

Numerical Geodynamics

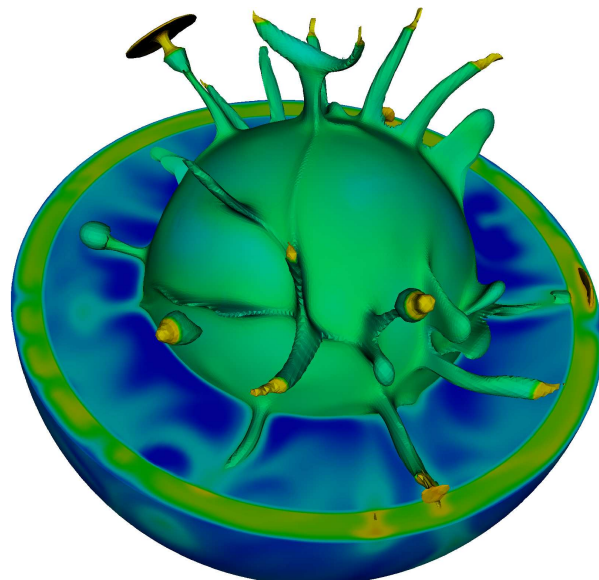
An introduction to computational methods with focus on solid Earth applications of continuum mechanics

A set of lecture notes and problem sets for
USC GEOL540 Geodynamics

Thorsten W. Becker
Department of Earth Sciences
University of Southern California
Los Angeles CA, USA

Boris J. P. Kaus
Department of Earth Sciences
ETH Zürich, Switzerland

Editing and additions by **Francois Cadieux**
University of Southern California
Los Angeles CA, USA



Version 1.1, August 5, 2010

Contents

1	Preliminaries	5
1.1	Acknowledgments	5
1.2	Availability and contact	5
1.3	Abbreviations used	5
1.4	Course objective	7
1.4.1	Example syllabus	8
1.4.2	Version history	10
2	Review material	11
2.1	Introductory notes on basic algebra and calculus	11
2.1.1	Linear algebra	11
2.1.2	Calculus	15
2.2	Continuum mechanics primer	21
2.2.1	Definitions and nomenclature	21
2.2.2	Stress tensor	22
2.2.3	Strain and strain rate tensors	23
2.2.4	Constitutive relationships (rheology)	24
2.2.5	Deriving a closed system of equations for a problem	24
2.2.6	Summary: The general system of equations for a continuum media in the gravity field.	26
2.3	Scaling analysis and non-dimensional numbers	28
2.3.1	Introduction	28
2.3.2	Problems	30
3	Introductory sketch of Numerical Geodynamics	33
3.1	Numerical methods in the Earth Sciences	33
3.1.1	Philosophy	33
3.1.2	Goals of this course	33
3.1.3	Textbooks and reading	34
3.1.4	Recommended Reading	34
3.1.5	Overview of applications of numerical methods for Earth sciences	34
3.1.6	Classification of numerical problems & solution methods	35
3.2	Examples of applications for numerical methods	35

3.2.1	Linear inverse problems	35
3.2.2	Ordinary differential equations	35
3.2.3	Partial differential equations	36
3.3	Computing	36
3.3.1	Hardware issues	37
3.3.2	Software - Computer Languages	37
3.3.3	Elements of a computer program	39
3.3.4	Guiding philosophy in writing a computer program	39
3.3.5	Guidelines for writing efficient code	40
3.4	Introduction to MATLAB	43
3.4.1	Introduction	43
3.4.2	Useful linear algebra (reprise)	43
3.4.3	Exploring MATLAB	44
4	Ordinary differential equations	50
4.1	Introduction	50
4.1.1	Initial Value Problems	51
4.1.2	Two-point Boundary Value Problems	52
4.2	Solution of initial value problem	52
4.3	Exercise: Solving Ordinary Differential Equations – Lorenz equations	55
4.3.1	Digression for background – not essential to solving this problem set	55
4.3.2	Problems	56
4.3.3	Additional experiments	58
5	Finite differences	59
5.1	Introduction to the finite difference method	59
5.1.1	Finite differences and Taylor series expansions	59
5.1.2	Finite difference approximations	61
5.1.3	Finite difference example	62
5.1.4	Exercises	64
5.2	Implicit FD schemes and boundary conditions	67
5.2.1	Variable time derivatives – explicit vs. implicit	67
5.2.2	Solution of example problem	69
5.2.3	Exercises	72
5.2.4	Derivation of flux boundary conditions (fictitious boundary points)	73
5.3	Non-linearities with FD methods	76
5.3.1	Example	76
5.4	Two-dimensional heat equation	78
5.4.1	Explicit method	78
5.4.2	Fully implicit method	79
5.4.3	Other methods	81
5.4.4	Exercises	82
5.5	Advection equations with FD	86

5.5.1	The diffusion-advection (energy) equation for temperature in convection . .	86
5.5.2	Advection (transport equations)	88
5.5.3	Advection and diffusion: operator splitting	96
6	Finite elements	99
6.1	Introduction to finite element methods	99
6.1.1	Philosophy of the finite element (FE) method	99
6.1.2	A one – dimensional example	101
6.1.3	Galerkin method	103
6.1.4	Discretization	104
6.2	A 1-D FE example	106
6.2.1	Local vs. global points of view	107
6.2.2	Matrix assembly	108
6.2.3	Element-local computations	109
6.3	Exercise: 1-D heat conduction and solution of linear systems	111
6.3.1	Implementation of 1-D heat equation example	111
6.3.2	Exercises	112
6.3.3	Solution of large, sparse linear systems of equations	113
6.4	Two-Dimensional boundary value problems with FE	118
6.4.1	Linear heat conduction	119
6.4.2	Matrix assembly	121
6.4.3	Isoparametric elements	123
6.4.4	Numerical integration	124
6.4.5	Simple elements, shape functions and Gaussian quadrature rules	126
6.5	Excercise: Heat equation in 2-D with FE	130
6.5.1	Implementation of 2-D heat equation	130
6.6	Exercise: Linear elastic, compressible finite element problem	134
6.6.1	Implementation of 2-D elasticity	135
6.7	Incompressible flow and elasticity and FE	141
6.7.1	Governing equations	141
6.7.2	FE solution to the incompressible elastic/flow problem	142
6.8	Exercise: Linear incompressible Stokes flow with FE	147
6.8.1	Implementation of incompressible, Stokes flow	147
6.8.2	Problem in strong form	149
6.8.3	Exercises	151
6.9	Time-dependent FE methods	154
6.9.1	Example: Heat equation	154
6.9.2	Solution of the semi-discrete heat equation	155

Chapter 1

Preliminaries

1.1 Acknowledgments

Parts of these course notes follow the treatment in the numerical analysis lecture notes by *Spiegelman* (2004), the textbook by *Press et al.* (1993) on numerical analysis, and the textbook by *Hughes* (2000) on finite element methods.

The finite difference exercises and the scaling homework assignment are loosely based on an ETH Zürich taught by Yuri Podladchikov. Most of the finite element exercises are built directly on the MILAMIN Matlab software which is openly distributed by *Dabrowski et al.* (2008). The multigrid problem set is based on an exercise by *Zhong* (2008), and one of the elasticity finite element exercises is inspired by a Numerical Analysis class taught by Harro Schmeling at Frankfurt University in 1997. Students who took the class at USC Earth Sciences in the Fall of 2005 and 2008 provided valuable feedback.

Partial funding for course development was provided by the US National Science Foundation under CAREER grant EAR-0643365.

1.2 Availability and contact

This PDF, its Latex source files and figures, as well as additional material for this class are available at the USC GEOL540 course web site

<http://geodynamics.usc.edu/~becker/teaching-540.html>.

In particular, the Matlab problem sets mentioned below are available at

http://geodynamics.usc.edu/~becker/teaching/540/matlab_problem_sets_usc_geol540.tgz.

Please send an email to Thorsten Becker (thorstinski -at- gmail.com) for any comments and questions.

1.3 Abbreviations used

BC Boundary conditions

FD Finite differences

FE Finite elements

IC Initial condition

ODE Ordinary differential equation

PDE Partial differential equation

1.4 Course objective

Assuming the use of shared, community software will continue to increase over the next years, geodynamics as a field faces the challenge to educate students in numerical analysis basics without having each PhD student write their own code. This class is supposed to help address this challenge and is geared toward all Earth science or engineering students. At the time of developing the class and lecture notes, no adequate textbook existed (but see [Gerya, 2009](#); [Ismail-Zadeh and Tackley, 2010](#)). This is why we developed our own set of documents for open web dissemination in the hope that they are useful.

The lecture notes and problems sets that are compiled in this document¹ form the basis of a one-semester, graduate level class at the University of Southern California that is geared toward students from both geology and geophysics. The goal is to introduce some of the fundamental concepts of numerical analysis within the context of solid Earth, mantle dynamics type of problems. The class includes regular lectures but is centered around hands-on, programming exercises using Matlab (see sec. 1.4.1).² As presented here, the class covers the solution of ordinary differential equations briefly, and then spends about equal time on finite difference and finite element methods for the solution of partial differential equations as they arise in continuum mechanics.

Students should ideally have had significant exposure to calculus, some linear algebra, a classic, introductory geodynamics (or continuum/rock mechanics) course (*e.g.*, based on the [Turcotte and Schubert, 2002](#), text), and have some introductory level knowledge of computer programming and Matlab. Earth science students have diverse backgrounds and often do not fulfill all of these math and programming prerequisites. We like to err on the side of learning by doing and supporting a broad group of students, and therefore also provide basic primers on calculus, linear algebra, continuum mechanics, and computer programming. The first week is spent learning some basic Matlab programming, later guiding students through the increasingly more involved programming needed with the problem sets as examples.

The class is meant to be an almost self-contained introductory survey of numerical modeling. This necessitates brushing over many technical or theoretical issues (for example, no mathematical proofs are given), and we also cannot give much room to the discussion of alternative, or cutting edge numerical methods. (PhD students at USC with a more geophysical background are encouraged to take more theoretically advanced classes in addition to this introductory course.)

Our goal is to provide all students with a sufficient working knowledge to solve simple research problems by reusing the Matlab codes, or by writing their own software. If complex, 3D problems are to be solved, students will often go on to use existing, shared software, such as the codes distributed by the Computational Infrastructure for Geodynamics (<http://geodynamics.org>). The basic insights into numerical analysis that are to be conveyed in this class should help make students educated and empowered users and developers of such codes.

¹The accompanying Matlab problem sets can be found at <http://geodynamics.usc.edu/~becker/teaching-540.html>, and a solved set of problem sets for instructors can be obtained by contacting TWB at thorstinski-at-gmail.com.

²We chose Matlab because of its ease in developing and debugging software and built-in visualization capabilities. The fact that the language is interpreted, and not compiled, does somewhat limit the scope of the applications of the solution methods discussed in class, pretty much to 2D problems, yet at high efficiencies [Dabrowski et al. \(2008\)](#).

Thorsten Becker and Boris Kaus, June 2010

1.4.1 Example syllabus

As an example for how the lecture notes and problem sets can be combined into a one semester course, we provide the syllabus as the class was taught in the Fall of 2008. Each week, the class met for a three hour slot which typically consisted of some formal instruction by means of lectures and joint, computer room, Matlab problem-set exercises. Some weeks, all class time is spent working on problem sets. The class culminates in a three week final project part where students are to either write their own code or combine codes used in class (for example, combine advection and diffusion solvers, and further with a Stokes solver to arrive at a self-contained convection code).

1. Introduction (chap. 3): Overview of numerical methods in Earth Sciences (sec. 3.1); Examples of applications for numerical methods in Earth Sciences (sec. 3.2); Computer hardware; Computer Language; Principles of Programming (sec. 3.3). Exercise: Matlab programming (sec. 3.4)

Notes: Introduction Handout: Math Problem set: Matlab

2. Ordinary differential equations (sec. 4): Definition of ODEs (sec. 4.1). Initial value problems (sec. 4.2). Euler method. Taylor expansions. Accuracy of numerical methods. Midpoint method, 4th order Runge Kutta. Exercise: Program and solve Lorentz equations (sec. 4.3).

Notes: ODEs Problem set: ODEs

3. Scaling analysis (sec. 2.3); Non-dimensionalization; Non-dimensional numbers (Rayleigh, Prandtl, Peclet, Reynolds, Deborah). Stokes velocities for Newtonian and non-Newtonian rheology; shear layers

Notes/problem set: Scaling

4. Finite differences I (sec. 5.1): 1-D heat equation. Explicit solution of diffusion problems. Stability.

Notes/problem set: Explicit FD

5. Finite differences II (sec. 5.2): Implicit methods. Crank-Nicolson method. Order of spatial and temporal accuracy. Stability conditions. Neumann and Dirichlet boundary conditions. Sparse matrices, triangularity. Linear systems of equations. Heat equation in 1-D.

Notes/problem set: Implicit FD methods

6. Finite differences III (sec. 5.3): Non-linear equations. Darcy flow equation for pressure-dependent diffusivity. Two-dimensional heat equation, solution with fully explicit and fully implicit methods (sec. 5.4). Comparison with analytical solutions.

Notes/problem set: Non-linear and 2-D FD methods

7. Finite differences IV (sec. 5.5): Advection equation for heat transport. FTCS method and stability. Lax method, Courant criterion. Upwind schemes. Staggered leapfrog. Semi-Lagrangian methods. Advection-diffusion combos in 2-D, operator splitting.

Notes/problem set: Advection equations and combos

8. Finite elements I (sec. 6.1): Introduction to the finite element method. Strong and weak forms of PDEs. Discretization of domains into finite elements. Shape functions. Bilinear forms. Variational approaches, virtual work. Galerkin method. One-dimensional heat equation example.

Notes: FE Intro

9. Finite elements II (sec. 6.2): Local and global coordinate systems. Change of variables during integration. Matrix assembly. Solution of linear systems of equations, direct and iterative methods. LU decomposition, Cholesky. Jacobi, Gauss-Seidel, Conjugate gradient, and multigrid methods.

Notes: FE Implementation Problem set: 1-D FE implementation and matrix inversion

10. Finite elements III (sec. 6.4): 2D boundary value problems. Isoparametric elements. Jacobian; global and element-local coordinates. Numerical integration using Gauss quadrature. Triangular and quadrilateral shape functions. Meshing using triangles. Solution of 2-D heat equation.

Notes: FE 2D, time dependent solution Problem set: 2-D FE heat equation

11. Finite elements IV (sec. 6.6): Compressible elastic problems. Elastic moduli, plane stress, plane strain. Gradient operator, elasticity matrix, engineering strain convection. Visualization of stress states, eigensystems.

Problem set: 2-D FE elastic

12. Finite elements V (sec. 6.8 & 6.7): Compressible and incompressible elasticity and Stokes flow. Mixed formulation with discontinuous pressure. Powell-Hestenen iterations.

Notes: Incompressible elastic/fluid problem Problem set: 2-D FE incompressible Stokes

13. Joint project work in computer lab.

1.4.2 Version history

1.1 (August 2010) - (Week 1) Exploring Matlab: GUI help and function browser

- (Week 2) Solving ODE's:
 - function for calculating derivatives used in R-K solver were put in a separate m-file <dydt.m>
 - code in <lorenz.m> and <rkstep.m> was changed accordingly
- (Week 3) Scaling:
 - suggestion for students to use a table and example headings
- (Week 4) FD Explicit 1-D Heat:
 - added detail about CFL (Courant) condition and concept of stability, numerical dissipation and aliasing with reference to proper textbook and chapter.
- (Week 5) FD Implicit 1-D Heat:
 - added requirement that students compare results to previous schemes
 - added requirement that students compare accuracy of different schemes near instability region.
- (Week 6) FD Non-Linear and 2-D Heat
 - added req that students compare linear and non-linear solns at each timestep to visualize the difference.
- (Week 7) FD Advection
 - Changed alpha (Courant number) in most <filename.m> to reflect defn in notes (it was incorrect before).
 - Added blurb about the reason why implicit methods are not a good choice for solving hyperbolic equations.
 - Added extra exercise and explanation: implement C-N and modified FE C-N scheme
 - Added solution file <exercise_1_c_n.m>
 - Added extra exercise and explanation: implement modified Galerkin Lax-Wendroff scheme
 - Added solution file <exercise_1_gLW.m>
- (Week 9) 1-D FE Example
 - Added another solver and an explanation: Successive Over Relaxation <sor.m>
 - Added code to implement option to use SOR or MATLAB CG in matlab_code dir
 - Removed code from <heatldfe.m> and added comments to ensure that students write their own stiffness matrix assembly code

1.0.3 (June 2010) Stylistic improvements including hyperlinks in PDF.

1.0 (June 2010) Initial completion of PDF lecture notes.

Chapter 2

Review material

2.1 Introductory notes on basic algebra and calculus

We here provide a few brief notes on math notation and background needed for this course (and USC classes GEOL440, GEOL534, or GEOL540). At places, these are not necessarily mathematically rigorous and you should refer to, *e.g.*, a math for engineers text for a more complete treatment. We will from now on assume that you are familiar with the matter treated in this section in class, please come and ask me if anything is unclear.

2.1.1 Linear algebra

The dot product

We will make use of the *dot product*, which is defined as

$$c = \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i, \quad (2.1)$$

where \mathbf{a} and \mathbf{b} are vectors of dimension n (n -dimensional, geometrical objects with a direction and length, like a velocity) and the outcome of this operation is a scalar (a regular number), c . In eq. (2.1), $\sum_{i=1}^n$ means “sum all that follows while increasing the index i from the lower limit, $i = 1$, in steps of of unity, to the upper limit, $i = n$ ”. In the examples below, we will assume a typical, spatial coordinate system with $n = 3$ so that

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3, \quad (2.2)$$

where 1, 2, 3 refer to the vector components along x , y , and z axis, respectively. When we write out the vector components, we put them on top of each other

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \quad (2.3)$$

or in a list, maybe with curly brackets, like so: $\mathbf{a} = \{a_1, a_2, a_3\}$. On the board, I usually write vectors as \underline{a} rather than \mathbf{a} , (which is used in these sets of lecture notes) because that's easier. You may also see vectors printed as bold face letters, like so: \mathbf{a} . (In the Einstein summation convention, we would rewrite $\sum_{i=1}^n a_i b_i$ simply as $a_i b_i$, where summation over repeated indices is implied, *i.e.* the \sum is not written.)

We can write the amplitude (or: length, L_2 norm) of a vector as

$$|\mathbf{a}| = \sqrt{\sum_i^n a_i^2} = \sqrt{a_1^2 + a_2^2 + a_3^2} = \sqrt{a_x^2 + a_y^2 + a_z^2}. \quad (2.4)$$

For instance, all of the base vectors defining the Cartesian coordinate system, \mathbf{e}_x , \mathbf{e}_y , and \mathbf{e}_z have unity length by definition, $|\mathbf{e}_i| = 1$. Those \mathbf{e}_i vectors point along the respective axes of the Cartesian coordinate system so that we can assemble a vector from its components like

$$\mathbf{a} = \{a_x, a_y, a_z\} = a_x \mathbf{e}_x + a_y \mathbf{e}_y + a_z \mathbf{e}_z. \quad (2.5)$$

For a spherical system, the \mathbf{e}_r , \mathbf{e}_θ , and \mathbf{e}_ϕ unity vectors can still be used to express vectors but the actual Cartesian components of \mathbf{e}_i depend on the coordinates at which the vectors are evaluated.

We can restate eq. (2.1) and give another definition of the dot product,

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \quad (2.6)$$

where θ is the angle between vectors \mathbf{a} and \mathbf{b} . The meaning of this is that if you want to know what component of vector \mathbf{a} is parallel to \mathbf{b} , you just take the dot product. Say, you have a velocity \mathbf{v} and want the normal velocity v_n along a vector \mathbf{n} with $|\mathbf{n}| = 1$ that is oriented rectangular to some plate boundary, you can use $v_n = \mathbf{v} \cdot \mathbf{n}$.

Also, eq. (2.5) only works because the base vectors \mathbf{e}_i of any coordinate system are, by definition, orthogonal (at right angle, perpendicular, at $\theta = 90^\circ$) to each other and $\mathbf{e}_i \cdot \mathbf{e}_j = 0$ for all $i \neq j$. Likewise, $\mathbf{e}_i \cdot \mathbf{e}_i = 1$ for all i since $\mathbf{a} \cdot \mathbf{a} = |\mathbf{a}|^2$, and base vectors have unity length by definition. Using the Kronecker δ

$$\delta_{ij} = 1 \quad \text{for } i = j, \quad \text{and} \quad \delta_{ij} = 0 \quad \text{for } i \neq j, \quad (2.7)$$

we can write the conditions for the basis vectors as

$$\mathbf{e}_i \cdot \mathbf{e}_j = \delta_{ij}. \quad (2.8)$$

Vector or cross product

This operation is written as $\mathbf{a} \times \mathbf{b}$ or $\mathbf{a} \wedge \mathbf{b}$ and its result is another vector

$$\mathbf{c} = \mathbf{a} \wedge \mathbf{b} \quad (2.9)$$

that is at a right angle to both \mathbf{a} and \mathbf{b} (hence the right-hand-rule, with thumb, index, and middle finger along \mathbf{a} , \mathbf{b} , and \mathbf{c} , respectively). vector \mathbf{c} 's length is given by

$$|\mathbf{c}| = |\mathbf{a} \wedge \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin \theta, \quad (2.10)$$

that is, \mathbf{c} is largest when \mathbf{a} and \mathbf{b} are orthogonal, and zero if they are parallel. Compare this relationship to eq. (2.6). In 3-D,

$$\mathbf{c} = \mathbf{a} \wedge \mathbf{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix} \quad (2.11)$$

(note that there is no i component of \mathbf{a} or \mathbf{b} in the i component of \mathbf{c} , this is the aforementioned orthogonality property). The cross product can also be written as the determinant of the matrix

$$\begin{pmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{pmatrix} \quad (2.12)$$

An example for a cross product is the velocity \mathbf{v} at a point with location \mathbf{r} in a body spinning with the rotation vector $\boldsymbol{\omega}$, $\mathbf{v} = \boldsymbol{\omega} \wedge \mathbf{r}$. The rotation vector $\boldsymbol{\omega}$ is different from, *e.g.*, \mathbf{r} in that $\boldsymbol{\omega}$ has a spin (a sense of rotation) to it (the other right-hand-rule, where your thumb points along the vector and your fingers indicate the counter-clockwise motion).

Matrices and tensors

A $n \times m$ matrix is a rectangular table of elements (or entries) with n rows and m columns which are filled with numbers. You will see matrices printed like so A , and we usually indicate them on the board by double underlining like $\underline{\underline{A}}$. The elements are referred to as A_{ij} where i is the row and j the column. Matrices can be added and or multiplied.

Multiplication of matrix with a scalar

$$fA = fA_{ij} = f \times \begin{pmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{pmatrix} = \begin{pmatrix} fa_{xx} & fa_{xy} & fa_{xz} \\ fa_{yx} & fa_{yy} & fa_{yz} \\ fa_{zx} & fa_{zy} & fa_{zz} \end{pmatrix} \quad (2.13)$$

Multiplication of a matrix with a vector

$$\begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} = \begin{pmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{pmatrix} \cdot \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_{xx}b_x + a_{xy}b_y + a_{xz}b_z \\ a_{yx}b_x + a_{yy}b_y + a_{yz}b_z \\ a_{zx}b_x + a_{zy}b_y + a_{zz}b_z \end{pmatrix} \quad (2.14)$$

or

$$c_i = \sum_j a_{ij} b_j. \quad (2.15)$$

Multiplication of two matrices works like this:

$$C = AB \quad (2.16)$$

$$c_{ij} = \sum_k a_{ik} b_{kj}, \quad (2.17)$$

where k goes from 1 to the number of columns in A , which has to be equal to the number of rows in B . Note that, in general, $AB \neq BA$!

Some special matrices and matrix properties

Quadratic matrices Have $n \times n$ rows and columns. All simple physical tensors, such as stress or strain, can be written as quadratic matrices in 3×3 .

Identity matrix $I = I, i_{ij} = \delta_{ij}$, i.e. this matrix is unity along the diagonal, and zero for all other elements.

Transpose of a matrix $(A^T)_{ij} = a_{ji}^T = a_{ji}$, i.e. the transpose has all elements flipped by row and column.

Inverse of A , A^{-1} : $A^{-1}A = AA^{-1} = I$. If the inverse exists, then $(A^{-1})^{-1} = A$, $(A^T)^{-1} = (A^{-1})^T$, and $(AB)^{-1} = B^{-1}A^{-1}$.

Orthogonal or rotation matrices: $AA^T = A^T A = I$

Eigenvalues and vectors: Any $n \times n$ symmetric matrix A has n eigenvectors \mathbf{v}_i that correspond to real eigenvalues λ_i such that

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad (2.18)$$

An example is the stress matrix which can be written in the principal axes system, where the eigenvectors of the Cartesian representation of the stress matrix are the principal axes.

Decomposition Any quadratic tensor A can be decomposed into a symmetric part A^s (for which $a_{ij}^s = a_{ji}^s$) and an anti-symmetric part A^a (for which $a_{ij}^a = -a_{ji}^a$) like $A = A^s + A^a$ (*Cartesian decomposition*). In the case of the deformation matrix F , we call the symmetric part *strain* E (the infinitesimal strain tensor, $\underline{\underline{\epsilon}}$), and the anti-symmetric part corresponds to a rotation R . The *polar decomposition* is also of interest; we can write $F = RU = VR$ where R is a rotation matrix and U and V are the right- and left-stretch matrices, respectively, and $V = (FF^T)^{1/2}$. The left-stretch matrix describes the deformation in the rotated coordinate system after the rotation R has been applied to the body.

Tensors

The stress σ and strain ϵ are examples of second order (rank $r = 2$) tensors which, for $n = 3$, 3-D operations, have 3^2 components and can be written as $n \times n$ matrices. You will see tensors printed like so $\underline{\underline{E}}$, and we usually indicate them on the board by double underlining like $\underline{\underline{E}}$, making no distinction between tensors and matrices.

Tensors in a Cartesian space are defined by their properties under coordinate transformation. If a quantity \mathbf{v} remains intact under rotation to a new coordinate system \mathbf{v}' such that

$$v'_i = L_{ij}v_j = \sum_{j=1}^3 L_{ij}v_j \quad (2.19)$$

holds, then \mathbf{v} , a vector, is a first order tensor. L_{ij} may be, for example, a rotation matrix. Likewise, a second order tensor \mathbf{T} is defined by remaining intact after rotation into another coordinate system where it is expressed as \mathbf{T}' such that

$$T'_{ij} = L_{ik}T_{kl}L_{jl} = \sum_k L_{ik} \sum_l T_{kl}L_{jl} = \mathbf{L}\mathbf{T}\mathbf{L}^T \quad (2.20)$$

2.1.2 Calculus

Full and partial derivatives

In calculus, we are interested in the *change* or *dependence* of some quantity, *e.g.* u , on small changes in some variable x . If u has value u_0 at x_0 and changes to $u_0 + \delta u$ when x changes to $x_0 + \delta x$, the incremental change can be written as

$$\delta u = \frac{\delta u}{\delta x}(x_0)\delta x. \quad (2.21)$$

The δ (or sometimes written as capital Δ) here means that this is a small, but finite quantity. If we let δx get asymptotically smaller around x_0 , we of course arrive at the *partial derivative*, which we denote with ∂ like

$$\lim_{\delta x \rightarrow 0} \frac{\delta u}{\delta x}(x_0) = \frac{\partial u}{\partial x}. \quad (2.22)$$

The limit in eq. (2.22) will work as long as u doesn't do any funny stuff as a function of x , like jump around abruptly. When you think of $u(x)$ as a function (some line on a plot) that depends on x , $\partial u / \partial x$ is the slope of this line that can be obtained by measuring the change δu over some interval δx , and then making the interval progressively smaller.

We call $\frac{\partial u}{\partial x}$ (we also write in shorthand $\partial_x u(x)$ or $u'(x)$; if the variable is time, t , we also use $\dot{u}(t)$ for $\partial u / \partial t$) the *partial derivative*, because u might also depend on other variables, *e.g.* y and z . If this is the case, the *total derivative* du at some $\{x_0, y_0, z_0\}$ (we will drop (*i.e.* not write down) the explicit dependence on the variables from now on) is given by the sum of the changes in all variables on which u depends:

$$du = \frac{\partial u}{\partial x}dx + \frac{\partial u}{\partial y}dy + \frac{\partial u}{\partial z}dz. \quad (2.23)$$

Here, dx and similar are placeholders for infinitesimal changes in the variables. This means that eq. (2.23) works as long as dx is small enough that a linear relationship between δu and δx still holds. In fact, we can (Taylor) approximate *any* $u(x)$ around x_0 by

$$u(x) = u(x_0) + \frac{\partial u}{\partial x}(x_0)(x - x_0) + \frac{\partial^2 u}{\partial x^2}(x_0)\frac{(x - x_0)^2}{2!} + \frac{\partial^3 u}{\partial x^3}(x_0)\frac{(x - x_0)^3}{3!} \dots \quad (2.24)$$

Here, $\frac{\partial^2 u}{\partial x^2}$ is the second derivative, the change of the change of u with x . $n!$ denotes the factorial, *i.e.*

$$n! = 1 \times 2 \times 3 \times \dots n. \quad (2.25)$$

So, as long as $dx = x - x_0$ is small, the derivative will work (for well behaved u). For example, if better approximations are needed, *e.g.* when the strain tensor is not infinitesimal anymore, quadratic and higher terms like the one that goes with the second derivative in the series eq. (2.24) and so on need to be taken into account.

How to compute derivatives Here are some of the most common derivatives of a few functions:

function $f(x)$	derivative $f'(x)$	comment
x^p	px^{p-1}	special case: $f(x) = c = cx^0 \rightarrow f'(x) = 0$ where c, p are constants
$\exp(x) = e^x$	e^x	that's what makes e so special
$\ln(x)$	$1/x$	
$\sin(x)$	$\cos(x)$	
$\cos(x)$	$-\sin(x)$	
$\tan(x)$	$\sec^2(x) = 1/\cos^2(x)$	

If you need to take derivatives of combinations of two or more functions, here called f , g , and h , there are four important rules (with a and b being constants):

Chain rule (inner and outer derivative):

$$\text{If } f(x) = h(g(x)) \quad (2.26)$$

$$f'(x) = h'(g(x))g'(x), \quad (2.27)$$

i.e. derivative of nested functions are given by the outer times the inner derivative.

Sum rule:

$$(af(x) + bg(x))' = af'(x) + bg'(x) \quad (2.28)$$

Product rule:

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \quad (2.29)$$

Quotient rule:

$$\text{If } f(x) = \frac{g(x)}{h(x)} \quad (2.30)$$

$$f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2} \quad (2.31)$$

If you need higher order derivatives, those are obtained by successively computing derivatives, e.g. the third derivative of $f(x)$ is

$$\frac{\partial^3 f(x)}{\partial x^3} = \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} f(x) \right) \right). \quad (2.32)$$

Say, $f(x) = x^3$, then

$$\frac{\partial^3 x^3}{\partial x^3} = \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} x^3 \right) \right) = \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} 3x^2 \right) = \frac{\partial}{\partial x} 6x = 6. \quad (2.33)$$

Divergence and curl

Operators are mathematical constructs that do something with the entity that is written to their right. For example, we had earlier introduced the *gradient operator*, ∇ (the del operator is represented by the “Nabla” symbol ∇), which takes derivatives in all directions and, in a Cartesian system, is given by $\nabla = \left\{ \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right\}$. Here, we explicitly write the vector symbol around ∇ but sometimes we just write ∇ because we know that it’s actually a vector operator. When applied to scalar field (a distribution of values that depends on spatial location), such as a temperature distribution $T(x, y, z)$, the *gradient* operation

$$\text{grad } T = \nabla T = \begin{pmatrix} \frac{\partial T(x, y, z)}{\partial x} \\ \frac{\partial T(x, y, z)}{\partial y} \\ \frac{\partial T(x, y, z)}{\partial z} \end{pmatrix} \quad (2.34)$$

generates a vector from the scalar field which points in the direction of the steepest increase in T .

Consider what ∇ can do to a vector field. If $\mathbf{v} = \{u, v, w\}$ is a velocity field, then the *divergence* (grad dot product) operation on a vector field

$$\text{div } \mathbf{v} = \nabla \cdot \mathbf{v} \quad (2.35)$$

is equivalent to finding the dilatancy (volumetric) strain Δ from the strain tensor components because

$$\Delta = \frac{\Delta V}{V} = \sum_i \epsilon_{ii} = \epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = \nabla \cdot \mathbf{v}. \quad (2.36)$$

(Here V is volume, and ΔV volume change and, mind you, $\epsilon_{ij} = 0.5 (\partial_j u_i + \partial_i u_j)$). Eq. (2.36) illustrates that the divergence has to do with sinks and sources, or volumetric effects. The volume

integral over the divergence of a velocity field is equal to the surface integral of the flow normal to the surface. (An electro-magnetics example: For the magnetic field: $\text{div } \mathbf{B} = 0$ because there are no magnetic monopoles, but for the electric field: $\text{div } \mathbf{E} = q$, with electric charges q being the “source”.)

If we take the vector instead of the dot product with the grad operator, we have the *curl* or *rot* operation

$$\text{curl } \mathbf{v} = \nabla \wedge \mathbf{v}. \quad (2.37)$$

The curl is a rotation vector just like ω . Indeed, if the velocity field is that of a the rigid body rotation, $\mathbf{v} = \omega \wedge \mathbf{r}$, one can show that $\nabla \wedge \mathbf{v} = \nabla \wedge (\omega \wedge \mathbf{r}) = 2\omega$.

Second derivatives enter into the *Laplace* operator which appears, *e.g.* in the diffusion equation:

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \quad (2.38)$$

Some rules for second derivatives:

$$\text{curl}(\text{grad} T) = \nabla \times (\nabla T) = 0 \quad (2.39)$$

$$\text{div}(\text{curl } \mathbf{v}) = \nabla \cdot \nabla \times \mathbf{v} = 0 \quad (2.40)$$

Integrals

Taking an integral

$$F(x) = \int f(x) dx, \quad (2.41)$$

in a general (indefinite) sense, is the inverse of taking the derivative of a function f ,

$$F\left(\frac{\partial f(x)}{\partial x}\right) = f(x) + c \quad (2.42)$$

$$\frac{\partial}{\partial x} F\left(\frac{\partial f(x)}{\partial x}\right) = \frac{\partial}{\partial x} (f(x) + c) = f'(x). \quad (2.43)$$

Any general integration of a derivative is thus only determined up to an integration constant, here c , because the derivative, which is the reverse of the integral, of a constant is zero.

Graphically, the definite (with bounds) integral over $f(x)$

$$\int_a^b f(x) dx = F(b) - F(a) \quad (2.44)$$

along x , adding up the value of $f(x)$ over little chunks of dx , from the left $x = a$ to the right $x = b$ corresponds to the area under the curve $f(x)$. This area can be computed by subtracting the analytical form of the integral at b from that at a , $F(b) - F(a)$. If $f(x) = c$ (c a constant), then

$$F(x) = cx + d \quad (2.45)$$

$$F(b) = cb + d \quad (2.46)$$

$$F(a) = ca + d \quad (2.47)$$

$$F(b) - F(a) = c(b - a), \quad (2.48)$$

the area of the box $(b - a) \times c$.

Here are the integrals (anti derivatives) of a few common functions, all only determined up to an integration constant C

function $f(x)$	integral $F(x)$	comment
x^p	$\frac{x^{p+1}}{p+1} + C$	special case: $f(x) = c = cx^0 \rightarrow F(x) = cx + C$
e^x	$e^x + C$	
$1/x$	$\ln(x) + C$	
$\sin(x)$	$-\cos(x) + C$	
$\cos(x)$	$\sin(x) + C$	

There are also a few very helpful definite integrals without closed-form anti derivatives, *e.g.*

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2} \quad (2.49)$$

A standard math textbook, table of integrals, the Mathematica software, or Wikipedia will be of help with more complicated integrals.

A few conventions and rules for integration:

Notation: Everything after the \int sign is usually meant to be integrated over up to the dx , or the next major mathematical operator if the dx is placed next to the \int if the context allows:

$$\int (af(x) + bg(x) + \dots) dx = \int af(x) + bg(x) \dots dx \quad (2.50)$$

$$\int dx f(x) = \int f(x) dx \quad (2.51)$$

Linearity:

$$\int_a^b (cf(x) + dg(x)) dx = c \int_a^b f(x) dx + d \int_a^b g(x) dx \quad (2.52)$$

Reversal:

$$\int_a^b f(x) dx = - \int_b^a f(x) dx \quad (2.53)$$

Zero length:

$$\int_a^a f(x) dx = 0 \quad (2.54)$$

Additivity:

$$\int_a^b f(x) dx = \int_a^c f(x) dx + \int_c^b f(x) dx \quad (2.55)$$

Product rules:

$$\int f'(x)f(x)dx = \frac{1}{2}(f(x))^2 + C \quad (2.56)$$

$$\int f'(x)g(x)dx = f(x)g(x) - \int f(x)g'(x)dx \quad (2.57)$$

Quotient rule:

$$\int \frac{f'(x)}{f(x)}dx = \ln|f(x)| + C \quad (2.58)$$

2.2 Continuum mechanics primer

The preparatory class for GEOL540 is GEOL534 Lithospheric Deformation where continuum mechanics is discussed in the context of geodynamics with focus on the lithosphere; a good reference for such problems is *Turcotte and Schubert (2002)*. However, here is a short and extremely simplified review of basic continuum mechanics as it pertains to the remainder of the class. You may wish to refer to our math review if notation or concepts appear unfamiliar, and consult chap. 1 of *Spiegelman (2004)* for some clean derivations.

2.2.1 Definitions and nomenclature

- Coordinate system. $\mathbf{x} = \{x, y, z\}$ or $\{x_1, x_2, x_3\}$ define points in 3D space. We will use the regular, Cartesian coordinate system throughout the class for simplicity.

Note: Earth science problems are often easier to address when inherent symmetries are taken into account and the governing equations are cast in specialized spatial coordinate systems. Examples for such systems are polar or cylindrical systems in 2-D, and spherical in 3-D. All of those coordinate systems involve a simpler description of the actual coordinates (*e.g.* $\{r, \theta, \phi\}$ for spherical radius, co-latitude, and longitude, instead of the Cartesian $\{x, y, z\}$) that do, however, lead to more complicated derivatives (*i.e.* you cannot simply replace $\partial/\partial y$ with $\partial/\partial \theta$, for example). We will talk more about changes in coordinate systems during the discussion of finite elements, but good references for derivatives and different coordinate systems are *Malvern (1977)*, *Schubert et al. (2001)*, or *Dahlen and Tromp (1998)*.

- Field (variable). For example $T(x, y, z)$ or $T(\mathbf{x})$ – temperature field – temperature varying in space.
- Indexed variables. For example, the velocity field $\mathbf{v}(\mathbf{x}) = v_i$ with $i = 1, 2, 3$ implies $\{v_1, v_2, v_3\}$, *i.e.* three variables that are functions of space $\mathbf{x} = \{x_1, x_2, x_3\}$.
- Repeated indices indicate summation over these components (also called Einstein summation convention).

$$\frac{\partial v_i}{\partial x_i} \quad \text{with } i = 1, 2, 3 \quad \text{implies} \quad \sum_{i=1}^3 \frac{\partial v_i}{\partial x_i} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_2} + \frac{\partial v_3}{\partial x_3} \quad (2.59)$$

- In a *Eulerian frame* one uses a reference system for computations that is fixed in space, for example a computational box in which we solve for advection of temperature T in a velocity field \mathbf{v} . Local changes in, *e.g.*, temperature are then given by

$$\frac{DT}{Dt} = \frac{\partial T}{\partial t} + \mathbf{v} \nabla T = \frac{\partial T}{\partial t} + v_i \frac{\partial T}{\partial x_i}, \quad (2.60)$$

where D/dt is the *total derivative* that we would experience if we were to ride on a fluid particle in the convection cell (*Lagrangian* reference frame). D/Dt takes into account local

changes in a property with time (*e.g.* due to radioactive heating for T) as well as advection of temperature anomalies by means of \mathbf{v} in and out of our local observation point.

- Tensor = indexed variable + the rule of transformation to another coordinate system.
- Useful tensor – the *Kronecker* δ (*delta*), in 3D:

$$\delta_{ij} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (2.61)$$

- Traction = a force per unit area acting on a plane (a vector).
- Mean stress (= –pressure, p), strain: $-p = \bar{\sigma} = \sigma_{ii}/3 = tr(\sigma)/3, \bar{\epsilon} = \epsilon_{ii}/3 = \theta$ (also called dilatation).
- Traction/stress sign convention. Compression is negative in physics, but usually taken positive in geology. Pressure is always positive compressive.
- Deviatoric stress, strain: $\tau = \tilde{\sigma} = \sigma_{ij} - \bar{\sigma}\delta_{ij}, \tilde{\epsilon} = \epsilon_{ij} - \bar{\epsilon}\delta_{ij}$.

2.2.2 Stress tensor

- A matrix, two indexed variables, tensor of rank two:

$$\sigma_{ij} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} (2D) \quad (2.62)$$

$$\sigma_{ij} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} (3D). \quad (2.63)$$

- Meaning of the elements: Each row are components of the traction vectors acting on the coordinate plane normal to the respective coordinate axis, the diagonal elements are normal stresses, and off-diagonal elements are shear stresses.

σ_{ij} : force/area (traction) on the i plane (plane with normal aligned with the i -th coordinate axis) along the j direction.

- Special properties: Symmetric, *i.e.* $\sigma_{ij} = \sigma_{ji}$. This means that only six components of σ need to be stored during computations since the other three can be readily computed. *Note:* There are different convention for the order of storing elements of σ (*e.g.* diagonal elements first, then off-diagonal; alternatively, upper right hand side ordering).

- Cauchy's formula: if you multiply the stress tensor (treated as a matrix) by a unit vector, n_j , which is normal to a certain plane, you will get the traction vector on this plane (see above):

$$T_{(n)i} = \sigma_{ij}n_j = \sum_{j=1}^3 \sigma_{ij}n_j \quad (2.64)$$

- In a model, the stress tensor is usually computed by solving the equilibrium equations.

Note: The number of equilibrium equations is less than the number of unknown stress tensor components.

