Introduction to Finite Element Modelling in Geosciences

Antoine Billy Rozel (antoine.rozel@erdw.ethz.ch)

Patrick Sanan (patrick.sanan@erdw.ethz.ch)

Introduction to Finite Element Modelling in Geosciences
ETH Zürich, Sonneggstrasse 5, 8092 Zürich, Switzerland

These notes were generated on July 10, 2020.
3318ab29cd3eae9447285b0b5450283359093688

Contents

1

Preface

Objectives

The emphasis of this course is to learn the fundamentals of the Finite Element (FE) method, and as importantly, that you learn to write a FE code from scratch. We present what we regard as the *minimum* amount of theory necessary to understand and implement the FE method. As such, this class is definitely a "hands-on" practical introduction to finite elements.

We encourage people to work on the exercises provided in these notes at their own speed. It is common that not everyone is able to complete all of the exercises within a one week (all day) time period. In the past, we have had students take the class two (or three) years in a row. Don't be discouraged if your progress in completing the exercises diminishes after day three.

Prerequisites

For teaching purposes, we advocate using MATLAB to implement your first FE code. If you are new to MATLAB, or have forgotten some of the essentials, in Appendix A you can find a small MATLAB refresher tutorial.

Our rationale to use MATLAB for teaching is based upon the following three facts:

1. MATLAB provides users with a complete development environment including: (i) a text editor; (ii) run-time error checking of array indices; (iii) a powerful debugger.

2. MATLAB contains a wide variety of visualisation routines.

3. MATLAB contains data-structures to describe sparse matrices and includes efficient sparse-direct solvers.

New as of 2020: If you are already familiar with them, we encourage the use of open source scientific Python tools which offer functionality similar to MATLAB: NumPy, SciPy, Matplotlib, IPython, etc. You may complete your exercises and projects using these tools, but please announce your intention to do so to the instructors. This is so that they can confirm that you are sufficiently comfortable with these tools to effectively learn the FEM material.

It is also highly advantageous if you already followed a class on programming the finite difference method; e.g. 651-4273-00L: "Numerical Modelling in Fortran" (Paul Tackley) or 651-4241-00L: "Numerical Modelling I and II: Theory and Applications" (Taras Gerya)

Further Reading

In addition to these notes, we recommend the following textbooks:

- The Finite Element Method using MATLAB. Y. W. Kwon and H. Bang (1997), CRC Press.
- The Finite Element Method, Vol. 1. O. C. Zienkiewicz and R. L. Taylor (2000), McGraw-Hill.
- Programming the Finite Element Method. I. M. Smith and D. V. Griffiths (1998), John Wiley & Sons.
- The Finite Element Method. T. J. R. Hughes (2000), Prentice-Hall.
- Finite Elements and Fast Iterative Solvers. H. C. Elman, D. J. Silvester & A. Wathan (2005), Oxford University Press.
- The Mathematical Theory of Finite Element Methods. S. C. Brenner & L. R. Scott (2005), Springer.
- Computational Techniques for Fluid Dynamics, Vol. 1. C. A. J. Fletcher (2000), Springer.
- Practical Finite Element Modeling in Earth Science Using Matlab. Guy Simpson (2017). John Wiley & Sons. [1]

More background on finite elements and there application in computational geodynamics, can be found in these references:

- Computational Geodyamics. A. Ismail-Zadeh and P. J. Tackley (2010), Cambridge University Press.
- Numerical Modeling of Earth Systems: An introduction to computational methods with focus on solid Earth applications of continuum mechanics. Lecture notes, University of SouthernCalifornia, Becker and Kaus (2014). Available at `http://geodynamics.usc.edu/~becker/preprints/Geodynamics557.pdf`

Acknowledgements

This class, including these notes, is built upon previous finite element classes taught at ETH which were established by Guy Simpson (now at University of Geneva), Boris Kaus (now at Johannes Gutenberg University Mainz) and Stefan Schmalholz (now at University of Lausanne), and continued by Marcel Frehner and Dave A. May (now at the University of Oxford).

[1] This book is based upon these notes.

2

Basic Principles

2.1 Introduction

The purpose of this course is to learn how to solve differential equations with the Finite Element Method (FEM). For the most part we assume that the equations governing the processes of interest are known and given. We will focus on the practical process of how one goes from the equation, to obtaining an approximate (numerical) solution. Although there are a variety of numerical techniques that one can use to obtain numerical solutions, here we will focus solely on the FEM.

The process of obtaining a computational solution consists of two stages shown schematically in Figure 21. The first stage converts the continuous partial differential equation (PDE) and auxiliary (boundary and initial) conditions into a discrete system of algebraic equations. This first stage is called discretisation. The second stage involves solving the system of algebraic equations to obtain an approximate solution to the original partial differential equation. The solution is approximate since errors are introduced by the replacement of continuous differential terms in the governing partial differential equation by algebraic expressions connecting nodal values on a finite grid. Errors are also introduced during the solution stage but these tend to be small in comparison to discretisation errors, unless the method failed to converge.

To convert the governing partial differential equation to a system of algebraic equations (or ordinary differential equations) a number of choices are available. The most common are the finite difference, finite element, finite volume and spectral methods. In principle, the solution does not depend on the method chosen. Each method has its own advantages and disadvantages. The best approach is to choose the method which best suits the problem being investigated. This course will only deal with the finite element method, which is widely used in practice and is extremely powerful. The technique is slightly harder to learn than, for example, the finite difference technique. However, as you will see, the effort invested in initially learning the FEM pays off due to the wide range of problems that the method is capable of solving. The FEM is especially well suited (though not restricted) to solving mechanical problems and problems with domains which are geometrically complex. Another advantage of programming in the finite element method is that the main structure of the code remains the same, even when the method is applied to different physical problems. Thus, once you learn this basic structure, you can modify it to solve a variety of problems with minimal effort.

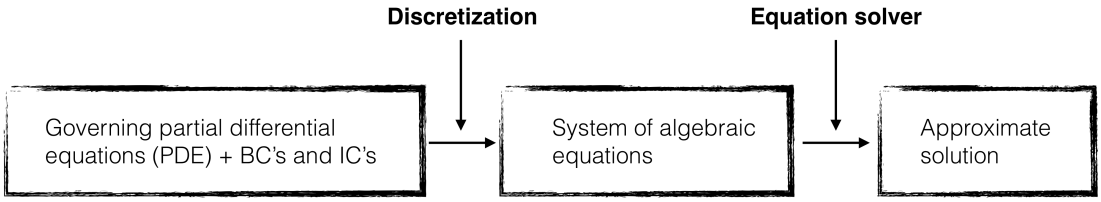


Figure 21: Overview of the computational solution technique.

2.2 The Finite Element Method

The finite element method is a technique for solving partial differential equations. Usually, only the spatial derivatives are discretised with the finite element method whereas finite differences are used to discretise time derivatives. Spatial discretisation is carried out locally over small regions of simple but arbitrarily shaped elements (the finite elements). This discretisation process results in matrix equations relating the loads (input) at specified points in the element (called nodes) to the displacements (output) at these same points. In order to solve equations over larger regions, one sums node-by-node the matrix equations for the smaller sub-regions (elements) resulting in a global matrix equation. This system of equations can then be solved simultaneously by standard linear algebra techniques to yield nodal displacements. This last step completes the numerical solution of the differential equation.

2.3 One-dimensional, time dependent diffusion

The various steps involved in performing the finite element method are best illustrated with a simple example. Consider the following partial differential equation

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial T}{\partial x}\right) + s(x), \tag{2.1}$$

which governs transient heat conduction in one dimension with a source term $s(x)$. The dependent variable in this equation is the temperature $T(x,t)$, the independent variables are time t, distance x, $\kappa(x)$ the thermal diffusivity and $s(x)$ the source. We are interested in computing the temperature function $T(x,t)$ which satisfies Equation (2.1) (i.e., the solution) over the domain $\Omega = [x_A, x_B]$ subject to either (i) Dirichlet boundary conditions of the form

$$\begin{aligned} T(x_A,t) &= T_A \\ T(x_B,t) &= T_B, \end{aligned} \tag{2.2}$$

where T_A, T_B are prescribed temperatures, or (ii) Neumann boundary conditions of the form

$$\begin{aligned} \kappa\frac{\partial T}{\partial x}\Big|_{x=x_A,t} &= q_A \\ \kappa\frac{\partial T}{\partial x}\Big|_{x=x_B,t} &= q_B, \end{aligned} \tag{2.3}$$

where q_A, q_B are prescribed fluxes, or (iii) some mixture of Dirichlet and Neumann conditions. We also require an initial condition

$$T(x, t=0) = T_0(x). \tag{2.4}$$

The first step of the finite element method involves choosing an element type which defines where and how the discretisation is carried out. The simplest element for one dimensional problems is a 2-node element (Figure 22a). As we will see, one can use more nodes per element which will have the effect of increasing accuracy, but also increasing the amount of equations and thus the cost of obtaining the numerical solution.

The second step of the finite element method involves approximating the continuous variable T in terms of nodal variables T_i using simple functions $N_i(x)$ called shape functions. If one focuses on one element (which contains 2 nodes), and one assumes that temperature varies linearly between two nodes, one can write

$$T(x) \approx N_1(x)T_1 + N_2(x)T_2, \tag{2.5}$$

or, using matrix notation

$$T(x) \approx \begin{bmatrix} N_1(x) & N_2(x) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \mathbf{NT}. \tag{2.6}$$

In these equations, T is the continuous variable which we are approximating within any given element in terms of the temperatures at the two nodes T_1 and T_2. Since we made the choice that temperature varies linearly between two nodes, we have to use the following shape functions,

$$N_1(x) = 1 - \frac{x}{L}, \qquad N_2(x) = \frac{x}{L}, \tag{2.7}$$

where L is the length of the element and x is the spatial variable which varies from 0 at node 1 to L at node 2 (Figure 22b). Note the following important properties of the shape functions

- $N_1 = 1$ at node 1 while $N_1 = 0$ at node 2.
- $N_2 = 0$ at node 1 while $N_2 = 1$ at node 2.
- $N_1(x) + N_2(x) = 1$ (over the entire element).
- The functions are only local (i.e., they only connect adjacent nodes).

Note that the shape functions are simply interpolating functions (i.e., they are used to interpolate the solution over a finite element). Also, as will become clear in the following lectures, the choice of the shape functions is directly related to the choice of an element type. For example in one-dimension, variation in a 2-node element cannot be uniquely described by a function with an order greater than linear (two parameter model), variation in a 3-node element cannot be uniquely described by a function with an order greater than quadratic (three parameter model), etc.

The next step of the finite element method is to substitute our approximation for the continuous variable into the governing differential equation. Thus, substituting Equation (2.6) into Equation (2.1) leads to

$$\frac{\partial}{\partial t}\left(\begin{bmatrix} N_1(x) & N_2(x) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}\right) - \frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial}{\partial x}\left(\begin{bmatrix} N_1(x) & N_2(x) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}\right)\right) - s(x) = R, \tag{2.8}$$

where R (the residual) is a measure of the error introduced during discretisation. Note that the original partial differential equation has now been replaced by an equation in

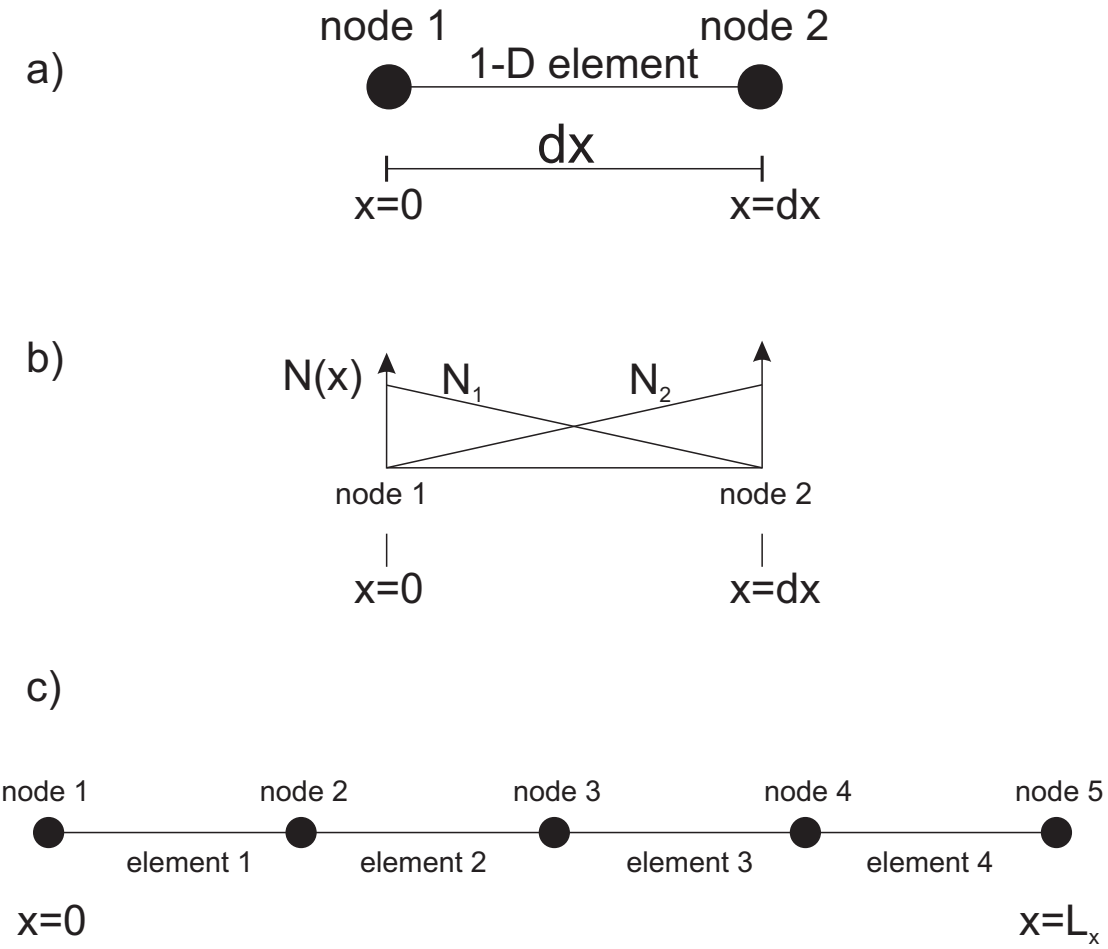


Figure 22: a) a 2-node, one dimensional finite element with b) linear shape functions, and c) a small one dimensional mesh consisting of four elements and a total of five nodes. See text for discussion.

the discretised (nodal) variables T_1 and T_2. The problem now reduces to finding values for T_1 and T_2 such that the residual is somehow minimized (ideally R is zero as in the original equation, but this is not possible to satisfy everywhere). This minimization should generate a system of equations where the number of equations equals the number of unknowns.

In the finite element method, the unknown coefficients T_i are determined by requiring that the integral of the weighted residual is zero on an element basis. To achieve this step practically, one must multiply (or "weight") the residual in Equation (2.8) by a set of weighting functions (each in turn), integrate over the element volume and equate to zero. Many methods (e.g., collocation, least squares and Galerkin) can be used to achieve this process, the difference between which depends on the choice of the weighting functions. In this course we will only consider the Galerkin method. In the Galerkin method, the weighting functions are chosen to be identical to the shape func-

tions N. By carrying out the steps just described one obtains

$$\int_0^L \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} \frac{\partial}{\partial t} \left(\begin{bmatrix} N_1 & N_2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \right) dx \\ - \int_0^L \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} \frac{\partial}{\partial x} \left(\kappa(x) \frac{\partial}{\partial x} \left(\begin{bmatrix} N_1 & N_2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \right) \right) dx - \int_0^L \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} s(x)\, dx = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad (2.9)$$

Note that in this example where the shape functions are linear, double differentiation of these functions would cause them to vanish (obviously not very desirable). This difficulty is resolved by applying Green's theorem (integration by parts). In one dimension, applied over the volume $\Omega = [x_A, x_B]$ this yields

$$\int_\Omega N_i \frac{\partial}{\partial x} \left(\kappa(x) \frac{\partial N_j}{\partial x} \right) dx = - \int_\Omega \kappa(x) \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x}\, dx + \kappa(x) N_i \frac{\partial N_j}{\partial x} \Big|_{x_A}^{x_B}. \quad (2.10)$$

We note that the final term in the above expression which is evaluated at the end points of the domain corresponds to a discrete flux. By invoking Green's theorem, we have implicitly introduced the Neumann boundary condition (see Equation (2.3)) into our discretisation. We will assume that $\kappa(x)$ and $s(x)$ are constant over each element volume $[x_A, x_B]$ and the constant values are denoted by $\bar{\kappa}$ and $\bar{s}$ respectively. Note that these choices do not preclude using a spatially dependent value for κ or s over the entire model domain, it only places a restriction on how these coefficients may vary *within* an element. Under these assumptions and using Equation (2.10), we can write Equation (2.9) as

$$\begin{bmatrix} \int_0^L N_1 N_1\, dx & \int_0^L N_1 N_2\, dx \\ \int_0^L N_2 N_1\, dx & \int_0^L N_2 N_2\, dx \end{bmatrix} \frac{\partial}{\partial t} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \\ + \bar{\kappa} \begin{bmatrix} \int_0^L \frac{\partial N_1}{\partial x} \frac{\partial N_1}{\partial x}\, dx & \int_0^L \frac{\partial N_1}{\partial x} \frac{\partial N_2}{\partial x}\, dx \\ \int_0^L \frac{\partial N_2}{\partial x} \frac{\partial N_1}{\partial x}\, dx & \int_0^L \frac{\partial N_2}{\partial x} \frac{\partial N_2}{\partial x}\, dx \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \\ - \bar{s} \begin{bmatrix} \int_0^L N_1\, dx \\ \int_0^L N_2\, dx \end{bmatrix} - \begin{bmatrix} N_1 q_A \big|_{x_A} \\ N_2 q_B \big|_{x_B} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad (2.11)$$

Note that one now has two equations for the two unknowns T_1 and T_2 as desired. On evaluating the integrals (using $\mathbf{N}$ defined in Equation (2.7)), Equation (2.11) becomes

$$\begin{bmatrix} \frac{L}{3} & \frac{L}{6} \\ \frac{L}{6} & \frac{L}{3} \end{bmatrix} \frac{\partial}{\partial t} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} + \bar{\kappa} \begin{bmatrix} \frac{1}{L} & -\frac{1}{L} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} - \bar{s} \begin{bmatrix} \frac{L}{2} \\ \frac{L}{2} \end{bmatrix} - \begin{bmatrix} N_1 q_A \big|_{x_A} \\ N_2 q_B \big|_{x_B} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (2.12)$$

which can be simplified using matrix notation to the following

$$\mathbf{M} \left(\frac{\partial \mathbf{T}}{\partial t} \right) + \mathbf{K}\, \mathbf{T} = \mathbf{F}, \quad (2.13)$$

where

$$\mathbf{M} = \begin{bmatrix} \frac{L}{3} & \frac{L}{6} \\ \frac{L}{6} & \frac{L}{3} \end{bmatrix}, \tag{2.14}$$

$$\mathbf{K} = \bar{\kappa} \begin{bmatrix} \frac{1}{L} & -\frac{1}{L} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix}, \tag{2.15}$$

$$\mathbf{F} = \bar{s} \begin{bmatrix} \frac{L}{2} \\ \frac{L}{2} \end{bmatrix} + \begin{bmatrix} N_1 q_A \big|_{x_A} \\ N_2 q_B \big|_{x_B} \end{bmatrix}, \tag{2.16}$$

and

$$\mathbf{T} = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}. \tag{2.17}$$

The next step we perform is to discretise the time derivative. Here this is achieved using a finite difference approximation. Assuming an implicit time discretisation, one approximates Equation (2.13) as

$$\mathbf{M} \left(\frac{\mathbf{T}^{n+1} - \mathbf{T}^n}{\Delta t} \right) + \mathbf{K}\, \mathbf{T}^{n+1} = \mathbf{F}, \tag{2.18}$$

where $\mathbf{T}^{n+1}$ is the future temperature at the nodes (i.e., the unknowns) and $\mathbf{T}^n$ is the vector of old (i.e., known) temperatures. Rearranging, one can write this as

$$\left(\frac{1}{\Delta t} \mathbf{M} + \mathbf{K} \right) \mathbf{T}^{n+1} = \frac{1}{\Delta t} \mathbf{M}\, \mathbf{T}^n + \mathbf{F}, \tag{2.19}$$

or more compactly as

$$\mathbf{L}\, \mathbf{T}^{n+1} = \mathbf{R}\, \mathbf{T}^n + \mathbf{F}, \tag{2.20}$$

where

$$\mathbf{L} = \begin{bmatrix} \frac{L}{3\Delta t} + \frac{\bar{\kappa}}{L} & \frac{L}{6\Delta t} - \frac{\bar{\kappa}}{L} \\ \frac{L}{6\Delta t} - \frac{\bar{\kappa}}{L} & \frac{L}{3\Delta t} + \frac{\bar{\kappa}}{L} \end{bmatrix} \tag{2.21}$$

and

$$\mathbf{R} = \begin{bmatrix} \frac{L}{3\Delta t} & \frac{L}{6\Delta t} \\ \frac{L}{6\Delta t} & \frac{L}{3\Delta t} \end{bmatrix} \tag{2.22}$$

and the $\mathbf{F}$ vector is

$$\mathbf{F} = \bar{s} L \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} + \begin{bmatrix} q_A \\ q_B \end{bmatrix}.$$

In Equation (2.20), everything appearing on the right hand side is known (and it combines to form a vector). The matrix $\mathbf{L}$ is referred to as the element stiffness matrix

whereas $\mathbf{T}$ is the unknown element vector (and the subscript $n+1$ has been dropped for clarity). For the purposes of following discussions we introduce the following notation:

$$\mathbf{L} = \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix}.$$

Thus, for example, the term L_{11} has the value $\frac{L}{3\Delta t} + \frac{\bar{\kappa}}{L}$. Similar notation is assumed for $\mathbf{R}$.

