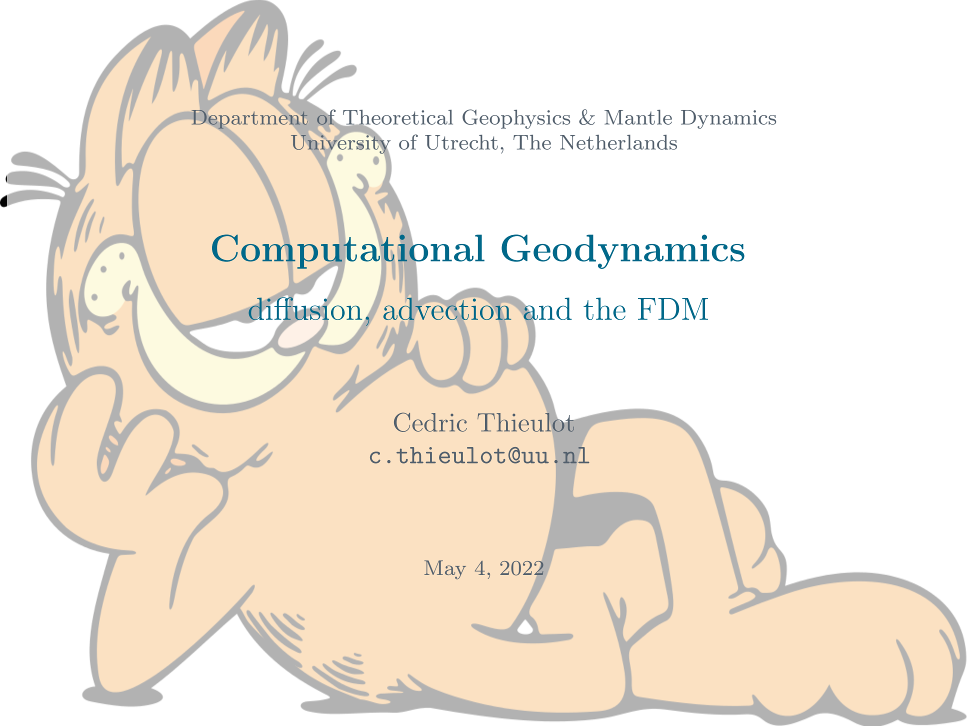


Department of Theoretical Geophysics & Mantle Dynamics
University of Utrecht, The Netherlands

Computational Geodynamics

diffusion, advection and the FDM

Cedric Thieulot
c.thieulot@uu.nl

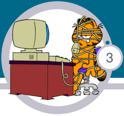
May 4, 2022

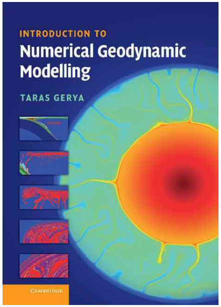
Content

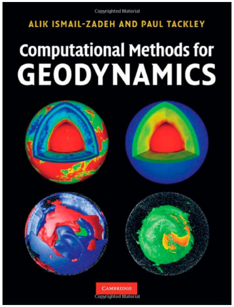
Introduction
Remarks

- many methods in geodynamics (FEM, FDM, FVM, Spectral, ...)
- many languages (C, C++, fortran, python, matlab, ...)
- research codes based on pre-existing libraries
- writing one's own code is fun but
  - modularise & test for robustness
  - strive for portability
  - comment
  - use structures
  - optimise
  - visualise
  - benchmark

Introduction
Sources






Copyrighted Material
ALIK ISMAIL-ZADEH AND PAUL TACKLEY
Computational Methods for GEODYNAMICS
CAMBRIDGE
Copyrighted Material

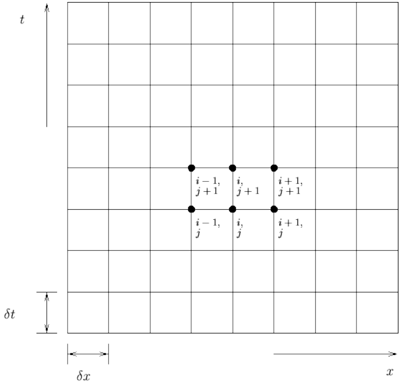
- Becker and Kaus
  `https://earth.usc.edu/~becker/Geodynamics557.pdf`
- Spiegelman
  `http://www.ldeo.columbia.edu/~mspieg/mmm/course.pdf`

The content of this presentation is mostly based on Becker & Kaus.

Finite Difference Method basics

Philosophy

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.



