Numerical Modeling of Earth Systems

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

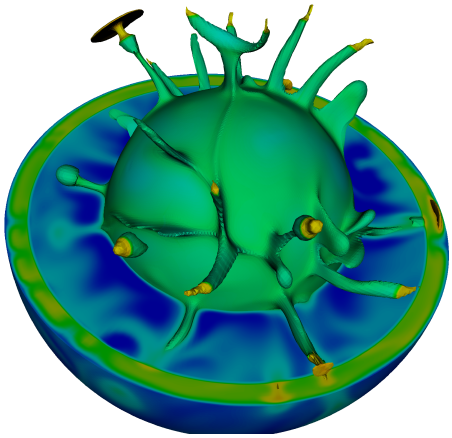
Lecture notes for USC GEOL557, v. 1.1.2

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

and

Boris J. P. Kaus
University of Mainz, Germany

October 20, 2014



Chapter 1

Preface

1.1 Purpose of these lecture notes

Numerical modeling in the solid Earth Sciences has come a long way over the last fourty years. Long-standing questions such as that of how surface tectonics arises from mantle convection can now be addressed with near-realistic convective vigor and material behavior.

The modeling software used for such research can be increasingly complex, which is why scientists often rely on existing codes, rather than writing them from scratch. Assuming the use of community software (such as those provided by the Computational Infrastructure for Geodynamics (CIG), `geodynamics.org`) will continue to increase over the next years, geodynamics as a field faces the challenge to educate graduate students in numerical analysis without having each PhD student write their own code. This one semester course is supposed to help address this challenge and is geared toward all Earth science or engineering students (grad and advanced undergrad), and not just geophysicists.

The course discusses the numerical solution of problems arising in the quantitative modeling of Earth systems. The focus is on continuum mechanics problems as applied to geological processes in the solid Earth, but the numerical methods have broad applications including in geochemistry or climate modeling. The quantitative skills which are to be learned are therefore useful for all Earth scientists, but the focus of the class is on mantle convection and seismology type problems.

After an introduction with few details on numerical analysis and programming, we briefly discuss ordinary differential equations (ODEs). Thereafter, the class spends the majority of the time discussing finite difference (FD) and finite element (FE) solutions to partial differential equations (PDEs). An example syllabus for how to use these notes is given in sec. 9.[1] A brief review of basic math and continuum mechanics fundamentals is provided in secs. 6 and 7, respectively.

Every subject in this class is accompanied by hands-on Matlab programming exer-

[1] The syllabus in sec. 9 is *not the 2013 version*, the latter can be found on the course web page.

cises. The course web site, geodynamics.usc.edu/~becker/teaching-557.html has the required Matlab files, and splits the associated exercises up into single documents rather than the complete set of notes provided here. The fact that many of the exercises are self-contained also means that some material, such as the governing equations, are repeated in several instances in these lecture notes.

We chose Matlab as a programming language because of its ease in developing and debugging, as well as the built-in visualization capabilities. The fact that the language is interpreted, and not compiled, does somewhat limit the direct applications of the implementations that are discussed, pretty much to 2D problems, yet at high efficiencies (*e.g. Dabrowski et al.*, 2008). Solutions for the Matlab exercises are available for instructors upon request, and a brief introduction to Matlab exercise is provided in sec. 8.

Students should ideally have had exposure to calculus, some linear algebra, a classic, introductory geodynamics (or continuum/rock mechanics) course (*e.g.*, based on *Turcotte and Schubert*, 2002), and have some introductory level knowledge of computer programming. 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 primers on calculus, linear algebra, continuum mechanics, and computer programming. After learning some basic programming skills, the students are then guided through increasingly more involved programming using the problem sets as examples.

These notes are an attempt at an almost self-contained, introductory survey of numerical modeling. This necessitates skimming 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, for which other textbooks exist. Our goal is to provide all students with a sufficient working knowledge to solve simple research problems by reusing the Matlab codes introduced in problem sets, or by writing their own software. The basic insights into numerical analysis that are conveyed in this class should also help make students educated and empowered users and developers of more complex, existing community software.

1.2 Acknowledgments

Parts of the original course notes are inspired by, or partially follow very closely, the treatment in the numerical analysis lecture notes by *Spiegelman* (2004), the notes by *Schmeling* (1994), and the textbook by *Press et al.* (1993) on numerical analysis, and the textbook by *Hughes* (2000) on finite element methods.

Some of the finite difference exercises and parts of the scaling homework assignment are loosely based on an ETH Zürich course taught by Yuri Podladchikov. Daoyuan Sun provided the wave propagation exercise, and 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 (*Schmeling*, 1994). A web blog at `http://gwlab.blogspot.com/2012/08/inverse-transformation-of-parametric.html` was helpful in coming to grips with inverse transformations.

Students who took the class at USC Earth Sciences in the Fall of 2005, 2008, and 2013 provided valuable feedback.

USC PhD candidate Francois Cadieux helped with the type setting of the original lecture notes in 2010, and also provided some additional material.

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

1.3 Abbreviations used

BC Boundary conditions

FD Finite differences

FE Finite elements

IC Initial condition

ODE Ordinary differential equation

PDE Partial differential equation

PDE Partial differential equation

ODE Ordinary differential equation

BC Boundary conditions

1.4 Typesetting conventions

Mathematical symbols are denoted by a for scalars, $\boldsymbol{a}$ for vectors, and A for tensors/matrices.

1.5 Other resources

This text is meant to be fairly self-contained, but there is some related materail which we refer to often. This includes the following online resources:

- *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).

When used for a class, some accompanying textbooks we recommend are:

- *Numerical Recipes* by *Press et al.* (1993), 2^{nd} or 3^{rd} edition;

- *Geodynamics* by *Turcotte and Schubert* (2002) for background on geodynamics;
- *The finite element method* by *Hughes* (2000), for further detailes on the finite element method.

For finite elements, for example, there are, of course, a number of good textbooks. Those include *Kwon and Bang* (1996), which provides a clear, step-by-step, introduction with many Matlab program examples, and the classic by *Bathe* (2007) for a comprehensive reference. For computational geodynamics, there is now *Ismail-Zadeh and Tackley* (2010) for a broader, but less-detailed overview of general methods and *Gerya* (2009) for more details on finite difference approaches.

Background material Short discussions of basic math, continuum mechanics, and Matlab are found in secs. 6, 7, and 8, respectively. Readers not familiar with this material may wish to review those chapters.

Availability and contact A PDF of the lecture notes and Matlab exercises as used for a graduate class at USC (GEOL557) can be found on course web site
`http://geodynamics.usc.edu/~becker/teaching-557.html`

For any questions, please contact

Thorsten W. Becker, University of Southern California: twb-at-usc-edu.

Part I

Introduction

Chapter 2

Introduction to numerical geodynamics

2.1 Numerical methods in the Earth Sciences

TO BE EXPANDED GREATLY, MORE DISCUSSION.

2.1.1 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. 5.4).
- Create, or understand, as much code and theory as possible yourself, no matter if you are geophysicist or geologist.
- There are no "Easy" or "Model my field data" buttons, but you don't have to be a math-whiz either!

2.1.2 Goals

- Provide you with a basic understanding of numerical modeling, using solid Earth science problems as an example.
- Allow you to solve simple research problems using tools presented here, and, more importantly, allow you to write new programs and solve different scientific questions yourself independently.
- Help you become an informed, empowered user of sophisticated numerical codes.
- Introduce some math and computer science along the way.

This text cannot

- be mathematically thorough (no proofs, etc.)
- be comprehensive
- be cutting edge in all aspects of numerical analysis

because we need to cover a lot of ground.

2.1.3 Overview of applications of numerical methods for Earth sciences

In general, numerical methods are important for *forward* and *inverse* problems. In particular, we may focus on

Differential equations of the partial (PDE) or ordinary (ODE) kind, which can be solved with

- finite difference methods
- integral methods, such as finite elements and spectral methods.

Inverse problems where a structural or physical model of the Earth is inferred from (a potentially very large) set of data. Methods may include

- linear matrix inversion, least squares (with challenges related to those in finite element methods)
- Monte Carlo, simulated annealing
- Genetic Algorithms

Example applications of numerical modeling in the Earth Sciences

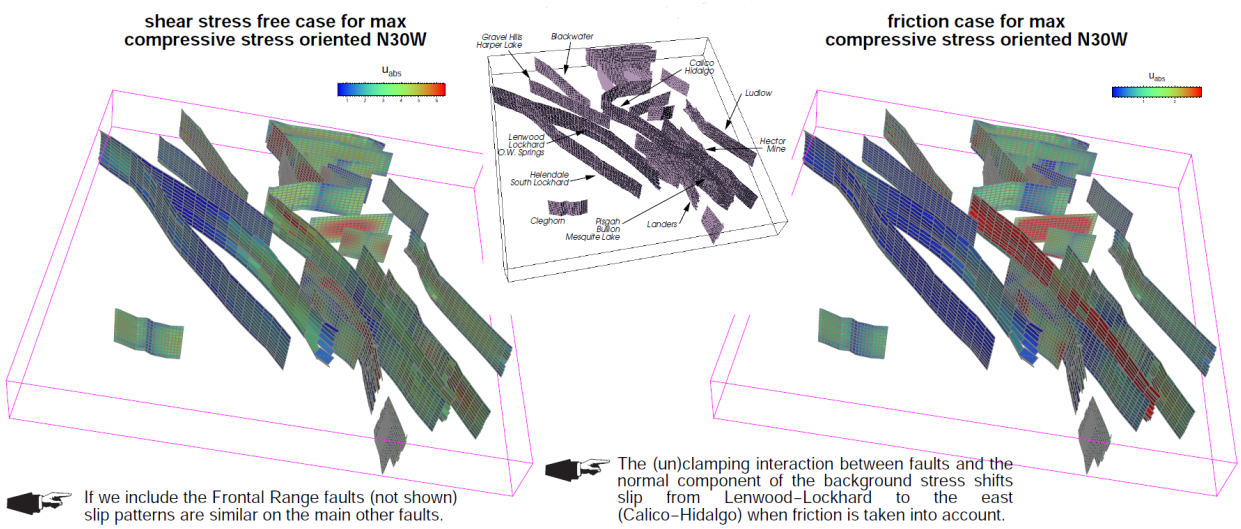


Figure 2.1: Estimated slip distribution on southern California faults for shear stress (left) and Coulomb stress free, simultaneous slip, boundary element method computations (modified from *Becker and Schott*, 2002). This method (*e.g. Crouch and Starfield*, 1983) uses Greens functions (here computed following *Okada*, 1992) and either summation for estimating τ given slip distributions, $\boldsymbol{u}$, on faults, or solution of a large, linear system of equations to infer $\boldsymbol{u}$ for specified stress on the faults.

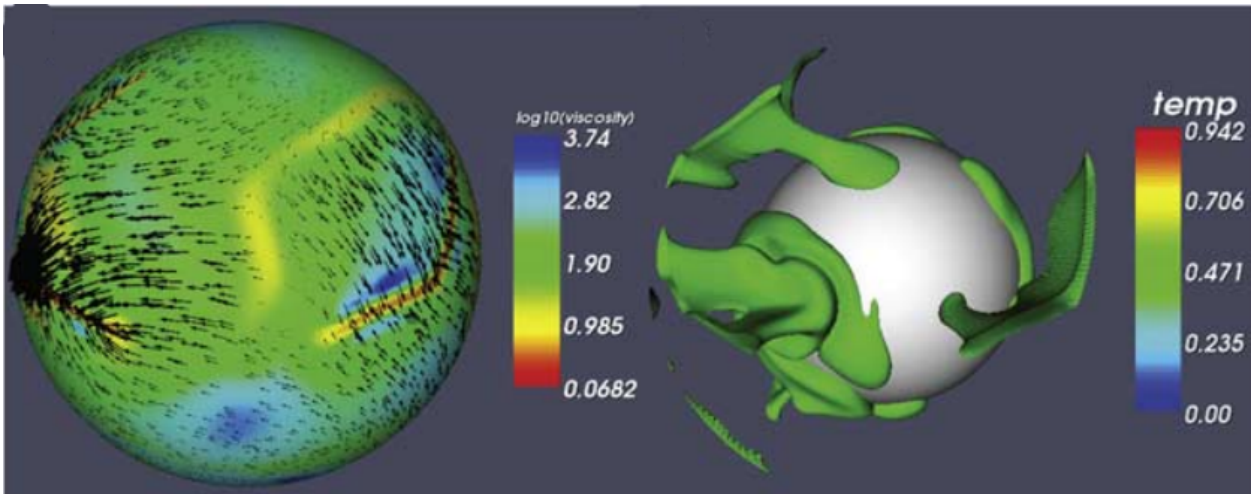


Figure 2.2: Spherical, visco-plastic mantle convection computation results showing plate-like surface motions (left: velocity vectors on top of viscosity, right: cold temperature isosurface in the interior showing one-sided subduction due to slab rollback), modified from *Foley and Becker* (2009). We use the CitcomS (*Zhong et al.*, 2000) finite element software to solve the Stokes and energy equations for an infinite Prantl number, incompressible fluid (*cf. Zhong et al.*, 2007).

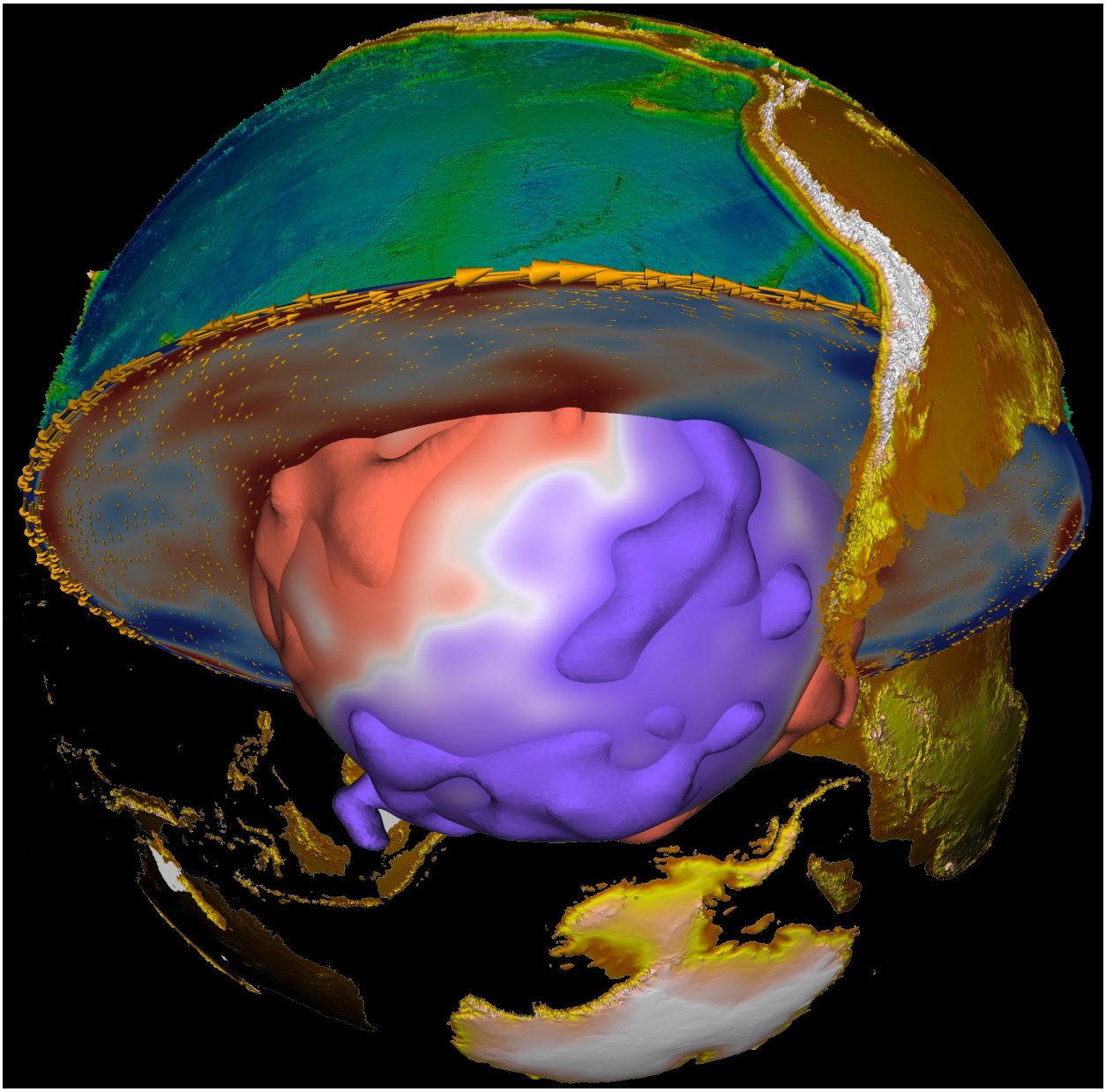


Figure 2.3: Global mantle circulation computation using mantle tomography to infer density distributions in the mantle and solving Stokes equation (modified from *Becker and Faccenna*, 2011), see (*cf. Hager and O'Connell*, 1981; *Zhong et al.*, 2000; *Becker*, 2006). Visualization with Paraview (*Kitware, Inc.*, 2006).

2.1.4 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

2.2 Examples of applications for numerical methods

2.2.1 Linear inverse problems

→ computational linear algebra

2.2.2 Ordinary differential equations

Examples:

$$\frac{\partial \boldsymbol{y}}{\partial t} = f(\boldsymbol{y}) \tag{2.1}$$

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{v} \tag{2.2}$$

→ Runge-Kutta, Burlisch-Stoer integration methods, for example (this will be dealt with next, see 3.2).

2.2.3 Partial differential equations

Many problems in the Earth sciences can be described by linear partial differential equations of second order. For constant coefficients k, we can write a general equation

$$k_1\frac{\partial^2 f}{\partial x^2} + k_2\frac{\partial^2 f}{\partial y^2} + k_3\frac{\partial^2 f}{\partial z^2} + k_4\frac{\partial^2 f}{\partial t^2} \quad + \tag{2.3}$$

$$p_1\frac{\partial f}{\partial x} + p_2\frac{\partial f}{\partial y} + p_3\frac{\partial f}{\partial z} + p_4\frac{\partial f}{\partial t} \quad + \tag{2.4}$$

$$f(x,y,z,t) = g(x,y,z,t). \tag{2.5}$$

Another common representation of a general, second order equation as a function of two variables is

$$A\frac{\partial^2 f}{\partial x^2} + B\frac{\partial^2 f}{\partial x \partial y} + C\frac{\partial^2 f}{\partial y^2} + D\frac{\partial f}{\partial x} + E\frac{\partial f}{\partial y} + Ff = R(x,y), \tag{2.6}$$

where x and y need not be spatial derivatives, *e.g.* y may be time t. Special cases of these general equations are then, in analogy to planar sections of a cone,

elliptic PDEs : $B^2 < 4AC$, or all k_i are non-zero and have the same sign;

hyperbolic PDEs : $B^2 > 4AC$, or all k_i are non-zero and one has a different sign from the others;

parabolic PDEs : $B^2 = 4AC$, or all k_i but one are non-zero and have the same sign.

Distinguishing between these different types of PDEs is important because the types determine which *boundary conditions* and which *initial conditions* are neded. For example, elliptic problems require boundary conditions, but parabolic and hyperbolic problems als require initial conditions.

As for the boundary types and conditions, the following rules apply

type	*boundary*	*conditions*
hyperbolic	open	Cauchy
parabolic	open	Dirichlet or Neumann
elliptic	closed	Dirichlet or Neumann

where the boundary condition type means

Dirichlet : specifying f itself;

Neumann : specifying $\frac{\partial f}{\partial n}$ where n is the *normal derivative* on the boundary, $\frac{\partial f}{\partial n} = \nabla f \cdot \boldsymbol{n}$, where $\boldsymbol{n}$ is the normal at each point;

Cauchy : where both f and $\frac{\partial f}{\partial n}$ need to be specified.

Stationary field problems (boundary value problems)

We are looking for a stationary ($\partial f/\partial t = 0$) solution for an elliptic PDE given boundary conditions on the edge ∂S of the domain S. ($k_4 = 0$ does not lead to a parabolic PDE in this case because the equation is not a function of time t in this case.) Examples include the

- stationary heat equation
$$k\nabla^2 T + \rho H = 0, \tag{2.7}$$
- gravity potential equation
$$\nabla^2 \Phi = -4\pi G\rho. \tag{2.8}$$

For an isotropic and homogeneous medium, these equations correspond to the Poisson

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = g(x, y, z), \tag{2.9}$$

or the Laplace equation

$$\nabla^2 f = 0. \tag{2.10}$$

Time-dependent field problems

If f can be a function of time, we can write

$$\nabla^2 f = g(x,y,z,t) + \mu \frac{\partial^2 f}{\partial t^2} + \gamma \frac{\partial f}{\partial t}, \tag{2.11}$$

where boundary conditions may also be a function of time t, and we need additional initial conditions which provide $f(x,y,z,t_0) = \Psi(x,y,z)$.

For $\mu = 0$, this is a parabolic PDE, with the classic example of the time-dependent heat equation, an example of a diffusion equation

$$k\nabla^2 T + \rho H = \rho c_p \frac{\partial T}{\partial t}, \tag{2.12}$$

with thermal diffusivity

$$\kappa = \frac{k}{\rho c_p}. \tag{2.13}$$

If $\gamma = 0$ and $\mu > 0$, the equation is a hyperbolic PDE, in the case of $g = 0$ a *wave equation*,

$$\frac{\partial^2 f}{\partial t^2} = c^2 \nabla^2 f, \tag{2.14}$$

where c is a velocity.

If we are interested in periodic solutions of eq. (2.14) only, we can use *normal modes* and turn the hyperbolic PDE in a eigenvalalue problem by assuming

$$f = F(x,y,z)e^{i\omega t}, \tag{2.15}$$

which yields

$$-\omega^2 F = c^2 \nabla^2 F. \tag{2.16}$$

The resulting frequencies ω are the eigenfrequencies, *e.g.* of the vibrational modes of the whole Earth induced by earthquakes.

In case both $\mu \neq 0$ and $\gamma \neq 0$, we have a damped wave equation. In this case, two initial conditions are needed,

$$f(x,y,z,t_0) = \Psi(x,y,z) \tag{2.17}$$

$$\frac{\partial f}{\partial t}(x,y,z,t_0) = \xi(x,y,z) \tag{2.18}$$

2.2.4 Numerical solution methods

Finite Differences (FD)

FD approximate differentials by Taylor series, then approximate equations to solve in a pointwise manner.

Take

$$\frac{\partial^2 T}{\partial x^2} - \frac{H(x)}{\kappa} = 0, \tag{2.19}$$

which can be written as

$$\frac{\partial^2 u}{\partial x^2} + s = 0. \tag{2.20}$$

We may approximate this equation with finite differences as

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} = -s_i. \tag{2.21}$$

Pros

- conceptually simple, approximates the PDE

Cons

- bad for sharp contrasts
- bad for complicated geometries
- code usually needs to be written from scratch for new problems

Finite Elements (FE)

The FE method is complicated conceptually, and provides an approximate solution to the solutions, rather than the equations themselves. It is an integral method, where we take eq. (2.20), multiply by a *virtual displacement* w and integrate over x

$$-\int_a^b w\frac{\partial^2 u}{\partial x^2} - \int_a^b ws = 0, \tag{2.22}$$

which gives with $w(a) = 0$ and integration by parts

$$\int_a^b \frac{\partial w}{\partial x}\frac{\partial u}{\partial x} = hw(b) + \int_a^b ws, \tag{2.23}$$

where $h = \frac{\partial u}{\partial x}(b)$. This is the weak form of the PDE, and picking w as a low order polynomial forms the basis of the FE method.

Pros

- good for sharp boundaries
- good for complicated geometries
- allow easy lateral mesh refinement
- usually, only minor modifications to the code are needed for new problems

Cons

- conceptually a bit more involved, approximates the solution of the PDE
- coding initially more complicated
- need to carefully choose elements, integration methods, etc.

Spectral element methods, a variant of finite elements

Spectral methods

Spectral methods expand the spatial solution as harmonic functions (can use FFT), and solve time evolution as an ODE for coefficients.

Example

$$\frac{\partial^2 u}{\partial x^2} = s, \tag{2.24}$$

assuming that s is periodic (*i.e.* $s(x) = s(x + 2\pi)$), we write u and s as Fourier series

$$u = \sum a_j e^{ijx} \tag{2.25}$$
$$s = \sum b_j e^{ijx} \tag{2.26}$$
$$\tag{2.27}$$

and substitute, which gives (for continuous, second order differentiable u)

$$\sum -a_j j^2 e^{ijx} = \sum b_j e^{ijx}, \tag{2.28}$$

from which

$$a_j = -\frac{b_j}{j^2}. \tag{2.29}$$

The approach then consists in solving by computing b_j by FFT of s, then use eq. (2.29), then find f from inverse FFT of a_j.

Pros

- just for homogeneous media

Cons

- non-local
- poor performance for lateral variations in material properties (need to iterate)

Distinct element methods

Boundary element methods

Gas lattice and other microscopic methods

These methods do not provide a clear relationship between micro rules and the continuum PDEs but are efficient for some problems.

2.3 Computing

Here, we very briefly review some hardware related issues, give a few general 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.

2.3.1 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 take up four bytes and is accurate up to $\sim 5 \cdot 10^{-7}$, a double precision float up to $\sim 5 \cdot 10^{-15}$. With a 32 bit operating system, the largest number you can represent is $2^{31} - 1 \sim 2.1$ billion. (Aside: this might seem like a big number but is not, as it corresponds roughly to a bit more then 800^3 resolution.)

→ 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 $abs(x) < \varepsilon \quad (abs(x) = |x|)$ where ε depends on implementation.

→ The detailed storage depends on the hardware, "big endian" *vrs.*"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 have likely have at least ~ 2 GB of Random Access Memory or RAM (as opposed to hard drive space) meaning you can store how many floats and doubles? 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.
GRAPHICAL PROCESSING UNITS

2.3.2 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 run time, can be slow
- may require paying license fees

Libraries

- NETLIB
- BLAS
- LAPACK
- PETSc

Low level languages

→ compile before run

Examples: "serial": C, Fortran 77, Fortran 90/95
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!

2.3.3 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".
```

2.3.4 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.
   - ensure that each subroutine gives error messages, in case non sensible input arguments are given.
   - do not ignore compiler warnings

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 re-usability, 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.

2.3.5 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, because memory access is faster that way. The storage depends on the computer language (C *vrs.* FORTRAN). *e.g.* in Matlab, you would write

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

   to multiply x elements by those of y and NOT the other way around,

```
 for j = 1:m     % row i computations across all columns j first
   for i = 1:n     % this will make things jump around in memort
                   % and slow things down
    x(i,j) = x(i,j) * y(i,j);
  end
end
```

Even better, in Matlab (and languages such as FORTRAN90) you can *vectorize*, *i.e.* write symbolically for a vector $\boldsymbol{x}$

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

if you want to add a scalar to each element, or

```
x = x .* y
```

for the example above. Matlab internally takes take that the looping is taking care of in the most efficient way. This can make a huge difference, vectorize whenever you can in.

3. *Avoid if statements* as much as possible. For example, if this test

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

if optional and usually zero, comment it out using pre-processor directives. *I.e.* in C, you would write the code like so

```
#ifdef DEBUG
    % code here for debugging version of program
#else
    % code here for the regular version of program
#end
```

and compile the program with or without

