TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Spectral element methods : theory and applications

Citation for published version (APA):
Vosse, van de, F. N., & Minev, P. D. (1996). *Spectral element methods : theory and applications*. (EUT report. W, Dept. of Mechanical Engineering; Vol. 96-W-001). Eindhoven University of Technology.

Document status and date:
Published: 01/01/1996

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Nov. 2023

Eindhoven
University of Technology
The Netherlands

Faculty of Mechanical Engineering

Spectral Element Methods: Theory and Applications

F.N. van de Vosse and P.D. Minev

EUT Report 96-W-001
ISBN 90-236-0318-5
©Eindhoven University of Technology, Eindhoven, June 1996.

Spectral Element Methods : Theory and Applications / F.N. van de Vosse and P.D. Minev. – Eindhoven : Eindhoven University of Technology, 1996. – 117 p. – (Eindhoven University of Technology Research Reports, ISSN 0167-9708, EUT Report 96-W-001). – ISBN 90-386-0318-5

Abstract

Some mathematical aspects of finite and spectral element discretizations for partial differential equations are presented. The weighted residual method is introduced and several kinds of collocation (finite difference and finite volume) and Galerkin (spectral and finite element) methods are derived as particular cases to that method. The concept of the spectral methods is described and an example of the application of the spectral element method to a second-order elliptic equation provides the reader practical information about it. Some direct and iterative methods to solve the resulting linear algebraic systems are described and some stabilization methods are introduced. An overview of the most commonly used time integration methods for unsteady problems is given in the context of the spectral space discretization. Different approaches for solution of the steady and unsteady Navier-Stokes are introduced in the context of the spectral and finite element methods. Some results of the practical implementation of SEM to 2-D problems are presented.

Keywords

Partial differential equations, Spectral element method, Time integration, Navier-Stokes equations

This publication can be ordered at:
Library Eindhoven University of Technology
P.O. Box 90159
NL 5600 RM Eindhoven
The Netherlands
fax: +31 40 244 70 15

Contents

1 Introduction **9**

2 Spatial discretization of partial differential equations **11**
2.1 Introduction 11
2.1.1 Strong formulation of a partial differential equation 11
2.1.2 Weighted residual formulation of a partial differential equation . . . 11
2.1.3 Weak formulation of a partial differential equation 13
2.1.4 Point collocation methods 13
2.1.5 Domain collocation methods 14
2.1.6 Galerkin methods 15
2.1.7 Numerical integration 15
2.2 Spectral methods 19
2.2.1 Spectral approximation 19
2.2.2 Chebyshev and Legendre polynomials 20
2.2.3 Pseudospectral approximation 21
2.3 Spectral element methods (SEM) 22
2.3.1 General remarks 22
2.3.2 Spectral element treatment of elliptic equations: 1-D example. . . . 23
2.3.3 Spectral element method in more dimensions 25
2.4 Solution methods for the algebraic system of equations 26
2.4.1 Direct methods 26
2.4.2 Iterative methods 27
2.5 Upwinding and other stabilization methods 29
2.5.1 Classical (finite difference) upwinding 29
2.5.2 Streamline upwind (SU) stabilization 30
2.5.3 Streamline upwind Petrov Galerkin (SUPG) stabilization 31
2.5.4 Galerkin least square (GLS) stabilization 31
2.6 Application of SEM to linear elasticity problems 31

3 Temporal discretization of partial differential equations **33**
3.1 Introduction 33
3.2 Standard implicit time integration methods 34
3.2.1 Adams-Moulton time integration schemes 35
3.2.2 Backward differencing time integration schemes 35
3.3 Standard explicit time integration methods 36
3.3.1 Adams-Bashforth time integration schemes 37
3.3.2 Runge-Kutta time integration schemes 37
3.4 Taylor-Galerkin methods 38
3.4.1 Explicit Taylor-Galerkin schemes 38

3.4.2 Implicit Taylor-Galerkin schemes . . . 39
3.5 Operator splitting . . . 39
3.6 Application of SEM to convection and convection diffusion problems . . . 41
3.6.1 One-dimensional linear convection . . . 41
3.6.2 One-dimensional non-linear convection . . . 42
3.6.3 One-dimensional unsteady strongly non-linear convection problem . 44
3.6.4 Two-dimensional linear convection . . . 44
3.6.5 1-D convection-diffusion of a Gaussian hill . . . 46
3.7 Application of SEM to wave equation . . . 49

4 Numerical solution of the Navier-Stokes equations **51**
4.1 Introduction . . . 51
4.2 Solution methods for the stationary Navier-Stokes equations . . . 51
4.2.1 Weak formulation . . . 51
4.2.2 Brezzi-Babuška stability condition . . . 53
4.2.3 Integrated method . . . 54
4.2.4 Linearization of the convective terms . . . 55
4.2.5 Penalty function method . . . 55
4.2.6 Uzawa methods . . . 56
4.3 Solution methods for the instationary Navier-Stokes equations . . . 57
4.3.1 Time integration methods . . . 57
4.3.2 Pressure correction and projection methods . . . 58
4.4 Solution of the Boussinesq equations . . . 60
4.5 Some numerical results of the SEM application to Navier-Stokes and Boussinesq problems . . . 61
4.5.1 Vortex shedding behind a cylinder . . . 61
4.5.2 Differentially heated cavity . . . 63

5 Problems **65**
5.1 Example 1: Introduction . . . 65
5.1.1 Example 1.1: Running a SEPRAN job . . . 67
5.1.2 Example 1.2: Mesh generation . . . 68
5.1.3 Example 1.3: Creation, printing and plotting a function . . . 70
5.2 Example 2: Numerical integration . . . 72
5.2.1 Example 2.1: Numerical integration . . . 73
5.3 Example 3: Steady convection-diffusion problems . . . 76
5.3.1 Example 3.1: Steady 1D diffusion . . . 76
5.3.2 Example 3.2: Steady 1D convection-diffusion . . . 82
5.3.3 Example 3.3: Steady 2D diffusion . . . 87
5.4 Example 4: Unsteady convection-diffusion problems . . . 90
5.4.1 Example 4.1: Euler implicit time integration . . . 90
5.5 Example 5: Unsteady convection problems . . . 98
5.5.1 Example 5.1: Euler implicit . . . 98
5.5.2 Example 5.2: Crank-Nicolson . . . 102

A Linear vector analysis **107**
A.1 Vector spaces . . . 107
A.2 Linear and bi-linear forms . . . 112

B Vector and tensor integrals **114**
B.1 Leibnitz formulae 114
B.2 Gauss-Ostrogradskii divergence theorem 114

Chapter 1

Introduction

In these lecture notes some mathematical aspects of finite and spectral element discretizations for partial differential equations are presented. The mathematics in these notes is not used to prove theorems and error estimates but only to obtain a better understanding of some aspects concerning the discretization of partial differential equations. As a consequence only little attention is paid on precise and formal mathematical fundamentals of the methods.
In chapter 2, the weighted residual method is introduced and several kinds of collocation (finite difference and finite volume) and Galerkin (spectral and finite element) methods are derived as particular cases to that method. Furthermore, the concept of the spectral methods is described and an example of the application of the spectral element method to a second-order elliptic equation provides the reader a practical information about it. Next, some direct and iterative methods to solve the resulting linear algebraic systems are described. At the end of the chapter some stabilization methods frequently used in the finite or spectral element formulations of convection-diffusion equations are introduced.
In chapter 3 an overview of the most commonly used time integration methods for unsteady problems is given in the context of the spectral space discretization. The possibilities to combine them using operator splitting are also discussed. At the end of this chapter, results of their practical application to some convection-diffusion problems are presented.
In chapter 4 different approaches for solution of the steady and unsteady Navier-Stokes are introduced in the context of the spectral and finite element methods. Some results of the practical implementation of SEM to 2-D problems are presented.

Chapter 2

Spatial discretization of partial differential equations

2.1 Introduction

Finite volume, finite element, spectral and also finite difference methods may be viewed as a specific application of the method of weighted residuals. In general the method of weighted residuals employs expansion functions [1] as basis functions for a truncated series expansion of the solution of the partial differential equation. In order to ensure that the approximate solution, defined by the truncated series expansion, satisfies the differential equation as closely as possible, test functions [2] are used to minimize the residual that is formed when the approximate solution is substituted into the partial differential equations. The combination of expansion and test functions distinguishes between the different spatial discretization methods mentioned above.

2.1.1 Strong formulation of a partial differential equation

To illustrate the framework of the weighted residual method consider a domain Ω with boundary Γ and assume that $f : \Omega \to \mathbb{R}$ is a given function. Then consider the following differential equation:

$$\begin{cases} \mathcal{L}u - f = 0 & \text{in } \Omega \\ u = u_\Gamma & \text{on } \Gamma \end{cases} \tag{2.1}$$

Here $\mathcal{L}$ is a continuous positive-definite differential operator. As an example we will consider the diffusion equation:

$$\begin{cases} -\dfrac{\partial^2 u}{\partial x^2} = f & \text{in } [0,1] \\ u(0) = 0 \qquad u(1) = 1 \end{cases} \tag{2.2}$$

2.1.2 Weighted residual formulation of a partial differential equation

If a set of trial functions, denoted by U, is defined as $U = \{u | u \in H^2(\Omega), u = u_\Gamma \text{ on } \Gamma\}$ and a set of test functions, denoted by W, is defined as $W = \{w | w \in L^2(\Omega), w = 0 \text{ on } \Gamma\}$,

[1] The expansion functions are also called trial or approximating functions.

[2] The test functions are also referred to as weighting functions.

a corresponding form of equation (2.1) is:

Find $u \in U$ such that:

$$(\mathcal{L}u - f, w)_W = 0 \qquad \forall_{w \in W} \tag{2.3}$$

Actually this form ensures the projection of the function $\mathcal{L}u - f$ on W to be zero. In terms of the $L^2(\Omega)$ inner product (2.3) reads:

Find $u \in U$ such that:

$$\int_\Omega (\mathcal{L}u - f) w d\Omega = 0 \qquad \forall_{w \in W} \tag{2.4}$$

The next step in the discretization scheme is to choose a finite dimensional subspace $U^h \subset U$ with basis φ_i, $(i = 0, ..., N)$. The trial functions φ_i are used as basis functions for a truncated series expansion of the solution. The approximate solution $u^h \in U^h$ is then written as:

$$u^h = \sum_{i=0}^{N} c_i \varphi_i \tag{2.5}$$

Depending on the choice of the space U^h, either the exact differential operator $\mathcal{L}$ or an appropriate discrete differential operator $\mathcal{L}^h$ can be used. If this approximation is substituted in the differential equation (2.1), it will not be identically zero but: $\mathcal{L}^h u^h - f = r^h$ in Ω where r^h is called the residual of the equation.
The expansion coefficients c_i are the unknowns that can be obtained by requiring the residual to be zero in the L_2-norm: $(r^h, w)_W = 0$, $\forall_{w \in W}$ [3]. Since the approximate solution and thus r^h now is an element of a finite dimensional subspace of U, also the space of test functions W can be reduced to a finite dimensional subspace $W^h \subset W$. To this end a basis ψ_j $(j = 0, ..., N)$ of test functions is introduced such that $W^h = \{\psi_j\}_{i=0}^N$ and the discrete weighted-residual formulation then reads:

Find $u^h \in U^h$ such that:

$$(\mathcal{L}^h u^h - f, w^h)_W = 0 \qquad \forall_{w^h \in W^h} \tag{2.6}$$

or equivalently again using the L_2-inner product:

Find c_i, $(i = 0, ...N)$ such that:

$$\sum_{i=0}^{N} c_i \int_\Omega (\mathcal{L}^h \varphi_i) \psi_j d\Omega = \int_\Omega f \psi_j d\Omega \qquad j = 0, ..., N \tag{2.7}$$

In matrix notation this yields:

$$\mathbf{Lc} = \mathbf{f} \tag{2.8}$$

with:

$$L_{ij} = \int_\Omega (\mathcal{L}^h \varphi_j) \psi_i d\Omega, \qquad f_i = \int_\Omega f \psi_i d\Omega. \tag{2.9}$$

[3] Least square methods minimize $(r^h, r^h)_W$.

and $\mathbf{c} = [c_0, ..., c_N]^T$, $\mathbf{f} = [f_0, ..., f_N]^T$. Once the coefficients c_i are obtained from the set of equations (2.8) the approximate solution u^h of the partial differential equation (2.1) can be computed from (2.5).
Different choice for the test function ψ_j results in different discretization methods. Some of them will be mentioned in (2.1.4÷2.1.6).

2.1.3 Weak formulation of a partial differential equation

If $\mathcal{L}$ is a second order differential operator (that is the case with a lot of the equations of the mathematical physics) it is convenient to perform an integration by parts of the weighted residual form (2.3). In many cases an equivalent bilinear form $a(u,w)_W$ can be derived such that (2.1) can be written as:
Find $u \in U$ such that:

$$a(u,w)_W = (f,w)_W \qquad \forall_{w \in W} \tag{2.10}$$

For the diffusion equation (2.2) we find:

$$a(u,w)_W = (\frac{\partial u}{\partial x}, \frac{\partial w}{\partial x})_W \tag{2.11}$$

According to the Lax-Milgram theorem (see Appendix A2), this problem has a unique solution u equivalent to the one of the original differential equation if the bilinear form $a(u,w)$ is coercive on W (positive definite) and bounded.
Note that the inner product $(\frac{\partial u}{\partial x}, \frac{\partial w}{\partial x})$ requires that now both $U \subset H^1(\Omega)$ and $W \subset H^1(\Omega)$. This weakens the restriction for u (originally $u \in H^2(\Omega)$ for second order differential equations). Often the weak formulation is derived from a variational form of a minimization problem and is referred to as the variational formulation of the differential equations (see e.g. Reddy and Rasmussen, 1982).
The integration by parts results in boundary integrals which vanish on the parts of the boundary where Dirichlet boundary conditions are prescribed. On the rest of the boundary the boundary conditions have to be formulated in a form which enables the evaluation of these integrals - so called natural boundary conditions of the problem.

2.1.4 Point collocation methods

In point collocation methods collocation points $\mathbf{x}_j$ are defined in Ω and the test functions ψ_j are chosen to be the Dirac delta functions according to:

$$\psi_j(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_j) \tag{2.12}$$

Substitution in the weighted residual equation (2.7) then yields:

Find u^h such that:

$$\mathcal{L}^h u^h|_{\mathbf{x}=\mathbf{x}_j} = f(\mathbf{x}_j) \qquad j = 0, ..., N \tag{2.13}$$

The residual r^h is forced to be zero in the set of collocation points $\{\mathbf{x}_j\}_{j=1}^N$. Typical examples of point collocation methods are:

Orthogonal collocation methods :
The approximating functions are chosen to be orthogonal polynomials in W i.e.:

$$(\phi_i, \phi_j)_W = 0 \qquad \text{for } i \neq j \tag{2.14}$$

Examples of orthogonal polynomials that are commonly used are Legendre and Chebyshev polynomials. The coefficients c_i of the truncated expansion functions (2.5) are chosen to be the values (u_i) of the approximate solution in the collocation points. As the polynomials are analytical functions, the discrete differential operator $\mathcal{L}^h$ can be equal to the original operator $\mathcal{L}$ but also a discrete version can be derived if the derivatives are expressed in terms of the coefficients of the approximate solution. An extended description can be found in Canuto *et al.* (1988).

Finite difference methods : The finite difference method can be seen as a point collocation method without the use of an approximate solution. Here a discrete differential operator $\mathcal{L}^h$ is derived using truncated Taylor-series around the collocation points:

Find $u(\mathbf{x}_j)$ *such that:*

$$\mathcal{L}^h u|_{\mathbf{x}=\mathbf{x}_j} = f(\mathbf{x}_j) \qquad j = 0, ..., N \tag{2.15}$$

The error of finite difference approximations is determined by both the number of collocation points chosen and the truncation error in the Taylor series used to approximate the differential operator. In Hirsch (1988) the finite difference method is treated in details.

2.1.5 Domain collocation methods

In domain collocation methods subdomains Ω_j are defined in Ω and the test functions ψ_j are chosen to be functions according to:

$$\psi_j(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{x} \in \Omega_j \\ 0 & \text{for } \mathbf{x} \notin \Omega_j \end{cases} \tag{2.16}$$

Equation (2.6) then yields:

Find u_h *such that:*

$$\int_{\Omega_j} (\mathcal{L}^h u^h - f) d\Omega_j = 0 \qquad j = 0, ..., N \tag{2.17}$$

Typical examples of domain collocation methods are:

Finite volume methods : Similar to finite difference methods there is no explicit introduction of an approximate solution. The volume integrals over the subdomains Ω_j are mostly expressed in surface integrals using Green's theorem. The approximation error is determined by both the number of subdomains and the accuracy of the integration method used. In Hirsch (1988) the finite volume method is treated in details.

2.1.6 Galerkin methods

If the spaces U^h and W^h are chosen to be the same and the weak formulation (2.10) is used as a starting point the method is called a Galerkin weighted-residual method:

Find $u^h \in U^h$ such that:

$$a(u^h, w^h)_W = (f, w^h)_W \qquad \forall_{w^h \in W^h} \tag{2.18}$$

Let u be the exact solution of the weighted-residual formulation (2.3). Then since $U^h \subset U$ it follows:

$$a(u, w^h)_U = (f, w^h)_U, \quad \forall w^h \in U^h \tag{2.19}$$

Subtracting (2.19) from (2.18):

$$(\mathcal{L}(u^h - u), w^h)_U = 0, \quad \forall w^h \in U^h \tag{2.20}$$

which may be interpreted as an orthogonal condition: the error $e = u - u^h$ of the Galerkin approximation u^h of the solution of (2.3) is orthogonal (in U-sense) to the subspace U^h. Now suppose that $a(u, w)$ is a symmetric and positive definite: $a(u, w) = a(w, u)$ and $a(u, u) \geq 0, \forall u, w \in U; a(u, u) = 0 \Longleftrightarrow u \equiv 0$. Then for arbitrary $w^h \in U^h$:

$$\begin{aligned} a(u - w^h, u - w^h) &= a(e + (u^h - w^h), e + (u^h - w^h)) && (2.21) \\ &= a(e, e) + a(u^h - w^h, u^h - w^h) && (2.22) \end{aligned}$$

where (2.20) is used. Since a is positive definite it follows that $a(u - w^h, u - w^h)$ reaches its minimum for $w^h = u^h$, i.e. from all the functions $w^h \in U^h$ the closest to the actual solution u (in the norm of U^h) is the Galerkin approximation u^h. That is why it is called the best approximation to u in U^h.
In case that $a(u^h, w^h)$ is continuous and positive definite on U^h the Lax-Milgram lemma holds and the Galerkin problem (2.18) has an unique solution. It is important to know that it may possess an unique solution even if the weighted-residual formulation (2.3) may not because in the approximate (Galerkin) problem we require $a(u^h, w^h)$ to be positive definite on a certain subspace of U but not in the whole U.
Typical examples of a Galerkin methods are:

Galerkin spectral methods : For spectral methods the trial functions are infinitely differentiable global functions. A more detailed description of spectral methods is given in section 2.2.

Galerkin finite element methods : In finite element methods, the domain Ω is divided into elements, and trial functions are specified in each element and are local in character (see section 2.3).

2.1.7 Numerical integration

All the methods which start from an integral formulation of the conservation laws (typical examples are the finite element method, finite volume method and the spectral methods), require evaluation of volume or surface integrals. Some of them (like the finite volume method) evaluate these integrals by means of a simple trapezoidal rule which retains

the accuracy of the method. The higher order methods, however, require higher order integration rules. Common feature of these methods (except the Fourier spectral methods) is that the solution is expanded over a certain polynomial basis. Thus, they require the calculation of integrals of polynomials of certain order. The quadratures derived from the requirement to be exact for all the polynomials of certain order are called Gauss quadratures. The derivation of such quadratures proceeds as follows. The general formula for numerical integration can be written as:

$$\int_a^b p(\xi)f(\xi)d\xi = \sum_{i=0}^{N} w_i f(\xi_i) + R_N(f) \tag{2.23}$$

where $p(\xi)$ is the weight function of the integration satisfying $p(\xi) \geq 0$ and $\int_a^b p(\xi)d\xi > 0$ and $R_N(f)$ is the error of the quadrature. The Gauss numerical integration problem then formulates as: find w_i and ξ_i such that $R_N(f) \equiv 0$ for polynomials of the maximal possible degree. Since (2.23) contains 2N+2 free parameters it cannot be generally exact for polynomials of order higher than $2N+1$. Let $Q_0 = 1, Q_1, ..., Q_N, ...$ is the system of orthogonal polynomials with respect to the weight function $p(\xi)$, i.e.:

$$\int_a^b p(\xi)Q_iQ_jd\xi = \delta_{ij}, \quad i,j = 0, ..., N, ... \tag{2.24}$$

with δ_{ij} being the Cronecker symbol. Note that for a given $p(\xi)$ the system Q_i is uniquely determined by (2.24). In case of finite element methods and many of the spectral methods $p(x) = 1$ and the corresponding orthogonal system consists of the so-called Legendre polynomials (see 2.2.2). Another important particular case is the system of Chebyshev polynomials orthogonal with respect to $p(\xi) = 1/\sqrt{1-\xi^2}$ which is used as a basis for some spectral methods (see 2.2.2). Let we take $\{\xi_i\}_{i=0}^N$ to be the zeros of Q_{N+1}. Then (2.23) defines unique sequence $\{w_i\}_{i=0}^N$ such that it is exact for all the polynomials of order N. We shall prove now that $R_N(f) \equiv 0$ for all the polynomials of order $2N+1$. Let Φ is an arbitrary polynomial of order $2N+1$. Then we can write it as:

$$\Phi(\xi) = Q_{N+1}(\xi)q(\xi) + r(\xi), \quad q, r \in P_N \tag{2.25}$$

with P_N being the linear space consisting of all the polynomials of order less or equal to N. From (2.24) and (2.25) it follows that:

$$\int_a^b p(\xi)\Phi(\xi)d\xi = \int_a^b p(\xi)Q_N(\xi)q(\xi)d\xi + \int_a^b p(\xi)r(\xi)d\xi \tag{2.26}$$

$$= \int_a^b p(\xi)r(\xi)d\xi \tag{2.27}$$

But since $\Phi(\xi_i) = r(\xi_i)$ (ξ_i are zeros of Q_{N+1}) then:

$$\int_a^b p(\xi)\Phi(\xi)d\xi \equiv \sum_{i=0}^{N} w_i\Phi(\xi_i) \tag{2.28}$$

The opposite can also be proved, i.e. if (2.23) is exact for all the polynomials of order $2N+1$ than $\{\xi_i\}_{i=0}^N$ must be the zeros of Q_{N+1} and $\{w_i\}_{i=0}^N$ should be chosen as given above.

In the most important cases of Legendre and Chebyshev orthogonal systems the weights and nodes of the corresponding quadratures are given below.

Chebyshev-Gauss :

The Gauss points are:

$$x_j = \cos \frac{(2j+1)\pi}{2N+2} \tag{2.29}$$

The weights for numerical integration are:

$$w_j = \frac{\pi}{N+1} \qquad 0 \leq j \leq N \tag{2.30}$$

Legendre-Gauss :

The Gauss points are:

$$x_j = \text{zeroes of } L_{N+1} \qquad 0 \leq j \leq N \tag{2.31}$$

The weights for numerical integration are:

$$w_j = \frac{2}{(1-x_j^2)[L_{N+1}(x_j)]^2} \qquad j = 0, ..., N \tag{2.32}$$

For many practical needs it is convenient to include the edges of the interval among the nodes of the quadrature. Since the number of the free parameters in (2.23) is than 2N one can expect that the resulting quadrature cannot be generally exact for polynomials of order higher than $2N-1$. Indeed, in a way similar to the one described above, a quadrature can be constructed which is exact for all the polynomials of order $2N-1$ and not exact for all the polynomials $2N$. It is called Gauss-Lobatto quadrature. In the case of Chebyshev and Legendre orthogonal systems the nodes and weights of the corresponding quadratures read:

Chebyshev-Gauss-Lobatto :

The Gauss-Lobatto points are:

$$x_j = \cos \frac{\pi j}{N} \tag{2.33}$$

The weights for numerical integration are:

$$w_0 = \frac{\pi}{2N}, \quad w_j = \frac{\pi}{N}, \quad w_N = \frac{\pi}{2N} \qquad 1 \leq j \leq N-1 \tag{2.34}$$

Legendre-Gauss-Lobatto :

The Gauss-Lobatto points are:

$$x_0 = -1, \quad x_j = \text{zeroes of } L_N', \quad x_N = 1 \qquad 1 \leq j \leq N-1 \tag{2.35}$$

The weights for numerical integration are:

$$w_j = \frac{2}{N(N+1)} \frac{1}{[L_N(x_j)]^2} \qquad j = 0, ..., N \tag{2.36}$$

For more detailed information on Gauss and Gauss-Lobatto integration the reader is referred to Canuto *et al.* (1988).

Example 1 *Legendre-Gauss-Lobatto integration of polynomials.*

Let we choose $N = 3$. The Gauss-Legendre-Lobatto points then are:

$$-\xi_o = \xi_3 = 1, -\xi_1 = \xi_2 = 0.4472... \tag{2.37}$$

and the corresponding weights:

$$w_0 = w_3 = \frac{1}{6}, \quad w_1 = w_2 = \frac{5}{6} \tag{2.38}$$

The integral:

$$\int_{-1}^{1} (1 + \xi + \xi^2 + \xi^3 + \xi^4 + \xi^5) d\xi = 3.0666... \tag{2.39}$$

is exactly calculated by means of GLL quadrature (check it).
Consider the integral:

$$\int_{-1}^{1} \xi^6 d\xi = \frac{2}{7} = 0.2857... \tag{2.40}$$

The GLL quadrature for $N = 3$ yields a value of 0.3466... which is about 20% higher.

2.2 Spectral methods

2.2.1 Spectral approximation

As mentioned, in the weighted residual method the solution $u \in U$ is expanded in a series of expansion functions:

$$u = \sum_{i=0}^{\infty} c_i \varphi_i \tag{2.41}$$

with c_i being the expansion coefficients and φ_i belonging to the orthogonal set of trial functions. The orthogonality with respect to a weight function w is defined by:

$$\int_{-1}^{1} \varphi_i(x)\varphi_j(x)w(x)dx = \delta_{ij} \tag{2.42}$$

Then the coefficients c_i in (2.41) are given by the weighted inner product:

$$c_i = \frac{1}{\|\varphi_i\|^2} \int_{-1}^{1} u(x)\varphi_i(x)w(x)dx \tag{2.43}$$

with:

$$\|\varphi_i\|^2 = \int_{-1}^{1} \varphi_i(x)\varphi_i(x)w(x)dx \tag{2.44}$$

The expansion (2.41) underlies all the spectral methods. A classical example of such a method is the Fourier spectral method using the set of functions:

$$\varphi_i(x) = e^{ikx} \tag{2.45}$$

which is orthogonal in the interval $(0, 2\pi)$ with weight 1. If u is infinitely smooth and periodic together with all its derivatives then the k-th coefficient of the expansion decays faster than any inverse power of k. In practice, of course, this never happens but this property (called spectral accuracy) is attainable also for non-periodic but smooth functions provided that the orthogonal set is properly constructed. Another classical result of the approximation theory (Gottlieb and Orszag, 1977) is that for analytical functions exponential (or spectral) decay of the coefficients can be obtained for trial functions that are eigenfunctions of singular Sturm-Liouville problems defined on $\Omega = (-1, 1)$.

$$-\frac{d}{dx}\left(a(x)\frac{d\varphi_i}{dx}\right) + b(x)\varphi_i = \lambda_i w(x)\varphi_i, \quad a > 0, b \geq 0 \tag{2.46}$$

In general, polynomial solutions of singular Sturm-Liouville problems are Jacobi polynomials like Chebyshev and Legendre polynomials (see section 2.2.2). Since the Jacobi polynomials are mutually orthogonal over the interval (-1,1) it can be proven that $\forall_{u \in U}$:

$$\lim_{N \to \infty} \|u - P_N^h u\|_U = 0 \tag{2.47}$$

If $u \in H^m(\Omega)$ so say if u is m times differentiable the truncation error can be approximated by (Canuto *et al.*, 1988):

$$\|u - P_N^h u\|_{L_2} \leq C_1 N^{-m} \|u\|_{H^m} \tag{2.48}$$

So an exponential convergence is obtained for infinitely smooth functions.
In practice, instead of (2.41) a finite expansion is used represented by the truncated series:

$$P_N^h u = \sum_{i=0}^{N} c_i \varphi_i \tag{2.49}$$

In spectral methods convergence is achieved by increasing N.

2.2.2 Chebyshev and Legendre polynomials

The most commonly used special cases of Jacobi polynomials are the Chebishev and Legendre polynomials.

