Abstractions for at-scale seismic inversion

Mathias Louboutin, Ali Siahkoohi, Ziyi Yin, Rafael Orozco, Thomas Grady,
Yijun Zhang, Philipp A. Witte*, Gabrio Rizzuti**, Felix J. Herrmann

* now at Azure
** now at Utrecht University

Rice Energy HPC, March 3rd, 2022

Georgia Tech College of Computing
School of Computational Science and Engineering

Georgia Tech College of Sciences
School of Earth and Atmospheric Sciences

Georgia Tech College of Engineering
School of Electrical and Computer Engineering

SLIM
Georgia Institute of Technology

Innovations – through abstractions

- vertical integration of abstractions at different levels (physics, lin. alg., ML)
- managing of complexity w/o loss in performance
- adding flexibility to facilitate integration w/
  - optimization (e.g. w/ SlimOptim.jl & SetIntersectionProjection.jl)
  - machine learning (e.g. w/ InvertibleNetworks.jl & Flux.jl)
  - fast automatic differentiation (e.g. w/ Julia's Zygote.jl)
  - other stakeholders (e.g. COFII)

Absolutely essential to

- increase rate of innovation
- reduce costs

[1] P. A. Witte, M. Louboutin, N. Kukreja, F. Luporini, M. Lange, G. G. Gorman and Felix J. Herrmann, 2019, *A large-scale framework for symbolic implementations of seismic inversion algorithms in Julia*, Geophysics, 84, 3.
[2] M. Louboutin, M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko and G. J. Gorman, 2019, Devito 3.1.0: an embedded domain-specific language for finite differences and geophysical exploration: Geoscientific model development, 12, 3.
[3] F. Luporini, M. Lange, M. Louboutin, N. Kukreja, J. Hückelheim, , C. Yount, P. A. Witte, P. H. J. Kelly, F. J. Herrmann and G. J. Gorman, 2019, Architecture and performance of Devito, a system for automated stencil computation, arXiv preprints.

The Julia Devito Inversion & ML framework

JUDI.jl & JUDI4Cloud:[1]

- matrix-free linear operators and abstract data containers
- parallel I/O based on look-up tables, parallelism (OMP + soon MPI)
- interface to Julia's machine learning package Flux
- *https://github.com/slimgroup/JUDI.jl/* (MIT license)

Devito: [2][3]

*Devito is a joint project between Imperial & Georgia Tech w/ Mathias one of the two main developers.

- a domain-specific language for finite-differences in Python
- code generation with automated performance optimizations
- model parallelism (MPI, multithreading)
- *https://github.com/devitocodes/devito* (MIT license)

InvertibleNetworks.jl:

- building blocks for invertible neural networks/normalizing flows
- scalable & gradients derived by hand
- *https://github.com/slimgroup/InvertibleNetworks.jl* (MIT license)

The Julia Devito Inversion framework (JUDI)

Julia

linear opeartors, data containers, IO

parallel modeling function

parallelization: distribute sources, data

serial modeling function

interface to Devito (Python)

