# Elmer/Ice advanced Workshop

30 Nov – 2 Dec 2015

# *Inverse Methods*

Fabien Gillet-Chaulet

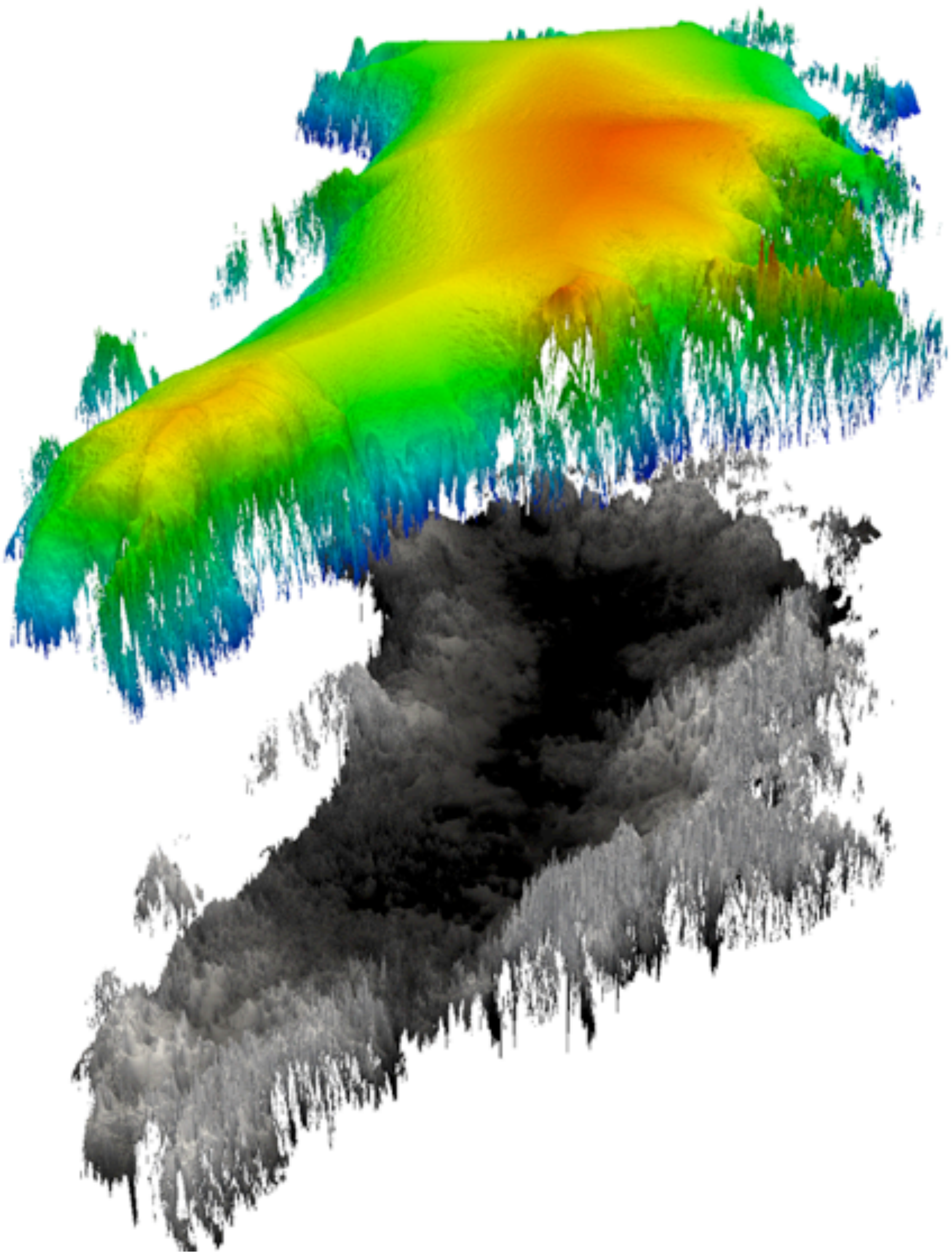
LGGE - Grenoble - France

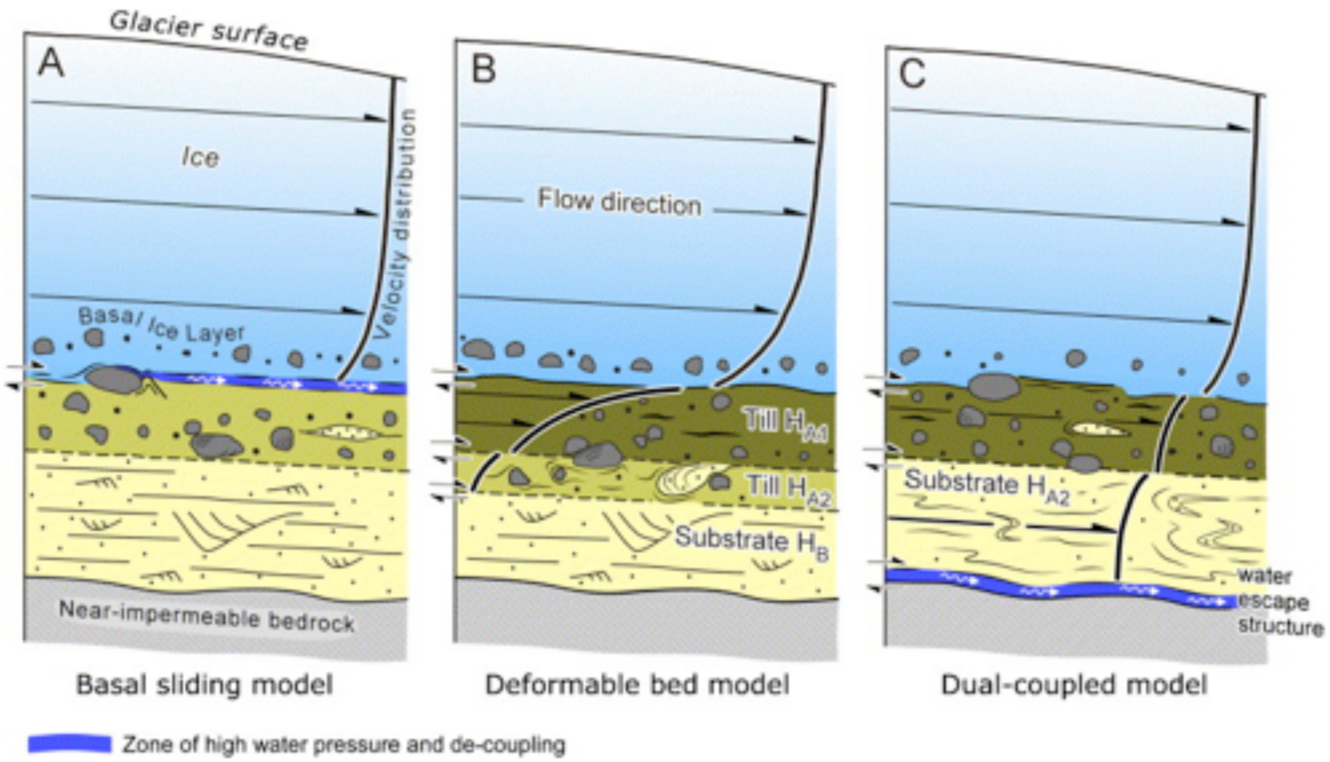
# Outline

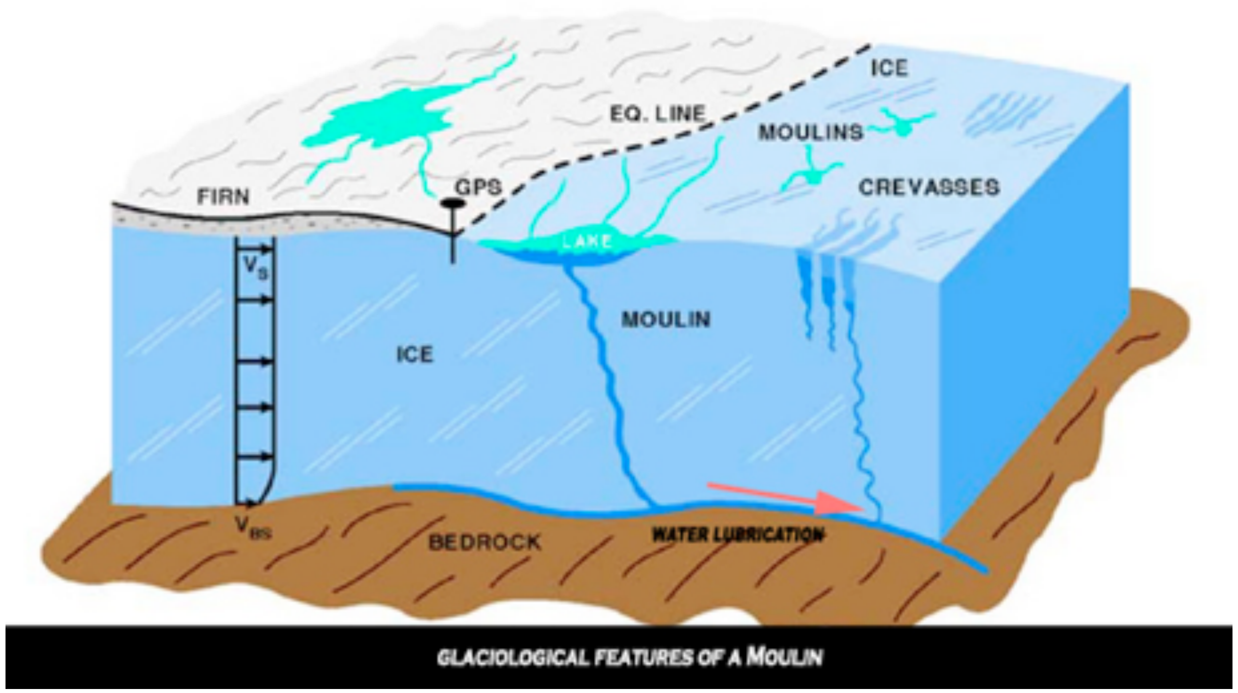
- **Short introduction**
- **Inverse methods in Elmer/Ice (Stokes solver)**
- **Current / planned developments**