Chebyshev polynomials If in (2.46) we take $a(x) = (1-x^2)^{1/2}$, $b(x) = 0$ and $w(x) = (1-x^2)^{-1/2}$ the solutions are Chebyshev polynomials given by the recurrence relation:

$$\begin{cases} T_0(x) & = 1 \\ T_1(x) & = x \\ T_{n+1}(x) & = 2xT_n(x) - T_{n-1}(x) \end{cases} \tag{2.50}$$

Legendre polynomials If in (2.46) we take $a(x) = (1-x^2)$, $b(x) = 0$ and $w(x) = 1$ the solutions are Legendre polynomials given by the recurrence relation:

$$\begin{cases} L_0(x) & = 1 \\ L_1(x) & = x \\ L_{n+1}(x) & = \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x) \end{cases} \tag{2.51}$$

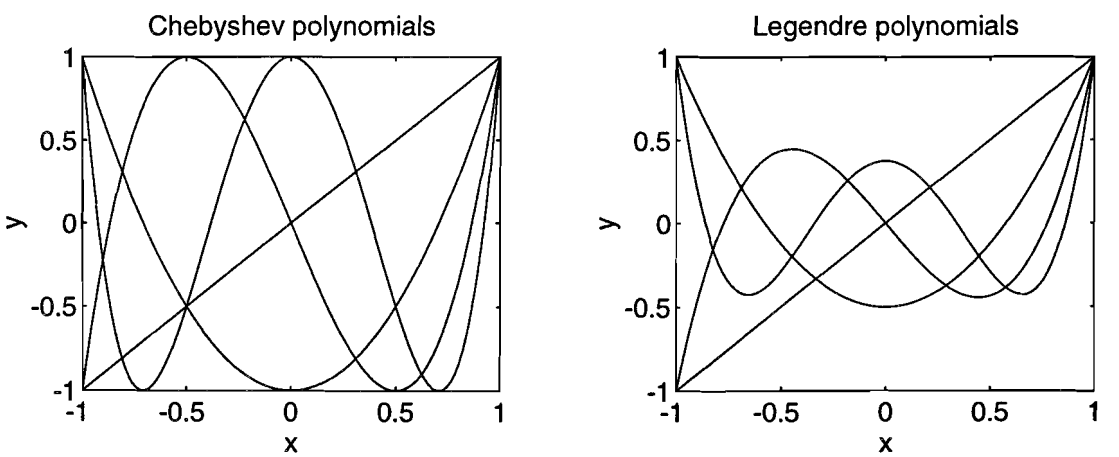

Figure 2.1: Chebyshev and Legendre polynomials for $n = 1, ...5$.

Note that for Legendre polynomials the weight function w is defined by $w(x) = 1$ which easily enables an integration by parts in Galerkin formulations of second order differential equations. For Chebyshev polynomials, where w is given by $w(x) = (1-x^2)^{-1/2}$ this is not the case. It is for this reason that in weak formulations mostly Legendre polynomials are used.

2.2.3 Pseudospectral approximation

Actually the spectral approximation defines a transform from the physical space to the spectral space (like the Fourier coefficients in a Fourier transform). The coefficients c_i in the spectral approximation depend on all the values of $u(x)$ in the physical space and can only be computed by numerical integration. Since this can not be performed exactly for arbitrary functions $u(x)$, in pseudospectral methods a set of approximate coefficients $\hat{c}_i$ is derived using an interpolating polynomial $\Pi_N^h u(x)$ of $u(x)$ defined by a finite set of interpolation points. So, an interpolant is constructed as:

$$\Pi_N^h u = \sum_{i=0}^{N} \hat{c}_i \varphi_i \tag{2.52}$$

The interpolating polynomial satisfies

$$\Pi_N^h u(x_k) = u(x_k), \qquad k = 0, ..., N \tag{2.53}$$

If x_k and w_k are the quadrature points and weights of some numerical quadrature rule, the discrete coefficients $\hat{c}_i$ can be approximated by:

$$\hat{c}_i = \frac{1}{\|\varphi_i\|^2} \sum_{k=0}^{N} u(x_k)\varphi_i(x_k)w_k \tag{2.54}$$

with

$$\|\varphi_i\|^2 = \sum_{k=0}^{N} \varphi_i(x_k)\varphi_i(x_k)w_k \tag{2.55}$$

It can be shown that spectral convergence is retained in replacing the continuous transform (2.49) by the interpolating polynomial (2.52) if the interpolation points are the corresponding Gauss-type quadrature points. The interpolation error then can be approximated by (Canuto *et al.*, 1988):

$$\|u - \Pi_N^h u\|_{L_2} \leq C_2 N^{1/2} N^{-m} \|u\|_{H^m} \tag{2.56}$$

Still the coefficients $\hat{c}_i$ have to be computed from (2.54). In practice, however, the interpolation polynomials are written as a linear combination of Lagrange interpolation polynomials through the Gauss-type quadrature points:

$$\Pi_N^h u = \sum_{i=0}^{N} u_i \phi_i \tag{2.57}$$

in this way the coefficients are just given by the value of the function in the interpolation points $u_i = u(x_i)$.

Chebyshev-Gauss-Lobatto-Lagrange interpolation polynomials :

The basisfunctions ϕ_i then are given by:

$$\phi_i = \frac{(-1)^{i+1}}{\alpha_i n^2} \frac{(1-x^2)T_n'(x)}{x - x_i} \tag{2.58}$$

with $\alpha_i = 1 (i = 1, ..., N-1), \alpha_0 = \alpha_N = 2$.

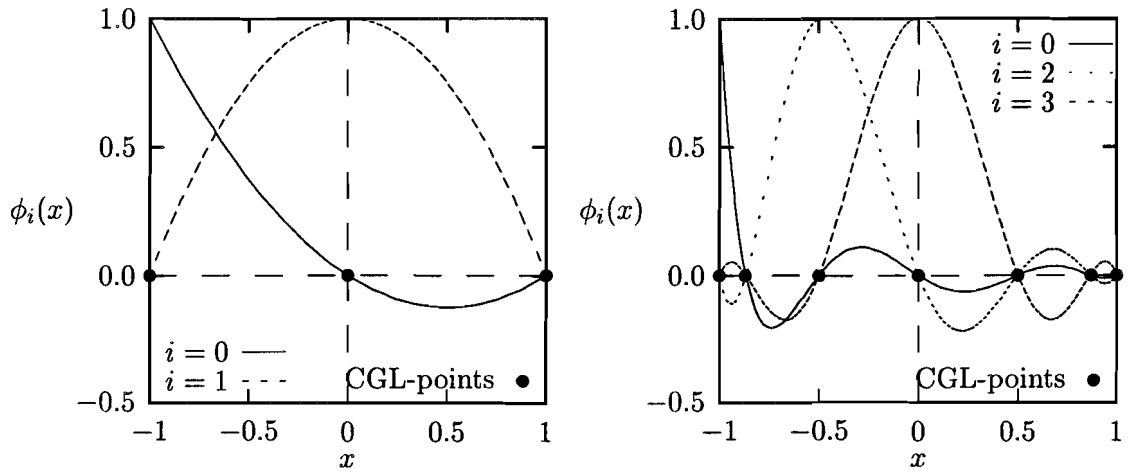

Figure 2.2: Lagrange interpolants $\phi_i(x)$ $(i = 0, \ldots, N)$ through the Chebyshev Gauss-Lobatto points (•) for $N = 2$ (left) and $N = 6$ (right).

Legendre-Gauss-Lobatto-Lagrange interpolation polynomials :
The basisfunctions ϕ_i then are given by:

$$\phi_i = \frac{-1}{N(N+1)L_N(x_i)} \frac{(1-x^2)L_N'(x)}{x - x_i} \tag{2.59}$$

In summary, it has been shown that the interpolation error of Lagrange interpolation polynomials shows spectral convergence if the interpolation points are Gauss-type quadrature points corresponding with Jacobi polynomials. In practice the Gauss-Lobatto points are taken in order to be able to prescribe function values at the boundary. The Gauss points are all located in the internal of the domain. As the weight function for Legendre polynomials is given by $w = 1$, for combination with variational (weak) formulations of partial differential equations Legendre polynomials are more suitable then Chebyshev polynomials.

2.3 Spectral element methods (SEM)

2.3.1 General remarks

Spectral elements, proposed by Patera (1984), combine the advantages and disadvantages of Galerkin spectral methods with those of finite element methods by a simple application of the spectral method per element. This means that, like in finite element methods, the domain is divided into N_{el} non-overlapping subdomains (elements) Ω_e:

$$\bar{\Omega} = \bigcup_{e=1}^{N_{el}} \bar{\Omega}_e, \qquad \bigcap_{e=1}^{N_{el}} \Omega_e = \emptyset \tag{2.60}$$

Again the space of approximation U^h is taken to be:

$$U^h = \{u \in U \mid u|_{\Omega_e} \in P_N(\Omega_e)\} \tag{2.61}$$

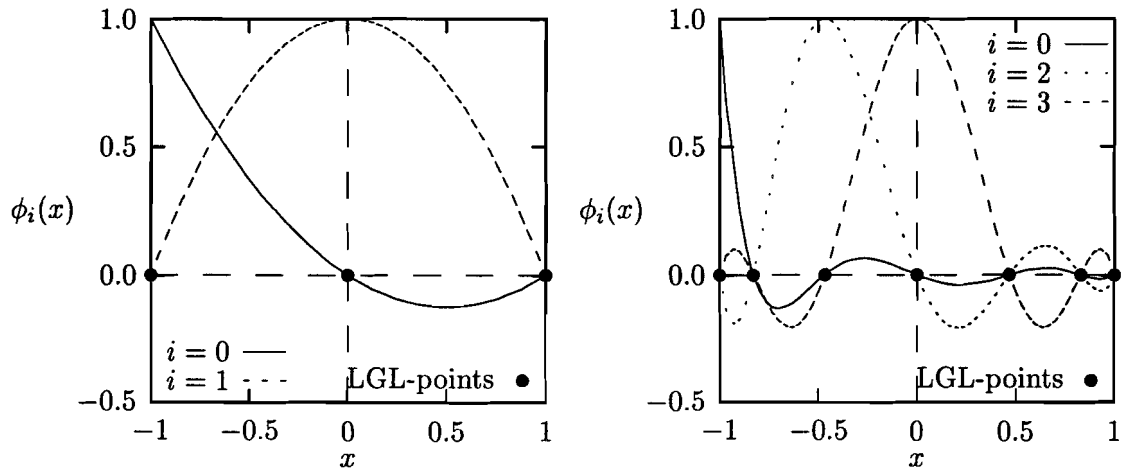

Figure 2.3: Lagrange interpolants $\phi_i(x)$ $(i = 0, \ldots, N)$ through the Legendre Gauss-Lobatto points (•) for $N = 2$ (left) and $N = 6$ (right).

where $P_N(\Omega_e)$ denotes the space of polynomials in Ω_e of degree $\leq N$. Convergence is either obtained by increasing the degree of the polynomials or by increasing the number of elements N_{el}. The basis functions ϕ_i are typically high-order Lagrange interpolation polynomials through the local Gauss-Lobatto integration points defined per element.
If $N_{el} = 1$ we obtain a spectral Galerkin method of order $N_{nd} - 1$. If $N = 1$ or $N = 2$ a standard Galerkin finite element method is obtained based on linear and quadratic elements repectively.

2.3.2 Spectral element treatment of elliptic equations: 1-D example.

Consider the one-dimensional Helmholtz problem:
Find u defined over $\Omega = [-1, 1]$ such that:

$$-\frac{d}{dx}(\eta \frac{du}{dx}) + \lambda^2 u = f \quad \text{in} \quad \Omega \tag{2.62}$$

$$u(-1) = u(1) = 0 \tag{2.63}$$

where λ is a real number and $\eta(x)$ is a function defined over Ω, bounded and positive. The starting point of the spectral element discretization is the Galerkin formulation of (2.62)-(2.63) which reads: Find $u \in H_0^1(\Omega)$ such that

$$\forall v \in H_0^1(\Omega), \quad a(u, v) = (f, v) \tag{2.64}$$

where the continuous bilinear form a is defined as

$$a(u, v) = \int_\Omega \eta \frac{du}{dx}\frac{dv}{dx} dx + \lambda^2 \int_\Omega u(x)v(x)dx \tag{2.65}$$

Further, the domain Ω is divided in K non-overlapping subdomains Ω_k. Since $u, v \in H^1(\Omega)$ the integrals in (2.64) can be decomposed as sums of the same integrals over $\Omega_k, k = 1, K$

$$\sum_{k=1}^{K} [\int_{\Omega_k} p\frac{du}{dx}(x)\frac{dv}{dx}(x)dx + \lambda^2 \int_{\Omega_k} u(x)v(x)dx] = \sum_{k=1}^{K} \int_{\Omega_k} f(x)v(x)dx \tag{2.66}$$

In order to complete the discretization one has to choose an approximation space for u: $X_h \subset H_0^1$ and a quadrature for the evaluation of the integrals in (2.66). Similar to the case of pseudospectral approximation discussed above the basis of X_h is formed of the elemental Lagrangian interpolants through the Gauss-Lobatto points in Ω_k extended with 0 outside the k-th element. Thus, the restriction of the solution u on Ω_k is approximated with $\Pi_{N,k}^h u$:

$$\Pi_{N,k}^h u = \sum_{j=0}^{N} u_j^k \phi_j^k \quad \text{in} \quad \Omega_k \tag{2.67}$$

with ϕ_j^k defined similarly to the one in (2.59) but for the interval Ω_k. Substitution of (2.67) into (2.66) and and choosing $v = \phi_i^k, i = 0, ..., N; k = 1, ..., K$ one finally arrives at a linear system of equations with respect to u_j^k:

$$\sum_{k=1}^{K} \sum_{j=o}^{N} C_{ij}^k u_j^k = \sum_{k=1}^{K} f_i^k, \quad i = 0, N \tag{2.68}$$

where

$$C_{ij}^k = \int_{\Omega_k} (\eta \frac{d\phi_i^k}{dx} \frac{d\phi_j^k}{dx} + \lambda^2 \phi_i^k \phi_j^k) dx \tag{2.69}$$

$$f_i^k = \int_{\Omega_k} f \phi_i^k dx \tag{2.70}$$

Here some comments on the choice of the basis of the approximation space X_h are in order. Note that the global interpolant:

$$u_h = \sum_{k=1}^{K} \sum_{j=0}^{N} u_j^k \phi_j^k \tag{2.71}$$

has to be in $H^1(\Omega)$ which requires its continuity over the elemental boundaries. The choice of Gauss-Lobatto Lagrangian interpolants as a local basis allows us to impose very easy this requirement by just setting $u_0^k = u_N^{k-1}$ and $u_N^k = u_0^{k+1}, k = 2, ..., K-1$. Moreover, in that way the elements are coupled only at the elemental boundaries resulting in a simple implementation and a relatively sparse matrix. The eventual use of Gauss Lagrangian interpolants would either couple all nodes of all the elements or would result in a discontinuous approximation.

The integrals in (2.69)-(2.70) have to be evaluated by means of a numerical quadrature. First, we use an affine mapping ${\Lambda_k}^{-1}$ of each element Ω_k into the standard interval $[-1, 1]$: $x = \Lambda_k(\xi)$. An integral of the type: $\int_{\Omega_k} r(x)dx$ is than transformed to: $\int_{-1}^{1} r(\xi) J_k d\xi$ where J_k is the determinant of the Jacobian of the transform Λ_k. This transform facilitates the implementation of the method. Moreover, in 2- and 3-D case it allows the usage of complex-shaped isoparametric elements and thus handling of complicated geometries. The choice of a quadrature formula is determined by the requirement that the integration error has to be of the same order or smaller than the approximation error. The quantities to be integrated are polynomials of order $2N-2$ in the case of the stiffness matrix and $2N$

in the case of the mass matrix. This suggests a Gauss type formula associated with the Legendre polynomials because such a formula based on $N+1$ nodes is exact for polynomials of order $2N+1$. It is very attractive to use a Gauss quadrature based on the Legendre-Gauss-Lobatto points in $[-1,1]$. This choice combined with the basic functions introduced above would result in a diagonal mass matrix which will prove important in the context of iterative or time-dependent procedures latter on. Moreover, in 2-D and 3-D case it allows a dramatic decrease of the number of operations and storage requirements for the construction of the stiffness matrix as well. The disadvantage is that this quadrature is exact only for polynomials of order $2N-1$. Maday and Patera (1989) proved , however, that if u, f and p are analytical functions this quadrature preserves the most attractive property of the spectral methods - their exponential convergence. If a Legendre-Gauss-Lobatto quadrature is applied the elements of the matrix given by (2.69) become:

$$C_{ij}^k = \sum_{l=0}^{N} w_l J_k \eta(x_l) \frac{d\phi_i}{d\xi}(\xi_l) \frac{d\phi_j}{d\xi}(\xi_l) + \sum_{l=0}^{N} w_l J_k \lambda^2 \phi_i(\xi_l) \phi_j(\xi_l) \tag{2.72}$$

where w_l and ξ_l are respectively the Legendre-Gauss-Lobatto weights and points in $[-1,1]$ (see 2.1.7) and x_l are the points in Ω_k corresponding to ξ_l after the transform Λ_k is used. Note that the superscript k of the basic functions is skipped here because after the affine mapping they become independent of the element. It is clear now that since the basic functions are chosen to be the Lagrangian interpolants through $\xi_i, i = 0, ..., N$ they satisfy: $\phi_i(\xi_j) = \delta_{ij}$. This simplifies considerably the second term on the right-hand side of (2.72) corresponding to the mass matrix of the problem and it becomes: $\lambda^2 M_{ij}^k = w_i J_k \lambda^2 \delta_{ij}$. In the same manner the right-hand side finally becomes: $f_i^k = M_{ii}^k f(x_i)$.

2.3.3 Spectral element method in more dimensions

The extension of the method described in the previous section towards two- and three-dimensional problems is straightforward. Just the more-dimensional basic functions are constructed as a tensor product of the one-dimensional ones:

$$\Psi_{lmn} = \phi_l \phi_m \phi_n, \quad \text{for} \quad l, m, n = 0, ..., N \tag{2.73}$$

The Legendre-Gauss-Lobatto quadrature is also a tensor-product extension of the one-dimensional quadrature with weights: $w_{lmn} = w_l w_m w_n, \quad l, m, n = 0, ..., N$ and nodes: $\xi_{lmn} = (\xi_l, \xi_m, \xi_n), \quad l, m, n = 0, ..., N$. The final algebraic system then reads:

$$\sum_{k=1}^{K} \sum_{p,q,r=0}^{N} C_{stvpqr}^k u_{pqr}^k = \sum_{k=1}^{K} \sum_{p,q,r=0}^{N} M_{stvpqr}^k f_{pqr}^k \tag{2.74}$$

for $s, t, v = 0, ..., N$. A direct computation of the residual on the left-hand side of (2.74) would require $O(N^6)$ operations since one must sum over $p, q, r = 0, ..., N$ for $s, t, v = 0, ..., N$. The storage requirement is of the same order since the matrix C is, in general, full. Using, however, (2.73) and the fact that $\phi_i(\xi_j) = \delta_{ij}$ the number of operations for evaluation of a stiffness matrix is reduced to $O(N^4)$ and the storage requirement $O(N^3)$. The mass matrix is again diagonal. The estimation for the storage requirement is valid only if an iterative method is used to invert the matrix requiring calculation only of residual vectors. If a direct method is applied (Gauss elimination, for example) the storage requirement increases a lot, depending on the storage strategy used. The choice between a direct or iterative solver for the linear system depends mainly on the number of

degrees of freedom involved and the type of the available computer and will be discussed in the section concerning the solution of the Navier-Stokes equations.
Now we can demonstrate the exponential convergence of the spectral element method on a 2-D example possessing an analytical solution. We consider the Helmholtz equation on a domain $\Omega = [0,1] \times [0,2]$:

$$\nabla^2 T - 2T = 0 \quad \text{in} \quad \Omega \tag{2.75}$$

$$T|_{\partial\Omega} = e^{x+y} \tag{2.76}$$

The solution of this boundary value problem is: $T = e^{x+y}$. Ω is divided into 2 square elements and the problem is solved using increasing orders of the approximation. The result for the maximum pointwise error of the spectral element solution is given in fig. 2.4. A clear exponential convergence is obtained which is to be expected since the solution is an analytical function.

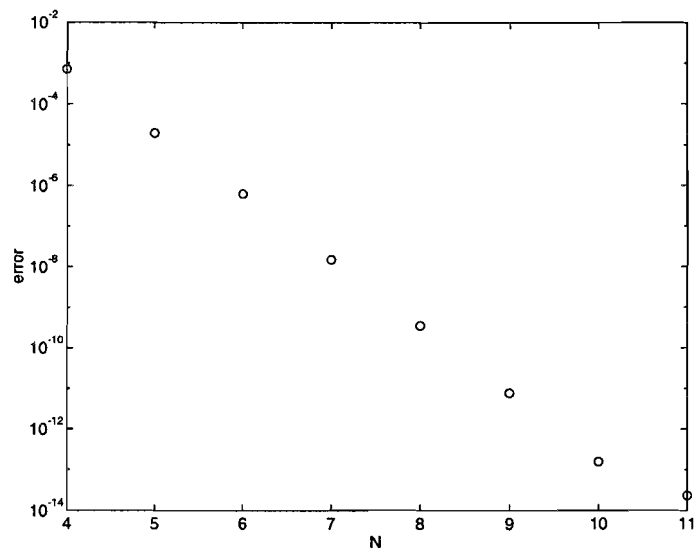

Figure 2.4: Maximum pointwise error in the spectral element solution of (2.76) as a function of the number of points in one direction

2.4 Solution methods for the algebraic system of equations

2.4.1 Direct methods

All direct methods for linear systems of equations are some variations of the Gaussian elimination technique. It is based on the fact that each non-singular matrix A can be written (after pivoting eventually) as: $A = LU$ where L is a lower triangular matrix with a unit main diagonal and U is an upper triangular matrix (see Strang (1976)). If $A\mathbf{u} = f$ is the system to be solved then it can be decomposed into:

$$U\mathbf{u} = \mathbf{y} \tag{2.77}$$

$$L\mathbf{y} = f \tag{2.78}$$

(2.78) can be solved directly since L is a lower triangular matrix and then (2.77) can be solved starting from the bottom. Further, if A is symmetric the decomposition reads: $A = LDL^T$ where D is a diagonal matrix. If, in addition, A is also positive definite then:

$$A = GG^T \tag{2.79}$$

with G being a lower triangular matrix - the so called Cholesky decomposition. Thus, in that case, if A is not time (or iteration) dependent, only one lower triangular matrix is needed to be stored after the decomposition (2.79) is performed once. As it will be seen in the next section this can be exploited in many cases when convection-diffusion or Navier-Stokes equations are to be solved. This concerns, however, mainly 2-D problems because in the 3-D case the storage requirement of the spectral (element) Cholesky decomposition is unacceptable for most problems of practical interest. That is why some iterative methods with less storage requirements have to be used.

2.4.2 Iterative methods

A basic iterative scheme (Richardson iteration) is given by:

$$\text{Choose initial guess} \quad \mathbf{u}_0 \tag{2.80}$$

$$\mathbf{u}^k = \mathbf{u}^{k-1} + \alpha(f - S\mathbf{u}^{k-1}) \tag{2.81}$$

Here α is a relaxation parameter. An optimal value for it is given by:

$$\alpha^{opt} = \frac{2}{|\lambda_{min}| + |\lambda_{max}|} \tag{2.82}$$

with λ_{min} and λ_{max} being the minimum and maximum eigenvalues of the matrix A. It can be proven that the number of iterations to achieve certain accuracy is proportional to the conditioning number of the matrix defined by: $c(A) = \frac{\lambda_{max}}{\lambda_{min}}$. In the case of spectral approximations it increases (see Canuto et al., 1988) as $O(N^4)$ with N being the maximum number of nodes in each spatial direction. In the case of spectral element method it is experimentally found to be $O(K_e N^3)$ with K_e - the number of elements used. As a consequence of the extremely ill-conditioning, the iterative scheme converges very slow. The only way to avoid that difficulty is to improve the conditioning of the spectral (element) matrix. That is usually done by multiplying the linear system with the invert of a matrix (called preconditioner) having eigenvalues close to those of A. There are different ways to construct such a preconditioner. Most of them, however, are based on the idea to use the invert of the matrix F resulting of some kind of finite difference or finite element discretization of the partial differential equation on a grid consisting of the nodes of the spectral or spectral element mesh (see fig. 2.5). Such a matrix is called spectrally equivalent to A. Since it is based on the same points it can be expected to have eigenvalues closed to those of A. Further, the iterative algorithm can be applied to the resulting system:

$$F^{-1}A = F^{-1}f \tag{2.83}$$

It reads:

$$Fu^0 = f \tag{2.84}$$

$$Fu^k = Fu^{k-1} + \alpha(f - AU^{k-1}) \tag{2.85}$$

An example of the distribution of the eigenvalues of the non-preconditioned and preconditioned spectral element matrix resulting from the Poisson equation with Neumann

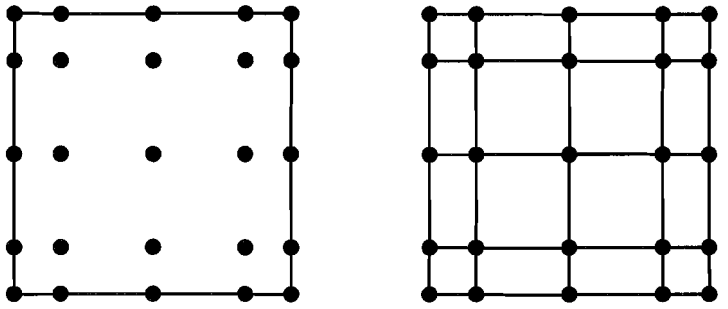

Figure 2.5: Spectral and corresponding finite element mesh.

boundary conditions in $[-1,1]^3$ is given in fig. 2.6. The decrease of the conditioning number due to the preconditioning is dramatic, and what is more important, it increases very slowly with the increase of the element order in the preconditioned case (see table 2.1).

The Richardson iteration has a convergence rate of order of $c(A)$. In case that the matrix A is symmetric and positive definite a substantial improvement can be achieved if a conjugate gradient or conjugate residual iteration technique is used. Their convergence rate is of order $\sqrt{c(A)}$. For an extensive description of these methods the reader is referred to Canuto et al. (1988).

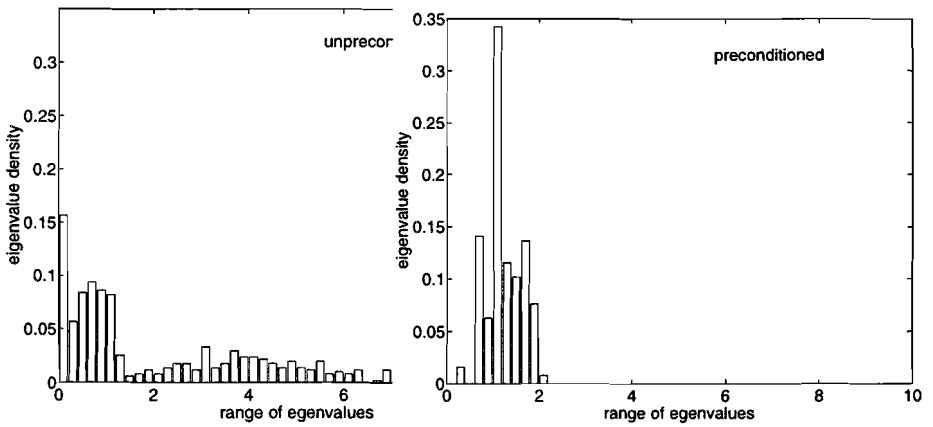

Figure 2.6: Block diagram of the eigenvalue density within certain range of the unpreconditioned (left) and preconditioned (right) SEM matrix using 1 element of 7-th order; Poisson equation with Neumann boundary conditions.

Table 2.1: Condition number for spectral elements of several orders

type	order of element		
	3	5	7
unpreconditioned	5862	42769	149000
preconditioned	4.88	5.19	5.57

2.5 Upwinding and other stabilization methods

2.5.1 Classical (finite difference) upwinding

As an example, the following classical differential equation is considered:

$$\begin{cases} -\dfrac{\partial^2 u}{\partial x^2} + \alpha \dfrac{\partial u}{\partial x} = 0 \qquad \text{in } \Omega = (0,1) \\ u(0) = 0 \\ u(1) = 1 \end{cases} \tag{2.86}$$

with the Peclet number $\alpha > 0$ and exact solution the monotonously increasing function:

$$u(x) = \frac{1 - e^{\alpha x}}{1 - e^{\alpha}} \tag{2.87}$$

If we choose a second order difference approximation and a central difference approximation for the first derivative a discrete version of (2.86) is:

$$\begin{cases} -\dfrac{u_{j-1} - 2u_j + u_{j+1}}{h^2} + \alpha \dfrac{u_{j+1} - u_{j-1}}{2h} = 0 \qquad j = 1, ..., N-1 \\ u_0 = 0 \\ u_N = 1 \end{cases} \tag{2.88}$$

The exact solution of this tri-diagonal system is given by:

$$u_i = \frac{1 - \delta^i}{1 - \delta^N} \qquad \text{with } \delta = \frac{1 + \frac{1}{2}h\alpha}{1 - \frac{1}{2}h\alpha} \tag{2.89}$$

If $\delta \geq 0$ or equivalently for $h < 2/\alpha$ the solution is monotonously increasing like the exact solution. For $\delta < 0$ and hereby $h > 2/\alpha$, however, the solution u_i behaves oscillatory (see figure 2.7). Note that the condition $h < 2/\alpha$ is nothing more than the requirement for diagonal dominance of the matrix to be inverted.
An often applied method to overcome the oscillatory behaviour of the solution is the use of a backward difference operator instead of the central difference operator:

$$\begin{cases} -\dfrac{u_{j-1} - 2u_j + u_{j+1}}{h^2} + \alpha \dfrac{u_j - u_{j-1}}{h} = 0 \qquad j = 1, ..., N-1 \\ u_0 = 0 \\ u_N = 1 \end{cases} \tag{2.90}$$

Now the resulting matrix is diagonal dominant for all $h > 0$ and no oscillations of the solution will occur. If, however, Taylor expansions are substituted in (2.90), we obtain for each collocation point x_j :

$$-\frac{\partial^2 u}{\partial x^2}|_{x_j} + \alpha \frac{\partial u}{\partial x}|_{x_j} - \frac{h\alpha}{2}\frac{\partial^2 u}{\partial x^2}|_{x_j} = \mathcal{O}(h^2) \tag{2.91}$$

In other words extra diffusion with magnitude $(h\alpha/2)$ is added to obtain a stable solution. The method then is only first order accurate and contains a mesh-depended diffusion. In the next sections this idea of adding extra diffusion is applied to Galerkin methods yielding the very popular streamline upwind methods described by Brooks and Hughes (1982) and Johnson (1987).