Python

Devito: symbolic definition of PDE

automatic code generation
and JIT compilation

C

solve PDE w/ OpenMP parallelism

```
g = J'*(Pr*A_inv*Ps'*q - d_obs)
```

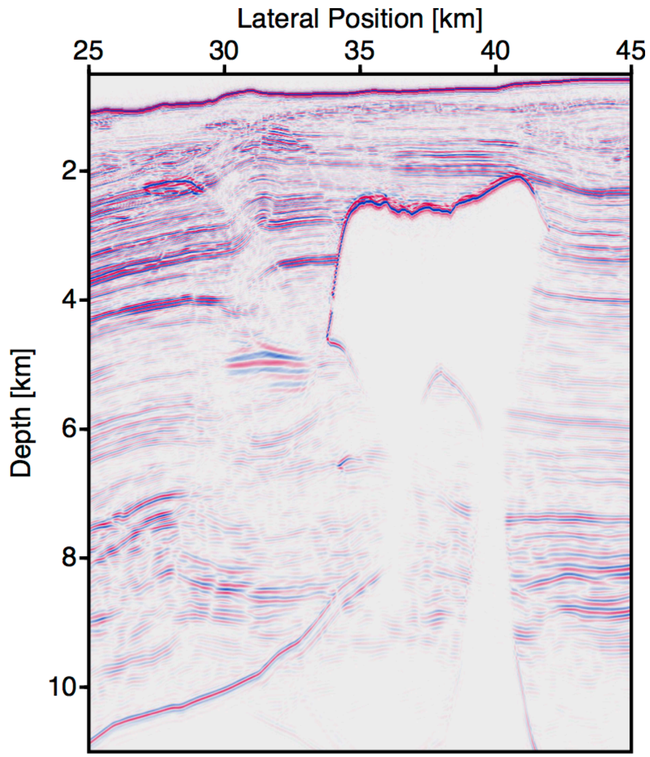
```
# Symbolic wavefield
u = TimeFunction(name='u', space_order=16)

# Acoustic wave equation
pde = model.m * u.dt2 - u.laplace
```

```
float (*rec_coords)[2] = (Float (*[2]) rec_coords_vec;
{
  struct timeval start_kernel, endkernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 0; i3 < 3; i3+=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
```

Example 1: Compressive seismic imaging

Linearized Bregman method with JUDI:

```
for j=1:maxiter

    # Compute residual and gradient
    i = randperm(d_obs.nsrc)[1:batchsize_source]
    select_frequencies!(J, batchsize_freq)
    r = Ml*J[i]*Mr*x - Ml*d_obs[i]
    g = Mr'*J[i]'*Ml'*proj_l2(r)

    # Residual and function value
    res[j] = norm(r, 2)
    fval[j] = λ*norm(C*z, 1) + .5f0*norm(C*z, 2)^2

    # Update variables
    global z -= α*g
    global x = C*soft_thresholding(C*z, λ)
end
```



Deep Learning

Integration JUDI operators w/ deep learning packages (e.g. Flux.jl):

```
network(x) = W2*(J*(W1*x + b1)) + b2
# Or:
network(x) = conv2(ℱ(conv1(x)))
loss(x, y) = Flux.mse(network(x), y)
```

Once again, abstractions pay off:

- No need to backpropagate through solvers using automatic differentiation (AD)
- Backpropagation through JUDI operators

```
@adjoint *(J::judiJacobian, x) = *(J, x), Δ -> (nothing, transpose(J) * Δ)
```