Finite Difference Method basics
Finite differences and Taylor series expansions

- Suppose we have a function $f(x)$, which is continuous and differentiable over the range of interest.
- Let's also assume we know the value $f(x_0)$ and all the derivatives at $x = x_0$.
- The forward Taylor-series expansion for $f(x_0 + h)$, away from the point x_0 by a small amount h gives

$$f(x_0+h) = f(x_0)+h\frac{\partial f}{\partial x}(x_0)+\frac{h^2}{2!}\frac{\partial^2 f}{\partial x^2}(x_0)+\cdots+\frac{h^n}{n!}\frac{\partial^n f}{\partial x^n}(x_0)+\mathcal{O}(h^{n+1})$$

- We can express the first derivative of f by rearranging

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

Finite Difference Method basics

Finite differences and Taylor series expansions

- If we now only compute the first term of this equation as an approximation:
$$\frac{\partial f}{\partial x}(x_i) = \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h^2)$$
where functions $f_i = f(x_i)$ are evaluated at discretely spaced x_i with $x_{i+1} = x_i + h$, where the node spacing, or resolution, h is assumed constant.
- $\mathcal{O}(h^2)$ indicates that the full solution would require additional terms of order h^2, h^3, and so on. $\mathcal{O}$ is called the truncation error: if the distance h is made smaller and smaller, the (numerical approximation) error decreases $\propto h^2$ in this case.
- The **forward FD derivative** as expressed above is called **first order accurate**, and this means that very small h is required for an accurate solution.

Finite Difference Method basics

Finite differences and Taylor series expansions

- We can also expand the Taylor series backward

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

- The **backward FD derivative** then writes:

$$\frac{\partial f}{\partial x}(x_i) = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h^2)$$

Finite Difference Method basics
Finite differences and Taylor series expansions

Introducing the notations:

$$f' = \frac{\partial f}{\partial x} \qquad \text{and} \qquad f'' = \frac{\partial^2 f}{\partial x^2}$$

we can derive higher order derivatives:

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

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

Finite Difference Method basics

Finite differences and Taylor series expansions

- Alternatively, we can form the average of the first order accurate forward and backward schemes, and dividing by two.
- The result is the **central difference** approximation, **second order accurate** of the first derivative

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

Finite Difference Method basics
Finite differences and Taylor series expansions

- By adding the taylor expansions (with $+h$ and $-h$) a **second order accurate** approximation of the second derivative is obtained
$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + \mathcal{O}(h^3)$$
- Another way to arrive at the same expression:

$$f'_{i+1/2} = \frac{f_{i+1} - f_i}{h} \qquad f'_{i-1/2} = \frac{f_i - f_{i-1}}{h}$$

$$f_i'' = \frac{f'_{i+1/2} - f'_{i-1/2}}{h} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$$

Finite Difference Method basics

Derivatives with variable coefficients

Note that derivatives with of the following form

$$\frac{\partial}{\partial x}\left(k\frac{\partial f}{\partial x}\right)$$

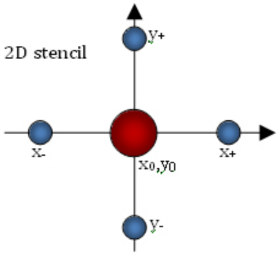
where k is a function of space, should be formed as follows

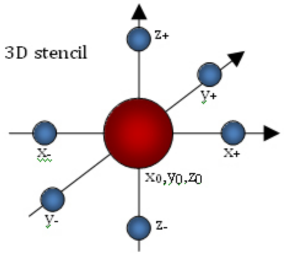
$$\frac{\partial}{\partial x}\left(k\frac{\partial f}{\partial x}\right)\bigg|_i = \frac{k_{i+1/2}\frac{f_{i+1}-f_i}{h} - k_{i-1/2}\frac{f_i-f_{i-1}}{h}}{h} + \mathcal{O}(h^3)$$

where $k_{i\pm 1/2}$ is evaluated between the points to maintain the second order accuracy.

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.

Finite Difference Method basics
Stencils





Stencils for the finite difference Laplacian operator, i.e., the geometric arrangement of points involved in calculating this discrete Laplacian.

Solving the 1D heat equation
Explicit approach

- Consider the one-dimensional, transient (i.e. time-dependent) heat conduction equation without heat generating sources

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

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 a diffusion equation:

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

where $\kappa = k/\rho c_p$ is the heat diffusivity.

