STANFORD UNVERSITY

SEP meeting 2017

A flexible out-of-core solver for linear/non-linear problems

Ettore Biondi* and Guillaume Barnier

19th April 2017

Inverse problem

From the given data obtain the physical model

$$\begin{tikzcd}
\boldsymbol{d(m)} \arrow[r, bend left, "\text{Inverse problem}"] & \boldsymbol{m} \arrow[l, bend left, "\text{Modeling}"]
\end{tikzcd}$$

Inverse problem

From the given data obtain the physical model

- Pressure
- Particle velocities
- Traveltimes
- Gather curvature (WEMVA)

$$\begin{tikzcd}
\boldsymbol{d(m)} \arrow[r, bend left, "\text{Inverse problem}"] & \boldsymbol{m} \arrow[l, bend left, "\text{Modeling}"]
\end{tikzcd}$$

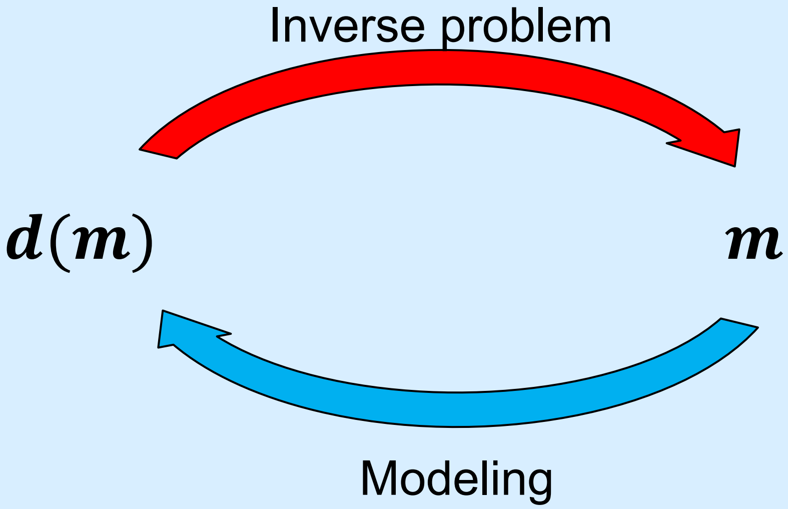
- Vp and/or Vs
- Density
- Attenuation
- Anisotropic parameters

f(m),Fm

non-linear,linear

Inverse problem

Complex machinery!
But many concepts are always there!



f(m),Fm

non-linear,linear

Class operator

FIRST HURDLE

How to combine different coding styles, programming languages, computers to work with a solver?

Class operator

FIRST HURDLE

How to combine different coding styles, programming languages, computers to work with a solver?

PYTHON
(very powerful scripting language!)

Class operator

FIRST HURDLE

How to combine different coding styles, programming languages, computers to work with a solver?

Create an operator object that:

- **Contains template commands for applying a given operator**

Class operator

FIRST HURDLE

How to combine different coding styles, programming languages, computers to work with a solver?

Create an operator object that:

- **Contains template commands for applying a given operator**
- **Runs the commands on an input file and outputs the result**

Class operator

FIRST HURDLE

How to combine different coding styles, programming languages, computers to work with a solver?

Create an operator object that:

- **Contains template commands for applying a given operator**
- **Runs the commands on an input file and outputs the result**

$$\boldsymbol{f}(\boldsymbol{m}) = \boldsymbol{d}$$

$\boldsymbol{m}$ => input model file (model vector)
$\boldsymbol{f}$ => operator (written in any computer language)
$\boldsymbol{d}$ => output data file (data vector)

The solver: structure

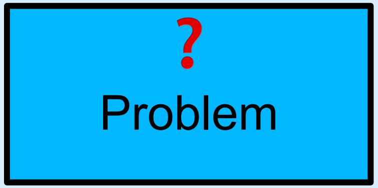
SECOND HURDLE

What kind of structure for the solver?
We spent a lot of time discussing, let's use that time! (See SEP160)

(Almomin, 2015)

The solver: structure

SECOND HURDLE

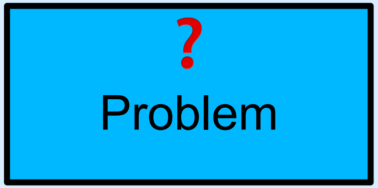


Problem object:

- **Objective function (L2/L1/Hybrid)**
- **Residual function (difference/cross-correlation)**
- **Linearized fw (useful for step-length initial guess)**
- **Linearized adj (fundamental for gradient)**

The solver: structure

SECOND HURDLE


?

