Adjoint-based PDE-constrained optimization using HPC techniques

Oscar F. Peredo

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A thesis submitted for the *Master's Degree in Information Technology*

October, 2013 - Barcelona, Spain

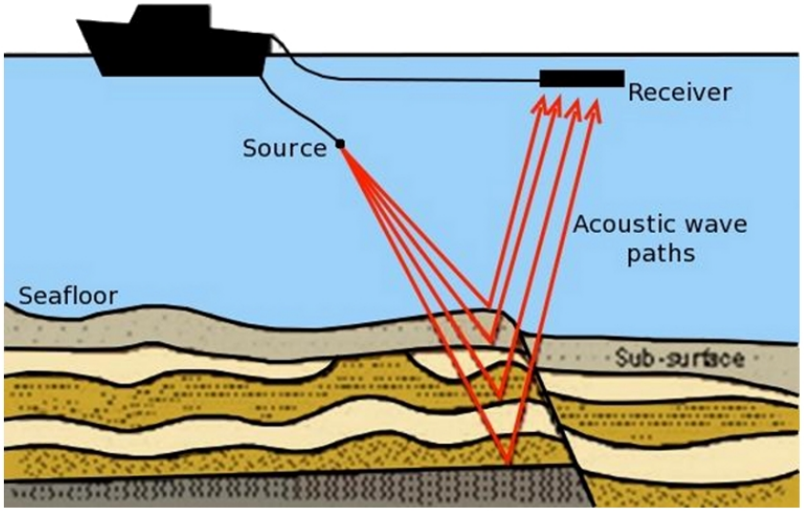
Outline

Outline

Motivation

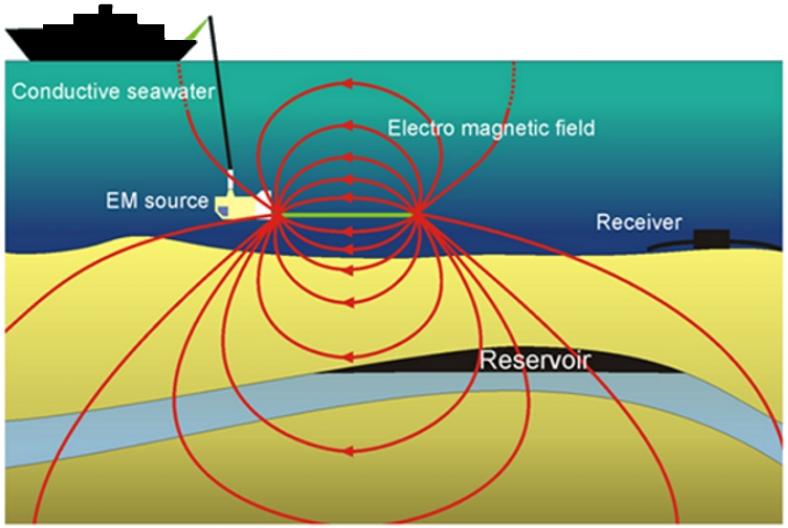
Inverse problems (Exploration Geophysics)

Seismic methods



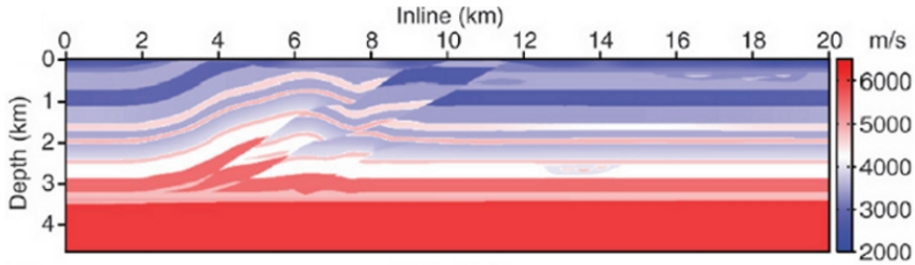
Source: www.soes.soton.ac.uk

Electromagnetic methods

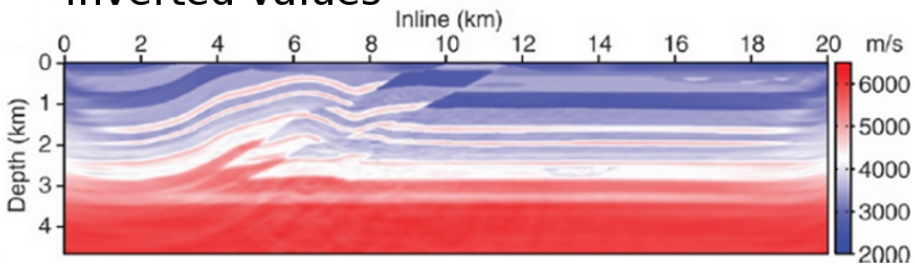


Source: noc.ac.uk

True model



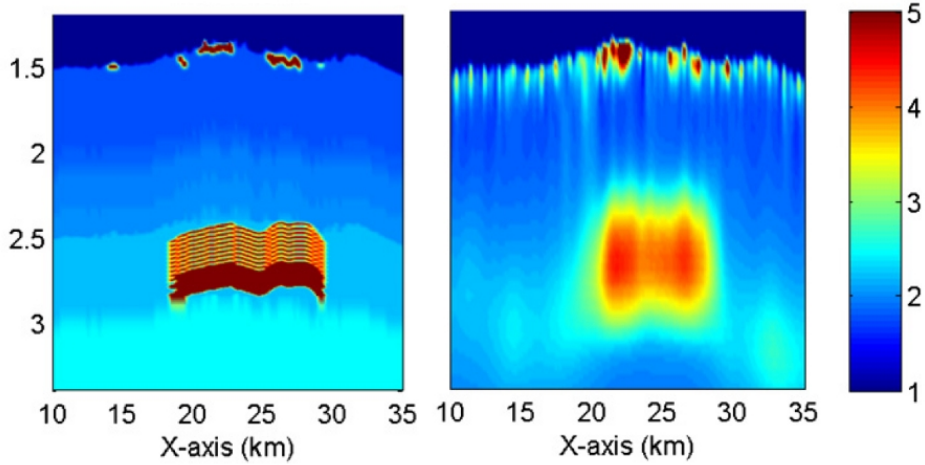
Inverted values



Source: Ben-Hadj-Ali, Operto, Virieux (2011)

True model

Inverted values



Source: Plessix, Mulder (2008)

Outline

PDE-constrained optimization

$$\begin{array}{ll} \underset{(\mathbf{u},\mathbf{d})\in\mathbb{R}^{n_u}\times\mathbb{R}^{n_d}}{\text{minimize}} & J(\mathbf{u},\mathbf{d}) \\ \text{subject to} & \mathbf{R}(\mathbf{u},\mathbf{d}) = \mathbf{0} \end{array} \qquad \text{(PDECO)}$$

with

- $J : \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \to \mathbb{R}$ a discrete cost function,
- $\mathbf{R} : \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \to \mathbb{R}^{n_u}$ a discretized PDE + boundary conditions, in our case we will work only with stationary linear models + Dirichlet conditions:

$$\mathbf{R}(\mathbf{u},\mathbf{d}) = \mathbf{A}(\mathbf{d})\mathbf{u} - \mathbf{b}(\mathbf{d})$$

- $\mathbf{u}$ *state* vector (solution of the discrete PDE),
- $\mathbf{d}$ *design* vector (input parameters of the PDE)

PDE-constrained optimization

How can we solve PDECO?

PDE-constrained optimization

How can we solve PDECO?

Reducing the problem to an unconstrained minimization.

$$\underset{\mathbf{d}\in\mathbb{R}^{n_d}}{\text{minimize}} \qquad j(\mathbf{d}) := J(\mathbf{u}(\mathbf{d}), \mathbf{d}) \qquad \text{(REDUCED-PDECO)}$$

with $\mathbf{u}(\mathbf{d})$ solution of $\mathbf{A}(\mathbf{d})\mathbf{u} = \mathbf{b}(\mathbf{d})$ (implicit dependency).

PDE-constrained optimization

How can we solve PDECO?

Reducing the problem to an unconstrained minimization.

$$\underset{\mathbf{d}\in\mathbb{R}^{n_d}}{\text{minimize}} \qquad j(\mathbf{d}) := J(\mathbf{u}(\mathbf{d}), \mathbf{d}) \qquad \text{(REDUCED-PDECO)}$$

with $\mathbf{u}(\mathbf{d})$ solution of $\mathbf{A}(\mathbf{d})\mathbf{u} = \mathbf{b}(\mathbf{d})$ (implicit dependency).

How can we solve REDUCED-PDECO?

PDE-constrained optimization

How can we solve PDECO?

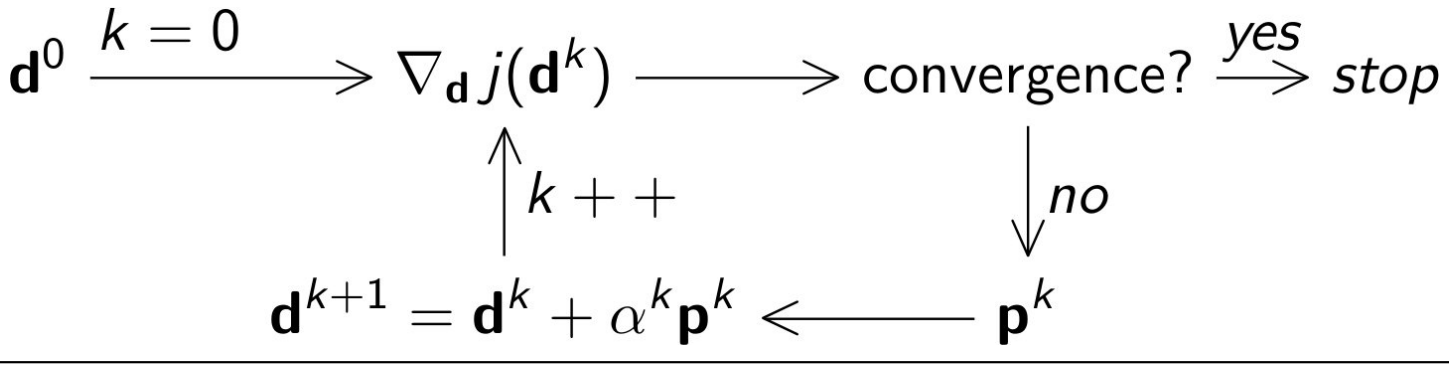
Reducing the problem to an unconstrained minimization.

$$\underset{\mathbf{d}\in\mathbb{R}^{n_d}}{\text{minimize}} \qquad j(\mathbf{d}) := J(\mathbf{u}(\mathbf{d}), \mathbf{d}) \qquad \text{(REDUCED-PDECO)}$$

with $\mathbf{u}(\mathbf{d})$ solution of $\mathbf{A}(\mathbf{d})\mathbf{u} = \mathbf{b}(\mathbf{d})$ (implicit dependency).

How can we solve REDUCED-PDECO?

Gradient-based methods (descent direction + line-search):



Outline

Objectives

- Propose a framework to solve PDECO through a gradient-based resolution of REDUCED-PDECO. The governing PDE must represent a stationary linear physical model.

- The proposed framework must be designed on top of a parallel multi-physics PDE-simulator called *Alya*. It will be able to:
  - calculate cost functions $j(\mathbf{d})$ defined by the user
  - calculate reduced gradients $\nabla_{\mathbf{d}} j(\mathbf{d}^k)$ with an *adjoint approach*
  - calculate descent directions $\mathbf{p}^k$ and step lengths α^k

Objectives

- Propose a framework to solve PDECO through a gradient-based resolution of REDUCED-PDECO. The governing PDE must represent a stationary linear physical model.

- The proposed framework must be designed on top of a parallel multi-physics PDE-simulator called *Alya*. It will be able to:
  - calculate cost functions $j(\mathbf{d})$ defined by the user
  - **calculate reduced gradients $\nabla_{\mathbf{d}}\, j(\mathbf{d}^k)$ with an *adjoint approach***
  - calculate descent directions $\mathbf{p}^k$ and step lengths α^k