Remember that so far we have only carried out discretisation for a single element, whereas we generally want to divide the solution domain into many elements so as to obtain an accurate solution. Accordingly, let us consider a small one-dimensional mesh, consisting of four elements (once you get the idea you can easily consider more elements). This situation is depicted in Figure 22c. Now, instead of having just two unknowns, we have five, related to the five nodes in the mesh. One now generates a global matrix equation by summing node-by-node the matrix equation derived for a single element (i.e., Equation (2.12)). Thus, for example, note that whereas node 1 contains a contribution only from element 1, node 2 has contributions from both elements 1 and 2 (Figure 22c). Performing this process (using the notation introduced above and assuming that each element matrix is the same) leads to

$$\begin{bmatrix} L_{11} & L_{12} & 0 & 0 & 0 \\ L_{21} & L_{22}+L_{11} & L_{12} & 0 & 0 \\ 0 & L_{21} & L_{22}+L_{11} & L_{12} & 0 \\ 0 & 0 & L_{21} & L_{22}+L_{11} & L_{12} \\ 0 & 0 & 0 & L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix}^{n+1}$$

$$= \begin{bmatrix} R_{11} & R_{12} & 0 & 0 & 0 \\ R_{21} & R_{22}+R_{11} & R_{12} & 0 & 0 \\ 0 & R_{21} & R_{22}+R_{11} & R_{12} & 0 \\ 0 & 0 & R_{21} & R_{22}+R_{11} & R_{12} \\ 0 & 0 & 0 & R_{21} & R_{22} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix}^{n}$$

$$+ sL \begin{bmatrix} \frac{1}{2} \\ 1 \\ 1 \\ 1 \\ \frac{1}{2} \end{bmatrix} + \begin{bmatrix} q_A \\ 0 \\ 0 \\ 0 \\ q_B \end{bmatrix}, \quad (2.23)$$

which, using matrix notation, becomes

$$\mathbf{L}_G \, \mathbf{T}^{n+1} = \mathbf{R}_G \, \mathbf{T}^n + \mathbf{F}_G, \quad (2.24)$$

where the subscript G indicates that the matrices and vectors refer to the entire "global" or assembled problem and not simply to a single element. Note that the matrices $\mathbf{L}_G$ and $\mathbf{R}_G$ are symmetric which is an important (though not a necessary) property when it comes to solving the system of equations. Note that most non-zero terms are clustered near the main diagonal – a property which can be exploited when solving the system of linear equations. Also, note that the two imposed *boundary fluxes*, q_A and q_B, only appear in rows 1 and 5. The reason for this is that the row 1 corresponds to node 1, which is located at the left hand side of the boundary ($x_A = 0$) and row 5 is related to node 5, which is located on the right most side of the boundary ($x_B = L_x$).

Equation (2.24) can also be written in the form

$$\mathbf{L}_G \, \mathbf{T}^{n+1} = \mathbf{b}, \tag{2.25}$$

which is the classic form for a system of linear equations. In this last expression, $\mathbf{L}_G$ is referred to as the stiffness (or coefficient) matrix (and is known), $\mathbf{b}$ is referred to as the "right hand side vector" or "load vector" and $\mathbf{T}^{n+1}$ is the unknown "solution vector" or "reaction vector".

The final step before one can solve the linear system of equations represented by Equation (2.25) is to impose boundary conditions. There are several possible choices; fixed temperature, fixed temperature gradient, or some combination of the two. Fixed temperature boundaries may be implemented by performing the following steps: (1) zeroing the entries in the relevant equation (2) placing a 1 on the diagonal entry of the stiffness matrix and (3) setting the value at the correct position of the right hand side vector equal to the desired value. Alternatively, if one wishes to implement zero-flux boundary conditions one does not have to do anything explicitly (i.e., it is the default boundary condition created when one ignored the boundary terms in Equation (2.10)). Non zero flux boundary conditions are slightly more complex to evaluate. In this class, if boundary fluxes are imposed, they will always be equal to zero. These will be referred to as the *natural* boundary condition.

You are now ready to compute the solution to Equation (2.25). The basic steps that need to be incorporated into a computer program can be summarized as follows:

1. Define all physical parameters (e.g., diffusivity, source term, length of spatial domain) and numerical parameters e.g., number of elements and nodes).

2. Define the spatial coordinates x_i for your nodes, and specify the time domain where you want to obtain the discrete solution.

3. Within an element loop, define the element matrices $\mathbf{M}$ and $\mathbf{K}$ and the element load vector $\mathbf{F}$ (see Equations (2.14) to (2.16)). Use these to compute $\mathbf{L}$ and $\mathbf{R}$ (see Equations (2.21) & (2.22)). Sum these matrices node-by-node to form the global matrices $\mathbf{L}_G$ and $\mathbf{R}_G$ and the global vector $\mathbf{F}_G$ (see Equations (2.23)). If the element properties do not depend on time these global matrices only need to be calculated once and can be saved for later use.

4. Within a time loop, perform the operations on the right hand side of Equation (2.25) (i.e., firstly multiply $\mathbf{R}_G$ with the old temperature vector $\mathbf{T}^n$ and then add the resulting vector to $\mathbf{F}_G$) to form the right-hand-side vector $\mathbf{b}$. Note that the vector $\mathbf{T}^0$ must contain the discrete form of your initial condition, $T(x, 0)$.

5. Apply boundary conditions.

6. Solve Equation (2.25) for the new temperature, and then continue to the next time step.

2.4 Exercises

1. Read the text above and try to understand it.

2. We solved the system for second order derivatives. Can we employ linear shape functions if we want to solve, for example, the bending equation for a thin plate, which has fourth order derivatives? If no, which shape functions should be employed?

3

Time for Programming

3.1 Introduction

The purpose of this session is to learn how to program the finite element method (FEM) in one dimension using MATLAB. To illustrate this we use the diffusion equation (discretised in Chapter 2) which is governed by the following element equations

$$\left(\frac{\mathbf{M}}{\Delta t} + \mathbf{K}\right) \mathbf{T}^{n+1} = \frac{\mathbf{M}}{\Delta t} \mathbf{T}^{n} + \mathbf{F}, \tag{3.1}$$

where

$$\mathbf{M} = \begin{bmatrix} \frac{\Delta x}{3} & \frac{\Delta x}{6} \\ \frac{\Delta x}{6} & \frac{\Delta x}{3} \end{bmatrix}, \tag{3.2}$$

$$\mathbf{K} = \kappa \begin{bmatrix} \frac{1}{\Delta x} & \frac{-1}{\Delta x} \\ \frac{-1}{\Delta x} & \frac{1}{\Delta x} \end{bmatrix}, \tag{3.3}$$

$$\mathbf{F} = s \begin{bmatrix} \frac{\Delta x}{2} \\ \frac{\Delta x}{2} \end{bmatrix} + \begin{bmatrix} N_1 q_A \big|_{x_A} \\ N_2 q_B \big|_{x_B} \end{bmatrix}, \tag{3.4}$$

and

$$\mathbf{T} = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}. \tag{3.5}$$

The superscripts n and $n+1$ refer to the old (known) and future (unknown) temperatures, respectively. These equations can be written more compactly as

$$\mathbf{L}\,\mathbf{T}^{n+1} = \mathbf{R}\,\mathbf{T}^{n} + \mathbf{F}, \tag{3.6}$$

where the element stiffness matrix $\mathbf{L}$ is

$$\mathbf{L} = \frac{\mathbf{M}}{\Delta t} + \mathbf{K} \tag{3.7}$$

and the element right hand side matrix $\mathbf{R}$ is

$$\mathbf{R} = \frac{\mathbf{M}}{\Delta t}. \tag{3.8}$$

Remember that for a 2-node element, these relations form a system of two equations for the two unknown node temperatures T_1 and T_2. It is important to distinguish

these element equations from the global equations formed by summing the element equations node-by-node within a finite element mesh, denoted as

$$\mathbf{L}_G \, \mathbf{T}^{n+1} = \mathbf{R}_G \, \mathbf{T}^n + \mathbf{F}_G = \mathbf{b}, \tag{3.9}$$

where the subscript G indicates that the matrices and vectors are global matrices. The size of the matrices $\mathbf{L}_G$ and $\mathbf{R}_G$ are $nn \times nn$ (where nn is the number of nodes in the finite element mesh) whereas the matrices $\mathbf{M}$ and $\mathbf{K}$ have dimensions 2×2.

3.2 The FE procedure

The basic steps that must be performed in order to solve these equations within a computer program can be summarized as follows:

1. Define all physical parameters (e.g., diffusivity, source term, length of spatial domain) and numerical parameters e.g., number of elements and nodes).

2. Define the spatial coordinates x_i for your nodes, and specify the time domain where you want to obtain the discrete solution.

3. Define the relationship between the element node numbers and the global node numbers.

4. Define boundary node indices and the values of the potential (e.g., temperature) at the boundary nodes.

5. Initialise the global matrices $\mathbf{L}_G$, $\mathbf{R}_G$ and $\mathbf{F}_G$ so that their dimensions are defined and that the matrices are filled with zeros.

6. Within an element loop, define the element matrices $\mathbf{M}$ and $\mathbf{K}$ and the element load vector $\mathbf{F}$ (see Equations (3.2), (3.3) and (3.4)). Use these to compute $\mathbf{L}$ and $\mathbf{R}$. Sum these matrices (and $\mathbf{F}$) node-by-node to form the global stiffness matrices $\mathbf{L}_G$ and $\mathbf{R}_G$ and the global vector $\mathbf{F}_G$ (see Equation (3.9)). If the element properties do not depend on time these global matrices only need to be calculated once and can be saved for later use.

7. Within a time loop, perform the operations on the right hand side of Equation (3.9) (i.e., firstly multiply $\mathbf{R}_G$ with the old temperature vector $\mathbf{T}^n$ and then add the resulting vector to $\mathbf{F}_G$ to form the right hand side vector $\mathbf{b}$). Note that the vector $\mathbf{T}^0$ must contain the discrete form of your initial condition, $T(x, 0)$.

8. Apply boundary conditions.

9. Solve Equation (3.9) for the new temperature, and then continue to the next time step.

We will now describe each of these steps in more detail. In the following, we will indicate where MATLAB commands are used via the syntax

```
>> alpha = 1.0;
```

[1] Parameter definitions

This step requires little further explanation. All physical (e.g., κ, domain length, etc.) and numerical (number of nodes, timestep) parameters must be defined in this section using commands of the form

```
>> kappa = 10.0;
```

[2] Define spatial and time domain

Similarly this step requires little further explanation. The spatial coordinate vector x can be defined in MATLAB using the command

```
>> x = [ 0 : dx : lx ];
```

where `lx` is the length of the spatial domain and `dx` is the element length. One can create a similar vector for time.

[3] Local to global mapping

In order to sum the element equations node-by-node to form the global matrices one must define the relationship between local node numbers and global node numbers. We achieve this by creating a matrix called `g_num` which has the following form for a 5-node (4 element) finite element mesh constructed with 2-node (linear) elements:

$$\mathrm{g\_num} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix},$$

which in MATLAB is defined via the command,

```
>> g_num = [ 1, 2, 3, 4 ; 2, 3, 4, 5 ];
```

The matrix `g_num` has the dimensions $nod \times nels$ where nod is the number of nodes per element and $nels$ is the total number of elements in the mesh (both of which must be defined in "[1] Parameter definitions" - see above). Thus, for example, the two local node numbers 1 and 2 in element 3 correspond to the global node numbers 3 and 4 respectively (i.e., `g_num(1,3) = 3` and `g_num(2,3) = 4`).

[4] Define boundary conditions

One must define to which global nodes boundary conditions are to be applied (and what the boundary value is). This can be done with the following commands:

```
>> bcdof = [ 1, nn ];
```

and

```
>> bcval = [ 0.0, 0.0 ];
```

The array `bcdof` specifies which which nodes, namely the first (`1`) and last (`nn`), in the mesh are to be constrained. The array `bcval` defines the value of the temperature constraint we wish to apply at each boundary node.

[5] Initialisation

One must initialise the global matrices $\mathbf{L}_G$, $\mathbf{R}_G$ and global vector $\mathbf{F}_G$ using commands of the form

```
>> LG = zeros( nn, nn );
```

and

```
>> FG = zeros( nn, 1 );
```

where $\mathtt{nn}$ is the number of equations in the finite element mesh (which also equals the total number of nodes in this problem).

[6] Element matrix assembly

The element matrices $\mathbf{M}$ and $\mathbf{K}$ can be formed using commands of the form

```
>> M = [ dx/3.0, dx/6.0 ; dx/6.0, dx/3.0 ];
```

whereas the vector $\mathbf{F}$ can be formed with the command

```
>> F = s*[ dx/2.0 ; dx/2.0 ];
```

Once these are defined, the matrices $\mathbf{L}$ and $\mathbf{R}$ can be computed according to Equations (3.7) and (3.8). The next task is, within a loop over all elements, to sum the element matrices node-by-node to form the global matrices. This step is achieved using the matrix $\mathtt{g\_num}$ just described. For example, in order to form the matrix $\mathbf{L}_G$ one can use the following commands:

```
>> LG(g_num(:,iel),g_num(:,iel))
    = LG(g_num(:,iel),g_num(:,iel)) + L;
```

whereas to form the vector $\mathbf{F}_G$ one can use

```
>> FG(g_num(:, iel)) = FG(g_num(:, iel)) + F;
```

The variable `iel` is the current element number which varies within the loop from 1 to $nels$.

[7] Form right hand side vector

Within a time loop one can now form the global right hand side vector $\mathbf{b}$ with the command

```
>> b = RG * T + FG;
```

where $\mathbf{T}$ is the vector (with dimensions $nn \times 1$) of old temperatures and the other vectors and matrices are described above.

[8] Implement boundary conditions

The boundary conditions can be imposed using the following sequence of commands:

```
>> for i=1:length(bcdof)
>>    LG( bcdof(i), : ) = 0.0;
>>    LG( bcdof(i),bcdof(i)) = 1.0;
>>    b( bcdof(i) ) = bcval(i);
>> end
```

The above piece of MATLAB code does the following. First we loop over all nodes that have a (fixed or Dirichlet) boundary condition. We place zeros throughout the entire row where the boundary conditions should be imposed. These row indices are listed in the array `bcdof`. Following that, we place a value 1.0 on the diagonal of that row, and place the corresponding value of the Dirichlet boundary condition in the right hand side vector $\mathbf{b}$.

[9] Solution

The final step which must be carried out is to solve the system of equations (i.e., Equation (3.9)) simultaneously, which can be performed using the MATLAB command

```
>> T = LG \ b;
```

This operation will compute the solution vector $\mathbf{T}$, which contains the temperature within the domain at the $n+1$ timestep. A new time step can be computed by repeating Steps 7,8,9.

3.3 Exercises

1. Write a code that can solve the 1D diffusion problem with the finite element method. For simplicity we will assume that the boundary integral term is zero. Note that this assumption implies that we are solving the diffusion with zero flux boundary conditions.

2. Modify the code, to take into variable grid spacing, variable source terms, and variable thermal diffusivity (assume sources and properties to be constant within one element).

3. A simple (time-dependent) analytical solution for the diffusion equation exists for the case when the forcing term $s = 0$, and the initial temperature distribution is given by

$$T(x, t = 0) = T_{\max} \exp\left[-\frac{x^2}{\sigma^2}\right].$$

Here σ is the half-width of the distribution and $T_{\max}$ is the maximum amplitude of the temperature perturbation at $x = 0$. The solution for this problem is given by

$$T(x, t) = \frac{T_{\max}}{\sqrt{1 + 4t\kappa/\sigma^2}} \exp\left[\frac{-x^2}{\sigma^2 + 4t\kappa}\right].$$

This solution is derived assuming that $T \to 0$ at $x = \pm\infty$. Program the analytical solution and compare the analytical solution with the numerical solution with the same initial condition. Use $T_{\max} = 100$, $\sigma = 1$ and $-5 \leq x \leq 5$.

4

Numerical Integration

4.1 Overview

In Chapter 3 we saw that 1D finite element discretisation of the diffusion equation may lead to the element matrices

$$\mathbf{M} = \begin{bmatrix} \int_0^L N_1 N_1 \, dx & \int_0^L N_1 N_2 \, dx \\ \int_0^L N_2 N_1 \, dx & \int_0^L N_2 N_2 \, dx \end{bmatrix} \tag{4.1}$$

and

$$\mathbf{K} = \begin{bmatrix} \kappa \int_0^L \frac{\partial N_1}{\partial x} \frac{\partial N_1}{\partial x} \, dx & \kappa \int_0^L \frac{\partial N_1}{\partial x} \frac{\partial N_2}{\partial x} \, dx \\ \kappa \int_0^L \frac{\partial N_2}{\partial x} \frac{\partial N_1}{\partial x} \, dx & \kappa \int_0^L \frac{\partial N_2}{\partial x} \frac{\partial N_2}{\partial x} \, dx \end{bmatrix} \tag{4.2}$$

and element force vector (assuming zero flux Neumann boundary conditions)

$$\mathbf{F} = s \begin{bmatrix} \int_0^L N_1 \, dx \\ \int_0^L N_2 \, dx \end{bmatrix}, \tag{4.3}$$

where the shape functions N_1 and N_2 were defined as

$$N_1(x) = 1 - \frac{x}{L}, \qquad N_2(x) = \frac{x}{L}. \tag{4.4}$$

As we will see later, many different partial differential equations (elasticity, viscous fluid flow) lead to these same (or at least very similar) set of matrices. A mass matrix $\mathbf{M}$ is always associated with discretisation of a first-order time derivative (e.g., $\partial T/\partial t$) whereas the $\mathbf{K}$ matrix comes from the discretisation of second-order spatial derivatives (e.g., $\partial^2 T/\partial x^2$). In the example we considered it was possible to evaluate the integrals in Equations (4.1) & (4.2) analytically. This was largely facilitated by the fact that we (i) considered a 1D problem using (ii) linear basis functions and (iii) we assumed that κ was constant over each element. Using analytically derived element stiffness matrices can be advantageous as it avoids the need to approximate the integrals which

removes one source of numerical error and may also be much more efficient than an approximate technique. However, in general, if any of the above three assumptions are violated, using analytically derived stiffness matrices is not feasible. In such circumstances one must *approximate* the integrals appearing within the stiffness matrix. Throughout the remainder of this course, we will utilise integrals which have been approximated numerically using an optimal and efficient technique.

4.2 Quadrature

The method we will employ to perform numerical integration is a type of quadrature rule. In general, a quadrature rule replaces the integral by a weighted sum over a set of n quadrature points. For example, the function $f(\xi)$ integrated over the range $[a, b]$ is approximated via

$$\int_a^b f(\xi)\, d\xi \approx \sum_{i=1}^{n} f(\xi_i) w_i. \tag{4.5}$$

In Equation (4.5), ξ_i is the coordinate (or abscissa) of the i^{th} quadrature point and w_i is the quadrature weight associated with the i^{th} quadrature point. Note that the quadrature rule only requires one to evaluate the integrand $f(\cdot)$ at the location of each quadrature point.

The choice of quadrature scheme defines the values of n, ξ_i and w_i. The type of quadrature scheme which should be used is entirely dependent on the nature of the integrand $f(\cdot)$. In finite element analysis, the integrand is defined via products of the basis functions N (see $\mathbf{M}$) or from products of the derivative of the basis functions (see $\mathbf{K}$). In general, integrands of this type are polynomials of some order k, i.e. they contain terms like ξ^s where $s = 1, 2, \ldots, k$. The family of *Gauss-Legendre* quadrature schemes have been specifically designed to exactly evaluate polynomial integrands in one dimension. We will denote the n-point Gauss-Legendre (GLg), as the rule which uses n quadrature points. In 1D, the n-point GLg rule *exactly* integrates polynomials of order $\leq (2n - 1)$. Tabulated values of n, ξ_i and w_i for GLg rules which are suitable for polynomials up to third order can be found in most finite element books. In Table 41 we provide the Gauss-Legendre quadrature rules for 1D from $n = 1 \rightarrow 4$.

