Numerical Geodynamics Modelling

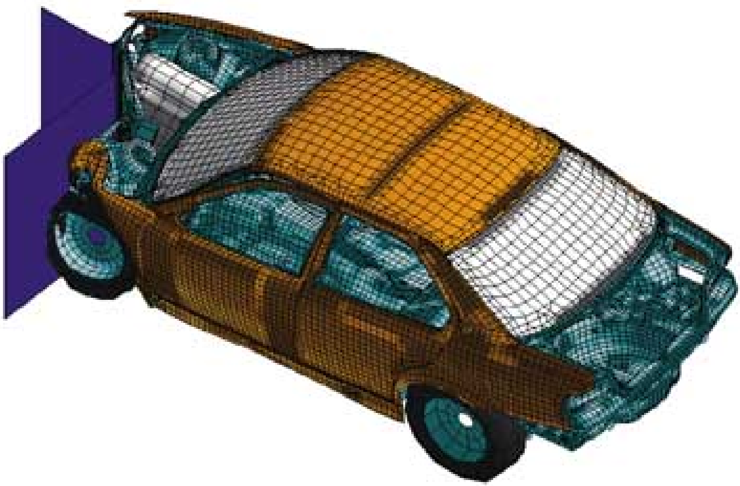
(there is no free lunch)

C. Thieulot (c.thieulot@uu.nl)

March 2014

Kinematical description (1)

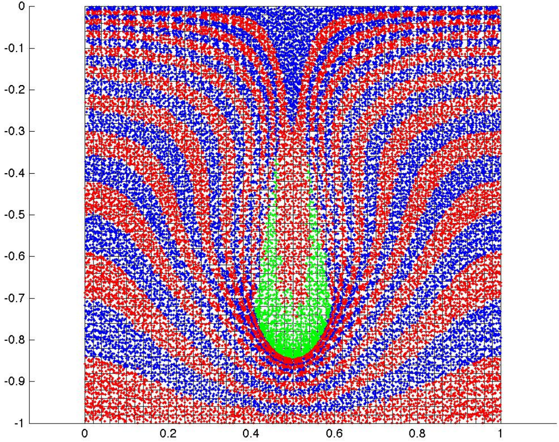
Lagrangian $\rightarrow$ the mesh deforms



$\rightarrow$ Finite Element method

Kinematical description (2)

Eulerian $\rightarrow$ the mesh does not deform

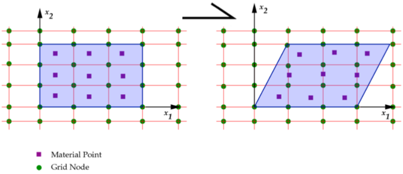


$\rightarrow$ Finite Difference Method, Finite Element Method

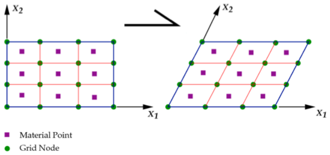
Gerya & Yuen, PEPI, 2007, Braun et al, PEPI, 2008, Jadamec & Billen, JGR, 2012

Kinematical description (2)

In Eulerian methods the mesh is fixed



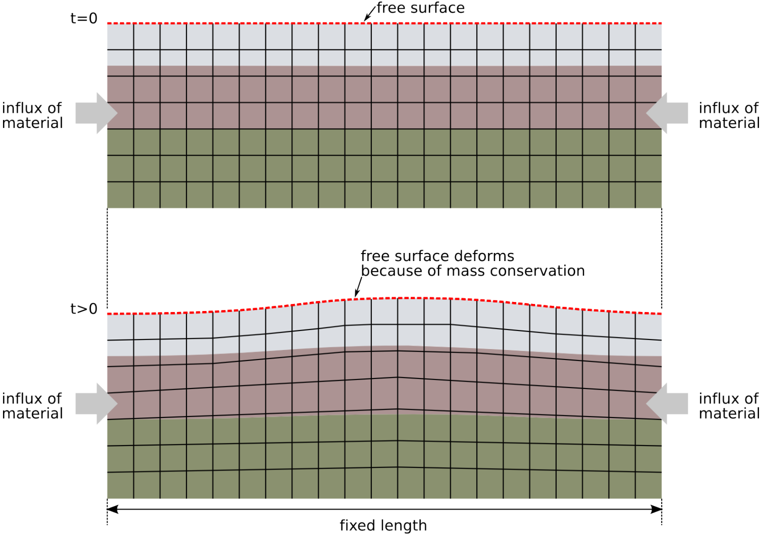
In Lagrangian methods the mesh deforms



	+	−
Lagrangian	Follows surfaces	Needs remeshing
Eulerian	No remeshing	Extra effort to follow surfaces

Kinematical description (3)

Arbitrary Lagrangian-Eulerian → the mesh somewhat deforms

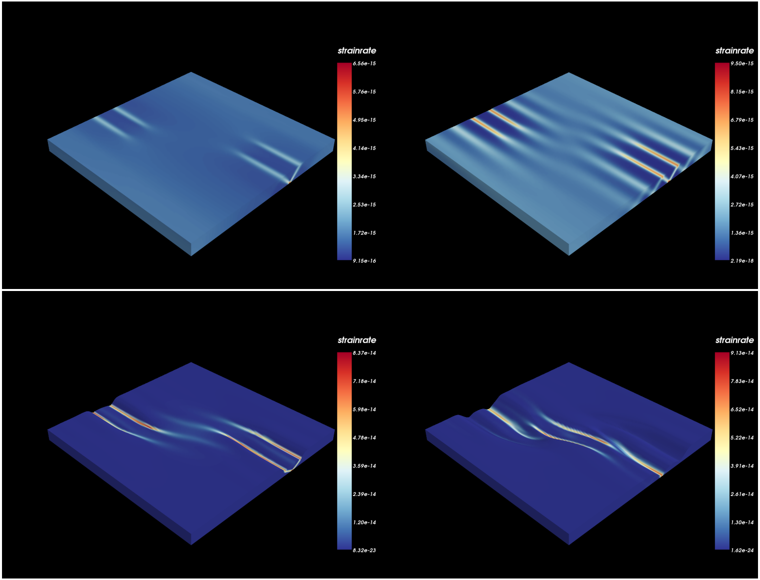


→ Finite Element method

Fullsack, GJI 1995, Thieulot, PEPI 2011

Kinematical description (3)

Arbitrary Lagrangian-Eulerian $\rightarrow$ the mesh somewhat deforms



Allken et al, JGR 2011, G^3 2012

Free surface (1)

At the surface of the Earth, the air layer exerts no stress on the crust $\rightarrow$ free surface.