```
gcc -DDEBUG
```

depending on if you want those statements to be executed when the program runs.

4. *Pre-compute* common factors 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. In Matlab, it's better still to use the vectorized version, `x=x.*fac`.

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).

7. *Use version control*!

   Use version control packages (such as subversion, RCS) during code development, as this might safe you an immense amount of time when you're trying to track down where and when that bug crept into the code.

2.4 Scaling analysis and non-dimensional numbers

Reading

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

2.4.1 Scaling analysis

Scaling analysis refers to order of magnitude estimates on how different processes work together and control a system. While this is a text on numerical analysis, such theoretical considerations are very useful if we are interested in getting a quick idea of the values that are of relevance for a problem, and for the order of magnitude for solutions. Comparing these estimates with the numerical results is always good practice and part of a basic set of plausibility checks that have to be conducted.

For example, shear stress τ (in units of Pa) for a Newtonian viscous rheology with viscosity η (in units of Pa s) is given by the simple constitutive law

$$\tau = 2\eta\dot{\varepsilon} \tag{2.30}$$

where $\dot{\varepsilon}$ is the strain-rate (in units of s^{-1}). Say, we wish to estimate the typical amplitudes of shear stress in a part of the crust that we know is being sheared at some (*e.g.* plate-) velocity u over a zone of width L. The strain-rate in 3-D is really a tensor, $\dot{\varepsilon}$, with 3×3 components that depends on the spatial derivatives of the velocity like so

$$\dot{\varepsilon} = \dot{\varepsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \tag{2.31}$$

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

$$\dot{\varepsilon} \propto \frac{v}{L} \tag{2.32}$$

where $\propto$ means "proportional to", or "scales as", to indicate that eq. (2.32) is not exact. Assuming we know the viscosity η, we can then estimate the typical stress in the shear zone to be

$$\tau \propto 2\eta\frac{v}{L}. \tag{2.33}$$

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. [1]

Note I: Whenever you work out, or type up, a new equation, it is always a good idea to check if the units on both sides make sense.

Note II: The scaling of velocities and stress for a buoyancy driven problem, such as the Stokes sinker discussed below, is entirely different!

2.4.2 Non-dimensionalization

A complementary approach that also takes into account the order of magnitude 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

$$\rho \frac{D\boldsymbol{v}}{Dt} = -\nabla p_d + \eta \nabla^2 \boldsymbol{v} + \rho_0 \alpha T \boldsymbol{g} \tag{2.34}$$

where D is the total, Lagrangian derivative operator

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla, \tag{2.35}$$

$\boldsymbol{v}$ 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 $\boldsymbol{g}$ gravitational acceleration. One can now choose (as mentioned before 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 (see, *e.g.*, discussion in *Ricard*, 2007).

Let us assume that we are dealing with a box heated from below and cooled from above, *i.e.* held at constant temperature difference of $\Delta T = T_{top} - T_{bot}$, with no internal heating (Rayleigh-Benard problem). A typical choice for a characteristic timescale is to use the diffusion time that can be constructed from the thermal diffusivity, κ, in the energy equation

$$\frac{DT}{Dt} = \kappa \nabla^2 T \tag{2.36}$$

[1] We will always use SI units unless it's inconvenient for Earth applications, where we might use multiples of SI units such as cm/yr instead of m/s for velocities. Also note that one year has roughly $\pi \cdot 10^7$ s (accurate up to 0.5%), *i.e.* 1 cm/yr is $\approx 3.2 \cdot 10^{-10}$ m/s, and that you should account for leap years for geological time conversions, meaning that $365.25 \times 24 \times 60 \times 60$ is the right number of seconds per year.

(no heat sources) that couples with the momentum equation, eq. (2.34). Because κ has units of length^2/time, any diffusion-related time scale t_d for a given length l has to work out like

$$t_d \propto \frac{d^2}{\kappa}, \tag{2.37}$$

by dimensional analysis. This relationship is hugely important for all diffusional processes.

Using the characteristic quantities f_c which result from this scaling and using $l = d$, for all variables in eq. (2.34) and eq. (2.36), all other properties can be derived, *e.g.*

$$v_c = \frac{d}{t_c} \quad \dot{\varepsilon}_c = \frac{v_c}{d} \quad \tau_c = \eta\dot{\varepsilon}_c \quad T_c = \Delta T \tag{2.38}$$

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. (2.34) can then be written as

$$\frac{1}{Pr}\frac{D'\boldsymbol{v}'}{D't'} = -\nabla' p' + (\nabla')^2\boldsymbol{v}' - RaT'\boldsymbol{e}_z \tag{2.39}$$

where we've used $\boldsymbol{g} = g\boldsymbol{e}_z$. I.e., all material parameters have been collected in two unit-less numbers after non-dimensionalization, the Prandlt number,

$$Pr = \frac{\eta}{\rho\kappa} \tag{2.40}$$

and the Rayleigh number [2]

$$Ra = \frac{\rho_0 g\alpha\Delta T d^3}{\kappa\eta}. \tag{2.41}$$

Exercise: Verify this recasting of the Navier-Stokes equation by plugging in the non-dimensionalized quantities.

Often, we then just drop the primes and write the equation like so

$$\frac{1}{Pr}\frac{D\boldsymbol{v}}{Dt} = -\nabla p + \nabla^2\boldsymbol{v} - RaT\boldsymbol{e}_z \tag{2.42}$$

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 I: The non-dimensional versions of the equations are also the best choice if you want to write a computer program for a physical problems. Using non-dimensionalized

[2] In the derivation above, we have assumed that the system is heated from below and viscosity is constant. The Rayleigh number is therefore valid for this bottom-heated case only. In Earth's mantle, internal heating (due to decay of radioactive elements in the mantle) is probably at least equally important (*Jaupart et al.*, 2007; *Lay et al.*, 2008), and the Rayleigh-number in this case is defined differently. Also, rock viscosity depends on a range of quantities, including temperature and strain-rate, making it imperative to properly (log) average viscosity when computing effective Rayleigh numbers (*e.g.* *Christensen*, 1984).

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, reducing round-off error (*e.g.* $\boldsymbol{v}$, what is the order of magnitude of η and of $|\boldsymbol{v}|$ for mantle convection?).

Note II: 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.

Note III: You'll also note that a few geodynamics papers will not provide the scaled quantities used so that you can go back to SI units; sometimes this is because the values used for the parameters in the models stray significantly from typical Earth values.

Particularly the Rayleigh number is key for mantle convection, because we typically use the infite Prandtl number approximation, ($Pr \rightarrow \infty$, why?). In this case, eq. (2.34) simplifies to the Stokes equation,

$$\eta \nabla^2 \boldsymbol{v} = \nabla p_d - \rho_0 \alpha T \boldsymbol{g}. \tag{2.43}$$

Both Pr and Ra are discussed 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; *Faccenna et al.*, 1999). 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).

2.4.3 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 (*i.e.* providing references) for individual parameters before com-

puting 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

$$Pe = \frac{dv}{\kappa} \tag{2.44}$$

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

$$Re = \frac{vd}{\nu} = \frac{vd\rho}{\eta}. \tag{2.45}$$

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

(e) Deborah number for the subducting oceanic lithosphere, and for a laboratory experiment on rock deformation. The Deborah number can be defined as

$$De = \frac{t_r}{t_p} \tag{2.46}$$

where you can use a Maxwell time

$$t_M = \frac{\eta}{\mu} \tag{2.47}$$

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 Stokes velocity 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, $F_S \propto \tau A \propto A\eta\dot{\varepsilon}_c$. 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.

   Note: The full equation for a Stokes sinker is only very weakly dependent on the viscosity of the sinker, η_s, itself, but scales mainly with the ambient viscosity η. For $\eta_s/\eta \to \infty$ and $\eta_s/\eta \to 0$, the prefactor changes from $2/9$ to $1/3$, respectively (see further discussion in *Becker and Faccenna*, 2009, for the subduction context).

   (b) For flow induced by a Stokes sinker, does the stress scale with η and/or $\Delta\rho$? How does that compare with the velocities?

   Note: The scaling of v and τ with combinations of $\Delta\rho$ and η are among the most fundamental relations of mantle dynamics (velocities v might be the plate velocities, dynamic topography scales with τ, for example).

   (c) Estimate the Stokes velocity by dimensional analysis as in (a), but now assuming that the viscosity of the fluid obeys a power-law,

$$\tau^n \propto \eta\dot{\varepsilon} \tag{2.48}$$

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

$$\tau \propto \eta\dot{\varepsilon} \tag{2.49}$$

   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 $\tau\,(\tau/\mu)^{n-1} = \eta\dot{\varepsilon}$, where η is a material parameter, but you may use eq. (2.48) 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 $v(z)$.

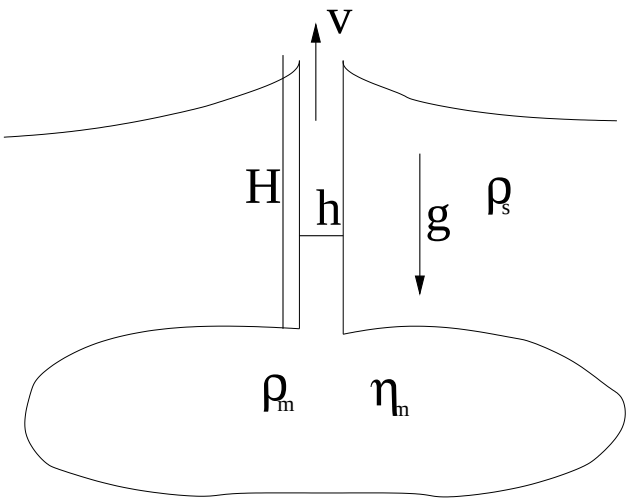


Figure 2.4: Illustration of the geometry of the volcanic eruption problem.

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 Figure 2.4.

Part II

Ordinary differential equations

Chapter 3

Solution of ordinary differential equations

3.1 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:

$$\frac{dy}{dx} = f(x), \quad \text{which can be solved by integration} \tag{3.1}$$

$$y = \int f(x)\,\mathrm{d}x + C \tag{3.2}$$

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.*

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x) \tag{3.3}$$

is "second order". However, we can always reduce ODEs to sets of first order equations. For eq. (3.3), define

$$\frac{dy}{dx} = z(x), \quad \text{then} \tag{3.4}$$

$$\frac{dz}{dx} = r(x) - q(x)\,z(x). \tag{3.5}$$

Or, in general

$$\frac{dy_i(x)}{dx} = f_i(x, y, \ldots, y_N) \quad \text{or} \quad \frac{d\boldsymbol{y}}{dx} = \boldsymbol{f}(x, \boldsymbol{y}) \tag{3.6}$$

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

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

3.1.1 Initial Value Problems

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

- *spring slider systems*

$$\frac{d\tau}{dt} = k(v - v_0) \qquad \tau = k \cdot x \qquad \text{(Hooke's law)}$$
$$\tau = f(v, \theta_1, \theta_2, \ldots) \qquad \text{(friction law)}$$
$$\frac{d\theta_i}{dt} = f(v, \theta_i) \qquad \text{(state variable evolution)}$$

- *geochemical box models*

$$\frac{d\boldsymbol{y}}{dt} = f(\boldsymbol{y}); \qquad \text{(concentration and fluxes)}$$

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

$$\boldsymbol{y}(\boldsymbol{x}, t) = \sum_{n=1}^{N} y_i(t)\, f_i(\boldsymbol{x})$$

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

- *parametrized convection models*

$$\dot{Q} = c_p M \frac{dT}{dt} = H(t) - Q_c(t) = H(t) - f(T, t) \tag{3.7}$$

- *particle tracking*

$$\frac{dc}{dt} = f(\boldsymbol{x}, t) \quad \text{for each particle} \tag{3.8}$$
$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v} \tag{3.9}$$

which is equivalent to the advection equation

$$\frac{\mathrm{D}c}{\mathrm{D}t} = \frac{dc}{dt} + \boldsymbol{v} \cdot \nabla \boldsymbol{c} = f \tag{3.10}$$

3.2 Solution of initial value problem

Let's consider the solution of

$$\frac{dy}{dt} = f(t, y(t)) \tag{3.11}$$

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

$$y(t) = y_0 + \int_{t_0}^{t_f} f(t, y(t))\, \mathrm{d}t. \tag{3.12}$$

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

$$I = \int_{t_i}^{t_i+h} f(t, y(t))\, \mathrm{d}t \tag{3.13}$$

as cheaply as possible numerically. The simplest approximation is

$$I = f(t_i, y(t_i))\, h \qquad \text{such that} \qquad \text{(from (3.12))}$$
$$y(t_i + h) = y(t_i) + h \cdot f(t_i, y(t_i)) \tag{3.14}$$

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 3.1, which shows that (3.14) is just a simple extrapolation of y based on the slope at t_i, which is given by equation (3.11). If y has some curvature to it, the Euler scheme will lead to large errors quickly!

We can Taylor expand (eq. 6.4) y around t_0 to get

$$y(t) \approx y(t_0) + (t - t_0)\frac{dy(t_0)}{dt} + \frac{(t - t_0)^2}{2!}\frac{d^2y(t_0)}{dt^2} + \frac{(t - t_0)^3}{3!}y''' + \dots \tag{3.15}$$

to gain some mathematical insight into the accuracy of the Euler scheme. For our problem, (3.15) becomes

$$y(t_i + h) \approx y(t_i) + h \cdot f(y(t_0), t_0) + \frac{h^2}{2}\frac{d^2y}{dt^2} + \dots \tag{3.16}$$

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

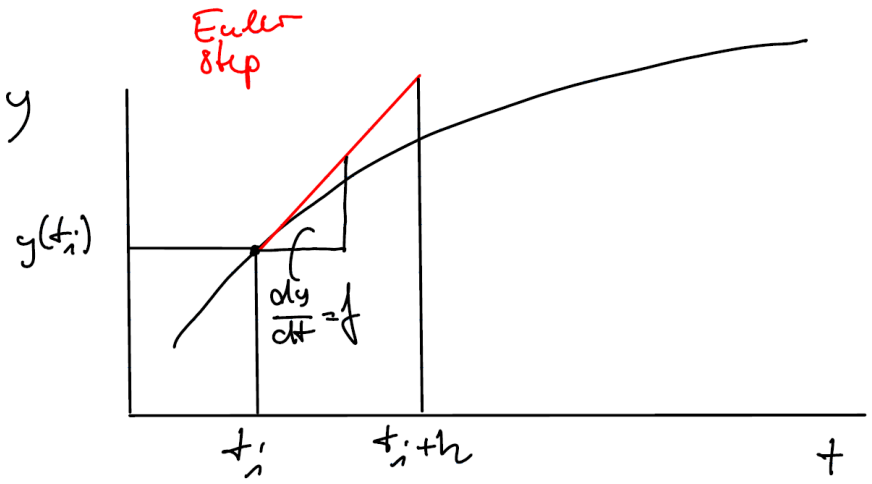


Figure 3.1: Sketch illustrating the Euler method.

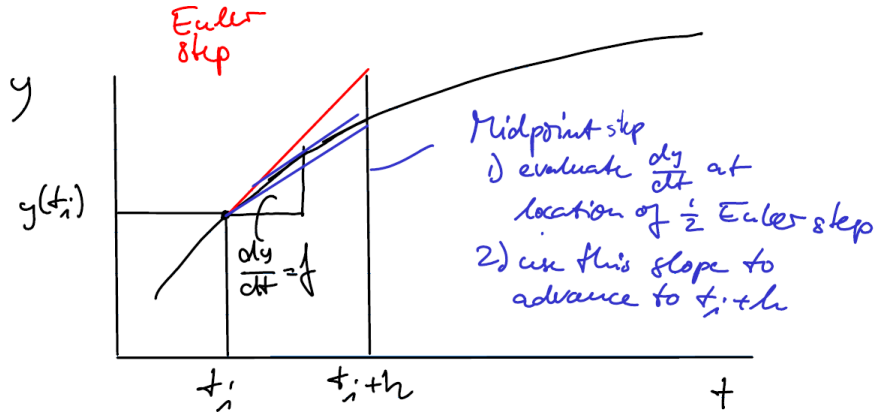


Figure 3.2: Sketch illustrating the midpoint method.

1. **The midpoint method** of Figure 3.2 evaluates the derivative of y w.r.t. to t first at half the Euler step

$$\frac{dy}{dt}\left(t_i + h/2, y\left(t_i\right) + \frac{dy}{dt}\left(t_i\right)\frac{h}{2}\right) \tag{3.17}$$

and then advances y by that slope

$$y\left(t_i + h\right) = y\left(t_i\right) + h\frac{dy}{dt}\left(t_i + h/2, y\left(t_i\right) + \frac{dy}{dt}\left(t_i\right)\frac{h}{2}\right) \tag{3.18}$$

written in terms of f

$$y\left(t_i + h\right) = y\left(t_i\right) + h\, f\left(t_i + \frac{h}{2}, y\left(t_i\right) + f\left(t_i, y_i\right)\frac{h}{2}\right) \tag{3.19}$$

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

$$k_1 = h f(t_i, y_i) \tag{3.20}$$

$$k_2 = h f\left(t_i + \frac{h}{2}, y_i + \frac{1}{2}k_1\right) \tag{3.21}$$

$$y_{i+1} = y_i + k_2 + O\left(h^3\right) \tag{3.22}$$

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 $\mathcal{O}\left(h^3\right)$, 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

$$\begin{aligned} k_1 &= h f(t_i, y_i) \\ k_2 &= h f\left(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\ k_3 &= h f\left(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) \\ k_4 &= h f(t_i + h, y_i + k_3) \\ y_{i+1} &= y_i + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O\left(h^5\right). \end{aligned} \tag{3.23}$$

The next improvement to the Runge Kutta method is to adapt the step size h during forward integration, according to an a comparison of an on-the-fly error estimate with some desired accuracy (*Press et al.*, 1993, sec. 16.2). If you have a simple problem and want to implement your own method, as in the problem set below, one way to test the behavior of the forward routine is to successively half h and keep track of the deviations of $|\boldsymbol{y}|$ at x_f to make sure things converge. This is of course assuming that the parameters for f remain within a reasonable range as used for the test, and adaptive step size is preferred.

Moreover, especially tricky functions f require special methods, and *Press et al.* (1993), chap. 16, discusses these. Some of the more fancy methods are implemented in Matlab, read the help material for how to use the built-in ODE solvers. Matlab implementation of ODE solvers are discussed in *Spencer and Ware* (2008), sec. 16.5. Typically, you want to try `ode45`, and if that fails, `ode113`, or `ode155`.

3.2.1 Two-point Boundary Value Problems

Here, $\boldsymbol{y}$ is given at both ends of the interval, x_0 and x_f. We will not deal with those kinds of problems here. They generally involve iteration to find the right solution based on

the initial value problems such as the "shooting method" which makes use of forward methods such as Runge Kutta (*e.g.* *Press et al.*, 1993, sec. 17.2). Alternatively, relaxation methods rely on iterative solution of a finite difference representation of the boundary value problem (*e.g.* *Press et al.*, 1993, sec. 17.3).

3.3 Exercise: Solving ODEs – Lorenz equations

Reading

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

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

$$\frac{d\boldsymbol{y}}{dt}(t) = \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{C}) \tag{3.24}$$

we made the dependence of f on constant parameters explicit in the $\boldsymbol{C}$.

Numerically, this is done by successively computing $\boldsymbol{y}_{n+1}$ for time $t + h$ from the last known solution for $\boldsymbol{y}_n$ at time t with time step h

$$\boldsymbol{y}_{n+1} \approx \boldsymbol{y}_n + h\boldsymbol{y}'(h, t, \boldsymbol{y}, \boldsymbol{C}) \tag{3.25}$$

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

In a real application, we would use adaptive step-size control by means of error checking depending on the accuracy of our approximate method, or employ an entirely different approach (*Press et al.*, 1993). *Spencer and Ware* (2008) discuss some of the algorithms that are implemented in Matlab, and the problem set file `rikitake.m`[1] is an example for how to use the Matlab function `ode45`. However, the Runge-Kutta is good example method and easy enough to implement.

3.3.1 The Lorenz equations solved with simple Runge Kutta

As an interesting example of a three-dimensional ($\boldsymbol{y} = \{y_1, y_2, y_3\}$) 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.

[1] All Matlab files for the problem sets are at `http://geodynamics.usc.edu/~becker/teaching-557.html`; solutions are available for instructors upon request.

3.3.2 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

$$\nabla \cdot \boldsymbol{v} = 0 \tag{3.26}$$

$$\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \nabla T = \kappa \nabla^2 T \tag{3.27}$$

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\, \boldsymbol{v} = \nu \nabla^2 \boldsymbol{v} - \frac{1}{\rho_0} \nabla P + \frac{\rho}{\rho_0} \boldsymbol{g}. \tag{3.28}$$

Here, $\nu = \eta/\rho_0$ is dynamic viscosity, $\boldsymbol{v}$ velocity, T temperature, κ thermal diffusivity, $\boldsymbol{g}$ 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 $\boldsymbol{g}$ 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

$$\frac{\partial T'}{\partial t'} + \boldsymbol{v}' \cdot \nabla T' = \nabla^2 T' \tag{3.29}$$

$$\frac{1}{Pr}\left(\frac{\partial \omega}{\partial t'} + \boldsymbol{v}' \cdot \nabla \omega\right) = \nabla^2 \omega - Ra \frac{\partial T'}{\partial x'}, \tag{3.30}$$

where the primed quantities are non-dimensionalized (more on this later). Eq. (3.30) is eq. (3.28) rewritten in terms of vorticity ω, such that $\nabla^2 \psi = -\omega$ where ψ is the stream-function, which relates to velocity as

$$\boldsymbol{v} = \nabla \times \psi \boldsymbol{k} = \{\partial\psi/\partial z, -\partial\psi/\partial x\} \tag{3.31}$$

and enforces incompressibility (mass conservation). The important part here are the two new non-dimensional quantities that arise, the Prandtl and the Rayleigh numbers, which were discussed previously (eqs. 2.40 and 2.41).

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

$$\psi \approx W(t) \sin(\pi a x) \sin(\pi z) \tag{3.32}$$

$$T \approx (1 - z) + T_1(t) \cos(\pi a x) \sin(\pi z) + T_2(t) \sin(2\pi z) \tag{3.33}$$

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.

3.3.3 Problems

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

$$\frac{dW}{dt} = Pr(T_1 - W) \tag{3.34}$$

$$\frac{dT_1}{dt} = -WT_2 + rW - T_1$$

$$\frac{dT_2}{dt} = WT_1 - bT_2$$

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

1. Identify W, T_1, and T_2 as y_1, y_2, y_3 and write up a Matlab code for a 4^{th} order Runge Kutta scheme to solve for the time-evolution of $\boldsymbol{y}$ using eq. (3.34) for derivatives.

   Hint: You can code this any way you want, but consider the following (Figure 3.4):

   - 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 functions.
   - You will want to make the Runge Kutta stepper independent of the actual function that is needed to compute $d\boldsymbol{y}/dt$ 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 then has to be referred to as
     `@func`. Alternatively, the function that computes the derivatives can be made into its own ".m" file, in the same directory as the other subroutines, making it available to all subroutines in that folder.
   - If you need some inspiration on how to do this, download the m-file fragments we provide for this sections problem set, `dydt.m`, `lorenz.m`, and `rkstep.m`.

2. Use initial condition $\boldsymbol{y}_0 = \{0, 0.5, 0.5\}$, 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 $\boldsymbol{y}$ 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 $\boldsymbol{y}_0$ 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.

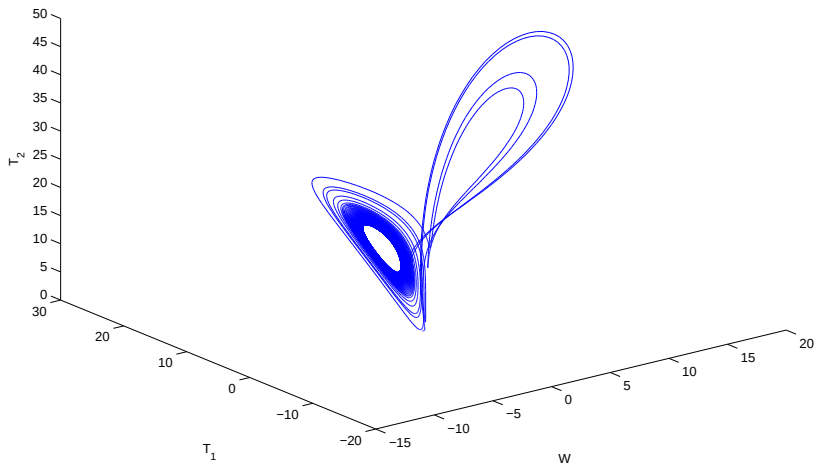


Figure 3.3: Solution to one of the problem set questions visualizing the behavior of the Lorenz equations (the Lorenz attractor).

For help with making simple plots with Matlab, see *Spencer and Ware* (2008), for example.

3.3.4 Additional examples

1. 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).

2. 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. The equations are

$$\begin{aligned} \frac{dx}{dt} &= -mx + yz \\ \frac{dy}{dt} &= -my + (z - a)x \\ \frac{dz}{dt} &= 1 - xy \end{aligned} \tag{3.35}$$

with typical parameters for a of 4 or 10, $m = 2$, at initial conditions $x, y, z = -5, 2, 2$. The file `rikitake.m` provides an example implementation of these equations using

```
function lorenz % this is only a function to allow function declarations
%
% Lorent'z equation solver
% the ... parts will have to be filled in by you
%
% values to solve for
%
% y(1) : W
% y(2) : T1
% y(3) : T2

% parameters for the equations
parameters.r = ...; % Rayleigh number
parameters.Pr = ..; % Prandtl number

% initial values
y= [...];
time =0;tstop=50;
h = 0.005;% timestep
save_each = 1;
nstep=0;save_step=0;
while(time < tstop) % loop while time is smaller than tstop
    if(mod(nstep,save_each)==0) % only save every save_each step
        save_step=save_step+1;
        ysave(save_step,:)=y;
...
    end
    % advance the y(1:3) solution by one 4th order Runge Kutta step
    y =  y + rkstep(....);

    nstep=nstep+1;
    time=time+h;
end

figure(1);clf % time series
plot(tsave,ysave(:,2))
xlabel('time');ylabel('temperature');
legend('T_1','T_2')



function dy = rkstep(.... )
%
%
% perform one 4th order Runge Kutta timestep and return
% the increment on y(t_n) by evaluating func(time,y,parameters)
%
% ... parts need to be filled in
%
%
% input values:
%  h: time step
%  t: time
%  y: vector with variables at time = t which are to be advanced
%  func: function which computes dy/dt
%  parameters: structure with any parameters the func function might need

% save computations
h2=h/2;

k1 = h .* dydt(...);
k2 = h .* dydt(...);
....
% return the y_{n+1} timestep
dy =  ....



function dydt = lorenz_dy(...)
%
% functions for Runge Kutta
%
%
% dW/dt = Pr (T1-w)
%
dydt(1) = ...
%
% dT1/dt = -W T2 + rW - T1
%
dydt(2) = ...
%
% dT2/dt = W T1 - b T2
%
dydt(3) = ... x
```

Figure 3.4: Suggested program structure for the Lorenz equation ODE solver exercise. Available online as `lorenz.m`, `rkstep.m`, and `dydt.m`.

a Matlab ODE solver.

3. 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 *Christensen* (1985). 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. I.e. we are averaging the PDEs governing convection spatially, to solve for the time-evolution of average mantle temperature.

   The idea is that a convective system with Rayleigh number

   $$Ra = \frac{\rho \alpha T g h^3}{\kappa \eta} \tag{3.36}$$

   transports heat at Nusselt number Nu following a scaling of

   $$Nu = \frac{Q}{cT} = aRa^{\beta} \tag{3.37}$$

   (with some debate about β, see, *e.g.* *Korenaga*, 2008, for a review). The energy balance for the mantle is

   $$C_p \frac{dT}{dt} = H(t) - Q \tag{3.38}$$

   where C_p is the total heat capacity, H the time-dependent heat production through radiogenic elements, and Q the heat loss through the surface. If viscosity is a function of temperature,

   $$\eta = \eta_0 \exp\left(\frac{H}{RT}\right) \tag{3.39}$$

   then the equations couple such that

   $$C_p \frac{dT}{dt} = H(t) - Q_0 \left(\frac{T}{T_0}\right)^{1+\beta} \left(\frac{\eta(T_0)}{\eta(T)}\right)^{\beta}. \tag{3.40}$$

   We provide an example, `thermal_all.m` online. You might want to experiment with the shooting method to explore feasible and unfeasable paths of Earth's thermal evolution from an initial to a final temperature.

4. 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).

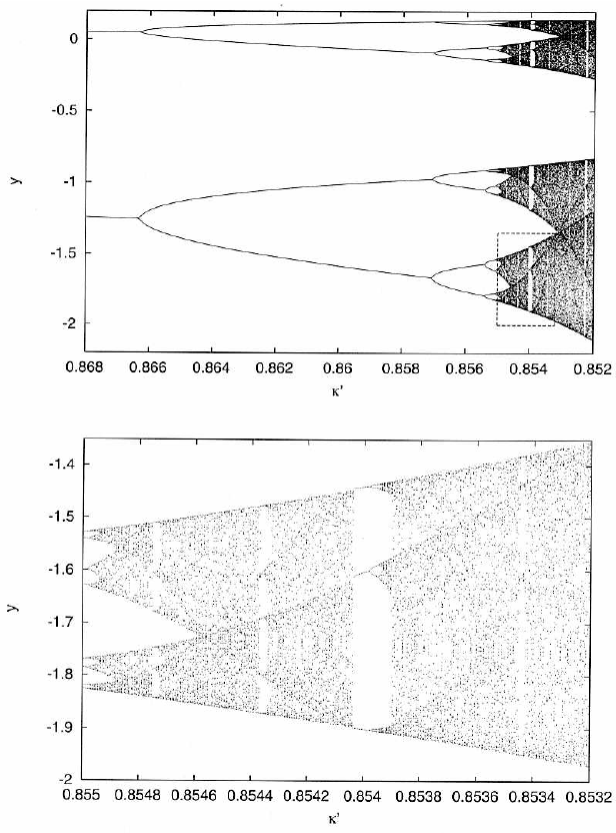


Figure 3.5: Poincare sections in y for the period doubling sequence to chaos for the spring-slider system, eq. (3.41), as a function of normalized spring stiffness, κ'. Bottom figure shows zoom into the dashed rectangular region highlighted on top (modified from *Becker*, 2000).

The equations are

$$\begin{aligned} \dot{x} = \frac{dx}{dt} &= e^x((\beta_1 - 1)x + y - z) + \dot{y} - \dot{z} \\ \dot{y} = \frac{dy}{dt} &= (1 - e^x)\kappa \\ \dot{z} = \frac{dz}{dt} &= -e^x\rho(\beta_2 x + z) \end{aligned} \tag{3.41}$$

with $\beta_1 = 1$, $\beta_2 = 0.84$, and $\rho = 0.048$ (*Gu et al.*, 1984).

This behavior includes the characteristic period-doubling route toward chaos (*Feigenbaum*, 1978) as a function of a material parameter (spring stiffness), at critical $\kappa = 0.08028$ (*Becker*, 2000). You might want to reproduce the bifurcation plot of Figure 3.5.

Part III

Partial differential equations

Chapter 4

Finite differences

4.1 Introduction to the finite difference method

We now turn to the solution of partial differential equations (PDEs), and the first method that will be discussed is finite differences (FD). The solution of PDEs by means of FD is based on approximating derivatives of continuous functions, *i.e.* the actual partial differential equation, by discretized versions of the derivatives based on discrete points of the functions of interest.

4.1.1 Finite differences and Taylor series expansions

Finite difference approximations to PDEs 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 (eq. 6.4) expansion for $f(x_0 + \Delta x)$, away from the point x_0 by a small amount h (sometimes here also denoted by Δx), gives

$$\begin{aligned} f(x_0+h) &= f(x_0) + \frac{\partial f(x_0)}{\partial x} h + \\ &\quad \frac{\partial^2 f(x_0)}{\partial x^2}\frac{h^2}{2!} + \frac{\partial^3 f(x_0)}{\partial x^3}\frac{h^3}{3!} + \ldots + \frac{\partial^n f(x_0)}{\partial x^n}\frac{h^n}{n!} + \\ &\quad \mathcal{O}(h^{n+1}). \end{aligned} \tag{4.1}$$

We can express the first derivative of f by rearranging eq. (4.1)

$$\frac{\partial f(x_0)}{\partial x} = \frac{f(x_0+h) - f(x_0)}{h} - \frac{\partial^2 f(x_0)}{\partial x^2}\frac{h}{2!} - \frac{\partial^3 f(x_0)}{\partial x^3}\frac{h^2}{3!}\ldots \tag{4.2}$$

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

$$\frac{\partial f(x_i)}{\partial x} = \frac{f_{i+1} - f_i}{h} + \mathcal{O}h \tag{4.3}$$

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 (or Δx) is assumed constant. Here, $\mathcal{O}(h)$ indicates that the full solution would require terms of order h, h^2, and so on. $\mathcal{O}$ 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. (4.3) 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

$$f(x_0 - h) = f(x_0) - \frac{\partial f(x_0)}{\partial x} h + \frac{\partial^2 f(x_0)}{\partial x^2} \frac{h^2}{2!} - \frac{\partial^3 f(x_0)}{\partial x^3} \frac{h^3}{3!} + ... \tag{4.4}$$

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

$$\frac{\partial f(x_i)}{\partial x} = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h). \tag{4.5}$$

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

$$f' = \frac{\partial f}{\partial x}, \quad f'' = \frac{\partial^2 f}{\partial x^2} \quad \dots \tag{4.6}$$

we can find, for example,

$$\begin{aligned} f''_{i+1} &= \frac{f'_{i+1} - f'_i}{h} + \mathcal{O}(h) && (4.7)\\ &= \frac{\frac{f_{i+2} - f_{i+1}}{h} - \frac{f_{i+1} - f_i}{h}}{h} + \mathcal{O}(h) && (4.8)\\ &= \frac{f_{i+2} - 2f_{i+1} + f_i}{h^2} + \mathcal{O}(h) && (4.9) \end{aligned}$$

which is the first order accurate, forward difference approximation for second order derivatives around f_{i+1}.

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,

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(x) + \dots && (4.10)\\ &= \frac{f(x+h) - f(x)}{h} - \\ &\quad \frac{h}{2} \left(\frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + \mathcal{O}(h) \right) + \mathcal{O}(h^2) \quad \text{or} \\ f'_{i+1} &= \frac{-f_{i+2} + 4f_{i+1} - 3f_i}{2h} + \mathcal{O}(h^2). && (4.11) \end{aligned}$$

Alternatively, we can form the average of the first order accurate forward and backward schemes, *i.e.* adding eqs. (4.3) and (4.5) and dividing by two. The result is the *central difference* approximation of the first derivative

$$f'_i = \frac{f_{i+1} - f_{i-1}}{2h} + \mathcal{O}(h^2) \tag{4.12}$$

and also second order accurate.

Note that eq. (4.12) involves fewer function evaluations than eq. (4.11), which is why eq. (4.12) is preferred for actual implementations. Also, 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. Moreover, improving accuracy by taking higher and higher order polynomial expansions into account only works if the function f is actually smooth (*i.e.* differentiable) in that way, and not "weird" or jumpy. In this case, lower order approximations may do just as well.

By adding eqs. (4.1) and (4.4) an approximation of the second derivative is obtained

$$f''_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + \mathcal{O}(h^2). \tag{4.13}$$

A different way to derive the second derivative of second order accuracy is by computing, by central differences which we now know should yield second accuracy, the first derivatives (theoretically!) as evaluated at the points in between, $i + 1/2$ and $i - 1/2$, and then computing the second derivative at i from this by using the central difference of those two first derivatives:

$$\begin{aligned} f'_{i+1/2} &= \frac{f_{i+1} - f_i}{h} \\ f'_{i-1/2} &= \frac{f_i - f_{i-1}}{h} \\ f''_i &= \frac{f'_{i+1/2} - f'_{i-1/2}}{h} = \frac{\frac{f_{i+1}-f_i}{h} - \frac{f_i - f_{i-1}}{\Delta x}}{h} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} \end{aligned} \tag{4.14}$$

(This is important for derivatives with variable coefficients, *cf.* sec. 4.1.3.) If h is not constant, it should be computed as $x_{i+1} - x_i$. The second order derivative in this case is identical to the equations above with $x_{i+1} - x_i = x_i - x_{i-1} = h$.

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 $d^n f/dx^n$ can be found in *Fornberg* (1996). Note that the highest order derivative that usually occurs in geodynamics is the 4^{th}-order derivative.

4.1.2 Finite difference approximations overview

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. One of the few cases where you want to use first order derivatives is for advection, where second order central is a very poor choice because it introduces large artificial diffusion.

Left-sided first derivative, first order

$$\frac{\partial u}{\partial x}\Big|_{i-1/2} = \frac{u_i - u_{i-1}}{h} + \mathcal{O}(h) \tag{4.15}$$

Right-sided first derivative, first order

$$\frac{\partial u}{\partial x}\Big|_{i+1/2} = \frac{u_{i+1} - u_i}{h} + \mathcal{O}(h) \tag{4.16}$$

Central first derivative, second order

$$\frac{\partial u}{\partial x}\Big|_{i} = \frac{u_{i+1} - u_{i-1}}{2h} + \mathcal{O}(h^2) \tag{4.17}$$

Central first derivative, fourth order

$$\frac{\partial u}{\partial x}\Big|_{i} = \frac{u_{i+2} - 8u_{i+1} + 8u_{i-1} - u_{i-2}}{12h} + \mathcal{O}(h^4) \tag{4.18}$$

Central second derivative, second order

$$\frac{\partial^2 u}{\partial x^2}\Big|_{i} = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \mathcal{O}(h^2) \tag{4.19}$$

Central second derivative, fourth order

$$\frac{\partial^2 u}{\partial x^2}\Big|_{i} = \frac{-u_{i+2} + 16u_{i+1} - 30u_i + 16u_{i-1} - u_{i-2}}{12h^2} + \mathcal{O}(h^4) \tag{4.20}$$

Central third derivative, second order

$$\frac{\partial^3 u}{\partial x^3}\Big|_{i} = \frac{-u_{i+2} + 2u_{i+1} - 2u_{i-1} + u_{i-2}}{2h^3} + \mathcal{O}(h^2) \tag{4.21}$$

Central third derivative, fourth order

$$\frac{\partial^3 u}{\partial x^3}\Big|_{i} = \frac{u_{i+3} - 8u_{i+2} + 13u_{i+1} - 13u_{i-1} + 8u_{i-2} - u_{i-3}}{8h^3} + \mathcal{O}(h^4) \tag{4.22}$$

Central fourth derivative, second order

$$\frac{\partial^4 u}{\partial x^4}\Big|_{i} = \frac{u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}}{h^4} + \mathcal{O}(h^2) \tag{4.23}$$

4.1.3 Derivatives with variable coefficients

Note that derivatives with of the following form

$$\frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right), \tag{4.24}$$

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

$$\frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right)\Big|_i = \frac{k_{i+1/2}\frac{u_{i+1}-u_i}{h} - k_{i-1/2}\frac{u_i-u_{i-1}}{h}}{h} + \mathcal{O}(h^2), \tag{4.25}$$

where $k_{i-1/2}$ is evaluated between u_i and u_{i-1}, to maintain the second order accuracy of the central difference approach for second derivatives (eq. 4.14)

$$f'' = \frac{f'(x+\frac{\Delta x}{2}) - f'(x-\frac{h}{2})}{h}, \tag{4.26}$$

i.e. the first derivatives are premultiplied with the cofficients in between the nodes. If k is spatially varying, the following, common approximations are therefore *inadequate* to maintain second order accuracy:

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

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

Note: If k has strong jumps from one grid point to another that are not aligned with the grid-nodes, most second-order methods will show first order accuracy at best.

4.2 Finite difference example: 1D explicit heat equation

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

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) \tag{4.27}$$

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

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \tag{4.28}$$

where

$$\kappa = \frac{k}{\rho c_p} \tag{4.29}$$

is the thermal diffusivity (a common value for rocks is $\kappa = 10^{-6}\,\mathrm{m^2 s^{-1}}$; also see discussion in sec. 2.4).

We are interested in the temperature evolution versus time, $T(x,t)$, which satisfies eq. (4.28), given an initial temperature distribution (Fig. 4.1A). 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. 4.1B). The next step is to replace the continuous derivatives of eq. (4.28) 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

$$\frac{\partial T}{\partial t} \approx \frac{T_i^{n+1} - T_i^n}{t^{n+1} - t^n} = \frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{T_i^{new} - T_i^{current}}{\Delta t}. \tag{4.30}$$

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. 4.1B). 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. (4.28) is replaced by a central finite difference approximation (*cf.* sec. 4.1.2), *i.e.*

$$\frac{\partial^2 T}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial T}{\partial x}\right) \approx \frac{\frac{T_{i+1}^n - T_i^n}{\Delta x} - \frac{T_i^n - T_{i-1}^n}{\Delta x}}{\Delta x} = \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2}. \tag{4.31}$$

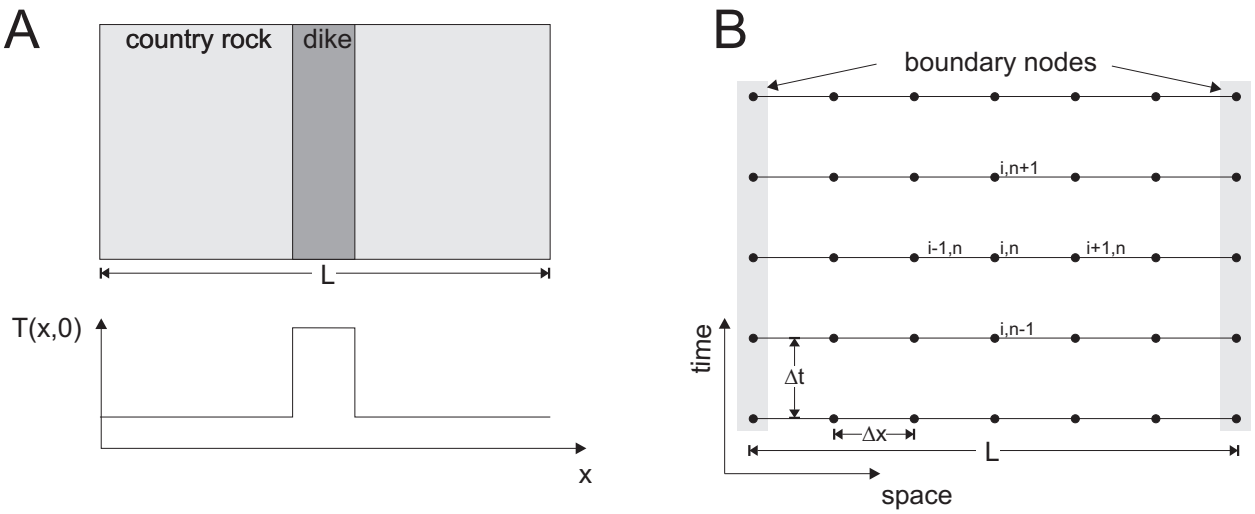


Figure 4.1: A) Setup of the thermal cooling model considered here. A hot basaltic dike intrudes cooler country rocks. Only variations in x-direction are considered; properties in the other directions are assumed to be constant. The initial temperature distribution $T(x,0)$ has a step-like perturbation, 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 Δx. The time-evolution is also computed at given times with time step Δt.

Substituting eqs. (4.31) and (4.30) into eq. (4.28) gives

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \left(\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \right). \tag{4.32}$$

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:

$$T_i^{n+1} = T_i^n + \kappa \Delta t \left(\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \right) \tag{4.33}$$

Because the temperature at the current time step (n) is known, we can use eq. (4.33) to compute the new temperature without solving any additional equations. Such a scheme is and *explicit* finite difference method 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 Δx and Δt because it has been shown to be consistent – that its discretization 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 Δt and Δx: 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:

$$T(x < -W/2, x > W/2, t = 0) = 300 \quad (4.34)$$
$$T(-W/2 \leq x \leq W/2, t = 0) = 1200 \quad (4.35)$$

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

$$T(x = -L/2, t) = 300 \quad (4.36)$$
$$T(x = L/2, t) = 300 \quad (4.37)$$

The MATLAB code in Figure 4.2, `heat1Dexplicit.m`, 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. (4.33).

4.2.1 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 time step). Compare the results for small Δx and Δt with those for larger Δx and Δt. How are these solutions different? Why? Notice also that if the time step 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 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).

```
%heat1Dexplicit.m
%
% 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 4.2: MATLAB script `heat1Dexplicit.m` to solve eq. (4.28) (once the blanks indicated by the questions marks are filled in . . .).

4. Record and plot the temperature evolution versus time at a distance of 5 m from the dikecountry 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. Add a test with an analytical solution for diffusion and plot error *vrs.* resolution. A good reference for analytical solutions for heat conduction problems is *Carslaw and Jaeger* (1959), or see sec. 4.8.

   The spatial discretization should be second order for a second order scheme.

7. Derive a finite-difference approximation for variable k (and variable Δx allowing for

uneven spacing between grid points should you so desire). Test the solution for the case of $k = 10$ inside the dike, and $k = 3$ in the country rock.

4.3 Implicit FD schemes and boundary conditions

Reading

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

Limited stability and numerical aliasing/dissipation are two major drawbacks of explicit finite difference codes such as the one presented in sec. 4.2. Next, we will discuss methods that do not have these limitations.

4.3.1 Time derivatives – explicit *vrs.* implicit

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

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \tag{4.38}$$

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. (4.38) is

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2}, \tag{4.39}$$

using central differences for the spatial derivatives (subscript i indicating the x location in 1-D, superscripts indicating the time). Eq. (4.39) 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)

$$T_i^{n+1} = T_i^n + \kappa \Delta t \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \tag{4.40}$$

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 4.3a, and an algorithm based on eq. (4.40), such as the one of last section's problem set, is called a *forward time, centered space (FTCS)* because of 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

$$0 < \frac{2\kappa \Delta t}{(\Delta x)^2} \leq 1 \quad \text{or} \quad \Delta t \leq \frac{(\Delta x)^2}{2\kappa} \quad \text{for given } \Delta x. \tag{4.41}$$

If this condition is not satisfied, the solution becomes unstable, starts to wildly oscillate, or "blow up".

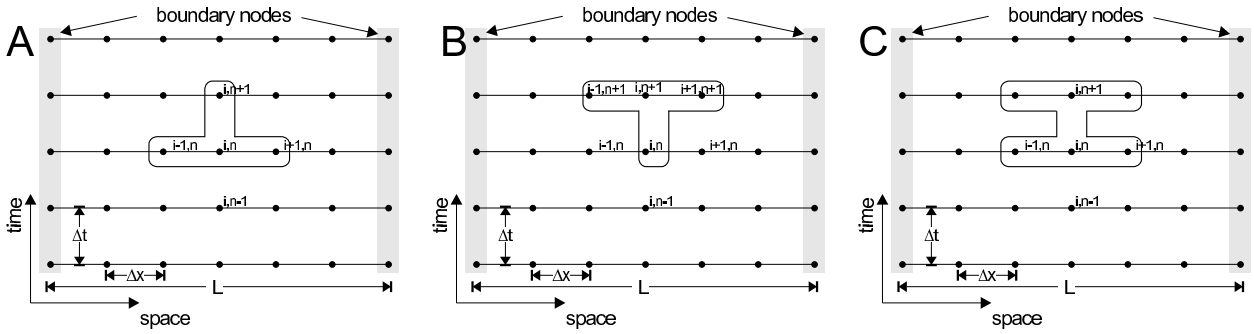


Figure 4.3: A) Explicit finite difference discretization. B) Implicit finite difference discretization. C) Crank-Nicolson finite difference discretization.

This can be shown by von Neumann stability analysis (*Press et al.*, 1993, chap. 19.2), analyzing the growth of of *eigenmodes* of the finite difference equation. However, physically, the stability condition eq. (4.41) 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 Δx (*cf.* diffusion time in sec. 3.3). The explicit solution, eq. (4.40), is an example of a *conditionally stable* method that only leads to well behaved solutions if a criterion like eq. (4.41) is satisfied.

Note that eq. (4.41) 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 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 (*cf. Ismail-Zadeh and Tackley*, 2010).

We will revisit an FTCS scheme similar to eq. (4.39) for advection that involves single derivatives in space later. Unlike eq. (4.39), 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 Δx, $\lambda \gg \Delta x$, because we want to resolve the solution features at least with a few nodes, the explicit scheme, eq. (4.39), will require

$$\left(\frac{\lambda}{\Delta x} \right)^2 \gg 1 \tag{4.42}$$

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. (4.38) is

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2}, \tag{4.43}$$

a *fully implicit* scheme where the time derivative is taken backward (Figure 4.3b). Eq. (4.43) can be rearranged so that unknown terms are on the left and known terms are on the right

$$-sT_{i+1}^{n+1} + (1+2s)T_i^{n+1} - sT_{i-1}^{n+1} = T_i^n \tag{4.44}$$

where

$$s = \frac{\kappa \Delta t}{(\Delta x)^2}. \tag{4.45}$$

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

$$\frac{\partial}{\partial x}\left(k(x)\frac{\partial}{\partial x}T\right) \tag{4.46}$$

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

$$-rk_rT_{i+1}^{n+1} + [1 + r(k_l + k_r)]T_i^{n+1} - rk_lT_{i-1}^{n+1} = T_i^n \tag{4.47}$$

has to be used instead of eq. (4.44). Here, $r = \Delta t/(\Delta x)^2$ 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. (4.44) is also suited to understand the overall behavior of an implicit method for large time steps. If we let $\Delta t \to \infty$, and then divide eq. (4.44) by $-s$, we get

$$T_{i+1} - 2T_i + T_{i-1} = 0, \tag{4.48}$$

which is a central difference approximation of the steady state solution of eq. (4.38),

$$\frac{\partial^2 T}{\partial x^2} = 0. \tag{4.49}$$

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. (4.43) is second order accurate in space but only first order accurate in time, *i.e.* $\mathcal{O}((\Delta x)^2, \Delta t)$. It is possible to write down a scheme which is second order accurate both in time and in space (*i.e.*

$\mathcal{O}((\Delta x)^2, (\Delta t)^2)$). One such scheme is the Crank-Nicolson scheme (see exercises, Fig. 4.3C), 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 $\mathcal{O}((\Delta x)^2)$. 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 solution (inversion) of a system of linear equations based on the known solution at time n. This is discussed next.