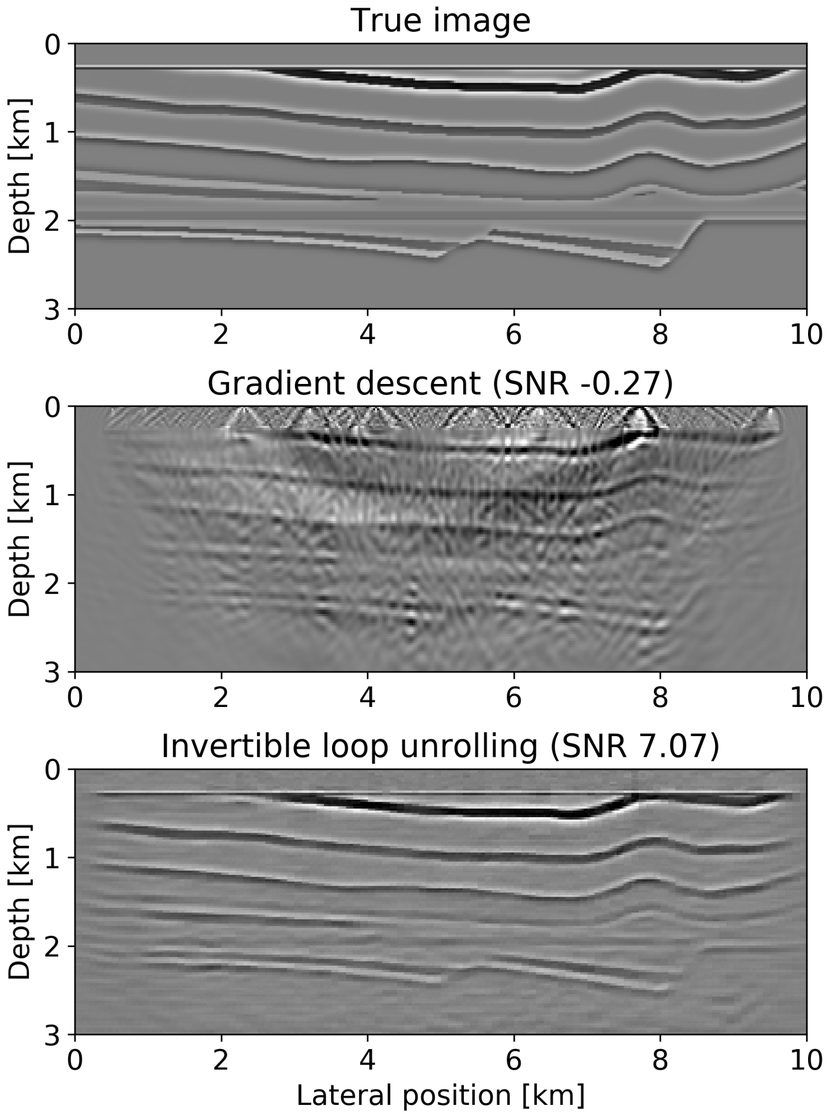
Loop unrolling LSRTM

Invertible version of loop-unrolled network:

- Inspired by i-RIM and Real NVP
- Invertible + tractable logdet

Migration-demigration

1. **function** $\mathcal{G}(\mathbf{J}, \mathbf{d})$
2. $\quad \mathbf{x} = 0$
3. $\quad$ **for** $j = 1, ..., n$
4. $\quad\quad \mathbf{x} = \mathbf{Q}\mathbf{x}$
5. $\quad\quad \mathbf{x}'_1 = \mathbf{x}_1$
4. $\quad\quad \mathbf{g} = \mathbf{J}^\top\left(\mathbf{J}\mathbf{x}'_1[1] - \mathbf{d}\right)$
5. $\quad\quad \mathbf{s}', \mathbf{t} = \mathrm{NN}\left([\mathbf{g}, \mathbf{x}'_1[2{:}\mathrm{end}]]\right)$
6. $\quad\quad \mathbf{s} = \sigma\left(\mathbf{s}'\right)$
6. $\quad\quad \mathbf{x}'_2 = \mathbf{x}_2 \odot \mathbf{s} + \mathbf{t}$
3. $\quad\quad \mathbf{x} = \mathbf{Q}^\top\mathbf{x}'$
12. $\quad$ **end**
13. $\quad$ return $\mathbf{x}$
14. **end**



Serverless abstractions on Azure

JUDI operators in Azure Batch w/ JUDI4Cloud.jl

```
JUDI4Cloud.init_clusterless(n_instances; n_julia_per_instance=2)
# Setup operators
Pr = judiProjection(info, recGeometry)
F = judiModeling(info, model; options=opt)
F0 = judiModeling(info, model0; options=opt)
Ps = judiProjection(info, srcGeometry)
J = judiJacobian(Pr*F0*adjoint(Ps), q)

# Nonlinear modeling
dobs = Pr*F*adjoint(Ps)*q
```

← Setup Azure Batch

← Run on Azure & gather results

Once again, abstractions pay off:

- no need to refactor code
- readable codes

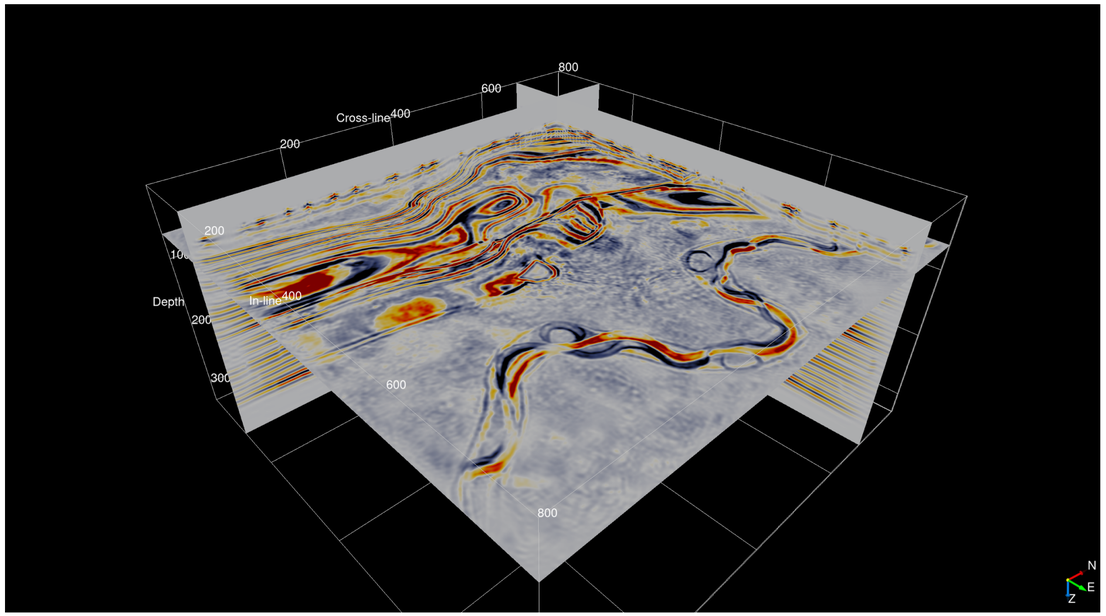
Witte, Philipp A., et al. "An event-driven approach to serverless seismic imaging in the cloud." *IEEE Transactions on Parallel and Distributed Systems* 31.9 (2020): 2032-2049.

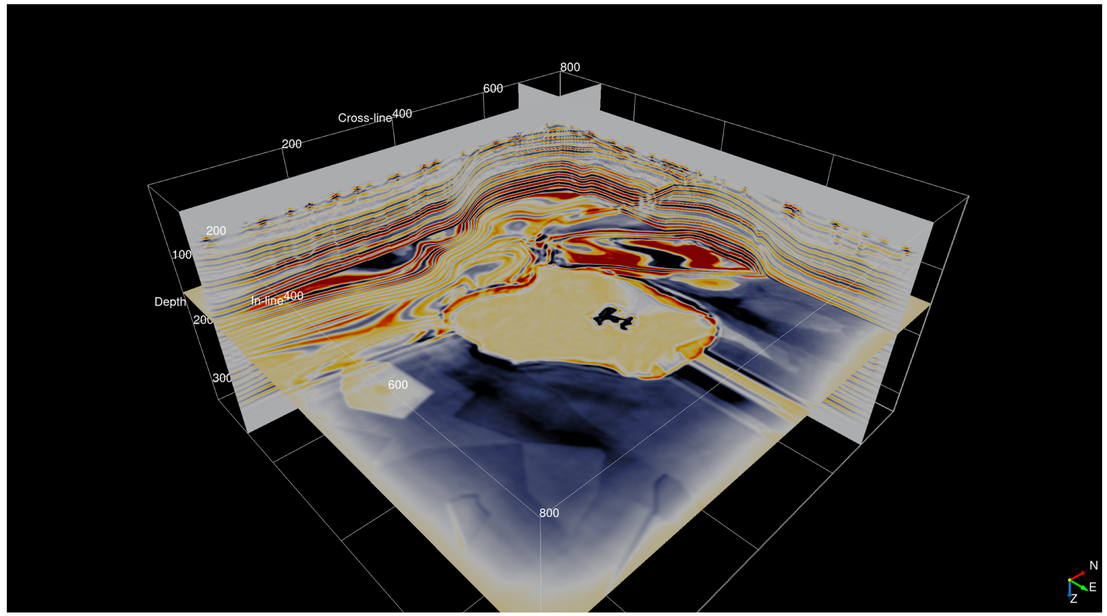
SLIM
ML4Seismic

Serverless seismic imaging on Azure

“Small” 3D Imaging case study w/ Devito DSL

- Data set: 1,500 source locations (~**2.1 TB** data)
- Model: 10 x 10 x 3.325 km (**270 million** unknowns)
- PDE: tilted transversely isotropic (TTI) wave equation, **3,500** time steps
- **Cost: < 10,000$** on 100 E64/E64s instances (2 VMs per gradient with MPI)
- **Peak performance: 140 TFLOPs**
- **cost single image is comparable to training a large NN**