Free surface (1)

At the surface of the Earth, the air layer exerts no stress on the crust $\rightarrow$ free surface.
There are three essential features needed to properly model free surfaces:

- A scheme is needed to describe the shape and location of a surface,
- An algorithm is required to evolve the shape and location with time
- Free-surface boundary conditions must be applied at the surface.

Free surface (1)

At the surface of the Earth, the air layer exerts no stress on the crust $\rightarrow$ free surface.
There are three essential features needed to properly model free surfaces:

- A scheme is needed to describe the shape and location of a surface,
- An algorithm is required to evolve the shape and location with time
- Free-surface boundary conditions must be applied at the surface.

If no (true) free surface: no topography, no loading, no erosion/sedimentation

Free surface (1)

At the surface of the Earth, the air layer exerts no stress on the crust $\rightarrow$ free surface.
There are three essential features needed to properly model free surfaces:

- A scheme is needed to describe the shape and location of a surface,
- An algorithm is required to evolve the shape and location with time
- Free-surface boundary conditions must be applied at the surface.

If no (true) free surface: no topography, no loading, no erosion/sedimentation

- Lagrangian formulation (or ALE): no special requirement
- Eulerian formulation: the mesh cannot conform to the Earth's surface $\rightarrow$ we need to model the air too.

Free surface (2) - Sticky air

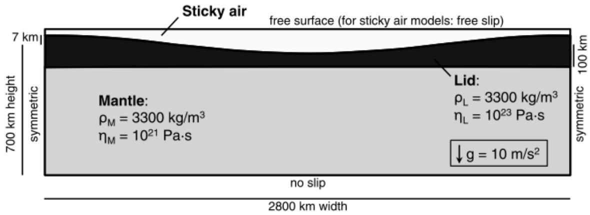
- This method requires the addition of a fluid layer in the model domain.
- pb: air viscosity $< 10^{-5} Pa.s$ vs mantle viscosity $\sim 10^{21} Pa.s$
- air is replaced by a proxy, i.e. a fluid with low density and a sufficiently small viscosity.
- typically, $\mu_{air} = 10^{17-18} Pa.s$.

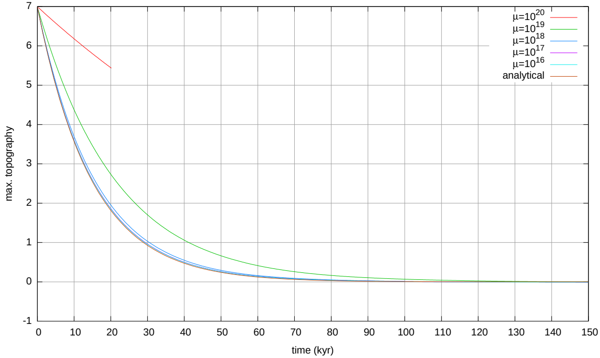
Free surface (2) - Sticky air

- This method requires the addition of a fluid layer in the model domain.
- pb: air viscosity $< 10^{-5} Pa.s$ vs mantle viscosity $\sim 10^{21} Pa.s$
- air is replaced by a proxy, i.e. a fluid with low density and a sufficiently small viscosity.
- typically, $\mu_{air} = 10^{17-18} Pa.s$.

$$\mu_{air} = 10^{18} Pa.s \ ?$$

Free surface (3) - Sticky air





Crameri et al, GJI 189, 2012

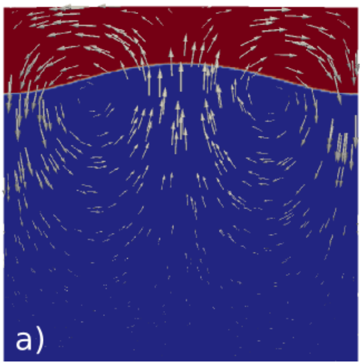
Free surface (4) - stabilisation

Let's relax ... and what about drunken sailors ?

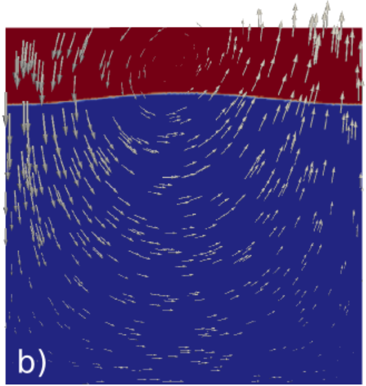
Free surface (4) - stabilisation

Let's relax ... and what about drunken sailors ?

$\Delta\rho = 100kg/m^3$, $500 \times 500km$, sinusoidal amplitude$=5km$



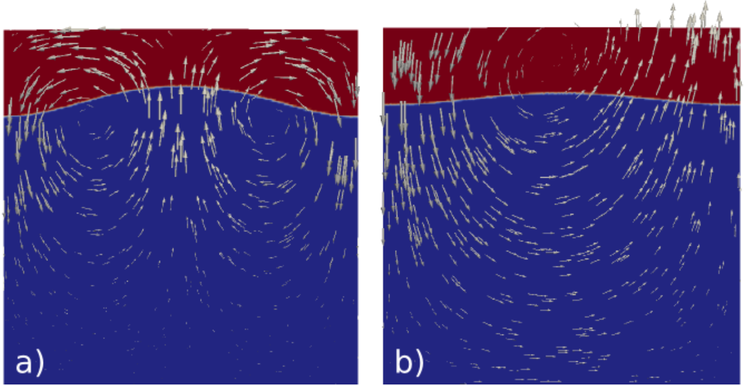
dt small



dt large

Free surface (4) - stabilisation

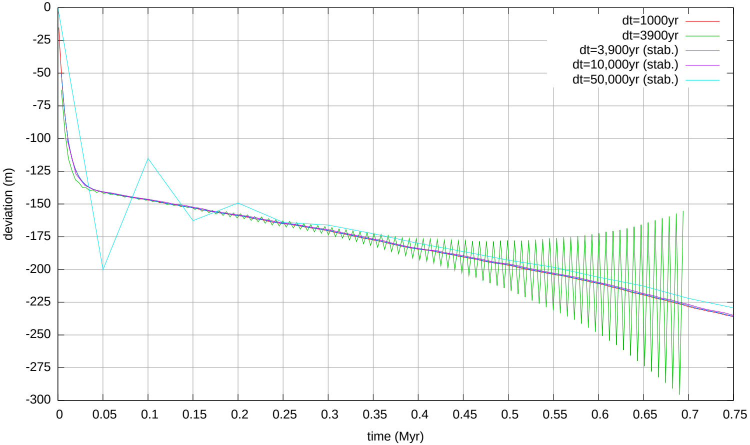
Let's relax ... and what about drunken sailors ?
$\Delta\rho = 100 kg/m^3$, $500 \times 500 km$, sinusoidal amplitude$=5km$



dt small — *dt* large

$\Rightarrow$ Need for stabilisation !