Solving the 1D heat equation
Explicit approach

- The derivative of temperature w.r.t. time can be approximated with a forward finite difference approximation as

$$\frac{\partial T}{\partial t} = \frac{T_i^{n+1} - T_i^n}{t^{n+1} - t^n} = \frac{T_i^{n+1} - T_i^n}{\delta t}$$

- n represents the temperature at the current time step whereas $n+1$ represents the new (future) temperature. The subscript i refers to the location.
- Both n and i are integers; n varies from 1 to nstep (total number of time steps) and i varies from 1 to nnx (total number of grid points in x-direction).
- The spatial derivative is replaced by a central FD approximation

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

Solving the 1D heat equation
Explicit approach

- We obtain

$$\frac{T_i^{n+1} - T_i^n}{\delta t} = \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

  and finally

$$T_i^{n+1} = T_i^n + \delta t \; \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

- Because the temperature at the current time step *n* is known, we can compute the new temperature without solving any additional equations.
- Such a scheme is an **explicit** finite difference method and was made possible by the choice to evaluate the temporal derivative with forward differences.

Solving the 1D heat equation
Explicit approach

- In order to solve this equation we need to
  - prescribe an initial temperature field
  - prescribe boundary conditions (T_1 cannot be computed by means of the above equation !)
- We know that this numerical scheme will converge to the exact solution for small h 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 h and δt: any spontaneous perturbations in the solution (such as round-off error) will either be bounded or will decay.

Solving the 1D heat equation
Explicit approach

- The main drawback of the explicit approach is that stable solutions are obtained *only* when

$$0 < \frac{2\kappa\delta t}{h^2} \leq 1$$

  or,

$$\delta t \leq \frac{h^2}{2\kappa}$$

- If this condition is not satisfied, the solution becomes unstable, starts to wildly oscillate and ultimately 'blows up'.
- The stability condition means that the maximum time step needs to be smaller than the time it takes for an anomaly to diffuse across the grid (nodal) spacing *h*.
- The explicit solution is an example of a conditionally stable method that only leads to well behaved solutions if a criterion like the one above is satisfied.

Solving the 1D heat equation
Implicit approach

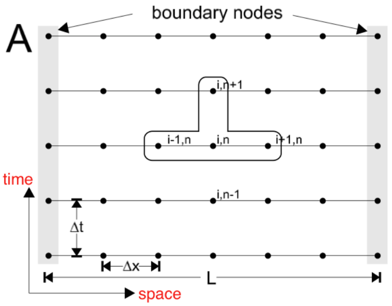
- An alternative approach is an implicit finite difference scheme, where the spatial derivatives of the Laplacian are evaluated (at least partially) at the new time step.
- The simplest implicit discretization of the 1D heat transport equation is

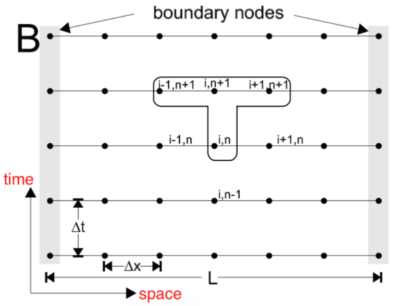
$$\frac{T_i^{n+1} - T_i^n}{\delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2}$$

It is a fully implicit scheme where the time derivative is taken backward.

Solving the 1D heat equation

Explicit vs implicit approach





Explicit FD discretisation

Implicit FD discretisation

Solving the 1D heat equation
Implicit approach

- Let us define

$$s = \frac{\kappa \delta t}{h^2}$$

- The previous equation can be rearranged as follows:

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

- 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.

$\rightarrow$ board: matrix structure and b.c.

Solving the 1D heat equation
Implicit approach

- 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.

Solving the 1D heat equation
Implicit approach

- Looking at

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

and dividing by $-s$ and letting $\delta t \to \infty$, we obtain:

$$T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1} = 0$$

which is a central difference approximation of the steady state solution

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

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

Solving the 1D heat equation
Implicit approach

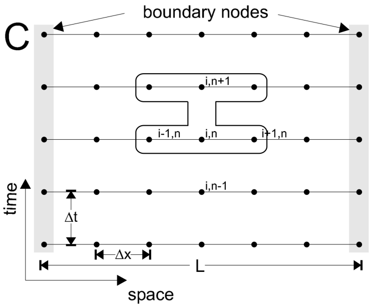
- It turns out that this fully implicit method is second order accurate in space but only first order accurate in time, i.e. the error goes as $\mathcal{O}(h^3, \delta t^2)$.
- It is possible to write down a scheme which is second order accurate both in time and in space (i.e. $\mathcal{O}(h^3, \delta t^3)$), e.g. the Crank-Nicolson scheme which is unconditionally stable.
- The Crank-Nicolson method is the time analog of central spatial differences and is given by

$$\frac{T_i^{n+1} - T_i^n}{\delta t} = \frac{\kappa}{2} \left[\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} + \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2} \right]$$

Solving the 1D heat equation

Implicit approach

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.



Crank-Nicolson stencil

Solving the 1D heat equation
Implicit approach

The implicit approach yields a linear system of the form:

$$\boldsymbol{A} \cdot \vec{T} = \vec{b}$$

where

- $\boldsymbol{A}$ is a (nnx$\times$nnx) matrix,
- $\vec{b}$ is a known vector of size nnx (the 'right-hand side', or rhs)
- $\vec{T}$ the vector of unknowns.