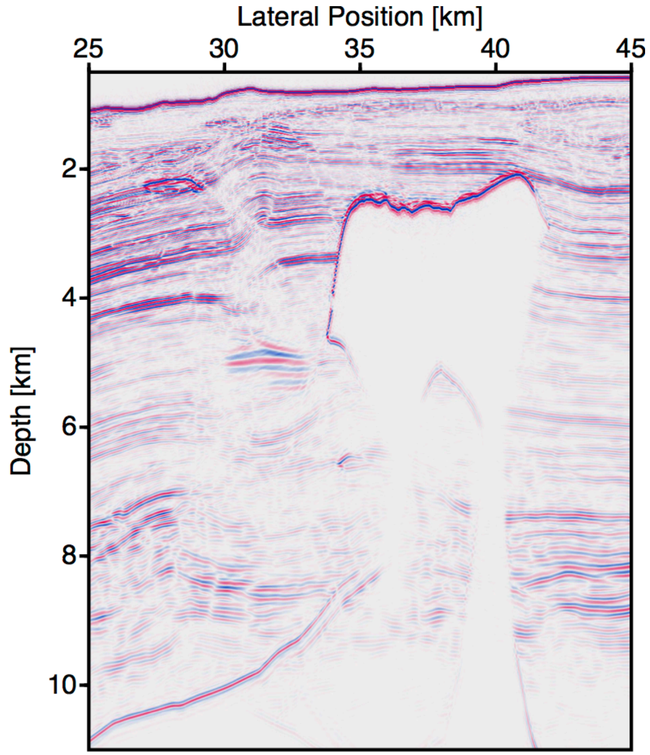
Compressive seismic imaging

JUDI4Cloud abstraction

Linearized Bregman method with JUDI:

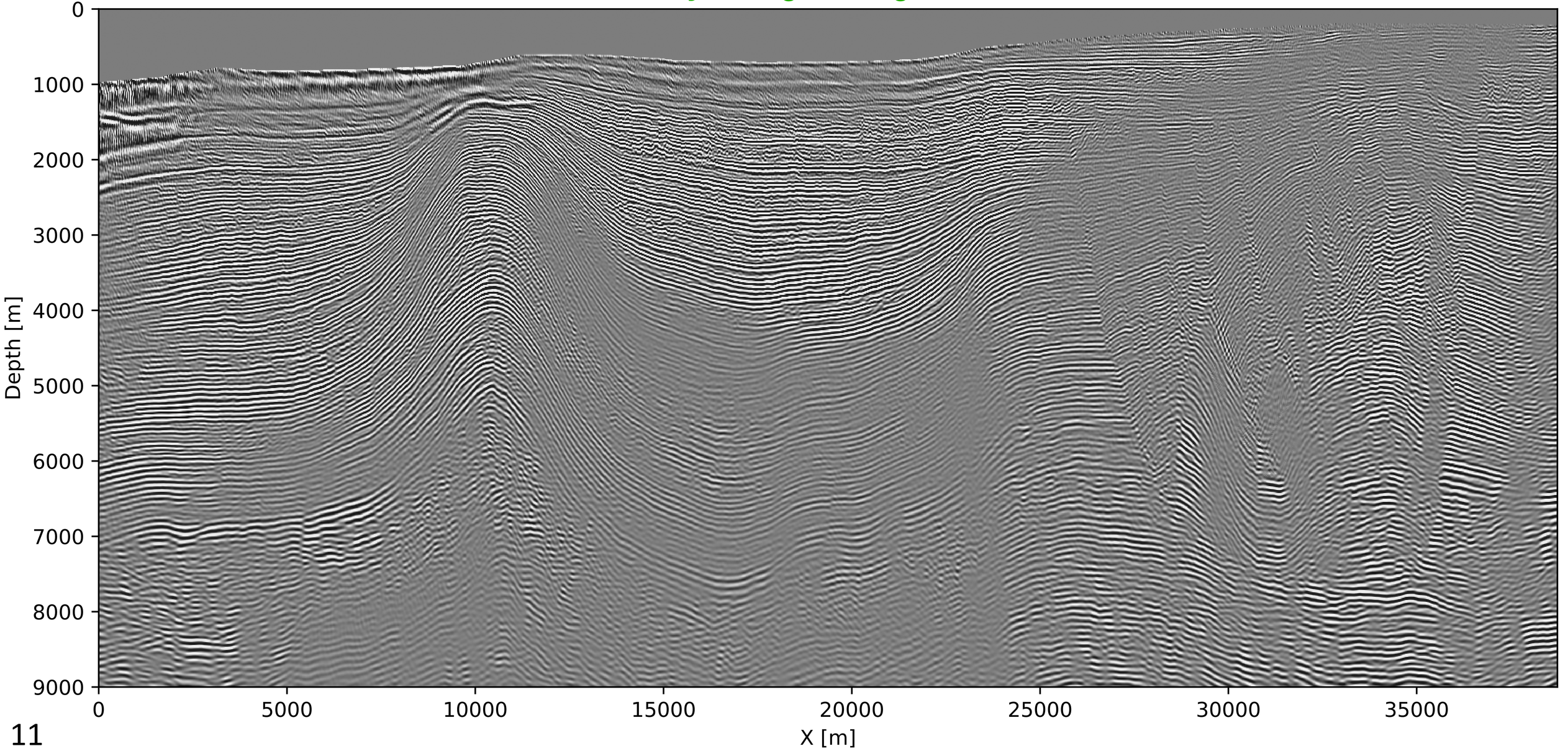
Now runs on Azure

```
for j=1:maxiter

    # Compute residual and gradient
    i = randperm(d_obs.nsrc)[1:batchsize_source]
    select_frequencies!(J, batchsize_freq)
    r = Ml*J[i]*Mr*x - Ml*d_obs[i]
    g = Mr'*J[i]'*Ml'*proj_l2(r)

    # Residual and function value
    res[j] = norm(r, 2)
    fval[j] = λ*norm(C*z, 1) + .5f0*norm(C*z, 2)^2

    # Update variables
    global z -= α*g
    global x = C*soft_thresholding(C*z, λ)
end
```