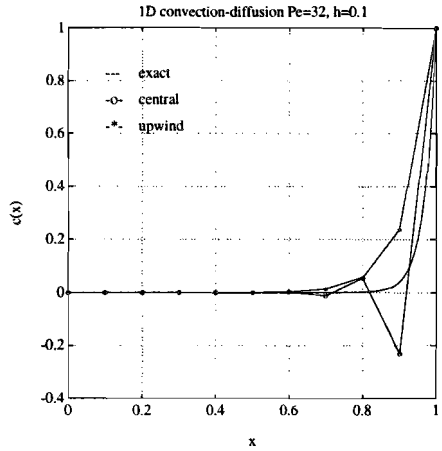

Figure 2.7: Exact and approximated solution for the 1D convection-diffusion equation with $\alpha = 32$ and $\Delta x = 0.1$.

2.5.2 Streamline upwind (SU) stabilization

Consider the convection diffusion equation:

$$\mathcal{L}u = \mathbf{v} \cdot \nabla u - \nabla \cdot \eta \nabla u = f \qquad \text{in } \Omega \tag{2.92}$$

with homogeneous Dirichlet conditions on the boundary of Ω. A standard Galerkin formulation of this problem is given by:

$$B(u, w)_\Omega - L(w)_\Omega = 0 \tag{2.93}$$

with:

$$\begin{aligned} B(u, w)_\Omega &= \int_\Omega ((\mathbf{v} \cdot \nabla u) w + \eta \nabla u \cdot \nabla w) d\Omega \\ L(w)_\Omega &= \int_\Omega f w d\Omega \end{aligned} \tag{2.94}$$

The same stability problems as described in the previous section can occur also if a Galerkin finite element method is used on too coarse grids. In order to overcome it Brooks and Hughes (1982) proposed to modify the weighting function according to:

$$\tilde{w} = w + \alpha \mathbf{v} \cdot \nabla w \tag{2.95}$$

in which α is a parameter that still has to be determined. In this way the information from upstream direction is weighted stronger (streamline upwinding). If this modified weighting is only applied to the convection term we obtain the streamline upwinding (SU) formulation:

$$B(u, w)_\Omega - L(w)_\omega + \int_\Omega \alpha (\mathbf{v} \cdot \nabla u)(\mathbf{v} \cdot \nabla w) d\Omega = 0 \tag{2.96}$$

In fact an extra term is added which adds extra diffusion in streamwise direction.

2.5.3 Streamline upwind Petrov Galerkin (SUPG) stabilization

A better way to use the modified weighting functions would be to apply them on the entire differential equation. This, however, introduces third order derivatives in the diffusion part of the equations and consequently demands more than C^0 continuity of the basisfunctions which is disadvantageous for domain decomposition methods like finite or spectral element methods. This can be avoided by introducing the modified weighting function on element level:

$$B(u,w)_\Omega = L(w)_\omega + \sum_e \int_{\Omega_e} \alpha(\mathcal{L}u - f)(\mathbf{v}\cdot\nabla w)d\Omega \tag{2.97}$$

Note that, in contradiction to the SU-formulation, the SUPG formulation is consistent since it involves the residual of the differential equation.

2.5.4 Galerkin least square (GLS) stabilization

Another, but based on the same idea, way to obtain stabilization is to modify the weighting functions according to:

$$\tilde{w} = w + \alpha\mathcal{L}w \tag{2.98}$$

In that case we obtain a Galerkin least squares (GLS) method:

$$B(u,w)_\Omega = L(w)_\omega + \sum_e \int_{\Omega_e} \alpha(\mathcal{L}u - f)(\mathcal{L}w - f)d\Omega \tag{2.99}$$

Disadvantage of these stabilization methods is that they introduce an extra parameter α which still has to be determined. Optimal values are given by (Johnson, 1987) but can not always be obtained easily. In the next section we will see that for time-dependent convection diffusion equations similar stabilizing terms can be obtained more naturally. For spectral and higher order spectral element methods it can be shown (Timmermans *et al.*, 1995) that the advantage of SUPG stabilization diminishes with increasing order of approximation.

2.6 Application of SEM to linear elasticity problems

The equilibrium between the stresses in the material and the external loading is expressed by:

$$\frac{\partial\rho\mathbf{u}}{\partial t} - \nabla\boldsymbol{\sigma} = \mathbf{F} \tag{2.100}$$

with $\boldsymbol{\sigma}$ being the stress tensor, $\mathbf{F}$ - a body force acting on an unit volume of the material and $\mathbf{u}$ - the displacement vector.
In order to express the stresses in displacements it is necessary to define a strain-displacement relation and the constitutive equations of the material, which define a relation between strains and stresses. A commonly used strain-displacement relation is:

$$\boldsymbol{\epsilon} = \mathbf{B}\mathbf{u} \tag{2.101}$$

where $\mathbf{B}$ represents the transpose of the divergence operator and $\boldsymbol{\epsilon}$ is the strain tensor.

The constitutive equations in case of linear elasticity read:

$$\sigma = D\epsilon \tag{2.102}$$

where **D** denotes the so-called elasticity matrix.
In 2D Cartesian coordinates the stress and strain tensors read:

$$\sigma = [\sigma_x, \sigma_y, \tau_{xy}]^T \tag{2.103}$$

$$\epsilon = [\frac{\partial u_x}{\partial x}, \frac{\partial u_y}{\partial y}, \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}]^T \tag{2.104}$$

In case of plain stress-isotropic material the elasticity matrix reads:

$$\mathbf{D} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-n}{2} \end{bmatrix}$$

where E denotes the Young's modulus and ν - the Poisson's ratio.
In most of the linear elasticity problems the resulting equations for the displacements are of elliptic type and thus are suitable for a spectral element treatment.
Some problems with the performance of SEM can be expected in the geometrically non-linear case. Than the deformation of the domain has to be taken into account which involves necessity of high order Jacobians during the computation of the mass matrix. If N-points GLL quadrature is used it is accurate for polynomials of $2N-1$ degree. The elements of the mass matrix are of $2N$ degree if the Jacobian is constant and thus the accuracy of its computation decreases rapidly with increasing the degree of the Jacobian. The empirical results show that it is not advisable to involve Jacobians of degree larger than 2 i.e. the sides of the spectral elements have to be at most second order curves.

Chapter 3

Temporal discretization of partial differential equations

3.1 Introduction

In this section some time integration methods are reviewed using the unsteady convection diffusion equation to illustrate them. Consider the convection and diffusion of a scalar function u for a divergence free velocity field $\mathbf{v}$:

$$\begin{aligned} &\frac{\partial u(\mathbf{x},t)}{\partial t} + (\mathbf{v}\cdot\nabla)u(\mathbf{x},t) - (\nabla\cdot\eta\nabla)u(\mathbf{x},t) = s \qquad \text{in } \Omega \\ &u(\mathbf{x},0) = u_0(\mathbf{x}) \end{aligned} \tag{3.1}$$

with η a diffusion constant and s some given source function. Note that for $u = \mathbf{v}$ this equation yields the non-linear convection diffusion equation known as Burger's equation which has a strong resemblance to the full Navier-Stokes equation for given pressure fields. After spatial discretization a semi-discrete version of (3.1) is:

$$\begin{aligned} &\mathbf{M}\dot{\mathbf{u}}(t) + \mathbf{N}(\mathbf{v})\mathbf{u}(t) + \mathbf{D}\mathbf{u}(t) = \mathbf{s} \\ &\mathbf{u}(0) = \mathbf{u}^0 \end{aligned} \tag{3.2}$$

where $\mathbf{u}(t)$ is the spatial approximation to $u(\mathbf{x},t)$, $\mathbf{N}(\mathbf{v})$ a discrete (eventually linearized) convection operator and $\mathbf{D}$ a discrete diffusion operator. $\mathbf{M}$ is the mass matrix which in finite difference methods is equal to the identity matrix.
If we combine the convection and diffusion operator and make use of the fact that the mass matrix can be inverted we obtain:

$$\begin{aligned} &\dot{\mathbf{u}}(t) = \mathbf{A}\mathbf{u}(t) + \mathbf{f} \\ &\mathbf{u}(0) = \mathbf{u}^0 \end{aligned} \tag{3.3}$$

with $\mathbf{A}(N\times N) = -\mathbf{M}^{-1}(\mathbf{N}+\mathbf{D})$ and $\mathbf{f} = \mathbf{M}^{-1}\mathbf{s}$. If we assume that $\mathbf{A}$ is non-defect, i.e. has N linear independent eigenvectors, a non-singular matrix $\mathbf{B}$ with complex coefficients exists defined by:

$$\mathbf{AB} = \mathbf{B}\Lambda \tag{3.4}$$

with $\Lambda = diag(\lambda_1, ..., \lambda_N)$ and λ_i the eigenvalues of $\mathbf{A}$.

The differential equation (and also its semi-discretized version is called to be stable when a finite error ε_0 in the initial condition u_0 results in a finite error $\varepsilon(t)$ in $u(t)$ for any t. If $\mathbf{u}(t)$ is the solution of (3.3) for the initial condition $\mathbf{u}(0) = \mathbf{u}^0$ and $\tilde{\mathbf{u}}(t)$ the solution for initial condition $\tilde{\mathbf{u}}(0) = \mathbf{u}^0 + \boldsymbol{\varepsilon}^0$, then if $\boldsymbol{\varepsilon} = \tilde{\mathbf{u}} - \mathbf{u}$ we have:

$$\begin{aligned} \dot{\boldsymbol{\varepsilon}} &= \mathbf{A}\boldsymbol{\varepsilon} \\ \boldsymbol{\varepsilon}(0) &= \boldsymbol{\varepsilon}^0 \end{aligned} \tag{3.5}$$

Since $\mathbf{B}$ is non-singular a variable $\boldsymbol{\eta}$ can be defined such that $\boldsymbol{\eta} = \mathbf{B}^{-1}\boldsymbol{\varepsilon}$ and the following equation holds:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{\Lambda}\boldsymbol{\eta} \\ \boldsymbol{\eta}(0) &= \mathbf{B}^{-1}\boldsymbol{\varepsilon}^0 = \boldsymbol{\eta}^0 \end{aligned} \tag{3.6}$$

The solution of this set is:

$$\eta_i = \eta_i^0 e^{\lambda_i t} \tag{3.7}$$

The differential equation is stable if for all i, η_i is a non-increasing function in time, hence if:

$$Re[\lambda_i] \leq 0 \qquad \text{for all } i = 1, ..., N \tag{3.8}$$

In other words, all eigenvalues of $\mathbf{A}$ must be non-positive. As will be shown in the next sections, time integration of the semi-discrete set of equations (3.2) will generally lead to the form:

$$\boldsymbol{\eta}^{n+1} = \mathbf{G}\boldsymbol{\eta}^n \tag{3.9}$$

with $\mathbf{G}$ the multiplication matrix of the error $\boldsymbol{\eta}$. Stability of the time discretization scheme will require that $||\mathbf{G}|| \leq 1$. The multiplication matrix $\mathbf{G}$ depends on the eigenvalues of $\mathbf{A}$ and hereby on the order of approximation N.

Eigenvalues of the diffusion and convection operator In Canuto *et al.* (1988) it is shown that for spectral methods the eigenvalues of the diffusion operator are negative and real and satisfy $\lambda = \mathcal{O}(N^4)$, with N being the order of approximation. For spectral elements empirically a growth of $\mathcal{O}(n_e N^3)$ (n_e being the number of elements) is found, whereas for low order finite elements and finite difference methods the eigenvalues of the diffusion operator globally grow with the number of collocation points like $\mathcal{O}(N^2)$. For the convective operator (yielding a non-symmetric set of discrete equations) the eigenvalues will have an imaginary part. The real parts are strictly negative and both the real and imaginary part of the largest eigenvalues grow like $\mathcal{O}(N^2)$ for spectral methods. Roughly spoken, the eigenvalues are located as indicated in figure 3.1.

3.2 Standard implicit time integration methods

Implicit time integration methods are methods that contain a matrix vector evaluation of the unknowns at the new time level $(n+1)$. As a consequence they demand to solve an algebraic system at each time step. Although this seems to be very costly, the superior stability properties of implicit methods make them useful for many applications. The two most important families of implicit methods are given below.

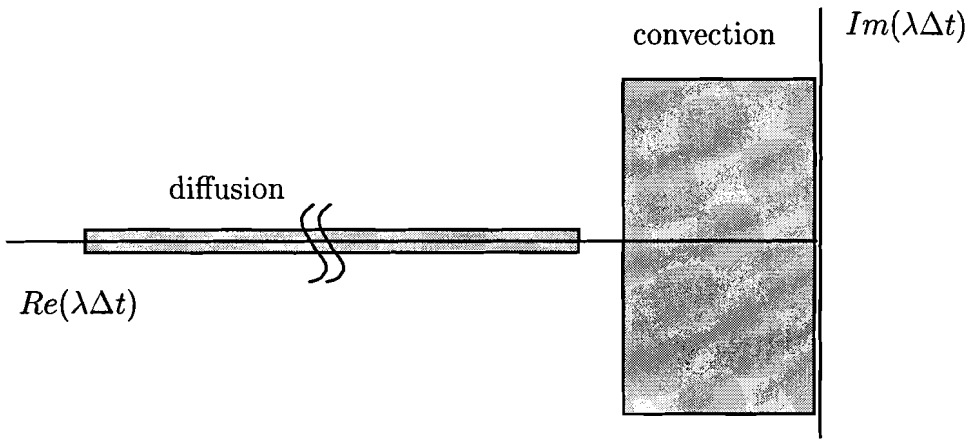

Figure 3.1: Location of eigenvalues of convection and diffusion operators

3.2.1 Adams-Moulton time integration schemes

A set of implicit methods are the Adams-Moulton methods defined by:

$$\mathbf{M}\mathbf{u}^{n+1} = \mathbf{M}\mathbf{u}^{n} + \Delta t \sum_{i=1}^{k} \beta_i \mathbf{A}^{n+2-i}\mathbf{u}^{n+2-i} \tag{3.10}$$

Table 3.1: Adams-Moulton schemes

k			β_1	β_2	β_3
1	Euler Implicit	EI	1	-	-
2	Crank-Nicolson	CN	1/2	1/2	-
3	Adams-Moulton	AM3	5/12	8/12	−1/12

The stability areas can easily be computed by substitution of $\mathbf{u}^{n+1} = \mathbf{G}\mathbf{u}^{n}$ in (3.10). This will result in an polynomial equation for $\mathbf{G}$ which can be solved as a function of the eigenvalue and the time step (i.e. $\lambda\Delta t$). Plots that are given are contour values of $||\mathbf{G}||$ at level $||\mathbf{G}|| = 1$.
As can be seen from figure 3.2 the Euler implicit and Crank-Nicolson schemes are unconditionally stable whereas the AM3-scheme is only conditionally stable. This means that the time step Δt must be chosen small enough to ensure that $\lambda\Delta t$ is located in the stability region of the method for all eigenvalues of the system. The region for which the methods are stable are indicated with the arrows. Both the Euler implicit and Crank-Nicolson schemes (or variants of them) are widely used due to there good stability properties.

3.2.2 Backward differencing time integration schemes

A second set of implicit methods are the backward-differencing methods defined by:

$$(\beta_0\mathbf{M} + \Delta t\mathbf{A})\mathbf{u}^{n+1} = \sum_{i=1}^{k} \beta_i \mathbf{M}\mathbf{u}^{n+1-i} \tag{3.11}$$

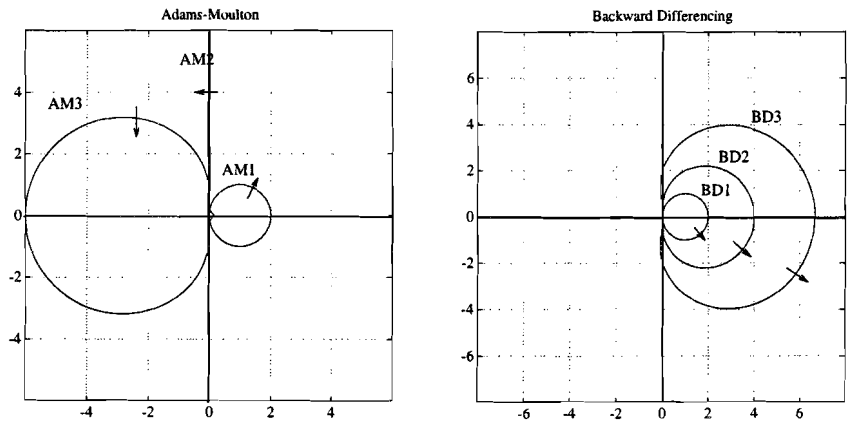

Figure 3.2: Stability areas of Adams-Moulton and backward-differencing schemes

Table 3.2: Backward-Differencing schemes

k			β_0	β_1	β_2	β_3
1	Euler Implicit	EI	1	1	-	-
2	Backward Differencing	BD2	3/2	2	$-1/2$	-
3	Backward Differencing	BD3	11/6	3	$-3/2$	1/3

The stability areas again can be computed by substitution of $u^{k+1} = \mathbf{G}u^k$ in the left hand side of (3.10).
Note that the backward differencing schemes are stable outside the regions defined by the closed contours. As can be seen from the figure only the Euler implicit (=BD1) scheme is unconditionally stable. All the other backward differencing schemes have a small region near the imaginary axis for which they are unstable. Using the information given in figure 3.1 it can be expected that higher order backward differencing can be used for diffusion equations but may give stability problems if convective forces become dominant.

3.3 Standard explicit time integration methods

In explicit time integration methods (two important sets are given below) the elliptic part of the equation is only evaluated at previous time levels and no matrix inversion or only a trivial matrix inversion of the mass matrix is required. As a consequence the time marching can be performed very efficiently. However, the severe restrictions imposed by the stability properties of explicit methods often cancel this advantage completely. Note that methods that are explicit in combination with a finite difference or finite volume space discretization (diagonal mass matrix) can hardly be called explicit in case of a Galerkin space discretization method since the inversion of the (non-diagonal) mass matrix is still required. In many cases lumping of the mass matrix (for instance by applying Gauss-Lobatto integration) is used. Especially for low order methods this, however, will result in a unacceptable loss of accuracy.

3.3.1 Adams-Bashforth time integration schemes

A first set of explicit methods are given by the Adams-Bashforth schemes, which can be written as:

$$\mathbf{M}\mathbf{u}^{n+1} = \mathbf{M}\mathbf{u}^{n} + \Delta t \sum_{i=1}^{k} \beta_i \mathbf{A}^{n+1-i}\mathbf{u}^{n+1-i} \tag{3.12}$$

Table 3.3: Adams-Bashforth schemes

k			β_1	β_2	β_3
1	Euler Explicit	EI	1	-	-
2	Adams-Bashforth	AB2	3/2	$-1/2$	-
3	Adams-Bashforth	AB3	23/12	$-16/12$	5/12

The stability areas again can easily be computed by substitution of $\mathbf{u}^{k+1} = \mathbf{G}\mathbf{u}^{k}$ in (3.10). All Adams-Bashforth schemes are conditionally stable and only third and higher order versions include a part of the imaginary axis. This makes Adams-Bashforth schemes almost exclusively appropriate for convection dominated problems. Often, in convection diffusion problems, third or higher order Adams-Bashforth methods are used to linearize (in time) the convection operator and are combined with implicit methods for the diffusion operator (see also section 3.5).

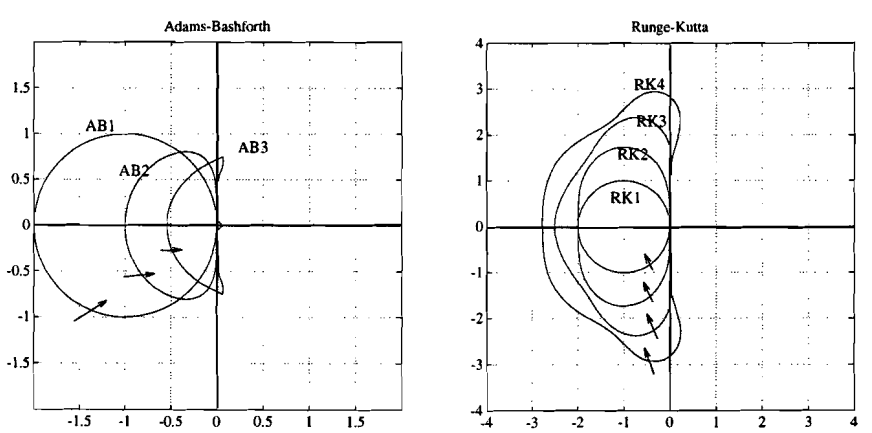

Figure 3.3: Stability areas of Adams-Bashforth and Runge-Kutta schemes

3.3.2 Runge-Kutta time integration schemes

Another set of explicit time integration methods are formed by the explicit Runge-Kutta time discretizations. An important class of Runge-Kutta schemes are given by:

$$\begin{cases} \mathbf{M}\mathbf{u}^{n+\frac{1}{k}} = \mathbf{M}\mathbf{u}^{n} + \frac{\Delta t}{k}\mathbf{A}^{n}\mathbf{u}^{n} \\ \\ \mathbf{M}\mathbf{u}^{n+\frac{1}{k-i}} = \mathbf{M}\mathbf{u}^{n} + \frac{\Delta t}{k-i}\mathbf{A}^{n+\frac{1}{k-i-1}}\mathbf{u}^{n+\frac{1}{k-i-1}} \qquad i = 1, ..., k-1 \end{cases} \tag{3.13}$$

Note that for $k = 1$ the Runge-Kutta method reduces to an Euler explicit method. The absolute stability areas are given in figure 3.3. As distinct from the Adams-Bashforth schemes, the stability regions expand with increasing order. Also here only third and higher order schemes include a part of the imaginary axis.

3.4 Taylor-Galerkin methods

In the previous subsections classical time discretization methods for sets of ordinary differential equations were applied to the semi-space-discretized equations. This procedure is often referred to as the method of lines. In this section we will apply first a time discretization and after that the space discretization. It will be shown that in case of convection diffusion equations this can lead to favorable stability properties. Consider the general non-linear form of the convection diffusion equation:

$$\frac{\partial u}{\partial t} = \mathcal{D}u - \nabla \cdot \mathbf{s}(u) \tag{3.14}$$

Here $\mathcal{D}u$ can contain diffusive but also other terms.

3.4.1 Explicit Taylor-Galerkin schemes

Point of departure is the Taylor expansion:

$$u^{n+1} = u^n + \Delta t \frac{\partial u}{\partial t}|_{t_n} + \tfrac{1}{2}\Delta t^2 \frac{\partial^2 u}{\partial t^2}|_{t_n} + \mathcal{O}(\Delta t^3) \tag{3.15}$$

Substitution of the original differential equation (3.14) yields:

$$\begin{aligned} \frac{u^{n+1} - u^n}{\Delta t} &= \mathcal{D}u|_{t_n} - \nabla \cdot s(u)|_{t_n} + \frac{\Delta t}{2}\frac{\partial}{\partial t}\left(\mathcal{D}u - \nabla \cdot \mathbf{s}(u)\right)_{t_n} + \mathcal{O}(\Delta t^2) \\ &= \mathcal{D}u|_{t_n} + \frac{\Delta t}{2}\frac{\partial}{\partial t}\mathcal{D}u|_{t_n} - \nabla \cdot \mathbf{s}(u)|_{t_n} - \frac{\Delta t}{2}\nabla \cdot \left(\frac{\partial \mathbf{s}(u)}{\partial u}\frac{\partial u}{\partial t}\right)_{t_n} \end{aligned} \tag{3.16}$$

And thus:

$$\frac{u^{n+1} - u^n}{\Delta t} = \tfrac{1}{2}(\mathcal{D}u|_{t_n} + \mathcal{D}u|_{t_{n+1}}) - \nabla \cdot s(u)|_{t_n} - \tfrac{1}{2}\Delta t \nabla \cdot \left(\frac{\partial \mathbf{s}}{\partial u}\left(\mathcal{D}u - \nabla \cdot \mathbf{s}(u)\right)\right) \tag{3.17}$$

Also higher order methods can be derived by subsequent substitution of the original differential equation. Mostly this will lead to relative complex and not always better schemes (Donea and Quartapelle, 1992). For the linear convection-diffusion equation $\mathbf{s}(u) = \mathbf{v}u$ and using $\nabla \cdot \mathbf{v} = 0$ this reduces to:

$$\frac{u^{n+1} - u^n}{\Delta t} = \tfrac{1}{2}(\mathcal{D}u|_{t_n} + \mathcal{D}u|_{t_{n+1}}) - \mathbf{v} \cdot \nabla u|_{t_n} - \tfrac{1}{2}\Delta t \mathbf{v} \cdot \nabla(\mathcal{D}u - \mathbf{v} \cdot \nabla u)_{t_n} \tag{3.18}$$

Note that we have a Crank-Nicolson based discretization of the diffusion term and an explicit discretization of the convection term. Moreover, the last term has the properties of a diffusion force and will stabilize the scheme. A disadvantage is that the combination $\nabla \mathcal{D}u$ contains third order space derivatives and demands high order regularity of the space

discretization methods that will be applied. For pure convection, however, only second order space derivatives are involved and an extra diffusion is introduced according to:

$$\tfrac{1}{2}\Delta t \mathbf{v}^2 \nabla^2 u \tag{3.19}$$

this strongly resembles the terms that are introduced in streamline upwinding techniques. Only here the coefficient $\frac{1}{2}\Delta t$ naturally follows from the discretization scheme.

3.4.2 Implicit Taylor-Galerkin schemes

Point of departure is the Taylor expansion:

$$u^{n+1} = u^n - \Delta t \frac{\partial u}{\partial t}|_{t_{n+1}} + \tfrac{1}{2}\Delta t^2 \frac{\partial^2 u}{\partial t^2}|_{t_{n+1}} - \mathcal{O}(\Delta t^3) \tag{3.20}$$

Substitution in the differential equation yields:
And thus similar as in the explicit Taylor-Galerkin method we have:

$$\frac{u^{n+1} - u^n}{\Delta t} = \tfrac{1}{2}(\mathcal{D}u|_{t_n} + \mathcal{D}u|_{t_{n+1}}) - \mathbf{v} \cdot \nabla u|_{t_{n+1}} + \tfrac{1}{2}\Delta t \mathbf{v} \cdot \nabla(\mathcal{D}u - \mathbf{v} \cdot \nabla u)_{t_{n+1}} \tag{3.21}$$

Due to the diffusion introduced by this scheme and the implicit treatment of the convection term, superior stability properties are obtained for convection dominated problems without unacceptable loss of accuracy (Donea and Quartapelle, 1992).

3.5 Operator splitting

From the previous sections we learned that diffusion dominated differential equations will give rise to eigenvalues along the negative real axis of the complex $\lambda\Delta t$-plane. They are proportional to the invert of the Reynolds number. Consequently, if an explicit time integration is performed the restriction on the time step becomes unacceptable even for relatively large Reynolds numbers. An alternative option is to use some implicit methods although at each time step a matrix has to be inverted. If the diffusion operator is time-independent and the LU-decomposition of the matrix (see section 4) can be stored then the system can be efficiently solved by means of a direct method. In many cases the convective part of the differential equation introduces time dependence of the matrix involved (for instance for time-dependent velocity fields) and the fully implicit methods become very inefficient. A way to avoid this is to use a combination of explicit and implicit time integration for the different operators involved. More general, it is possible to apply an operator splitting technique (Maday *et al.*, 1990) that enables any combination of time integration schemes for the different operators the original equation contains.
As an example we will treat the unsteady convection-diffusion problems by an operator splitting technique in which the problem is decomposed in a pure convection problem and a pure diffusion problem (Timmermans *et al.*, 1994). Both problems are then solved by suitable time-integrations with different time-steps, if necessary.
Thereto the convection-diffusion problem is rewritten as follows

$$\frac{\partial \mathbf{c}}{\partial t} = \mathcal{D}(\mathbf{c}) + \mathcal{C}(\mathbf{c}) + \mathbf{f}, \tag{3.22}$$

where $\mathcal{D}(\mathbf{c}) = (\nabla \cdot \eta\nabla)\mathbf{c}$ is the diffusion operator and $\mathcal{C}(\mathbf{c}) = -(\mathbf{u} \cdot \nabla)\mathbf{c}$ is the convection operator. Following the idea of Maday *et al.*, equation (3.22) is written in terms of an integrating factor in $\mathcal{C}$

$$\frac{\partial}{\partial t}\left(\mathcal{Q}_{\mathcal{C}}^{(t^*,t)}\mathbf{c}(t)\right) = \mathcal{Q}_{\mathcal{C}}^{(t^*,t)}(\mathcal{D}(c) + \mathbf{f}), \tag{3.23}$$

with t^* an arbitrary fixed time. The integrating factor $\mathcal{Q}_{\mathcal{C}}^{(t^*,t)}$ is defined by the initial-value problem:

$$\frac{\partial}{\partial t}\mathcal{Q}_{\mathcal{C}}^{(t^*,t)} = -\mathcal{Q}_{\mathcal{C}}^{(t^*,t)}\mathcal{C}(c), \quad \mathcal{Q}_{\mathcal{C}}^{(t^*,t^*)} = \mathcal{I}, \tag{3.24}$$

where $\mathcal{I}$ is the identity operator. Equation (3.23) is integrated by a suitable time-integration for the diffusion operator $\mathcal{D}(c)$. A useful class of $A(\alpha)$-stable time-integration methods is given by the backward differences formulae. These schemes are accurate for all components around the origin in the stability diagram and absolutely stable away from the origin in the left imaginary plane. Thus, it is possible to use high-order backward differences schemes without the severe constraints on the time-step that are needed for general high-order multistep schemes like the Adams–Moulton methods, which are not $A(\alpha)$-stable for any order higher than 2.
Application of a backward differences scheme to equation (3.23) gives the following semi-discrete system

$$\frac{\gamma_0\mathbf{c}^{n+1} - \sum_{i=1}^{k}\beta_i\mathcal{Q}_{\mathcal{C}}^{(t^{n+1-i},t^{n+1})}\mathbf{c}^{n+1-i}}{\Delta t} = \mathcal{D}(\mathbf{c}^{n+1}) + \mathbf{f}^{n+1}, \tag{3.25}$$

where e.g. the superscript $n+1$ denotes the approximation at time $t^{n+1} = (n+1)\Delta t$ with Δt the time-step. For consistency it is required that

$$\gamma_0 = \sum_{i=1}^{k}\beta_i. \tag{3.26}$$

The coefficients of the first-order scheme ($k = 1$), which is in fact a backward Euler scheme, are $\gamma_0 = 1, \beta_1 = 1$. For the second-order scheme ($k = 2$) they read $\gamma_0 = \frac{3}{2}, \beta_1 = 2, \beta_2 = -\frac{1}{2}$. To evaluate the terms $\mathcal{Q}_{\mathcal{C}}^{(t^{n+1},t^{n+1-i})}\mathbf{c}^{n+1-i}(i = 1, 2, \dots)$ the following associated initial value problem is solved

$$\begin{cases} \dfrac{\partial\tilde{\mathbf{c}}(s)}{\partial s} = \mathcal{C}(\tilde{\mathbf{c}})(s), \quad 0 < s < i\Delta t, \\ \tilde{\mathbf{c}}(0) = \mathbf{c}^{n+1-i}, \end{cases} \tag{3.27}$$

from which it then follows that

$$\mathcal{Q}_{\mathcal{C}}^{(t^{n+1-i},t^{n+1})}\mathbf{c}^{n+1-i} = \tilde{\mathbf{c}}(i\Delta t). \tag{3.28}$$

Problem (3.27), accounting for the convection part, can be solved using a suitable (and preferably explicit) scheme with a time-step Δs which can be taken different from Δt. Note that the integrating factor $\mathcal{Q}_{\mathcal{C}}^{(t^{n+1},t^{n+1-i})}$ is in fact never constructed explicitly; rather, the 'action' of the integrating factor is evaluated through solution of the associated convection problem (3.27).