Problem

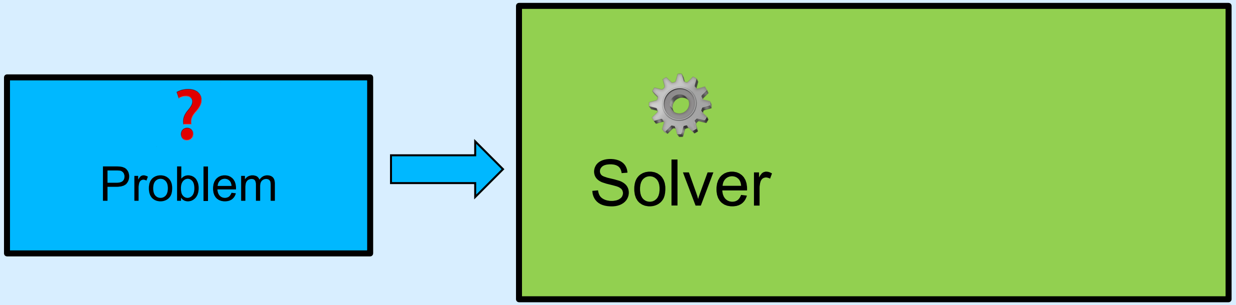
Problem object:

- **Objective function (L2/L1/Hybrid)**
- **Residual function (difference/cross-correlation)**
- **Linearized fw (useful for step-length initial guess)**
- **Linearized adj (fundamental for gradient)**

Needed if gradient-based method is used

The solver: structure

SECOND HURDLE

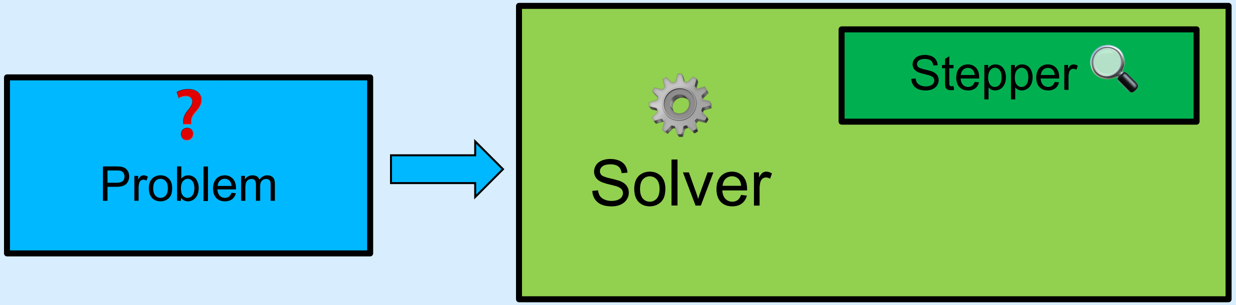


Solver object:

- **Different for various algorithms (CG/LBFGS/MCMC)**

The solver: structure

SECOND HURDLE

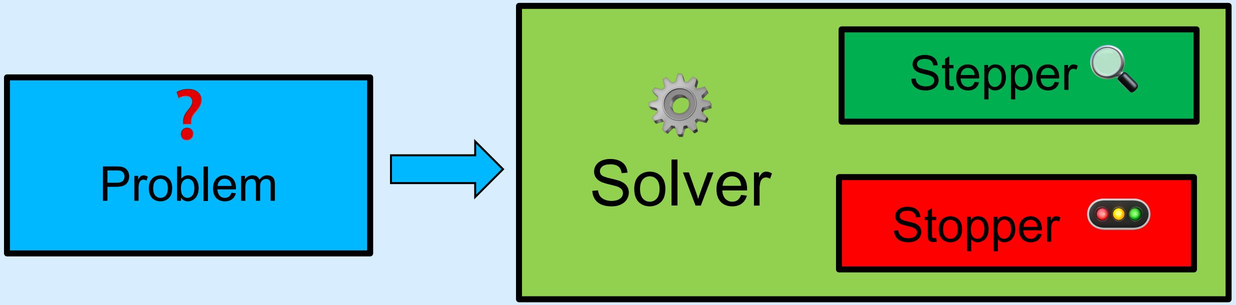


Stepper object:

- **Finds step length (linear/parabolic/backtracking)**

The solver: structure

SECOND HURDLE

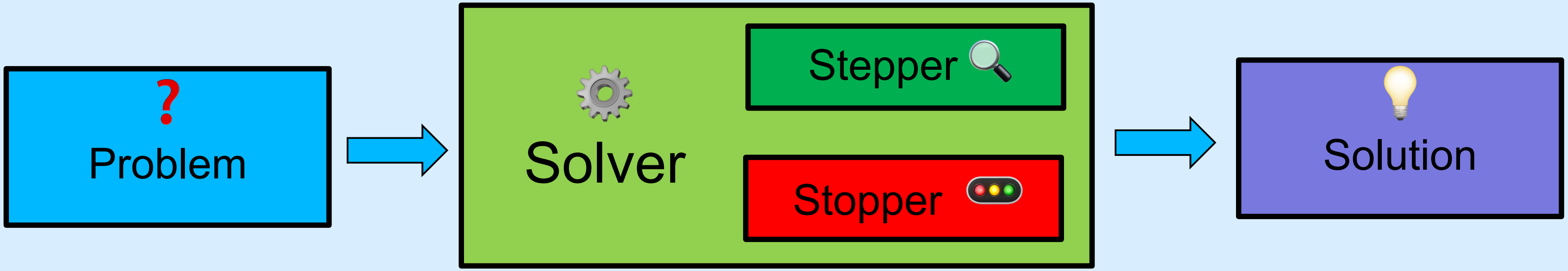


Stopper object:

- **Stopping criteria (iterations/evaluations/time)**

The solver: structure

SECOND HURDLE



Stopper object:

- **Stopping criteria (iterations/evaluations/time)**

The solver: usage

Does the user have to know all of this?