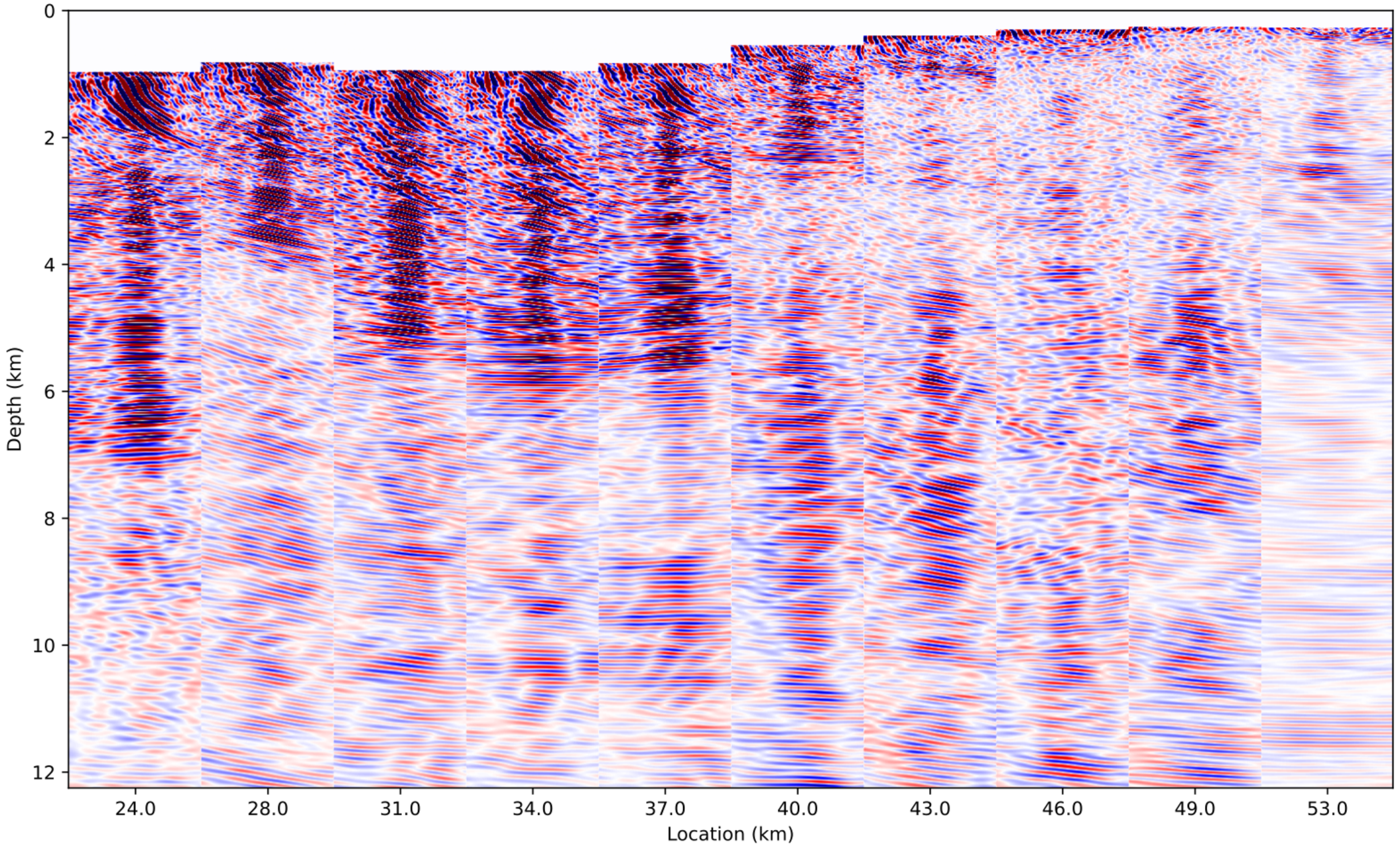


GPU accelerated clusterless TTI RTM

160 × memory savings through trace estimation



And subsurface CIG!



Inter-operability

Combine JOLI operator with Chevron's COFII's Jets

```
using JOLI, JetPack, Jets
# 1D real DFT with 1000 samples
J = joDFT(1000; DDT=Float64, RDT=Float64)
# Jet operator for circular shift by 25 samples
jop = JopCircShift(JetSpace(Float64, 1000), 25);

a = randn(1000)
# Parenthesis needed to prevent julia to compute J*jop first
a = randn(1000);b = J*(jop*a); c = jop'*(J'*b);
# dot test
@show dot(b, b),    dot(c, a),  dot(b, b) - dot(c, a), dot(b, b) / dot(c, a)
# 490.00330785, 490.003307859, 1.1368683772161603e-13, 1.0000000000000002)
```

Once again, abstractions pay off:

- no need to refactor code
- readable codes
- seamless integration JUDI & COFII

https://github.com/slimgroup/ConstrainedFWIExamples/blob/master/notebooks/02_constr_fwi_jetpack.ipynb
https://github.com/ChevronETC/Examples/blob/main/50_fwi/11_constrained_fwi_pqn.ipynb