# 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)

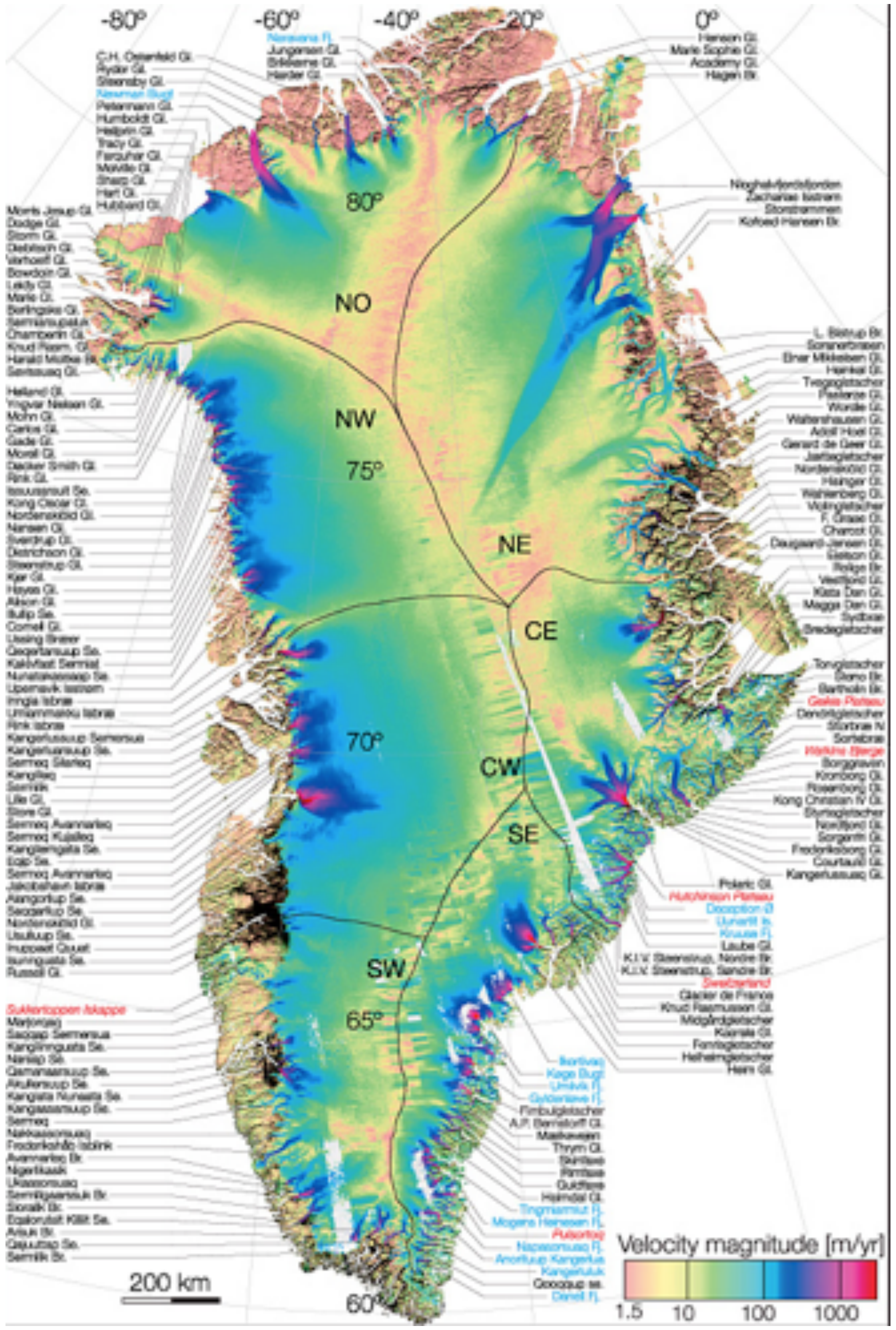




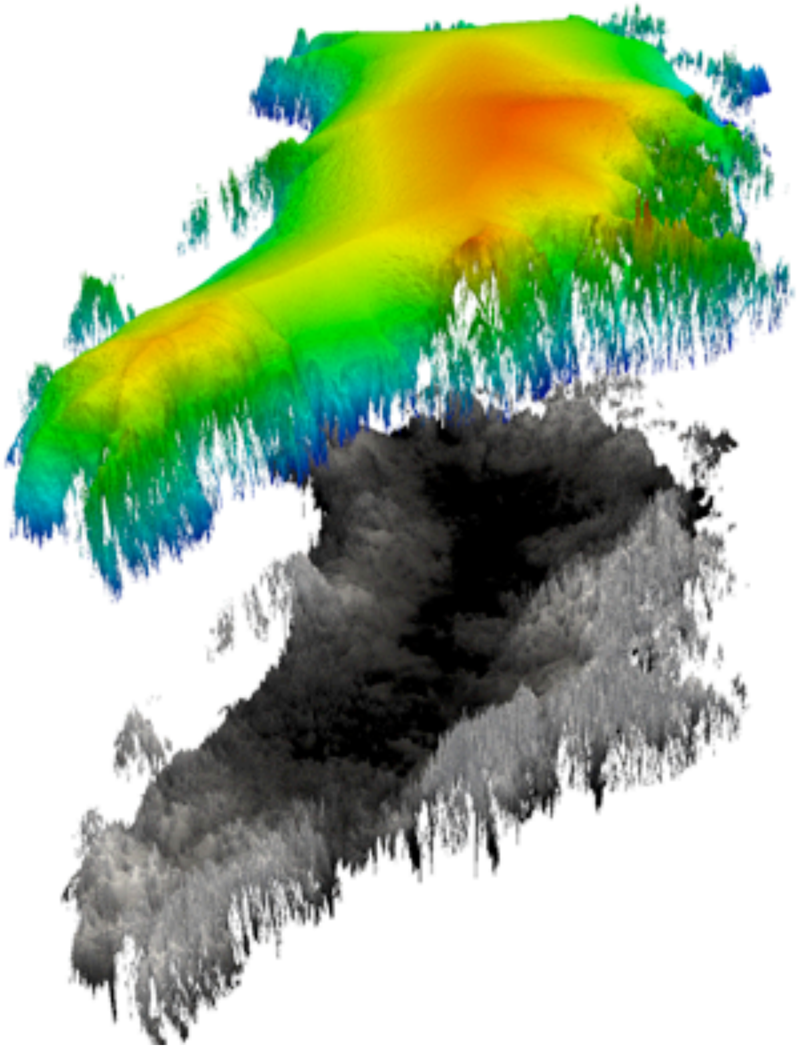



# *More and more available observations*

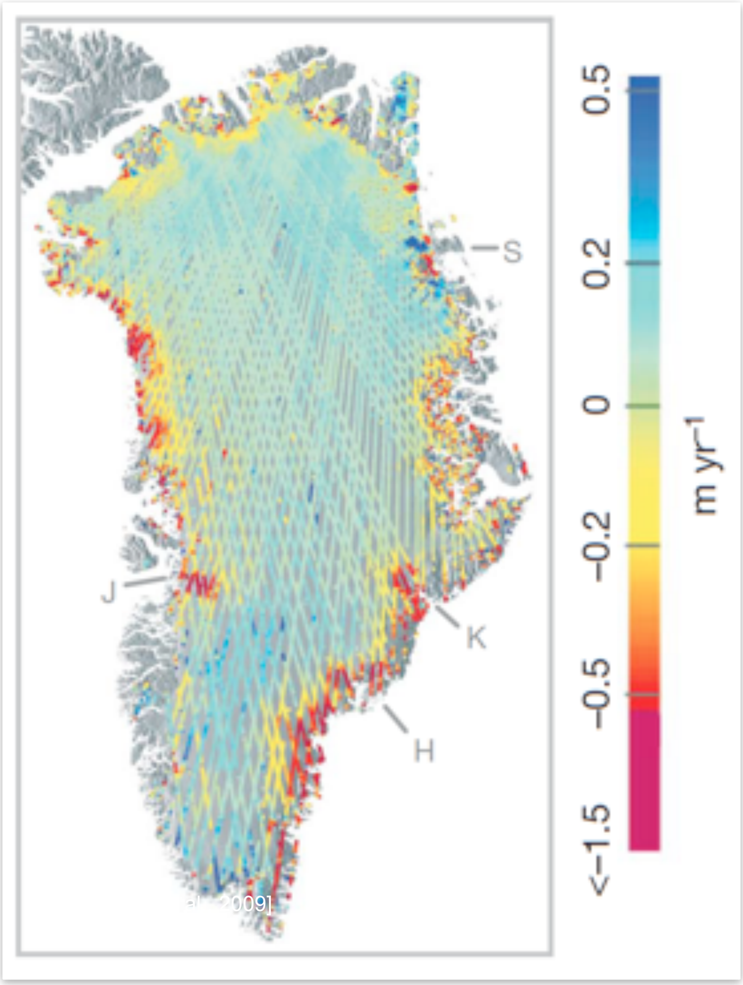
Surface velocities




Topography



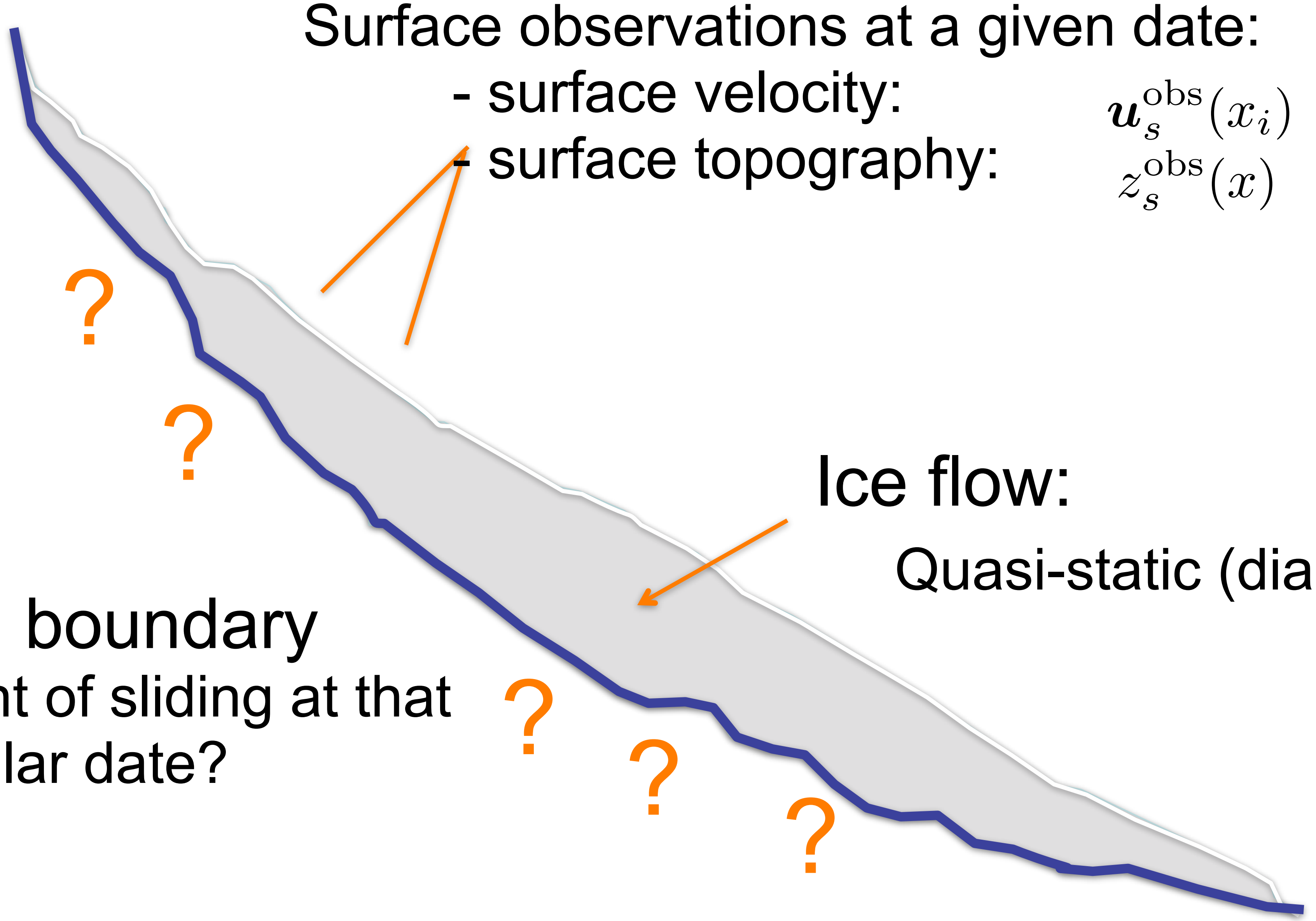
ds/dt



# Specificity of ice flow

## Very low Reynolds -> no history in the velocity

Surface observations at a given date:
- surface velocity: $\boldsymbol{u}_s^{\mathrm{obs}}(x_i)$
- surface topography: $z_s^{\mathrm{obs}}(x)$




**Ice flow:**

Quasi-static (diagnostic)

**Basal boundary**

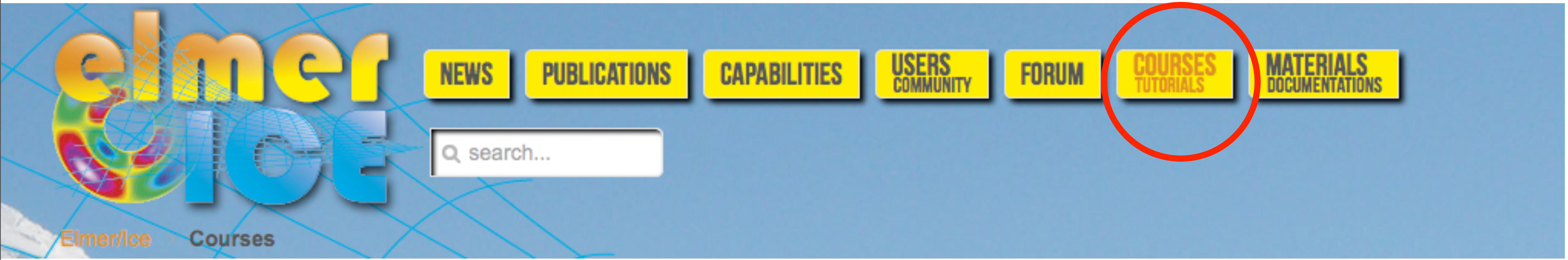
Amount of sliding at that particular date?