Free surface (5) - stabilisation



Kaus et al, PEPI 181, 2010, Duretz, Gcubed, 2011, Quinquis et al, Tectonophysics 497, 2011

2D vs 3D

Earth is 3D. Why are 99% of all modelling papers 2D ?

2D vs 3D

Earth is 3D. Why are 99% of all modelling papers 2D ?

	2D	3D	ratio
grid	100x100	100x100x100	
# nodes	10^4	10^6	
# dofs	3×10^4	4×10^6	> 100
memory solver	$< 10 Mb$	$\sim 100 Gb$	$> 10^5$
solve time	$\sim 1s$	$1h$	> 1000
# tracers	$5^2 \times 10^4$	$5^3 \times 10^6$	500

$\Rightarrow$ 100-fold increase in memory and computational time
$\hookrightarrow$ optimised code, dedicated methods, parallelism, ...

2D vs 3D

Jadamec & Billen, 2012: *The mesh contains 960 x 648 x 160 elements in the longitudinal, latitudinal, and radial directions, respectively. Models were run using 360 processors on Lonestar, a Linux cluster, for approximately 48 hours per job in models with the composite viscosity and for less time in models with the Newtonian only viscosity.*

2D vs 3D

Jadamec & Billen, 2012: *The mesh contains 960 x 648 x 160 elements in the longitudinal, latitudinal, and radial directions, respectively. Models were run using 360 processors on Lonestar, a Linux cluster, for approximately 48 hours per job in models with the composite viscosity and for less time in models with the Newtonian only viscosity.*
The increasing incorporation of high performance computing and massive data sets into scientific research has led to the need for high fidelity tools to analyze and interpret the information. [...] Immersive 3D visualization facilities provide one approach to fill this gap in the workflow [...]. The open source software 3DVisualizer was used in the Keck Center for Active Visualization in the Earth Sciences (KeckCAVES) for rapid inspection and interactive exploration of the 3D plate boundary geometry and thermal structure output.

2D vs 3D

Jadamec & Billen, 2012: *The mesh contains 960 x 648 x 160 elements in the longitudinal, latitudinal, and radial directions, respectively. Models were run using 360 processors on Lonestar, a Linux cluster, for approximately 48 hours per job in models with the composite viscosity and for less time in models with the Newtonian only viscosity.*
The increasing incorporation of high performance computing and massive data sets into scientific research has led to the need for high fidelity tools to analyze and interpret the information. [...] Immersive 3D visualization facilities provide one approach to fill this gap in the workflow [...]. The open source software 3DVisualizer was used in the Keck Center for Active Visualization in the Earth Sciences (KeckCAVES) for rapid inspection and interactive exploration of the 3D plate boundary geometry and thermal structure output.

Li et al, EPSL 2013: *The Cartesian spatial domain is resolved by 501x341x165 grid points with the resolution of 2x2km in the x-y plane and 4 km in the along-strike z-direction. The lithological structure of the model is represented by a dense grid of about 330 million randomly distributed markers used for advecting various material properties and temperatures.*

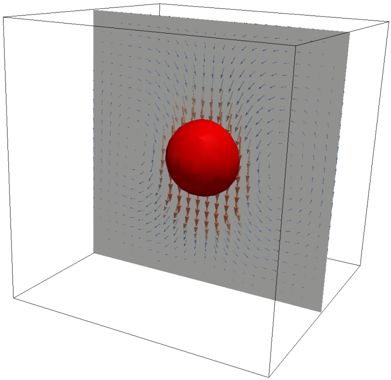
Boundary conditions (1)

- Free slip (flow tangential to boundary) Jadamed & billen, JGR, 2012, Leng & Gurnis, 2011
- No-slip (no flow along the boundary)
- kinematical (precribed velocity) Gurnis et al, Gcubed, 2004
- stress (prescribed stress)
- Open boundaries are implemented by constraining zero tangential velocity on the boundary and by imposing a lithostatic pressure condition for the normal stress on the boundary Chertova et al, 2012.

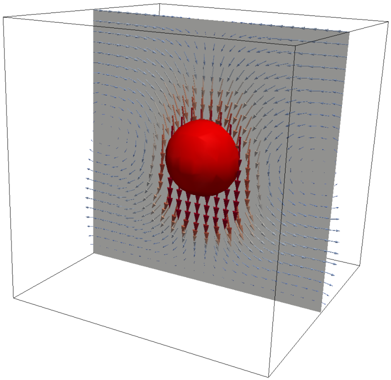
Your model is only as good as the boundary conditions you apply.

Boundary conditions (2) - Open Boundary conditions

$$-\nabla p + \nabla(2\mu\dot{\epsilon}) = \rho g \qquad p = p_{lith} + \delta p$$



free slip side walls



open b.c. side walls

Boundary conditions (3) - Open Boundary conditions

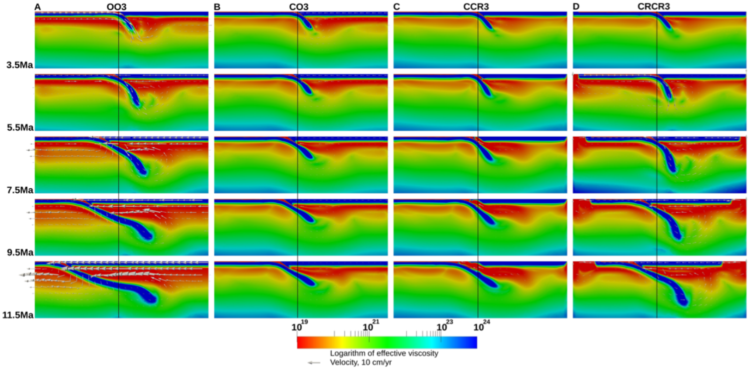
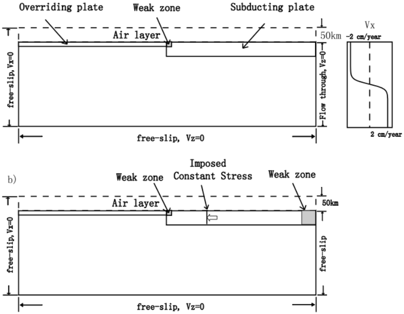


