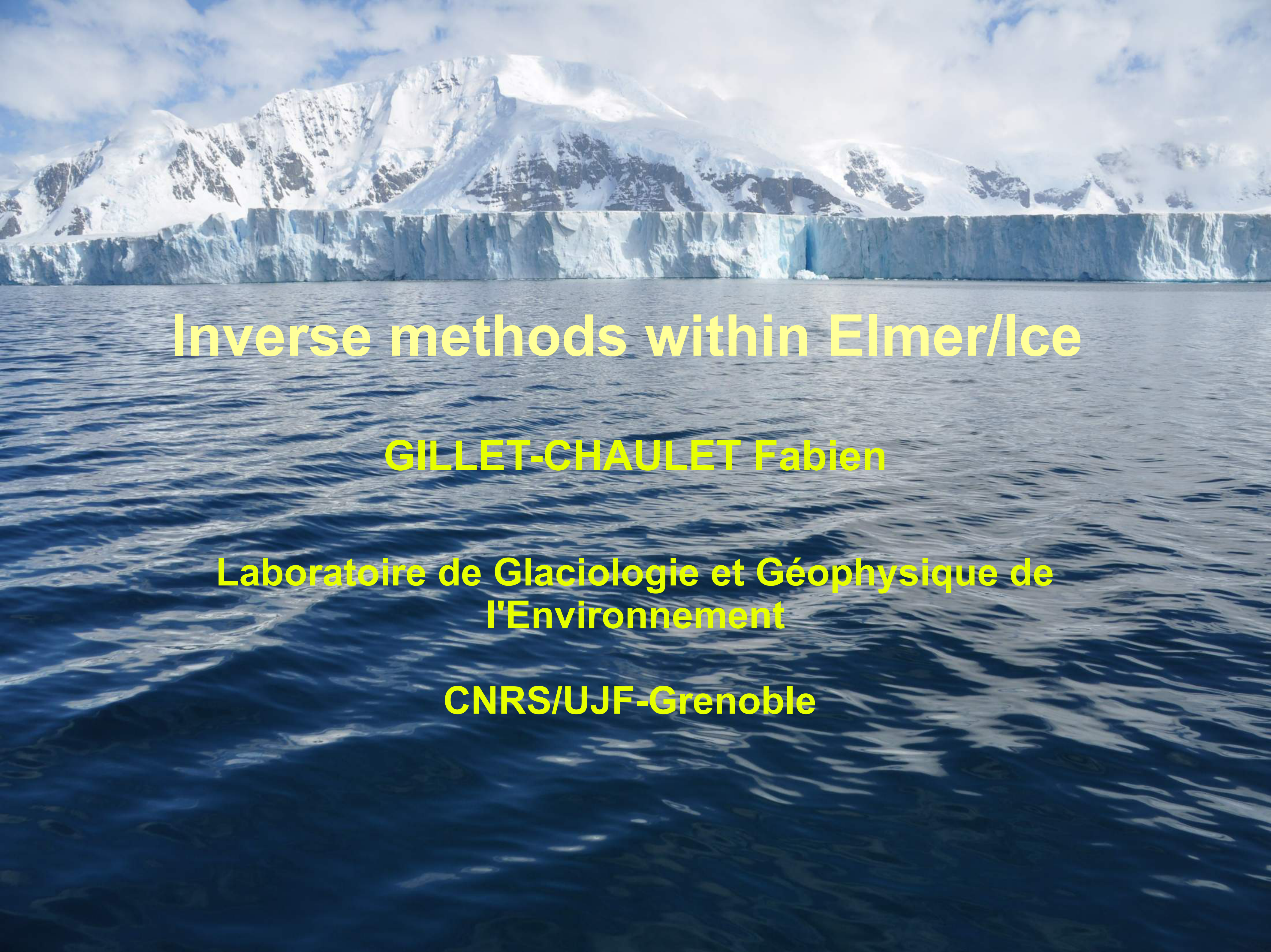


Inverse methods within Elmer/Ice

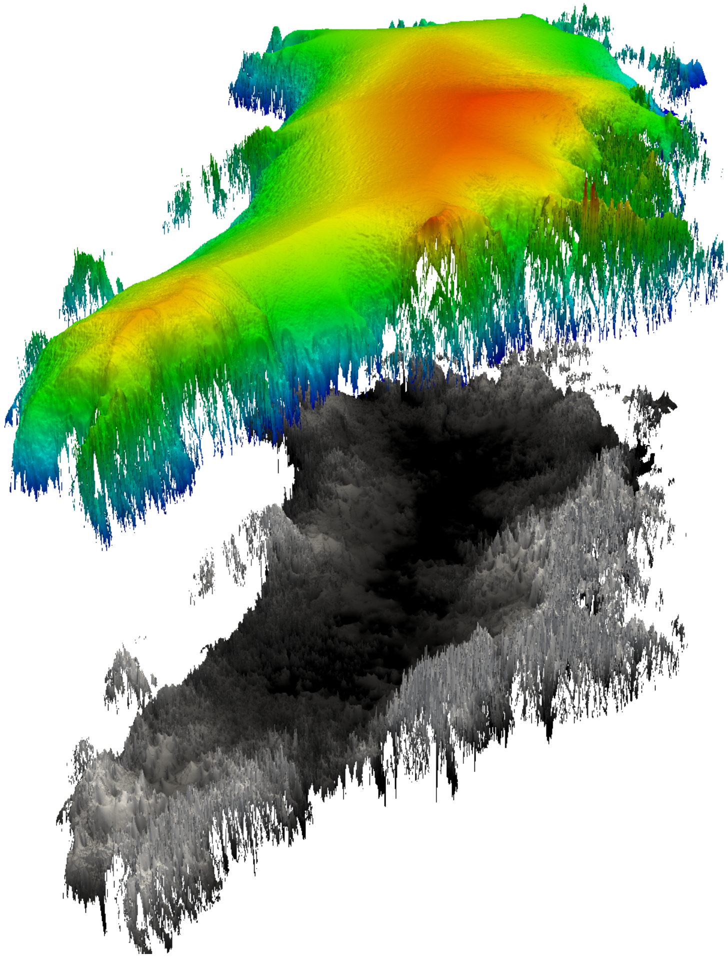
GILLET-CHAULET Fabien

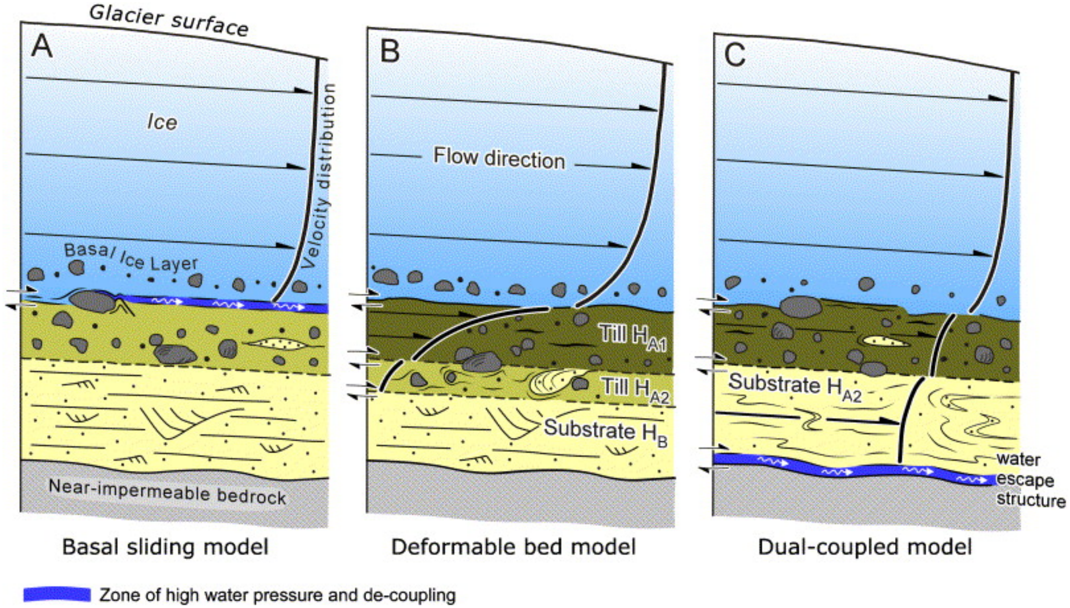
Laboratoire de Glaciologie et Géophysique de l'Environnement

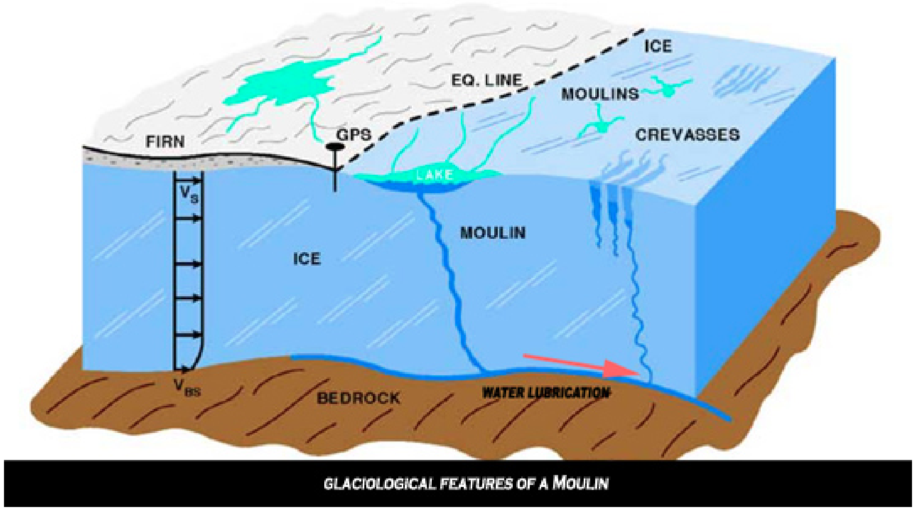
CNRS/UJF-Grenoble

Uncertain parameterisations

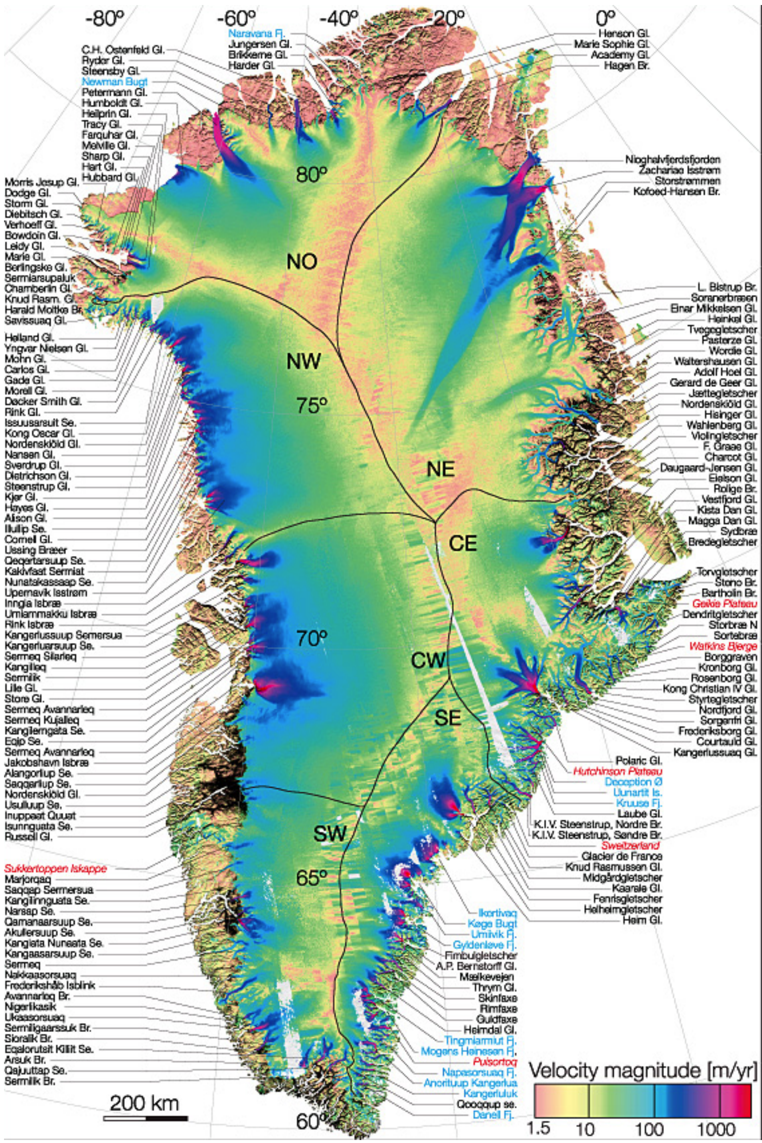
e.g. friction of the ice on the bedrock highly variable in space and time
Usually prescribed as a friction law Tau=f(u)