4.4 Finite difference example: 1D implicit heat equation

4.4.1 Boundary conditions – Neumann and Dirichlet

We solve the transient heat equation

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) \tag{4.50}$$

on the domain $-L/2 \leq x \leq L/2$ subject to the following boundary conditions for fixed temperature

$$\begin{aligned} T(x = -L/2, t) &= T_{left} \\ T(x = L/2, t) &= T_{right} \end{aligned} \tag{4.51}$$

with the initial condition

$$T(x < -W/2, x > W/2, t = 0) = 300 \tag{4.52}$$

$$T(-W/2 \leq x \leq W/2, t = 0) = 1200, \tag{4.53}$$

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

Boundary conditions (BCs, see also sec. 2.2.3) for PDEs that specify values of the solution function (here T) to be constant, such as eq. (4.51), 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):

$$\begin{aligned} \frac{\partial T}{\partial x}(x = -L/2, t) &= 0 \\ \frac{\partial T}{\partial x}(x = L/2, t) &= 0. \end{aligned} \tag{4.54}$$

4.4.2 Solving an implicit finite difference scheme

As before, 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. (4.44). Starting with fixed temperature BCs (eq. 4.51), the boundary condition on the left boundary gives

$$T_1 = T_{left} \tag{4.55}$$

and the one on the right

$$T_{n_x} = T_{right}. \tag{4.56}$$

Eqs. (4.44), (4.55), and (4.56) can be written in matrix form as

$$A\boldsymbol{x} = \boldsymbol{b}. \tag{4.57}$$

For a six-node grid, for example, the coefficient matrix A is

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -s & (1+2s) & -s & 0 & 0 & 0 \\ 0 & -s & (1+2s) & -s & 0 & 0 \\ 0 & 0 & -s & (1+2s) & -s & 0 \\ 0 & 0 & 0 & -s & (1+2s) & -s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{4.58}$$

the unknown temperature vector $\boldsymbol{x}$ is

$$\boldsymbol{x} = \begin{pmatrix} T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_5^{n+1} \\ T_6^{n+1} \end{pmatrix}, \tag{4.59}$$

and the known right-hand-side vector $\boldsymbol{b}$ is

$$\boldsymbol{b} = \begin{pmatrix} T_{left} \\ T_2^n \\ T_3^n \\ T_4^n \\ T_5^n \\ T_{right} \end{pmatrix}. \tag{4.60}$$

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. (4.72) and (4.73) 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 efficiently by LU decomposition using the Thomas algorithm (*e.g.* *Press et al.*, 1993, sec. 2.4).

4.4.3 Matlab implementation

Within Matlab, we declare matrix A to be sparse by initializing it with the `sparse` function. This will ensure a computationally efficient internal treatment within Matlab. Once the coefficient matrix A and the right-hand-side vector $\boldsymbol{b}$ have been constructed, MATLAB functions can be used to obtain the solution $\boldsymbol{x}$ 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 $\boldsymbol{b}$ can be constructed with

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

The only thing that remains to be done is to solve the system of equations and find $\boldsymbol{x}$. MATLAB does this with

```
x = A\b;
```

The vector $\boldsymbol{x}$ 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 Δx, 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 $\boldsymbol{b}$ and $\boldsymbol{x}$ need to be recomputed. (Note: Having a constant matrix helps a lot for large systems because operations such as `x = A\b` can then be optimized further by storing A in a special form.)

4.4.4 Exercises

1. Save the script `heat1Dexplicit.m` from last section as `heat1Dimplicit.m`. Program the implicit finite difference scheme explained above. Compare the results with results from last section's explicit code.

2. Time-dependent, analytical solutions for the heat equation exists. For example, if the initial temperature distribution (initial condition, IC) is

$$T(x, t = 0) = T_{max} \exp\left(-\left(\frac{x}{\sigma}\right)^2\right) \tag{4.61}$$

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

$$T(x, t) = \frac{T_{max}}{\sqrt{1 + 4t\kappa/\sigma^2}} \exp\left(\frac{-x^2}{\sigma^2 + 4t\kappa}\right) \tag{4.62}$$

   (for $T = 0$ BCs at infinity). (See *Carslaw and Jaeger*, 1959, for useful analytical solutions to heat conduction problems).

   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 section to the analytical solution near the instability region of FTCS,

$$s = \frac{\kappa \Delta t}{(\Delta x)^2} < \frac{1}{2}. \tag{4.63}$$

   Note: Eq. (4.62) 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. 4.38) 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. (4.72) and (4.73) need to replace the first and last columns of your A matrix.

5. Derive and program the Crank-Nicolson method (*cf.* Figure 4.3C). This "best of both worlds" method is obtained by computing the average of the fully implicit and fully explicit schemes:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{\kappa}{2}\left(\frac{\left(T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}\right) + \left(T_{i+1}^n - 2T_i^n + T_{i-1}^n\right)}{(\Delta x)^2}\right). \tag{4.64}$$

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. (4.64) and sorting terms into those that depend on the solution at time step $n + 1$ and those at time step n, as for eq. (4.44).

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 Δx is constant. For extra bonus, allow for variable grid spacing and variable conductivity.

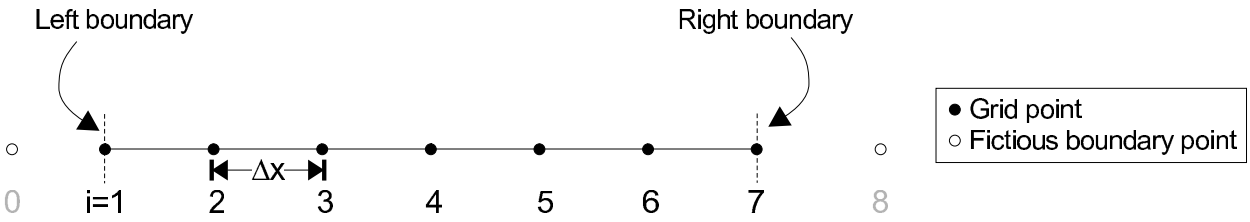


Figure 4.4: Discretization of the numerical domain with fictitious boundary points, that are employed to set flux boundary conditions.

4.5 Derivation of flux boundary conditions (fictitious boundary points)

A Neumann boundary condition can be expressed as

$$\begin{aligned} \frac{\partial T(x=0,t)}{\partial x} &= c_1 \\ \frac{\partial T(x=L,t)}{\partial x} &= c_2 \end{aligned} \tag{4.65}$$

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

$$\frac{T_2 - T_0}{2\Delta x} = c_1 \tag{4.66}$$

and at $i = n_x$ by

$$\frac{T_{n_x+1} - T_{n_x-1}}{2\Delta x} = c_2. \tag{4.67}$$

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 4.4). A way around this can be found by noting that the equation for the center nodes is given by

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2}. \tag{4.68}$$

Writing this expression for the first node gives

$$\frac{T_1^{n+1} - T_1^n}{\Delta t} = \kappa \frac{T_2^{n+1} - 2T_1^{n+1} + T_0^{n+1}}{(\Delta x)^2}. \tag{4.69}$$

An explicit expression for T_0^{n+1} is obtained from eq. (4.66)

$$T_0^{n+1} = T_2^{n+1} - 2\Delta x c_1, \tag{4.70}$$

i.e. we are simply extrapolating from T_2 to T_0 with the slope given by c_1. Substituting into eq. (4.69) yields

$$\frac{T_1^{n+1} - T_1^n}{\Delta t} = \kappa \frac{2T_2^{n+1} - 2T_1^{n+1} - 2\Delta x c_1}{(\Delta x)^2}. \tag{4.71}$$

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

$$(1 + 2s)T_1^{n+1} - 2sT_2^{n+1} = T_1^n - 2s\Delta x c_1. \tag{4.72}$$

On the other end of the domain (verify!)

$$-2sT_{n_x-1}^{n+1} + (1 + 2s)T_{n_x}^{n+1} = T_{n_x}^n + 2s\Delta x c_2. \tag{4.73}$$

These equations now only involves grid points that are part of the computational grid, and eqs. (4.72) and (4.73) can be incorporated into the matrix A and right-hand-side $\boldsymbol{b}$ (compare with eq. 4.44).

4.6 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).

Whether 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; the most important of which is linearization of the nonlinear terms and solution of the (more complicated) PDE (*e.g. Newton-Rhapson* iterations). This method is converges quadratically, as long as the initial guess is close to the final solution. It is, however, more difficult to implement and will therefore not be discussed here.

4.6.1 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

$$\frac{\partial P}{\partial t} = \frac{\partial}{\partial z}\left(\kappa(P)\frac{\partial P}{\partial z}\right) \tag{4.74}$$

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. (4.74), we need a constitutive law, and we assume that the hydraulic diffusivity is given by

$$\kappa(P) = \kappa_0 + cP^m \tag{4.75}$$

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:

$$\frac{P_i^{n+1} - P_i^n}{\Delta t} = \frac{\kappa_{i+1/2}^{n+1}\frac{P_{i+1}^{n+1} - P_i^{n+1}}{\Delta x} - \kappa_{i-1/2}^{n+1}\frac{P_i^{n+1} - P_{i-1}^{n+1}}{\Delta x}}{\Delta x} \tag{4.76}$$

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

$$\kappa_{i\pm1/2}^{n+1} = \frac{\kappa_i^{n+1} + \kappa_{i\pm1}^{n+1}}{2}. \tag{4.77}$$

(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

$$A\boldsymbol{P} = \boldsymbol{b} \tag{4.78}$$

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 $\kappa^{n+1}_{i\pm1/2}$ using eqs. (4.75) and (4.77).
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:

$$error = \frac{max(abs(\boldsymbol{P}^{it} - \boldsymbol{P}^{it-1}))}{max(abs(\boldsymbol{P}^{it}))} \tag{4.79}$$

   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.

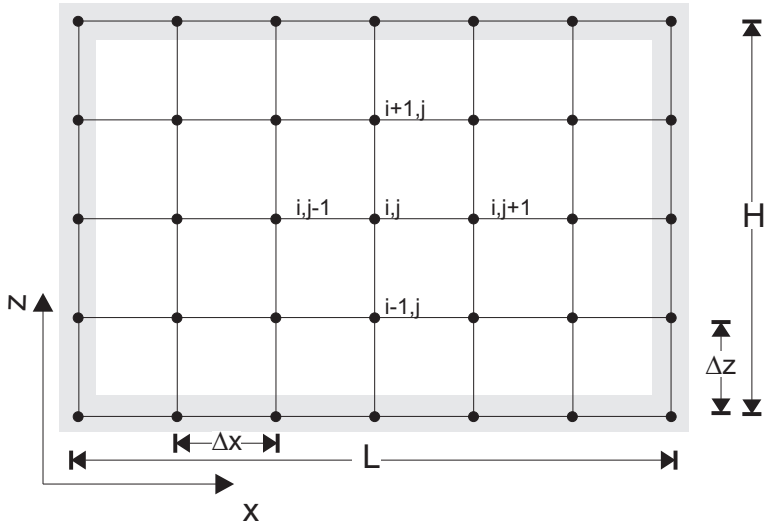


Figure 4.5: Finite difference discretization of the 2D heat problem.

4.7 Two-dimensional heat equation with FD

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

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k_x \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial z}\left(k_z \frac{\partial T}{\partial z}\right) + Q \tag{4.80}$$

where, ρ is density, c_p heat capacity, $k_{x,z}$ the thermal conductivities in x and z direction, and Q radiogenic heat production.

If the thermal conductivity is isostropic ($k_x = k_z$) and constant, we can rewrite

$$\frac{\partial T}{\partial t} = \kappa\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2}\right) + \frac{Q}{\rho c_p}. \tag{4.81}$$

4.7.1 Explicit method

The simplest way to discretize eq. (4.81) on a domain, *e.g.* a box with width L and height H, is to employ an FTCS, explicit method like in 1-D

$$\frac{T^{n+1}_{i,j} - T^n_{i,j}}{\Delta t} = \kappa\left(\frac{T^n_{i,j+1} - 2T^n_{i,j} + T^n_{i,j-1}}{(\Delta x)^2} + \frac{T^n_{i+1,j} - 2T^n_{i,j} + T^n_{i-1,j}}{(\Delta z)^2}\right) + \frac{Q^n_{i,j}}{\rho c_p}, \tag{4.82}$$

where Δx and Δz indicates the node spacing in both spatial directions, and there are now two indices for space, i and j for z_i and x_j, respectively (Figure 4.5). Rearranging gives

$$T_{i,j}^{n+1} = T_{i,j}^n + s_x \left(T_{i,j+1}^n - 2T_{i,j}^n + T_{i,j-1}^n \right) + s_z \left(T_{i+1,j}^n - 2T_{i,j}^n + T_{i-1,j}^n \right) + \frac{Q_{i,j}^n \Delta t}{\rho c_p}, \tag{4.83}$$

where

$$s_x = \frac{\kappa \Delta t}{(\Delta x)^2} \quad \text{and} \quad s_z = \frac{\kappa \Delta t}{(\Delta z)^2}. \tag{4.84}$$

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

$$T_{i,j=1} = T_{left} \quad \text{for all } i. \tag{4.85}$$

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

$$\begin{aligned} \frac{\partial T}{\partial x} &= c_1 \\ \frac{T_{i,2} - T_{i,0}}{2\Delta x} &= c_1 \\ T_{i,0} &= T_{i,2} - 2\Delta x c_1 \end{aligned} \tag{4.86}$$

which can then be plugged into eq. (4.83) so that for $j = 1$, for example,

$$\begin{aligned} T_{i,1}^{n+1} = T_{i,1}^n \quad &+ \quad s_x \left(2T_{i,2}^n - 2(T_{i,1}^n + \Delta x c_1) \right) \\ &+ \quad s_z \left(T_{i+1,1}^n - 2T_{i,1}^n + T_{i-1,1}^n \right) + \frac{Q_{i,1}^n \Delta t}{\rho c_p}. \end{aligned} \tag{4.87}$$

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

$$\texttt{T} = \texttt{zeros}(\texttt{nz}, \texttt{nx}); \tag{4.88}$$

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

$$\frac{2\kappa \Delta t}{\min((\Delta x)^2, (\Delta z)^2)} \leq 1. \tag{4.89}$$

4.7.2 Fully implicit method

If we employ a fully implicit, unconditionally stable discretization scheme as for the 1D exercise, eq. (4.81) can be written as

$$\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = \kappa \left(\frac{T_{i,j+1}^{n+1} - 2T_{i,j}^{n+1} + T_{i,j-1}^{n+1}}{(\Delta x)^2} + \frac{T_{i+1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i-1,j}^{n+1}}{(\Delta z)^2} \right) + \frac{Q_{i,j}^n}{\rho c_p}. \tag{4.90}$$

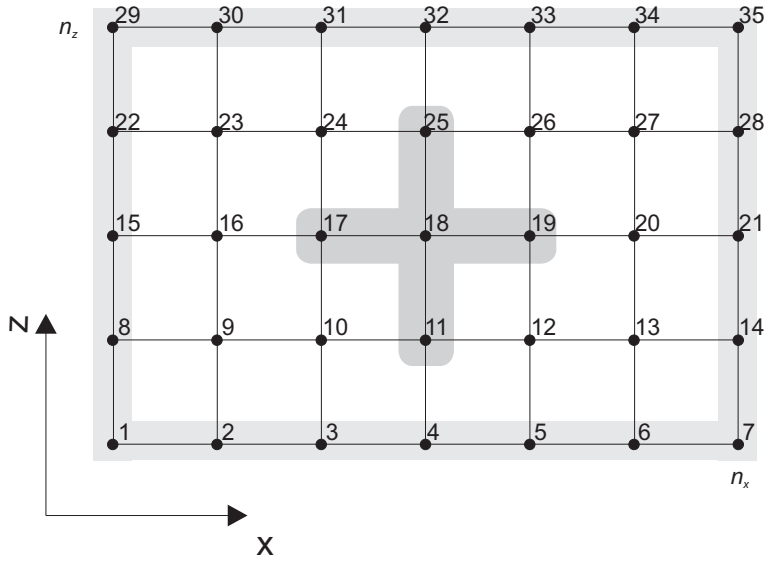


Figure 4.6: Numbering scheme for a 2D grid with $n_x = 7$ and $n_z = 5$.

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

$$-s_z T^{n+1}_{i+1,j} - s_x T^{n+1}_{i,j+1} + (1 + 2s_z + 2s_x) T^{n+1}_{i,j} - s_z T^{n+1}_{i-1,j} - s_x T^{n+1}_{i,j-1} = T^n_{i,j} + \frac{Q^n_{i,j} \Delta t}{\rho c_p}. \quad (4.91)$$

As in the 1D case, we have to write these equations in a matrix A and a vector $\boldsymbol{b}$ (and use Matlab `x = A\b` to solve for T^{n+1}). From a practical point of view, this is a bit more complicated than 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 `T(k)` (as opposed to the more intuitive matrix `T(i,j)` we could use for the explicit scheme).

If a 2D temperature field is to be solved for with an equivalent vector `T`, the nodes have to be numbered continuously, for example as in Figure 4.6. The derivative versus x-direction is then *e.g.*

$$\frac{\partial^2 T}{\partial x^2}|_{i=3,j=4} = \frac{1}{(\Delta x)^2} \left(T_{19} - 2T_{18} + T_{17} \right), \quad (4.92)$$

and the derivative versus z-direction is given by

$$\frac{\partial^2 T}{\partial z^2}|_{i=3,j=4} = \frac{1}{(\Delta z)^2} \left(T_{25} - 2T_{18} + T_{11} \right). \quad (4.93)$$

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. (4.92) and (4.93) in a more general way as:

$$\frac{\partial^2 T}{\partial x^2}|_{i,j} = \frac{1}{(\Delta x)^2}\left(T_{(i-1)n_x+j+1} - 2T_{(i-1)n_x+j} + T_{(i-1)n_x+j-1}\right) \tag{4.94}$$

$$\frac{\partial^2 T}{\partial z^2}|_{i,j} = \frac{1}{(\Delta z)^2}\left(T_{i\cdot n_x+j} - 2T_{(i-1)n_x+j} + T_{(i-2)n_x+j}\right). \tag{4.95}$$

In matrix format this gives something like (*cf.* eq. 4.91)

$$A = \begin{pmatrix}
1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\
0 & 1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\
\vdots & \vdots & & & & & & & & & & & & \\
0 & 0 & & -s_z & \dots & -s_x & (1+2s_x+2s_z) & -s_x & \dots & -s_z & 0 & & 0 & 0 \\
0 & 0 & & 0 & -s_z & \dots & -s_x & (1+2s_x+2s_z) & -s_x & \dots & -s_z & & 0 & 0 \\
\vdots & \vdots & & & & & & & & & & & \vdots & \vdots \\
0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 \\
0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1
\end{pmatrix}. \tag{4.96}$$

Note that we now have five diagonals filled with non-zero entries as opposed to three diagonals in the 1D case. The solution vector $\boldsymbol{x}$ is given by

$$\boldsymbol{x} = \begin{pmatrix}
T_1^{n+1} = T_{1,1} \\
T_2^{n+1} = T_{1,2} \\
\vdots \\
T_{(i-1)n_x+j}^{n+1} = T_{i,j} \\
T_{(i-1)n_x+j+1}^{n+1} = T_{i,j+1} \\
\vdots \\
T_{n_x n_z-1}^{n+1} = T_{n_z,n_x-1} \\
T_{n_x n_z}^{n+1} = T_{n_z,n_x}
\end{pmatrix}, \tag{4.97}$$

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

$$\boldsymbol{b} = \begin{pmatrix}
T_{bottom} \\
T_{bottom} \\
\vdots \\
T_{(i-1)n_x+j}^{n} \\
T_{(i-1)n_x+j+1}^{n} \\
\vdots \\
T_{top} \\
T_{top}
\end{pmatrix}. \tag{4.98}$$

4.7.3 Other methods

The fully implicit method discussed above works fine, but is only first order accurate in time (sec. 4.3). 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 often preferred and does not cost much in terms of additional programming. You may consider using it for diffusion-type equations.

A different, and more serious, issue is the fact that the cost of solving `x = A\b` 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).

4.8 Exercise: 2D heat equation with FD

You are to program the diffusion equation in 2D both with an explicit and an implicit discretization scheme, as discussed above. 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 $T = 1500^\circ$ C impinges at the bottom of the lithosphere. What happen 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 4.7), 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 4.8), by programming the implicit finite difference approximation of the 2D temperature equation.

```
% 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
```

Figure 4.7: MATLAB script `heat2D_explicit.m` to solve the 2D heat equation using the explicit approach.

3. A simple (time-dependent) analytical solution for the temperature equation exists for the case that the initial temperature distribution is

$$T(x,z,t=0) = T_{max} \exp\left[\frac{-(x^2+z^2)}{\sigma^2}\right] \tag{4.99}$$

where T_{max} is the maximum amplitude of the temperature perturbation at $(x,z) =$

```
% 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  ))  =   ??;
        A( ii, Number(i  ,j+1))  =   ??;
  ??
    end
end
% Set lower and upper BC
for j = 1:nx
??
end
% Set left and right BC
for i = 1:nz
??
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);
          ??
        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,10)==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
```

Figure 4.8: MATLAB script `heat2D_explicit.m` to solve the 2D heat equation using the implicit approach.

$(0,0)$ and σ its half-width. The solution is

$$T(x,z,t) = \frac{T_{max}}{\sqrt{1+4t\kappa/\sigma^2}} \exp\left[\frac{-(x^2+z^2)}{\sigma^2+4t\kappa}\right]. \tag{4.100}$$

(As for the 1D example, note that this uses a characteristic, time-dependent length-scale of $l_c \propto \sqrt{\kappa t}$, as expected for a diffusion problem (*cf.* sec. 2.4), and see *Carslaw and Jaeger* (1959) for more analytical solutions.)

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 dt.

4. 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:* 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 hydro-geological problems (Darcy flow, *e.g.* how far did the chemical waste go into the aquifer?), fluid movements through the crust and through fault zones (which is related to the creation of ore deposits), magma migration through the mantle, geochemistry and mineral reactions at grain-boundary scale, and, aftershocks and fluids.

4.9 Advection equations with FD

Reading

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

4.9.1 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, sec. 2.4). 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 4.9 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 material transport, dominant in the interior the domain). How strongly these two effects are partitioned is indicated globally by the Rayleigh number, and locally by the Peclet number (sec. 2.4).

Mathematically, the temperature equation gets an additional term for advection in a Eulerian (fixed grid) system, and the partial time derivative, $\partial/\partial t$, is replaced by the total derivative

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla, \tag{4.101}$$

where this is equation is for an operator, that applies to a quantity, such as temperature.

In 1-D and in the absence of heat sources, the diffusion-advection equation becomes (sec. 7)

$$\rho c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} \right) = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) \tag{4.102}$$

or in 2-D

$$\rho c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} \right) = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) \tag{4.103}$$

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

$$\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \nabla T = \kappa \nabla^2 T. \tag{4.104}$$

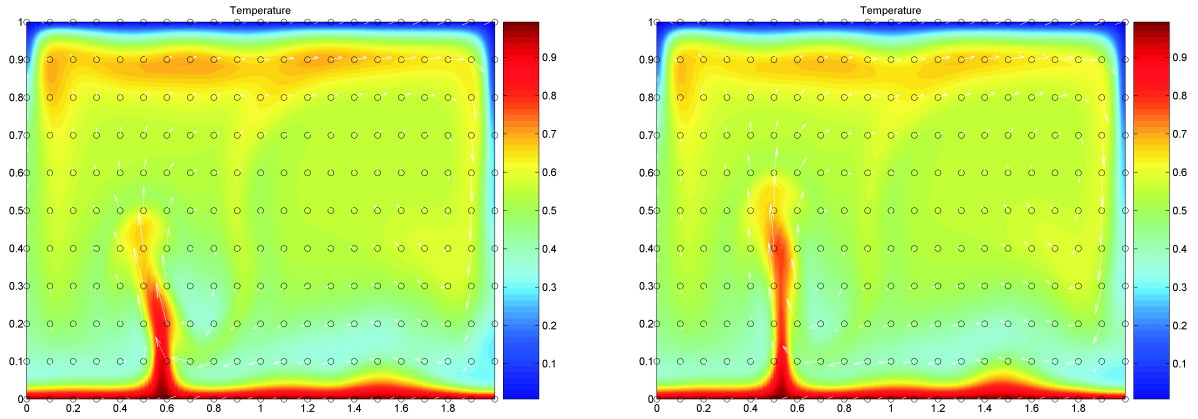


Figure 4.9: Snapshots of a bottom heated thermal convection model with a Rayleigh-number of 5×10^5 and constant viscosity (no internal heating). Temperature is advected through a fixed (Eulerian) grid (circles) with a velocity (arrows) that is computed with a Stokes solver.

Heat sources would lead to additional terms on the right hand side. Since temperature variations lead to buoyancy forces, the energy equation is coupled with the Stokes (conservation of momentum) equation from which velocities $\boldsymbol{v}$ can be computed to close the system needed for a convection algorithm.

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 non-physical artifacts in cases that are advection-dominated. One example would be if a chemical composition C is to be treated akin to T,

$$\frac{\partial C}{\partial t} + \boldsymbol{v} \cdot \nabla C = \kappa_c \nabla^2 C. \tag{4.105}$$

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

$$\frac{\partial C}{\partial t} + \boldsymbol{v} \cdot \nabla C = 0 \tag{4.106}$$

(*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, sec. 3)

$$\frac{dC}{dt} = 0 \quad \text{and} \quad \frac{d\boldsymbol{x}_i}{dt} = \boldsymbol{v} \tag{4.107}$$

where $\boldsymbol{x}_i$ is the location of the i-th tracer moving through the fluid. We will return to a hybrid approach (the semi-Lagrangian scheme) 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).

4.9.2 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

$$\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} = 0 \tag{4.108}$$

and

$$\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} = 0. \tag{4.109}$$

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. 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. (4.108) is by employing a central difference scheme in space, and go forward in time (another example of a forward-time, central space, FTCS, scheme):

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = -v_{x,i} \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x}, \tag{4.110}$$

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 (see Figure 4.10 and `exercise_1_ftcs.m`).

- Program the FTCS method in the code of Figure 4.10 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?

```
%
% 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) = ???
    end
    % BCs
    Tnew(1)     =   ???
    Tnew(nx)    =   ???
    % Update Solution & time incrememement
    T           =   Tnew;
    time        =   itime*dt;
    % Analytical solution for this case
    T_anal      =   ???
    figure(1),clf, plot(x,T,x,T_anal), ...
legend('Numerical','Analytical')
    xlabel('x')
    ylabel('temperature')
    drawnow
end
```

Figure 4.10: MATLAB script to be used with FTCS exercise 1.

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*, blows 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.

Lax method

The Lax approach consists of replacing the T_i^n in the time-derivative of eq. (4.110) with $(T_{i+1}^n + T_{i-1}^n)/2$. The resulting equation is

$$\frac{T_i^{n+1} - (T_{i+1}^n + T_{i-1}^n)/2}{\Delta t} = -v_{x,i}\frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x} \tag{4.111}$$

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:

$$\alpha = \frac{v_x \Delta t}{\Delta x} \tag{4.112}$$

- 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 following equation:

$$\left[M_x - \frac{\alpha^2}{(\Delta x)^2} \frac{\partial^2}{\partial x^2} \right] (T_i^{n+1} - T_i^n) + \alpha \Delta x \frac{\partial}{\partial x} T_i^n - \frac{\alpha^2 (\Delta x)^2}{2} \frac{\partial}{\partial x} T_i^n = 0 \tag{4.113}$$

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

$$\begin{aligned} &\frac{1}{6}\left(1 - c^2(\Delta x)^2\right)(T_{i+1}^{n+1} - T_{i-1}^{n+1}) + \frac{2}{3}\left(1 + c^2(\Delta x)^2\right) T_i^{n+1} \\ &\quad = \left[\frac{1}{6} - \frac{\alpha}{2} + \frac{\alpha^2(\Delta x)^2}{3}\right] T_{i+1}^n + \frac{2}{3}\left(1 - \alpha^2(\Delta x)^2\right) T_i^n + \left[\frac{1}{6} - \frac{\alpha}{2} + \frac{\alpha^2(\Delta x)^2}{3}\right] T_{i-1}^n \end{aligned} \tag{4.114}$$

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 exercise 2, 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 discretized 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 4.9 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

$$\alpha = \frac{|V| \Delta t}{\Delta x} \leq 1 \tag{4.115}$$

is called the *Courant criterion* (Figure 4.11).

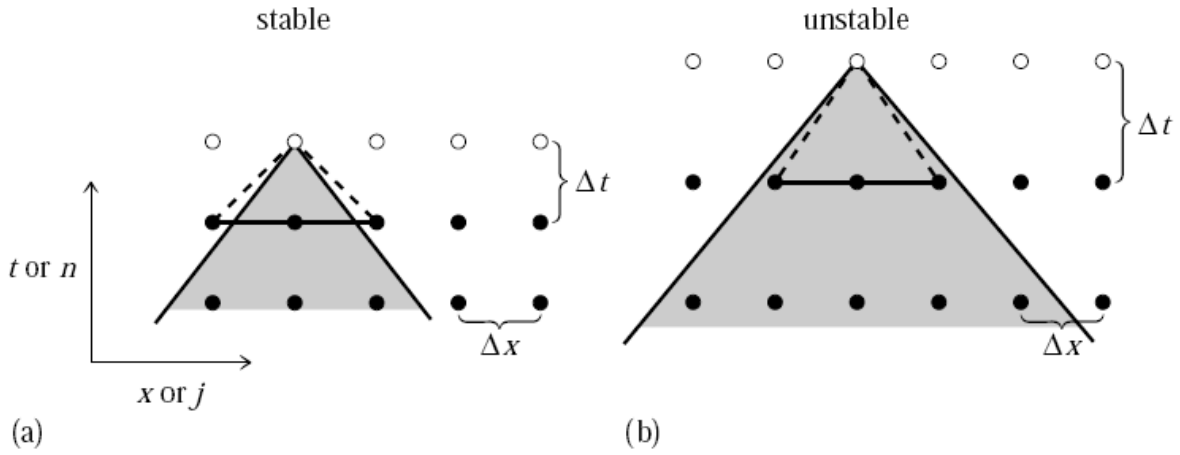


Figure 19.1.3. Courant condition for stability of a differencing scheme. The solution of a hyperbolic problem at a point depends on information within some domain of dependency to the past, shown here shaded. The differencing scheme (19.1.15) has its own domain of dependency determined by the choice 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 4.11: Illustration of the Courant criterion (from *Press et al.*, 1993, chap 19.1).

Streamline upwind scheme

A popular scheme is the so-called (streamline) upwind approach (Figure 4.12a). Here, the spatial finite difference scheme depends on the sign of the velocity:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = -v_{x,i} \begin{cases} \frac{T_i^n - T_{i-1}^n}{\Delta x}, \text{ if } v_{x,i} > 0 \\ \frac{T_{i+1}^n - T_i^n}{\Delta x}, \text{ if } v_{x,i} < 0 \end{cases} \tag{4.116}$$

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?

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 PDEs 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 $M_x = \{\delta, 1 - 2\,\delta, \delta\}$.

$$M_x \left[\frac{T_i^{n+1} - T_i^n}{\Delta t} + \frac{v}{2} \frac{(T_{i+1}^n - T_{i-1}^n) + (T_{i+1}^{n+1} - T_{i-1}^{n+1})}{2\Delta t} \right] = 0 \tag{4.117}$$

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. (4.117) written out with $\delta = \frac{1}{6}$.

$$\begin{aligned} \left[\frac{1}{6} - \frac{1}{4}\left(v\frac{\Delta t}{\Delta x}\right)\right] T_{i-1}^{n+1} - \left(1 - \frac{1}{3}\right) T_i^{n+1} + \left[\frac{1}{6} + \frac{1}{4}\left(v\frac{\Delta t}{\Delta x}\right)\right] T_{i+1}^{n+1} \\ = \left[\frac{1}{6} + \frac{1}{4}\left(v\frac{\Delta t}{\Delta x}\right)\right] T_{i-1}^{n} + \left(1 - \frac{1}{3}\right) T_i^{n} + \left[\frac{1}{6} - \frac{1}{4}\left(v\frac{\Delta t}{\Delta x}\right)\right] T_{i+1}^{n} \end{aligned} \tag{4.118}$$

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 Δt 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

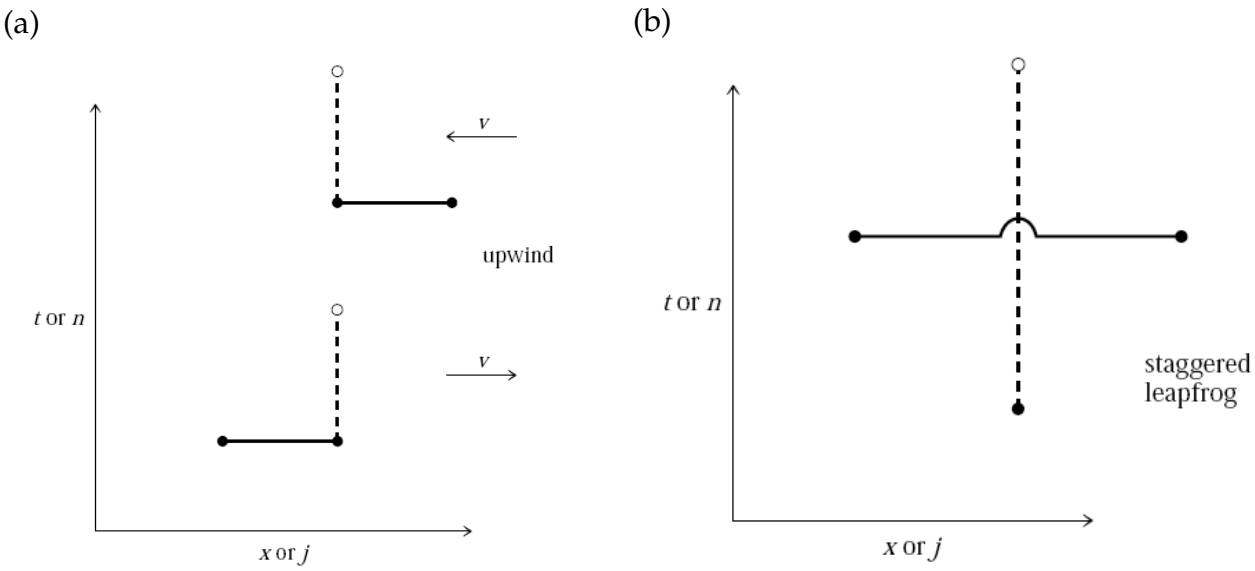


Figure 4.12: Illustration of the upwind (a) and leapfrog (b) schemes (from *Press et al.*, 1993, chap 19.1).

space

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = -v_{x,i}\frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x}, \tag{4.119}$$

called *staggered leapfrog* because of the way it's centered in shifted space-time (Figure 4.12b). 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 $T^{n-1} = T^n$).
- Try with different values of the Courant number α and compare the accuracy and stability of the different methods.
- Also, make the width of the Gaussian curve smaller.
- *Bonus:* Also program the two formulations of the Crank-Nicolson method with $\delta = 0$ and $\delta = \frac{1}{6}$.

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 4.9). Onward ever, backward never.

Similarly, the Crank-Nicolson method works well for $v \leq 0.1$ 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 approaches

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 4.13A 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 $\boldsymbol{X}$ where the particle came from by $\boldsymbol{X} = x_i - \Delta t v_x(t_{n+1}, x_i)$.

2. 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 as an option when calling the `interp1` function.

3. 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 4.13A. Is there a Courant criterion for stability?

Some improvements The algorithm described in Figure 4.13A 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 4.13B):

For each point i

1. 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).

2. 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

$$v_x(t_{n+1/2}, x_i) = \frac{v_x(t_{n+1}, x_i) + v_x(t_n, x_i)}{2}. \tag{4.120}$$

   Use linear interpolation for the spatial interpolation of velocity $v_x(t_{n+1/2}, X')$.

3. 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.

4. Compute the location X at t_n with the centered velocity $X = x_i - \Delta t v_x(t_{n+1/2}, X')$.

5. Use cubic interpolation to find the temperature at point X as before.

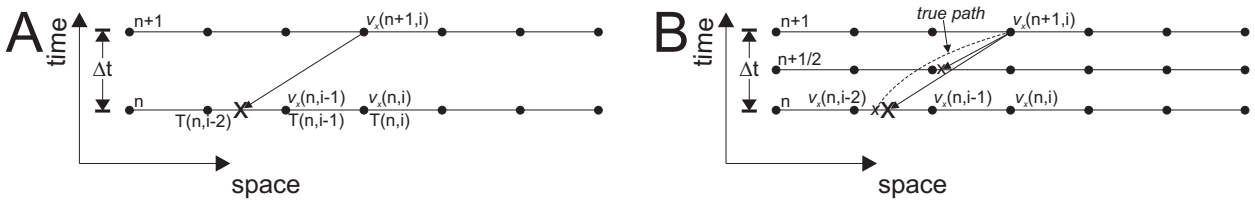


Figure 4.13: Basics of the semi-Lagrangian method. See text for explanation.

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 4.13B (*cf.* *Spiegelman*, 2004, p. 67).

  Some care has to be taken if point $\boldsymbol{X}$ 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

$$v_x(x,z) = z \tag{4.121}$$
$$v_z(x,z) = -x \tag{4.122}$$

Moreover, assume that the initial temperature distribution is Gaussian and given by

$$T(x,z) = 2\exp\left(\frac{(x+0.25)^2+z^2}{0.1^2}\right) \tag{4.123}$$

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 4.14 (`semi_lagrangian_2D_1.m`).

4.9.3 Advection and diffusion: operator splitting

In geodynamics, we often want to solve the coupled advection-diffusion equation, which is given by eq. (4.102) in 1-D and by eq. (4.103) 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

$$\frac{\tilde{T}^{n+1} - T^n}{\Delta t} + v_x \frac{\partial T}{\partial x} = 0 \tag{4.124}$$

for example by using a semi-Lagrangian advection scheme. Then solve the diffusion equation

$$\rho c_p \frac{\partial \tilde{T}}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial \tilde{T}}{\partial x}\right) + Q. \tag{4.125}$$

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 section's exercise). Base your code on the script of Figure 4.14.

```
% semi_lagrangian_2D: 2D semi-lagrangian  with center midpoint time stepping method
%

clear all

W       =   40;     % width of domain
sigma   =   .1;
Ampl    =   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       =   Ampl*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);
   %         for ??
%                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 4.14: MATLAB script to be used with exercise 7.

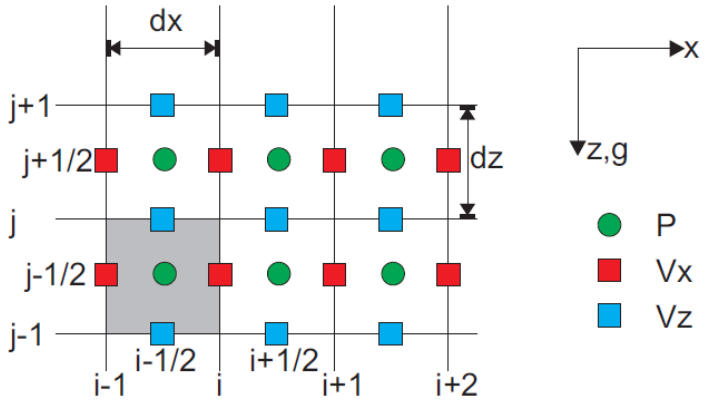


Figure 4.15: Staggered grid definition. Properties such as viscosity and density inside a control volume (gray) are assumed to be constant. Moreover, a constant grid spacing in x and z-direction is assumed.

4.10 2D Stokes equations on a staggered grid using primitive variables

4.10.1 Introduction

The basis of basically all mantle convection and lithospheric dynamics codes are the so-called Stokes equations for slowly moving viscous fluids. These equations describe the balance between buoyancy forces (*e.g.* due to temperature variations in the fluid) and viscous drag (sec. 7). Here, we will describe the governing equations. There are several ways to solve those equations, and the goal of this project is to use a staggered finite difference approach in primitive variables.

For this, we solve the governing equations for $\boldsymbol{v} = \{v_x; v_z\}$ (velocities) and p (pressure). Staggered finite differences means that the different unknowns v_x, v_z, p are defined at physically different grid points. The main challenges of this project are, 1), having several variables instead of only one (*e.g.* temperature), and, 2), to do the bookkeeping for the present case that the variables are at different grid points. (While the governing equations are different, those computational challenges are similar to those arising in the staggered grid, finite difference approach for wave propagation discussed in sec. 4.12.2.)

4.10.2 Governing equations

It is assumed that the rheology is incompressible and that the rheology is Newtonian viscous, *i.e.* $\sigma = 2\mu\dot{\varepsilon}$ with μ no function of $\dot{\varepsilon}$, where σ is the stress tensor, μ viscosity, and $\dot{\varepsilon}$ strain-rate tensor. In this case, the governing equations in 2D (x and z) are (see sec. 7):

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = 0 \tag{4.126}$$

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z} = 0 \tag{4.127}$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z} - \rho g = 0 \tag{4.128}$$

$$\sigma_{xx} = -p + 2\mu \frac{\partial v_x}{\partial x} \tag{4.129}$$

$$\sigma_{zz} = -p + 2\mu \frac{\partial v_z}{\partial z} \tag{4.130}$$

$$\sigma_{xz} = \mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right), \tag{4.131}$$

where ρ is density and $\boldsymbol{g} = \{0, g\}$ the gravitational acceleration. The density is where these continuum and force balance equations (eqs. 4.126 to 4.128) couple to the the energy equation, *e.g.* the diffusion and advection of temperature for mantle convection, discussed in the previous sections.

It has been suggested that a particularly nice way to solve these equations is to use a staggered grid (more about this later) and to keep as variables v_x, v_z and p (*Gerya and Yuen*, 2003; *Gerya*, 2009).[1] Since there are three variables, we need three equations. Substituting eqs. (4.129)-(4.131) into eq. (4.127) and eq. (4.128) leads to:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = \frac{p}{\gamma} \tag{4.132}$$

$$-\frac{\partial P}{\partial x} + 2\frac{\partial}{\partial x}\left(\mu \frac{\partial v_x}{\partial x}\right) + \frac{\partial}{\partial z}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) = 0 \tag{4.133}$$

$$-\frac{\partial P}{\partial z} + 2\frac{\partial}{\partial z}\left(\mu \frac{\partial v_z}{\partial z}\right) + \frac{\partial}{\partial x}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) = \rho g \tag{4.134}$$

Note that we added the term $\frac{p}{\gamma}$ to the incompressibility equations. This is a "trick" called the penalty method, which ensures that the system of equations does not become ill-posed. For this to work, γ should be sufficiently large ($\sim 10^4$ or so), so that the condition of incompressibility (conservation of mass, eq. 4.126) is approximately satisfied.

4.10.3 Exercise

1. Discretize eqs. (4.132)-(4.134) on a staggered grid as shown on Figure 4.15.
2. A MATLAB subroutine is shown on Figure 4.16. The subroutine sets up the grid, the node numbering and discretizes the incompressibility equations.

[1] For a comparison of different finite difference approaches, see **?**, for example.