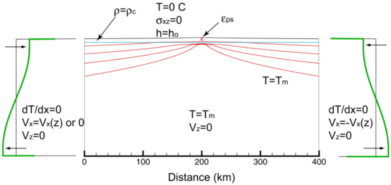
Fig. 4. Evolution of the subduction process for model OO3 with open boundaries, model CO_3 closed left and open right boundary, model CCR3 with closed right and left boundaries with spreading centre on the right boundary and model CRCR3 with closed boundaries. Arrows show the direction and magnitude of flow field. Identical scaling of the velocity vectors applies to all cases.

Chertova et al, Solid Earth 3, 2012

Boundary conditions (4) - In/Outflow

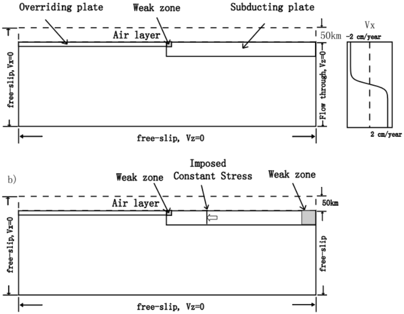


Leng & Gurnis, 2011

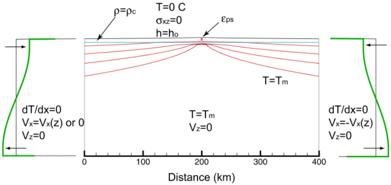


Gurnis et al, 2004

Boundary conditions (4) - In/Outflow



Leng & Gurnis, 2011



Gurnis et al, 2004

Eulerian computational domain + incompressible flow:
$\Rightarrow$ inflow must balance outflow !

The art of benchmarking (1)

- ASPECT $>$ 500,000 lines
- ELEFANT $>$ 100,000 lines
- Complex codes are made of multiple algorithms interacting with each other:
  Solving Stokes Eq + Solving Temp. Eq. + Advecting material + Phase change + brittle-ductile transition +Surface processes + ...

The art of benchmarking (1)

- ASPECT $>$ 500,000 lines
- ELEFANT $>$ 100,000 lines
- Complex codes are made of multiple algorithms interacting with each other:
  Solving Stokes Eq + Solving Temp. Eq. + Advecting material + Phase change + brittle-ductile transition +Surface processes + ...

You <u>need</u> to thoroughly test your code.

The art of benchmarking (1)

- ASPECT $>$ 500,000 lines
- ELEFANT $>$ 100,000 lines
- Complex codes are made of multiple algorithms interacting with each other:
  Solving Stokes Eq + Solving Temp. Eq. + Advecting material + Phase change + brittle-ductile transition +Surface processes + ...

You need to thoroughly test your code.

This process is called *benchmarking*

The art of benchmarking (2)

You have two options

The art of benchmarking (2)

You have two options

1. Run a simulation to which there is an analytical solution and compare the outcome of your code with the analytical solution.

The art of benchmarking (2)

You have two options

1. Run a simulation to which there is an analytical solution and compare the outcome of your code with the analytical solution.
2. Run the same simulation on a variety of codes (preferably using different techniques) and compare outcome.

The art of benchmarking (2)

You have two options

1. Run a simulation to which there is an analytical solution and compare the outcome of your code with the analytical solution.
2. Run the same simulation on a variety of codes (preferably using different techniques) and compare outcome.

Since (1) is not always possible, (2) is widely used:

"A comparison of numerical surface topography calculations: an evaluation of the sticky air method", Crameri et al, GJI 189, 2012

"A community benchmark for 2-D Cartesian compressible convection in the Earths mantle", King et al, GJI 180, 2010

"A comparison of methods for the modeling of thermochemical convection", van Keken, JGR 102, 1997

"The numerical sandbox: comparison of model results for a shortening and an extension experiment", Buiter et al, 2006

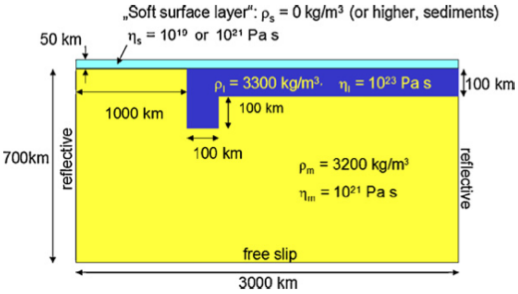
"3D convection at infinite Prandtl number in Cartseian geometry - a benchmark comparison", Busse et al, 1993

"A two- and three-dimensional numerical comparison study of slab detachment", C. thieulot et al, 2014 ?

"A benchmark comparison of spontaneous subduction modelsTowards a free surface", H. Schmeling et al, PEPI 2008

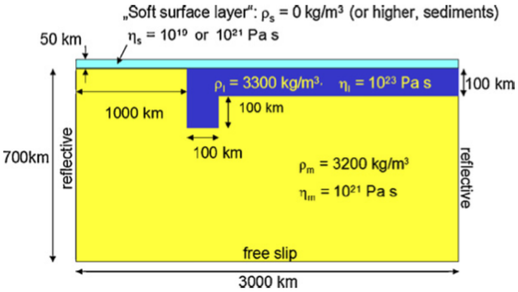
The art of benchmarking (2) - Example

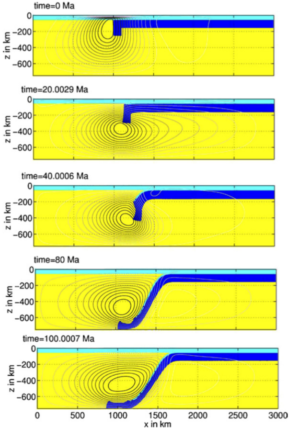
Schmeling et al, PEPI, 2008



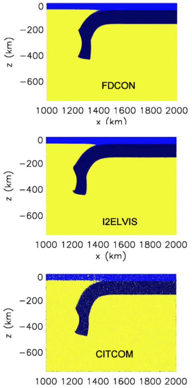
The art of benchmarking (2) - Example

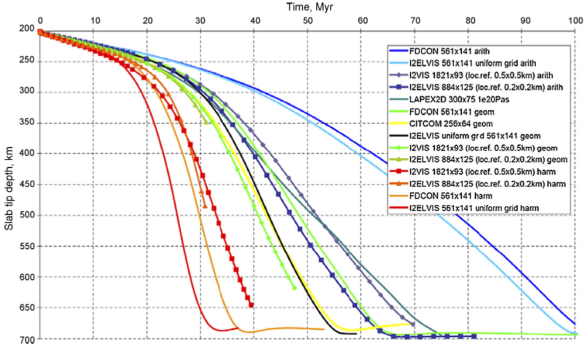
Schmeling et al, PEPI, 2008





The art of benchmarking (2) - Example





Material tracking (1)