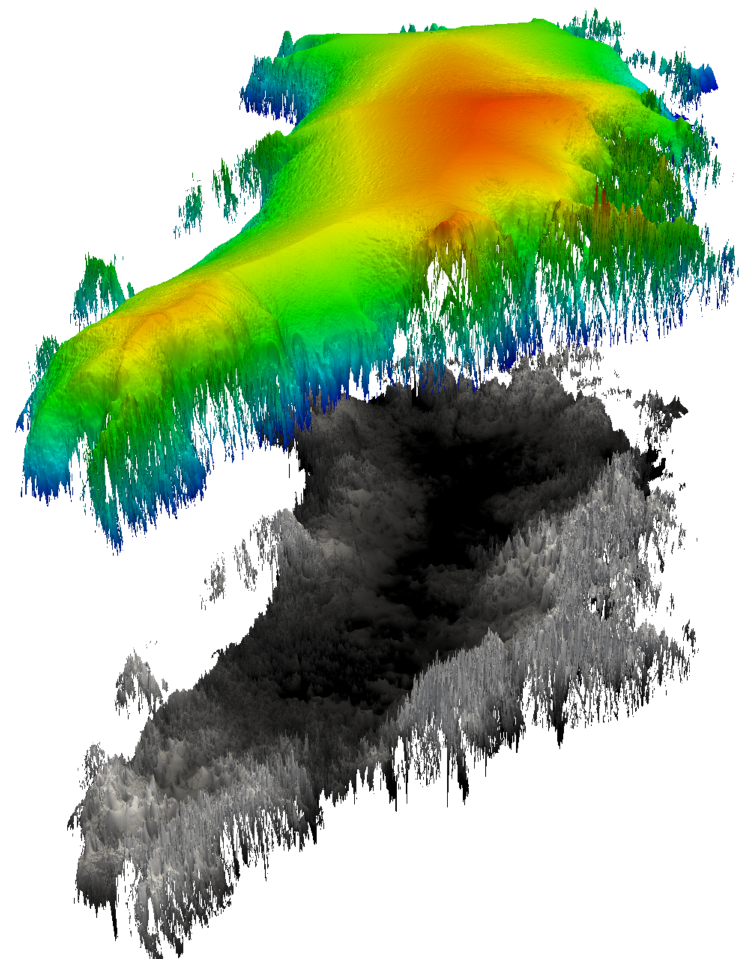




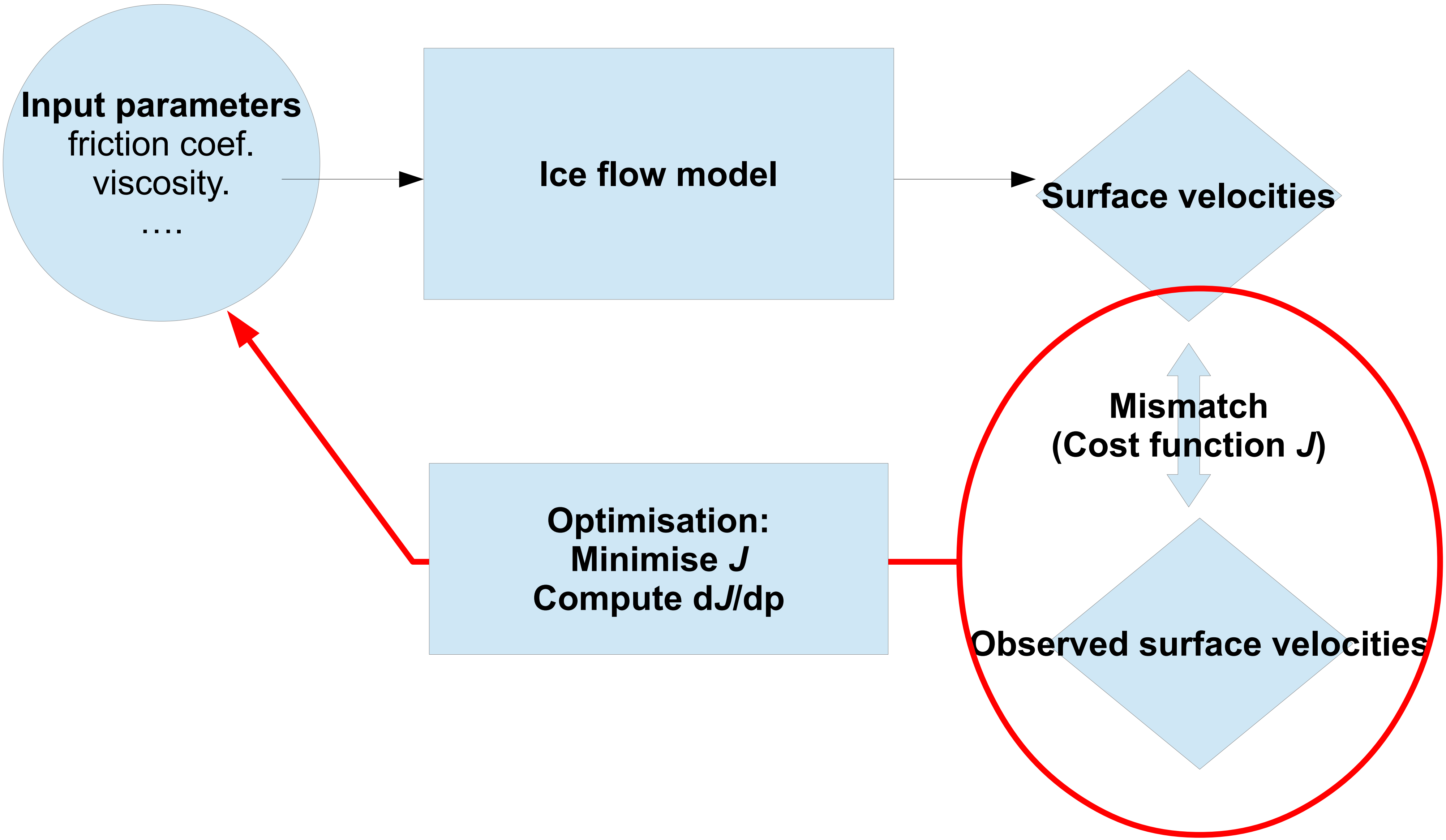


More and more available observations





Inverse modelling: variational data assimilation



Inverse methods in Elmer/Ice

- **2 inverse methods** implemented in **Elmer/Ice**:
  - **Robin inverse method** (arthern and Gudmundsson, 2010)
  - **Adjoint method** (Mac Ayeal, 1993; Morlighem et al., 2010; Petra et al., 2012)

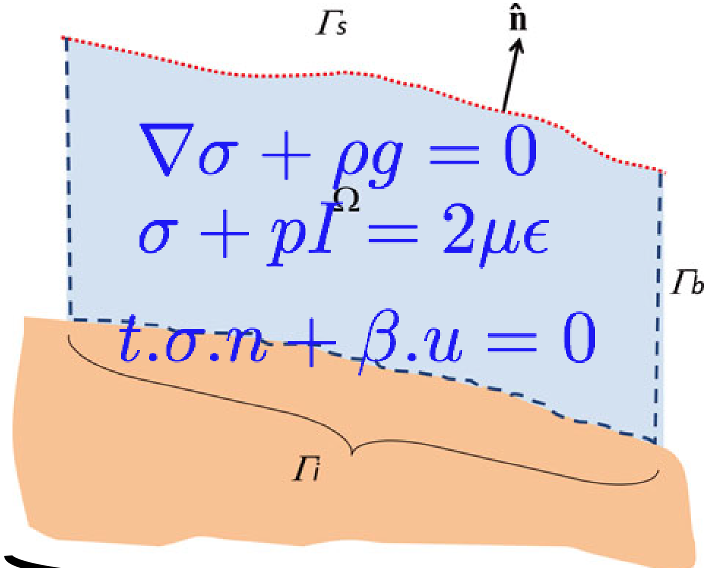
  Characteristics:

  => restricted to **diagnostic** (no time evolution)
  => **slip coefficient** (Linear sliding law)
  => **ice viscosity**
  => could also do Neumann and Dirichlet BC (Adjoint method)

- **Efficient minimisation library** (quasi-Newton algorithm)

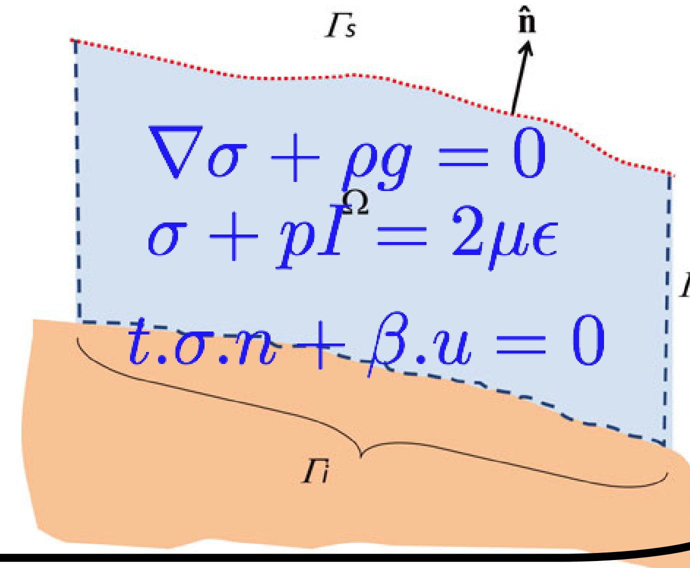
Neumann problem

$\sigma.n = 0$



Dirichlet problem

$u = u^{obs}$



$$J = \int_{\Gamma_s} (u^N - u^D).(\sigma^N - \sigma^D).n d\Gamma$$

$$d_\beta J = \int_{\Gamma_b} \beta'(\|u^D\|^2 - \|u^N\|^2) d\Gamma$$

$$d_\mu J = \int_{\Omega} 2\mu'(\|\epsilon^D\|^2 - \|\epsilon^N\|^2) d\Omega$$

Adjoint method (Mac Ayeal, 1993)