Solving a linear system of equations
Direct solvers

- A general strategy to solve $\boldsymbol{A} \cdot \vec{x} = \vec{b}$ is then LU decomposition:

$$\boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}$$

  where $\boldsymbol{L}$ and $\boldsymbol{U}$ are lower and upper triangular matrices, respectively, which only have zeros in the other part of the matrix.

- The solution of $\boldsymbol{A} \cdot \vec{x} = \vec{b}$ can then be obtained efficiently from

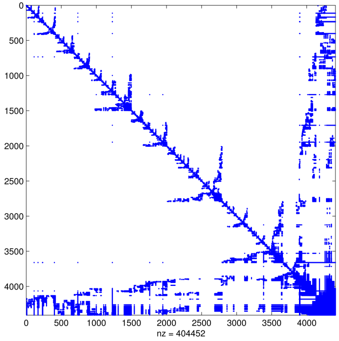
$$\boldsymbol{L} \cdot \boldsymbol{U} \cdot \vec{x} = \vec{b}$$

  by solving $\vec{y} = \boldsymbol{L}^{-1} \cdot \vec{b}$ and then $\vec{x} = \boldsymbol{U}^{-1} \cdot \vec{y}$ because the inverse of $\boldsymbol{L}$ and $\boldsymbol{U}$ are computationally fast to obtain. 'LU' is often how general matrix inversion is implemented on a computer.

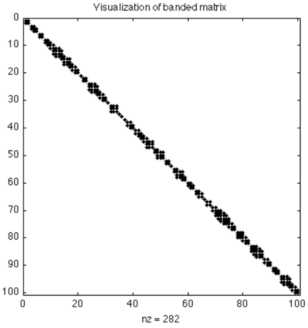
Solving a linear system of equations

Direct solvers

- For most FE/FD problems, the ***A*** matrix will be sparse and banded.



sparse matrix



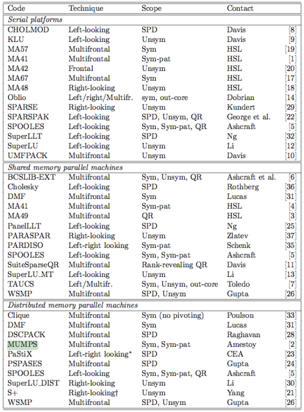
sparse banded matrix

Solving a linear system of equations
Direct solvers

- Special algorithms exist to exploit this feature such that the run time is ideally dominated by the number of non-zero entries of $\boldsymbol{A}$, rather than the full size.
- If $\boldsymbol{A}$ is symmetric and positive definite (i.e. $\vec{x} \cdot \boldsymbol{A} \cdot \vec{x} > 0, \; \forall \vec{x}$), we can use the Cholesky decomposition for which $\boldsymbol{U} = \boldsymbol{L}^T$ and computations are twice as fast as for the general LU case.
- For complex, 3D 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.

Solving a linear system of equations
Direct solvers

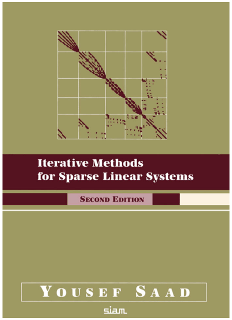
Do not write your own. Do not even try.



Code	Technique	Scope	Contact	
Serial platforms				
CHOLMOD	Left-looking	SPD	Davis	[8]
KLU	Left-looking	Unsym	Davis	[9]
MA57	Multifrontal	Sym	HSL	[19]
MA41	Multifrontal	Sym-pat	HSL	[1]
MA42	Frontal	Unsym	HSL	[20]
MA67	Multifrontal	Sym	HSL	[17]
MA48	Right-looking	Unsym	HSL	[18]
Oblio	Left/right/Multifr.	sym, out-core	Dobrian	[14]
SPARSE	Right-looking	Unsym	Kundert	[29]
SPARSPAK	Left-looking	SPD, Unsym, QR	George et al.	[22]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
SuperLLT	Left-looking	SPD	Ng	[32]
SuperLU	Left-looking	Unsym	Li	[12]
UMFPACK	Multifrontal	Unsym	Davis	[10]
Shared memory parallel machines				
BCSLIB-EXT	Multifrontal	Sym, Unsym, QR	Ashcraft et al.	[6]
Cholesky	Left-looking	SPD	Rothberg	[36]
DMF	Multifrontal	Sym	Lucas	[31]
MA41	Multifrontal	Sym-pat	HSL	[4]
MA49	Multifrontal	QR	HSL	[3]
PanelLLT	Left-looking	SPD	Ng	[25]
PARASPAR	Right-looking	Unsym	Zlatev	[37]
PARDISO	Left-right looking	Sym-pat	Schenk	[35]
SPOOLES	Left-looking	Sym, Sym-pat	Ashcraft	[5]
SuiteSparseQR	Multifrontal	Rank-revealing QR	Davis	[11]
SuperLU_MT	Left-looking	Unsym	Li	[13]
TAUCS	Left/Multifr.	Sym, Unsym, out-core	Toledo	[7]
WSMP	Multifrontal	SPD, Unsym	Gupta	[26]
Distributed memory parallel machines				
Clique	Multifrontal	Sym (no pivoting)	Poulson	[33]
DMF	Multifrontal	Sym	Lucas	[31]
DSCPACK	Multifrontal	SPD	Raghavan	[28]
MUMPS	Multifrontal	Sym, Sym-pat	Amestoy	[2]
PaStiX	Left-right looking*	SPD	CEA	[23]
PSPASES	Multifrontal	SPD	Gupta	[24]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
SuperLU_DIST	Right-looking	Unsym	Li	[30]
S+	Right-looking†	Unsym	Yang	[21]
WSMP	Multifrontal	SPD, Unsym	Gupta	[26]