- The Earth consists of an upper crust, a middle crust, a lower crust, a lithospheric mantle, an asthenospheric mantle, sediments, melts, ...
  $\Rightarrow$ realistic setups require multiple materials.

Material tracking (1)

- The Earth consists of an upper crust, a middle crust, a lower crust, a lithospheric mantle, an asthenospheric mantle, sediments, melts, ...
  $\Rightarrow$ realistic setups require multiple materials.
- This is not unique to Geodynamics, but very common in CFD too.

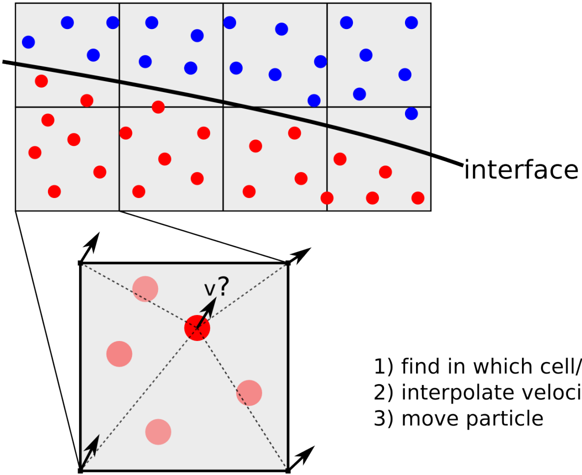
Material tracking (1)

- The Earth consists of an upper crust, a middle crust, a lower crust, a lithospheric mantle, an asthenospheric mantle, sediments, melts, ...
  $\Rightarrow$ realistic setups require multiple materials.
- This is not unique to Geodynamics, but very common in CFD too.
- Multiple methods have been designed over the past decades
  - marker-and-cell (MAC) , Particle-in-Cell (PIC)
    McKee et al, Computers & Fluids, 2008, Gerya book
  - Compositional fields
    ASPECT manual, ConMan code
  - Level set functions
    hillebrand, subm. 2014
  - Particle Level set
    Braun et al, PEPI 2008, Samuel & Evonuk, C3, 2010
  - Marker-Chain
    van Keken et al, JGR 1997
  - all kinds of hybrid methods

None is perfect, none is trivial, none is the best.

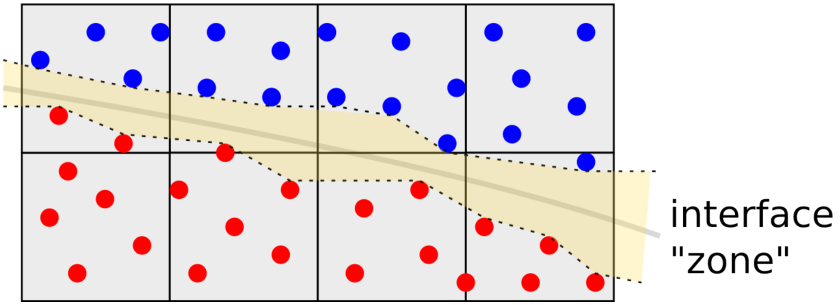
Material tracking (2) - particle/marker advection

Purely Eulerian grid, particles/markers are used to track crustal and lithospheric material.

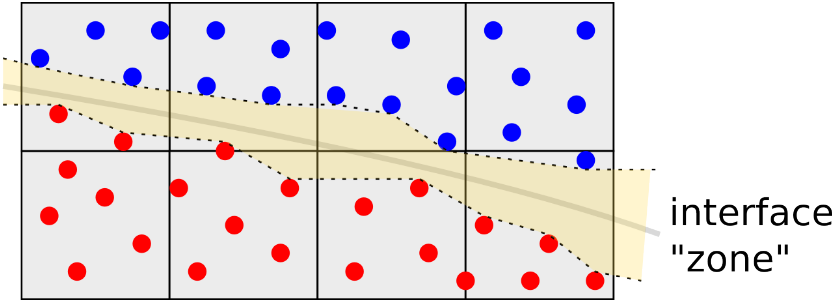


1) find in which cell/elt the particle is
2) interpolate velocity on particle
3) move particle

Material tracking (3) - particle/marker advection



Material tracking (3) - particle/marker advection



If average spacing between particles is $\sim 500m$, free surface is known with $\pm 250m$ precision.

Material tracking (4) - particle/marker advection

Task 1: "Find in which cell/element the particle is"

Assuming a 3D simulation with 100x100x100 grid and 10 particles per cell, doing 1000 timesteps.
$\rightarrow 10^6$ cells , 10^7 particles

Material tracking (4) - particle/marker advection

Task 1: "Find in which cell/element the particle is"

Assuming a 3D simulation with 100x100x100 grid and 10 particles per cell, doing 1000 timesteps.
$\rightarrow 10^6$ cells , 10^7 particles

```
do i=1,nb of timesteps
   do j=1,nb of particles
      do k=1,nb of cells
         Q: is (xj,yj) inside cell k ?
      end do
   end do
end do
```

Material tracking (4) - particle/marker advection

Task 1: "Find in which cell/element the particle is"

Assuming a 3D simulation with 100x100x100 grid and 10 particles per cell, doing 1000 timesteps.
$\rightarrow 10^6$ cells , 10^7 particles

```
do i=1,nb of timesteps
   do j=1,nb of particles
      do k=1,nb of cells
         Q: is (xj,yj) inside cell k ?
      end do
   end do
end do
```

Question `Q` needs to be answered 10^{16} times ...