Reduced gradient: adjoint approach

Input: $\mathbf{d}$ design vector
Output: $\nabla_{\mathbf{d}} j(\mathbf{d})$

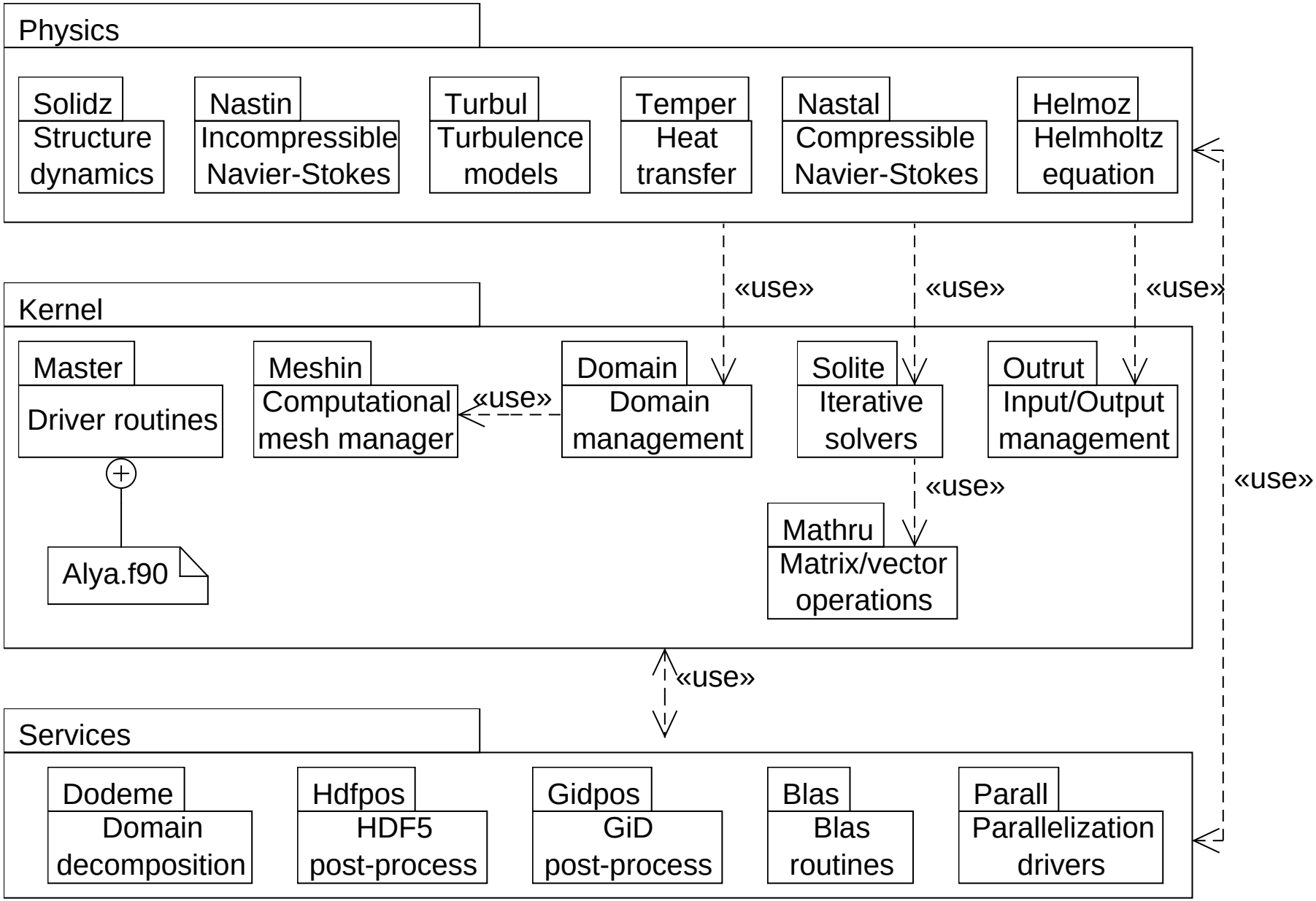
1. $\mathbf{u} \leftarrow$ solve $\mathbf{A}(\mathbf{d})\mathbf{u} = \mathbf{b}(\mathbf{d})$; (*forward problem*)
2. Obtain explicit derivative $\mathbf{c}(\mathbf{u}, \mathbf{d})^T := \nabla_{\mathbf{u}} J(\mathbf{u}, \mathbf{d})$;
3. $\boldsymbol{\lambda} \leftarrow$ solve $\mathbf{A}(\mathbf{d})^T \boldsymbol{\lambda} = \mathbf{c}(\mathbf{u}, \mathbf{d})$; (*adjoint problem*)
4. Obtain explicit derivatives $\nabla_{\mathbf{d}} J(\mathbf{u}, \mathbf{d})$ and $\nabla_{\mathbf{d}} \mathbf{R}(\mathbf{u}, \mathbf{d})$;
5. $\nabla_{\mathbf{d}} j(\mathbf{d}) = -\boldsymbol{\lambda}^T \nabla_{\mathbf{d}} \mathbf{R}(\mathbf{u}, \mathbf{d}) + \nabla_{\mathbf{d}} J(\mathbf{u}, \mathbf{d})$;

If $\mathbf{d}$ and $\mathbf{u}$ are large enough, this algorithm is cheaper in terms of FLOPS w.r.t. other alternatives.

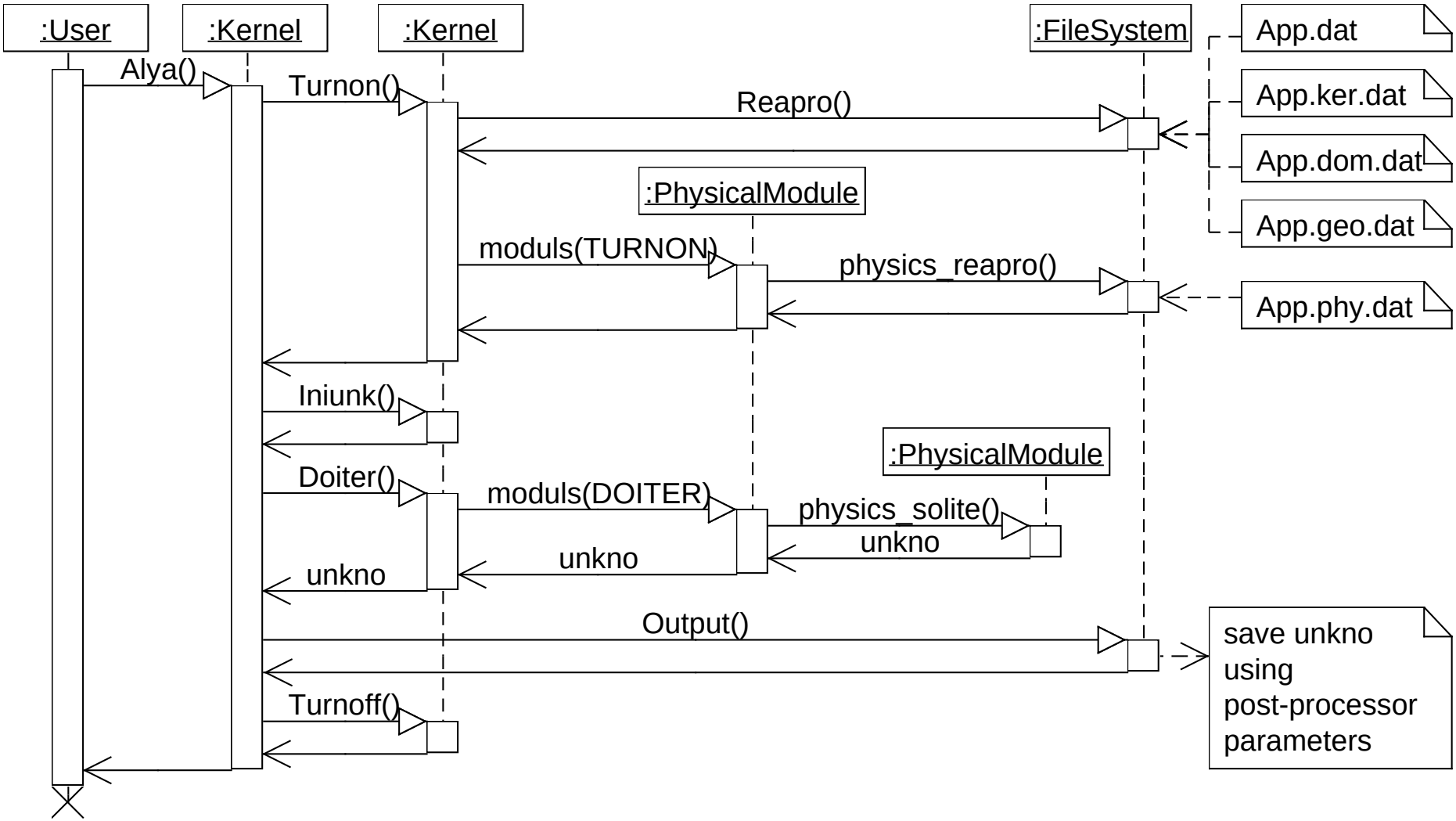
Outline

Outline

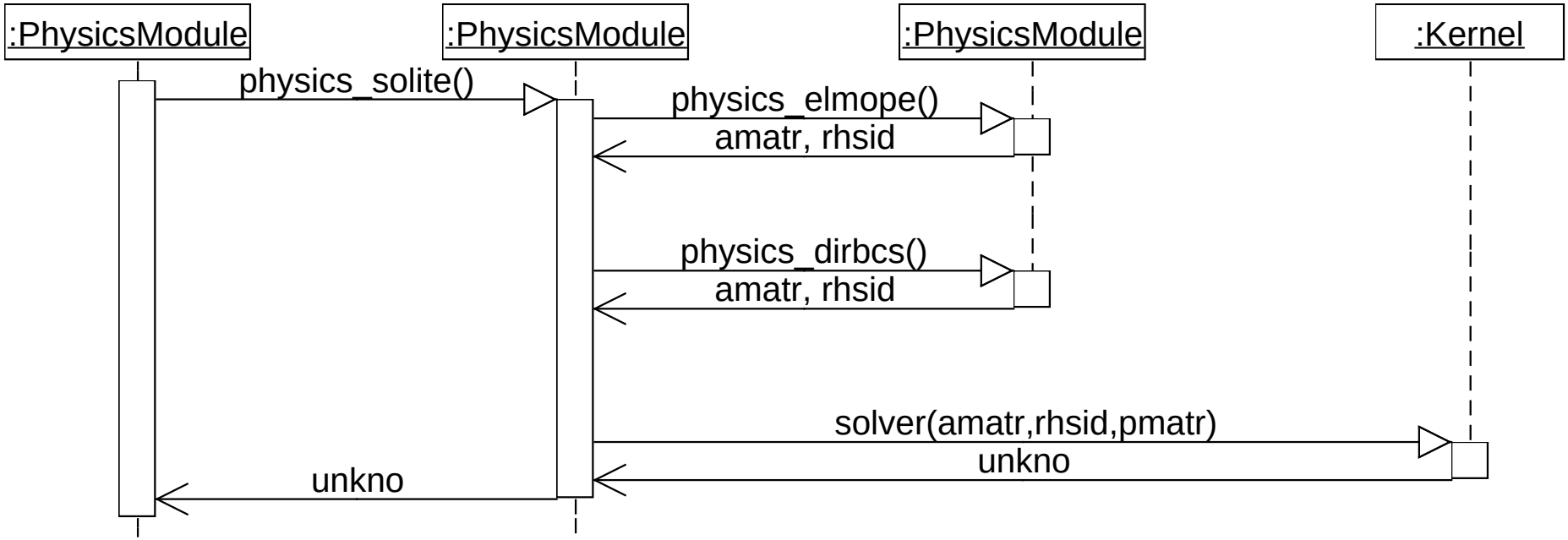
Alya: architecture (simplified)