To be used in a general manner, tabulated quadrature rules assume an integration domain. GLg rules typically assume that the integration is performed over the domain $[-1, 1]$. The choice of integration domain is arbitrary. To use the tabulated rule in practice, one needs to define a coordinate transform from the domain of interest (say $[a, b]$ in Equation (4.5), or $[0, L]$ in Equation (4.1)) to the domain used by the tabulated quadrature rule. To illustrate the necessary steps to numerically evaluate a finite element stiffness matrix via GLg quadrature, we now consider evaluating

$$\int_0^L N_1(x) N_1(x)\, dx. \tag{4.6}$$

The first task is to transform the limit of integration, from what we will refer to as the global coordinate system $0 \leq x \leq L$ to local coordinate system used by the quadrature rule given by $-1 \leq \xi \leq 1$. Note that the shape functions defined in Equation (4.4) are written in terms of the global coordinate x varying over the interval $[0, L]$. We can linearly map the $[0, L]$ interval for x onto the $[-1, 1]$ interval for ξ using the following

n	ξ_i	w_i	k
1	0.0	2.0	1
2	$\pm\sqrt{\frac{1}{3}}$	1.0	3
3	0.0	0.88888888888889	
	$\pm$0.774596669241483	0.55555555555556	5
4	$\pm$0.339981043584859	0.65214515486255	
	$\pm$0.861136311594053	0.34785484513745	7

Table 41: A selection of different order accuracy 1D Gauss-Legendre quadrature rules. n denotes the number of quadrature points. The quantities w_i, ξ_i indicate the weight and coordinate of the i^{th} quadrature point respectively. k is the polynomial order for which the integration rule is exact.

transformation:

$$x = \frac{L}{2}(\xi + 1). \tag{4.7}$$

The reader should check that this transformation gives us the desired limits of $[-1, +1]$. This is achieved by inserting $x = 0$ (and $x = L$) into Equation (4.7) and solving for ξ. Substituting Equation (4.7) into Equation (4.4) leads to the shape functions defined in terms of local coordinates

$$N_1(\xi) = \frac{1}{2}(1 - \xi), \qquad N_2(\xi) = \frac{1}{2}(1 + \xi). \tag{4.8}$$

To perform the integration over the local coordinate system $[-1, +1]$, we must first transform the variable of integration. This is obtained by differentiating Equation (4.7) with respect to ξ and rearranging to yield

$$dx = \frac{L}{2}\, d\xi. \tag{4.9}$$

Thus, one can rewrite the integral in Equation (4.6) as

$$\int_0^L N_1(x)N_1(x)\, dx = \int_{-1}^1 N_1(\xi)N_1(\xi)\frac{L}{2}\, d\xi. \tag{4.10}$$

As will be seen in the following lectures, the term $\frac{L}{2}$ appearing in Equation (4.10), which is responsible for converting the integral from the global to the local coordinate system, is referred to as the determinant of the Jacobian, det$(\mathbf{J})$.

Now we are ready to numerically approximate the integral using the summation formula in Equation (4.5). For example, using the 2 point Gauss-Legendre rule (i.e.,

$n = 2$, see Table 41) the integral can be computed as

$$\begin{aligned}\int_{-1}^{1} N_1(\xi)N_1(\xi)\frac{L}{2}\,d\xi &\approx N_1\left(-\sqrt{1/3}\right)N_1\left(-\sqrt{1/3}\right)\times\frac{L}{2}\times 1.0\\ &\quad + N_1\left(\sqrt{1/3}\right)N_1\left(\sqrt{1/3}\right)\times\frac{L}{2}\times 1.0\\ &= \frac{L}{2}(0.6220084681\times 1.0 + 0.04465819869\times 1.0)\\ &= 0.333L,\end{aligned}$$

noting that $N_1(\xi)N_1(\xi)$ evaluated at $\sqrt{1/3}$ is 0.04465819869 and $N_1(\xi)N_1(\xi)$ at $\xi = -\sqrt{1/3}$ is 0.6220084681 and the weighting factors are both 1.0 (see Table 41). Note that this result is identical to the result obtained by the analytic calculation (i.e., $L/3$).

As a second example, consider calculation of the term

$$\int_0^L \frac{\partial N_1}{\partial x}\frac{\partial N_1}{\partial x}\,dx, \tag{4.11}$$

appearing in Equation (4.2). Note firstly that, by using the chain rule,

$$\frac{\partial N_1}{\partial x} = \frac{\partial N_1}{\partial \xi}\frac{\partial \xi}{\partial x}. \tag{4.12}$$

In general, the derivative of the global coordinate with respect to local coordinate (i.e., $\partial x/\partial\xi$) is known as the Jacobian (denoted $\mathbf{J}$). The inverse relation, $\partial\xi/\partial x$ (appearing in Equation (4.12)) is known as the inverse of the Jacobian, and this must always be calculated to convert derivatives from global coordinates to local coordinates.

Therefore, one can rewrite Equation (4.11) as

$$\begin{aligned}\int_0^L \frac{\partial N_1}{\partial x}\frac{\partial N_1}{\partial x}\,dx &= \int_0^L \left(\frac{\partial N_1}{\partial \xi}\frac{\partial \xi}{\partial x}\right)\left(\frac{\partial N_1}{\partial \xi}\frac{\partial \xi}{\partial x}\right)dx\\ &= \int_{-1}^{1} \left(\frac{\partial N_1}{\partial \xi}\frac{\partial \xi}{\partial x}\right)\left(\frac{\partial N_1}{\partial \xi}\frac{\partial \xi}{\partial x}\right)\frac{L}{2}\,d\xi\\ &= \int_{-1}^{1} \frac{\partial N_1}{\partial \xi}\frac{\partial N_1}{\partial \xi}\frac{2}{L}\,d\xi,\end{aligned} \tag{4.13}$$

where we used the fact that $\partial x/\partial\xi = \frac{L}{2}$ (see Equation (4.9)). Note that $\partial N_1/\partial\xi = -1/2$, which is constant (see Equation (4.8)). Thus by using the 2-point GLg quadrature rule, the term represented by Equation (4.13) can be integrated to give

$$\begin{aligned}\int_{-1}^{1} \frac{\partial N_1}{\partial \xi}\frac{\partial N_1}{\partial \xi}\frac{2}{L}\,d\xi &\approx \frac{2}{L}\left(\left(-\frac{1}{2}\right)\left(-\frac{1}{2}\right)\times 1.0 + \left(-\frac{1}{2}\right)\left(-\frac{1}{2}\right)\times 1.0\right)\\ &= \frac{1}{L},\end{aligned} \tag{4.14}$$

which is once again the exact result.

From these simple examples, we see that numerical integration gives very accurate results with minimal computation (in this case using only 2 integration points). This accuracy is due the strength of the Gauss-Legendre method and the fact that the functions being integrated are polynomials of a known order.

For the remainder of this course we will assume that shape functions and their derivatives are given in terms of the local coordinates on the interval $[-1, 1]$. Moreover, we will use Gauss-Legendre quadrature which also requires integration over the interval $[-1, 1]$. Based on these assumptions, the following steps must be carried out in order to perform numerical integration of the terms in the element matrices:

1. Define the number of integration points and obtain the integration points and weights. Evaluate the shape functions and their derivatives (defined in terms of local coordinates) at each integration point, and save the results for later use.

2. Do a loop over all elements and initialize the element matrices $\mathbf{M}$, $\mathbf{K}$ and element vector $\mathbf{F}$.

3. Within the element loop, start a loop over all integration points. One must convert derivatives from the local coordinates to the global coordinates using the inverse of the Jacobian, by performing an operation such as that in Equation (4.12).

4. Perform the vector multiplication involving the shape functions or their derivatives (evaluated at integration points), multiplied by the relevant weight, and multiplied by the determinant of the Jacobian (to convert integrals from global to local coordinates, see Equation (4.10)), which leads to the matrices $\mathbf{M}$, $\mathbf{K}$ and $\mathbf{F}$. The result should be added to the multiplication from proceeding integration points.

5. Once one has exited the loop over integration points, the integration is complete (i.e., the element matrices have been integrated).

6. The element matrices can then be added to the global element stiffness matrix. This process is then continued for each element.

In the 1D example discussed here, it might seem like "overkill" to use numerical integration. However, as you will soon see, utilising quadrature rules to evaluate element stiffness matrices and force vectors has immense advantages when (i) using high order elements (e.g. quadratic), and or (ii) when considering 2D and 3D problems. An immediate advantage in 2D (and 3D) which comes from using quadrature rules, is that we can deform the finite element mesh and still easily evaluate the integrals, without having to modify our code. Therefore it is good to practice using quadrature rules within the context of this 1D example.

4.3 Exercises

1. Perform the numerical integration of Equation (4.11) with a 3-point quadrature rule, instead of 2-point rule. How do the results differ?

2. Using the program developed in the last session, write a FE code which solves the diffusion equation in 1D where the element matrices and vectors are integrated numerically. Check that the results are identical to those obtained when using analytically computed element matrices/vectors.

3. Explain which order quadrature rule should be used if one was to use quadratic elements to solve the diffusion equation? Modify your code to solve the 1D diffusion equation with quadratic elements, instead of linear elements. The 1D

quadratic shape functions defined in the local coordinates system are given by

$$N_1(\xi) = \frac{1}{2}\xi\left(\xi - 1\right) \tag{4.15}$$

$$N_2(\xi) = 1 - \xi^2 \tag{4.16}$$

$$N_3(\xi) = \frac{1}{2}\xi\left(\xi + 1\right). \tag{4.17}$$

Note that each 1D quadratic element is composed of three nodes, instead of two nodes (as with the linear elements). Compare the results of the solution obtained using a 3-point GLg quadrature rule and a 2-point GLg quadrature rule.

5

The Diffusion Equation: From 1D to 2D

5.1 Introduction

One of the strengths of the finite element method is the relative ease with which it is possible to extend your code from one dimension to two (or three) dimensions. To demonstrate the finite element method in two dimensions, we once again use the example of the diffusion equation. Remember that the diffusion equation is derived by combining a statement for the conservation of energy

$$\rho c_p \frac{\partial T}{\partial t} = -\nabla^T \mathbf{q} + s = -\begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} q_x \\ q_y \end{bmatrix} + s, \tag{5.1}$$

with a general anisotropic constitutive relationship

$$\mathbf{q} = \begin{bmatrix} q_x \\ q_y \end{bmatrix} = -\begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} T = -\mathbf{D}\nabla T, \tag{5.2}$$

to yield

$$\rho c_p \frac{\partial T}{\partial t} = \nabla^T (\mathbf{D}\nabla T) + s. \tag{5.3}$$

This equation governs transient heat conduction in two dimensions with a source term $s(x, y)$. The dependent variable in this equation is the temperature T, the independent variables are time t and distance x, and $k_{ij}(x, y)$ are the components of the thermal conductivity tensor, $\rho(x, y)$ is density and $c_p(x, y)$ heat capacity. If the thermal conductivity is isotropic and constant in space, Equation (5.3) in 2D reduces to

$$\frac{\partial T}{\partial t} = \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + \frac{s}{\rho c_p}, \tag{5.4}$$

where $\kappa = k/(\rho c_p)$ is the thermal diffusivity. For the isotropic case, we have that

$$\mathbf{D} = \begin{bmatrix} \kappa & 0 \\ 0 & \kappa \end{bmatrix}. \tag{5.5}$$

Nevertheless, as we will see, it is more natural and general to leave the equation in the form shown in Equation (5.3). In fact one of the benefits of the finite element method is that introducing an anisotropic diffusivity tensor does not introduce any additional complexity to the discretisation procedure.

5.2 Derivation of the element matrices

As in our one dimensional example, the first step of the finite element method is to choose an element type. We choose here one of the simplest two-dimensional elements which is the 4-node quadrilateral (see Figure 51) element. The four shape functions $N_i(x, y)$ which go together with this type of element are polynomials of the form

$$N_i(x, y) = a_i + b_i x + c_i y + d_i xy,$$

where a_i, b_i, c_i, d_i are coefficients which relate to each node i. This element appears to be linear, but due to the presence of the cross term xy, the quadrilateral basis functions in 2D are referred to as *bilinear*. For the moment we shall simply refer to the shape functions using the following notation

$$\mathbf{N}(x, y) = \begin{bmatrix} N_1(x, y) & N_2(x, y) & N_3(x, y) & N_4(x, y) \end{bmatrix}, \tag{5.6}$$

and we note that the shape functions are expressed in terms of the global coordinate system (i.e., they are a function of x and y). The second step of the finite element method involves approximating the continuous variable T in Equation (5.3) in terms of nodal variables T_i using the shape functions just defined, i.e.,

$$\begin{aligned} T(x, y) &\approx \begin{bmatrix} N_1(x, y) & N_2(x, y) & N_3(x, y) & N_4(x, y) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \\ &= \mathbf{NT}. \end{aligned} \tag{5.7}$$

Substituting this approximation for T into Equation (5.3) yields the following equation

$$\rho c_p \frac{\partial}{\partial t} (\mathbf{NT}) = \nabla^T (\mathbf{D} \nabla \mathbf{N}) \mathbf{T} + s. \tag{5.8}$$

Multiplying this equation by the weighting functions (the shape functions) and integrating over the element volume Ω^e leads to

$$\iint_{\Omega^e} \rho c_p \mathbf{N}^T \frac{\partial}{\partial t} (\mathbf{NT}) \, dxdy = \iint_{\Omega^e} \mathbf{N}^T \nabla^T (\mathbf{D} \nabla \mathbf{N}) \mathbf{T} \, dxdy + \iint_{\Omega^e} \mathbf{N}^T s \, dxdy. \tag{5.9}$$

Integration by parts over the domain Ω^e and denoting the element boundaries lying on the domain boundary via Γ^e gives

$$\begin{aligned} \left[\iint_{\Omega^e} \rho c_p \mathbf{N}^T \mathbf{N} \, dxdy \right] \left(\frac{\partial \mathbf{T}}{\partial t} \right) + \left[\iint_{\Omega^e} \nabla (\mathbf{N})^T (\mathbf{D} \nabla \mathbf{N}) \, dxdy \right] \mathbf{T} \\ = \iint_{\Omega^e} \mathbf{N}^T s \, dxdy + \oint_{\Gamma^e} \mathbf{N}^T \, \widehat{\mathbf{q}}^T \mathbf{n} \, dS, \end{aligned} \tag{5.10}$$

where $\widehat{\mathbf{q}}$ is an applied flux and $\mathbf{n}$ is the outward point normal to the boundary $\partial\Omega$. Recall that the boundary integral term corresponds to the Neumann boundary condition. This equation can be simplified to

$$\mathbf{M} \left(\frac{\partial \mathbf{T}}{\partial t} \right) + \mathbf{K} \, \mathbf{T} = \mathbf{F}, \tag{5.11}$$

where

$$\mathbf{M} = \iint_{\Omega^e} \rho c_p \mathbf{N}^T \mathbf{N}\, dxdy, \tag{5.12}$$

$$\mathbf{K} = \iint_{\Omega^e} (\nabla \mathbf{N})^T \mathbf{D} \nabla \mathbf{N}\, dxdy \tag{5.13}$$

and

$$\mathbf{F} = \iint_{\Omega^e} \mathbf{N}^T s\, dxdy + \oint_{\Gamma^e} \mathbf{N}^T\, \widehat{\mathbf{q}}^T \mathbf{n}\, dS. \tag{5.14}$$

Note that we leave $p, c_p, \mathbf{D}$ and s inside the integrals as we have not assumed anything about how these coefficients vary within the domain, or within the element.

Note that the element matrices and vectors have a virtually identical form to those derived in Chapter 3 in one dimension. In fact, there are only two significant differences. First, the integration domain is now defined in two dimensions. As we shall see, one can use the same numerical integration as used in the 1D problems. Second, the shape functions have derivatives in both the x and y directions.

Before continuing it is useful to explicitly define the form (i.e., the dimensions) of some of the matrices defined above. For example, the shape functions $\mathbf{N}$ are defined as

$$\mathbf{N}(x,y) = \begin{bmatrix} N_1(x,y) & N_2(x,y) & N_3(x,y) & N_4(x,y) \end{bmatrix}, \tag{5.15}$$

and the shape function derivatives are defined as

$$\nabla \mathbf{N}(x,y) = \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial x} & \dfrac{\partial N_3}{\partial x} & \dfrac{\partial N_4}{\partial x} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_2}{\partial y} & \dfrac{\partial N_3}{\partial y} & \dfrac{\partial N_4}{\partial y} \end{bmatrix}. \tag{5.16}$$

The conductivity matrix is defined as [1]:

$$\mathbf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}. \tag{5.17}$$

Thus, one can see that the dimensions of the element matrices $\mathbf{M}$ and $\mathbf{K}$ are 4×4 whereas the element vector $\mathbf{F}$ has dimensions 4×1.

The next step is to discretise the time derivative in Equation (5.11) using finite differences. This is carried out in an identical manner to that already performed in 1D (see script 1) leading to

$$\mathbf{M} \left(\frac{\mathbf{T}^{n+1} - \mathbf{T}^n}{\Delta t} \right) + \mathbf{K}\, \mathbf{T}^{n+1} = \mathbf{F}, \tag{5.18}$$

where $\mathbf{T}^{n+1}$ is the new temperature at the nodes (i.e. the unknowns) and $\mathbf{T}^n$ is the vector of the old (known) temperatures. Rearranging, one can write this as

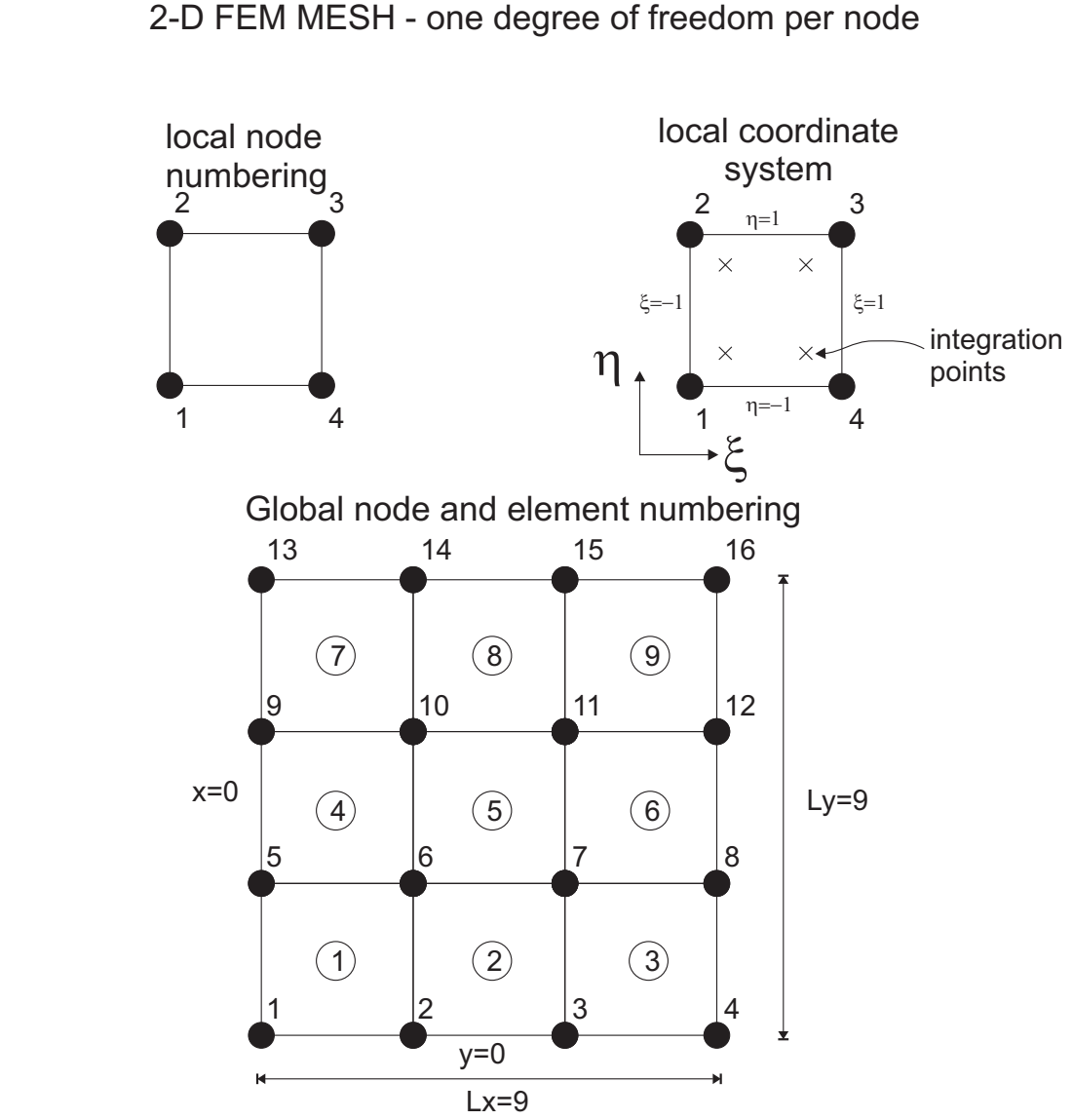
$$\left(\frac{\mathbf{M}}{\Delta t} + \mathbf{K} \right) \mathbf{T}^{n+1} = \frac{\mathbf{M}}{\Delta t}\, \mathbf{T}^n + \mathbf{F}, \tag{5.19}$$

or more compactly as

$$\mathbf{L}\, \mathbf{T}^{n+1} = \mathbf{R}\, \mathbf{T}^n + \mathbf{F}, \tag{5.20}$$

where $\mathbf{M}$, $\mathbf{K}$ and $\mathbf{F}$ have been previously defined.