http://crd-legacy.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf

Solving a linear system of equations
Iterative solvers


Iterative Methods
for Sparse Linear Systems
Second Edition
Yousef Saad
siam

- Stationary iterative methods (Jacobi, Gauss-Seidel, SSOR)
- Krylov subspace methods (conjugate gradients (CG), generalized minimal residual method (GMRES), biconjugate gradient method (BiCG))

Solving a linear system of equations
Iterative solvers

⚠ World of applied maths, computational science and linear algebra.

"The approximating operator that appears in stationary iterative methods can also be incorporated in Krylov subspace methods such as GMRES (alternatively, preconditioned Krylov methods can be considered as accelerations of stationary iterative methods), where they become transformations of the original operator to a presumably better conditioned one. The construction of preconditioners is a large research area."

https://en.wikipedia.org/wiki/Iterative_method

Solving a linear system of equations

Iterative solvers - Jacobi method

- The simplest iterative solution of $\boldsymbol{A}\cdot\vec{x}=\vec{b}$ is the Jacobi method.
- If $\boldsymbol{A}$ is decomposed as $\boldsymbol{A}=\boldsymbol{L}+\boldsymbol{D}+\boldsymbol{U}$ then an iterative solution for $\vec{x}$ starting from an initial guess $\vec{x}_0$ can be obtained from

$$\vec{x}^{k+1}=\boldsymbol{D}^{-1}\left(\vec{b}-(\boldsymbol{L}+\boldsymbol{U})\cdot\vec{x}^{k}\right)$$

- A sufficient (but not necessary) condition for the method to converge is that the matrix $\boldsymbol{A}$ is strictly or irreducibly diagonally dominant. Strict row diagonal dominance means that for each row, the absolute value of the diagonal term is greater than the sum of absolute values of other terms $|a_{ii}|>\sum_{j\neq i}|a_{ij}|$

Note: this algorithm will fail if one or more diagonal terms of $\boldsymbol{A}$ is nul

Solving a linear system of equations
Iterative solvers - Gauss-Seidel method

- If $\boldsymbol{A}$ is decomposed as $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$ then an iterative solution for $\vec{x}$ starting from an initial guess $\vec{x}_0$ can be obtained from

$$\vec{x}^{k+1} = (\boldsymbol{L} + \boldsymbol{D})^{-1} \left(\vec{b} - \boldsymbol{U} \cdot \vec{x}^k \right)$$

- The convergence properties of the Gauss–Seidel method are dependent on the matrix $\boldsymbol{A}$. Namely, the procedure is known to converge if either:
  - $\boldsymbol{A}$ is symmetric positive-definite, or
  - $\boldsymbol{A}$ is strictly or irreducibly diagonally dominant.

  The Gauss–Seidel method sometimes converges even if these conditions are not satisfied.