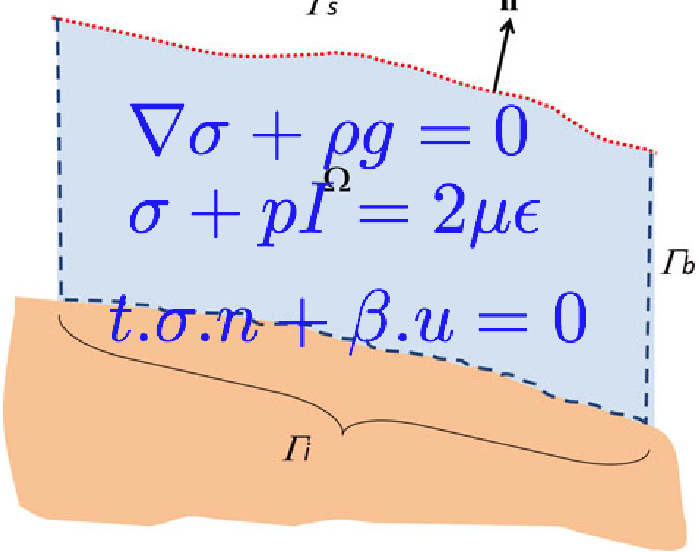
Direct problem

$$\sigma.n = 0$$

$$\nabla\sigma + \rho g = 0$$

$$\sigma + pI = 2\mu\epsilon$$

$$t.\sigma.n + \beta.u = 0$$



1. Define a cost function

$$J = f(u)$$

$$e.g. \quad J = \int_{\Gamma_S} \frac{1}{2}(u - u^{obs})^2 d\Gamma$$

2. Insure that *u* is solution of your problem

$$J' = J(u) + \Lambda(\nabla\sigma + \rho g)$$

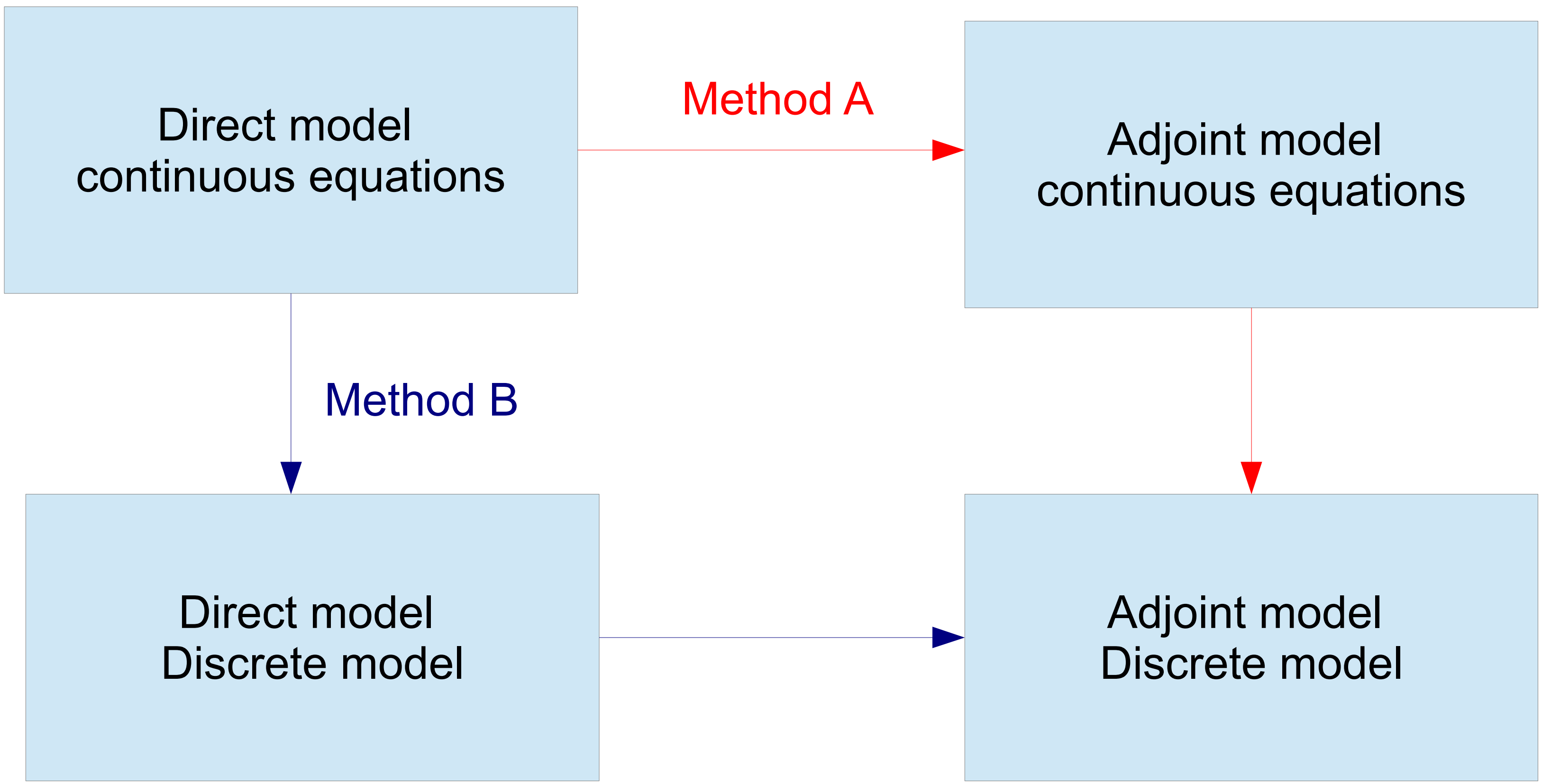
3. Minimisation of *J'* requires that all variations are 0

$d_\Lambda J' = 0$ => direct problem equation is satisfied

$d_u J' = 0$ => adjoint equations

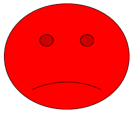
=> gradient of *J* w.r. To input parameters *p* $\quad d_p J = f(\Lambda, u)$

Getting the adjoint model



Usually Method A $\neq$ Method B

Method B should be preferred

Can be done using automatic differentiation  Pointer arrays not yet supported

=> crucial parts have been derived by hand (from Rev 6366)

Inverse method comparison

Robin Inverse method

- Easy to understand/implement
- Only exact for linear viscosity
- Cost function given

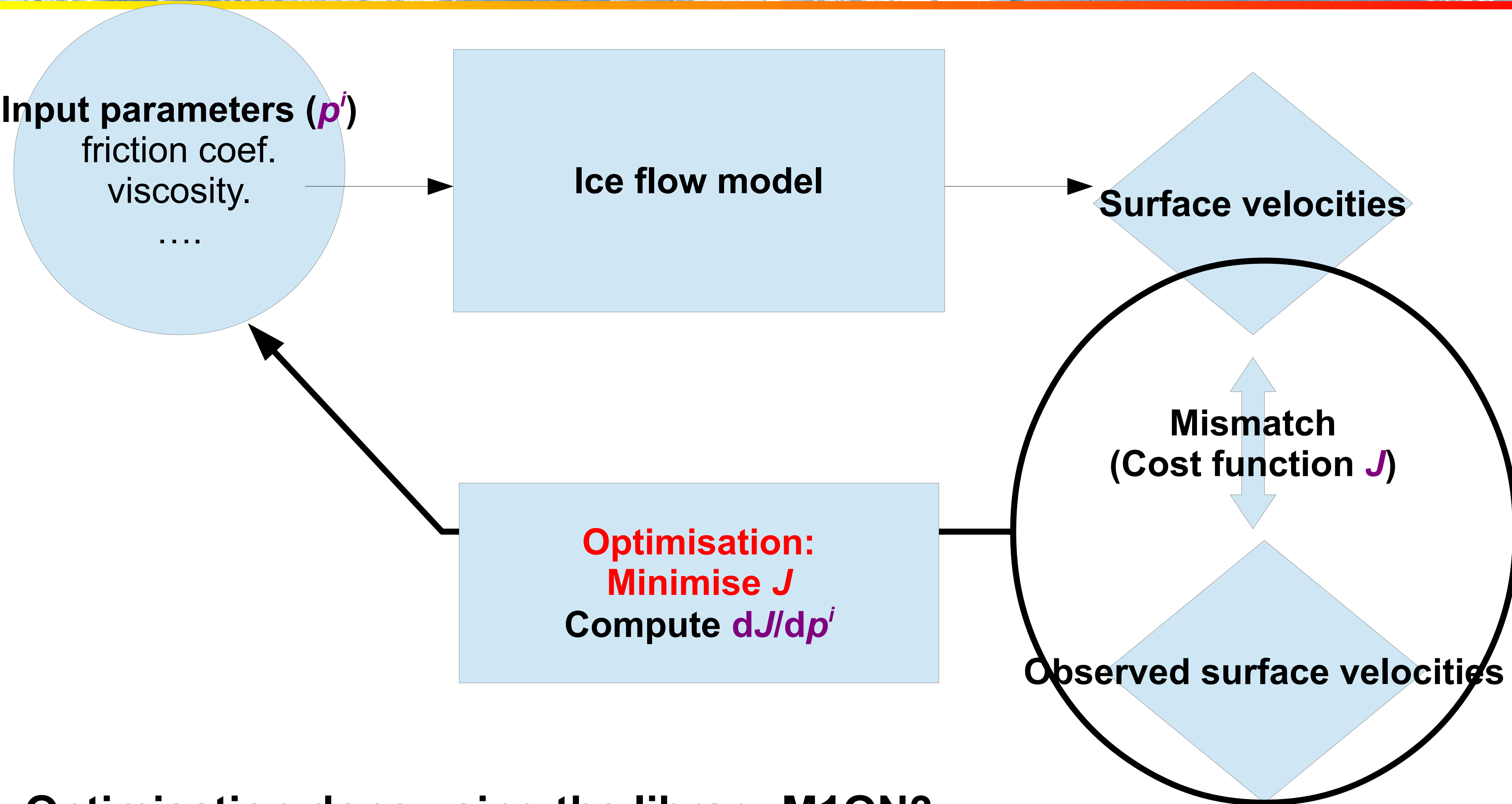
Adjoint method

- Implementation issues
- Remain self-adjoint with non-linear viscosity if solver use newton linearisation (Petra et al.; 2012)
- Cost function can be user-defined

- Some work has been done recently (Rev 6366) to improve the adjoint method.
- When compared with finite differences, gradients obtained with the adjoint method are now more accurate

=> I advise to use the adjoint method from now

Optimisation algorithm: M1QN3



Optimisation done using the library M1QN3:

- Limited memory quasi-newton algorithm
- Implemented in reverse communication (i.e. called by Elmer within a solver)
- Iterative procedure: Input: p^i, J^i, dJ/dp^i – Output p^{i+1}
- https://who.rocq.inria.fr/Jean-Charles.Gilbert/modulopt/optimization-routines/m1qn3/m1qn3.html

Setting up the adjoint method

- Here we will see how to set-up the adjoint method by constructing a "twin" experiment, step by step

- Set-up of the experiment based on Mac Ayeal, 1993

- Finally we will apply it to infer the slip coefficient under the Jacobshavn Isbrae drainage basin

Step 0: Create a reference solution

Domain Geometry

```
$ function zs(tx) {\
  Lx = 200.0e3;\
  Ly = 50.0e03;\
  _zs=500.0-1.0e-03*tx(0)+20.0*(sin(3.0*pi*tx(0)/Lx)*sin(2.0*pi*tx(1)/Ly));\
}

$ function zb(tx) {\
  _zb=zs(tx)-1500.0+2.0e-3*tx(0);\
}
```

Material properties

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Material 1
  Density = Real $rhoi