2.2.3 Strain and strain rate tensors

- A matrix, two indexed variables, tensor of rank two, like the stress matrix.
- Meaning of the elements: Diagonal elements are elongation (rate), *i.e.* the relative changes of length in coordinate axes directions), off-diagonal elements are shears, *i.e.* deviations from 90° of the angles between lines coinciding with the coordinate axes directions before deformation.
- Special properties: symmetric.
- Strain and strain-rate tensors are a measure of the infinitesimal (small, of order %, as opposed to finite, *i.e.* large) deformation (rate). Strain and strain-rates connect to stress (forces) via the rheological (constitutive) relationships.
- Computed from the spatial gradients of displacements u and velocities v for strain and strain-rate, respectively.

$$\epsilon_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (2.65)$$

$$\dot{\epsilon}_{ij} = \frac{1}{2} \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) \quad (2.66)$$

$$= \begin{pmatrix} \frac{\partial v_1}{\partial x_1} & \frac{1}{2} \left(\frac{\partial v_1}{\partial x_2} + \frac{\partial v_2}{\partial x_1} \right) & \frac{1}{2} \left(\frac{\partial v_1}{\partial x_3} + \frac{\partial v_3}{\partial x_1} \right) \\ \frac{1}{2} \left(\frac{\partial v_2}{\partial x_1} + \frac{\partial v_1}{\partial x_2} \right) & \frac{\partial v_2}{\partial x_2} & \frac{1}{2} \left(\frac{\partial v_2}{\partial x_3} + \frac{\partial v_3}{\partial x_2} \right) \\ \frac{1}{2} \left(\frac{\partial v_3}{\partial x_1} + \frac{\partial v_1}{\partial x_3} \right) & \frac{1}{2} \left(\frac{\partial v_3}{\partial x_2} + \frac{\partial v_2}{\partial x_3} \right) & \frac{\partial v_3}{\partial x_3} \end{pmatrix} \quad (2.67)$$

- *Note:* The number of velocity components is smaller than the number of strain rate components.
- *Note:* Engineering strain, γ , is often used by commercial finite element packages and $\gamma = 2\epsilon_{xy}$.

2.2.4 Constitutive relationships (rheology)

- A functional relationship between second rank tensors for kinematics ($\dot{\epsilon}$, ϵ) and dynamics (forces, σ). For example,

elastic: $\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij}$

incompressible viscous: $\sigma_{ij} = -p \delta_{ij} + 2\eta \dot{\epsilon}_{ij}$

Maxwell visco-elastic (for deviators): $\dot{\epsilon}_{ij} = \frac{\tilde{\sigma}_{ij}}{2\mu} + \frac{\tilde{\sigma}_{ij}}{2\eta}$

Here, λ, μ are elastic moduli (for an isotropic medium, there are two (bulk and shear) independent moduli which can be related to all other commonly used parameters such as Poisson's ratio). η is (dynamic, shear) viscosity, bulk viscosities are usually assumed infinite. Sometimes, kinematic viscosity $\nu = \eta/\rho$ is used.

- To solve a problem starting from the equilibrium equations for force balance, one can replace stress by strain (rate) via the constitutive law, and then replace strain (rate) by displacement (velocities). This results in a “closed” system of equations in “fundamental” variables, meaning that the number of equations is equal to the number of unknowns, the basic displacements (velocities).
- Material parameters for solid Earth problems can ideally be obtained by measuring rheology in the lab. Alternatively, indirect inferences from seismology or geodynamic modeling augmented by constraints such as post-glacial rebound need to be used.
- There are three major classes of rheologies:
 - Reversible elastic rheology at small stresses and strains over short time scales.
 - Irreversible fluid flow (creep) at large strains and over long time scales. Examples are Newtonian viscous (rate-independent) or power-law (rate/stress dependent) rheology; usually thermally activated. Intermediate stress levels.
 - Rate-independent (instantaneous), catastrophic yielding at large, limit stresses. Pressure sensitive, often temperature independent. Also called plastic, or frictional (brittle), behavior. Important for cold material over long time-scales.

2.2.5 Deriving a closed system of equations for a problem

Conservation laws

Conservation of mass (continuity equation)

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \mathbf{v}) = \frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0, \quad (2.68)$$

where ρ is density and \mathbf{v} velocity. For an incompressible medium, this simplifies to

$$\nabla \cdot \mathbf{v} = 0 \quad \text{or} \quad \frac{\partial v_i}{\partial x_i} = \sum_{i=1}^3 \frac{\partial v_i}{\partial x_i} = 0. \quad (2.69)$$

In 2D, the incompressibility constraint can be incorporated by solving for a *stream function* (see the Lorenz problem) instead of the actual velocities. If, instead, the fundamental variables v are solved for, special care needs to be taken to ensure eq. (2.69) holds.

Conservation of momentum (equilibrium force balance)

$$\frac{D\mathbf{v}}{Dt} = \nabla \sigma + \rho \mathbf{g}, \quad (2.70)$$

or

$$\frac{Dv_i}{Dt} = \rho \left(\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j} \right) = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \quad (2.71)$$

where \mathbf{g} is gravitational acceleration.

Conservation of energy

$$\left(\frac{\partial E}{\partial t} + v_j \frac{\partial E}{\partial x_j} \right) + \frac{\partial q_i}{\partial x_i} = \rho Q \quad (2.72)$$

where E is energy, q_i the energy flux vector, and Q an energy source (heat production).

Thermodynamic relationships

Energy (heat) flux vector *vrs.* temperature gradient (Fick's law)

$$q_i = -k \frac{\partial T}{\partial x_i} \quad (2.73)$$

where k is the thermal conductivity.

Equation of state 1 (“caloric” equation)

$$E = c_p \rho T \quad (2.74)$$

where c_p is heat capacity, and T is temperature. If all material parameters are constant (homogeneous medium), we can then write conservation of energy as

$$\frac{DT}{Dt} = \frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \kappa \nabla^2 T + H \quad (2.75)$$

or

$$\frac{\partial T}{\partial t} + v_j \frac{\partial T}{\partial x_j} = \kappa \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_i} T + H = \kappa \sum_i \frac{\partial^2 T}{\partial x_i^2} + H \quad (2.76)$$

with $H = Q/\rho$ and the thermal diffusivity

$$\kappa = \frac{k}{\rho c_p}. \quad (2.77)$$

Equation of state 2: relationships for the isotropic parts of the stress/strain tensors

$$\rho = f(T, p) \quad (2.78)$$

where p is pressure (note: $\rho = \rho_0 \epsilon_{kk}$).

Equation of state 3: Boussinesq approximation Material is assumed incompressible for all equations but the momentum equation where density anomalies are taken to be temperature dependent

$$\Delta \rho = \alpha \rho_0 \Delta T, \quad (2.79)$$

with α the thermal expansivity and $\Delta \rho$ the density difference from reference state ρ_0 for temperature difference ΔT from reference temperature T .

2.2.6 Summary: The general system of equations for a continuum media in the gravity field.

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0 \quad (2.80)$$

$$\rho \left(\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{\partial x_j} \right) = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \quad (2.81)$$

$$\left(\frac{\partial E}{\partial t} + v_j \frac{\partial E}{\partial x_j} \right) + \frac{\partial q_i}{\partial x_i} = \rho Q \quad (2.82)$$

$$E = c_p \rho T \quad (2.83)$$

$$\rho = f(T, P) \quad (2.84)$$

$$\tilde{\epsilon}_{ij} = R(\tilde{\sigma}_{ij}, \tilde{\sigma}_{ij}) \quad (2.85)$$

$$q_i = -k \frac{\partial T}{\partial x_i} \quad (2.86)$$

where ρ is density, v_i velocity, g_i gravitational acceleration vector, E energy, q_i heat flux vector, Q an energy source (heat production, *e.g.* by radioactive elements), c is heat capacity, T temperature, p pressure and k thermal conductivity. R indicates a general constitutive law.

Known functions, tensors and coefficients: g_i , c_p , $f(\dots)$, ρ_0 , $R(\dots)$, and k

Unknown functions: ρ , v_i , p , $\tilde{\sigma}_{ij}$, q_i , and T . The number of unknowns is thus equal to the number of equations.

Example: The Stokes system of equations for a slowly moving incompressible linear viscous (Newtonian) continuum

$$\frac{\partial v_i}{\partial x_i} = 0 \quad (2.87)$$

$$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho_0 g_i = 0 \quad (2.88)$$

$$\rho_0 c_p \left(\frac{\partial T}{\partial t} + v_j \frac{\partial T}{\partial x_j} \right) = \frac{\partial}{\partial x_i} \left(k \frac{\partial T}{\partial x_i} \right) + \rho_0 Q \quad (2.89)$$

$$\tilde{\epsilon}_{ij} = \frac{\tilde{\sigma}_{ij}}{2\eta} \quad (2.90)$$

$$\sigma_{ij} = -p\delta_{ij} + \tilde{\sigma}_{ij} \quad (2.91)$$

Major simplifications: No inertial ($D\rho/Dt$) terms (infinite Prandtl number, see non-dimensional analysis), incompressible flow, linear viscosity.

2D version, spelled out

Choice of coordinate system and new notation for 2D:

$$g_i = \{0, -g\}, x_i = \{x, z\}, v_i = \{v_x, v_z\}, \sigma_{ij} = \begin{pmatrix} \sigma_{xx} & \sigma_{xz} \\ \sigma_{zx} & \sigma_{zz} \end{pmatrix} (\sigma_{zx} = \sigma_{xz}).$$

The 2D Stokes system of equations (the basis for basically every mantle convection/lithospheric deformation code):

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = 0 \quad (2.92)$$

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z} = 0 \quad (2.93)$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z} - \rho g = 0 \quad (2.94)$$

$$\sigma_{xx} = -p + 2\eta \frac{\partial v_x}{\partial x} \quad (2.95)$$

$$\sigma_{zz} = -p + 2\eta \frac{\partial v_z}{\partial z} \quad (2.96)$$

$$\sigma_{xz} = \eta \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right) \quad (2.97)$$

$$\rho_0 c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} \right) = k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2} \right) + \rho_0 Q \quad (2.98)$$

2.3 Scaling analysis and non-dimensional numbers

Reading

- *Spiegelman* (2004), sec. 1.4
- *Turcotte and Schubert* (2002), Google, and Wikipedia for reference and material parameters

2.3.1 Introduction

Scaling analysis

Scaling analysis refers to order of magnitude estimates on how different processes work together if we are interested in getting a quick idea of the values that are of relevance for a problem. For example, shear stress τ for a Newtonian viscous rheology with viscosity η is given by

$$\tau = 2\eta\dot{\epsilon} \quad (2.99)$$

where $\dot{\epsilon}$ is the strain-rate. Say, we wish to estimate the shear stress in a part of the crust that we know is being sheared at some (*e.g.* plate-) velocity v over a zone of width L . The strain-rate in 3-D is really a tensor with 3×3 components that depends on the spatial derivatives of the velocity like so

$$\dot{\epsilon}_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \quad (2.100)$$

and has to be either constrained by kinematics or inferred for the full solution. However, for our problem, we only need a “characteristic” value, *i.e.* correct up to a factor of ten or so. Strain-rate is physically the change in velocity over length, and the characteristic strain-rate is then given by

$$\dot{\epsilon} \propto \frac{v}{L} \quad (2.101)$$

where \propto means “proportional to”, or “scales as”, to indicate that eq. (2.101) is not exact. Assuming we know the viscosity η , we can then estimate the typical stress in the shear zone to be

$$\tau \propto 2\eta \frac{v}{L}. \quad (2.102)$$

If you think about the units of all quantities involved (“dimensional analysis”), then this scaling could not have worked out any other way. Viscosity is Pa s (stress times time), velocity m/s (length over time), so stress=Pa s m/(s m)=Pa as it should be. (We will always use SI units unless it’s convenient for Earth applications, where we might use multiples of SI units such as cm/yr instead of m/s for velocities.)

Non-dimensionalization

A complementary approach that also takes into account the dimension of variables is to simplify the governing equations by defining “characteristic” quantities and then dividing all properties by those to make them “non-dimensional”. This way, the non-dimensional quantities that enter the equation on their own should all be of order unity so that the resulting collection of parameters in some part of the equation measures their relative importance.

A classic example for this is based on the Navier Stokes equation for an incompressible, Newtonian fluid. When body forces driving flow are due to temperature T fluctuations in (the Earth’s) gravitational field

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p_d + \eta \nabla^2 \mathbf{v} + \rho_0 \alpha T \mathbf{g} \quad (2.103)$$

where D is the total, Lagrangian derivative ($D = \partial/\partial t + \mathbf{v} \cdot \nabla$), \mathbf{v} velocity, ∇ the Nabla derivative operator $\nabla = \{\partial/\partial x, \partial/\partial y, \partial/\partial z\}$, t time, p_d the dynamic pressure (without the hydrostatic part), η the viscosity, ρ_0 reference density, α , and \mathbf{g} gravitational acceleration. One can now choose (as mentioned as for the Lorenz equations) typical quantities that can be derived from the given parameters such as a ΔT temperature difference, a fluid box height d , and some choice for the timescale. All other characteristic values for physical properties can then be derived from those choices.

A typical one is to use the diffusion time that can be constructed from the thermal diffusivity, κ , in the energy equation

$$\frac{DT}{dt} = \kappa \nabla^2 T \quad (2.104)$$

(no heat sources) that couples with the momentum equation, eq. (2.103). Because κ has units of $\text{length}^2/\text{time}$, any diffusion-related time scale t_d for a given length d has to work out like

$$t_d \propto d^2/\kappa \quad (2.105)$$

(see above). Using the characteristic quantities f_c which result from this scaling for all variables in eq. (2.103) and eq. (2.104), *e.g.*

$$v_c = \frac{d}{t_c} \quad \dot{\epsilon}_c = \frac{v_c}{d} \quad \tau_c = \eta \dot{\epsilon}_c \quad T_c = \Delta T \quad (2.106)$$

we divide all variables (spatial and temporal derivatives are dealt with like space and time variables) to make them unit-less, non-dimensional $f' = f/f_c$, and eq. (2.103) can then be written as

$$\frac{1}{Pr} \frac{D'v'}{D't'} = -\nabla' p' + (\nabla')^2 \mathbf{v}' - Ra T' \mathbf{e}_z \quad (2.107)$$

where we’ve used $\mathbf{g} = g \mathbf{e}_z$. Often, we just drop the primes and write the equation like so

$$\frac{1}{Pr} \frac{Dv}{Dt} = -\nabla p + \nabla^2 \mathbf{v} - Ra T \mathbf{e}_z \quad (2.108)$$

where it is implied that all quantities are used non-dimensionalized. This equation may still be hard to solve, but at least we now have sorted all material parameters into two numbers, Ra and Pr .

Note: The non-dimensional versions of the equations are also the best choice if you want to write a compute program for any physical problem. Using non-dimensionalized equations, all terms should be roughly of order unity, and the computer will not have to multiply terms that are very large in SI units (*e.g.* η) with those that are very small (*e.g.* \mathbf{v} , how large are η and $|\mathbf{v}|$ typically for mantle convection?). This also means that when some geophysicist's convection code spits out, say, velocities, you will have to check what units those have, and more often than not you'll have to multiply by the v_c from above to get back m/s, which you'll then convert to cm/yr. You'll also note that a few geodynamics papers will not provide you with the scaled quantities so that you can go back to SI units; sometimes this is because the parameters values used in the models are actually quite far from what they should be compared to the Earth.

Going back to eq. (2.108), all material parameters have been collected in two unit-less numbers after non-dimensionalization, the Prandtl number,

$$Pr = \frac{\eta}{\rho\kappa} \quad (2.109)$$

and the Rayleigh number

$$Ra = \frac{\rho_0 g \alpha \Delta T d^3}{\kappa \eta}. \quad (2.110)$$

Particularly the latter is key for mantle convection, and you will get to discuss both below. Fluid dynamics is full of these non-dimensional numbers which are usually named after some famous person because they are so powerful. Any fluid that has the same Ra and Pr number as another fluid will behave exactly the same way in terms of the overall style of dynamics, such as the resulting average temperature structure and up and downwelling morphology.

The actual time scales of convection, *e.g.*, may, however, be very different for two systems at the same Rayleigh number (because of v_c being different). This behavior allows, for example, to conduct analog, laboratory experiments of mantle convection (*e.g.* [Jacoby and Schmeling, 1981](#)). When conducting such experiments, care needs to be taken that all relevant non-dimensional numbers agree between the real Earth problem and the laboratory experiment (*e.g.* [Weijermars and Schmeling, 1986](#)). Also, when changing length scales and material, different physical effects such as surface tension may matter in the lab, while they are irrelevant for mantle convection in general (see, *e.g.*, sec. 6.7 of [Ricard, 2007](#), for a discussion of Mahagoni convection).

From an analytical point of view, if the non-dimensional quantities are either very large or very small, we can simplify the full equations to more tractable special cases. For a nice and more comprehensive treatment of this section, you may want to refer to [Ricard \(2007\)](#) (the PDF is on our web page).

2.3.2 Problems

1. For all of the following non-dimensional numbers, discuss briefly (2-3 sentences) the processes which these numbers measure, *e.g.* by contrasting system behavior for $Th = 0$ and $Th = \infty$, where Th is some non-dimensional number.

For each number, give numerical estimates for the Earth, at the present day. Document your choices for individual parameters before computing joint quantities, mention where you got

the estimates from, and what the implications for Earth in terms of the dynamics are. A neat way to organize this might be to use a table for each dimensionless number with appropriate headings (*e.g.* Parameter, estimate, reference).

You might have to look up definitions and other reference material, *e.g.* in a geodynamics text, or on google (*note*: don't trust everything on the web ...). There are no unique answers for this part of the problem set, and you will often have to decide on an example problem for which you'll pick a characteristic quantity such as length. Some answers are actively debated in the literature.

- (a) Rayleigh number for whole and upper mantle convection.
- (b) Peclet number for ridges, slabs, and general mantle convection. The Peclet number is defined as

$$Pe = \frac{vd}{\kappa} \quad (2.111)$$

with characteristic length d , velocity v , and thermal diffusivity κ .

- (c) Prandtl number for the mantle and the atmosphere. Once you've figured out the meaning of the Prandtl number, think of the different response of the mantle to an applied pulse of change in plate motion, compared to an applied pulse of heating.
- (d) Reynolds number for the mantle, the ocean, and a tornado. The Reynolds number is defined as

$$Re = \frac{vd}{\nu} = \frac{vd\rho}{\eta}. \quad (2.112)$$

Note: Take care to distinguish between velocity v , kinematic viscosity $\nu = \eta/\rho$ and dynamic viscosity η .

- (e) Deborah number for a pancake of the subducting oceanic lithosphere. The Deborah number can be defined as

$$De = \frac{t_r}{t_p} \quad (2.113)$$

where you can use a Maxwell time

$$t_M = \frac{\eta}{\mu} \quad (2.114)$$

for the relaxation time t_r , and t_p is the time scale of observation. The Maxwell time measures the visco-elastic relaxation time of a body with viscosity η and shear modulus μ (think post-glacial rebound).

- What are characteristic Maxwell times for the crust? The upper mantle?

2. (a) Consider a solid, sinking sphere of radius a in a fluid of viscosity η and gravitational pull g , and a density contrast between sphere and fluid of $\Delta\rho$. Solve for the approximate sinking velocity of this "Stokes" sinker by equating the gravitational pull force $F_P = \Delta M g = V \Delta \rho g$ with the shear force acting on the sphere's area A , $F_S \propto \tau A \propto A \eta \dot{\epsilon}_c$. Here, I've used ΔM for the mass anomaly, and V for the volume of the sphere. All equations follow from $F = ma$ and stress = force / area and some geometry.

- (b) For flow induced by a Stokes sinker, does the stress scale with η and/or Δp ? How does that compare with the velocities?
- (c) Estimate the Stokes velocity by dimensional analysis as in (a), but now assuming that the viscosity of the fluid obeys a power-law,

$$\tau^n \propto \eta \dot{\epsilon} \quad (2.115)$$

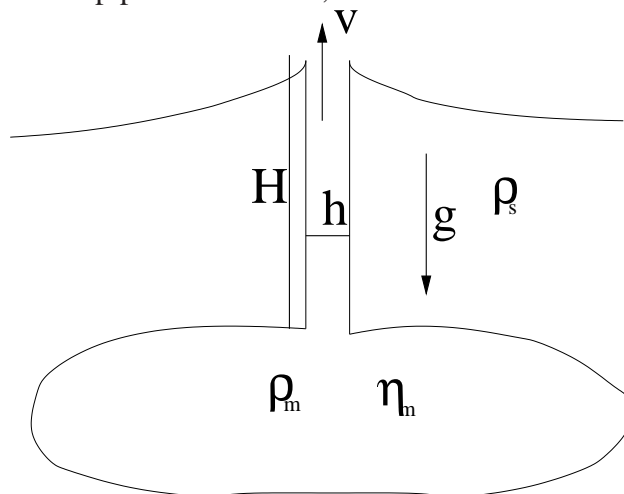
(for rocks, $n \sim 3$) instead of

$$\tau \propto \eta \dot{\epsilon} \quad (2.116)$$

for Newtonian creep as assumed above. (These equations are written sloppily and don't have the right units. For correct units, consider a relationship like $\tau(\tau/\mu)^{n-1} = \eta \dot{\epsilon}$, but you may use eq. (2.115) for the scaling analysis.)

- (d) Estimate the rise velocity of a plume head large enough to cause the Deccan traps.
3. You are moving the top of a fluid layer of height d at constant speed $v(z = d) = v_0$, and the fluid is held fixed at the bottom at $z = 0$. In this case, the laminar solution for the flow velocity is a linear decrease of velocity with depth to $v(0) = 0$ at the bottom.
- (a) What material parameters set the stress in the fluid? What determines the strain-rate and how does it vary with depth?
 - (b) Now consider two fluid layers, with the top fluid viscosity larger than the bottom one by a factor of two. Sketch the solution for the dependence of $v(z)$.
4. Using dimensional analysis, such as used above for the Stokes sinker, estimate the velocity of a volcanic eruption (see Figure below for parameters).

Hint: You might want to proceed by first using the equations for laminar, pressure-driven (look up “Hagen-Poiseuille”) flow in a pipe of radius R , and then estimate the pressure



difference from the Figure below.

Chapter 3

Introductory sketch of Numerical Geodynamics

3.1 Numerical methods in the Earth Sciences

3.1.1 Philosophy

- Avoid black boxes (*e.g.* commercial codes) in general. They may or may not do what you like them to do; if they don't, you're out of luck because if you cannot modify the source code. Exception for “good” black boxes are matrix solver and linear algebra packages, generally speaking (but see sec. [6.3.3](#)).
- Create, or understand, as much code and theory as possible yourself, no matter if you are geophysicist or geologist.
- There is no “Easy” button, but you don't have to be a math-whiz either!

3.1.2 Goals of this course

- Provide you with a basic understanding of numerical modeling, using solid Earth science problems as an example.
- Ideally, you should be able to solve a simple research problem using tools we've discussed in class plus those you write yourself. (Start thinking about a project!)
- Make you an informed, empowered user.
- Introduce some math and computer science along the way.

This course cannot

- be mathematically thorough (no proofs, etc.)

- be comprehensive
- be cutting edge

because we need to cover a lot of ground.

3.1.3 Textbooks and reading

- These lecture notes (available at geodynamics.usc.edu/~becker/preprints/Geodynamics540.pdf) and handouts
- *Myths and Methods in Modeling* by [Spiegelman \(2004\)](#)
- *Matlab Introduction* by [Spencer and Ware \(2008\)](#)
- Elsevier Treatise article on *Numerical methods in mantle convection* by [Zhong et al. \(2007\)](#)
- Possibly [Gerya \(2009\)](#) and [Ismail-Zadeh and Tackley \(2010\)](#), for reference (haven't read those yet)

3.1.4 Recommended Reading

- *Numerical Recipes*, 2nd or 3rd edition (possibly available online as PDF) ([Press et al., 1993](#))
- *Geodynamics* by [Turcotte and Schubert \(2002\)](#) for background
- *The finite element method* by [Hughes \(2000\)](#), if you need more detail on FE.
- The finite element classic by [Bathe \(2007\)](#) also makes for good reading.

3.1.5 Overview of applications of numerical methods for Earth sciences

Slides showing examples from

- inverse theory (seismic tomography)
- lithospheric dynamics
- mantle dynamics
- seismology
- magneto - hydrodynamics

3.1.6 Classification of numerical problems & solution methods

Forward Problem

1. formulate model
2. identify theoretical description
3. solve
 - dimensional analysis
 - analytical solution
 - check if this is a standard problem someone else has solved
 - check if terms can be neglected to simplify
 - check if equations can be linearized
 - numerical solution

Distinguish between model and simulation

A good model is as simple as possible to satisfy the most important constraints with the smallest number of parameters, to understand the underlying physics. A simulation tries to mimic what a system looks like, in a kitchen sink, lots of parameters kind of approach. To some extent, this distinction is a matter of taste, but a good model can provide the fundamental description needed to understand the *why* of Earth's dynamics.

Inverse problem

1. Formulate a model (*e.g.* Earth's mantle wave speed variations are smooth)
2. Identify theory (*e.g.* can treat seismic waves as rays)
3. Collect data
4. Solve a (linear) inverse problem

3.2 Examples of applications for numerical methods

3.2.1 Linear inverse problems

→ computational linear algebra

3.2.2 Ordinary differential equations

→ Runge-Kutta, Burlisch-Stoer integration methods, for example (this will be dealt with in the next lecture, see [4.2](#))

3.2.3 Partial differential equations

Finite Differences (FD)

approximate differentials by Taylor series, then approximate equations

- Pros**
- conceptually simple
- Cons**
- bad for sharp contrasts
 - bad for complicated geometries

We will look at FD during the first part of the course.

Finite Elements (FE)

complicated conceptually, approximate solution to eqn.

- Pros**
- good for sharp boundaries
 - good for complicated geometries
 - allow easy lateral mesh refinement
- Cons**
- coding more complicated
 - need to carefully choose elements, integration methods, etc.

Spectral methods

expand spatial solution as harmonic functions (can use FFT), solve time evolution as ODE for coefficients

- Pros**
- just for homogeneous media
- Cons**
- non-local
 - poor performance for lateral variations in material properties (need to iterate)

Gas lattice and other microscopic methods

unclear how to relate micro rules to continuum PDE's.

3.3 Computing

At USC, GEOL425 (Quantitative methods in the Earth Sciences) and the C language summer short course will provide you with some more exposure to basic computer programming skills. Here, we very briefly note some hardware related issues, give a few programming tips, and then move on to get started programming using the Matlab language in the next section. You can also refer to the hardware notes of *Press et al. (1993)* for some background on machine architecture.

3.3.1 Hardware issues

At a low level, a computer stores information in the binary system, *i.e.* in bits that can hold the values of either zero or one. You can then use a byte (8 bits) to encode numbers from 0 to $2^8 - 1 = 255$ using the binary system. For floating point or larger integers, more memory is required. A single precision float takes up four bytes and is accurate up to $\sim 5 \cdot 10^{-7}$, a double precision float up to $\sim 5 \cdot 10^{-15}$.

- A numerical representation of a float will always be approximate (only integers are exact). This means to not test for $x == 0$ (equal to zero) but $abs(x) < \epsilon$ ($abs(x) = |x|$) where ϵ depends on implementation.
- The detailed storage depends on the hardware, “big endian” *vs.* “small endian” ?
- Some mathematical operations that are theoretically valid will lead to large round off errors. *e.g.* $\cos^{-1}(x)$ for small x , subtracting large numbers from each other.
- The memory requirements for a float vector will be half of that of a double.

Memory

1 MB (megabyte) corresponds 1024×1024 bytes; 1 GB = 1024 MB. As of 2008, your PC will likely have at least ~ 2 GB of Random Access Memory or RAM (as opposed to hard drive space) meaning you can store how many float and double? To increase the available memory, one can use formerly called “supercomputers”. Those consist these days mainly of

Distributed memory machines *e.g.* $200 \times 2 \times$ quadcore (8 Central Processing Units or CPUs) \times 8 GB RAM machines which need specially designed software to make use of parallelism, *e.g.* Message Passing Interface or MPI.

Shared memory machines This is the more expensive, old school approach where several CPUs can share a larger than normal (*e.g.* 256 GB) memory. Compilers can sometimes help make your code make use of “parallelism”, *i.e.* having the computational time decrease by using more than one core or CPU. Right now, typical PCs can be considered shared memory (multi-core, *i.e.* CPU) machines.

Note how hardware and software are intertwined.

3.3.2 Software - Computer Languages

High level languages

- run interpreted
 - Examples: Matlab - numerical computations
 - Octave (a free Matlab clone)
 - Mathematica for symbolic math
 - Python for programming and scripting

Pros:

- rapid prototyping, convenient abstractions
- convenient debugging
- easy access to visualization (key for validation)

Cons:

- interpreted at runtime, can be slow
- may require paying license fees

Low level languages

→ compile before run

Examples: “serial”: C, Fortran 77, Fortran 95
 object-oriented: C++, Java, Python

Pros:

- freely available compilers and development tools
- fast, particularly C, and Fortran
- numerous libraries and code fragments available

Cons:

- Need to compile
- “As is”, no standard interface to plotting
- More hands-on & detail-oriented work required, *e.g.* memory allocation

Lowest level languages

Assembler code:

This is what the CPU actually understands and consists of basic operations, *e.g.* “place number on stack, multiply with second number”. A “compiler”’s job is to translate low level to lowest level language, and do this as efficiently as possible. Note that compiler “optimization” can improve run times by factors of 10-100, and care should be taken when writing low level code to help the compiler.

Likely, you will never see assembler code, but you might be able to benefit from the work others have put into this (see hardware optimization below).

How to choose a computer language?

The best choice of language, hardware, and method will always depend on the problem at hand. For simple analysis, Matlab or the free python language plus extensions may be all you need. If you need more highly optimized (*e.g.* faster, 3-D, parallel) performance, F95 and C are good choices. As with all other crafts, experience will bring you closer to perfection, but keep in mind that paying attention to detail may save you a lot of time in the end!

3.3.3 Elements of a computer program

Here's a non-sensical program written in the Matlab language to illustrate a few concepts.

```
% This is the main program. Notice the '%' symbol - it means this line is
% a comment and will be ignored at run time.
i = 0;    % assign integer variable for loop
n = 100;  % some number of elements
x = zeros(n,1); % allocate and initialize a vector x[] with n elements
y = 1;
for i = 1:n    % loop from i = 1, 2, ..., n
    x(i) = y^2;    % assign some value
    y = y+2;      % increment variable
end           % close loop
% notice the statements inside the loop are indented.
i = 1;
while (i <= n)    % different loop construct
    x(i) = mysin(x(i));    % function call
    i = i+1;
    printf("%g\n", x(i));    % output statement
end

% This is the subroutine or function 'mysin'
function result = mysin(xloc)
    result = sin(xloc);
% Note that this subroutine will not know the main programs
% variables, they are "local".
```

3.3.4 Guiding philosophy in writing a computer program

1. *Modularize* and test for robustness.

- Break the task down into small into small pieces that can be reused within the same program or in another program

- Test each part well before using it in a larger project to make code more robust.

2. Strive for *portability*

Don't use special tricks/packages that might not be available on other platforms.

3. *Comment*

- Add explanatory notes for each major step, strive for a fraction of comments to code $\geq 30\%$

This will help reusability, should you or someone else want to modify the code later.

4. Use “*structures*”, avoid globals

- If variables are needed in several subroutines, do not use “global” declaration, but pass a structure that contains a set of variables.

5. *Avoid unnecessary computations*

See below for common speed up tricks.

6. *Visualize* you intermediate results often (But don't print it all out in color!)

Bugs in the code can often be seen easily when output is analyzed graphically, and may show up as, *e.g.*

- lines being wiggly when they should be smooth
- solutions being skewed when they should be symmetrical
- etc.

Object oriented programming forces you to follow rules 1 & 4 (not so much 2). Editors and advanced development environments (such as the Matlab DE) help with 3 & 6.

3.3.5 Guidelines for writing efficient code

1. *Avoid reading and writing* intermediate steps to “*file*”, *i.e.* on the hard drive (Input/Output or IO) if at all possible.
2. Use nested loops that are sorted by the fastest/major index (depends on language). *e.g.* in Matlab


```
for i = 1:n      % increment i across all rows
    for j = 1:m  % row i computations across all columns j first
        x(i,j) = x(i,j) + 5;
    end
end
end
```

and not the other way around, or *vectorize*

```
x = x + 5; % x here can be a matrix or a vector
```

3. *Avoid if statements* as much as possible.

```
if(optional == 1) % evaluating this expression will take time
    % do this
else
    % do that
end
```

if optional is usually zero, comment it out using preprocessor directives.

4. *Precompute* common factor to avoid redundant computations.
For example, instead of

```
for i = 1:n
    x(i) = x(i)/180*pi;
end
```

It is better to do

```
fac = pi/180;
for i = 1:n
    x(i) = x(i)*fac;
end
```

because it entails one less division per step.

5. *Share the code!*

The more eyes, the less bugs, and the better the performance.

6. Use *hardware optimized packages* for standard tasks, *e.g.*

- LAPACK for linear algebra
This package is available highly optimized for several architectures.
- FFTW for FFT,
an automatically adapting package.

Different hardware makes certain chunks of memory sized (“cache”) operations highly efficient (see, *e.g.* [Dabrowski et al., 2008](#), as used later in class).

3.4 Introduction to MATLAB

Reading

- *Spencer and Ware (2008)*, secs. 1-7, 9-9.3, 12-12.4.
- For reference: Matlab online help desk

3.4.1 Introduction

Matlab is a commercial software that provides a computing environment that allows for sophisticated ways of developing and debugging computer code, executing programs, and visualizing the output. Matlab is also a computer language (sort of a mix between C and Fortran) and this exercise for you to work through is mainly concerned with some of the language aspects that we will use extensively throughout the course. Please read through the more comprehensive and verbose Matlab Intro and familiarize yourself with Matlab.

All of our Windows and Linux machines have Matlab installed and after starting up the program, you will be presented with an interactive window where you can type in commands as we indicate below. Please also familiarize yourself with the other components of the development environment, such as the built-in editor for Matlab programs, which are called “m-files”, so that you can be more efficient in writing and debugging codes. There are numerous Matlab-provided help resources accessible through the environment, including video tutorials, access to the help pages, along with extensive documentation on the web.

Also note that there is a free clone of Matlab called octave. Given that Matlab often uses freely available computational routines underneath the hood, it was fairly easy to reproduce the computational basics of Matlab. However, the Matlab people also added a bunch of proprietary visualization tools which are not available in octave. Another alternative is to use the freely available Python language and its Matplotlib package, but we will not have time to explore such intriguing options in class.

MATLAB is entirely vector or linear algebra based. It is therefore useful to briefly review some basic linear algebra.

3.4.2 Useful linear algebra (reprise)

Let's define a vector \mathbf{b} as:

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

and a 3 by 2 matrix D as:

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

The transpose (denoted with T) is given by:

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

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

Matrix-vector multiplication:

$$\mathbf{D}^T \mathbf{b}^T = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 3 & 6 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 17 \end{pmatrix} = \begin{pmatrix} 130 \\ 142 \end{pmatrix}$$

Vector-vector multiplication (dot product):

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

Matrix-matrix multiplication:

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

If you don't know what's going on here, and what the rules for such multiplications are, please consult sec. 2.1.

In numerical modeling, or in geophysical inverse problems, we frequently end up with linear system of equations of the form:

$$\mathbf{A} \mathbf{c} = \mathbf{Rhs}$$

where \mathbf{A} is a $n \times m$ matrix and \mathbf{Rhs} is a $n \times 1$ vector whose coefficients are both known, and \mathbf{c} is a $m \times 1$ vector with unknown coefficients. If we take $\mathbf{A} = \mathbf{D}$ and $\mathbf{Rhs} = \mathbf{b}^T$, \mathbf{c} is (check!):

$$\mathbf{c} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

3.4.3 Exploring MATLAB

Getting started

To start the program on the Linux machines type `Matlab` at the UNIX prompt, or click on the relevant Windows item. The MATLAB environment, including the command window, starts. (If

you want to avoid bringing up the whole environment on Linux, use “matlab -nojvm” for no-java-virtual-machine.)

1. Type 2+3. You’ll get the answer. Type $2 + 3*9 + 5^2$.

2. Type the following commands and note how Matlab deals with vectors

```
>>x=3
>>x=3;
>>x
>>y=x^2
>>x = [2, 5.6]
>>y=2 * x;
>>y=x^2;
>>y=x.^2
>>y = [3, 4]
>>x * y
>>x * y'
>>x .* y
>>pi
>>a=x*pi
```

3. Type demo and explore some examples. Also note the introductory tutorial videos you might want to watch later.

4. Type help. You see a list of all help functions. Type help log10 to get information about the log10 command. Type help logTAB where logTAB means typing log and then pressing the TAB key without adding a white space. Notice the command completion selection within the Matlab shell. Note also that you can use the Up and Down arrows to retrieve previous commands and navigate through your command history, and pUP will bring up the last command line that started with a p. MATLAB also offers a graphical user interface (GUI) to explore all of its features: click help in the menu bar, then product help. Moreover, the function browser offers you a graphical way to find the suitable function for what you are trying to accomplish. The function browser can be found under help, functions browser, or can be brought up using a keyboard shortcut: Shift+F1.

Vectors/arrays and plotting

5. Create an array of x-coordinates

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

6. 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 is it?

```
>>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 short radius 1 and long radius 2. Also change the color 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 product of x and y. Typing

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

performs an element by element product of the two vectors (note the dot)

```
>>x'
```

returns the transpose

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

the “dot” or scalar product of two matrices

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

the matrix product - returns a matrix.

Some commands (try them):

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

10. Create 2D matrices.

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

```
>>[x2d,y2d] = meshgrid(0:.1:2*pi,1:.1:2*pi)
```

You can get the size of an array with:

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

11. Plotting of the function $\sin(x2d \cdot y2d)$.

```
>>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;
```

Other useful commands:

clf: clear current active figure

close all: close all figure windows

Matlab scripting

By now you must be tired from typing all those commands all the time. Luckily there is a Matlab script language which basically allows you to type the commands in a text editor. Matlab scripts are text files that end with the suffix ".m".

12. Use the built in editor (or another text editor e.g. Emacs) 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 put lots of comments inside a Matlab script. 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. Alternatively, within the Matlab editor, you can press F5 to run. Also note that there are various debugging features in the editor that are very helpful, such as real-time syntax checking and addition of breakpoints.

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 x-coordinate array: $dx=0.01$; $y=\cos([0:dx:10])$. Compute the integral of $y=\cos(x)$ on the x-interval $0 < x < 10$. 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, ask `help cumsum`

IF command

23. Ask `help if`. Find maxima of the above array `thickness`, and compare it with the in built function `max(thickness)`

FIND command

24. Ask `help find`. Find which bed has the maximum thickness:

`find(thickness==max(thickness))`. Is there a way to do this without invoking the `find` command? find out by typing `help max`

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 $Ac=Rhs$ for c , you can use the backslash operator:
`c = A\Rhs`

Functions

Matlab allows you to declare functions that return a value and use m-files to store those functions. If you save

```
function xs = mysqr(x)
xs = x.^2;
```

as a `mysqr.m` in your working directory, you can then use your function just like a regular Matlab command.