[1] We note that to be physically valid we require that $k_{xx}k_{yy} - k_{xy}k_{yx} > 0$ and that $k_{xy} = k_{yx}$.



$$g\_num = \begin{bmatrix} 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 \\ 5 & 6 & 7 & 9 & 10 & 11 & 13 & 14 & 15 \\ 6 & 7 & 8 & 10 & 11 & 12 & 14 & 15 & 16 \\ 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 \end{bmatrix}$$

Relationship between elements and global node numbers

Global coordinates of nodes

$$g\_coord = \begin{bmatrix} 0.0 & 3.0 & 6.0 & 9.0 & 0.0 & 3.0 & 6.0 & 9.0 & 0.0 & 3.0 & 6.0 & 9.0 & 0.0 & 3.0 & 6.0 & 9.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 3.0 & 3.0 & 3.0 & 3.0 & 6.0 & 6.0 & 6.0 & 6.0 & 9.0 & 9.0 & 9.0 & 9.0 \end{bmatrix}$$

Figure 51: A two-dimensional finite element mesh with one degree of freedom on each node.

5.3 Integration of the element matrices

The element matrices just derived contain integrals of the shape functions, or derivatives of the shape functions, both expressed in terms of global coordinates (x and y). To evaluate such matrices two transformations are required. First, since we will provide shape functions (and their derivatives) in terms of local coordinates, it is necessary to devise some means of expressing the shape functions (and their derivatives) appearing in the integrals in terms of local coordinates. Second, the area over which the integration has to be carried out must be expressed in terms of local coordinates (with an appropriate change in the limits of integration).

As in one dimension, we will use Gauss-Legendre quadrature to approximate the integrals defining the element stiffness matrices. The formula for Gauss-Legendre quadrature in two dimensions is

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)\,d\xi d\eta \approx \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} f(\xi_i,\eta_j)w_i w_j = \sum_{n=1}^{n_{\text{ip}}} f(\xi_n,\eta_m)W_n \tag{5.21}$$

where n_x and n_y are number of integration points in each direction, ξ_i and η_j are the local spatial coordinates of the integration points, and w_i and w_j, or W_n are the weights. The quadrature weights and locations of the integration points are once again obtained from a table such as that provided in Chapter 4, Table 41. Note that the integration in two dimensions is very similar to the integration in one dimension. One simply sums over the total number of integration points $n_{\text{ip}} = n_x \times n_y$ and multiplies by the weights $W_n = w_i w_j$.

Derivatives are converted from one coordinate system to another by means of the chain rule, expressed in matrix form as

$$\begin{bmatrix} \dfrac{\partial}{\partial\xi} \\ \dfrac{\partial}{\partial\eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial\xi} & \dfrac{\partial y}{\partial\xi} \\ \dfrac{\partial x}{\partial\eta} & \dfrac{\partial y}{\partial\eta} \end{bmatrix} \begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{bmatrix}, \tag{5.22}$$

where $\mathbf{J}$ is the Jacobian matrix. The Jacobian matrix can be found by differentiating the global coordinates with respect to the local coordinates. In practice this requires one to multiply the coordinates of a particular element with the derivatives of the shape functions. For example, if we considered the 4 node quadrilateral element we would have:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial N_1}{\partial\xi} & \dfrac{\partial N_2}{\partial\xi} & \dfrac{\partial N_3}{\partial\xi} & \dfrac{\partial N_4}{\partial\xi} \\ \dfrac{\partial N_1}{\partial\eta} & \dfrac{\partial N_2}{\partial\eta} & \dfrac{\partial N_3}{\partial\eta} & \dfrac{\partial N_4}{\partial\eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}, \tag{5.23}$$

where (x_1, y_1) is the x, y coordinates of node 1, etc. The derivatives of the shape functions with respect to the global coordinates can thus be found via

$$\begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dfrac{\partial}{\partial\xi} \\ \dfrac{\partial}{\partial\eta} \end{bmatrix}, \tag{5.24}$$

where $\mathbf{J}^{-1}$ is the inverse of the Jacobian matrix. The shape functions themselves do not have to be converted from the local to the global coordinate system. This is because

we have chosen a special type of "isoparametric" element (which essentially means that the shape functions defining the geometry are the same as those interpolating the unknown function).

Transformation of integration from the global to local coordinate system is performed using the determinant of the Jacobian, $\det(\mathbf{J})$, according to the following relation

$$\iint f(x,y)\,dxdy = \int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)\,\det(\mathbf{J}(\xi,\eta))\,d\xi d\eta. \tag{5.25}$$

We note that in contrast to the one dimensional example, the Jacobian $\mathbf{J}$ is in general not a constant. Only under special conditions is $\mathbf{J}$ a constant. Such conditions for bilinear basis functions include when the elements have edges which are parallel to the x and y axis, or the elements define parallelograms. Combining this result with Equation (5.21) yields the final integration formula

$$\iint f(x,y)\,dxdy \approx \sum_{i=1}^{n_{\text{ip}}} f(\xi_i,\eta_i)\,\det(\mathbf{J}(\xi_i,\eta_i))W_i. \tag{5.26}$$

Again we emphasize that $\det(\mathbf{J})$ should be evaluated at the quadrature point.

Finally, to complete the definition of the element stiffness matrices, we provide the definition for the shape functions associated with a bilinear, 2D quadrilateral element such as is shown in Figure 51. In local coordinates, they are defined via

$$N_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta) \tag{5.27}$$

$$N_2(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta) \tag{5.28}$$

$$N_3(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta) \tag{5.29}$$

$$N_4(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta), \tag{5.30}$$

and their derivatives with respect to ξ and η are (computed by hand, or with MAPLE);

$$\begin{aligned}
\frac{\partial N_1(\xi,\eta)}{\partial\xi} &= -\frac{1}{4}(1-\eta), \quad & \frac{\partial N_1(\xi,\eta)}{\partial\eta} &= -\frac{1}{4}(1-\xi)\\
\frac{\partial N_2(\xi,\eta)}{\partial\xi} &= -\frac{1}{4}(1+\eta), \quad & \frac{\partial N_2(\xi,\eta)}{\partial\eta} &= \frac{1}{4}(1-\xi)\\
\frac{\partial N_3(\xi,\eta)}{\partial\xi} &= \frac{1}{4}(1+\eta), \quad & \frac{\partial N_3(\xi,\eta)}{\partial\eta} &= \frac{1}{4}(1+\xi)\\
\frac{\partial N_4(\xi,\eta)}{\partial\xi} &= \frac{1}{4}(1-\eta), \quad & \frac{\partial N_4(\xi,\eta)}{\partial\eta} &= -\frac{1}{4}(1+\xi).
\end{aligned} \tag{5.31}$$

In summary, the following steps must be carried out to perform integration of the element matrices:

1. Define the number of integration points (n_{ip}) and obtain and save the integration points and weights.

2. Perform a loop over all elements and initialize the element matrices $\mathbf{M}$ and $\mathbf{K}$.

3. Within the element loop, do a loop over each integration point. Evaluate the shape functions and their derivatives at the relevant integration point.

4. Calculate the Jacobian matrix by performing the operation in Equation (5.23).

5. Convert the derivatives from the local coordinates to the global coordinates by performing the operation in Equation (5.24).

6. Calculate the determinant of the Jacobian.

7. Perform the vector multiplication involving the shape functions or their derivatives (evaluated at integration points), multiplied by the relevant weights, which leads to the matrices $\mathbf{M}$ and $\mathbf{K}$. The result should be added to the multiplication from proceeding integration points.

8. Once one has exited the loop over integration points, the integration is complete (i.e., the element matrices have been integrated).

9. Form the matrices $\mathbf{L}$ and $\mathbf{R}$ defined in Equation (5.20)

10. The element matrices can then be added to the global element stiffness matrix. This process is then continued for each element.

5.4 Assembling the global stiffness matrix

The final step that needs to be performed is the node-by-node summing-up of the element matrices to form the global stiffness matrices. Once again, this is carried out in a manner very similar to that already done in one dimension. The only difference is that now one has to make more decisions in how the unknowns in the mesh are numbered globally. An example of a small 2D finite element mesh is shown in Figure 51. Note that each node represents one unknown (i.e. temperature) and is given a global index (and a global coordinate).

The global numbering of the equations used in your finite element is not unique. There are no strict rules as to how one should order the global indices. On structured grids with quadrilateral elements (like in our example) the traditional rule of thumb was to begin numbering in the direction with the fewest nodes in an effort to minimise the bandwidth of the global matrix and thus yield more efficient solves. In practice, when using modern factorisation techniques such as those employed in MATLAB, this rule of thumb is no longer applicable. Furthermore it does not generalise to unstructured meshes. Here we advocate using the numbering scheme which is most convenient to implement. The relationship between the elements and global node numbers should be specified early in the program and saved in a matrix (for example in `g_num`. This matrix is then used to "steer" the entries within each element stiffness matrix into the correct position within the global matrix.

To illustrate this process with an example, we return to the 2D diffusion equation and consider assembling the global stiffness matrix which corresponds to a very small mesh which is illustrated in Figure 52. Referring to the element matrices defined in

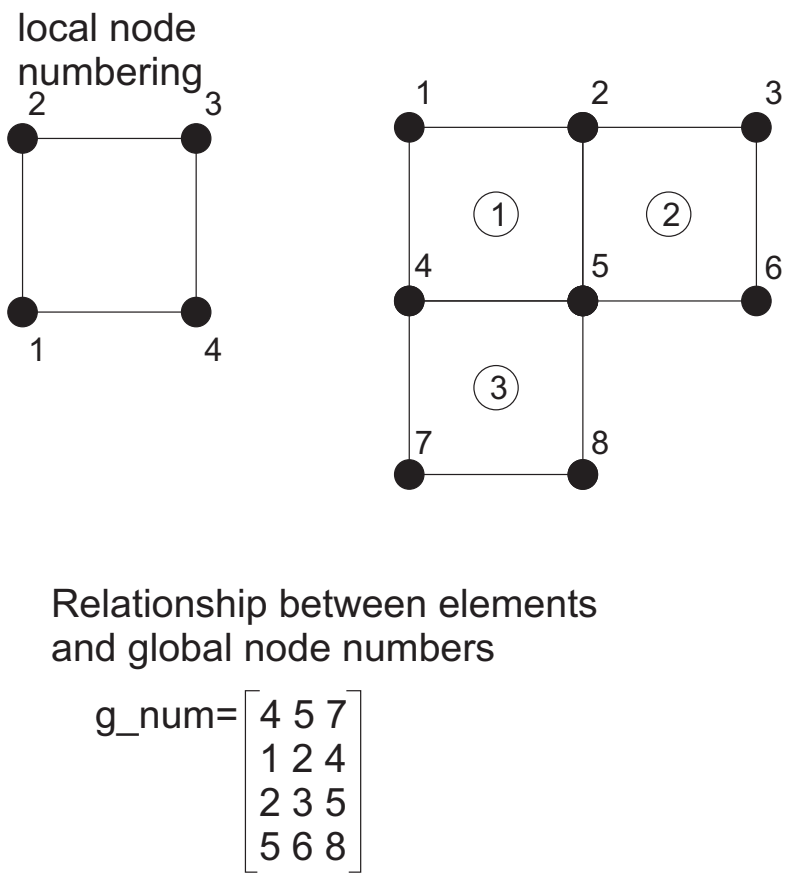


Figure 52: A small three element finite element mesh. In the text the diffusion equation is discretised for this mesh. The corresponding system of global equations are shown in Equation (5.33).

Equation (5.20), we introduce the following notation

$$\mathbf{L}^e = \begin{bmatrix} L^e_{11} & L^e_{12} & L^e_{13} & L^e_{14} \\ L^e_{21} & L^e_{22} & L^e_{23} & L^e_{24} \\ L^e_{31} & L^e_{32} & L^e_{33} & L^e_{34} \\ L^e_{41} & L^e_{42} & L^e_{43} & L^e_{44} \end{bmatrix},$$

where the superscript e is introduced to represent the element number. Thus, for example, the term L^3_{11} is the term L_{11} from element 3. Similar notation is assumed for $\mathbf{R}_G$ (refer to Equation (5.20)). The matrix $\mathtt{g\_num}$ for the mesh in Figure 52 is given by:

$$\mathtt{g\_num} = \begin{bmatrix} 4 & 5 & 7 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 5 & 6 & 8 \end{bmatrix}. \tag{5.32}$$

Note that there are three columns, one for each element, while there are four rows, one for each node. As already stated, $\mathtt{g\_num}$ is used to place the entries from element

matrices in the correct position within the global matrix. As an example, consider the term L^1_{13}. Since the term is from element 1, we must look at the first column of `g_num` where we see that the local nodes 1 and 3 (i.e., the first and third row) map to the global indices 4 and 2 respectively. This indicates that the entry L^1_{13} will be inserted (summed) within the global matrix $\mathbf{L}_G$ at the position $(4, 2)$. Repeating this process for all terms leads to the global equations which will be denoted via

$$\mathbf{L}_G\mathbf{T}^{n+1} = \mathbf{R}_G\mathbf{T}^n + \mathbf{F}_G, \tag{5.33}$$

where

$$\mathbf{T}^k = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{bmatrix}^k, \tag{5.34}$$

$$\mathbf{L}_G = \begin{bmatrix} L^1_{22} & L^1_{23} & 0 & L^1_{21} & L^1_{24} & 0 & 0 & 0 \\ L^1_{32} & L^1_{33}+L^2_{22} & L^2_{23} & L^1_{31} & L^1_{34}+L^2_{21} & L^2_{24} & 0 & 0 \\ 0 & L^2_{32} & L^2_{33} & 0 & L^2_{31} & L^2_{34} & 0 & 0 \\ L^1_{12} & L^1_{13} & 0 & L^1_{11}+L^3_{22} & L^1_{14}+L^3_{23} & 0 & L^3_{21} & L^3_{24} \\ L^1_{42} & L^1_{43}+L^2_{12} & L^2_{13} & L^1_{41}+L^3_{32} & L^1_{44}+L^2_{11}+L^3_{33} & L^2_{14} & L^3_{31} & L^3_{34} \\ 0 & L^2_{42} & L^2_{43} & 0 & L^2_{41} & L^2_{44} & 0 & 0 \\ 0 & 0 & 0 & L^3_{12} & L^3_{13} & 0 & L^3_{11} & L^3_{14} \\ 0 & 0 & 0 & L^3_{42} & L^3_{43} & 0 & L^3_{41} & L^3_{44} \end{bmatrix} \tag{5.35}$$

and

$$\mathbf{R}_G = \begin{bmatrix} R^1_{22} & R^1_{23} & 0 & R^1_{21} & R^1_{24} & 0 & 0 & 0 \\ R^1_{32} & R^1_{33}+R^2_{22} & R^2_{23} & R^1_{31} & R^1_{34}+R^2_{21} & R^2_{24} & 0 & 0 \\ 0 & R^2_{32} & R^2_{33} & 0 & R^2_{31} & R^2_{34} & 0 & 0 \\ R^1_{12} & R^1_{13} & 0 & R^1_{11}+R^3_{22} & R^1_{14}+R^3_{23} & 0 & R^3_{21} & R^3_{24} \\ R^1_{42} & R^1_{43}+R^2_{12} & R^2_{13} & R^1_{41}+R^3_{32} & R^1_{44}+R^2_{11}+R^3_{33} & R^2_{14} & R^3_{31} & R^3_{34} \\ 0 & R^2_{42} & R^2_{43} & 0 & R^2_{41} & R^2_{44} & 0 & 0 \\ 0 & 0 & 0 & R^3_{12} & R^3_{13} & 0 & R^3_{11} & R^3_{14} \\ 0 & 0 & 0 & R^3_{42} & R^3_{43} & 0 & R^3_{41} & R^3_{44} \end{bmatrix}. \tag{5.36}$$

The process of summing the element matrix into the global matrix is easily achieved in a computer program. For example, once the matrix $\mathbf{L}^e$ for a certain element e has been integrated, it is added to the global matrix using a command such as

```
>> LG( g_num( :, iel ), g_num( :, iel ) ) = ...
          LG( g_num( :, iel ), g_num( :, iel ) ) + L;
```

where `iel` is the current element index. Now that the global equations have been constructed all that remains is to apply boundary conditions and solve the system of equations. This is carried out in the same manner as the one dimensional problem considered earlier.

5.5 Exercises

1. Write a 2D FE code that solves the diffusion equation using bilinear, quadrilateral elements. Assume a constant, isotropic conductivity tensor. Apply constant temperature at the top and bottom boundaries, and zero flux at the side boundaries (i.e. set no boundary condition here).

2. Confirm that the code works when the grid is deformed (if not, it is likely that you made a mistake defining the Jacobian or your global derivatives). Define a mesh which has significant topography on the upper surface. Compute the temperature distribution within this domain.

3. Modify the code to take into account (i) a spatially variable, anisotropic conductivity tensor, (ii) variable grid spacing and (iii) a spatially variable heat source.

6

The Weak Form

6.1 A prototype PDE

Consider the Poisson equation

$$\nabla^2 u + f = 0. \tag{6.1}$$

In order to obtain a unique solution of Equation (6.1) in some domain Ω, we require the prescription of the boundary conditions. We will denote the boundary of Ω via $\partial\Omega$ and the interior of the domain as $\bar{\Omega} = \Omega \setminus \partial\Omega$. Dirichlet boundary conditions specify the value of u along some region of the boundary which we denote by $\partial\Omega_D$. A Neumann boundary specifies the value of ∇u along some boundary segment $\partial\Omega_N$. At every point in space $\mathbf{x}$ along the entire boundary of Ω, a boundary condition must be specified. That is, $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$ and these segments do not overlap, i.e. $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. Formally we can state this problem as:
Find u such that

$$\nabla^2 u + f = 0 \qquad \text{in } \bar{\Omega} \tag{6.2}$$

subject to

$$u = g_D \quad \text{on } \partial\Omega_D \qquad \text{and} \qquad \nabla u \cdot \mathbf{n} = g_N \quad \text{on } \partial\Omega_N, \tag{6.3}$$

where $\mathbf{n}$ is the outward pointing normal to the boundary $\partial\Omega$.

6.2 The weak form

To construct the weak form we introduce a *test function* v and we will require that

$$\int_\Omega \left(\nabla^2 u + f\right) v \, dV = 0. \tag{6.4}$$

Using Green's theorem, the above can be equivalently stated as

$$\int_\Omega \nabla u \cdot \nabla v \, dV = \int_\Omega v f \, dV + \oint_{\partial\Omega} v \frac{\partial u}{\partial n} \, dS. \tag{6.5}$$

We now discuss some details related to the weak form defined in Equation (6.5). From Equation (6.4) it is apparent that any solution u which satisfies Equation (6.1) satisfies Equation (6.4) for *any* choice of v. To consider whether the converse is true, we first observe that the required smoothness of u has been reduced, that is the derivative

operating on u has changed from second order to first order. Thus Equation (6.5) may have a solution say u', but u' may not be smooth enough to also be a solution of Equation (6.1). For this reason, solutions of Equation (6.5) are referred to as weak solutions. Given their weak nature, many possible weak solutions will exist which satisfy Equation (6.5). We will discuss the class of functions from which weak solutions live within in the following paragraphs.