Material tracking (4) - particle/marker advection

Task 1: "Find in which cell/element the particle is"

Assuming a 3D simulation with 100x100x100 grid and 10 particles per cell, doing 1000 timesteps.
$\rightarrow 10^6$ cells , 10^7 particles

```
do i=1,nb of timesteps
   do j=1,nb of particles
      do k=1,nb of cells
         Q: is (xj,yj) inside cell k ?
      end do
   end do
end do
```

Question `Q` needs to be answered 10^{16} times ...

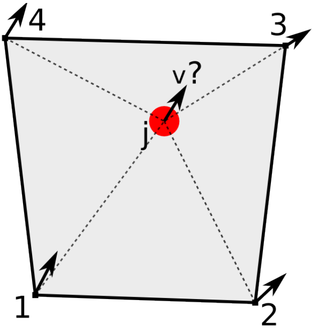
Do not use the force Luke ...

Material tracking (5) - particle/marker advection

Task 2: "interpolate velocity on particle"

Material tracking (5) - particle/marker advection

Task 2: "interpolate velocity on particle"

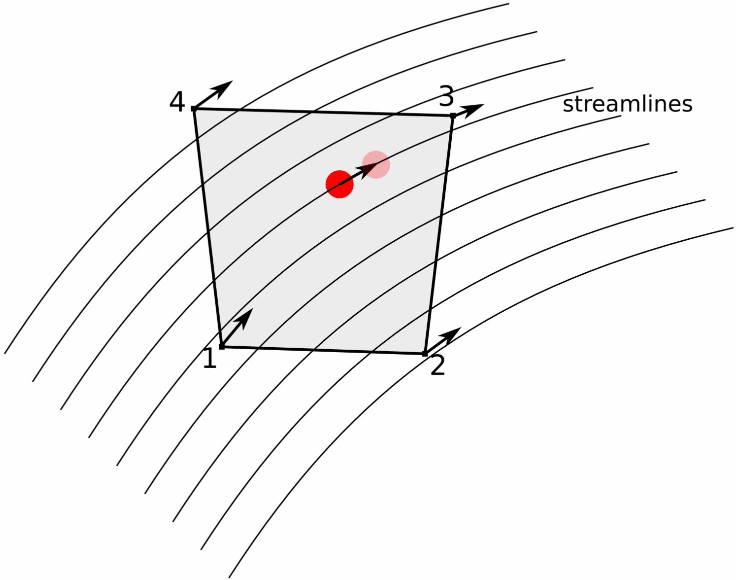


u(xj,yj)=fct(u1,u2,u3,u4, x1,x2,x3,x4,y1,y2,y3,y4)

v(xj,yj)=fct(v1,v2,v3,v4, x1,x2,x3,x4,y1,y2,y3,y4)

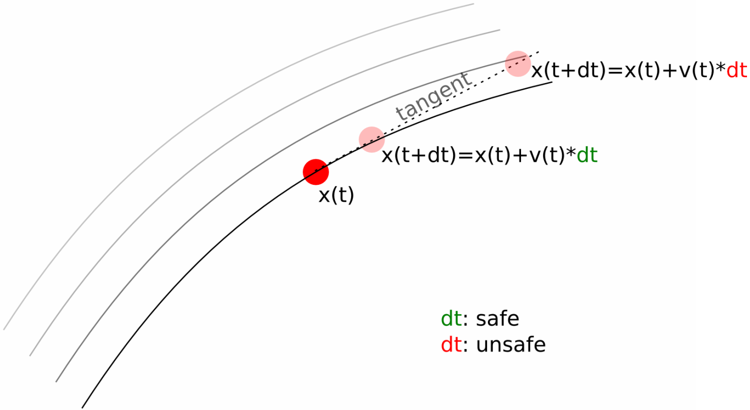
Material tracking (6) - particle/marker advection

Task 3: "move particle with velocity v"



Material tracking (7) - particle/marker advection

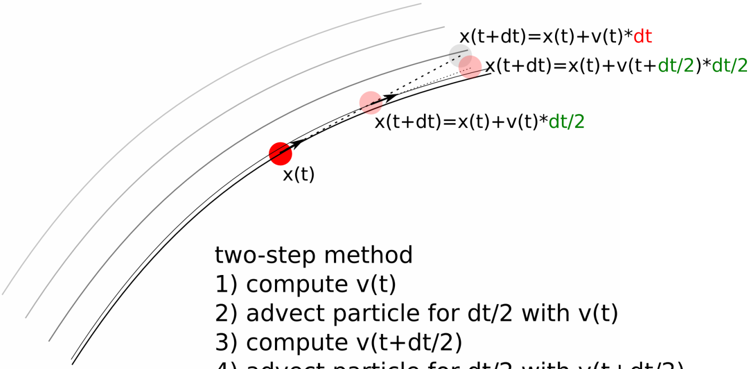
Task 3: "move particle with velocity v"



$\Rightarrow$ small *dt* is better, but increases computational time.

Material tracking (8) - particle/marker advection

Task 3: "move particle with velocity v"



two-step method
1) compute v(t)
2) advect particle for dt/2 with v(t)
3) compute v(t+dt/2)
4) advect particle for dt/2 with v(t+dt/2)

Solvers (1)

- Most methods yield a very large linear system of equations. $N \simeq 10^6 - 10^8$
- Corresponding matrices are *very* sparse (nonzero terms $< 0.001\%$ of matrix terms)
- A solver is a piece of code which solves the system as efficiently as possible (and possibly in parallel)

Solvers (1)

- Most methods yield a very large linear system of equations. $N \simeq 10^6 - 10^8$
- Corresponding matrices are *very* sparse
  (nonzero terms $< 0.001\%$ of matrix terms)
- A solver is a piece of code which solves the system as efficiently as possible (and possibly in parallel)

$\rightarrow$ more than 40 years of research in computer science, applied mathematics, linear algebra, ...

Solvers (1)

- Most methods yield a very large linear system of equations. $N \simeq 10^6 - 10^8$
- Corresponding matrices are *very* sparse (nonzero terms $< 0.001\%$ of matrix terms)
- A solver is a piece of code which solves the system as efficiently as possible (and possibly in parallel)

$\rightarrow$ more than 40 years of research in computer science, applied mathematics, linear algebra, ...
$\rightarrow$ unless it is for educational purposes, do not attempt to write your own solver. Find one that suits your application and use it wisely.