Solving the 2D heat equation
Explicit approach

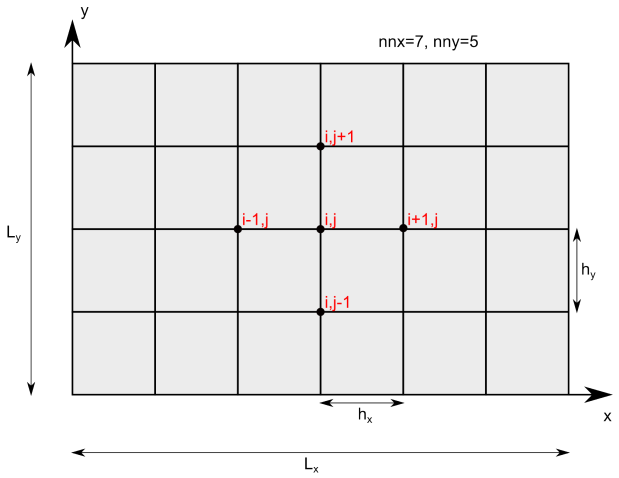
- We now revisit the transient heat equation, this time with sources/sinks for 2D problems:

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

  where Q is the radiogenic heat production.
- If the heat conductivity is constant, it writes:

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

Solving the 2D heat equation
Explicit approach



Solving the 2D heat equation
Explicit approach

The simplest way to discretize the last equation on a domain, e.g. a box with width L_x and height L_y, is to employ an FTCS (forward time, centered space) explicit method like in 1D:

$$\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\delta t} = \kappa \left(\frac{T_{i-1,j}^n - 2T_{i,j}^n + T_{i+1,j}^n}{h_x^2} + \frac{T_{i,j-1}^n - 2T_{i,j}^n + T_{i,j+1}^n}{h_y^2} \right) + \frac{Q_{i,j}^n}{\rho c_p}$$

We define s_x and s_y as follows:

$$s_x = \frac{\kappa \delta t}{h_x^2} \qquad s_y = \frac{\kappa \delta t}{h_y^2}$$

so that

$$T_{i,j}^{n+1} = T_{i,j}^n + s_x (T_{i-1,j}^n - 2T_{i,j}^n + T_{i+1,j}^n) + s_y (T_{i,j-1}^n - 2T_{i,j}^n + T_{i,j+1}^n) + \frac{Q_{i,j}^n \delta t}{\rho c_p}$$

Solving the 2D heat equation
Explicit approach

- The scheme is stable for

$$\delta t \leq \frac{min(h_x^2, h_y^2)}{2\kappa}$$

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

$$T_{i,j} = T_{left} \qquad \forall\ j$$

Solving the 2D heat equation
Implicit approach

If we employ a fully implicit, unconditionally stable discretization scheme as for the 1D exercise:

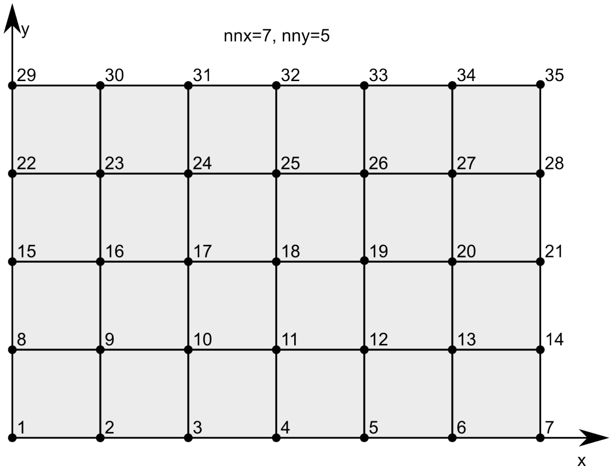
$$\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\delta t} = \kappa \left(\frac{T_{i-1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i+1,j}^{n+1}}{h_x^2} + \frac{T_{i,j-1}^{n+1} - 2T_{i,j}^{n+1} + T_{i,j+1}^{n+1}}{h_y^2} \right) + \frac{Q_{i,j}^{n}}{\rho c_p}$$

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

$$-s_x T_{i+1,j}^{n+1} - s_y T_{i,j+1}^{n+1} + (1+2s_x+2s_y) T_{i,j}^{n+1} - s_x T_{i-1,j}^{n+1} - s_y T_{i,j-1}^{n+1} = T_{i,j}^{n} + \frac{Q_{i,j}^{n}}{\rho c_p}$$

which yields a linear system of equations written $\boldsymbol{A} \cdot \boldsymbol{T} = \boldsymbol{b}$ where $\boldsymbol{A}$ is a $(np \times np)$ matrix.

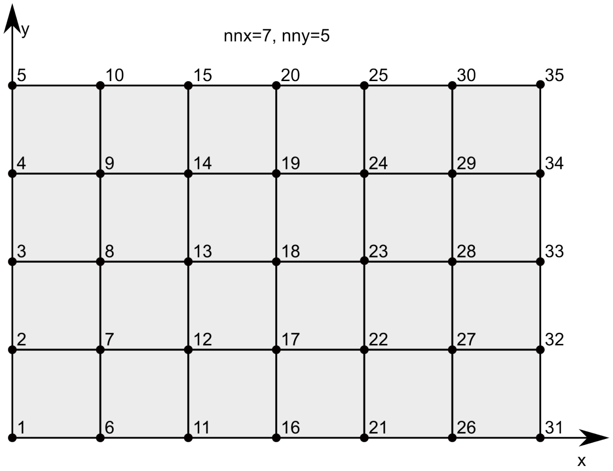
Solving the 2D heat equation
Explicit approach



Numbering scheme for a 2D grid with nnx=7 and nny=5.