As of yet we have not discussed what types of functions are valid choices for u and v. One restriction on selecting v comes from the observation that the weak form in Equation (6.5) contains a flux term $\frac{\partial u}{\partial n} = \nabla u \cdot \mathbf{n}$, defined over the entire boundary. Recalling the problem statement required to obtain a solution to the PDE (see Equation (6.3)), we observe that the flux term in the weak form exactly matches the Neumann boundary condition, except that that Neumann condition is only enforced over the boundary segment $\partial\Omega_N$ and not over all $\partial\Omega$. Therefore, to enforce that the weak form is consistent with the boundary conditions of the PDE we require that $v = 0$ along the Dirichlet boundary $\partial\Omega_D$. Thus, for solutions of the PDE which only possess Dirichlet boundaries, i.e. $\Omega_N = \emptyset$, the surface integral in the weak form vanishes as v is constructed to be identically zero along this boundary region. The Dirichlet boundary also provides a restriction on the choice of u, name that any function used to define the weak solution must satisfy $u = g_D$ on $\partial\Omega_D$. Using this restrictions on u, v, we can state the weak form problem as:
Find u such that

$$\int_\Omega \nabla u \cdot \nabla v \, dV = \int_\Omega v f \, dV + \oint_{\partial\Omega_N} v \, g_N \, dS \qquad \text{in } \bar{\Omega}. \tag{6.6}$$

Another property to consider in choosing a valid v, and in defining the class of permissible functions within which weak solutions to Equation (6.6) live within, come from the consideration of *function smoothness*. To define smoothness, we use an L_2 measure defined as

$$||u||_2 := \left(\int_\Omega u^2 \, dV \right)^{1/2}. \tag{6.7}$$

Any function u which satisfies

$$||u||_2 < \infty$$

is said to live within the space (i.e. the set of all functions) of L_2 functions. Considering the left hand side of Equation (6.6) we observed that the equation is well-defined (doesn't blow up) if all the derivatives of u and v are in L_2. Accordingly, if this true and both f and g_N also live within L_2, then the right hand side of Equation (6.6) will also be well bounded.

6.3 The discrete weak form

We will assume that the problem domain Ω, has been partitioned into Me elements and Nn nodes. One each node, i we will define a test function $v(\mathbf{x}) = \hat{\phi}_i(\mathbf{x})$. Inserting this expression into Equation (6.6) yields

$$\sum_i^{Nn} \int_\Omega \nabla u \cdot \nabla \hat{\phi}_i \, dV = \sum_i^{Nn} \int_\Omega \hat{\phi}_i f \, dV + \oint_{\partial\Omega_N} \hat{\phi}_i \, g_N \, dS. \tag{6.8}$$

The discrete solution space is then defined via

$$u(\mathbf{x}) = \phi_1(\mathbf{x})u_1 + \phi_2(\mathbf{x})u_2 + \cdots = \sum_{i=1}^{Nn} \phi_i(\mathbf{x})u_i,$$

where each u_i represents the approximate to u at the node i. Upon substitution we have

$$\sum_i^{Nn}\sum_j^{Nn} \int_\Omega \nabla\hat{\phi}_i \cdot \nabla\phi_j u_j \, dV = \sum_i^{Nn} \int_\Omega \hat{\phi}_i f \, dV + \oint_{\partial\Omega_N} \hat{\phi}_i \, g_N \, dS. \tag{6.9}$$

By making the choice that the trial functions should be identical to the discrete solution space, that is choosing $\hat{\phi}_i = \phi_i$, we obtain the Galerkin approximation,

$$\sum_i^{Nn}\sum_j^{Nn} \int_\Omega \nabla\phi_i \cdot \nabla\phi_j u_j \, dV = \sum_i^{Nn} \int_\Omega \phi_i f \, dV + \oint_{\partial\Omega_N} \phi_i \, g_N \, dS. \tag{6.10}$$

The equation above defines a system of linear equation which can be more compactly expressed via a matrix-vector form.

6.4 Further reading

- For a discussion about the weak form, see sections 1.1 to 1.3 of Elman, Silvester & Wathan.
- For a discussion about evaluating the discrete weak form, see sections 1.4 of Elman, Silvester & Wathan.
- For a discussion on the same concepts with, read and work through pages 1-13 of Hughes.

7

Elasticity in Two Dimensions

7.1 Introduction

Up to now we have only considered PDE's which involve scalar unknowns (i.e. temperature). When discretised, such problems possess one degree of freedom (i.e., one unknown) per node. However, many PDE's such as those describing the behaviour of elastic solids or viscous fluids, have vector unknowns (e.g., the displacement or velocity in each direction). When discretised these PDE's translate to FE problems with multiple degrees of freedom per node. The purpose of this script is to introduce you the finite element programming of problems having two degrees of freedom per node. This approach is illustrated by solving a solid mechanics problem.

7.2 Governing equations

The equations governing the two-dimensional displacement of a solid are derived by combining the conservation of momentum and a constitutive relationship relating stresses and strain. The conservation of momentum defined in a domain Ω with boundary $\partial\Omega$ is given by:

$$\begin{aligned} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} &= 0 \\ \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= 0, \end{aligned} \tag{7.1}$$

where σ_{xx}, σ_{yy} and σ_{xy} are stresses (gravity is ignored). In case of an elastic medium, the constitutive relation is

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix} \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix}, \tag{7.2}$$

where E is the Youngs modulus, ν the Poisson's ratio and $\epsilon_{xx}, \epsilon_{yy}, \gamma_{xy}$ are strains. In this case we assumed a linear elastic material subjected to plane strain conditions. In the more general case, the constitutive relationship becomes more complex (one can, for example, incorporate anisotropy by changing the constitutive matrix in Equation

(7.2)). The relationship between strain and displacement is given by

$$\begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}, \tag{7.3}$$

where u_x, u_y are displacements in the x and y direction, respectively. Using matrix notation, these three relations can be written as

$$\begin{aligned} \mathbf{B}^T \hat{\boldsymbol{\sigma}} &= \mathbf{0}, \\ \hat{\boldsymbol{\sigma}} &= \mathbf{D}\hat{\boldsymbol{\epsilon}}, \\ \hat{\boldsymbol{\epsilon}} &= \mathbf{B}\mathbf{e}, \end{aligned} \tag{7.4}$$

where

$$\mathbf{e} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}, \tag{7.5}$$

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix}, \tag{7.6}$$

and

$$\hat{\boldsymbol{\epsilon}} = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix}, \qquad \hat{\boldsymbol{\sigma}} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix}. \tag{7.7}$$

In the displacement formulation, such as the one presented here, one eliminates $\hat{\boldsymbol{\sigma}}$ and $\hat{\boldsymbol{\epsilon}}$ in the following steps:

$$\begin{aligned} \mathbf{B}^T \hat{\boldsymbol{\sigma}} &= \mathbf{0} \\ \mathbf{B}^T \mathbf{D} \hat{\boldsymbol{\epsilon}} &= \mathbf{0} \\ \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{e} &= \mathbf{0} \end{aligned} \tag{7.8}$$

This last relation is a pair or partial differential equations for the two unknowns, u_x and u_y (i.e., the two components of the vector $\mathbf{e}$).

7.3 FE discretisation

To proceed with a finite element discretisation one introduces an element type and set of shape functions. We use 4-node quadrilaterals and the corresponding bilinear shape functions. Thus, u_x and u_y within a single element are approximated as

$$u_x(x,y) \approx \begin{bmatrix} N_1(x,y) & N_2(x,y) & N_3(x,y) & N_4(x,y) \end{bmatrix} \begin{bmatrix} u_{x(1)} \\ u_{x(2)} \\ u_{x(3)} \\ u_{x(4)} \end{bmatrix} = \mathbf{N}\mathbf{u}_x \tag{7.9}$$

and

$$u_y(x,y) \approx \begin{bmatrix} N_1(x,y) & N_2(x,y) & N_3(x,y) & N_4(x,y) \end{bmatrix} \begin{bmatrix} u_{y(1)} \\ u_{y(2)} \\ u_{y(3)} \\ u_{y(4)} \end{bmatrix} = \mathbf{N}\mathbf{u}_y. \tag{7.10}$$

Substituting these approximations into Equation (7.8), weighting each equation with the shape functions, integrating over the element volume Ω^e and integrating by parts where necessary yields the following set of discrete element equations:

$$\mathbf{K}\,\mathbf{r} = \mathbf{F}, \tag{7.11}$$

where

$$\mathbf{K} = \int_{\Omega^e} \hat{\mathbf{B}}^T \mathbf{D} \hat{\mathbf{B}}\, dV, \tag{7.12}$$

$$\hat{\mathbf{B}} = \begin{bmatrix} \frac{\partial \mathbf{N}}{\partial x} & 0 \\ 0 & \frac{\partial \mathbf{N}}{\partial y} \\ \frac{\partial \mathbf{N}}{\partial y} & \frac{\partial \mathbf{N}}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \end{bmatrix} \tag{7.13}$$

and

$$\mathbf{F} = \begin{bmatrix} -\oint_{\partial\Omega^e \cap \partial\Omega} \mathbf{N}^T \left(\sigma_{xx} n_x + \sigma_{xy} n_y\right)\, dS \\ -\oint_{\partial\Omega^e \cap \partial\Omega} \mathbf{N}^T \left(\sigma_{yx} n_x + \sigma_{yy} n_y\right)\, dS \end{bmatrix} = \begin{bmatrix} -\oint_{\partial\Omega^e \cap \partial\Omega} \mathbf{N}^T\, t_x\, dS \\ -\oint_{\partial\Omega^e \cap \partial\Omega} \mathbf{N}^T\, t_y\, dS \end{bmatrix}, \tag{7.14}$$

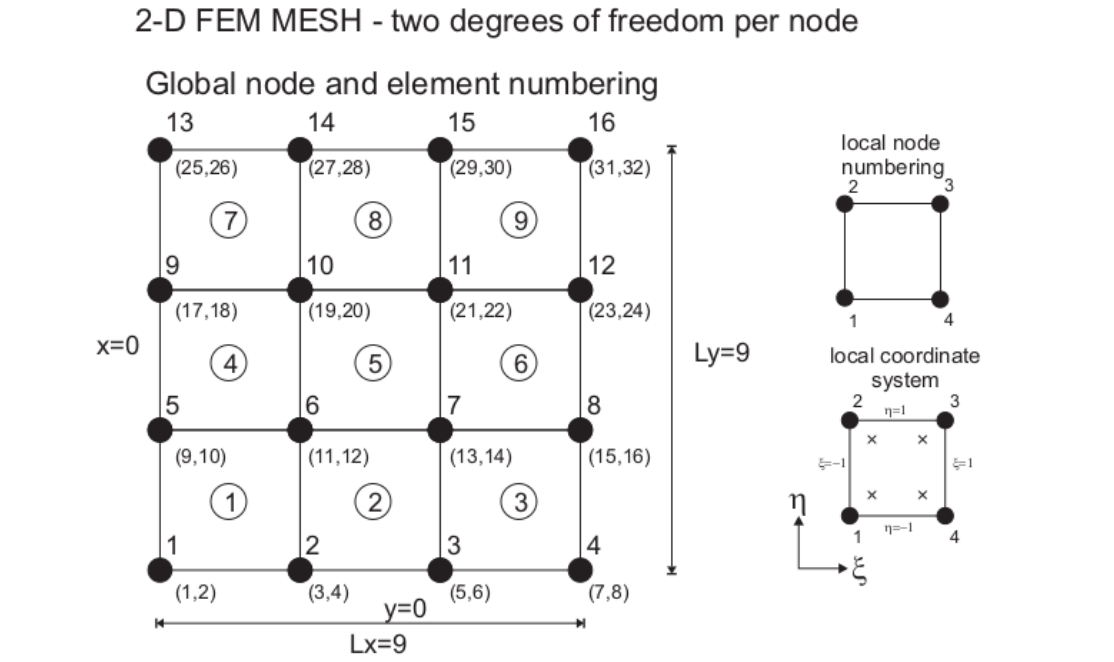
where $t_i = \sigma_{ij} n_j$ is an applied traction along the piece of the element edge $\partial\Omega^e$ contained on the boundary of the domain $\partial\Omega$ and the element displacement vector $\mathbf{r}$ is given by

$$\mathbf{r} = \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} = \begin{bmatrix} u_{x(1)} \\ u_{x(2)} \\ u_{x(3)} \\ u_{x(4)} \\ u_{y(1)} \\ u_{y(2)} \\ u_{y(3)} \\ u_{y(4)} \end{bmatrix}. \tag{7.15}$$

The terms within the element matrix $\mathbf{K}$ can be integrated via Gauss-Legendre quadrature rules using the same procedure as was utilised for the diffusion equation.

7.4 Matrix assembly

In the diffusion problem we considered previously, each node represented only one unknown and it was reasonably straightforward to "steer" the element matrices to the correct location in the global matrix. Now that we have two degrees of freedom per node



$$g\_num = \begin{bmatrix} 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 \\ 5 & 6 & 7 & 9 & 10 & 11 & 13 & 14 & 15 \\ 6 & 7 & 8 & 10 & 11 & 12 & 14 & 15 & 16 \\ 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 \end{bmatrix}$$

Relationship between elements and global node numbers

$$nf = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 & 20 & 22 & 24 & 26 & 28 & 30 & 32 \end{bmatrix}$$

Relationship between nodes and equation numbers

$$g\_g = \begin{bmatrix} 1 & 3 & 5 & 9 & 11 & 13 & 17 & 19 & 21 \\ 9 & 11 & 13 & 17 & 19 & 21 & 25 & 27 & 29 \\ 11 & 13 & 15 & 19 & 21 & 23 & 27 & 29 & 31 \\ 3 & 5 & 7 & 11 & 13 & 15 & 19 & 21 & 23 \\ 2 & 4 & 6 & 10 & 12 & 14 & 18 & 20 & 22 \\ 10 & 12 & 14 & 18 & 20 & 22 & 26 & 28 & 30 \\ 12 & 14 & 16 & 20 & 22 & 24 & 28 & 30 & 32 \\ 4 & 6 & 8 & 12 & 14 & 16 & 20 & 22 & 24 \end{bmatrix}$$

Relationship between elements and equation numbers

Figure 71: A small finite element mesh with two degrees of freedom per node. Global space numbering for nodes and displacements is shown in upper left image. Upper right panel shows the local space numbering of the nodes and the reference element. Lower panels indicate element-node map (`g_num`), the node-equation map (`nf`) and the element-equation map (`g_g`).

we have to perform some additional steps to define how we assemble the global matrix. Namely, we will require the specification of three matrices, denoted `g_num`, `nf` and `g_g`. We have already encountered `g_num`; this matrix defines the relationship between

global element numbers and global node numbers. The matrix `nf` is used to specify the relationship between the global node numbers and the global equation numbers. Lastly, the matrix `g_g` defines the relationship between the global element numbers and the global equation numbers. Examples of these matrices for a small mesh are shown in Figure 71. Once these matrices are constructed, then the assembly of the global matrix is straightforward. Within the main element loop, `g_num` is used to retrieve the nodes within a particular element, which is necessary for example to obtain the global node coordinates of each element (e.g., `coord = g_coord(:,g_num(:,iel))'`). The matrix `nf` is used if one wishes to refer to a specific degree of freedom. For example, say our solution is stored in the array `displ`. Then `displ(nf(1, :))` can be used to refer to the first degree of freedom (i.e, u_x) for each node in the mesh whereas `displ(nf(2, :))` can be used to refer to the second degree of freedom (i.e, u_y). Finally, the matrix `g_g` is used to "steer" the integrated element matrix $\mathbf{K}$ to the correct position in the global matrix $\mathbf{L}_G$, e.g.

```
>> LG( g_g(:,iel), g_g(:,iel) ) = ...
     LG( g_g(:,iel), g_g(:,iel) ) + K;
```

Now that the global matrix has been assembled, one can impose any Dirichlet boundary conditions and solve the system of equations.

7.5 Exercises

1. Write a 2D, plane strain, elastic code to solve the equations above. Test it with a pure-shear setup, in which you prescribe u_x at the left and right boundaries and set $u_y = 0$ at the lower boundary. Leave all other boundaries unconstrained, implying the traction $t_i = 0$. If you choose $\nu = 0.49$, the material behaves almost incompressibility and you should see a pure shear displacement field.

2. Model the deformation of a thin elastic beam under the influence of gravity. To do this, you will need to add gravity to the force balance equations (which will result in right-hand side terms in your equations). Create a setup in which you keep the left boundary of the domain fixed ($u_x = u_y = 0$) and all other boundaries free (i.e. enforce zero normal stress). Introduce time-evolution into the model by updating the global coordinates with `gcoord = gcoord + displacement * factor`, where `factor` is the fraction of the displacement field you wish to use to advect the mesh.

3. Examine the stress field obtained from your FE solution. Plot the components of the stress at each integration plot. Plot σ_{xx} at the nodal points of your deformed mesh.

4. Repeat the exercises above using 2D biquadratic shape functions (instead of bilinear shape functions). The biquadratic basis functions are defined in Appendix D. You will need to use at least 3×3 integration points with the biquadratic shape functions. Once this element type is added to your code and you have verified the correctness of the implementation (e.g you obtain very similar looking results as when using bilinear elements), you will be well prepared to tackle the Stokes equations (next chapter).

8

Stokes Flow in Two Dimensions

8.1 Governing equations

Many problems in Earth science involve the flow of fluids. One of the most important examples of which is the slow viscous deformation of rock. The purpose of this script is to present an introduction to the equations which govern the motion of very viscous fluid and to study how these equations can be solved using the finite element method.

The mechanics of a fluid is governed by four sets of equations, conservation of mass (continuity), conservation of momentum, a relationship between strain rate and velocity and a constitutive relationship. The force balance in two dimensions is governed by the equations

$$\begin{aligned} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} &= 0 \\ \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} - \rho g &= 0, \end{aligned} \tag{8.1}$$

where σ_{ij} is the stress tensor, ρ is density and g is gravitational acceleration (note inertial terms are ignored). We assume a coordinate system in which the direction of gravity acts in the negative y direction The equation for the conservation of mass of an incompressible fluid is

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{8.2}$$

where u and v are velocities in x and y direction respectively. The constitutive relationship for an incompressible, Newtonian viscous material is

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} = -p \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 2\eta & 0 & 0 \\ 0 & 2\eta & 0 \\ 0 & 0 & 2\eta \end{bmatrix} \begin{bmatrix} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{bmatrix}, \tag{8.3}$$

where η is the viscosity, $\dot{\epsilon}_{ij}$ are strain rates and p is the pressure. Finally, the kinematic relationship between strain rates and velocities is defined as

$$\begin{bmatrix} \dot{\epsilon}_{xx} \\ \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \end{bmatrix}. \tag{8.4}$$

When written out in full, these equations in 2D have the form

$$-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left(2\eta\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right) = 0 \tag{8.5}$$

$$-\frac{\partial p}{\partial y}+\frac{\partial}{\partial y}\left(2\eta\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial x}\left(\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right) = \rho g \tag{8.6}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y} = 0. \tag{8.7}$$

This set of coupled partial differential equations are commonly known as the Stokes equations. Using matrix notation, Equations (8.1), (8.2), (8.3) & (8.4) can be written compactly as

$$\mathbf{B}^T\hat{\boldsymbol{\sigma}}=\mathbf{f} \tag{8.8}$$

$$\mathbf{m}^T\mathbf{B}\mathbf{u}=0 \tag{8.9}$$

$$\hat{\boldsymbol{\sigma}}=-\mathbf{m}p+\mathbf{D}\dot{\mathbf{e}} \tag{8.10}$$

$$\dot{\mathbf{e}}=\mathbf{B}\mathbf{u}, \tag{8.11}$$

where

$$\mathbf{u}=\begin{bmatrix}u\\v\end{bmatrix}, \qquad \mathbf{f}=\begin{bmatrix}0\\\rho g\end{bmatrix}=\begin{bmatrix}f_x\\f_y\end{bmatrix}, \tag{8.12}$$

$$\mathbf{m}=\begin{bmatrix}1\\1\\0\end{bmatrix}, \qquad \mathbf{B}=\begin{bmatrix}\frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}, \tag{8.13}$$

$$\mathbf{D}=\begin{bmatrix}2\eta & 0 & 0\\ 0 & 2\eta & 0\\ 0 & 0 & \eta\end{bmatrix}, \tag{8.14}$$

and

$$\dot{\mathbf{e}}=\begin{bmatrix}\dot{\epsilon}_{xx}\\ \dot{\epsilon}_{yy}\\ \dot{\gamma}_{xy}\end{bmatrix}, \qquad \hat{\boldsymbol{\sigma}}=\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{bmatrix}. \tag{8.15}$$

Note in the above equation that $\dot{\gamma}_{xy}=2\dot{\epsilon}_{xy}$ and that we have generalised the definition of the force to allow for non-zero contributions to the x-momentum equation. The reader should verify that Equations (8.8) - (8.9) are consistent with Equations (8.5), (8.6) and (8.7). In the Stokes formulation considered here, one eliminates $\hat{\boldsymbol{\sigma}}$ and $\dot{\mathbf{e}}$ in the following manner

$$\mathbf{B}^T\hat{\boldsymbol{\sigma}}=\mathbf{f} \tag{8.16}$$

$$\mathbf{B}^T\mathbf{D}\dot{\mathbf{e}}-\mathbf{B}^T\left(\mathbf{m}p\right)=\mathbf{f} \tag{8.17}$$

$$\mathbf{B}^T\mathbf{D}\mathbf{B}\mathbf{u}-\mathbf{B}^T\left(\mathbf{m}p\right)=\mathbf{f}, \tag{8.18}$$

to obtain an expression only in terms of the velocity $\mathbf{u}$ and pressure p. Thus, the governing equations are

$$\mathbf{B}^T\mathbf{D}\mathbf{B}\mathbf{u}-\mathbf{B}^T\left(\mathbf{m}p\right)=\mathbf{f} \tag{8.19}$$

and

$$\mathbf{m}^T\mathbf{B}\mathbf{u}=0, \tag{8.20}$$

which is a set of three equations (remember that Equation (8.19) consists of two equations), for the three unknowns u, v and p.