```
y=[2,3,4]
mysqr(y);
```

Variables and structures

Matlab stores all regular variables as arrays of size 1×1 which are by default of type “double”. To write more efficient programs, you might at times consider declaring integers as actual integers.

More importantly, Matlab affords you with the possibility to collect variables that logically belong together into a “structure”. This variable will hold as many sub-variable as you want which are each addressed with a “.”. For example, if dealing with earthquakes, you might want to use a structure like

```
quake.lon = 100.1;quake.lat = 120.1;quake.depth = 15;
```

The benefit of this is that you can now, for example, pass “quake” to functions and the function will locally know that quake actually has the components lon, lat, and depth which can be addressed within the subroutine.

26. Exercise: Write and test function that has two inputs, x and a polynomial. The polynomial structure should have two entries, the order of the polynomial expansion n and a vector a with n entries that hold the coefficients such that the function returns

$$y = \sum_{i=1}^n a_i x^{n-1} \quad (3.1)$$

Chapter 4

Ordinary differential equations

4.1 Introduction

Reading: *Press et al.* (1993), Chap. 17; *Spiegelman* (2004), Chap. 4; *Spencer and Ware* (2008), sec. 16.

ODE An equation that involves the derivative of the function we want to solve for, and that has only one independent variable (else it's a PDE).

For example:

$$\frac{\partial y}{\partial x} = f(x), \quad \text{which can be solved by integration} \quad (4.1)$$

$$y = \int f(x) dx + C \quad (4.2)$$

where C needs to be determined by additional information, such as a *boundary condition* on y . If $f(x, y)$ depends non-linearly on y , the ODE will normally have to be solved numerically.

The order of an ODE is determined by the largest number of derivatives involved, *e.g.*

$$\frac{\partial^2 y}{\partial x^2} + q(x) \frac{\partial y}{\partial x} = r(x) \quad (4.3)$$

is “*second order*”. However, we can always reduce ODEs to sets of first order equations. For eq. (4.3), define

$$\frac{\partial y}{\partial x} = z(x), \quad \text{then} \quad (4.4)$$

$$\frac{\partial z}{\partial x} = r(x) - q(x)z(x). \quad (4.5)$$

Or, in general

$$\frac{\partial y_i(x)}{\partial x} = f_i(x, y, \dots, y_N) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial x} = \mathbf{f}(x, \mathbf{y}) \quad (4.6)$$

is a system for N coupled ODEs, all dependent on the independent variable x , which is typically time. \mathbf{y} is the solution vector we want to solve for. The actual solution of ODEs will depend on the types of boundary conditions on \mathbf{y} and the initial conditions.

We can distinguish between initial value and two point boundary values problems.

4.1.1 Initial Value Problems

We focus here on initial value problems, where \mathbf{y} is known for some $x = x_0$, and the system evolves from there to some x_f (final time).

Examples are

- *spring slider systems*

$$\frac{\partial \tau}{\partial t} = k(v - v_0) \quad \tau = k \cdot x \quad (\text{Hooke's law})$$

$$\tau = f(v, \theta_1, \theta_2, \dots) \quad (\text{friction law})$$

$$\frac{\partial \theta_i}{\partial t} = f(v, \theta_i)$$

- *geochemical box models*

$$\frac{\partial y}{\partial t} = f(y); \quad (\text{concentration and fluxes})$$

- *low order spectral models, e.g. for convection*

$$\mathbf{y}(\mathbf{x}, t) = \sum_{n=1}^N y_n(t) f_n(\mathbf{x})$$

(harmonic basis functions for spatial solution (problem set will deal with those))

- *parametrized convection models*

$$\dot{Q} = c_p M \frac{\partial T}{\partial t} = H(t) - Q_c(t) = H(t) - f(T, t) \quad (4.7)$$

- *particle tracking*

$$\frac{\partial c}{\partial t} = f(\mathbf{x}, t) \quad \text{for each particle} \quad (4.8)$$

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{v} \quad (4.9)$$

which is equivalent to the advection equation

$$\frac{Dc}{Dt} = \frac{\partial c}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{c} = f \quad (4.10)$$

4.1.2 Two-point Boundary Value Problems

Here, \mathbf{y} is given at x_0 and x_f . We will not deal with those, see [Press et al. \(1993\)](#), chap. 18. They generally involve iteration to find the right solution based on the initial value problems such as the “shooting method”.

4.2 Solution of initial value problem

Let's consider the solution of

$$\frac{\partial y}{\partial t} = f(t, y(t)) \quad (4.11)$$

from $t = t_0$ to $t = t_f$ with $y_0 = y(t = t_0)$

$$y(t) = y_0 + \int_{t_0}^{t_f} f(t, y(t)) dt \quad (4.12)$$

We can break down the integral into *step sizes* h from t_i to $t_i + h$ with $n = \frac{t_f - t_0}{h}$ partial integrals such that we only need to solve

$$I = \int_{t_i}^{t_i+h} f(t, y(t)) dt \quad (4.13)$$

as cheaply as possible numerically. The simplest approximation is

$$\begin{aligned} I &= f(t_i, y(t_i)) h \quad \text{such that} & (\text{from (4.12)}) \\ y(t_i + h) &= y(t_i) + h \cdot f(t_i, y(t_i)) & (4.14) \end{aligned}$$

becomes the rule to advance y from t_i to $t_i + h$. This is the *Euler* method, and a really bad idea. Consider the graphical representation in [Figure 4.1](#), which shows that (4.14) is just a simple extrapolation of y based on the slope at t_i , which is given by equation (4.11). If y has some curvature to it, the Euler scheme will lead to large errors!

We can Taylor expand y around t_0 to get

$$y(t) \approx y(t_0) + (t - t_0) \frac{\partial y(t_0)}{\partial t} + \frac{(t - t_0)^2}{2!} \frac{\partial^2 y(t_0)}{\partial t^2} + \frac{(t - t_0)^3}{3!} y''' + \dots \quad (4.15)$$

to gain some mathematical insight into the accuracy of the Euler scheme. For our problem, (4.15) becomes

$$y(t_i + h) \approx y(t_i) + h \cdot f(y(t_0), t_0) + \frac{h^2}{2} \frac{\partial^2 y}{\partial t^2} + \dots \quad (4.16)$$

Notice that the error of the Euler scheme goes as $O(\text{“order of”})(h^2)$, and the scheme itself is only accurate to *first order*. This means that tiny timesteps would have to be taken for a good solution. There are several improvements to the Euler method.

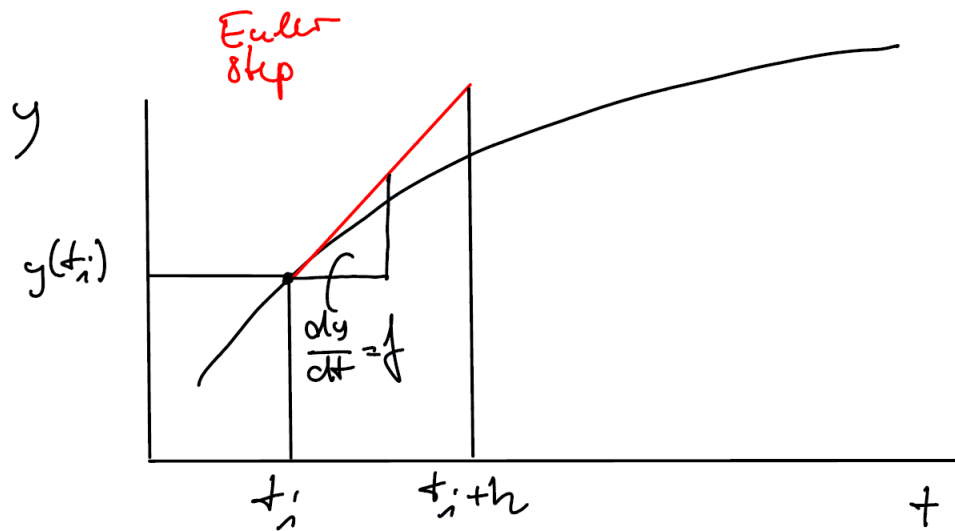


Figure 4.1: Example Euler method

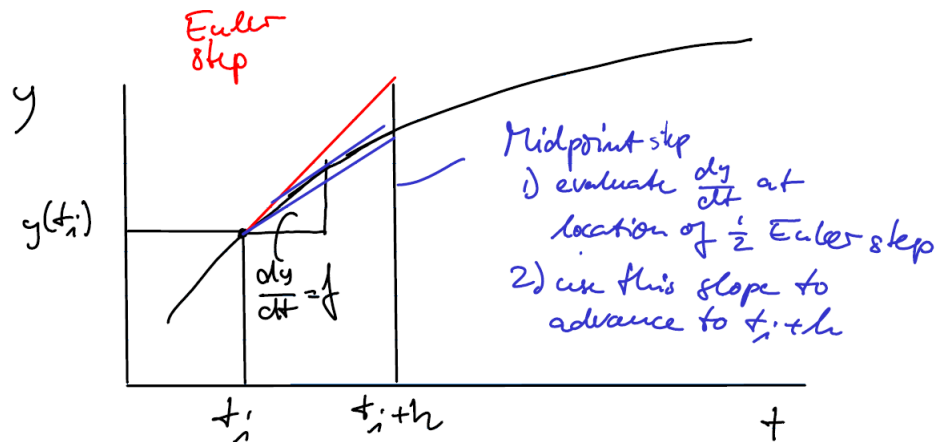


Figure 4.2: Example midpoint method

1. **The midpoint method** of Figure 4.2 evaluates the derivative of y w.r.t. to t first at half the Euler step

$$\frac{\partial y}{\partial t}(t_i + h/2, y(t_i) + \frac{\partial y}{\partial t}(t_i) \frac{h}{2}) \quad (4.17)$$

and then advances y by that slope

$$y(t_i + h) = y(t_i) + h \frac{\partial y}{\partial t}(t_i + h/2, y(t_i) + \frac{\partial y}{\partial t}(t_i) \frac{h}{2}) \quad (4.18)$$

written in terms of f

$$y(t_i + h) = y(t_i) + h f(t_i + \frac{h}{2}, y(t_i) + f(t_i, y_i) \frac{h}{2}) \quad (4.19)$$

or letting $y_{i+1} = y(t_i + h)$, $y_i = y(t_i)$, we can write

$$k_1 = h f(t_i, y_i) \quad (4.20)$$

$$k_2 = h f(t_i + \frac{h}{2}, y_i + \frac{1}{2}k_1) \quad (4.21)$$

$$y_{i+1} = y_i + k_2 + O(h^3) \quad (4.22)$$

and this method is *second order accurate*. Note that higher accuracy has come at a cost, f now needs to be evaluated twice and once at a y value different from y_i , and there are overall more operations per time step. However, since the error is now $O(h^3)$, we can take larger time steps.

There are several avenues to refine the midpoint method further, but in general the

2. **4th order Runge-Kutta** works well. The rules are

$$k_1 = h f(t_i, y_i) \quad (4.23)$$

$$k_2 = h f(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}) \quad (4.24)$$

$$k_3 = h f(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}) \quad (4.25)$$

$$k_4 = h f(t_i + h, y_i + k_3) \quad (4.26)$$

$$y_{i+1} = y_i + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5) \quad (4.27)$$

The next improvement is to adapt the stepsize h during forward integration, and especially tricky functions f require special methods. [Press et al. \(1993\)](#) discusses these, and some are implemented in Matlab.

Read the Matlab help material for how to use the built-in ODE solvers. This is discussed in [Spencer and Ware \(2008\)](#), sec. 16.5. Typically, you want to try `ode45`, and if that fails, `ode113`, or `ode15s`.

4.3 Exercise: Solving Ordinary Differential Equations – Lorenz equations

Reading

- *Spiegelman* (2004), chap. 4
- *Press et al.* (1993), chap. 17 (16 in 2nd ed.)
- *Spencer and Ware* (2008), sec. 16

In class, we discussed the 4th order Runge Kutta method as a simple method to solve initial value problems where the task is to forward integrate a vector $\mathbf{y}(t)$ from an initial condition $\mathbf{y}_0(t = t_0)$ to some time t_f when the time derivatives of \mathbf{y} are given by

$$\frac{d\mathbf{y}}{dt}(t) = \mathbf{f}(t, \mathbf{y}, \mathbf{C}) \quad (4.28)$$

where \mathbf{C} is a vector that holds all parameters that function \mathbf{f} might need. Numerically, this is done by successively computing \mathbf{y}_{n+1} for time $t + h$ from the last known solution for \mathbf{y}_n at time t with time step h

$$\mathbf{y}_{n+1} \approx \mathbf{y}_n + h\mathbf{y}'(h, t, \mathbf{y}, \mathbf{C}) \quad (4.29)$$

where \mathbf{y}' denotes the approximate time-derivatives for \mathbf{y} .

As an interesting example of a three-dimensional ($\mathbf{y} = \{y_1, y_2, y_3\}$) ODE system are the *Lorenz* (1963) equations. These equations are a simplified description of thermal convection in the atmosphere and an example of a low order, spectral numerical solution.

4.3.1 Digression for background – not essential to solving this problem set

For an incompressible fluid, conservation of mass, energy, and momentum for the convection problem can be written as

$$\nabla \cdot \mathbf{v} = 0 \quad (4.30)$$

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \kappa \nabla^2 T \quad (4.31)$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{v} = \nu \nabla^2 \mathbf{v} - \frac{1}{\rho_0} \nabla P + \frac{\rho}{\rho_0} \mathbf{g}. \quad (4.32)$$

Here, $\nu = \eta/\rho_0$ is dynamic viscosity, \mathbf{v} velocity, T temperature, κ thermal diffusivity, \mathbf{g} gravitational acceleration, ρ density, and P pressure. In the Boussinesq approximation, $\rho(T) = \rho_0(1 - \alpha(T - T_0))$, where α is thermal expansivity and ρ_0 and T_0 reference density and temperature, respectively.

If we assume two-dimensionality (2-D) in x and z direction, and a bottom-heated box of fluid, the box height d provides a typical length scale. If \mathbf{g} only acts in z direction and all quantities are non-dimensionalized by d , the diffusion time, d^2/κ , and the temperature contrast between top and bottom ΔT , we can write

$$\frac{\partial T'}{\partial t'} + \mathbf{v}' \cdot \nabla T' = \nabla^2 T' \quad (4.33)$$

$$\frac{1}{Pr} \left(\frac{\partial \omega}{\partial t'} + \mathbf{v}' \cdot \nabla \omega \right) = \nabla^2 \omega - Ra \frac{\partial T'}{\partial x'}, \quad (4.34)$$

where the primed quantities are non-dimensionalized (more on this later). Eq. (4.34) is eq. (4.32) rewritten in terms of vorticity ω , such that $\nabla^2 \psi = -\omega$ where ψ is the stream-function, which relates to velocity as

$$\mathbf{v} = \nabla \times \psi \mathbf{k} = \{\partial \psi / \partial z, -\partial \psi / \partial x\} \quad (4.35)$$

and enforces incompressibility (mass conservation). The important part here are the two new non-dimensional quantities that arise, the Prandtl number

$$Pr = \frac{\nu}{\kappa} = \frac{\eta}{\rho \kappa} \quad (4.36)$$

which measures viscous to thermal diffusion. Pr is huge for the mantle, which is why mantle convection models drop the left part of eq. (4.34). The other quantity is the Rayleigh number

$$Ra = \frac{\rho_0 \alpha g \Delta T d^3}{\eta \kappa} \quad (4.37)$$

and measures the vigor of convection and is the most important parameter for thermal convection.

[Lorenz \(1963\)](#) used a very low order spectral expansion to solve the convection equations. He assumed that

$$\psi \approx W(t) \sin(\pi a x) \sin(\pi z) \quad (4.38)$$

$$T \approx (1 - z) + T_1(t) \cos(\pi a x) \sin(\pi z) + T_2(t) \sin(2\pi z) \quad (4.39)$$

for convective cells with wavelength $2/a$. This is an example of a spectral method where spatial variations in properties such as T are dealt with in the frequency domain, here with one harmonic. Such an analysis is also common when examining barely super-critical convective instabilities. Digression end.

The resulting equations for the time dependent parameters of the approximate convection equations are

$$\begin{aligned} \frac{dW}{dt} &= Pr(T_1 - W) \\ \frac{dT_1}{dt} &= -WT_2 + rW - T_1 \\ \frac{dT_2}{dt} &= WT_1 - bT_2 \end{aligned} \quad (4.40)$$

where $b = 4/(1 + a^2)$, $r = Ra/Ra_c$ with the critical Rayleigh number Ra_c .

4.3.2 Problems

1. Identify W , T_1 , and T_2 as y_1, y_2, y_3 and write up a Matlab code for a 4th order Runge Kutta scheme to solve for the time-evolution of \mathbf{y} using eq. (4.40) for derivatives.

Hint: You can code this any way you want, but consider the following:

- You will want to separate a “driver” that deals with initial conditions, controlling the total time steps, plotting, etc., and an actual routine that computes the Runge Kutta step following the formula we discussed in class. Those should be separate m-files, or at least separate files.

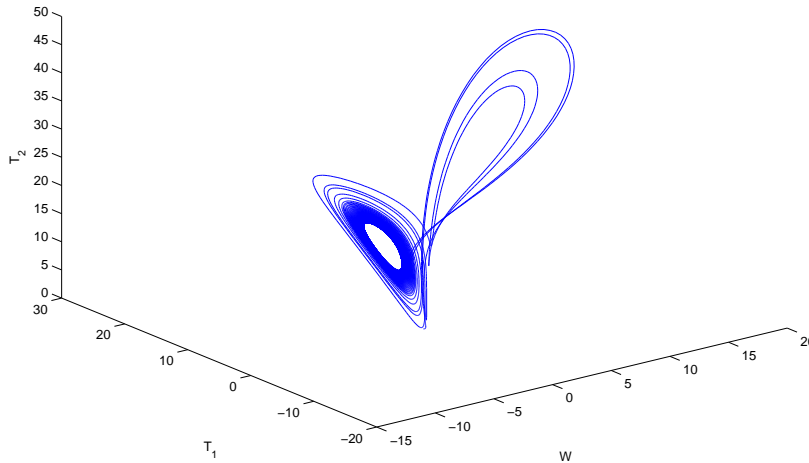


Figure 4.3: Solution to one of the problem set questions visualizing the behavior of the Lorenz equations.

- You will want to make the Runge Kutta stepper independent of the actual function that is needed to compute dy/dt so that you can reuse it for other problems later. This can be done in Matlab by defining a function `myfunc` that computes the derivatives, and then passing the function name `myfunc` as an argument to the Runge Kutta time stepper. Within the time stepper, the function then has to be referred to as `@func`.
 - If you need some inspiration on how to do this, download the m-file fragments I provide for this week's problem set.
2. Use initial condition $\mathbf{y}_0 = \{0, 0.5, 0.5\}$, parameters $b = 8/3$, $Pr = 10$ and solve for time evolution for all three variables from $t = 0$ to $t = 50$, using a time step $h = 0.005$. Use $r = 2$ and plot T_1 and T_2 against time. Comment on the temporal character of the solution, what does it correspond to physically?
 3. Change r to 10, and then 24. Plot T_1 and T_2 against time, and also plot the “phase space trajectory” of the system by plotting \mathbf{y} in W , T_1 , and T_2 space using Matlab `plot3`. Comment on how these solutions differ.
 4. Increase r to 25 and plot both time behavior of T and the phase space trajectory. What happened? Compare the $r = 25$ solution with the $r = 24$ solution from the last question. Do you think $r = 24$ will remain steady for all times? Why? Why not?

5. Use $r = 30$ and show on one plot how T_1 evolves with time for two different initial conditions, the \mathbf{y}_0 from before, $\{0, 0.5, 0.5\}$, and a second initial condition $\{0, 0.5, 0.50001\}$. Comment.
6. Compare your solution with $h = 0.005$ for T_1 and an initial condition of your choice in the $r = 30$ regime with the Matlab-internal ODE solver you deem most appropriate. Plot the absolute difference of the solutions against time. Comment.

Please hand in all your results before the next lecture in hard copy form. Print out all code you have written and also key figures, like the ones I've asked for. Label all plots with axes and title descriptions, identify which curves show what. Explain all steps, and comment your code. When discussing results, 2-3 sentences should normally be sufficient. Handwritten notes, if legible, plus print-outs for code and graphics are sufficient.

4.3.3 Additional experiments

If you are curious about additional Earth Science applications of ODEs, the literature of geochemical modeling is full of it because it is often easiest, or most appropriate, to consider fluxes between reservoirs of different chemical species with averages properties, so-called “box models” (e.g. [Albarede, 1995](#)). A classic example from magneto-hydrodynamics is the 3-D Rikitake dynamo model that consists of two conducting, coupled rotating disks in a background magnetic field. The Rikitake dynamo shows behavior similar to the Lorenz system and serves as an analog for magnetic field reversal.

Examples from our own research where we have used simple ODE solutions, include some work on parameterized convection ([Loyd et al., 2007](#)), a method that goes back at least to [Schubert et al. \(1980\)](#), see [Korenaga \(2008\)](#) for a review. In this case, the box is the mantle, and the total heat content of the mantle, as parameterized by the mean temperature, is the property one solves for.

Another example, from the brittle regime, are spring sliders. Instead of dealing with full fault dynamics, one may consider a block that has a friction law apply at its base and pulled by a string. Depending on the assumptions on the friction law, such a system exhibits stick-slip behavior akin to the earthquake cycle. For rate-and-state (i.e. velocity and heal-time) dependent friction (e.g. [Marone, 1998](#)) with two “state” variables, spring-slider models exhibit interesting, chaotic behavior ([Becker, 2000](#)). This behavior includes the characteristic period-doubling route toward chaos as a function of a material parameter (spring stiffness) ([Feigenbaum, 1978](#)).

Chapter 5

Finite differences

5.1 Introduction to the finite difference method

5.1.1 Finite differences and Taylor series expansions

We now turn to the solution of partial differential equations (PDEs), and the first method we discuss are finite differences (FD). The solution of PDEs by means of FD is based on approximating derivatives of continuous functions by a discretized version of the derivative based on a discretized function. FD approximations can be derived through the use of Taylor series expansions. Suppose we have a function $f(x)$, which is continuous and differentiable over the range of interest. Let's also assume we know the value $f(x_0)$ and all the derivatives at $x = x_0$. The forward Taylor-series expansion for $f(x_0 + h)$, away by an amount of h from a point x_0 gives

$$f(x_0 + h) = f(x_0) + \frac{\partial f(x_0)}{\partial x} h + \frac{\partial^2 f(x_0)}{\partial x^2} \frac{h^2}{2!} + \frac{\partial^3 f(x_0)}{\partial x^3} \frac{h^3}{3!} + \dots + \frac{\partial^n f(x_0)}{\partial x^n} \frac{h^n}{n!} + O(h^{n+1}) \quad (5.1)$$

We can express the first derivative of f by rearranging eq. (5.1)

$$\frac{\partial f(x_0)}{\partial x} = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{\partial^2 f(x_0)}{\partial x^2} \frac{h}{2!} - \frac{\partial^3 f(x_0)}{\partial x^3} \frac{(\Delta x)^2}{3!} \dots \quad (5.2)$$

If we now only compute the first term of this equation as an approximation, we can write a discretized version

$$\frac{\partial f(x_i)}{\partial x} = \frac{f_{i+1} - f_i}{h} + O(h) \quad (5.3)$$

where functions $f_i = f(x_i)$ are evaluated at discretely spaced x_i with $x_{i+1} = x_i + h$, where the node spacing, or *resolution* h is assumed constant. Here, $O(h)$ indicates that the full solution would require terms of order h , $(h)^2$, and so on. O is called the truncation error, which means that if the distance h is made smaller and smaller, the (numerical approximation) error decreases $\propto h$. The *forward FD derivative* as expressed by eq. (5.3) is therefore called first order accurate, and this means that very small h is required for an accurate solution.

We can also expand the Taylor series backward

$$f(x_0 - h) = f(x_0) - \frac{\partial f(x_0)}{\partial x} h + \frac{\partial^2 f(x_0)}{\partial x^2} \frac{h^2}{2!} - \frac{\partial^3 f(x_0)}{\partial x^3} \frac{h^3}{3!} + \dots \quad (5.4)$$

In this case, the first, *backward difference* can be obtained by

$$\frac{\partial f(x_i)}{\partial x} = \frac{f_i - f_{i-1}}{h} + O(h). \quad (5.5)$$

Proceeding in a similar fashion, we can derive higher order derivatives. Introducing the abbreviation

$$f' = \frac{\partial f}{\partial x} \quad f'' = \frac{\partial^2 f}{\partial x^2} \quad \dots \quad (5.6)$$

we can find, for example,

$$f_i'' = \frac{f_{i+1}' - f_i'}{h} + O(h) \quad (5.7)$$

$$= \frac{\frac{f_{i+2} - f_{i+1}}{h} - \frac{f_{i+1} - f_i}{h}}{h} + O(h) \quad (5.8)$$

$$= \frac{f_{i+2} - 2f_{i+1} + f_i}{h^2} + O(h) \quad (5.9)$$

which is the first order accurate forward difference approximation for second derivatives.

If we wish to improve on accuracy, we can proceed by taking higher order terms of the Taylor expansion and using first order accurate estimates for the derivatives. For example,

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(x) + \dots \quad (5.10)$$

$$= \frac{f(x+h) - f(x)}{h} - \frac{h}{2} \left(\frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + O(h) \right) + O(h^2) \quad \text{or (5.11)}$$

$$f_i' = \frac{-f_{i+2} + 4f_{i+1} - 3f_i}{2h} + O(h^2). \quad (5.12)$$

Alternatively, we can form the average of the first order accurate forward and backward schemes, *i.e.* adding eqs. (5.3) and (5.5) and dividing by two. The result is the *central difference* approximation of the first derivative

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2) \quad (5.13)$$

and also second order accurate. Note that eq. (5.13) involves fewer function evaluations than eq. (5.12), which is why eq. (5.13) is preferred. Note also that both equations now require knowledge of f over three lateral grid points (three point *stencil*), rather than two as was needed for first order accuracy.

By adding eqs. (5.1) and (5.4) an approximation of the second derivative is obtained

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2). \quad (5.14)$$

A different way to derive the second derivative is by computing the first derivatives at $i + 1/2$ and at $i - 1/2$ and computing the second derivative at i by using the central difference of those two first derivatives:

$$f'_{i+1/2} = \frac{f_{i+1} - f_i}{h} \quad (5.15)$$

$$f'_{i-1/2} = \frac{f_i - f_{i-1}}{h} \quad (5.16)$$

$$f''_i = \frac{f'_{i+1/2} - f'_{i-1/2}}{h} = \frac{\frac{f_{i+1} - f_i}{h} - \frac{f_i - f_{i-1}}{h}}{h} \quad (5.17)$$

Similarly, we can derive higher order derivatives, and higher order accuracy (but only if f is of polynomial form). Both require more input values, a larger stencil. A general approach to forming interpolations of f and $d^n f/dx^n$ is, *e.g.*, in [Fornberg \(1996\)](#). Note that the highest order derivative that usually occurs in geodynamics is the 4th-order derivative.

5.1.2 Finite difference approximations

The following equations are common finite difference approximations of derivatives which are here provided for reference. Central differences with second order accuracy are typically good choices and highlighted in bold face.

Left-sided first derivative, first order

$$\left. \frac{\partial u}{\partial x} \right|_{i-1/2} = \frac{u_i - u_{i-1}}{h} + O(h) \quad (5.18)$$

Right-sided first derivative, first order

$$\left. \frac{\partial u}{\partial x} \right|_{i+1/2} = \frac{u_{i+1} - u_i}{h} + O(h) \quad (5.19)$$

Central first derivative, second order

$$\left. \frac{\partial u}{\partial x} \right|_i = \frac{u_{i+1} - u_{i-1}}{2h} + O(h^2) \quad (5.20)$$

Central first derivative, fourth order

$$\left. \frac{\partial u}{\partial x} \right|_i = \frac{-u_{i+2} + 8u_{i+1} - 8u_{i-1} + u_{i-2}}{12h} + O(h^4) \quad (5.21)$$

Central second derivative, second order

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2) \quad (5.22)$$

Central second derivative, fourth order

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_i = \frac{-u_{i+2} + 16u_{i+1} - 30u_i + 16u_{i-1} - u_{i-2}}{12h^2} + O(h^4) \quad (5.23)$$

Central third derivative, second order

$$\left. \frac{\partial^3 u}{\partial x^3} \right|_i = \frac{u_{i+2} - 2u_{i+1} + 2u_{i-1} - u_{i-2}}{2h^3} + O(h^2) \quad (5.24)$$

Central third derivative, fourth order

$$\left. \frac{\partial^3 u}{\partial x^3} \right|_i = \frac{-u_{i+3} + 8u_{i+2} - 13u_{i+1} + 13u_{i-1} - 8u_{i-2} + u_{i-3}}{8h^3} + O(h^4) \quad (5.25)$$

Central fourth derivative, second order

$$\left. \frac{\partial^4 u}{\partial x^4} \right|_i = \frac{u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}}{h^4} + O(h^2) \quad (5.26)$$

Note that derivatives with of the following form

$$\frac{\partial}{\partial x} \left(k(x) \frac{\partial u}{\partial x} \right), \quad (5.27)$$

where k is a function of space, should be formed as follows

$$\left. \frac{\partial}{\partial x} \left(k(x) \frac{\partial u}{\partial x} \right) \right|_i = \frac{k_{i+1/2} \frac{u_{i+1} - u_i}{h} - k_{i-1/2} \frac{u_i - u_{i-1}}{h}}{h} + O(h^2), \quad (5.28)$$

to maintain the second order accuracy of the central difference approach for second derivatives (see above)

$$f'' = \frac{f'(x + h/2) - f'(x - h/2)}{h}. \quad (5.29)$$

If k is spatially varying, the following, common approximations are therefore inadequate to maintain second order accuracy:

$$\left. \frac{\partial}{\partial x} \left(k \frac{\partial u}{\partial x} \right) \right|_i = \frac{k_{i+1} \frac{u_{i+1} - u_i}{h} - k_i \frac{u_i - u_{i-1}}{h}}{h} \quad (5.30)$$

$$\left. \frac{\partial}{\partial x} \left(k \frac{\partial u}{\partial x} \right) \right|_i = k_i \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \quad (5.31)$$

5.1.3 Finite difference example

Finite difference methods are perhaps best understood with an example. Consider the one-dimensional, transient (*i.e.* time-dependent) heat conduction equation without internal sources

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) \quad (5.32)$$

where ρ is density, c_p heat capacity, k thermal conductivity, T temperature, x distance and t time. If the thermal conductivity, density and heat capacity are constant over the model domain, the equation can be simplified to

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \quad (5.33)$$

where

$$\kappa = \frac{k}{\rho c_p} \quad (5.34)$$

is the thermal diffusivity (a common value for rocks is $\kappa = 10^{-6} \text{ m}^2\text{s}^{-1}$). We are interested in the temperature evolution versus time $T(x, t)$ which satisfies eq. (5.33), given an initial temperature distribution (Fig. 5.1A). An example would be the intrusion of a basaltic dike in cooler country rocks. How long does it take to cool the dike to a certain temperature? What is the maximum temperature that the country rock experiences?

The first step in the finite differences method is to construct a grid with points on which we are interested in solving the equation (this is called discretization, see Fig. 5.1B).

The next step is to replace the continuous derivatives of eq. (5.33) with their finite difference approximations. The derivative of temperature versus time $\frac{\partial T}{\partial t}$ can be approximated with a forward finite difference approximation as

$$\frac{\partial T}{\partial t} \approx \frac{T_i^{n+1} - T_i^n}{t^{n+1} - t^n} = \frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{T_i^{\text{new}} - T_i^{\text{current}}}{\Delta t} \quad (5.35)$$

here n represents the temperature at the current time step whereas $n + 1$ represents the new (future) temperature. The subscript i refers to the location (Fig. 5.1B). Both n and i are integers; n varies from 1 to n_t (total number of time steps) and i varies from 1 to n_x (total number of grid points in x -direction). The spatial derivative of eq. (5.33) is replaced by a central finite difference approximation, *i.e.*,

$$\frac{\partial^2 T}{\partial x^2} = \frac{\partial}{\partial x} \left(\frac{\partial T}{\partial x} \right) \approx \frac{\frac{T_{i+1}^n - T_i^n}{h} - \frac{T_i^n - T_{i-1}^n}{h}}{h} = \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}. \quad (5.36)$$

Substituting eq. (5.36) and (5.35) into eq. (5.33) gives

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \left(\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \right) \quad (5.37)$$

The third and last step is a rearrangement of the discretized equation, so that all known quantities (*i.e.* temperature at time n) are on the right hand side and the unknown quantities on the left-hand side (properties at $n + 1$). This results in:

$$T_i^{n+1} = T_i^n + \kappa \Delta t \left(\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \right) \quad (5.38)$$

Because the temperature at the current timestep (n) is known, we can use eq. (5.38) to compute the new temperature without solving any additional equations. Such a scheme is called *explicit* and

was made possible by the choice to evaluate the temporal derivative with forward differences. We know that this numerical scheme will converge to the exact solution for small h and dt because it has been shown to be consistent - that its discretisation process can be reversed, through a Taylor series expansion, to recover the governing partial differential equation - and because it is stable for certain values of dt and h : any spontaneous perturbations in the solution (such as round-off error) will either be bounded or will decay.

The last step is to specify the initial and the boundary conditions. If for example the country rock has a temperature of 300°C and the dike a total width $W = 5$ m, with a magma temperature of 1200°C , we can write as initial conditions:

$$T(x < -W/2, x > W/2, t = 0) = 300 \quad (5.39)$$

$$T(-W/2 \leq x \leq W/2, t = 0) = 1200 \quad (5.40)$$

In addition we assume that the temperature far away from the dike center (at $|L/2|$) remains at a constant temperature. The boundary conditions are thus

$$T(x = -L/2, t) = 300 \quad (5.41)$$

$$T(x = L/2, t) = 300 \quad (5.42)$$

The attached MATLAB code shows an example in which the grid is initialized, and a time loop is performed. In the exercise, you will fill in the question marks and obtain a working code that solves eq. (5.38).

5.1.4 Exercises

1. Open MATLAB and an editor and type the Matlab script in an empty file; alternatively use the template provided on the web if you need inspiration. Save the file under the name `heat1Dexplicit.m`. If starting from the template, fill in the question marks and then run the file by typing `heat1Dexplicit` in the MATLAB command window (make sure you're in the correct directory). (Alternatively, type F5 to run from within the editor.)
2. Study the time evolution of the spatial solution using a variable y-axis that adjusts to the peak temperature, and a fixed axis with range `axis([-L/2 L/2 0 Tmagma])`. Comment on the nature of the solution. What parameter determines the relationship between two spatial solutions at different times?

Does the temperature of the country rock matter for the nature of the solution? What about if there is a background gradient in temperature such that the country rock temperature increases from 300° at $x = -L/2$ to 600° at $x = L/2$?

3. Vary the parameters (e.g. use more grid points, a larger or smaller timestep). Compare the results for small dx and dt with those for larger dx and dt . How are these solutions different? Why? Notice also that if the timestep is increased beyond a certain value, the numerical method becomes unstable and does not converge - it grows without bounds and exhibits non-physical features. Investigate which parameters affect stability, and find out what ratio

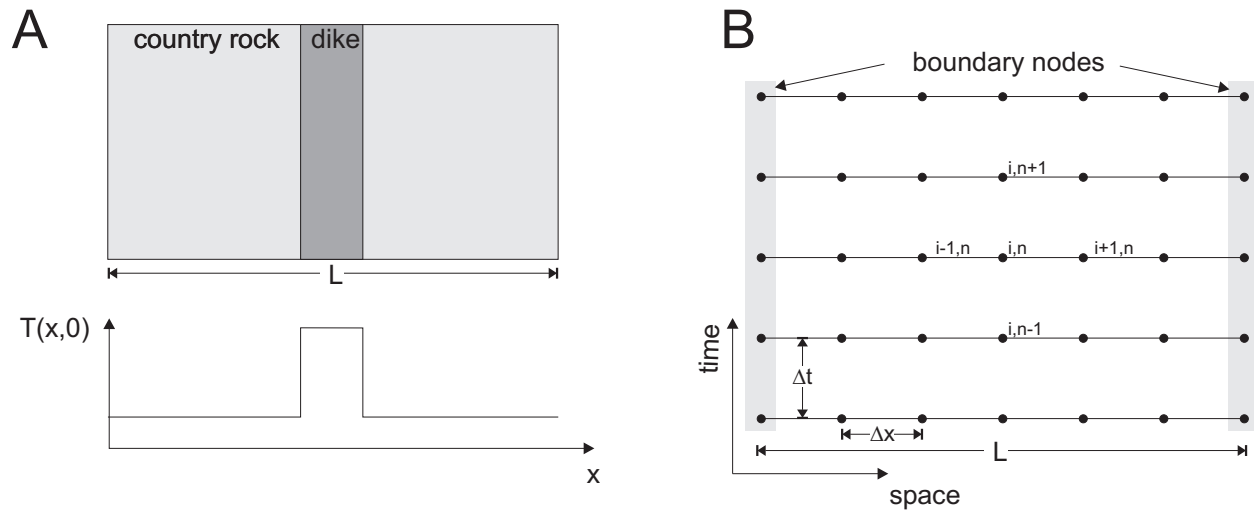


Figure 5.1: A) Setup of the model considered here. A hot basaltic dike intrudes cooler country rocks. Only variations in x -direction are considered; properties in the other directions are assumed to be constant. The initial temperature distribution $T(x,0)$ has a step-like perturbation, centered around the origin with $[-W/2; W/2]$ B) Finite difference discretization of the 1D heat equation. The finite difference method approximates the temperature at given grid points, with spacing h . The time-evolution is also computed at given times with timestep Δt .

of these parameters delimits this scheme's stability region. This is called the CFL condition, see von Neumann stability analysis in (cf. chap 5 of [Spiegelman, 2004](#)).

Limited stability and numerical aliasing/dissipation are two major drawbacks of explicit finite difference codes such as the one presented here. In the next week, we will discuss methods that do not have these limitations.

4. Record and plot the temperature evolution versus time at a distance of 5 m from the dike-country rock contact. What is the maximum temperature the country rock experiences at this location and when is it reached? Assume that the country rock was composed of shales, and that those shales were transformed to hornfels above a temperature of 600°C . What is the width of the metamorphic aureole?
5. Think about how one would write a non-dimensionalized version of the temperature solver.
6. Bonus question: Derive a finite-difference approximation for variable k (and variable h , should you so desire). Test the solution for the case of $k = 10$ inside the dike, and $k = 3$ in the country rock.

```

% Solves the 1D heat equation with an explicit finite difference scheme
clear
%Physical parameters
L      = 100;          % Length of modeled domain      [m]
Tmagma = 1200;         % Temperature of magma          [C]
Trock  = 300;          % Temperature of country rock [C]
kappa  = 1e-6;         % Thermal diffusivity of rock [m2/s]
W      = 5;           % Width of dike                  [m]
day     = 3600*24;     % # seconds per day
dt      = 1*day;       % Timestep                      [s]
% Numerical parameters
nx      = 201;         % Number of gridpoints in x-direction
nt      = 500;         % Number of timesteps to compute
dx      = L/(nx-1);    % Spacing of grid
x       = -L/2:dx:L/2;% Grid
% Setup initial temperature profile
T       = ones(size(x))*Trock;
T(find(abs(x)<=W/2)) = Tmagma;
time    = 0;
for n=1:nt % Timestep loop
    % Compute new temperature
    Tnew = zeros(1,nx);
    for i=2:nx-1
        Tnew(i) = T(i) + ?????;
    end
    % Set boundary conditions
    Tnew(1) = T(1);
    Tnew(nx) = T(nx);
    % Update temperature and time
    T = Tnew;
    time = time+dt;
    % Plot solution
    figure(1), clf
    plot(x,Tnew);
    xlabel('x [m]')
    ylabel('Temperature [^oC]')
    title(['Temperature evolution after ',num2str(time/day),' days'])
    drawnow
end

```

Figure 5.2: MATLAB script to solve eq. (5.33) (once the blanks indicated by the questions marks are filled in ...).

5.2 Implicit FD schemes and boundary conditions

Reading

- *Press et al.* (1993), sec. 19.2
- *Spiegelman* (2004), sec. 6.1-6.5

5.2.1 Variable time derivatives – explicit vs. implicit

Last week we solved the transient (time-dependent) heat equation in 1D. In the absence of heat sources, the governing equation is

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \quad (5.43)$$

if material parameters are homogeneous.

In *explicit* finite difference schemes, the temperature at time $n + 1$ depends only on the already known temperature at time n . The explicit finite difference discretization of eq. (5.43) is

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}, \quad (5.44)$$

using central differences for the spatial derivatives (subscript i indicating the x location in 1-D, superscripts indicating the time). Eq. (5.44) can be rearranged in the following manner (with all quantities at time $n + 1$ on the left and quantities at time n on the right-hand-side)

$$T_i^{n+1} = T_i^n + \kappa \Delta t \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \quad (5.45)$$

Since we know T_{i+1}^n , T_i^n and T_{i-1}^n , we can compute T_i^{n+1} . This is schematically shown on Figure 5.3a, and an algorithm based on eq. (5.45), such as the one of last week’s problem set, is called a *forward time, centered space (FTCS)* for the way it is computed.

The major advantage of explicit finite difference methods is that they are relatively simple, only one solution for T needs to be stored, and the method is computationally fast for each time step. However, the main drawback is that stable solutions are obtained only when

$$0 < \frac{2\kappa\Delta t}{h^2} \leq 1 \quad \text{or} \quad \Delta t \leq \frac{h^2}{2\kappa} \quad \text{for given } h. \quad (5.46)$$

If this condition is not satisfied, the solution becomes unstable, starts to wildly oscillate, or “blow up”. *Press et al.* (1993) discuss how to derive stability bounds for different wavelengths of perturbations, but physically, the stability condition eq. (5.46) means that the maximum time step needs to be smaller than the time it takes for an anomaly to diffuse across the grid (nodal) spacing h (cf. diffusion time in sec. 4.3). The explicit solution, eq. (5.45), is an example of a *conditionally stable* method that only leads to well behaved solutions if a criterion like eq. (5.46) is satisfied.

Note that eq. (5.46) can only hold for $\kappa\Delta t > 0$; having a negative diffusivity, or using a time-reversed $\Delta t < 0$, will invariably lead to blow up since small features will get emphasized rather than

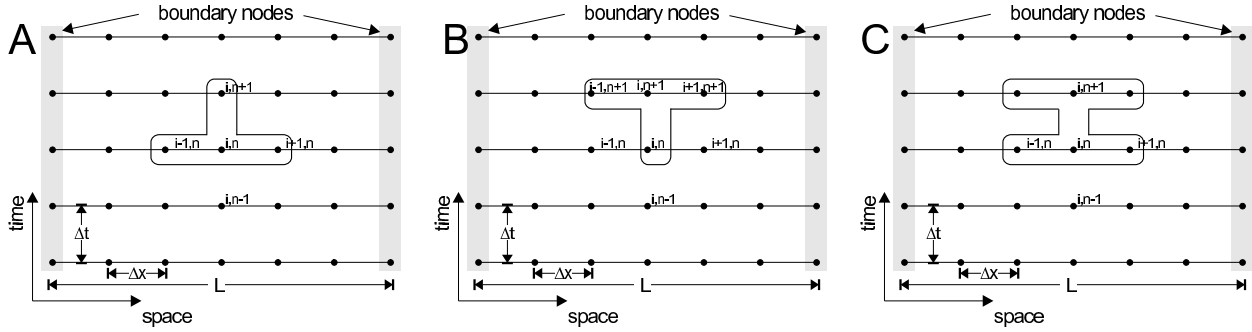


Figure 5.3: A) Explicit finite difference discretization. B) Implicit finite difference discretization. C) Crank-Nicolson finite difference discretization.

smoothed out. This is an issue if one wishes to reconstruct diffusive-advective processes (such as mantle convection), going from the present-day temperature field back in time. We will revisit an FTCS scheme similar to eq. (5.44) for advection that involves single derivatives in space later. Unlike eq. (5.44), the FTCS scheme for advection is *always* unstable. Not a good idea.

Even if the FTCS for diffusion can be made stable, the stability condition leads to numerical convenience issues. Given that we are typically interested in spatial features with wavelength, λ , within the solution that are much larger than h , $\lambda \gg h$, because we want to resolve the solution features at least with a few nodes, the explicit scheme, eq. (5.44), will require $(\lambda/h)^2 \gg 1$ steps per relevant time scale for the evolution of λ features, which is usually prohibitive.

An alternative approach is an *implicit* finite difference scheme, where the spatial derivatives $\partial^2 T / \partial x^2$ are evaluated (at least partially) at the new time step. The simplest implicit discretization of eq. (5.43) is

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2}, \quad (5.47)$$

a *fully implicit* scheme where the time derivative is taken backward (Figure 5.3b). Eq. (5.47) can be rearranged so that unknown terms are on the left and known terms are on the right

$$-sT_{i+1}^{n+1} + (1 + 2s)T_i^{n+1} - sT_{i-1}^{n+1} = T_i^n \quad (5.48)$$

where $s = \kappa \Delta t / h^2$. Note that in this case we no longer have an explicit relationship for T_{i-1}^{n+1} , T_i^{n+1} and T_{i+1}^{n+1} . Instead, we have to solve a linear system of equations, which is discussed further below.

Note: If the spatial derivative is of type

$$\frac{\partial}{\partial x} \left(k(x) \frac{\partial}{\partial x} T \right) \quad (5.49)$$

as for the case of the laterally varying conductivity in the explicit FD exercise for the heat equation, then

$$-rk_r T_{i+1}^{n+1} + [1 + r(k_l + k_r)]T_i^{n+1} - rk_l T_{i-1}^{n+1} = T_i^n \quad (5.50)$$

has to be used instead of eq. (5.48). Here, $r = \Delta t / h^2$ and k_l and k_r are the material parameters to the “left” ($x_{i-1/2}$) and “right” ($x_{i+1/2}$), respectively.

The main advantage of implicit methods is that there are no restrictions on the time step, the fully implicit scheme is *unconditionally stable*. This does not mean that it is accurate. Taking large time steps may result in an inaccurate solution for features with small spatial scales. For any application, it is therefore always a good idea to check the results by decreasing the time step until the solution does not change anymore (this is called a convergence check), and to ensure the method can deal with small and large scale features robustly at the same time.

Eq. (5.48) is also suited to understand the overall behavior of an implicit method for large time steps. If we let $\Delta t \rightarrow \infty$, and then divide eq. (5.48) by $-s$, we get

$$T_{i+1} - 2T_i + T_{i-1} = 0, \quad (5.51)$$

which is a central difference approximation of the steady state solution of eq. (5.43),

$$\frac{\partial^2 T}{\partial x^2} = 0. \quad (5.52)$$

Therefore, the fully implicit scheme will always yield the right equilibrium solution but may not capture small scale, transient features.

It turns out that the fully implicit method described by eq. (5.47) is second order accurate in space but only first order accurate in time, *i.e.* $O(h^2, \Delta t)$. It is possible to write down a scheme which is second order accurate both in time and in space (*i.e.* $O(h^2, \Delta t^2)$). One such scheme is the Crank-Nicolson scheme (see exercises, Fig. 5.3C), which is unconditionally stable. Note the analogy with the previous derivation of spatial derivatives: forward or backward differences were only first order accurate, while the central difference approach achieved second order accuracy $O(h^2)$. The Crank-Nicolson method is the time analog of central spatial differences. However, any partially implicit method is more tricky to compute as we need to infer the future solution at time $n + 1$ by inversion of a system of linear equations based on the known solution at time n . This is discussed next.

5.2.2 Solution of example problem

Boundary conditions – Neumann and Dirichlet

We solve the transient heat equation, eq. (5.43), on the domain $-L/2 \leq x \leq L/2$ subject to the following boundary conditions for fixed temperature

$$\begin{aligned} T(x = -L/2, t) &= T_{left} \\ T(x = L/2, t) &= T_{right} \end{aligned} \quad (5.53)$$

with the initial condition

$$T(x < -W/2, x > W/2, t = 0) = 300 \quad (5.54)$$

$$T(-W/2 \leq x \leq W/2, t = 0) = 1200, \quad (5.55)$$

where we have again assumed a hot dike intrusion for $-W/2 \leq x \leq W/2$.

Boundary conditions (BCs) for PDEs that specify values of the solution function (here T) to be constant, such as eq. (5.53), are called *Dirichlet boundary conditions*. We can also choose to specify the gradient of the solution function, e.g. $\partial T/\partial x$ (*Neumann boundary condition*). This gradient boundary condition corresponds to heat flux for the heat equation and we might choose, e.g., zero flux in and out of the domain (isolated BCs):

$$\begin{aligned}\frac{\partial T}{\partial x}(x = -L/2, t) &= 0 \\ \frac{\partial T}{\partial x}(x = L/2, t) &= 0.\end{aligned}\tag{5.56}$$

Solving an implicit finite difference scheme

As usual, the first step is to discretize the spatial domain with n_x finite difference points. The implicit finite difference discretization of the temperature equation within the medium where we wish to obtain the solution is eq. (5.48). Starting with fixed temperature BCs (eq. (5.53)), the boundary condition on the left boundary gives

$$T_1 = T_{left}\tag{5.57}$$

and the one on the right

$$T_{n_x} = T_{right}.\tag{5.58}$$

Eqs. (5.48), (5.57), and (5.58) can be written in matrix form as

$$\mathbf{A}\mathbf{c} = \mathbf{rhs}.\tag{5.59}$$

For a six-node grid, for example, the coefficient matrix \mathbf{A} is

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -s & (1+2s) & -s & 0 & 0 & 0 \\ 0 & -s & (1+2s) & -s & 0 & 0 \\ 0 & 0 & -s & (1+2s) & -s & 0 \\ 0 & 0 & 0 & -s & (1+2s) & -s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},\tag{5.60}$$

the unknown temperature vector \mathbf{c} is

$$\mathbf{c} = \begin{pmatrix} T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_5^{n+1} \\ T_6^{n+1} \end{pmatrix},\tag{5.61}$$

and the known right-hand-side vector **rhs** is

$$\mathbf{rhs} = \begin{pmatrix} T_{left} \\ T_2^n \\ T_3^n \\ T_4^n \\ T_5^n \\ T_{right} \end{pmatrix}. \quad (5.62)$$

Note that matrix **A** will have a unity entry on the diagonal and zero else for each node where Dirichlet (fixed temperature) boundary conditions apply; see derivation below and eqs. (5.73) and (5.74) for how to implement Neumann boundary conditions.

Matrix **A** also has an overall peculiar form because most entries off the diagonal are zero. This “sparseness” can be exploited by specialized linear algebra routines, both in terms of storage and speed. By avoiding computations involving zero entries of the matrix, much larger problems can be handled than would be possible if we were to store the full matrix. In particular, the fully implicit FD scheme leads to a “tridiagonal” system of linear equations that can be solved by LU decomposition (*e.g.* [Press et al., 1993](#), sec. 2.4).

Matlab implementation

Within Matlab, we declare matrix **A** to be sparse by initialization with the `sparse` function. This will ensure computationally fairly efficient internal treatment within Matlab. (Later, we will take recourse to a `sparse2` function that improves on built-in `sparse`.) Once the coefficient matrix **A** and the right-hand-side vector **rhs** have been constructed, MATLAB functions can be used to obtain the solution **c** and you will not have to worry about choosing a proper matrix solver for now.

First, however, we have to construct the matrices and vectors. The coefficient matrix **A** can be constructed with a simple loop:

```
A = sparse(nx,nx);
for i=2:nx-1
    A(i,i-1) = -s;
    A(i,i) = (1+2*s);
    A(i,i+1) = -s;
end
```

and the boundary conditions are set by:

```
A(1,1) = 1;
A(nx,nx) = 1;
```

(Exercise: Improve on the loop formulation for **A** assembly by using Matlab vector functionality.) Once the coefficient matrix has been constructed, its structure can be visualized with the command

```
>>spy(A)
```

(Try it, for example by putting a “break-point” into the Matlab code below after assembly.)

The right-hand-side vector **rhs** can be constructed with

```
rhs = zeros(nx,1);
rhs(2:nx-1) = Told(2:nx-1);
rhs(1) = Tleft; rhs(nx) = Tright;
```

The only thing that remains to be done is to solve the system of equations and find **c**. MATLAB does this with

```
c = A\rhs;
```

The vector **c** is now filled with new temperatures T^{n+1} , and we can go to the next time step. Note that, for constant Δt , κ , and h , the matrix **A** does not change with time. Therefore we have to form it only once in the program, which speeds up the code significantly. Only the vectors **rhs** and **c** need to be recomputed. (Note: Having a constant matrix helps a lot for large systems because operations such as $\mathbf{c} = \mathbf{A} \backslash \mathbf{rhs}$ can then be optimized further by storing **A** in a special form.)

5.2.3 Exercises

1. Save the script `heat1Dexplicit.m` from last week as `heat1Dimplicit.m`. Program the implicit finite difference scheme explained above. Compare the results with results from last week's explicit code.
2. Time-dependent, analytical solutions for the heat equation exists. For example, if the initial temperature distribution (initial condition, IC) is

$$T(x, t = 0) = T_{max} \exp \left(- \left(\frac{x}{\sigma} \right)^2 \right) \quad (5.63)$$

where T_{max} is the maximum amplitude of the temperature perturbation at $x = 0$ and σ its half-width of the perturbation (use $\sigma < L$, for example $\sigma = W$). The solution is then

$$T(x, t) = \frac{T_{max}}{\sqrt{1 + 4t\kappa/\sigma^2}} \exp \left(\frac{-x^2}{\sigma^2 + 4t\kappa} \right) \quad (5.64)$$

(for $T = 0$ BCs at infinity). Program the analytical solution and compare the analytical solution with the numerical solution with the same initial condition. Compare results of the implicit and FTCS scheme used last week to the analytical solution near the instability region of FTCS ($s = \frac{\kappa \Delta t}{\Delta x^2} < \frac{1}{2}$)

Note: Eq. (5.64) can be derived using a *similarity variable*, $\tilde{x} = x/x_c$ with $x_c \propto \sqrt{\kappa t}$. Looks familiar?

3. A steady-state temperature profile is obtained if the time derivative $\partial T / \partial t$ in the heat equation (eq. 5.43) is zero. There are two ways to do this.

- (a) Wait until the temperature does not change anymore.
- (b) Write down a finite difference discretization of $\partial^2 T / \partial x^2 = 0$ and solve it. (See the limit case consideration above.)

Employ both methods to compute steady-state temperatures for $T_{left} = 100^\circ$ and $T_{right} = 1000^\circ$. Derive the analytical solution and compare your numerical solutions' accuracies. Use the implicit method for part (a), and think about different boundary conditions, and the case with heat production.

4. Apply no flux boundary conditions at $|x| = L/2$ and solve the dike intrusion problem in a fully implicit scheme. Eqs. (5.73) and (5.74) need to replace the first and last columns of your **A** matrix.
5. Derive and program the Crank-Nicolson method (*cf.* Figure 5.3C). This “best of both worlds” method is obtained by computing the average of the fully implicit and fully explicit schemes:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{\kappa}{2} \left(\frac{(T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}) + (T_{i+1}^n - 2T_i^n + T_{i-1}^n)}{h^2} \right). \quad (5.65)$$

This scheme should generally yield the best performance for any diffusion problem, it is second order time and space accurate, because the averaging of fully explicit and fully implicit methods to obtain the time derivative corresponds to evaluating the derivative centered on $n + 1/2$. Such centered evaluation also lead to second order accuracy for the spatial derivatives. Compare the accuracy of the Crank-Nicolson scheme with that of the FTCS and fully implicit schemes for the cases explored in the two previous problems, and for ideal values of Δt and Δx and for large values of Δt that are near the instability region of FTCS.

Hint: Proceed by writing out eq. (5.65) and sorting terms into those that depend on the solution at time step $n + 1$ and those at time step n , as for eq. (5.48).

6. Bonus question: Write a code for the thermal equation with variable thermal conductivity k : $\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right)$. Assume that the grid spacing h is constant. For extra bonus, allow for variable grid spacing and variable conductivity.

5.2.4 Derivation of flux boundary conditions (fictitious boundary points)

A Neumann boundary condition can be expressed as

$$\begin{aligned} \frac{\partial T(x=0,t)}{\partial x} &= c_1 \\ \frac{\partial T(x=L,t)}{\partial x} &= c_2 \end{aligned} \quad (5.66)$$

These conditions can be implemented with a forward or a backward FD expression. However, this is not preferred since such finite difference approximations are only first order accurate in

space (see last section). A better way to incorporate a flux boundary conditions is therefore to use a central finite difference approximation, which is given (at $i = 1$) by

$$\frac{T_2 - T_0}{2h} = c_1 \quad (5.67)$$

and at $i = n_x$ by

$$\frac{T_{n_x+1} - T_{n_x-1}}{2h} = c_2. \quad (5.68)$$

The problem is, of course, that the expressions above involve points that are not part of the original numerical grid (T_0^{n+1} and $T_{n_x+1}^{n+1}$). These points are called *fictitious boundary points* (Figure 5.4). A way around this can be found by noting that the equation for the center nodes is given by

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2}. \quad (5.69)$$

Writing this expression for the first node gives

$$\frac{T_1^{n+1} - T_1^n}{\Delta t} = \kappa \frac{T_2^{n+1} - 2T_1^{n+1} + T_0^{n+1}}{h^2}. \quad (5.70)$$

An explicit expression for T_0^{n+1} is obtained from eq. (5.67)

$$T_0^{n+1} = T_2^{n+1} - 2hc_1, \quad (5.71)$$

i.e. we are simply extrapolating from T_2 to T_0 with the slope given by c_1 . Substituting into eq. (5.70) yields

$$\frac{T_1^{n+1} - T_1^n}{\Delta t} = \kappa \frac{2T_2^{n+1} - 2T_1^{n+1} - 2hc_1}{h^2}. \quad (5.72)$$

To apply this formulation in a fully explicit scheme, we can again rearrange terms to bring known quantities on the right-hand-side:

$$(1 + 2s)T_1^{n+1} - 2sT_2^{n+1} = T_1^n - 2shc_1. \quad (5.73)$$

On the other end of the domain (verify!)

$$-2sT_{n_x-1}^{n+1} + (1 + 2s)T_{n_x}^{n+1} = T_{n_x}^n + 2shc_2. \quad (5.74)$$

These equations now only involves grid points that are part of the computational grid, and eqs. (5.73) and (5.74) can be incorporated into the matrix **A** and right-hand-side **rhs** (compare with eq. (5.48)).

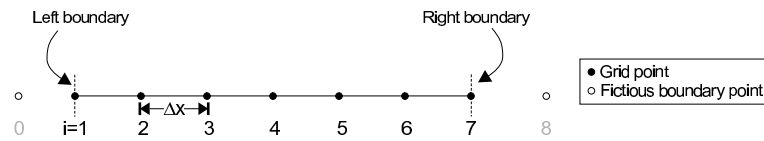


Figure 5.4: Discretization of the numerical domain with fictitious boundary points, that are employed to set flux boundary conditions.

5.3 Non-linearities with FD methods

So far, we considered linear partial differential equations, where the coefficients in the equations are either constant or only spatially variable, but are independent of time or the solution itself. If the coefficients are dependent on the solution, a nonlinear problem results.

There are a number of ways to solve such nonlinear problems. The easiest, rough and ready way, which works in many cases, is to replace the nonlinear PDE by a linear one, guess initial values for the solution and the parameters that depend on it, and then perform iterations until the solution converges (*Picard* iterations). If this method will converge will depend on the quality of the initial guess, which becomes harder when the non-linearities are strong. More sophisticated methods exist of which the most important is linearization of the nonlinear terms and solution of the (more complicated) PDE (*e.g. Newton-Rhapson* iterations). This method is more robust and converges quadratically. It is, however, more complicated to implement and will therefore not be discussed here.

5.3.1 Example

We consider a case of fluid flow in a porous media (governed by the Darcy equation) whose diffusivity κ is a function of the fluid pressure (high fluid pressure increases permeability, $p \uparrow \rightarrow \kappa \uparrow$). In a 1-D, vertical (z) column the governing equation shall be

$$\frac{\partial P}{\partial t} = \frac{\partial}{\partial z} \left(\kappa(P) \frac{\partial P}{\partial z} \right) \quad (5.75)$$

where P is the fluid pressure, and $\kappa(P)$ the hydraulic diffusivity. The equation is nonlinear because the diffusivity can be written as a function of the fluid pressure P , which is related to the effect of dilation and cracking under enhanced fluid pressure.

To solve eq. (5.75), we need a constitutive law, and we assume that the hydraulic diffusivity is given by

$$\kappa(P) = \kappa_0 + cP^m \quad (5.76)$$

where κ_0 is the background diffusivity, and c and m (semi-empirical) constants.

We will use a fully implicit scheme, so that the discretization is done in analogy (second order accurate second spatial derivative) to the implicit 1-D thermal diffusion problem:

$$\frac{P_i^{n+1} - P_i^n}{\Delta t} = \frac{\kappa_{i+1/2}^{n+1} \frac{P_{i+1}^{n+1} - P_i^{n+1}}{h} - \kappa_{i-1/2}^{n+1} \frac{P_i^{n+1} - P_{i-1}^{n+1}}{h}}{h} \quad (5.77)$$

where the material parameters are evaluated in between nodes, for example by computing an average

$$\kappa_{i\pm 1/2}^{n+1} = \frac{\kappa_i^{n+1} + \kappa_{i\pm 1}^{n+1}}{2}. \quad (5.78)$$

(If diffusivity were merely heterogeneous (such as in the previous explicit heat equation example), but not dependent on the solution itself, we could use a “staggered” grid where κ would be specified at nodes located in between the locations where P is to be computed.)

The implicit equations can again be solved in matrix form as

$$A\mathbf{p} = \mathbf{rhs} \quad (5.79)$$

for P^{n+1} . The problem, however, is that κ depends on P^{n+1} . Therefore we have to perform iterations for the true P^{n+1} and recompute A at each time step before advancing time. The general recipe is

1. Use the pressure P^n to compute the diffusivities $\kappa_{i\pm 1/2}^{n+1}$ using eqs. (5.76) and (5.78).
2. Determine the coefficients in A using the estimated diffusivities.
3. Solve the system of equations to find the new pressure P^{n+1} .
4. Use this new pressure estimate to recalculate diffusivities and the coefficients in A .
5. Return to step 2 and continue until the pressure P_{n+1} stops changing, which indicates that the solution has converged. Use as an indication of convergence the following error estimate:

$$error = \frac{\max(abs(\mathbf{p}^{it} - \mathbf{p}^{it-1}))}{\max(abs(\mathbf{p}^{it}))} \quad (5.80)$$

If convergence is reached (*e.g.* relative error $< 10^{-4}$), continue to the next time step.

Exercise

- Write a program that solves the equations described above on the domain $z \in [0; 1]$ from $t = 0$ to $t = 0.2$. Assume that we have zero flux boundary conditions (*i.e.* gradient $\partial P / \partial z = 0$ on top and bottom). Use non-dimensional parameter values $\kappa_0 = 0.05$, $c = 1$, $m = 2$ and time-step 0.005. The initial pressure is to be unity within $[0.4; 0.6]$ and zero else. At each time step, compare the nonlinear solution to the linear one, obtained by setting $c = 0$, to visualize the effect of the non-linear terms.

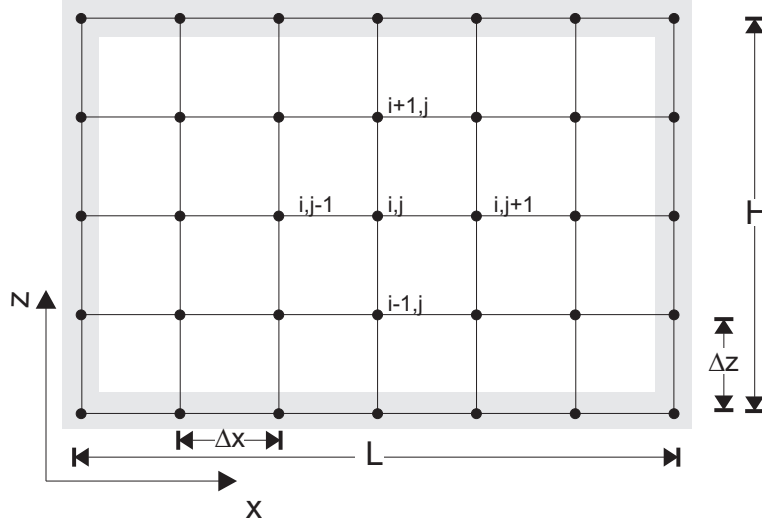


Figure 5.5: Finite difference discretization in 2D

5.4 Two-dimensional heat equation

We now revisit the transient heat equation, this time with sources/sinks, as an example for two-dimensional FD problem. In 2D ($\{x, z\}$ space), we can write

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) + Q \quad (5.81)$$

where, ρ is density, c_p heat capacity, k thermal conductivity and Q radiogenic heat production.

If the thermal conductivity is constant, we can rewrite

$$\frac{\partial T}{\partial t} = \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2} \right) + \frac{Q}{\rho c_p}. \quad (5.82)$$

5.4.1 Explicit method

The simplest way to discretize eq. (5.82) on a domain, *e.g.* a box with width L and height H , is to employ an FTCS, explicit method like in 1-D

$$\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = \kappa \left(\frac{T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n}{\Delta x^2} + \frac{T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n}{\Delta z^2} \right) + \frac{Q_{i,j}^n}{\rho c_p}. \quad (5.83)$$

There are now two indices for space, i and j for z_i and x_j , respectively (Figure 5.5). Rearranging gives

$$T_{i,j}^{n+1} = T_{i,j}^n + s_x (T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n) + s_z (T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n) + \frac{Q_{i,j}^n \Delta t}{\rho c_p}, \quad (5.84)$$

where

$$s_x = \frac{\kappa \Delta t}{\Delta x^2} \quad \text{and} \quad s_z = \frac{\kappa \Delta t}{\Delta z^2}. \quad (5.85)$$

Boundary conditions can be set the usual way. A constant (Dirichlet) temperature on the left-hand side of the domain (at $j = 1$), for example, is given by

$$T_{i,j=1} = T_{left} \quad \text{for all } i. \quad (5.86)$$

A constant flux (Neumann BC) on the same boundary at $\{i, j = 1\}$ is set through fictitious boundary points

$$\begin{aligned} \frac{\partial T}{\partial x} &= c_1 \\ \frac{T_{i,2} - T_{i,0}}{2\Delta x} &= c_1, \\ T_{i,0} &= T_{i,2} - 2\Delta x c_1, \end{aligned} \quad (5.87)$$

which can then be plugged into eq. (5.84) so that for $j = 1$, for example,

$$T_{i,1}^{n+1} = T_{i,1}^n + s_x (2T_{i,2}^n - 2(T_{i,1}^n + \Delta x c_1)) + s_z (T_{i+1,1}^n - 2T_{i,1}^n + T_{i-1,1}^n) + \frac{Q_{i,1}^n \Delta t}{\rho c_p}. \quad (5.88)$$

The implementation of this approach is straightforward as T can be represented as a matrix with Matlab, to be initialized, for example, for n_z and n_x rows and columns, respectively, as

$$T = \text{zeros}(n_z, n_x); \quad (5.89)$$

and then accessed as $T(i, j)$ for $T_{i,j}$. The major disadvantage of fully explicit schemes is, of course, that they are only stable if

$$\frac{2\kappa \Delta t}{\min(\Delta x^2, \Delta z^2)} \leq 1. \quad (5.90)$$

5.4.2 Fully implicit method

If we employ a fully implicit, unconditionally stable discretization scheme as for the 1D exercise, eq. (5.82) can be written as

$$\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = \kappa \left(\frac{T_{i,j+1}^{n+1} - 2T_{i,j}^{n+1} + T_{i,j-1}^{n+1}}{\Delta x^2} + \frac{T_{i+1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i-1,j}^{n+1}}{\Delta z^2} \right) + \frac{Q_{i,j}^n}{\rho c_p}. \quad (5.91)$$

Rearranging terms with $n + 1$ on the left and terms with n on the right hand side gives

$$-s_z T_{i+1,j}^{n+1} - s_x T_{i,j+1}^{n+1} + (1 + 2s_z + 2s_x) T_{i,j}^{n+1} - s_z T_{i-1,j}^{n+1} - s_x T_{i,j-1}^{n+1} = T_{i,j}^n + \frac{Q_{i,j}^n \Delta t}{\rho c_p}. \quad (5.92)$$

As in the 1D case, we have to write these equations in a matrix A and a vector **rhs** (and use Matlab `c = A \ rhs` to solve for T^{n+1}). From a practical point of view, this is a bit more complicated than

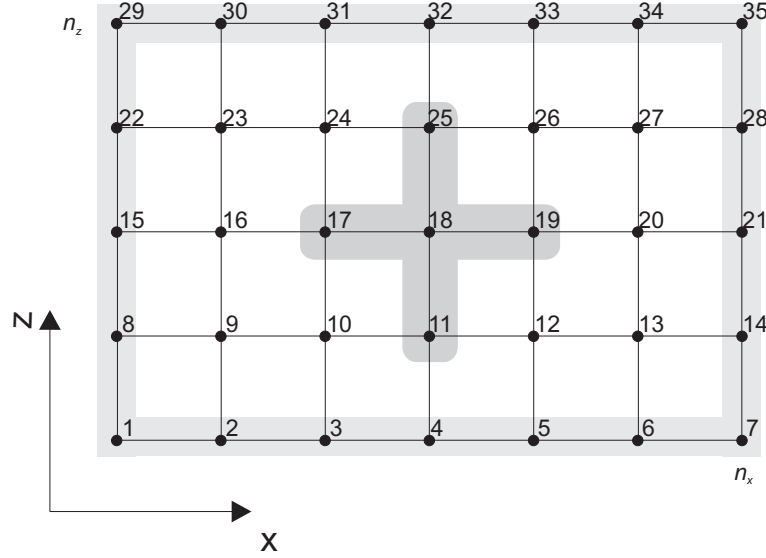


Figure 5.6: Numbering scheme for a 2D grid with $n_x = 7$ and $n_z = 5$.

in the 1D case, since we have to deal with “book-keeping” issues, *i.e.* the mapping of $T_{i,j}$ to the entries of a temperature vector $\mathbf{T}(k)$ (as opposed to the more intuitive matrix $\mathbf{T}(i, j)$ we could use for the explicit scheme).

If a 2D temperature field is to be solved for with an equivalent vector \mathbf{T} , the nodes have to be numbered continuously, for example as in Figure 5.6. The derivative versus x -direction is then *e.g.*

$$\frac{\partial^2 T}{\partial x^2} \big|_{j=3, i=4} = \frac{1}{\Delta x^2} (T_{19} - 2T_{18} + T_{17}), \quad (5.93)$$

and the derivative versus z -direction is given by

$$\frac{\partial^2 T}{\partial z^2} \big|_{j=3, i=4} = \frac{1}{\Delta z^2} (T_{25} - 2T_{18} + T_{11}). \quad (5.94)$$

If n_x are the number of grid points in x -direction and n_z the number of points in z -direction, we can write eqs. (5.93) and (5.94) in a more general way as:

$$\frac{\partial^2 T}{\partial x^2} \big|_{i,j} = \frac{1}{\Delta x^2} (T_{(i-1)n_x+j+1} - 2T_{(i-1)n_x+j} + T_{(i-1)n_x+j-1}) \quad (5.95)$$

$$\frac{\partial^2 T}{\partial z^2} \big|_{i,j} = \frac{1}{\Delta z^2} (T_{i \cdot n_x + j} - 2T_{(i-1)n_x + j} + T_{(i-2)n_x + j}). \quad (5.96)$$

In matrix format this gives something like (cf. eq. 5.92)

$$A = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & & & & & & & & & & & & \\ 0 & 0 & & -s_z & \dots & -s_x & (1 + 2s_x + 2s_z) & -s_x & \dots & -s_z & 0 & & 0 & 0 \\ 0 & 0 & & 0 & -s_z & \dots & -s_x & (1 + 2s_x + 2s_z) & -s_x & \dots & -s_z & & 0 & 0 \\ \vdots & \vdots & & & & & & & & & & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}. \quad (5.97)$$

Note that we now have 5 diagonals filled with non-zero entries as opposed to 3 diagonals in the 1D case. The solution vector \mathbf{c} is given by

$$\mathbf{c} = \begin{pmatrix} T_1^{n+1} = T_{1,1} \\ T_2^{n+1} = T_{1,2} \\ \vdots \\ T_{(i-1)n_x+j}^{n+1} = T_{i,j} \\ T_{(i-1)n_x+j+1}^{n+1} = T_{i,j+1} \\ \vdots \\ T_{n_x n_z - 1}^{n+1} = T_{n_z, n_x - 1} \\ T_{n_x n_z}^{n+1} = T_{n_z, n_x} \end{pmatrix}, \quad (5.98)$$

and the load (right hand side) vector is given by ($Q = 0$ for simplicity)

$$\mathbf{rhs} = \begin{pmatrix} T_{bottom} \\ T_{bottom} \\ \vdots \\ T_{(i-1)n_x+j}^n \\ T_{(i-1)n_x+j+1}^n \\ \vdots \\ T_{top} \\ T_{top} \end{pmatrix}. \quad (5.99)$$

5.4.3 Other methods

The fully implicit method discussed above works fine, but is only first order accurate in time (sec. 5.2). A simple modification is to employ a Crank-Nicolson time step discretization which is second order accurate in time. In practice, this often does not make a big difference, but Crank-Nicolson is preferred and does not cost much in terms of additional programming, so you may consider using it for diffusion-type equations.

A different, and more serious, issue is the fact that the cost of solving $c = A \backslash \text{rhs}$ is a strong function of the size of A . This size depends on the number of grid points in x - (n_x) and z -direction (n_z). For a 2D problem with $n_x \times n_z$ internal points, $(n_x \times n_z)^2 \times (n_x \times n_z)^2$ equations have to be solved at every time step. This quickly fills the computer memory (especially if going to 3D cases).

For the special case of the temperature equation, different techniques have therefore been developed. One such technique, is the *alternating direction implicit* (ADI) method. It basically consists of solving the 2D equations half-explicit and half-implicit along 1D profiles (what you do is the following: (1) discretize the heat equation implicitly in the x -direction and explicit in the z -direction. (2) solve it for time $n + 1/2$, and (3) repeat the same but with an implicit discretization in the z -direction). Compared to the other methods, ADI is fast. However, ADI-methods only work if the governing equations have time-derivatives, and unfortunately this is often not the case in geodynamics. In the exercises, we therefore focus on the fully implicit formulation. If, however, you have to write a thermal solver at some point, you may strongly consider to use the ADI method (which is still very fast in 3D).

5.4.4 Exercises

In the first two exercises you are to program the diffusion equation in 2D both with an explicit and an implicit discretization scheme. The problem to be considered is that of the thermal structure of a lithosphere of 100 km thickness, with an initial linear thermal gradient of 13 K/km. Suddenly a plume with $T = 1500^\circ \text{C}$ impinges at the bottom of the lithosphere. What happens with the thermal structure of the lithosphere? A related (structural geology) problem is that of the cooling of batholiths (like the ones in the Sierra Nevada).

1. Fill in the question marks in the script `heat2Dexplicit.m` (Figure 5.7), by programming the explicit finite difference scheme. Employ zero flux boundary conditions $\frac{\partial T}{\partial x} = 0$ on the left and on the right-side of the domain (outside the top and bottom edges), and constant temperature conditions on the top and bottom. Ignore the effects of radioactive heat.
2. Finish the code `heat2Dimplicit.m` (Figure 5.8), by programming the implicit finite difference approximation of the 2D temperature equation.
3. A simple (time-dependent) analytical solution for the temperature equation exists for the case that the initial temperature distribution is

$$T(x, z, t = 0) = T_{max} \exp \left[\frac{-(x^2 + z^2)}{\sigma^2} \right] \quad (5.100)$$

where T_{max} is the maximum amplitude of the temperature perturbation at $(x, z) = (0, 0)$ and σ its half-width. The solution is then

$$T(x, z, t) = \frac{T_{max}}{1 + 4t\kappa/\sigma^2} \exp \left[\frac{-(x^2 + z^2)}{\sigma^2 + 4t\kappa} \right] \quad (5.101)$$

Program the analytical solution and compare it with the explicit and fully implicit numerical solutions with the same initial conditions at each time step. Comment on the accuracy of both methods for different values of Δt .

4. Bonus question 1: Add the effects of radioactive heat to the explicit/implicit equations above. Use *Turcotte and Schubert (2002)* or Google to find typical values of Q, ρ, c_p for rocks.
5. Bonus question 2: Write a code for the thermal equation with variable thermal conductivity k . Assume that the grid spacing Δx is constant. This type of code is not only relevant for thermal problems, but also for problems like (1) hydro-geological problems (Darcy flow, *e.g.* how far did the chemical waste go into the aquifer?), (2) fluid movements through the crust and through fault zones (which is related to the creation of ore deposits), (3) magma migration through the mantle, (4) geochemistry and mineral reactions at grain-boundary scale, (5) aftershocks and fluids.

```

% Solves the 2D heat equation with an explicit finite difference scheme
clear
%Physical parameters
L = 150e3; % Width of lithosphere [m]
H = 100e3; % Height of lithosphere [m]
Tbot = 1300; % Temperature of bottom lithosphere [C]
Tsurf = 0; % Temperature of country rock [C]
Tplume = 1500; % Temperature of plume [C]
kappa = 1e-6; % Thermal diffusivity of rock [m2/s]
Wplume = 25e3; % Width of plume [m]
day = 3600*24; % # seconds per day
year = 365.25*day; % # seconds per year

% Numerical parameters
nx = 101; % # gridpoints in x-direction
nz = 51; % # gridpoints in z-direction
nt = 500; % Number of timesteps to compute
dx = L/(nx-1); % Spacing of grid in x-direction
dz = H/(nz-1); % Spacing of grid in z-direction
[x2d,z2d] = meshgrid(-L/2:dx:L/2, -H:dz:0); % create grid
% Compute stable timestep
dt = min([dx,dz])^2/kappa/4;
% Setup initial linear temperature profile
T = abs(z2d./H)*Tbot;
% Imping plume beneath lithosphere
ind = find(abs(x2d(1,:)) <= Wplume/2);
T(1,ind) = Tplume;
time = 0;
for n=1:nt

    % Compute new temperature
    Tnew = zeros(nz,nx);
    sx = kappa*dt/dx^2;
    sz = kappa*dt/dz^2;
    for j=2:nx-1
        for i=2:nz-1
            Tnew(i,j) = ???;
        end
    end
    % Set boundary conditions
    Tnew(1,:) = T(1, :);
    Tnew(nz,:) = ?;
    for i=2:nz-1
        Tnew(i,1) = ?
        Tnew(i,nx) = ?
    end
    T = Tnew;
    time = time+dt;
    % Plot solution every 50 timesteps
    if (mod(n,50)==0)
        figure(1), clf
        pcolor(x2d/1e3,z2d/1e3,Tnew); shading interp, colorbar
        hold on
        contour(x2d/1e3,z2d/1e3,Tnew,[100:100:1500],'k');
        xlabel('x [km]')
        ylabel('z [km]')
        zlabel('Temperature [^oC]')
        title(['Temperature evolution after ',num2str(time/year/1e6),' Myrs'])
        drawnow
    end
end
end

```

Figure 5.7: MATLAB script heat2D_explicit.m to solve the 2D heat equation.

```

% Solves the 2D heat equation with an implicit finite difference scheme
clear
%Physical parameters
L = 150e3; % Width of lithosphere [m]
H = 100e3; % Height of lithosphere [m]
Tbot = 1300; % Temperature of bottom lithosphere [C]
Tsurf = 0; % Temperature of country rock [C]
Tplume = 1500; % Temperature of plume [C]
kappa = 1e-6; % Thermal diffusivity of rock [m2/s]
Wplume = 25e3; % Width of plume [m]
day = 3600*24; % # seconds per day
year = 365.25*day; % # seconds per year
dt = 100e6*year; % timestep
% Numerical parameters
nx = 51; % # gridpoints in x-direction
nz = 51; % # gridpoints in z-direction
nt = 100; % Number of timesteps to compute
dx = L/(nx-1); % Spacing of grid in x-direction
dz = H/(nz-1); % Spacing of grid in z-direction
[x2d,z2d] = meshgrid(-L/2:dx:L/2, -H:dz:0); % create grid
% Setup initial linear temperature profile
T = abs(z2d./H)*Tbot;
% Imping plume beneath lithosphere
ind = find(abs(x2d(1,:)) <= Wplume/2);
T(1,ind) = Tplume;
% Setup numbering
num = 1;
for i=1:nz
    for j=1:nx
        Number(i,j) = num;
        num = num+1;
    end
end
% Construct the A matrix
A = sparse(nx*nz,nx*nz);
sx = kappa*dt/dx^2;
sz = kappa*dt/dz^2;
for i = 2:nz-1
    for j = 2:nx-1
        ii = Number(i,j);
        A(ii, Number(i+1,j)) = sx;
        A(ii, Number(i,j+1)) = sz;
    end
end
% Set lower and upper BC
for j = 1:nx
    T(1,j) = Tbot;
    T(nz,j) = Tsurf;
end
% Set left and right BC
for i = 1:nz
    T(i,1) = Tbot;
    T(i,nx) = Tsurf;
end
time = 0;
for n=1:nt
    % Compute rhs
    rhs = zeros(nx*nz,1);
    for i = 1:nz
        for j = 1:nx
            ii = Number(i,j);
            rhs(ii) = T(i,j);
        end
    end
    % Compute solution vector
    Tnew_vector = A\rhs;
    % Create 2D matrix from vector
    Tnew = Tnew_vector(Number);
    T = Tnew;
    time = time+dt;
    % Plot solution every 50 timesteps
    if (mod(n,50)==0)
        figure(1), clf
        pcolor(x2d/1e3,z2d/1e3,Tnew); shading interp, colorbar
        hold on
        contour(x2d/1e3,z2d/1e3,Tnew,[100:100:1500],'k');
        xlabel('x [km]')
        ylabel('z [km]')
        zlabel('Temperature [°C]')
        title(['Temperature evolution after ',num2str(time/year/1e6),' Myrs'])
        drawnow
    end
end
end

```

Figure 5.8: MATLAB script heat2D.implicit.m to solve the 2D heat equation.

5.5 Advection equations with FD

Reading

- *Spiegelman* (2004), chap. 5
- *Press et al.* (1993), sec. 19.1

5.5.1 The diffusion-advection (energy) equation for temperature in convection

So far, we mainly focused on the diffusion equation in a non-moving domain. This is maybe relevant for the case of a dike intrusion or for a lithosphere which remains undeformed. However, more often, we want to consider problems where material moves during the time period under consideration and takes temperature anomalies with it. An example is a plume rising through a convecting mantle. The plume is hot and hence its density is low compared to the colder mantle around it. The hot material rises with a velocity that depends on the density anomaly and viscosity (see Stokes velocity). If the numerical grid remains fixed in the background, the hot temperatures should be moved to different grid points at each time step (see Figure 5.9 for an illustration of this effect).

More generally speaking, mantle convection is an example of a system where heat is transported by diffusion (temperature changes without moving mass, particularly important in the boundary layers) and advection (temperature changes by transport, dominant in the interior the domain). How strongly these two effects are partitioned is controlled globally by the Rayleigh number, and locally by the Peclet number (see sec. 2.3 on scaling analysis).

Mathematically, the temperature equation gets an additional term for advection in a Eulerian (fixed grid) system. In 1-D and in the absence of heat sources, the diffusion-advection equation becomes (see continuum mechanics, sec. 2.2)

$$\rho c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} \right) = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) \quad (5.102)$$

or in 2-D

$$\rho c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} \right) = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) \quad (5.103)$$

where v_x, v_z are velocities in x -, respectively z -direction. If k is constant, the general equation can be written as

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \kappa \nabla^2 T. \quad (5.104)$$

Heat sources would lead to additional terms on the right hand side. Since temperature variations lead to buoyancy forces, the energy equation couples into the Stokes (momentum equilibrium) equation from which velocities \mathbf{v} can be computed to close the system needed for a convection code. You can explore the coupling of an energy and a Stokes solver in your term project.

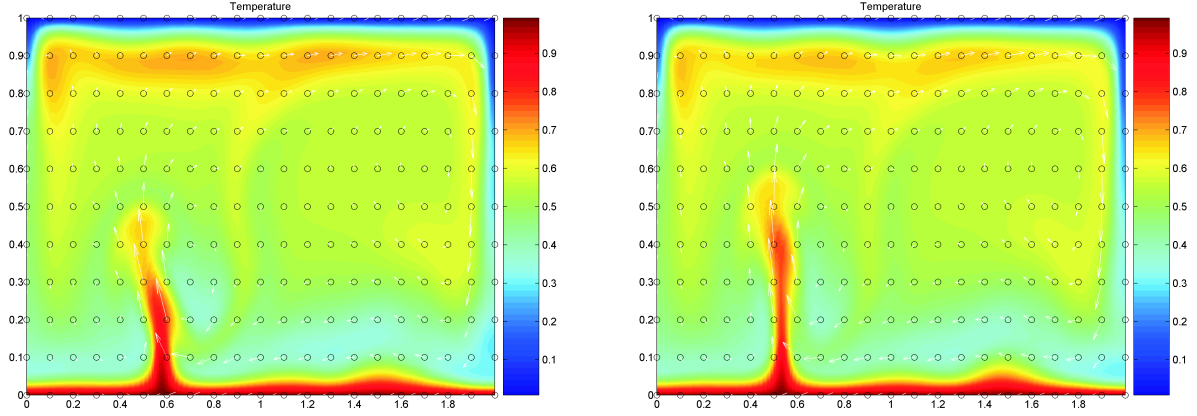


Figure 5.9: Snapshots of a bottom heated thermal convection model with a Rayleigh-number of 5×10^5 and constant viscosity (no internal heating). Temperature is advected through a fixed (eulerian) grid (circles) with a velocity (arrows) that is computed with a Stokes solver.

Mantle convection codes typically deal with advection of a temperature field assuming that there is significant diffusion at the same time, $\kappa > 0$, and will produce artifacts in cases that are advection-dominated. One example would be if a chemical composition C is to be treated akin to T ,

$$\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C = \kappa_c \nabla^2 C. \quad (5.105)$$

Chemical diffusivities are for mantle purposes zero, $\kappa_c \approx 0$, and special tricks are required to use field methods to solve

$$\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C = 0 \quad (5.106)$$

(e.g. *Lenardic and Kaula, 1993*), as discussed below. Often, one therefore uses tracer-based, or “particle methods” where C is assigned to virtual particles that are then advected with an ODE approach (to be solved with, e.g., Runge Kutta)

$$\frac{dC}{dt} = 0 \quad \text{and} \quad \frac{d\mathbf{x}_i}{dt} = \mathbf{v} \quad (5.107)$$

where \mathbf{x}_i is the location of the i -th tracer moving through the fluid. We will return to a hybrid approach below, but see, e.g., *Tackley and King (2003)* for a recent discussion of different tracer approaches. A related method is based on marker chains (e.g. *van Keken et al., 1997*), this works well if we are mainly interested in tracking a single interface between different materials with C_1 and C_2 . For the latter problem, “level set” methods are also promising (e.g. *Suckale et al., 2010; Samuel and Evonuk, 2010*).

5.5.2 Advection (transport equations)

We will return to the combined (“combo”) solution of both diffusion and advection below, but for now focus on the advection part. In the absence of diffusion (*i.e.* $k, \kappa = 0$), the 1-D equations are

$$\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} = 0 \quad (5.108)$$

and

$$\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} = 0. \quad (5.109)$$

We will now evaluate some options on how to solve these equations with a finite difference scheme on a fixed grid. Even though the equations appear simple, it is quite tricky to solve them accurately, more so than for the diffusion problem. This is particularly the case if there are large gradients in the quantity that is to be advected, *e.g.* ∇T . If not done carefully, one can easily end up with strong numerical artifacts such as wiggles (oscillatory artifacts) and numerical diffusion (artificial smoothing of the solution).

FTCS method

In 1-D, the simplest way to discretize eq. (5.108) is by employing a central difference scheme in space, and go forward in time (another example of a forward-time, central space, FTCS, scheme):

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = -v_{x,i} \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x}, \quad (5.110)$$

where $v_{x,i}$ is the v_x velocity at location i .

Exercise 1 We will consider a 1-D problem, with constant v_x velocity in which an exponential pulse of temperature is getting advected along the x axis.

- Program the FTCS method in the code of Figure 5.10 and watch what happens. Change the sign of the velocity. Change the time step and grid spacing and compute the non-dimensional parameter $|v_x| \Delta t / \Delta x$. When do unstable results occur? Put differently, can you find a Δt small enough to avoid blow-up?

As you can see from the exercise, the FTCS method does not work so well ... In fact, it is a nice example of a scheme that looks logical on paper, but looks can be deceiving. The FTCS method is *unconditionally unstable*, blow up for any Δt , as can be shown by *von Neumann stability analysis* (*cf.* chap 5 of [Spiegelman, 2004](#)). The instability is related to the fact that this scheme produces negative diffusion, which is numerically unstable.