Remark

An alternative approach for the diffusion step is to use the θ-method or the trapezoidal method. The semi-discrete equation for the diffusion operator then becomes

$$\frac{\mathbf{c}^{n+1} - \mathcal{Q}_C^{(t^{n+1},t^n)}\mathbf{c}^n}{\Delta t} = \theta(\mathcal{D}(\mathbf{c}^{n+1}) + \mathbf{f}^{n+1}) + (1-\theta)\mathcal{Q}_C^{(t^{n+1},t^n)}(\mathcal{D}(\mathbf{c}^n) + \mathbf{f}^n). \tag{3.29}$$

The terms $\mathcal{Q}_C^{(t^{n+1},t^n)}\mathbf{c}^n$ and $\mathcal{Q}_C^{(t^{n+1},t^n)}(\mathcal{D}(\mathbf{c}^n)+\mathbf{f}^n)$ are calculated according to a convection problem similar to (3.27).
For $\theta = \frac{1}{2}$ this scheme results in a second-order accurate Crank-Nicolson method. This scheme is commonly used for diffusion problems. In Navier–Stokes calculations it is frequently applied to the viscous and pressure terms. Although the Crank–Nicolson scheme is $A(\alpha)$-stable for such terms, it has the disadvantage that it damps high frequency components very weakly, whereas these components in reality decay very rapidly. In cases where this is undesirable, a possible strategy is to use $\theta = \frac{1}{2} + \delta\Delta t$, where δ is a small positive constant. This method damps all components of the solution and is formally second-order in time.

3.6 Application of SEM to convection and convection diffusion problems

Here some test problems solved by means of the spectral element method will be presented. The time integration is performed by means of Euler explicit Taylor-Galerkin (EETG) scheme, Crank-Nicolson scheme and and a 2-step version of the EETG scheme given by:

$$\mathbf{u}^{k+1/2} = \mathbf{u}^k + \frac{\Delta t}{2}\mathbf{C}^{k+1/2}\mathbf{u}^k \tag{3.30}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \Delta t\mathbf{C}^{k+1}\mathbf{u}^{k+1/2} \tag{3.31}$$

for the equation:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{C}(t)\mathbf{u} \tag{3.32}$$

This scheme can be regarded also as a 2-step Runge-Kutta scheme. In case of convection-diffusion problems the operator-splitting approach is used. The results are originally provided in (Timmermans and van de Vosse, 1993) and (Timmermans *et al.*, 1994)

3.6.1 One-dimensional linear convection

Consider a one-dimensional test case the convection of a Gaussian hill described by:

$$c(x,t) = e^{-\frac{(x-x_0-ut)^2}{2\sigma^2}}. \tag{3.33}$$

The initial hill ($t = 0$) is centered around $x_0 = 0.15$ and has a standard deviation of $\sigma = 0.04$. The hill is convected with constant velocity $u = 1$ and $t \in [0, 0.6]$ according to the equation:

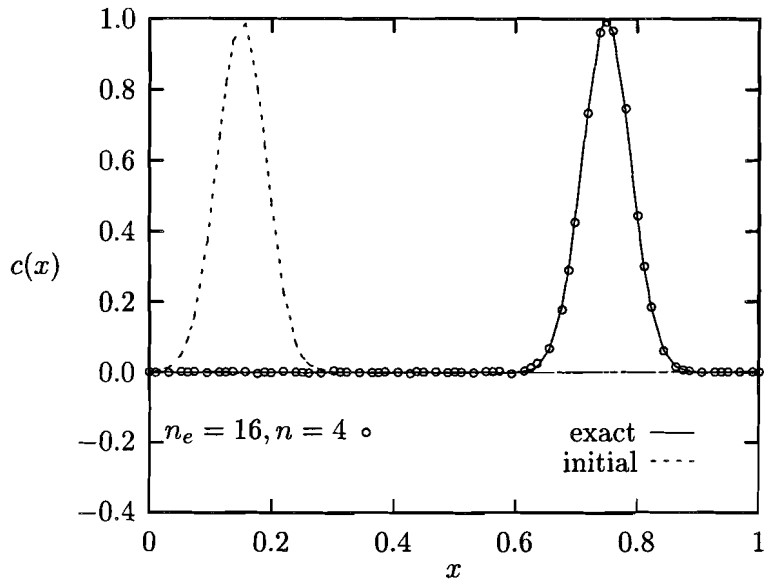

Figure 3.4: Convection of a Gaussian hill; exact solution and two-step EETG approximation for $n_e = 16, n = 4$ with 256 time-steps

$$\frac{\partial \mathbf{c}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{c} \quad \text{in} \quad \mathbf{\Omega} = [0,1] \tag{3.34}$$

For this problem the Taylor–Galerkin schemes for linear convection are compared with a Crank–Nicolson time-integration. The spatial discretization is a spectral element one using $n_e = 16$ elements of degree of approximation $n = 2, 4$ and 8. The discrete maximum error $\varepsilon = \|c - c_h\|_{\infty,gl}$ for these cases is given in table 3.4. Here c_h denotes the approximate solution and the subscript ∞, gl means that the maximum error is evaluated in the Gauss–Lobatto points of the spectral element approximation. The exact solution and the approximation for $n_e = 16, n = 4$ for 256 time-steps is shown in fig. 3.4.

Note also that the solution becomes much more accurate if the degree of approximation increases. The Crank–Nicolson scheme is only slightly more accurate. All schemes show second-order accuracy if the degree of the approximation is large enough. Taking into account that the explicit Taylor–Galerkin schemes require far less processing time, it is obvious that it is in fact preferable for this problem.

3.6.2 One-dimensional non-linear convection

Consider the one-dimensional non-linear Burgers equation with zero diffusion:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} \quad \text{in}\mathbf{\Omega}, \tag{3.35}$$

Table 3.4: Discrete maximum error $\|c - c_h\|_{\infty,gl}$ for the convection of a Gaussian hill; $n_e = 16$ elements with varying degree of approximation n

method	n	number of time-steps			
		128	256	512	1024
two-step EETG	2	$0.20\ 10^{0}$	$0.21\ 10^{0}$	$0.21\ 10^{0}$	$0.21\ 10^{0}$
	4	$0.44\ 10^{-1}$	$0.10\ 10^{-1}$	$0.90\ 10^{-2}$	$0.91\ 10^{-2}$
	8	unstable	unstable	$0.30\ 10^{-2}$	$0.74\ 10^{-3}$
one-step EETG	2	$0.16\ 10^{0}$	$0.19\ 10^{0}$	$0.20\ 10^{0}$	$0.21\ 10^{0}$
	4	$0.47\ 10^{-1}$	$0.10\ 10^{-1}$	$0.77\ 10^{-2}$	$0.84\ 10^{-2}$
	8	unstable	$0.12\ 10^{-1}$	$0.30\ 10^{-2}$	$0.74\ 10^{-3}$
Crank–Nicolson	2	$0.22\ 10^{0}$	$0.22\ 10^{0}$	$0.21\ 10^{0}$	$0.21\ 10^{0}$
	4	$0.30\ 10^{-1}$	$0.13\ 10^{-1}$	$0.93\ 10^{-2}$	$0.92\ 10^{-2}$
	8	$0.24\ 10^{-1}$	$0.59\ 10^{-2}$	$0.15\ 10^{-2}$	$0.37\ 10^{-3}$

Table 3.5: Relative discrete maximum error $\|u - u_h\|_{\infty,gl}/0.02$ for the Burgers problem; $n_e = 16$ elements of with varying degree of approximation n

method	n	number of time-steps			
		128	256	512	1024
two-step EETG	2	$0.92\ 10^{-1}$	$0.99\ 10^{-1}$	$0.10\ 10^{0}$	$0.10\ 10^{0}$
	4	$0.21\ 10^{-1}$	$0.14\ 10^{-1}$	$0.11\ 10^{-1}$	$0.11\ 10^{-1}$
	8	unstable	unstable	$0.33\ 10^{-2}$	$0.16\ 10^{-2}$
Crank–Nicolson	2	$0.11\ 10^{0}$	$0.10\ 10^{0}$	$0.10\ 10^{0}$	$0.10\ 10^{0}$
	4	$0.22\ 10^{-1}$	$0.15\ 10^{-1}$	$0.12\ 10^{-1}$	$0.12\ 10^{-1}$
	8	$0.15\ 10^{-1}$	$0.57\ 10^{-2}$	$0.29\ 10^{-2}$	$0.18\ 10^{-2}$

in the domain $\Omega = (0, 4)$ and $t \in [0, 2]$. The initial condition is given by

$$u(x,0) = g(x) = \begin{cases} a - b\cos(2\pi x), & 0 \le x \le 1, \\ a - b, & \text{elsewhere}, \end{cases} \tag{3.36}$$

with $a = 1, b = 0.01$. The boundary conditions are given by

$$u(0,t) = u(4,t) = a - b. \tag{3.37}$$

The exact solution to this problem is given by Whitham (1974)

$$u(x,t) = g(y), \quad x = y + u(g(y))t. \tag{3.38}$$

For this initial solution no shock arises in the given time-segment.

This non-linear problem is solved with the explicit two-step EETG scheme and compared to a time-linearized Crank–Nicolson scheme. Since the boundary conditions are non-homogeneous, for this case the two-step scheme is easier to implement than the one-step EETG scheme which involves the evaluation of a boundary integral. The spectral element method uses the same number of elements and degree of approximation as in the linear

case. Since the solution only varies over an interval of 0.02, the numerical solution is verified with respect to the following relative error

$$\varepsilon = \frac{\|u - u_h\|_{\infty, gl}}{0.02}. \tag{3.39}$$

The results for the relative error of the three different spectral element discretizations are shown in table 3.5.
For non-linear convection the results are quite the same as for the linear convection problem, although no second-order accuracy is achieved due to the non-linearity. The two-step EETG scheme is quite comparable in accuracy to the Crank–Nicolson scheme. Again, for an increasing degree of approximation the solution becomes much more accurate; but then also more time-steps are needed to obtain a stable numerical scheme. However, as was already stated in the linear convection case, due to the efficiency of the two-step scheme it is more suited for this problem than the Crank–Nicolson method.

3.6.3 One-dimensional unsteady strongly non-linear convection problem

Consider in this section the strongly non-linear convection problem as described by:

$$\frac{\partial c}{\partial t} + \mathbf{u}(c)\frac{\partial c}{\partial x} = 0 \quad \text{in}[0,2] \tag{3.40}$$

$$c(0) = c(2) = 0.5 \tag{3.41}$$

$$\mathbf{u}(c) = 5c^4 \tag{3.42}$$

with an initial condition:

$$c(x,0) = \begin{cases} 1 - 0.5cos(2\pi x) & \text{if} \quad x \in [0,1] \\ 0.5 \quad \text{elsewhere} \end{cases} \tag{3.43}$$

which describes the convection of a shock. For this non-linear problem implicit time-integration proves to be necessary. However, in this case, the stabilization of the second-order Taylor-Galerkin methods must also be applied. The most stable scheme appears to be the IETG scheme. Application to this particular problem using again a linearization in time of the implicit non-linear advective term gives

$$\left(\mathbf{M} + \Delta t \mathbf{N}(\mathbf{c}^n) - \frac{\Delta t^2}{2}\mathbf{S}_{TG}(\mathbf{c}^n)\right)\mathbf{c}^{n+1} = \mathbf{M}\mathbf{c}^n, \tag{3.44}$$

where $\mathbf{S}_{TG}(\mathbf{c}^n)$ denotes the diffusion matrix $\mathbf{S}$ with coefficient $\eta = \mathbf{u}(\mathbf{c}^n)^2$.
In fig. 3.5 (left-top) a spectral element solution ($n_e = 32, n = 4$) for the strong non-linear advection problem is given using the IETG method with 128 time steps. It is clearly seen that the shock has not traveled far enough. Obviously the explicit time-linearization of $u(c)$ is not accurate enough. Significantly better results are obtained if a simple Picard iteration at each time step is performed. Fig. 3.5 (right-top) shows that the shock now is transported quite accurate. As can be seen in fig. 3.5 even better results can be obtained using higher-order approximations ($n_e = 32, n = 8$ (left-bottom), $n_e = 32, n = 16$ (right-bottom)).

3.6.4 Two-dimensional linear convection

In more dimensions the choice of the time-integration becomes more and more important with respect to efficiency. From the previous sections it appears that the two-step EETG

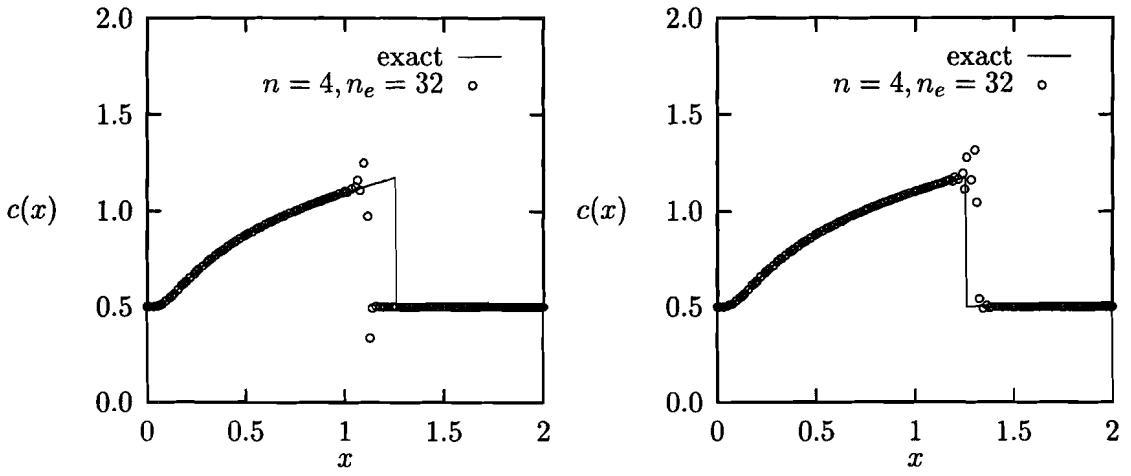

Table 3.6: Discrete maximum error $\|c - c_h\|_{\infty,gl}$ for the rotation of a Gaussian hill; number of elements $n_e = 4$ fixed with varying degree of approximation n

time-steps	$n = 4$	$n = 8$	$n = 12$	$n = 16$
256	$0.33\ 10^{0}$	$0.67\ 10^{-1}$	$0.17\ 10^{-1}$	unstable
512	$0.33\ 10^{0}$	$0.67\ 10^{-1}$	$0.29\ 10^{-2}$	$0.29\ 10^{-2}$
1024	$0.33\ 10^{0}$	$0.67\ 10^{-1}$	$0.29\ 10^{-2}$	$0.33\ 10^{-3}$

scheme is the most suitable for large more-dimensional problems. In order to check the performance of the two-step scheme, consider the unsteady rotation of a Gaussian hill described by the convection equation in two dimensions with domain $\Omega = (-1,1)\times(-1,1)$ and $t \in [0, 0.5]$. The time-dependent velocity is given by

$$\mathbf{u}(\mathbf{x}, t) = [-\pi^2 \sin(2\pi t)x_2,\ \pi^2 \sin(2\pi t)x_1]^T. \tag{3.45}$$

The initial solution is given by

$$c(\mathbf{x}, t) = 0.01^{4\left((x_1+\frac{1}{2})^2+x_2^2\right)}. \tag{3.46}$$

It represents a smooth Gaussian hill with height equal to 1 and with radius equal to $\frac{1}{4}$ centered at $(-\frac{1}{2}, 0)$. At $t = 0.5$ the hill is rotated halfway without diffusion, and therefore without loss of shape.

The problem is solved using the two-step EETG scheme. Two types of convergence are examined. To check the p-convergence the number of elements is kept fixed at $n_e = 4$; the degree of approximation is varying ($n = 4, 8, 12, 16$). To check the h-convergence the degree of approximation is kept fixed at $n = 2$ and the number of elements varies ($n_e = 16, 64, 144, 256$). The total number of degrees of freedom in the corresponding discretizations is the same. The results for the discrete maximum error $\varepsilon = \|c - c_h\|_{\infty,gl}$ for the first discretization are given in table 3.6; for the second discretization they are found in table 3.6.

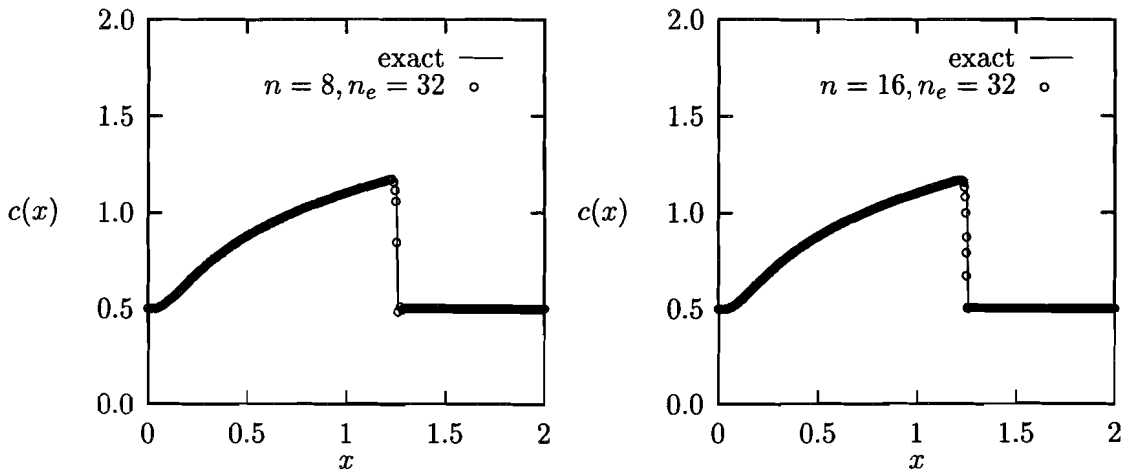

Figure 3.5: Time-linearized IETG spectral element approximation of a shock using 128 time steps with $n_e = 32$. Left-top: $n = 4$ no Picard iteration. Right-top: $n = 4$ Picard iteration. Left-bottom: $n = 8$ Picard iteration. Right-bottom: $n = 16$ Picard iteration.

Table 3.7: Discrete maximum error $\|c - c_h\|_{\infty,gl}$ for the rotation of a Gaussian hill; degree of approximation $n = 2$ fixed with varying number of elements n_e

time-steps	$n_e = 16$	$n_e = 64$	$n_e = 144$	$n_e = 256$
256	$0.53\ 10^{0}$	$0.18\ 10^{0}$	$0.77\ 10^{-1}$	$0.34\ 10^{-1}$
512	$0.53\ 10^{0}$	$0.19\ 10^{0}$	$0.82\ 10^{-1}$	$0.37\ 10^{-1}$
1024	$0.53\ 10^{0}$	$0.19\ 10^{0}$	$0.82\ 10^{-1}$	$0.38\ 10^{-1}$

It is evident that the two-step scheme performs very well for this problem. The results of table 3.6 show that the Gaussian hill is convected very accurately if the degree of the approximation increases (p-convergence). From table 3.7 it can be deduced that also h-convergence is obtained; the solutions obtained by increasing the degree of approximation however, are much more accurate. In fig. 3.6 (left) the solution for $n = 8$ is shown. There are still some 'wiggles' visible in this solution. Fig. 3.6 (right) shows the solution for $n = 16$, which is convected in an extremely accurate way.

3.6.5 1-D convection-diffusion of a Gaussian hill

In order to test the performance of the operator splitting approach, in this section a one-dimensional convection-diffusion problem is solved using an implicit time-integration for the diffusion step and the explicit two-step EETG scheme for the convection step.
Consider as a test case for the operator splitting scheme the problem of a Gaussian hill in one dimension traveling with a constant velocity $u = 1$ and spreading isotropically with a

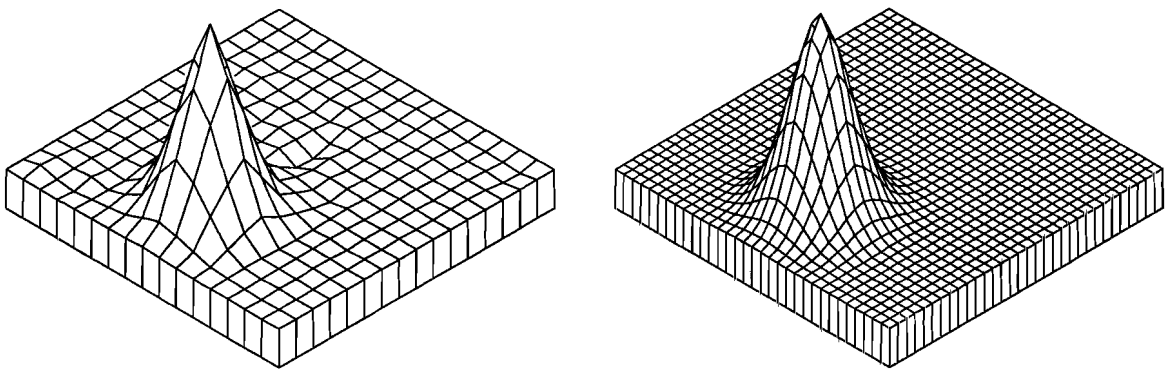

Figure 3.6: Unsteady rotation of a Gaussian hill; two-step EETG approximation using 1024 time-steps for $n_e = 4, n = 8$ (left) and for $n_e = 4, n = 16$ (right)

viscosity $\eta = 0.005$. The exact solution has the form

$$c(x,t) = \frac{\sigma(0)}{\sigma(t)} e^{-\frac{(x - x_0 - ut)^2}{2\sigma(t)^2}}, \tag{3.47}$$

where $\sigma(t) = \sqrt{\sigma(0) + 2\eta t}$. The initial hill ($t = 0$) is centered around $x_0 = 0.15$ and has a standard deviation of $\sigma(0) = 0.04$. The hill is convected with constant velocity $u = 1$ and $t \in [0, 0.3]$.

This problem is solved using the splitting scheme described above for both a first- and a second-order backward differences (BDF) scheme and for a Crank-Nicolson scheme (CN-new). The number of time-steps for the convection step is equal to 64. The spectral element discretization uses $n_e = 16$ elements with degree of approximation $n = 4$. The discrete maximum error $\varepsilon = \|c - c_h\|_{\infty,gl}$ is given in table 3.8. Fig. 3.7 shows the exact solution and the approximation for $n_e = 16, n = 4$ using a second-order backward differences scheme with 4 diffusion time-steps.

The performance of the operator splitting scheme is quite good. Only very few expensive diffusion steps are needed to obtain accurate solutions. It can also be seen that the backward differences schemes and the Crank–Nicolson scheme achieve the theoretical order of accuracy for sufficient diffusion steps. The performance of the 'classical' Crank–Nicolson approach is very bad compared to the other results. For the small number of diffusion steps that are needed to obtain accuracy for the other schemes, the solution is not very accurate. The large number of convection steps in each diffusion cycle does not require much extra processing time, since each convection step is solved explicitly.

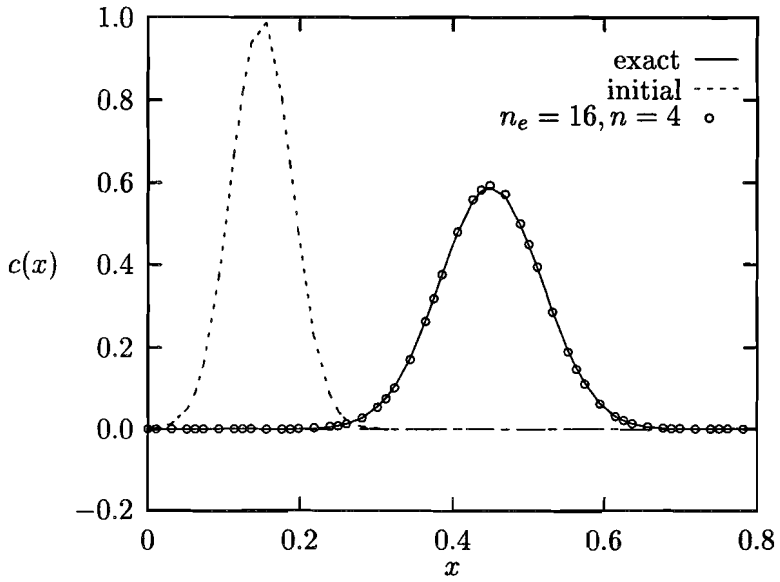

Figure 3.7: Convection and diffusion of a Gaussian hill; exact solution and two-step EETG/second-order backward differences approximation for $n_e = 16, n = 4$ with 4 diffusion steps containing 64 convection steps each

Table 3.8: Discrete maximum error $\|c - c_h\|_{\infty,gl}$ for the convection and diffusion of a Gaussian hill; $n_e = 16$ elements of degree $n = 4$; 64 two-step EETG convection steps per diffusion step

diffusion steps	BDF 1st-order	BDF 2nd-order	CN-new	CN-classical
2	$0.42\ 10^{-1}$	$0.24\ 10^{-1}$	$0.58\ 10^{-2}$	$0.23\ 10^{0}$
4	$0.22\ 10^{-1}$	$0.39\ 10^{-2}$	$0.16\ 10^{-2}$	$0.26\ 10^{0}$
8	$0.11\ 10^{-1}$	$0.87\ 10^{-3}$	$0.42\ 10^{-3}$	$0.11\ 10^{0}$
16	$0.58\ 10^{-2}$	$0.31\ 10^{-3}$	$0.24\ 10^{-3}$	$0.43\ 10^{-1}$
32	$0.30\ 10^{-2}$	$0.17\ 10^{-3}$	$0.17\ 10^{-3}$	$0.20\ 10^{-1}$

3.7 Application of SEM to wave equation

Consider the following model problem:

$$\frac{\partial^2 u}{\partial t^2}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 0 \quad \text{in} \quad \mathbf{R}\times]0,T[, \tag{3.48}$$

$$u(x,0) = u_0(x), \quad \frac{\partial u}{\partial t}(x,o) = u_(x) \quad \text{in} \quad \mathbf{R} \tag{3.49}$$

which describes the propagation of a 1D wave over the real axis $\mathbf{R}$.
Its Galerkin formulation reads:

$$\frac{d^2}{dt^2}\int_{\mathbf{R}} uv dx + \int_{\mathbf{R}} \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} dx = 0 \qquad \forall v \in H^1(\mathbf{R}) \tag{3.50}$$

Further, a spectral element discretization can be applied to (3.50) resulting in a ordinary differential equation (in time) for the values of the solution in the collocation points. The algorithm is exactly the same as the one for the second order elliptic equation described in section 2.3.1.
The easiest way for time discretization is to use a standard second-order finite difference scheme for $\frac{d}{dt}$:

$$\frac{\partial^2 u_h(t^n)}{\partial t^2} = \frac{u_h^{n+1} - 2u_h^n + u_h^{n-1}}{\Delta t^2} + O(\Delta t^2) \tag{3.51}$$

If this accuracy in time is not satisfactory some more sophisticated approaches can be applied to derive higher-order stable schemes in a way similar to the Taylor-Galerkin schemes described in section 3.4.