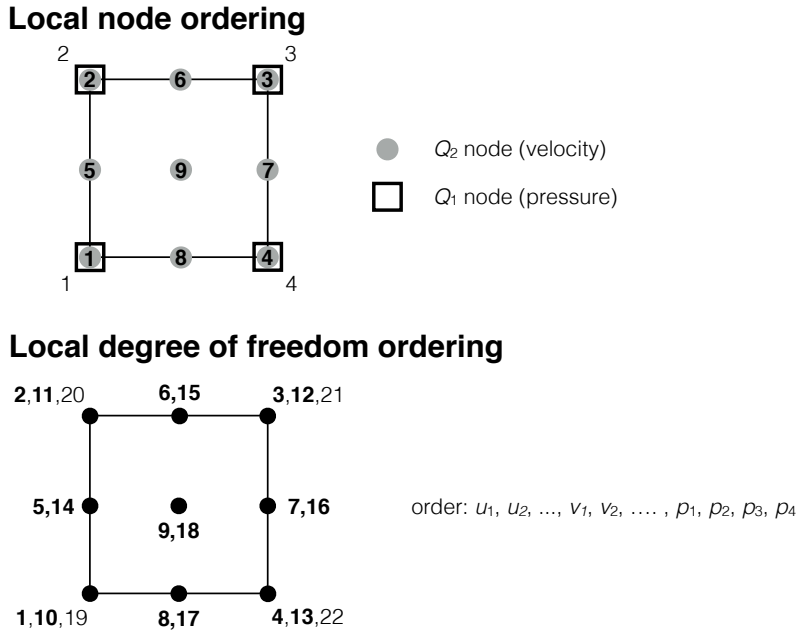


Figure 81: A single finite element illustrating the Q_2-Q_1 velocity-pressure formulation used to solve Stokes flow. A biquadratic shape function (Q_2) is used for velocity and a bilinear shape function (Q_1) for pressure. Refer to Appendix D for the definition of the Q_2 basis function. The nodes associated with velocity (filled grey circles – bold numbers) possess two degrees of freedom (for u, v) and the overlapping nodes associated with the pressure (open squares) possess one degree of freedom (for p).

8.2 FE discretisation

One simple and (barely) adequate method for discretising Equations (8.5), (8.6) and (8.7) with quadrilateral elements is to use the 9 node biquadratic shape functions (Q_2) for velocities and the 4 node bilinear shape function (Q_1) for the pressure. This configuration is depicted in Figure 81. Note that the same quadrilateral geometry is used to define the approximation for both velocity and pressure. We also note that the vertices of the velocity element coincide with the vertices of the pressure element. The two velocities are approximated as

$$u(x,y) \approx \begin{bmatrix} N_1 & N_2 & N_3 & N_4 & N_5 & N_6 & N_7 & N_8 & N_9 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \mathbf{N}\mathbf{u}^e, \tag{8.21}$$

$$v(x,y) \approx \begin{bmatrix} N_1 & N_2 & N_3 & N_4 & N_5 & N_6 & N_7 & N_8 & N_9 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{bmatrix} = \mathbf{N}\mathbf{v}^e, \tag{8.22}$$

or

$$\begin{bmatrix} u \\ v \end{bmatrix} \approx \begin{bmatrix} \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{N} \end{bmatrix} \begin{bmatrix} \mathbf{u}^e \\ \mathbf{v}^e \end{bmatrix} = \hat{\mathbf{N}}\mathbf{U}^e. \tag{8.23}$$

The pressure is approximated via

$$p(x,y) \approx \begin{bmatrix} \bar{N}_1 & \bar{N}_2 & \bar{N}_3 & \bar{N}_4 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \mathbf{N}_p\mathbf{p}^e. \tag{8.24}$$

Note in the above that we denote the basis functions for the velocity via N_i and those for pressure via $\bar{N}_i$.

First we consider the weak form of the momentum equations. To simplify the derivation, we multiply the stress gradient (in Voigt notation) by the velocity basis functions for u and v and integrate over the element volume Ω^e. In 2D this yields

$$\int_{\Omega^e} \begin{bmatrix} \mathbf{N}^T & 0 \\ 0 & \mathbf{N}^T \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} dV = \int_{\Omega^e} \begin{bmatrix} \mathbf{N}^T & 0 \\ 0 & \mathbf{N}^T \end{bmatrix} \begin{bmatrix} f_x \\ f_y \end{bmatrix} dV. \tag{8.25}$$

Then applying integration by parts yields

$$\begin{aligned} \int_{\Omega^e} \begin{bmatrix} \frac{\partial \mathbf{N}^T}{\partial x} & 0 & \frac{\partial \mathbf{N}^T}{\partial y} \\ 0 & \frac{\partial \mathbf{N}^T}{\partial y} & \frac{\partial \mathbf{N}^T}{\partial x} \end{bmatrix} \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} dV = \\ - \int_{\Omega^e} \begin{bmatrix} \mathbf{N}^T & 0 \\ 0 & \mathbf{N}^T \end{bmatrix} \begin{bmatrix} f_x \\ f_y \end{bmatrix} dV + \oint_{\partial\Omega^e} \begin{bmatrix} \mathbf{N}^T(\sigma_{xx}n_x + \sigma_{xy}n_y) \\ \mathbf{N}^T(\sigma_{yx}n_x + \sigma_{yy}n_y) \end{bmatrix} dS. \end{aligned} \tag{8.26}$$

Denoting the Neumann boundary conditions as $t_i = \sigma_{ij}n_j$ and the boundary segment of each element as Γ^e, we can simplify the weak form above to

$$\int_{\Omega^e} \begin{bmatrix} \frac{\partial \mathbf{N}^T}{\partial x} & 0 & \frac{\partial \mathbf{N}^T}{\partial y} \\ 0 & \frac{\partial \mathbf{N}^T}{\partial y} & \frac{\partial \mathbf{N}^T}{\partial x} \end{bmatrix} \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} dV = - \int_{\Omega^e} \begin{bmatrix} \mathbf{N}^T f_x \\ \mathbf{N}^T f_y \end{bmatrix} dV + \oint_{\Gamma^e} \begin{bmatrix} \mathbf{N}^T t_x \\ \mathbf{N}^T t_y \end{bmatrix} dS. \tag{8.27}$$

To complete the derivation, we introduce the discrete strain rate operator

$$\hat{\mathbf{B}} = \mathbf{B}\hat{\mathbf{N}}, \tag{8.28}$$

and then insert the definition of the discrete stress

$$\hat{\boldsymbol{\sigma}} = \mathbf{D}\hat{\mathbf{B}}\mathbf{U}^e - \mathbf{m}\mathbf{N}_p\mathbf{p}^e,$$

to give

$$\int_{\Omega^e} \begin{bmatrix} \frac{\partial \mathbf{N}^T}{\partial x} & 0 & \frac{\partial \mathbf{N}^T}{\partial y} \\ 0 & \frac{\partial \mathbf{N}^T}{\partial y} & \frac{\partial \mathbf{N}^T}{\partial x} \end{bmatrix} \left(\mathbf{D}\hat{\mathbf{B}}\mathbf{U}^e - \mathbf{m}\mathbf{N}_p\mathbf{p}^e \right) dV = - \int_{\Omega^e} \begin{bmatrix} \mathbf{N}^T f_x \\ \mathbf{N}^T f_y \end{bmatrix} dV + \oint_{\Gamma^e} \begin{bmatrix} \mathbf{N}^T t_x \\ \mathbf{N}^T t_y \end{bmatrix} dS, \tag{8.29}$$

or

$$\mathbf{K}^e\mathbf{U}^e + \mathbf{G}^e\mathbf{p}^e = \mathbf{f}^e, \tag{8.30}$$

in which

$$\mathbf{K}^e = \int_{\Omega^e} \hat{\mathbf{B}}^T \mathbf{D} \hat{\mathbf{B}} \, dV, \tag{8.31}$$

$$\mathbf{G}^e = - \int_{\Omega^e} \hat{\mathbf{B}}^T \mathbf{m} \mathbf{N}_p \, dV \tag{8.32}$$

and

$$\mathbf{f}^e = - \int_{\Omega^e} \begin{bmatrix} \mathbf{N}^T f_x \\ \mathbf{N}^T f_y \end{bmatrix} dV + \oint_{\Gamma^e} \begin{bmatrix} \mathbf{N}^T t_x \\ \mathbf{N}^T t_y \end{bmatrix} dS. \tag{8.33}$$

In comparison, the weak form of the continuity equation is much simpler to derive. All that is required is to multiply the continuity equation by $\mathbf{N}_p^T$ and integrate over the element volume to yield

$$- \int_{\Omega} \mathbf{N}_p^T \mathbf{m}^T \hat{\mathbf{B}} \mathbf{U}^e \, dV = \mathbf{0}, \tag{8.34}$$

or

$$(\mathbf{G}^e)^T \mathbf{U}^e = \mathbf{0}. \tag{8.35}$$

Note that we multiplied the weak continuity equation by minus one simply to make formulation symmetric (The reader should verify the system is symmetric). This doesn't change the system of equations as the right side of the continuity equations is the zero vector. The complete discretised Stokes problem (on the element level) can be compactly expressed via

$$\begin{bmatrix} \mathbf{K}^e & \mathbf{G}^e \\ (\mathbf{G}^e)^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{U}^e \\ \mathbf{p}^e \end{bmatrix} = \begin{bmatrix} \mathbf{f}^e \\ \mathbf{0} \end{bmatrix}. \tag{8.36}$$

One should note that the element matrices all have different dimensions. Using the 9-noded biquadratic elements for velocity, there are 18 velocity unknowns per element. The bilinear pressure element results in 4 pressure degrees of freedom per element. Consequently, $\mathbf{K}^e$ has dimensions 18×18 and $\mathbf{G}^e$ has dimensions 18×4. Care should be taken in carrying out the matrix-vector products as the discrete velocity and pressure vectors are of different length. All the element matrices defined above can be integrated and assembled in the standard finite element fashion.

8.3 Exercises

1. Write a FE code to solve the Stokes equations for the unknowns u, v and p using boundary conditions and the Rayleigh-Taylor setup shown in Figure 82. Use biquadratic shape functions for velocity and continuous bilinear shape functions

for pressure. Free slip conditions are implemented by only setting the boundary-normal velocities to zero (so e.g., $u = 0$ on the left and right boundary). Model time evolution by advecting the grid, and also study what happens if the upper boundary is a free surface, $\boldsymbol{\sigma} \cdot \mathbf{n} = \mathbf{0}$. To introduce this boundary condition, simply do not set any Dirichlet boundary conditions on the upper boundary.

2. Plot the second invariant of the strain rate tensor ($\dot{\epsilon}_{II} = \frac{1}{2}(\dot{\epsilon}_{xx}^2 + \dot{\epsilon}_{yy}^2 + 2\epsilon_{xy}^2)^{0.5}$) for the previous model.

3. Create a code to simulate viscous detachment folding subjected to a background pure shear deformation. Ask the instructors for hints.

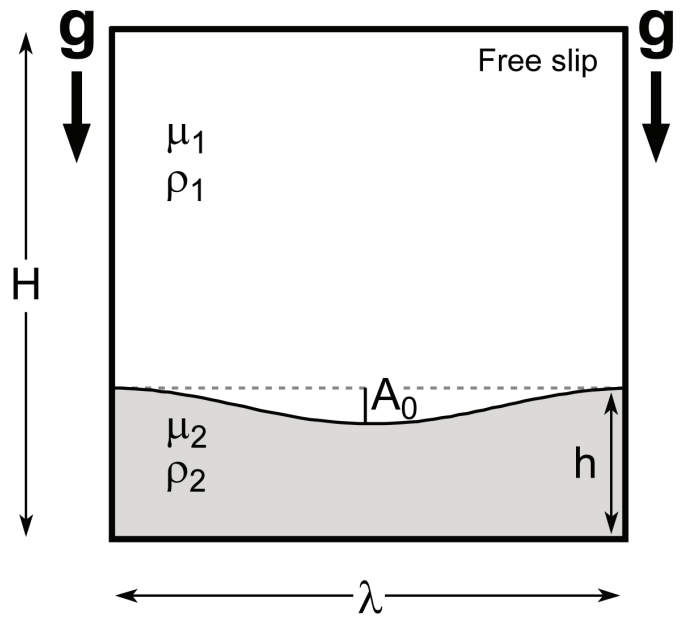


Figure 82: Rayleigh Taylor setup in which a fluid of higher density is superposed on top of a fluid with lower density and the interface between the two fluids is perturbed in a sinusoidal manner. Employ $\lambda = H = 1$, $h = 0.5, A_0 = 10^{-2}, g = 1, \mu_1 = 1, \mu_2 = 1, \rho_1 = 1, \rho_2 = 0$.

9

Code Verification

9.1 Introduction

The purpose of this session is to learn how to determine if your implementation of the finite element method (i.e. your MATLAB code) is "working correctly". By "working correctly" we mean that your FE code is solving the partial differential equations (PDE's) accurately, and most importantly that the numerical errors associated with the discrete solution of the PDE decrease at the expected rate when the grid is refined.

9.2 Taylor series approximations

Suppose we had a smooth, infinitely differentiable function $f(x)$, then we can locally approximate the function via an infinite Taylor series expansion

$$\begin{aligned} f(x+h) &= f(x) + h\frac{\partial f}{\partial x} + \frac{h^2}{2}\frac{\partial^2 f}{\partial x^2} + \frac{h^3}{6}\frac{\partial^3 f}{\partial x^3} + \dots \\ &= \sum_{i=0}^{N} \frac{h^n}{n!}\frac{\partial^n f}{\partial x^n}, \end{aligned} \tag{9.1}$$

where h is a small perturbation. The more terms N we include, the more accurate the expansion is. For example, consider the truncated Taylor series expansion

$$f(x+h) = f(x) + h\frac{\partial f}{\partial x} + \frac{h^2}{2}\frac{\partial^2 f}{\partial x^2} + \frac{h^3}{6}\frac{\partial^3 f}{\partial x^3}, \tag{9.2}$$

where all terms with powers ≥ 4 have been neglected. We will use the notation $\mathcal{O}(h^k)$ to indicate the order of accuracy of discrete solutions. We can express the approximation error in Equation (9.2) which results from dropping the terms with order $h^\alpha, \alpha \geq 4$ via

$$f(x+h) = f(x) + h\frac{\partial f}{\partial x} + \frac{h^2}{2}\frac{\partial^2 f}{\partial x^2} + \frac{h^3}{6}\frac{\partial^3 f}{\partial x^3} + \mathcal{O}(h^4), \tag{9.3}$$

which indicates that the truncated Taylor series is fourth order accurate.

9.3 Errors and norms

Consider the PDE

$$\nabla^2 u = f, \tag{9.4}$$

where we denote by u, the exact solution. If we apply the FE method to the above we obtain the following matrix problem

$$L^h u^h = F^h. \tag{9.5}$$

Here L^h is the discrete Laplacian and F^h is the discrete right hand side of the PDE. We denote by u^h the solution of the discrete problem. Note that in general, u^h will only ever by an *approximate* solution to the PDE. Accordingly, we can define the discretisation error e, as

$$e = u - u^h. \tag{9.6}$$

The superscript h notation is frequently used to indicate that the discrete solution is a function of the grid resolution h. In the above, e is function of space. In practice, it is more convenient to quantify the error via a single number, i.e. via some measure of the global discretisation error. Possible choices for global error measure include the L_1 measure

$$E(\Omega)_{L_1} = \int_\Omega \left| u - u^h \right| \, dV \tag{9.7}$$

and the L_2 measure

$$E(\Omega)_{L_2} = \sqrt{\int_\Omega (u - u^h)^2 \, dV}. \tag{9.8}$$

Sources of errors

We consider that there are two main sources of error in the FE implementations discussed in this course. These can be classified as *discretisation errors* or *numerical errors*. Errors associated with discretisation related to the size of the element used h, and the order of the polynomial used p. For example, the first exercise used ten ($h = 1/10$), 1D linear elements ($p = 1$). Numerical errors relate to round-off (i.e. due to finite precision arithmetic) and the accuracy of the solver used to obtain u^h in Equation (9.5). In all the examples we considered in this course, the solver employed by MATLAB is an exact LU factorisation - which for the purpose of this class we will regard as having zero numerical error.

Expected errors

The FE method benefits from having a very solid and rigorous mathematical background. This mathematical foundation provides many results regarding the errors we can expect from finite element calculations. We will not derive the errors estimates for the discretisation errors here, but rather state them and leave the interested reader to discover the world of finite element analysis in their own time.

In the following error estimates the coefficients C_i are constants which are independent of the grid resolution h, and the polynomial order of the basis function p. Note that a linear element implies $p = 1$, quadratic implies $p = 2$, cubic implies $p = 3$, etc. Each error estimate is associated with a different constant C_i. In general, the constants C_i cannot be derived analytically as they are problem dependent (e.g. there depend on the domain geometry, boundary conditions, coefficients in the PDE), thus if desired, they must be computed from numerical experiments. In practical computations we are usually less interested in the specific value of C_i and are primarily concerned with only the order of accuracy of the method.

STEADY STATE DIFFUSION (1D, 2D, 3D)

In the exercise considered in previous lessons, we used linear-1D (bilinear-2D, trilinear-3D) elements to solve the steady diffusion equation. In this case, the polynomial order of the shape function p, was $p = 1$. Using this element, the errors expected are given by

$$\int_\Omega \left| T - T^h \right| \, dV \leq C_1 h,$$

in the L_1 norm and

$$\left[\int_\Omega (T - T^h)^2 \, dV \right]^{1/2} \leq C_2 h^2,$$

in the L_2 norm. In general, for a shape function of order polynomial p, we expect the following:

$$\int_\Omega \left| T - T^h \right| \, dV \leq C_3 h^p, \tag{9.9}$$

$$\left[\int_\Omega (T - T^h)^2 \, dV \right]^{1/2} \leq C_4 h^{p+1}. \tag{9.10}$$

ELASTICITY (2D)

$$\int_\Omega \left| u - u^h \right| + \left| v - v^h \right| \, dV \leq C_5 h^p \tag{9.11}$$

$$\left[\int_\Omega (u - u^h)^2 + (v - v^h)^2 \, dV \right]^{1/2} \leq C_6 h^{p+1} \tag{9.12}$$

STOKES FLOW - MIXED METHOD (2D)

For convenience we will define the error of each velocity component as

$$e_u = u - u^h, e_v = v - v^h,$$

then the expected errors for pressure are given by

$$\left[\int_\Omega (p - p^h)^2 \, dV \right]^{1/2} \leq C_7 h^{q+1} \tag{9.13}$$

and for the velocity we have

$$\left[\int_\Omega e_u^2 + e_v^2 \, dV \right]^{1/2} \leq C_8 h^{p+1}. \tag{9.14}$$

$$\left[\int_\Omega \left(\frac{\partial e_u}{\partial x} \right)^2 + \left(\frac{\partial e_u}{\partial y} \right)^2 + \left(\frac{\partial e_v}{\partial x} \right)^2 + \left(\frac{\partial e_v}{\partial y} \right)^2 \, dV \right]^{1/2} \leq C_9 h^p \tag{9.15}$$

Note that for the mixed method (used when solving Stokes flow), the pressure basis function usually employs a different order polynomial to the velocity basis, which here we denote via q.

Limitations

The mathematical analysis performed to obtain the above error estimates required numerous assumptions to be made. The assumptions include: homogenous material properties (diffusivity, viscosity, density, etc.); that the model domain Ω can be completely resolved by the element with zero discretisation error (i.e. the domain Ω is a rectangle and a structured grid of quadrilateral elements was used to discretise it); there is zero error associated with the boundary conditions; there is zero error associated with the matrix solver.

Thus, we emphasize very strongly, that the results above are by no means completely universal. Whilst some of the assumptions can be relaxed and the error estimates above will remain valid - however, violating these assumptions can sometimes result in completely different (usually lower) error estimates.