  Viscosity Model = String "power law"
  Viscosity = Real $ 1.8e8*1.0e-6*(2.0*yearinsec)^(-1.0/3.0)
  Viscosity Exponent = Real $1.0e00/3.0e00
  Critical Shear Rate = Real 1.0e-10
End
```

Boundary Conditions

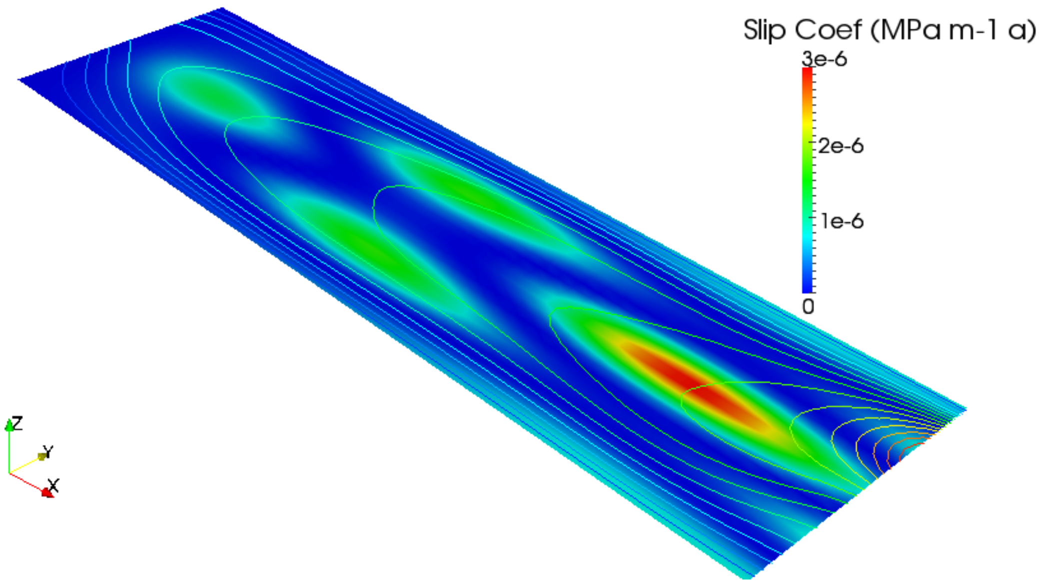
```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Boundary Condition 1
  Name = "Side Walls"
  Target Boundaries(2) = 1 3

  Velocity 1 = Real 0.0
  Velocity 2 = Real 0.0
End

Boundary Condition 2
  Name = "Inflow"
  Target Boundaries = 4

   Velocity 1 = Variable Coordinate 2
      REAL MATC "4.753e-6*yearinsec*(sin(2.0*pi*(Ly-tx)/Ly)+2.5*sin(pi*(Ly-tx)/Ly))"
   Velocity 2 = Real 0.0
End

Boundary Condition 3
  Name = "Front"
  Target Boundaries = 2

   Velocity 1 = Variable Coordinate 2
      REAL MATC "1.584e-5*yearinsec*(sin(2.0*pi*(Ly-tx)/Ly)+2.5*sin(pi*(Ly-tx)/Ly)+0.5*sin(3.0*pi*(Ly-tx)/Ly))"
   Velocity 2 = Real 0.0

End
```

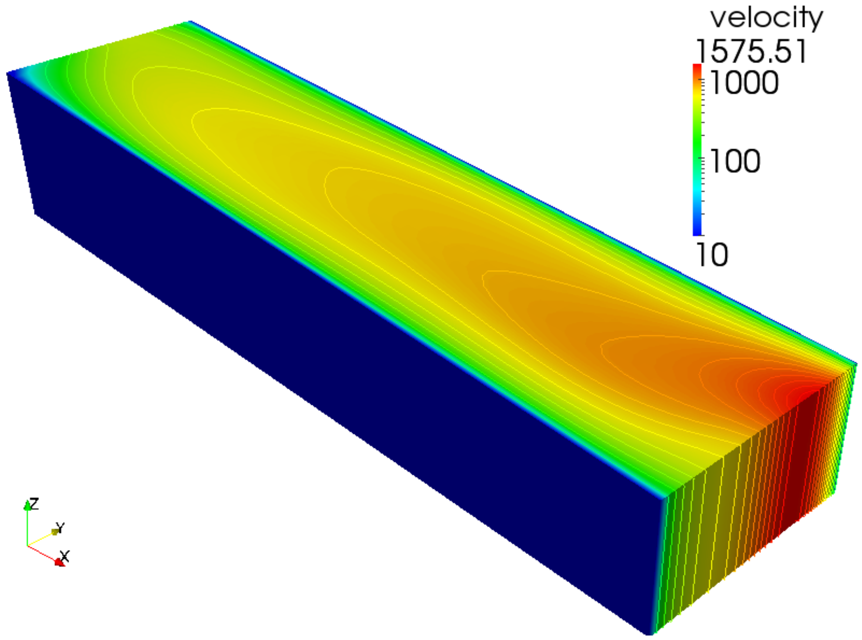
Step 0: Create a reference solution

Reference slip coefficient

```
!Reference Slip Coefficicient used to construct surface velocities
$ function betaSquare(tx) {\
  Lx = 200.0e3;\
  Ly = 50.0e03;\
  yearinsec = 365.25*24*60*60;\
  F1=sin(3.0*pi*tx(0)/Lx)*sin(pi*tx(1)/Ly);\
  F2=sin(pi*tx(0)/(2.0*Lx))*cos(4.0*pi*tx(1)/Ly);\
  beta=5.0e3*F1+5.0e03*F2;\
  _betaSquare=beta*beta/(1.0e06*yearinsec);\
}
```

Ideal observed surface velocities





Step 1: Compute the cost function and the gradient

1. Take an initial guess for the slip coefficient

```
! initial guess for (square root) slip coeff.
  Beta = REAL $ 1.0e3/sqrt(1.0e06*yearinsec)
```

2. Compute the cost function: Cost Solver

```
!!! Compute Cost function
!!!!!!!!   Has to be run before the Adjoint Solver as adjoint forcing is computed here !!!!!
Solver 3

  Equation = "Cost"

!!  Solver need to be associated => Define dumy variable
  Variable = -nooutput "CostV"
  Variable DOFs = 1

  procedure = "ElmerIceSolvers" "CostSolver_Adjoint"


  Cost Variable Name = String "CostValue"  ! Name of Cost Variable

  Optimized Variable Name = String "Beta"  ! Name of Beta for Regularization
  Lambda = Real  $Lambda                   ! Regularization Coef
! save the cost as a function of iterations
  Cost Filename = File "Cost_$name".dat"
end
```

Step 1: Compute the cost function and the gradient

2. Compute the cost function: Boundary conditions

```
! Upper Surface
Boundary Condition 5
  !Name= "Surface" mandatory to compute cost function
  Name = "Surface"

  Save Line = Logical True

  ! Used by StructuredMeshMapper for initial surface topography
  ! here interpolated from a regular DEM
  Top Surface = Variable Coordinate 1
     REAL procedure "Executables/USF_Init" "zsIni"

  ! Definition of the Cost function
  Adjoint Cost = Variable Velocity 1 , Vsurfini 1 , Velocity 2 , Vsurfini 2
    Real MATC "0.5*((tx(0)-tx(1))*(tx(0)-tx(1))+(tx(2)-tx(3))*(tx(2)-tx(3)))"

  !  derivative of the cost function wr u and v
  Adjoint Cost der 1 = Variable Velocity 1 , Vsurfini 1
    Real MATC "tx(0)-tx(1)"
  Adjoint Cost der 2 = Variable Velocity 2 , Vsurfini 2
    Real MATC "tx(0)-tx(1)"

End
```

Step 1: Compute the cost function and the gradient

2. Compute the cost function: Boundary conditions

```
! Upper Surface
Boundary Condition 5
  !Name= "Surface" mandatory to compute cost function
  Name = "Surface"

  Save Line = Logical True

  ! Used by StructuredMeshMapper for initial surface topography
  ! here interpolated from a regular DEM
  Top Surface = Variable Coordinate 1
     REAL procedure "Executables/USF_Init" "zsIni"

  ! Definition of the Cost function
  Adjoint Cost = Variable Velocity 1 , Vsurfini 1 , Velocity 2 , Vsurfini 2
    Real MATC "0.5*((tx(0)-tx(1))*(tx(0)-tx(1))+(tx(2)-tx(3))*(tx(2)-tx(3)))"

  !  derivative of the cost function wr u and v
  Adjoint Cost der 1 = Variable Velocity 1 , Vsurfini 1
    Real MATC "tx(0)-tx(1)"
  Adjoint Cost der 2 = Variable Velocity 2 , Vsurfini 2
    Real MATC "tx(0)-tx(1)"

End
```

$$J = \int_{\Gamma_S} \frac{1}{2}(u - u^{obs})^2 d\Gamma + \lambda \frac{1}{2} \int_{\Gamma_b} (\frac{d\beta}{dx})^2 d\Gamma$$

Hard coded inside the solver,
May be changed to allow
more flexible regularisation,
e.g. *a-priori* estimate

Step 1: Compute the cost function and the gradient

2. Compute the cost function: Boundary conditions

```
! Upper Surface
Boundary Condition 5
  !Name= "Surface" mandatory to compute cost function
  Name = "Surface"

  Save Line = Logical True

  ! Used by StructuredMeshMapper for initial surface topography
  ! here interpolated from a regular DEM
  Top Surface = Variable Coordinate 1
     REAL procedure "Executables/USF_Init" "zsIni"

  ! Definition of the Cost function
  Adjoint Cost = Variable Velocity 1 , Vsurfini 1 , Velocity 2 , Vsurfini 2
    Real MATC "0.5*((tx(0)-tx(1))*(tx(0)-tx(1))+(tx(2)-tx(3))*(tx(2)-tx(3)))"

  !  derivative of the cost function wr u and v
  Adjoint Cost der 1 = Variable Velocity 1 , Vsurfini 1
    Real MATC "tx(0)-tx(1)"
  Adjoint Cost der 2 = Variable Velocity 2 , Vsurfini 2
    Real MATC "tx(0)-tx(1)"

End
```

Used to compute the **forcing term of the adjoint system** (part differentiated by hand)

Step 1: Compute the cost function and the gradient

3. Compute the Adjoint solution

```
!!!!  Adjoint Solution
Solver 4

  Equation = "Adjoint"
  Variable = Adjoint
  Variable Dofs = 4

  procedure = "ElmerIceSolvers" "AdjointSolver"

!Name of the flow solution solver
  Flow Solution Equation Name = string "Navier-Stokes"

  Linear System Solver = Iterative
   Linear System Iterative Method = GMRES
   Linear System GMRES Restart = 100
   Linear System Preconditioning= ILU0
   Linear System Convergence Tolerance= 1.0e-12
   Linear System Max Iterations = 1000
End
```

- **Take the last NS bulk matrix**
- **Apply BC**
- **Solve**

This part has not been differentiated

Step 1: Compute the cost function and the gradient

```
.......................................
Boundary Condition 1
  Name = "Side Walls"
  Target Boundaries(2) = 1 3

!Dirichlet BC

  Velocity 1 = Real 0.0
  Velocity 2 = Real 0.0

!Dirichlet BC => Dirichlet = 0 for Adjoint
  Adjoint 1 = Real 0.0
  Adjoint 2 = Real 0.0
End

Boundary Condition 2
  Name = "Inflow"
  Target Boundaries = 4

   Velocity 1 = Variable Coordinate 2
      REAL MATC "4.753e-6*yearinsec*(sin(2.0*pi*(Ly-tx)/Ly)+2.5*sin(pi*(Ly-tx)/Ly))"
   Velocity 2 = Real 0.0

!Dirichlet BC => Dirichlet = 0 for Adjoint
  Adjoint 1 = Real 0.0
  Adjoint 2 = Real 0.0
End

Boundary Condition 3
  Name = "Front"
  Target Boundaries = 2

   Velocity 1 = Variable Coordinate 2
      REAL MATC "1.584e-5*yearinsec*(sin(2.0*pi*(Ly-tx)/Ly)+2.5*sin(pi*(Ly-tx)/Ly)+0.5*sin(3.0*pi*(Ly-tx)/Ly))"
   Velocity 2 = Real 0.0