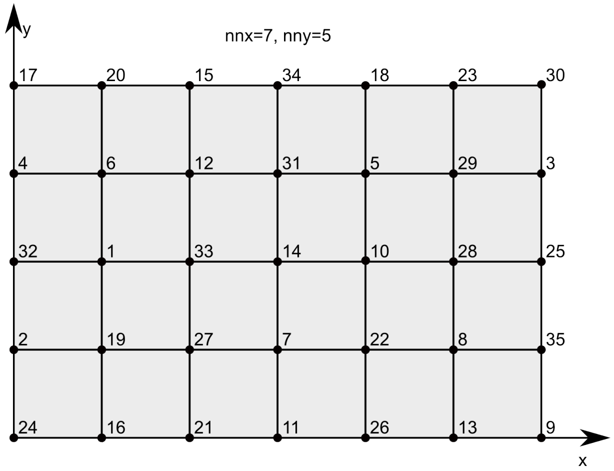
Solving the 2D heat equation

Explicit approach



Alternative numbering scheme for a 2D grid with nnx=7 and nny=5.

Solving the 2D heat equation
Explicit approach



Yet another alternative numbering scheme ...

Solving the 2D heat equation
Explicit approach

- In 2D we need a 'function' which associates to every (i, j) a global index k.
- For the first grid: $1 \leq i \leq 7$, $1 \leq j \leq 5$ so that $1 \leq k \leq 35$

$$k = (j-1) * nnx + i$$

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{i=3,j=4} = \frac{1}{h_x^2}(T_{2,4} - 2T_{3,4} + T_{4,4}) = \frac{1}{h_x^2}(T_{23} - 2T_{24} + T_{25})$$

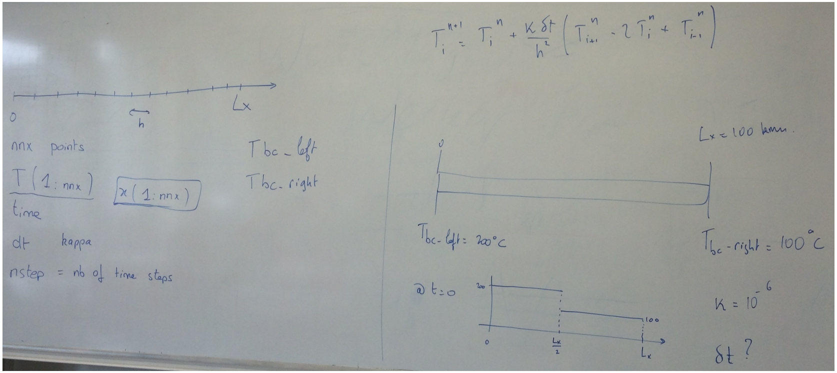
  Note that we now have five diagonals filled with non-zero entries as opposed to three diagonals in the 1D case.
- More generally $1 \leq i \leq nnx$, $1 \leq j \leq nny$ so that $1 \leq k \leq np = nnx * nny$

Other topics

- nonlinear coefficients, e.g. $k = k(T)$
- Neumann boundary conditions (heat flux as b.c.)
- other stencils
- other schemes

Exercises

Ex.1



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

nnx points

$\underline{T(1:nnx)}$ $\boxed{x(1:nnx)}$

time

dt kappa

nstep = nb of time steps

Tbc_left

Tbc_right

$L_x = 100$ km.

T_{bc}-left $= 200°C$

T_{bc}-right $= 100°C$

@ t=0

$\kappa = 10^{-6}$

δt ?

Exercises
Ex.1

Aim: building a 1D code which computes the temperature as a function of time

- explicit vs explicit
- timestep value ?
- use the provided direct solver subroutine
- build your own jacobi solver subroutine
- try Crank-Nicolson
- add a source term Q

Exercises
Ex.2

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

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

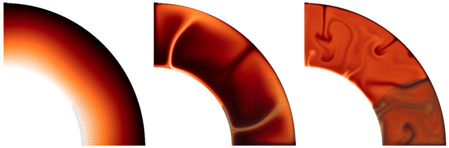
where T_{max} is the maximum amplitude of the temperature perturbation at $(x, y) = (0, 0)$ and σ its half-width. The solution is

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

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.