```
% Solve the 2D Stokes equations on a staggered grid, using the Vx,Vz,P formulation.
clear
% Material properties
%               phase #1  phase #2
mu_vec      =   [1      1   ];
rho_vec     =   [1      2   ];
% Input parameters
Nx          =   20;
Nz          =   .9*Nx;
W           =   1;
H           =   1;
g           =   1;
% Setup the interface
x_int       =   0:.01:W;
z_int       =   cos(x_int*2*pi/W)*1e-2 - 0.5;
% Setup the grids-------------------------------------------------------------
dz          =   H/(Nz-1);
dx          =   W/(Nx-1);
[X2d,Z2d]   =   meshgrid(0:dx:W,-H:dz:0);
XVx         =   [X2d(2:end,:) + X2d(1:end-1,:)]/2; % Vx
ZVx         =   [Z2d(2:end,:) + Z2d(1:end-1,:)]/2;
XVz         =   [X2d(:,2:end) + X2d(:,1:end-1)]/2; % Vz
ZVz         =   [Z2d(:,2:end) + Z2d(:,1:end-1)]/2;
XP          =   [X2d(2:end,2:end) + X2d(1:end-1,1:end-1)]/2; % p
ZP          =   [Z2d(2:end,2:end) + Z2d(1:end-1,1:end-1)]/2;
% Compute material properties from interface, properties are computed in the center of a control volume
Rho             =   ones(Nz-1,Nx-1)*rho_vec(2);
Mu              =   ones(Nz-1,Nx-1)*mu_vec(2);
z_int_intp      =   interp1(x_int,z_int,XP(1,:));
for ix = 1:length(z_int_intp)
    ind         =   find(ZVz(:,1)<z_int_intp(ix));
    Rho(ind(1:end-1),ix) = mu_vec(1);
    Mu(ind(1:end-1),ix)  = rho_vec(1);
    fac         =  (z_int_intp(ix) - ZVz(ind(end),1))/dz;
    Rho(ind(end),ix)  =  fac*rho_vec(1) + (1-fac)*rho_vec(2);
    Mu(ind(end),ix)   =  fac*mu_vec( 2) + (1-fac)*mu_vec( 2);
end
% Setup numbering scheme------------------------------------------------------
Number_Phase      =   zeros(Nz + Nz-1, Nx + Nx-1);  % Create the general numbering scheme
Number_ind        =   zeros(Nz + Nz-1, Nx + Nx-1);  % Create the general numbering scheme
Number_Vx         =   zeros(Nz-1,Nx  );Number_Vz        =   zeros(Nz  ,Nx-1);
Number_P          =   zeros(Nz-1,Nx-1);
for ix=1:2:Nx+Nx-1, for iz=2:2:Nz+Nz-1, Number_Phase(iz,ix) = 1; end; end % Vx equations
for ix=2:2:Nx+Nx-1, for iz=1:2:Nz+Nz-1, Number_Phase(iz,ix) = 2; end; end % Vz equations
for ix=2:2:Nx+Nx-1, for iz=2:2:Nz+Nz-1, Number_Phase(iz,ix) = 3; end; end % P  equations
num             =     1;
for ix=1:size(Number_Phase,2)
    for iz=1:size(Number_Phase,1)
        if Number_Phase(iz,ix)~=0
            Number_ind(iz,ix)   =   num;
            num                 =   num+1;
        end
    end
end
num_eqns        =   num-1;
ind_Vx          =   find(Number_Phase==1);  Number_Vx(find(Number_Vx==0)) = Number_ind(ind_Vx);
ind_Vz          =   find(Number_Phase==2);  Number_Vz(find(Number_Vz==0)) = Number_ind(ind_Vz);
ind_P           =   find(Number_Phase==3);  Number_P (find(Number_P ==0)) = Number_ind(ind_P );
% Setup the stiffness matrix
A                   =   sparse(num_eqns,num_eqns);
Rhs_vec             =   zeros(num_eqns,1);
% Setup the incompressibility equations---------------------------------------
ind_list = [];ind_val  = [];
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_Vx(:,2:end  ),  ( 1/dx));%dVx/dx
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_Vx(:,1:end-1),  (-1/dx));
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_Vz(2:end,:  ),  ( 1/dz));%dVz/dz
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_Vz(1:end-1,:),  (-1/dz));
% Add local equations to global matrix
for i=1:size(ind_list,2)
    A = A + sparse([1:size(ind_list,1)].',ind_list(:,i),ind_val(:,i),num_eqns,num_eqns);
end
num_incomp = length(ind_list);
% Perform testing of the system of equation, setup some given matrixes
mu  = mu_vec(1);
Vx                      =         cos(XVx).*sin(ZVx);
Vz                      =        -sin(XVz).*cos(ZVz);
P                       =    2*mu*sin(XP ).*sin(ZP );
C                       =   zeros(num_eqns,1);
C(Number_Vx(:))         =   Vx(:);
C(Number_Vz(:))         =   Vz(:);
C(Number_P(:))          =   P(:);
Rhs                     =   A*C;
% Check whether the compressibility equations are implemented correctly
max(abs(Rhs(1:num_incomp)))
```

Figure 4.16: MATLAB script `Staggered_Stokes.m` that sets up numbering, matrix A and that solves the incompressibility equations.

```
function [ind_list,ind_val] = Add_coeffs(ind_list,ind_val,ind_add,val_add)% Add coefficients to an array
if (length(val_add(:))==1)
   val_add  =   ones(size(ind_add))*val_add;
end
ind_list    =   [ind_list, ind_add(:)];ind_val     =   [ind_val , val_add(:)];
```

Figure 4.17: MATLAB script `Add_coeffs.m`, used by `Staggered_Stokes.m`.

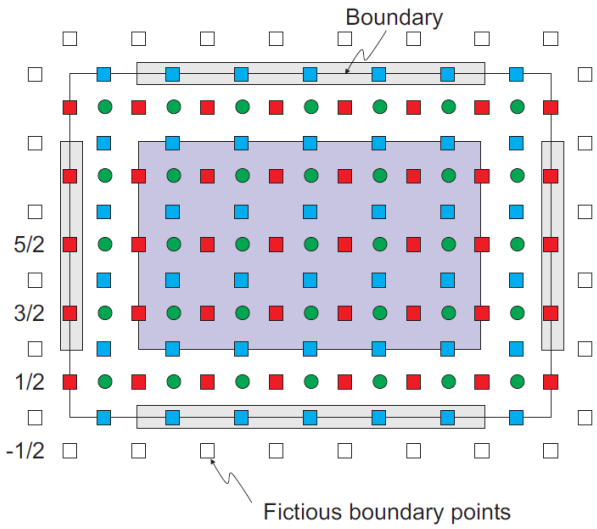


Figure 4.18: Staggered grid definition with the boundary points. Within the purple domain, the finite difference scheme for center points can be applied. At the boundaries, we have to apply a special finite difference scheme which employ fictious boundary nodes.

Add the discretization of the force balance equations (including the effects of gravity) into the equation matrix A. Assume that the viscosity is constant and $\mu = 1$ in a first step, but density is variable.

An example is given in how to verify that the incompressibility equation is incorporated correctly. This is done by assuming a given (sinusoidal) function for, let's say, v_x (*e.g.* $v_x = \cos(\omega x)\cos(\omega z)$). From the incompressibility equation (eq. 4.126) a solution for v_z than follows. By setting those solutions in the $\boldsymbol{c}$ vector, we can compute $A\boldsymbol{c}$ and verify that $\boldsymbol{rhs}$ for those equations is indeed zero.

3. Add free-slip boundary conditions on all sides (which means $v_z = 0, \sigma_{xz} = 0$ on the lower and upper boundaries and $\sigma_{xz} = 0, v_x = 0$ on the side boundaries). Use fictious boundary points to incorporate the σ_{xz} boundary conditions.

4. Assume a model domain $x = [0;1], \quad z = [0;1]$, and assume that the density below $z = 0.1\cos(2\pi x) + 0.5$ is 1, whereas the density above it is 2. Compute the velocity and pressure, and plot the velocity vectors.

5. Write the code for the case of variable viscosity (which is relevant for the Earth since rock properties are a strong function of temperature).

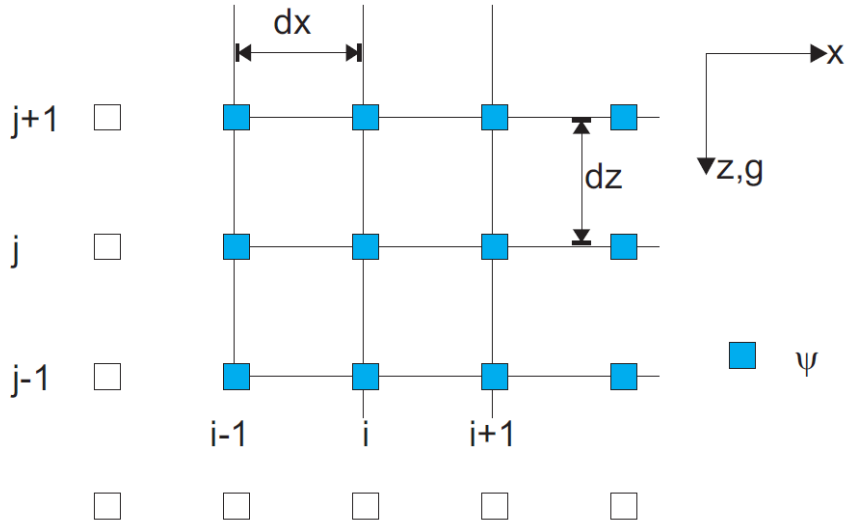


Figure 4.19: Discretization for the streamfunction approach. The boundary conditions are set through fictious boundary points.

4.11 Stokes equations with FD on a staggered grid using the stream-function approach.

4.11.1 Introduction

As was discussed in sec. 4.10, the basis of basically all mantle convection and lithospheric dynamics codes are the Stokes equations for slowly moving viscous fluids.

There are several ways to solve those equations, and the goal of this exercise is to use a streamfunction, finite difference approach. Stream function means that there is a potential field which we solve for, and then obtain velocities from the derivatives of this field. The advantage of this approach is that the continuity equation for incompressible flow can be satisfied implicitly, rather than having to use a panelty parameter as for the primitive variable approach of sec. 4.10. (It is, however, possible to formulate the stream function method for compressible convection approximations, *e.g.* **?**). For a comparison of different finite difference approaches, see **?**, for example.

The main challenges of this project are, 1), having fairly high-order and mixed derivatives (up to 4^{th} order) and, 2), setting of boundary conditions.

4.11.2 Governing equations

It is assumed that the rheology is incompressible and that the rheology is Newtonian viscous. In this case, the governing equations are (see sec. 7):

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = 0 \tag{4.135}$$

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z} = 0 \tag{4.136}$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z} - \rho g = 0 \tag{4.137}$$

$$\sigma_{xx} = -p + 2\mu \frac{\partial v_x}{\partial x} \tag{4.138}$$

$$\sigma_{zz} = -p + 2\mu \frac{\partial v_z}{\partial z} \tag{4.139}$$

$$\sigma_{xz} = \mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right) \tag{4.140}$$

By substituting eqs. (4.138)-(4.140) into eqs. (4.135)-(4.137), we obtain (compare sec. 4.10)

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = 0 \tag{4.141}$$

$$-\frac{\partial p}{\partial x} + 2\frac{\partial}{\partial x}\left(\mu \frac{\partial v_x}{\partial x}\right) + \frac{\partial}{\partial z}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) = 0 \tag{4.142}$$

$$-\frac{\partial p}{\partial z} + 2\frac{\partial}{\partial z}\left(\mu \frac{\partial v_z}{\partial z}\right) + \frac{\partial}{\partial x}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) = \rho g \tag{4.143}$$

We can eliminate pressure from eqs. (4.142) and (4.143) by taking the derivative of eq. (4.142) versus z and subtracting eq. (4.143) derived versus x. This results in:

$$\begin{aligned} 2\frac{\partial^2}{\partial x \partial z}\left(\mu \frac{\partial v_x}{\partial x}\right) - 2\frac{\partial^2}{\partial x \partial z}\left(\mu \frac{\partial v_z}{\partial z}\right) \quad + \\ \frac{\partial^2}{\partial z^2}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) - \frac{\partial^2}{\partial x^2}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) = -\frac{\partial}{\partial x}\rho g. \end{aligned} \tag{4.144}$$

We can also use the incompressibility constraint (4.141) to simplify things a little bit more:

$$\begin{aligned} -4\frac{\partial^2}{\partial x \partial z}\left(\mu \frac{\partial v_z}{\partial z}\right) \quad + \\ \frac{\partial^2}{\partial z^2}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) - \frac{\partial^2}{\partial x^2}\left(\mu \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)\right) = -\frac{\partial}{\partial x}\rho g \end{aligned} \tag{4.145}$$

Now we introduce a variable Ψ (the stream function) which is defined by its relationship to the velocities as

$$v_x = \frac{\partial \Psi}{\partial z} \tag{4.146}$$

$$v_z = -\frac{\partial \Psi}{\partial x} \tag{4.147}$$

Note that Ψ satisfies incompressibility by plugging eqs. (4.146) and (4.147) into eq. (4.135).

By using Ψ, we can write eq. (4.145) as:

$$\begin{aligned} 4\frac{\partial^2}{\partial x \partial z}\left(\mu \frac{\partial^2 \Psi}{\partial x \partial z}\right) \quad + \\ \frac{\partial^2}{\partial z^2}\left(\mu\left(\frac{\partial^2 \Psi}{\partial z^2}-\frac{\partial^2 \Psi}{\partial x^2}\right)\right)-\frac{\partial^2}{\partial x^2}\left(\mu\left(\frac{\partial^2 \Psi}{\partial z^2}-\frac{\partial^2 \Psi}{\partial x^2}\right)\right) \quad = \quad -\frac{\partial}{\partial x}\rho g. \end{aligned} \tag{4.148}$$

Note that this equation now has 4^{th} order derivatives for Ψ (easier to see for constant μ, where we can pull the viscosity out of the derivatives.) The challenge is to solve eq. (4.148) for Ψ given then density gradients.

4.11.3 Exercise

1. Discretize eq. (4.148) on a grid as shown on Figure 4.19.
2. A MATLAB subroutine is shown on Figure 4.20. The subroutine sets up the grid and the node numbering. Finish the code by programming the discretized eq. (4.148). To start simple, assume that viscosity is constant.
3. Add free-slip boundary conditions on all sides (which means $v_z = 0, \sigma_{xz} = 0$ on the lower and upper boundaries and $\sigma_{xz} = 0, v_x = 0$ on the side boundaries; you'll have to write these equations in terms of Ψ and employ fictious boundary points).
4. Assume a model domain $x = [0;1], \quad z = [0;1]$, and assume that the density below $z = 0.1\cos(2\pi x) + 0.5$ is 1, whereas the density above it is 2. Compute the velocity, and plot the velocity vectors.
5. Write the code for the case of variable viscosity (which is relevant for the Earth since rock properties are a strong function of temperature).

```
% Solve the 2D Stokes equations on a staggered grid, using the Vx,Vz,P
% formulation.
clear
% Material properties         phase #1 phase #2
mu_vec =  [1 1 ];
rho_vec = [1 2 ];
% Input parameters
Nx = 6;
Nz = 6;
W = 1;H = 1;g = 1;
% Setup the interface
x_int = 0:.01:W;
z_int = cos(x_int*2*pi/W)*1e-2 - 0.5;
% Setup the grids-------------------------------------------------------------
dz = H/(Nz-1);dx = W/(Nx-1);
[X2d,Z2d] = meshgrid(0:dx:W,-H:dz:0);
%--------------------------------------------------------------------------
% Compute material properties from interface----------------------------
% Properties are computed in the center of a control volume
Rho = ones(Nz,Nx)*rho_vec(2);
Mu = ones(Nz,Nx)*mu_vec(2);
z_int_intp = interp1(x_int,z_int,X2d(1,:));
for ix = 1:length(z_int_intp)
  ind = find(Z2d(:,1)<z_int_intp(ix));
  Rho(ind(1:end-1),ix) = mu_vec(1);
  Mu(ind(1:end-1),ix) = rho_vec(1);
  fac = (z_int_intp(ix) - Z2d(ind(end),1))/dz;
  Rho(ind(end),ix) = fac*rho_vec(1) + (1-fac)*rho_vec(2);
  Mu(ind(end),ix) = fac*mu_vec( 2) + (1-fac)*mu_vec( 2);
end
%--------------------------------------------------------------------------
% Setup numbering scheme------------------------------------------------------
Number_ind = zeros(Nz, Nx); % Create the general numbering scheme
num = 1;
for ix=1:Nx
  for iz=1:Nz
    Number_ind(iz,ix) = num;
    num = num+1;
  end
end
num_eqns = num-1;
%--------------------------------------------------------------------------
% Setup the stiffness matrix
A = sparse(num_eqns,num_eqns);Rhs_vec = zeros(num_eqns,1);
% Compute coefficients for mu*d4 Psi/dx4
ind_list = [];ind_val = [];
mu = mu_vec(1);
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_ind(1:end,5:end ), mu*( 1/dx4));
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_ind(1:end,4:end-1), mu*(-4/dx4));
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_ind(1:end,3:end-2), mu*( 6/dx4));
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_ind(1:end,2:end-3), mu*(-4/dx4));
[ind_list,ind_val] = Add_coeffs(ind_list,ind_val, Number_ind(1:end,1:end-4), mu*( 1/dx4));
% Compute coefficients for d4 Psi/dx2/dz2
% compute coefficients for ....
% Add local equations to global matrix
ii = 1;
for i=1:size(ind_list,2)
  A = A + sparse([ii+1:ii+size(ind_list,1)].,ind_list(:,i),ind_val(:,i),num_eqns,num_eqns);
end
% set rhs
%Rhs_vec([ii+1:ii+size(ind_list,1)]) = Rhs_vec([ii+1:ii+size(ind_list,1)]) + rho(:)*g;
% Set boundary conditions
% Solve system of equations.
```

Figure 4.20: Code `Streamfunction_Stokes.m` that initializes the grid and node numbering for the 2D streamfunction approach.