Alya: main driver `Alya.f90`

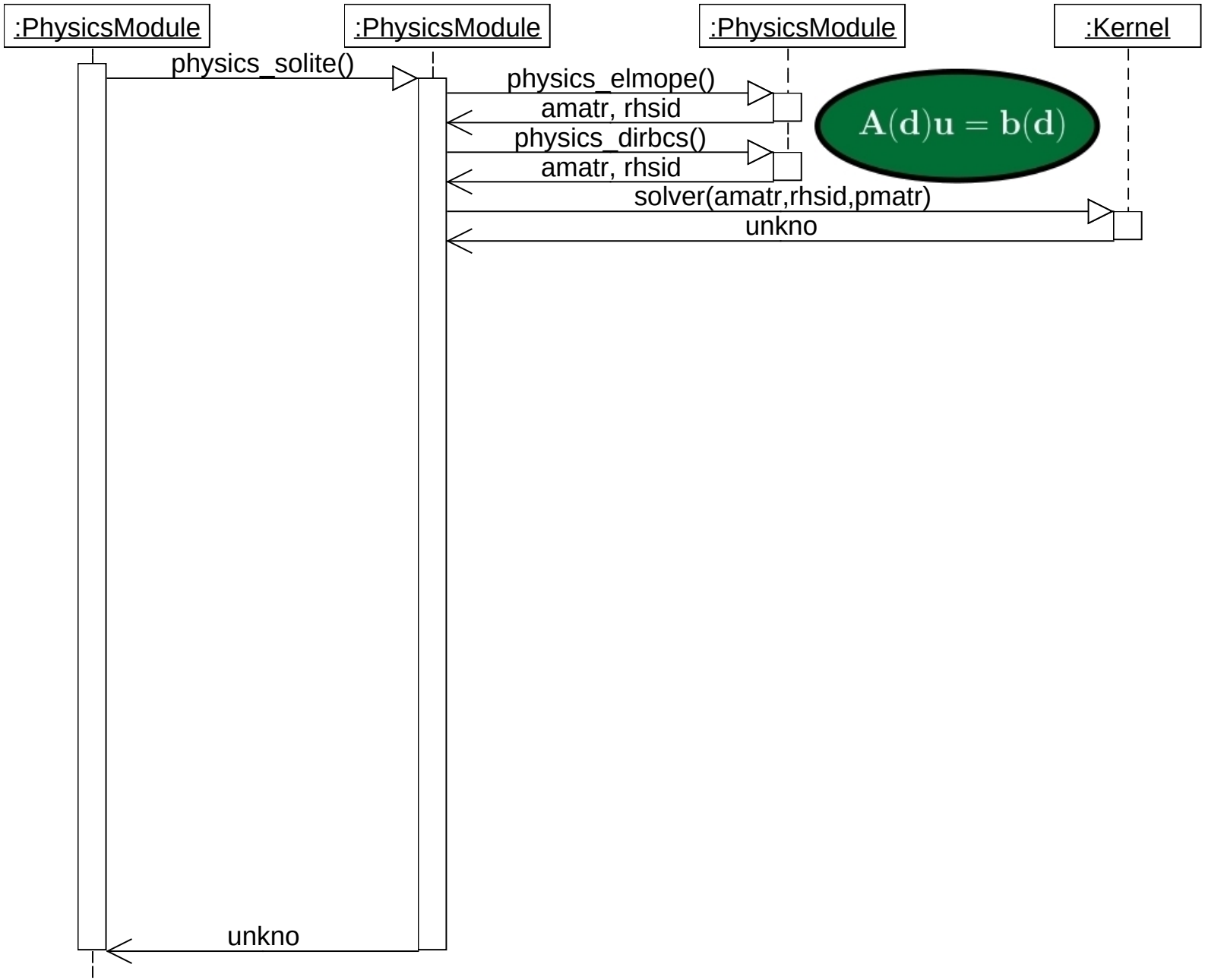


Alya: assemble/solve $\mathbf{Au} = \mathbf{b}$

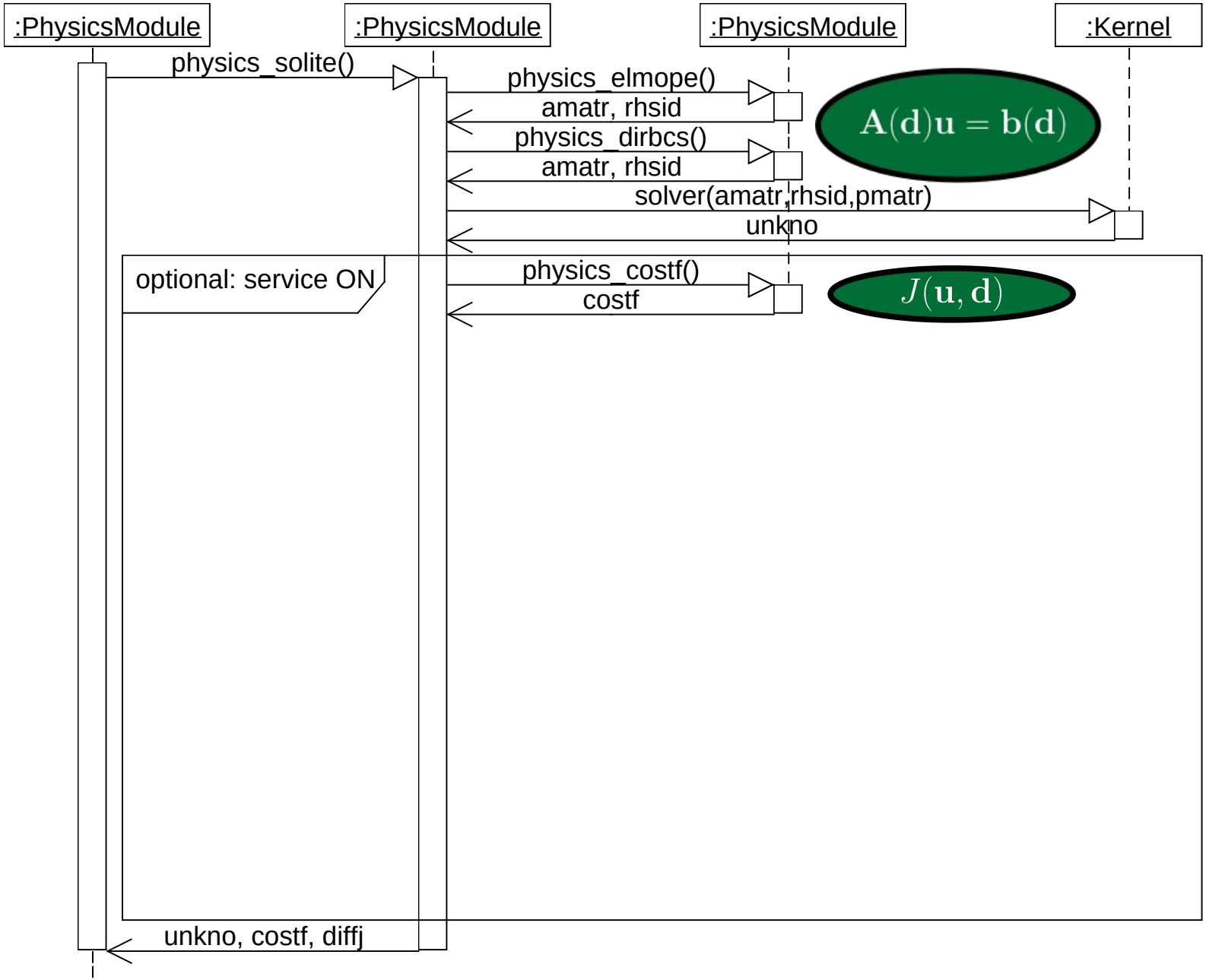


Outline

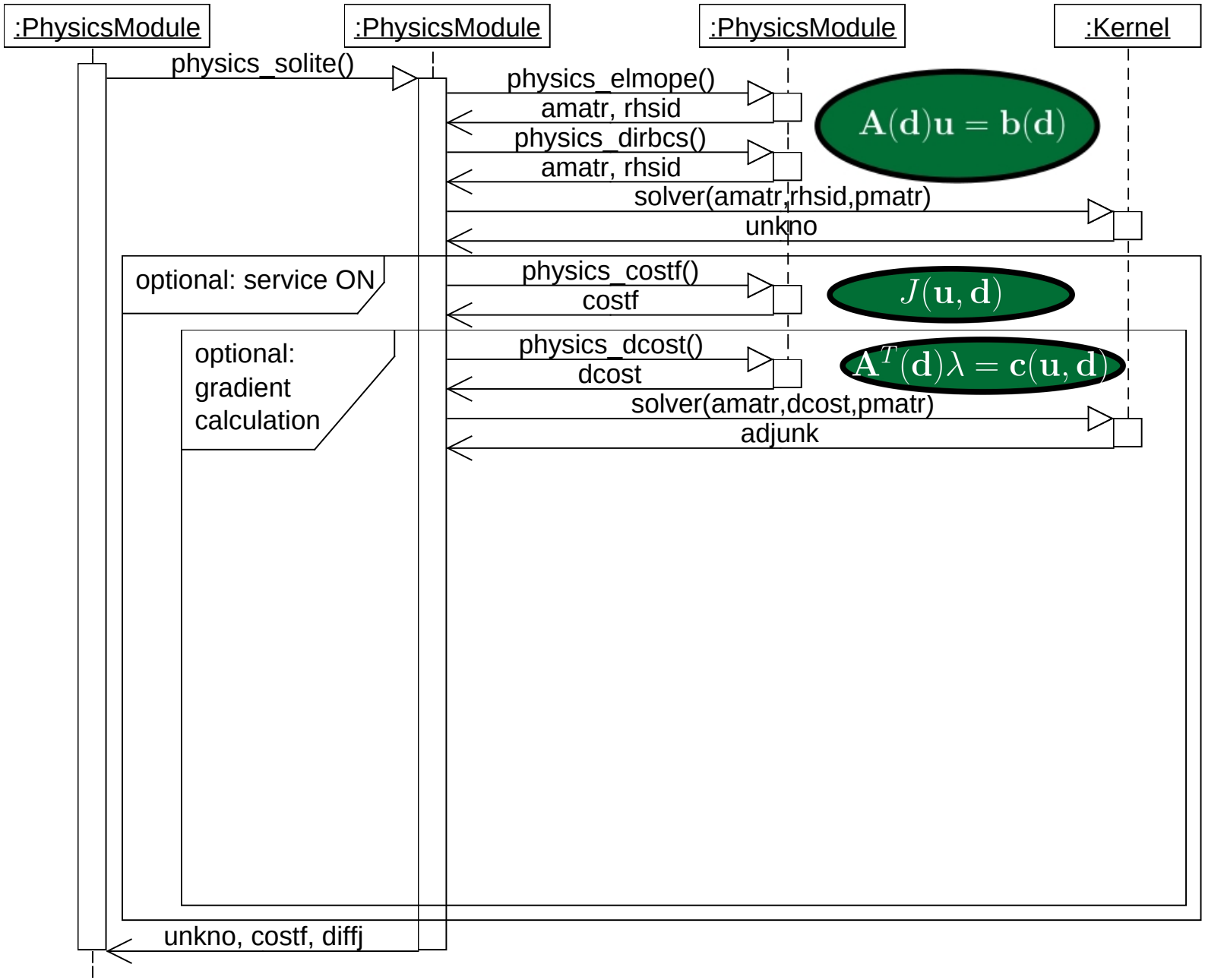
Alya: assemble/solve $\mathbf{Au} = \mathbf{b}$



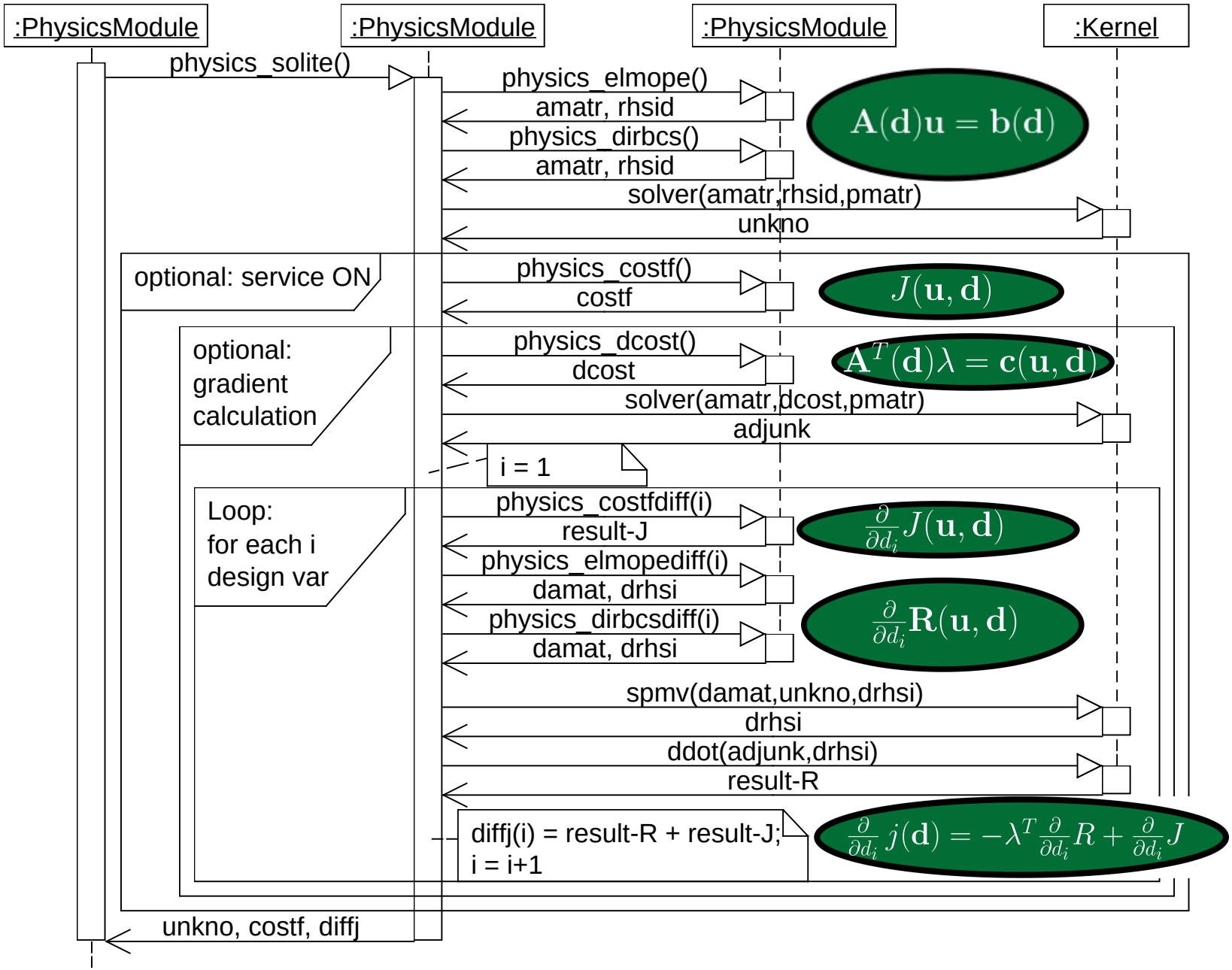
Alya: assemble/solve $\mathbf{Au} = \mathbf{b}$ + reduced gradient