Chapter 4

Numerical solution of the Navier-Stokes equations

4.1 Introduction

In this section the spatial and temporal discretization of the incompressible Navier-Stokes equations is considered. They form a set of coupled equations for both velocity and pressure. The pressure is an implicit variable which instantaneously 'adjusts itself' in such a way that the velocity remains divergence free. As the coupled set of equations for velocity and pressure forms a saddle-point problem (Girault and Raviart, 1986) the approximation spaces for the velocity has to be taken different from that for the pressure in order to obtain a unique pressure solution. For the instationary Navier-Stokes equations the situation is different if pressure-correction or projection methods that decouple the momentum and continuity equation are chosen.

4.2 Solution methods for the stationary Navier-Stokes equations

4.2.1 Weak formulation

The stationary Navier-Stokes equations for incompressible flow are given by:

$$\begin{cases} \rho(\mathbf{v}\cdot\nabla)\mathbf{v} - \nabla\cdot\boldsymbol{\sigma} = \rho\mathbf{f} \\ \\ \nabla\cdot\mathbf{v} = 0 \end{cases} \tag{4.1}$$

The boundary conditions can have the form:

$$\begin{cases} \mathbf{v} = \mathbf{g}_0 & \text{on } \Gamma_0 \\ \\ \boldsymbol{\sigma}\cdot\mathbf{n} = \mathbf{g}_1 & \text{on } \Gamma_1 \end{cases} \tag{4.2}$$

Also combinations of these two types of boundary conditions in different directions are possible but give rise to complex writing and therefore will not be considered here. A weak form of (4.1) can be derived by introducing weighting functions for the momentum and continuity equations $\mathbf{w} \in L^2(\Omega)$ and $q \in L^2(\Omega)$. The pressure is determined up to a constant which can be fixed by the choice $q \in Q$ with:

$$Q = \left\{ q \in L^2(\Omega) | \int_\Omega q d\Omega = 0 \right\} \tag{4.3}$$

Then the weak form reads:

$$\begin{cases} \int_\Omega [\rho(\mathbf{v}\cdot\nabla)\mathbf{v} - \nabla\cdot\boldsymbol{\sigma}]\cdot\mathbf{w}d\Omega = \int_\Omega \rho\mathbf{f}\cdot\mathbf{w}d\Omega \\ \int_\Omega (\nabla\cdot\mathbf{v})qd\Omega = 0 \end{cases} \tag{4.4}$$

If we choose the weighting function $\mathbf{w}$ to be in $\mathbf{H}^1(\Omega)$ (see appendix A.1) we can integrate by parts the second term in (4.4) and obtain [1]:

$$\begin{cases} \int_\Omega [\rho\mathbf{w}\cdot(\mathbf{v}\cdot\nabla)\mathbf{v} + \boldsymbol{\sigma} : (\nabla\mathbf{w})]\, d\Omega = \int_\Omega \rho\mathbf{f}\cdot\mathbf{w}d\Omega + \int_\Gamma (\boldsymbol{\sigma}\cdot\mathbf{n})\cdot\mathbf{w}d\Gamma \\ \int_\Omega (\nabla\cdot\mathbf{v})qd\Omega = 0 \end{cases} \tag{4.5}$$

After substitution of the constitutive equation for Newtonian flow ($\boldsymbol{\sigma} = -p\mathbf{I} + \eta\dot{\boldsymbol{\gamma}}$) the following weak form is obtained:

$$\begin{cases} \int_\Omega [\rho\mathbf{w}\cdot(\mathbf{v}\cdot\nabla)\mathbf{v} + \eta\dot{\boldsymbol{\gamma}}\,(\mathbf{v}) : \dot{\boldsymbol{\gamma}}\,(\mathbf{w}) - p\nabla\cdot\mathbf{w}]\, d\Omega = \int_\Omega \rho\mathbf{f}\cdot\mathbf{w}d\Omega + \int_\Gamma (\boldsymbol{\sigma}\cdot\mathbf{n})\cdot\mathbf{w}d\Gamma \\ \int_\Omega (\nabla\cdot\mathbf{v})qd\Omega = 0 \end{cases} \tag{4.6}$$

Here $\dot{\boldsymbol{\gamma}}\,(\mathbf{u}) = \frac{1}{2}[\nabla\mathbf{u} + (\nabla\mathbf{u})^c]$. With the aid of the space of trial solutions:

$$\mathbf{V} = \{\mathbf{v}|\mathbf{v}\in\mathbf{H}^1(\Omega), \mathbf{v} = \mathbf{g}_0 \text{ on } \Gamma_0\} \tag{4.7}$$

and the space of weigthing functions defined as:

$$\mathbf{W} = \{\mathbf{w}|\mathbf{w}\in\mathbf{H}^1_0(\Omega), \mathbf{w} = \mathbf{0} \text{ on } \Gamma_0\} \tag{4.8}$$

the weak formulation of the set of equations and boundary conidtions given by (4.1) and (4.2) reads:

Find $\mathbf{v}\in\mathbf{V}$ *and* $p\in Q$ *such that:*

$$\begin{cases} \mathcal{N}(\mathbf{v},\mathbf{v},\mathbf{w}) + \mathcal{D}(\mathbf{v},\mathbf{w}) + \mathcal{L}(\mathbf{w},p) = \ell(\mathbf{w}) & \forall_{\mathbf{w}\in\mathbf{W}} \\ \mathcal{L}(\mathbf{v},q) = 0 & \forall_{q\in Q} \end{cases} \tag{4.9}$$

with:

$$\mathcal{N}(\mathbf{u},\mathbf{v},\mathbf{w}) = \int_\Omega \rho[(\mathbf{u}\cdot\nabla)\mathbf{v}]\cdot\mathbf{w}d\Omega \tag{4.10}$$

$$\mathcal{D}(\mathbf{v},\mathbf{w}) = \int_\Omega \eta\dot{\boldsymbol{\gamma}}\,(\mathbf{v}) : \dot{\boldsymbol{\gamma}}\,(\mathbf{w})d\Omega \tag{4.11}$$

[1] Here the tensor identity for symmetric tensors $\boldsymbol{\sigma}$ given by: $(\boldsymbol{\sigma} : \nabla\mathbf{w}) = \nabla\cdot(\boldsymbol{\sigma}\cdot\mathbf{w}) - \mathbf{w}\cdot(\nabla\cdot\boldsymbol{\sigma})$ is used.

$$\mathcal{L}(\mathbf{v},q) = -\int_{\Omega}(\nabla\cdot\mathbf{v})qd\Omega \tag{4.12}$$

$$\ell(\mathbf{w}) = \int_{\Omega}(\mathbf{f}\cdot\mathbf{w})qd\Omega + \int_{\Gamma_1}(\mathbf{g}_1\cdot\mathbf{w})qd\Gamma \tag{4.13}$$

4.2.2 Brezzi-Babuška stability condition

Let we introduce the weakly divergence-free vector space:

$$\mathbf{V}_0 = \{\mathbf{v}\in\mathbf{V}|\mathcal{L}(\mathbf{v},q) = 0, \forall_{q\in Q}\} \tag{4.14}$$

and choose $\mathbf{W}_0 \equiv \mathbf{V}_0$. Then the weak form (4.9) without the contibution of the convection terms reduces to:

Find $\mathbf{v}\in\mathbf{V}_0$ *such that:*

$$\mathcal{D}(\mathbf{v},\mathbf{w}) = \ell(\mathbf{w}) \qquad \forall_{\mathbf{w}\in\mathbf{W}_0} \tag{4.15}$$

Using the Lax-Millgram theorem it can be proved that the Stokes equation (4.15) has a unique solution [2]. Then the pressure follows from:

Find $p\in Q$ *such that:*

$$\mathcal{L}(\mathbf{w},p) = \ell(\mathbf{w}) - \mathcal{D}(\mathbf{v},\mathbf{w}) \qquad \forall_{\mathbf{v}\in\mathbf{V}} \tag{4.16}$$

This equation only has a unique solution for the pressure if the following condition holds:

$$\exists_{\beta>0}\sup_{\mathbf{v}\in\mathbf{V}}\frac{\mathcal{L}(\mathbf{v},q)}{||\mathbf{v}||_V} \geq \beta||q||_Q \qquad \forall_{q\in Q} \tag{4.17}$$

This condition is called the Brezzi-Babuška condition but was originally derived by Ladyzhenskaya (1969). The interpretation of this condition is not easy but it will be clear that it restricts the choice of the spaces $\mathbf{V}$ and Q in the sense that not any combination will satisfy (4.17). This is illustrated by the following. Assume that the velocity approximation is taken in the space $\mathbf{V}^h$ given by:

$$\mathbf{V}^h = \left\{\mathbf{v}\in\mathbf{H}^1(\Omega), \mathbf{v}\in P_k(\Omega)\right\} \tag{4.18}$$

with $P_k(\Omega)$ the space of polynomials in Ω of order $\leq k$. Assume also that the pressure is taken in the space Q_x^h given by:

$$Q_x^h = \left\{q\in L^2(\Omega), q\in P_k(\Omega), \int_{\Omega} qd\Omega = 0\right\} \tag{4.19}$$

If the set $(\mathbf{v}^h,p^h)\in\mathbf{V}^h\times Q_x^h$ is a solution of the weak form (4.9), then also the sets $(\mathbf{v}^h,p^h+p^x)$ are solutions as long as $p^x\in X^h$ with:

$$X^h = \left\{q\in Q_x^h, \mathcal{L}(\mathbf{v},q) = 0, \forall_{\mathbf{v}^h\in V^h}\right\} \tag{4.20}$$

[2] For the Navier-Stokes equations no such proof can be given and indeed non-unique solutions can exist (see also chapter 4)

The space $X^h \subset Q_x^h$ contains all spurious pressure modes. The Brezzi-Babuška condition can be seen as a condition needed to ensure that the space $Q^h \subset Q_x^h$ is such that it does not contain spurious pressure modes, i.e. $Q^h \cap X^h = \emptyset$. This is clear from (4.17) since for all $q^h \in X^h$ the left hand side is zero by virtue of $\mathcal{L}(\mathbf{v}, q) = 0$ while the right hand side is a possitive real. In practice the Brezzi-Babuška condition implies that the pressure approximation must be taken one or two orders lower than the velocity approximation. Note that in domain decomposition methods like finite and spectral element methods the velocity must be continuous over the domain boundaries since it must be taken in $\mathbf{H}^1$. The pressure however may be discontinuous over the domain boundaries. An overview of admissible finite element spaces $\mathbf{V}^h$ and Q^h is given by Fortin (1981) and Fortin and Fortin (1985).

4.2.3 Integrated method

In this section, for the sake of simplicity, we assume that only homogenious Dirichlet boundary conditions are imposed. The non-homogenious case can also be treated but this involves some complications. If one of the space discretization methods described in section 2 is applied to the weak formulation given in (4.9) the following algebraic system of equations will be obtained:

$$\begin{cases} \mathbf{N}(\mathbf{v}^h)\mathbf{v}^h + \mathbf{D}\mathbf{v}^h + \mathbf{L}^T p^h &= \mathbf{f} \\ \mathbf{L}\mathbf{v}^h &= \mathbf{0} \end{cases} \tag{4.21}$$

In case that Galerkin method is used the approximate solutions for velocity and pressure are expended over a finite basis:

$$\mathbf{v}^h = \sum_{i=1}^{N} \mathbf{v}_i^h \boldsymbol{\phi}_i \tag{4.22}$$

$$p^h = \sum_{i=1}^{M} p_i^h \psi_i \tag{4.23}$$

with $\boldsymbol{\phi}_i \in H^1(\Omega), \boldsymbol{\phi}_i|_\Gamma = 0$ and linearly independent and $\psi_i \in L_2(\Omega)$. Furthermore, if we introduce the finite-dimensional spaces $\mathbf{V}_{\boldsymbol{\phi}}^h$ consisting of all the linear combinations of $\{\boldsymbol{\phi}_i\}_{i=1}^N$ and Q_ψ^h consisting of all the linear combinations of $\{\psi_i\}_{i=1}^M$ the discrete Galerkin analog of (4.9) reads:

Find $\mathbf{v}^h \in \mathbf{V}_{\boldsymbol{\phi}}^h$ *and* $p^h \in Q_\psi^h$ *such that:*

$$\begin{aligned} \mathcal{N}(\mathbf{v}^h, \mathbf{v}^h, \boldsymbol{\phi}_i) + \mathcal{D}(\mathbf{v}^h, \boldsymbol{\phi}_i) + \mathcal{L}(\boldsymbol{\phi}_i, p^h) &= \ell(\boldsymbol{\phi}_i) \qquad i = 1, .., N \\ \mathcal{L}(\mathbf{v}^h, \psi_j) &= 0 \qquad j = 1, ..., M \end{aligned} \tag{4.24}$$

The expressions for $\mathbf{N}(\mathbf{v}^h)$, $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{f}$ then follow straight forward by substitution of the functions $\mathbf{v}^h$, p^h, $\boldsymbol{\phi}_i$ and ψ_i in equations (4.10) to (4.13). From here we will drop the superscript ^{h} in situations that it is clear whether the discrete version $\mathbf{v}^h$ or the continuous version $\mathbf{v}$ is meant. There are two main problems involved in solving the set of equations given in (4.21). First, the set of equations is non-linear because of the convective term $\mathbf{N}(\mathbf{v})$. Secondly, the set of conditions is difficult to solve due to the fact that the matrix contains zeros on the main diagonal as there are no pressure unknowns in the continuity equation. The sequel of this section will deal with those two problems.

4.2.4 Linearization of the convective terms

Linearization is performed by using an iterative procedure with:

$$\begin{aligned}
&\text{Picard 0:} && \mathbf{N}(\mathbf{v}^{n+1})\mathbf{v}^{n+1} = \mathbf{N}(\mathbf{v}^{n})\mathbf{v}^{n} \\
&\text{Picard 1:} && \mathbf{N}(\mathbf{v}^{n+1})\mathbf{v}^{n+1} = \mathbf{N}(\mathbf{v}^{n})\mathbf{v}^{n+1} \\
&\text{Picard 2:} && \mathbf{N}(\mathbf{v}^{n+1})\mathbf{v}^{n+1} = \mathbf{N}(\mathbf{v}^{n+1})\mathbf{v}^{n} \\
&\text{Newton-Raphson:} && \mathbf{N}(\mathbf{v}^{n+1})\mathbf{v}^{n+1} = \mathbf{J}(\mathbf{v}^{n})\mathbf{v}^{n+1} - \mathbf{N}(\mathbf{v}^{n})\mathbf{v}^{n}
\end{aligned} \tag{4.25}$$

Here $\mathbf{J}(\mathbf{v}^n)$ is the Jacobian:

$$\mathbf{J}(\mathbf{v}^n) = \frac{d}{d\mathbf{v}}\mathbf{N}(\mathbf{v}^n)\mathbf{v}^n \tag{4.26}$$

Since the convective term is only quadratic in $\mathbf{v}$ the Jacobian gets the simple form (van de Vosse *et al.*, 1989):

$$\mathbf{J}(\mathbf{v}^n) = \mathbf{N}(\mathbf{v}^n)\mathbf{v}^{n+1} + \mathbf{N}(\mathbf{v}^{n+1})\mathbf{v}^n \tag{4.27}$$

The Picard iteration schemes have a relatively large convergence region but a slow (or no) rate of convergence in contrast with the Newton-Raphson iteration which shows fast convergence but with a relatively small convergence region. In practice a few Picard iterations can be used to move the initial guess (mostly the solution of the Stokes equations or a solution with a lower Reynolds number) into the convergence region of the Newton-Raphson method.
The set of linearized equations to be solved each iteration can be written as:

$$\begin{cases} \mathbf{A}(\mathbf{v}^n)\mathbf{v}^{n+1} + \mathbf{L}^T p^{n+1} &= \mathbf{f}(\mathbf{v}^n) \\ \mathbf{L}\mathbf{v}^{n+1} &= \mathbf{0} \end{cases} \tag{4.28}$$

which still is difficult to solve due to the zero elements on the main diagonal. Partial pivoting or special numbering of the unknowns will demolish the band structure of the matrix and hereby is inefficient with respect to computing time and memory usage. In the next two sections different ways of decoupling the set of equations will be described briefly.

4.2.5 Penalty function method

An often used way to decouple the system of equations is provided by the penalty function method. Here the continuity equation is perturbed with a small term proportional to the pressure:

$$\nabla \cdot \mathbf{v} = -\varepsilon p \tag{4.29}$$

This will yield a discrete system of the form:

$$\begin{cases} \mathbf{A}(\mathbf{v})\mathbf{v} + \mathbf{L}^T\mathbf{p} &= \mathbf{f} \\ \mathbf{L}\mathbf{v} &= \varepsilon \mathbf{M}_p \mathbf{p} \end{cases} \tag{4.30}$$

or equivalently the decoupled system:

$$\begin{aligned} &[\mathbf{A}(\mathbf{v}) + \frac{1}{\varepsilon}\mathbf{L}^T\mathbf{M}_p^{-1}\mathbf{L}]\mathbf{v} = \mathbf{f} \\ &\mathbf{p} = \frac{1}{\varepsilon}\mathbf{M}_p^{-1}\mathbf{L}\mathbf{v} \end{aligned} \tag{4.31}$$

Since the the pressure mass-matrix $\mathbf{M}_p$ must be inverted, the penalty function method can only be applied efficiently in combination with a discontinuous pressure approximation. In that case $\mathbf{M}_p$ can be inverted element-by-element. Due to the small parameter ε the system is ill-conditioned and mostly direct matrix solvers have to be used for the velocity equation. The pressure can be computed in a post-processing step from the second equation of (4.31).

4.2.6 Uzawa methods

Another way of decoupling the momentum and mass equations is provided by the Uzawa algorithm (see Fortin and Glowinski, 1983). This is an iterative procedure where the initial pressure is guessed and the velocity and pressure at iteration $n+1$ are computed from:

$$\begin{cases} \mathbf{A}\mathbf{v}^{n+1} = \mathbf{f} + \mathbf{L}^T\mathbf{p}^n \\ \mathbf{p}^{n+1} = \mathbf{p}^n - \beta\mathbf{L}\mathbf{v}^{n+1} \end{cases} \tag{4.32}$$

It can be proved that the solution of this iterations scheme converges to the solution of the original equations for $0 < \beta < 2/\max(\lambda_i)$, with λ_i the eigenvalues of $\mathbf{A}^{-1}\mathbf{L}^T\mathbf{L}$.
Better convergence properties can be established by the addition of a kind of penalty term (Fortin and Glowinski, 1983):

$$\begin{cases} (\mathbf{A} + \gamma\mathbf{L}^T\mathbf{L})\mathbf{v}^{n+1} = \mathbf{f} + \mathbf{L}^T\mathbf{p}^n \\ \mathbf{p}^{n+1} = \mathbf{p}^n - \beta\mathbf{L}\mathbf{v}^{n+1} \end{cases} \tag{4.33}$$

This is referred to as the Powell-Hestenes method and can be seen as a iterative penalty function method. Advantage of this scheme compared to the penalty function method is that the parameter γ is not very large so that the condition of the matrices involved is not altered too much.
Maday and Patera (1989) obtained a decoupling of the set of equations (4.28) by writing:

$$\begin{cases} \mathbf{v} = -\mathbf{A}^{-1}[\mathbf{L}^T\mathbf{p} - \mathbf{f}] \\ \mathbf{L}\mathbf{v} = \mathbf{0} \end{cases} \tag{4.34}$$

Multiplication of the first equation with $\mathbf{L}$ and substitution of the second equation yields:

$$\mathbf{L}\mathbf{v} = -\mathbf{L}\mathbf{A}^{-1}[\mathbf{L}^T\mathbf{p} - \mathbf{f}] = \mathbf{0} \tag{4.35}$$

and thus an equation for the pressure:

$$\mathbf{L}\mathbf{A}^{-1}\mathbf{L}^T\mathbf{p} = \mathbf{L}\mathbf{A}^{-1}\mathbf{f} \tag{4.36}$$

Once the pressure is solved using for instance an iterative solver for (4.36), the velocity can be computed from:

$$\mathbf{A}\mathbf{v} = -\mathbf{L}^T\mathbf{p} + \mathbf{f} \tag{4.37}$$

4.3 Solution methods for the instationary Navier-Stokes equations

Consider the instationary Navier-Stokes equations for incompressible flow given by:

$$\begin{cases} \rho\dfrac{\partial \mathbf{v}}{\partial t} + \rho(\mathbf{v}\cdot\nabla)\mathbf{v} - \nabla\cdot\boldsymbol{\sigma} = \rho\mathbf{f} \\ \nabla\cdot\mathbf{v} = 0 \end{cases} \tag{4.38}$$

together with the boundary conditions given in (4.2) and initial conditions for the velocity and the pressure. Application of the space discretization method as prescribed in the previous section will yield a set of equations similar to (4.28):

$$\begin{cases} \mathbf{M}\dot{\mathbf{v}} + \mathbf{N}(\mathbf{v})\mathbf{v} + \mathbf{D}\mathbf{v} + \mathbf{L}^T\mathbf{p} &= \mathbf{f}(\mathbf{v}) \\ \mathbf{L}\mathbf{v} &= \mathbf{0} \end{cases} \tag{4.39}$$

with $\mathbf{M}$ is the mass-matrix. Not all the temporal discretization schemes described in section 3 can be applied directly to this system. As fully explicit treatment is not possible because then the pressure unknowns disappear from the system and the incompressibility constraint is not satisfied anymore, all time discretizations schemes directly applied to (4.39) will need some kind of an implicit treatment (see section 4.3.1) of the pressure unless in some way a correction of the solution with the aid of the incompressibility constraint can be performed (see section 4.3.2).

4.3.1 Time integration methods

Both Adams-Moulton and Backward-differencing methods could be used to discretize the space-discretized Navier-Stokes equations given by (4.39). The backward-difference schemes are only conditionally stable because they have a small part of the imaginary axis for which the multiplication matrix is larger then 1. This area increases with higher order. The Adams-Moulton schemes are unconditionally stable only for the first (EI) and second (CN) order ones. An unconditionally stable time integration scheme for the complete (unsplitted) set of equations (4.39) can be constructed by a combination of the EI and CN method. Such a combination is provided by the θ-method:

$$\begin{cases} \mathbf{M}\dfrac{\mathbf{v}^{n+1}-\mathbf{v}^n}{\Delta t} + \theta[\mathbf{N}(\mathbf{v}^{n+1}) + \mathbf{D}]\mathbf{v}^{n+1} + \theta\mathbf{L}^T\mathbf{p}^{n+1} = \\ \qquad \theta\mathbf{f}^{n+1}(\mathbf{v}) + (1-\theta)\mathbf{f}^n(\mathbf{v}) - (1-\theta)[\mathbf{N}(\mathbf{v}^n) + \mathbf{D}]\mathbf{v}^n - (1-\theta)\mathbf{L}^T\mathbf{p}^n \\ \mathbf{L}\mathbf{v}^{n+1} = \mathbf{0} \end{cases} \tag{4.40}$$

For $\theta = 1$ this scheme is equivalent to an Euler implicit scheme which is first order accurate in time and for $\theta = 0.5$ this scheme is a Crank- Nicolson scheme which is second order accurate. The pressure can be eliminated by using a penalty function method, the nonlinear convective terms can be linearized in time by using one step of a Newton-

Raphson iteration:

$$\begin{cases} \mathbf{M}\dfrac{\mathbf{v}^{n+1}-\mathbf{v}^n}{\Delta t} + \theta[\mathbf{J}(\mathbf{v}^n)+\mathbf{D}+\dfrac{1}{\varepsilon}\mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T]\mathbf{v}^{n+1} = \\ \qquad \theta\mathbf{f}^{n+1}(\mathbf{v}) + (1-\theta)\mathbf{f}^n(\mathbf{v}) - \theta[\mathbf{N}(\mathbf{v}^n)+\mathbf{D}+\dfrac{1}{\varepsilon}\mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T]\mathbf{v}^n - \theta\mathbf{N}(\mathbf{v}^n)\mathbf{v}^n \\ \mathbf{p}^{n+1} = \dfrac{1}{\varepsilon}\mathbf{M}^{-1}\mathbf{L}\mathbf{v}^{n+1} \end{cases} \tag{4.41}$$

For large negative eigenvalues the Crank-Nicolson method has a multiplication factor equal to -1. As a consequence, small perturbations in the solution will damp only very slowly and will show an oscillatory behaviour in time. Although the amplitude of the oscillation may be very small, this will impose huge oscillations in the pressure because of the penalty parameter ε. Better results with respect to this can be obtained by the Euler implicit method, however, then also oscillatory behaviour of physical origin (like vortex shedding and flow instability) will be damped. A way to overcome this difficulty is to substitute:

$$\mathbf{v}^{n+\theta} = \theta\mathbf{v}^{n+1} + (1-\theta)\mathbf{v}^n \tag{4.42}$$

and eliminate $\mathbf{v}^{n+1}$. This will give a simple two-step alternative:

$$\begin{cases} \text{1a:} & \mathbf{M}\dfrac{\mathbf{v}^{n+\theta}-\mathbf{v}^n}{\theta\Delta t} + [\mathbf{J}(\mathbf{v}^n)+\mathbf{D}+\dfrac{1}{\varepsilon}\mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T]\mathbf{v}^{n+\theta} = \mathbf{f}^{n+\theta}(\mathbf{v}) + \mathbf{N}(\mathbf{v}^n)\mathbf{v}^n \\ \text{1b:} & \mathbf{p}^{n+\theta} = \dfrac{1}{\varepsilon}\mathbf{M}^{-1}\mathbf{L}\mathbf{v}^{n+\theta} \\ \text{2:} & \mathbf{v}^{n+1} = \dfrac{1}{\theta}\left(\mathbf{v}^{n+\theta} - (1-\theta)\mathbf{v}^n\right) \end{cases} \tag{4.43}$$

This is an Euler implicit step to time $t+\theta\Delta t$ followed by a simple extrapolation to $t+\Delta t$ (see equation 4.42). The order of the method is equal to the order of the one-step version. In figure 4.1 the above is illustrated clearly. The vortex shedding downstream a cylinder is computed using both the EI and CN method for a Reynolds number based on the diameter of the cylinder equal to 100. The EI method damps the oscillations and finally yields a steady solution (top figure left) while the CN method is able to find a nice periodic shedding of the vortices (top figure right). The one-step and two-step method show the same result for the velocity (van de Vosse, 1987) but a clear difference in the pressure approximation (bottom figures left). Note that in the two-step method the pressure is evaluated at the time levels $t+\theta\Delta t$ and not at $t+\Delta t$. In the one-step method each modification in the time step induces new spurious pressure oscillations while the two-step method behaves relatively stable.

4.3.2 Pressure correction and projection methods

The pressure correction method has been introduced by Chorin (1968) in a finite difference context. He first derived an intermediate velocity $\mathbf{v}^*$ by neglecting the pressure terms in the discrete momentum equations. Since the pressure unknowns are removed, this intermediate velocity can not satisfy the incompressibility constraint. By subtracting the equation for the intermediate velocity from the original momentum equation and applying the divergence operator on the result of this subtraction, the new pressure can be derived

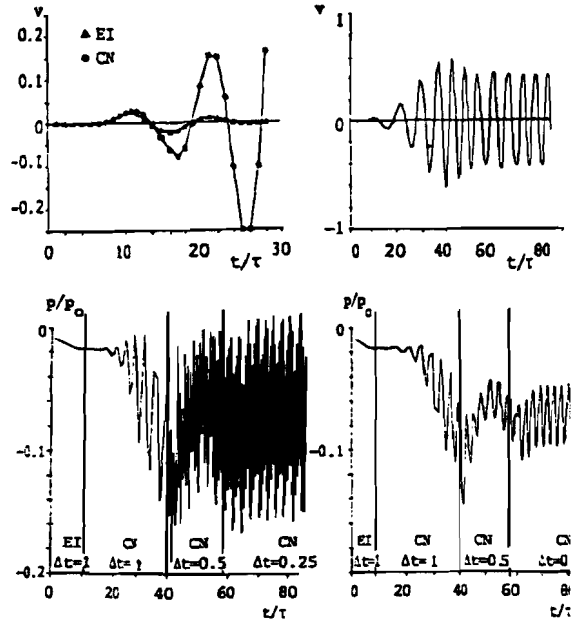

Figure 4.1: Vortex shedding downstream a cylinder at $Re_D = 100$. **left top**: vertical velocity at 10 diameters downstream the cylinder for the Euler implicit (left) and Crank-Nicolson (left and right) method. **left bottom**: pressure at 10 diameters downstream the cylinder for the one-step (left) and two-step (right) Crank-Nicolson method. **right**: streamline patterns during one shedding cycle. (From van de Vosse, 1987)

from a discrete Poisson equation if the difference between the discrete diffusion operator applied to the intermediate velocity and the new velocity is neglected. This new pressure then can be used to update the velocity. In this way Chorin obtained a first order accurate in time method for unsteady Navier-Stokes equations. Later van Kan (1986) improved this scheme by not neglecting the pressure but making use of the pressure at the previous time step. In combination with a Crank-Nicolson time integration he was able to proof second order convergence in time. The same procedure can be applied more generally to the space and time discretized equations that follow from a Galerkin method:

$$\begin{cases} \mathbf{M}\dfrac{\mathbf{v}^{n+1}-\mathbf{v}^n}{\Delta t} + \mathbf{A}(\mathbf{v}^{n+1},\mathbf{v}^n,...) + \mathbf{L}^T\mathbf{p}^{n+1} = \mathbf{f} \\ \mathbf{L}\mathbf{v}^{n+1} = \mathbf{0} \end{cases} \tag{4.44}$$

The intermediate velocity $\mathbf{v}^*$ can be computed from the first equation of:

$$\begin{cases} \mathbf{M}\dfrac{\mathbf{v}^{*}-\mathbf{v}^n}{\Delta t} + \mathbf{A}(\mathbf{v}^{*},\mathbf{v}^n,...) = \mathbf{f} - \mathbf{L}^T\mathbf{p}^n \\ \mathbf{M}\dfrac{\mathbf{v}^{n+1}-\mathbf{v}^n}{\Delta t} + \mathbf{A}(\mathbf{v}^{n+1},\mathbf{v}^n,...) + \mathbf{L}^T\mathbf{p}^{n+1} = \mathbf{f} \\ \mathbf{L}\mathbf{v}^{n+1} = \mathbf{0} \end{cases} \tag{4.45}$$

Subtraction of the first from the second equation yields after neglecting of $\mathbf{A}(\mathbf{v}^{n+1}, \mathbf{v}^n, ...) - \mathbf{A}(\mathbf{v}^*, \mathbf{v}^n, ...)$ the second equation in:

$$\begin{cases} \mathbf{M}\dfrac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t} + \mathbf{A}(\mathbf{v}^*, \mathbf{v}^n, ...) = \mathbf{f} - \mathbf{L}^T\mathbf{p}^n \\ \mathbf{M}\dfrac{\mathbf{v}^{n+1} - \mathbf{v}^*}{\Delta t} = -\mathbf{L}^T(\mathbf{p}^{n+1} - \mathbf{p}^n) \\ \mathbf{L}\mathbf{v}^{n+1} = \mathbf{0} \end{cases} \tag{4.46}$$

Applying the discrete divergence operator $\mathbf{L}$ on the second equation yields:

$$\begin{cases} \mathbf{M}\dfrac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t} + \mathbf{A}(\mathbf{v}^*, \mathbf{v}^n, ...) = \mathbf{f} - \mathbf{L}^T\mathbf{p}^n & \longrightarrow \mathbf{v}^* \\ \mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T(\mathbf{p}^{n+1} - \mathbf{p}^n) = \dfrac{1}{\Delta t}\mathbf{L}\mathbf{v}^* & \longrightarrow \mathbf{p}^{n+1} \\ \mathbf{v}^{n+1} = \mathbf{v}^* - \Delta t\mathbf{M}^{-1}\mathbf{L}^T(\mathbf{p}^{n+1} - \mathbf{p}^n) & \longrightarrow \mathbf{v}^{n+1} \end{cases} \tag{4.47}$$

Note that in this method the inverse of the mass-matrix is involved. Normally a lumped mass matrix is used to overcome this disadvantage. The procedure described above is a form of a discrete pressure correction scheme also described by Hawken *et al.* (1990) and successfully used by Perktold and Peter (1990) for the simulation of pulsatile flow in three-dimensional bifurcation models. The decomposition or projection of the equations is performed on the discrete set of equations.
Also methods are developed where the projection is performed on the continuous strong form of the equations, yielding a set of decoupled equations that do not have the form of a saddle-point problem anymore and thus avoid the need to satisfy the Brezzi-Babuška condition (see e.g. Timmermans *et al.*, 1995). More details on projection methods using the strong form of the equations as a point of departure are given in the papers by Gresho (1990); Gresho and Chan (1990).