```
function [ind_list,ind_val] = Add_coeffs(ind_list,ind_val,ind_add,val_add)% Add coefficients to an array
if (length(val_add(:))==1)
   val_add  =   ones(size(ind_add))*val_add;
end
ind_list    =   [ind_list, ind_add(:)];ind_val     =   [ind_val , val_add(:)];
```

Figure 4.21: MATLAB script `Add_coeffs.m`, used by `Streamfunction_Stokes.m`.

4.12 Wave propagation

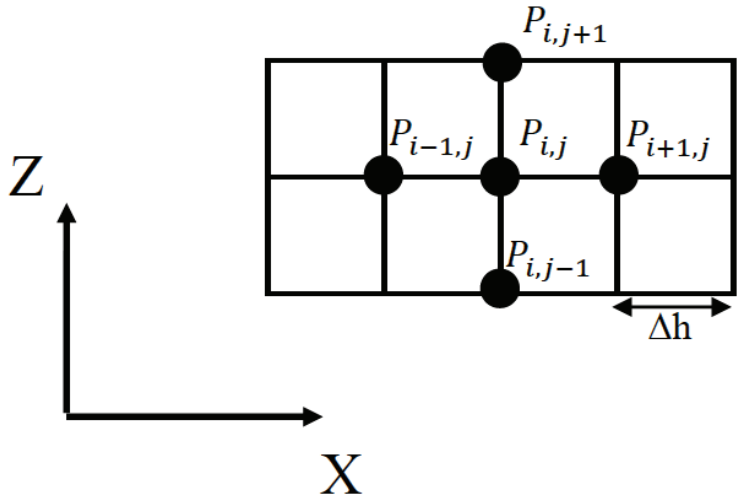


Figure 4.22: Finite difference discretization of the 2D acoustic problem.

We briefly discuss two examples for solving wave propagation type problems with finite differences, the acoustic and the seismic problem.

4.12.1 Acoustic problem with standard grid

In an isotropically elastic medium, acoustic wave propagation, where we are only taking care of a single type of wave, can be described by a set of two partial differential equations, leading to an hyperbolic problem. Firstly, Newtons 2^{nd} law,

$$\frac{\partial^2 \boldsymbol{u}}{\partial t^2} = \ddot{\boldsymbol{u}} = \frac{1}{\rho}\nabla^2 p = b\nabla^2 p \tag{4.149}$$

and, secondly, a constitutive law,

$$p = K\nabla \cdot \boldsymbol{u}. \tag{4.150}$$

Here, $\boldsymbol{u} = \{u_x, u_y, u_z\}$ are the three components of particle displacement, p is pressure, b is buoyancy, which is the inverse of density, ρ, and K is the bulk modulus, or compressibility.

Substituting eq. (4.150) for $\boldsymbol{u}$ into eq. (4.149), we can find

$$\frac{\partial^2 p}{\partial t^2} = K\nabla \cdot (b\nabla p). \tag{4.151}$$

If we assume that density is constant, then

$$\frac{\partial^2 p}{\partial t^2} = v^2\nabla^2 p, \tag{4.152}$$

where

$$v = \sqrt{Kb} = \sqrt{\frac{K}{\rho}} \tag{4.153}$$

denotes the bulk sound velocity. Simplifying to a 2D case, we have

$$\frac{\partial^2 p}{\partial t^2} = v^2(x,z)\left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial z^2}\right). \tag{4.154}$$

The equation for propogation of SH waves, the transverse components of S waves, in seismology has a similar form as eq. (4.154):

$$\frac{\partial^2 u}{\partial t^2} = v_{SH}^2(x,z)\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2}\right), \tag{4.155}$$

where u is the displacement and V_{SH} is the velocity of the SH component.

Likewise, a similar equation also applies for tsunami waves at long wavelengths, in the "shallow water approximation",

$$\frac{\partial^2 \xi}{\partial t^2} = v^2(x,z)\left(\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial z^2}\right). \tag{4.156}$$

Here, ξ is the height of the tsunami wave, v is the velocity defined by the water depth H as

$$v = \sqrt{gH}. \tag{4.157}$$

To solve equations (4.154)-(4.156), with finite differences, we use the mesh shown in Fig. 4.22. Here, we have $p_{i,j}^n = P(i\Delta h, j\Delta h, n\Delta t)$ and $v_{i,j} = v(i\Delta h, j\Delta h)$. Applying the 2^{nd}-order, second derivative formula to the acoustic wave equation eq. (4.154),

$$\frac{p_{i,j}^{n-1} - 2p_{i,j}^n + p_{i,j}^{n+1}}{\Delta t^2} = v_{i,j}^2\left[\frac{p_{i-1,j}^n - 2p_{i,j}^n + p_{i+1,j}^n}{\Delta h^2} + \frac{p_{i,j-1}^n - 2p_{i,j}^n + p_{i,j+1}^n}{\Delta h^2}\right]. \tag{4.158}$$

After rearranging, we have

$$p_{i,j}^{n+1} = -p_{i,j}^{n-1} + (2 - 4a_{i,j})2p_{i,j}^n + a_{i,j}\left(p_{i-1,j}^n + p_{i+1,j}^n + p_{i,j-1}^n + p_{i,j+1}^n\right), \tag{4.159}$$

where

$$a_{i,j} = v_{i,j}^2 \frac{\Delta h^2}{\Delta t^2}. \tag{4.160}$$

Then, the pressure or displacement at time step $n+1$ can be derived explicitly from time step n and $n-1$ as in eq. (4.159), though two solutions have to be stored.

Note that we use 2^{nd}-order second derivatives in eq. (4.159). Two considerations are required for choosing suitable time step Δt and spatial step Δh: grid dispersion and stability.

When waves propagate on a discrete grid, they produces an artificial variation of velocity with frequency, which is called grid dispersion. The higher frequency signals, with slower velocity, are delayed relative to the lower frequency arrivals. This dispersion increases as Δh becomes larger. In other words, a small Δh is required to avoid grid dispersion.

To achieve an accurate solution, we need at least 12 points per wavelength for space for a scheme with 2^{nd} order accuracy. For a 4^{th} order scheme, a minimum of 6.5 points per wavelength are required. For a fixed frequency, this minimum wavelength is determined by the minimum velocity (v_{min}), so the accuracy of the system is governed by (v_{min}). Following a stability analysis, we can derive the stability requirement here as:

$$\Delta t \leq \frac{1}{\sqrt{2}} \frac{\Delta h}{v_{max}} \tag{4.161}$$

where v_{max} is the maximum velocity on the grid.

Exercise 1

- Program the 2D acoustic wave propagation in standard grid scheme as in Fig. 4.22 (`wave_acoustic_2D.m`). Study the wavefield and seismograms with different choices of Δt and Δh and demonstrate how Δt and Δh affect the stability and grid dispersion in the program.
- Introduce heterogeneities in the velocities, such as a thin layer with half velocity, and describe the difference from the isotropic model, especially how this layer affects the observed seismograms. Run the code with the velocity inside the thin layer being zero and explain the result.

4.12.2 Elastic wave problem with staggered grid

For 2D elastic wave case (P-SV system), we have the equations as:

$$\rho \frac{\partial^2 u_x}{\partial t^2} = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xz}}{\partial z}, \tag{4.162}$$

$$\rho \frac{\partial^2 u_z}{\partial t^2} = \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{zz}}{\partial z}, \tag{4.163}$$

$$\tau_{xx} = (\lambda + 2\mu) \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_z}{\partial z}, \tag{4.164}$$

$$\tau_{zz} = (\lambda + 2\mu) \frac{\partial u_z}{\partial z} + \lambda \frac{\partial u_x}{\partial x}, \tag{4.165}$$

and

$$\tau_{xz} = \mu \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right). \tag{4.166}$$

Here, (u_x, u_z) are the particle displacements. For seismic waves, they are typically called radial and vertical components, respectively, if they are recorded at surface. Further, τ is the stress tensor, and $\lambda(x,z)$ and $\mu(x,z)$ the elastic, Lamé coefficients (μ is shear modulus). Typically, those equations are solved for particle velocities as $u = \frac{\partial u_x}{\partial t}$ and $v = \frac{\partial u_z}{\partial t}$. Then, the system is transformed into the frst-order hyperbolic system:

$$\frac{\partial u}{\partial t} = b\left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xz}}{\partial z}\right), \tag{4.167}$$

$$\frac{\partial v}{\partial t} = b\left(\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{zz}}{\partial z}\right), \tag{4.168}$$

$$\frac{\partial \tau_{xx}}{\partial t} = (\lambda + 2\mu)\frac{\partial u}{\partial x} + \lambda\frac{\partial v}{\partial z}, \tag{4.169}$$

$$\frac{\partial \tau_{zz}}{\partial t} = (\lambda + 2\mu)\frac{\partial v}{\partial x} + \lambda\frac{\partial u}{\partial z}, \tag{4.170}$$

$$\frac{\partial \tau_{xz}}{\partial t} = \mu\left(\frac{\partial u}{\partial z} + \frac{\partial v}{\partial x}\right). \tag{4.171}$$

A typical seismic wave propagation problem needs to deal with medium with variable Poisson's ratio, ν, which can be defined as

$$\nu = \frac{\lambda}{2(\lambda + \mu)}. \tag{4.172}$$

For the special case of $\lambda = \mu$, $\nu = 0.25$, and many rocks have Poisson's ratios not far from $1/4$. For liquids, $\nu \to 0.5$. For seismic wave propagation, this is particularly important when ocean water or the outer core of the Earth are needed to be considered in the problem, which is hard to be resolved with the traditional set up of grid as in Fig. 4.22.

To satisfy both requirements for stability and grid dispersion at those problems, a *P-SV* staggered-grid scheme is applied. Note the structure of the elastic wave problem, eq. (4.167)-eq. (4.171), they allow the stress and particle velocity to be spatially interlaced on the grids as in Fig. 4.23. The staggered-grid scheme allows the spatial derivative to be computed to a much higher accuracy (*e.g. Levander*, 1988). (This computational aspect is similar to the staggered grid finite difference approach to the Stokes problem, discussed in sec. 4.10.)

To add the complexity, the stress and velocity field can also staggered in time. Follow the explicit scheme, the second order difference equations for Equation (4.167)-(4.171) are:

$$\begin{aligned} u^{k+1/2}_{i+1/2,j} = u^{k-1/2}_{i+1/2,j} &+ b_{i+1/2,j}\frac{\Delta t}{\Delta h}\left(\Sigma^k_{i+1,j} - \Sigma^k_{i,j}\right) \\ &+ b_{i+1/2,j}\frac{\Delta t}{\Delta h}\left(\Gamma^k_{i+1/2,j+1/2} - \Gamma^k_{i+1/2,j-1/2}\right), \end{aligned} \tag{4.173}$$

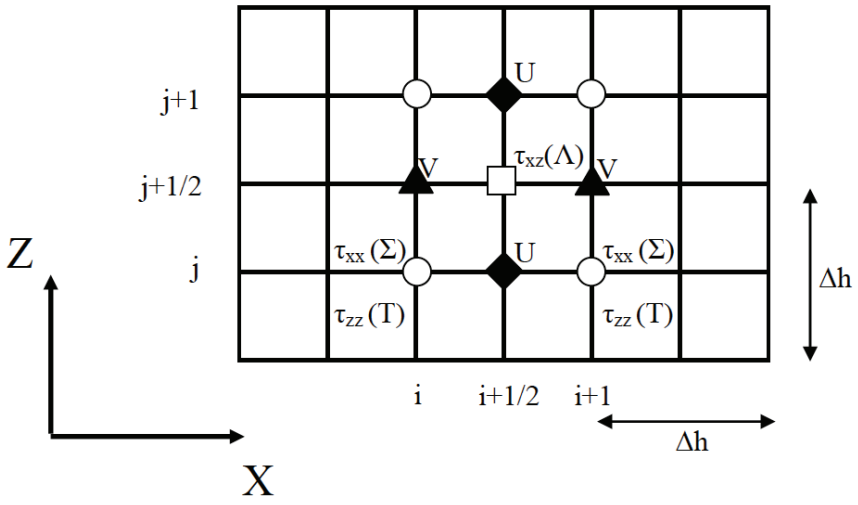


Figure 4.23: 2D staggered finite difference grid for wave propagation.

$$v_{i,j+1/2}^{k+1/2} = v_{i,j+1/2}^{k-1/2} + b_{i,j+1/2}\frac{\Delta t}{\Delta h}\left(\Gamma_{i+1/2,j+1/2}^{k} - \Gamma_{i-1/2,j+1/2}^{k}\right) + b_{i,j+1/2}\frac{\Delta t}{\Delta h}\left(\Xi_{i,j+1}^{k} - \Xi_{i,j}^{k}\right), \tag{4.174}$$

$$\Sigma_{i,j}^{k+1} = \Sigma_{i,j}^{k} + (\lambda + 2\mu)_{i,j}\frac{\Delta t}{\Delta h}\left(u_{i+1/2,j}^{k+1/2} - u_{i-1/2,j}^{k+1/2}\right) + \lambda_{i,j}\frac{\Delta t}{\Delta h}\left(v_{i,j+1/2}^{k+1/2} - v_{i,j-1/2}^{k+1/2}\right), \tag{4.175}$$

$$\Xi_{i,j}^{k+1} = \Xi_{i,j}^{k} + (\lambda + 2\mu)_{i,j}\frac{\Delta t}{\Delta h}\left(v_{i,j+1/2}^{k+1/2} - v_{i,j-1/2}^{k+1/2}\right) + \lambda_{i,j}\frac{\Delta t}{\Delta h}\left(u_{i+1/2,j}^{k+1/2} - v_{i-1/2,j}^{k+1/2}\right), \tag{4.176}$$

$$\Gamma_{i+1/2,j+1/2}^{k+1} = \Gamma_{i+1/2,j+1/2}^{k} + \mu_{i+1/2,j+1/2}\frac{\Delta t}{\Delta h}\left(v_{i+1,j+1/2}^{k+1/2} - v_{i,j+1/2}^{k+1/2}\right) + \mu_{i+1/2,j+1/2}\frac{\Delta t}{\Delta h}\left(u_{i+1/2,j+1}^{k+1/2} - u_{i+1/2,j}^{k+1/2}\right). \tag{4.177}$$

To time-evolve the solution for one full Δt, we follow:

1. update velocities from the stress;
2. update the stress from the velocities.

For a homogeneous medium, the stability condition is

$$v_P \frac{\Delta t}{\Delta h} < \frac{1}{\sqrt{2}}, \tag{4.178}$$

where

$$v_P = \sqrt{\frac{\lambda + 2\mu}{\rho}} \tag{4.179}$$

is the P-wave velocity. The stability condition is independent of the S-wave velocity

$$v_S = \sqrt{\frac{\mu}{\rho}} \tag{4.180}$$

because information will propagate at the P wave speed.

To minimize the grid dispersion, the spatial sampling required at least 10 gridpoints per wavelength, which is defined by the v_P. For a 4^{th}-order approach, the sampling rate can be reduced to 5 gridpoints/wavelength (*Levander*, 1988).

Several other issues are also very important in wave propagation in practice:

1. If a boundary condition is not well implemented, the related reflected waves from the boundaries of the domain will affect the results strongly. Depending on the problem, different boundary conditions can be applied to the edges: free-surface conditions, absorbing boundaries (*Clayton and Engquist*, 1977), and the recently widely adopted Perfectly Matched Layer (PML) absorbing boundary (*Collino and Tsogka*, 2001).

2. The source excitation, which initializes the wave propagation, also has to be treated with care. In general, a source can be implemented by simply adding a prescribed source time function to the source mesh. For example, an explosion point source time function $S(t)$ can be added to the 2D elastic case as:

$$\tau_{xx\ or\ zz}(source\ grid) = \tau_{xx\ or\ zz}(FD\ solution\ at\ source\ grid) + S(t)$$

Exercise 2

- Program the 2D elastic wave propagation in staggered grid scheme as in Fig. 4.23 (`wave_elastic_staggered_2D.m`). Choose Δt and Δh and describe the wavefield (both vertical and horizontal components) for the model with uniform velocities. Identify the first P and SV arrivals on the recorded seismograms.

- Include a thin liquid layer (v_S = 0) in the model and explain the result. Note for a typical wave propagation problem, the input models are v_P, v_S, and density ρ, so the conversions to λ and μ are required in the program.

Chapter 5

Finite elements

5.1 Introduction to finite element methods

Reading:

- Textbooks
  - The recommended background reading for this, finite element part of the 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) (see also *Zhong et al.*, 2007).
  - For additional reference, you might want to consider the Matlab-based finite element course by *Kwon and Bang* (1996) and the comprehensive and high-level treatment by *Bathe* (2007) and
- *Hughes* (2000), chap. 1, secs. 1.1-1.15
- *Bathe* (2007), sec. 1.2

5.1.1 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 $u(\boldsymbol{x})$ satisfies the PDE:

$$F(u(\boldsymbol{x})) = s(\boldsymbol{x}) \tag{5.1}$$

where F is some differential operator and s a source term (Figure 5.1).

As boundary conditions, we can have

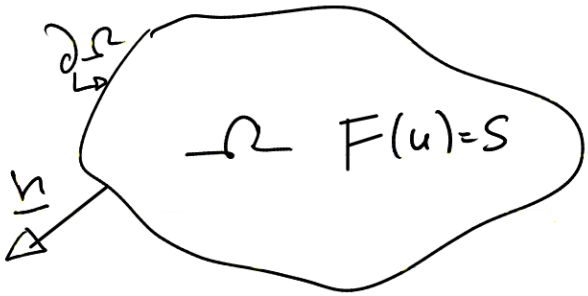


Figure 5.1: Illustration of the finite element domain and boundary.

Dirichlet (fixed value, "essential")

$$u|_{\partial\Omega} = g \tag{5.2}$$

type constraints where the value of u is given on $\partial\Omega$, and/or

Neumann (flux, "natural")

$$n_i \frac{\partial u}{\partial x_i} = \boldsymbol{n} \cdot \nabla \boldsymbol{u} = h \tag{5.3}$$

conditions, where we specify the derivatives at the boundary.

(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).

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, numerical solution for u is to be obtained using simplified polynomials, so called "shape 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)	**Finite Elements** (FE)
approximates the PDE	approximates the solution of the PDE
mainly restricted to simple, rectangular domains	complex geometries fairly easily implemented
regional, or adaptive mesh refinement hard to implement	regional mesh refinement easy, adaptive refinement fairly straightforward (problems with "smoothing")
simple implementation special case of FE	involved first implementation
requires programming from scratch if solving new equations	for well-written existing code, only minor changes are needed to solve different equations

5.1.2 A one – dimensional example

Consider

$$\frac{\partial^2 u}{\partial x^2} + s = 0 \tag{5.4}$$

on the domain $[0; 1]$ where s is given. Eq. (5.4) could be a 1-D steady-state heat equation, for example, where

$$\frac{\partial^2 T}{\partial x^2} - \frac{H(x)}{\kappa} = 0. \tag{5.5}$$

Mathematically, we require s to be smooth for a solution for u to exist. Additionally, we will require

$$u(0) = g \quad \textit{(Dirichlet)} \tag{5.6}$$

and

$$\frac{\partial u(1)}{\partial x} = h \quad \textit{(Neumann)} \tag{5.7}$$

boundary conditions (BCs), 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. (5.4) and (5.6) can, of course, be solved analytically by integration.

For example, if $s = x$ such that

$$\frac{\partial^2 u}{\partial x^2} + x = 0 \tag{5.8}$$

then by integration

$$\frac{\partial u}{\partial x} + \frac{1}{2}x^2 + c_1 = 0 \tag{5.9}$$

and integrating again

$$u + \frac{1}{6}x^3 + c_1 x + c_2 = 0 \tag{5.10}$$

$$u(x) = -\frac{1}{6}x^3 - c_1 x - c_2 \tag{5.11}$$

and use BCs (5.6) & (5.7)

$$\Rightarrow u(0) = -c_2 = g; \qquad \frac{\partial u(1)}{\partial x} = -\frac{1}{2} - c_1 = h \tag{5.12}$$

$$\Rightarrow u(x) = -\frac{1}{6}x^3 + \left(h + \frac{1}{2}\right)x + g \tag{5.13}$$

The analytical solution (5.13) will be used in the numerical problem set to test the approximate, numerical 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

$$u(0) = g \tag{5.14}$$

and that the trial solutions are square integrable

$$\int_0^1 \left(\frac{\partial u}{\partial x}\right)^2 dx < \infty; \tag{5.15}$$

(b) the weighting functions, or variations, w, satisfy

$$w(0) = 0, \tag{5.16}$$

the homogeneous counterpart of eq. (5.14).

We can then write (sloppily):

$$\frac{\partial^2 u}{\partial x^2} + s = 0 \tag{5.17}$$

multiply by $-w$ and integrate

$$-\int w \frac{\partial^2 u}{\partial x^2} - \int w\, s = 0 \tag{5.18}$$

from the integration by parts rule

$$\int a'b = ab \mid - \int b'a \tag{5.19}$$

$$\Rightarrow \int_0^1 \frac{\partial w}{\partial x}\frac{\partial u}{\partial x} - \left[\frac{\partial u}{\partial x} w\right]_0^1 - \int w\, s = 0. \tag{5.20}$$

With $\frac{\partial u(1)}{\partial x} = h$ and $w(0) = 0$, this can be written as

$$\boxed{\int_0^1 \frac{\partial w}{\partial x}\frac{\partial u}{\partial x} = h\, w(1) + \int_0^1 w\, s\, dx} \tag{5.21}$$

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. (5.21) 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

$$a(w,u) = \int_0^1 \frac{\partial w}{\partial x}\frac{\partial u}{\partial x}\, dx \quad \text{and} \quad (w,s) = \int_0^1 w\, s\, dx \tag{5.22}$$

Then, we can write eq. (5.21) as

$$\boxed{a(w,u) = (w,s) + h\, w(1)} \tag{5.23}$$

$a(.,.)$ and $(.,.)$ are *symmetric*

$$(w,s) = (s,w) \tag{5.24}$$

and *bilinear*

$$(c_1 u + c_2 v, w) = c_1(u,w) + c_2(v,w) \tag{5.25}$$

forms.

5.1.3 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

$$\tilde{u}(0) = g \quad \text{and} \quad \tilde{w}(0) = 0, \tag{5.26}$$

then we can construct a solution with $\tilde{v} \in \mathcal{W}$

$$\tilde{u} = \tilde{v} + \tilde{g} \tag{5.27}$$

where $\tilde{g}$ is a given function such that $\tilde{g}(0) = g$, which satisfies the BCs because

$$\tilde{u}(0) = \tilde{v}(0) + \tilde{g}(0) = 0 + g \tag{5.28}$$

as $\tilde{v}(0) = 0$, since $\tilde{v} \in \mathcal{W}$. If we substitute eq. (5.27) into eq. (5.23), we get

$$a(\tilde{w}, \tilde{v}) = (\tilde{w}, s) + \tilde{w}(1)h \; - a(\tilde{w}, \tilde{g}) \tag{5.29}$$

where we solve for the LHS and the RHS is determined by BCs.

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 $\tilde{w}$* .

5.1.4 Shape functions and discretization

Let's assume that there are n nodes such that we can write the weighting functions as

$$\tilde{w}(x) = \sum_{A=1}^{n} c_A \, N_A(x) \tag{5.30}$$

where the $N_A(x)$ are called *shape*, basis, or interpolation functions. We require

$$N_A(0) = 0 \qquad \forall A \tag{5.31}$$

such that $\tilde{w}(0) = 0$ can be fulfilled. If we also introduce a shape function

$$\hat{N}_1(0) = 1 \qquad (\text{with } \hat{N}_1 \notin \mathcal{W}) \tag{5.32}$$

then

$$\tilde{g} = g \, \hat{N}_1 \qquad \text{so that } \tilde{g}(0) = g. \tag{5.33}$$

We can then write

$$\begin{aligned} \tilde{u} &= \tilde{v} + \tilde{g} \\ &= \sum_{A=2}^{n} d_A \, N_A + g \, \hat{N}_1 \end{aligned} \tag{5.34}$$

such that $\tilde{u}(0) = g$. If we substitute eqs. (5.30) and (5.34) into eq. (5.29), then

$$a\left(\sum_A c_A\, N_A, \sum_B d_b\, N_B\right) = \left(\sum_A c_A\, N_A, s\right) + \left[\sum_A c_A\, N_A(1)\right]\, h - a\left(\sum_A c_A\, N_A, g\,\hat{N}_1\right). \tag{5.35}$$

Because of bilinearity, we can write $\sum_A c_A\, G_A = 0$ with

$$G_A = \sum_B a(N_A, N_B)\, d_B - (N_A, s) - N_A(1)\, h + a(N_A, \hat{N}_1)\, g. \tag{5.36}$$

The Galerkin equation (5.29) is supposed to hold for all w, therefore all c_A, which means that $a_A = 0$. So, for all A

$$\boxed{\sum_B a(N_A, N_B)\, d_B = (N_A, s) + N_A(1)\, h - a\,(N_A, \hat{N}_1)\, g.} \tag{5.37}$$

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. (5.37) becomes

$$\boxed{K\,\boldsymbol{d} \;=\; \boldsymbol{F}} \tag{5.38}$$

where K is the *stiffness matrix* , $\boldsymbol{d}$ is the *displacement vector* , and $\boldsymbol{F}$ is the *force, or load, vector*.

Once the $K\boldsymbol{d} = \boldsymbol{F}$ system is assembled, one may solve for $\boldsymbol{d}$ and then obtain the spatial solution from

$$\tilde{u}(x) = \sum_{A=2}^{n} d_A\, N_A(x) + g\,\hat{N}_1(x) \tag{5.39}$$

$$\text{or} \quad \tilde{u}(x) = \sum_{A}^{n} d_A\, N_A(x) \quad \text{with } d_1 = g. \tag{5.40}$$

Note that K is symmetric,

$$K = K_{AB} \;=\; a(N_A, N_B) \tag{5.41}$$

$$\;=\; a(N_B, N_A) = K_{BA} = K^T, \tag{5.42}$$

which facilitates computations, because special numerical solvers can be used depending on the matrix properties of K. Also, see sec. 5.4 for the solution of large, "sparse", linear systems of equations such as eq. (5.38).

5.2 A 1-D FE example implementation

We now provide a numerical implementation of the previously discussed 1-D FE example. We subdivide the $[0;1]$ interval into n sub-intervals (*"elements"*) delimited by $n+1$ *nodes* or nodal points such that $x_1 = 0$ and $x_{n+1} = 1$. The sub-intervals are denoted by

$$[x_A; x_{A+1}] \text{ with } h_A = x_{A+1} - x_A \tag{5.43}$$

where the element size 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

$$N_A(x) = \begin{cases} \frac{1}{h_{A-1}}(x - x_{A-1}) & \text{for} \quad x_{A-1} \leq x \leq x_A, \\ \frac{1}{h_A}(x_{A+1} - x) & \text{for} \quad x_A \leq x \leq x_{A+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.44}$$

For the boundaries, we use special shape functions

$$\hat{N}_1(x) = \frac{1}{h_1}(x_2 - x) \quad \text{for} \;\; x_1 \leq x \leq x_{n+1} \quad \text{and} \tag{5.45}$$

$$\hat{N}_{n+1}(x) = \frac{1}{h_n}(x - x_n) \quad \text{for} \;\; x_n \leq x \leq x_{n+1}. \tag{5.46}$$

An illustration of interior and boundary shape functions is shown in Figure 5.2; note that $N_A = 1$ at $x = x_A$ and zero for other nodes.

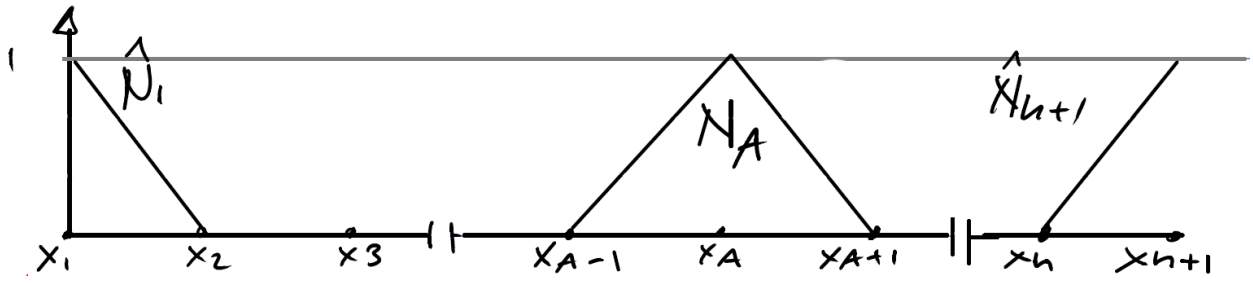


Figure 5.2: Example of 1-D shape functions

With this choice, the shape functions are zero outside the vicinity of A. They have *local support*, which means that K is a sparse matrix because the

$$a(N_B, N_A) = \int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial N_B}{\partial x} \, dx \tag{5.47}$$

integral is zero for $B > A + 1$. The K matrix is *banded*, banded matrix 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

$$\boldsymbol{c}^T K\, \boldsymbol{c} \;\geq\; 0 \tag{5.48}$$

for all $\boldsymbol{c}$ such that $\boldsymbol{c}^T\; K\; \boldsymbol{c} \;=\; 0 \;\;\Rightarrow\;\; \boldsymbol{c} \;=\; 0$. These properties allow efficient solution of $K\; \boldsymbol{d} = \boldsymbol{F}$ (sec. 5.4).

5.2.1 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 5.3.

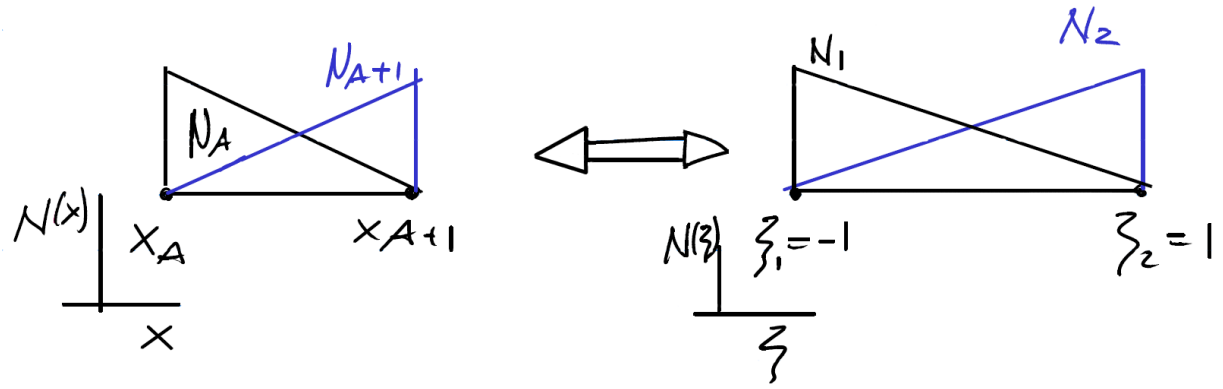


Figure 5.3: Example of 1-D shape functions in global (left, $N(X)$) and element-local (right, $N(\xi)$. with $\xi \in [-1;1]$) coordinates.

The mappings are as follows:

$$x(\xi) = \frac{1}{2}\,(h_A\,\xi + x_A + x_{A+1}) \quad \Leftrightarrow \quad \xi(x) = \frac{1}{h_A}\,(2x - x_A - x_{A+1}) \tag{5.49}$$

$$u(x) = \sum d_A\, N_A(x) \quad \Leftrightarrow \quad u(\xi) = N_1(\xi)\, d_1 + N_2(\xi)\, d_2. \tag{5.50}$$

We can express the global shape function N_A of eq. (5.44) in a local coordinate system as

$$N_a(\xi) = \frac{1}{2}\,(1 + \xi_a \xi) \quad \text{for } a = 1, 2 \tag{5.51}$$

$$(i.e.\; N_1(\xi) = \frac{1}{2}\,(1 - \xi); \quad N_2(\xi) = \frac{1}{2}\,(1 + \xi) \quad \text{with } \xi \in [-1;1]). \tag{5.52}$$

Likewise, we can express the global coordinate within the element as

$$x^e(\xi) = \sum_{a=1}^{2} N_a(\xi)\; x_a^e \tag{5.53}$$

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

$$\frac{\partial N_a}{\partial \xi} = \frac{\xi_a}{2} = \frac{(-1)^a}{2}, \tag{5.54}$$

$$\frac{\partial x}{\partial \xi} = \frac{h^e}{2}, \tag{5.55}$$

and

$$\frac{\partial \xi}{\partial x} = \left(\frac{\partial x^e}{\partial \xi}\right)^{-1} = \frac{2}{h^e}. \tag{5.56}$$

Note 1: For higher dimensional problems, the term $\left(\frac{\partial x^e}{\partial \xi}\right)^{-1}$ will be a matrix inverse.

Note 2: The choice of shape function is determined by the type of element. E.g. with a two node element, shape functions can only be linear. If the solution to be approximated is u, then u will vary linearly over the element, and derivatives, $\partial_x u$, will be constant.

5.2.2 Matrix assembly

With n elements, we therefore have *globally*

$$K = [K_{AB}] \quad \text{an } n \times n \text{ matrix and} \quad \boldsymbol{F} = \{F_A\} \quad \text{an } n \times 1 \text{ vector} \tag{5.57}$$

where (from last section)

$$K_{AB} = a(N_A, N_B) = \int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial N_B}{\partial x} \, \mathrm{d}x \tag{5.58}$$

$$F_A = (N_A, s) + h\,\delta_{A,n+1} - a(N_A, \hat{N}_1)\, g \tag{5.59}$$

$$= \int_0^1 N_A\, s \, \mathrm{d}x + \delta_{A,n+1}\, h - g \int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial \hat{N}_1}{\partial x} \, \mathrm{d}x \tag{5.60}$$

where $N_A(x_{n+1}) = \delta_{A,n+1}$ is assumed, and δ_{ij} is the Kronecker δ, $\delta_{ii} = 1$ and $\delta_{ij} = 0$ for $i \neq j$.

The integrals over the problem domain $[0; 1]$ can be written as summations over elements, therefore

$$K = \sum_{e=1}^{n} K^e \quad \text{with} \quad K^e = [K_{AB}^e] \tag{5.61}$$

$$\tag{5.62}$$

$$\boldsymbol{F} = \sum_e \boldsymbol{F}^e \quad \text{with} \quad \boldsymbol{F}^e = \{F_A^e\} \tag{5.63}$$

$$K^e_{AB} = \int d\Omega^e \frac{\partial N_A}{\partial x} \frac{\partial N_B}{\partial x} \, \mathrm{d}x \tag{5.64}$$

$$F^e_A = \int_{\Omega^e} N_A \, s \, \mathrm{d}x + h \, \delta_{e,n} \delta_{A,n+1} - g \int_{\Omega^e} \frac{\partial N_A}{\partial x} \frac{\partial \hat{N}_1}{\partial x} \, \mathrm{d}x \tag{5.65}$$

where the element domain $\Omega^e = [x^e_1; x^e_2]$.

Since the N_A only have local support $K^e_{AB} = 0$ if $A \neq \{e \text{ or } e+1\}$ or $B \neq \{e \text{ or } e+1\}$, and $F^e_A = 0$ if $A \neq \{e \text{ or } e+1\}$, and we can obtain the global stiffness matrix and force vector by summing up elemental contributions K^e and $\boldsymbol{f}^e$

$$K^e = [K_{ab}] \text{ a } 2 \times 2 \text{ matrix}, \quad \boldsymbol{f}^e = \{f_a\} \text{ a } 2 \times 1 \text{ vector} \tag{5.66}$$

(2 is the number of nodes per element!)

$$K_{ab} = a(N_a, N_b)^e = \int_{\Omega^e} \frac{\partial N_a}{\partial x} \frac{\partial N_b}{\partial x} \, \mathrm{d}x \tag{5.67}$$

$$\tag{5.68}$$

$$f_a = \int_{\Omega^e} N_a \, s \, \mathrm{d}x + \begin{cases} K^e_{a,1} g & \text{for} \quad e = 1, \\ 0 & \text{for} \quad e = 2, \ldots, n-1, \\ \delta_{a,n+1} h & \text{for} \quad e = h. \end{cases} \tag{5.69}$$

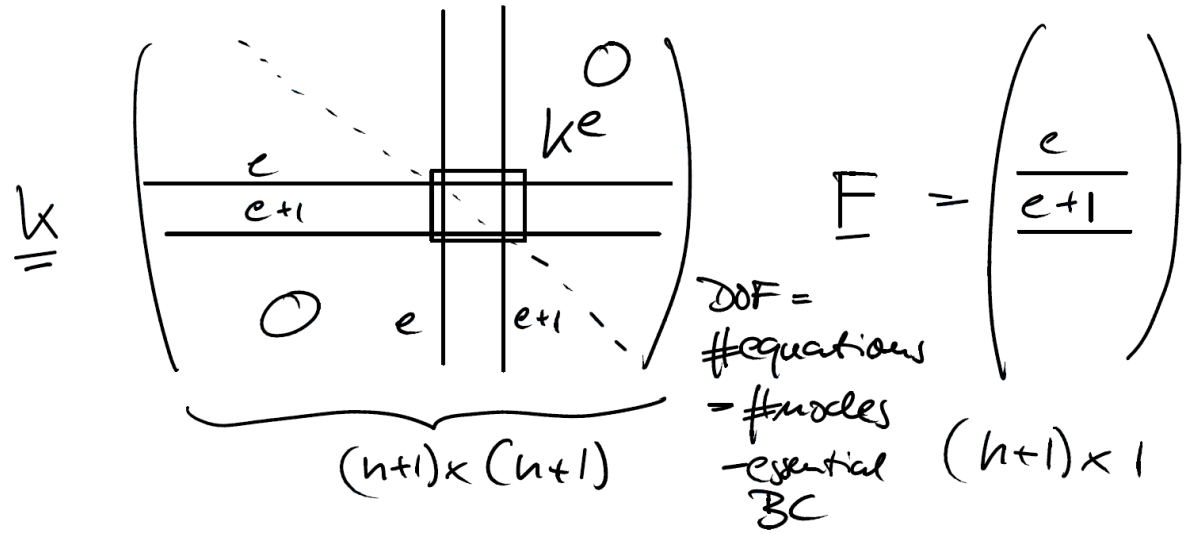


Figure 5.4: Cartoon of stiffness matrix assembly process

The assembly proceeds as symbolized in Figure 5.4, and placing the 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.

5.2.3 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

$$\int_{x_1}^{x_2} \mathrm{d}x\, f(x) = \int_{\xi(x_1)}^{\xi(x_2)} \mathrm{d}\xi\, [\frac{\partial x(\xi)}{\partial \xi}]\, f(x(\xi)) = \int_{\xi_1}^{\xi_2} \mathrm{d}\xi\, \frac{\partial x}{\partial \xi}\, f(\xi). \tag{5.70}$$

Therefore, we can compute

$$\begin{aligned} K^e_{ab} &= \int_{\Omega^e} \frac{\partial N_a(x)}{\partial x}\, \frac{\partial N_b(x)}{\partial x}\, \mathrm{d}x \\ &= \int_{-1}^{1} \frac{\partial N_a(x(\xi))}{\partial x}\, \frac{\partial N_b(x(\xi))}{\partial x}\, \frac{\partial x(\xi)}{\partial \xi}\, \mathrm{d}\xi \end{aligned} \tag{5.71}$$

using a change of variables. Then, using the the chain rule,

$$\frac{\partial f(x(\xi))}{\partial \xi} = \frac{\partial f(x(\xi))}{\partial x}\, \frac{\partial x(\xi)}{\partial \xi}, \tag{5.72}$$

we can get

$$\frac{\partial N_a(x(\xi))}{\partial \xi} = \frac{\partial N_a(x(\xi))}{\partial x}\, \frac{\partial x(\xi)}{\partial \xi} \quad \Rightarrow \quad \frac{\partial N_a(x)}{\partial x} = \left(\frac{\partial x}{\partial \xi}\right)^{-1} \frac{\partial N_a(\xi)}{\partial \xi}. \tag{5.73}$$

Then, plugging the result back into eq. (5.71)

$$K^e_{ab} = \int_{-1}^{1} \left(\frac{\partial x}{\partial \xi}\right)^{-2} \frac{\partial N_a(\xi)}{\partial \xi}\, \frac{\partial N_b(\xi)}{\partial \xi}\, \frac{\partial x(\xi)}{\partial \xi}\, \mathrm{d}\xi \tag{5.74}$$

$$= \int_{-1}^{1} \left(\frac{\partial x}{\partial \xi}\right)^{-1} \frac{\partial N_a(\xi)}{\partial \xi}\, \frac{\partial N_b(\xi)}{\partial \xi}\, \mathrm{d}\xi \tag{5.75}$$

$$= \frac{1}{h^e}(-1)^{a+b} \quad \text{(eq. 5.54)} \tag{5.76}$$

$$\tag{5.77}$$

$$\Rightarrow \; K^e = \frac{1}{h^e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \tag{5.78}$$

Since $\frac{\partial N_a}{\partial \xi}$ is independent of element data, the computation only has to be performed once, independent of the actual nodal values for the solution. The derivatives $\frac{\partial x}{\partial \xi}$ and $\frac{\partial \xi}{\partial x}$ do, however, depend on the element shape and need to be computed for each geometric (mesh) configuration.

For the source term, we use the approximation

$$\tilde{s} = \sum_{a=1}^{2} s_a\, N_a \tag{5.79}$$

i.e. the source term is assumed to vary linearly across the element with N_a. Then, we can write

$$\int_{\Omega^e} N_a(x)\ \tilde{s}(x)\,\mathrm{d}x = \int_{-1}^{1} N_a(x(\xi))\ \tilde{s}(x(\xi))\frac{\partial x(\xi)}{\partial \xi}\,\mathrm{d}\xi \tag{5.80}$$

$$= \frac{h^e}{2}\sum_{b=1}^{2}\int_{-1}^{1} N_a(\xi)\ N_b(\xi)\,\mathrm{d}\xi\ s_b. \tag{5.81}$$

Since $\int_{-1}^{1} N_a\ N_b = \frac{1}{3}(1+\delta_{ab})$,

$$s^e = \frac{h^e}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}s_1\\ s_2\end{bmatrix} \quad + \quad \text{boundary terms} \tag{5.82}$$

$$\tag{5.83}$$

$$= \frac{h^e}{6}\begin{bmatrix}2\,s_1 + s_2\\ s_1 + 2\,s_2\end{bmatrix} \quad + \quad \text{boundary terms} \tag{5.84}$$

5.3 Exercise: 1-D heat conduction with finite elements

Reading

This finite element example is based on *Hughes* (2000), sec. 1.1-1.15.

5.3.1 Implementation of the 1-D heat equation example

In the previous two sections, we considered the example PDE

$$\frac{\partial^2 u}{\partial x^2} + s = 0 \tag{5.85}$$

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, $\frac{\partial u(1)}{\partial x} = h$ on the right. Equation (5.85) may be considered a general version of the steady-state heat equation

$$\frac{\partial^2 T}{\partial x^2} + H = 0 \tag{5.86}$$

with sources $s = H$, for example.

See sec. 5.2, but in brief: If we have n elements between $n+1$ global nodes, the weak form of eq. (5.85) can be written for each global node A as

$$\sum_B a(N_A, N_B)d_B = (N_A, s) + N_A(1)h - a(N_A, \hat{N}_1)g. \tag{5.87}$$

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

$$K_{AB} = a(N_A, N_B) = \int_0^1 \frac{\partial N_A}{\partial x}\frac{\partial N_B}{\partial x} dx \tag{5.88}$$

$$F_A = (N_A, s) + N_A(1)h - a(N_A, \hat{N}_1)g \tag{5.89}$$

$$= \int_0^1 N_A s dx + \delta_{A,n+1}h - \left(\int_0^1 \frac{\partial N_A}{\partial x}\frac{\partial \hat{N}_1}{\partial x} dx\right) g, \tag{5.90}$$

where we have used the definitions of the bi-linear forms $a(\cdot,\cdot)$ and $(\cdot,\cdot)$ from before, and the Kronecker delta

$$\delta_{i,j} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{else} \end{cases} \tag{5.91}$$

is used for the flux boundary condition (also see *Hughes*, 2000, chap 1). Note that it is sometimes helpful to not think about nodes but about elements. The weak form of the equations are satisfied on average per element, and by constructing an appropriate mapping/numbering we can easily go from a single element to a 1D or 2D domain.

The approximate solution of u after discretization of the weak form is given by

$$\tilde{u}(x) = \sum_{A=2}^{n+1} d_A N_A(x) + \hat{N}_1 g = \sum d_A N_A(x), \tag{5.92}$$

where the latter summation implies choosing the boundary shape function and BC if needed. The vector $\boldsymbol{d} = \{d_A\}$ values have to be obtained by solution of the matrix equation

$$\mathsf{K}\boldsymbol{d} = \boldsymbol{F}, \tag{5.93}$$

with $\mathsf{K} = \{K_{AB}\}$ and $\boldsymbol{F} = \{F_A\}$.

We discussed previously how the integration over the domain can be broken down into summation over integrals over each element (see sec. 3.2). 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 $x(\xi)$,

$$x(\xi) = \sum_A N_A(\xi) x_A \quad \text{and} \quad u(\xi) = \sum N_A(\xi) d_A. \tag{5.94}$$

The global K matrix and the $\boldsymbol{F}$ vector are then assembled by looping over all elements $1 \leq e \leq n$ and adding each element's contribution for shared nodes. By change of integration variables and the chain rule, those elemental contributions follow as

$$\mathsf{k}^e = \frac{1}{\Delta x}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{5.95}$$

where Δx is the element size, $x_{A+1} - x_A$, and for the force term

$$f^e = \frac{\Delta x}{6} \begin{pmatrix} 2s_1 + s_2 \\ s_1 + 2s_2 \end{pmatrix} + \begin{cases} k_{a1} g & \text{for} \quad e = 1 \\ \delta_{a,n+1} & \text{for} \quad e = n \\ 0 & \text{else} \end{cases} \tag{5.96}$$

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 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.

5.3.2 Exercises

(a) Download `heat1dfe.m`, and all helper routines for this section. 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. (5.85) 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.

  (c) Reads the following section on solving linear systems of equations, if you have time, and work on the examples (optional).

5.4 Solution of large, sparse linear systems of equations

Parts of this exercise are based on *Zhong* (2008).

The finite element method and implicit finite difference methods quickly leads to very large, linear systems of equations

$$\mathsf{K}\boldsymbol{d} = \boldsymbol{F} \tag{5.97}$$

whose solution can be quite involved. Ideally, we would hand off the solution of eq. (5.97) 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, such as inverse theory, and some exposure to computational linear algebra is needed to fully 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.

5.4.1 Direct solvers

For the finite element method, we can always write our problem in the form of eq. (5.97), where K is a square, $n \times n$ matrix. A general strategy to solve eq. (5.97) is then *LU* decomposition

$$\mathsf{K} = \mathsf{L}\mathsf{U}, \tag{5.98}$$

where L and U are lower and upper triangular matrices, respectively, which only have zeros in the other part of the matrix. The solution of eq. (5.97) can then be obtained efficiently from

$$\mathsf{K}\boldsymbol{d} = \mathsf{L}\mathsf{U}\boldsymbol{d} = \boldsymbol{F} \tag{5.99}$$

by solving for $\boldsymbol{y} = \mathsf{L}^{-1}\boldsymbol{F}$ and then $\boldsymbol{d} = \mathsf{U}^{-1}\boldsymbol{y}$, because the inverse of U and L are computationally fast to obtain. *LU* is often how general matrix inversion is implemented on a computer.

For most FE problems, the 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 K, rather than the full size. Moreover, if K is symmetric and positive definite, as in our example above, we can use the Cholesky decomposition for which $\mathsf{U} = \mathsf{L}^T$ 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.

Notes:

- Symmetry means $\mathsf{K} = \mathsf{K}^T$, where K^T is the transpose, $K_{ij}^T = K_{ji}$.
- Positive definite means that $\boldsymbol{c}^T \mathsf{K} \boldsymbol{c} > 0$ for any non-zero $\boldsymbol{c}$. 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

$$\mathsf{A}\boldsymbol{x} = \boldsymbol{b} \tag{5.100}$$

in the sense that $|\mathsf{A}\boldsymbol{x} - \boldsymbol{b}| = \min$, *i.e.* deviations from the true solution are minimized, for a matrix A that may be under-determined, *i.e.* not simply invertible. It can be shown that the general least squares solution $\boldsymbol{x}_{LS}$ is given by

$$\boldsymbol{x}_{LS} = \left(\mathsf{A}^T \cdot \mathsf{A}\right)^{-1} \cdot \mathsf{A}^T \cdot \boldsymbol{d}, \tag{5.101}$$

where $\left(\mathsf{A}^T \cdot \mathsf{A}\right)^{-1}$ is the generalized inverse (which exists even if the inverse of A, A^{-1}, does not exist because A is singular). $\mathsf{A}^T \cdot \mathsf{A}$ is symmetric and positive definite, meaning that Cholesky is also the method of choice for direct approaches to find $\boldsymbol{x}_{LS}$.

5.4.2 Iterative solvers

Jacobi method

The simplest iterative solution of eq. (5.97) is given by the Jacobi method. If K is *LU* decomposed and we write the diagonal matrix (only non-zero along diagonal) of K as D, then an iterative solution for $\boldsymbol{d}$ starting from an initial guess $\boldsymbol{d}^1$ (*e.g.* $\mathbf{0}$) can be obtained from

$$\mathsf{D}\boldsymbol{d}^{i+1} = \boldsymbol{F} - (\mathsf{U} + \mathsf{L})\boldsymbol{d}^i, \tag{5.102}$$

where the iteration is over i and is stopped once $\boldsymbol{d}^{i+1}$ is not changing more than some tolerance from the previous solution estimate $\boldsymbol{d}^i$. On an element by element basis, this can be written as

$$d_j^{i+1} = \frac{1}{K_{jj}} \left(F_j - \sum_{l=1,\, l \neq j}^{n} K_{jl} d_l^i \right) \tag{5.103}$$

where the summation is over all l but for $l = j$. The Jacobi method following eq. (5.103) is implemented in `jacobi.m`. It serves mainly illustrative purposes but is guaranteed

to converge, albeit slowly (see below), if 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 method

An improvement over the Jacobi method is the Gauss-Seidel (GS) approach, where the iterative rule is

$$(\mathsf{D}+\mathsf{K})\boldsymbol{d}^{i+1} = \boldsymbol{F} - \mathsf{U}\boldsymbol{d}^{i}. \tag{5.104}$$

The main benefit is that $\boldsymbol{d}^{i+1}$ can be computed from $\boldsymbol{d}^{i}$ directly, without having to store a full previous solution, following

$$d_j^{i+1} = \frac{1}{K_{jj}}\left(F_j - \sum_{l<j} K_{jl} d_l^{i+1} - \sum_{l>j} K_{jl} d_l^{i}\right). \tag{5.105}$$

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 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.

$$d_j^{i+1} = (1-w)\, d_j^{i} + \frac{w}{K_{jj}}\left(F_j - \sum_{l<j} K_{jl} d_l^{i+1} - \sum_{l>j} K_{jl} d_l^{i}\right) \tag{5.106}$$

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 1

- 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 (GS) 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

$$\mathsf{M}^{-1}\mathsf{K}\boldsymbol{d} = \mathsf{M}^{-1}\boldsymbol{F}, \tag{5.107}$$

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 2

- 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 $\boldsymbol{d}^i$ (*e.g.* all zero, random numbers), record the convergence and comment on the solution.

Multigrid method

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 $\boldsymbol{d}$ ("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 $\boldsymbol{d}_0$ 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 coarse 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 $\mathcal{O}(N)$ 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

$$n_i = b \times 2^{i-1} + 1 \quad \text{for} \quad i = 1, 2, \ldots, L, \tag{5.108}$$

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 $\mathsf{K}\boldsymbol{d} = \boldsymbol{F}$, and the forcing is only needed to be specified at $\boldsymbol{F}_L$ (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 $\boldsymbol{d}_L$ from

$$\mathsf{K}_L \boldsymbol{d}_L = \boldsymbol{F}_L \tag{5.109}$$

to remove the short wavelength misfit starting from an initial guess $\boldsymbol{d}_L = \boldsymbol{0}$. The residual is then given by

$$\boldsymbol{R}_L = \boldsymbol{F}_L - \mathsf{K}\boldsymbol{d}_L. \tag{5.110}$$

We then *project*, or restrict, the residual to a coarser grid at $L - 1$ by a projection operator P

$$\boldsymbol{R}_{L-1} = P_{L \to L-1} \boldsymbol{R}_L. \tag{5.111}$$

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

$$\mathsf{K}_i \delta \boldsymbol{d}_i = \boldsymbol{R}_i \tag{5.112}$$

for $i = L - 1$ for another small number of iterations (initializing $\boldsymbol{d}_i$ again with $\boldsymbol{0}$), performing another "smoothing" step, reducing short wavelength fluctuations. Note that eq. (5.112) now operates on the residual and not the load vector $\boldsymbol{F}$ such that we are computing corrections of $\boldsymbol{d}$, $\delta\boldsymbol{d}$. We then repeat the smoothing and projection steps down to $i = 1$ where eq. (5.112) 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 $\delta \boldsymbol{d}_1$ 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

$$\delta \boldsymbol{d}_{i+1} = I_{i \to i+1} \delta \boldsymbol{d}_i. \tag{5.113}$$

I may be a linear interpolation, for example, which is easy to compute for the mesh structure eq. (5.108). This upward interpolated $\delta \boldsymbol{d}_{i+1}$ can then be smoothed by using it as a starting guess for a fixed number of GS iterations for

$$\mathsf{K}_{i+1} \delta \boldsymbol{d}_{i+1} = \boldsymbol{R}_{i+1} \tag{5.114}$$

with $\delta \boldsymbol{d}_{i+1}$. We can now correct

$$\begin{aligned} \delta \boldsymbol{d}_{i+1} &= \delta \boldsymbol{d}_{i+1} - \alpha \delta \boldsymbol{d}_{i+1} && (5.115) \\ \boldsymbol{R}_{i+1} &= \boldsymbol{R}_{i+1} - \alpha \delta \boldsymbol{R}_{i+} 1, && (5.116) \\ & && (5.117) \end{aligned}$$

with $\delta \boldsymbol{R}_{i+1} = -\mathsf{K}_{i+1} \delta \boldsymbol{d}_{i+1}$ and weighting $\alpha = (\delta \boldsymbol{R}_{i+1} \cdot \boldsymbol{R}_{i+1}) / |\delta \boldsymbol{R}_{i+1}|^2$. We continue by projecting $\delta \boldsymbol{d}_i$ in this fashion up to $i = L$, where we update $\boldsymbol{d}_L = \boldsymbol{d}_L + \delta \boldsymbol{d}_L$, which completes the upward leg of the V. The whole V cycle is then repeated until the desired tolerance for $\boldsymbol{d}_L$ is reached at which point $\boldsymbol{d}_L = \boldsymbol{d}$. 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 3

- 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.

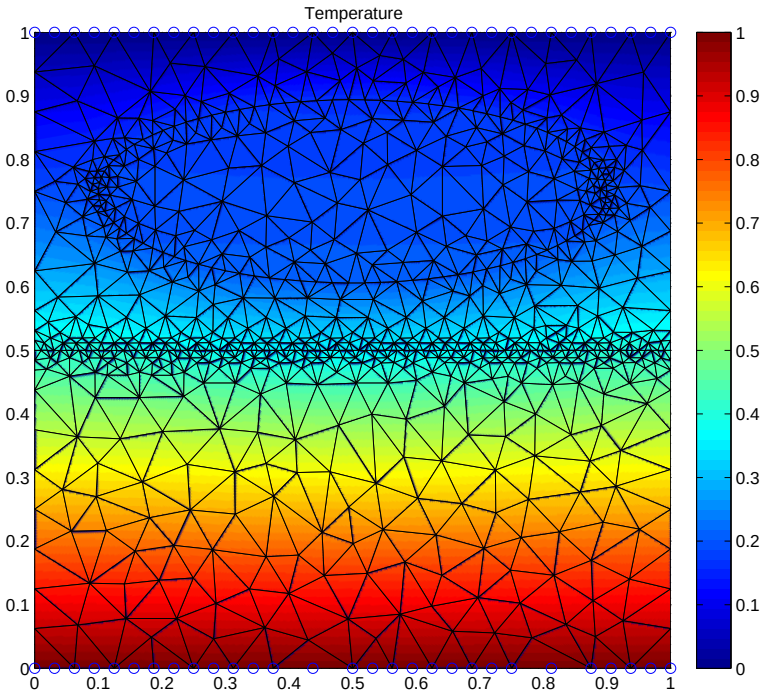


Figure 5.5: Coarse finite element mesh with solution for temperature, allowing for an elliptical inclusion and a boundary at mid-model height.

5.5 Two-Dimensional boundary value problems with FE

Reading: *Hughes* (2000) secs. 2.1 - 2.6, 3.1, 3.4, 3.8 - 3.9

We will now consider the solution of 2D boundary value problems using finite elements, which can be easily expanded to three dimensions. We write $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x \\ z \end{pmatrix}$ for a location vector x_i with $i = 1, 2$ and a normal vector $\boldsymbol{n}$ on the boundary Γ of the domain Ω. As an example problem we will now revisit the linear heat conduction problem.

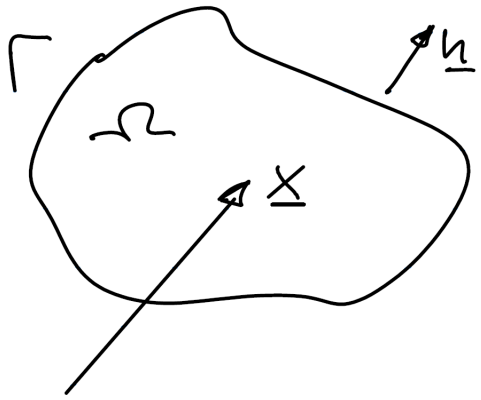


Figure 5.6: Visual representation of domain, vector, and normal

5.5.1 Linear heat conduction

If we allow for anisotropic diffusivity (which may apply to the oceanic plates), Fick's Law can be written as

$$q_i = -\kappa_{ij} \frac{\partial T}{\partial x_j} = -\sum_{j=1}^{2} \kappa_{ij} \frac{\partial T}{\partial x_j} \tag{5.118}$$

where repeated indices imply summation, $\boldsymbol{q}$ 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,

$$\boxed{\boldsymbol{q} = -\kappa \cdot \nabla \boldsymbol{T}} \qquad \kappa = [\kappa_{ij}] \tag{5.119}$$

and, usually, for the isotropic case

$$\kappa_{ij} = \kappa(\boldsymbol{x})\delta_{ij} \tag{5.120}$$

is assumed. In steady-state, the energy equation is

$$\boxed{\nabla \cdot \boldsymbol{q} = H} \tag{5.121}$$

$$\text{or} \quad \frac{\partial q_i}{\partial x_i} = \frac{\partial \kappa_{ij}}{\partial x_i} \frac{\partial T}{\partial x_j} = H \quad \text{for } \Omega \qquad \text{(Poisson Eq.)}$$

with BCs:

$$\begin{aligned} T &= g \quad \text{on } \Gamma_g \quad \text{and} \\ -q_i\, n_i &= h \quad \text{on } \Gamma_h \end{aligned} \tag{5.122}$$

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

$$\frac{\partial}{\partial x_i}\,\delta_{ij}\kappa\,\frac{\partial T}{\partial x_j} = \kappa\,\frac{\partial}{\partial x_i}\frac{\partial T}{\partial x_i} = H \tag{5.123}$$

$$\text{conductivity } \rightarrow \kappa\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2}\right) = H\,. \tag{5.124}$$

The weak form representation (first substitute the test function and integrate over the domain, second integrate by parts) of eq. (5.122) is given by