Solvers (1)

- Most methods yield a very large linear system of equations. $N \simeq 10^6 - 10^8$
- Corresponding matrices are *very* sparse (nonzero terms $< 0.001\%$ of matrix terms)
- A solver is a piece of code which solves the system as efficiently as possible (and possibly in parallel)

$\rightarrow$ more than 40 years of research in computer science, applied mathematics, linear algebra, ...
$\rightarrow$ unless it is for educational purposes, do not attempt to write your own solver. Find one that suits your application and use it wisely.

There are two main types of solvers

- Direct
- Iterative

Solvers (2) - Direct Methods

Pros:

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Cons:

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Cons:

- They typically need to build the entire matrix of the system, which means that big systems may not fit in memory and not be usable.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Cons:

- They typically need to build the entire matrix of the system, which means that big systems may not fit in memory and not be usable.
- Memory requirements for the storage of the numerical factor (number of nonzeros in the Cholesky matrix) grow very fast, esp in 3D

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Cons:

- They typically need to build the entire matrix of the system, which means that big systems may not fit in memory and not be usable.
- Memory requirements for the storage of the numerical factor (number of nonzeros in the Cholesky matrix) grow very fast, esp in 3D
- Same observation for the operation count.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Cons:

- They typically need to build the entire matrix of the system, which means that big systems may not fit in memory and not be usable.
- Memory requirements for the storage of the numerical factor (number of nonzeros in the Cholesky matrix) grow very fast, esp in 3D
- Same observation for the operation count.
- Their abstraction ignores/sacrifices the specifics of the problem.

Solvers (2) - Direct Methods

Pros:

- They can solve ill-conditioned matrices (robustness to ill- conditioning).
- They can reuse the factorized matrix and apply it to the solutions of multiple right-hand sides.
- They can be used as black boxes with little or no need for tuning by users.
- They are versatile and application independent, being based on algebra and graph theory rather than on any specific construction of the system of equations.
- They have a high computational intensity and can execute well in hierarchical computer memories.

Cons:

- They typically need to build the entire matrix of the system, which means that big systems may not fit in memory and not be usable.
- Memory requirements for the storage of the numerical factor (number of nonzeros in the Cholesky matrix) grow very fast, esp in 3D
- Same observation for the operation count.
- Their abstraction ignores/sacrifices the specifics of the problem.
- They are harder to parallelize efficiently on a large number of processors

Solvers (2) - Iterative Methods

Solvers (2) - Iterative Methods

- many many variants (CG, GMRES, Jacobi, Gauss-Seidel, ...)

Solvers (2) - Iterative Methods

- many many variants (CG, GMRES, Jacobi, Gauss-Seidel, ...)
- less memory consuming

Solvers (2) - Iterative Methods

- many many variants (CG, GMRES, Jacobi, Gauss-Seidel, ...)
- less memory consuming
- Different physics do require different iterative solver settings, depending on the nature of the governing equation being solved.

Solvers (2) - Iterative Methods

- many many variants (CG, GMRES, Jacobi, Gauss-Seidel, ...)
- less memory consuming
- Different physics do require different iterative solver settings, depending on the nature of the governing equation being solved.
- Contrary to direct solvers, iterative methods approach the solution gradually, rather than in one large computational step.

Solvers (2) - Iterative Methods

- many many variants (CG, GMRES, Jacobi, Gauss-Seidel, ...)
- less memory consuming
- Different physics do require different iterative solver settings, depending on the nature of the governing equation being solved.
- Contrary to direct solvers, iterative methods approach the solution gradually, rather than in one large computational step.
- Therefore, when solving a problem with an iterative method, you can observe the error estimate in the solution decrease with the number of iterations.

Solvers (2) - Iterative Methods

- many many variants (CG, GMRES, Jacobi, Gauss-Seidel, ...)
- less memory consuming
- Different physics do require different iterative solver settings, depending on the nature of the governing equation being solved.
- Contrary to direct solvers, iterative methods approach the solution gradually, rather than in one large computational step.
- Therefore, when solving a problem with an iterative method, you can observe the error estimate in the solution decrease with the number of iterations.
- For well-conditioned problems, this convergence should be quite monotonic. If you are working on problems that are not as well-conditioned, then the convergence will be slower.

Solvers (2) - an example of iterative method

The Gauss-Seidel method is an iterative technique for solving a square system of n linear equations with unknown x:

$$A\mathbf{x} = \mathbf{b}.$$

It is defined by the iteration

$$L_*\mathbf{x}^{(k+1)} = \mathbf{b} - U\mathbf{x}^{(k)},$$

where the matrix A is decomposed into a lower triangular component L_*, and a strictly upper triangular component U: $A = L_* + U$

Solvers (2) - an example of iterative method

The Gauss-Seidel method is an iterative technique for solving a square system of n linear equations with unknown x:

$$A\mathbf{x} = \mathbf{b}.$$

It is defined by the iteration

$$L_*\mathbf{x}^{(k+1)} = \mathbf{b} - U\mathbf{x}^{(k)},$$

where the matrix A is decomposed into a lower triangular component L_*, and a strictly upper triangular component U: $A = L_* + U$
In more detail, write out A, x and b in their components:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Solvers (2) - an example of iterative method

The decomposition of A into its lower triangular component and its strictly upper triangular component is given by:

$$A = L_* + U \qquad \text{where} \qquad L_* = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

The system of linear equations may be rewritten as:

$$L_* \mathbf{x} = \mathbf{b} - U\mathbf{x}$$

The Gauss-Seidel method now solves the left hand side of this expression for x, using previous value for x on the right hand side. Analytically, this may be written as:

$$\mathbf{x}^{(k+1)} = L_*^{-1}(\mathbf{b} - U\mathbf{x}^{(k)}).$$

Solvers (2) - an example of iterative method

The decomposition of A into its lower triangular component and its strictly upper triangular component is given by:

$$A = L_* + U \qquad \text{where} \qquad L_* = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

The system of linear equations may be rewritten as:

$$L_* \mathbf{x} = \mathbf{b} - U\mathbf{x}$$

The Gauss-Seidel method now solves the left hand side of this expression for x, using previous value for x on the right hand side. Analytically, this may be written as:

$$\mathbf{x}^{(k+1)} = L_*^{-1}(\mathbf{b} - U\mathbf{x}^{(k)}).$$

The procedure is generally continued until the changes made by an iteration are below some tolerance, such as a sufficiently small residual.