Reconstruction of the basal conditions from surface measurements

# *Outline*

- **Short introduction**
- **Inverse methods in Elmer/Ice (Stokes solver)**
- **Current / planned developments**

# *Nothing really new since the CSC - 2013 Advanced Course*




## CSC - Espoo - 4-6 November 2013

A 3-day Elmer/Ice advanced workshop was organised at CSC (Espoo, Finland) from the 4th to the 6th of November 2013. The course was held by Fabien Gillet-Chaulet (LGGE), Mika Malinen (CSC), Peter Råback (CSC) and Thomas Zwinger (CSC).

| Title | Presentation | Material |
|---|---|---|
| Introduction to Elmer | pdf | - |
| Elmer Glaciological Modelling | pdf | - |
| Simple Hydro Toymodel | pdf | tar file |
| Structured Meshes | pdf | tar file |
| Enhanced pre-processing | pdf | USB stick |
| Block pre-conditioner | pdf | USB stick |
| Enhanced post-processing | pdf | ZIP archive |
| Make-file for YAMS on Ubuntu 64bit | - | tar archive |
| Mesh Adaptation using YAMS (see also these notes) | pdf | tar archive |
| Inverse methods | pdf | tar archive |

# See also the Elmer/Ice reference paper

Geosci. Model Dev., 6, 1299–1318, 2013
www.geosci-model-dev.net/6/1299/2013/
doi:10.5194/gmd-6-1299-2013

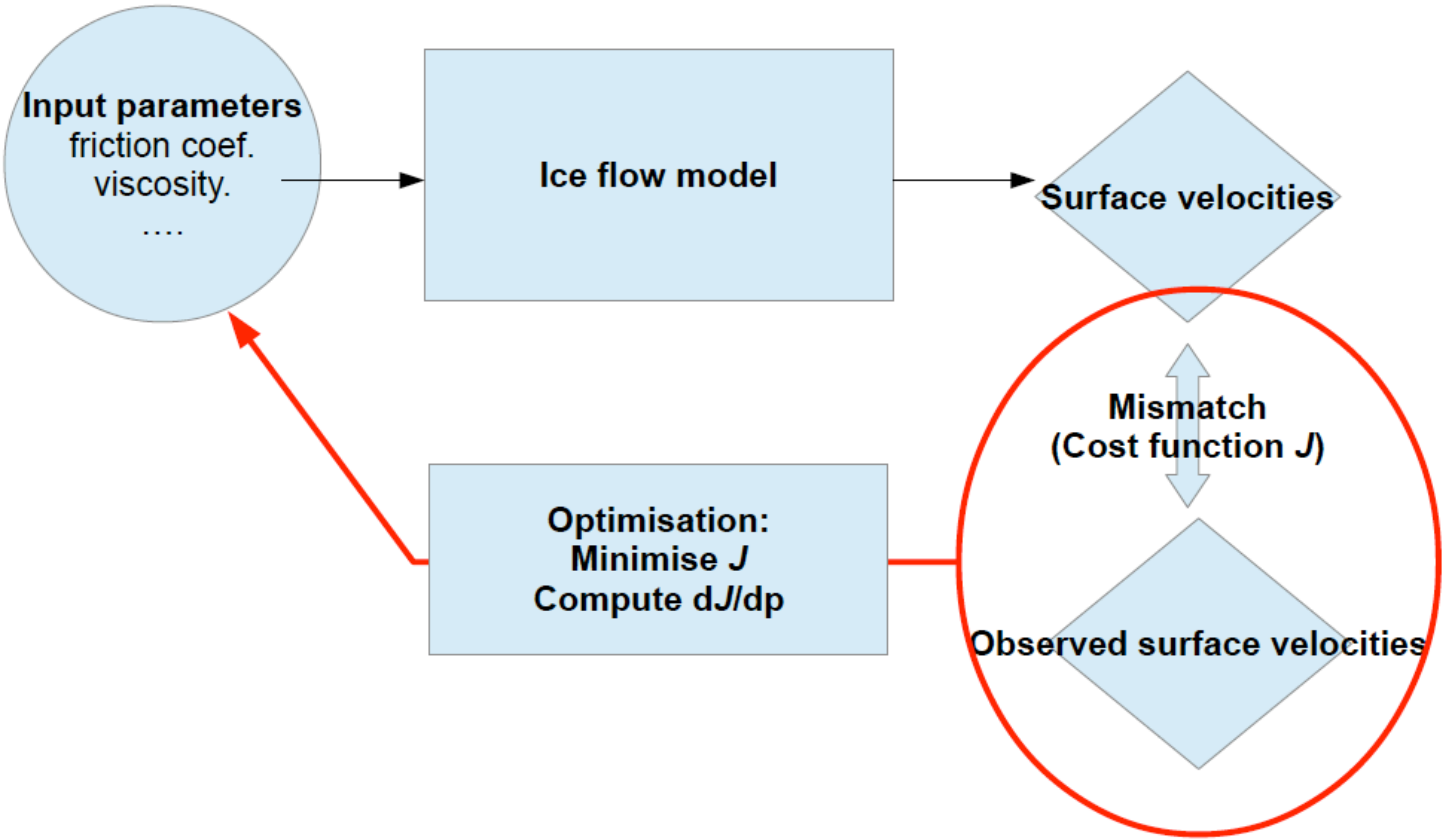

Geoscientific Model Development
Open Access

## Capabilities and performance of Elmer/Ice, a new-generation ice sheet model

**O. Gagliardini[1,2], T. Zwinger[3], F. Gillet-Chaulet[1], G. Durand[1], L. Favier[1], B. de Fleurian[1], R. Greve[4], M. Malinen[3], C. Martín[5], P. Råback[3], J. Ruokolainen[3], M. Sacchettini[1], M. Schäfer[6], H. Seddik[4], and J. Thies[7]**

[1]Laboratoire de Glaciologie et Géophysique de l'Environnement, UJF-Grenoble, CNRS – UMR5183, Saint-Martin-d'Hères, France
[2]Institut Universitaire de France, Paris, France
[3]CSC-IT Center for Science Ltd., Espoo, Finland
[4]Institute of Low Temperature Science, Hokkaido University, Sapporo, Japan
[5]British Antarctic Survey, Cambridge, UK
[6]Arctic Centre, University of Lapland, Rovaniemi, Finland
[7]Uppsala University, Uppsala, Sweden

# *Variational data assimilation*



# *Inverse methods in Elmer*

- **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)

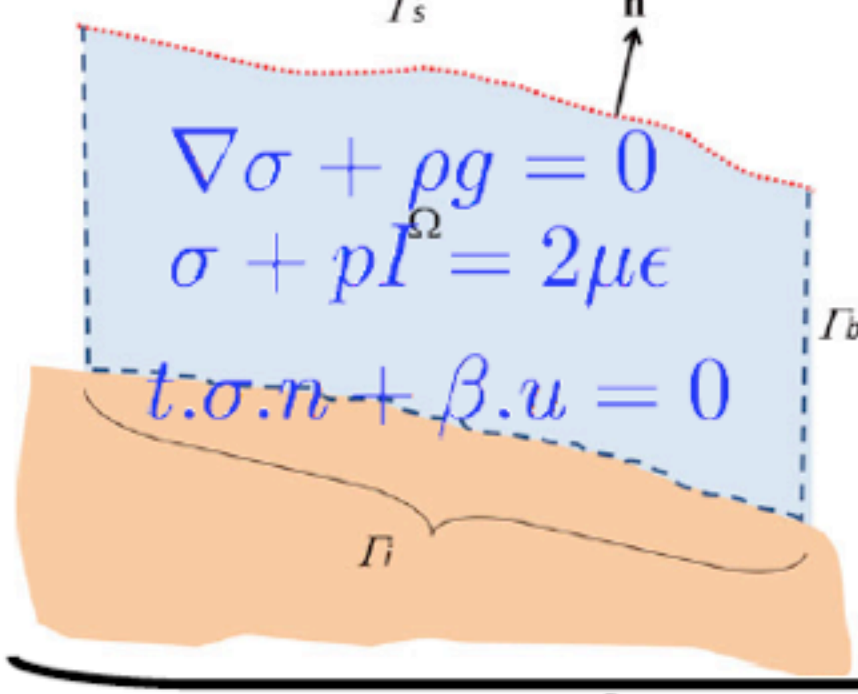
    **Characteristics:**
    => restricted to **diagnostic** (no time evolution)
    => **slip coefficient** (Linear sliding law)
    => **ice viscosity**
    => could also do Neumann and Dirichlet BC (Adjoint method)