```

%
% FTCS advection schem
%
clear all

nx      = 201;
W       = 40;    % width of domain
Vel     = -4;    % velocity
sigma   = 1;
Ampl    = 2;
nt      = 500;   % number of timesteps
dt      = 1e-2;  % timestep
dx      = W/(nx-1);
x       = 0:dx:W;
% Initial Gaussian T-profile
xc      = 20;
T       = Ampl*exp(-(x-xc).^2/sigma^2);
% Velocity
Vx      = ones(1,nx)*Vel;
abs(Vel)*dt/dx
cfac = dt/(2*dx);
% Central finite difference discretization
for itime=1:nt
    % central fin. diff
    for ix=2:nx-1
        Tnew(ix) = T(ix) - Vx(ix)*cfac *(T(ix+1)-T(ix-1));
    end
    % BCs
    Tnew(1) = T(1);
    Tnew(nx) = T(nx);
    T = Tnew;
    time = itime*dt;
    % Analytical solution for this case
    T_anal = Ampl*exp(-(x-xc-time*Vel).^2/sigma^2);
    figure(1),clf, plot(x,T,x,T_anal), ...
    legend('Numerical','Analytical')
    xlabel('x')
    ylabel('temperature')
    drawnow
end

```

Figure 5.10: MATLAB script to be used with FTCS exercise 1.

Lax method

The Lax approach consists of replacing the T_i^n in the time-derivative of eq. (5.110) with $(T_{i+1}^n + T_{i-1}^n)/2$. The resulting equation is

$$\frac{T_i^{n+1} - (T_{i+1}^n + T_{i-1}^n)/2}{\Delta t} = -v_{x,i} \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x} \quad (5.111)$$

Exercise 2

- Program the Lax method by modifying the script of the last exercise. Try different velocities and Δt settings and compute the *Courant number* α , which is given by the following equation:

$$\alpha = \frac{v_x \Delta t}{\Delta x} \quad (5.112)$$

Is the numerical scheme stable for all Courant numbers? What is the physical meaning of α ? What happens for $\alpha = 1$ and why?

- *Bonus question:* Implement a generalized Galerkin-Lax-Wendroff method using the follow-

ing equation:

$$[M_x - \frac{\alpha^2}{\Delta x^2} \frac{\partial^2}{\partial x^2}](T_i^{n+1} - T_i^n) + \alpha \Delta x \frac{\partial}{\partial x} T_i^n - \frac{\alpha^2 \Delta x^2}{2} \frac{\partial}{\partial x} T_i^n = 0 \quad (5.113)$$

where $M_x = \{\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\}$ and spatial derivatives are discretized using second order central differences:

$$\frac{1}{6}(1 - c^2 \Delta x^2)(T_{i+1}^{n+1} - T_{i-1}^{n+1}) + \frac{2}{3}(1 + c^2 \Delta x^2)T_i^{n+1} = (\frac{1}{6} - \frac{\alpha}{2} + \frac{\alpha^2 \Delta x^2}{3})T_{i+1}^n + \frac{2}{3}(1 - \alpha^2 \Delta x^2)T_i^n + (\frac{1}{6} - \frac{\alpha}{2} + \frac{\alpha^2 \Delta x^2}{3})T_{i-1}^n \quad (5.114)$$

This formulation gives us much better accuracy ($O(\Delta t^2, \Delta x^2)$) by using a higher order discretization in both time and space. But what is its stability range in terms of Courant number? Notice the difference in terms of artificial diffusion, and oscillations with respect to the simple Lax method.

As you saw from the exercise the Lax method does not blow up, but does have a lot of numerical diffusion for $\alpha \neq 1$ (which is hard to attain for realistic problems, as v will vary in space and time). In fact, the Lax criterion stabilized the advection equation by adding some artificial diffusion. So, it's an improvement but it's far from perfect, since you may now lose the plumes of Figure 5.9 around mid-mantle purely due to numerical diffusion. As for the case of the implicit versus explicit solution of the diffusion equation, you see that there are trade-offs between stability and accuracy. There is no free lunch, and numerical modeling is also a bit of an art.

The stability requirement

$$\alpha = \frac{|V|\Delta t}{\Delta x} \leq 1 \quad (5.115)$$

is called the *Courant criterion* (Figure 5.11).

Streamline upwind scheme

A popular scheme is the so-called (streamline) upwind approach (Figure 5.12a). Here, the spatial finite difference scheme depends on the sign of the velocity:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = -v_{x,i} \begin{cases} \frac{T_i^n - T_{i-1}^n}{\Delta x}, & \text{if } v_{x,i} > 0 \\ \frac{T_{i+1}^n - T_i^n}{\Delta x}, & \text{if } v_{x,i} < 0 \end{cases} \quad (5.116)$$

Note that we have replaced central with forward or backward derivatives, depending on the flow direction. The idea is that the flux into the local cell at x_i will only depend on the gradient of temperature in the direction “upstream”, *i.e.* where the inflowing velocity is coming from.

Exercise 3

- Program the upwind scheme method. Try different velocity distributions (not just constant) and compute the Courant numbers α . Is the numerical scheme stable for all Courant numbers?

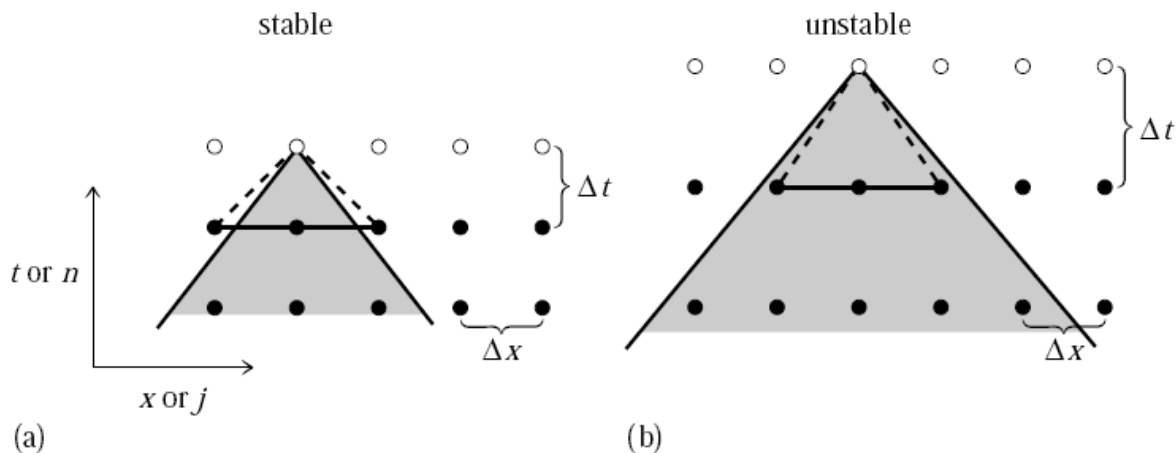


Figure 19.1.3. Courant condition for stability of a differencing scheme. The solution of a hyperbolic problem at a point depends on information within some domain of dependency to the past, shown here shaded. The differencing scheme (19.1.15) has its own domain of dependency determined by the choice of points on one time slice (shown as connected solid dots) whose values are used in determining a new point (shown connected by dashed lines). A differencing scheme is Courant stable if the differencing domain of dependency is larger than that of the PDEs, as in (a), and unstable if the relationship is the reverse, as in (b). For more complicated differencing schemes, the domain of dependency might not be determined simply by the outermost points.

Figure 5.11: Illustration of the Courant criterion (from *Press et al.*, 1993, chap 19.1).

The upwind scheme also suffers from numerical diffusion, and it is only first order accurate in space. For some applications, particularly if there's also diffusion, it might just be good enough because the simple trick of doing FD forward or backward is closer to the underlying physics of transport than, say, FTCS. There are some mantle convection codes that use streamline upwind schemes.

So far, we employed explicit discretizations. You're probably wondering whether implicit discretizations will save us again this time. Bad news: they are not well-suited for this type of problem (try it and see). Implicit schemes behave like parabolic partial differential equations (*e.g.* the diffusion equation) in that a perturbation at node (j, n) will affect the solution at all nodes at time level $n+1$. With hyperbolic PDE's like the advection equation or the wave equation, disturbances travel at a finite speed (the speed of the material displacement) and will not affect all nodes at time level $n+1$. So we have to come up with something else.

Modified Crank-Nicolson

One approach to solving the advection equation is the previously introduced Crank-Nicolson semi-implicit scheme. Here we modify it slightly by introducing a general mass operator $M_x = \{\delta, 1 - 2\delta, \delta\}$.

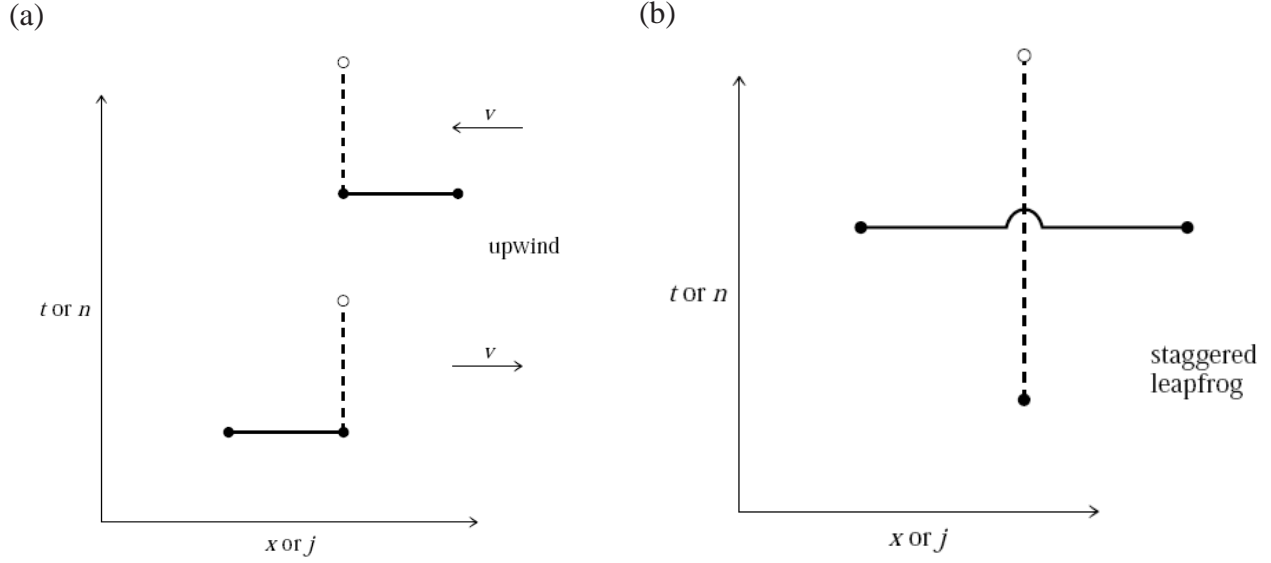


Figure 5.12: Illustration of the upwind (a) and leapfrog (b) schemes (from *Press et al., 1993*, chap 19.1).

$$M_x \left[\frac{T_i^{n+1} - T_i^n}{\Delta t} + v \left[\frac{1}{2} \frac{(T_{i+1}^n - T_{i-1}^n) + (T_{i+1}^{n+1} - T_{i-1}^{n+1})}{2\Delta t} \right] \right] = 0 \quad (5.117)$$

Setting the mass operator to $\delta = 0$ gives us the previously seen Crank-Nicolson semi-implicit finite difference discretization, while setting $\delta = \frac{1}{6}$ gives us the finite element formulation. Below is eq. (5.117) written out with $\delta = \frac{1}{6}$.

$$\left(\frac{1}{6} - \frac{1}{4} \left(v \frac{\delta t}{\delta x} \right) \right) T_{(i-1)}^{(n+1)} - \left(1 - \frac{1}{3} \right) T_i^{(n+1)} + \left(\frac{1}{6} + \frac{1}{4} \left(v \frac{\delta t}{\delta x} \right) \right) T_{(j+1)}^{(n+1)} = \left(\frac{1}{6} + \frac{1}{4} \left(v \frac{\delta t}{\delta x} \right) \right) T_{(j-1)}^n + \left(1 - \frac{1}{3} \right) T_i^n + \left(\frac{1}{6} - \frac{1}{4} \left(v \frac{\delta t}{\delta x} \right) \right) T_{(i+1)}^n \quad (5.118)$$

The finite element Crank-Nicolson advection scheme is stable for $\alpha \leq 1$ and provides an improvement over previous schemes in that it is accurate to $O(\delta t, \delta x^3)$. This allows us to reduce the number of grid points to reach the same accuracy as the other schemes presented, as long as dt is kept small enough.

Staggered leapfrog

The explicit discretizations discussed so far were second order accurate in time, but only first order in space. We can also come up with a scheme that is second order in time and space

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = -v_{x,i} \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x}, \quad (5.119)$$

called *staggered leapfrog* because of the way it's centered in shifted space-time (Figure 5.12b). The computational inconvenience in this scheme is that two time steps have to be stored, T^{n-1} and T^n .

Exercise 4

- Program the staggered leapfrog method (assume that at the first time step $T^{n-1} = T^n$) and the two formulations of the Crank-Nicolson method with $\delta = 0$ and $\delta = \frac{1}{6}$. Try with different values of the Courant number α and compare the accuracies and stability of the three methods. Also make the width of the Gaussian curve smaller.

The staggered leapfrog method works quite well regarding the amplitude and transport phase as long as α is close to one. If, however, $\alpha \ll 1$ and the length scale of the to-be-transported quantity is small compared to the number of grid points (*e.g.* we have a thin plume), numerical oscillations again occur (those are due to the lack of communication between cells, which can be remedied by ... artificial diffusion). The conditions where leapfrog does not work well are typically the case in mantle convection simulations (*cf.* Figure 5.9). Onward ever, backward never.

Similarly, the Crank-Nicolson method works well for $\nu \leq 0.1$ and $\alpha \leq 0.1$, and eliminates the staggered problem. But what happens for $\alpha \geq 0.1$? What about the finite element formulation? What about computational time? Is Crank-Nicolson's increased accuracy worth the extra work? Is it well-suited for mantle convection problems?

MPDATA

This is a technique that is frequently applied in (older) mantle convection codes. The idea is based on [Smolarkiewicz \(1983\)](#) and represents an attempt to improve on the upwind scheme by adding some anti-diffusion, which requires iterative corrections. The results are pretty good, but MPDATA is somewhat more complicated to implement. Moreover we still have a restriction on the time step (given by the Courant criterion), for details see [Spiegelman \(2004\)](#).

Semi-Lagrangian approach

What we want is a scheme that is stable, has only small numerical diffusion and is not limited by the Courant criterion. A contender is the semi-Lagrangian method, which is often used for climate modeling. The method is related to tracer-based advection by solving ODEs and has little to do with the finite difference schemes we discussed so far. Since this scheme could be the one that is most important for practical purposes we will go in more detail. It has few drawbacks, one being that it is not necessarily flux conserving.

Basic idea The basic idea of the semi-Lagrangian method is illustrated in Figure 5.13A and is given by the following, simplified scheme. Instead of allowing the numerical scheme to transport noise in from unknown regions, the semi-Lagrangian method uses transport by going back one (*e.g.* Euler) step.

For each point i at x_i and time t_n :

1. Assume that the future velocity $v_x(t_{n+1}, x_i)$ at x_i is known. Under the assumption that the velocity at the old time step is close to the future velocity ($v_x(t_{n+1}, x_i) \approx v_x(t_n, x_i)$) and that velocity does not vary spatially ($v_x(t_n, x_{i-1}) \approx v_x(t_n, x_i) \approx v_x(t_n, x_{i+1})$), we can compute the location \mathbf{X} where the particle came from by $\mathbf{X} = x_i - \Delta t v_x(t_{n+1}, x_i)$.

- Interpolate temperature from grid points $\{x_i\}$ to the location X at time t_n , $T(t_n, X)$. For example, use cubic interpolation (in MATLAB use the command `interp1(x,T, X, 'cubic')` for interpolation, where x is supposed to be the vector that holds the $\{x_i\}$).

Note 1: Be careful with interpolation. For smooth functions, polynomial interpolation, say of cubic order, is often a good idea. However, at edges, or if the function is otherwise discontinuous, “ringing”, *i.e.* large, wiggly excursions, can occur. Linear, or spline, interpolation may be preferred.

Note 2: Most of the Matlab interpolation functions will by default not extrapolate outside the

$$[\min(x_i), \max(x_i)]$$

range and return NaN (not a number). If extrapolation is desired, ‘`extrap`’ needs to be set.

- Assume that $T(t_{n+1}, x_i) = T(t_n, X)$, *i.e.* temperature has been transported (along “characteristics”) without any modification (*e.g.* due to diffusion).

This scheme assumes that no heat-sources were active during the advection of T from $T(t_n, X)$ to $T(t_{n+1}, x_i)$. If heat sources are present *and* are spatially variable, some extra care needs to be taken ([Spiegelman, 2004](#), sec. 5.6.1).

Exercise 5

- Program the semi-Lagrangian advection scheme illustrated in Figure 5.13A. Is there a Courant criterion for stability?

Some improvements The algorithm described in Figure 5.13A illustrates the basic idea of the semi-Lagrangian scheme. However, it has two problems. First it assumes that velocity is spatially constant (which is clearly not the case in mantle convection simulations). Second, it assumes that velocity does not change between time n and $n + 1$. We can overcome both problems by using a more accurate time stepping algorithm (see the ODE section). An example is an iterative mid-point scheme which works as follows (*cf.* Figure 5.13B):

For each point i

- Use the velocity $v_x(t_{n+1}, x_i)$ to compute the location X' at time $t_{n+1}/2$ (*i.e.* take half a time step backward in time).
- Find the velocity at the location X' at half time step $t_{n+1}/2$. Assume that the velocity at the half time step can be computed as $v_x(t_{n+1}/2, x_i) = \frac{v_x(t_{n+1}, x_i) + v_x(t_n, x_i)}{2}$. Use linear interpolation for the spatial interpolation of velocity $v_x(t_{n+1}/2, X')$.
- Go back to point 1, but use the velocity $v_x(t_{n+1}/2, X')$ instead of $v_x(t_{n+1}, x_i)$ to move the point $x_i(t_{n+1})$ backward in time. Repeat this process a number of times (*e.g.* 5 times). This gives a fairly accurate centered velocity.
- Compute the location X at t_n with the centered velocity $X = x_i - \Delta t v_x(t_{n+1}/2, X')$.
- Use cubic interpolation to find the temperature at point X as before.

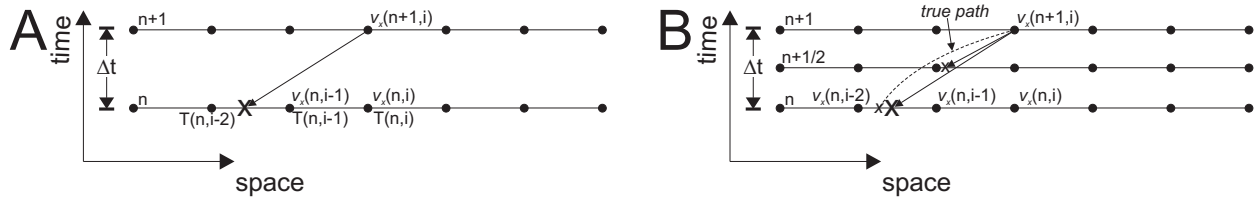


Figure 5.13: Basics of the semi-Lagrangian method. See text for explanation.

Other ODE-motivated methods such as 4th order Runge Kutta are also possible (but take a bit more work). Note that the various velocity interpolation and iteration schemes add overhead that is, however, typically made up for by not needing to obey the Courant criterion.

Exercise 6

- Program the semi-Lagrangian advection scheme with the centered midpoint method as illustrated in Figure 5.13B (cf. *Spiegelman, 2004*, p. 67). Some care has to be taken if point \mathbf{X} is outside of the computational domain, since MATLAB will return NaN for the velocity (or temperature) of this point. If no extrapolation is desired, use the velocity $v_x(t_{n+1}, x_i)$ in this case. A pseudo-code is given by

```

if isnan(Velocity)
    Velocity = Vx(i)
end

```

2D advection example

The semi-Lagrangian method is likely a good, general advection algorithm (except in the case of pseudo spectral methods), so this is the one we will implement in 2D.

Assume that velocity is given by

$$v_x(x, z) = z \quad (5.120)$$

$$v_z(x, z) = -x \quad (5.121)$$

Moreover, assume that the initial temperature distribution is Gaussian and given by

$$T(x, z) = 2 \exp \left(-\frac{((x + 0.25)^2 + z^2)}{0.1^2} \right) \quad (5.122)$$

with $x \in [-0.5, 0.5]$, $z \in [-0.5, 0.5]$.

Exercise 7

- Program advection in 2D using the semi-Lagrangian advection scheme with the centered midpoint method. Use the MATLAB routine `interp2` for interpolation and employ linear interpolation for velocity and cubic interpolation for temperature. A MATLAB script that will get you started is shown on Figure 5.14.

5.5.3 Advection and diffusion: operator splitting

In geodynamics, we often want to solve the coupled advection-diffusion equation, which is given by eq. (5.102) in 1-D and by eq. (5.103) in 2-D. We can solve this pretty easily by taking the equation apart and by computing the advection part separately from the diffusion part. This is called operator-splitting, and what is done in 1-D is, for example: First solve the advection equation

$$\frac{\tilde{T}^{n+1} - T^n}{\Delta t} + v_x \frac{\partial T}{\partial x} = 0 \quad (5.123)$$


```

% semi_lagrangian_2D: 2D semi-lagrangian with center midpoint time stepping method
%

clear all

W      = 40;      % width of domain
sigma  = .1;
Amp1   = 2;
nt     = 500;    % number of timesteps
dt     = 5e-1;

% Initial grid and velocity
[x,z]  = meshgrid(-0.5:.025:0.5,-0.5:.025:0.5);
nz     = size(x,1);
nx     = size(x,2);

% Initial gaussian T-profile
T      = Amp1*exp(-(x+0.25).^2+z.^2)/sigma^2);

% Velocity
Vx     = z;
Vz     = -x;

for itime=1:nt

    Vx_n = Vx;          % Velocity at time=n
    Vx_n1 = Vx;         % Velocity at time=n+1
    Vx_n1_2 = ??;       % Velocity at time=n+1/2
    Vz_n = ??;          % Velocity at time=n
    Vz_n1 = ??;         % Velocity at time=n+1
    Vz_n1_2 = ??;       % Velocity at time=n+1/2
    Tnew = zeros(size(T));
    for ix=2:nx-1
        for iz=2:nz-1

            Vx_cen = Vx(iz,ix);
            Vz_cen = Vz(iz,ix);
            %
            %      X = ?
            %      Z = ?

            %linear interpolation of velocity
            Vx_cen = interp2(x,z,?,?, ?, 'linear');
            Vz_cen = interp2(x,z,?,?, ?, 'linear');

            if isnan(Vx_cen)
                Vx_cen = Vx(iz,ix);
            end
            if isnan(Vz_cen)
                Vz_cen = Vz(iz,ix);
            end

        end
        %
        %      X = ?;
        %      Z = ?;

        % Interpolate temperature on X
        T_X = interp2(x,z,?,?,?, 'cubic');
        if isnan(T_X)
            T_X = T(iz,ix);
        end

        Tnew(iz,ix) = T_X;
    end

end

Tnew(1,:) = T(1,:);
Tnew(nx,:) = T(nx,:);
Tnew(:,1) = T(:,1);
Tnew(:,nx) = T(:,nx);

T      = Tnew;
time   = itime*dt;