The solver: usage

Does the user have to know all of this?
Not at all!

The solver: usage

For example, linear L2 difference problem:

```
$ ./python_solver/generic_linear_prob.py \
      fwd_cmd_file=./Par/tmp_fwd.txt       adj_cmd_file=./Par/tmp_adj.txt \
      data=data.H          init_model=init_model.H          niter=10
```

generic_linear_prob.py => Python script running the solver

The solver: usage

For example, linear L2 difference problem:

```
$ ./python_solver/generic_linear_prob.py \
     fwd_cmd_file=./Par/tmp_fwd.txt    adj_cmd_file=./Par/tmp_adj.txt \
     data=data.H          init_model=init_model.H          niter=10
```

fwd_cmd_file => $\boldsymbol{Fm = d}$

adj_cmd_file => $\boldsymbol{F^{*}d = \tilde{m}}$

The solver: usage

For example, linear L2 difference problem:

```
$ ./python_solver/generic_linear_prob.py \
     fwd_cmd_file=./Par/tmp_fwd.txt      adj_cmd_file=./Par/tmp_adj.txt \
     data=data.H        init_model=init_model.H        niter=10
```

data => observed data file

init_model => initial model file

niter => # of iterations

The solver: usage

For example, linear L2 difference problem:

```
$ ./python_solver/generic_linear_prob.py \
      fwd_cmd_file=./Par/tmp_fwd.txt   adj_cmd_file=./Par/tmp_adj.txt \
      data=data.H         init_model=init_model.H         niter=10
```

fwd_cmd_file => $Fm = d$

```
#This is a comment
#Applying forward modeling operator
RUN: ./bin/forward.x < input.H par=Par/parfile.p > output.H
```

The solver: usage

For example, non-linear L2 difference problem:

```
$ ./python_solver/generic_non_linear_prob.py   fwd_nl_cmd_file=./Par/tmp_nl.txt \
      fwd_cmd_file=./Par/tmp_fwd.txt        adj_cmd_file=./Par/tmp_adj.txt \
      data=data.H          init_model=init_model.H          niter=10
```

fwd_nl_cmd_file => $\boldsymbol{f}(\boldsymbol{m}) = \boldsymbol{d}$

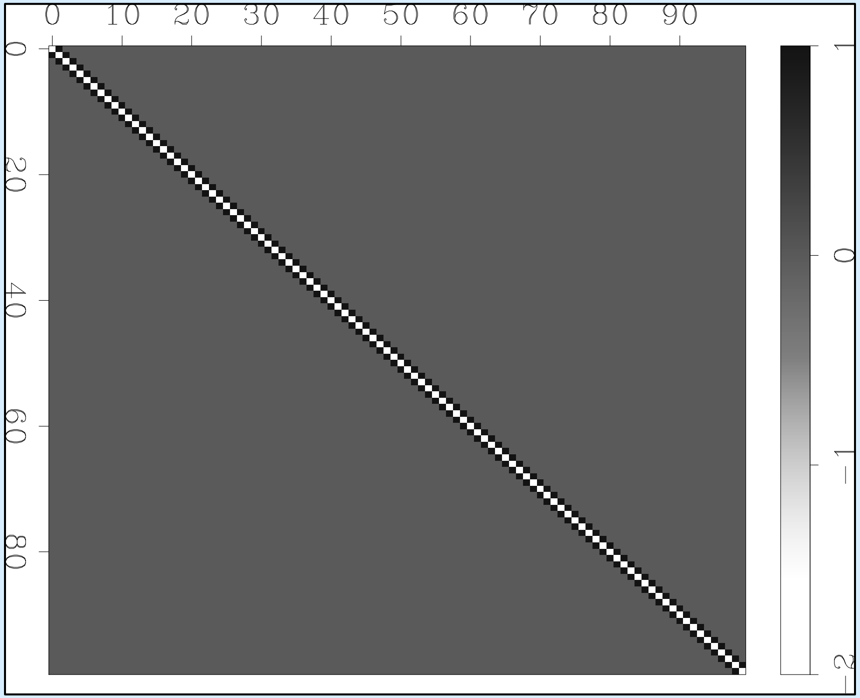
fwd_cmd_file => $\boldsymbol{F}(\boldsymbol{m}_0)\boldsymbol{m} = \boldsymbol{d}$

adj_cmd_file => $\boldsymbol{F}(\boldsymbol{m}_0)^*\boldsymbol{d} = \widetilde{\boldsymbol{m}}$

The solver: linear test

Invertible matrix:

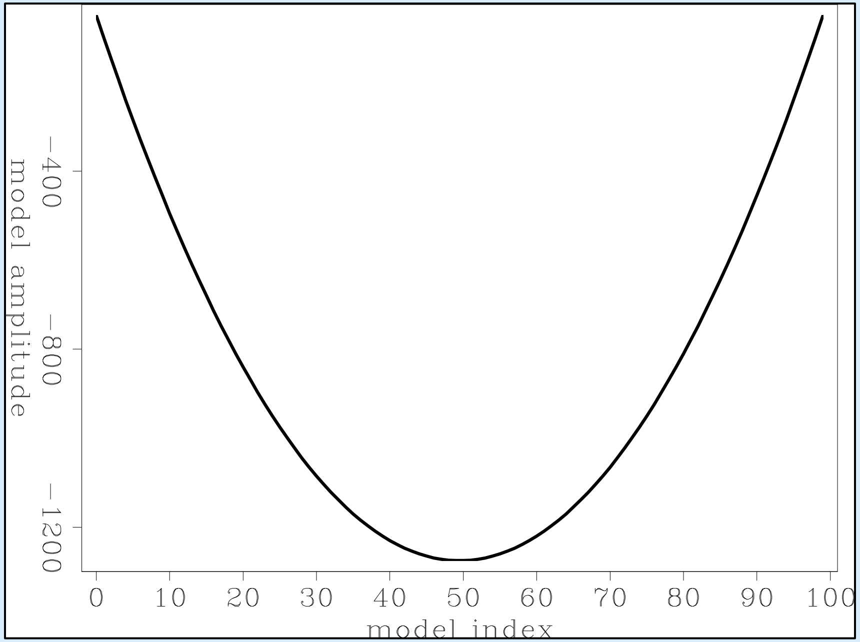
- **Second-order derivative operator with boundary conditions**
- **Data vector is constant => parabola**



The solver: linear test

Invertible matrix:

- **Second-order derivative operator with boundary conditions**
- **Data vector is constant => parabola**

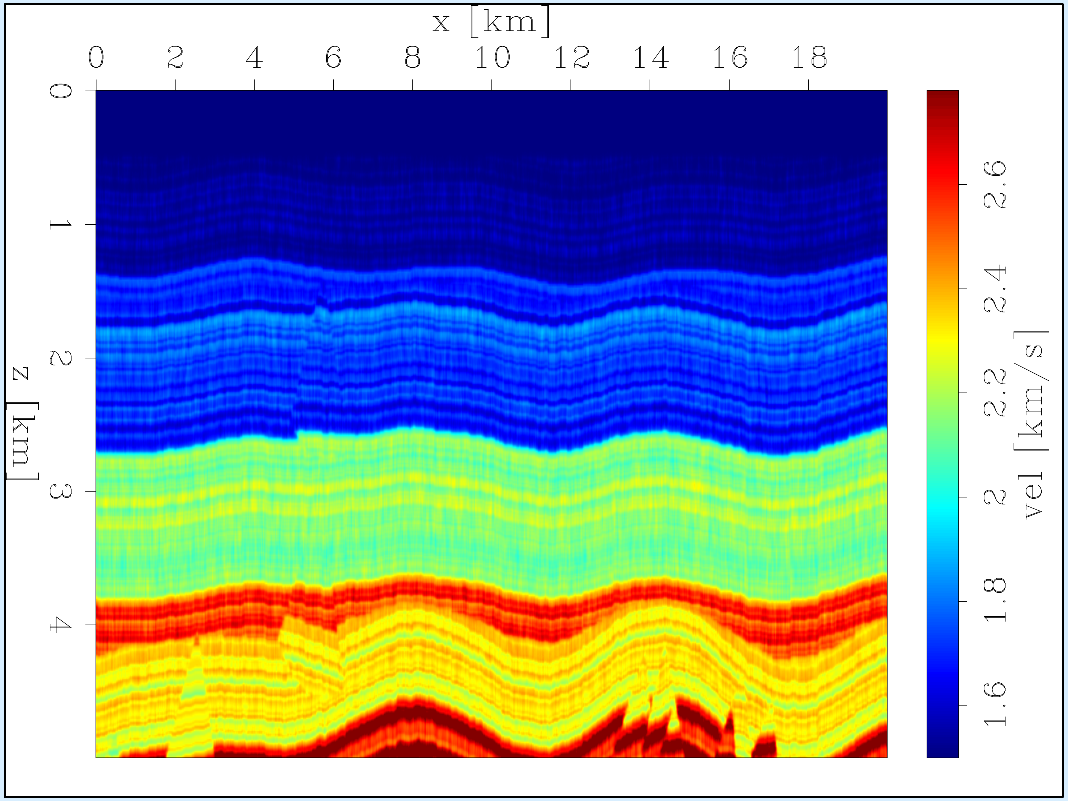


The solver: non-linear test

Acoustic Full Waveform Inversion test

- **200 shots managed/submitted on a cluster by python script**
- **2000 receivers at the surface**
- **Inverting 0-5 Hz in the recorded data**
- **Run 14 iterations of non-linear CG**