The advection-diffusion equation
Philosophy

- So far, we mainly focused on the diffusion equation in a non-moving domain (relevant for the case of a dike intrusion or for a lithosphere which remains undeformed).
- we now want to consider problems where material moves during the time period under consideration and takes temperature anomalies with it (e.g. a plume rising through a convecting mantle).
- If the numerical grid remains fixed in the background, the hot temperatures should be moved to different grid points at each time step.



The advection-diffusion equation
Formulation

- in 1D

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

- in 2D/3D

$$\rho c_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} T \right) = \boldsymbol{\nabla} \cdot (k \boldsymbol{\nabla} T) + Q$$

The advection-diffusion equation
Formulation

- in 1D

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

- in 2D/3D

$$\rho c_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} T \right) = \boldsymbol{\nabla} \cdot (k \boldsymbol{\nabla} T) + Q$$

Since temperature variations lead to buoyancy forces, the energy equation is coupled with the Stokes equations from which velocities v can be computed to close the system needed for a convection algorithm.

The advection-diffusion equation
Formulation

- in 1D

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

- in 2D/3D

$$\rho c_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} T \right) = \boldsymbol{\nabla} \cdot (k \boldsymbol{\nabla} T) + Q$$

Since temperature variations lead to buoyancy forces, the energy equation is coupled with the Stokes equations from which velocities v can be computed to close the system needed for a convection algorithm.
The main unknowns are then $(\boldsymbol{v}, p, T)$.

The advection-diffusion equation
"pure" advection

In the absence of diffusion ($k = 0$) we have to solve in 1D:

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = 0$$

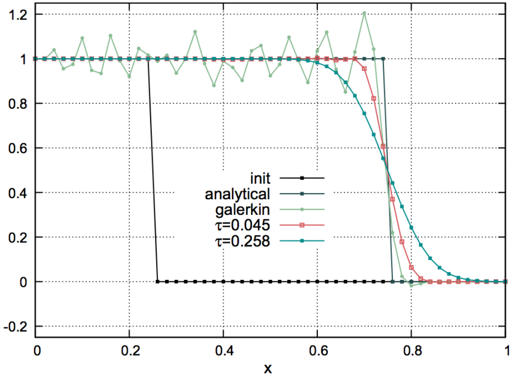
and in 2D:

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = 0$$

- 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.

The advection-diffusion equation
"pure" advection

- 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).



Thieulot, pepi 188, 2011.

The advection-diffusion equation
"pure" advection

- central difference scheme in space, and go forward in time (FTCS scheme):

$$\frac{T_i^{n+1} - T_i^n}{\delta t} = -u_i \frac{T_{i+1}^n - T_{i-1}^n}{2h_x}$$

where u_i is the velocity at location i.

The FTCS method is **unconditionally** unstable !

The advection-diffusion equation
"pure" advection

- central difference scheme in space, and go forward in time (FTCS scheme):

$$\frac{T_i^{n+1}-T_i^n}{\delta t} = -u_i \frac{T_{i+1}^n - T_{i-1}^n}{2h_x}$$

where u_i is the velocity at location i.

The FTCS method is **unconditionally** unstable !
i.e., it blows up for any δt.

The advection-diffusion equation
"pure" advection

- central difference scheme in space, and go forward in time (FTCS scheme):

$$\frac{T_i^{n+1} - T_i^n}{\delta t} = -u_i \frac{T_{i+1}^n - T_{i-1}^n}{2h_x}$$

where u_i is the velocity at location i.

The FTCS method is **unconditionally** unstable !
i.e., it blows up for any δt.
The instability is related to the fact that this scheme produces negative diffusion, which is numerically unstable.

The advection-diffusion equation
"pure" advection

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

$$\frac{T_i^{n+1} - (T_{i+1}^n + T_{i-1}^n)/2}{\delta t} = -u_i \frac{T_{i+1}^n - T_{i-1}^n}{2h_x}$$

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

$$\frac{T_i^{n+1} - (T_{i+1}^n + T_{i-1}^n)/2}{\delta t} = \begin{cases} -u_i \frac{T_i^n - T_{i-1}^n}{h_x} & \text{if} \quad u_i < 0 \\ -u_i \frac{T_{i+1}^n - T_i^n}{h_x} & \text{if} \quad u_i > 0 \end{cases}$$

We have replaced central with forward or backward derivatives, depending on the flow direction.

Exercises
Ex.3

- Program the above FTCS method
- Change the sign of the velocity.
- Change the time step and grid spacing and compute the non-dimensional Courant number $|u|\delta t/h_x$.
- When do unstable results occur? Put differently, can you find a δ small enough to avoid blow-up?
- Program the Lax method by modifying the previous code
- Try different velocities and δt settings and compute the Courant number
- Is the numerical scheme stable for all Courant numbers?
- BONUS: Program the upwind scheme method. Is the numerical scheme stable for all Courant numbers?