4.4 Solution of the Boussinesq equations

In order to model many non-isothermal flows of practical interest, it is usually sufficient to assume that the density and viscosity of the flow are all temperature independent except for the density in the source term of the momentum equations, which results in the so-called Boussinesq equations:

$$\frac{\partial v}{\partial t} + (v \cdot \nabla)v = -\nabla p + RPrTg + Pr\nabla^2 v \tag{4.48}$$

$$\nabla \cdot v = 0 \tag{4.49}$$

$$\frac{\partial T}{\partial t} + (v \cdot \nabla)T = \nabla^2 T \tag{4.50}$$

where $R = (g\beta\Delta Tl^3)/(\kappa\nu), Pr = \nu/\kappa$ are the commonly used Rayleigh and Prandtl numbers and $g = (0, 1)^T$. Here, g is the acceleration of gravity, β is the thermal expansion

coefficient, l is the characteristic length, ΔT is the characteristic temperature difference, κ is the thermal diffusivity and ν is the kinematic viscosity of the fluid.
This system of equations resembles a lot the Navier-Stokes equations except for the buoyancy term in the right-hand side of the momentum equations and the energy equation added to the system. This, however, involves a coupling between the momentum and energy equations which makes the solution of the whole system more difficult than in the case of the Navier-Stokes equations. The most strightforward way to avoid this coupling is to use some extrapolation for either the temperature or the velocity on the corresponding time levels. The following two options are available. The first one is to calculate the velocity according to (4.48) with a source term $R.Pr.T^n$ and then to interpolate its value for $t^n < t < t^{n+1}$. For many flows of practical interest, however, this term is dominant in the momentum equations because the Rayleigh number is very high. That is why the second option seems to be better: first calculate the temperature with an explicit second order extrapolation for the velocity at t^{n+1}:

$$v^{n+1} = (1 + 1/\Delta t)v^n - 1/\Delta t v^{n-1} \tag{4.51}$$

Then the velocity/ pressure problem (4.48)-(4.49) can be solved with an implicit source term using the methods described in the previous section.

4.5 Some numerical results of the SEM application to Navier-Stokes and Boussinesq problems

4.5.1 Vortex shedding behind a cylinder

A frequently used example for testing the performance of unsteady solvers is the von Karman vortex shedding behind a circular cylinder. At $Re \geq 40$ the flow around a circular cylinder becomes essentially unsteady undergoing its first bifurcation towards a periodical regime - so called von Karman vortex shedding (see fig. 4.3). This flow is simulated using the mesh in 4.2 consisting of 68 elements of 8 order (see fig. 4.2) at Reynolds number $Re = 100$. The approximate projection scheme combined with the convection splitting described above are used for time integration. The Strouhal number of the computed vortex shedding is 0.1709. This value compares well with the measurements of Braza *et al.* (1986) who report an average value of 0.17. Engelman and Jamnia (1990) have employed the traditional finite element method to model the same flow. The reported value of the Strouhal number is 0.1724. The number of nodes they used is 14000 compared to the 4352 nodes in the SEM mesh.

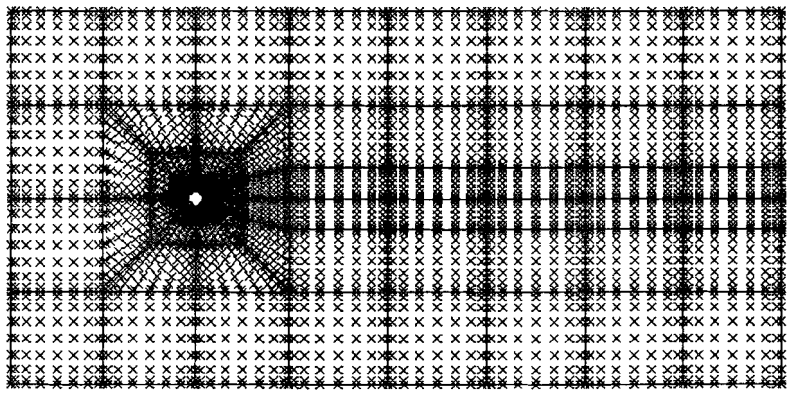

Figure 4.2: Spectral element mesh for the flow past a cylinder

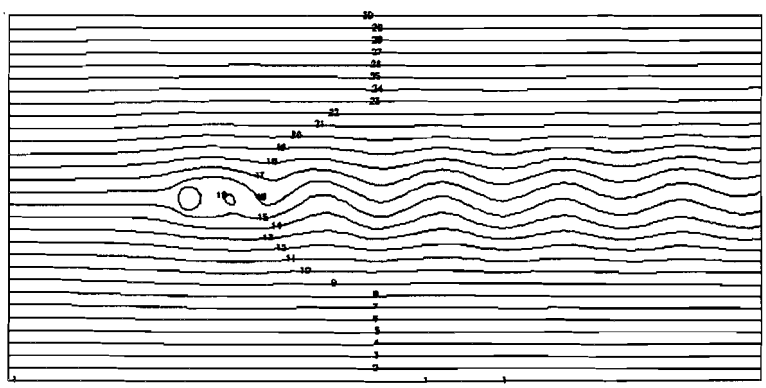

Figure 4.3: Flow past a cylinder at Re = 100; instantaneous steamlines picture.

4.5.2 Differentially heated cavity

Another frequently used example is the Boussinesq flow in a differentially heated cavity. de Vahl Davis (1983) provided a benchmark solution for the flow in a square cavity with a hot left wall and a cold right wall. The top and the bottom walls are kept adiabatic. The Prandtl number is 0.71. The results (see fig. 4.4 and 4.5) on a mesh of 4×4 elements of 8 order are compared with the benchmark solution at mesh size tending to 0 in table 4.1. Results at four different values of the Rayleigh number R are reported. At Rayleigh number $R = 2 \times 10^8$ the flow undergoes a bifurcation towards a periodic regime. The frequency of the oscillations reported by Paolucci and Chenowith (1989) is 630.3. They have used a second order finite difference method to simulate the flow. The spectral element calculations yield a value of 604 which is 4.1% lower.

Figure 4.4: Streamlines (left) and isotherms (right) for the buoyancy-driven flow in an enclosed cavity at $R = 10^3$. Modified pressure correction/operator splitting scheme using 4×4 elements of degree N = 8.

Figure 4.5: Streamlines (left) and isotherms (right) for the buoyancy-driven flow in an enclosed cavity at $R = 10^6$. Modified pressure correction/operator splitting scheme using 4×4 elements of degree N = 8.

Table 4.1: Buoyancy-driven flow in an enclosed cavity. Present results (P) compared with the benchmark numerical solution (B) and the deviation (D) for $R = 10^3$ through $R = 10^6$. Modified pressure correction/operator splitting scheme using 4 × 4 elements of degree N = 8.
* These results were obtained using 16 × 16 finite elements of degree N = 2.

variable	source	$R = 10^3$	$R = 10^4$	$R = 10^5$	$R = 10^6$	$R = 10^{6*}$
$u_{1,\max}$	B	3.649	16.178	34.73	64.63	64.63
	P	3.630	16.171	34.15	63.02	68.17
	D (%)	−0.5	0.0	−1.6	−2.3	+5.5
$x_2(u_1)$	B	0.813	0.823	0.855	0.850	0.850
	P	0.830	0.830	0.875	0.830	0.844
$u_{2,\max}$	B	3.697	19.617	68.59	219.39	219.39
	P	3.693	19.604	66.85	219.69	211.98
	D (%)	−0.1	−0.1	−2.5	+0.1	−3.3
$x_1(u_2)$	B	0.178	0.119	0.066	0.0379	0.0379
	P	0.170	0.125	0.079	0.0404	0.0313
$Nu_{\max}$	B	1.505	3.528	7.717	17.925	17.925
	P	1.507	3.531	7.717	17.350	14.169
	D (%)	+0.1	+0.1	0.0	−3.2	−20.95
$x_2(Nu)$	B	0.092	0.143	0.081	0.0378	0.0378
	P	0.080	0.125	0.080	0.0404	0.0625
$Nu_{\min}$	B	0.692	0.586	0.729	0.989	0.989
	P	0.692	0.586	0.726	0.972	0.989
	D (%)	0.0	0.0	−0.3	−1.7	0.0
$x_2(Nu)$	B	1.0	1.0	1.0	1.0	1.0
	P	1.0	1.0	1.0	1.0	1.0

Chapter 5

Problems

5.1 Example 1: Introduction

With the aid of a number of sample programs we will help novell users of the SEPRAN package to get started. Experienced users may skip the first two examples of this introductory part and start with example 1.3 . Since we do not intend to give a typing course, all sample programs (`progxx.f`) that are mentioned and all the input files (`progxx.dat`) that are needed can be copied with the unix copy command:

```
cp /usr/local/sepran/progs/progxx.* .
```

Here xx denotes the program number that must be copied and . stands for the current directory.
The sample program progxx.f is a FORTRAN main program that calls subroutines from the SEPRAN library. In order to link the main program with the SEPRAN library the unix script `seprun` can be used. Detailed information on the usage of that script can be obtained with the command:

```
seprun -help
```

For the practical work in this course only the options in the following command are of importance:

```
seprun progxx -i progxx.dat -o progxx.out [-cl] [-dev X11]
```

In words this command reads: run (`seprun`) my SEPRAN program (`progxx`) and use the inputfile `progxx.dat` (`-i progxx.dat`). The results must be written in the output file `progxx.out` (`-o progxx.out`). If the program must be compiled and linked with the SEPRAN library this must be indicated with the `-cl` option. In that case the `seprun`-script will generate an executable with the name `progxx` that can be run subsequently without the `-cl` option. Apart from the output file also a plotfile with default name `sepran.cgm` will be generated. This file can be handled in several ways. It can be viewed on X11 or tektronix screens (X-terminal) with the commands `cgm2x sepran.cgm` and `cgm2tek sepran.cgm` respectively. Moreover it can be transposed to postscript with the command `cgm2ps sepran.cgm`. In the latter case a postscript file with name `sepran.ps` will be generated.

In order to view the pictures that are generated on the screen while running the program the option `-dev X11` must be added.

5.1.1 Example 1.1: Running a SEPRAN job

Subroutines SEPRAN is a software library that contains a large number of FORTRAN subroutines that must be called in a main program suplied by the user. The most important subroutines that are needed to build a SEPRAN program are:

`start`	pg 2.2	Starting and initializing the program
`mesh`	pg 3.1	Generation of a computational domain and mesh
`probdf`	pg 4.1	Definition of the problem
`commat`	pg 4.4	Definition of the solution procedure
`presdf`	pg 5.5	Definition of the boundary conditions
`build`	pg 5.1	Construction of the system matrix
`solve`	pg 6.8	Solution of the matrix equation
`finish`	pg 2.5	Finishing of the program

More subroutines for printing and plotting will be introduced below.

Main program Let us first run our first SEPRAN job. Copy the file `prog11.f` and the input file `prog11.dat` (which is empty) to the current directory:

```
cp /usr/local/sepran/progs/prog11.* .
```

The program looks like:

```
      program prog11
c=====================================================================
c     Program to illustrate the sepran subroutines start and finish
c=====================================================================

c==== LOCAL PARAMETERS ================================================

      implicit none
      integer kmesh(100), idum
      double precision rdum

c==== Start sepran (pg 2.2) ===========================================

      call start(0,1,1,1)

c==== Finish the job (pg 2.5) =========================================

      call finish(0)

c=====================================================================
      end
```

It calls two routines `start` and `finish`. Find out what they are meant for in the SEPRAN manual. The manual sections are indicated in the program (pg 2.2 and pg 2.5).

Running Just use:

```
seprun prog11 -i prog11.dat -o prog11.out -cl
```

Check the contents of the outputfile `prog11.out`.

5.1.2 Example 1.2: Mesh generation

Database The data (like coordinates, topology, solution vectors etc.) in SEPRAN are all stored in one huge buffer array (called `ibuffr`). This array can not be accessed easily by the user to put or get data from it. Therefore all SEPRAN subroutines communicate with the buffer array with the aid of small integer arrays that can be seen as the visiting-card of the real data. This visiting-card contains information concerning starting adress, length and type of the real data. Mostly (for instance for the solution array) this visiting-card is an integer array with a length of 5 (`isol`). In some cases however (information of the mesh and the problem to be solved) this visiting-card is more complex and longer arrays are used (`kmesh, kprob`).

Mesh generator A finite element or spectral element mesh is generated with the aid of subroutine `mesh`. An extended description can be found in the programmers guide (pg 3.1). Do not read this complete story because in this course we only need a small subset. Instead use the time to study the following sample program:

```
      program prog12
c=====================================================================
c     Program to illustrate the usage of the mesh generator
c=====================================================================

c==== LOCAL PARAMETERS ================================================

      implicit none
      integer kmesh(100), idum
      double precision rdum

c==== Start sepran (pg 2.2) ===========================================

      call start(0,1,1,1)

c==== Generate the mesh (pg 3.1) ======================================

      kmesh(1) = 100
      call mesh (0, idum, rdum, kmesh)

c==== Plot the mesh (pg 9.3) ==========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) ==================================

      call prinrv( idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Finish the job (pg 2.5) =========================================

      call finish(0)

c=====================================================================
      end
```

and its corresponding input file:

```
# input file for prog12.f ==============================================
mesh1d
```

```
  points
    p1=(-1.0)
    p2=(1.0)
  curves
    c1=line1(p1,p2,nelm=1)
  intermediate points
    sidepoints=15,subdivision=equidistant,midpoints=filled
end
# ==========================================================================
```

Run the program and check what happens.
In order to generate a finite element mesh, the same progam but with the following inputfile can be used:

```
# input for prog12.f ======================================================
mesh1d
  points
    p1=(-1.0)
    p2=(1.0)
  curves
    c1=line1(p1,p2,nelm=15)
*  intermediate points
*    sidepoints=15,subdivision=equidistant,midpoints=filled
end
# ==========================================================================
```

5.1.3 Example 1.3: Creation, printing and plotting a function

Now we have generated a grid with points we will define a solution vector (in this case a scalar function) depending on the coordinates and print and plot it. We need to call subroutine **probdf** (see pg 4.1) in order to provide SEPRAN with the information of this solution vector. Study and run the following program:

```
      program prog13
c=====================================================================
c     Program to illustrate the how to create print and plot a user
c     defined function
c=====================================================================

c==== LOCAL PARAMETERS ==============================================

      implicit none
      integer kmesh(100), kprob(100), iexact(5), icurvs(2)
      integer idum
      double precision rdum

c==== Start sepran (pg 2.2) =========================================

      call start(0,1,1,1)

c==== Generate the mesh (pg 3.1) ====================================

      kmesh(1) = 100
      call mesh (0, idum, rdum, kmesh)

c==== Plot the mesh (pg 9.3) ========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) ================================

      call prinrv( idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) ===================================

      kprob(1) = 100
      call probdf(0, kprob, kmesh, idum)

c==== Create the exact solution vector (pg 5.3) =====================

      call create(0, kmesh, kprob, iexact)

c==== Plot the exact solution vector (pg 9.8) =======================

      icurvs(1) = 0
      icurvs(2) = 1
      call plotfn(0, 1, 1, kmesh, kprob, iexact, 1, icurvs, 20d0, 1d0,
     $            'x', 'u(x)', rdum, rdum)

c==== Print the exact solution (pg 8.6) =============================

      call prinov(iexact, kmesh, kprob, 1, 'Exact Solution', rdum, idum)

c==== Finish the job (pg 2.5) =======================================

      call finish(0)
```

```
c=======================================================================
      end

      double precision function func(ichois, x, y, z)
c=======================================================================
c     Fill SEPRAN vector with Legendre or Chebyshev polynomial of
c     order n in the following way:
c
c     if ichois < 100 : Legendre of order ichois
c     if ichois > 100 : Chebyshev of order ichois-100
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

c==== LOCAL PARAMETERS ================================================
      integer n, k
      double precision pnm1, pn, pnp1

      n = mod(ichois,100)
      if ( n .eq. 0 ) then
         func = 1
         return
      else if ( n .eq. 1 ) then
         func = x
      else
         if ( ichois .le. 100 ) then

c==== Legendre polynomial of order n ==================================

            pnm1 = 1
            pn = x
            do 100 k = 2,n
               pnp1 = dble(2*n+1)*x*pn/dble(n+1) -
     $                              dble(n)*pnm1/dble(n+1)
               pnm1=pn
               pn = pnp1
  100       continue
            func = pnp1
         else

c==== Chebyshev polynomial of order n =================================

            pnm1 = 1
            pn = x
            do 200 k = 2,n
               pnp1 = 2d0*x*pn - pnm1
               pnm1=pn
               pn = pnp1
  200       continue
            func = pnp1
         endif
      endif

c=======================================================================
      end
```

From the function func it can be seen that the following functions are computed:

Legendre polynomials:

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= x \\ L_{n+1}(x) &= \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x) \end{aligned} \tag{5.1}$$

Chebyshev polynomials:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \end{aligned} \tag{5.2}$$

The order and type of the polynomials are determined in the input file:

```
# input for prog13.f ================================================
meshid
  points
    p1=(-1.0)
    p2=(1.0)
  curves
    c1=line1(p1,p2,nelm=2)
  intermediate points
    sidepoints=11,subdivision=legendre,midpoints=filled
  renumber(method=2)
end
problem
  types
    elgrp1=(type=1)
    numdegfd=1
  essbouncond
    degfd1=points(p1,p2)
end
create vector
  type=solution vector
  function=5
end
# ====================================================================
```

by the parameter (function=..) in the create vector part of the input file.
Compare the location of the maxima and minima of the Legendre polynomials with the location of the Gauss-Lobatto points.

5.2 Example 2: Numerical integration

The program below illustrates the difference between piecewise linear or piecewise quadratic integration (as is the case in linear and quadratic finite elements) and Gauss-Lobatto integration.

5.2.1 Example 2.1: Numerical integration

```
      program prog21
c=====================================================================
c Program to demonstarte numerical integration rules for some functions
c=====================================================================

c==== LOCAL PARAMETERS ================================================

      implicit none
      integer kmesh(100), kprob(100), iexact(5), icurvs(2)
      integer idum
      double precision rdum, retval, volint, exact

c==== Start sepran (pg 2.2) ===========================================

      call start(0,1,1,1)

c==== Generate the mesh (pg 3.1) ======================================

      kmesh(1) = 100
      call mesh (0, idum, rdum, kmesh)

c==== Plot the mesh (pg 9.3) ==========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) ==================================

      call prinrv( idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) =====================================

      kprob(1) = 100
      call probdf(0, kprob, kmesh, idum)

c==== Create the exact solution vector (pg 5.3) =======================

      call create(0, kmesh, kprob, iexact)

c==== Plot the exact solution vector (pg 9.8) =========================

      icurvs(1) = 0
      icurvs(2) = 1
      call plotfn(0, 1, 1, kmesh, kprob, iexact, 1, icurvs, 20d0, .3d0,
     $            'x', 'u(x)', rdum, rdum)

c==== Print the exact solution (pg 8.6) ================================

      call prinov(iexact, kmesh, kprob, 1, 'Exact Solution', rdum, idum)

c==== Compute the integral of iexact (pg 6.4) =========================

      retval = volint(0, 1, 1, kmesh, kprob, iexact, idum, rdum, idum)
      write(*,*)'retval = ',retval
      exact = 3.06666667
      write(*,*)'error1 = ',abs(retval-exact)
      exact = 2d0*sin(6d0)
      write(*,*)'error2 = ',abs(retval-exact)
      exact = 3.29616184
      write(*,*)'error3 = ',abs(retval-exact)
```

```
c==== Finish the job (pg 2.5) ========================================

      call finish(0)

c=====================================================================
      end

      double precision function func(ichois, x, y, z)

c=====================================================================
c     Fill SEPRAN vector
c=====================================================================

      implicit none
      integer ichois
      double precision x,y,z

      if ( ichois .eq. 1) then
         func = 1 + x + x*x + x*x*x + x*x*x*x + x*x*x*x*x
      else if ( ichois .eq. 2) then
         func = 6d0*cos(6d0*x)
      else
         func = 6d0/(1+25d0*x*x)
      endif

c=====================================================================
      end

      double precision function elint( icheli, jdegfd, coor, iuser,
     &                     user, vector, index1, index2 )
c=====================================================================
c     compute integral of a function over one element
c=====================================================================

      integer icheli, jdegfd, iuser(*), index1(*), index2(*)
      double precision coor(*), user(*), vector(*)

c==== COMMON BLOCKS ==================================================

      integer ielem, itype, ielgrp, inpelm, icount, ifirst,
     .        notmat, notvec, irelem, nusol, nelem, npoint
      common /cactl/ ielem, itype, ielgrp, inpelm, icount, ifirst,
     .               notmat, notvec, irelem, nusol, nelem, npoint
      save /cactl/
c
c                /cactl/
c    Contains element dependent information for the various element
c    subroutines. cactl is used to transport information from main
c    subroutines to element subroutines
c
c    icount   i   Number of unknowns in element
c    ielem    i   Element number
c    ielgrp   i   Element group number
c    ifirst   i   Indicator if the element subroutine is called for the
c                 first time in a series (0) or not (1)
c    inpelm   i   Number of nodes in element
c    irelem   i   Relative element number with respect to element group
c                 number ielgrp
c    itype    i   Type number of element
```

```
c    nelem    i   Number of elements in the mesh
c    notmat  i/o  Indicator if the element matrix is zero (1) or not (0)
c    notvc   i/o  Indicator if the element vector is zero (1) or not (0)
c    npoint   i   Number of nodes in the mesh
c    nusol    i   Number of degrees of freedom in the mesh
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c==== LOCAL VARIABLES ================================================

      integer i, m
      double precision x(128), xi(128), det, w(128), value, rdum

c==== Gauss-Lobatto weights ==========================================

      call elp640( 0, xi, w, rdum, rdum, inpelm)

c==== Determinant ====================================================

      x(1) = coor(index1(1))
      x(inpelm) = coor(index1(inpelm))
      det = 2d0/( x(inpelm) - x(1) )

c==== Perform numerical integration ==================================

      value = 0d0
      do 60 i=1,inpelm
         value = value + w(i)*vector(index2(i))
   60 continue
      elint = value/det

c=====================================================================
      end
```

Study the program (especially subroutine elint). Note that the following integration rule is programmed:

$$\int_{\Omega_e} f(x)dx = \sum_{i=1}^{npelm} w_i f(x_i) \tag{5.3}$$

Compare the difference in piecewise quadratic and high order Gauss-Lobatto integration by computing the integrals for 2,4,8,16,... quadratic finite elements and 2 spectral elements of order 2,4,8,16,... So fill in the followin tables:

function 1:

2 spectral elements		quadratic finite elements	
order N	error	error	number
2			2
3			3
4			4
			8
			16
			32
N convergence		h convergence	

function 2:

2 spectral elements		quadratic finite elements	
order N	error	error	number
2			2
4			4
8			8
16			16
			32
N convergence		h convergence	

function 3:

2 spectral elements		quadratic finite elements	
order N	error	error	number
2			2
4			4
8			8
16			16
			32
N convergence		h convergence	

Do we have spectral convergence in all cases?
Try the last function with linear finite elements.
By the way, the following input file can be used:

```
# input for prog21.f ================================================
meshid
  points
    p1=(-1.0)
    p2=(1.0)
  curves
    c1=line1(p1,p2,nelm=32)
*  intermediate points
*    sidepoints=15,subdivision=legendre,midpoints=filled
end
problem
  types
    elgrp1=(type=1)
    numdegfd=1
  essbouncond
    degfd1=points(p1,p2)
end
create vector
  type=solution vector
  function=3
end
# ===================================================================
```

5.3 Example 3: Steady convection-diffusion problems

5.3.1 Example 3.1: Steady 1D diffusion

We will now implement a spectral element for 1D diffusion problems given by:

$$-\eta \frac{\partial^2 u}{\partial x^2} = f(x) \tag{5.4}$$

where η is a positive diffusion coefficient. In order to test the performence of the element we will first assume $\eta = 1$ and the right hand side to be:

$$f(x) = cos(x) \tag{5.5}$$

and thus an exact solution:

$$u(x) = cos(x) \tag{5.6}$$

The program is given by:

```
      program prog31
c=====================================================================
c     Program to illustrate how to solve a one dimensional diffusion
c     equation
c=====================================================================

c==== LOCAL PARAMETERS ================================================

      implicit none
      integer kmesh(100), kprob(100), iexact(5), icurvs(2), intmat(5),
     $        isol(5), matr(5), irhsd(5), iinbld(10), iinvec(10), idum
      double precision rdum, error, user(100)

c==== Start sepran (pg 2.2) ===========================================

      call start(0,1,1,1)

c==== Generate the mesh (pg 3.1) ======================================

      kmesh(1) = 100
      call mesh (0, idum, rdum, kmesh)

c==== Plot the mesh (pg 9.3) ==========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) ==================================

      call prinrv( idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) =====================================

      kprob(1) = 100
      call probdf(0, kprob, kmesh, idum)

c==== Create the exact solution vector (pg 5.3) =======================

      call create(0, kmesh, kprob, iexact)

c==== Plot the exact solution vector (pg 9.8) =========================

      icurvs(1) = 0
      icurvs(2) = 1
      call plotfn(0, 1, 2, kmesh, kprob, iexact, 1, icurvs, 20d0, 1d0,
     $            'x', 'u(x)', rdum, rdum)

c==== Print the exact solution (pg 8.6) ===============================

      call prinov(iexact, kmesh, kprob, 1, 'Exact Solution', rdum, idum)
```

```
c==== Define the solution method (pg 4.4) ==============================

      call commat(1, kmesh, kprob, intmat)

c==== Define the essential boundary conditions (pg 5.5) ================

      call presdf(kmesh, kprob, isol)

c==== Fill coefficients ================================================

      user(1) = 1d0

c==== Build the system matrix (pg 5.1) =================================

      iinbld(1) = 0
      call build(iinbld, matr, intmat, kmesh, kprob, irhsd, idum,
     &           isol, idum, idum, user )

c==== Solve the system of equations (pg 6.8) ===========================

      call solve(1, matr, isol, irhsd, intmat, kprob )

c==== Print the solution (pg 8.4) ======================================

      call prinrv( isol, kmesh, kprob, 4, 0, 'Solution' )

c==== Plot the solution  (pg 8.4) ======================================

      call plotfn( 1100, 2, 2, kmesh, kprob, isol, 1, icurvs, 20d0,
     $             1d0, 'x', 'u(x)', rdum, rdum )

c==== Compute the error (pg 6.5) =======================================

      iinvec(1) = 6
      iinvec(2) = 5
      iinvec(3) = 0
      iinvec(4) = 0
      iinvec(5) = 1
      iinvec(6) = 3
      CALL MANVEC( iinvec, error, isol, iexact, idum, kmesh, kprob )
      write ( *, * ) ' Error =', error
c
c==== Finish the job (pg 2.5) ==========================================

      call finish(0)

c=======================================================================
      end

      double precision function func(ichois, x, y, z)
c=======================================================================
c     Fill SEPRAN vector
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      func = cos(x)
```

```
c=======================================================================
      end

      double precision function funccf(ichois, x, y, z)

c=======================================================================
c     Fill SEPRAN right hand side vector
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      funccf = cos(x)

c=======================================================================
      end

      subroutine elem( coor, elemmt, elemvc, iuser, user, uold, matrix,
     &                 vector, index1, index2 )
c=======================================================================
c     spectral element for the 1D diffusion equation
c=======================================================================

      implicit none
      double precision coor(*), elemmt(*), elemvc(*), user(*), uold(*)
      integer iuser(*), index1(*), index2(*)
      logical matrix, vector

c==== COMMON BLOCKS ===================================================

      integer ielem, itype, ielgrp, inpelm, icount, ifirst,
     +        notmat, notvec, irelem, nusol, nelem, npoint
      common /cactl/ ielem, itype, ielgrp, inpelm, icount, ifirst,
     +               notmat, notvec, irelem, nusol, nelem, npoint
      save /cactl/
c
c                /cactl/
c    Contains element dependent information for the various element
c    subroutines. cactl is used to transport information from main
c    subroutines to element subroutines
c
c    icount   i   Number of unknowns in element
c    ielem    i   Element number
c    ielgrp   i   Element group number
c    ifirst   i   Indicator if the element subroutine is called for the
c                 first time in a series (0) or not (1)
c    inpelm   i   Number of nodes in element
c    irelem   i   Relative element number with respect to element group
c                 number ielgrp
c    itype    i   Type number of element
c    nelem    i   Number of elements in the mesh
c    notmat  i/o  Indicator if the element matrix is zero (1) or not (0)
c    notvc   i/o  Indicator if the element vector is zero (1) or not (0)
c    npoint   i   Number of nodes in the mesh
c    nusol    i   Number of degrees of freedom in the mesh
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c==== LOCAL PARAMETERS ================================================
```

```
      integer i, j, m, ij, im, jm
      double precision  dlphi(16384), x(128), xi(128), w(128), det,
     $                  funccf, rdum, dcof
      external funccf

c==== Nodel points ===================================================

      do 5 m = 1,inpelm
         x(m) = coor(index1(m))
    5 continue

c==== Weights and derivatives of the basisfunctions ==================

      call elp640( 1, xi, w, dlphi, rdum, inpelm)

c==== Determinant ====================================================

      det = 2d0/(x(inpelm) - x(1))

c==== Fill element matrix (diffusion part) ===========================

      dcof = user(1)
      do 30 i = 1,inpelm
         do 20 j = 1,inpelm
            ij = inpelm*(i-1) + j
            elemmt(ij) = 0d0
            do 10 m = 1,inpelm
               im = inpelm*(i-1) + m
               jm = inpelm*(j-1) + m
               elemmt(ij) = elemmt(ij) + w(m)*dcof*dlphi(jm)*dlphi(im)
   10       continue
            elemmt(ij) = elemmt(ij)*det
   20    continue
   30 continue

c==== Fill right hand side vector (may depend on x) ==================

      do 60 i = 1,inpelm
         elemvc(i) = w(i)*funccf(1,x(i),x,x)/det
   60 continue

c=====================================================================
      end
```

The new part of the main program starts at the location where subroutine `commat` is called and ends after the call of subroutine `solve`. Four subroutines are new for us:

`commat`	pg 4.4	Definition of the solution procedure
`presdf`	pg 5.5	Definition of the boundary conditions
`build`	pg 5.1	Construction of the system matrix
`solve`	pg 6.8	Solution of the matrix equation

Consult the programmers guide to find out what these subroutines can do for us.