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

# *Robin inverse method* *(Arthern and Gudmundsson, 2010)*

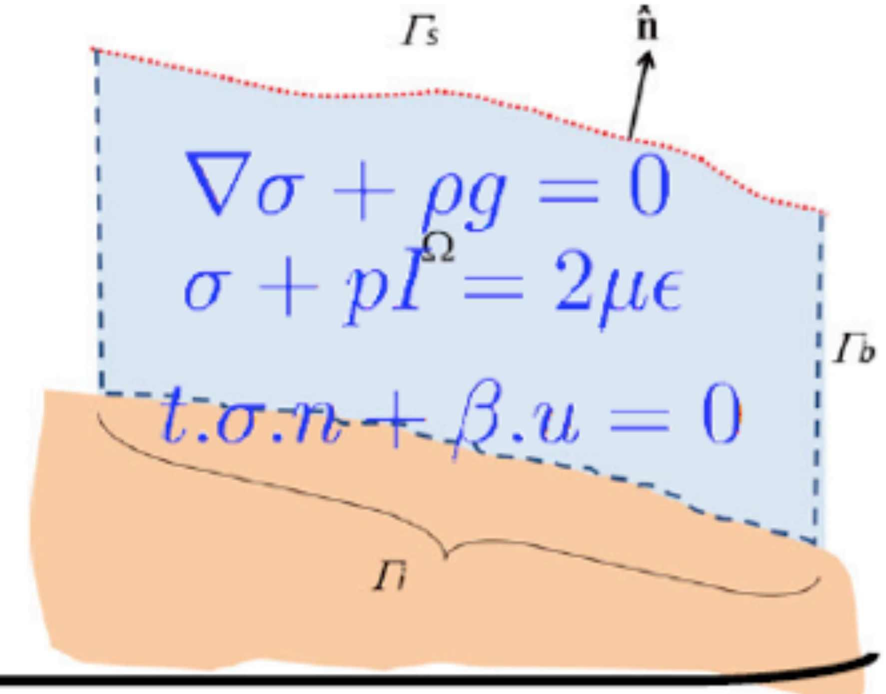
## Neumann problem

$\sigma.n = 0$




## 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)*

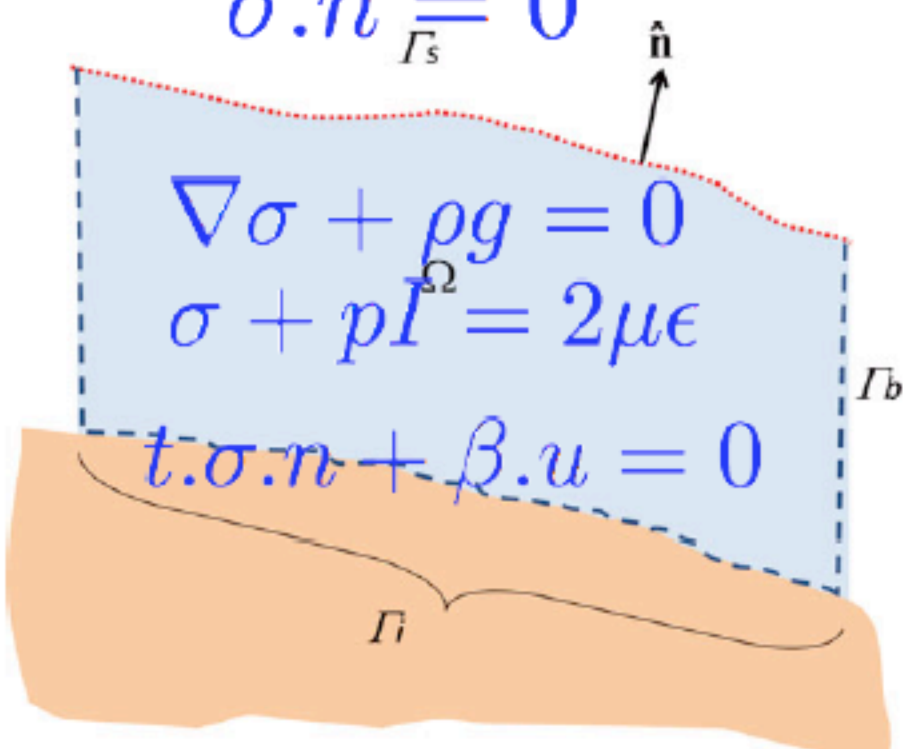
Direct problem

$$\sigma.n \underset{\Gamma_s}{=} 0$$

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

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

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




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

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

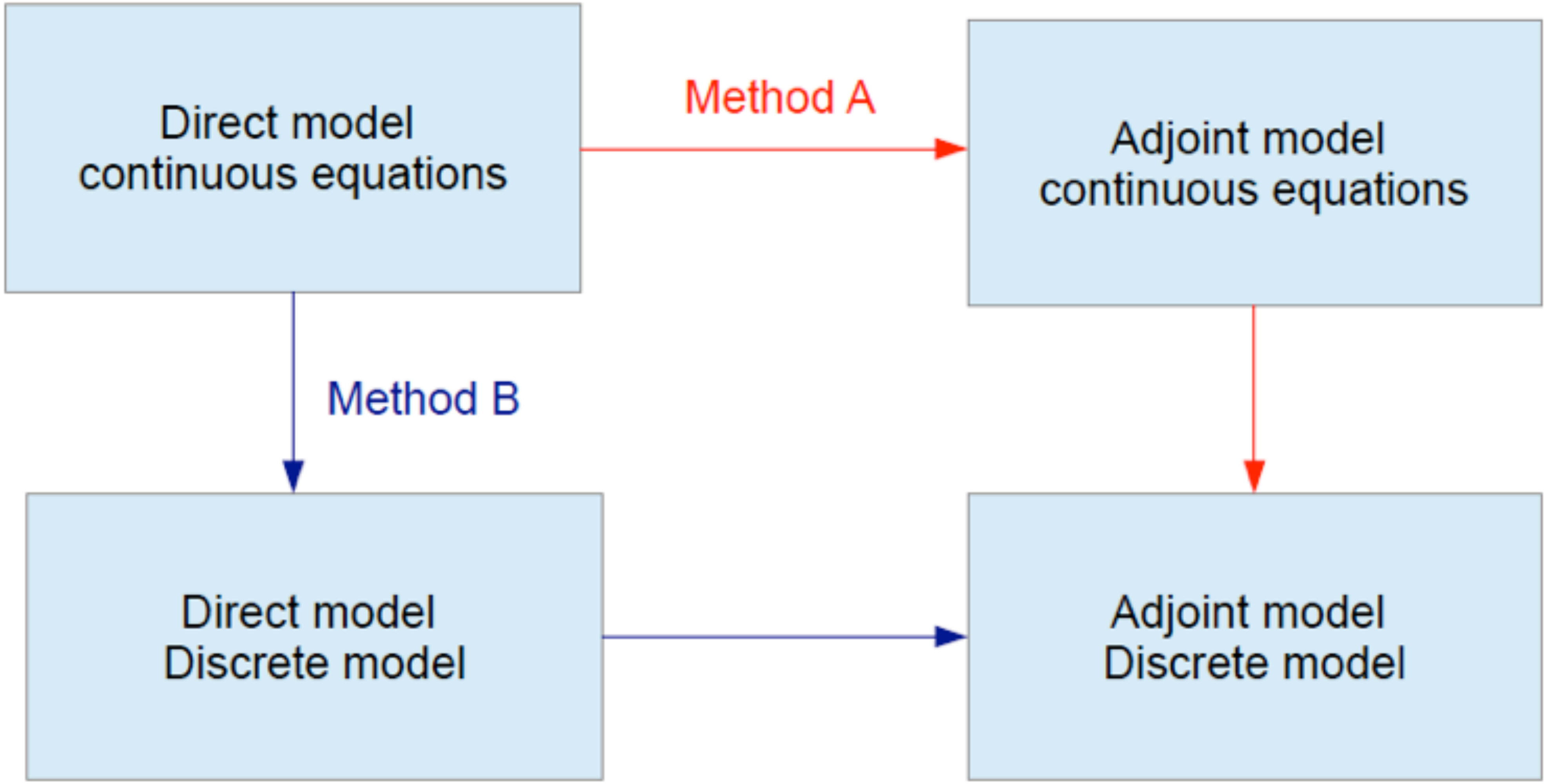
3. Minimisation of *J'* requires that all variations are 0

$d_\Lambda J' = 0$ => 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*




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 methods comparison*

**Robin Inverse method**

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

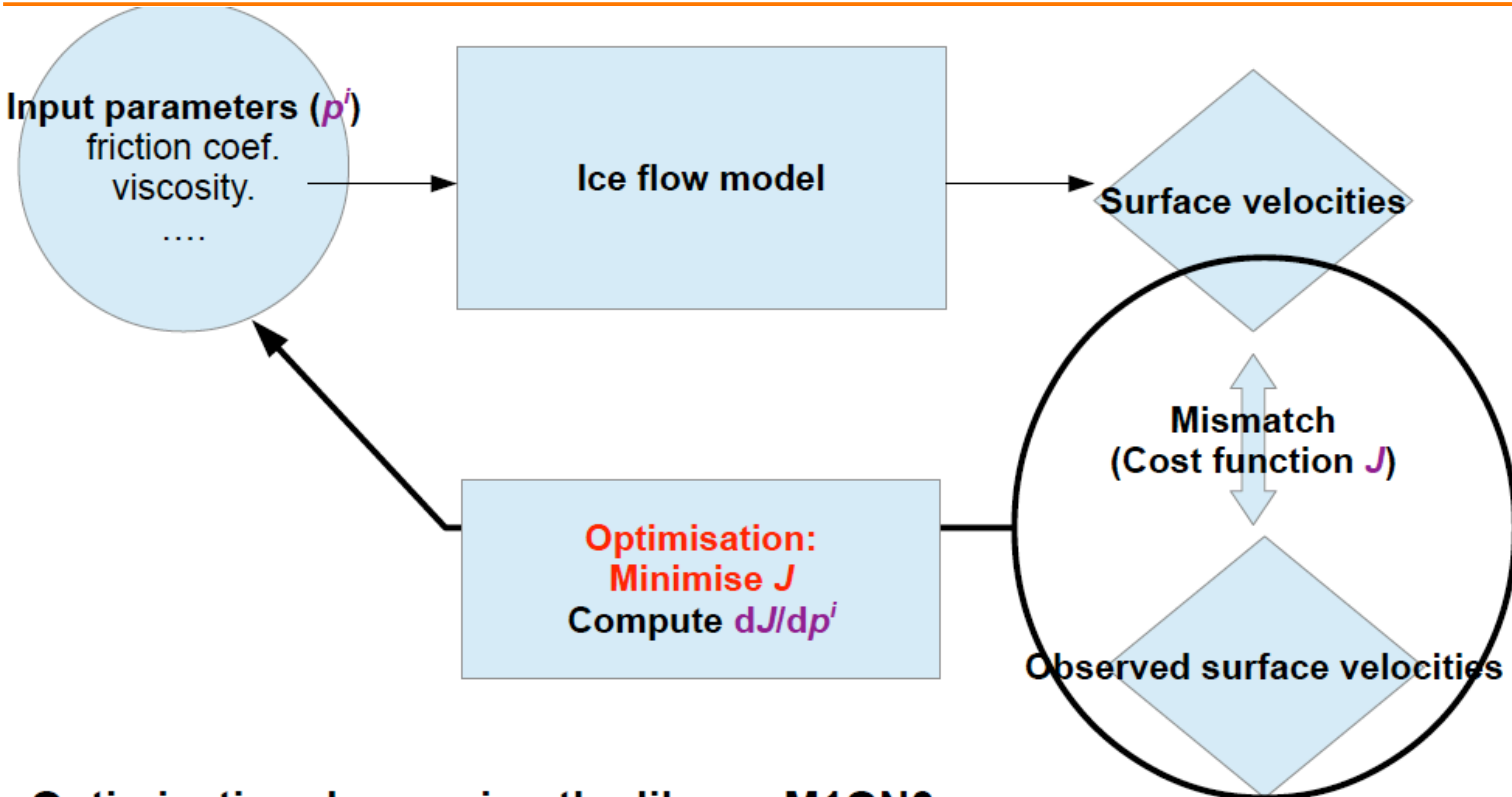
**Adjoint method**

- 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

## *Start with the adjoint method in Elmer/Ice*

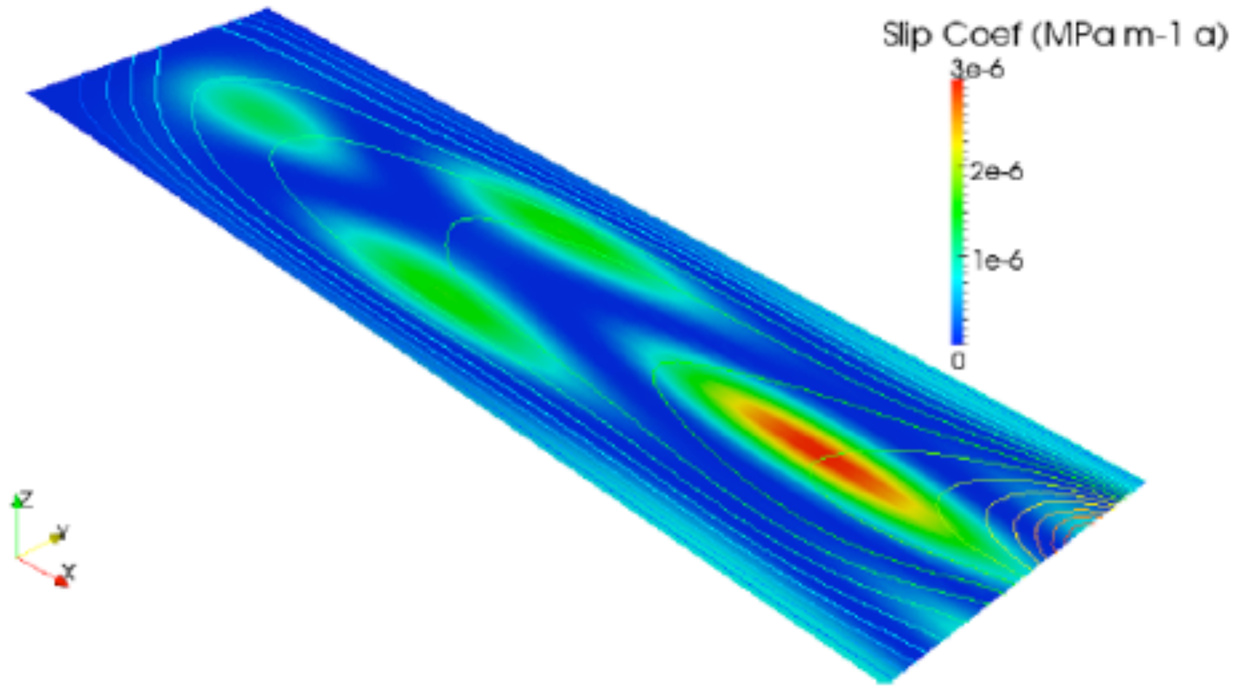
See the CSC-2013-Advanced Course Material:

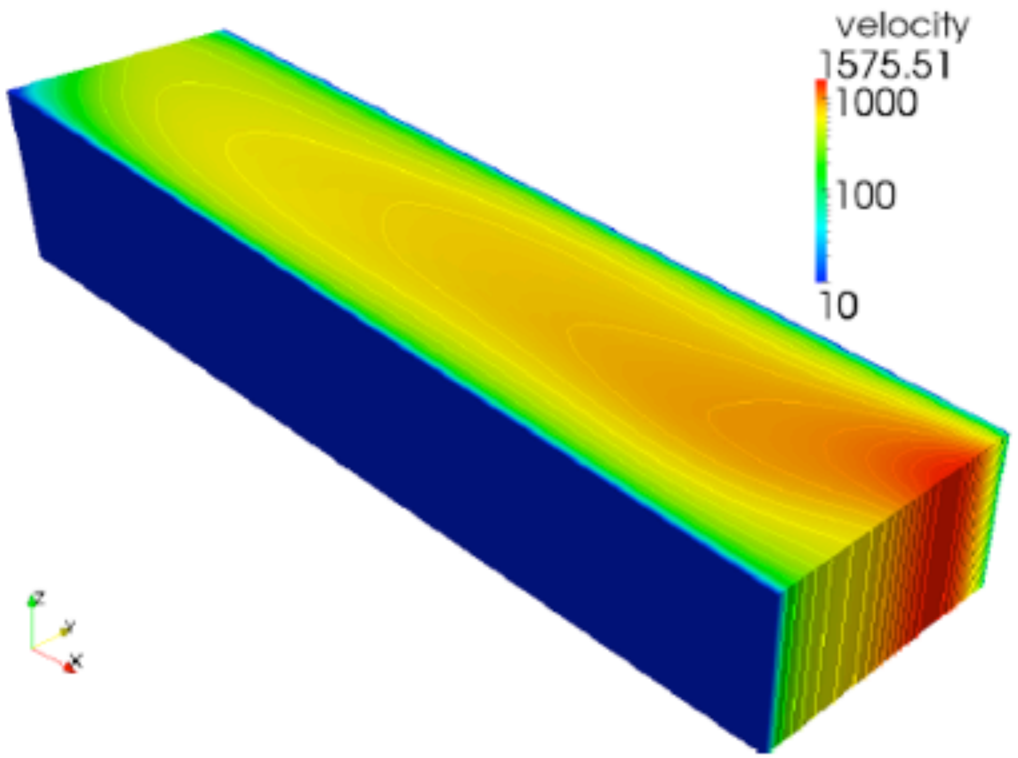
- step by step construction of a «twin experiment»
- set-up based on Mac Ayeal, 1993
- Application to Jacobshavn Isbrae drainage basin