True model



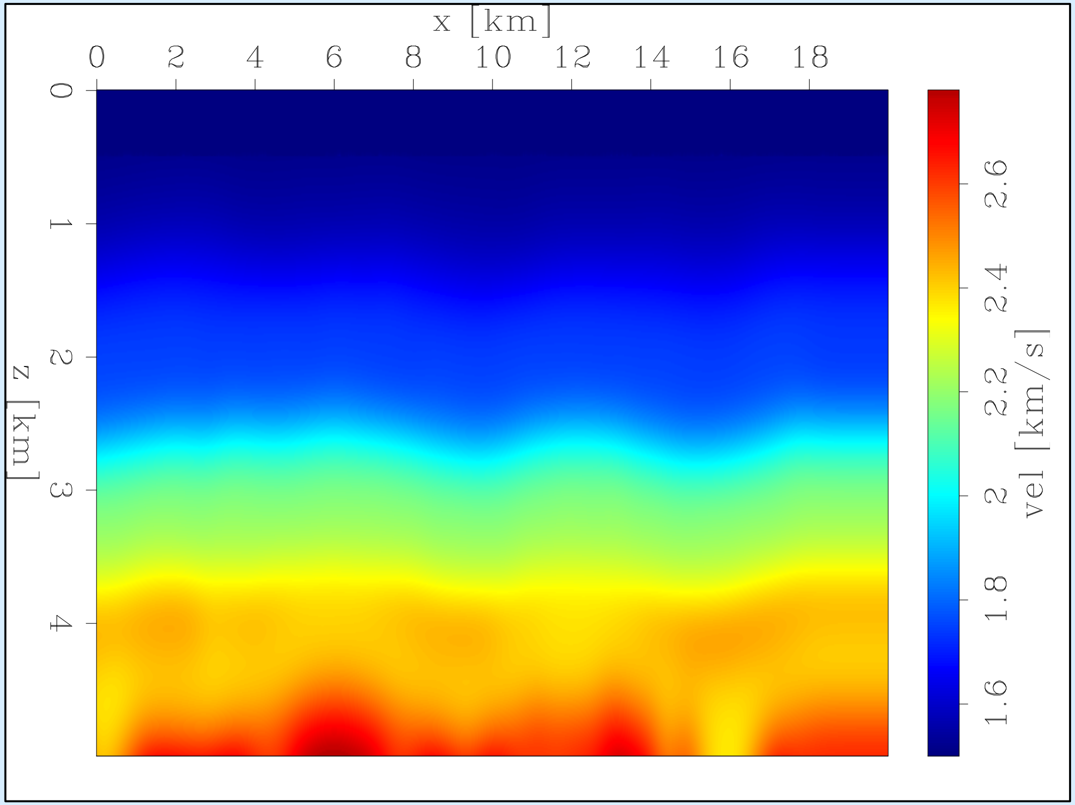
(Clapp, 2014)

The solver: non-linear test

Acoustic Full Waveform Inversion test

- 200 shots managed/submitted on a cluster by python script
- 2000 receivers at the surface
- Inverting 0-5 Hz in the recorded data
- Run 14 iterations of non-linear CG