figure(1),clf
pcolor(x,z,T), shading interp, hold on, colorbar
contour(x,z,T,[.1:1:2],'k'),
hold on, quiver(x,z,Vx,Vz,'w')
axis equal, axis tight
drawnow
pause
end

```

Figure 5.14: MATLAB script to be used with exercise 7.

for example by using a semi-Lagrangian advection scheme. Then solve the diffusion equation

$$\rho c_p \frac{\partial \tilde{T}}{\partial t} = \frac{\partial}{\partial x} \left(k \frac{\partial \tilde{T}}{\partial x} \right) + Q. \quad (5.124)$$

For this, we assumed that Q is spatially constant; if not, one should consider to slightly improve the advection scheme by introducing source terms. A good general method would be to combine Crank-Nicolson for diffusion with a semi-Lagrangian solver for advection ([Spiegelman, 2004](#), sec. 7.2), but we will try something simpler first:

Exercise 8

- Program diffusion-advection in 2D using the semi-Lagrangian advection scheme coupled with an implicit 2D diffusion code (from last week's exercise). Base your code on the script of figure [5.14](#).

Chapter 6

Finite elements

6.1 Introduction to finite element methods

Reading:

- Textbooks
 - The recommended source for this, finite element part of the class is the nice and well-priced textbook by *Hughes* (2000). This book is in wide use in mantle convection modeling and aspects of the codes ConMan (*King et al.*, 1990) and CitcomS (*Moresi and Solomatov*, 1995; *Zhong et al.*, 2000) (both now maintained and developed by CIG, geodynamics.org) follow the approach and notation of *Hughes* (2000).
 - For additional reference, you might want to consider the classic, comprehensive and high-level treatment by *Bathe* (2007) which is held by the library.
 - However, as before your lecture notes and the handouts are meant to be sufficient.
- *Hughes* (2000), chap. 1, secs. 1.1-1.15
- *Bathe* (2007), sec. 1.2

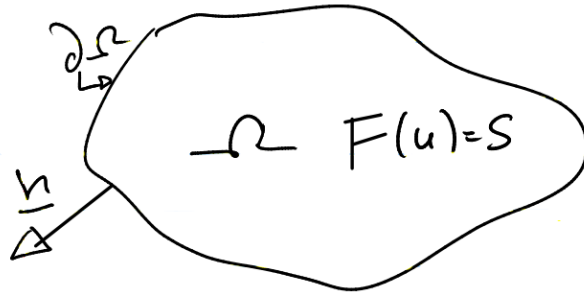
6.1.1 Philosophy of the finite element (FE) method

Consider a boundary value problem (and many physical problems in solid mechanics can be converted into a boundary value problem) given on a “domain” Ω with a boundary $\Gamma = \partial\Omega$ such that a solution $u(\mathbf{x})$ satisfies the PDE:

$$F(u(\mathbf{x})) = s(\mathbf{x}) \tag{6.1}$$

where F is some differential operator and s a source term.

As boundary conditions, we can have



Dirichlet (fixed value, “essential”)

$$u|_{\partial\Omega} = g \quad (6.2)$$

type constraints where the value of u is given on $\partial\Omega$, and/or

Neumann (flux, “natural”)

$$n_i \partial_i u = \mathbf{n} \cdot \nabla \mathbf{u} = h \quad (6.3)$$

conditions, where we specify the derivatives.

(If the PDE is, for example, an elastic deformation problem, then u would be displacements, and Dirichlet conditions of $g = 0$ correspond to “no-slip”, *i.e.* no deformation at the boundaries. Likewise, $h = 0$ Neumann conditions would correspond to zero tractions (the derivative of displacement times modulus are stresses), “free slip”.)

The FE analysis then proceeds by two steps:

1. Converting the governing PDE from the regular, “strong” form (which we used for FD) to the weak integral form (see below).
2. Discretizing the domain Ω into “elements” on which an approximate solution for u is to be obtained using simplified polynomials, so called “form functions” or basis functions.

We will provide a highly abbreviated treatment, lacking any mathematical rigor. Moreover, we will omit the detailed discussion of different element types, or shape functions, as well as implementation issues such as order of integration. Those issues are very important in practice, as choices in shape functions and element type may strongly affect solution robustness and accuracy. However, we don’t have time to delve into this (see, *e.g.* [Hughes, 2000](#)). For a specific mantle convection code pertaining to the commonly used ConMan and Citcom, see also [Zhong et al. \(2007\)](#).

To contrast finite elements with finite difference methods, we can summarize the main differences and how they affect usage:

Finite Differences (FD)

- approximates the PDE
- mainly restricted to simple, rectangular domains
- regional, or adaptive mesh refinement hard to implement
- simple implementation
- special case of FE

Finite Elements (FE)

- approximates the solution of the PDE
- complex geometries fairly easily implemented
- regional mesh refinement easy, adaptive refinement fairly straight forward (problems with “smoothing”)
- involved implementation

6.1.2 A one – dimensional example

Consider

$$\partial_{xx}u + s = 0 \quad (6.4)$$

on the domain $[0, 1]$ where s is given. Eq. (6.4) could be a 1-D steady-state heat equation, for example, where

$$\partial_{xx}T - \frac{H(x)}{k} = 0. \quad (6.5)$$

Mathematically, we require s to be smooth for a solution for u to exist. Additionally, we will require

$$u(0) = g \quad (\text{Dirichlet}) \quad (6.6)$$

and

$$\partial_x u(1) = h \quad (\text{Neumann}) \quad (6.7)$$

boundary conditions (BC’s), which closes the system for a two point boundary value problem. This formulation of the PDE with all original derivatives in place is called the *strong form*. Eqs. (6.4) and (6.6) can, of course, be solved analytically by integration. For example, if $s = x$ such that

$$\partial_{xx}u + x = 0 \quad (6.8)$$

then by integration

$$\partial_x u + \frac{1}{2}x^2 + c_1 = 0 \quad (6.9)$$

and integrating again

$$u + \frac{1}{6}x^3 + c_1x + c_2 = 0 \quad (6.10)$$

$$u(x) = -\frac{1}{6}x^3 - c_1x - c_2 \quad (6.11)$$

and use BC's (6.6) & (6.7)

$$\Rightarrow u(0) = -c_2 = g; \quad \partial_x u(1) = -\frac{1}{2} - c_1 = h \quad (6.12)$$

$$\Rightarrow u(x) = -\frac{1}{6}x^3 + (h + \frac{1}{2})x + g \quad (6.13)$$

The analytical solutions (6.13) will be used in the numerical problem set to test the approximate solutions. For more complicated, realistic problems, typically no analytical solutions can be found (which is why we do numerical analysis in the first place).

From the strong form, we will now move to a variation, or “*weak form*”. We require that

- (a) the *trial solutions* of u , among all possible solutions satisfy the essential BC

$$u(0) = g \quad (6.14)$$

and that the trial solutions are square integrable

$$\int_0^1 (\partial_x u)^2 dx < \infty \quad (6.15)$$

- (b) the weighting functions, or variations, w satisfy $w(0) = 0$, the homogeneous counterpart of eq. (6.14).

We can then write (sloppily):

$$\partial_{xx}u + s = 0 \quad (6.16)$$

multiply by $-w$ and integrate

$$-\int w \partial_{xx}u - \int w s = 0 \quad (6.17)$$

from the integration by parts rule

$$\int a'b = ab \big| - \int b'a \quad (6.18)$$

$$\Rightarrow \int_0^1 \partial_x w \partial_x u - \partial_x u w \big|_0^1 - \int w s = 0 \quad (6.19)$$

with $\partial_x u(1) = h$ and $w(0) = 0$, this can be written as

$$\boxed{\int_0^1 \partial_x w \partial_x u = h w(1) + \int_0^1 w s dx} \quad (6.20)$$

This is the weak form of the PDE. Equations of this type in mechanics are called “virtual work”, or virtual displacement formulations (the w are the virtual displacements). It can be shown that the weak and the strong form are identical (*Hughes (2000)*, sec. 1.4) and the FE method proceeds from eq. (6.20) by assuming u and w can be taken from a simplified functional space, typically based on low order polynomials.

It is useful to define a shorthand notation

$$a(w, u) = \int_0^1 \partial_x w \partial_x u dx \quad \text{and} \quad (w, s) = \int_0^1 w s dx \quad (6.21)$$

Then, we can write eq. (6.20) as

$$\boxed{a(w, u) = (w, s) + h w(1)} \quad (6.22)$$

$a(., .)$ and $(., .)$ are *symmetric*

$$(w, s) = (s, w) \quad (6.23)$$

and *bilinear*

$$(c_1 u + c_2 v, w) = c_1 (u, w) + c_2 (v, w) \quad (6.24)$$

forms.

6.1.3 Galerkin method

If we consider a finite dimensional approximation of u and w on a FE *mesh*, \tilde{u} and \tilde{w} from a function space \mathcal{U} and \mathcal{W} , where $\tilde{u} \in \mathcal{U}$, $\tilde{w} \in \mathcal{W}$ such that

$$\tilde{u}(0) = g \quad \text{and} \quad \tilde{w}(0) = 0 \quad (6.25)$$

We can then construct a solution with $\tilde{v} \in \mathcal{W}$

$$\tilde{u} = \tilde{v} + \tilde{g} \quad (6.26)$$

where \tilde{g} is a given function such that $\tilde{g}(0) = g$, which satisfies the BCs because

$$\tilde{u}(0) = \tilde{v}(0) + \tilde{g}(0) = 0 + g \quad (6.27)$$

as $\tilde{v}(0) = 0$, since $\tilde{v} \in \mathcal{W}$. If we substitute eq. (6.27) into eq. (6.22), we get

$$a(\tilde{w}, \tilde{v}) = (\tilde{w}, s) + \tilde{w}(1)h - a(\tilde{w}, \tilde{g}) \quad (6.28)$$

where we solve for the LHS and the RHS is determined by BC's.

This is an example of a *weighted residual method*, there are other approaches such as the Petrov-Galerkin method. The Galerkin method is the simplest because it assumes that \tilde{v} and \tilde{w} are from the same function space, i.e. *the same shape functions (see below) are used for the solution \tilde{u} and the weights \tilde{w}* .

6.1.4 Discretization

Let's assume that there are n nodes such that we can write the weighting functions as

$$\tilde{w}(x) = \sum_{A=1}^n c_A N_A(x) \quad (6.29)$$

where the $N_A(x)$ are called *shape*, basis, or interpolation functions. We require

$$N_A(0) = 0 \quad \forall A \quad (6.30)$$

such that $\tilde{w}(0) = 0$ can be fulfilled. If we also introduce a shape function

$$\hat{N}_1(0) = 1 \quad (\text{with } \hat{N}_1 \notin \mathcal{W}) \quad (6.31)$$

then

$$\tilde{g} = g \hat{N}_1 \quad \text{so that } \tilde{g}(0) = g \quad (6.32)$$

We can then write

$$\begin{aligned} \tilde{u} &= \tilde{v} + \tilde{g} \\ &= \sum_{A=2}^n d_A N_A + g \hat{N}_1 \end{aligned} \quad (6.33)$$

such that $\tilde{u}(0) = g$. If we substitute eqs. (6.29) and (6.33) into eq. (6.28), then

$$a \left(\sum_A c_A N_A(1), \sum_B d_B N_B \right) = \left(\sum_A c_A N_{A,s} \right) + \left[\sum_A c_A N_A(1) \right] h - a \left(\sum_A c_A N_{A,g} \hat{N}_1 \right). \quad (6.34)$$

Because of bilinearity, we can write $\sum_A c_A G_A = 0$ with

$$G_A = \sum_B a(N_A, N_B) d_B - (N_A, s) - N_A(1) h + a(N_A, \hat{N}_1) g. \quad (6.35)$$

The Galerkin equation (6.28) is supposed to hold for all w , therefore all c_A , which mean that $a_A = 0$, so for all A

$$\boxed{\sum_B a(N_A, N_B) d_B = (N_A, s) + N_A(1) h - a(N_A, \hat{N}_1) g} \quad (6.36)$$

if we write $K_{AB} = a(N_A, N_B)$ and $F_A = (N_A, s) + N_A(1) h - a(N_A, \hat{N}_1) g$, then eq. (6.36) becomes

$$\boxed{\mathbf{K} \mathbf{d} = \mathbf{F}} \quad (6.37)$$

where \mathbf{K} is the *stiffness matrix*, \mathbf{d} is the *displacement vector*, and \mathbf{F} is the *force, or load, vector*. Once the $\mathbf{Kd} = \mathbf{F}$ system is assembled, one may solve for \mathbf{d} and then obtain the spatial solution from

$$\tilde{u}(x) = \sum_{A=2}^n d_A N_A(x) + g \hat{N}_1(x) \quad (6.38)$$

$$\text{or } \tilde{u}(x) = \sum_A^n d_A N_A(x) \quad \text{with } d_1 = g. \quad (6.39)$$

Note that \mathbf{K} is symmetric,

$$\mathbf{K} = K_{AB} = a(N_A, N_B) \quad (6.40)$$

$$= a(N_B, N_A) = K_{BA} = \mathbf{K}^T \quad (6.41)$$

which facilitates computations.

6.2 A 1-D FE example

We now provide a numerical implementation of the 1-D FE example of the previous lecture. We subdivide the $[0, 1]$ interval into n subintervals (“*elements*”) delimited by $n + 1$ *nodes* or nodal points such that $x_1 = 0$ and $x_{n+1} = 1$. The subintervals are denoted by

$$[x_A, x_{A+1}] \text{ with } h_A = x_{A+1} - x_A \quad (6.42)$$

where h_A may vary and a general grid spacing may be defined as $h = \max(h_A)$.

We can then choose interior shape functions for $2 \leq A \leq n$ as

$$N_A(x) = \begin{cases} \frac{1}{h_{A-1}}(x - x_{A-1}) & \text{for } x_{A-1} \leq x \leq x_A, \\ \frac{1}{h_A}(x_{A+1} - x) & \text{for } x_A \leq x \leq x_{A+1}, \\ 0 & \text{otherwise.} \end{cases} \quad (6.43)$$

For the boundaries, we use special shape functions

$$\hat{N}_1(x) = \frac{1}{h_1}(x_2 - x) \quad \text{for } x_1 \leq x \leq x_{n+1} \quad \text{and} \quad (6.44)$$

$$\hat{N}_{n+1}(x) = \frac{1}{h_n}(x - x_n) \quad \text{for } x_n \leq x \leq x_{n+1}. \quad (6.45)$$

An illustration of interior and boundary shape functions is shown in Figure 6.1; note that $N_A = 1$ at $x = x_A$ and zero for other nodes.

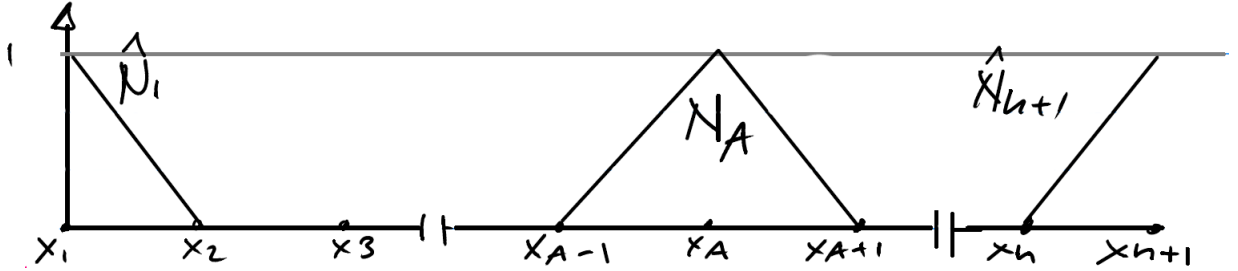


Figure 6.1: Example of 1-D shape functions

With this choice, the shape functions are zero outside the vicinity of A . They have *local support*, which means that K is sparse because the $a(N_B, N_A) = \int_0^1 \partial_x N_A \partial_x N_B dx$ integral is zero for $B > A + 1$. The K matrix is *banded*, and the bandwidth depends on how the nodes are numbered (leading to an optimization problem during mesh design) and what basis functions are used. Besides symmetry and bandedness, K is also *positive definite*, which means that

$$\mathbf{c}^T K \mathbf{c} \geq 0 \quad (6.46)$$

for all \mathbf{c} such that $\mathbf{c}^T K \mathbf{c} = 0 \Rightarrow \mathbf{c} = 0$. These properties allow efficient solution of $K \mathbf{d} = \mathbf{F}$.

6.2.1 Local vs. global points of view

It is useful to compare the $(.,.)$ and $a(.,.)$ operations in local coordinate systems that are referenced to each element as is shown below in Figure 6.2.

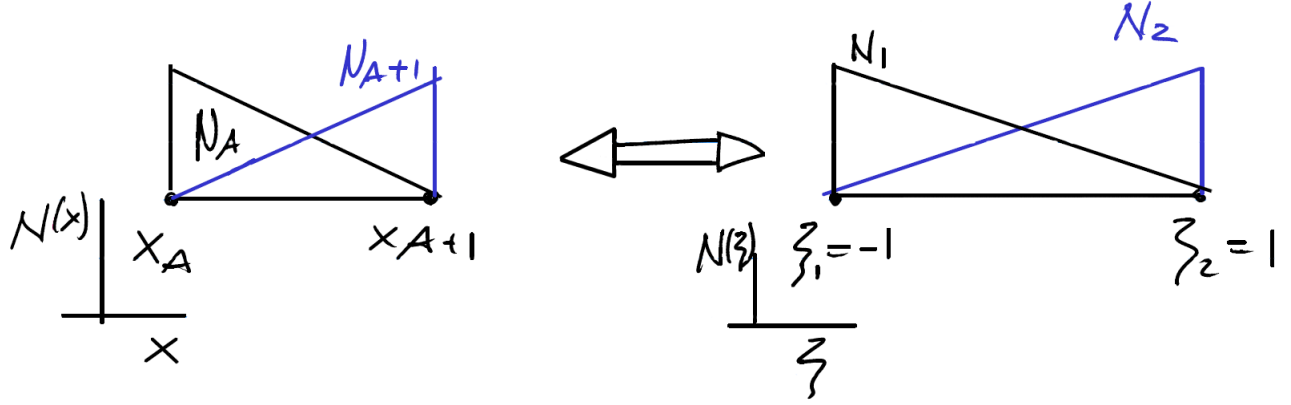


Figure 6.2: Example of 1-D shape functions in global (left) and element-local (right) coordinates

$$x(\xi) = \frac{1}{2}(h_A \xi + x_A + x_{A+1}) \Leftrightarrow \xi = \frac{1}{h_A}(2x - x_A - x_{A+1}) \quad (6.47)$$

$$u(x) = \sum d_A N_A(x) \Leftrightarrow u(\xi) = N_1(\xi) d_1 + N_2(\xi) d_2. \quad (6.48)$$

We can express the global shape function N_A of eq. (6.43) in a local coordinate system as

$$N_a(\xi) = \frac{1}{2}(1 + \xi_a \xi) \quad \text{for } a = 1, 2 \quad (6.49)$$

$$(i.e. N_1(\xi) = \frac{1}{2}(1 - \xi); \quad N_2(\xi) = \frac{1}{2}(1 + \xi) \quad \text{with } \xi = \{-1, 1\}). \quad (6.50)$$

Likewise, we can express the global coordinate within the element as

$$x^e(\xi) = \sum_{a=1}^2 N_a(\xi) x_a^e \quad (6.51)$$

where x_a^e are the global nodes that belong to the element e . For the assembly of the stiffness matrix, derivatives of N_a and x^e with respect to ξ are required. We note that

$$\partial_\xi N_a = \frac{\xi_a}{2} = \frac{(-1)^a}{2} \quad \text{and} \quad \partial_\xi x = \frac{h^e}{2} \quad (6.52)$$

$$\partial_x \xi = (\partial_\xi x^e)^{-1} = \frac{2}{h^e} \quad (6.53)$$

where for higher dimensions the term $(\partial_\xi x^e)^{-1}$ will be a matrix inverse.

Note: The choice of shape function is determined by the element. E.g. with a two node element, shape functions can only be linear.

6.2.2 Matrix assembly

With n elements, we therefore have *globally*

$$\mathbf{K} = [K_{AB}] \quad \text{an } n \times n \text{ matrix and} \quad \mathbf{F} = \{F_A\} \quad \text{an } n \times 1 \text{ vector} \quad (6.54)$$

where (from last lecture)

$$K_{AB} = a(N_A, N_B) = \int_0^1 \partial_x N_A \partial_x N_B \, dx \quad (6.55)$$

$$F_A = (N_A, s) + h \delta_{A,n+1} - a(N_A, \hat{N}_1) g \quad (6.56)$$

$$= \int_0^1 N_A s \, dx + \delta_{A,n+1} h - g \int_0^1 \partial_x N_A \partial_x \hat{N}_1 \, dx \quad (6.57)$$

where $N_A(x_{n+1}) = \delta_{A,n+1}$ is assumed. The integrals over the problem domain $[0, 1]$ can be written as summations over elements, therefore

$$\mathbf{K} = \sum_{e=1}^n \mathbf{K}^e \quad \text{with} \quad \mathbf{K}^e = [K_{AB}^e] \quad (6.58)$$

$$(6.59)$$

$$\mathbf{F} = \sum_e \mathbf{f}^e \quad \text{with} \quad \mathbf{F}^e = \{F_A^e\} \quad (6.60)$$

$$K_{AB}^e = \int d\Omega^e \partial_x N_A \partial_x N_B \, dx \quad (6.61)$$

$$F_A^e = \int_{\Omega^e} N_A s \, dx + h \delta_{e,n} \delta_{A,n+1} - g \int_{\Omega^e} \partial_x N_A \partial_x \hat{N}_1 \, dx \quad (6.62)$$

where the element domain $\Omega^e = [x_1^e, x_2^e]$.

Since the N_A only have local support $K_{AB}^e = 0$ if $(A \neq e \text{ or } e+1)$ or $(B \neq e \text{ or } e+1)$ and $F_A^e = 0$ if $A \neq e \text{ or } e+1$, and we can obtain the global stiffness matrix and force vector by summing up elemental contributions \mathbf{K}^e and \mathbf{f}^e

$$\mathbf{K}^e = [K_{ab}] \text{ a } 2 \times 2 \text{ matrix, } \mathbf{f}^e = \{f_a\} \text{ a } 2 \times 1 \text{ vector} \quad (6.63)$$

(2 is the number of nodes per element!)

$$K_{ab} = a(N_a, N_b)^e = \int_{\Omega^e} \partial_x N_a \partial_x N_b \, dx \quad (6.64)$$

$$(6.65)$$

$$f_a = \int_{\Omega^e} N_a s \, dx + \begin{cases} K_{a,1}^e g & \text{for } e = 1, \\ 0 & \text{for } e = 2, \dots, n-1, \\ \delta_{a,n+1} h & \text{for } e = n. \end{cases} \quad (6.66)$$

The assembly proceeds as symbolized in Figure 6.3, and placing the \mathbf{K}^e element-local matrix into the global stiffness matrix requires use of an assignment operator or array. This is discussed in the worked example of the problem set.

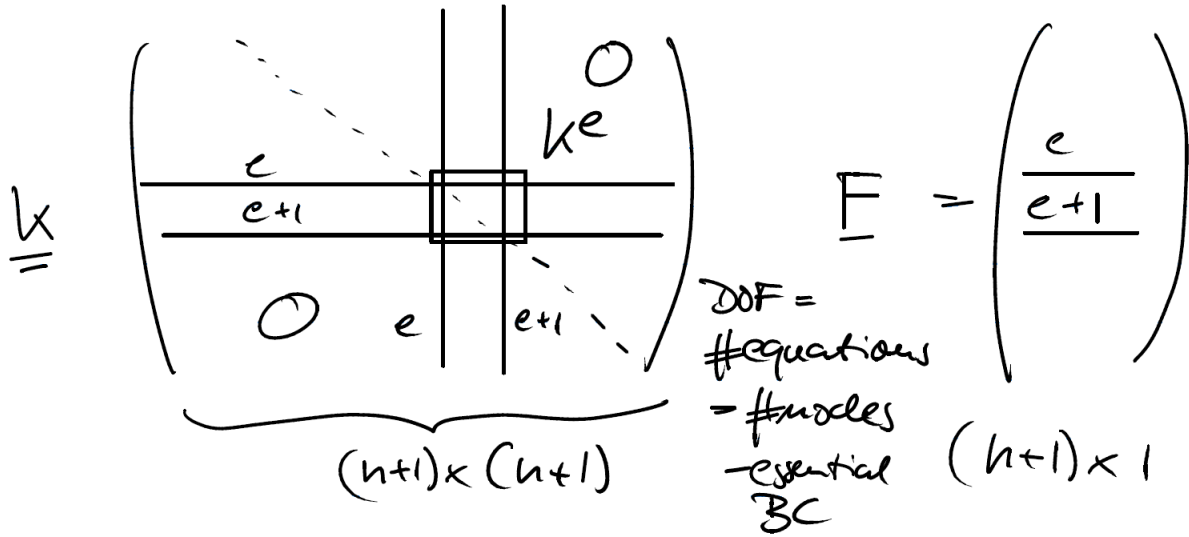


Figure 6.3: Visual example of stiffness matrix assembly process

6.2.3 Element-local computations

We wish to perform integrations in a local coordinate system. If the original interval $x \in [x_1, x_2]$ is smoothly mapped into $\xi \in [\xi_1, \xi_2]$, there exists a *change of variables* such that

$$\int_{x_1}^{x_2} dx f(x) = \int_{\xi_1}^{\xi(x_2)} d\xi [\partial_\xi x(\xi)] f(x(\xi)) = \int_{\xi_1}^{\xi_2} d\xi \partial_\xi x f(\xi). \quad (6.67)$$

By means of the chain rule we have

$$\partial_\xi f(x(\xi)) = \partial_x f(x(\xi)) \partial_\xi x(\xi). \quad (6.68)$$

Therefore, we can compute

$$\begin{aligned} K_{ab}^e &= \int_{\Omega^e} \partial_x N_a(x) \partial_x N_b(x) dx \\ &= \int_{-1}^1 \partial_x N_a(x(\xi)) \partial_x N_b(x(\xi)) \partial_\xi x(\xi) d\xi \end{aligned} \quad (6.69)$$

using the change of variable described above. Then using the chain rule we have

$$\partial_\xi N_a(x(\xi)) = \partial_x N_a(x(\xi)) \partial_\xi x(\xi) \quad \Rightarrow \quad \partial_x N_a(x) = (\partial_\xi x)^{-1} \partial_\xi N_a(\xi) \quad (6.70)$$

Then, plugging result back into eq. (6.69)

$$K_{ab}^e = \int_{-1}^1 (\partial_{\xi} x)^{-2} \partial_{\xi} N_a(\xi) \partial_{\xi} N_b(\xi) \partial_{\xi} x(\xi) d\xi \quad (6.71)$$

$$= \int_{-1}^1 (\partial_{\xi} x)^{-1} \partial_{\xi} N_a(\xi) \partial_{\xi} N_b(\xi) d\xi \quad (6.72)$$

$$= \frac{1}{h^e} (-1)^{a+b} \quad (\text{see above}) \quad (6.73)$$

$$(6.74)$$

$$\Rightarrow K^e = \frac{1}{h^e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \quad (6.75)$$

Since $\partial_{\xi} N_a$ is independent of element data, the computation only has to be performed once, derivatives $\partial_{\xi} x$ and $\partial_x \xi$ depend on the element shape and need to be computed for each configuration. For the source term, we use the approximation

$$\tilde{s} = \sum_{a=1}^2 s_a N_a \quad (6.76)$$

i.e. the source term is assumed to vary linearly across the element with N_a . Then we can write

$$\int_{\Omega^e} N_a(x) \tilde{s}(x) dx = \int_{-1}^1 N_a(x(\xi)) \tilde{s}(x(\xi)) \partial_{\xi} x(\xi) d\xi \quad (6.77)$$

$$= \frac{h^e}{2} \sum_{b=1}^2 \int_{-1}^1 N_a(\xi) N_b(\xi) d\xi s_b. \quad (6.78)$$

Since $\int_{-1}^1 N_a N_b = \frac{1}{3}(1 + \delta_{ab})$,

$$s^e = \frac{h^e}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \text{boundary terms} \quad (6.79)$$

$$(6.80)$$

$$= \frac{h^e}{6} \begin{bmatrix} 2 s_1 + s_2 \\ s_1 + 2 s_2 \end{bmatrix} + \text{boundary terms} \quad (6.81)$$

6.3 Exercise: 1-D heat conduction and solution of linear systems

Reading

This finite element example is based on

- [Hughes \(2000\)](#), sec. 1.1-1.15

and multigrid solver part based on an exercise by [Zhong \(2008\)](#).

6.3.1 Implementation of 1-D heat equation example

In the previous two lectures, we considered the example PDE

$$\partial_{xx}u + s = 0 \quad (6.82)$$

on the domain $x \in [0, 1]$, $u(x)$, $s(x)$, and subject to essential (Dirichlet) boundary condition $u(0) = g$ on the left, and natural (Neumann) BCs, $\partial_x u(1) = h$ on the right (here, $\partial_x = \frac{\partial}{\partial x}$). Equation (6.82) may be considered as a simplified version of the steady-state heat equation

$$\partial_{xx}T + H = 0 \quad (6.83)$$

with sources $s = H$.

You should consult your lecture notes for details (see sec. 6.2), but in brief: If we have n elements between $n + 1$ global nodes, the weak form of eq. (6.82) can be written for each global node A as

$$\sum_B a(N_A, N_B) d_B = (N_A, s) + N_A(1)h - a(N_A, \hat{N}_1)g. \quad (6.84)$$

Here, N_A are the shape functions in the interior, B is another global node, and \hat{N}_1 the boundary shape function for the essential boundary condition g . This can be further abbreviated by

$$K_{AB} = a(N_A, N_B) = \int_0^1 \partial_x N_A \partial_x N_B dx \quad (6.85)$$

$$F_A = (N_A, s) + N_A(1)h - a(N_A, \hat{N}_1)g = \int_0^1 N_A s dx + \delta_{A,n+1}h - \left(\int_0^1 \partial_x N_A \partial_x \hat{N}_1 dx \right) g \quad (6.86)$$

where we have used the definitions of the bi-linear forms $a(\cdot, \cdot)$ and (\cdot, \cdot) from lecture, and the Kronecker delta

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases} \quad (6.87)$$

is used for the flux boundary condition (also see [Hughes, 2000](#), chap 1).

The approximate solution of u after discretization of the weak form is given by

$$\tilde{u}(x) = \sum_{A=2}^{n+1} d_A N_A(x) + \hat{N}_1 g = \sum d_A N_A(x), \quad (6.88)$$

where the latter summation implies choosing the boundary shape function and BC if needed. The vector $\mathbf{d} = \{d_A\}$ values have to be obtained by solution of the matrix equation

$$\mathbf{K}\mathbf{d} = \mathbf{F}, \quad (6.89)$$

with $\mathbf{K} = \{K_{AB}\}$ and $\mathbf{F} = \{F_A\}$.

We discussed in lecture how the integration over the domain can be broken down into summation over integrals over each element (see sec. 4.2). This integration is most easily performed in a local coordinate system $-1 \leq \xi \leq 1$ between the two nodes of each element, which has a mapping to the corresponding, global coordinate interval $[x_A, x_{A+1}]$. We can also express the shape functions as $x(\xi)$,

$$x(\xi) = \sum_A N_A(\xi)x_A \quad \text{and} \quad u(\xi) = \sum N_A(\xi)d_A. \quad (6.90)$$

The global \mathbf{K} matrix and the \mathbf{F} vector are then assembled by looping over all elements $1 \leq e \leq n$ and adding each element's contribution. By change of integration variables and the chain rule, those elemental contributions follow as

$$\mathbf{k}^e = \frac{1}{\Delta x} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad (6.91)$$

where Δx is the element size, $x_{A+1} - x_A$, and for the force term

$$\mathbf{f}^e = \frac{\Delta x}{6} \begin{pmatrix} 2s_1 + s_2 \\ s_1 + 2s_2 \end{pmatrix} + \begin{cases} k_{a1g} & \text{for } e = 1 \\ \delta_{a,n+1} & \text{for } e = n \\ 0 & \text{else} \end{cases} \quad (6.92)$$

where we have assumed that the source function s varies linearly over each element, and s_1 and s_2 are the contributions from each local node a within the element from $s(x)$. After assembly, one needs to ensure that each row of the global \mathbf{K} matrix that corresponds to a fixed value (Dirichlet) boundary condition, will only have a diagonal entry and the other columns for this row are zero.

6.3.2 Exercises

- (a) Download `heat1dfe.m`, and all helper routines for this week from the course web site. Read through the implementation of what is summarized above, `heat1dfe.m`, and understand this code.
- (b) Fill in the blanks in `heat1dfe.m` and experiment with a solution of eq. (6.82) for $n = 3$ elements.
 - (a) Print out the stiffness matrix (`full(stiffness)`) to appreciate its banded structure. Does this look familiar to you?
 - (b) Choose the Matlab solver (`solver=0`) and plot the finite element solution at the nodes, as interpolated within the elements, and compare with the analytical solution.

6.3.3 Solution of large, sparse linear systems of equations

The finite element method quickly leads to very large, linear systems of equations

$$\mathbf{K}\mathbf{d} = \mathbf{F} \quad (6.93)$$

whose solution can be quite involved. Ideally, we would hand off the solution of eq. (6.93) to a computational scientist and use a “black box” solver. However, practice shows that the nature of the physical problem and the best solution method are often intertwined. Choosing a different solver might also allow addressing larger, *e.g.* 3-D, problems because of improved efficiency. Moreover, it is very hard to make solvers bullet-proof and one often encounters problematic (*e.g.* unstable, or no convergence) performance in practice. Linear systems of equations also arise in other fields of geophysics, and some exposure to computational linear algebra is needed to understand the MILAMIN (*Dabrowski et al., 2008*) finite element implementation which we will use later. We therefore digress a bit here. If your research has you deal with matrices a lot, *Golub and Van Loan (1996)* is a classic numerical linear algebra text that might come in handy.

Direct solvers

For the finite element method, we can always write our problem in the form of eq. (6.93), where \mathbf{K} is a square, $n \times n$ matrix. A general strategy to solve eq. (6.93) is then *LU* decomposition

$$\mathbf{K} = \mathbf{LU}, \quad (6.94)$$

where \mathbf{L} and \mathbf{U} are lower and upper triangular matrices, respectively, which only have zeros in the other part of the matrix. The solution of eq. (6.93) can then be obtained efficiently from

$$\mathbf{K}\mathbf{d} = \mathbf{LU}\mathbf{d} = \mathbf{F} \quad (6.95)$$

by solving for $\mathbf{y} = \mathbf{L}^{-1}\mathbf{F}$ and then $\mathbf{d} = \mathbf{U}^{-1}\mathbf{y}$, because the inverse of \mathbf{U} and \mathbf{L} are computationally fast to obtain. *LU* is often how general matrix inversion is implemented on a computer.

For most FE problems, the \mathbf{K} matrix will also be sparse and banded. Special algorithms exist to exploit this feature such that the run time is ideally dominated by the number of non-zero entries of \mathbf{K} , rather than the full size. Moreover, if \mathbf{K} is symmetric and positive definite, as in our example above, we can use the Cholesky decomposition for which $\mathbf{U} = \mathbf{L}^T$ and computations are twice as fast as for the general *LU* case. However, for complex, 3-D FE problems, current computational limitations often prohibit the use of direct solvers which is why iterative methods which do not require matrix decomposition or inversion, are used.

Note: Symmetry means $\mathbf{K} = \mathbf{K}^T$, where \mathbf{K}^T is the transpose, $K_{ij}^T = K_{ji}$. Positive definite means that $\mathbf{c}^T \mathbf{K} \mathbf{c} > 0$ for any non-zero \mathbf{c} . Graphically, this corresponds for a 2×2 matrix to a well defined minimum (lowest) point in a curved landscape, which is important for iterative methods (*e.g.* *Shewchuk, 1994*). Positive definite, symmetric matrices also arise in least-squares problems in geophysical inversions (*e.g.* seismic tomography, see for example *Boschi and Dziewoński, 1999*, for a nice introduction to linear algebra in this framework). Least-squares means that we wish to solve

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (6.96)$$

in the sense that $\|\mathbf{Ax} - \mathbf{b}\| = \min$, *i.e.* deviations from the true solution are minimized, for a matrix \mathbf{A} that may be under-determined, *i.e.* not simply invertible. It can be shown that the least squares solution \mathbf{x}_{LS} is given by

$$\mathbf{x}_{LS} = (\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot \mathbf{A}^T \cdot \mathbf{d}, \quad (6.97)$$

where $(\mathbf{A}^T \cdot \mathbf{A})^{-1}$ is the generalized inverse (which exists even if the inverse of \mathbf{A} , \mathbf{A}^{-1} , does not exist because \mathbf{A} is singular). $\mathbf{A}^T \cdot \mathbf{A}$ is symmetric and positive definite, meaning that Cholesky is also the method of choice for direct approaches to find \mathbf{x}_{LS} .

Iterative solvers

Jacobi method The simplest iterative solution of eq. (6.93) is given by the Jacobi method. If \mathbf{K} is LU decomposed and we write the diagonal matrix (only non-zero along diagonal) of \mathbf{K} as \mathbf{D} , then an iterative solution for \mathbf{d} starting from an initial guess \mathbf{d}^1 (*e.g.* $\mathbf{0}$) can be obtained from

$$\mathbf{D}\mathbf{d}^{i+1} = \mathbf{F} - (\mathbf{U} + \mathbf{L})\mathbf{d}^i, \quad (6.98)$$

where the iteration is over i and is stopped once \mathbf{d}^{i+1} is not changing more than some tolerance from the previous solution estimate \mathbf{d}^i . On an element by element basis, this can be written as

$$d_j^{i+1} = \frac{1}{K_{jj}} \left(F_j - \sum_{l=1, l \neq j}^n K_{jl} d_l^i \right) \quad (6.99)$$

where the summation is over all l but for $l = j$. The Jacobi method following eq. (6.99) is implemented in `jacobi.m`. It serves mainly illustrative purposes but is guaranteed to converge, albeit slowly (see below), if \mathbf{K} is “diagonally dominant” which is satisfied strictly when the absolute value of the diagonal elements is larger than the sum of the absolute values of each row.

Gauss-Seidel An improvement over the Jacobi method is the Gauss-Seidel (GS) approach, where the iterative rule is

$$(\mathbf{D} + \mathbf{K})\mathbf{d}^{i+1} = \mathbf{F} - \mathbf{U}\mathbf{d}^i. \quad (6.100)$$

The main benefit is that \mathbf{d}^{i+1} can be computed from \mathbf{d}^i directly, without having to store a full previous solution, following

$$d_j^{i+1} = \frac{1}{K_{jj}} \left(F_j - \sum_{l < j} K_{jl} d_l^{i+1} - \sum_{l > j} K_{jl} d_l^i \right). \quad (6.101)$$

Note that this operation can be done “in place”, and is implemented in `gauss_seidel.m`. The GS method will converge (somewhat faster than the Jacobi method) for diagonally dominant, or positive definite and symmetric \mathbf{K} .

Successive Over Relaxation (SOR) Successive Over Relaxation is a more general variant of the Gauss-Seidel method that can lead to faster convergence. This is obtained by adding a parameter w which determines the weight of the current solution in the weighted average used to compute the next solution.

$$d_j^{i+1} = (1 - w) d_j^i + \frac{w}{K_{jj}} \left(F_j - \sum_{l < j} K_{jl} d_l^{i+1} - \sum_{l > j} K_{jl} d_l^i \right) \quad (6.102)$$

Setting $w = 1$ will reduce SOR to the GS method. The optimal value of w is dependent upon the matrix K , but setting $w = 0.5$ is a good starting point. The method has been rigorously shown to converge for symmetric, positive definite matrices K for $0 < w < 2$.

Exercise

- Plot the Jacobi, GS, and SOR solutions for 32 elements and a tolerance of 10^{-4} , 10^{-5} , and 10^{-6} on one plot each; comment on the accuracy and number of iterations required. Can you improve the definition of tolerance for the Jacobi method?

Choose a tolerance of 10^{-6} , and record the number of iterations required to solve the 1-D FE example problem using the Jacobi and GS methods for increasing number of elements, *e.g.* for 8, 16, 32, 64, and 128 elements. (You might want to automate these computations and not wait until convergence and record the results by hand.) Plot the number of iterations against number of elements for both methods. Comment on the “scaling” of iteration numbers with size.

Conjugate gradient You have now seen that while the Gauss-Seidel method is an improvement on the Jacobi approach, it still requires a large number of iterations to converge. This makes both methods impractical in real applications and other approaches are commonly used. One of those is the conjugate gradient (CG) method which works for positive-definite, symmetric, square ($n \times n$) matrices. The CG method is explained in a nice, geometric fashion by [Shewchuk \(1994\)](#). We cannot explore the motivation behind CGs in detail, but `conjgrad.m` provides a pretty straightforward Matlab implementation which you should check out.

The CG method provides an exact solution after n iterations, which is often a prohibitively large number for real systems, and approximate solutions may sometimes be reached for a significantly smaller number of iterations. There are numerous tweaks involving modifications to the conjugate gradient method that pertain to “pre-conditioners” where we solve

$$M^{-1}K\mathbf{d} = M^{-1}\mathbf{F}, \quad (6.103)$$

for some M which approximates K but is simpler to handle than K . The best choice of these is, for some applications, an active area of research. For sparse least-squares problems, such as for seismic tomography, the LSQR approach of [Paige and Saunders \(1982\)](#) is a popular choice that is used by most researchers at present for linear inversions.

Exercise

- Switch the solver from the GS method to conjugate gradient and increase the maximum iteration number stepwise from a fraction of n to the full n (as determined by the number of elements which you should choose large, *e.g.* 200, for this exercise). Test different initial guesses for \mathbf{d}^i (*e.g.* all zero, random numbers), record the convergence and comment on the solution.

Multigrid An interesting philosophy to solving PDEs of the type we are considering for the 1-D finite element example is by using several layers of variable resolution grids (*e.g.* [Press et al., 1993](#), sec. 19.6). The insight is based on the observation that the Gauss-Seidel method is very good at reducing short-wavelength residuals in the iterative solution for \mathbf{d} (“smoothing”), but it takes a long time to reduce the largest wavelength components of the residual. (You should try to plot successive solutions of the GS method compared to the analytical solution for different starting \mathbf{d}_0 to visualize this behavior.)

For the multigrid (MG) method, the idea is to solve the equations to within the desired tolerance only at a very coarse spatial discretization, where only a few iterations are required. Then, the solution is interpolated up to finer and finer levels where only a few GS iterations are performed at each level to smooth the solution. One then cycles back and forth until convergence is achieved at the finest, true solution level. There are several different approaches that are all called “multigrid” methods and basically only share the same philosophy. Differences are, for example found in terms of the way the cycling between fine and course resolutions are conducted (*e.g.* [Briggs et al., 2000](#)), and we will only discuss the “V cycle” method. Multigrid methods are now implemented in most 3-D finite element methods ([Zhong et al., 2007](#)) because MG has, ideally, the perfect scaling of $O(N)$ where N is the size of the problem. MG methods areas of active research (*e.g.* algebraic multigrid, which is related to adaptive mesh refinement).

The multigrid method is based on expressing the PDE on L MG levels of resolution where the number of nodes in each level, n_i , is given by

$$n_i = b \times 2^{i-1} + 1 \quad \text{for } i = 1, 2, \dots, L, \quad (6.104)$$

where b is the base, typically a small number such as 2 or 4. At each i^{th} level, we need to construct separate stiffness matrices, K_i , and the corresponding force vector where the resolution for the $i = L$ solution is the best approximation to $K\mathbf{d} = \mathbf{F}$, and the forcing is only needed to be specified at \mathbf{F}_L (see below).

An example implementation may proceed like so (see, *e.g.* [Press et al., 1993](#), sec. 19.6 for some alternatives): We start at the highest level, L , and perform only a few, fixed number of GS iterations for an rough approximate \mathbf{d}_L from

$$K_L \mathbf{d}_L = \mathbf{F}_L \quad (6.105)$$

to remove the short wavelength misfit starting from an initial guess $\mathbf{d}_L = \mathbf{0}$. The residual is then given by

$$\mathbf{R}_L = \mathbf{F}_L - K_L \mathbf{d}_L. \quad (6.106)$$

We then *project*, or restrict, the residual to a coarser grid at $L - 1$ by a projection operator P

$$\mathbf{R}_{L-1} = P_{L \rightarrow L-1} \mathbf{R}_L. \quad (6.107)$$

P will be some stencil giving more weight to the fine resolution nodes that are closer to the coarse resolution node to which we project. We next GS iterate

$$\mathbf{K}_i \delta \mathbf{d}_i = \mathbf{R}_i \quad (6.108)$$

for $i = L - 1$ for another small number of iterations (initializing \mathbf{d}_i again with $\mathbf{0}$), performing another “smoothing” step, reducing short wavelength fluctuations. Note that eq. (6.108) now operates on the residual and not the load vector \mathbf{F} such that we are computing corrections of \mathbf{d} , $\delta \mathbf{d}$. We then repeat the smoothing and projection steps down to $i = 1$ where eq. (6.108) can be solved quickly and exactly. This completes the downward leg of the V cycle where the longest wavelength residual has been addressed.

Next, we have to propagate the correction $\delta \mathbf{d}_1$ from $i = 1$ to $i = 2$ and higher resolutions by means of a “prolongation”, *i.e.* an interpolation to higher resolution by an interpolation operator I

$$\delta \mathbf{d}_{i+1} = I_{i \rightarrow i+1} \delta \mathbf{d}_i. \quad (6.109)$$

I may be a linear interpolation, for example, which is easy to compute for the mesh structure eq. (6.104). This upward interpolated $\delta \mathbf{d}_{i+1}$ can then be smoothed by using it as a starting guess for a fixed number of GS iterations for

$$\mathbf{K}_{i+1} \delta \mathbf{d}_{i+1} = \mathbf{R}_{i+1} \quad (6.110)$$

with $\delta \mathbf{d}_{i+1}$. We can now correct

$$\delta \mathbf{d}_{i+1} = \delta \mathbf{d}_{i+1} - \alpha \delta \mathbf{d}_{i+1} \quad (6.111)$$

$$\mathbf{R}_{i+1} = \mathbf{R}_{i+1} - \alpha \delta \mathbf{R}_{i+1}, \quad (6.112)$$

$$(6.113)$$

with $\delta \mathbf{R}_{i+1} = -\mathbf{K}_{i+1} \delta \mathbf{d}_{i+1}$ and weighting $\alpha = (\delta \mathbf{R}_{i+1} \cdot \mathbf{R}_{i+1}) / |\delta \mathbf{R}_{i+1}|^2$. We continue by projecting $\delta \mathbf{d}_i$ in this fashion up to $i = L$, where we update $\mathbf{d}_L = \mathbf{d}_L + \delta \mathbf{d}_L$, which completes the upward leg of the V. The whole V cycle is then repeated until the desired tolerance for \mathbf{d}_L is reached at which point $\mathbf{d}_L = \mathbf{d}$. Details of the implementations of the MG method, such as the smoothing, restriction, and prolongation operations, depend on the problem and the boundary conditions (*e.g.* [Press et al., 1993](#); [Briggs et al., 2000](#)).

Exercise

- Download the MG implementation of the 1-D FE example (based on C code by [Zhong, 2008](#)), `multigrid.m`. Read through the implementation, compare with the above recipe, and understand the approach. Compare the number of iterations needed for the MG solver with that of the GS method for 32, 64, 128, 256 numbers of elements. Plot the scaling of the number of iterations, or time spent in the multigrid subroutine, with the number of elements.

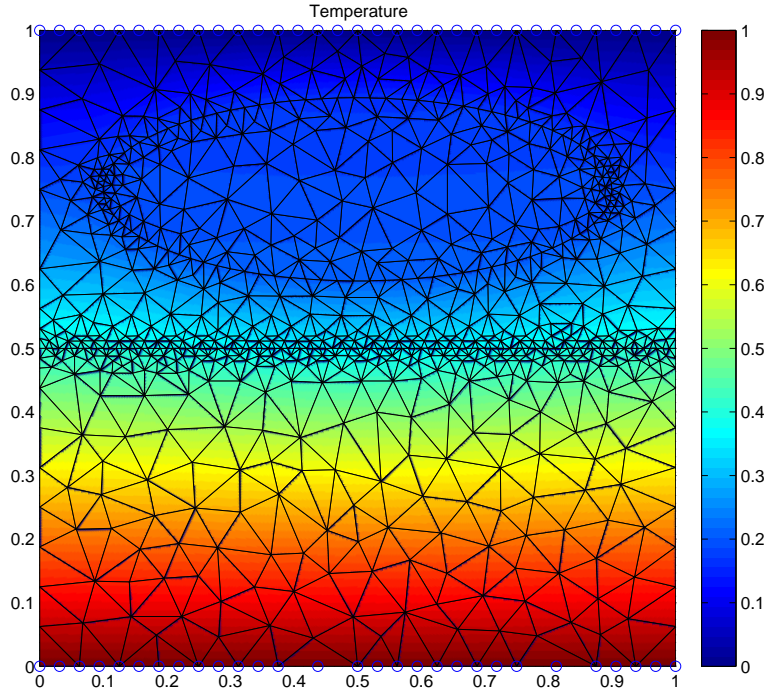


Figure 6.4: Coarse finite element mesh with solution for temperature, allowing for an elliptical inclusion and a boundary at mid-model height.

6.4 Two-Dimensional boundary value problems with FE