Alya: assemble/solve **Au** = **b** + reduced gradient

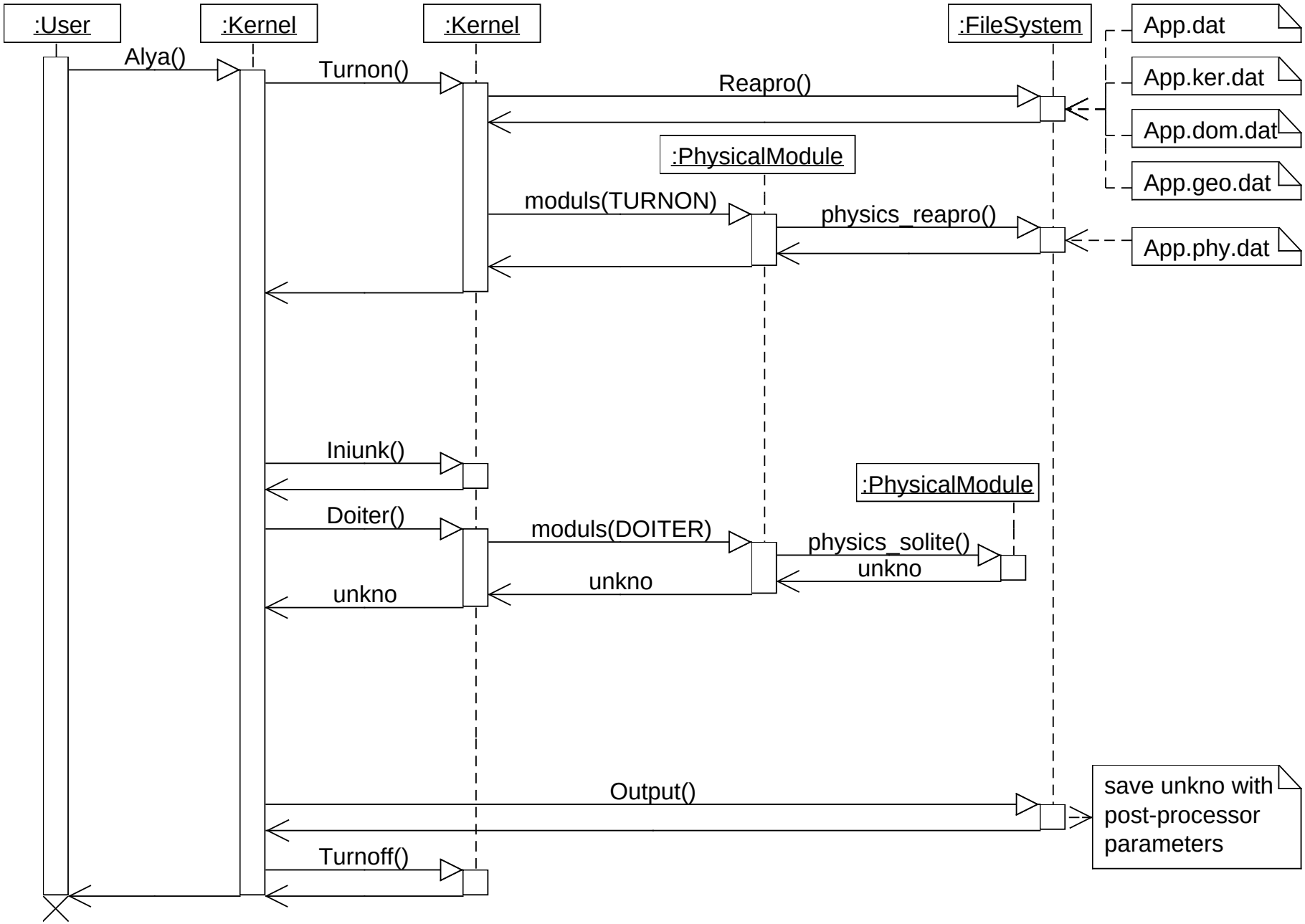


Alya: assemble/solve **Au** = **b** + reduced gradient

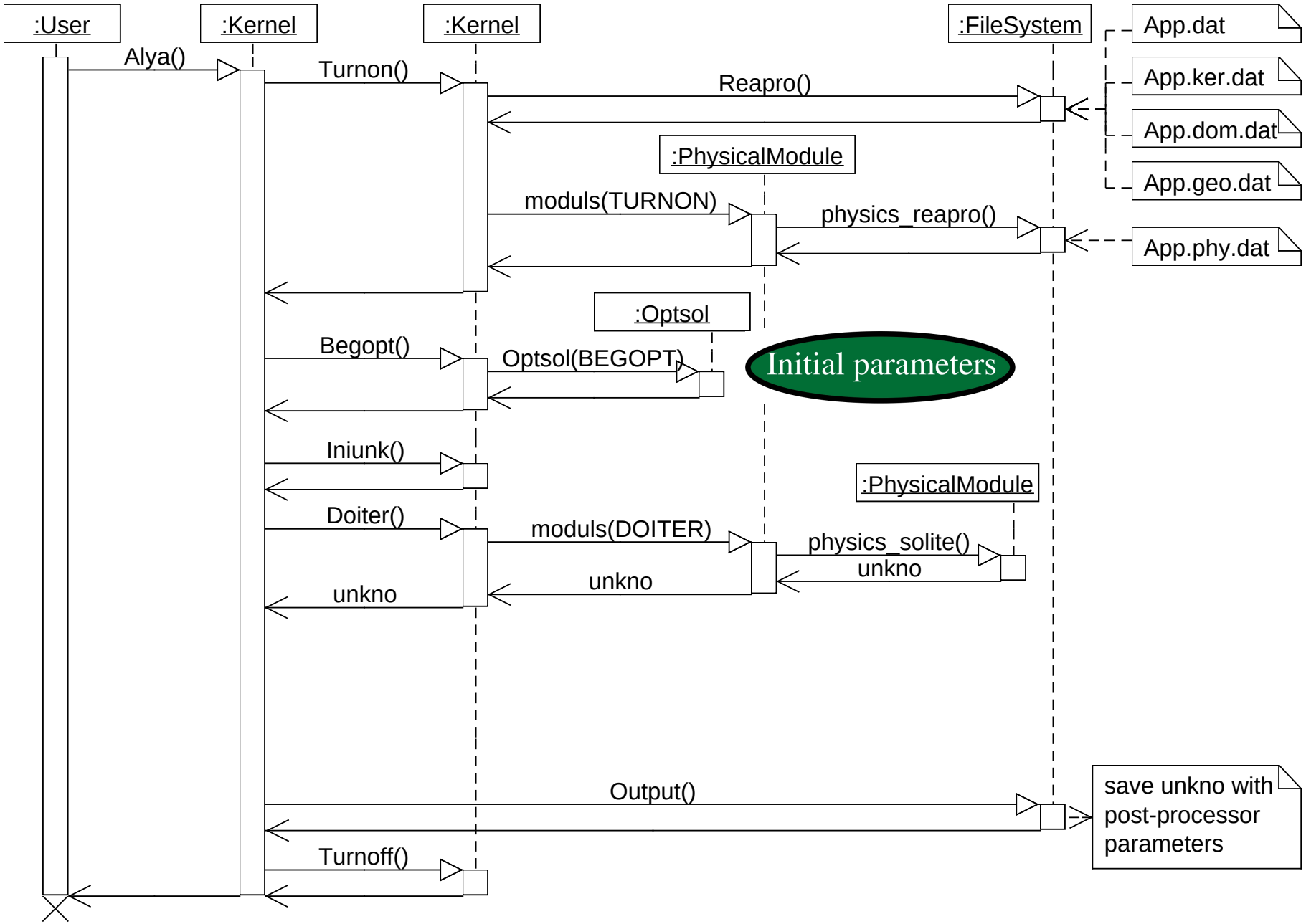


Outline

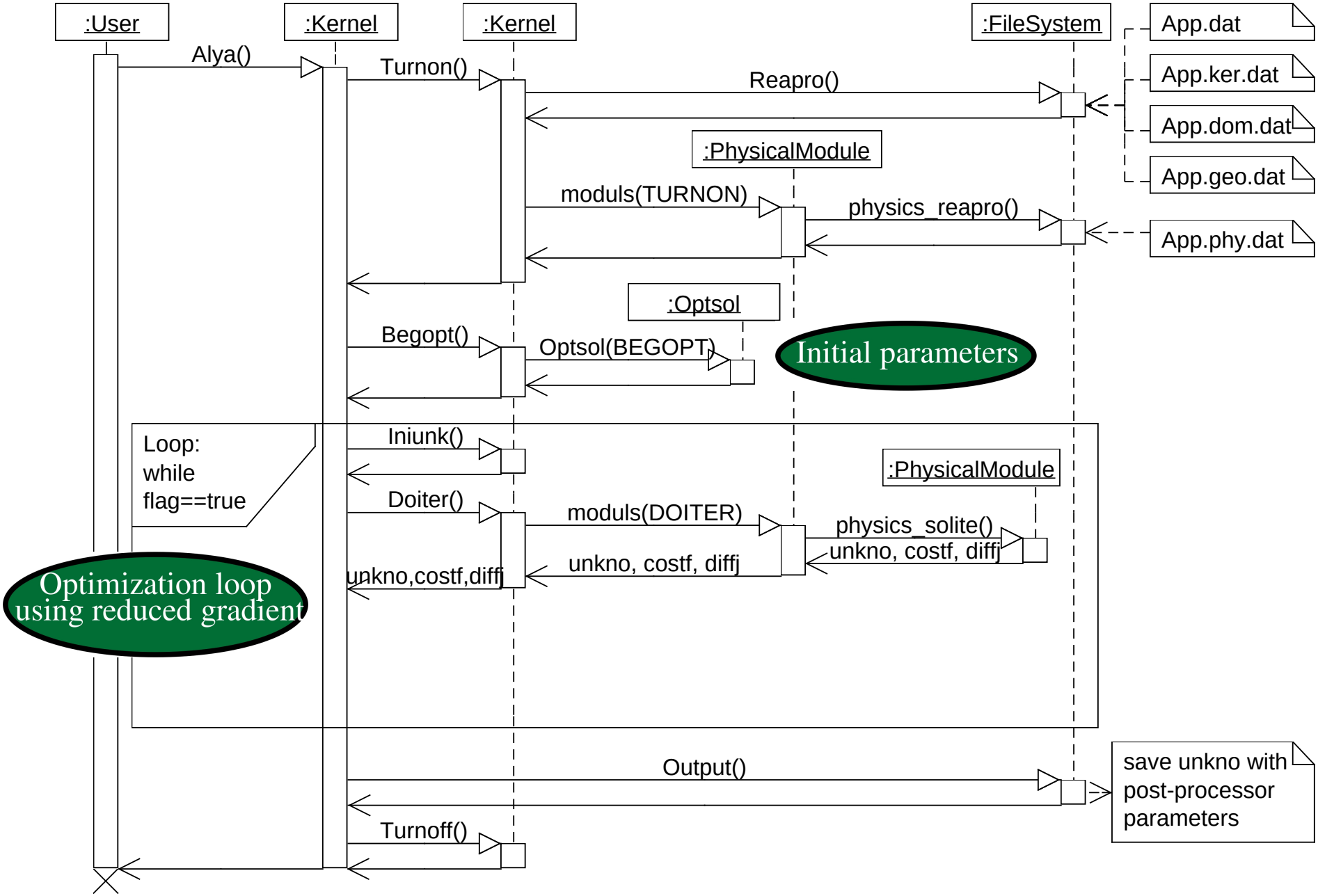
Alya: main driver `Alya.f90`



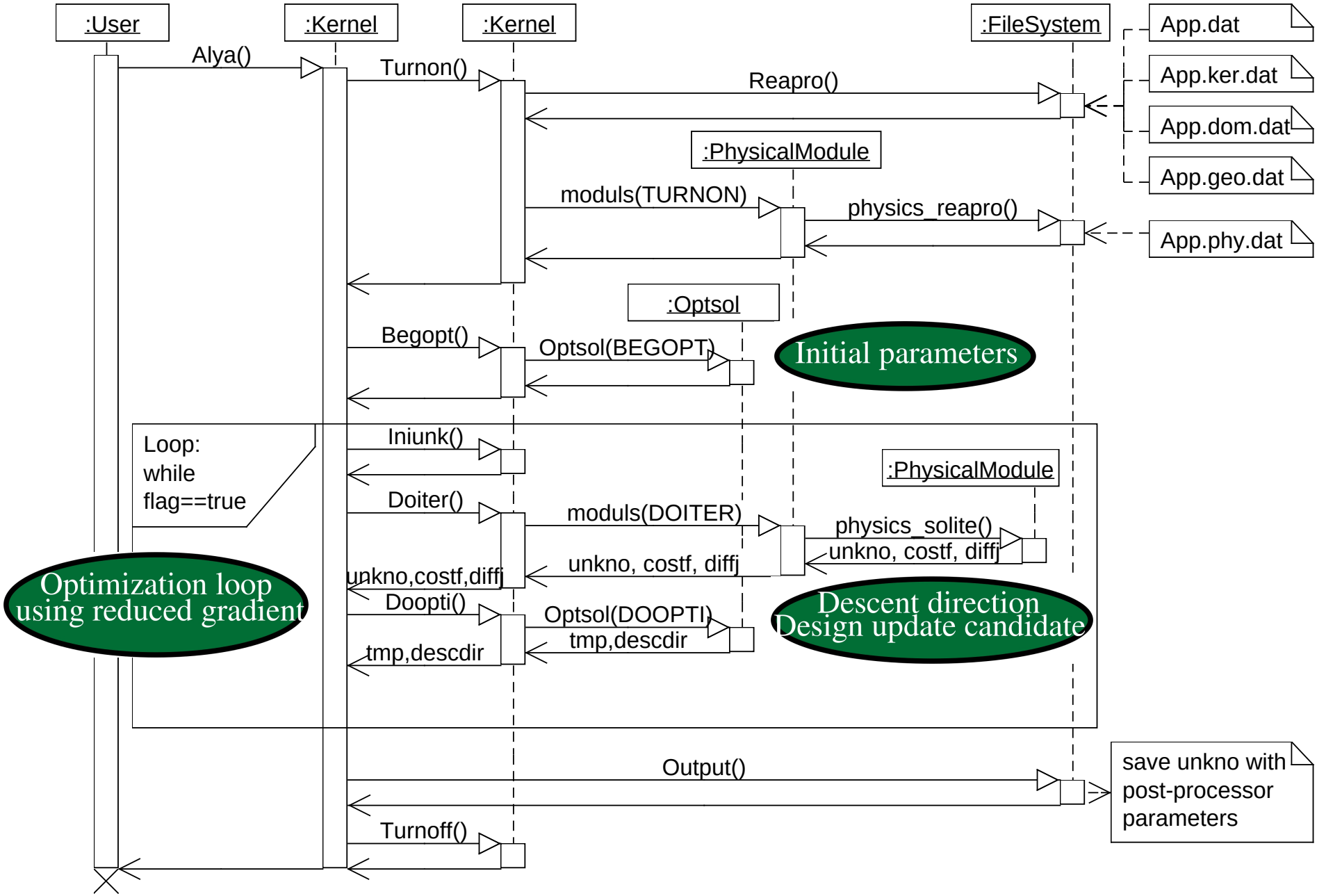
Alya: main driver `Alya.f90` + *Optsol*