# *Step0: Reference solution generated with the model*

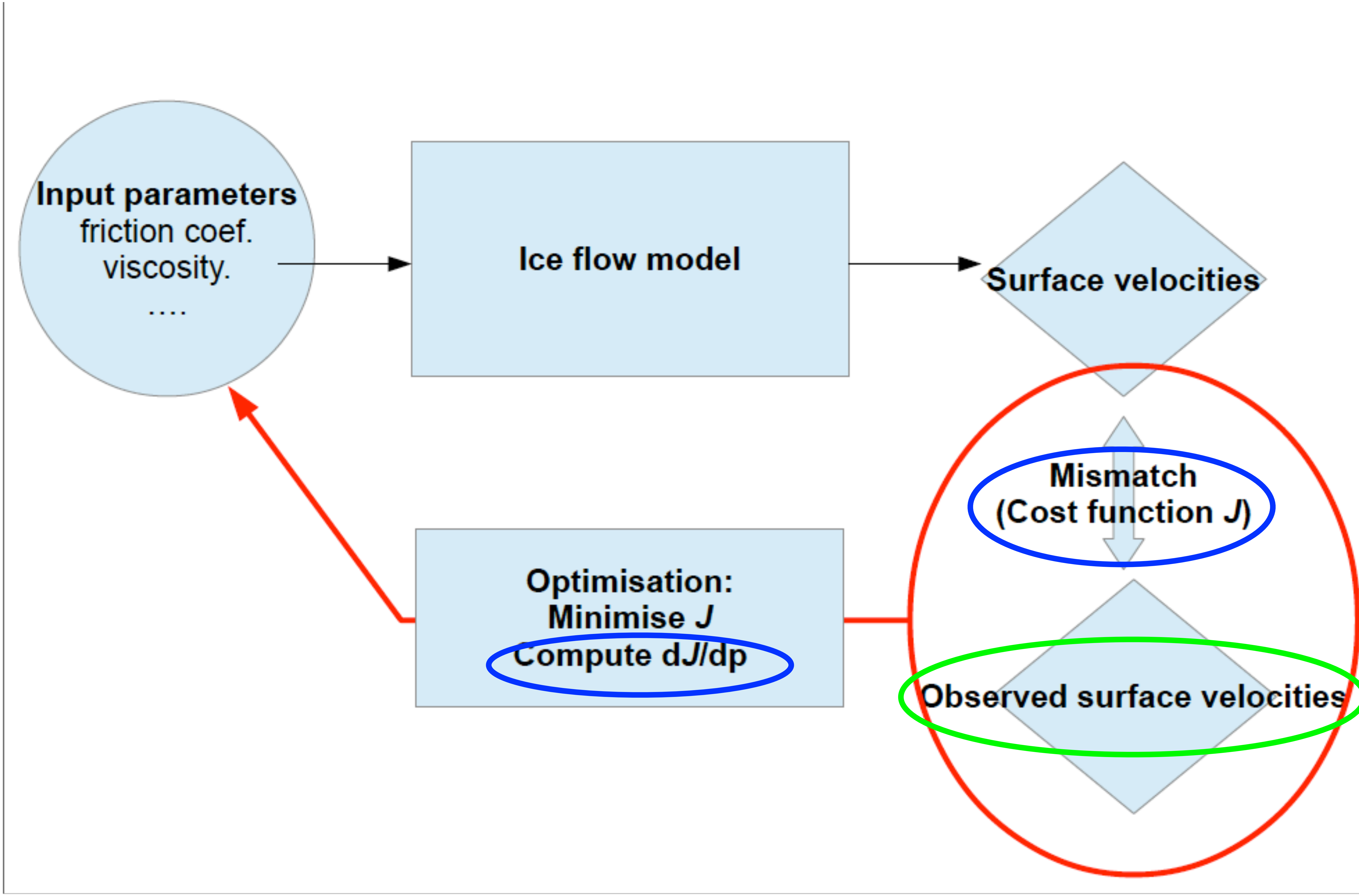
```
!Reference Slip Coefficicient used to construct surface velocities
$ function betaSquare(tx) {\
  Lx = 200.0e3;\
  Ly = 50.0e03;\
  yearinsec = 365.25*24*60*60;\
  F1=sin(3.0*pi*tx(0)/Lx)*sin(pi*tx(1)/Ly);\
  F2=sin(pi*tx(0)/(2.0*Lx))*cos(4.0*pi*tx(1)/Ly);\
  beta=5.0e3*F1+5.0e03*F2;\
  _betaSquare=beta*beta/(1.0e06*yearinsec);\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

## Ideal observed surface velocities



# Step 1: Compute the cost function and the gradient



# 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. Solve your problem (solver Stokes)

## 3. Compute the cost function: 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*

**3. 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

**3. 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,
**Will be changed to allow more flexible regularisation,**
e.g. a-priori estimate

# Step 1: Compute the cost function and the gradient

**3. 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

## 4. 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

## 5. 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

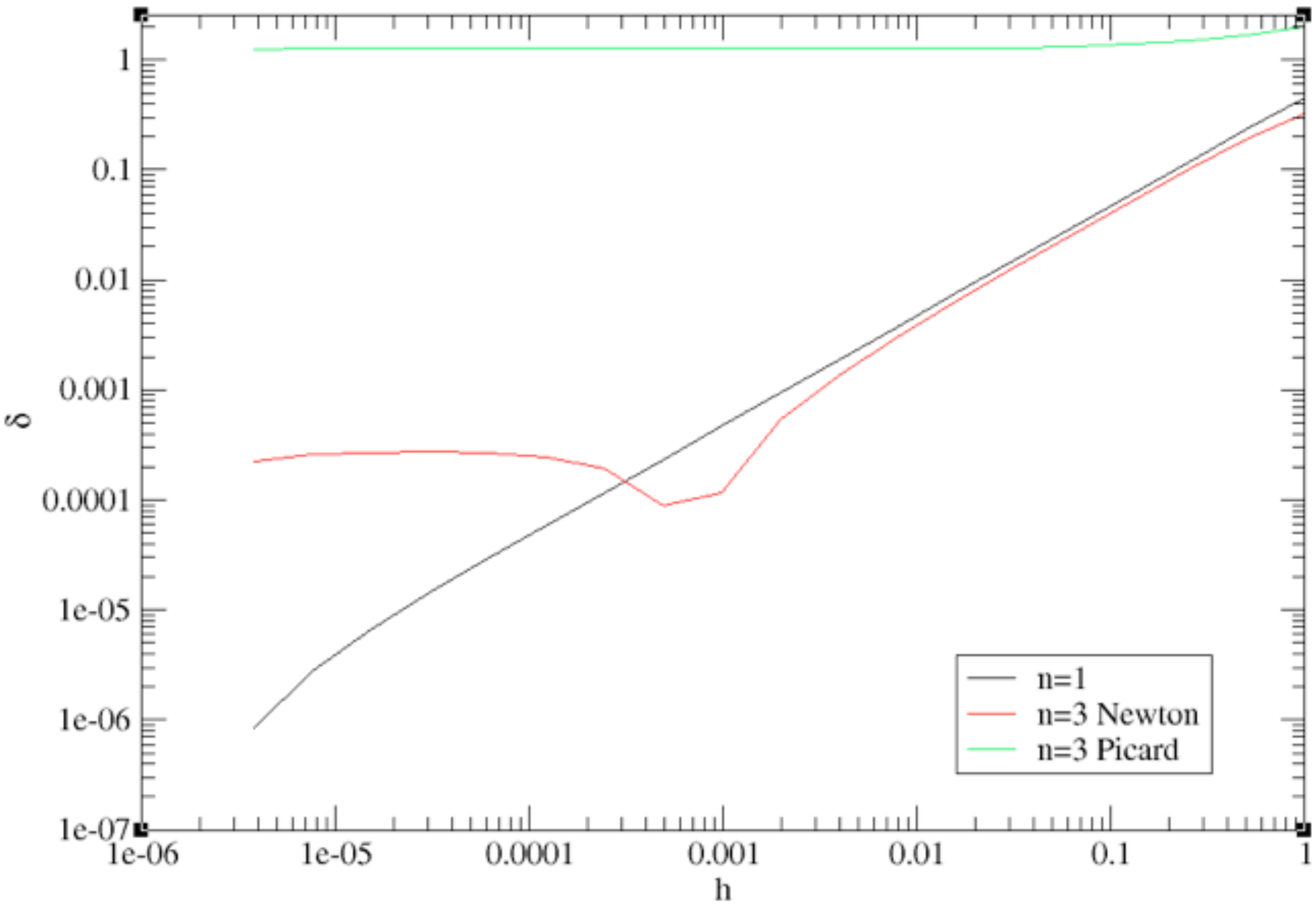
# Step 2 (OPTIONNAL) : Check the accuracy of the gradient

Validate the computation of the gradient with a finite difference scheme

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