!Dirichlet BC => Dirichlet = 0 for Adjoint
  Adjoint 1 = Real 0.0
  Adjoint 2 = Real 0.0
End
```

Step 1: Compute the cost function and the gradient

```
Boundary Condition 4
  !Name= "bed" mandatory to compute regularistaion term of the cost function (int (dbeta/dx) 2)
  Name = "bed"
  !Body Id used to solve
  Body ID = Integer 2

  Save Line = Logical True

  Bottom Surface = Variable Coordinate 1
    REAL   procedure "Executables/USF_Init" "zbIni"

  Normal-Tangential Velocity = Logical True
  Normal-Tangential Adjoint = Logical True

  Adjoint Force BC = Logical True

  Velocity 1 = Real 0.0e0
  Adjoint 1 = Real 0.0e0

  Slip Coefficient 2 = Variable Beta
     REAL MATC "tx*tx"

  Slip Coefficient 3 = Variable Beta
     REAL MATC "tx*tx"

End
```

Step 1: Compute the cost function and the gradient

4. Compute the gradient of the cost function

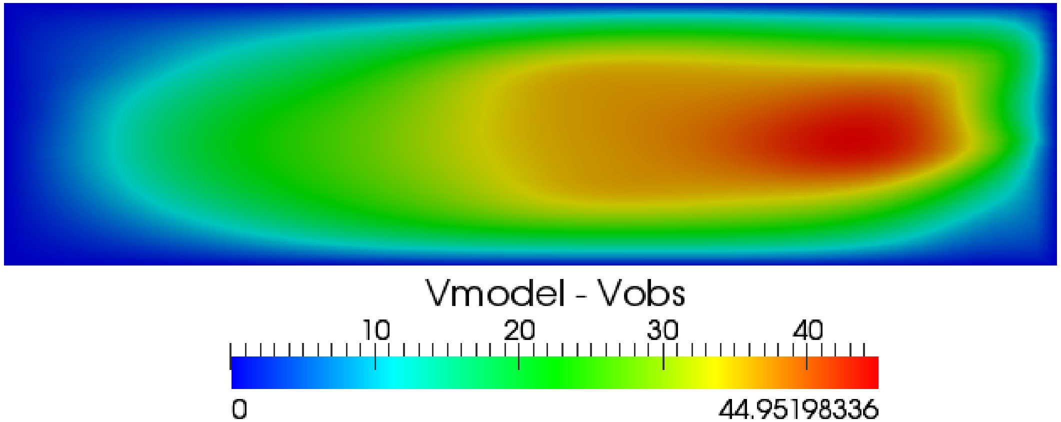
```
!!!!!  Compute Derivative of Cost function / Beta
Solver 5
  Equation = "DJDBeta"

!!  Solver need to be associated => Define dumy variable
  Variable = -nooutput "DJDB"
  Variable DOFs = 1

  procedure = "ElmerIceSolvers" "DJDBeta_Adjoint"

  Flow Solution Name = String "Flow Solution"
  Adjoint Solution Name = String "Adjoint"
  Optimized Variable Name = String "Beta"  ! Name of Beta variable
  Gradient Variable Name = String "DJDBeta"   ! Name of gradient variable
  PowerFormulation = Logical False
  Beta2Formulation = Logical True        ! SlipCoef define as Beta^2

  Lambda = Real  $Lambda                   ! Regularization Coef
end
```

Compute the **gradient of the cost function** with respect to the **Beta** variable (~slip coef.) from the **direct and adjoint solutions**

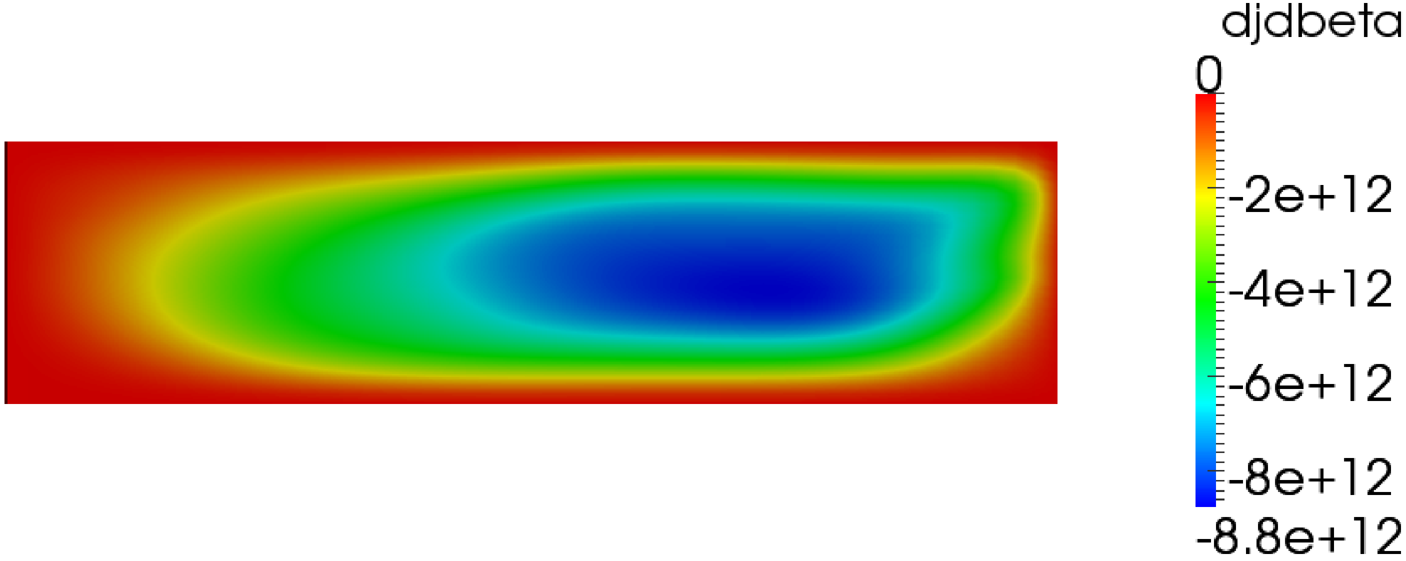
This part has been differentiated by hand

Step 1: Compute the cost function and the gradient

Visualise the difference between model and “observed” surface velocities



Visualise the gradient of the cost function with respect to the slip coefficient



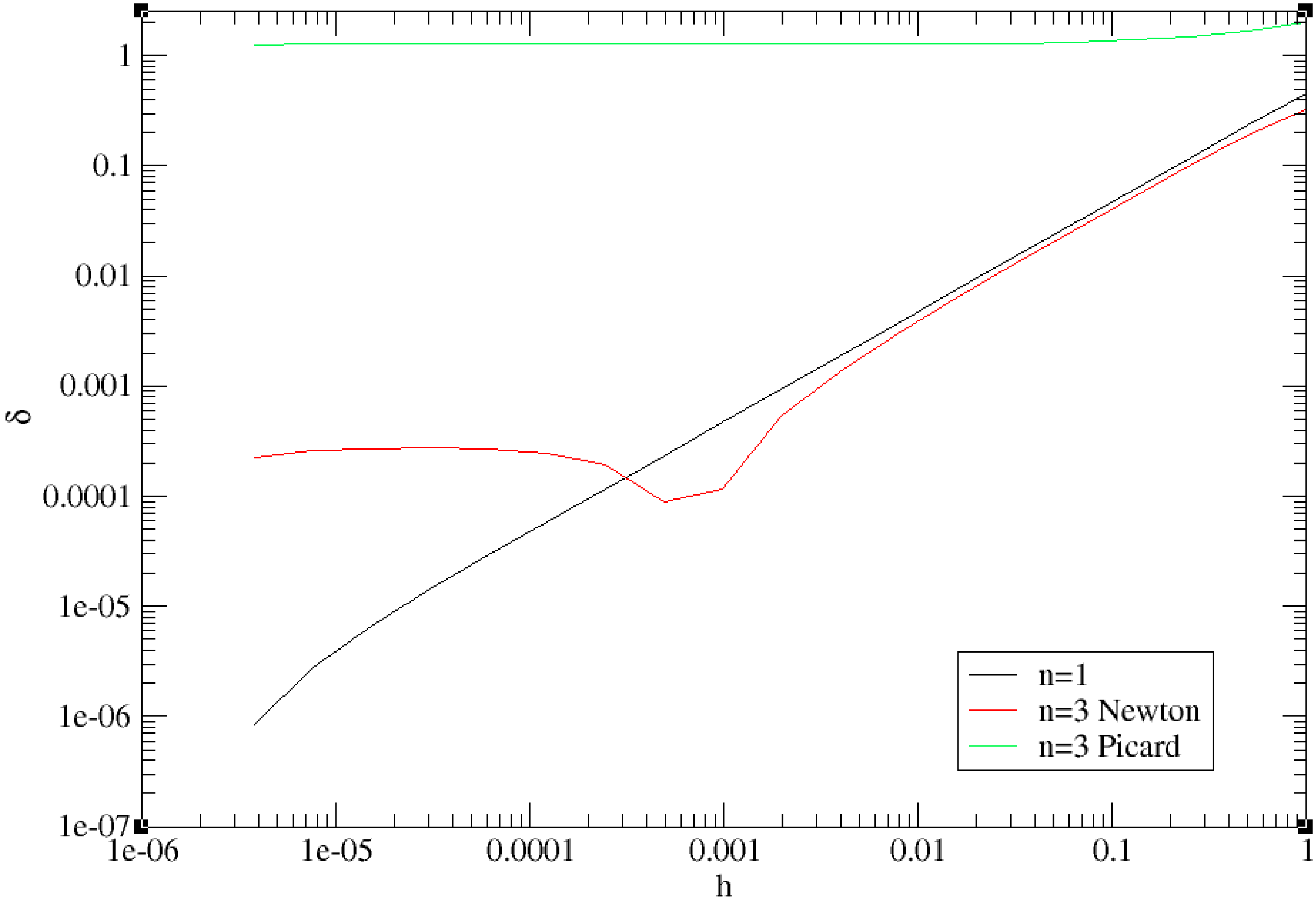
Step 2: Check the accuracy of your gradient

Validate the computation of the gradient with a finite difference scheme

```
!!!!! Gradient Validation
!!!!!!  Compute total derivative and update the step size for the finite difference computation
Solver 6
  Equation = "GradientValidation"

  !!  Solver need to be associated => Define dumy variable
   Variable = -nooutput "UB"
   Variable DOFs = 1

    procedure = "./Executables/GradientValidation" "GradientValidation"