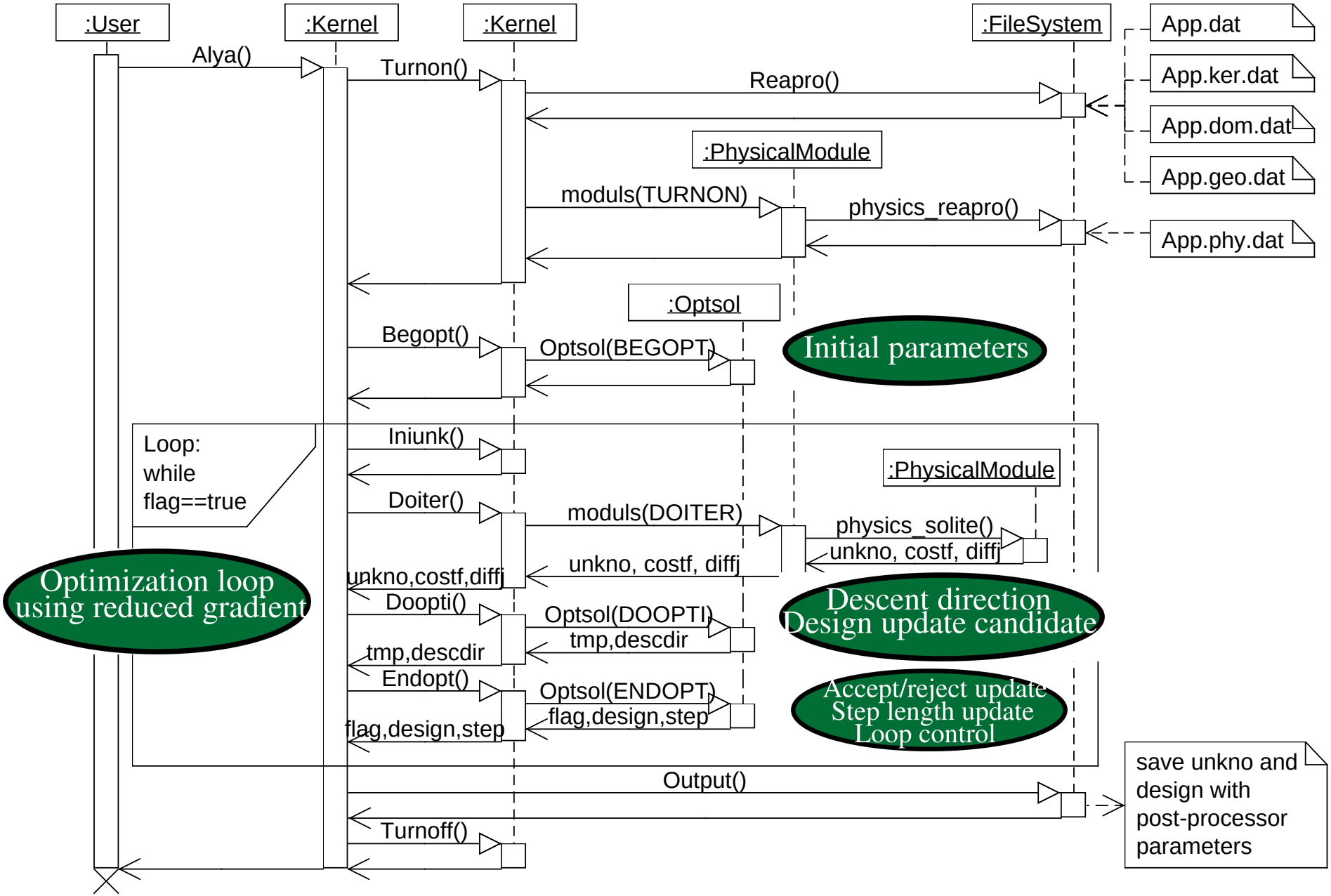
Alya: main driver `Alya.f90` + *Optsol*



Alya: main driver `Alya.f90` + *Optsol*

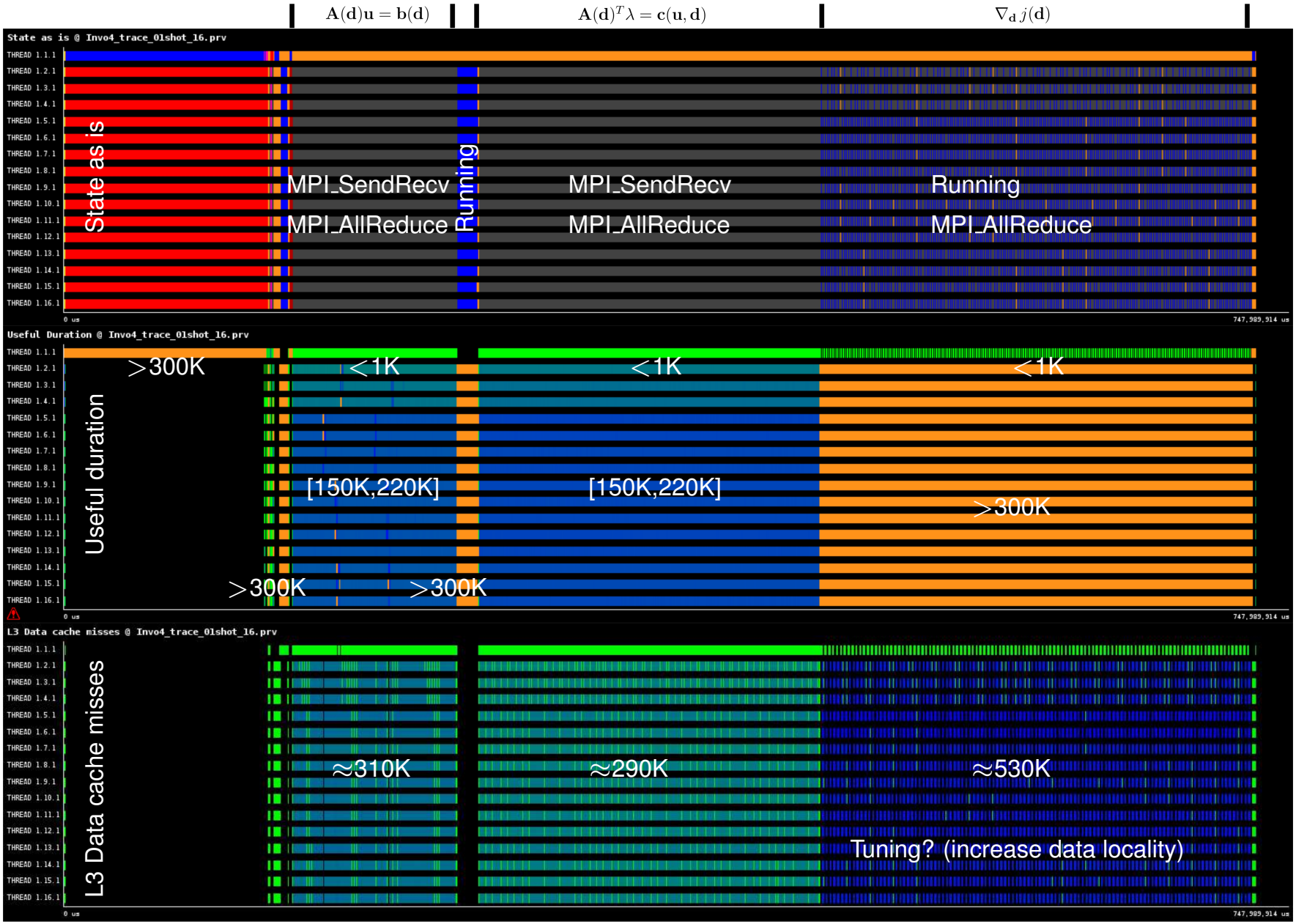


Alya: main driver `Alya.f90` + *Optsol*



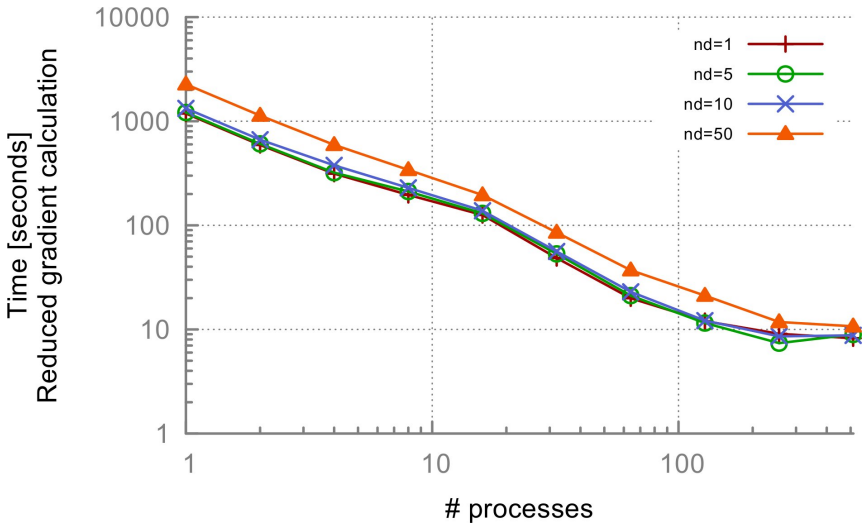
Outline

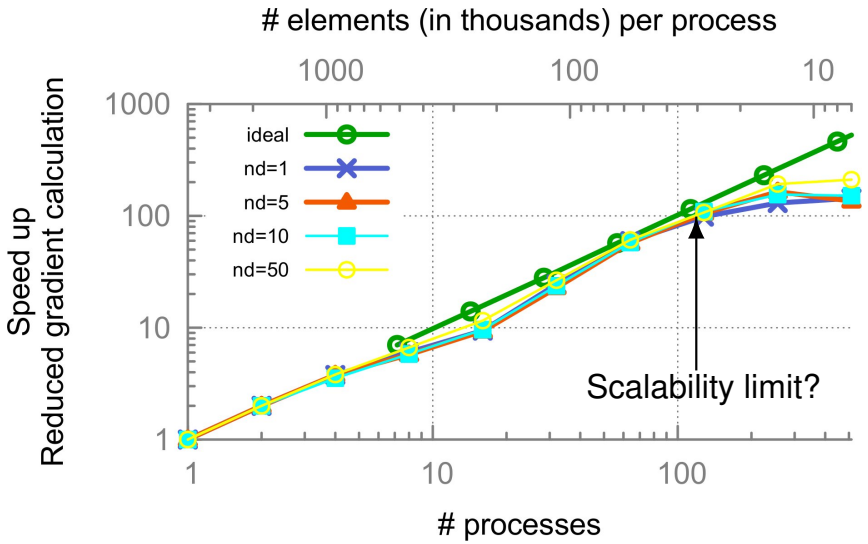
Performance: Reduced gradient calculation