9.4 Measuring the order of accuracy

To keep the discussion general, let's assume that the problem of interest can be described by the abstract PDE

$$\mathcal{L}u = f,$$

which is valid over the domain of interest Ω. For this problem, we will consider that u is subject to the following Dirichlet boundary condition

$$u(\mathbf{x}) = g(\mathbf{x}) \qquad \text{for } \mathbf{x} \in \partial\Omega,$$

where the boundary of Ω is denoted by $\partial\Omega$ and $g(\mathbf{x})$ is a prescribed function (i.e. it's known). Furthermore we will assume that given f, g and the differential operator $\mathcal{L}$, we have (by some means) obtained a function u which satisfies the boundary and the PDE above, i.e. we have a solution to the problem of interest.

The procedure to measure the order of accuracy of the discretisation error can be summarised as follows:

1. Define a mesh on your domain Ω^h, with a mesh resolution of h;
2. Discretise the differential operator $\mathcal{L}$ and f, and solve for u^h;
3. Compute a global error measure, say $E(\Omega^h)_{L_2}$ for example;
4. Generate a new mesh, $\Omega^{h/2}$ by sub-dividing all the elements within the original mesh Ω^h;
5. Repeat the steps above.

If the sequence above is repeated for say 4-8 different meshes, each with successively higher resolution, the results of E (the error measured in some norm) versus h should be plotted on a $\log_{10} - \log_{10}$ graph. If the meshes used in the study are suitably fine and resolve the analytic solution well, then the plotted data in log-log space should lie along a straight line. Performing a linear regression should result in a correlation coefficient very close of 1.0. The gradient of the this log-log plot will define the order of accuracy of our numerical solution.

To understand this result, recall that the error one a given mesh domain Ω^h is assumed to vary with h according to

$$E(\Omega^h) \approx Ch^k, \tag{9.16}$$

where C is a constant independent of h. C is considered a function of the model problem, i.e. it relates to the shape of the domain used, the choice of boundary conditions and material properties used in the PDE. On a finer mesh, $\Omega^{h/r}$, we have

$$E(\Omega^{h/r}) \approx C\left(\frac{h}{r}\right)^k, \tag{9.17}$$

where r is a refinement factor. For simplicity in the steps 1-5 above we considered a refinement factor of $r = 2$. Combining Equations (9.16) & (9.17) we have

$$\frac{E(\Omega_1)}{E(\Omega_2)} = r^k \tag{9.18}$$

and now taking the $\log_{10}$ of both sides yields an expression for the order of accuracy k:

$$k = \log_{10}\left(\frac{E(\Omega_1)}{E(\Omega_2)}\right).\left[\log_{10}(r)\right]^{-1}. \tag{9.19}$$

Note that we are most interested in the behaviour of the error in the asymptotic limit, $h \to 0$. If your initial mesh was too coarse and failed to resolve the analytic solution, the computed errors may not lie along a straight line. In this case, one should consider only using the errors associated with the finest meshes considered.

Asymptotic behaviour

Measuring the order of accuracy k of a method only provides information concerning how the error changes (ideally decreases) as the mesh resolution h is decreased. For example, if a method is second order accurate $\mathcal{O}(h^2)$ (in some norm), ie. $k = 2$, then we can expect that the error (measure in the appropriate norm) obtained on a grid with resolution h, will reduce be a factor of four if we solve then solve on grid with resolution $h/2$. *The order of accuracy tells us nothing about the absolute value of the error*. Another way of stating this, is the order of accuracy gives us information about the slope of the $\log(E) - \log(h)$ graph, but doesn't provide us with the offset, i.e. the constant C in Equation (9.16) is unknown.

In the asymptotic limit of $h \to 0$, then a fourth order method will always be more accurate than a second order method. However, in practice we are often far from this asymptotic limit, thus the question of which method is better cannot be answered with only knowing the order of accuracy. This point is illustrated in Figure 91. Pure analysis *may* provide upper bounds on what the constant C might be (under quite strict assumptions), but for all practical purposes the constant C must be determined via numerical experiments.

9.5 The Method of Manufactured Solutions (MMS)

In all the previous sections it was taken for granted that we also had an analytic solution u which we could use to measure the error within our numerical solution, u^h. The classical approach to obtain an analytic solution is as follows. Given the PDE of interest, $\mathcal{L}u = f$:

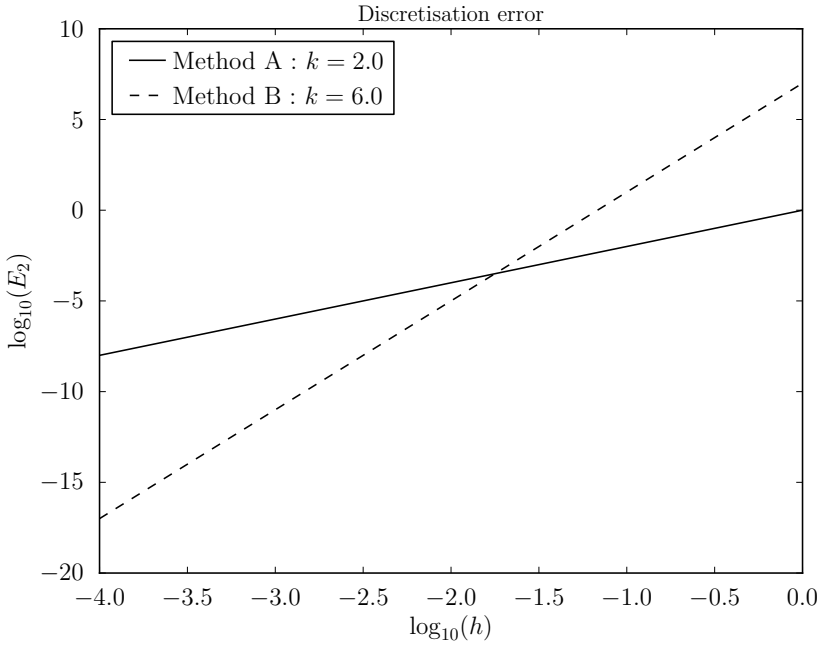


Figure 91: Discretisation errors for two methods with an order of accuracy of 6 (Method B) and 2 (Method A). Note the cross over point at $h \approx 1/56$

1. Choose a simple domain Ω (i.e. a box/cube);
2. Define the right hand side f;
3. Select your Dirichlet or Neumann boundary conditions;
4. Use all your skills and find a function u which satisfies the PDE and the boundary conditions chosen.

In general, analytic solutions for multi-dimensional problems can be difficult to obtain. The complexity of the obtaining analytic solutions continues to increase if we wish to consider coefficients (i.e. diffusivity, viscosity) which vary as a function of space, or if we were to consider a domain which was no longer naturally defined in the chosen coordinate system (i.e. a rectangle/cube in x, y, z). Up to this point, we have only considered *linear* problems. The problem is compounded if one seeks an analytic solution for non-linear problems. Frequently researchers have to resort to using analytic solutions for 1D, homogenous coefficient problems to verify their numerical model (e.g. Poiseuille flow). Using the solution from 1D Poiseuille flow is hardly appropriate to verify a code designed to solve 3D, variable viscosity Stokes flow. An alternative approach must be employed - this approach is called the *Method of Manufactured Solutions (MMS)*.

The basic idea to constructed manufactured solutions is completely general and extremely versatile. The concepts can be applied to arbitrarily complex PDE's (linear, non-linear, time dependent) and to arbitrarily complex domains and boundary conditions. The basic process of manufacturing a solution are described below. Given the PDE, $\mathcal{L}u = f$:

1. Choose the solution u_{MS};
2. Define any coefficients within $\mathcal{L}$;
3. Compute $f_{MS} = \mathcal{L}u_{MS}$ and *define the result to be your right hand side*, i.e. $f = f_{MS}$;
4. Given any domain Ω, evaluate u_{MS} or $\partial u_{MS}/\partial n$ to define any Dirichlet or Neumann boundary conditions respectively.

The procedure only requires to choose a function for the solution, and then apply the action of the differential operator $\mathcal{L}$. Whilst the algebra involved may be somewhat tedious to define the right hand side function f, we are only required to perform differentiation. The action of differentiation is easy (compared to integration) and can be readily performed by symbolic algebra packages such as Maple, MATLAB (using the symbolic toolbox) Mathematica, Maxima (`http://maxima.sourceforge.net`) or SymPy (symbolic python, `http://http://sympy.org`).

It is important to emphasize that the function chosen for u does not have to be physically meaningful. It is not important whether u "looks" like a solution you would typically obtain from your application. The purpose of manufacturing solutions is solely to ascertain that what is implemented in your code is producing discrete solutions of the quality expected by that particular numerical method.

There are some limitations to the method of manufactured solutions which should be mentioned. The function chosen for u and the coefficients in the operator $\mathcal{L}$, must be smooth and differentiable. Care should also be exercised that the constructed function f_{MS} is "well behaved", in the sense that it doesn't contain any singularities. This is easily confirmed by simply plotting the function. Furthermore, f_{MS} should be able to be resolved on the grid sequence you use when computing the order of accuracy. The coefficients should be chosen so that are physically reasonable, e.g. coefficients representing viscosity should not be negative.

Example: A linear PDE

The linear differential operator associated with the 1D diffusion equation, with a spatially variable coefficient is given by

$$\mathcal{L}u := \frac{d}{dx}\left(\kappa(x)\frac{d}{dx}\right)T. \tag{9.20}$$

First I choose the coefficient for the diffusivity to be

$$\kappa(x) = (4\tanh(10x) + 6)\exp(0.8x), \tag{9.21}$$

and the temperature to be

$$T_{MS}(x) = 8\sin(x)x^4 + 20. \tag{9.22}$$

Using sympy (see `mms-linear-Diffusion1D.py`), I obtain the following right hand side

$$\begin{aligned} f_{MS} &= \frac{d}{dx}\left(\kappa(x)\frac{d}{dx}\right)T_{MS} \\ &= [6 + 4\tanh(10x)][-8x^4\sin(x) + 64x^3\cos(x) + 96x^2\sin(x)]\exp(0.8x) \\ &+ [40 - 40(\tanh(10x)^2)][8x^4\cos(x) + 32x^3\sin(x)]\exp(0.8x) \\ &+ 0.8[6 + 4\tanh(10x)][8x^4\cos(x) + 32x^3\sin(x)]\exp(0.8x) \end{aligned} \tag{9.23}$$

The manufactured functions $\kappa(x)$ and u_{MS}, together with f_{MS} are shown in Figure 92. This example is demonstrated in the script `ML_FEM_1D_DiffusionVariableCoeff_MMS.m` and the order of accuracy is determined using the resulting output. The output is processed in the script `ML_FEM_OrderOfAccuracy.m`.

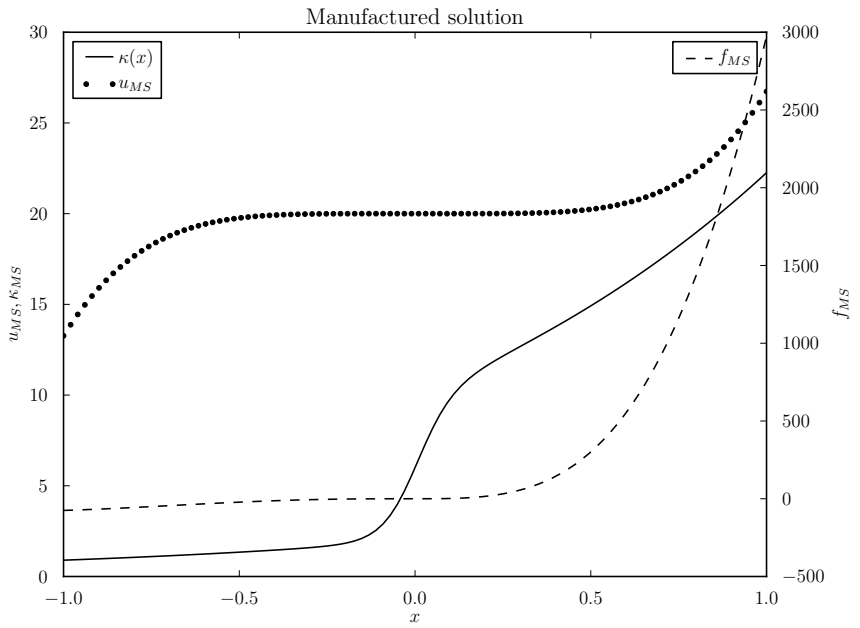


Figure 92: Manufactured solution for the 1D diffusion equation with a spatially variable diffusivity.

9.6 Exercises

1. Verify your 2D steady state, diffusion code which uses Q_1 (bilinear) elements. Choose a smooth, differentiable function for u_{MS} for the temperature. Use (i) a mesh of uniform quadrilateral elements, (ii) constant thermal diffusivity $\kappa = 13.4$ and (iii) the manufactured solution to specify Dirichlet boundary conditions everywhere. Use the global error measure in Equation (9.10). Follow the programming style in the example script `ML_FEM_1D_DiffusionVariableCoeff_MMS.m`. One important difference to note is that the example script uses 1-point Gauss quadrature to evaluate the error. In this example, you should use the 2×2 Gauss quadrature rule to evaluate Equation (9.10). Follow steps 1-5 in Section 9.4 and measure the order of accuracy of your FE solution by modify the script `ML_FEM_OrderOfAccuracy.m`. Deform the grid so that the elements are no longer uniform quadrilaterals and measure the order of accuracy. Does the order of accuracy change?

2. Verify your 2D elasticity code which uses Q_2 (biquadratic) elements. Choose a function for the displacement solution (u_{MS}, v_{MS}) and use (i) a mesh of uniform quadrilateral elements, (ii) constant elastic parameters and (iii) the manufactured

solution to specify Dirichlet boundary conditions for both the displacement degrees of freedom (u, v). Use the error defined in Equation (9.12). Evaluate this integral using the 3×3 Gauss quadrature rule. Compute the order of accuracy of your FE solution. As an additional test, make the elasticity coefficients vary in space and see if the order of accuracy changes. Note that you will have to re-compute $\mathbf{f}_{MS}$.

A

MATLAB Introduction

A.1 Introduction

This main goal of this part of the course is to learn how to solve the partial differential equations we spoke about earlier in the lecture. Most of the equations that are of interest to you as an Earth scientist are known. However, they can typically not be solved analytically (or if they can - it is a very complicated). At the same time, however, it is relatively straightforward to solve them with the aid of a computer. Although there are a variety of numerical techniques that one can use, the main focus here is on the Finite Element Method.

The focus here is on the practical problem of going from an equation to a solution. At the beginning of each lecture, a short introduction will be given and the rest of the lecture consists in writing code and solving exercises. The computer language we will use is MATLAB, which has a number of neat features, such as plotting or solving of linear systems of equations.

A.2 Useful linear algebra

MATLAB is entirely vector or linear algebra based. It is therefore useful to remind you of some of the linear algebra that you learned a long time ago.
Let's define a (column) vector $\mathbf{b}$ as

$$\mathbf{b} = \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix}$$

and a 2×3 matrix $\mathbf{D}$ as

$$\mathbf{D} = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix}.$$

The transpose (denoted with ^{T}) of $\mathbf{D}$ and $\mathbf{b}$ is given by:

$$\begin{aligned} \mathbf{D}^T &= \begin{pmatrix} 1 & 2 \\ 4 & 3 \\ 5 & 6 \end{pmatrix}, \\ \mathbf{b}^T &= \begin{pmatrix} 5 & 10 & 17 \end{pmatrix}. \end{aligned}$$

An example of matrix-vector multiplication:

$$\mathbf{Db} = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix} = \begin{pmatrix} 130 \\ 142 \end{pmatrix}.$$

An example of vector-vector multiplication (e.g. scalar product or dot product):

$$\mathbf{b}^T\mathbf{b} = \begin{pmatrix} 5 & 10 & 17 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix} = \begin{pmatrix} 414 \end{pmatrix}.$$

An example of matrix-matrix multiplication:

$$\mathbf{DD}^T = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 4 & 3 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 42 & 44 \\ 44 & 49 \end{pmatrix}.$$

In numerical modelling, after discretisation we will frequently end up with a system of linear equations of the form:

$$\mathbf{Ax} = \mathbf{f},$$

where $\mathbf{A}$ is an $n \times m$ matrix and $\mathbf{f}$ is a $n \times 1$ vector. The coefficients A_{ij} and f_i are all known, however the $m \times 1$ vector $\mathbf{x}$ is unknown. If we take $\mathbf{A} = \mathbf{D}^T$ and $\mathbf{f} = \mathbf{b}$, $\mathbf{x}$ is (check!):

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

A.3 Exploring MATLAB

MATLAB is a vector based computer language, which is available for Windows, Apple and Unix operating systems. It comes with its own programming language, which it is extremely useful due to its simplicity, though it can be slow.

Getting started

To start the program, type `matlab` at the unix prompt or launch a MATLAB application installed on your computer. The MATLAB command window starts.

1. Type `2+3`. You'll get the answer. Type `2 + 3*9 + 5^2`.
2. Type

```
>> x=3
>> y=x
>> z=x*y
>> pi
>> a=x*pi
```

3. Type `demo` and explore some examples.
4. Type `help`. You see a list of all help functions. Type `help log10` to get information about the log10 command.

Vectors/arrays and plotting

5. Create an array of x-coordinates:

```
>> dx=2
>> x=[0:dx:10]
```

6. Define a set of y-coordinates as a function of x:

```
>> y=x.^2 + exp(x/2)
```

7. Plot it:

```
>> plot(x,y)
```

8. Exercise: Make a plot of a parametric function. What does it look like?

```
>> t=0:.1:2*pi
>> x=sin(t); y=cos(t); plot(x,y,'o-')
>> xlabel('x')
>> ylabel('y')
>> axis image, title('fun␣with␣plotting')
```

Exercise: Make an ellipse out of it with a short radius of 1 and a long radius of 2. Change the colour of the curve to red.

Matrices and 3D plotting

First create `x` and `y` arrays, for example: `x=[1:5];y=x;`
9. Play with matrix products of `x` and `y`. Perform an element by element product of the two vectors (note the dot) via,

```
>> x.*y
```

The following command

```
>> x.'
```

returns the transpose of the vector. The "dot" or scalar product of two matrices is given by

```
>> x*y.'
```

The matrix product

```
>> x'*y
```

returns a matrix. Some commands (try them):

```
>> ones(1,5), ones(6,1)
>> length(x)
>> whos
```

10. Creating 2D matrices.
Matrics can be created directly, for example

```
>> M = [1 2 3; 4 5 6]
```

A useful function is meshgrid, which creates 2D arrays:

```
>> [x2d,y2d] = meshgrid(0:.1:5,1:.1:8)
```

You can obtain the size of an array using:

```
>> size(x2d)
```

11. Plot the function `sin(x2d.*y2d)`.

```
>> z2d = sin(x2d.*y2d)
>> surf(x2d,y2d,z2d)
>> mesh(x2d,y2d,z2d)
>> contour(x2d,y2d,z2d), colorbar
>> contourf(x2d,y2d,z2d), colorbar
```

Some cool stuff (1)

```
>> [x2d,y2d,z2d] = peaks(30);
>> surf(x2d,y2d,z2d); shading interp
>> light; lighting phong
```

Some cool stuff (2): perform the example given at the end of

```
>> help coneplot;
```

Some additional useful commands:
`clf`: clear current active figure
`close all`: close all figure windows

MATLAB scripting

By now you must be tired of typing all those commands all the time. Luckily, there is a MATLAB script language which allows you to type the commands in a text editor, save the result and re-use it later. MATLAB scripts are text files which end with the suffix ".m".
12. Open a text editor (e.g. emacs, vi, Xcode) and create a file "mysurf.m".
13. Type the plotting commands from the last section in the text file. A good programming convention is to start the script with `clear`, which clears the memory of MATLAB. Another good programming practice is to include lots of comments within your MATLAB script which describe what each line/block of code actually does. A comment can be placed after %, e.g.

```
% This is my first matlab script
```

14. Start the script from within MATLAB by going to the directory where the text file is saved. Type

```
>> mysurf
```

from within MATLAB and you should see the plot.

Loops

Create an array `na=100; a=sin(5*[1:na]/na); plot(a)`.
15. Ask instructions on using "for":

```
>> help for
```

16. Compute the sum of an array:

```
>> mysum=0; for i=1:length(a), mysum = mysum + a(i); end; mysum
```

17. Compare the result with the MATLAB inbuilt function **sum**

```
>> sum(a)
```

18. Exercise. Create an x-coordinate array: `dx=0.01; y=cos([0:dx:10])`. Compute the integral of $y = \cos(x)$ on the x-interval $0 \leq x \leq 10$. Use `sum(y)` and write a MATLAB script. Compare it with $\sin(10)$, the analytical solution.

Cumulative sum

19. Create a number of sedimentary layers with variable thickness:

```
>> thickness = rand(1,10); plot(thickness)}
```

20. Compute the depth of the interface between different layers:

```
>> depth(1)=0;
>> for i=2:length(thickness)
>>    depth(i) = depth(i-1)+thickness(i);
>> end;
>> plot(depth)
```

21. Compare the results with the built in MATLAB function cumsum:

```
>> bednumber=1:length(depth)
>> plot(bednumber,depth,bednumber,cumsum(thickness))
```

22. What causes the discrepancy? Try to remove it, read the description of the function

```
>> help cumsum
```

IF command

23. Ask MATLAB for information;

```
>> help if
```

Find maximum value within the above array `thickness`, and compare it with the built-in MATLAB function `max(thickness)`.