  Cost Variable Name = String "CostValue"
  Optimized Variable Name = String "Beta"
  Perturbed Variable Name = String "BetaP"
  Gradient Variable Name = String "DJDBeta"
  Result File = File "GradientValidation_$name".dat"

end
```

$$\left.\begin{aligned} dJ^{adj} &= \frac{dJ}{dp}.p' \\ dJ^{FD} &= lim_{h->0}\frac{J(p+hp')-J(p)}{h} \end{aligned}\right\} \quad \delta(h) = abs(\frac{dJ^{adj}-dJ^{FD}}{dJ^{adj}})$$

Step 2: Check the accuracy of your gradient



Check with the improvement by comparing with the gradient test in Gagliardini et al. 2012

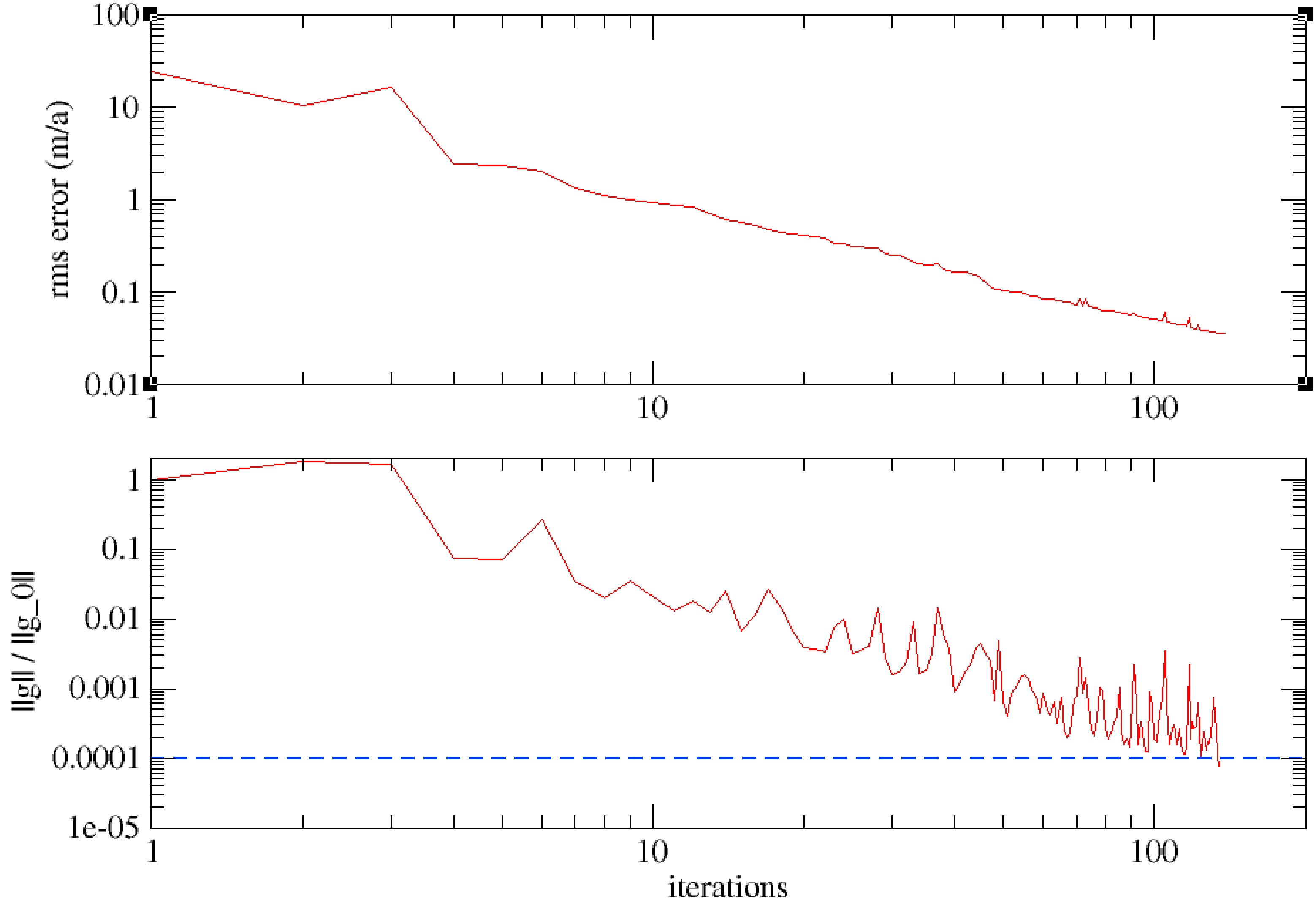
Step 3: Minimise your cost function

Retrieve the original nodal slip coefficients by minimising the cost function using M1QN3

```
!!!!! Optimization procedure
Solver 6
  Equation = "Optimize_m1qn3"

!!  Solver need to be associated => Define dumy variable
  Variable = -nooutput "UB"
  Variable DOFs = 1

  procedure = "ElmerIceSolvers" "Optimize_m1qn3Parallel"

  Cost Variable Name = String "CostValue"
  Optimized Variable Name = String "Beta"
  Gradient Variable Name = String "DJDBeta"
  gradient Norm File = String "GradientNormAdjoint_$name".dat"


! M1QN3 Parameters
  M1QN3 dxmin = Real 1.0e-10
  M1QN3 epsg = Real  1.e-4
  M1QN3 niter = Integer 400
  M1QN3 nsim = Integer 400
  M1QN3 impres = Integer 5
  M1QN3 DIS Mode = Logical False
  M1QN3 df1 = Real 0.5
  M1QN3 normtype = String "dfn"
  M1QN3 OutputFile = File  "M1QN3_$name".out"
  M1QN3 ndz = Integer 20