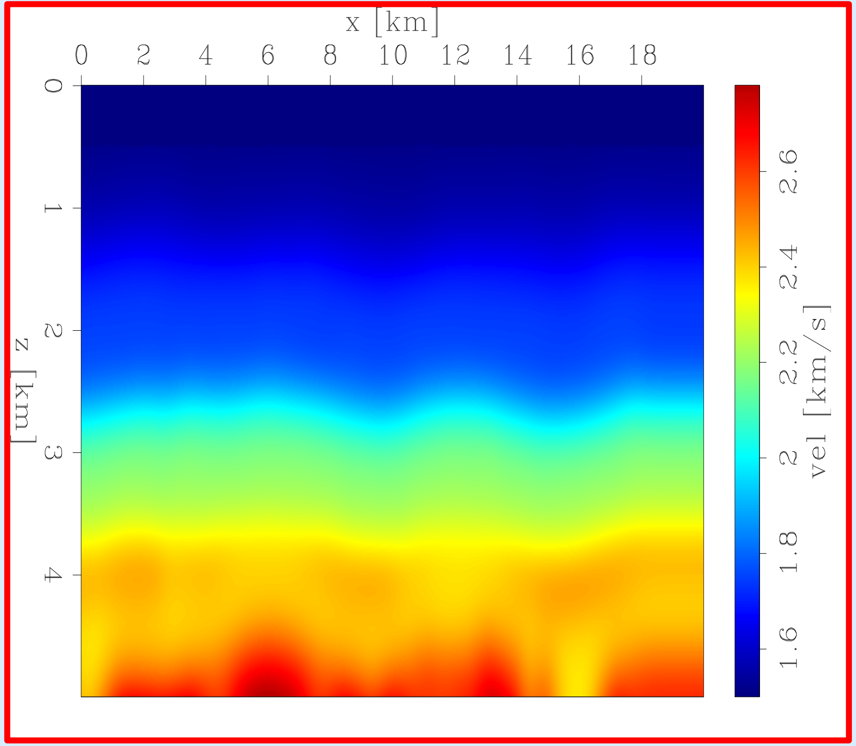
Starting model



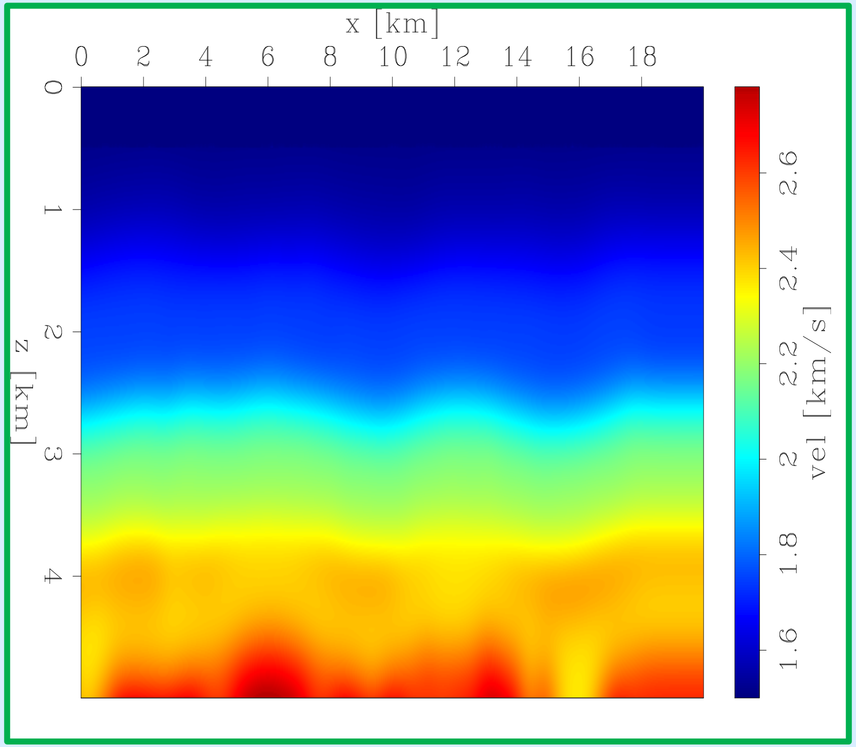
The solver: non-linear test

Starting model

Un-preconditioned

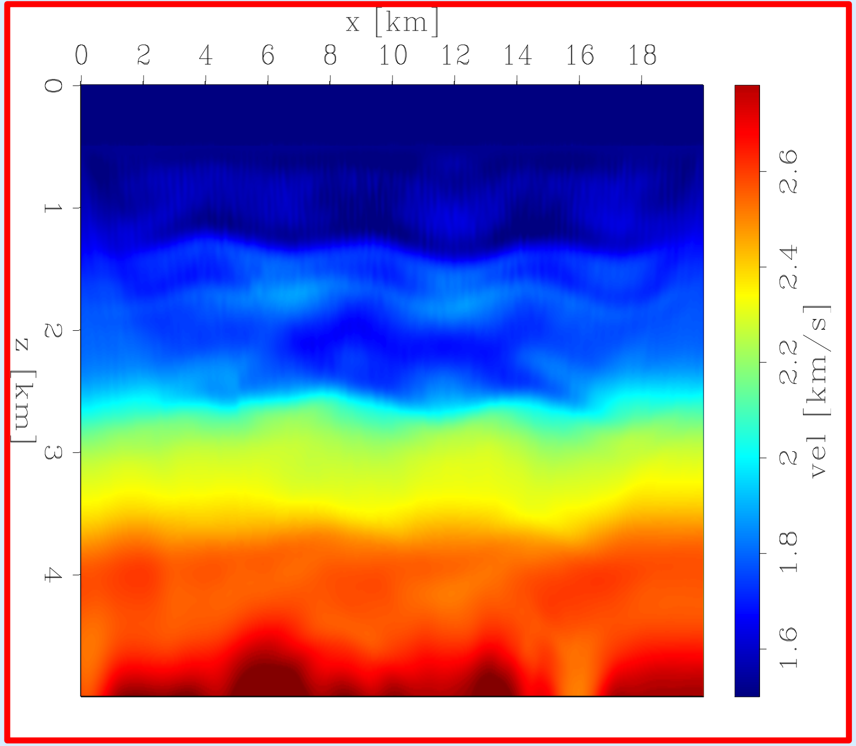


Preconditioned

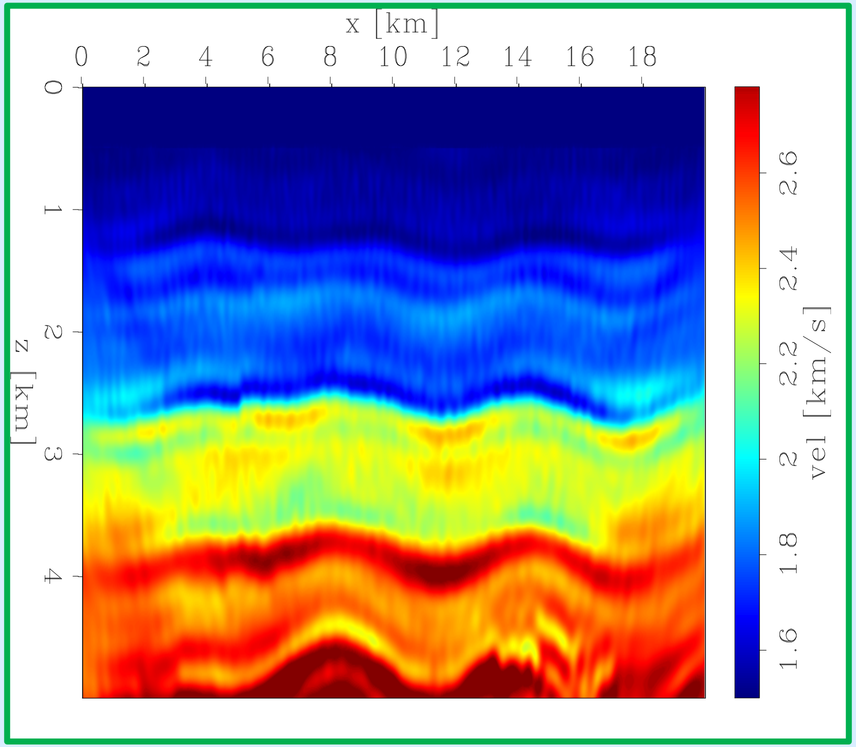


Inverted model

Un-preconditioned



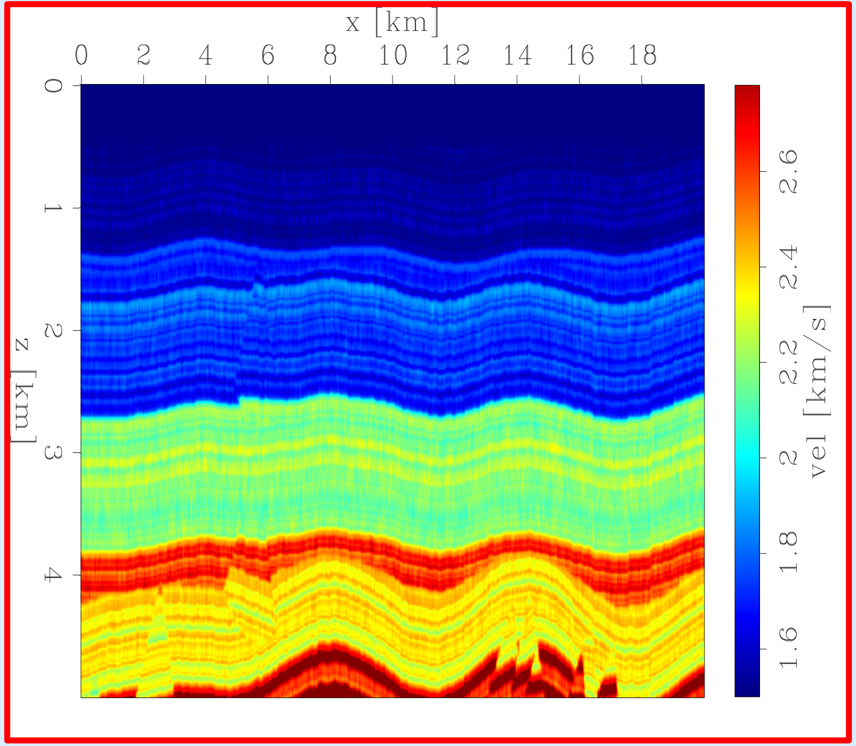
Preconditioned



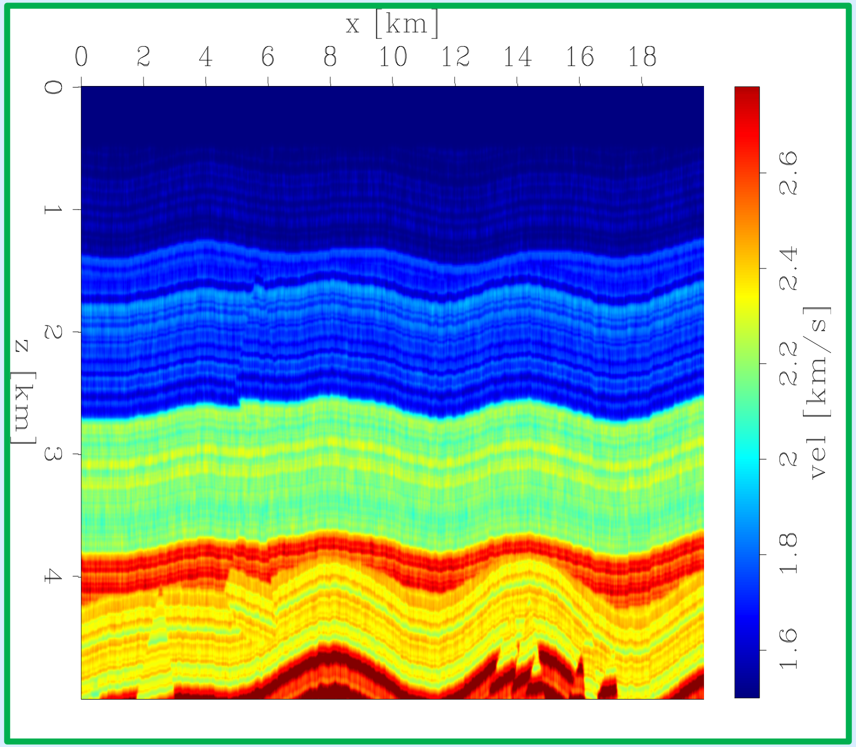
The solver: non-linear test

True model

Un-preconditioned

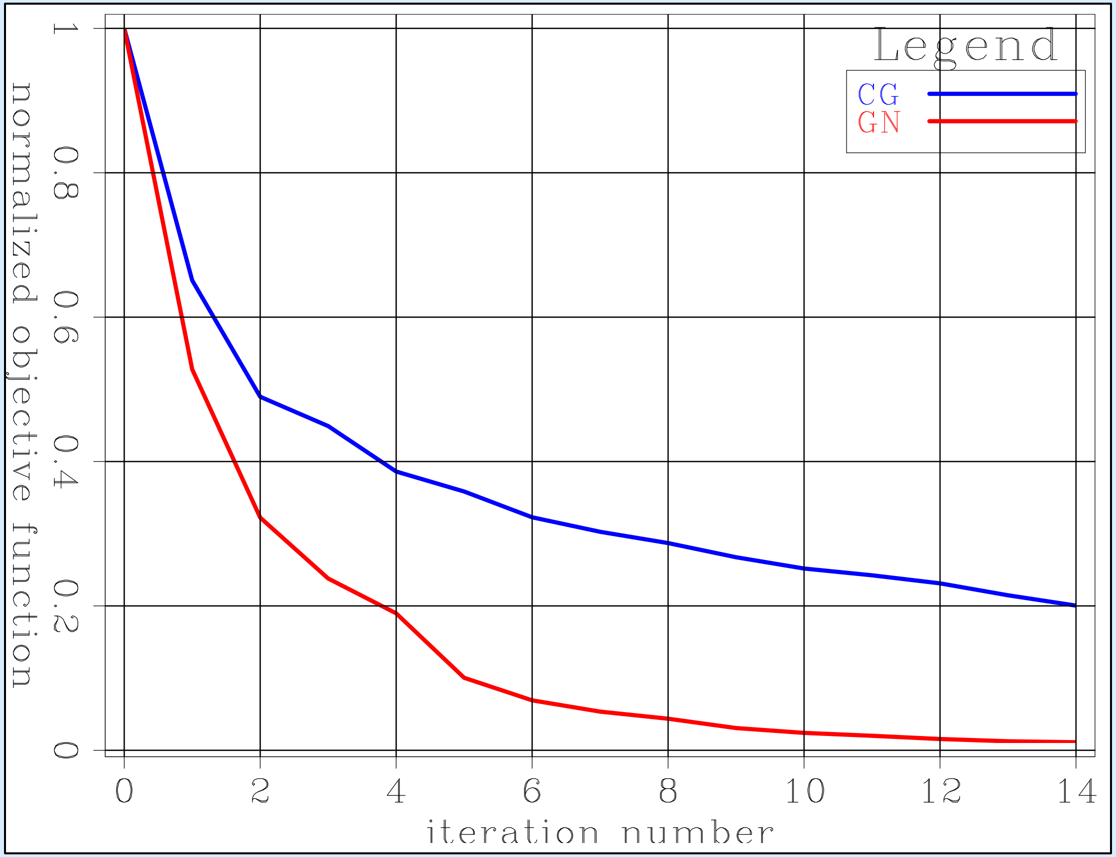


Preconditioned



The solver: non-linear test

Objective function comparison



Un-preconditioned (CG)

Preconditioned (GN)

The solver: current

What do we have now?

Problem object:

- Linear L2-norm problem with/without regularization
- Non-linear L2-norm problem with/without regularization

Solver object:

- Linear CG (symmetric/rectangular systems)
- Non-linear CG

Stepper object:

- Parabolic
- Sampling
- Linear

Stopper object:

- Criteria: time, residual norm, gradient norm, function evaluations, iterations

The solver: future

What we are going to add:

Problem object:

- Linear L2-norm problem with/without regularization (L1 reg)
- Non-linear L2-norm problem with/without regularization (L1 reg)

Solver object:

- Linear CG (symmetric/rectangular systems)
- Non-linear CG
- LBFGS

Stepper object:

- Parabolic
- Sampling
- Linear
- GN/Newton
- Brent
- Backtracking

Stopper object:

- Criteria: time, residual norm, gradient norm, function evaluations, iterations

Thank you for your attention Questions?