Reading: *Hughes (2000)* secs. 2.1 - 2.6, 3.1, 3.4, 3.8 - 3.9
eqs1cm]

We will now consider the solution of 2D boundary value problems using finite elements, which can be easily expanded to three dimensions. We write $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x \\ z \end{pmatrix}$ for a location vector x_i with $i = 1, 2$ and a normal vector \mathbf{n} on the boundary Γ of the domain Ω . As an example problem we will now revisit the linear heat conduction problem.

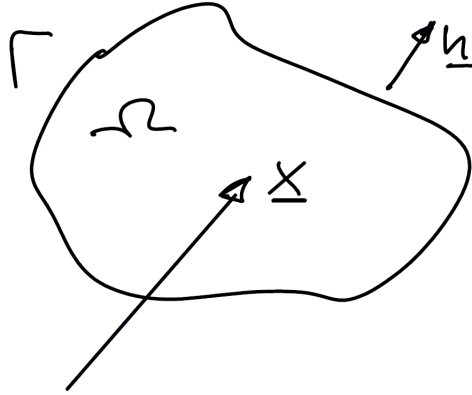


Figure 6.5: Visual representation of domain, vector, and normal

6.4.1 Linear heat conduction

If we allow for anisotropic diffusivity (which may apply to the oceanic plates), Fick's Law can be written as

$$q_i = -\kappa_{ij} \partial_j T = - \sum_{j=1}^2 \kappa_{ij} \frac{\partial T}{\partial x_j} \quad (6.114)$$

where repeated indices imply summation, \mathbf{q} is heat flux, $\kappa_{ij} = \kappa_{ji}$ is the conductivity matrix (we use k normally for diffusivity but we wish to distinguish it from the stiffness matrix), and T is temperature. In vector notation,

$$\boxed{\mathbf{q} = -\kappa \cdot \nabla T} \quad \kappa = [\kappa_{ij}] \quad (6.115)$$

and, usually, for the isotropic case

$$\kappa_{ij} = \kappa(\mathbf{x}) \delta_{ij} \quad (6.116)$$

is assumed. In steady-state, the energy equation is

$$\boxed{\nabla \cdot \mathbf{q} = H} \quad (6.117)$$

$$\text{or } \partial_i q_i = \partial_i \kappa_{ij} \partial_j T = H \quad \text{for } \Omega \quad (\text{Poisson Eq.})$$

with BCs:

$$\begin{aligned} T &= g & \text{on } \Gamma_g & \quad \text{and} \\ -q_i n_i &= h & \text{on } \Gamma_h \end{aligned} \quad (6.118)$$

where Γ_g and Γ_h are the parts of the boundary where fixed temperature or fixed flux conditions apply, respectively. For isotropy, we recover

$$\partial_i \delta_{ij} \kappa \partial_j T = \kappa \partial_i \partial_i T = H \quad (6.119)$$

$$\text{conductivity} \rightarrow \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2} \right) = H. \quad (6.120)$$

The weak form representation of eq. (6.118) is given by

$$-\int_{\Omega} d\Omega \partial_i w q_i = \int_{\Omega} d\Omega w H + \int_{\Gamma_h} d\Gamma w h \quad (6.121)$$

where the LHS is the diffusion term, the first term on the RHS corresponds to volumetric heating, and the second term on the RHS is the flux through the boundary. See [Hughes \(2000\)](#), sec. 2.3 for the derivation. Eq. (6.121) can then be expressed as

$$\boxed{a(w, T) = (w, f) + (w, h)_{\Gamma}} \quad (6.122)$$

with

$$\begin{aligned} a(w, T) &= \int_{\Omega} d\Omega \partial_i w \kappa_{ij} \partial_j T \\ (w, f) &= \int_{\Omega} d\Omega w H && \text{(area integral over } \Omega) \\ (w, h)_{\Gamma} &= \int_{\Gamma} d\Gamma w h. && \text{(line integral over } \Gamma) \end{aligned}$$

It is convenient to use vector/matrix notation. $\partial_i w \kappa_{ij} \partial_j T$ can then be written as

$$(\nabla \mathbf{w})^T \kappa \nabla T \quad \text{with} \quad \nabla \mathbf{w} = \begin{pmatrix} \partial_1 w \\ \partial_2 w \end{pmatrix} \quad \text{and} \quad \nabla T = \begin{pmatrix} \partial_1 T \\ \partial_2 T \end{pmatrix} \quad (6.123)$$

such that

$$a(w, T) = \int_{\Omega} d\Omega (\nabla \mathbf{w})^T \kappa \nabla T \quad (6.124)$$

with $\kappa = \kappa \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ for isotropy in 2-D.

Using the Galerkin approach of choosing the trial and weighting functions from the same function space, we again posit for the solution

$$\tilde{T} = \tilde{v} + \tilde{g} \quad (6.125)$$

where $\tilde{v} = 0$ on Γ_g and \tilde{g} allows satisfying the Dirichlet BCs with $\tilde{T} = \tilde{g}$ on Γ_g .

The weak form becomes

$$a(\tilde{w}, \tilde{v}) = (\tilde{w}, H) + (\tilde{w}, h)_{\Gamma_h} - a(\tilde{w}, \tilde{g}) \quad (6.126)$$

(compare to 1D case from two lectures ago).

Introduce the shape functions N_A for a global node A out of n_{np} total number of nodes. With

$$\tilde{v}(\mathbf{x}) = \sum_{A \in I} N_A(\mathbf{x}) d_A \quad \text{and} \quad \tilde{g}(\mathbf{x}) = \sum_{A \in \mathcal{B}} \tilde{N}_A(\mathbf{x}) g_A \quad (6.127)$$

where we have again distinguished between interior nodes and shape function $A \in I$ and those on the Dirichlet boundary with $A \in \mathcal{B}$. Arguing as for the 1-D case, the following assembly rules result

$$\mathbf{K} \mathbf{d} = \mathbf{F} \quad (6.128)$$

$$\mathbf{K} = [K_{PQ}] \quad K_{PQ} = a(N_A, N_B), \quad (6.129)$$

where $1 \leq P, Q \leq n_{eq}$ and the number of free equations is given by the total number of nodes minus the number of nodes on the Dirichlet boundary \mathcal{B} .

P and Q can be computed from a 1D array that maps a global node A into a global equation number

$$ID(A) = \begin{cases} P & \text{for } A \in I \\ 0 & \text{for } A \in \mathcal{B} \end{cases} \quad (6.130)$$

such that $P = ID(A)$ and $Q = ID(B)$. $\mathbf{d} = \{d_Q\}$ for the solution temperatures $\tilde{v}(\mathbf{x}) = \sum N_A d_A$ and

$$\mathbf{F} = \{F_P\} \quad (6.131)$$

where

$$F_P = (N_A, H) + (N_A, h)_{\Gamma_h} - \sum_{B \in \mathcal{B}} a(N_A, N_B) q_B \quad (6.132)$$

and \mathbf{K} is again symmetric and positive definite.

6.4.2 Matrix assembly

As before, we compute \mathbf{K} and \mathbf{F} based on summation over all n_{el} elements.

$$\mathbf{K} = \sum_{e=1}^{n_{el}} \mathbf{K}^e \quad \mathbf{K}^e = \{K_{PQ}^e\} \quad (6.133)$$

$$K_{PQ}^e = a(N_A, N_B)^e = \int_{\Omega^e} (\nabla \mathbf{N}_A)^T \kappa (\nabla \mathbf{N}_B) d\Omega. \quad (6.134)$$

The RHS in eq. (6.134) corresponds to integrating over each element's area.

$$\mathbf{F} = \sum_{e=1}^{n_{el}} \mathbf{F}^e \quad \mathbf{F}^e = \{F_P^e\} \quad (6.135)$$

$$F_P^e = \int_{\Omega^e} d\Omega N_A H + \int_{\Gamma_h^e} d\Gamma N_A h - \sum_{B \in \mathcal{B}} a(N_A, N_B)^e q_B \quad (6.136)$$

where Γ_h^e is the part of the Neumann (flux) boundary within element e , and $P = ID(A)$, $Q = ID(B)$. Within each element we compute for new nodes per element with $1 \leq a, b \leq n_{en}$

$$\mathbf{K}^e = \{K_{ab}^e\} \quad K_{ab}^e = a(N_a, N_b)^e = \int_{\Omega^e} d\Omega (\nabla \mathbf{N}_a)^T \boldsymbol{\kappa} (\nabla \mathbf{N}_b) \quad (6.137)$$

$$\mathbf{f}^e = [f_a^e] \quad (6.138)$$

$$f_a = \int_{\Omega^e} N_a f_a d\Omega + \int_{\Gamma_h^e} N_a h d\Gamma - \sum_{b=1}^{n_{en}} K_{ab}^e g_b^e \quad (6.139)$$

where $g_b^e = g(\mathbf{x}_b^e)$ for prescribed g and zero otherwise. It is convenient to write

$$\mathbf{K}^e = \int_{\Omega^e} d\Omega \mathbf{B}^T \mathbf{D} \mathbf{B} \quad (6.140)$$

where \mathbf{D} is $n_{sd} \times n_{sd}$ (rows \times columns); $n_{sd} \triangleq$ number of spatial dimensions. In our case \mathbf{D} is 2×2 and $\mathbf{D} = \boldsymbol{\kappa}$. \mathbf{B} is $n_{sd} \times n_{en}$ such that $\mathbf{B} = \{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_{n_{en}}\}$ and $B_a = \nabla \mathbf{N}_a$ is $n_{sd} \times 1$.

The \mathbf{B} and \mathbf{D} matrices' general meaning is that of a gradient operator and that of a material parameter matrix at an element level, respectively. For example, if the temperatures at an element level are given by

$$\mathbf{d}^e = \{d_a^e\} = \begin{pmatrix} d_1^e \\ d_2^e \\ \vdots \\ d_{n_{en}}^e \end{pmatrix} \quad (6.141)$$

then

$$\mathbf{q}(\mathbf{x}) = -\mathbf{D}(\mathbf{x}) \mathbf{B}(\mathbf{x}) \mathbf{d}^e = -\mathbf{D}(\mathbf{x}) \sum_{a=1}^{n_{en}} \mathbf{B}_a d_a^e \quad (6.142)$$

can be used to compute the heat flux within each element. We will revisit this for the elastic problem where \mathbf{B} converts the nodal displacement solution into the strain.

6.4.3 Isoparametric elements

It is convenient to use elements where the shape functions that are used to map from a local coordinate system, for example for a four node quad ($1 \leq a \leq n_{en} = 4$) spanned by the local coordinates

$$\xi_1 = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \quad \xi_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \quad (6.143)$$

$$\xi_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \xi_4 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad (6.144)$$

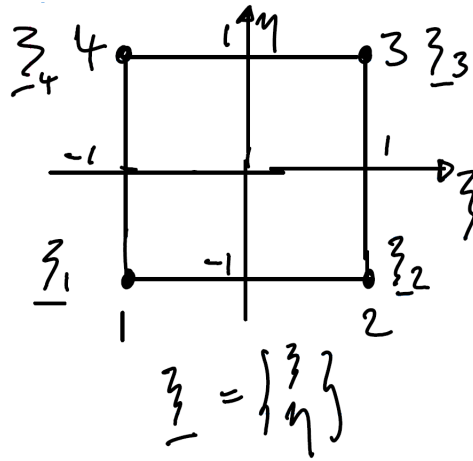


Figure 6.6: Quad element in element-local (ξ, η) coordinate space

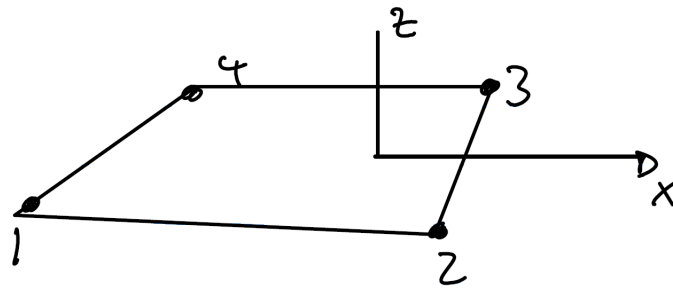


Figure 6.7: Quad element in global (x, z) coordinate space

to the total global domain where the element may be deformed

$$\mathbf{x}(\xi) = \sum_{a=1}^{n_{en}} N_a(\xi) \mathbf{x}_a^e, \quad (6.145)$$

are the same shape functions that are used to represent the solutions

$$\tilde{v}(\xi) = \sum_{a=1}^{n_{en}} N_a(\xi) d_a^e. \quad (6.146)$$

If the mapping from the element local, ξ , to real coordinate space, \mathbf{x} , is differentiable, the determinant j of the Jacobian \mathbf{J}

$$j = \det \mathbf{J} = \det \begin{pmatrix} \partial_{\xi} x & \partial_{\eta} x \\ \partial_{\xi} z & \partial_{\eta} z \end{pmatrix} \quad (6.147)$$

is $j(\xi) > 0$ for all ξ within the element ($\det \mathbf{J} = \partial_{\xi} x \partial_{\eta} z - \partial_{\xi} z \partial_{\eta} x$). $j(\xi)$ may, in practice, become very small, which indicates that the element is greatly deformed (two edges almost align, for example) which is to be avoided.

The practical use of j arises from element-local integration. Recall from the 1-D case

$$\int_{\Omega^e} f(x) d\Omega = \int_{-1}^1 f(x(\xi)) d\xi \frac{\partial x}{\partial \xi} = \int_{-1}^1 f(x(\xi)) \partial_{\xi} x d\xi \quad (6.148)$$

which generalizes to 2-D as

$$\int_{\Omega^e} f(\mathbf{x}) d\Omega = \int_{-1}^1 d\xi \int_{-1}^1 d\eta f(x(\xi, \eta), z(\xi, \eta)) j(\xi, \eta). \quad (6.149)$$

The above equation is a result of the change of variables and can be used to evaluate the $a(.,.)$ type integrals.

6.4.4 Numerical integration

While the integral over simple shape functions may be easily evaluated analytically, it is most convenient to perform a numerical integration over the element area or volume Ω^e .

In 1-D, the objective is to optimally approximate

$$\int_{-1}^1 d\xi g(\xi) \approx \sum_{i=1}^{n_{int}} g(\tilde{\xi}_i) W_i \quad (6.150)$$

for a small number of integration points n_{int} . The W_i are the weights for the function values at the integration points $\tilde{\xi}_i$. For example, the

Trapezoidal rule corresponds to $n_{int} = 2$; $\tilde{\xi}_1 = -1$; $\tilde{\xi}_2 = 1$; $W_i = 1$ and is second order accurate.

$$\int_a^b f(x) dx \approx (b-a) \frac{f(a) + f(b)}{2} \quad (6.151)$$

Simpson's rule corresponds to $n_{int} = 3$; $\xi_1 = -1$; $\xi_2 = 0$; $\xi_3 = 1$; $W_1 = \frac{1}{3}$; $W_2 = \frac{4}{3}$; $W_3 = \frac{1}{3}$ and is fourth order accurate (*i.e.* Simpson's rule integrates a cubic polynomial exactly).

$$\int_a^b f(x) dx \approx \frac{b-a}{2} \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right] \quad (6.152)$$

Gaussian quadrature is the optimal (fewest integration points for maximum accuracy) strategy. For n_{int} , it is defined by

$$W_i = \frac{2}{(1 - \xi_i^2)(\partial_{\xi} P_{n_{int}}(\xi_i)^2)} \quad 1 \leq i \leq n_{int} \quad (6.153)$$

where ξ_i is the i^{th} root of the Legendre polynomial

$$P_n(\xi) = \frac{1}{2^n n!} \frac{\partial^n}{\partial \xi^n} (\xi^2 - 1)^n. \quad (6.154)$$

This rule is $O(2n_{int})$ accurate in 1-D, and weights and ξ locations are tabulated (see below).

In 2-D, we can compute

$$\int_{-1}^1 d\xi \int_{-1}^1 d\eta g \approx \sum_{i=1}^{n_{int}} \sum_{j=1}^{n_{int}} g(\xi_i, \eta_j) W_i W_j. \quad (6.155)$$

Finally, we often need to convert the derivatives of the shape functions with respect to the global coordinates to local coordinates: $\xi(\mathbf{x})$. By means of the chain rule we obtain

$$\partial_x N_a = \partial_{\xi} N_a \partial_x \xi + \partial_{\eta} N_a \partial_x \eta \quad (6.156)$$

$$\partial_z N_a = \partial_{\xi} N_a \partial_z \xi + \partial_{\eta} N_a \partial_z \eta. \quad (6.157)$$

This can be written in matrix form as

$$\{\partial_x N_a, \partial_z N_a\} = \{\partial_{\xi} N_a, \partial_{\eta} N_a\} \begin{pmatrix} \partial_x \xi & \partial_z \xi \\ \partial_x \eta & \partial_z \eta \end{pmatrix} \quad (6.158)$$

(multiply and add, column wise)

and $\partial_{\xi} N_a$ as well as $\partial_{\eta} N_a$ can be easily computed from the shape function definition. However, the $\partial_x \xi$ type derivatives are not available explicitly. We do know the inverse relationships

$$x(\xi, \eta) = \sum_{a=1}^{n_{en}} N_a(\xi, \eta) x_a^e \quad (6.159)$$

$$z(\xi, \eta) = \sum_{a=1}^{n_{en}} N_a(\xi, \eta) z_a^e \quad (6.160)$$

from which we can compute

$$\mathbf{J} = \partial_{\xi} \mathbf{x} = \begin{pmatrix} \partial_{\xi} x & \partial_{\eta} x \\ \partial_{\xi} z & \partial_{\eta} z \end{pmatrix} \quad (6.161)$$

(e.g. $\partial_{\xi} x = \sum \partial_{\xi} N_a x_a^e$; $\partial_{\eta} z = \sum \partial_{\eta} N_a z_a^e$).

It turns out that \mathbf{J} is the inverse of eq. (6.158)

$$\begin{pmatrix} \partial_x \xi & \partial_z \xi \\ \partial_x \eta & \partial_z \eta \end{pmatrix} = \mathbf{J}^{-1} = \frac{1}{j} \begin{pmatrix} \partial_{\eta} z & -\partial_{\eta} x \\ -\partial_{\xi} z & \partial_{\xi} x \end{pmatrix} \quad (6.162)$$

with $j = \det(\partial_{\xi} \mathbf{x}) = \partial_{\xi} x \partial_{\eta} z - \partial_{\eta} x \partial_{\xi} z$. Therefore

$$\boxed{\{\partial_x N_a, \partial_z N_a\} = \{\partial_{\xi} N_a, \partial_{\eta} N_a\} \mathbf{J}^{-1}} \quad (6.163)$$

6.4.5 Simple elements, shape functions and Gaussian quadrature rules

1-D linear shape functions

$N_a(\xi) = \frac{1}{2}(1 + \xi_a \xi)$ $a = 1, 2$ with two nodes at $\xi_1 = -1$; $\xi_2 = 1$. $\partial_{\xi} N_a(\xi) = \frac{\xi_a}{2} = \frac{(-1)^a}{2}$

Quadrature

n_{int}	ξ_i	w_i	accuracy	for integration
1	0	2	2 nd order	$\int_{-1}^1 d\xi$
2	$\{-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\}$	$\{1, 1\}$	4 th order	„

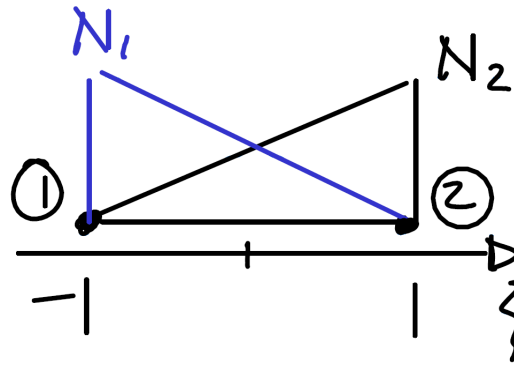


Figure 6.8: 1-D linear shape functions

Bilinear quadrilateral (“quad”) element

four nodes are located at

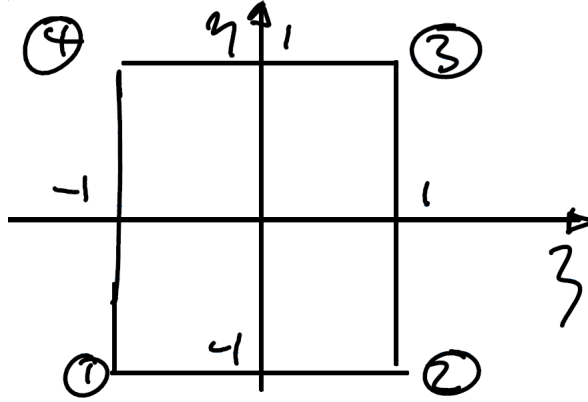


Figure 6.9: Quad element nodes in local coordinates

$$\xi_1 = \{-1, -1\} \quad \xi_2 = \{1, -1\} \quad (6.164)$$

$$\xi_3 = \{1, 1\} \quad \xi_4 = \{-1, 1\} \quad (6.165)$$

$$\xi_a = \{\xi_a, \eta_a\} \quad \text{etc. for } a = 1, 2, 3, 4 \quad (6.166)$$

$$N_a(\xi) = N_a(\xi, \eta) = \frac{1}{4}(1 + \xi_a \xi)(1 + \eta_a \eta) \quad (6.167)$$

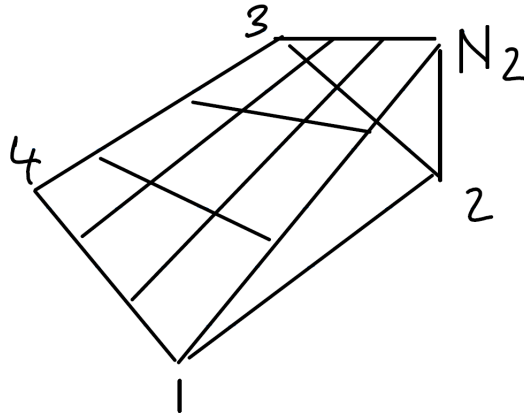


Figure 6.10: Quad element shape functions

Quadrature in 2-D				
n_{int}	visual	$\tilde{\xi}_i$	w_i	for integration
1	$[\times]$	$\{0, 0\}$	4	$\int_{-1}^1 d\xi \int_{-1}^1 d\eta$
2	$\begin{pmatrix} \times & \times \\ \times & \times \end{pmatrix}$	$\{\frac{-1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}\}$	1	"
		$\{\frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}\}$	1	"
		$\{\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\}$	1	"
		$\{\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\}$	1	"

For higher order quads, see [Hughes \(2000\)](#), sec. 3.7

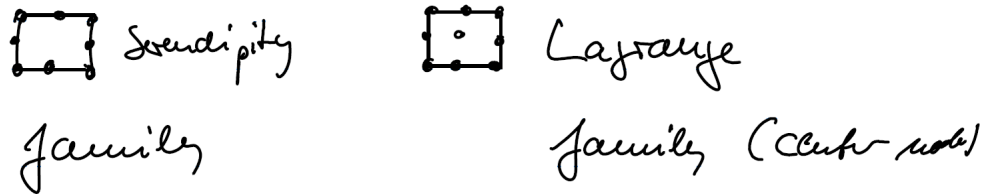


Figure 6.11: Other quad element families

[Hughes \(2000\)](#) p. 191 discusses the required level of Gaussian quadrature for adequate convergence for different element types.

Triangular elements

Linear triangle

$$t(r, s) = 1 - r - s \quad (6.168)$$

$$\mathbf{r} = \{r, s\} \quad (6.169)$$

$$\mathbf{r}_1 = \{1, 0\} \quad N_1(r, s) = r \quad (6.170)$$

$$\mathbf{r}_2 = \{0, 1\} \quad N_2(r, s) = s \quad (6.171)$$

$$\mathbf{r}_3 = \{0, 0\} \quad N_3(r, s) = t = 1 - r - s \quad (6.172)$$

Quadratic (six node) triangle

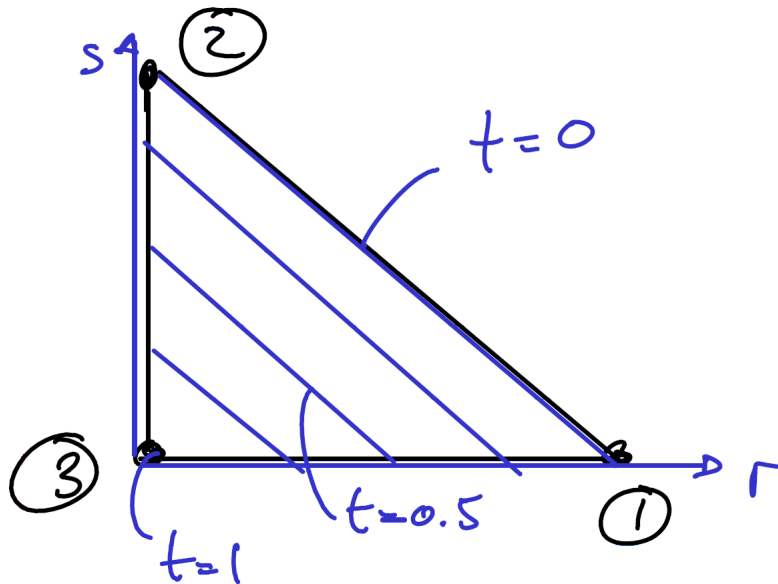


Figure 6.12: Linear triangle element nodes in local coordinates

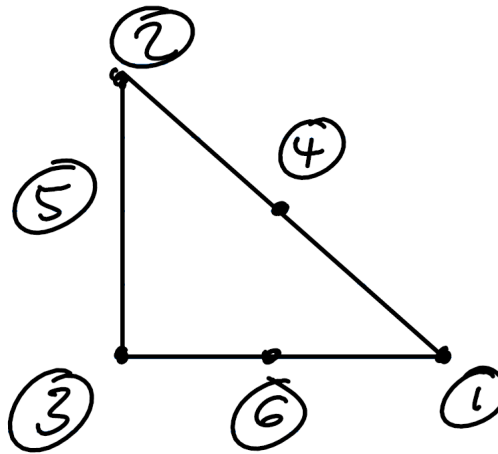


Figure 6.13: Quadratic triangle element

$$N_1 = r(2r - 1) \qquad N_4 = 4rs \qquad (6.173)$$

$$N_2 = s(2s - 1) \qquad N_5 = 4st \qquad (6.174)$$

$$N_3 = t(2t - 1) \qquad N_6 = 4rt \qquad (6.175)$$

$$(6.176)$$

See [Hughes \(2000\)](#) Appendix 3.1 for Gauss quadrature formulae.

6.5 Exercise: Heat equation in 2-D with FE

Reading

- *Hughes (2000)*, sec. 2.3-2.6
- *Dabrowski et al. (2008)*, sec. 1-3, 4.1.1, 4.1.3, 4.2.1

This FE exercise and most of the following ones are based on the MILAMIN package by *Dabrowski et al. (2008)* which provides a set of efficient, 2-D Matlab-based FE routines including a thermal and a Stokes fluid solver. Given that the code uses Matlab, MILAMIN is remarkably efficient and certainly a good choice for simple 2-D research problems that lend themselves to FE modeling. For your final project, you may want to consider working on expanding the MILAMIN capabilities, *e.g.* by adding advection to the thermal solver and combining it with the Stokes solver for a convection code.

For this and the following exercises, I slightly rewrote the original MILAMIN routines to simplify things. Over the next weeks, we will however strive to discuss all of the issues described in *Dabrowski et al. (2008)*. This paper will be a good reference, along with my lecture notes, and the original MILAMIN Matlab codes can be downloaded from <http://milamin.org/> (the latter will not be of help with the exercises).

6.5.1 Implementation of 2-D heat equation

We spent the last three lectures discussing the fundamentals of finite element analysis building up to the solution of the 2-D, stationary heat equation, which is given by

$$\frac{\partial}{\partial x} \left(\kappa \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial z} \left(\kappa \frac{\partial T}{\partial z} \right) = H, \quad (6.177)$$

where κ is conductivity (not diffusivity, we use κ to distinguish from the stiffness matrix K), and H are heat sources. Both κ and H may vary in space, and, unlike for FD, the solution domain can now be irregular.

The FE approach casts the boundary value problem (boundary conditions are assumed given) in the weak (variational) form, discretized on elements on which shape functions, N , approximate the solution of the PDE as \tilde{T} . The solution is given by nodal temperatures $\mathbf{T} = \{T_A\}$ for all *NNOD* nodes of the mesh, which can be combined to

$$\tilde{T}(x, z) = \sum_{A=1}^{NNOD} N_A(x, z) T_A. \quad (6.178)$$

Following, *e.g.*, *Hughes (2000)*, we use the Galerkin approach for which the resulting stiffness matrix components, on an element level, results as

$$K_{ab}^e = \int_{\Omega^e} \kappa^e \left(\frac{\partial N_a}{\partial x} \frac{\partial N_b}{\partial x} + \frac{\partial N_a}{\partial z} \frac{\partial N_b}{\partial z} \right) d\Omega. \quad (6.179)$$

Here, a and b are node numbers local to element e , and integration Ω_e is over the element area.

If we express the spatial coordinates $\mathbf{x} = \{x, z\}$ in a node-local coordinate system $\xi = \{\xi, \eta\}$ and use Gaussian quadrature with $NINT$ points and weights W_i for integration, we need to evaluate terms of the kind

$$K_{ab}^e = \int_{-1}^1 d\xi \int_{-1}^1 d\eta \kappa \left(\frac{\partial N_a}{\partial \xi} \frac{\partial N_b}{\partial \xi} + \frac{\partial N_a}{\partial \eta} \frac{\partial N_b}{\partial \eta} \right) J^{-1} |J| \quad (6.180)$$

$$K_{ab}^e = \sum_i^{NINT} W_i \kappa_i \left(\frac{\partial N_a}{\partial \xi} \frac{\partial N_b}{\partial \xi} + \frac{\partial N_a}{\partial \eta} \frac{\partial N_b}{\partial \eta} \right) J^{-1} |J| \quad (6.181)$$

where J^{-1} and $|J|$ are the inverse and determinant of the Jacobian matrix

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{pmatrix}, \quad (6.182)$$

respectively.

The load vector \mathbf{F} has to be assembled on an element basis as

$$\mathbf{F}_a^e = \int_{\Omega^e} N_a H d\Omega - K_{ab}^e \hat{T}_b, \quad (6.183)$$

where the terms on the right hand side are due to heat sources, H , and a correction due to prescribed temperatures on the boundaries \hat{T} (zero flux BCs need no specific treatment, see [Hughes, 2000](#), p. 69 and [Dabrowski et al. \(2008\)](#)). The global \mathbf{K} and \mathbf{F} are assembled by looping through all elements and adding up the K^e and \mathbf{F}^e contributions, while eliminating those rows that belong to nodes where essential boundary conditions (\hat{T}) are supplied. The solution is then obtained from solving

$$\mathbf{K}\mathbf{T} = \mathbf{F}. \quad (6.184)$$

Meshing

Download `generate_mesh.m`. Start by reading through this Matlab code, it is a modification of the MILAMIN wrapper for `triangle`. `Triangle` is a 2-D triangular mesh generator that was written by [Shewchuk \(2002\)](#). That work is freely available as C source code and a flexible, production quality “Delaunay” mesh generator. Delaunay mesh means that all nodes are connected by elements such that any circle which is drawn through the three nodes of an element has no other nodes within its circumference.

Note: The “dual graph” (sort of the graphic opposite) of a Delaunay mesh are the Voronoi cells around each node. Those can be constructed based on the triangulation by connecting lines that are orthogonal to each of the triangles’ sides and centered half-way between nodes. Those two properties are important for computational geometry, inverse theory, and interpolation problems.

A Delaunay triangulation is the best possible mesh for a given number of nodes in the sense that the triangles are closest to equilateral. For FE analysis, we always strive for nicely shaped elements (*i.e.* not distorted from their ideal, local coordinate system form) so that the J^{-1} does

not go haywire. Typically, meshers like triangle will allow you to refine the mesh (*i.e.* add more nodes) for a given boundary structure and overall domain by enforcing minimum area and/or angle constraints. Those refinements may also be iteratively applied based on an initial solution of the PDE, *e.g.* to refine in local regions of large variations (adaptive mesh refinement, AMR).

Exercise Download a test driver for the triangle wrapper, `mesher_test.m`. You will have to fill in the blanks after reading through `generate_mesh.m`, and make sure the triangle binary (program) is installed on your machine in the directory you are running in. You can download several different architecture binaries from our web site, also check with John Yu.

Note: For this exercise and those below, please plot and inspect graphs on the screen while playing with the code, but no need to hand in printouts, unless I denote those plots which you should hand in in **bold face**. Also, no need to print out code for this problem set.

- (a) Create a triangular grid using three node triangles for the domain $0 \leq x \leq 1, 0 \leq z \leq 1$ using minimum area constraint 0.1 and minimum angle 20° . Create a plot of this mesh highlighting nodes that are on the outer boundary.
- (b) Change the area constraint to 0.01, remesh, and replot.
- (c) Use second order triangles and an area constraint of 0.005 and minimum angle of 30° .
- (d) Using the same quality constraints, create and **plot** a mesh of an elliptical inclusion of radius 0.2, ellipticity 0.8, and 50 nodes on its perimeter. Color the elements of the inclusion differently from those of the exterior. Denote nodes on the boundary of the inclusion.
- (e) Create and plot a mesh with a circular hole and a circular inclusion of radius 0.1.

Thermal solver

Download `thermal2d_std.m`; this is a simplified version of the MILAMIN thermal solver (`thermal2d.m`) which should be easier to read than the version of [Dabrowski et al. \(2008\)](#); it also allows for heat production. Read through this Matlab code and identify the matrix assembly and solution method we discussed in class and briefly reviewed above. Also download and read through `shp_deriv_triangle.m` and `ip_triangle.m` which implement linear (three node) and quadratic (six node), triangular shape functions and derivatives, and weights for Gauss quadratures, respectively.

Exercise

- (a) Download a rudimentary driver for the mesher and thermal solver, `thermal2d_test2.m`. You will need to fill in the blanks.
- (b) Generate a regular mesh with area constraint 0.003 and solve the heat equation with linear shape functions, without heat sources, given no flux on the sides, unity temperature at the bottom, and zero temperature at the top. Plot your results. Use constant conductivity.

- (c) Place an elliptical inclusion with radius 0.4, ellipticity 0.8, and ten times higher conductivity than the ambient material in the medium, and plot the resulting temperatures. Experiment with variable resolutions and second order triangles. Comment on the how the solution changes (visually only is OK).
- (d) Set the heat production of the inclusion to 10 and 100, and **plot** the solution. Compare with boundary conditions where zero temperatures are prescribed on all boundary conditions.

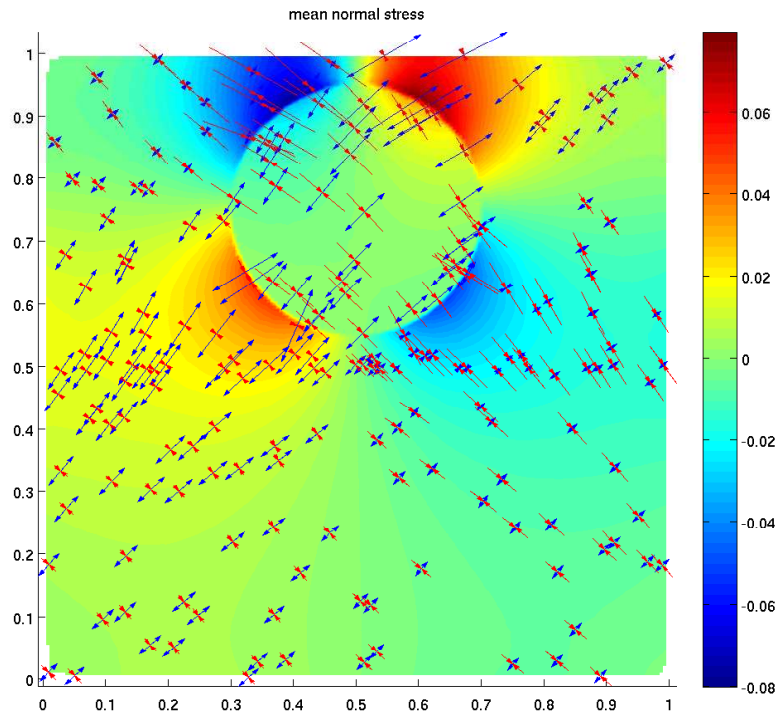


Figure 6.14: Stress solution for a sheared, elastic box with an inclusion of different strength (see problem set for details).

6.6 Exercise: Linear elastic, compressible finite element problem

Reading

- [Hughes \(2000\)](#), secs. 2.7, 2.9 - 2.11, 3.10
- [Dabrowski et al. \(2008\)](#)

This FE exercise is again based on the MILAMIN package by [Dabrowski et al. \(2008\)](#). I rewrote their “mechanical” solver (incompressible Stokes fluid, to be discussed next week) for the elastic problem, and simplified it to reduce the dependency on packages external to Matlab. A highly optimized version of the code that, for example, uses matrix reordering for K is available from me (closer to the original [Dabrowski et al. \(2008\)](#) code). When inspecting the source codes, you should find many similarities (same mesher, same variable structure, etc.) with last week’s 2-D heat equation exercise.

6.6.1 Implementation of 2-D elasticity

Problem in strong form

The strong form of the PDE that governs force balance in a medium is given by

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{f} = \mathbf{0}, \quad (6.185)$$

where $\boldsymbol{\sigma} = \sigma_{ij}$ is the stress tensor and \mathbf{f} a body force (such as due to gravity). Written in component form as PDEs for the finite element domain Ω for each of the three spatial coordinates i this is

$$\partial_j \sigma_{ij} + f_i = 0 \quad \text{on } \Omega \quad (6.186)$$

with essential boundary conditions for displacements $\mathbf{u} = \mathbf{g}$ on Γ_g and natural boundary conditions for tractions $\mathbf{h} = \boldsymbol{\sigma} \cdot \mathbf{n}$ on Γ_h with vector \mathbf{n} normal to the boundary such that

$$u_i = g_i \quad \text{on } \Gamma_{g_i} \quad (6.187)$$

$$\sigma_{ij} n_j = h_i \quad \text{on } \Gamma_{h_i}. \quad (6.188)$$

Here, Γ_h and Γ_{h_i} , and similar for g , denotes that different components of the traction vector may be specified on different parts of the domain boundary Γ .

In the case of linear, elastic behavior, the constitutive law linking dynamic with kinematic properties is given by the generalized Hooke's law

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\varepsilon} \quad \text{or} \quad \sigma_{ij} = C_{ijkl}\varepsilon_{kl}, \quad (6.189)$$

with the elasticity tensor \mathbf{C} , and the strain-tensor $\boldsymbol{\varepsilon}$, computed as

$$\varepsilon_{ij} = u_{(i,j)} = \frac{1}{2} (\partial_i u_j + \partial_j u_i). \quad (6.190)$$

Note the definition of the $_{(i,j)}$ derivative short-hand. The engineering shear strain is given by $\gamma_{ij} = 2\varepsilon_{ij}$ (note the factor of two which often causes confusion).

For an isotropic material, the constitutive law simplifies to

$$\sigma_{ij} = \lambda \varepsilon_{kk} \delta_{ij} + 2\mu \varepsilon_{ij}, \quad (6.191)$$

where μ and λ are the shear modulus and Lamè parameter, respectively; the latter specifies how incompressible a body is. (The dilation $\varepsilon_{ii} = \sum_{i=1}^3 \varepsilon_{ii}$, and the Kronecker δ , $\delta_{ij} = 1$ for $i = j$ and zero for $i \neq j$.) These moduli can also be expressed differently, *e.g.* we can write

$$\lambda = \frac{2\nu\mu}{1-2\nu} = \frac{\nu E}{(1+\nu)(1-2\nu)} \quad \text{with} \quad E = 2\mu(1+\nu), \quad (6.192)$$

with the Poisson ratio ν and Young's modulus E . If a block is fixed at the base and loaded in z -directions without constraints, then ν measures the deformation in the horizontal $\nu = -\varepsilon_{xx}/\varepsilon_{zz}$. E measures the stress exerted for the same experiment if the material is not allowed to give way sideways (free-slip in z direction) by $E = \sigma_{zz}/\varepsilon_{zz}$. Note that $\lambda = \mu$ for $\nu = 1/4$, which is often close to values measured for rocks.

Problem in weak form

It can be shown (e.g. [Hughes, 2000](#), p. 77ff) that the equivalent weak form formulation of the elastic equilibrium PDE is given by the following statement: Find the displacements \mathbf{u} for all virtual displacements \mathbf{w} such that

$$a(\mathbf{w}, \mathbf{u}) = (\mathbf{w}, \mathbf{f}) + (\mathbf{w}, \mathbf{h})_{\Gamma_h} \quad (6.193)$$

with

$$a(\mathbf{w}, \mathbf{u}) = \int d\Omega w_{(i,j)} C_{ijkl} u_{(k,l)} \quad (6.194)$$

$$(\mathbf{w}, \mathbf{f}) = \int d\Omega w_i f_i \quad (6.195)$$

$$(\mathbf{w}, \mathbf{h})_{\Gamma_h} = \sum_{i=1}^3 \left(\int_{\Gamma_{h_i}} d\Gamma w_i h_i \right). \quad (6.196)$$

Note that unlike the thermal problem, the solution function we wish to obtain using the finite element method is a vector, \mathbf{u} , rather than a scalar.

Matrix assembly

In the finite element approximation, we then solve for the nodal displacements \mathbf{d} which approximate \mathbf{u} within the elements with shape functions N from

$$\mathbf{K}\mathbf{d} = \mathbf{F}. \quad (6.197)$$

The global \mathbf{K} is assembled from the element level by

$$k_{ab}^e = \int_{\Omega^e} d\Omega B_a^T D B_b \quad (6.198)$$

where a, b are local node numbers. The elemental force vector at local node a is given by

$$f_i^e = \int_{\Omega^e} d\Omega N_a f_i + \int_{\Gamma_{h_i}^e} d\Gamma N_a h_i - \sum_b k_{ab} g_b. \quad (6.199)$$

\mathbf{B} connects displacements at the nodal level with strains. For 2-D,

$$\mathbf{B}_a = \begin{pmatrix} \partial_x N_a & 0 \\ 0 & \partial_z N_a \\ \partial_z N_a & \partial_x N_a \end{pmatrix}. \quad (6.200)$$

We can represent the strain tensor ϵ as a strain vector \mathbf{e} that can be computed from displacements \mathbf{u} by a gradient operator \mathbf{L} , like

$$\mathbf{e} = \mathbf{L}\mathbf{u} \quad \text{or} \quad e_j = L_{jk} u_k. \quad (6.201)$$

In 2-D, for example,

$$\mathbf{e} = \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{zz} \\ \gamma_{xz} \end{pmatrix} = \begin{pmatrix} \partial_x & 0 \\ 0 & \partial_z \\ \partial_z & \partial_x \end{pmatrix} \begin{pmatrix} u_x \\ u_z \end{pmatrix}, \quad (6.202)$$

where the definition of $\gamma_{xy} = 2\varepsilon_{xy}$ simplifies the notation. Within each finite element the displacements can be obtained by summation over the shape functions for each node a , N_a , times the nodal displacements,

$$\mathbf{u} = u_k = N_a \mathbf{d}_a = N_a d_a^k \quad (6.203)$$

where \mathbf{d}_a is the displacement at the local node a , and d_a^k is the k -th spatial component of this displacement. Then,

$$\mathbf{e} = e_j = L_{jk} N_a d_a^k = B_{jka} d_a^k = B_a \mathbf{d}_a \quad (6.204)$$

defines B_a . If we define a stress vector

$$\mathbf{s} = \begin{pmatrix} \sigma_{xx} \\ \sigma_{zz} \\ \tau_{xz} \end{pmatrix} \quad (6.205)$$

(with $\tau_{xz} = 2\sigma_{xy}$ in analogy to γ_{xy}), then the (symmetric) elasticity matrix D can be used to obtain stresses from displacements like

$$\mathbf{s} = D\mathbf{e} = DB_a \mathbf{d}_a. \quad (6.206)$$

The D matrix is a “condensed” version of C ,

$$D_{IJ} = C_{ijkl}, \quad (6.207)$$

where $I, J = 1, 2, \dots, n_{sd}(n_{sd} + 1)/2$ in n_{sd} dimensions, which exploits symetries in C such that

$$w_{(i,j)} C_{ijkl} u_{(k,l)} = \mathbf{e}(\mathbf{w})^T D \mathbf{e}(\mathbf{u}). \quad (6.208)$$

Equation (6.194) can then be written as

$$a(\mathbf{w}, \mathbf{u}) = \int d\Omega \mathbf{e}(\mathbf{w})^T D \mathbf{e}(\mathbf{u}), \quad (6.209)$$

where $\mathbf{e}(\mathbf{w})$ indicates applying the gradient operator to the virtual displacements, as opposed to $\mathbf{e}(\mathbf{u})$ as in eq. (6.201). In the isotropic, 2-D *plane strain* approximation, D takes the simple form

$$D = \begin{pmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & \mu \end{pmatrix} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix}, \quad (6.210)$$

where plane strain means that no deformation is allowed in the y direction, $\varepsilon_{yy} = 0$. For the case of *plane stress*, where deformation is allowed and $\sigma_{yy} = 0$,

$$D = \begin{pmatrix} \bar{\lambda} + 2\mu & \bar{\lambda} & 0 \\ \bar{\lambda} & \bar{\lambda} + 2\mu & 0 \\ 0 & 0 & \mu \end{pmatrix} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}, \quad (6.211)$$

with

$$\bar{\lambda} = \frac{2\lambda\mu}{\lambda + 2\mu}. \quad (6.212)$$

From eq. (6.212), it is apparent that plane stress reduces the effective, volumetric stiffness of a body, for $\nu = 1/4$, $\bar{\lambda} = 2/3\lambda$, because out of plane deformation is permitted.

Viscous equivalence

The constitutive law for linear viscous flow with viscosity η , and deviatoric stress σ , $\sigma = 2\eta\dot{\epsilon}$, is analogous to the elastic case with $\sigma = 2\mu\epsilon$, assuming the material is incompressible. The latter can, in theory, be achieved by letting $\nu \rightarrow 1/2$ such that the linear FE approach can be used to solve simple fluid problems. In practice, however, special care needs to be taken to allow for the numerical solution of the incompressible elastic, or the Stokes flow case, which we discuss in sec. 6.7.

Exercises

- (a) Make sure you have the Matlab subroutines `ip_triangle.m`, `shp_deriv_triangle.m`, `generate_mesh.m`, and the `triangle` binary from last week in your working directory. Both shape functions and the mesher will be reused.
- (b) Download `elastic2d_std.m`, a simple linear elasticity solver, and `calc_el_D.m` which assembles D . Also download the driver routine `elastic2d_test2.m`. You will have to fill in the blanks.
- (c) Inspect `elastic2d_std.m`, compare with the notes above for linear elasticity, and the heat solver from sec. 6.5.
- (d) Download and inspect `det2D.m`, `inv2D.m`, and `eig2d.m` (for computing the determinant, inverse, and eigensystem of 2×2 matrices, respectively) from the course web site. Writing out these operations specifically slightly improves performance compared to using Matlab's `inv` and `eig` functions. Also download `arrow.m`, which is a routine to plot vectors I got from the web, and download and inspect `calc_el_stress.m` and `plot2d_strain_cross.m`, which are used to compute element integration node stresses and plot strain- or stress, crossed-vectors symbols for visualization of the stress tensor in the eigensystem coordinates, respectively.
- (e) Consider a square, homogeneous elastic body with shear modulus $\mu = 1$, Poisson's ratio $\nu = 1/4$ and size 1×1 in x and z directions.
 - (a) Assume the body is fixed at the base (zero displacement \mathbf{u} for all $z = 0$), and sheared with a uniform u_x displacement of $u_0 = 0.1$ at the top ($z = 1$) (Load case a of Figure 6.15a). Assume the plane strain approximation and zero density (*i.e.* zero body forces). What kind of geologic deformation state does this correspond to? What kinds

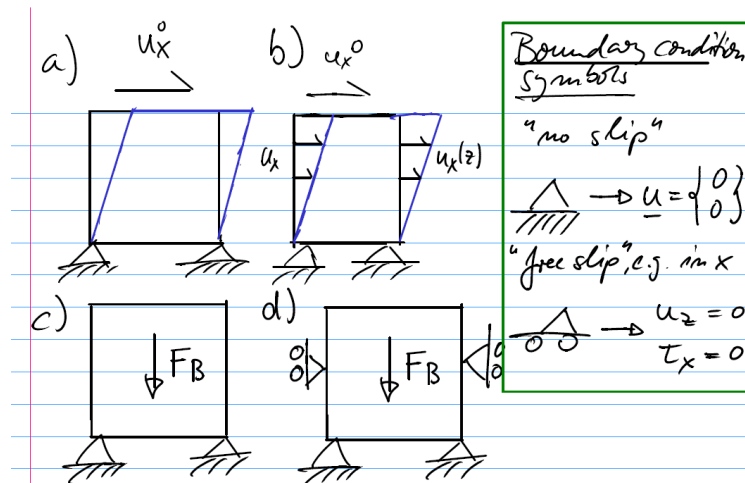


Figure 6.15: Load case sketches for some of the exercises, along with common symbols for kinematic boundary conditions.

of displacements would you expect, and how should the major (σ_1) extensional and the major compressional (σ_2) stress axis align?

- (b) Compute the displacements and stresses using the 2-D FE programs provided. Use linear, three node triangles and experiment with the integration order. Use a coarse mesh with area constraint 0.01 and angle constraint 25° .

For this and each subsequent problem, hand in three plots: i), of the deformed mesh, indicating the shape of the deformed body, possibly exaggerating the displacements of each node; ii) a plot where the background field (colored) is the amplitude of displacement, and the foreground has displacement vectors, plotted with origin at each original node location; and, iii), a plot of mean (normal) stress (colored in the background), along with extensional and compressional stress axes vector-crosses. The Matlab routines I provide can, with some alterations, perform all of these tasks.

- (c) Compare the predicted stress and displacements for plane strain and plane stress approximations. Comment.
- (d) Compare the distorted mesh shape for linear triangles with that for six node, quadratic shape functions. Increase the number of elements and compare the predicted stress fields. Does the displacement and stress field agree with your expectations for this load case?
- (e) Consider Figure 6.15b and prescribe u_x displacements linearly tapered from $u_x = u_0$ at $z = 1$ down to $u_x = 0$ at $z = 0$. Compare the predicted displacements and stresses with load case Figure 6.15a. Comment on the stress and displacement fields.
- (f) Relax the kinematic boundary conditions on the sides and top and include body forces with density $\rho = 1$ at a fixed (no slip) bottom boundary condition (Figure 6.15c). Compute the displacements and stresses, plot those, and comment.

- (g) Compute the body force load case of Figure 6.15d with free-slip (no horizontal displacements, $u_x = 0$, and no “vertical” shear stresses, $\tau_{xz} = 0$) conditions on the left and right sides. Compare the stress field with the previous, unconstrained case and comment.
- (f) Consider the square elastic medium in 2-D plane strain plus a centered, spherical inclusion with radius 0.2, shear modulus 0.001. Increase the resolution (*e.g.* use 100 nodes on the outline of the inclusion, 0.001 minimum element area, and 30° triangle edge angle). Load the system as in Figure 6.15b, compute and plot the stress field, and comment.
- (g) *Bonus:* Write a subroutine that computes the stresses at the global node locations, as opposed to the integration points within each element as is currently implemented. Use the nodal stresses and `trisurf` to generate a plot of triangles colored according to their normal stress. Compare with the previous plot.

6.7 Incompressible flow and elasticity and FE

Reading: [Hughes \(2000\)](#) sec. 4.2-4.4

6.7.1 Governing equations

- As for the thermal or the elastic problem, we will only consider the static case (but see section 21 of the notes). In the absence of inertia case (infinite Prandtl number), this reduces the fluid equilibrium (Navier-Stokes) equations to the *Stokes equations* which are formally similar to the elastic problem considered in sec. 18.
- Since most fluids are (nearly) incompressible, we will revisit the general problem of elastic deformation.

The ratio of the bulk, B , and shear, μ , modulus can be expressed as a function of Poisson's ratio ν

$$\frac{B}{\mu} = \frac{2(1+\nu)}{3(1-2\nu)}.$$