end
```

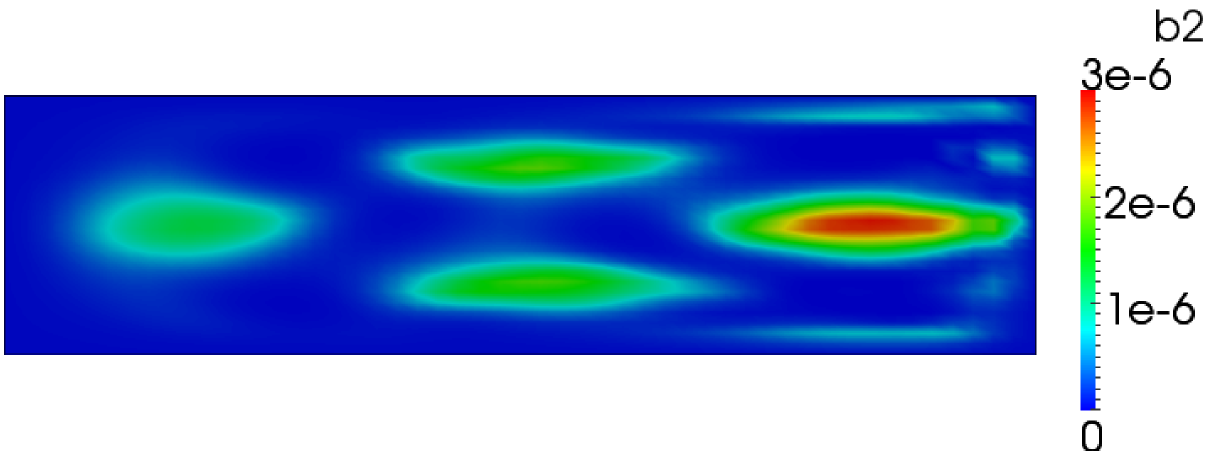
Step 3: Minimise your cost function

Check the evolution of the cost function and gradient norm as a function of the number of iterations

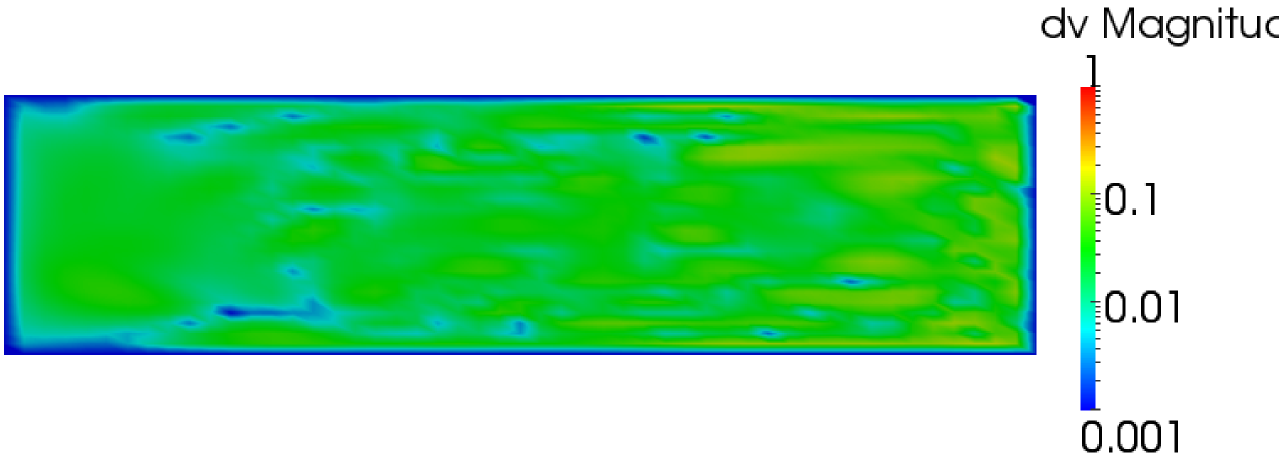


Step 3: Minimise your cost function

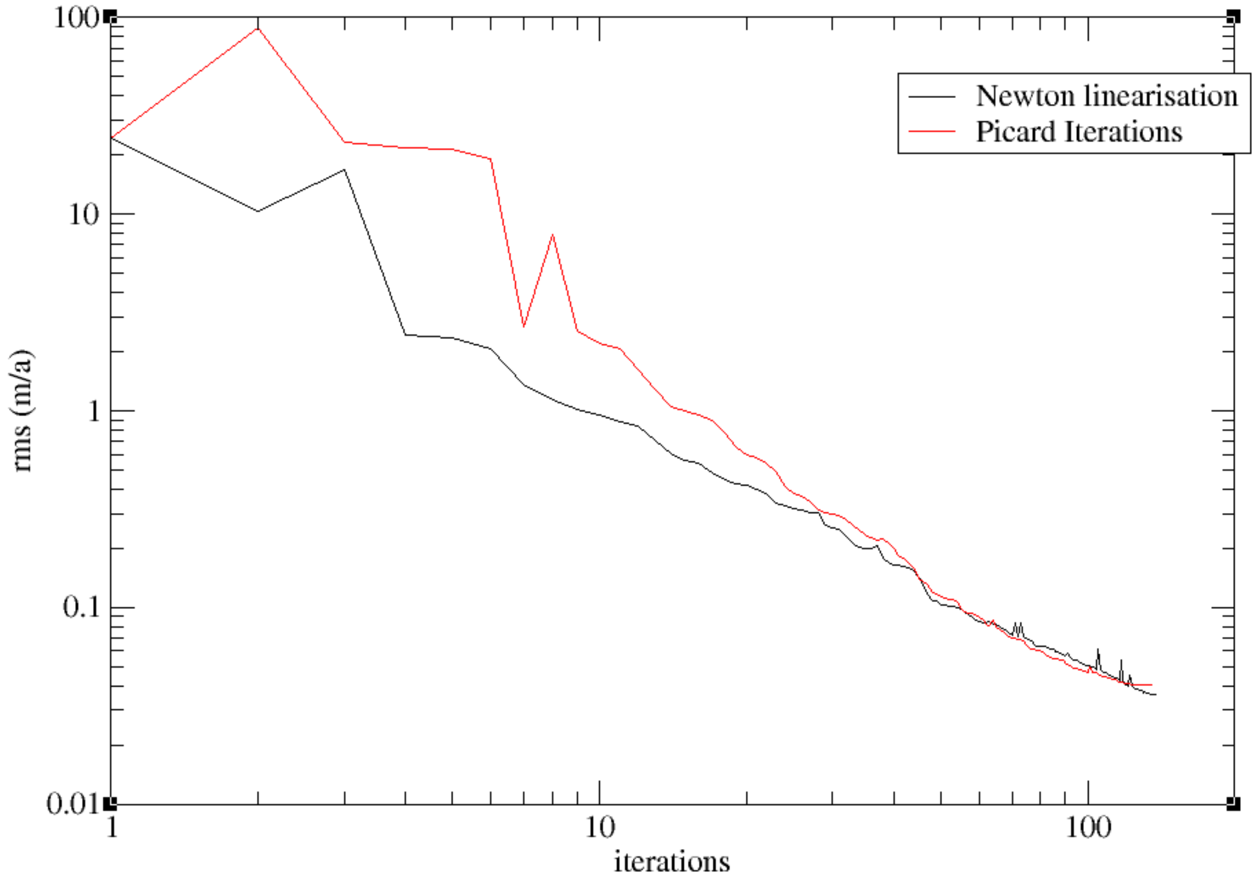
Visualise the final slip coefficient distribution



Visualise the mismatch between model and observed velocities



Step 3: try with the “inexact” adjoint

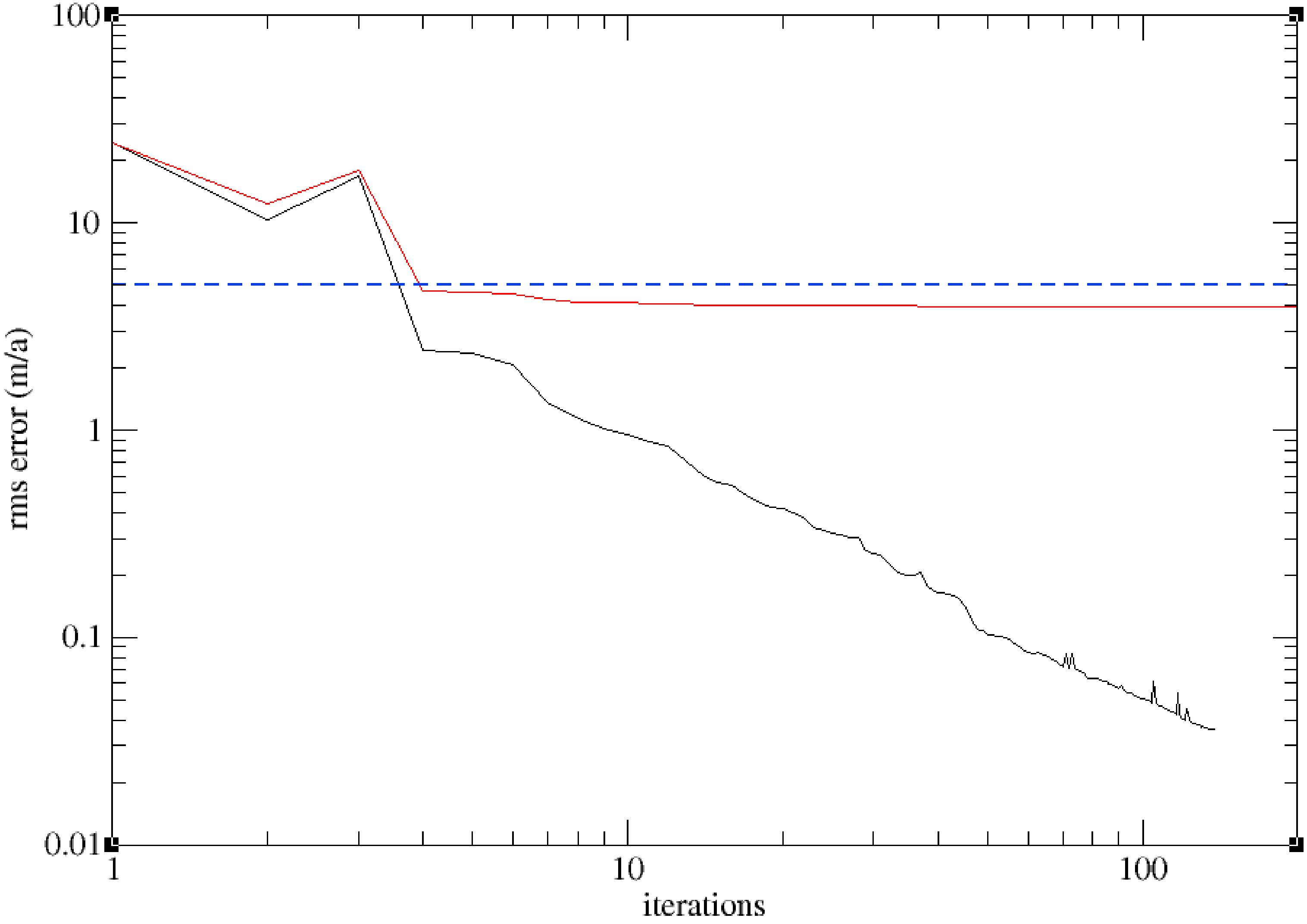


Even with an **inaccurate gradient** computation (i.e. neglecting the non-linearity due to the viscosity) you **may** be able to **minimise your cost function**....

But you have more chance to get lost in real applications

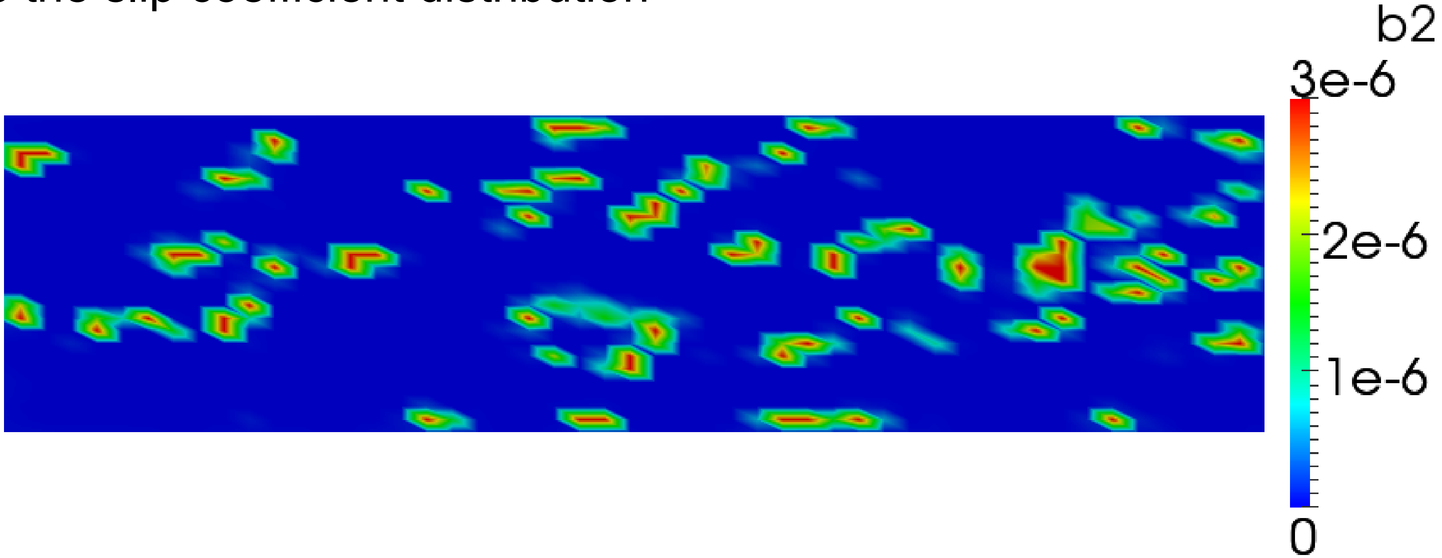
Step 4: Add noise to your “observations”

- Use the script **AddNoise.sh** to add random noise to your perfect observations
- Check the evolution of the cost function and gradient norm as a function of the number of iterations

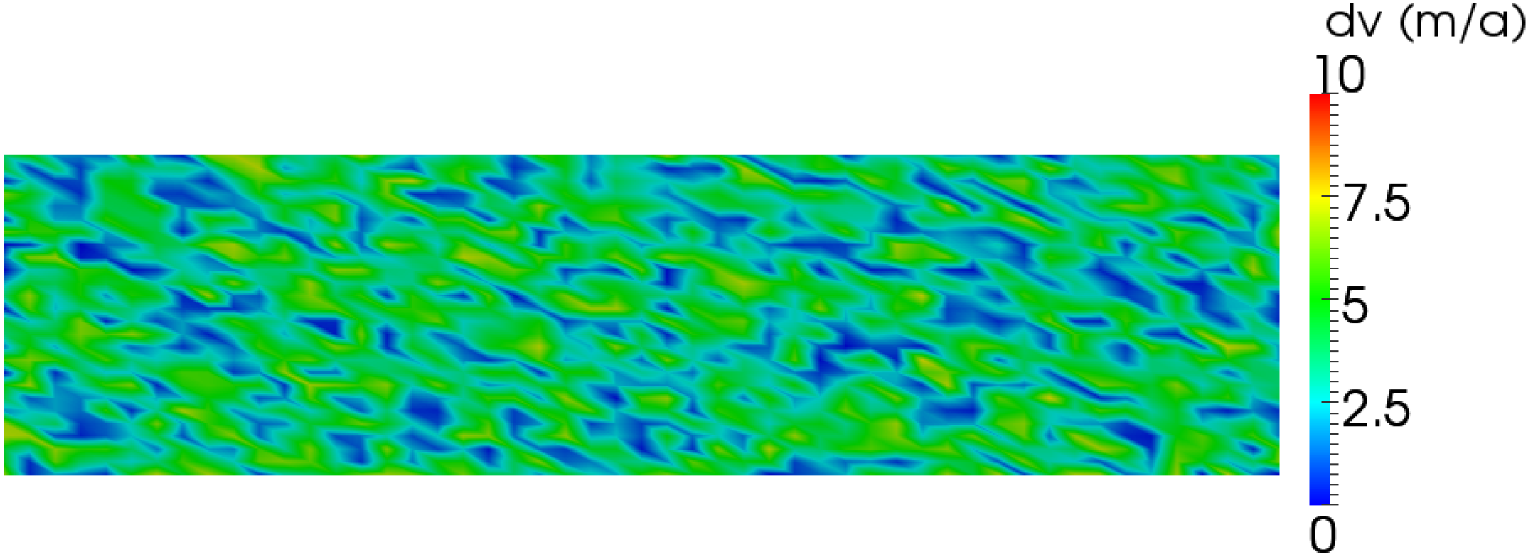


Step 4: Add noise to your "observations"

Visualise the slip coefficient distribution



Visualise the mismatch between model and observed velocities



Step 4: Add noise to your “observations”

Remedies :