```
!!!!! 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
```

# Step 2 (OPTIONNAL) : Check the accuracy of the gradient




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

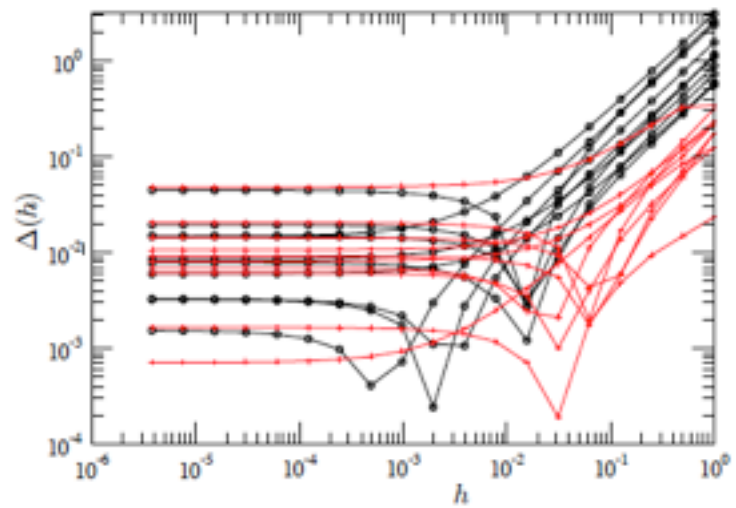
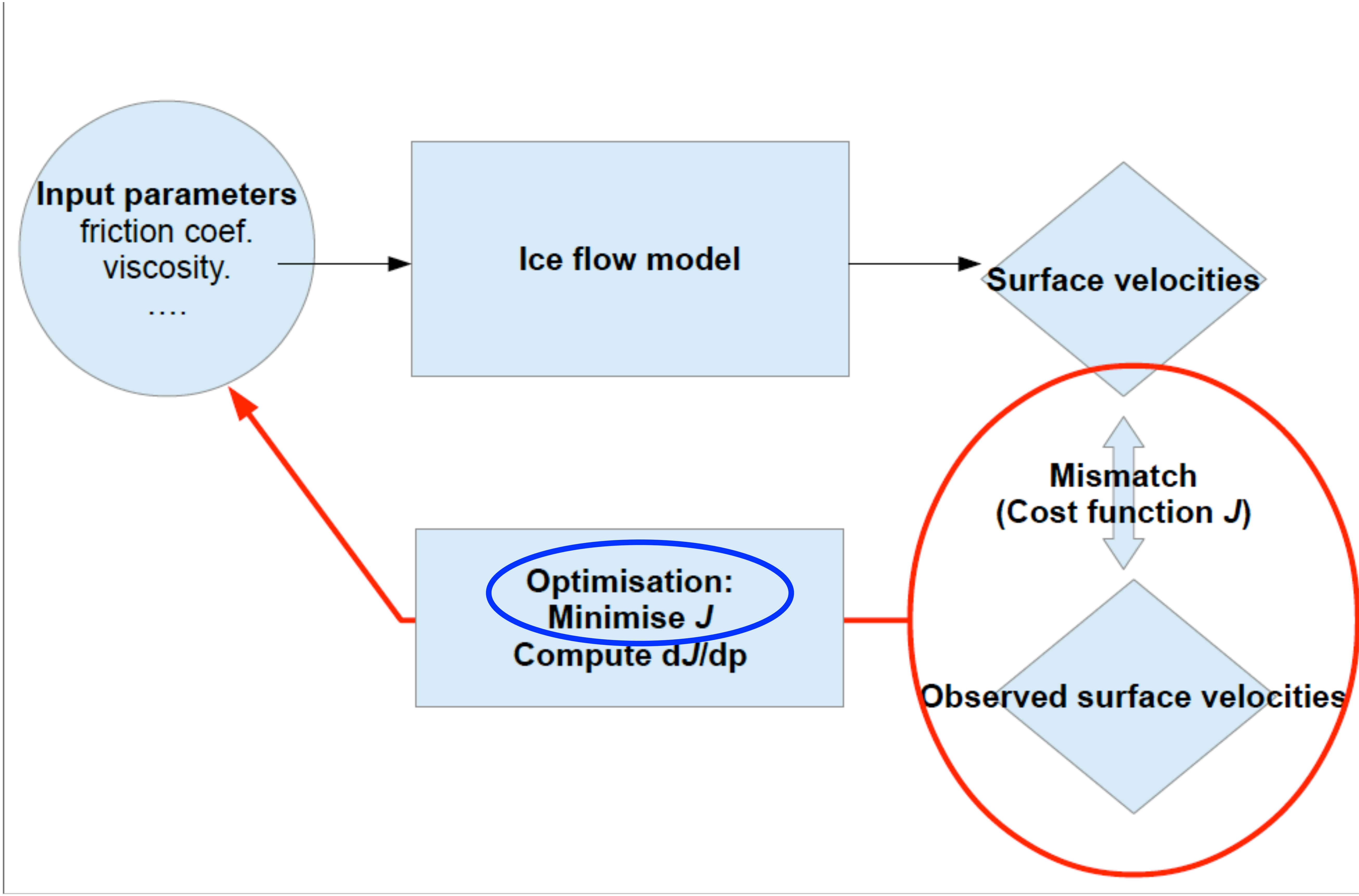


**Fig. 6.** Ratio $\Delta(h)$ obtained with the Robin (black curves) and control (red curves) inverse methods for perturbations of the enhancement factor to the viscosity $E_\eta(x, y)$.

# Step 3 : Minimise the cost function



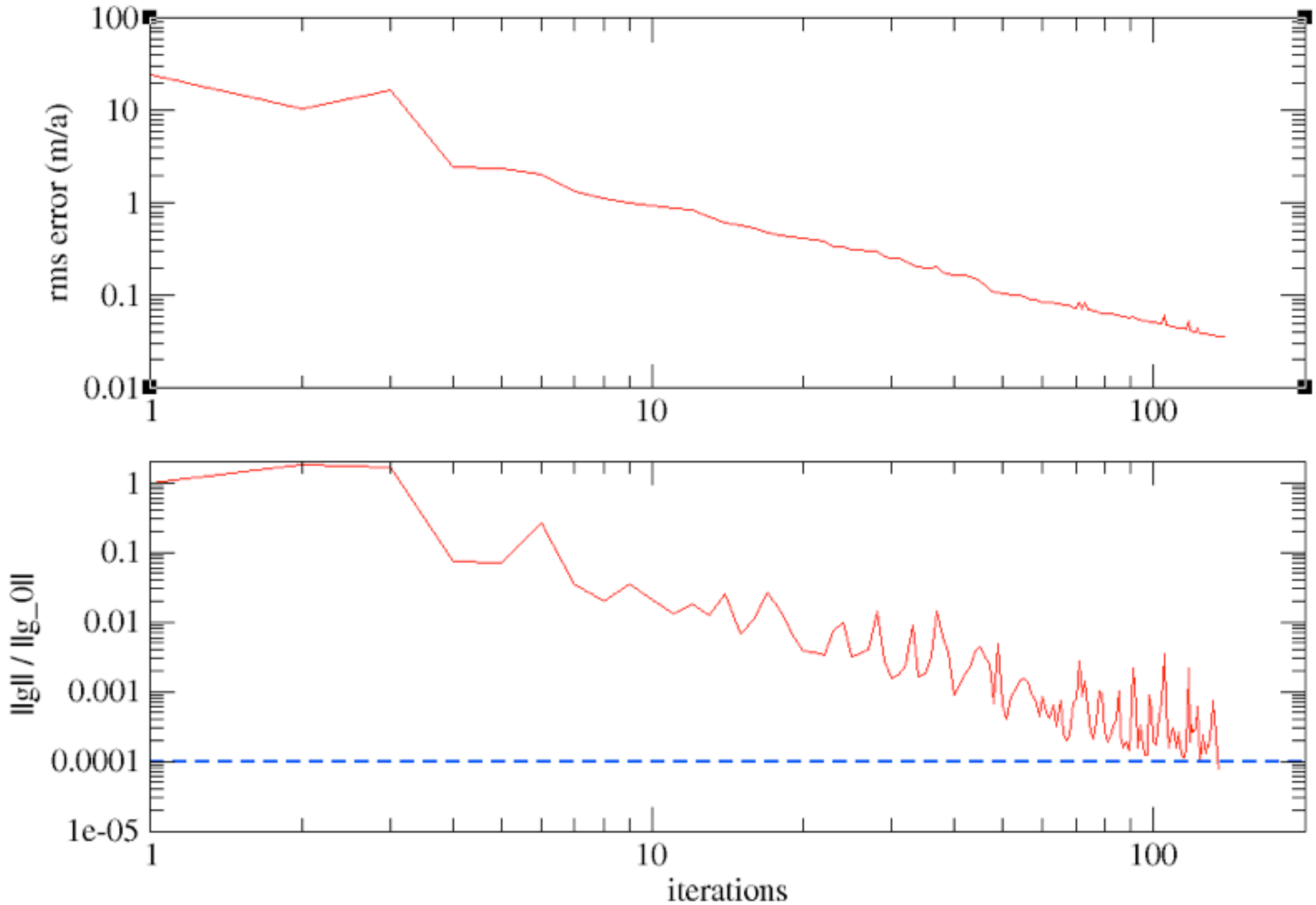
# Step 3 : Minimise the cost function

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 the cost function

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



# M1QN3 output: header

step4 — vim — 126x39

```
******* M1QN3 Output file ************
11/26/2015     15:39:17
 *****************************************
 -----------------------------------------------------------------------------
 M1QN3 Copyright (C) 2008, J. Ch. Gilbert, Cl. Lemarechal.
 -----------------------------------------------------------------------------
 This program comes with ABSOLUTELY NO WARRANTY. This is free software, and you
 are welcome to redistribute it under certain conditions. See the file COPYING
 in the root directory of the M1QN3 distribution for details.
 -----------------------------------------------------------------------------

 M1QN3 (Version 3.3, October 2009): entry point
     dimension of the problem (n):          1326
     absolute precision on x (dxmin):          1.00D-10
     expected decrease for f (df1):            1.47D+12
     relative precision on g (epsg):           1.00D-04 (dfn-norm)
     maximal number of iterations (niter):   400
     maximal number of simulations (nsim):   400
     printing level (impres):                  5
     reverse communication

 m1qn3: Scalar Initial Scaling mode

     allocated memory (ndz) :    57048
     used memory :               57038
     number of updates :            20
     (y,s) pairs are stored in core memory

 m1qn3: cold start

     f              =  2.93628863D+12
     dfn-norm of g =  1.62823848D+14

 m1qn3a: descent direction -g: precon =  0.111D-15

 -----------------------------------------------------------------------------

 m1qn3: iter 1, simul 1, f= 2.93628863D+12, h'(0)=-2.93629D+12
                                                                   1,2           Top
```

# M1QN3 output: output mode

step4 — vim — 126×39

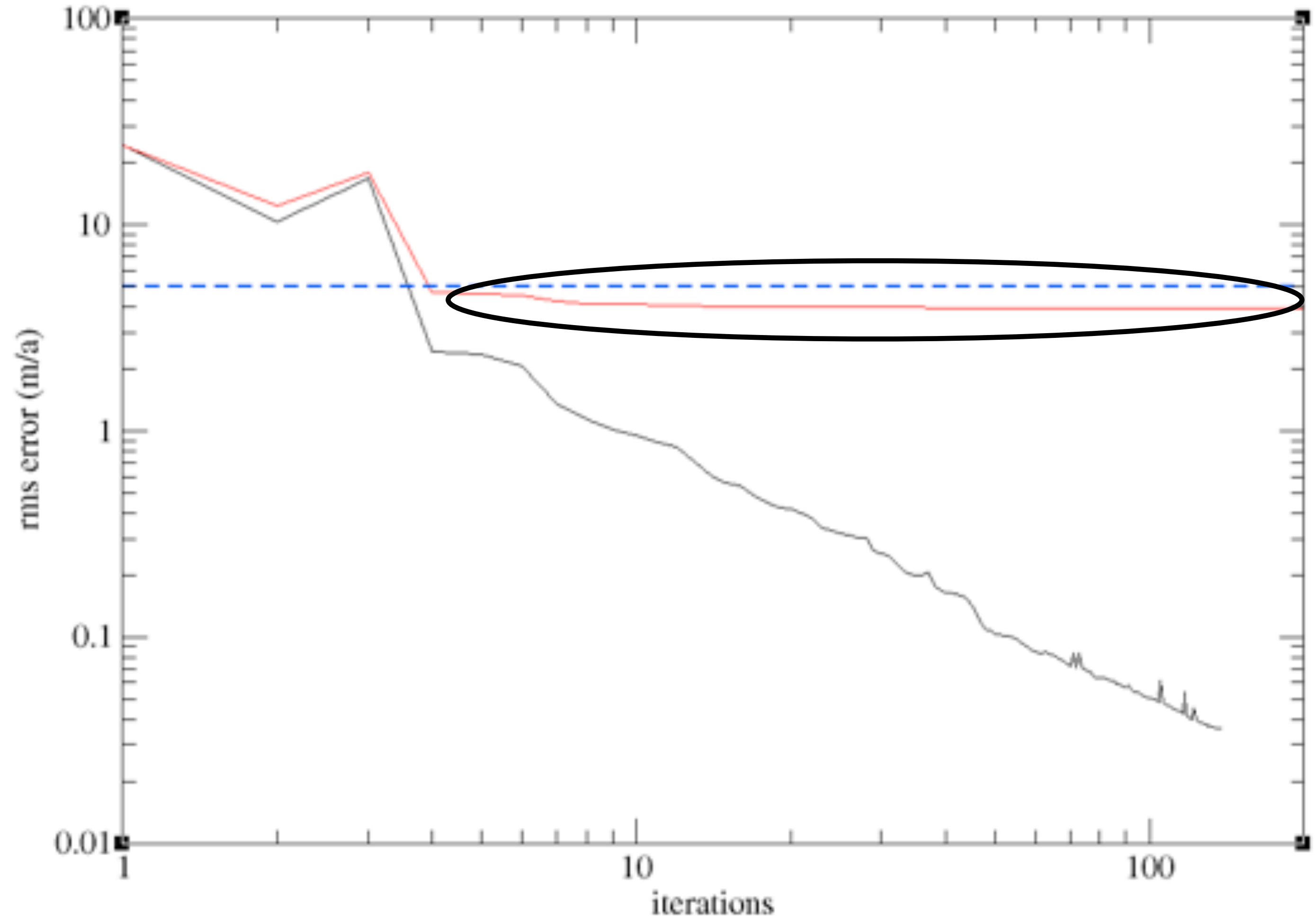
```
m1qn3: line search

    mlis3       fpn=-3.988D+05 d2= 1.40D-08  tmin= 6.90D-06 tmax= 1.00D+20
    mlis3     1.000D+00 -3.054D+05 -2.163D+05

m1qn3: convergence rate, s(k)/s(k-1) =  2.21598D+00

m1qn3: matrix update:
    Oren-Spedicato factor =  0.291D-15

m1qn3: stopping criterion on g:  1.91799D-04

m1qn3: descent direction d: angle(-g,d) =  83.1 degrees

--------------------------------------------------------------------------------

m1qn3: iter 96, simul 103, f= 1.24779848D+07, h'(0)=-5.19684D+05

m1qn3: line search

    mlis3       fpn=-5.197D+05 d2= 1.89D-08  tmin= 6.32D-06 tmax= 1.00D+20
    mlis3     1.000D+00 -3.384D+05 -1.661D+05

m1qn3: convergence rate, s(k)/s(k-1) =  1.16213D+00

m1qn3: matrix update:
    Oren-Spedicato factor =  0.278D-15

m1qn3: stopping criterion on g:  9.63919D-05

--------------------------------------------------------------------------------

m1qn3: output mode is  1
    number of iterations:            96
    number of simulations:          104
    realized relative precision on g:  9.64D-05
    f             =  1.21396118D+07
    dfn-norm of g =  1.56949054D+10
                                                                                         1774,6        Bot
```

# M1QN3 output: lost in line search