Code structure

- One or multiple folders containing fortran/C/C++/matlab files
- Makefile/configure file
- Cookbooks
- Post-processing tools

Languages


Perl
JavaScript
PHP
Markup
DOSBatch
Tcl-Tk
LabVIEW
Logic-based
Logo
Visual
Haskell
Occam
Scala
Functional
Simula
Miranda
C++
SQL
.NET
XML
Simkin
Moto
CLU
Pascal
VisualBasic
Euphoria
RPG
Cocoa
Pliant
Python
ObjectiveCaml
Database
BETA
MATLAB
Cobol
Maple
Prototype-based
DataStructured
Erlang
Declarative
Ruby
UML
GLSL
ICI
Turing
ABC
Eiffel
Delphi
Scripting
Multiparadigm
Squeak
Ada
Yorick
Assembly
Mumps
Verilog
CurlyBracket
Groovy
PowerBuilder
S-Lang
Object-Oriented
LaTeX
Clean
Oberon
Scheme
Java
Clojure
ADL
Fortran
IDL
Bistro
Caml
FP
BASIC
HyperCard
Procedural
PostScript
Prolog
Reflective
HTML
Imperative
Interpreted
Distributed
AppleScript
C-sharp
Smalltalk
Befunge
ElasticC
Awk
Pike
Sather
CobolScript
Obfuscated
Lisp

Which code to use and where to get it ?

Which code to use and where to get it ?

- from your wise and enlightened supervisor

Which code to use and where to get it ?

- from your wise and enlightened supervisor
- from a company

Which code to use and where to get it ?

- from your wise and enlightened supervisor
- from a company
- from www.geodynamics.org

  $\rightarrow$ ASPECT code

Which code to use and where to get it ?

- from your wise and enlightened supervisor
- from a company
- from www.geodynamics.org

  $\rightarrow$ ASPECT code
- from free code sources on the internet (www.netlib.org, ...)

Should you write your own code ?

Pros

- Easier to use than commercial/academic software

Should you write your own code ?

Pros

- Easier to use than commercial/academic software
- Taylored to your application

Should you write your own code ?

Pros

- Easier to use than commercial/academic software
- Taylored to your application
- No copyrights problem, nor conflict of interests

Should you write your own code ?

Pros

- Easier to use than commercial/academic software
- Taylored to your application
- No copyrights problem, nor conflict of interests
- Forces you to answer all questions, make decisions, justify choices

Should you write your own code ?

Pros

- Easier to use than commercial/academic software
- Taylored to your application
- No copyrights problem, nor conflict of interests
- Forces you to answer all questions, make decisions, justify choices

Cons

- Wasting time redoing what many people have already done before

Should you write your own code ?

Pros

- Easier to use than commercial/academic software
- Taylored to your application
- No copyrights problem, nor conflict of interests
- Forces you to answer all questions, make decisions, justify choices

Cons

- Wasting time redoing what many people have already done before
- You are not a computer scientist nor an applied mathematician

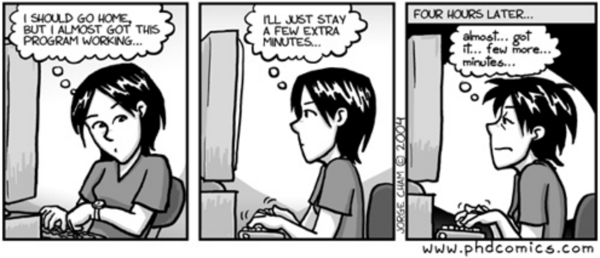
Should you write your own code ?

Pros

- Easier to use than commercial/academic software
- Taylored to your application
- No copyrights problem, nor conflict of interests
- Forces you to answer all questions, make decisions, justify choices

Cons

- Wasting time redoing what many people have already done before
- You are not a computer scientist nor an applied mathematician
- Spending more time debugging (not fun) than coding (fun)

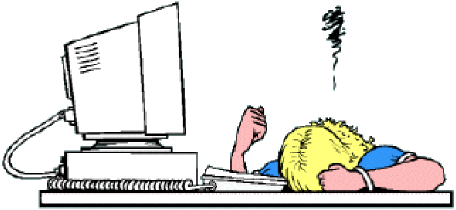


Using a code you did not write

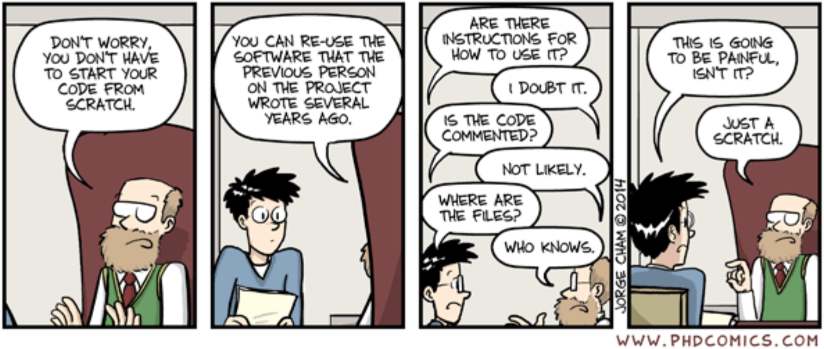
- The manual (heaven or hell ?)
- You need to understand the mindset of its developer(s)
- (useful ?) Cookbooks
- do a few benchmarks
- Existing input files

Using a code you did not write

- The manual (heaven or hell ?)
- You need to understand the mindset of its developer(s)
- (useful ?) Cookbooks
- do a few benchmarks
- Existing input files
- Legacy code: you inherit a code written 20 years ago by your supervisor, in a deprecated language, and consequently modified by 5 generations of phd students ...



Using a code you did not write (2)



I wish I had the time to talk about

- implementation of phase change
- implementation of two phase flow
- rheologies (elasto-visco-plasticity)
- parametrisation & uncertainties
- treatment of nonlinarities
- existing competing codes in the community
- setup design
- lengthscales and timescales

I wish I had the time to talk about

- implementation of phase change
- implementation of two phase flow
- rheologies (elasto-visco-plasticity)
- parametrisation & uncertainties
- treatment of nonlinarities
- existing competing codes in the community
- setup design
- lengthscales and timescales

Let's read the papers and learn some more.

Journals

- GJI: Geophysical Journal International
- JGR: Journal of Geophysical Research
- G3: Geochemistry, Geophysics, Geosystems