Subroutine `build` assembles the element matrices into the large system matrix and asks the user to provide a subroutine to generate the element matrix. This subroutine is called

`elem` and (for the diffusion matrix) must compute:

$$D_{ij} = \int_{\Omega_e} \eta \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x} dx = \int_{-1}^{1} \eta J^{-1} \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x} dx = \sum_{m=1}^{npelm} \eta w(m) \frac{\partial \phi_i}{\partial x}|_{x_m} \frac{\partial \phi_j}{\partial x}|_{x_m} * det \tag{5.7}$$

and

$$f_i = \int_{\Omega_e} f(x)\phi_i dx = \int_{-1}^{1} Jf(x)\phi_i dx = \sum_{m=1}^{npelm} w(m)f(x_m)\phi_i(x_m)/det = w(i)f(x_i)/det \tag{5.8}$$

Here we made use of the fact that $\phi_i(x_m) = \delta_{im}$. Compare this with what is programmed in subroutine elem.
Check the h- and N-convergence of this element by solving the diffusion equation given above for several values of h and N.

5.3.2 Example 3.2: Steady 1D convection-diffusion

In this example we will extend the program of the previous example to an element for the convection-diffusion equation:

$$-\eta\frac{\partial^2 u}{\partial x^2}+\alpha\frac{\partial u}{\partial x}=f(x) \tag{5.9}$$

To this end we have to add an extra term to the element matrix of the form:

$$C_{ij}=\int_{\Omega_e}\alpha\frac{\partial\phi_j}{\partial x}\phi_i dx=\int_{-1}^{1}\eta\frac{\partial\phi_j}{\partial x}\phi_i dx=\sum_{m=1}^{npelm}\eta w(m)\frac{\partial\phi_j}{\partial x}|_{x_m}\phi_i(x_m)=\eta w(i)\frac{\partial\phi_j}{\partial x}|_{x_i} \tag{5.10}$$

Here again it is used that $\phi_i(x_m)=\delta_{im}$.

In FORTRAN code this may look like:

```
      program prog31
c=====================================================================
c     The one dimensional convection-diffusion equation
c=====================================================================


c==== LOCAL PARAMETERS ================================================

      implicit none
      integer kmesh(100), kprob(100), iexact(5), icurvs(2), intmat(5),
     $        isol(5), matr(5), irhsd(5), iinbld(10), iinvec(10)
      integer idum
      double precision rdum, error, user(100)

c==== Start sepran (pg 2.2) ===========================================

      call start(0,1,1,1)

c==== Generate the mesh (pg 3.1) ======================================

      kmesh(1) = 100
      call mesh (0, idum, rdum, kmesh)

c==== Plot the mesh (pg 9.3) ==========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) ==================================

      call prinrv( idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) =====================================

      kprob(1) = 100
      call probdf(0, kprob, kmesh, idum)

c==== Create the exact solution vector (pg 5.3) =======================

      call create(0, kmesh, kprob, iexact)

c==== Plot the exact solution vector (pg 9.8) =========================
```

```
      icurvs(1) = 0
      icurvs(2) = 1
      call plotfn(0, 1, 2, kmesh, kprob, iexact, 1, icurvs, 20d0, 1d0,
     $            'x', 'u(x)', rdum, rdum)

c==== Print the exact solution (pg 8.6) ================================

      call prinov(iexact, kmesh, kprob, 1, 'Exact Solution', rdum, idum)

c==== Define the solution method (pg 4.4) ==============================

      call commat(2, kmesh, kprob, intmat)

c==== Define the essential boundary conditions (pg 5.5) ================

      call presdf(kmesh, kprob, isol)

c==== Fill coefficients ================================================

      user(1) = 1d0
      user(2) = 32d0

c==== Build the system matrix (pg 5.1) =================================

      iinbld(1) = 0
      call build(iinbld, matr, intmat, kmesh, kprob, irhsd, idum,
     &            isol, idum, idum, user )

c==== Solve the system of equations (pg 6.8) ===========================

      call solve(1, matr, isol, irhsd, intmat, kprob )

c==== Print the solution (pg 8.4) ======================================

      call prinrv( isol, kmesh, kprob, 4, 0, 'Solution' )

c==== Plot the solution  (pg 8.4) ======================================

      call plotfn( 1100, 2, 2, kmesh, kprob, isol, 1, icurvs, 20d0,
     $             1d0, 'x', 'u(x)', rdum, rdum )

c==== Compute the error (pg 6.5) =======================================

      iinvec(1) = 6
      iinvec(2) = 5
      iinvec(3) = 0
      iinvec(4) = 0
      iinvec(5) = 1
      iinvec(6) = 3
      CALL MANVEC( iinvec, error, isol, iexact, idum, kmesh, kprob )
      write ( *, * ) ' Error =', error

c==== Finish the job (pg 2.5) ==========================================

      call finish(0)

c=======================================================================
      end

      double precision function func(ichois, x, y, z)
```

```
c=======================================================================
c     Fill SEPRAN vector
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      func = (1d0-exp(32d0*x))/(1d0-exp(32d0))

c=======================================================================
      end

      double precision function funccf(ichois, x, y, z)

c=======================================================================
c     Fill SEPRAN right hand side vector
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      funccf = 0d0

c=======================================================================
      end

      subroutine elem( coor, elemmt, elemvc, iuser, user, uold, matrix,
     &                 vector, index1, index2 )
c=======================================================================
c     Spectral element for the 1D convection diffusion equation
c=======================================================================

      implicit none
      double precision coor(*), elemmt(*), elemvc(*), user(*), uold(*)
      integer iuser(*), index1(*), index2(*)
      logical matrix, vector


c==== COMMON BLOCKS ===================================================

      integer ielem, itype, ielgrp, inpelm, icount, ifirst,
     +        notmat, notvec, irelem, nusol, nelem, npoint
      common /cactl/ ielem, itype, ielgrp, inpelm, icount, ifirst,
     +               notmat, notvec, irelem, nusol, nelem, npoint
      save /cactl/
c
c                /cactl/
c    Contains element dependent information for the various element
c    subroutines. cactl is used to transport information from main
c    subroutines to element subroutines
c
c    icount   i   Number of unknowns in element
c    ielem    i   Element number
c    ielgrp   i   Element group number
c    ifirst   i   Indicator if the element subroutine is called for the
c                 first time in a series (0) or not (1)
c    inpelm   i   Number of nodes in element
```

```
c    irelem   i   Relative element number with respect to element group
c                 number ielgrp
c    itype    i   Type number of element
c    nelem    i   Number of elements in the mesh
c    notmat  i/o  Indicator if the element matrix is zero (1) or not (0)
c    notvc   i/o  Indicator if the element vector is zero (1) or not (0)
c    npoint   i   Number of nodes in the mesh
c    nusol    i   Number of degrees of freedom in the mesh
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c**** LOCAL PARAMETERS ************************************************

      integer i, j, m, ij, im, jm, ji
      double precision  d1phi(16384), x(128), xi(128), w(128), det,
     $                  funccf, rdum, dcof, ccof
      external funccf

c**** Nodel points ****************************************************

      do 5 m = 1,inpelm
         x(m) = coor(index1(m))
    5 continue

c**** Weights and derivatives of the basisfunctions ********************

      call elp640( 1, xi, w, d1phi, rdum, inpelm)

c**** Determinant ******************************************************

      det = 2d0/(x(inpelm) - x(1))

      if ( matrix ) then

c**** Fill element matrix (diffusion part) *****************************

         dcof = user(1)
         do 30 i = 1,inpelm
            do 20 j = 1,inpelm
               ij = inpelm*(i-1) + j
               elemmt(ij) = 0d0
               do 10 m = 1,inpelm
                  im = inpelm*(i-1) + m
                  jm = inpelm*(j-1) + m
                  elemmt(ij) = elemmt(ij) +
     $                         w(m)*dcof*d1phi(jm)*d1phi(im)*det
   10          continue
   20       continue
   30    continue

c**** Fill element matrix (convection part) ****************************

         ccof = user(2)
         do 50 i = 1,inpelm
            do 40 j = 1,inpelm
               ij = inpelm*(i-1) + j
               ji = inpelm*(j-1) + i
               elemmt(ij) = elemmt(ij) + w(i)*ccof*d1phi(ji)
   40       continue
   50    continue
```

```
      endif

c==== Fill right hand side vector (may depend on x) ====================

      if ( vector ) then
         do 60 i = 1,inpelm
            elemvc(i) = w(i)*funccf(1,x(i),x,x)/det
   60    continue
      endif

c=======================================================================
      end
```

Note that the system matrix is not symmetric anymore. So the input parameter in `commat` must change!
Test this element with the 'classical' example:

$$\begin{aligned} -\eta\frac{\partial^2 u}{\partial x^2} + \alpha\frac{\partial u}{\partial x} &= f(x) && 0 \le x \le 1 \\ u &= 0 && x = 0 \\ u &= 1 && x = 1 \end{aligned} \tag{5.11}$$

Analyse again h- and N-convergence.

5.3.3 Example 3.3: Steady 2D diffusion

A beautiful 2-dimensional example (the tea-towel) is given below:

$$\nabla^2 u = 32\pi^2 sin(4\pi x) sin(4\pi y) \tag{5.12}$$

with exact solution:

$$u = sin(4\pi x) sin(4\pi y) \tag{5.13}$$

This program uses the standard spectral element 604. In order to link the objects that are needed use:
`seprun prog33 -i prog33.dat -o prog33.out -cl -O /usr/local/sepran/SPEC`

```
      program prog33
c=====================================================================
c  A 2D poisson solver using spectral elements
c=====================================================================

c==== LOCAL VARIABLES ================================================

      implicit none
      integer idum, kmesh(100), kprob(200), intmat(5,1), iinbld(11),
     $        matr(5), isol(5), iuser(100), iexact(5), iinvec(6),
     $        kemesh(100), indcol(27), i, idm(5),itrsh(5)

      double precision user(100), rdum, error

c==== Start sepran (pg 2.2) ==========================================

      call start(0, 1, 1, 1)

c==== Generate the mesh (pg 3.1) =====================================

      kmesh(1)=100
      call mesh(0, idum, rdum, kmesh)

c==== Generate finite element mesh for plotting ======================

      call femesh(kmesh, kemesh, 22)

c==== Define the problem  (pg 4.1) ===================================

      kprob(1) = 1000
      call probdf( 0, kprob, kmesh, idum )

c==== Create the exact solution (pg 5.3) =============================

      call creavc( 0, 1, idum, iexact, kmesh, kprob, 5, rdum,
     $             idum, rdum )

c==== Fill boundary conditions =======================================

      call presdf(kmesh, kprob, isol)

c==== Define the solution method  (pg 4.4) ===========================

      call commat( 1, kmesh, kprob, intmat )

c==== Fill coefficients  (pg 5.11) ===================================
```

```
      iuser(1)=100
      user(1)=100
      call filcof(iuser, user, kprob, kmesh, 1)

c==== Build the system  (pg 5.1) ======================================

      iinbld(1) = 0
      call build( iinbld, matr, intmat, kmesh, kprob, itrsh, idum,
     $            isol ,idm, iuser, user )

c==== Solve the system  (pg 5.1) ======================================

      call solve( 1, matr, isol, itrsh, intmat, kprob )

c==== Fill color array ================================================

      indcol(1) =    27
      do 100 i = 4,14
         indcol(3+i) = i
  100 continue

c==== Plot the solution  (pg 9.5) =====================================

      call plotc2(1, kemesh, kprob, iexact, rdum, -12, indcol, 20d0,
     $            1d0, 1)
      call plotc2(1, kemesh, kprob, isol, rdum, -12, indcol, 20d0,
     $            1d0, 1)

c==== Compute the error ===============================================

      iinvec(1) = 6
      iinvec(2) = 5
      iinvec(3) = 0
      iinvec(4) = 1
      iinvec(5) = 1
      iinvec(6) = 1
      call manvec( iinvec, error, isol, iexact, idum, kmesh, kprob )
      write ( 6, * ) ' Error =', error

c==== Stop the program (pg 2.5) =======================================

      call finish( 0 )

c=======================================================================
      end

      double precision function funcbc(ichois,x,y,z)
c=======================================================================
c     Fill boundary conditions
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      funcbc =0.0d0

c=======================================================================
      end
```

```
      double precision function func(ichois,x,y,z)
c=======================================================================
c     Fill sepran array
c=======================================================================

      implicit none
      integer ichois,k,l
      double precision x,y,z,pi
c
      pi = datan(1.0d0)*4.0d0
      if (ichois.eq.2)
     $ func =32.0d0*pi**2* dsin(4.0d0*pi*x)*dsin(4.0d0*pi*y)
      if (ichois.eq.5)
     $ func = dsin(4.0d0*pi*x)*dsin(4.0d0*pi*y)

c=======================================================================
      end
c
      double precision function funccf(ichois,x,y,z)
c=======================================================================
c     Fill sepran right hand side
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z,pi

      pi = datan(1.0d0)*4.0d0
      funccf=32.0d0*pi**2*dsin(4.0d0*pi*x)*dsin(4.0d0*pi*y)

c=======================================================================
      end

# input file for prog33.f ==============================================
mesh2d
  points
    p1=(-1.0d0, -1.0d0)
    p2=( 1.0d0, -1.0d0)
    p3=( 1.0d0, 1.0d0)
    p4=(-1.0d0, 1.0d0)
  curves
    c1=line1(p1,p2,nelm=3)
    c2=line1(p2,p3,nelm=3)
    c3=line1(p3,p4,nelm=3)
    c4=line1(p4,p1,nelm=3)
  surfaces
    s1=rectangle5(c1,c2,c3,c4)
  intermediate points
    sidepoints=11 ,subdivision=legendre,midpoints=filled
  renumber
  plot(jmark=3, numsub=1)
end
problem
  types
    elgrp1 = ( type = 604)
  essbouncond
    degfd1=curves0(c1,c4)
end
essential boundary conditions
  degfd1=curves0(c1,c4)
```

```
end
coefficients
  elgrp1(nparm=8)
  coef1 = 1.0d0
  coef3 = 1.0d0
  coef7 = (func=2)
  icoef8 = 0
end
# ========================================================================
```

Use this program as a sample program for your own 2D Poisson application.

5.4 Example 4: Unsteady convection-diffusion problems

We will now consider the 1D unsteady convection diffusion equation given by:

$$\frac{\partial u}{\partial t} - \eta \frac{\partial^2 u}{\partial x^2} + \alpha \frac{\partial u}{\partial x} = f \tag{5.14}$$

First we will solve this equation for the boundary conditions:

$$\begin{aligned} u(0,t) &= 0 \\ u(1,t) &= 1 \end{aligned} \tag{5.15}$$

and initial condition:

$$u(x,0) = 0 \tag{5.16}$$

5.4.1 Example 4.1: Euler implicit time integration

Using an Euler implicit time integration method we obtain:

$$M \frac{u^{n+1} - u^n}{\Delta t} + S u^{n+1} = f^{n+1} \tag{5.17}$$

or:

$$[M + \Delta t S] u^{n+1} = M u^n + \Delta t f^{n+1} \tag{5.18}$$

The matrix of the system to be solved now consists of two parts: the mass matrix M and Δt times the matrix S which is known from the steady equations. The computation of the mass matrix is performed in the element (`elem1` now instead of `elem`). Check the element given below with respect to this.
Some special remarks have to be made on the incorporation of the boundary conditions. Up until now we were used to the fact that SEPRAN takes care of that while assembling the system matrix. It does this by renumbering the unknowns in such a way that all the Dirichlet boundary conditions are at the lower part of the solution array:

$$u = \begin{bmatrix} u_f \\ u_p \end{bmatrix} \tag{5.19}$$

The system then is partitioned as follows:

$$Su = \begin{bmatrix} S_{ff} & S_{fp} \\ S_{pf} & S_{pp} \end{bmatrix} \begin{bmatrix} u_f \\ u_p \end{bmatrix} = \begin{bmatrix} f_f \\ f_p \end{bmatrix} = f \tag{5.20}$$

The actual system dat is solved then reads:

$$S_{ff}u_f = f_f - S_{fp}u_p \tag{5.21}$$

This procedure is followed by subroutine `build` if it is called with `iinbld(3) = 0`. As for time dependent problems we have to add to matrices $(M + \Delta t S)$ we can not follow the procedure given above. Therefor now `build` is called with `iinbld(3)=1` and the incorporation of the boundary conditions is performed in the main program.

The program is given by:

```
      program prog41
c=======================================================================
c     The one dimensional time dependent convection diffusion equation
c     Euler implicit
c
c     [ [ matm ] + dt*[ mats ] ]*isoln = dt*irhsdn + [ m ]*isolo
c=======================================================================

c==== LOCAL VARIABLES ================================================

      implicit none
      integer idum, kmesh(100), kprob(100), intmat(5), isolo(5), ipict,
     $        npict, icurvs(2), isoln(5), iinbld(4), mats(5), irhsdo(5),
     $        matm(5), iinvec(2), ihelp1(5), ihelp2(5), matl(5),
     $        irhsdn(5), irhsd(5), istep, nstep
      double precision rdum, user(10), t0, t, dt, tend, rinvec(2)

c==== COMMON VARIABLES ===============================================

      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer
      save /cmcdpi/
c
c                /cmcdpi/
c    Unit numbers to use for certain standard in- and output files.
c
c    irefwr     Unit number to use for "normal" writes
c    irefre     Same for standard reads (mostly keyboard input)
c    irefer     Same for error messages
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c==== Set some parameters ============================================

      t0 = 0d0
      tend = 5d-1
      dt = 5d-2
      nstep = (tend-t0)/dt
      t = t0

c==== Start Sepran (pg 2.2) ==========================================

      call start(0, 1, 1, 1)

c==== Generate the mesh (pg 3.1) =====================================

      kmesh(1) = 100
      call mesh(0, idum, rdum, kmesh )

c==== Plot the mesh (pg 9.3) =========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) =================================

      call prinrv(idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) ====================================

      kprob(1) = 100
```

```
      call probdf(0, kprob, kmesh, idum )

c==== Create the exact solution (pg 5.3) ==============================

      call create(0, kmesh, kprob, isolo)

c==== Plot the exact (initial) solution (pg 9.8) ======================

      ipict = 1
      npict = nstep+1
      icurvs(1)= 0
      icurvs(2) = 1
      call plotfn(0, ipict, npict, kmesh, kprob, isolo, 1,
     $             icurvs, 20d0, 1d0, 'x', 'u(x)', rdum, rdum )

c==== Print the exact solution (pg 8.6) ===============================

      call prinov(isolo, kmesh, kprob, 1, 'Initial solution', rdum,
     $            idum)

c==== Define the solution method ======================================

      call commat(2, kmesh, kprob, intmat )

c==== Define the essential boundary conditions (pg 5.5) ===============

      call prestm(0, kmesh, kprob, isoln)

c==== Fill coefficients ===============================================

      user(1)=1d0
      user(2)=1d0

c==== build mats, matm and irhsdo, no boundary conditions (pg 5.1) =====

      iinbld(1) = 4
      iinbld(2) = 1
      iinbld(3) = 1
      iinbld(4) = 1
      call build( iinbld, mats, intmat, kmesh, kprob, irhsdo, matm,
     $            idum, idum, idum, user )

c==== Calculate [ matl ] = [ [ matm ] + dt*[ mats ] ] =================

      call copymt( mats, matl, kprob )
      call addmat( kprob, matl, matm, intmat, dt, 0d0, 1d0, 0d0 )

c==== Loop over timesteps =============================================

      do 100 istep=1,nstep
         t = t + dt

c======= Build irhsdn, no boundary conditions =========================

         iinbld(1) = 4
         iinbld(2) = 2
         iinbld(3) = 1
         iinbld(4) = 1
         call build( iinbld, idum, intmat, kmesh, kprob, irhsdn, matm,
     $               idum, idum, idum, user )
```

```
c======= Calculate   ihelp1 = irhsdn*dt + [ matm ]*isolo ===============

         call maver( matm, isolo, ihelp2, intmat, kprob, 4 )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = dt
         call manvec( iinvec, rinvec, ihelp2, irhsdn, ihelp1,
     $                kmesh, kprob )

c======= Incorporate the boundary conditions ============================

         call maver( matl, isoln, ihelp2, intmat, kprob, 6 )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = -1d0
         call manvec( iinvec, rinvec, ihelp1, ihelp2, irhsd,
     $                kmesh, kprob )

c======= Solve the system (pg 6.8) ======================================

         call solve(0, matl, isoln, irhsd, intmat, kprob )

c======= Print the solution (pg 8.4) ====================================

         write ( irefwr, * )' Output for timestep ',istep,' time = ',t
         call prinrv( isoln, kmesh, kprob, 4, 0, 'Solution' )

c======= Plot the solution (pg 8.4) =====================================

         ipict = ipict + 1
         call plotfn( 0, ipict, npict, kmesh, kprob, isoln, 1,
     $                icurvs, 20d0, 1d0, 'x', 'u(x)', rdum, rdum )

c======= save solution for next timestep ================================

         call copyvc(isoln,isolo)

  100 continue

c==== Finish the job (pg 2.5) ========================================

      call finish(0)

c=======================================================================
      end

      double precision function func(ichois, x, y, z)

c=======================================================================
c     Fill SEPRAN vector
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      func = 0d0
```

```
c=======================================================================
      end

      double precision function funccf(ichois, x, y, z)

c=======================================================================
c     Fill SEPRAN right hand side vector
c=======================================================================

      implicit none
      integer ichois
      double precision x,y,z

      funccf = 0d0

c=======================================================================
      end

      subroutine elem1( coor, elemmt, elemvc, elemms, iuser, user,
     $                  uold, matrix, vector, index1, index2, notmas )
c=======================================================================
c     Spectral element for the time-dependent convection diffusion
c     equation
c=======================================================================

      double precision coor(*), elemmt(*), elemvc(*), elemms(*),
     $                 user(*), uold(*)
      integer iuser(*), index1(*), index2(*), notmas
      logical matrix, vector

c=====COMMON BLOCKS ====================================================

      integer ielem, itype, ielgrp, inpelm, icount, ifirst,
     +        notmat, notvec, irelem, nusol, nelem, npoint
      common /cactl/ ielem, itype, ielgrp, inpelm, icount, ifirst,
     +               notmat, notvec, irelem, nusol, nelem, npoint
      save /cactl/
c
c                /cactl/
c    Contains element dependent information for the various element
c    subroutines. cactl is used to transport information from main
c    subroutines to element subroutines
c
c    icount   i   Number of unknowns in element
c    ielem    i   Element number
c    ielgrp   i   Element group number
c    ifirst   i   Indicator if the element subroutine is called for the
c                 first time in a series (0) or not (1)
c    inpelm   i   Number of nodes in element
c    irelem   i   Relative element number with respect to element group
c                 number ielgrp
c    itype    i   Type number of element
c    nelem    i   Number of elements in the mesh
c    notmat  i/o  Indicator if the element matrix is zero (1) or not (0)
c    notvc   i/o  Indicator if the element vector is zero (1) or not (0)
c    npoint   i   Number of nodes in the mesh
c    nusol    i   Number of degrees of freedom in the mesh
c - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
c==== LOCAL PARAMETERS ==============================================

      integer i, j, m, ij, im, jm, ji
      double precision  d1phi(16384), x(128), xi(128), w(128), det,
     $                  funccf, rdum, dcof, ccof
      external funccf

c==== Nodeal points =================================================

      do 5 m = 1,inpelm
         x(m) = coor(index1(m))
    5 continue

c==== Weights and derivatives of the basisfunctions ==================

      call elp640( 1, xi, w, d1phi, rdum, inpelm)

c==== Determinant ====================================================

      det = 2d0/(x(inpelm) - x(1))
      if ( matrix ) then

c==== Fill element matrix (diffusion part) ===========================

         dcof = user(1)
         do 30 i = 1,inpelm
            do 20 j = 1,inpelm
               ij = inpelm*(i-1) + j
               elemmt(ij) = 0d0
               do 10 m = 1,inpelm
                  im = inpelm*(i-1) + m
                  jm = inpelm*(j-1) + m
                  elemmt(ij) = elemmt(ij) +
     $                         w(m)*dcof*d1phi(jm)*d1phi(im)*det
   10          continue
   20       continue
   30    continue

c==== Fill element matrix (convection part) ==========================

         ccof = user(2)
         do 50 i = 1,inpelm
            do 40 j = 1,inpelm
               ij = inpelm*(i-1) + j
               ji = inpelm*(j-1) + i
               elemmt(ij) = elemmt(ij) + w(i)*ccof*d1phi(ji)
   40       continue
   50    continue

      endif

c==== Fill right hand side vector (may depend on x) ==================

      if ( vector ) then
         do 60 i = 1,inpelm
            elemvc(i) = w(i)*funccf(1,x(i),x,x)/det
   60    continue
      endif

c==== Fill mass matrix ===============================================
```

```
      if ( notmas .eq. 0 ) then
         do 80 i=1,inpelm
            elemms(i) = w(i)/det
   80    continue
      endif

c=======================================================================
      end
```

The corresponding input file is:

```
# input for prog41.dat ================================================
meshld
  points
    p1=(0.0)
    p2=(1.0)
  curves
    c1=line1(p1,p2,nelm=20)
intermediate points
  sidepoints=1,subdivision=legendre,midpoints=filled
  plot(jmark=0)
end
problem
  types
    elgrp1=(type=7)
    numdegfd=1
  essbouncond
    degfd1=points(p1,p2)
end
create vector
  type=solution vector
  function=1
end
essential boundary conditions
  points(p1), degfd1=(value=0.0)
  points(p2), degfd1=(value=1.0)
end
# =====================================================================
```

Run this program and study convergence of the solution to the steady solution.

5.5 Example 5: Unsteady convection problems

A nice convection problem that can be solved is the traveling Gauss-distribution:

$$u(x,t) = e^{-\frac{1}{2}\left(\frac{x-m_0-\alpha t}{\sigma}\right)^2} \tag{5.22}$$

This function is a solution of:

$$\frac{\partial u}{\partial t} + \alpha \frac{\partial u}{\partial x} = 0 \tag{5.23}$$

Next you will find both an Euler implicit and a Crank-Nicolson implementation. Verify the data given in section 3.6.5 of the course material.