```
step4 — vim — 126×39
 m1qn3: stopping criterion on g:  2.63987D-04

 m1qn3: descent direction d: angle(-g,d) =  87.9 degrees

 ---------------------------------------------------------------------------

 m1qn3: iter 485, simul 510, f= 4.07807663D+05, h'(0)=-6.26337D+03

 m1qn3: line search

     mlis3       fpn=-6.263D+03 d2= 4.23D-02  tmin= 6.36D-09 tmax= 1.00D+20
     mlis3                                      1.000D+00  1.761D+04  3.281D+04
     mlis3                                      1.003D-01  2.297D+03  4.656D+04
     mlis3                                      1.003D-03  7.624D-01 -6.266D+03
     mlis3                                      9.525D-04  7.180D-01 -6.266D+03
     mlis3                                      7.144D-04  5.165D-01 -6.266D+03
     mlis3                                      5.001D-04  3.479D-01 -6.265D+03
     mlis3                                      3.500D-04  2.369D-01 -6.265D+03
     mlis3                                      2.450D-04  1.626D-01 -6.264D+03
     mlis3                                      1.715D-04  1.123D-01 -6.264D+03
     mlis3                                      1.201D-04  7.784D-02 -6.264D+03
     mlis3                                      8.405D-05  5.411D-02 -6.264D+03
     mlis3                                      5.883D-05  3.770D-02 -6.264D+03
     mlis3                                      4.118D-05  2.630D-02 -6.264D+03
     mlis3                                      2.883D-05  1.837D-02 -6.264D+03
     mlis3                                      2.018D-05  1.284D-02 -6.263D+03
     mlis3                                      1.413D-05  8.974D-03 -6.263D+03
     mlis3                                      9.888D-06  6.277D-03 -6.263D+03
     mlis3                                      6.922D-06  4.391D-03 -6.263D+03
     mlis3                                      4.845D-06  3.073D-03 -6.263D+03
     mlis3                                      3.392D-06  2.150D-03 -6.263D+03
     mlis3                                      2.374D-06  1.505D-03 -6.263D+03
     mlis3                                      1.662D-06  1.053D-03 -6.263D+03
     mlis3                                      1.163D-06  7.372D-04 -6.263D+03
     mlis3                                      8.143D-07  5.160D-04 -6.263D+03
     mlis3                                      5.700D-07  3.612D-04 -6.263D+03
     mlis3                                      3.990D-07  2.528D-04 -6.263D+03
     mlis3                                      2.793D-07  1.770D-04 -6.263D+03
"MC_Vallot/M1QN3_hoptim_l1.out" 8804L, 256279C                                        8804,6        Bot
```

# *Step 4 : «noisy» observations*

The cost function stop to decrease
The inverse field becomes «noisy»




Remedies :

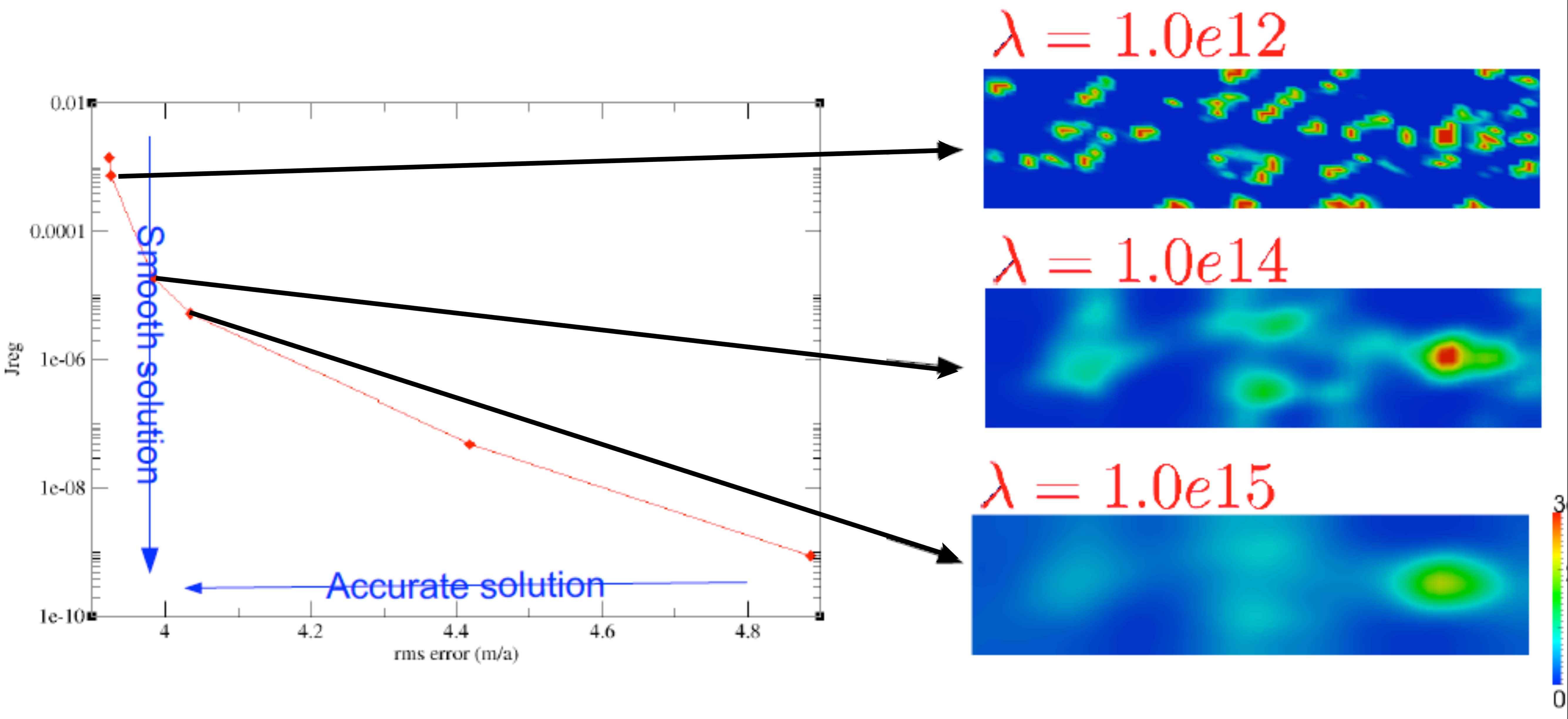
1. Stop when you reach your rms error (i.e. avoid over-fitting)