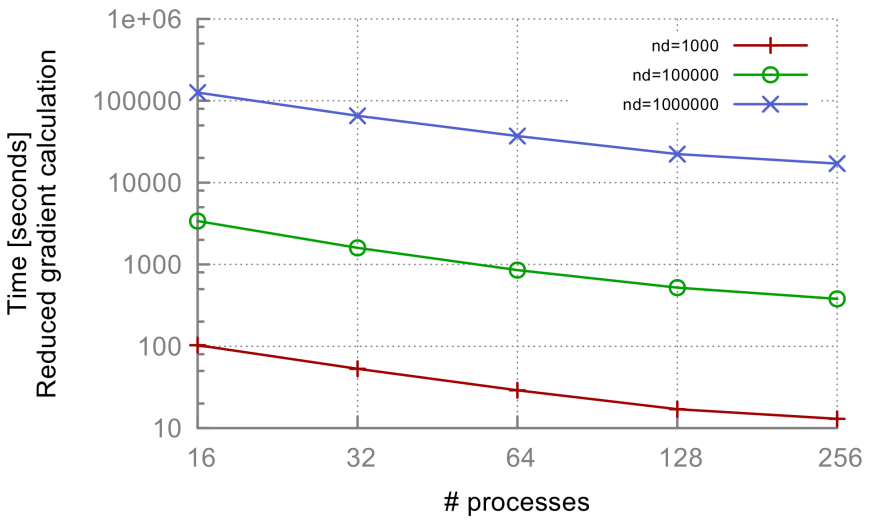
Speedup: Reduced gradient calculation

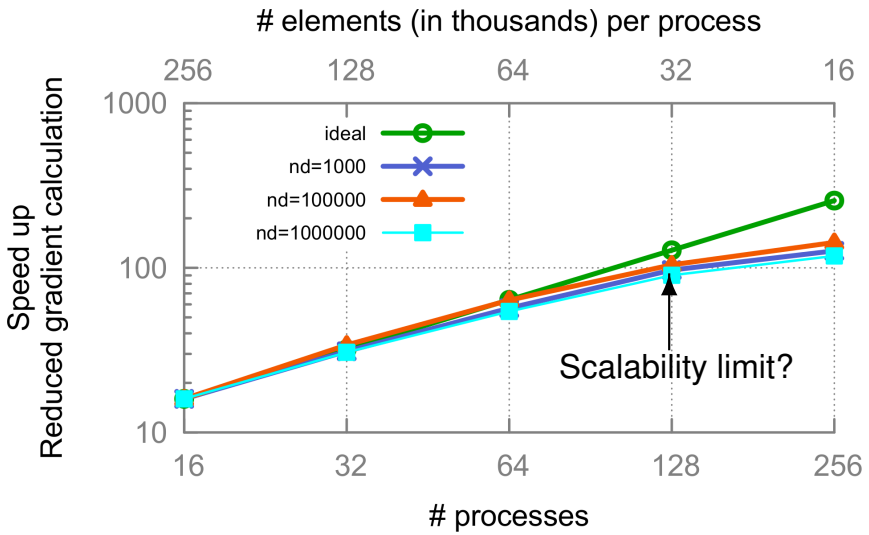
2D mesh, 3.7M tetra elems





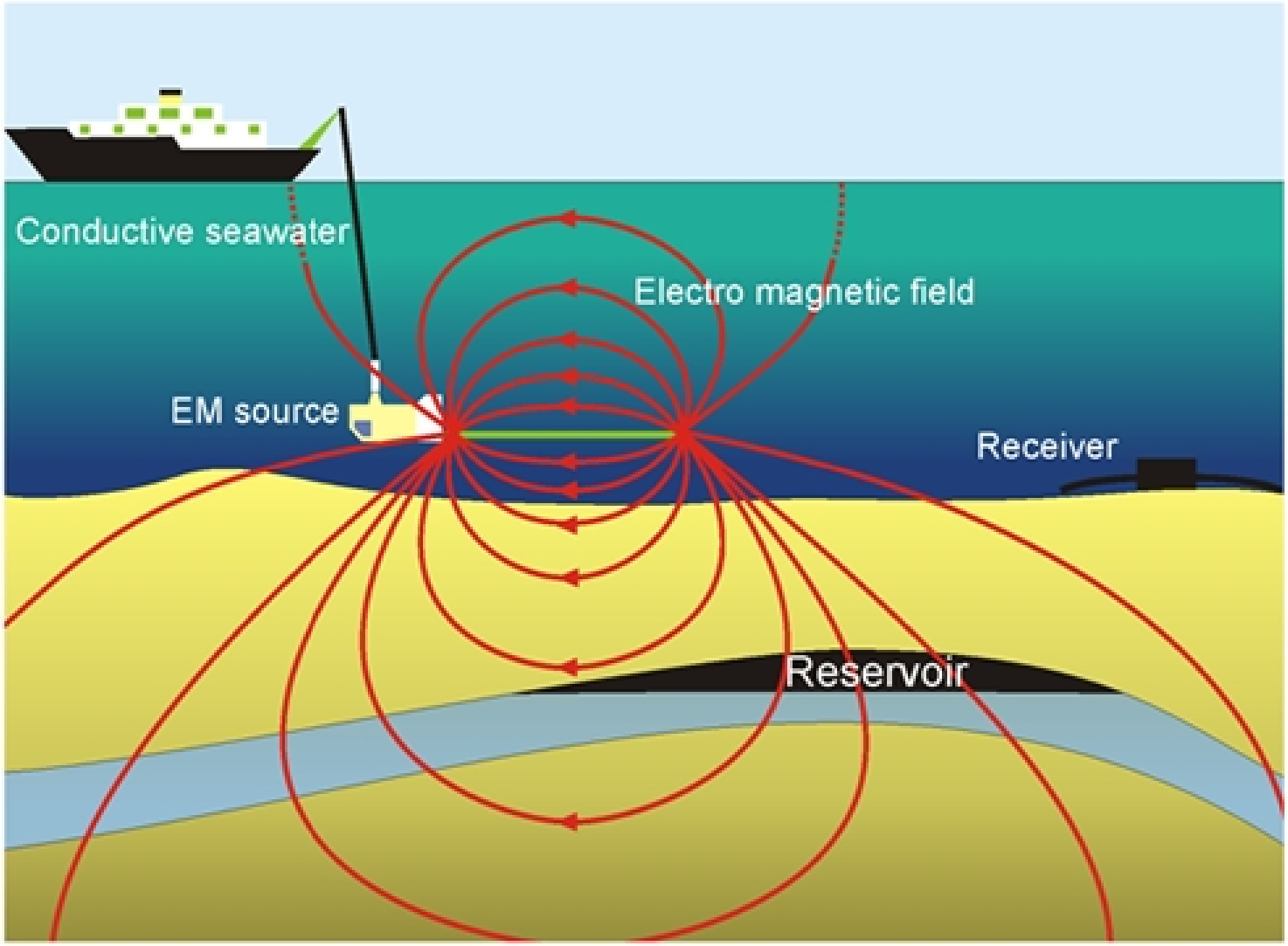
3D mesh, 4.7M tetra elems





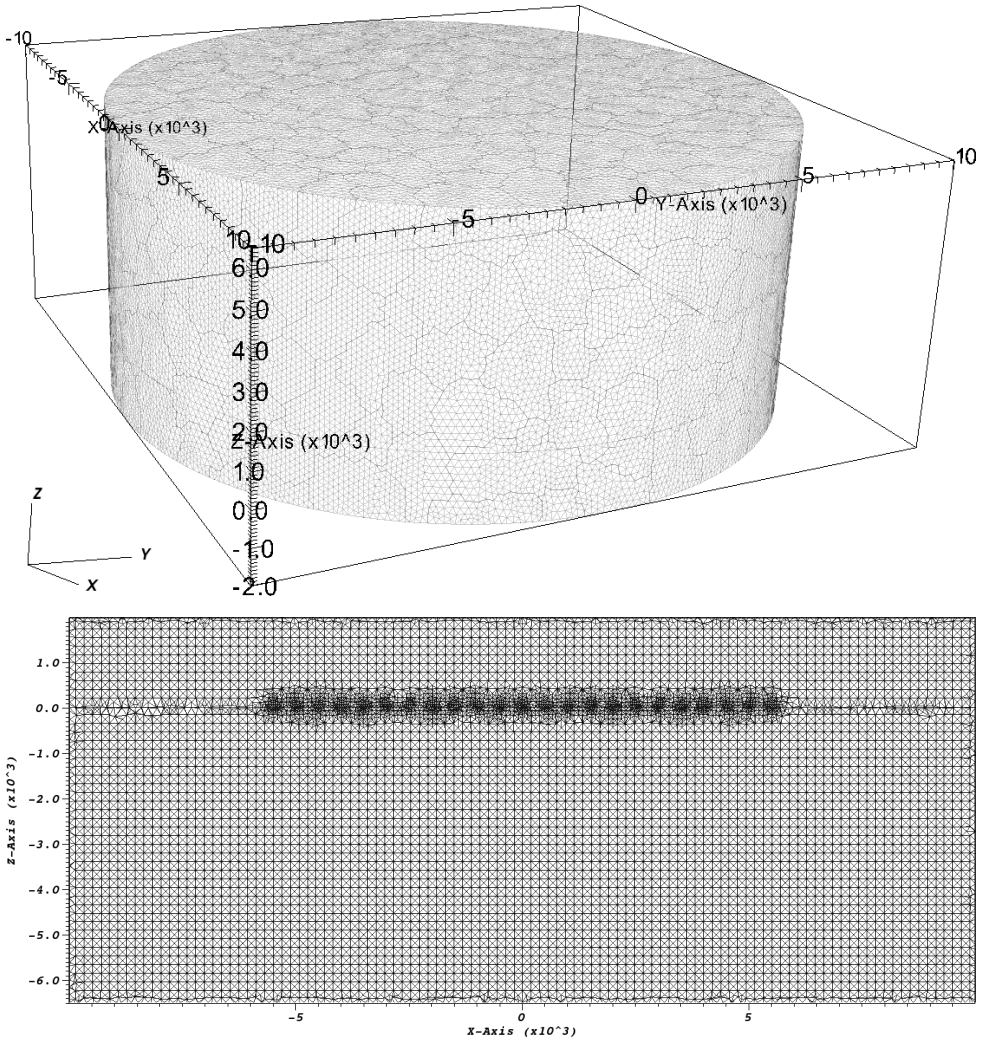
Outline

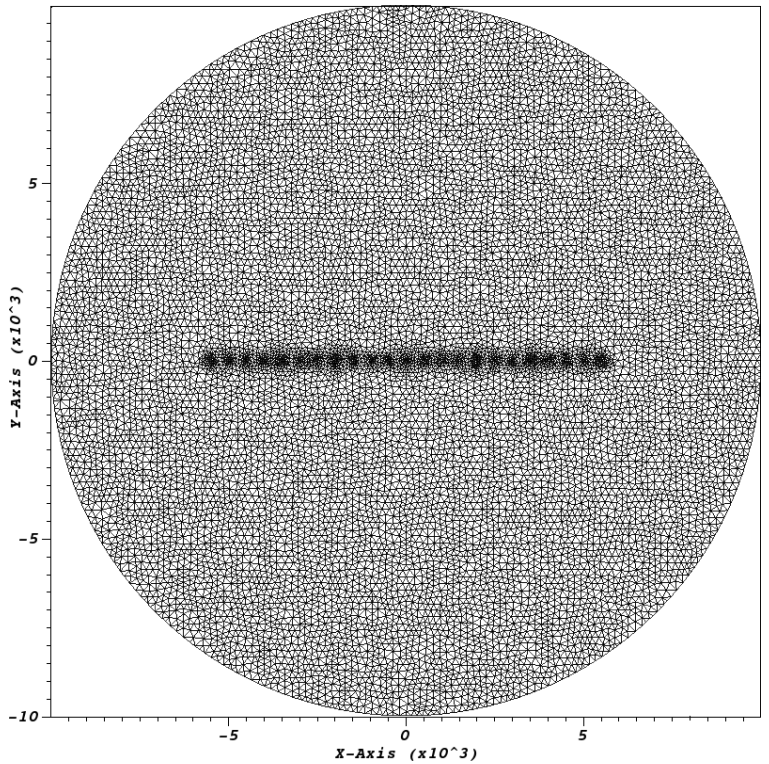
3D EM Inversion



Source: noc.ac.uk

Computational domain



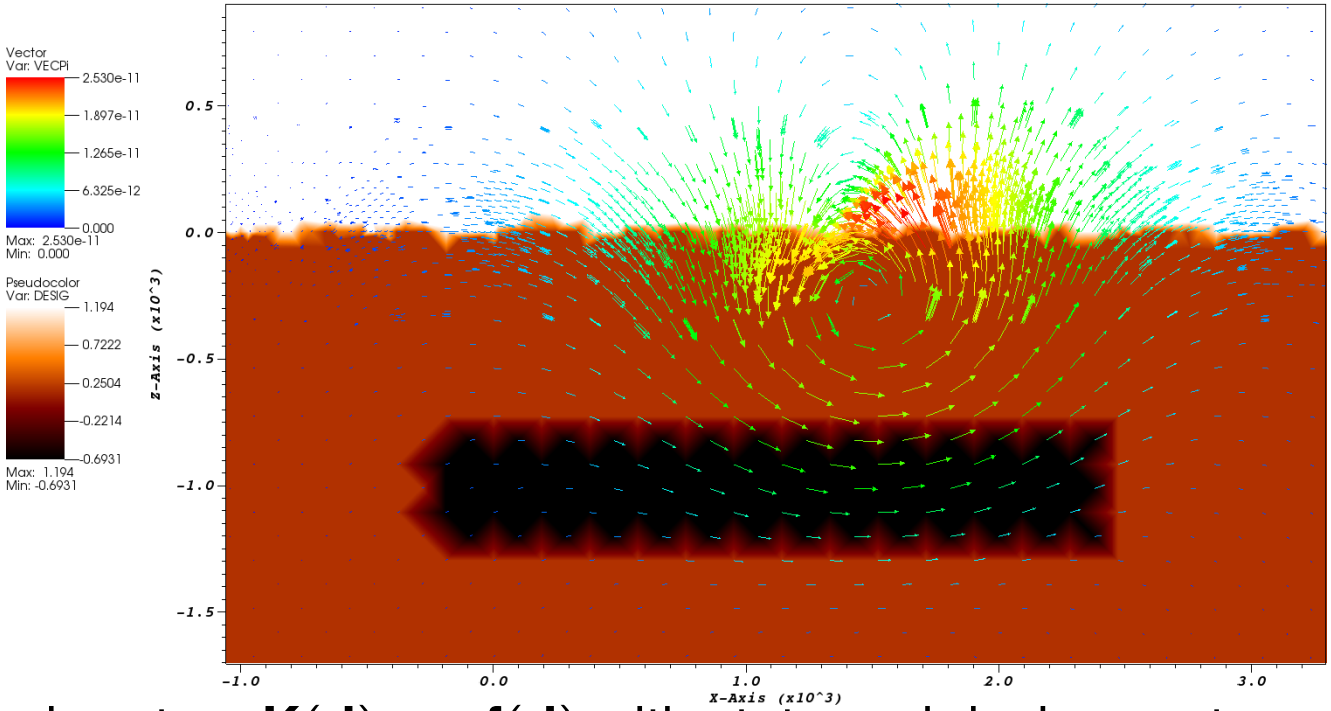


Forward problem

Maxwell's equations using secondary EM potentials [Badea,2001]:

$$\nabla^2 \mathbf{A}_s + i\omega\mu_0\sigma(\mathbf{A}_s + \nabla\phi_s) = -i\omega\mu_0\delta\sigma(\mathbf{A}_p + \nabla\phi_p) \quad (1)$$

$$\nabla \cdot (i\omega\mu_0\sigma(\mathbf{A}_s + \nabla\phi_s)) = -\nabla \cdot (i\omega\mu_0\delta\sigma(\mathbf{A}_p + \nabla\phi_p)) \quad (2)$$



Discretized system $\mathbf{K}(\mathbf{d})\mathbf{u} = \mathbf{f}(\mathbf{d})$ with state and design vectors

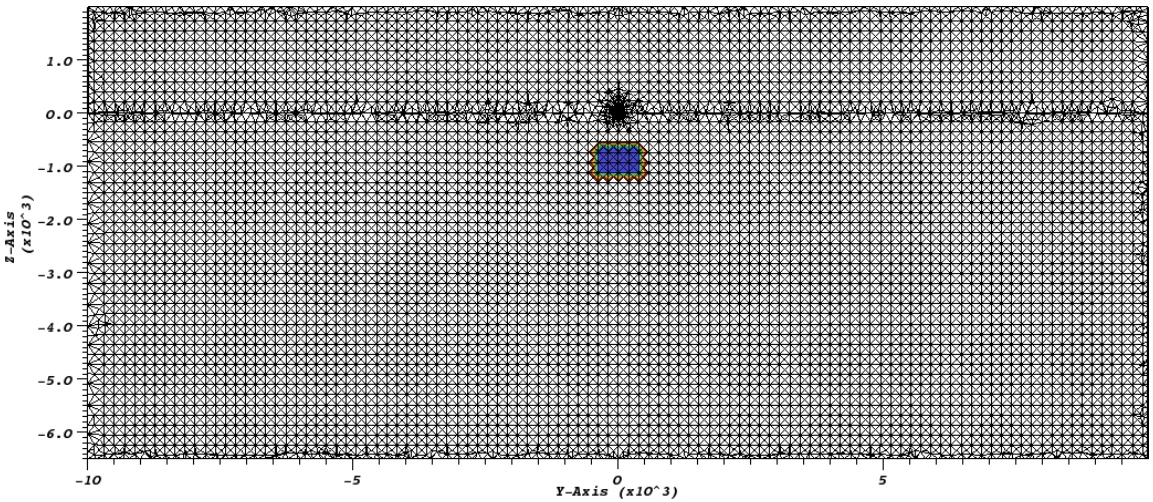
$\mathbf{u} = (\mathbf{A}_s^x, \mathbf{A}_s^y, \mathbf{A}_s^z, \phi_s)$

$\mathbf{d} = \sigma(x, y, z)$ (electrical conductivity in each nodal point)