$$-\int_\Omega \mathrm{d}\Omega\,\frac{\partial w}{\partial x_i} q_i = \int_\Omega \mathrm{d}\Omega\; wH + \int_{\Gamma_h} \mathrm{d}\Gamma\; wh \tag{5.125}$$

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. (5.125) can then be expressed as

$$\boxed{a(w,T) = (w,H) + (w,h)_\Gamma} \tag{5.126}$$

with

$$\begin{aligned} a(w,T) &= \int_\Omega \mathrm{d}\Omega\,\frac{\partial w}{\partial x_i}\kappa_{ij}\,\frac{\partial T}{\partial x_j} \\ (w,H) &= \int_\Omega \mathrm{d}\Omega\; wH && \text{(area integral over } \Omega) \\ (w,h)_\Gamma &= \int_\Gamma \mathrm{d}\Gamma\; wh. && \text{(line integral over } \Gamma) \end{aligned}$$

It is convenient to use vector/matrix notation

$$\frac{\partial w}{\partial x_i}\kappa_{ij}\,\frac{\partial g}{\partial x_j}T \tag{5.127}$$

can then be written as

$$(\nabla \boldsymbol{w})^T \kappa\,\nabla T \qquad \text{with} \quad \nabla \boldsymbol{w} = \begin{pmatrix} \frac{\partial w}{\partial x_1} \\ \frac{\partial w}{\partial x_2} \end{pmatrix} \qquad \text{and} \quad \nabla \boldsymbol{T} = \begin{pmatrix} \frac{\partial T}{\partial x_1} \\ \frac{\partial T}{\partial x_2} \end{pmatrix} \tag{5.128}$$

such that

$$a(w,T) = \int_\Omega \mathrm{d}\Omega\;(\nabla \boldsymbol{w})^T \kappa\,\nabla \boldsymbol{T} \tag{5.129}$$

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

$$\tilde{T} = \tilde{v} + \tilde{g} \tag{5.130}$$

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

$$a(\tilde{w}, \tilde{v}) = (\tilde{w}, H) + (\tilde{w}, h)_{\Gamma_h} - a(\tilde{w}, \tilde{g}) \tag{5.131}$$

(compare to previous 1D case, eq. 5.29).
Let's introduce the shape functions N_A for a global node A out of n_{np} total number of nodes. With

$$\tilde{v}(\boldsymbol{x}) = \sum_{A \in \mathcal{I}} N_A(\boldsymbol{x})\, d_A \quad \text{and} \quad \tilde{g}(\boldsymbol{x}) = \sum_{A \in \mathcal{B}} \tilde{N}_A(\boldsymbol{x})\, g_A \tag{5.132}$$

where we have again distinguished between interior nodes and shape functionm $A \in \mathcal{I}$, and those on the Dirichlet boundary, with $A \in \mathcal{B}$. Arguing as for the 1-D case, the following assembly rules result

$$K\ \boldsymbol{d} = \boldsymbol{F} \tag{5.133}$$

$$K = \left[K_{PQ}\right] \qquad K_{PQ} = a(N_A, N_B), \tag{5.134}$$

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

$$ID(A) = \begin{cases} P & \text{for } A \in \mathcal{I} \\ 0 & \text{for } A \in \mathcal{B} \end{cases} \tag{5.135}$$

such that $P = ID(A)$ and $Q = ID(B)$. $\boldsymbol{d} = \{d_Q\}$ for the solution temperatures

$$\tilde{v}(\boldsymbol{x}) = \sum N_A\, d_A \tag{5.136}$$

and

$$\boldsymbol{F} = \{F_p\} \tag{5.137}$$

where

$$F_P = (N_A, H) + (N_A, h)_{\Gamma_h} - \sum_{B \in \mathcal{B}} a(N_A, N_B)\, q_B \tag{5.138}$$

and K is again symmetric and positive definite.

5.5.2 Matrix assembly

As before, we compute K and $\boldsymbol{F}$ based on summation over all n_{el} elements.

$$K = \sum_{e=1}^{n_{el}} K^e \qquad K^e = \{K^e_{PQ}\} \tag{5.139}$$

$$\boxed{K^e_{PQ} = a(N_A, N_B)^e = \int_{\Omega^e} (\nabla \boldsymbol{N_A})^T \kappa \, (\nabla \boldsymbol{N_B}) \, \mathrm{d}\Omega}. \tag{5.140}$$

The RHS in eq. (5.140) corresponds to integrating over each element's area.

$$\boldsymbol{F} = \sum_{e=1}^{n_{el}} \boldsymbol{F}^e \qquad \boldsymbol{F}^e = \{F^e_P\} \tag{5.141}$$

$$F^e_P = \int_{\Omega^e} \mathrm{d}\Omega \; N_A \, H + \int_{\Gamma^e_h} \mathrm{d}\Gamma \; N_A \, h - \sum_{B \in \mathcal{B}} a(N_A, N_B)^e \, q_B \tag{5.142}$$

where Γ^e_h 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 $1 \leq a, b \leq n_{en}$

$$K^e = \{K^e_{ab}\} \qquad K^e_{ab} = a(N_a, N_b)^e = \int_{\Omega^e} \mathrm{d}\Omega \; (\nabla \boldsymbol{N_a})^T \kappa \, (\nabla \boldsymbol{N_b}) \tag{5.143}$$

$$\boldsymbol{f}^e = [f^e_a] \tag{5.144}$$

$$f_a = \int_{\Omega^e} N_a \, f_a \, \mathrm{d}\Omega + \int_{\Gamma^e_h} N_a \, h \, \mathrm{d}\Gamma - \sum_{b=1}^{n_{en}} K^e_{ab} \, g^e_b \tag{5.145}$$

where $g^e_b = g(\boldsymbol{x}^e_b)$ for prescribed g and zero otherwise. It is convenient to write

$$\boxed{K^e = \int_{\Omega^e} \mathrm{d}\Omega \; B^T D \; B} \tag{5.146}$$

where D is $n_{sd} \times n_{sd}$ (rows $\times$ columns); $n_{sd} \triangleq$ number of spatial dimensions. In our case D is 2×2 and $D = \kappa$. B is $n_{sd} \times n_{en}$ such that $B = \{\boldsymbol{B}_1, \boldsymbol{B}_2, \ldots, \boldsymbol{B}_{n_{en}}\}$ and

$$B_a = \nabla \boldsymbol{N_a} \tag{5.147}$$

is $n_{sd} \times 1$.

The B and 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

$$\boldsymbol{d}^e = \{d^e_a\} = \begin{pmatrix} d^e_1 \\ d^e_2 \\ \vdots \\ d^e_{n_{en}} \end{pmatrix} \tag{5.148}$$

then

$$\boldsymbol{q}(\boldsymbol{x}) = -D(\boldsymbol{x})\; B(\boldsymbol{x})\; \boldsymbol{d}^e = -D(\boldsymbol{x}) \sum_{a=1}^{n_{en}} \boldsymbol{B}_a\, d_a^e \tag{5.149}$$

can be used to compute the heat flux within each element. We will revisit this for the elastic problem where B converts the nodal displacement solution into the strain.

5.5.3 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 $(1 \le a \le n_{en} = 4)$ spanned by the local coordinates

$$\xi_1 = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \qquad \xi_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{5.150}$$

$$\xi_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \xi_4 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \tag{5.151}$$

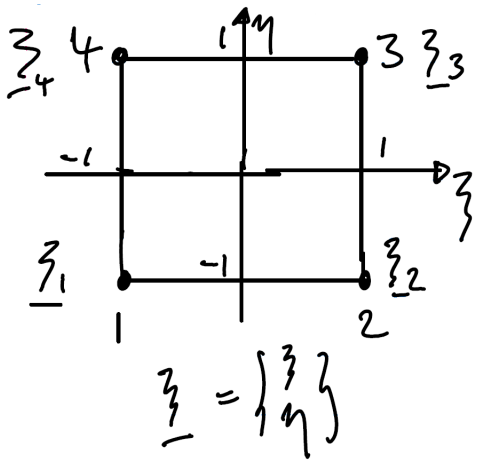


Figure 5.7: Quad element in element-local (ξ, η) coordinate space

to the total global domain where the element may be deformed

$$\boxed{\boldsymbol{x}(\boldsymbol{\xi}) = \sum_{a=1}^{n_{en}} N_a(\xi)\, \boldsymbol{x}_a^e}\,, \tag{5.152}$$

are the same shape functions that are used to represent the solutions

$$\boxed{\tilde{v}(\boldsymbol{\xi}) = \sum_{a=1}^{n_{en}} N_a(\boldsymbol{\xi})\, d_a^e}. \tag{5.153}$$

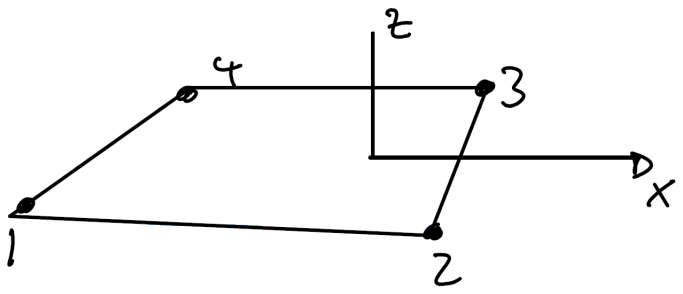


Figure 5.8: Quad element in global (x, z) coordinate space

If the mapping from the element local, $\boldsymbol{\xi}$, to real coordinate space, $\boldsymbol{x}$, is differentiable, the determinant j of the Jacobian J

$$j = \det J = \det \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{pmatrix} = \frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \eta} - \frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \eta} \tag{5.154}$$

is $j(\boldsymbol{\xi}) > 0$ for all $\boldsymbol{\xi}$ within the element. $j(\boldsymbol{\xi})$ 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 (eq. 5.70)

$$\int_{\Omega^e} f(x)\,\mathrm{d}\Omega = \int_{-1}^{1} f(x(\xi))\,\mathrm{d}\xi\,\frac{\partial x}{\partial \xi} = \int_{-1}^{1} f(x(\xi))\frac{\partial x}{\partial \xi}\,\mathrm{d}\xi \tag{5.155}$$

which generalizes to 2-D as

$$\int_{\Omega^e} f(\boldsymbol{x})\,\mathrm{d}\Omega = \int_{-1}^{1}\mathrm{d}\xi \int_{-1}^{1}\mathrm{d}\eta\; f(x(\xi,\eta), z(\xi,\eta))\, j(\xi,\eta). \tag{5.156}$$

The above equation is a result of the change of variables and can be used to evaluate the $a(.,.)$ type integrals.

5.5.4 Numerical integration

While the integral over simple shape functions (and for non-deformed quadrilaterals) may be easily evaluated analytically, it is most convenient to perform a numerical integration over the element area or volume Ω^e. Also, if the element is deformed, one must perform numerical integration.

In 1-D, the objective is to optimally approximate (*i.e.* replace the integral over the element

with a summation)

$$\int_{-1}^{1} \mathrm{d}\xi \; g(\xi) \approx \sum_{i=1}^{n_{int}} g(\tilde{\xi}_i) \, W_i \tag{5.157}$$

for a small number of integration points n_{int}. The W_i are the weights for the function values at the integration points ξ_i. For example, the

Trapezoidal rule corresponds to $n_{int} = 2;\ \tilde{\xi}_1 = -1;\ \tilde{\eta}_2 = 1;\ W_i = 1$ and is second order accurate.

$$\int_a^b f(x)\,\mathrm{d}x \approx (b-a)\frac{f(a)+f(b)}{2} \tag{5.158}$$

Simpson's rule corresponds to $n_{int} = 3;\ \tilde{\xi}_1 = -1;\ \tilde{\xi}_2 = 0;\ \tilde{\xi}_3 = 1;\ W_1 = \frac{1}{3};\ W_2 = \frac{4}{3};\ W_3 = \frac{1}{3}$ and is fourth order accurate (*i.e.* Simpson's rule integrates a cubic polynomial exactly).

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \tag{5.159}$$

Gaussian quadrature is the optimal (fewest integration points for maximum accuracy) strategy. For n_{int}, it is defined by

$$W_i = \frac{2}{(1-\tilde{\xi}_i^2)\left(\frac{\partial P_{n_{int}}}{\partial \tilde{\xi}}(\tilde{\xi}_i)^2\right)} \quad 1 \le i \le n_{int} \tag{5.160}$$

where $\tilde{\xi}_i$ is the i^{th} root of the Legendre polynomial

$$P_n(\xi) = \frac{1}{2^n\, n!}\frac{\partial^n}{\partial \xi^n}(\xi^2 - 1)^n. \tag{5.161}$$

This rule is $\mathcal{O}(2n_{int})$ accurate in 1-D, and weights and $\tilde{\xi}$ locations are tabulated (see below).

In 2-D, we can compute

$$\int_{-1}^{1} \mathrm{d}\xi \int_{-1}^{1} \mathrm{d}\eta \; g \approx \sum_{i=1}^{n_{int}} \sum_{j=1}^{n_{int}} g(\tilde{\xi}_i, \tilde{\eta}_j)\, W_i\, W_j. \tag{5.162}$$

Finally, we often need to convert the derivatives of the shape functions with respect to the global coordinates to local coordinates: $\boldsymbol{\xi}(\boldsymbol{x})$. By means of the chain rule we obtain

$$\frac{\partial N_a}{\partial x} = \frac{\partial N_a}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial N_a}{\partial \eta}\frac{\partial \eta}{\partial x} \tag{5.163}$$

$$\frac{\partial N_a}{\partial z} = \frac{\partial N_a}{\partial \xi}\frac{\partial \xi}{\partial z} + \frac{\partial N_a}{\partial \eta}\frac{\partial \eta}{\partial z}. \tag{5.164}$$

This can be written in matrix form as

$$\{\frac{\partial N_a}{\partial x}, \frac{\partial N_a}{\partial z}\} = \{\frac{\partial N_a}{\partial \xi}, \frac{\partial N_a}{\partial \eta}\} \begin{pmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial z} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial z} \end{pmatrix} \tag{5.165}$$

(multiply and add, column wise)

and $\frac{\partial N_a}{\partial \xi}$ as well as $\frac{\partial N_a}{\partial \eta}$ can be easily computed from the shape function definition. However, the $\frac{\partial \xi}{\partial x}$ type derivatives are not available explicitly. How can we compute them then? This involves one of the tricks of the FE method, namely by noting that the coordinates of the element are also approximated by shape functions, in the following manner: We do know the inverse relationships

$$x(\xi, \eta) = \sum_{a=1}^{n_{en}} N_a(\xi, \eta)\, x_a^e \tag{5.166}$$

$$z(\xi, \eta) = \sum_{a=1}^{n_{en}} N_a(\xi, \eta)\, z_a^e \tag{5.167}$$

from which we can compute the derivatives of the coordinates versus natural coordinate as

$$J = \partial_{\boldsymbol{\xi}} \boldsymbol{x} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{pmatrix} \tag{5.168}$$

$$\Big(e.g.\ \frac{\partial x}{\partial \xi} = \sum \frac{\partial N_a}{\partial \xi}\, x_a^e;\ \frac{\partial z}{\partial \eta} = \sum \frac{\partial N_a}{\partial \eta}\, z_a^e\Big).$$

It turns out that the Jacobian J is the inverse of eq. (5.165)

$$\begin{pmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial z} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial z} \end{pmatrix} = J^{-1} = \frac{1}{j} \begin{pmatrix} \frac{\partial z}{\partial \eta} & -\frac{\partial x}{\partial \eta} \\ -\frac{\partial z}{\partial \xi} & \frac{\partial x}{\partial \xi} \end{pmatrix} \tag{5.169}$$

with

$$j = \det\left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}\right) = \frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial z}{\partial \xi}. \tag{5.170}$$

Therefore

$$\boxed{\left\{\frac{\partial N_a}{\partial x}, \frac{\partial N_a}{\partial z}\right\} = \left\{\frac{\partial N_a}{\partial \xi}, \frac{\partial N_a}{\partial \eta}\right\} J^{-1}} \tag{5.171}$$

5.5.5 Simple elements, shape functions and Gaussian quadrature rules

1-D linear shape functions

$N_a(\xi) = \frac{1}{2}(1 + \xi_a \xi) \quad a = 1, 2$
with two nodes at $\xi_1 = -1; \quad \xi_2 = 1.$
$\frac{\partial}{\partial \xi} N_a(\xi) = \frac{\xi_a}{2} = \frac{(-1)^a}{2}$

Quadrature

n_{int}	$\tilde{\xi}_i$	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	"

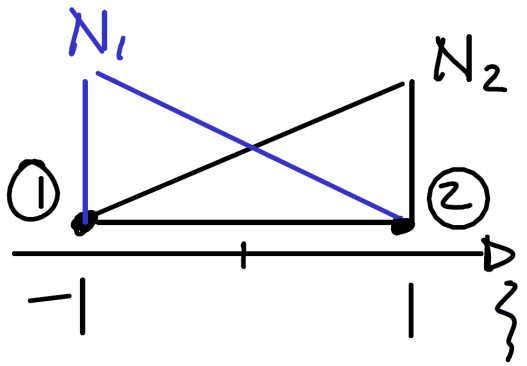


Figure 5.9: 1-D linear shape functions

Bilinear quadrilateral ("quad") element

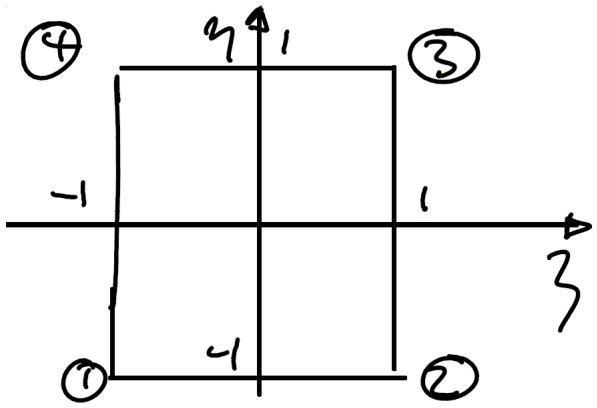


Figure 5.10: Quad element nodes in local coordinates

four nodes are located at

$$\boldsymbol{\xi}_1 = \{-1, -1\} \qquad \boldsymbol{\xi}_2 = \{1, -1\} \tag{5.172}$$
$$\boldsymbol{\xi}_3 = \{1, 1\} \qquad \boldsymbol{\xi}_4 = \{-1, 1\} \tag{5.173}$$
$$\boldsymbol{\xi}_a = \{\xi_a, \eta_a\} \qquad \text{etc. for } a = 1, 2, 3, 4 \tag{5.174}$$

$$N_a(\boldsymbol{\xi}) = N_a(\xi, \eta) = \frac{1}{4}(1 + \xi_a\,\xi)(1 + \eta_a\,\eta) \tag{5.175}$$

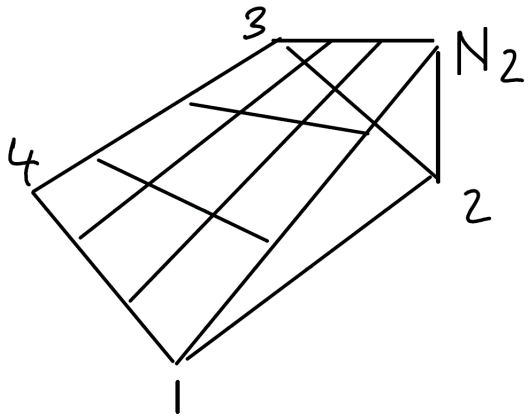


Figure 5.11: Quad element shape functions

Quadrature in 2-D

n_{int}	visual	$\tilde{\boldsymbol{\xi}}_i$	w_i	for integration
1	$[\times]$	$\{0, 0\}$	4	$\int_{-1}^{1} \mathrm{d}\xi \int_{-1}^{1} \mathrm{d}\eta$
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

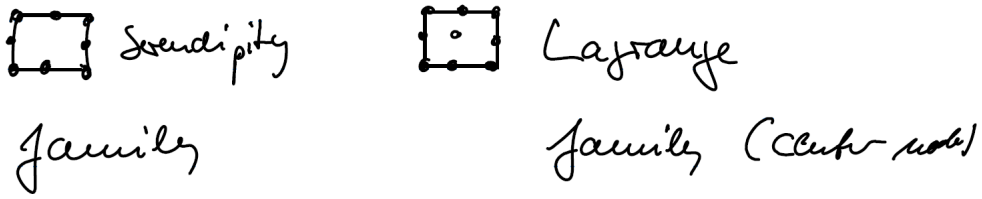


Figure 5.12: Other quad element families

Hughes (2000) p. 191 discusses the required level of Gaussian quadrature for adequate convergence for different element types.

Triangular elements

Linear triangle

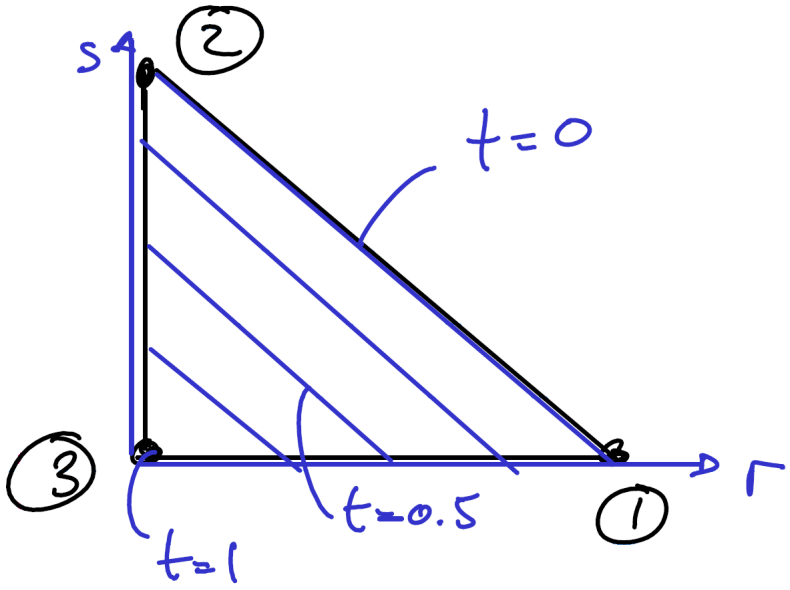


Figure 5.13: Linear triangle element nodes in local coordinates

$$t(r,s) = 1 - r - s \tag{5.176}$$

$$\boldsymbol{r} = \{r, s\} \tag{5.177}$$

$$\boldsymbol{r}_1 = \{1, 0\} \qquad N_1(r,s) = r \tag{5.178}$$

$$\boldsymbol{r}_2 = \{0, 1\} \qquad N_2(r,s) = s \tag{5.179}$$

$$\boldsymbol{r}_3 = \{0, 0\} \qquad N_3(r,s) = t = 1 - r - s \tag{5.180}$$

Quadratic (six node) triangle

$$N_1 = r(2r-1) \qquad N_4 = 4\,r\,s \tag{5.181}$$

$$N_2 = s(2s-1) \qquad N_5 = 4\,s\,t \tag{5.182}$$

$$N_3 = t(2t-1) \qquad N_6 = 4\,r\,t \tag{5.183}$$

See *Hughes* (2000) Appendix 3.1 for Gauss quadrature formula.

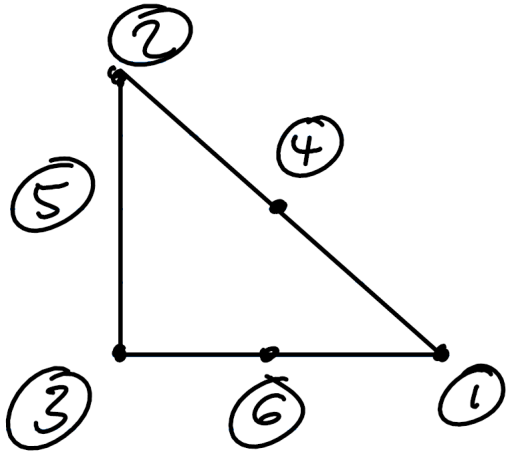


Figure 5.14: Quadratic triangle element

5.5.6 Inverse transformation of parametric elements

It is straighforward to convert from element-local, $\boldsymbol{\xi}$, to global coordinates, $\boldsymbol{x}$, using the shape functions within each element and the element-local coordinates $\boldsymbol{x}_a$,

$$\boldsymbol{x} = \sum N_a(\boldsymbol{\xi})\boldsymbol{x}_a, \tag{5.184}$$

where $\boldsymbol{\xi} = \{\xi, \eta\}$ or $\boldsymbol{\xi} = \{r, s\}$. However, the inverse transformation, going from an arbitrary global $\boldsymbol{x}$ to the element-local $\boldsymbol{\xi}$, is a bit more tricky. This conversion involves two steps, finding the element in which $\boldsymbol{x}$ lies, and then projecting into the local coordinate system.

Finding an element can be very fast, for example when the mesh consists of regular quads, in which case this step only involves finding which row i and column j the point $\boldsymbol{x} = \{x, z\}$ lies in by means of division of the grid spacings Δx and Δz, $i = \texttt{fix}(x/(\Delta x)) + 1$ and $j = \texttt{fix}(z/(\Delta z)) + 1$, where the element number might be $n = i + (j-1)n_x$. Here, n_x is the number of elements in each row, and `fix` the Matlab real to integer conversion that does not round.

For irregular meshes, finding the element containing $\boldsymbol{x}$ can be tricky and care should be taken to use optimally fast, geometric methods to find if the point lies within the polygons defined, *e.g.* by an irregular triangular mesh. Matlab provides the function `pointLocation` for native Delaunay meshes and `inpolygon` for general meshes, for example.

Triangular elements

Once the element is found, the type of projection depends on the type of element. For triangular, 2D elements, the operation is fairly straightforward. If x_i and z_i are the nodal

coordinates ($i = 1, 2, 3$) and $\boldsymbol{x} = \{x, z\}$, then

$$r = \frac{x_2 z_3 - x_3 z_2 - x_2 z + x z_2 + x_3 z - x z_3}{x_1 z_2 - x_2 z_1 - x_1 z_3 + x_3 z_1 + x_2 z_3 - x_3 z_2} \tag{5.185}$$

$$s = -\frac{x_1 z_3 - x_3 z_1 - x_1 z + x z_1 + x_3 z - x z_3}{x_1 z_2 - x_2 z_1 - x_1 z_3 + x_3 z_1 + x_2 z_3 - x_3 z_2} \tag{5.186}$$

for the shape functions defined in eqs. (5.178) and (5.179).

Quadratic elements

If the mesh is regular, one can easily convert $\boldsymbol{x}$ to $\boldsymbol{\xi} = \{\xi, \eta\}$ given the distance from the center of the element in question. If the element is irregular, different cases have to be distinguished and those and the corresponding equations are discussed in **?**.

5.6 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. 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.

Over the next sections, we will discuss all of the issues described in *Dabrowski et al.* (2008). This paper will be a good additional reference, and the original MILAMIN Matlab codes can be downloaded from `http://milamin.org/` (the latter will not be of help with the exercises).

5.6.1 Implementation of 2-D heat equation

We spent the last three sections discussing the fundamentals of finite element analysis building up to the solution of the 2-D, stationary heat equation, which is given by

$$\frac{\partial}{\partial x}\left(\kappa \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial z}\left(\kappa \frac{\partial T}{dz}\right) = H, \tag{5.187}$$

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 $\boldsymbol{T} = \{T_A\}$ for all $NNOD$ nodes of the mesh, which can be combined to

$$\tilde{T}(x,z) = \sum_{A=1}^{NNOD} N_A(x,z) T_A. \tag{5.188}$$

Following, *e.g.*, *Hughes* (2000), we use the Galerkin approach for which the resulting stiffness matrix components, on an element level, is

$$K^e_{ab} = \int_{\Omega^e} \kappa^e \left(\frac{\partial N_a}{\partial x}\frac{\partial N_b}{\partial x} + \frac{\partial N_a}{\partial z}\frac{\partial N_b}{\partial z} \right) d\Omega. \tag{5.189}$$

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 $\boldsymbol{x} = \{x,z\}$ in a node-local coordinate system $\boldsymbol{\xi} = \{\xi,\eta\}$ and use Gaussian quadrature with $NINT$ points and weights W_i for integration, we need to evaluate terms of the kind

$$K^e_{ab} = \int_{-1}^{1} d\xi \int_{-1}^{1} d\eta \kappa \left(\frac{\partial N_a}{\partial \xi}\frac{\partial N_b}{\partial \xi} + \frac{\partial N_a}{\partial \eta}\frac{\partial N_b}{\partial \eta} \right) J^{-1} j \tag{5.190}$$

$$K^e_{ab} = \sum_{i}^{NINT} W_i \kappa_i \left(\frac{\partial N_a}{\partial \xi}\frac{\partial N_b}{\partial \xi} + \frac{\partial N_a}{\partial \eta}\frac{\partial N_b}{\partial \eta} \right) J^{-1} j \tag{5.191}$$

where J^{-1} is the inverse and $j = det(J) = |J|$ the determinant of the Jacobian matrix

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{pmatrix}, \tag{5.192}$$

respectively.

The load vector $\boldsymbol{F}$ has to be assembled on an element basis as

$$\boldsymbol{F}^e_a = \int_{\Omega^e} N_a H d\Omega - K^e_{ab} \hat{T}_b, \tag{5.193}$$

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 K and $\boldsymbol{F}$ are assembled by looping through all elements and adding up the K^e and $\boldsymbol{F}^e$ contributions, while eliminating those rows that belong to nodes where essential boundary conditions ($\hat{T}$) are supplied.

The solution is then obtained from solving

$$K\boldsymbol{T} = \boldsymbol{F}. \tag{5.194}$$

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 by *Shewchuk* (2002). That work is freely available as `C` source code and a flexible, production quality "Delaunay" mesh generator. A Delaunay mesh is such that all nodes are connected by elements in a way that any circle which is drawn through the three nodes of an element has no other nodes within its circumference.

Aside: 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, and $j = det(J)$ remains positive.

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 executing your Matlab commands in.

Note: For this exercise and those below, please first inspect graphs on the screen while playing with the code, and then only print out a few geometries.

(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 print out 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 print out a mesh plot 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 sec. 5.6.1. Please make sure you take this step seriously.

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_test.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.

(e) Compute the temperature as well as the geothermal gradient at a specific location. This exercise is so that you gain experience using shape functions and derivatives of shape functions and requires you to identify the N and ∂N equivalents.

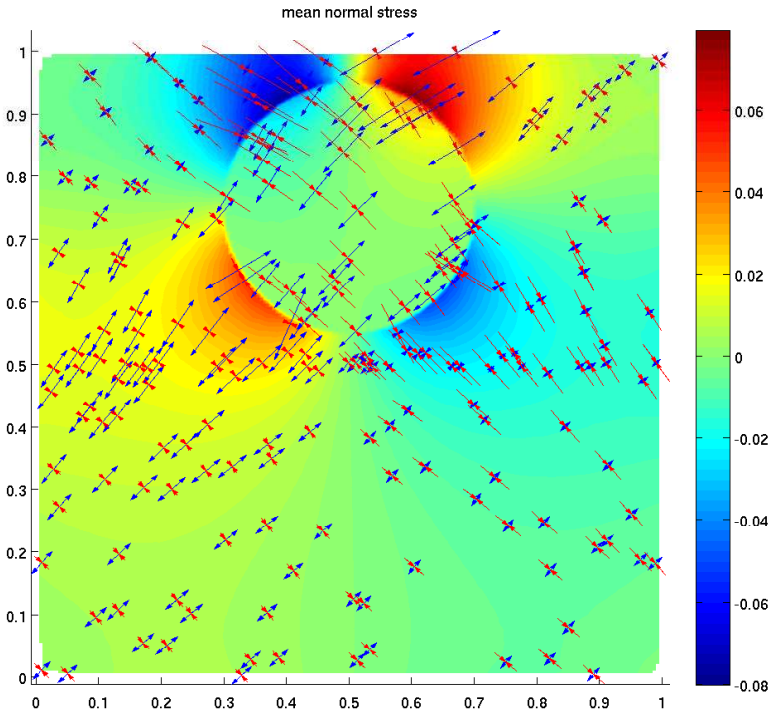


Figure 5.15: Stress solution for a sheared, elastic box with an inclusion of different strength (see problem set for details).

5.7 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). Their "mechanical" solver (incompressible Stokes fluid, to be discussed in the next section) was rewritten for the elastic problem, and simplified 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 us (this one is 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 section's 2-D heat equation exercise.

5.7.1 Implementation of static 2-D elasticity

Problem in strong form

The strong form of the PDE that governs force balance in a medium is given by

$$\nabla \cdot \sigma + \boldsymbol{f} = \mathbf{0}, \tag{5.195}$$

where $\sigma = \sigma_{ij}$ is the stress tensor and $\boldsymbol{f}$ a body force (such as due to gravity). (Note that this equation is a general force balance equation in the absence of inertia. You can use it for static elastic deformation, as we do here, or the Stokes fluid flow problem, as we will discuss subsequently. The difference arises in the constitutive law.)

Written in component form as PDEs for the finite element domain Ω for each of the three spatial coordinates i this is

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0 \quad \text{on} \quad \Omega \tag{5.196}$$

with essential boundary conditions for displacements $\boldsymbol{u} = \boldsymbol{g}$ on $\Gamma_{\boldsymbol{g}}$. Natural boundary conditions for tractions $\boldsymbol{h} = \sigma \cdot \boldsymbol{n}$ shall apply on $\Gamma_{\boldsymbol{h}}$ with vector $\boldsymbol{n}$ normal to the boundary such that

$$u_i = g_i \quad \text{on} \quad \Gamma_{g_i} \tag{5.197}$$

$$\sigma_{ij} n_j = h_i \quad \text{on} \quad \Gamma_{h_i}. \tag{5.198}$$

Here, $\Gamma_{\boldsymbol{h}}$ 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

$$\sigma = C\epsilon \quad \text{or} \quad \sigma_{ij} = C_{ijkl}\epsilon_{kl}, \tag{5.199}$$

with the elasticity tensor C, and the strain-tensor ε, computed as

$$\varepsilon_{ij} = u_{(i,j)} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right). \tag{5.200}$$

Note 1: Notice the definition of the $_{(i,j)}$ derivative short-hand, *e.g.* operating on $\boldsymbol{u}$ to get the tensor ε, like $u_{(i,j)}$.

Note 2: C is a 4^{th} order tensor and somewhat cumbersome to deal with. Noticing that there are only 6 independent components in σ and ε, we can write the 21 independent components of C in the Voigt notation, as a 6×6 matrix, C_V. However, this matrix has different definitions (see, *e.g.* *Browaeys and Chevrot*, 2004, for a discussion), and is not a tensor anymore. I.e. you can do math with it, such as multiplying $C_V \cdot \boldsymbol{\varepsilon}_6$ to get the stress state, where $\boldsymbol{\varepsilon}_6$ has the six independent components of ε, but you cannot rotate C_V anymore. For this, the full 4^{th} order C has to be restored.

For an isotropic material, the constitutive law between total stress and strain thankfully simplifies to

$$\sigma_{ij} = \lambda \varepsilon_{kk} \delta_{ij} + 2\mu \varepsilon_{ij} = \lambda \Delta \delta_{ij} + 2\mu \varepsilon_{ij}, \tag{5.201}$$

where μ and λ are the shear modulus and Lamè parameter, respectively; the latter specifies how incompressible a body is. This formulation uses the isotropic dilation,

$$\Delta = \varepsilon_{ii} = \sum_{i=1}^{3} \varepsilon_{ii}, \tag{5.202}$$

and the Kronecker δ, $\delta_{ij} = 1$ for $i = j$ and zero for $i \neq j$.

The elastic moduli can also be expressed differently, *e.g.* we can write

$$\lambda = \mu \frac{2\nu}{1-2\nu} = \frac{\nu E}{(1+\nu)(1-2\nu)} \quad \text{with} \quad E = 2\mu(1+\nu), \tag{5.203}$$

with the Poisson ratio ν and Young's modulus E. (There are only two independent material parameters for isotropic elasticity.) 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 $E = \sigma_{zz}/\varepsilon_{zz}$.

The incompressibility, K, is defined as

$$p = -K\Delta = -K\varepsilon_{ii} \tag{5.204}$$

where p is pressure with

$$p = -\frac{1}{3}\sum \sigma_{ii} = -\frac{1}{3}\sigma_{ii}, \tag{5.205}$$

and can be computed from

$$K = \lambda + \frac{2}{3}\mu = \frac{E}{3(1-2\nu)}, \tag{5.206}$$

or

$$\mu = \frac{3K(1-2\nu)}{2(1+\nu)}. \tag{5.207}$$

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 $\boldsymbol{u}$ for all virtual displacements $\boldsymbol{w}$ such that

$$a(\boldsymbol{w}, \boldsymbol{u}) = (\boldsymbol{w}, \boldsymbol{f}) + (\boldsymbol{w}, \boldsymbol{h})_{\Gamma_{\boldsymbol{h}}} \tag{5.208}$$

with

$$a(\boldsymbol{w}, \boldsymbol{u}) = \int d\Omega \; w_{(i,j)} C_{ijkl} u_{(k,l)} \tag{5.209}$$

$$(\boldsymbol{w}, \boldsymbol{f}) = \int d\Omega \; w_i f_i \tag{5.210}$$

$$(\boldsymbol{w}, \boldsymbol{h})_{\Gamma_h} = \sum_{i=1}^{3} \left(\int_{\Gamma_{h_i}} d\Gamma \; w_i h_i \right). \tag{5.211}$$

Note that unlike the thermal problem, the solution function we wish to obtain using the finite element method is a vector, $\boldsymbol{u}$, rather than a scalar.

Matrix assembly

In the finite element approximation, we then solve for the nodal displacements $\boldsymbol{d}$ which approximate $\boldsymbol{u}$ within the elements with shape functions N from

$$K\boldsymbol{d} = \boldsymbol{F}. \tag{5.212}$$

The global K is assembled from the element level by

$$k^e_{ab} = \int_{\Omega^e} d\Omega \; B_a^T D B_b \tag{5.213}$$

where a, b are local node numbers. The elemental force vector at local node a is given by

$$f^e_i = \int_{\Omega^e} d\Omega \; N_a f_i + \int_{\Gamma^e_{h_i}} d\Gamma \; N_a h_i - \sum_b k_{ab} g_b. \tag{5.214}$$

B connects displacements at the nodal level with strains. For 2-D,

$$B_a = \begin{pmatrix} \frac{\partial N_a}{\partial x} & 0 \\ 0 & \frac{\partial N_a}{\partial z} \\ \frac{\partial N_a}{\partial z} & \frac{\partial N_a}{\partial x} \end{pmatrix}. \tag{5.215}$$

We can represent the strain tensor ε as a strain vector $\boldsymbol{e}$ that can be computed from displacements $\boldsymbol{u}$ by a gradient operator L, like

$$\boldsymbol{e} = L\boldsymbol{u} \quad \text{or} \quad e_j = L_{jk} u_k. \tag{5.216}$$

In 2-D, for example,

$$\boldsymbol{e} = \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{zz} \\ \gamma_{xz} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial z} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} u_x \\ u_z \end{pmatrix}, \tag{5.217}$$

where the definition of $\gamma_{xy} = 2\varepsilon_{xy}$ simplifies the notation, and it is where the "engineering strain" convention arises. Make sure to distinguish it from ε, *i.e.* convert with the factor 2 if needed.

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,

$$\boldsymbol{u} = u_k = N_a \boldsymbol{d}_a = N_a d_a^k \tag{5.218}$$

where $\boldsymbol{d}_a$ is the displacement at the local node a, and d_a^k is the k-th spatial component of this displacement. Then,

$$\boldsymbol{e} = e_j = L_{jk} N_a d_a^k = B_{jka} d_a^k = B_a \boldsymbol{d}_a \tag{5.219}$$

defines B_a. If we define a stress vector

$$\boldsymbol{s} = \begin{pmatrix} \sigma_{xx} \\ \sigma_{zz} \\ \tau_{xz} \end{pmatrix} \tag{5.220}$$

(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

$$\boldsymbol{s} = D\boldsymbol{e} = DB_a \boldsymbol{d}_a. \tag{5.221}$$

The D matrix is a "condensed" version of C,

$$D_{IJ} = C_{ijkl}, \tag{5.222}$$

where $I, J = 1, 2, \ldots, n_{sd}(n_{sd}+1)/2$ in n_{sd} dimensions, which exploits symmetries in C such that

$$w_{(i,j)} C_{ijkl} u_{(k,l)} = \boldsymbol{e}(\boldsymbol{w})^T \mathsf{D} \boldsymbol{e}(\boldsymbol{u}). \tag{5.223}$$

Note that D may or may not be identical to C in the Voigt notation, C_V.

Equation (5.209) can then be written as

$$a(\boldsymbol{w}, \boldsymbol{u}) = \int d\Omega \; \boldsymbol{e}(\boldsymbol{w})^T \mathsf{D} \boldsymbol{e}(\boldsymbol{u}), \tag{5.224}$$

where $\boldsymbol{e}(\boldsymbol{w})$ indicates applying the gradient operator to the virtual displacements, as opposed to $\boldsymbol{e}(\boldsymbol{u})$ as in eq. (5.216).

In the isotropic, 2-D *plane strain* approximation, D takes the simple form

$$\mathsf{D} = \begin{pmatrix} \lambda+2\mu & \lambda & 0 \\ \lambda & \lambda+2\mu & 0 \\ 0 & 0 & \mu \end{pmatrix} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix}, \tag{5.225}$$

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 $\sigma_{yy} = 0$,

$$\mathsf{D} = \begin{pmatrix} \bar{\lambda}+2\mu & \bar{\lambda} & 0 \\ \lambda & \bar{\lambda}+2\mu & 0 \\ 0 & 0 & \mu \end{pmatrix} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}, \tag{5.226}$$

with

$$\bar{\lambda} = \frac{2\lambda\mu}{\lambda+2\mu}. \tag{5.227}$$

From eq. (5.227), 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, $\sigma = 2\eta\dot{\varepsilon}$, is analogous to the elastic case with $\sigma = 2\mu\varepsilon$, assuming the material is incompressible.

The latter can, in theory, be achieved by letting $\nu \to 1/2$ for which $K/\mu \to \infty$ 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. 5.8.

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 section 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_test.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. 5.6.

(d) Download and inspect `det2D.m`, `inv2D.m`, and `eig2d.m` (for computing the determinant, inverse, and eigen system of 2×2 matrices, respectively). 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 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 eigen system 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.

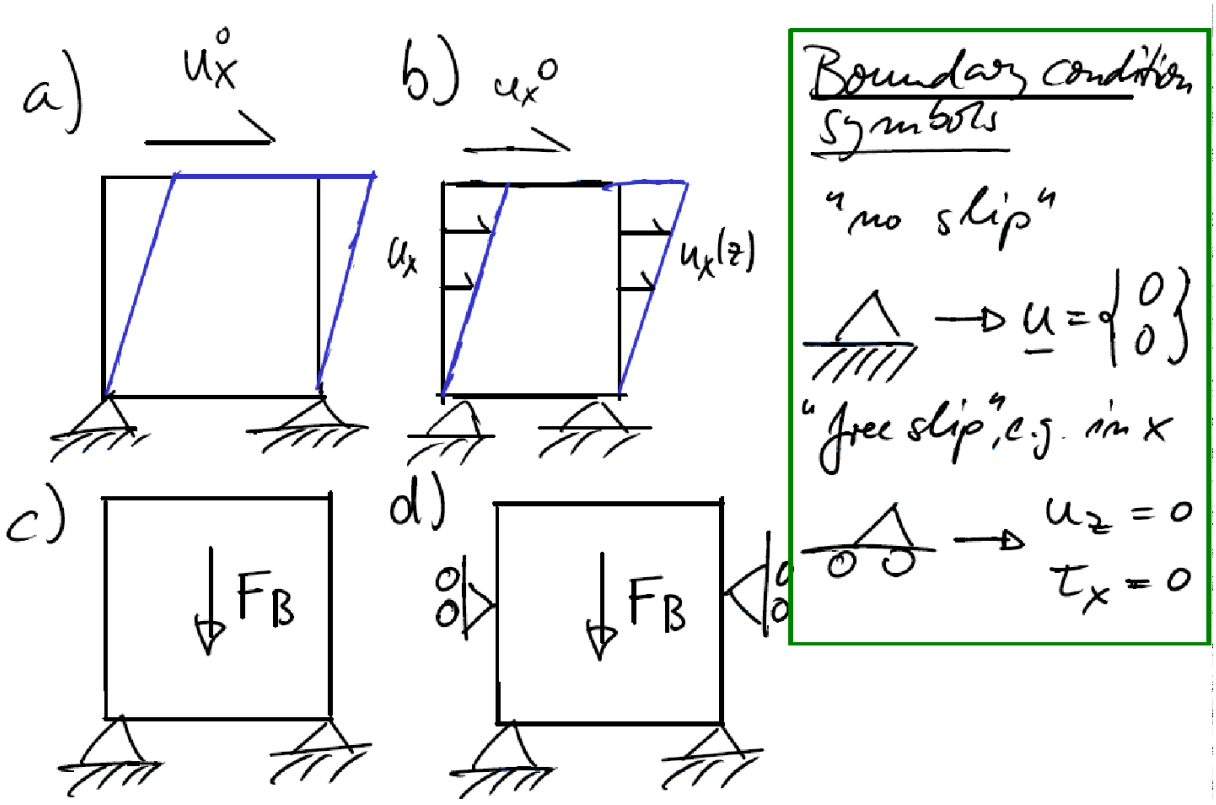


Figure 5.16: Load case sketches for some of the exercises, along with common symbols for kinematic boundary conditions.

(a) Assume the body is fixed at the base (zero displacement $\boldsymbol{u}$ 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 5.16a). 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 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 provided 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 5.16b 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 5.16a. 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 5.16c). Compute the displacements and stresses, plot those, and comment.

(g) Compute the body force load case of Figure 5.16d 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 5.16b, 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.

5.8 Incompressible flow and elasticity with FE

Reading: *Hughes* (2000) sec. 4.2-4.4

5.8.1 Governing equations

- As for the thermal or the elastic problem, we will only consider the static case (but see sec. 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 (see secs. 4.10 and 7) 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 modulus, or incompressibility, K, and shear, μ, modulus can be expressed as a function of Poisson's ratio ν

$$\frac{K}{\mu} = \frac{2(1+\nu)}{3(1-2\nu)}.$$

As noted earlier, for the $\nu \to \frac{1}{2}$, incompressible case, $\frac{K}{\mu} \to \infty$.

However, in this case we cannot use the regular, elastic (isotropic, linear) constitutive law

$$\sigma_{ij} = \lambda \frac{\partial u_k}{\partial x_k} \delta_{ij} + 2\mu \, \varepsilon_{ij} = \lambda \varepsilon_{kk} \delta_{ij} + 2\mu \varepsilon_{ij} \tag{5.228}$$

because $\lambda = \frac{2\nu\mu}{1-2\nu}$ becomes unbounded for $\nu = \frac{1}{2}$. Therefore, we need to use

$$\sigma_{ij} = -p \, \delta_{ij} + 2\mu \, \varepsilon_{ij} \tag{5.229}$$

instead, where the hydrostatic pressure is

$$p = -\frac{1}{3}\sigma_{kk} \, . \tag{5.230}$$

The fluid equivalent of eq. (5.229) is

$$\sigma_{ij} = -p \, \delta_{ij} + 2\eta \, \dot{\varepsilon}_{ij} \tag{5.231}$$

where we have replaced strain ε with strain rate $\dot{\varepsilon}$, 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 ($\nabla \sigma = \boldsymbol{f}$) which is given by the continuity (of mass) equation. In the case of an incompressible medium

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u_i}{\partial x_i} = 0 \tag{5.232}$$

so that the strong form of the incompressible elastic and fluid problems become

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0 \qquad \frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0 \tag{5.233}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \qquad \frac{\partial v_i}{\partial x_i} = 0 \tag{5.234}$$

$$u_i = g_i \qquad v_i = g_i \tag{5.235}$$

$$\sigma_{ij} n_j = h_i \qquad \sigma_{ij} n_j = h_i \tag{5.236}$$

where eqs. (5.233) and (5.234) hold in the domain Ω, eqs. (5.235) and (5.236) are boundary conditions and hold on Γ_{g_i} and Γ_{h_i} respectively. $\boldsymbol{u}$ and $\boldsymbol{v}$ are displacement and velocity, respectively, and

$$\varepsilon_{ij} = u_{(i,j)} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \tag{5.237}$$

$$\dot{\varepsilon}_{ij} = v_{(i,j)} = \frac{1}{2}\left(\frac{\partial v_j}{\partial x_i} + \frac{\partial v_i}{\partial x_j}\right). \tag{5.238}$$

Note that from eq. (5.232) and by Gauß' Theorem

$$\int \mathrm{d}\Omega \, \frac{\partial u_i}{\partial x_i} = \int \mathrm{d}\Gamma \, u_i \, n_i = \int \mathrm{d}\Gamma \, g_i \, n_i = 0 \tag{5.239}$$

and if there are only displacement/velocity BCs, and $\boldsymbol{f} = 0$, then pressures are only determined up to a constant.

5.8.2 FE solution to the incompressible elastic/flow problem

Different approaches exist involving Lagrange multipliers, penalty methods (see sec. 4.10), 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. $\lambda \to \infty$ would lead to the desired case of $\nabla \cdot \boldsymbol{u} = 0$, 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

$$\sigma_{ij} = -p \, \delta_{ij} + 2\mu \, \varepsilon_{ij} \tag{5.240}$$

$$\frac{\partial u_i}{\partial x_i} + \frac{p'}{\lambda} = 0 \tag{5.241}$$

where eq. (5.241) corresponds to the elastic case. For $\nu \to \frac{1}{2} \Rightarrow \lambda \to \infty \Rightarrow \frac{\partial u_i}{\partial x_i} = 0$.
For $\nu < \frac{1}{2}$, we can eliminate eq. (5.241)

$$p' = -\lambda \frac{\partial u_i}{\partial x_i} \tag{5.242}$$

such that (5.240) recovers

$$\sigma_{ij} = \lambda \frac{\partial u_i}{\partial x_i} \delta_{ij} + 2\mu \, \varepsilon_{ij}.$$

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. (5.242)

$$p' = -\lambda\varepsilon_{ii}.$$

$p' \approx p$ and $\lambda \approx K$ only for $\mu \ll \lambda$, the nearly incompressible case. The major results are outlined below.

Equations in strong form

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0 \qquad \text{on } \Omega \tag{5.243}$$

$$\frac{\partial u_i}{\partial x_i} + \frac{p}{\lambda} = 0 \qquad \text{on } \Omega \tag{5.244}$$

$$u_i = g_i \qquad \text{on } \Gamma_{g_i} \tag{5.245}$$

$$\sigma_{ij} n_j = h_i \qquad \text{on } \Gamma_{h_i} \tag{5.246}$$

Equations in weak form

$$\int \mathrm{d}\Omega \, w_{(i,j)} \sigma_{ij} - \int \mathrm{d}\Omega \, q \left(\frac{\partial u_i}{\partial x_i} + \frac{p}{\lambda}\right) = \int \mathrm{d}\Omega \, w_i \, f_i + \sum_i^{n_{sd}} \int \mathrm{d}\Gamma_{h_i} \, w_i \, h_i \tag{5.247}$$

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.* linear) than the displacements (*e.g.* quadratic).

Matrix formulation

$$\begin{pmatrix} \bar{K} & G \\ G^T & M \end{pmatrix} \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \bar{\boldsymbol{F}} \\ \boldsymbol{H} \end{pmatrix} \tag{5.248}$$

where the LHS is symmetric but not positive definite (it has negative eigen values or $\sim$ zero eigen values for pressure modes), $\boldsymbol{p}$ are the pressures at nodes (*e.g.* center of element), $\boldsymbol{d}$ are the displacements at nodes (*e.g.* edges of elements), $\bar{\boldsymbol{F}}$ are the body forces, and $\boldsymbol{H}$ 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 (including finite compressibility) case.

$$\bar{K} \qquad \bar{a}(\boldsymbol{w}, \boldsymbol{d}) \qquad \text{stiffness matrix (symm, pos. def.)} \tag{5.249}$$

$$G \qquad -(\nabla \boldsymbol{w}, \boldsymbol{p}) \qquad \text{gradient operator} \tag{5.250}$$

$$G^T \qquad -(\boldsymbol{q}, \nabla \cdot \boldsymbol{v}) \qquad \text{divergence operator} \tag{5.251}$$

$$M \qquad -\left(\boldsymbol{q}, \frac{p}{\lambda}\right) \qquad \text{symm, neg. def. for } \nu \to \frac{1}{2} \quad \text{and} \quad M \to 0 \tag{5.252}$$

In general, we can distinguish 3 cases:

(a) **The compressible case**

$$\bar{K}\,\boldsymbol{d} + G\,\boldsymbol{p} = \bar{\boldsymbol{F}} \tag{5.253}$$

$$G^T\,\boldsymbol{d} + M\,\boldsymbol{p} = \boldsymbol{H} \tag{5.254}$$

solve for $\boldsymbol{p}$

$$\boldsymbol{p} = M^{-1}(\boldsymbol{H} - G^T\,\boldsymbol{d}) \tag{5.255}$$

substitute eq. (5.256) into eq. (5.253)

$$\begin{aligned} \bar{K}\,\boldsymbol{d} + G\,M^{-1}(\boldsymbol{H} - G^T\,\boldsymbol{d}) &= \bar{\boldsymbol{F}} \\ (\bar{K} - G\,M^{-1}\,G^T)\boldsymbol{d} &= \bar{\boldsymbol{F}} - G\,M^{-1}\,\boldsymbol{H} \end{aligned} \tag{5.256}$$

which reduces to solving the following system of equations

$$K\,\boldsymbol{d} = \boldsymbol{F}$$

where K is symmetric and positive definite and $K = \bar{K} - G\,M^{-1}\,G^T$ and $\boldsymbol{F} = \bar{\boldsymbol{F}} - G\,M^{-1}\,\boldsymbol{H}$ from eq. (5.256). As the condition number of K is larger then that of $\bar{K}$, we generally need to use direct solvers for this approach.

If p is discontinuous on the elements, eq. (5.256) can be solved locally, on the element level.
$\to$ determine $\boldsymbol{p}$ from eq. (5.241).

(b) **The incompressible case**
Solve eq. (5.253) for $\boldsymbol{d}$, pre-multiply with G^T and use eq. (5.254) to get the pressure.

$$(G^T \bar{K}^{-1} G)\boldsymbol{p} = G^T \bar{K}^{-1} F - \boldsymbol{H} \tag{5.257}$$
$$K \quad \boldsymbol{p} = \quad \boldsymbol{F} \tag{5.258}$$

(c) **The element-by-element, discontinuous pressure case**

$$K\,\boldsymbol{d} = \boldsymbol{F} \tag{5.259}$$
$$\boldsymbol{u}(\boldsymbol{x}) = \sum_a N_a(\boldsymbol{x})\,\boldsymbol{d}^a \tag{5.260}$$
$$p(x) = \sum_{\tilde{a}} \tilde{N}_{\tilde{a}}(\boldsymbol{x}) p_{\tilde{a}} \tag{5.261}$$
$$K \leftarrow K^e \quad \text{from element levels} \tag{5.262}$$
$$\boldsymbol{F} \leftarrow f^e \tag{5.263}$$

$$K^e = \bar{K}^e - g^e (m^e)^{-1} (g^e)^T$$
$$\boldsymbol{f}^e = \bar{\boldsymbol{f}} - g^e (m^e)^{-1} \boldsymbol{h}^e$$
$$\boldsymbol{p}^e = -(m^e)^{-1} (g^e)^T \boldsymbol{d}$$

Matrix assembly for the element-by-element, discontinuous pressure case

$$\bar{\boldsymbol{K}}_{ab} = \int_{\Omega^e} \mathrm{d}\Omega\; B_a^T \bar{D}\, B_b$$

D here only has deviatoric terms, for the plane strain case

$$\bar{D} = \mu \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

and B transforms displacements into strains (as before).

$$\bar{f}_p^e = \int \mathrm{d}\Omega\; N_a\, f_i + \int_{\Gamma_{h_i}} \partial\Gamma\; N_a\, h_i - \sum_q k_{pq} g_q^e$$

Pressure components

$$m^e_{\tilde{a}\tilde{b}} = \int_{\Omega^e} \mathrm{d}\Omega - \frac{1}{\lambda}\,\tilde{N}_{\tilde{a}}\,\tilde{N}_{\tilde{b}} \tag{5.264}$$

Mixed (5.265)

$$g_{a\tilde{a}} = -\int \mathrm{d}\Omega\; \nabla\cdot(N_a\,\boldsymbol{e})\,\tilde{N}_{\tilde{a}} \tag{5.266}$$

$$h^e_{\tilde{a}} = -\sum g^e_{p\tilde{a}}\,g^e_p \tag{5.267}$$

Stress vector for 2-D plane strain

$$\boldsymbol{\sigma}(\boldsymbol{x}) = -\sum_{\tilde{a}} \tilde{N}_{\tilde{a}}(\boldsymbol{x}) p_{\tilde{a}} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + \bar{D}(\boldsymbol{x}) \sum_a B_a(\boldsymbol{x})\,\boldsymbol{d}_a$$

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. Another way to think of it is that we actually solve a case with a compressible bulk deformation; iterations are performed until the resulting system is incompressible.

$\boldsymbol{p}^0 = 0$
while $max(\Delta\boldsymbol{p}^i) >$ tolerance

$$\boldsymbol{d}^i = (\bar{K} - G\,M^{-1}\,G^T)^{-1}(\boldsymbol{F} - G\,\boldsymbol{p}^i) \tag{5.268}$$

$$\Delta\boldsymbol{p}^i = -M^{-1}G^T\boldsymbol{d}^i \quad \leftarrow \text{ quasi-divergence} \tag{5.269}$$

$$\boldsymbol{p}^{i+1} = \boldsymbol{p}^i + \Delta\boldsymbol{p}^i \tag{5.270}$$

$$i = i + 1 \tag{5.271}$$

end

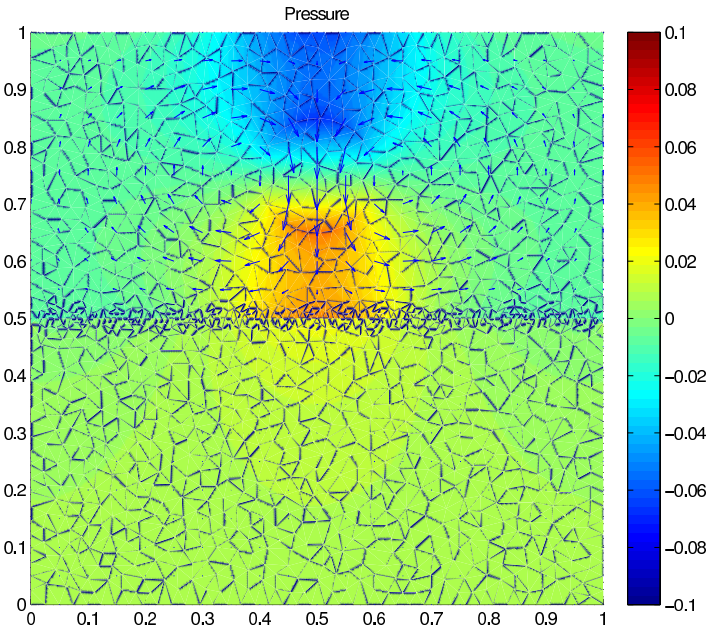


Figure 5.17: Pressure and velocity solution for a sinking, fluid slab impinging on viscosity contrast problem.

5.9 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). As for the heat and elasticity problems, we 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 us or the original authors; it uses, *e.g.*, reordering of node numbers to improve matrix solutions which comes an important memory issue for larger problems. The notation here is close to *Dabrowski et al.* (2008), for simplicity, but *Hughes* (2000) has a somewhat clearer exposition.

5.9.1 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

secs. 4.10 and 5.7). These approximations transform the general, Navier-Stokes equation for fluids into the Stokes equation, which is easier to solve, on the one hand, because there is no turbulence. On the other hand, it is more complicated numerically as Stokes requires implicit solution methods, whereas turbulent equations can often be solved in an explicit manner.

The static force-balance equations for body forces due to gravity are given by

$$\nabla \cdot \sigma = \boldsymbol{f} = \rho \boldsymbol{g} \quad \text{or} \quad \frac{\partial \sigma_{ij}}{\partial x_j} = \rho g_i, \tag{5.272}$$

where σ is the stress tensor, ρ density, and $\boldsymbol{g}$ gravitational acceleration ($g_i = g\delta_{iz}$).

We assume that the medium is incompressible and a linear (Newtonian) fluid constitutive law holds,

$$\sigma_{ij} = -p\delta_{ij} + 2\eta\dot{\varepsilon}'_{ij}, \tag{5.273}$$

where η is the viscosity, p pressure, and $\dot{\varepsilon}'$ the deviatoric strain-rate tensor,

$$\dot{\varepsilon}'_{ij} = v_{(i,j)} - \frac{1}{3}\frac{\partial v_k}{\partial x_k}\delta_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) - \frac{1}{3}\frac{\partial v_k}{\partial x_k}\delta_{ij}, \quad \text{or} \quad \dot{\varepsilon}' = \dot{\varepsilon} - \frac{1}{3}tr(\dot{\varepsilon}), \tag{5.274}$$

where $\boldsymbol{v}$ are the velocities, and $\dot{\varepsilon}'$ is the total strain-rate reduced by the isotropic part.

We can define

$$\dot{e} = \frac{1}{3}\sum_i \dot{\varepsilon}_{ii} = \frac{1}{3}\dot{\varepsilon}_{ii} = \frac{1}{3}tr(\dot{\varepsilon}) \tag{5.275}$$

in analogy to the pressure

$$p = -\frac{1}{3}\sum_i \sigma_{ii} \tag{5.276}$$

such that deviatoric stress and strain-rate are defined from the isotropic quantities as

$$\tau_{ij} = \sigma_{ij} + p\delta_{ij}, \quad \text{and} \tag{5.277}$$

$$\dot{\varepsilon}'_{ij} = \dot{\varepsilon}_{ij} - \dot{e}\delta_{ij}, \quad \text{and} \tag{5.278}$$

$$\tag{5.279}$$

Using the constitutive law, and assuming 2-D (x-z space), the Stokes equation can be written as (also see sec. 7)

$$\frac{\partial}{\partial x}\left(\eta\left(\frac{4}{3}\frac{\partial v_x}{\partial x} - \frac{2}{3}\frac{\partial v_z}{\partial z}\right)\right) + \frac{\partial}{\partial z}\left(\eta\left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x}\right)\right) - \frac{\partial p}{\partial x} = 0 \tag{5.280}$$