(cf e.g. Arthern and Gudmundsson, 2010)

2. Add a regularisation term to the cost function $$J_{tot} = J_0 + \lambda J_{reg}$$

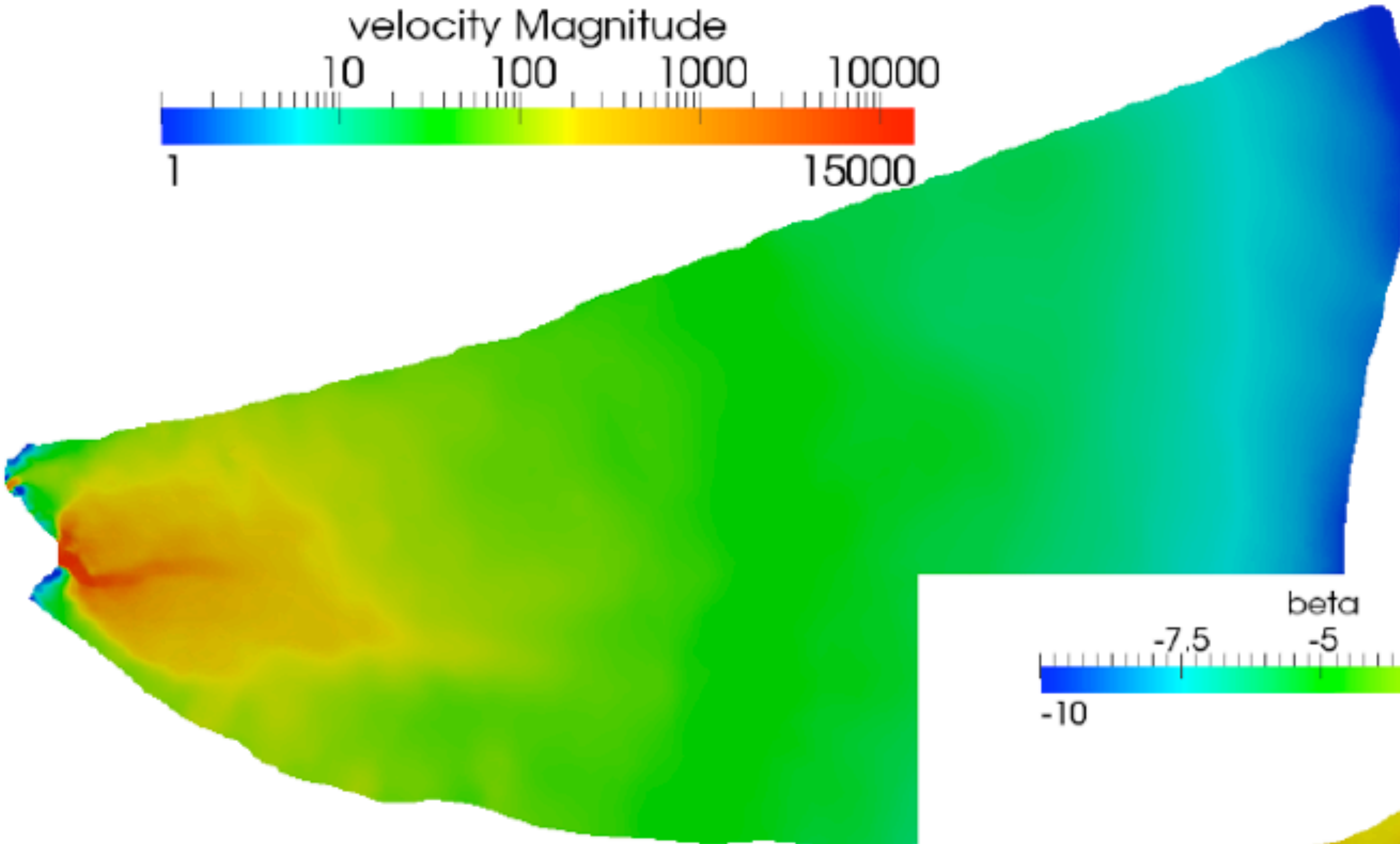
Here, penalise spatial derivatives of the input parameter: $$J_{reg} = \frac{1}{2}\int_{\Gamma_b}\left(\frac{d\beta}{dx}\right)^2$$

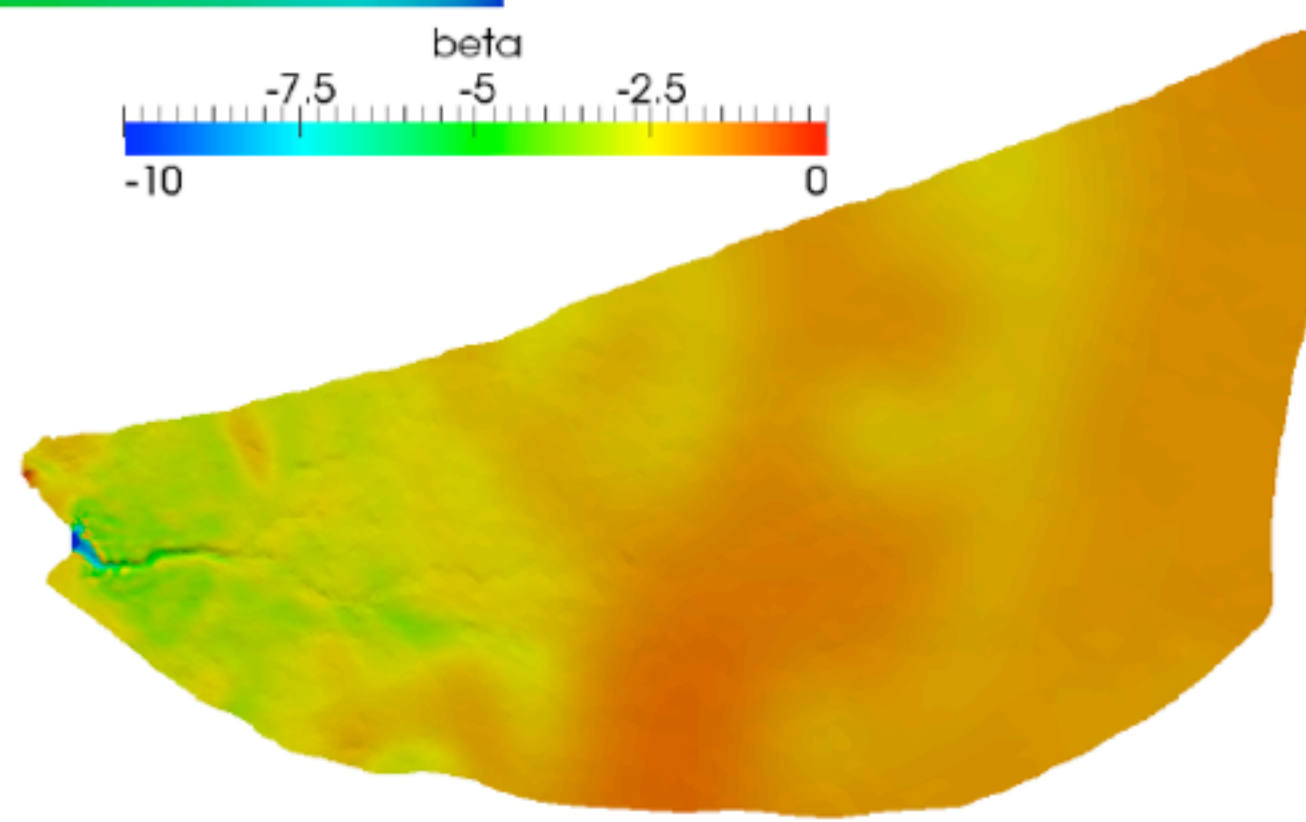
# Step 4 : «noisy» observations

L-Curve analyses



# Step 6: A Real case application Jacobshavn Isbrae



# *Outline*

- **Short introduction**
- **Inverse methods in Elmer/Ice (Stokes solver)**
- **Current / planned developments**

# *1. Adjoint of the shallow shelf model*

## Field equations:

$$\begin{cases} \dfrac{\partial}{\partial x}\left(2H\nu\left(2\dfrac{\partial u}{\partial x}+\dfrac{\partial v}{\partial y}\right)\right)+\dfrac{\partial}{\partial y}\left(H\nu\left(\dfrac{\partial v}{\partial x}+\dfrac{\partial u}{\partial y}\right)\right)-\beta u=\rho g H\dfrac{\partial z_s}{\partial x} \\ \dfrac{\partial}{\partial x}\left(H\nu\left(\dfrac{\partial v}{\partial x}+\dfrac{\partial u}{\partial y}\right)\right)+\dfrac{\partial}{\partial y}\left(2H\nu\left(\dfrac{\partial u}{\partial x}+2\dfrac{\partial v}{\partial y}\right)\right)-\beta v=\rho_i g H\dfrac{\partial z_s}{\partial y} \end{cases}$$

Work already done, see applications

- ***Fürst et al.*, Assimilation of Antarctic velocity observations provides evidence for uncharted pinning points, *The Cryosphere*, 2015**
- ***Fürst et al.,* Passive shelf ice: the safety band of Antarctic ice 1shelves, *Nature Climate Change*, accepted**

still requires some cleaning/re-arrangement before submission to the Elmer/Ice distrib

# 2. Adjoint of the thickness solver

see Morlighem *et al.*, 2011, **a mass consservation approach for mapping glacier ice thickness**

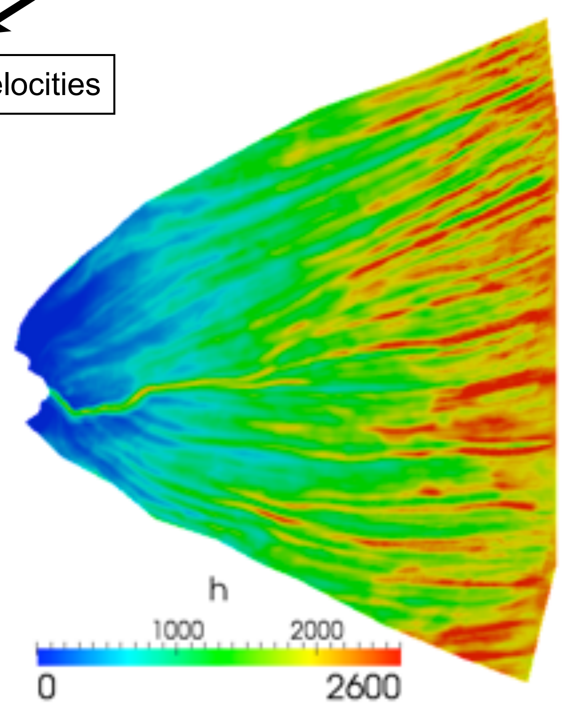
Work already done, D. Vallot is using it. Still requires some cleaning/re-arrangement before submission to the Elmer/Ice distrib.

1. Compute the blance thickness

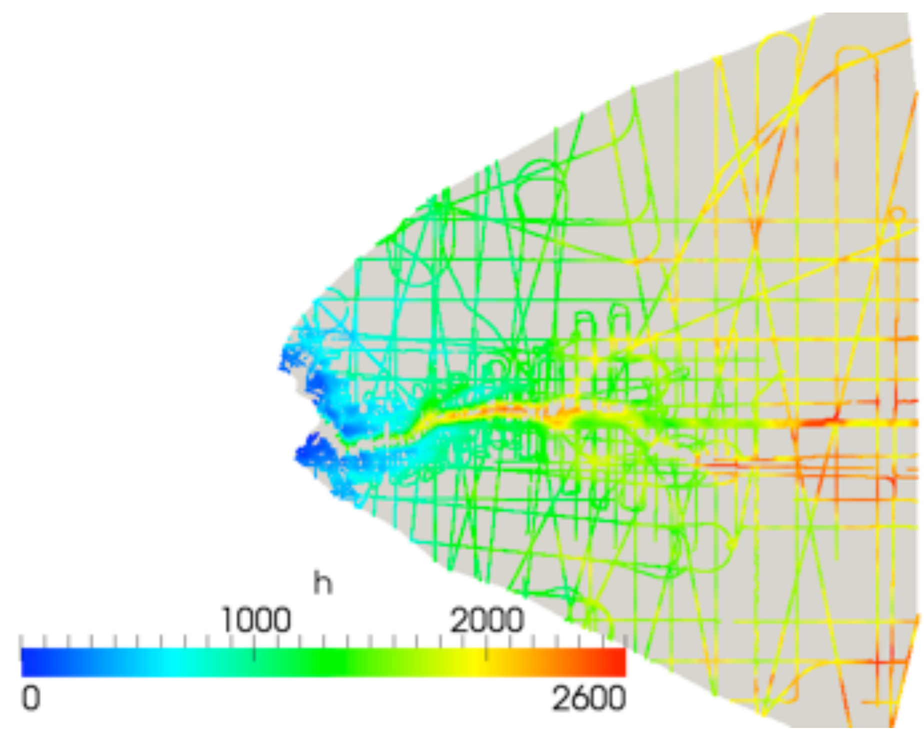
$$\nabla(\bar{u}H) = a_s$$

Effective smb (model smb+dhdt)

Observed surface velocities



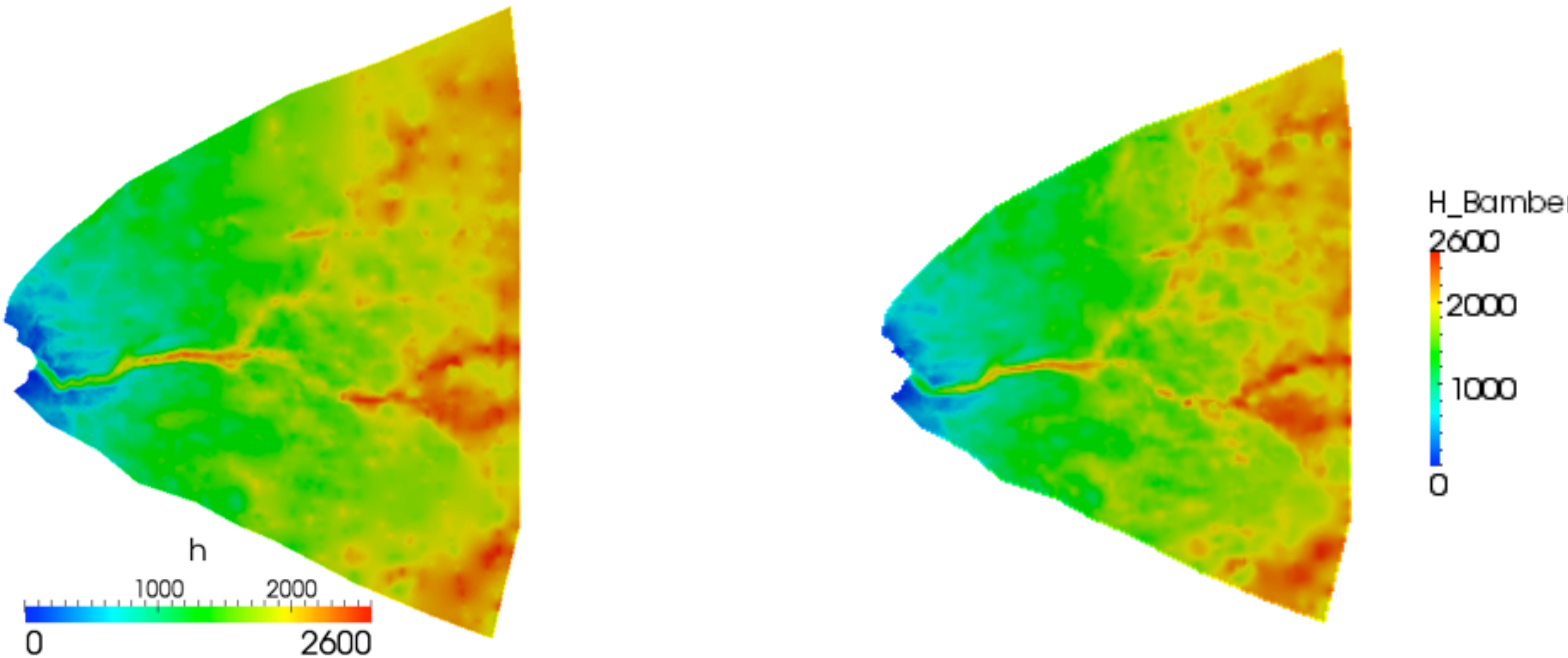

2. Compare with observations




$$J(\bar{u}, a_s) = \sum_{1}^{N_{obs}} 0.5(H - H_{obs})^2 + \lambda J_{Reg}$$

# 2. Adjoint of the thickness solver

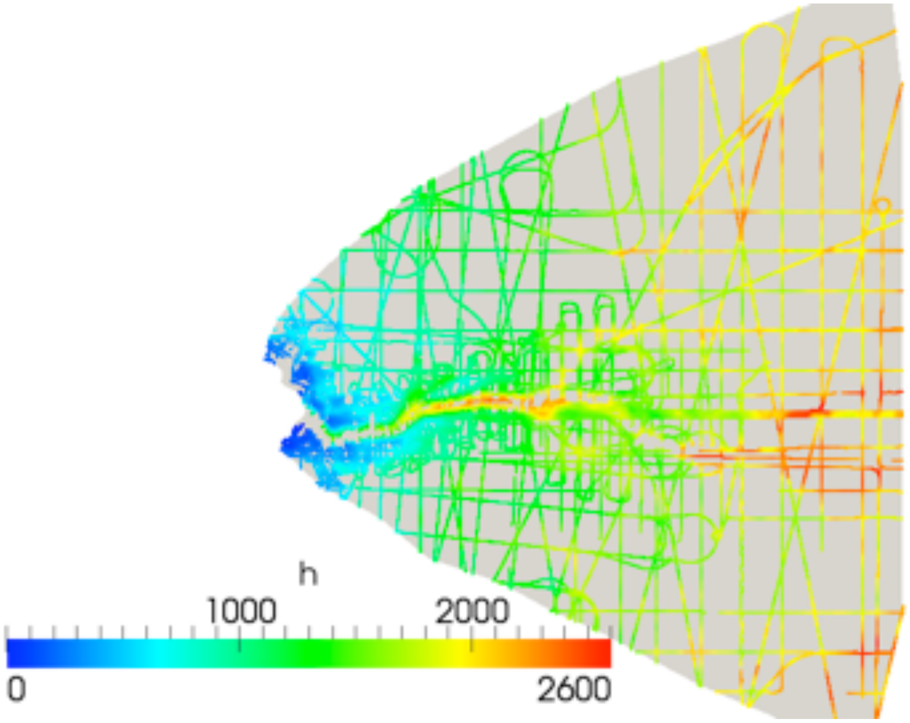
3. Use the adjoint to optimise u and a and reduced mismatch



# 3. Evaluate cost where observations are availabe

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

$$J = \sum_{1}^{N_{obs}} 0.5(u - u^{obs})^2$$




=> the model results are interpolated where observations are available (using the FE basis functions)

## *4. Couple optimisation of bedrock elevation and slip coef.*

phD work of C. Mosbeux (LGGE)

=> SSA adjoint model (gradient w.r.t. zb and beta)

=> adjoint beta + controlled relaxation of zb