5.5.1 Example 5.1: Euler implicit

```
      program prog51
c=====================================================================
c     The one dimensional time dependent convection equation
c     Euler implicit
c
c     [ [ matm ] + dt*[ mats ] ]*isoln = dt*irhsdn + [ m ]*isolo
c=====================================================================

c==== LOCAL VARIABLES ================================================

      implicit none
      integer idum, kmesh(100), kprob(100), intmat(5), isolo(5), ipict,
     $        npict, icurvs(2), isoln(5), iinbld(4), mats(5), irhsdo(5),
     $        matm(5), iinvec(2), ihelp1(5), ihelp2(5), matl(5),
     $        irhsdn(5), irhsd(5), istep, nstep
      double precision rdum, user(10), t0, t, dt, tend, rinvec(2)

c==== COMMON VARIABLES ===============================================

      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer
      save /cmcdpi/
c
c                /cmcdpi/
c    Unit numbers to use for certain standard in- and output files.
c
c    irefwr     Unit number to use for "normal" writes
c    irefre     Same for standard reads (mostly keyboard input)
c    irefer     Same for error messages
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c==== Set some parameters ============================================

      t0 = 0d0
      tend = 6d-1
      dt = 5d-3
      nstep = (tend-t0)/dt
      t = t0

c==== Start Sepran (pg 2.2) ==========================================

      call start(0, 1, 1, 1)
```

```
c==== Generate the mesh (pg 3.1) ======================================

      kmesh(1) = 100
      call mesh(0, idum, rdum, kmesh )

c==== Plot the mesh (pg 9.3) ==========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) ==================================

      call prinrv(idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) =====================================

      kprob(1) = 100
      call probdf(0, kprob, kmesh, idum )

c==== Create the exact solution (pg 5.3) ==============================

      call create(0, kmesh, kprob, isolo)

c==== Plot the exact (initial) solution (pg 9.8) ======================

      ipict = 1
      npict = 3
      icurvs(1)= 0
      icurvs(2) = 1
      call plotfn(0, ipict, npict, kmesh, kprob, isolo, 1,
     $             icurvs, 20d0, 1d0, 'x', 'u(x)', rdum, rdum )

c==== Print the exact solution (pg 8.6) ===============================

      call prinov(isolo, kmesh, kprob, 1, 'Initial solution', rdum,
     $            idum)

c==== Define the solution method ======================================

      call commat(2, kmesh, kprob, intmat )

c==== Define the essential boundary conditions (pg 5.5) ===============

      call prestm(0, kmesh, kprob, isoln)

c==== Fill coefficients ===============================================

      user(1)=0d0
      user(2)=1d0

c==== build mats, matm and irhsdo, no boundary conditions (pg 5.1) =====

      iinbld(1) = 4
      iinbld(2) = 1
      iinbld(3) = 1
      iinbld(4) = 1
      call build( iinbld, mats, intmat, kmesh, kprob, irhsdo, matm,
     $            idum, idum, idum, user )

c==== Calculate [ matl ] = [ [ matm ] + dt*[ mats ] ] =================
```

```
      call copymt( mats, matl, kprob )
      call addmat( kprob, matl, matm, intmat, dt, 0d0, 1d0, 0d0 )

c==== Loop over timesteps ================================================

      do 100 istep=1,nstep
         t = t + dt

c======= Build irhsdn, no boundary conditions ============================

         iinbld(1) = 4
         iinbld(2) = 2
         iinbld(3) = 1
         iinbld(4) = 1
         call build( iinbld, idum, intmat, kmesh, kprob, irhsdn, matm,
     $               idum, idum, idum, user )

c======= Calculate   ihelp1 = irhsdn*dt + [ matm ]*isolo ==============

         call maver( matm, isolo, ihelp2, intmat, kprob, 4 )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = dt
         call manvec( iinvec, rinvec, ihelp2, irhsdn, ihelp1,
     $                kmesh, kprob )

c======= Incorporate the boundary conditions =============================

         call maver( matl, isoln, ihelp2, intmat, kprob, 6 )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = -1d0
         call manvec( iinvec, rinvec, ihelp1, ihelp2, irhsd,
     $                kmesh, kprob )

c======= Solve the system (pg 6.8) ======================================

         call solve(0, matl, isoln, irhsd, intmat, kprob )

c======= Print the solution (pg 8.4) ====================================

         write ( irefwr, * )' Output for timestep ',istep,' time = ',t
         call prinrv( isoln, kmesh, kprob, 4, 0, 'Solution' )

c======= Plot the solution (pg 8.4) =====================================

         if ( istep .eq. nstep/2 .or. istep .eq. nstep) then
            ipict = ipict + 1
            call plotfn( 0, ipict, npict, kmesh, kprob, isoln, 1,
     $                   icurvs, 20d0, 1d0, 'x', 'u(x)', rdum, rdum )
         endif

c======= save solution for next timestep ================================

         call copyvc(isoln,isolo)

  100 continue
```

```
c==== Finish the job (pg 2.5) ========================================

      call finish(0)

c=====================================================================
      end

      double precision function func(ichois, x, y, z)

c=====================================================================
c     Fill SEPRAN vector
c=====================================================================

      implicit none
      integer ichois
      double precision x,y,z
      double precision a, m0, s

      m0 = .15d0
      s = .04d0
      a = - ( ( (x-m0)/s )**2 )/2d0
      func = exp(a)

c=====================================================================
      end

      double precision function funccf(ichois, x, y, z)

c=====================================================================
c     Fill SEPRAN right hand side vector
c=====================================================================

      implicit none
      integer ichois
      double precision x,y,z

      funccf = 0d0

c=====================================================================
      end
```

5.5.2 Example 5.2: Crank-Nicolson

```
      program prog52
c=====================================================================
c     The one dimensional time dependent convection equation
c     Crank-Nicolson  (Theta-method)
c
c     [ [ matm ] + theta*dt*[ mats ] ]*isoln =
c                  theta*dt*irhsdn + (1-theta)*dt*irhsdo +
c                  [ m - (1-theta)*dt*[ mats ] ]*isolo
c=====================================================================

c==== LOCAL VARIABLES ================================================

      implicit none
      integer idum, kmesh(100), kprob(100), intmat(5), isolo(5), ipict,
     $        npict, icurvs(2), isoln(5), iinbld(4), mats(5), irhsdo(5),
     $        matm(5), iinvec(2), ihelp1(5), ihelp2(5), matl(5),
     $        irhsdn(5), irhsd(5), istep, nstep, matr(5)
      double precision rdum, user(10), t0, t, dt, tend, rinvec(2),
     $        theta, dtt, dt1mt

c==== COMMON VARIABLES ===============================================

      integer irefwr, irefre, irefer
      common /cmcdpi/ irefwr, irefre, irefer
      save /cmcdpi/
c
c                /cmcdpi/
c    Unit numbers to use for certain standard in- and output files.
c
c    irefwr     Unit number to use for "normal" writes
c    irefre     Same for standard reads (mostly keyboard input)
c    irefer     Same for error messages
c  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

c==== Set some parameters ============================================

      t0 = 0d0
      tend = 6d-1
      dt = 5d-3
      nstep = (tend-t0)/dt
      t = t0
      theta = 0.5d0
      dtt = dt*theta
      dt1mt = dt*(1d0-theta)

c==== Start Sepran (pg 2.2) ==========================================

      call start(0, 1, 1, 1)

c==== Generate the mesh (pg 3.1) =====================================

      kmesh(1) = 100
      call mesh(0, idum, rdum, kmesh )

c==== Plot the mesh (pg 9.3) =========================================

      call plotm1(0, kmesh, idum, 20d0)

c==== Print the coordinates (pg 8.4) =================================
```

```
      call prinrv(idum, kmesh, kmesh, 4, -1, 'Coordinates')

c==== Define the problem (pg 4.1) ======================================

      kprob(1) = 100
      call probdf(0, kprob, kmesh, idum )

c==== Create the exact solution (pg 5.3) ===============================

      call create(0, kmesh, kprob, isolo)

c==== Plot the exact (initial) solution (pg 9.8) =======================

      ipict = 1
      npict = 3
      icurvs(1)= 0
      icurvs(2) = 1
      call plotfn(0, ipict, npict, kmesh, kprob, isolo, 1,
     $             icurvs, 20d0, 1d0, 'x', 'u(x)', rdum, rdum )

c==== Print the exact solution (pg 8.6) ================================

      call prinov(isolo, kmesh, kprob, 1, 'Initial solution', rdum,
     $            idum)

c==== Define the solution method =======================================

      call commat(2, kmesh, kprob, intmat )

c==== Define the essential boundary conditions (pg 5.5) ================

      call prestm(0, kmesh, kprob, isoln)

c==== Fill coefficients ================================================

      user(1)=0d0
      user(2)=1d0

c==== build mats, matm and irhsdo, no boundary conditions (pg 5.1) =====

      iinbld(1) = 4
      iinbld(2) = 1
      iinbld(3) = 1
      iinbld(4) = 1
      call build( iinbld, mats, intmat, kmesh, kprob, irhsdo, matm,
     $            idum, idum, idum, user )

c==== Calculate [ matl ] = [ [ matm ] + theta*dt*[ mats ] ] ============

      call copymt( mats, matl, kprob )
      call addmat( kprob, matl, matm, intmat, dtt, 0d0, 1d0, 0d0 )

c==== Calculate [ matr ] = [ [ matm ] + (1-theta)*dt*[ mats ] ] ========

      call copymt( mats, matr, kprob )
      call addmat( kprob, matr, matm, intmat, -dt1mt, 0d0, 1d0, 0d0 )

c==== Loop over timesteps ==============================================
```

```
      do 100 istep=1,nstep
         t = t + dt

c======= Build irhsdn, no boundary conditions ==========================

         iinbld(1) = 4
         iinbld(2) = 2
         iinbld(3) = 1
         iinbld(4) = 1
         call build( iinbld, idum, intmat, kmesh, kprob, irhsdn, matm,
     $               idum, idum, idum, user )

c======= Calculate   ihelp1 = theta*dt*irhsdn +
c                             (1-theta)*dt*irhsdo +
c                             [ matr ]*isolo              ===============

         call maver( matr, isolo, ihelp1, intmat, kprob, 4 )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = dtt
         call manvec( iinvec, rinvec, ihelp1, irhsdn, ihelp2,
     $                kmesh, kprob )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = dt1mt
         call manvec( iinvec, rinvec, ihelp2, irhsdo, ihelp1,
     $                kmesh, kprob )

c======= Incorporate the boundary conditions ============================

         call maver( matl, isoln, ihelp2, intmat, kprob, 6 )
         iinvec(1) = 2
         iinvec(2) = 27
         rinvec(1) = 1d0
         rinvec(2) = -1d0
         call manvec( iinvec, rinvec, ihelp1, ihelp2, irhsd,
     $                kmesh, kprob )

c======= Solve the system (pg 6.8) ======================================

         call solve(0, matl, isoln, irhsd, intmat, kprob )

c======= Print the solution (pg 8.4) ====================================

         write ( irefwr, * )' Output for timestep ',istep,' time = ',t
         call prinrv( isoln, kmesh, kprob, 4, 0, 'Solution' )

c======= Plot the solution (pg 8.4) =====================================

         if ( istep .eq. nstep/2 .or. istep .eq. nstep) then
            ipict = ipict + 1
            call plotfn( 0, ipict, npict, kmesh, kprob, isoln, 1,
     $                   icurvs, 20d0, 1d0, 'x', 'u(x)', rdum, rdum )
         endif

c======= save solution for next timestep ================================

         call copyvc(isoln,isolo)
```

```
  100 continue

c==== Finish the job (pg 2.5) ========================================

      call finish(0)

c=====================================================================
      end

      double precision function func(ichois, x, y, z)

c=====================================================================
c     Fill SEPRAN vector
c=====================================================================

      implicit none
      integer ichois
      double precision x,y,z
      double precision a, m0, s

      m0 = .15d0
      s = .04d0
      a = - ( ( (x-m0)/s )**2 )/2d0
      func = exp(a)

c=====================================================================
      end

      double precision function funccf(ichois, x, y, z)

c=====================================================================
c     Fill SEPRAN right hand side vector
c=====================================================================

      implicit none
      integer ichois
      double precision x,y,z

      funccf = 0d0

c=====================================================================
      end
```

Change one of the programs above to a time integration which you think is the best for the convection problem given above.

Appendix A

Linear vector analysis

A.1 Vector spaces

In order to discuss the concept of weighted residual formulations of partial differential equations, without claiming to be complete, first some basic theory concerning linear vector spaces will be given. Most of the theory is extensively described in Reddy and Rasmussen (1982).

Linear vector spaces

Definition 1: linear vector space

A linear vector space V is a set of elements (vectors) $u, v, w, ...$ satisfying the following properties:

1. For each pair of vectors $u \in V$ and $v \in V$ there exists a unique vector $u + v = w \in V$. Moreover the following properties must hold for vector addition:
   a) $u + v = v + u$
   b) $(u + v) + w = u + (v + w)$
   c) $\exists_{\theta \in V}$ such that $u + \theta = u$
   d) $\exists_{-u \in V}$ such that $u + (-u) = \theta$
2. For each vector $u \in V$ and real number $\alpha \in I\!R$ there exists a unique vector $w = \alpha u \in V$. Moreover the following properties must hold for scalar multiplication:
   a) $\alpha(\beta u) = (\alpha\beta)u \ \ \forall_{\beta \in I\!R}$
   b) $(\alpha + \beta)u = \alpha u + \beta u \ \ \forall_{\beta \in I\!R}$
   c) $\alpha(u + v) = \alpha u + \alpha v \ \ \forall_{v \in V}$
   d) $1u = u$

Example 1: linear vector space

1. $V = I\!R^3$ is a linear vector space with elements v represented by $v = (v_1, v_2, v_3)$ with vector addition:

   $$v + w = (v_1 + w_1, v_2 + w_2, v_3 + w_3)$$

   and scalar multiplication:

   $$\alpha v = (\alpha v_1, \alpha v_2, \alpha v_3)$$

2. $V = C^m([a,b]), m \geq 0$ is a linear vector space of m times differential functions $u : [a,b] \to \mathbb{R}$ with vector addition:

$$(u+v)(x) = u(x) + v(x)$$

and scalar multiplication:

$$(\alpha u)(x) = \alpha u(x)$$

Banach spaces

Definition 2: norm

Given a linear vector space V in which a function $n(u) : V \to \mathbb{R}$ is defined. The function $n(u) := \|u\|_V$ is called a norm in V if:

a) $\|u+v\|_V \leq \|u\|_V + \|v\|_V$

b) $\|\alpha u\|_V = |\alpha|\|u\|_V$

c) $\|u\|_V \geq 0$

d) $\|u\|_V = 0 \Leftrightarrow u = 0$

Definition 3: Cauchy sequence

A Cauchy sequence in V with norm $\|\cdot\|_V$ is a sequence of elements $\{u_1, u_2, ...\}$ for which:

$$\forall_{\epsilon>0}\exists_{N(\epsilon)>0}\forall_{k,m>N(\epsilon)}\|u_k - u_m\|_V < \epsilon$$

Definition 4: convergent sequence

A sequence is called convergent in V with norm $\|\cdot\|_V$ if:

$$\exists_{u\in V} \lim_{k\to\infty} \|u_k - u\|_V = 0$$

Definition 5: complete space

A vector space V is called complete if each Cauchy sequence converges in V.

Definition 6: Banach space

A linear vector space is called a Banach space if it is equipped with a norm for which the space is complete.

Example 2: Banach space

1. $V = \mathbb{R}^3$ is a Banach space for the norm:
$$\|u\|_2 = \sqrt{u_1^2 + u_2^2 + u_3^2}$$
2. $V = L^p(a,b), p \geq 1$ is a Banach space of piecewise continuous functions $u : (a,b) \to \mathbb{R}$ with norm :
$$\|u\|_{L^p(a,b)} = \left(\int_a^b |u(x)|^p dx\right)^{\frac{1}{p}}$$

Hilbert spaces

Definition 7: inner product

Given a linear vector space V in which a function $i(u,v) : V \times V \to \mathbb{R}$ is defined. The function $i(u,v) := (u,v)_V$ is called a inner product in V if:

a) $(u,v)_V = (v,u)_V$

b) $(\alpha u, v)_V = \alpha(u,v)_V$

c) $(u+v,w)_V = (u,w)_V + (v,w)_V$

d) $(u,u)_V \geq 0$

d) $(u,u)_V = 0 \Leftrightarrow u = 0$

Note that $\sqrt{(u,u)_V}$ is a proper norm in V.

Definition 8: Hilbert space

A Hilbert space is a linear vector space equipped with an inner product $(\cdot,\cdot)_V$ and for which the space is complete with respect to a norm defined as:

$$\|\cdot\|_V = \sqrt{(\cdot,\cdot)_V}$$

Example 3: Hilbert space

1. $V = \mathbb{R}^3$ is a Hilbert space for the inner product:

   $$(u \cdot v) = u_1 v_1 + u_2 v_2 + u_3 v_3$$

   and norm:

   $$\|u\|_2 = \sqrt{u_1^2 + u_2^2 + u_3^2}$$

2. $V = L^2(a,b)$ is a Hilbert space of piecewise continuous functions $u : (a,b) \to \mathbb{R}$ with inner product:

   $$(u,v)_{L^2(a,b)} = \int_a^b uv dx$$

   and norm :

   $$\|u\|_{L^2(a,b)} = \left(\int_a^b u^2 dx \right)^{\frac{1}{2}}$$

An often used property of the inner product is the Cauchy-Schwarz inequality.

Theorem 1: Cauchy-Schwarz

$$|(u,v)_V| \leq \|u\|_V \cdot \|v\|_V$$

Proof.

From the properties of the inner product for all $u \in V$, $v \in V$ and $\alpha \in \mathbb{R}$ it follows that:

$$0 \leq (u - \alpha v, u - \alpha v)_V = (u,u)_V - 2\alpha(u,v)_V + \alpha^2 (v,v)_V$$

This is a non-negative quadratic form in α so:

$$4(u,v)_V^2 - 4(v,v)(u,u)_V \leq 0$$

and thus:

$$|(u,v)_V| \leq \|u\|_V \cdot \|v\|_V$$

□

Sobolev spaces

Definition 9: Sobolev spaces

A Sobolev space of order m is a space of square integrable functions that possesses m derivatives that are representable as square integrable functions:

$$H^m(a,b) = \left\{ u \in L^2(a,b) \mid \frac{\partial^k u}{\partial x^k} \in L^2(a,b), 1 \leq k \leq m \right\}$$

$H^m(a,b)$ is endowed with the inner product:

$$(u,v)_{H^m(a,b)} = \sum_{k=0}^{m} \int_a^b \frac{\partial^k u}{\partial x^k} \frac{\partial^k v}{\partial x^k} dx$$

and norm:

$$\|u\|_{H^m(a,b)} = \sqrt{(u,u)_{H^m(a,b)}}$$

The following properties can be derived:

$$H^{m+1}(a,b) \subset H^m(a,b) \subset \ldots \subset H^0(a,b) \equiv L^2(a,b)$$

$$C^m([a,b]) \subset H^m(a,b)$$

$$H^m(a,b) \subset C^{m-1}([a,b])$$

A.2 Linear and bi-linear forms

Definition 10: linear form

Let V be a Hilbert space. The form $l(u) : V \to I\!R$, is called a linear form if $\forall_{u,v \in V}$:

$$l(\alpha u + \beta v) = \alpha l(u) + \beta l(v)$$

Definition 11: linear continuous form

Let V be a Hilbert space. The form $l(u) : V \to I\!R$, is called a linear continuous form if $\forall_{u \in V}$:

$$|l(u)| \leq C\|u\|_V$$

In other words, since $|l(u) - l(v)| = |l(u - v)| \leq C\|u - v\|_V$ and hence $\forall_{\epsilon > 0}$ with $|l(u) - l(v)|_V < \epsilon$, a δ can be found such that $\|u - v\| < \delta$. So a linear form is continuous if it is bounded.

Definition 12: bilinear form

Let V be a Hilbert space. The form $a(u, v) : V \times V \to I\!R$, is called a bilinear form if $\forall_{u,v,w \in V}$:

$$a(\alpha u + \beta v, w) = \alpha a(u, w) + \beta a(v, w)$$

and

$$a(u, \gamma v + \delta w) = \gamma a(u, v) + \delta a(u, w)$$

Definition 13: bilinear continuous form

Let V be a Hilbert space. The form $a(u, v) : V \times V \to I\!R$, is called a bilinear continuous form if $\forall_{u,v \in V}$:

$$|a(u, v)| \leq \beta \|u\|_V \|v\|_V$$

Definition 14: positive-definite form

Let V be a Hilbert space. The form $a(u, v) : V \times V \to I\!R$, is called a positive-definite, or V-coercive, or V-elliptic form if $\forall_{u \in V, \alpha > 0}$:

$$|a(u, u)| \geq \alpha \|u\|_V^2$$

The Lax-Milgram theorem

Theorem 2: Lax-Milgram

Let V be a Hilbert space and let $a(u, v) : V \times V \to I\!R$ be a linear continuous V-coercive form on V. Then for each continuous linear form $l(v) : V \to I\!R$ there exists a unique solution $u \in V$ to the problem:

$$a(u, v) = l(v) \qquad \forall_{v \in V}$$

Moreover this solution is stable in the sense that the following estimate holds:

$$\|u\|_V \leq \frac{\beta}{\alpha} \|f\|_V$$

showing that the solution u depends continuously on the data f.

Lemma of Céa

Let $\mathcal{L}$ be a linear continuous positive-definite differential operator, i.e.:

$$|(\mathcal{L}u, v)_V| \leq \beta \|u\|_V \|v\|_V$$
$$|(\mathcal{L}u, u)_V| \geq \alpha \|u\|_V^2$$

A standard Galerkin discrete weighted residual formulation of the differential equation $\mathcal{L}u = f$ then is given by:

Lemma 1: Lemma of Céa

The error of the Galerkin approximation behaves like the error of the best approximation in the norm for which stability is proven using the Lax-Milgram theorem.

Proof.

Since $V^h \subset V$ we also have

$$(\mathcal{L}u, w^h)_V = (f, w^h)_V \qquad \forall_{w^h \in V^h}$$

and hereby:

$$(\mathcal{L}(u^h - u), w^h)_V = 0 \qquad \forall_{w^h \in V^h}$$

Since this must hold for all $w^h \in V^h$ this must also hold for $w^h = u^h - v^h$ and thus:

$$(\mathcal{L}(u^h - u), u^h - v^h)_V = 0 \qquad \forall_{v^h \in V^h}$$

or alternatively:

$$(\mathcal{L}(u^h - u), u^h - u + u - v^h)_V = 0 \qquad \forall_{v^h \in V^h}$$

yielding:

$$(\mathcal{L}(u^h - u), u^h - u)_v = (\mathcal{L}(u^h - u), v^h - u)_V \qquad \forall_{v^h \in V^h}$$

Using the properties of the differential operator we finally obtain:

$$\|u^h - u\|_V \leq \frac{\beta}{\alpha} \|u - v^h\|_V \qquad \forall_{v^h \in V^h}$$

or equivalently:

$$\|u^h - u\|_V \leq \frac{\beta}{\alpha} \inf_{v^h \in V^h} \|u - v^h\|_V \tag{A.1}$$

□

Appendix B

Vector and tensor integrals

B.1 Leibnitz formulae

If Ω is a moving region with boundary Γ and u_Γ the velocity of the moving boundary, then:

$$\frac{d}{dt}\int\limits_{\Omega(t)} s d\Omega = \int\limits_{\Omega(t)} \frac{\partial s}{\partial t} d\Omega + \int\limits_{\Gamma(t)} s(u_\Gamma \cdot n) d\Gamma = 0 \tag{B.1}$$

B.2 Gauss-Ostrogradskii divergence theorem

If Ω is a closed region with boundary Γ then:

$$\int\limits_{\Omega} (\nabla \cdot u) d\Omega = \int\limits_{\Gamma} (u \cdot n) d\Gamma \tag{B.2}$$

$$\int\limits_{\Omega} (a(\nabla \cdot u) + (u \cdot \nabla)a) d\Omega = \int\limits_{\Gamma} a(u \cdot n) d\Gamma \tag{B.3}$$

$$\int\limits_{\Omega} (\nabla \cdot \tau^c) d\Omega = \int\limits_{\Gamma} (\tau \cdot n) d\Gamma \tag{B.4}$$

Bibliography

Braza, M., Chassaing, P., and Minh, H. (1986). Numerical study and physical analysis of the pressure and velociry fields in the near wake of a circular cylinder. *J. Fluid Mech.*, **165**, 79–130.

Brooks, A. N. and Hughes, T. J. R. (1982). Streamline upwind/petrov-galerkin formulations for convection dominated flows with special emphasis on the incompressible navier-stokes equations. *Comp. Meth. Appl. Mech. Eng.*, **32**, 199–259.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. (1988). *Spectral methods in FLuid Dynamics*. Springer Verlag.

Chorin, A. J. (1968). Numerical solution of he navier-stokes equations. *Math. Comp.*, **22**, 745–761.

Ciarlet, P. G. (1978). *The finite element method for elliptic problems*. North-Holland.

de Vahl Davis, G. (1983). Natural convection in a square cavity: a benchmark numerical solution. *Int. J. Num. Meth. Fluids*, **3**, 249–264.

Donea, J. and Quartapelle, L. (1992). An introduction to finite element methods for transient advection problems. *Comp. Meth. Appl. Mech. Eng.*, **45**, 169–203.

Engelman, M. and Jamnia, M.-A. (1990). Transient flow past a circular cylinder: a benchmark solution. *Int. J. Num. Meth. Fluids*, **11**, 985–1000.

Fortin, M. (1981). Old and new finite elements for incompressible flows. *Int. J. Num. Meth. Fluids*, **1**, 347–364.

Fortin, M. and Fortin, A. (1985). Experiments with several elements for viscous incompressible flows. *Int. J. Num. Meth. Fluids*, **5**, 911–928.

Fortin, M. and Glowinski, R. (1983). *Augmented Lagrangian methods:applications to the numerical solution of boundary value problems*. North-Holland.

Girault, V. and Raviart, P. A. (1986). *Finite element methods for Navier-Stokes equations*. Springer-Verlag.

Gottlieb, D. and Orszag, S. A. (1977). *Numerical analysis of spectral methods*. SIAM.

Gresho, P. M. (1990). On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. part 1: Theory. *Int. J. Numer. Meth. Fluids*, **11**, 587–620.

Gresho, P. M. and Chan, S. T. (1990). On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix. part 2: Implementation. *Int. J. Numer. Meth. Fluids*, **11**, 621–659.

Hawken, D. M., Tamaddon-Jahromi, H. R., Townsend, P., and Webster, M. F. (1990). A taylor-galerkin based algorithm for viscous incompressible flow. *Int. J. Numer. Meth. Fluids*, **10**, 327–351.

Hirsch, C. (1988). *Numerical computation of internal and external flows*, volume 1. John Wiley & sons.

Johnson, C. (1987). *Numerical solutions of partial differential equations by the finite element method.* Cambridge University Press.

Ladyzhenskaya, O. A. (1969). *The matematical theory of viscous incompressible flow.* Gordon and Breach.

Maday, Y. and Patera, A. T. (1989). Spectral element methods for theincompressible navier-stokes equations. In A. Noor, editor, *State-of-the-Art surveys on computaional mechnics*. ASME, New York.

Maday, Y., Patera, A. T., and Ronquist, E. M. (1990). An operator-integration-factor splitting method for time-dependent problems: application to incompressible fluid flow. *J. Sci. Comp.*, **5**, 263–292.

Paolucci, S. and Chenowith, D. (1989). Transition to chaos in a differentially heated vertical cavity. *J. Fluid Mech.*, **201**, 379–410.

Patera, A. T. (1984). A spectral element method for fluid dynamics: laminar flow in a channel expansion. *J. Comput. Phys.*, **54**, 468–488.

Perktold, K. and Peter, R. (1990). Numerical 3d-simulation of pulsatile wall shear stress in an arterial t-bifurcation model. *J. Biomed. Eng.*, **12**, 2–12.

Reddy, J. N. and Rasmussen, M. L. (1982). *Advanced engineering analysis.* John Wiley & Sons.

Strang, G. (1976). *Linear algebra and its applications.* Academic Press.

Timmermans, L. J. P. and van de Vosse, F. N. (1993). Spectral methods for advection-diffusion problems. In C. Vreugdenhil and B. Koren, editors, *Notes in Numerical Methods in Fluid Mechanics: 'Numerical advection-diffusion'*. Vieweg, Braunschweig.

Timmermans, L. J. P., van de Vosse, F. N., and Minev, P. D. (1994). Taylor-galerkin based spectral element methods for convection-diffusion problems. *Int. J. Num. Meth. Fluids*, **18**, 853–870.

Timmermans, L. J. P., Minev, P. D., and van de Vosse, F. N. (1995). An approximate projection scheme for incompressible flow using spectral elements. *Int. J. Num. Meth. Fluids*. in press.

van de Vosse, F. N. (1987). *Numerical analysis of carotid artery flow.* PhD thesis, University of Technology, Eindhoven.

van de Vosse, F. N., van Steenhoven, A. A., Segal, A., and Janssen, J. D. (1989). A finite element approximation of the steady laminar entrance flow in a 90^o curved tube. *Int. J. Num. Meth. Fluids*, **9**, 275–287.

van Kan, J. (1986). A second order accurate pressure-correction scheme for viscous incompressible flow. *SIAM J. Sci. Stat. Comp.*, **7**, 870–891.

Whitham, G. (1974). *Linear and nonlinear waves*. Wiley-Interscience.