$$\frac{\partial}{\partial z}\left(\eta\left(\frac{4}{3}\frac{\partial v_z}{\partial z} - \frac{2}{3}\frac{\partial v_x}{\partial x}\right)\right) + \frac{\partial}{\partial x}\left(\eta\left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x}\right)\right) - \frac{\partial p}{\partial z} = \rho g_z. \tag{5.281}$$

Often, we write the constitutive law for deviatoric quantities only,

$$\tau_{ij} = 2\eta\dot{\varepsilon}'_{ij} \quad \text{with} \quad \tau_{ij} = \sigma_{ij} + p = \sigma_{ij} - \sigma_{kk}/3. \tag{5.282}$$

Incompressibility translates to a constraint on the divergence of the velocity

$$\nabla \cdot \boldsymbol{v} = 0 \quad \text{or} \quad \frac{\partial v_i}{\partial x_i} = 0, \tag{5.283}$$

which allows solving eq. (5.272) for the additional unknown, pressure. For $\nabla \cdot \boldsymbol{v} = \mathbf{0}$,

$$tr(\dot{\varepsilon}) = 0 \quad \rightarrow \quad \dot{\varepsilon}' = \dot{\varepsilon}, \tag{5.284}$$

but we made the distinction between deviatoric and total strain-rate because we numerically only approximate the incompressible continuity equation, eq. (5.283), by requiring the divergence to be smaller than some tolerance.

There are several approaches to do this (*e.g.* penalty methods (sec. 4.10) or Lagrange methods) which typically involve iterations to progressively introduce additional "stiffness" to the medium (sec. 5.8). We shall allow for a finite, large bulk viscosity, κ, such that eq. (5.283) is approximated by

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = -\frac{p}{\kappa}, \tag{5.285}$$

the right hand side would $\rightarrow 0$ for $\kappa \rightarrow \infty$. Eq. (5.285) is valid for the incompressible and the compressible cases. However, for the compressible case, where the constitutive law, eq. (5.273), is replaced by

$$\sigma_{ij} = \kappa \frac{\partial v_k}{\partial x_k} \delta_{ij} + 2\eta \dot{\varepsilon}_{ij}, \tag{5.286}$$

p cannot be interpreted as the actual pressure, $P = -\sigma_{ii}/3$, rather it is a pressure parameter because

$$P = -(\kappa + 2\eta/3) \frac{\partial v_i}{\partial x_i} \tag{5.287}$$

and

$$p = -\kappa \frac{\partial v_i}{\partial x_i}. \tag{5.288}$$

Note: The general, compressible case is identical to the elastic formulation where $\boldsymbol{v} \rightarrow \boldsymbol{u}$ and the constitutive law is

$$\sigma_{ij} = \lambda \frac{\partial v_k}{\partial x_k} \delta_{ij} + 2\mu \dot{\varepsilon}_{ij}. \tag{5.289}$$

5.9.2 Problem in strong form

The (finite element) solution is to be found for the problem stated by eqs. (5.272) and (5.285),

$$\frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0 \tag{5.290}$$

$$\frac{\partial v_i}{\partial x_i} + \frac{p}{\kappa} = 0 \tag{5.291}$$

with boundary conditions

$$v_i = g_i \quad \text{on} \quad \Gamma_{g_i} \tag{5.292}$$
$$\sigma_{ij} n_j = h_i \quad \text{on} \quad \Gamma_{h_i} \tag{5.293}$$

for velocities and tractions, respectively.

Problem in weak form

The pressure equation modifies the stiffness matrix component such that

$$\int d\Omega \, w_{(i,j)} \sigma_{ij} - \int d\Omega \, q \left(\frac{\partial v_i}{\partial x_i} + \frac{p}{\kappa} \right) = \int d\Omega \, w_i f_i + \sum_i^{n_{sd}} \int_{\Gamma_{h_i}} d\Gamma \, w_i h_i, \tag{5.294}$$

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 $\dot{\boldsymbol{e}} = \{\dot{\varepsilon}_{xx}, \dot{\varepsilon}_{zz}, \dot{\gamma}_{xz} = 2\dot{\varepsilon}_{xz}\}$ such that strain-rates on an element level can be computed from

$$\dot{\boldsymbol{e}} = B\boldsymbol{v}, \tag{5.295}$$

where $\boldsymbol{v}$ are velocities given at the element-local nodes, and B holds the derivatives, as before. When expressed for the local node a and shape functions N_a,

$$B_a = \begin{pmatrix} \frac{\partial N_a}{\partial x} & 0 \\ 0 & \frac{\partial N_a}{\partial z} \\ \frac{\partial N_a}{\partial z} & \frac{\partial N_a}{\partial x} \end{pmatrix}. \tag{5.296}$$

Likewise, deviatoric stresses can be computed from $\boldsymbol{t} = D\dot{\boldsymbol{e}}$, where the property matrix D shall be given by

$$D = \eta \begin{pmatrix} \frac{4}{3} & -\frac{2}{3} & 0 \\ -\frac{2}{3} & \frac{4}{3} & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{5.297}$$

for a plane-strain approximation (compare the elastic case). This allows to express the stress vector with pressure part as

$$\boldsymbol{s} = -p\boldsymbol{m} + D\dot{\boldsymbol{e}}, \tag{5.298}$$

where $\boldsymbol{m} = \{1, 1, 0\}$. The deviatoric-only version of D is

$$D' = \eta \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.299}$$

In analogy to the displacement, $\boldsymbol{u}$, representation for the elastic problem, interpolated velocities, $\boldsymbol{v}$, are assumed to be given by the summation over the nodal velocities times the shape functions within each element

$$\boldsymbol{v}(\boldsymbol{x}) \approx \sum_{a=1} N_a(\boldsymbol{x}) \boldsymbol{v}_a. \tag{5.300}$$

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 $\boldsymbol{v}$, and we use linear (constant) shape functions for

$$p(\boldsymbol{x}) = \sum_{a'} \tilde{N}_{a'}(\boldsymbol{x}) p_{a'} = \tilde{N}_{a'} p_{a'}, \tag{5.301}$$

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 B_v, such that

$$\nabla \cdot \boldsymbol{v} = \dot{\varepsilon}_v = B_v \boldsymbol{v}^e, \tag{5.302}$$

and $p^e = -\kappa B_v \boldsymbol{v}^e$.

The global system of equations for velocity, $\boldsymbol{V}$, and pressure, $\boldsymbol{P}$, at the nodes is given by

$$\begin{pmatrix} A & Q^T \\ Q & M \end{pmatrix} \begin{pmatrix} \boldsymbol{V} \\ \boldsymbol{P} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F} \\ \boldsymbol{H} \end{pmatrix}, \tag{5.303}$$

where $\boldsymbol{F}$ are the load vectors, *e.g.* due to body forces, and $\boldsymbol{H}$ is due to the divergence that may be imposed traction loads for the compressible case ($\boldsymbol{H} = \boldsymbol{0}$ for incompressible case).

On an element-level, the stiffness matrix is given by

$$\begin{aligned} k^e_{ab} &= \int_{\Omega_e} d\Omega \begin{pmatrix} A & Q^T \\ Q & -\frac{1}{\kappa} M \end{pmatrix} \\ &= \int_{\Omega_e} d\Omega \begin{pmatrix} B_a^T D B_b & -B_v^T \tilde{N}^T \\ -\tilde{N} B_v & -\frac{1}{\kappa} \tilde{N}_a \tilde{N}_b^T \end{pmatrix}, \end{aligned} \tag{5.304}$$

i.e. $Q = -\tilde{N} B_v$, $M = \tilde{N}\tilde{N}^T$, and A corresponds to the total stiffness matrix k in the elastic case. We have omitted the dependence on the local node number in eq. (5.304). Note

that all operations involving Q and M involve the pressure, and not the velocity, shape functions.

We avoid having to actually solve for the global p by using the "static condensation". This means that we locally (element by element) invert M to obtain the pressure from

$$p \approx \tilde{N}_{a'} p_{a'} = \kappa \tilde{N}^T \left(M^{-1} Q \boldsymbol{v}^e \right) = -\kappa B_v \boldsymbol{v}^e. \tag{5.305}$$

(This is not a good idea if combined with iterative solvers.)

We can then simplify eq. (5.304) to the global, linear equation system

$$A' \boldsymbol{V} = \boldsymbol{f}, \tag{5.306}$$

which is to be solved for the nodal velocities $\boldsymbol{V}$. Here, $\boldsymbol{f} = \{\boldsymbol{f}^e\} = \{\rho^e \boldsymbol{g}^e\}$ and (the Schur complement)

$$A' = A + \kappa Q^T M^{-1} Q. \tag{5.307}$$

A' is now symmetric and positive-definite, and the regular, efficient matrix solution methods can be applied. (Note that the A is only symmetric if the Dirichlet boundary conditions are applied carefully. If implemented straightforwardly, A is not symmetric.)

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 $\boldsymbol{V}$ and $\boldsymbol{P}$ (*cf.* *Dabrowski et al.*, 2008), as in

$$\begin{aligned}
\boldsymbol{P}^0 = 0, \quad i = 0 & & & \\
\text{while } \max(\Delta \boldsymbol{P}^i) & > & \text{tolerance} & \\
\boldsymbol{V}^i & = & (A')^{-1}(\boldsymbol{f} - Q^T \boldsymbol{P}^i) & \\
\Delta \boldsymbol{P}^i & = & M^{-1} Q \boldsymbol{V}^i & \\
\boldsymbol{P}^{i+1} & = & \boldsymbol{P}^i + \Delta \boldsymbol{P}^i & \\
i & = & i + 1 & \\
\text{end while} & & &
\end{aligned} \tag{5.308}$$

If and when the algorithm converges, the pressure correction $\Delta \boldsymbol{P}^i = M^{-1} Q \boldsymbol{V}$, which depends on the divergence, $M^{-1} Q \boldsymbol{V}$, goes to zero. Above, all matrices are meant to be the global, not element-local representation.

5.9.3 Exercises

(a) Make sure you have the common FE Matlab subroutines from the earlier exercises (`ip_-triangle.m`, `shp_deriv_triangle.m`, `genereate_mesh.m`), and the `triangle` binary in your working directory.

(b) Download the `mechanical2d_test.m` driver, and the `mechanical2d_std.m` solver. Inspect both and compare with 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 ($\boldsymbol{v} = \mathbf{0}$ 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 $\boldsymbol{x} = \{0.5, 0.5\}$.

(j) Provide an analytical estimate for the sinking velocities 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, power-law fluid with $\dot{\varepsilon}'_{II} \propto \tau^n_{II}$ 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, $\dot{\varepsilon}_{II}$. (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.

5.10 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

5.10.1 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

$$q_i = -\kappa_{ij} \frac{\partial}{\partial x_j} T \qquad \text{(heat flux)}$$

where κ_{ij} is the conductivity matrix.

$$\rho c_p \frac{\partial}{\partial t} T + \frac{\partial}{\partial x_i} q_i = H \tag{5.309}$$

$$(\rho c_p \frac{\partial T}{\partial t} - k \nabla^2 T = H \quad \text{for isotropic conductivity}) \tag{5.310}$$

Boundary conditions

$$T = g \quad \text{on } \Gamma_g \qquad \text{(essential)}$$

$$-q_i n_i = h \quad \text{on } \Gamma_h \qquad \text{(natural)}$$

Initial conditions

$$T(\boldsymbol{x}, t = 0) = T_0(\boldsymbol{x}) \tag{5.311}$$

Weak form

$$(w, \rho c_p \dot{T}) + a(w, T) = (w, H) + (w, h)_\Gamma \tag{5.312}$$

$$(w, \rho c_p T(0)) = (w, \rho c_p T_0) \tag{5.313}$$

$$\dot{T} = \frac{\partial T}{\partial t} \tag{5.314}$$

Galerkin approximation, in analogy to static case

$$T(\boldsymbol{x},t) \simeq v(\boldsymbol{x}.t) + g(\boldsymbol{x},t) \tag{5.315}$$
$$(w, \rho c_p \dot{v}) + a(w,v) = (w,H) + (w,h)_\Gamma - (w, \rho c_p \dot{g}) - a(w,g) \tag{5.316}$$
$$\tag{5.317}$$

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

$$v(\boldsymbol{x},t) = \sum N_A(x)\, d_A(t) \tag{5.318}$$

approximation with shape functions
N_A for all global nodes

The new matrix equation is

$$\text{initial condition} \quad M\,\dot{\boldsymbol{d}} + K\,\boldsymbol{d} = \boldsymbol{F}, \qquad \boldsymbol{d}(0) = \boldsymbol{d}_0 \tag{5.319}$$

with

$$\text{assembly from element level } m^e \ : \qquad M \leftarrow m^e \tag{5.320}$$
$$\text{local nodes a,b :} \qquad m^e = [m^e_{ab}]$$
$$\text{“mass” or “}\textit{capacity}\text{” matrix :} \qquad m^e_{ab} = \int_{\Omega^e} \mathrm{d}\Omega\ N_a \rho c_p N_b$$
$$\text{conductivity matrix :} \qquad K \leftarrow K^e \tag{5.321}$$
$$K^e = [K^e_{ab}]$$
$$\text{same as the static case :} \qquad K^e_{ab} = \int_{\Omega^e} \mathrm{d}\Omega\ B_a^T D\ B_b$$
$$\boldsymbol{F} = \text{heat supply vector :} \qquad \boldsymbol{F} \leftarrow \boldsymbol{f}^e \tag{5.322}$$
$$\boldsymbol{f}^e = [f^e_a]$$
$$\text{from BCs :} \qquad f^e_a = \int_{\Omega^e} N_a\, H + \int_{\Gamma_h} \mathrm{d}\Gamma N_a\, h - \sum (K^e_{ab}\, g_b + m_{ab}\, \dot{g}_b)$$
$$\boldsymbol{d}_0 = M^{-1}\boldsymbol{d}\ ; \boldsymbol{d} \leftarrow \boldsymbol{d}^e \tag{5.323}$$
$$\text{Initial condition :} \qquad \boldsymbol{d}^e = [d^e_a]$$
$$d_a = \int_{\Omega^e} N_a \rho c_p T_0 - \sum m_{ab}\, g^e_b(0)$$

(See *Hughes*, 2000, p. 421).

The main difference with the static sets of equation for the heat equation is the introduction of the M matrix and the need to solve eq. (5.319) as an ODE.

5.10.2 Solution of the semi-discrete heat equation

Solve

$$\begin{aligned} M\,\dot{\boldsymbol{d}} + K\,\boldsymbol{d} &= \boldsymbol{F} \\ \text{with IC} \quad \boldsymbol{d} &= \boldsymbol{d}_0 \end{aligned} \tag{5.324}$$

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. (5.324) is by the generalized trapezoidal method (see *Hughes*, 2000, p. 459).

Generalized Trapezoidal Method

$$M\,\boldsymbol{v}^{n+1} + K\,\boldsymbol{d}^{n+1} = \boldsymbol{F}^{n+1}$$

$$\boldsymbol{d}^{n+1} = \boldsymbol{d}^n + \Delta t\,\boldsymbol{v}^{n+\alpha} \tag{5.325}$$

$$\boldsymbol{v}^{n+\alpha} = (1-\alpha)\boldsymbol{v}^n + \alpha\boldsymbol{v}^{n+1} \tag{5.326}$$

where $\boldsymbol{d}^n$ and $\boldsymbol{v}^n$ are the approximations to $\boldsymbol{d}(t=t^n)$ and $\dot{\boldsymbol{d}}(t=t^n)$, respectively, with

$$t^{n+1} = t^n + \Delta t \tag{5.327}$$

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

v - form implementation

(a) Start at $t = t_0$ with $\boldsymbol{d} = \boldsymbol{d}_0$ given for $n = 0$.

(b) Estimate $\boldsymbol{v}_0 \simeq \dot{\boldsymbol{d}}_0$ from

$$M\,\boldsymbol{v}_0 = \boldsymbol{F}_0 - K\,\boldsymbol{d}_0 \tag{5.328}$$

(c) Compute predictor

$$\tilde{\boldsymbol{d}}^{n+1} = \boldsymbol{d}_n + (1-\alpha)\Delta t\,\boldsymbol{v}^n \tag{5.329}$$

Combine eq. (5.325) & (5.326) with (5.329)

$$\boldsymbol{d}^{n+1} = \tilde{\boldsymbol{d}}^{n+1} + \alpha\,\Delta t\,\boldsymbol{v}^{n+1} \tag{5.330}$$

into eq. (5.324)

$$(M + \alpha\,\Delta t\,K)v^{n+1} = \boldsymbol{F}^{n+1} - K\,\tilde{\boldsymbol{d}}^{n+1} \tag{5.331}$$

(d) Solve eq. (5.331) for $\boldsymbol{v}^{n+1}$ (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 (5.331) (*i.e.* diagonal) does not involve any equation solving for time-stepping. M is lumped for ρ and c_p constant.

d - form implementation

Instead of eq. (5.331), we use (for $\alpha \neq 0$)

$$\frac{1}{\alpha\,\Delta t}(M + \alpha\,\Delta t\,K)\boldsymbol{d}^{n+1} = \boldsymbol{F}^{n+1} + \frac{1}{\alpha\,\Delta t}M\,\tilde{\boldsymbol{d}}^{n+1} \tag{5.332}$$

to obtain $\boldsymbol{d}^{n+1}$, and then update

$$\boldsymbol{v}^{n+1} = \frac{\boldsymbol{d}^{n+1} - \tilde{\boldsymbol{d}}^{n+1}}{\alpha\,\Delta t} \tag{5.333}$$

The right hand side of eq. (5.332) is fast to compute for diagonal M.

The generalized trapezoidal methods are $\alpha < \frac{1}{2}$ conditionally stable for

$$\Delta t \lesssim \frac{2}{(1-2\alpha)\,h^2} \tag{5.334}$$

where h is the smallest grid spacing in the mesh ($h \triangleq$ "mesh parameter"). For the fully explicit method ($\alpha = 0$), we recover

$$\Delta t \leq \frac{2}{h^2} \tag{5.335}$$

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.

Part IV

Appendix

Chapter 6

Basic calculus and algebra review

This section provides a few brief notes on math notation and concepts needed for this text. Not all concepts and formula are presented in a mathematically rigorous way, and you should refer to something like a Math for Engineers text for a more complete treatment. For most of this text, it will be assumed that the reader is familiar with the material treated in this chapter.

6.1 Calculus

6.1.1 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

$$\delta u = \frac{\delta u}{\delta x}(x_0)\delta x. \tag{6.1}$$

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

$$\lim_{\delta x \to 0} \frac{\delta u}{\delta x}(x_0) = \frac{\partial u}{\partial x}. \tag{6.2}$$

The limit in eq. (6.2) 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:

$$\mathrm{d}u = \frac{\partial u}{\partial x}\,\mathrm{d}x + \frac{\partial u}{\partial y}\,\mathrm{d}y + \frac{\partial u}{\partial z}\,\mathrm{d}z. \tag{6.3}$$

Here, $\mathrm{d}x$ and similar are placeholders for infinitesimal changes in the variables. This means that eq. (6.3) works as long as $\mathrm{d}x$ is small enough that a linear relationship between δu and δx still holds. In fact, we can perform a Taylor approximation on *any* $u(x)$ around x_0 by

$$u(x) = u(x_0) + \frac{\partial u}{\partial x}(x_0)(x - x_0) + \frac{\partial^2 u}{\partial x^2}(x_0)\frac{(x - x_0)^2}{2!} + \frac{\partial^3 u}{\partial x^3}(x_0)\frac{(x - x_0)^3}{3!}\ldots \tag{6.4}$$

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.*

$$n! = 1 \times 2 \times 3 \times \ldots n. \tag{6.5}$$

So, as long as $\mathrm{d}x = x - x_0$ 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. (6.4) and so on need to be taken into account. Finite difference methods essentially use Taylor approximations to approximate derivatives, as we will see later.

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):

$$\begin{aligned} \text{If} \quad f(x) &= h(g(x)) && (6.6)\\ f'(x) &= h'(g(x))g'(x), && (6.7) \end{aligned}$$

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

Sum rule:

$$(af(x) + bg(x))' = af'(x) + bg'(x) \tag{6.8}$$

Product rule:

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \tag{6.9}$$

Quotient rule:

$$\text{If} \quad f(x) = \frac{g(x)}{h(x)} \tag{6.10}$$

$$f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2} \tag{6.11}$$

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

$$\frac{\partial^3 f(x)}{\partial x^3} = \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} f(x)\right)\right). \tag{6.12}$$

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

$$\frac{\partial^3 x^3}{\partial x^3} = \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} x^3\right)\right) = \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} 3x^2\right) = \frac{\partial}{\partial x} 6x = 6. \tag{6.13}$$

6.1.2 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 $\nabla = \{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\}$. Note that the operatore ∇ is a vector. When applied to scalar field (a distribution of values that depends on spatial location), such as a temperature distribution $T(x, y, z)$ (meaning T is variable with coordinates x, y, and z, assumed implicitly for all properties from now on), the *gradient* operation

$$grad\ T = \nabla T = \begin{pmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial z} \end{pmatrix} \tag{6.14}$$

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 (*i.e.* vectors that vary in space, $\boldsymbol{x}$). If

$$\boldsymbol{v}(\boldsymbol{x}) = \{v_1(\boldsymbol{x}), v_2(\boldsymbol{x}), v_3(\boldsymbol{x})\} \tag{6.15}$$

is a velocity field, then the *divergence* (grad dot product) operation on a vector field

$$div\ \boldsymbol{v} = \nabla \cdot \boldsymbol{v} \tag{6.16}$$

is equivalent to finding the dilatancy (volumetric) strain-rate $\dot{\triangle}$ from the strain-rate tensor components because

$$\dot{\triangle} = \frac{\dot{\Delta V}}{V} = tr(\dot{\varepsilon}) = \sum_i \dot{\varepsilon}_{ii} = \dot{\varepsilon}_{11} + \dot{\varepsilon}_{22} + \dot{\varepsilon}_{33} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_2} + \frac{\partial v_3}{\partial x_3} = \nabla \cdot \boldsymbol{v}. \tag{6.17}$$

Here V is volume, and $\dot{\Delta V}$ volume rate-change and, mind you, the strain-rate tensor, $\dot{\varepsilon}$, is defined as

$$\dot{\varepsilon} = \dot{\varepsilon}_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right). \tag{6.18}$$

In complete analogy, if the vector field are displacements $\boldsymbol{u}(\boldsymbol{x})$, then $\nabla \cdot \boldsymbol{u}$ yields the dilatancy, *i.e.* the trace of the strain tensor, ε,

$$\varepsilon = \varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{6.19}$$

Eq. (6.17) 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: $div\ \boldsymbol{B} = 0$ because there are no magnetic monopoles, but for the electric field: $div\ \boldsymbol{E} = q$, 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

$$curl\ \boldsymbol{v} = \nabla \wedge \boldsymbol{v}. \tag{6.20}$$

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

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

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \tag{6.21}$$

Some rules for second derivatives:

$$curl(grad\ T) = \nabla \times (\nabla T) \quad = \quad 0 \tag{6.22}$$

$$div(curl\ \boldsymbol{v}) = \nabla \cdot \nabla \times \boldsymbol{v} = 0 \tag{6.23}$$

6.1.3 Integrals

Taking an integral

$$F(x) = \int f(x)dx, \tag{6.24}$$

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

$$F\left(\frac{\partial f(x)}{\partial x}\right) = f(x) + c \tag{6.25}$$

$$\frac{\partial}{\partial x}F\left(\frac{\partial f(x)}{\partial x}\right) = \frac{\partial}{\partial x}\left(f(x) + c\right) = f'(x). \tag{6.26}$$

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 $f(x)$

$$\int_a^b f(x)dx = F(b) - F(a) \tag{6.27}$$

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

$$F(x) = cx + d \tag{6.28}$$

$$F(b) = cb + d \tag{6.29}$$

$$F(a) = ca + d \tag{6.30}$$

$$F(b) - F(a) = c(b - a), \tag{6.31}$$

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	
$1/x$	$\ln(\lvert x\rvert) + C$	
$\sin(x)$	$-\cos(x) + C$	
$\cos(x)$	$\sin(x) + C$	

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

$$\int_0^\infty e^{-x^2}dx = \frac{\sqrt{\pi}}{2} \tag{6.32}$$

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:

$$\int (af(x) + bg(x) + \ldots)\, dx = \int af(x) + bg(x) \ldots dx \tag{6.33}$$

$$\int dx\, f(x) = \int f(x) dx \tag{6.34}$$

Linearity:

$$\int_a^b (cf(x) + dg(x))\, dx = c \int_a^b f(x) dx + d \int_a^b g(x) \tag{6.35}$$

Reversal:

$$\int_a^b f(x) dx = -\int_b^a f(x) dx \tag{6.36}$$

Zero length:

$$\int_a^a f(x) dx = 0 \tag{6.37}$$

Additivity:

$$\int_a^c f(x) dx = \int_a^b f(x) dx + \int_b^c f(x) dx \tag{6.38}$$

Product rules:

$$\int f'(x) f(x) dx = \frac{1}{2} (f(x))^2 + C \tag{6.39}$$

$$\int f'(x) g(x) dx = f(x) g(x) - \int f(x) g'(x) dx \tag{6.40}$$

Quotient rule:

$$\int \frac{f'(x)}{f(x)} dx = \ln |f(x)| + C \tag{6.41}$$

Gauß theorem The integral over the area Ω of the divergence of a vector field $\boldsymbol{f}$ is equivalent to the boundary integral, $\partial\Omega$, over the local normal (to the boundary), $\boldsymbol{n}$, dotted with $\boldsymbol{f}$:

$$\int_\Omega dA\ \nabla \cdot \boldsymbol{f} = \int_{\partial\Omega} ds\ \boldsymbol{n} \cdot \boldsymbol{f}. \tag{6.42}$$

6.2 Linear algebra

TO BE ADDED: Matlab conventions for mathematical operations such as dot and cross products.

6.2.1 The dot product

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

$$c = \boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i, \tag{6.43}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ 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. (6.43), $\sum_{i=1}^{n}$ 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

$$\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2 + a_3 b_3, \tag{6.44}$$

where 1, 2, 3 refer to the vector components along x, y, and z axis, respectively (ADD FIGURE HERE). 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.

When we write out the vector components, we put them on top of each other

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \tag{6.45}$$

or in a list, maybe with curly brackets, like so: $\boldsymbol{a} = \{a_1, a_2, a_3\}$. Here, we will use a bold font $\boldsymbol{a}$ to denote vectors as opposed to scalar a, but another common form is $\vec{a}$, and on a blackboard, you might also see vectors written as $\underline{a}$ because that's easier.

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

$$|\boldsymbol{a}| = \sqrt{\sum_{i}^{n} a_i^2} = \sqrt{a_1^2 + a_2^2 + a_3^2} = \sqrt{a_x^2 + a_y^2 + a_z^2}. \tag{6.46}$$

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

$$\boldsymbol{a} = \{a_x, a_y, a_z\} = a_x \boldsymbol{e}_x + a_y \boldsymbol{e}_y + a_z \boldsymbol{e}_z. \tag{6.47}$$

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

We can restate eq. (6.43) and give another definition of the dot product,

$$\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| \cos\theta \tag{6.48}$$

where θ is the angle between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. The meaning of this is that if you want to know what component of vector $\boldsymbol{a}$ is parallel to $\boldsymbol{b}$, you just take the dot product. Say, you have a velocity $\boldsymbol{v}$ and want the normal velocity v_n along a vector $\boldsymbol{n}$ with $|\boldsymbol{n}| = 1$ that is oriented at a 90° angle (perpendicular) to some plate boundary, you can use $v_n = \boldsymbol{v} \cdot \boldsymbol{n}$.

Also, eq. (6.47) only works because the basis vectors $\boldsymbol{e}_i$ of any coordinate system are, by definition, orthogonal (at right angle, perpendicular, at $\theta = 90°$) to each other and $\boldsymbol{e}_i \cdot \boldsymbol{e}_j = 0$ for all $i \neq j$. Likewise, $\boldsymbol{e}_i \cdot \boldsymbol{e}_i = 1$ for all i since $\boldsymbol{a} \cdot \boldsymbol{a} = |\boldsymbol{a}|^2$, and basis vectors have unity length by definition. Using the Kronecker δ

$$\delta_{ij} = 1 \quad \text{for} \quad i = j, \quad \text{and} \quad \delta_{ij} = 0 \quad \text{for} \quad i \neq j, \tag{6.49}$$

we can write the conditions for the basis vectors as

$$\boldsymbol{e}_i \cdot \boldsymbol{e}_j = \delta_{ij}. \tag{6.50}$$

6.2.2 Vector or cross product

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

$$\boldsymbol{c} = \boldsymbol{a} \wedge \boldsymbol{b} \tag{6.51}$$

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

$$|\boldsymbol{c}| = |\boldsymbol{a} \wedge \boldsymbol{b}| = |\boldsymbol{a}||\boldsymbol{b}| \sin\theta, \tag{6.52}$$

that is, $\boldsymbol{c}$ is largest when $\boldsymbol{a}$ and $\boldsymbol{b}$ are orthogonal, and zero if they are parallel. Compare this relationship to eq. (6.48).

In 3-D,

$$\boldsymbol{c} = \boldsymbol{a} \wedge \boldsymbol{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix} \tag{6.53}$$

(note that there is no i component of $\boldsymbol{a}$ or $\boldsymbol{b}$ in the i component of $\boldsymbol{c}$, this is the aforementioned orthogonality property).

An example for a cross product is the velocity $\boldsymbol{v}$ at a point with location $\boldsymbol{r}$ in a body spinning with the rotation vector $\boldsymbol{\omega}$, $\boldsymbol{v} = \boldsymbol{\omega} \wedge \boldsymbol{r}$. The rotation vector $\boldsymbol{\omega}$ is different from, *e.g.*, $\boldsymbol{r}$ 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). ADD FIGURE

6.2.3 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. For example, if A is 3×3,

$$A = \begin{pmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}. \tag{6.54}$$

You will see matrices printed like so A, with the blackboard version being 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

$$fA = fa_{ij} = f \times \begin{pmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{pmatrix} = \begin{pmatrix} fa_{xx} & fa_{xy} & fa_{xz} \\ fa_{yx} & fa_{yy} & fa_{yz} \\ fa_{zx} & fa_{zy} & fa_{zz} \end{pmatrix} \tag{6.55}$$

Multiplication of a matrix with a vector

$$\begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} = \begin{pmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{pmatrix} . \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_{xx}b_x + a_{xy}b_y + a_{xz}b_z \\ a_{yx}b_x + a_{yy}b_y + a_{yz}b_z \\ a_{zx}b_x + a_{zy}b_y + a_{zz}b_z \end{pmatrix} \tag{6.56}$$

or

$$c_i = \sum_j a_{ij} b_j. \tag{6.57}$$

Multiplication of two matrices works like this:

$$C = AB \tag{6.58}$$

$$c_{ij} = \sum_k a_{ik} b_{kj}, \tag{6.59}$$

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, $AB \neq BA$!

Special types of matrices and matrix operations

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 $1 = I$, $i_{ij} = \delta_{ij}$, *i.e.* this matrix is unity along the diagonal, and zero for all other elements.

Trace The trace of a $n \times n$ matrix A is the sum of its diagonal elements

$$tr(A) = \sum_{i=1}^{n} a_{ii}. \tag{6.60}$$

Determinant The determinant for a 2×2 matrix is computed as

$$det(A) = a_{11}a_{22} - a_{12}a_{21} \tag{6.61}$$

and is a measure of area change. For 3×3,

$$\begin{aligned} det(A) &= a_{11}\left(a_{22}a_{33} - a_{23}a_{32}\right) \\ &- a_{12}\left(a_{21}a_{33} - a_{23}a_{31}\right) \\ &+ a_{13}\left(a_{21}a_{32} - a_{22}a_{31}\right) \end{aligned} \tag{6.62}$$

(note how the 3×3 determinant is assembled from a pattern of 2×2 determinants; for $n > 3$, a correspondingly more complicated formula applies.
ADD FIGURE

Vector cross product based on the determinant The cross product $\boldsymbol{c} = \boldsymbol{a} \wedge \boldsymbol{b}$ (eq. 6.53) can also be written as the determinant of the matrix

$$\begin{pmatrix} \boldsymbol{e}_x & \boldsymbol{e}_y & \boldsymbol{e}_z \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{pmatrix} \tag{6.63}$$

Invariants The trace

$$I_A = tr(A) = a_i i \tag{6.64}$$

and determinant

$$III_A = det(A) \tag{6.65}$$

of a matrix A are two of the three invariants, *i.e.* properties of a tensor (expressed as a matrix) that are independent of a coordinate system. The third is the "second invariant",

$$II_A = a_{11}a_{22} + a_{11}a_{33} + a_{22}a_{33} - a_{12}^2 - a_{13}^2 - a_{23}^2. \tag{6.66}$$

These expressions arise when finding the eigenvectors and values of a tensor, eq. (6.71).

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

Inverse of A, A^{-1} : The inverse of a matrix is defined via

$$A^{-1}A = AA^{-1} = I. \tag{6.67}$$

If the inverse exists, then $(A^{-1})^{-1} = A$, $(A^T)^{-1} = (A^{-1})^T$, and $(AB)^{-1} = B^{-1}A^{-1}$. The inverse only exists if $det(A) \neq 0$.

Orthogonal or rotation matrices: For those matrices,

$$AA^T = A^TA = 1 \tag{6.68}$$

holds.

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

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \tag{6.69}$$

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

Eigenvalues can be found using

$$det(A - \lambda I) = 0 \tag{6.70}$$

and eigen vectors subsequently by using the first property, which leads to

$$det(A - \lambda I) = -\lambda^3 + I_A\lambda^2 - II_A\lambda + III_A = 0. \tag{6.71}$$

If a symmetric matrix A is transformed into the principal axes system, A', there are no off-diagonal elements

$$A = \begin{pmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{pmatrix} \rightarrow A' = \begin{pmatrix} a_1 & 0 & 0 \\ 0 & a_2 & 0 \\ 0 & 0 & a_3 \end{pmatrix} \tag{6.72}$$

where the a_1, a_2, and a_3 correspond to the three eigenvalues λ_i. (The coordinate system reference of A' is then contained in the orientation of the eigen vectors $\boldsymbol{v}_i$.) For a matrix in the principal axis system, the invariants are very easily computed:

$$tr(A') = I_{A'} \quad = I_A \quad = a_1 + a_2 + a_3 \tag{6.73}$$

$$II_{A'} \quad = II_A \quad = a_1a_2 + a_1a_3 + a_2a_3 \tag{6.74}$$

$$det(A') = III_{A'} \quad = III_A \quad = a_1a_2a_3. \tag{6.75}$$

See also sec. 7.2 for definitions of invariatns using deviators, such as for the deviatoric stress tensor.

Matrix decomposition Any quadratic tensor A can be decomposed into a symmetric part A^s (for which $a^s_{ij} = a^s_{ji}$) and an anti-symmetric part A^a (for which $a^a_{ij} = -a^a_{ji}$) 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, $\underline{\underline{\varepsilon}}$), 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 $V = \left(FF^T\right)^{1/2}$. The left-stretch matrix describes the deformation in the rotated coordinate system after the rotation R has been applied to the body.

6.2.4 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^r components and can be written as $n \times n$ matrices. You will see tensors printed like so E, and the blackboard version again double underlining like $\underline{\underline{\varepsilon}}$, making no distinction between tensors and matrices.

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

$$v'_i = L_{ij} v_j = \sum_{j=1}^{3} L_{ij} v_j \tag{6.76}$$

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

$$T'_{ij} = L_{ik} T_{kl} L_{jl} = \sum_k L_{ik} \sum_l T_{kl} L_{jl} = LTL^T \tag{6.77}$$

Chapter 7

Continuum mechanics review

We will assume some familiarity with continuum mechanics as discussed in the context of an introductory geodynamics course; 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.

TO BE REWRITTEN, MORE DISCUSSION ADDED.

7.1 Definitions and nomenclature

- Coordinate system. $\boldsymbol{x} = \{x, y, z\}$ 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 $T(\boldsymbol{x})$ – temperature field – temperature varying in space.

- Indexed variables. For example, the velocity field $\boldsymbol{v}(\boldsymbol{x}) = v_i$ with $i = 1, 2, 3$ implies $\{v_1, v_2, v_3\}$, *i.e.* three variables that are functions of space $\boldsymbol{x} = \{x_1, x_2, x_3\}$.

- Repeated indices indicate summation over these components (also called Einstein summation convention).

$$\frac{\partial v_i}{\partial x_i} \quad \text{with} \quad i = 1,2,3 \quad \text{implies} \quad \sum_{i=1}^{3} \frac{\partial v_i}{\partial x_i} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_2} + \frac{\partial v_3}{\partial x_3} \tag{7.1}$$

- 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 $\boldsymbol{v}$. Local changes in, *e.g.*, temperature are then given by

$$\frac{DT}{Dt} = \frac{\partial T}{\partial t} + \boldsymbol{v}\nabla T = \frac{\partial T}{\partial t} + v_i \frac{\partial T}{\partial x_i}, \tag{7.2}$$

  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 $\boldsymbol{v}$ in and out of our local observation point.

- Tensor = indexed variable + the rule of transformation to another coordinate system.

- Traction = a force per unit area acting on a plane (a vector).

- Mean stress (= −pressure, p): $-p = \bar{\sigma} = \sigma_{ii}/3 = tr(\sigma)/3$

- Mean strain: $\bar{\varepsilon} = \varepsilon_{ii}/3 = tr(\varepsilon)/3 = \theta$ (also called dilatation).

- Traction/stress sign convention. Compression is negative in physics, but usually taken positive in geology. Pressure is always positive compressive.

7.2 Stress tensor

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

$$\sigma_{ij} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} \text{(2D)} \tag{7.3}$$

$$\sigma_{ij} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} \text{(3D)}. \tag{7.4}$$

- 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 within, for example, a finite element program).

- 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):

$$T_{(n)i} = \sigma \boldsymbol{n} = \sigma_{ij} n_j = \sum_{j=1}^{3} \sigma_{ij} n_j \tag{7.5}$$

- 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.

- Stress deviator

  We often decompose the stress tensor, σ, into a hydrostatic pressure, p, which is minus the mean stress tensor, $\bar{\sigma}$, and defined as

$$p = -\bar{\sigma} = -\frac{1}{3} tr(\sigma) = -\frac{\sigma_{ii}}{3} = -\frac{I_\sigma}{3}, \tag{7.6}$$

  where I_σ is the first invariant, eq. (6.64). The deviator, or deviatoric stress, is defined as

$$\tau_{ij} = \sigma_{ij} - \delta_{ij}\bar{\sigma} = \sigma_{ij} + \delta_{ij} p. \tag{7.7}$$

  The deviatoric stress tensor invariants of τ are typically denoted as J (as opposed to

I for the full stress tensor, σ), and given by

$$\begin{aligned} J_\tau = \quad J_1 &= \tau_{ii} = 0 && (7.8)\\ JJ_\tau = \quad J_2 &= \frac{1}{2}\left(\tau_1^2 + \tau_2^2 + \tau_3^2\right) && (7.9)\\ &= \frac{1}{6}\left[(\sigma_1-\sigma_2)^2 + (\sigma_2-\sigma_3)^2 + (\sigma_3-\sigma_1)^2\right] && (7.10)\\ &= \frac{1}{3}I_\sigma^2 - II_\sigma && (7.11)\\ JJJ_\tau = \quad J_3 &= \tau_1\tau_2\tau_3. && (7.12) \end{aligned}$$

The equivalent stress or van Mises stress is defined as

$$\sigma_e = \sqrt{3J_2} = \sqrt{\frac{1}{2}\left[(\sigma_1-\sigma_2)^2 + (\sigma_2-\sigma_3)^2 + (\sigma_3-\sigma_1)^2\right]}. \tag{7.13}$$

7.3 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.

$$\begin{aligned} \varepsilon_{ij} &= \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) && (7.14)\\ \dot{\varepsilon}_{ij} &= \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) && (7.15)\\ &= \begin{pmatrix} \frac{\partial v_1}{\partial x_1} & \frac{1}{2}\left(\frac{\partial v_1}{\partial x_2} + \frac{\partial v_2}{\partial x_1}\right) & \frac{1}{2}\left(\frac{\partial v_1}{\partial x_3} + \frac{\partial v_3}{\partial x_1}\right) \\ \frac{1}{2}\left(\frac{\partial v_2}{\partial x_1} + \frac{\partial v_1}{\partial x_2}\right) & \frac{\partial v_2}{\partial x_2} & \frac{1}{2}\left(\frac{\partial v_2}{\partial x_3} + \frac{\partial v_3}{\partial x_2}\right) \\ \frac{1}{2}\left(\frac{\partial v_3}{\partial x_1} + \frac{\partial v_1}{\partial x_3}\right) & \frac{1}{2}\left(\frac{\partial v_3}{\partial x_2} + \frac{\partial v_2}{\partial x_3}\right) & \frac{\partial v_3}{\partial x_3} \end{pmatrix} && (7.16) \end{aligned}$$

- *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 $\gamma = 2\varepsilon_{xy}$.

7.4 Constitutive relationships (rheology)

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

  Elastic rheology: $\sigma_{ij} = \lambda\varepsilon_{kk}\delta_{ij} + 2\mu\varepsilon_{ij}$

  Incompressible viscous rheology: $\sigma_{ij} = -p\delta_{ij} + 2\eta\dot{\varepsilon}_{ij}$

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

  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.

7.5 Deriving a closed system of equations for a problem

7.5.1 Conservation laws

Conservation of mass (continuity equation)

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = \frac{\partial \rho}{\partial t} + \frac{\partial (\rho v_i)}{\partial x_i} = 0, \tag{7.17}$$

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

$$\nabla \cdot \boldsymbol{v} = 0 \quad \text{or} \quad \frac{\partial v_i}{\partial x_i} = \sum_{i=1}^{3} \frac{\partial v_i}{\partial x_i} = 0. \tag{7.18}$$

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. (7.18) holds.

Conservation of momentum (equilibrium force balance)

$$\frac{D\boldsymbol{v}}{Dt} = \nabla \sigma + \rho \boldsymbol{g}, \tag{7.19}$$

or

$$\frac{Dv_i}{Dt} = \rho \left(\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{x_j} \right) = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \tag{7.20}$$

where $\boldsymbol{g}$ is gravitational acceleration.

Conservation of energy

$$\left(\frac{\partial E}{\partial t} + v_j \frac{\partial E}{x_j} \right) + \frac{\partial q_i}{\partial x_i} = \rho Q \tag{7.21}$$

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

7.5.2 Thermodynamic relationships

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

$$\boldsymbol{q} = -k \nabla T \tag{7.22}$$

or

$$q_i = -k \frac{\partial T}{\partial x_i} \tag{7.23}$$

where k is the thermal conductivity.

Equation of state 1 ("caloric" equation)

$$E = c_p \rho T \tag{7.24}$$

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

$$\frac{DT}{Dt} = \frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \nabla T = \kappa \nabla^2 T + H \tag{7.25}$$

or

$$\frac{\partial T}{\partial t} + v_j \frac{\partial T}{\partial x_j} = \kappa \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_i} T + H = \kappa \sum_i \frac{\partial^2 T}{\partial x_i^2} + H \tag{7.26}$$

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

$$\kappa = \frac{k}{\rho c_p}. \tag{7.27}$$

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

$$\rho = f(T, p) \tag{7.28}$$

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

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

$$\Delta\rho = \alpha \rho_0 \Delta T, \tag{7.29}$$

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

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

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_i}{\partial x_i} = 0 \tag{7.30}$$

$$\rho \left(\frac{\partial v_i}{\partial t} + v_j \frac{\partial v_i}{x_j} \right) = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \tag{7.31}$$

$$\left(\frac{\partial E}{\partial t} + v_j \frac{\partial E}{x_j} \right) + \frac{\partial q_i}{\partial x_i} = \rho Q \tag{7.32}$$

$$E = c_p \rho T \tag{7.33}$$

$$\rho = f(T, P) \tag{7.34}$$

$$\tilde{\dot{\varepsilon}}_{ij} = R(\tilde{\sigma}_{ij}, \tilde{\sigma}_{ij}) \tag{7.35}$$

$$q_i = -k \frac{\partial T}{\partial x_i} \tag{7.36}$$

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, $f(\ldots)$, ρ_0 , $R(\ldots)$, 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.

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

$$\frac{\partial v_i}{\partial x_i} = 0 \tag{7.37}$$

$$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho_0 g_i = 0 \tag{7.38}$$

$$\rho_0 c_p \left(\frac{\partial T}{\partial t} + v_j \frac{\partial T}{x_j} \right) = \frac{\partial}{\partial x_i} \left(k \frac{\partial T}{\partial x_i} \right) + \rho_0 Q \tag{7.39}$$

$$\tilde{\dot{\varepsilon}}_{ij} = \frac{\tilde{\sigma}_{ij}}{2\eta} \tag{7.40}$$

$$\sigma_{ij} = -p\delta_{ij} + \tilde{\sigma}_{ij} \tag{7.41}$$

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

7.6.2 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):

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z} = 0 \tag{7.42}$$

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z} = 0 \tag{7.43}$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z} - \rho g = 0 \tag{7.44}$$

$$\sigma_{xx} = -p + 2\eta \frac{\partial v_x}{\partial x} \tag{7.45}$$

$$\sigma_{zz} = -p + 2\eta \frac{\partial v_z}{\partial z} \tag{7.46}$$

$$\sigma_{xz} = \eta \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right) \tag{7.47}$$

$$\rho_0 c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} \right) = k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2} \right) + \rho_0 Q \tag{7.48}$$

Chapter 8

Introduction to MATLAB

Reading

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

8.1 Introduction

Matlab is 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 book. Please read through the more comprehensive and verbose Matlab Intro and familiarize yourself with Matlab.

We will assume that your Windows, Mac, or Linux machine has Matlab installed. After starting up the program with the graphical user interface enabled, you will be presented with a number of windows, including an interactive window ("shell") 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.

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

8.2 Useful linear algebra (reprise)

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

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

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:

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

Matrix-vector multiplication:

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

Vector-vector multiplication (dot product):

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

Matrix-matrix multiplication:

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

If you don't know what's going on here, and what the rules for such multiplications are, please consult sec. 6.

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

$$\texttt{Ac} = \texttt{Rhs}$$

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

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

8.3 Exploring MATLAB

8.3.1 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`.

8.3.2 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.

8.3.3 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.*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

8.3.4 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 pop up in a new figure window. 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.

8.3.5 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.

8.3.6 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`

8.3.7 IF command

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

8.3.8 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.

8.3.9 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`

8.3.10 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);
```