1. Stop when you reach your rms error (i.e. avoid over-fitting)

   (cf e.g. Arthern and Gudmundsson, 2010)

2. Add a regularisation term to the cost function

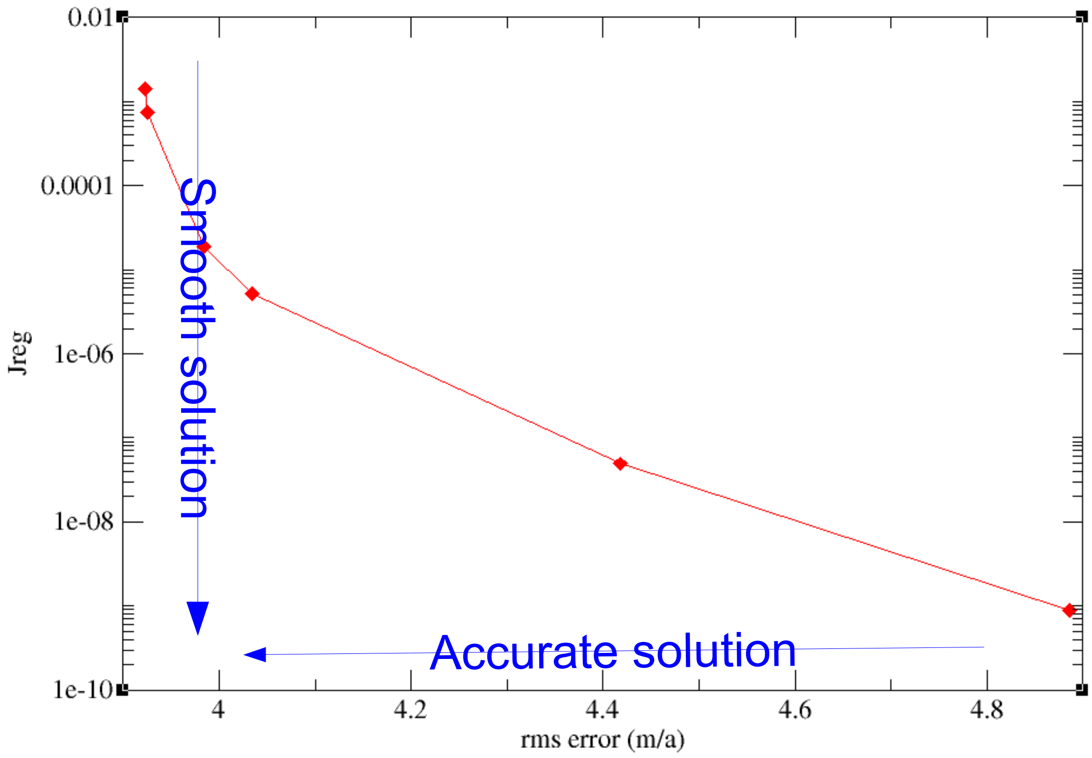
$$J_{tot} = J_0 + \lambda J_{reg}$$

Here, penalise spatial derivatives of the input parameter:

$$J_{reg} = \frac{1}{2} \int_{\Gamma_b} (\frac{d\beta}{dx})^2$$

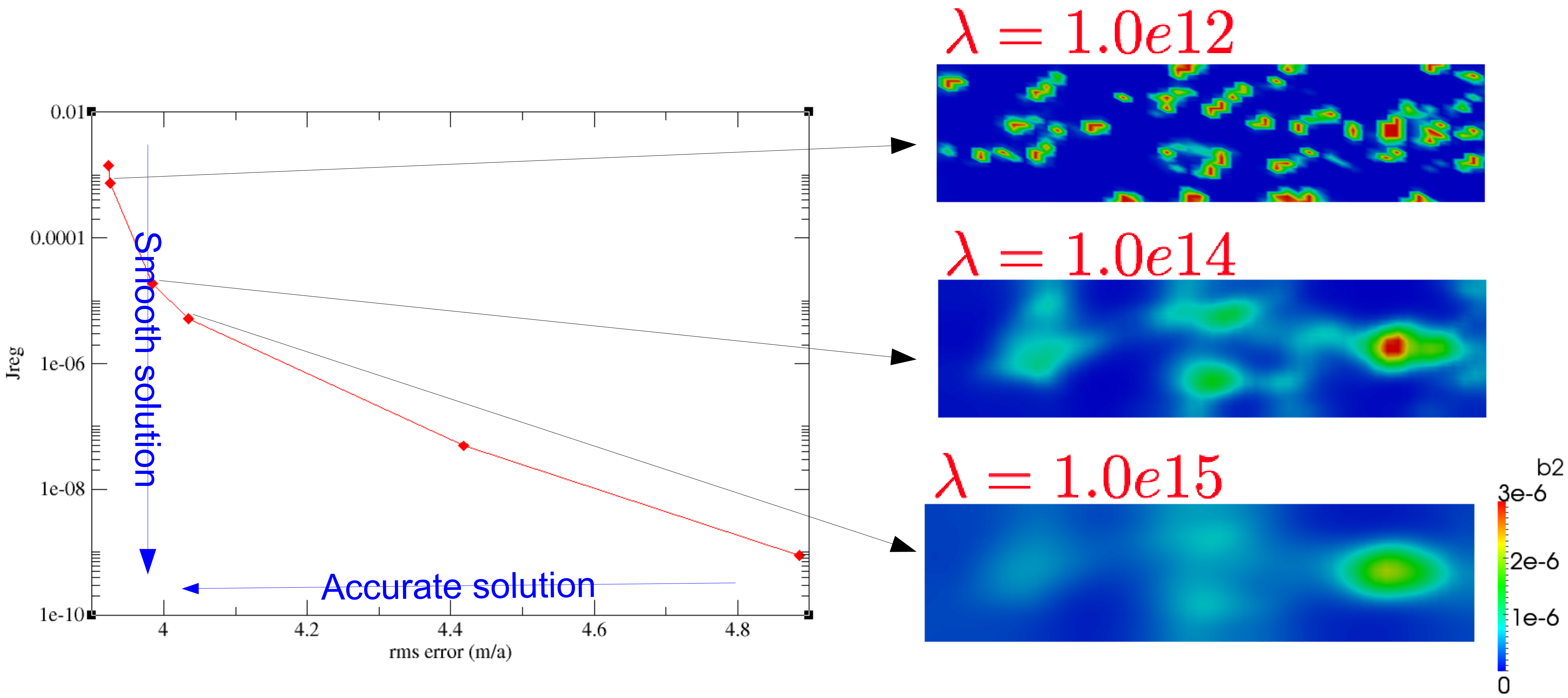
Step 5: Add regularisation

Change the value of the regularisation weight, observe the final results and plot the L_Curve

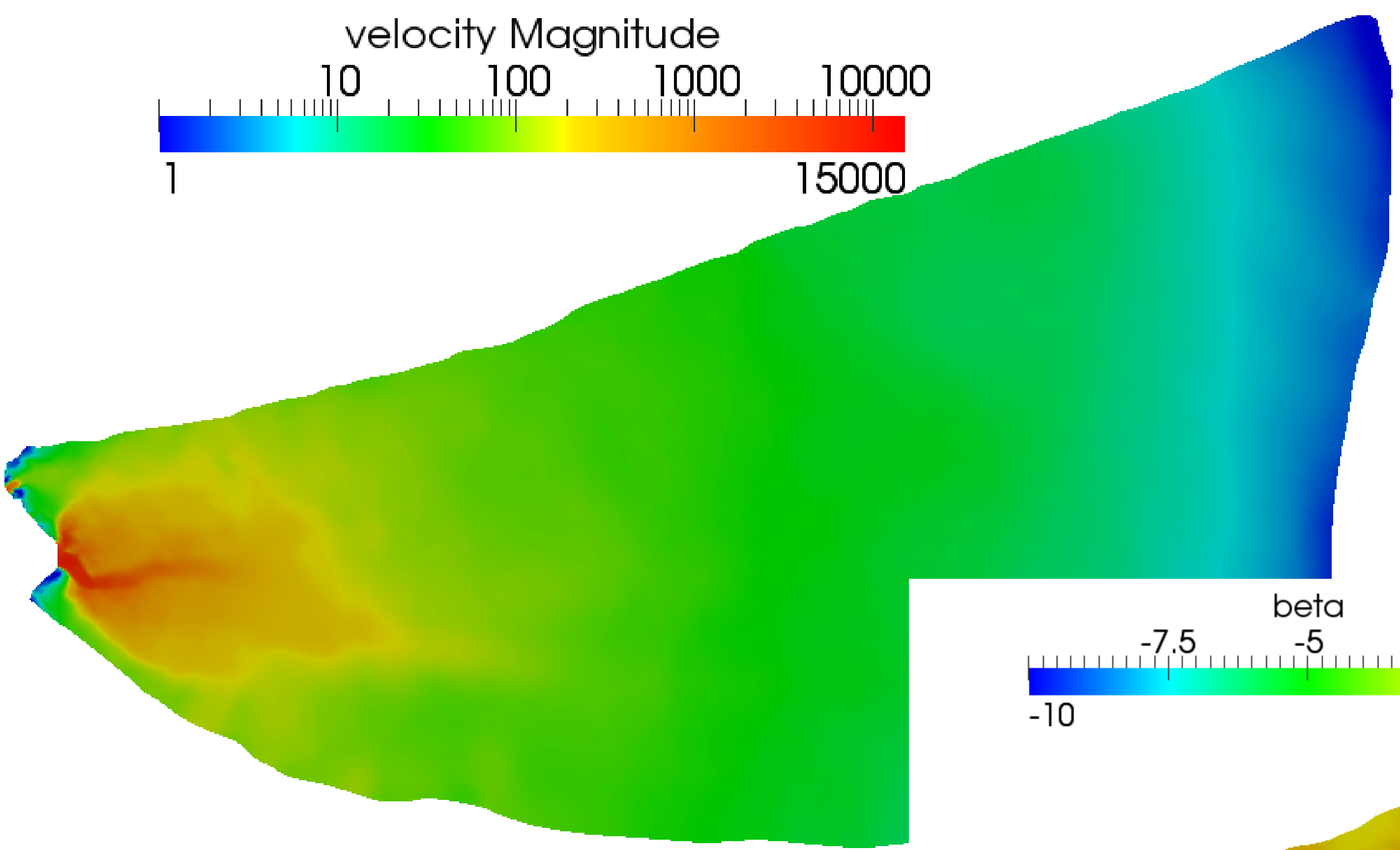


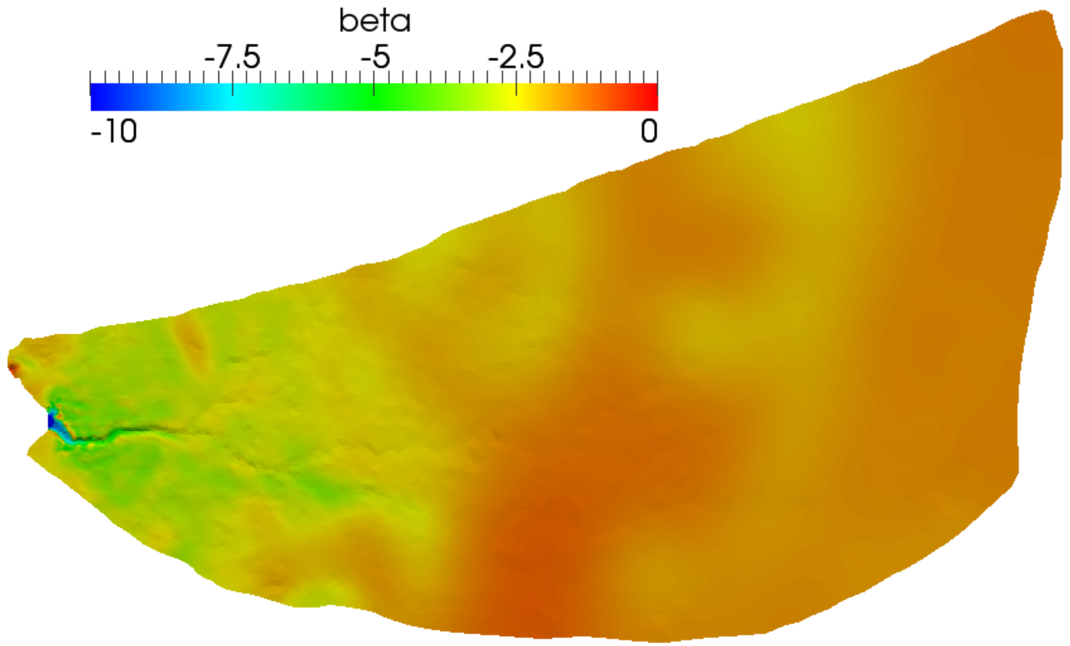
Step 5: Add regularisation

Change the value of the regularisation weight, observe the final results and plot the L_Curve



Step 6: A Real case application **Jacobshavn Isbrae**





Conclusion/Perspectives

- Should be relatively easy to use for your own applications
- Ask me for help if needed; I will be very happy to collaborate on this
- Please refer to the Elmer/Ice capabilities paper (Gagliardini et al, 2013) if you use these solvers
- Next steps:
  - Easy: assimilation of boundary conditions; use inverse methods with SSA;SIA solvers
  - Not easy: move to transient data assimilation. Shape optimisation (bedrock topography)