As noted earlier, for the $\nu \rightarrow \frac{1}{2}$, incompressible case $\frac{B}{\mu} \rightarrow \infty$. However, in this case we cannot use the regular, elastic (isotropic, linear) constitutive law

$$\sigma_{ij} = \lambda \partial_k u_k \delta_{ij} + 2\mu \epsilon_{ij} \quad (6.213)$$

because $\lambda = \frac{2\nu\mu}{1-2\nu}$ becomes unbounded for $\nu = \frac{1}{2}$. Therefore, we need to use

$$\sigma_{ij} = -p \delta_{ij} + 2\mu \epsilon_{ij} \quad (6.214)$$

instead, where the hydrostatic pressure is

$$p = -\frac{1}{3}\sigma_{kk}. \quad (6.215)$$

The fluid equivalent of eq. (6.214) is

$$\sigma_{ij} = -p \delta_{ij} + 2\eta \dot{\epsilon}_{ij} \quad (6.216)$$

where we've replaced strain ϵ with strain rate $\dot{\epsilon}$, and η is the dynamic viscosity. Since the addition of p has introduced another unknown, we require an additional constraint in addition to force balance ($\nabla \sigma = \mathbf{f}$) which is given by the continuity (of mass) equation. In the case of an incompressible medium

$$\nabla \cdot \mathbf{u} = \partial_i u_i = 0 \quad (6.217)$$

so that the strong form of the incompressible elastic and fluid problems become

$$\partial_j \sigma_{ij} + f_i = 0 \quad \partial_j \sigma_{ij} + f_i = 0 \quad (6.218)$$

$$\partial_i u_i = 0 \quad \partial_i v_i = 0 \quad (6.219)$$

$$u_i = g_i \quad v_i = g_i \quad (6.220)$$

$$\sigma_{ij} n_j = h_i \quad \sigma_{ij} n_j = h_i \quad (6.221)$$

where eqs. (6.218) and (6.219) hold in the domain Ω , eqs. (6.220) and (6.221) are boundary conditions and hold on Γ_{g_i} and Γ_{h_i} respectively. \mathbf{u} and \mathbf{v} are displacement and velocity, respectively, and

$$\varepsilon_{ij} = u(i, j) = \frac{1}{2}(\partial_i u_j + \partial_j u_i) \quad (6.222)$$

$$\dot{\varepsilon}_{ij} = v(i, j) = \frac{1}{2}(\partial_i v_j + \partial_j v_i). \quad (6.223)$$

Note that from eq. (6.217) and by Gauß' Theorem

$$\int d\Omega \partial_i u_i = \int d\Gamma u_i n_i = \int d\Gamma g_i n_i = 0 \quad (6.224)$$

and if there are only displacement/velocity BCs, and $\mathbf{f} = 0$, then pressures are only determined up to a constant.

6.7.2 FE solution to the incompressible elastic/flow problem

Different approaches exist involving Lagrange multipliers, penalty methods, or Uzawa iterations. See for example, *Zhong et al. (2007)*. All methods typically involve a stiffening of the deforming structure using some parameter λ that controls the degree of compression. $\lambda \rightarrow \infty$ would lead to the desired case of $\nabla \cdot \mathbf{u} = 0$, but may lead to an ill-conditioned (hard or impossible to invert) matrix. We will pursue a mixed formulation.

Mixed formulation

This is valid both for compressible and incompressible behavior, such that

$$\sigma_{ij} = -p \delta_{ij} + 2\mu \varepsilon_{ij} \quad (6.225)$$

$$\partial_i u_i + \frac{p'}{\lambda} = 0 \quad (6.226)$$

where eq. (6.226) corresponds to the elastic case. For $\nu \rightarrow \frac{1}{2} \Rightarrow \lambda \rightarrow \infty \Rightarrow \partial_i u_i = 0$. For $\nu < \frac{1}{2}$, we can eliminate eq. (6.226)

$$p' = -\lambda \partial_i u_i \quad (6.227)$$

such that (6.225) recovers

$$\sigma_{ij} = \lambda \partial_i u_i \delta_{ij} + 2\mu \varepsilon_{ij}.$$

However, p' is the proper hydrostatic pressure

$$p = -\frac{1}{3}\sigma_{ij}$$

only for the incompressible case. For the compressible case

$$p = -\frac{1}{3}\sigma_{ij} = -\left(\lambda + \frac{2\mu}{3}\right)\varepsilon_{ii} = -K\varepsilon_{ii}$$

with the incompressible modulus $K = \lambda + \frac{2\mu}{3}$, but from eq. (6.227)

$$p' = -\lambda\varepsilon_{ii}.$$

$p' \approx p$ and $\lambda \approx K$ only for $\mu \ll \lambda$, the nearly incompressible case. The major results are outlined below.

Equations in strong form

$$\partial_j \sigma_{ij} + f_i = 0 \quad \text{on } \Omega \quad (6.228)$$

$$\partial_i u_i + \frac{p}{\lambda} = 0 \quad \text{on } \Omega \quad (6.229)$$

$$u_i = g_i \quad \text{on } \Gamma_{g_i} \quad (6.230)$$

$$\sigma_{ij} n_j = h_i \quad \text{on } \Gamma_{h_i} \quad (6.231)$$

Equations in weak form

$$\int d\Omega w(i, j) \sigma_{ij} - \int d\Omega q \left(\partial_i u_i + \frac{p}{\lambda} \right) = \int d\Omega w_i f_i + \sum_i^{n_{sd}} \int d\Gamma_{h_i} w_i h_i \quad (6.232)$$

where w, g are virtual displacements and pressures, respectively, and n_{sd} is the number of dimensions. Special care must be taken in the next step: the choice of shape functions for pressure and velocities/displacements (see [Hughes, 2000](#), sec. 4.3), but in general, the pressure shape functions should be lower order (*e.g.* constant) than the displacements (*e.g.* quadratic).

Matrix formulation

$$\begin{pmatrix} \bar{K} & G \\ G^T & M \end{pmatrix} \begin{pmatrix} \mathbf{d} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{F}} \\ \mathbf{H} \end{pmatrix} \quad (6.233)$$

where the LHS is symmetric but not positive definite (it has negative eigen values or \sim zero eigen values for pressure modes), \mathbf{p} are the pressures at nodes (*e.g.* center of element), \mathbf{d} are the displacements at nodes (*e.g.* edges of elements), $\bar{\mathbf{F}}$ are the body forces, and \mathbf{H} is the divergence source in the boundary conditions.

This is called the “segregated d/p form” of the equations and is valid for the general case (including finite compressibility case).

$$\bar{K} \quad \bar{a}(\mathbf{w}, \mathbf{d}) \quad \text{stiffness matrix (symm, pos. def.)} \quad (6.234)$$

$$G \quad -(\nabla \mathbf{w}, \mathbf{p}) \quad \text{gradient operator} \quad (6.235)$$

$$G^T \quad -(\mathbf{q}, \nabla \cdot \mathbf{v}) \quad \text{divergence operator} \quad (6.236)$$

$$M \quad -\left(\mathbf{q}, \frac{p}{\lambda}\right) \quad \text{symm, neg. def. for } v \rightarrow \frac{1}{2} M \rightarrow 0 \quad (6.237)$$

In general, we can distinguish 3 cases (*in class, we will only consider the element-by-element, discontinuous pressure case 3*).

(a) **The compressible case**

$$\bar{K} \mathbf{d} + G \mathbf{p} = \bar{\mathbf{F}} \quad (6.238)$$

$$G^T \mathbf{d} + M \mathbf{p} = \mathbf{H} \quad \text{solve for } \mathbf{p} \quad (6.239)$$

$$\mathbf{p} = M^{-1}(\mathbf{H} - G^T \mathbf{d}) \quad (6.240)$$

substitute eq. (6.241) into eq. (6.238)

$$\begin{aligned} \bar{K} \mathbf{d} + G M^{-1}(\mathbf{H} - G^T \mathbf{d}) &= \bar{\mathbf{F}} \\ (\bar{K} - G M^{-1} G^T) \mathbf{d} &= \bar{\mathbf{F}} - G M^{-1} \mathbf{H} \end{aligned} \quad (6.241)$$

which reduces to solving the following system of equations

$$\mathbf{K} \cdot \mathbf{d} = \mathbf{F}$$

where \mathbf{K} is symmetric and positive definite and $\mathbf{K} = \bar{K} - G M^{-1} G^T$ and $\mathbf{F} = \bar{\mathbf{F}} - G M^{-1} \mathbf{H}$ from eq. (6.241).

If p is discontinuous on the elements, eq. (6.241) can be solved locally, on the element level.

→ determine \mathbf{p} from eq. (6.226).

(b) **The incompressible case**

Solve eq. (6.238) for \mathbf{d} , pre-multiply with G^T and use eq. (6.239) to get the pressure.

$$(G^T \bar{K}^{-1} G) \mathbf{p} = G^T \bar{K}^{-1} \bar{\mathbf{F}} - \mathbf{H} \quad (6.242)$$

$$\mathbf{K} \quad \mathbf{p} = \quad \mathbf{F} \quad (6.243)$$

(c) **The element-by-element, discontinuous pressure case**

$$\mathbf{K} \mathbf{p} = \mathbf{F} \quad (6.244)$$

$$\mathbf{u}(\mathbf{x}) = \sum_a N_a(\mathbf{x}) \mathbf{d}^a \quad (6.245)$$

$$p(x) = \sum_{\tilde{a}} \tilde{N}_{\tilde{a}}(\mathbf{x}) p_{\tilde{a}} \quad (6.246)$$

$$\mathbf{K} \leftarrow \mathbf{K}^e \quad \text{from element levels} \quad (6.247)$$

$$\mathbf{F} \leftarrow \mathbf{f}^e \quad (6.248)$$

$$\mathbf{K}^e = \bar{\mathbf{K}}^e - \mathbf{g}^e (\mathbf{m}^e)^{-1} (\mathbf{g}^e)^T$$

$$\mathbf{f}^e = \bar{\mathbf{f}} - \mathbf{g}^e (\mathbf{m}^e)^{-1} \mathbf{h}^e$$

$$\mathbf{p}^e = -(\mathbf{m}^e)^{-1} (\mathbf{g}^e)^T \mathbf{d}$$

Matrix assembly for the element-by-element, discontinuous pressure case

$$\bar{\mathbf{K}}_{ab} = \int_{\Omega^e} d\Omega \mathbf{B}_a^T \bar{\mathbf{D}} \mathbf{B}_b$$

$\bar{\mathbf{D}}$ here only has deviatoric terms, for the plane strain case

$$\bar{\mathbf{D}} = \mu \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

and \mathbf{B} transforms displacements into strains (as before).

$$\tilde{f}_p^e = \int d\Omega N_a f_i + \int_{\Gamma_{h_i}} \partial \Gamma N_a h_i - \sum_q k_{pq} g_q^e$$

Pressure components

$$\mathbf{m}_{\tilde{a}\tilde{b}}^e = \int_{\Omega^e} d\Omega - \frac{1}{\lambda} \tilde{N}_{\tilde{a}} \tilde{N}_{\tilde{b}} \quad (6.249)$$

$$\text{Mixed} \quad (6.250)$$

$$g_{a\tilde{a}} = - \int d\Omega \nabla \cdot (N_a \mathbf{e}) \tilde{N}_{\tilde{a}} \quad (6.251)$$

$$h_{\tilde{a}}^e = - \sum_p g_{p\tilde{a}}^e g_p^e \quad (6.252)$$

Stress vector for 2-D plane strain

$$\boldsymbol{\sigma}(\mathbf{x}) = - \sum_{\tilde{a}} \tilde{N}_{\tilde{a}}(\mathbf{x}) p_{\tilde{a}} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + \bar{\mathbf{D}}(\mathbf{x}) \sum_a \mathbf{B}_a(\mathbf{x}) \mathbf{d}_a$$

in each element.

Powell and Hestenes iterations

As detailed in the problem set on the 2-D Matlab implementation (*cf. Dabrowski et al., 2008*), iterations are needed to stabilize the solution of the segregated form for the incompressible problem, or for large λ .

$\mathbf{p}^0 = 0$

while $\max(\Delta \mathbf{p}^i) > \text{tolerance}$

$$\mathbf{d}^i = (\bar{\mathbf{K}} - \mathbf{G} \mathbf{M}^{-1} \mathbf{G}^T)^{-1} (\mathbf{F} - \mathbf{G} \mathbf{p}^i) \quad (6.253)$$

$$\Delta \mathbf{p}^i = -\mathbf{M}^{-1} \mathbf{G}^T \mathbf{d}^i \quad \leftarrow \text{quasi-divergence} \quad (6.254)$$

$$\mathbf{p}^{i+1} = \mathbf{p}^i + \Delta \mathbf{p}^i \quad (6.255)$$

$$i = i + 1 \quad (6.256)$$

end

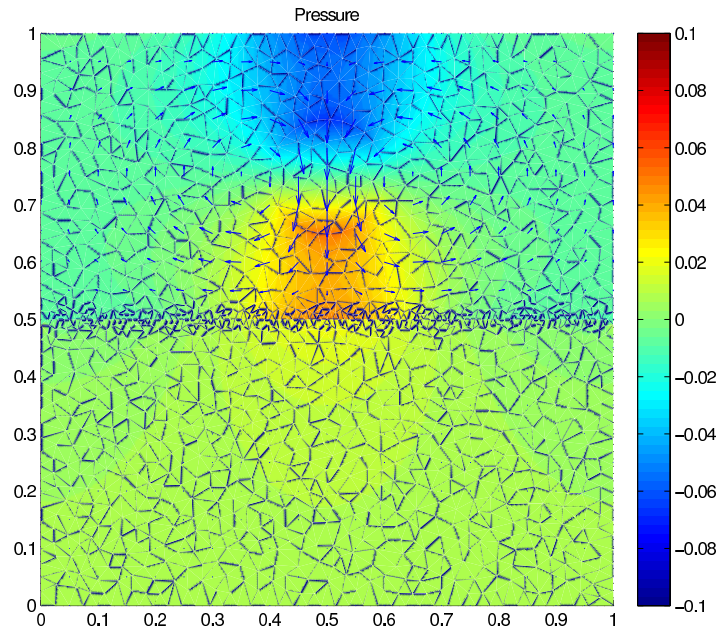


Figure 6.16: Pressure and velocity solution for a sinking, fluid slab impinging on viscosity contrast problem.

6.8 Exercise: Linear incompressible Stokes flow with FE

Reading

- *Hughes (2000)*, sec. 4.2-4.4
- *Dabrowski et al. (2008)*, sec. 4.1.2, 4.3.1, 4.4-4.7

This FE exercise is again based on the MILAMIN package by *Dabrowski et al. (2008)*. I simplified their “mechanical”, incompressible Stokes fluid solver to reduce the dependency on packages external to Matlab. *Dabrowski et al. (2008)* have a highly optimized version, which you can obtain from me; it uses, *e.g.*, reordering of node numbers to improve matrix solutions which comes an important memory issue for larger problems. The notation is close to *Dabrowski et al. (2008)*, but *Hughes (2000)* has a somewhat clearer exposition.

6.8.1 Implementation of incompressible, Stokes flow

We are interested in the instantaneous solution of a fluid problem in the absence of inertia (infinite Prandtl number limit), as is appropriate for the Earth’s mantle, for example (see earlier problem set, sec. 6.6). These approximations transform the general, Navier-Stokes equation for fluids into the Stokes equation, which is quite a bit easier to solve, because there is no turbulence.

The static force-balance equations for body forces due to gravity are given by

$$\nabla \cdot \boldsymbol{\sigma} = \mathbf{f} = \rho \mathbf{g} \quad \text{or} \quad \partial_j \sigma_{ij} = \rho g_i, \quad (6.257)$$

where $\boldsymbol{\sigma}$ is the stress tensor, ρ density, and \mathbf{g} gravitational acceleration ($g_i = g\delta_{iz}$). We assume that the medium is incompressible and a linear (Newtonian) fluid constitutive law holds,

$$\sigma_{ij} = -p\delta_{ij} + 2\eta\dot{\epsilon}'_{ij}, \quad (6.258)$$

where η is the viscosity, p pressure, and $\dot{\epsilon}'$ the deviatoric strain-rate tensor,

$$\dot{\epsilon}'_{ij} = v_{(i,j)} - \frac{1}{3}\partial_k v_k \delta_{ij} = \frac{1}{2}(\partial_j v_i + \partial_i v_j) - \frac{1}{3}\partial_k v_k \delta_{ij}, \quad \text{or} \quad \dot{\epsilon}' = \dot{\epsilon} - tr(\dot{\epsilon}), \quad (6.259)$$

where \mathbf{v} are the velocities, and $\dot{\epsilon}'$ is the total strain-rate reduced by the isotropic part. Using the constitutive law, and assuming 2-D (x - z space), the Stokes equation can be written as (also see continuum mechanics cheat sheet, sec. 2.2)

$$\partial_x \left(\eta \left(\frac{4}{3}\partial_x v_x - \frac{2}{3}\partial_z u_z \right) \right) + \partial_z (\eta (\partial_z v_x + \partial_x v_z)) - \partial_x p = 0 \quad (6.260)$$

$$\partial_z \left(\eta \left(\frac{4}{3}\partial_z v_z - \frac{2}{3}\partial_x u_x \right) \right) + \partial_x (\eta (\partial_z v_x + \partial_x v_z)) - \partial_z p = \rho g_z. \quad (6.261)$$

Often we write the constitutive law for deviatoric quantities only,

$$\tau_{ij} = 2\eta\dot{\epsilon}'_{ij} \quad \text{with} \quad \tau_{ij} = \sigma_{ij} + p = \sigma_{ij} - \sigma_{kk}/3. \quad (6.262)$$

Incompressibility translates to a constraint on the divergence of the velocity

$$\nabla \cdot \mathbf{v} = 0 \quad \text{or} \quad \partial_i v_i = 0, \quad (6.263)$$

which allows solving eq. (6.257) for the additional unknown, pressure. For $\nabla \cdot \mathbf{v} = 0$,

$$tr(\dot{\epsilon}) = 0 \quad \rightarrow \quad \dot{\epsilon}' = \dot{\epsilon}, \quad (6.264)$$

but we made the distinction between deviatoric and total strain-rate because we numerically only approximate the incompressible continuity equation, eq. (6.263), by requiring the divergence to be smaller than some tolerance. There are several approaches to do this (*e.g.* penalty or Lagrange methods) which typically involve iterations to progressively introduce additional “stiffness” to the medium. We shall allow for a finite, large bulk viscosity, κ , such that eq. (6.263) is approximated by

$$\partial_x v_x + \partial_z v_z = -\frac{p}{\kappa}, \quad (6.265)$$

the right hand side would $\rightarrow 0$ for $\kappa \rightarrow \infty$. Eq. (6.265) is valid for the incompressible and the compressible cases. However, for the compressible case, where the constitutive law, eq. (6.258), is replaced by

$$\sigma_{ij} = \kappa\partial_k v_k \delta_{ij} + 2\eta\dot{\epsilon}_{ij}, \quad (6.266)$$

p cannot be interpreted as the actual pressure, $P = -\sigma_{ii}/3$, rather it is a pressure parameter because $P = -(\kappa + 2\eta/3)\partial_i v_i$ and $p = -\kappa\partial_i v_i$. (The general, compressible case is identical to the elastic formulation where $\mathbf{v} \rightarrow \mathbf{u}$ and the constitutive law is $\sigma_{ij} = \lambda\partial_k v_k \delta_{ij} + 2\mu\dot{\epsilon}_{ij}$.)

6.8.2 Problem in strong form

The (finite element) solution is to be found for the problem stated by eqs. (6.257) and (6.265),

$$\partial_j \sigma_{ij} + f_i = 0 \quad (6.267)$$

$$\partial_i v_i + p/\kappa = 0 \quad (6.268)$$

with boundary conditions

$$v_i = g_i \quad \text{on} \quad \Gamma_{g_i} \quad (6.269)$$

$$\sigma_{ij} n_j = h_i \quad \text{on} \quad \Gamma_{h_i} \quad (6.270)$$

for velocities and tractions, respectively.

Problem in weak form

The pressure equation modifies the stiffness matrix component such that

$$\int d\Omega w_{(i,j)} \sigma_{ij} - \int d\Omega q (\partial_i v_i + p/\kappa) = \int d\Omega w_i f_i + \sum_i^{n_{sd}} \int_{\Gamma_{h_i}} d\Gamma w_i h_i, \quad (6.271)$$

with n_{sd} the number of spatial dimensions. We again use the Galerkin approach, which leads to the matrix equations.

Matrix assembly

In analogy to the elastic problem, we define a (total) strain-rate vector $\dot{\mathbf{e}} = \{\dot{\epsilon}_{xx}, \dot{\epsilon}_{zz}, \dot{\gamma}_{xz} = 2\dot{\epsilon}_{xz}\}$ such that strain-rates on an element level can be computed from

$$\dot{\mathbf{e}} = \mathbf{B}\mathbf{v}, \quad (6.272)$$

where \mathbf{v} are velocities given at the element-local nodes, and \mathbf{B} holds the derivatives, as before. When expressed for the local node a and shape functions N_a ,

$$\mathbf{B}_a = \begin{pmatrix} \partial_x N_a & 0 \\ 0 & \partial_z N_a \\ \partial_z N_a & \partial_x N_a \end{pmatrix}. \quad (6.273)$$

Likewise, deviatoric stresses can be computed from $\mathbf{t} = \mathbf{D}\dot{\mathbf{e}}$, where the property matrix \mathbf{D} shall be given by

$$\mathbf{D} = \eta \begin{pmatrix} \frac{4}{3} & -\frac{2}{3} & 0 \\ -\frac{2}{3} & \frac{4}{3} & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad (6.274)$$

for a plane-strain approximation (compare the elastic case). This allows to express the stress vector with pressure part as

$$\mathbf{s} = -p\mathbf{m} + \mathbf{D}\dot{\mathbf{e}}, \quad (6.275)$$

where $\mathbf{m} = \{1, 1, 0\}$. The deviatoric-only version of \mathbf{D} is

$$\mathbf{D}' = \eta \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (6.276)$$

In analogy to the displacement, \mathbf{u} , representation for the elastic problem, interpolated velocities, \mathbf{v} , are assumed to be given by the summation over the nodal velocities times the shape functions within each element

$$\mathbf{v}(\mathbf{x}) \approx \sum_{a=1} N_a(\mathbf{x}) \mathbf{v}_a. \quad (6.277)$$

Given the incompressibility constraint, special care has to be taken in the choice of shape functions, and we will use the seven-node, *Crouzeix and Raviart (1973)* triangle with quadratic shape functions N_a (cf. *Dabrowski et al., 2008*). As detailed in *Hughes (2000)*, one can either choose “conforming” elements for the problem at hand and get a nice solution for the velocities and pressure right away (which is what we do here), or choose theoretically inappropriate shape functions and later correct the pressure (e.g. for so-called “checkerboard modes”). The latter, rough-and-ready approach may seem less appealing, but works just as well if done properly.

A departure from the elastic problem is that the pressure is treated differently from \mathbf{v} , and we use linear (constant) shape functions for

$$p(\mathbf{x}) = \sum_{a'} \tilde{N}_{a'}(\mathbf{x}) p_{a'} = \tilde{N}_{a'} p_{a'}, \quad (6.278)$$

where a' indicates an element-local node, to be distinguished from a which we use for the velocity shape function, and the respective total node number per element may be different (e.g. seven for velocities, one for pressure). This approach is called the “mixed formulation”. Correspondingly, we introduce an isotropic strain operator \mathbf{B}_v , such that

$$\nabla \cdot \mathbf{v} = \dot{\epsilon}_v = \mathbf{B}_v \mathbf{v}^e, \quad (6.279)$$

and $p^e = -\kappa \mathbf{B}_v \mathbf{v}^e$.

The global system of equations for velocity, \mathbf{V} , and pressure, \mathbf{P} , at the nodes is given by

$$\begin{pmatrix} \mathbf{A} & \mathbf{Q}^T \\ \mathbf{Q} & \mathbf{M} \end{pmatrix} \begin{pmatrix} \mathbf{V} \\ \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{F} \\ \mathbf{H} \end{pmatrix}, \quad (6.280)$$

where \mathbf{F} are the load vectors, e.g. due to body forces, and \mathbf{H} is due to the divergence that may be imposed traction loads for the compressible case ($\mathbf{H} = \mathbf{0}$ for incompressible case).

On an element-level, the stiffness matrix is given by

$$\begin{aligned} \mathbf{k}_{ab}^e &= \int_{\Omega_e} d\Omega \begin{pmatrix} \mathbf{A} & \mathbf{Q}^T \\ \mathbf{Q} & \mathbf{M} \end{pmatrix} \\ &= \int_{\Omega_e} d\Omega \begin{pmatrix} \mathbf{B}_a^T \mathbf{D} \mathbf{B}_b & -\mathbf{B}_v^T \tilde{N}^T \\ -\tilde{N} \mathbf{B}_v & \tilde{N}_a \tilde{N}_b^T \end{pmatrix}, \end{aligned} \quad (6.281)$$

i.e. $\mathbf{Q} = -\tilde{\mathbf{N}}\mathbf{B}_v$, $\mathbf{M} = -\kappa^{-1}\tilde{\mathbf{N}}\tilde{\mathbf{N}}^T$, and \mathbf{A} corresponds to the total stiffness matrix \mathbf{k} in the elastic case. We have omitted the dependence on the local node number in eq. (6.281).

We avoid having to actually solve for the global p by using the “static condensation”. This means that we locally (element by element) invert \mathbf{M} to obtain the pressure from

$$p \approx \tilde{\mathbf{N}}_\pi p_\pi = \tilde{\mathbf{N}}^T (\mathbf{M}^{-1} \mathbf{Q} \mathbf{v}^e) = -\kappa \mathbf{B}_v \mathbf{v}^e. \quad (6.282)$$

We can then simplify eq. (6.281) to the global, linear equation system

$$\mathbf{A}' \mathbf{V} = \mathbf{f}, \quad (6.283)$$

which is to be solved for the nodal velocities \mathbf{V} . Here, $\mathbf{f} = \{\mathbf{f}^e\} = \{\rho^e \mathbf{g}^e\}$ and

$$\mathbf{A}' = \mathbf{A} + \mathbf{Q}^T \mathbf{M}^{-1} \mathbf{Q}. \quad (6.284)$$

\mathbf{A}' is now symmetric and positive-definite, and the regular, efficient matrix solution methods can be applied. However, the matrix becomes ill-conditioned (hard to invert) for the desired large values of κ , which is why iterations for the velocity solution are needed in order to achieve the incompressibility constraint. Our example code applies “Powell and Hestenes” iterations for the global velocity and pressure vectors \mathbf{V} and \mathbf{P} (cf. [Dabrowski et al., 2008](#)), as in

$$\begin{aligned} \mathbf{P}^0 &= 0, \quad i = 0 \\ \text{while } \max(\Delta \mathbf{P}^i) &> \text{tolerance} \\ \mathbf{V}^i &= (\mathbf{A}')^{-1} (\mathbf{f} - \mathbf{Q}^T \mathbf{P}^i) \\ \Delta \mathbf{P}^i &= \mathbf{M}^{-1} \mathbf{Q} \mathbf{V}^i \\ \mathbf{P}^{i+1} &= \mathbf{P}^i + \Delta \mathbf{P}^i \\ i &= i + 1 \\ \text{end while} \end{aligned} \quad (6.285)$$

If and when the algorithm converges, the pressure correction $\Delta \mathbf{P}^i = \mathbf{M}^{-1} \mathbf{Q} \mathbf{V}$, which depends on the divergence, $\mathbf{M}^{-1} \mathbf{Q} \mathbf{V}$, goes to zero. Above, all matrices are meant to be the global, not element-local representation.

6.8.3 Exercises

- Make sure you have the common FE Matlab subroutines from the earlier exercises (`ip_triangle.m`, `shp_deriv_triangle.m`, `generate_mesh.m`), and the `triangle` binary in your working directory.
- Download the `mechanical2d_test2.m` driver, and the `mechanical2d_std.m` solver. Inspect both and compare with the lecture notes and above for implementation. You will have to fill in the blanks in the driver.

- (c) Compute the sinking velocity of a dense sphere (*i.e.* disk in 2-D) with radius 0.1 that is centered in the middle of the 1×1 box with free-slip boundary conditions (no shear stress tangentially to the boundary, no motion perpendicular to the boundary) on all sides.

Ensure that the sphere is well resolved by choosing ~ 50 points on its circumference and using a high quality mesh. Use the second order triangles (six nodes on the edges plus one added in the center), and six integration points.

- (a) Note how boundary conditions are implemented in the Matlab code, and comment on essential and natural types.
 - (b) Compute the solution for the dense sphere with the same viscosity as the background. **Plot** the velocities on top of the pressure within the fluid. You may choose whichever absolute parameter values you like but will have to be consistent subsequently.
 - (c) Change the number of integration points to three, and replot. Change the type of element to linear, replot. Comment on the velocity and pressure solution.
 - (d) The solver applies a finite bulk viscosity (it should be ∞ for an incompressible fluid). For increasing sphere/medium viscosity contrasts upward of 10^3 , experiment with increasing the pseudo-incompressibility and comment on the stability of the solution. After this experiment, reset to the starting value.
 - (e) The solver applies iterations to enforce the incompressibility constrain. Change the tolerance criterion and comment on the resulting velocity and pressure solutions.
 - (f) Change back to seven node triangles with six integration points. **Plot** the vertical velocity, v_z , along a profile for $x \in [0; 1]$ at $z = 0.5$.
 - (g) Vary the radius of the sphere and comment on how the v_z profiles are affected by the size of the sinker relative to the box size. How small does the sphere have to be to not feel the effect of the boundaries?
 - (h) Change the boundary conditions to no-slip ($\mathbf{v} = \mathbf{0}$ on all domain edges), replot the vertical velocity profile for a sphere of radius 0.1. Comment. Change back to free-slip subsequently.
 - (i) Compute the sinking velocity of a dense sphere with radius 0.1 that is 0.001, 1, and 1,000 times the background viscosity. Define the sinking velocity as the maximum velocity at the sphere's origin at $\mathbf{x} = \{0.5, 0.5\}$.
 - (j) Provide an analytical estimate for the sinking velocities (*e.g.* check your notes from earlier in the course) and compare with the numerical estimates.
- (d) Compute the sinking velocities of a highly elliptical (choose ellipticity 0.975, radius 0.25) body whose viscosity is 1,000 times the background viscosity. Investigate the case where this “needle” is oriented horizontally (*i.e.* perpendicular to the sinking velocity at its center) and when it is oriented vertically (*i.e.* aligned with the sinking velocity at its center). Comment on the difference in the maximum sinking velocity between the two elliptical and the spherical cases.

- (e) *Bonus (somewhat involved)*: Compute the sinking velocity for a non-Newtonian, powerlaw fluid with $\dot{\epsilon}'_{II} \propto \tau^n_{II}$ where $n = 3$, and II indicated the second, shear invariants.

Hints: You will have to convert the constitutive law to a viscosity, for which you can assume constant strain-rates. Then, you will have to modify the code to compute the strain-rate tensor to obtain the second invariant, $\dot{\epsilon}_{II}$. (You might want to check the elastic exercise for the use of D and B to obtain strain and stress.) This strain-rate will then enter the viscosity, and you will have to use a second iteration loop, starting with a Newtonian viscosity, then updating the viscosity from the first velocity solution, and repeat until velocities do not change by more than some tolerance.

6.9 Time-dependent FE methods

So far, we have only considered static solutions for heat and continuum mechanics problems. Finite elements can also be used to solve dynamic, or time evolving problems. In analogy to our treatment of FD methods, the ODE part of the equations (the time-derivative) can be dealt with by implicit or explicit methods.

Reading: *Hughes (2000)* sec. 7.1, 8.1-8.2

6.9.1 Example: Heat equation

We return to the heat equation as an example of a “parabolic” PDE (as opposed to “hyperbolic”, *e.g.* wave propagation problems).

Strong form of the problem

$$q_i = -\kappa_{ij} \partial_j T \quad (\text{heat flux})$$

where κ_{ij} is the conductivity matrix.

$$\rho c_p \partial_t T + \partial_i q_i = H \quad (6.286)$$

$$\left(\rho c_p \frac{\partial T}{\partial t} - k \nabla^2 T = H \quad \text{for isotropic conductivity} \right) \quad (6.287)$$

Boundary conditions

$$\begin{aligned} T &= g \quad \text{on } \Gamma_g & (\text{essential}) \\ -q_i n_i &= h \quad \text{on } \Gamma_h & (\text{natural}) \end{aligned}$$

Initial conditions

$$T(\mathbf{x}, t = 0) = T_0(\mathbf{x}) \quad (6.288)$$

Weak form

$$(w, \rho c_p \dot{T}) + a(w, T) = (w, H) + (w, h)_\Gamma \quad (6.289)$$

$$(w, \rho c_p T(0)) = (w, \rho c_p T_0) \quad (6.290)$$

$$\dot{T} = \partial_t T = \frac{\partial T}{\partial t} \quad (6.291)$$

Galerkin approximation, in analogy to static case

$$T(\mathbf{x}, t) \simeq v(\mathbf{x}, t) + g(\mathbf{x}, t) \quad (6.292)$$

$$(w, \rho c_p \dot{v}) + a(w, v) = (w, H) + (w, h)_\Gamma - (w, \rho c_p \dot{g}) - a(w, g) \quad (6.293)$$

$$(6.294)$$

where we have assumed that the spatial derivatives are now approximated by FE as expressed by v but time is still left continuous,

Matrix assembly

$$v(\mathbf{x}, t) = \sum N_A(x) d_A(t) \quad (6.295)$$

approximation with shape functions
 N_A for all global nodes

The new matrix equation is

$$\text{initial condition} \quad \mathbf{M} \dot{\mathbf{d}} + \mathbf{K} \mathbf{d} = \mathbf{F}, \quad \mathbf{d}(0) = \mathbf{d}_0 \quad (6.296)$$

with

$$\text{assembly from element level } m^e : \quad \mathbf{M} \leftarrow m^e \quad (6.297)$$

$$\text{local nodes } a, b : \quad m^e = [m_{ab}^e]$$

$$\text{“mass” or “capacity” matrix :} \quad m_{ab}^e = \int_{\Omega^e} d\Omega N_a \rho c_p N_b$$

$$\text{conductivity matrix :} \quad \mathbf{K} \leftarrow \mathbf{K}^e \quad (6.298)$$

$$\mathbf{K}^e = [K_{ab}^e]$$

$$\text{same as the static case :} \quad K_{ab}^e = \int_{\Omega^e} d\Omega B_a^T D B_b$$

$$\mathbf{F} = \text{heat supply vector :} \quad \mathbf{F} \leftarrow \mathbf{f}^e \quad (6.299)$$

$$\mathbf{f}^e = [f_a^e]$$

$$\text{from BC's :} \quad f_a^e = \int_{\Omega^e} N_a H + \int_{\Gamma_h} d\Gamma N_a h - \sum (K_{ab}^e g_b + m_{ab} \dot{g}_b)$$

$$\mathbf{d}_0 = \mathbf{M}^{-1} \mathbf{d} ; \mathbf{d} \leftarrow \mathbf{d}^e \quad (6.300)$$

$$\text{Initial condition :} \quad \mathbf{d}^e = [d_a^e]$$

$$d_a = \int_{\Omega^e} N_a \rho c_p T_0 - \sum m_{ab} g_b^e(0)$$

(See [Hughes, 2000](#), p. 421).

The main difference with the static sets of equation for the heat equation is the introduction of the \mathbf{M} matrix and the need to solve eq. (6.296) as an ODE.

6.9.2 Solution of the semi-discrete heat equation

Solve

$$\begin{aligned} \mathbf{M} \dot{\mathbf{d}} + \mathbf{K} \mathbf{d} &= \mathbf{F} \\ \text{with IC } \mathbf{d} &= \mathbf{d}_0 \end{aligned} \quad (6.301)$$

Note that M, K are symmetric, M is positive definite and K is positive semi-definite (not pos. def. anymore). A general approach to solve eq. (6.301) is by the generalized trapezoidal method (see [Hughes, 2000](#), p. 459).

Generalized Trapezoidal Method

$$M \mathbf{v}^{n+1} + K \mathbf{d}^{n+1} = \mathbf{F}^{n+1}$$

$$\mathbf{d}^{n+1} = \mathbf{d}^n + \Delta t \mathbf{v}^{n+\alpha} \quad (6.302)$$

$$\mathbf{v}^{n+\alpha} = (1 - \alpha) \mathbf{v}^n + \alpha \mathbf{v}^{n+1} \quad (6.303)$$

where \mathbf{d}^n and \mathbf{v}^n are the approximations to $\mathbf{d}(t = t^n)$ and $\dot{\mathbf{d}}(t = t^n)$, respectively, with

$$t^{n+1} = t^n + \Delta t \quad (6.304)$$

as for the finite difference method. For the following α 's the methods in the table below are recovered.

α	
0	forward Euler, fully explicit
0.5	midpoint, Crank-Nicolson
1	backward Euler, fully implicit

\mathbf{v} - form implementation

(a) Start at $t = t_0$ with $\mathbf{d} = \mathbf{d}_0$ given for $n = 0$.

(b) Estimate $\mathbf{v}_0 \simeq \dot{\mathbf{d}}_0$ from

$$M \mathbf{v}_0 = \mathbf{F}_0 - K \mathbf{d}_0 \quad (6.305)$$

(c) Compute predictor

$$\tilde{\mathbf{d}}^{n+1} = \mathbf{d}_n + (1 - \alpha) \Delta t \mathbf{v}^n \quad (6.306)$$

Combine eq. (6.302) & (6.303) with (6.306)

$$\mathbf{d}^{n+1} = \tilde{\mathbf{d}}^{n+1} + \alpha \Delta t \mathbf{v}^{n+1} \quad (6.307)$$

into eq. (6.301)

$$(M + \alpha \Delta t K) \mathbf{v}^{n+1} = \mathbf{F}^{n+1} - K \tilde{\mathbf{d}}^{n+1} \quad (6.308)$$

(d) Solve eq. (6.308) for \mathbf{v}^{n+1} (rest is known)

(e) Advance $t = t + \Delta t$ and return to step 3.

Note that for the fully explicit case with $\alpha = 0$ and a “lumped” M matrix (6.308) (*i.e.* diagonal) does not involve any equation solving for time-stepping. M is lumped for p and c_p constant.

d - form implementation

Instead of eq. (6.308), we use (for $\alpha \neq 0$)

$$\frac{1}{\alpha \Delta t} (\mathbf{M} + \alpha \Delta t \mathbf{K}) \mathbf{d}^{n+1} = \mathbf{F}^{n+1} + \frac{1}{\alpha \Delta t} \mathbf{M} \tilde{\mathbf{d}}^{n+1} \quad (6.309)$$

to obtain \mathbf{d}^{n+1} , and then update

$$\mathbf{v}^{n+1} = \frac{\mathbf{d}^{n+1} - \tilde{\mathbf{d}}^{n+1}}{\alpha \Delta t} \quad (6.310)$$

The right hand side of eq. (6.309) is fast to compute for diagonal \mathbf{M} .

The generalized trapezoidal methods are $\alpha < \frac{1}{2}$ conditionally stable for

$$\Delta t \lesssim \frac{2}{(1 - 2\alpha) h^2} \quad (6.311)$$

where h is the smallest grid spacing in the mesh ($h \triangleq$ “mesh parameter”). For the fully explicit method ($\alpha = 0$), we recover

$$\Delta t \leq \frac{2}{h^2} \quad (6.312)$$

as in the finite difference method.

- For $\alpha \geq \frac{1}{2}$, the method is unconditionally stable. The best accuracy is obtained by the Crank-Nicolson scheme for $\alpha = \frac{1}{2}$, the extremes of $\alpha = 0$ and $\alpha = 1$ are only first order accurate. It is therefore a good idea to use the $\alpha = \frac{1}{2}$ scheme if the equation solving required for implicit methods is feasible.
- If the complete matrix inversion required for implicit schemes is not feasible, the element-by-element approach of *Hughes (2000)* p. 484 (preconditioned conjugate gradient with Crout factorization) can be used.
- For solutions of wave propagation (hyperbolic and parabolic - hyperbolic) problems, see *Hughes (2000)*, chap. 9.

Bibliography

Albarede, F. (1995), *Introduction to geochemical modeling*, Cambridge University Press.

Bathe, K.-J. (2007), *Finite Element Procedures*, Prentice-Hall, London.

Becker, T. W. (2000), Deterministic chaos in two state-variable friction sliders and the effect of elastic interactions, in *GeoComplexity and the physics of earthquakes*, *Geophysical Monographs*, vol. 120, edited by J. B. Rundle, D. L. Turcotte, and W. Klein, pp. 5–26, American Geophysical Union, Washington, DC.

Boschi, L., and A. M. Dziewoński (1999), ‘High’ and ‘low’ resolution images of the Earth’s mantle – Implications of different approaches to tomographic modeling, *J. Geophys. Res.*, *104*, 25,567–25,594.

Briggs, W. L., V. E. Henson, and S. F. McCormick (2000), *A multigrid tutorial*, 2 ed., The Society for Industrial and Applied Mathematics.

Crouzeix, M., and P. A. Raviart (1973), Conforming and nonconforming finite elements methods for solving the stationary Stokes equation, *Rev. Franc. d’Automat. Informat. Rech. Opér.*, *3*, 33–76.

Dabrowski, M., M. Krotkiewski, and D. W. Schmid (2008), MILAMIN: MATLAB-based finite element method solver for large problems, *Geochemistry, Geophysics, Geosystems*, *9*(Q04030), doi:10.1029/2007GC001719.

Dahlen, F. A., and J. Tromp (1998), *Theoretical Global Seismology*, Princeton University Press, Princeton, New Jersey.

Feigenbaum, M. J. (1978), Quantitative universality for a class of nonlinear transformations, *J. Stat. Phys.*, *19*, 25.

Fornberg, B. (1996), *A practical guide to pseudospectral methods*, Cambridge University Press, Cambridge UK.

Gerya, T. (2009), *Introduction to Numerical Geodynamic Modelling*, Cambridge University Press, Cambridge UK.

- Golub, G. H., and C. F. Van Loan (1996), *Matrix computations*, 3 ed., Johns Hopkins University Press.
- Hughes, T. J. R. (2000), *The finite element method*, Dover Publications.
- Ismail-Zadeh, A., and P. Tackley (2010), *Computational Methods for Geodynamics*, Cambridge University Press.
- Jacoby, W. R., and H. Schmeling (1981), Convection experiments and driving mechanism, *Geol. Rundschau*, 24, 217–284.
- King, S. D., D. A. Raefsky, and B. H. Hager (1990), ConMan: vectorizing a finite element code for incompressible two-dimensional convection in the Earth's mantle, *Phys. Earth Planet. Inter.*, 59, 195–207.
- Korenaga, J. (2008), Urey ratio and the structure and evolution of Earth's mantle, *Rev. Geophys.*, 46, doi:10.1029/2007RG000241.
- Lenardic, A., and W. M. Kaula (1993), A numerical treatment of geodynamic viscous flow problems involving the advection of material interfaces, *J. Geophys. Res.*, 98, 8243–8260.
- Lorenz, E. N. (1963), Deterministic nonperiodic flow, *J. Atmos. Sci.*, 20, 130.
- Loyd, S. J., T. W. Becker, C. P. Conrad, C. Lithgow-Bertelloni, and F. Corsetti (2007), Time-variability in Cenozoic reconstructions of mantle heat flow: plate tectonic cycles and implications for Earth's thermal evolution, *Proc. Nat. Acad. Sci.*, 104, 14,266–14,271.
- Malvern, L. E. (1977), *Introduction to the Mechanics of a Continuous Medium*, Prentice-Hall.
- Marone, C. (1998), Laboratory-derived friction laws and their application to seismic faulting, *Annu. Rev. Earth Planet. Sci.*, 26, 643–696.
- Moresi, L. N., and V. S. Solomatov (1995), Numerical investigations of 2D convection with extremely large viscosity variations, *Phys. Fluids*, 7, 2154–2162.
- Paige, C. C., and M. A. Saunders (1982), LSQR: an algorithm for sparse linear equations and sparse least-squares, *Trans. Math. Software*, 8, 43–71.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1993), *Numerical Recipes in C: The Art of Scientific Computing*, 2 ed., Cambridge University Press, Cambridge.
- Ricard, Y. (2007), Physics of mantle convection, in *Treatise on Geophysics*, edited by G. Schubert and D. Bercovici, Elsevier.
- Samuel, H., and M. Evonuk (2010), Modeling advection in geophysical flows with particle level sets, *Geochemistry, Geophysics, Geosystems*, in press, doi:10.1029/2010GC003081.

- Schubert, G., D. Stevenson, and P. Cassen (1980), Whole planet cooling and the radiogenic heat source contents of the Earth and Moon, *J. Geophys. Res.*, 85, 2531–2538.
- Schubert, G., D. L. Turcotte, and P. Olson (2001), *Mantle Convection in the Earth and Planets*, Cambridge University Press.
- Shewchuk, J. R. (1994), An introduction to the conjugate gradient method without the agonizing pain, available online at <http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf>, accessed 10/2008.
- Shewchuk, J. R. (2002), Delaunay refinement algorithms for triangular mesh generation, *Comput. Geom.: Theor. Appl.*, 22, 21–74.
- Smolarkiewicz, P. K. (1983), A simple positive definite advection scheme with small implicit diffusion, *Mon. Weather Rev.*, 111, 479–486.
- Spencer, R. L., and M. Ware (2008), *Introduction to Matlab*, Brigham Young University, available online, accessed 07/2008.
- Spiegelman, M. (2004), *Myths and Methods in Modeling*, Columbia University Course Lecture Notes, available online at <http://www.ldeo.columbia.edu/~mspieg/mmm/course.pdf>, accessed 06/2006.
- Suckale, J., J.-C. Nave, and B. H. Hager (2010), It takes three to tango 1: Simulating buoyancy-driven flow in the presence of large viscosity contrasts, *J. Geophys. Res.*, *in press*, doi:10.1029/2009JB006916.
- Tackley, P. J., and S. D. King (2003), Testing the tracer ratio method for modeling active compositional fields in mantle convection simulations, *Geochemistry, Geophysics, Geosystems*, 4, doi:10.1029/2001GC000214.
- Turcotte, D. L., and G. Schubert (2002), *Geodynamics*, 2 ed., Cambridge University Press, Cambridge.
- van Keken, P. E., S. King, H. Schmeling, U. Christensen, D. Neumeister, and M.-P. Doin (1997), A comparison of methods for the modeling of thermochemical convection, *J. Geophys. Res.*, 102, 22,477–22,495.
- Weijermars, R., and H. Schmeling (1986), Scaling of Newtonian and non-Newtonian fluid dynamics without inertia for quantitative modelling of rock flow due to gravity (including the concept of rheological similarity), *Phys. Earth Planet. Inter.*, 43, 316–330.
- Zhong, S. (2008), Iterative solutions of PDE, available online at http://anquetil.colorado.edu/szhong/TEMP/tutorial_mg.tar.gz, accessed 10/2008.

Zhong, S., M. T. Zuber, L. Moresi, and M. Gurnis (2000), Role of temperature-dependent viscosity and surface plates in spherical shell models of mantle convection, *J. Geophys. Res.*, *105*, 11,063–11,082.

Zhong, S. J., D. A. Yuen, and L. N. Moresi (2007), Numerical methods in mantle convection, in *Treatise in Geophysics*, vol. 7, edited by G. Schubert and D. Bercovici, pp. 227–252, Elsevier.