8.3.11 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

$$y = \sum_{i=1}^{n} a_i x^{n-1} \tag{8.1}$$

Chapter 9

Example syllabus as USC GEOL540 – 2008

As an example for how the textbook can be used for a one semester course, we provide the syllabus as Numerical Geodynamics was taught at the University of Southern California as GEOL540 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 Matlab problem-set exercises in a computer lab. 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).

(a) **Introduction** (chap. 2)

1.1 Overview of numerical methods in Earth Sciences (sec. 2.1)

1.2 Examples of applications for numerical methods in Earth Sciences (sec. 2.2)

1.3 Computer hardware, Computer Language, Principles of Programming (sec. 2.3)

1.4 Exercise: Matlab programming (sec. 8)

Notes: Introduction Handout, Math Problem set, Matlab

(b) **Ordinary differential equations** (sec. 3)

2.1 Definition of ODEs (sec. 3.1)

2.2 Initial value problems (sec. 3.2)

2.3 Euler method, Taylor expansions, Accuracy of numerical methods, Midpoint method, 4th order Runge Kutta. sec. 3.3)

2.4 Exercise: Program and solve Lorentz equations (end of sec. 3.3).

Notes: ODEs Problem set, ODEs

(c) **Scaling analysis** (sec. 2.4); Non-dimensionalization; Non-dimensional numbers (Rayleigh, Prandtl, Peclet, Reynolds, Deborah). Stokes velocities for Newtonian and non-Newtonian rheology; shear layers.

Notes/problem set: Scaling

(d) **Finite differences I** (sec. 4.1): 1-D heat equation. Explicit solution of diffusion problems. Stability.

Notes/problem set: Explicit FD

(e) **Finite differences II** (sec. 4.3): 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

(f) **Finite differences III** (sec. 4.6): Non-linear equations. Darcy flow equation for pressure-dependent diffusivity. Two-dimensional heat equation, solution with fully explicit and fully implicit methods (sec. 4.7). Comparison with analytical solutions.

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

(g) **Finite differences IV** (sec. 4.9): 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

(h) **Finite elements I** (sec. 5.1): 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

(i) **Finite elements II** (sec. 5.2): 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

(j) **Finite elements III** (sec. 5.5): 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

(k) **Finite elements IV** (sec. 5.7): 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

(l) **Finite elements V** (sec. 5.9 & 5.8): Compressible and incompressible elasticity and Stokes flow. Mixed formulation with discontinuous pressure. Powell-Hestenes iterations.

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

(m) **Joint project work in computer lab.**

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*, *Geophys. Monograph*, vol. 120, edited by J. B. Rundle, D. L. Turcotte, and W. Klein, pp. 5–26, American Geophysical Union, Washington, DC.

Becker, T. W. (2006), On the effect of temperature and strain-rate dependent viscosity on global mantle flow, net rotation, and plate-driving forces, *Geophys. J. Int.*, *167*, 943–957.

Becker, T. W., and C. Faccenna (2009), A review of the role of subduction dynamics for regional and global plate motions, in *Subduction Zone Geodynamics*, edited by F. Funiciello and S. Lallemand, Int. J. Earth Sci., pp. 3–34, Springer.

Becker, T. W., and C. Faccenna (2011), Mantle conveyor beneath the Tethyan collisional belt, *Earth Planet. Sci. Lett.*, *310*, 453–461.

Becker, T. W., and B. Schott (2002), On boundary-element models of elastic fault interaction (abstract), *Eos Trans. AGU*, *83*(47), NG62A–0925.

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.

Browaeys, J., and S. Chevrot (2004), Decomposition of the elastic tensor and geophysical applications, *Geophys. J. Int.*, *159*, 667–678.

Carslaw, H. S., and J. C. Jaeger (1959), *Conduction of Heat in Solids*, 2nd ed., Oxford University Press, London, p. 243.

Christensen, U. R. (1984), Convection with pressure- and temperature-dependent non-Newtonian rheology, *Geophys. J. R. Astr. Soc.*, *77*, 343–384.

Christensen, U. R. (1985), Thermal evolution models for the Earth, *J. Geophys. Res.*, *90*, 2995–3007.

Clayton, R., and H. Engquist (1977), Absorbing boundary conditions for acoustic and elastic wave equations, *Bull. Seismol. Soc. Am.*, *67*, 1529–1540.

Collino, F., and C. Tsogka (2001), Application of the perfectly matched absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media, *Geophysics*, *66*, 294–307.

Crouch, S. L., and A. M. Starfield (1983), *Boundary Element Methods in Solid Mechanics. With Applications in Rock Mechanics*, Allen and Unwin, London.

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, *Geochem., Geophys., Geosys.*, *9*(Q04030), doi:10.1029/2007GC001719.

Dahlen, F. A., and J. Tromp (1998), *Theoretical Global Seismology*, Princeton University Press, Princeton, New Jersey.

Faccenna, C., D. Giardini, P. Davy, and A. Argentieri (1999), Initiation of subduction at Atlantic type margins: Insights from laboratory experiments, *J. Geophys. Res.*, *104*, 2749–2766.

Feigenbaum, M. J. (1978), Quantitative universality for a class of nonlinear transformations, *J. Stat. Phys.*, *19*, 25.

Foley, B., and T. W. Becker (2009), Generation of plate tectonics and mantle heterogeneity from a spherical, visco-plastic convection model, *Geochem., Geophys., Geosys.*, *10*(Q08001), doi:10.1029/2009GC002378.

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.

Gerya, T. V., and D. Yuen (2003), Characteristics-based marker-in-cell method with conservative finite-differences schemes for modeling geological flows with strongly variable transport properties, *Phys. Earth Planet. Inter.*, *140*, 293–318.

Golub, G. H., and C. F. Van Loan (1996), *Matrix computations*, 3 ed., Johns Hopkins University Press.

Gu, J.-C., J. R. Rice, A. L. Ruina, and S. T. Tse (1984), Slip motion and stability of a single degree of freedom elastic system with rate and state dependent friction, *J. Mech. Phys. Solids*, *32*, 167–196.

Hager, B. H., and R. J. O'Connell (1981), A simple global model of plate dynamics and mantle convection, *J. Geophys. Res.*, *86*, 4843–4867.

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.

Jaupart, C., S. Labrosse, and J.-C. Marechal (2007), Temperatures, heat and energy in the mantle of the Earth, in *Treatise on Geophysics*, edited by G. Schubert and D. Bercovici, pp. 253–303, Elsevier.

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.

Kitware, Inc. (2006), Paraview: Parallel Visualization Application, online at http://www.paraview.org/, accessed 06/2006.

Korenaga, J. (2008), Urey ratio and the structure and evolution of Earth's mantle, *Rev. Geophys.*, *46*, doi:10.1029/2007RG000241.

Kwon, Y. W., and H. Bang (1996), *The Finite Element Method Using Matlab*, CRC Press.

Lay, T., J. Hernlund, and B. Buffett (2008), Core-mantle boundary heat flow, *Nature Geosc.*, *1*, 25–32.

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.

Levander, A. R. (1988), Fourth-order finite-difference P-SV seismograms, *Geophysics*, *53*, 1425–1436.

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. A. 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.

Okada, Y. (1992), Internal deformation due to shear and tensile faults in a half-space, *Bull. Seismol. Soc. Am.*, *82*, 1018–1040.

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, *Geochem., Geophys., Geosys., in press*, doi:10.1029/2010GC003081.

Schmeling, H. (1994), *Skriptum: Numerische Methoden in der Geophysik*, Institut für Meteorologie und Geophysik, Universiät Frankfurt am Main.

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, *Geochem., Geophys., Geosys.*, *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.

Index