FIND command

24. Ask MATLAB for information;

```
>> help find
```

Find which layer has the maximum thickness: `find(thickness==max(thickness))`.
25. Find the number of beds with a maximum thickness less than 0.5.

Matrix operations

26. Exercise: Reproduce the linear algebra exercises in the beginning of this document. Hint: If you want to solve the system of linear equations `A x = f` for `x`, you can use the backslash operator: `x = A\f`.

B

Useful Formulae

Vector operations

$$\text{grad}(u) = \nabla u = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} u \tag{B.1}$$

$$\text{div}(\mathbf{v}) = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix} \cdot \mathbf{v} = \nabla^T \mathbf{v} = \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y} \tag{B.2}$$

Integration by parts (1D)

$$\int_a^b f(x) \frac{dg(x)}{dx}\, dx = -\int_a^b \frac{df(x)}{dx} g(x)\, dx + [f(x)g(x)]_a^b \tag{B.3}$$

Integration by parts (higher dimensions)

$$\int_\Omega f \frac{\partial g}{\partial x_i}\, dV = -\int_\Omega \frac{\partial f}{\partial x_i} g\, dV + \oint_{\partial\Omega} f g\, n_i\, dS \tag{B.4}$$

$$\int_\Omega \boldsymbol{f} \cdot \nabla g\, dV = -\int_\Omega g \nabla \cdot \boldsymbol{f}\, dV + \oint_{\partial\Omega} g \boldsymbol{f} \cdot \boldsymbol{n}\, dS \tag{B.5}$$

$$\int_\Omega \nabla f \cdot \nabla g\, dV = -\int_\Omega f \nabla^2 g\, dV + \oint_{\partial\Omega} f \nabla g \cdot \boldsymbol{n}\, dS \tag{B.6}$$

Tensor products

Given a vector of 1D functions, $P(s) = (P_1(s), P_2(s), \ldots, P_m(s))$, we can define a 2D tensor product with the coordinates (ξ, η)

$$\begin{aligned}
N_{11}(\xi,\eta) &= P_1(\xi)P_1(\eta) \\
N_{21}(\xi,\eta) &= P_2(\xi)P_1(\eta) \\
N_{31}(\xi,\eta) &= P_3(\xi)P_1(\eta) \\
&\vdots \\
N_{m1}(\xi,\eta) &= P_m(\xi)P_1(\eta) \\
N_{12}(\xi,\eta) &= P_1(\xi)P_2(\eta) \\
&\vdots \\
N_{m2}(\xi,\eta) &= P_m(\xi)P_2(\eta) \\
&\vdots \\
N_{mm}(\xi,\eta) &= P_m(\xi)P_m(\eta)
\end{aligned} \tag{B.7}$$

In general we have,

$$N_{ij} = P_i(\xi)P_j(\eta), \qquad \text{for all } 1 \leq i, j \leq m \tag{B.8}$$

A 3D tensor product can similarily be defined with the coordinates (ξ, η, ζ) via

$$N_{ijk} = P_i(\xi)P_j(\eta)P_k(\zeta), \qquad \text{for all } 1 \leq i, j, k \leq m \tag{B.9}$$

We note that the 1D basis used need not be the same length, for example one could define

$$N_{ijk} = U_i(\xi)V_j(\eta)W_k(\zeta), \tag{B.10}$$

where $U(s) = (U_1(s), \ldots, U_m(s))$, $V(s) = (V_1(s), \ldots, V_n(s))$, $W(s) = (W_1(s), \ldots, W_p(s))$ and where $m \neq n \neq p$.

C

1D Steady State Diffusion Example

```
%%
%% Script: FEM_1D_2nd_order_steadystate.m
%%
%% FEM code for 1D, 2nd order steady state
%% diffusion equation with a non-zero source term
%% Marcel Frehner, ETH Zurich, 2012
%%

% GENERAL STUFF
clear all
close all
clc

% GEOMETRICAL PARAMETERS
Lx = 10;

% PHYSICAL PARAMETERS
kappa  = 1;
source = -1;

% NUMERICAL PARAMETERS
el_tot   = 10;
n_tot    = el_tot+1;
n_per_el = 2;

% CALCULATED FROM ABOVE
dx      = Lx/el_tot;
GCOORD  = 0:dx:Lx;

% LOCAL-TO-GLOBAL MAPPING
EL_N = [ 1:n_tot-1 ; 2:n_tot ];

% BOUNDARY CONDITIONS
bc_dof = [ 1 n_tot ];
bc_val = [ 0 0 ];

% INITIALIZE GLOBAL STIFFNESS MATRIX AND GLOBAL FORCE VECTOR
KG = zeros(n_tot,n_tot);
FG = zeros(n_tot,1);

% ELEMENT LOOP
for iel = 1:el_tot
```

```
  % GET NODE INDICES FOR ELEMENT iel
  el_nodes   =   EL_N(:,iel);

  % DEFINE ELEMENT STIFFNESS MATRIX AND VECTOR
  Kloc    =   kappa/dx*[1 -1;-1 1];
  Floc    =   source*dx/2;

  % INSERT/SUM LOCAL ELEMENT STIFFNESS MATRICES
  % INTO GLOBAL STIFFNESS MATRIX
  KG(el_nodes,el_nodes)  = KG(el_nodes,el_nodes) + Kloc;
  FG(el_nodes)           = FG(el_nodes)          + Floc;

end

% APPLY BOUNDARY CONDITIONS
for ibc = 1:length(bc_dof)

  % GET NODE INDICES FOR BOUNDARY CONDITION AT NODE ibc
  bc_idx              =   bc_dof(ibc);

  % ZERO ROW AND INSERT 1 ON THE DIAGONAL
  KG(bc_idx,:)        =   0;
  KG(bc_idx,bc_idx)   =   1;

  % FORCE DIRICHLET VALUE INTO RHS VECTOR
  FG(bc_idx)          =   bc_val(ibc);

end

% SOLVE
T = KG\FG;

% COMPUTE ANALYTICAL SOLUTION
x_ana = 0:Lx/1000:Lx;
T_ana = -1/2*source/kappa.*x_ana.^2 + 1/2*source*Lx/kappa*x_ana;

% PLOT SOLUTIONS
plot(x_ana,T_ana,'k-',GCOORD,T,'ro');
xlim([0 Lx])
xlabel('x-coordinate␣[m]')
ylabel('T␣[K]')
legend('analytical␣solution','fem␣solution')
drawnow
```

D

Quadratic Basis Functions

One-dimension

Three node iso-parametric element

$$N_1(\xi) = \frac{1}{2}\xi\,(\xi - 1) \tag{D.1}$$

$$N_2(\xi) = 1 - \xi^2 \tag{D.2}$$

$$N_3(\xi) = \frac{1}{2}\xi\,(\xi + 1)\,. \tag{D.3}$$

Two-dimensions

Nine node iso-parametric element

Vertices (bottom left, clock-wise ordering)

$$N_1(\xi,\eta) = \frac{1}{4}(\xi^2 - \xi)(\eta^2 - \eta) \tag{D.4}$$

$$N_2(\xi,\eta) = \frac{1}{4}(\xi^2 - \xi)(\eta^2 + \eta) \tag{D.5}$$

$$N_3(\xi,\eta) = \frac{1}{4}(\xi^2 + \xi)(\eta^2 + \eta) \tag{D.6}$$

$$N_4(\xi,\eta) = \frac{1}{4}(\xi^2 + \xi)(\eta^2 - \eta) \tag{D.7}$$

Mid-faces (left face, clock-wise ordering

$$N_5(\xi,\eta) = \frac{1}{2}(\xi^2 - \xi)(1 - \eta^2) \tag{D.8}$$

$$N_6(\xi,\eta) = \frac{1}{2}(1 - \xi^2)(\eta^2 + \eta) \tag{D.9}$$

$$N_7(\xi,\eta) = \frac{1}{2}(\xi^2 + \xi)(1 - \eta^2) \tag{D.10}$$

$$N_8(\xi,\eta) = \frac{1}{2}(1 - \xi^2)(\eta^2 - \eta) \tag{D.11}$$

Center

$$N_9(\xi,\eta) = (1 - \xi^2)(1 - \eta^2) \tag{D.12}$$

E

MMS 1D Diffusion Example

```
%%
%% Script: ML_FEM_1D_DiffusionVariableCoeff_MMS.m
%%
%% FEM code for 1D diffusion equation with variable coefficient
%% diffusivity.
%% Demonstration of the Method of Manufactured Solutions (MMS)
%% 26.07.2011, Dave May, ETH Zurich
%%


function [] = main()

clear all, close all, clc

% Numerical parameters
% Mesh sequence,
% no_elements = { 4, 8, 16, 32, 64, 128, 256, 512, 1024 }
no_el       = 128;
no_nodes    = no_el + 1;
no_nodes_el = 2;
width       = 2;
X           = [-1:width/(no_el):-1+width];
dx          = X(2)-X(1);
% Initialization of matrices and vectors
KK  = zeros(no_nodes,no_nodes);
u   = zeros(no_nodes,1);
F   = zeros(no_nodes,1);

% Node numbering
for i=1:no_el
    iel = i;
    nodes(iel,1) = i;
    nodes(iel,2) = i+1;
end

% Setup system matrix
for iel=1:no_el
    ileft  = nodes(iel,1);
    iright = nodes(iel,2);
    x_centroid = 0.5 * ( X(iright) + X(ileft) );

    % Physical parameters evaluated at the element centroid
```

```
    A   = MMS_Coefficient( x_centroid );
    B   = MMS_RightHandSide( x_centroid );

    % Setup element matrix

    % Matrix for conductive terms, see Maple script
    K_loc   = [-A/dx  A/dx; A/dx  -A/dx];
    % Right hand side vector
    F_loc   = [B*dx/2 B*dx/2];

    for i=1:no_nodes_el
        ii = nodes(iel,i);
        for j=1:no_nodes_el
            jj = nodes(iel,j);
            KK(ii,jj) = KK(ii,jj) + K_loc(i,j);
        end
        F(ii) = F(ii) + F_loc(i);
    end
end

% Boundary conditions
% Evaluate the manufactured solution at the end points
KK(1,:)   = 0; KK(1,1)     = 1;
F(1)   = MMS_Solution( X(1)   ); % Left  boundary
KK(end,:) = 0; KK(end,end) = 1;
F(end) = MMS_Solution( X(end) ); % Right boundary

% Solve matrix
u = KK\F;

% Compute the L2 error using a 1-point quadrature over each element
L2_error = 0.0;
for iel=1:no_el
    ileft  = nodes(iel,1);
    iright = nodes(iel,2);

    % element volume
    dV             = X(iright) - X(ileft);

    % coordinate of element centroid
    x_centroid     = 0.5 * ( X(iright) + X(ileft) );

    % FE solution at centroid
    T_fem_centroid = 0.5 * ( u(iright) + u(ileft) );

    % MMS at centroid
    T_mms          = MMS_Solution( x_centroid );

    % Discretisation error at centroid
    T_error        = T_mms - T_fem_centroid;

    L2_error = L2_error + T_error * T_error * dV;
end
L2_error = sqrt( L2_error );
fprintf(1,'h␣=␣␣␣␣␣␣␣␣␣␣E_2␣=␣␣\n' );
fprintf(1,'%1.4e,␣␣%1.6e␣;␣\n', dx, L2_error );



% Evaluate the analytical solution and plot
```

```
L = width;
x = [-1:dx/2.0:-1+L];
for i=1:length(x)
    u_mms(i)  = MMS_Solution( x(i) );
end

% Graphics
plot(X,u,'-ok',x,u_mms,'-b');
xlabel('x'); ylabel('u'); grid on;
legend('finite element soln.','manufactured soln.',...
       'Location','SouthEast');
drawnow;


% =======================================================
%
% Manufactured coefficients, solution and right hand side
%
function val = MMS_Coefficient(x)
   val = ( 4.0 * tanh(10.0 * x) + 6.0 ) * exp(0.8 * x);

function val = MMS_Solution(x)
   val = 8.0 * sin(x)*x^4  + 20.0;

function val = MMS_RightHandSide(x)
   val = (6 + 4*tanh(10*x))*(-8*x^4*sin(x) + 64*x^3*cos(x) ...
   + 96*x^2*sin(x))*exp(0.8*x) ...
   + (40 - 40*tanh(10*x)^2)*(8*x^4*cos(x) + 32*x^3*sin(x))*exp(0.8*x) ...
   + 0.8*(6 + 4*tanh(10*x))*(8*x^4*cos(x) + 32*x^3*sin(x))*exp(0.8*x);
```

F

MMS Order of Accuracy Example

```
%%
%% Script: ML_FEM_OrderOfAccuracy.m
%%
%% Compute order of accuracy using L2 error measure
%%

clear all, close all, clc

%
% Data obtained from running
%  ML_FEM_1D_DiffusionVariableCoeff_MMS.m
% with the following mesh sequence
%   no_elements = { 4, 8, 16, 32, 64, 128, 256, 512, 1024 }
%

data = [
5.0000e-01,  2.375693e+00;
2.5000e-01,  6.843966e-01;
1.2500e-01,  1.770845e-01;
6.2500e-02,  4.460979e-02;
3.1250e-02,  1.117224e-02;
1.5625e-02,  2.794296e-03;
7.8125e-03,  6.986512e-04;
3.9062e-03,  1.746675e-04;
1.9531e-03,  4.366748e-05;
];

h = data(:,1);
E = data(:,2);

logh = log10(h);
logE = log10(E);

% plot log(h) vs log(E)
plot(logh,logE,'-ok', logh,logE,'-b');
xlabel('log10(h)'); ylabel('log10(E)'); grid on;
legend('convergence',4); drawnow;

% perform linear regression and obtain the correlation coefficient
p = polyfit(logh,logE,1);
R = corrcoef(logh,logE);
```

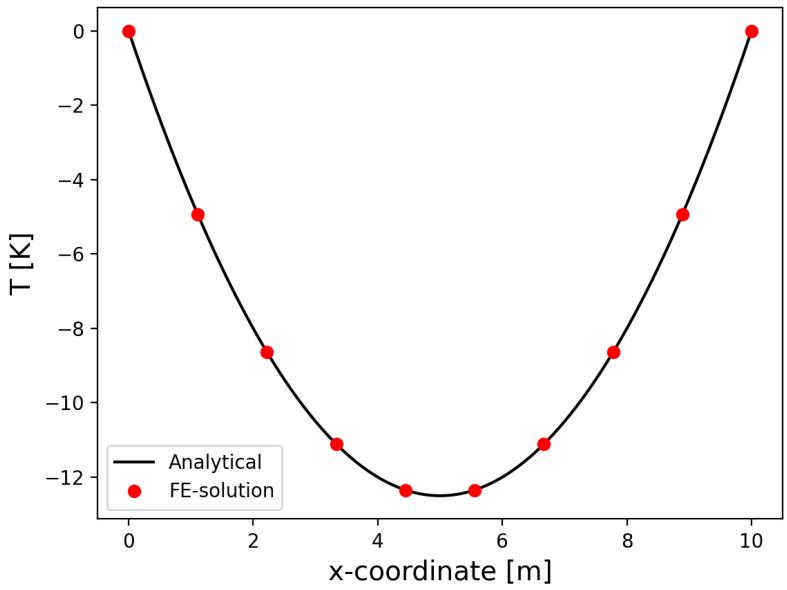
```
fprintf(1,'Order␣of␣accuracy:␣␣␣␣␣␣␣␣%1.4f␣\n', p(1) );
fprintf(1,'Correlation␣coefficient:␣%1.4f␣\n', R(1,2) );
```

G

NumPy 1D Steady State Diffusion Example

```
#!/usr/bin/env python
""" Steady-state diffusion, using finite elements and NumPy

Patrick Sanan, 2019
Based on a MATLAB template by Marcel Frehner
"""

import matplotlib.pyplot as plt
import numpy as np

# GEOMETRICAL PARAMETERS
Lx = 10.0  # length of model [m]

# PHYSICAL PARAMETERS
kappa = 1.0  # thermal diffusivity [m2/s]
source = -1.0  # heat source [K/s]
#
## NUMERICAL PARAMETERS
el_tot = 9  # #elements total
n_tot = el_tot + 1
n_per_el = 2  # #nodes per element

# CREATE NUMERICAL GRID
dx = Lx / el_tot  # distance between two nodes
GCOORD = np.linspace(0, Lx, n_tot)  # array of coordinates

# LOCAL-TO-GLOBAL MAPPING
# relates local to global node numbers per element
EL_N = np.array([np.arange(0, n_tot - 1),
                 np.arange(1, n_tot)])

# BOUNDARY CONDITIONS
bc_dof = [0, n_tot - 1]  # DOFs to which Dirichlet BCs are assigned
bc_val = np.array([[0.], [0.]])  # value for these dof's

## INITIALIZATION OF ALL KINDS OF STUFF
KG = np.zeros([n_tot, n_tot])  # global stiffness matrix
FG = np.zeros([n_tot, 1])  # global force vector

for iel in range(el_tot):  # ELEMENT LOOP
    n_now = EL_N[:, iel]  # which nodes are in the current element?
```

```
    # local stiffness matrix (3.3)
    Kloc = kappa / dx * np.array([[1., -1.], [-1., 1.] ])

    # local force vector (3.4)
    Floc = 0.5 * source * dx * np.ones([2, 1])

    #  add local matrices to global ones
    KG[np.ix_(n_now, n_now)] += Kloc  # note use of np.ix_()
    FG[n_now] += Floc

# APPLY BOUNDARY CONDITIONS
KG[bc_dof, :] = 0  # set bc-rows to zero
KG[bc_dof, bc_dof] = 1.0  # put 1 onto the main diagonal
FG[bc_dof] = bc_val  # set bc-value to the rhs

# SOLVER
T = np.linalg.solve(KG, FG)

# ANALYTICAL SOLUTION
x_ana = np.linspace(0, Lx, 1000)
T_ana = (-1 / 2 * source / kappa * x_ana**2 +
         (1 / 2 * source * Lx / kappa +
          (bc_val[1] - bc_val[0]) / Lx) * x_ana + bc_val[0])

# PLOTTING
plt.close()
plt.plot(x_ana, T_ana, 'k-')  # plot analytical solution
plt.plot(GCOORD, T, 'ro')  # plot numerical solution
plt.xlabel('x-coordinate␣[m]', size=14)  # label of the x-axis
plt.ylabel('T␣[K]', size=14)  # label of the y-axis
plt.legend(['Analytical', 'FE-solution'])  # add a legend
plt.show(block=False)
```



H

MMS SymPy Example

```
##
## Example demonstrating the method of
## manufactured solutions (MMS) using SymPy.
## The PDE used is the 1D diffusion equation
## with a spatially variable diffusivity, e.g.
##   d/dx ( kappa(x) dT/dx ) = f, for x \in [-1,1]
## The solution (T), and diffusivity (kappa)
## are chosen. The RHS (f) is manufactured
## using symbolic differentiation
##

from sympy import *
from sympy import var, Plot

x = Symbol('x')

# Chosen solution
T     = 8*sin(x) * x**4 + 20

# Chosen coefficient
kappa = ( 4*tanh(10*x) + 6 ) * exp(0.8*x)

# Compute, fMS = Lu
#               = d/dx(kappa(x).dT/dx)
dT_dx       = diff( T, x )
kappa_dT_dx = kappa * dT_dx

fMS         = diff( kappa_dT_dx, x )

print('[MMS␣chosen␣solution]')
print('␣␣T␣=␣' + ccode(T) + '\n')

print('[MMS␣chosen␣coefficients]')
print('␣␣kappa␣=' + ccode(kappa) + '\n')

print('[MMS␣right␣hand␣side]')
print('␣␣f_MS␣=␣' + ccode(fMS) + '\n')

print('[MMS␣boundary␣conditions]')

print('␣␣Dirichlet:')
T_eval = T.subs({x:'-1.0'})
```

```
result = T_eval.evalf()
print('␣␣␣␣T(x)|(x=-1.0)␣=␣' + str(result))

T_eval = T.subs({x:'1.0'})
result = T_eval.evalf()
print('␣␣␣␣T(x)|(x=+1.0)␣=␣' + str(result))

print('␣␣Neumann:')
kgradT_eval = kappa_dT_dx.subs({x:'-1.0'})
result      = kgradT_eval.evalf()
print('␣␣␣␣kappa(x).dT(x)/dx|(x=-1.0)␣=␣' + str(result))

kgradT_eval = kappa_dT_dx.subs({x:'1.0'})
result      = kgradT_eval.evalf()
print('␣␣␣␣kappa(x).dT(x)/dx|(x=+1.0)␣=␣' +  str(result))
```