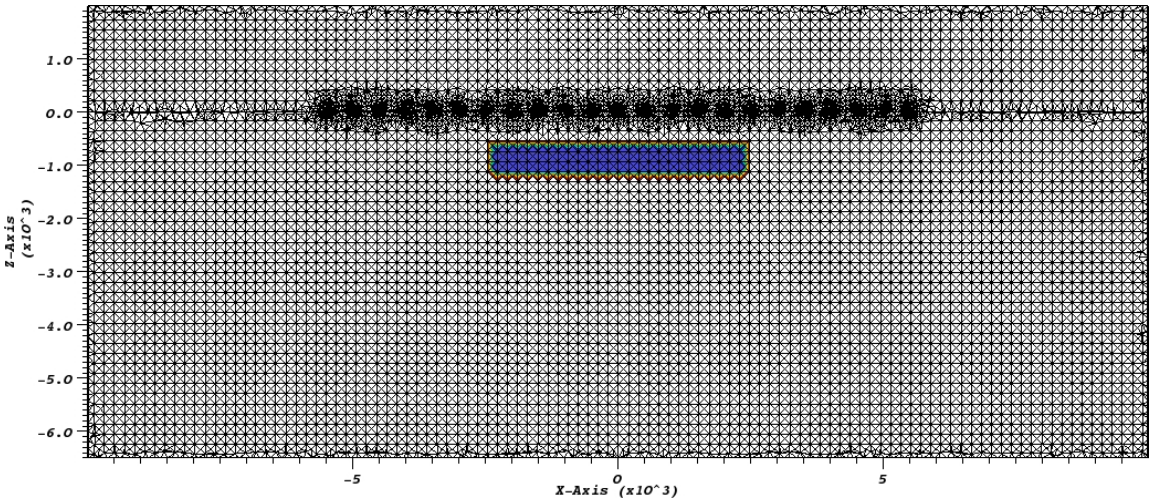
$\gamma = \ln(\mathbf{d})$ (auxiliary variable, electrical log-conductivity in each nodal point)

Inverse problem: reservoir geometry

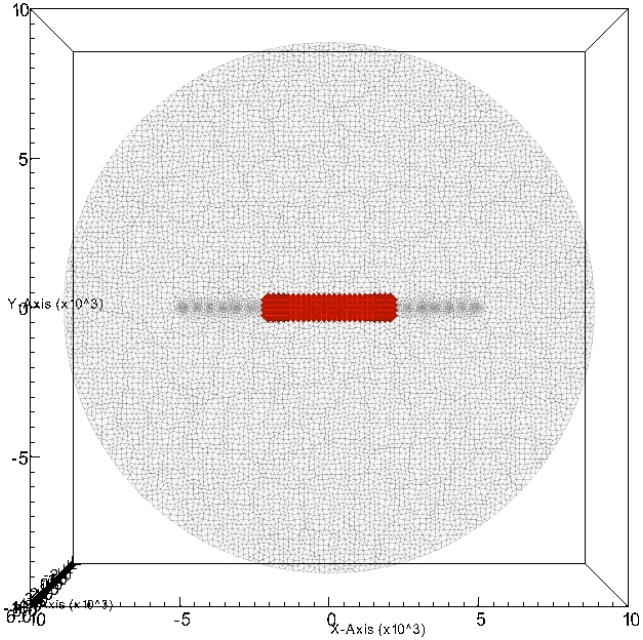
Slice in $x = 0$:



Slice in $y = 0$:



Slice in $z = 0$:

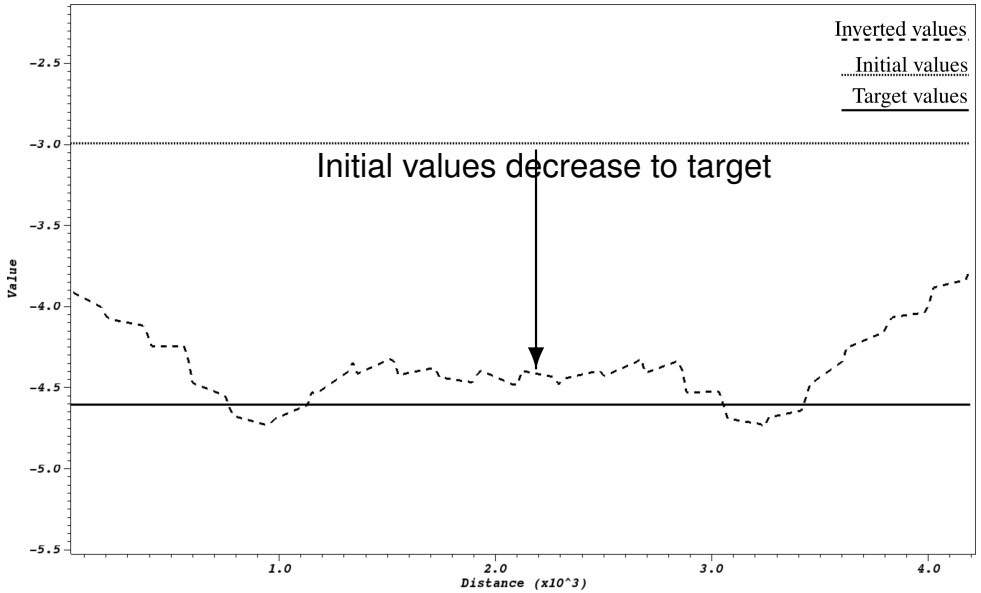


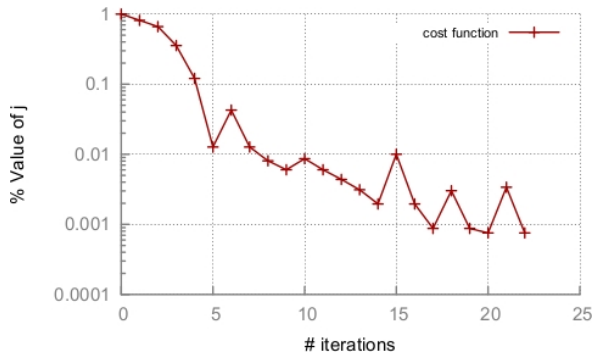
Inverse problem: test 1 ($\sigma_{initial} > \sigma_{target}$)

Cost function and reduced gradient:

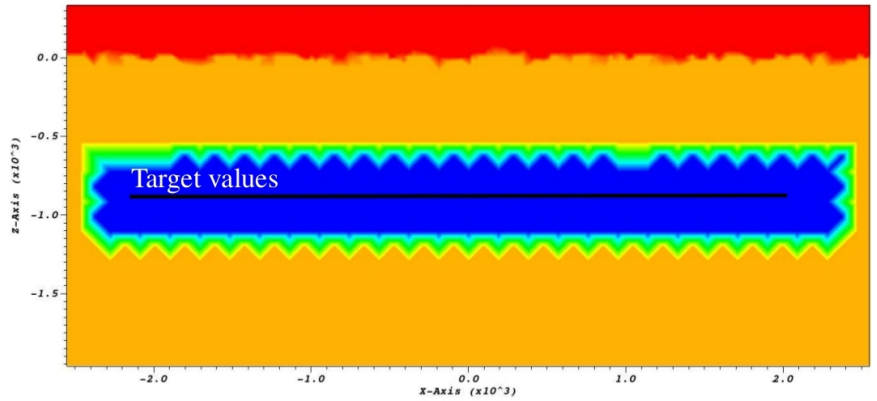
$$J(\mathbf{z},\mathbf{d}) = \sum_{i=1}^{\#\text{shots}} \sum_{k=1}^{\#\text{obs}(i)} \delta(\text{Im}z_k - \text{Im}z_k^{obs_i})^2$$

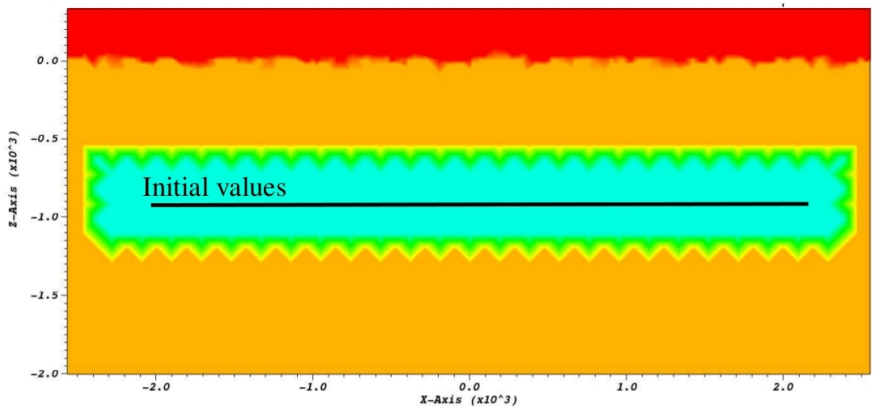
$$\nabla_\gamma j(\gamma) = \sum_{i=1}^{\#\text{shots}} -2\text{Re}\left\{\overline{\lambda}_i^T \nabla_{\mathbf{d}}\mathbf{R}(\mathbf{z},\overline{\mathbf{z}},\mathbf{d})\right\} \cdot \nabla_\gamma \mathbf{d}(\gamma)$$

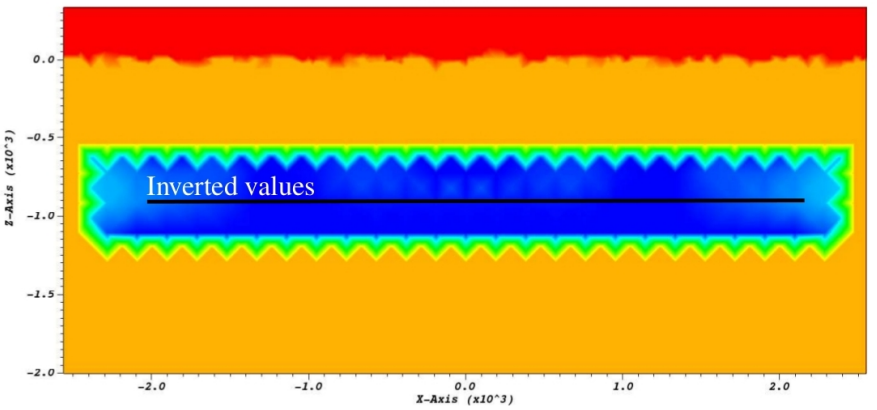




Steepest descent, 5 line-searches, 10 iters max, BiCGSTAB, 4 shots (5 obs per shot)





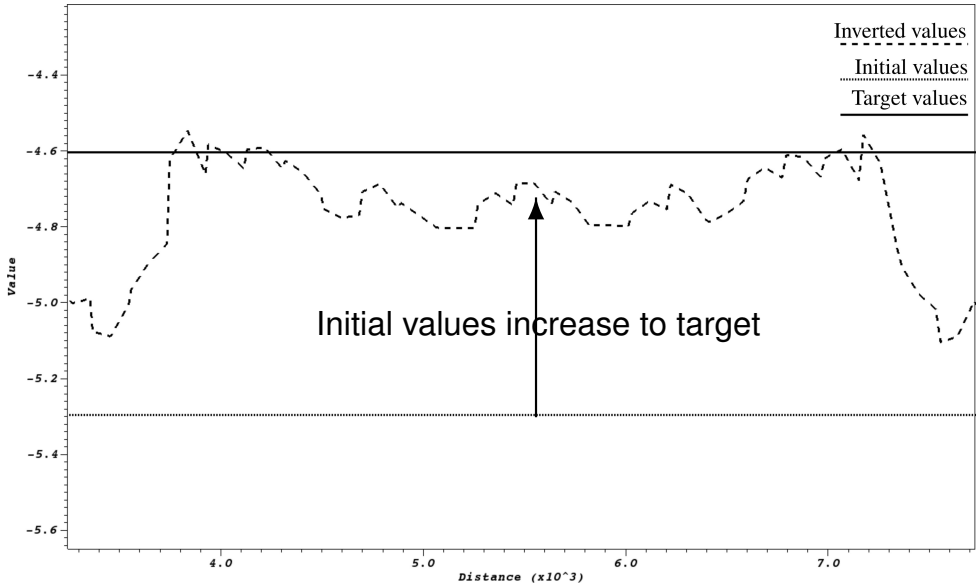


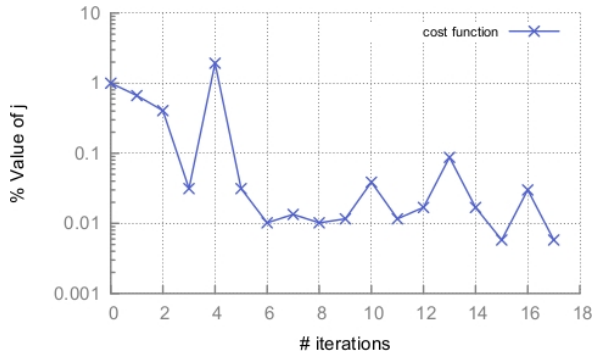
Inverse problem: test 2 ($\sigma_{initial} < \sigma_{target}$)

Cost function and reduced gradient:

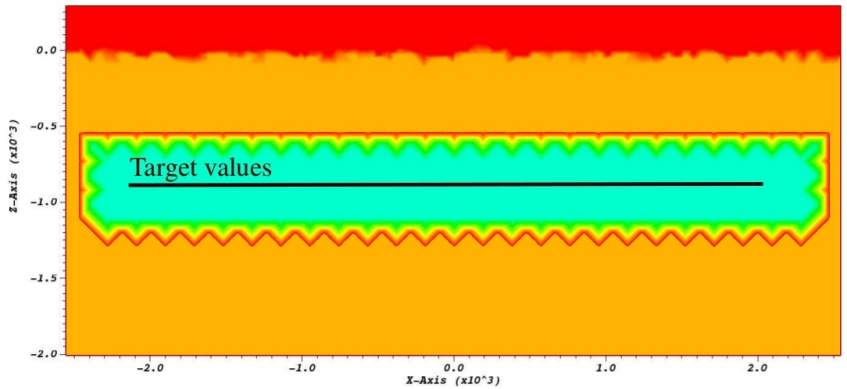
$$J(\mathbf{z},\mathbf{d}) = \sum_{i=1}^{\#\text{shots}} \sum_{k=1}^{\#\text{obs}(i)} \delta(\text{Im}z_k - \text{Im}z_k^{obs_i})^2$$

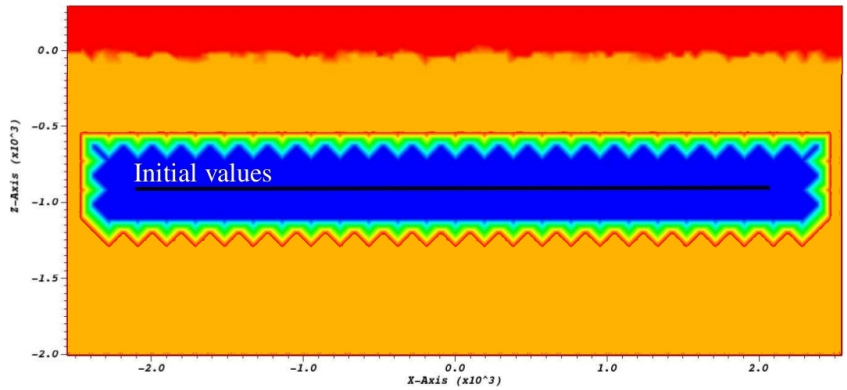
$$\nabla_\gamma j(\gamma) = \sum_{i=1}^{\#\text{shots}} -2\text{Re}\left\{\overline{\boldsymbol{\lambda}}_i^T \nabla_{\mathbf{d}}\mathbf{R}(\mathbf{z},\bar{\mathbf{z}},\mathbf{d})\right\} \cdot \nabla_\gamma \mathbf{d}(\gamma)$$

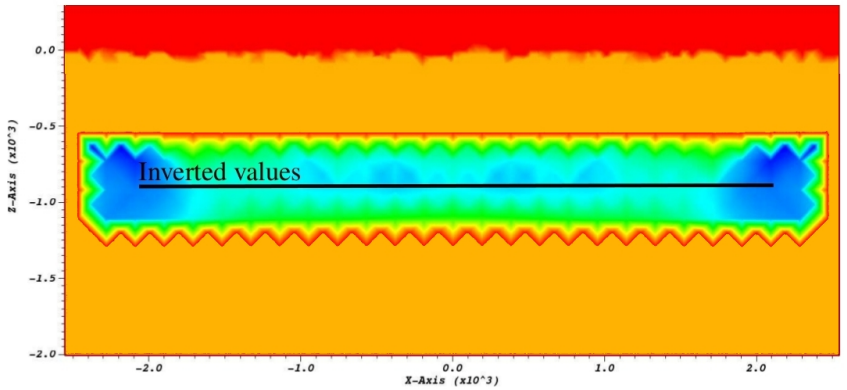




Steepest descent, 5 line-searches, 10 iters max, BiCGSTAB, 4 shots (5 obs per shot)





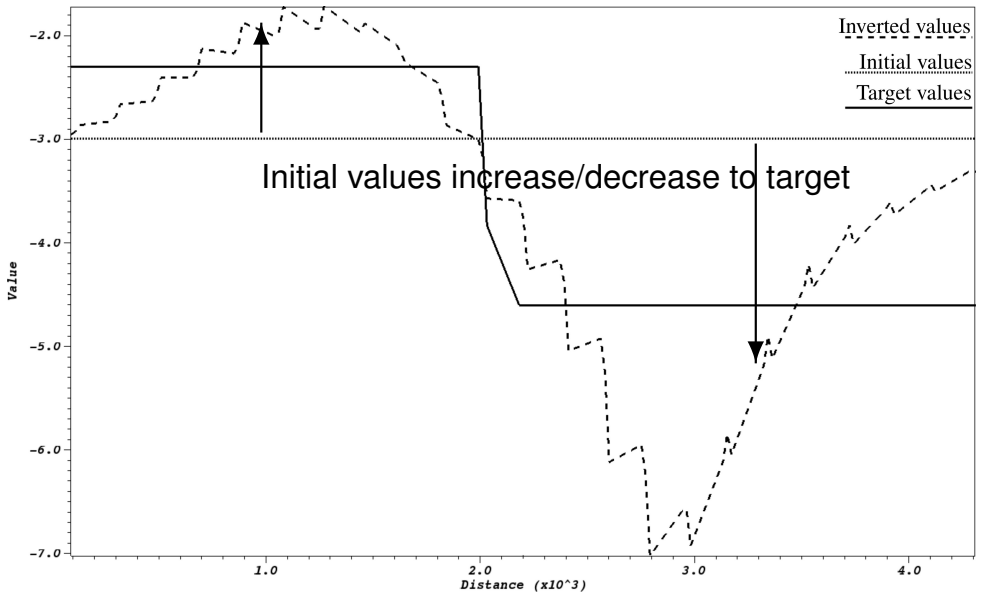


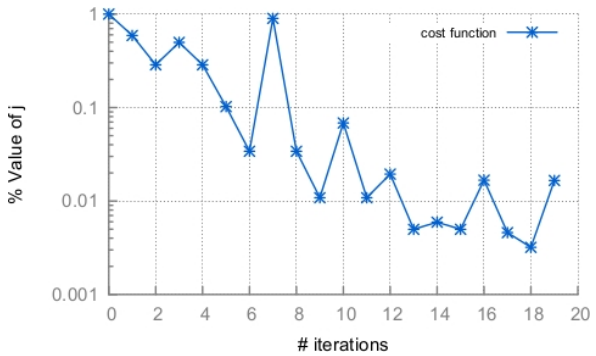
Inverse problem: test 3 (σ_{target} no homogeneous)

Cost function and reduced gradient:

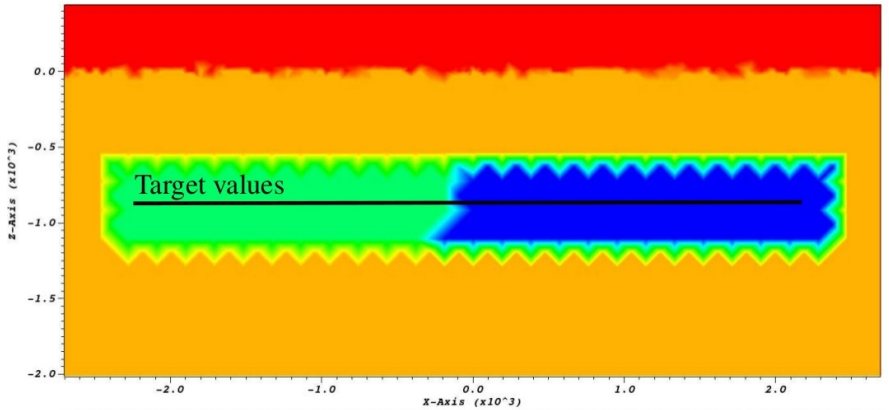
$$J(\mathbf{z},\mathbf{d}) = \sum_{i=1}^{\#\text{shots}} \sum_{k=1}^{\#\text{obs}(i)} \delta(\text{Im}z_k - \text{Im}z_k^{obs_i})^2$$

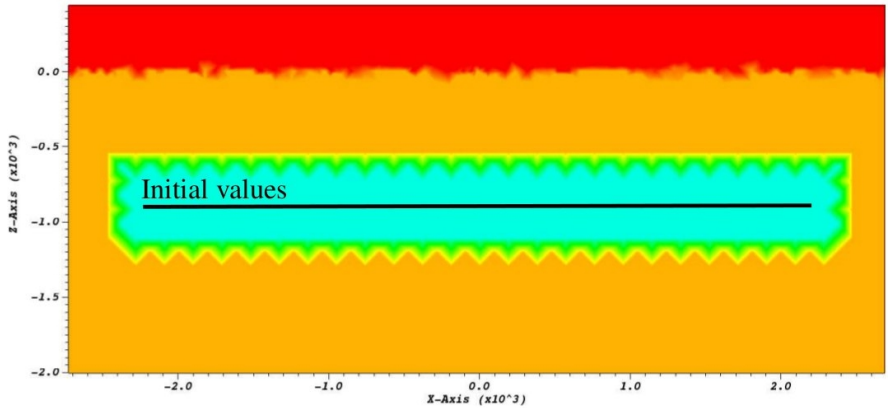
$$\nabla_\gamma j(\gamma) = \sum_{i=1}^{\#\text{shots}} -2\text{Re}\left\{\overline{\boldsymbol{\lambda}}_i^T \nabla_{\mathbf{d}}\mathbf{R}(\mathbf{z},\bar{\mathbf{z}},\mathbf{d})\right\} \cdot \nabla_\gamma \mathbf{d}(\gamma)$$

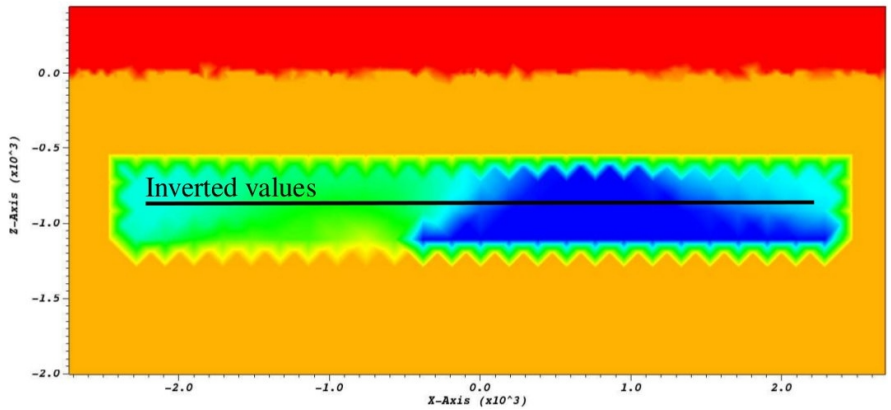




Steepest descent, 5 line-searches, 10 iters max, BiCGSTAB, 4 shots (5 obs per shot)







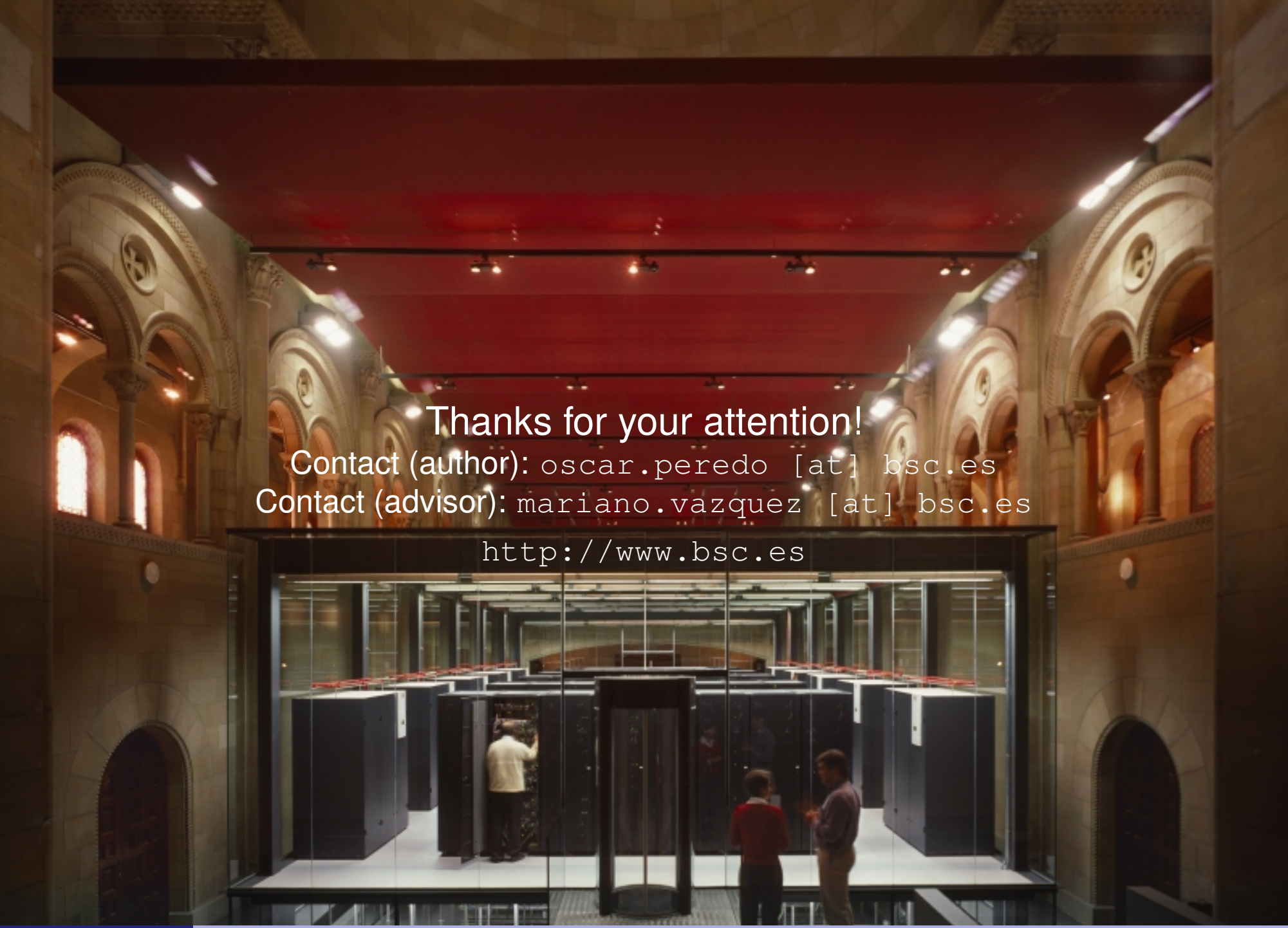
Outline

Conclusions

- We have developed a simple framework on top of Alya to solve PDECO problems (linear stationary physical models).
- Acceptable scalability and convergence results (expert knowledge of the forward model and several tests are required).
- A novel application was developed (synthetic examples): 3D electromagnetic inversion. Potential usage in hydrocarbon exploration.

Future work

- Inclusion of new physical models into the proposed framework (transient, non-linear and multi-physics models).
- Add sophisticated optimization methods to *Optsol* (SAND, external constraints).
- Research in new applications: Oil&Gas/Mining industry, Optimal design in engineering, Biomedical simulations, . . .


Thanks for your attention!
Contact (author): oscar.peredo [at] bsc.es
Contact (advisor): mariano.vazquez [at] bsc.es
http://www.bsc.es