SLIM

More advanced interplay

COFII propagator

SetIntersectionProjection.jl
Projection onto intersection of constraints

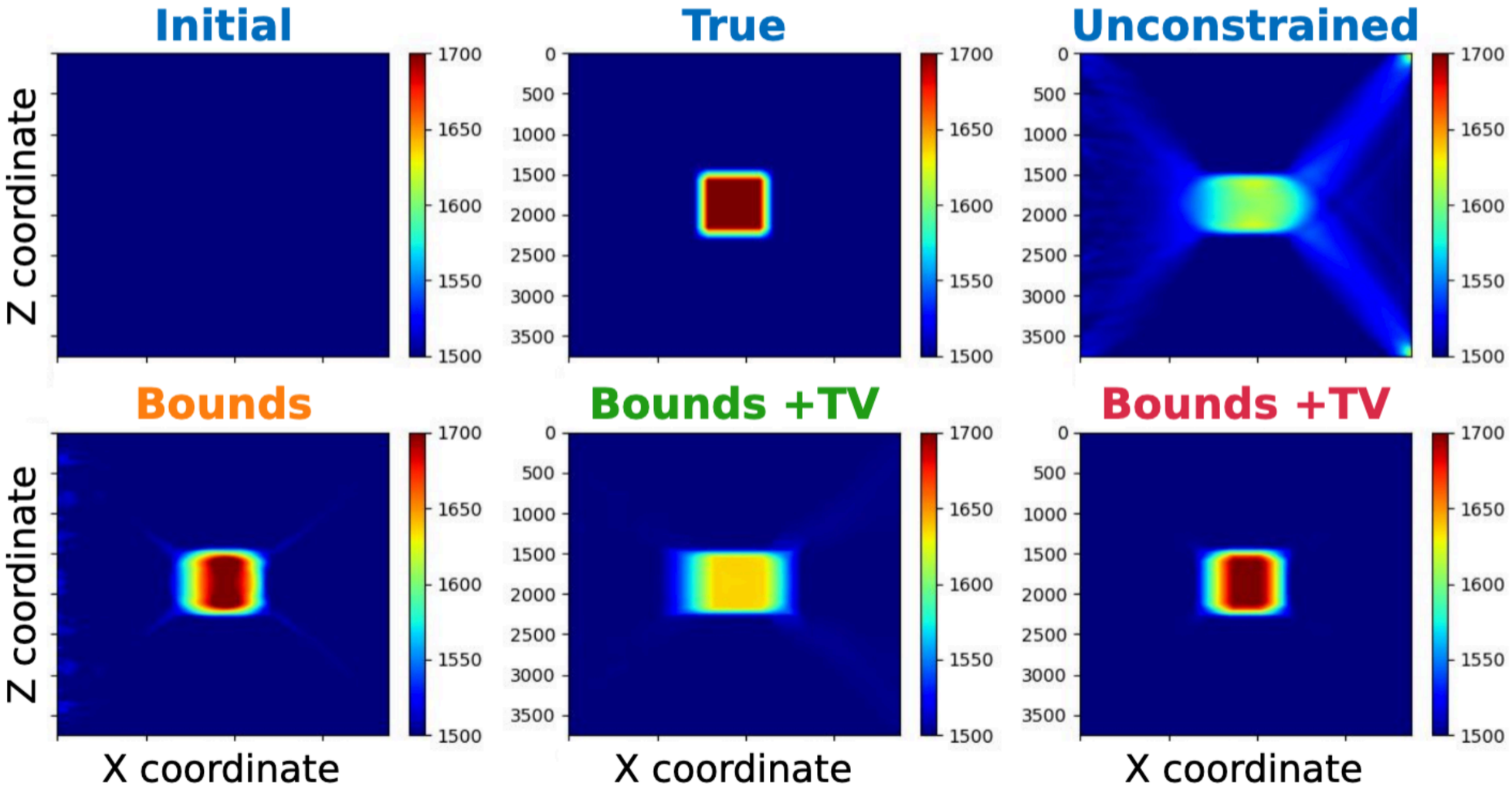
SlimOptim.jl
quasi-newton

```
sol = pqn(g, vec(v2), prj, options_pqn);
```

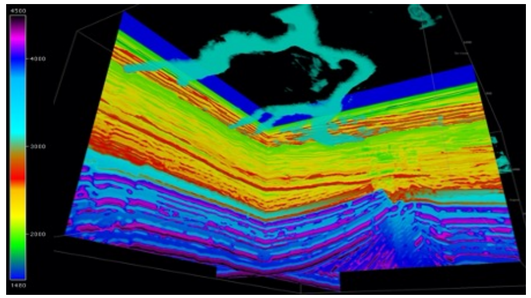
```
Running PQN...
Number of L-BFGS Corrections to store: 10
Spectral initialization of SPG: 1
Maximum number of SPG iterations: 10
SPG optimality tolerance: 1.00e-06
SPG progress tolerance: 1.00e-07
PQN optimality tolerance: 1.00e-05
PQN progress tolerance: 0.00e+00
Quadratic initialization of line search: 0
Maximum number of iterations: 25
input to PARSDMM is feasible, returning
p.gscale = 163.19803f0
 Iteration   FunEvals  GradEvals Projections     Step Length    Function Val        Opt Cond
input to PARSDMM is feasible, returning
         0          0          0          0     0.00000e+00     5.47293e+06     1.00000e+01
input to PARSDMM is feasible, returning
input to PARSDMM is feasible, returning
input to PARSDMM is feasible, returning
input to PARSDMM is feasible, returning
         1          3          3          6     1.00000e+00     5.28328e+06     9.59644e+00
```

https://github.com/ChevronETC/Examples/blob/main/50_fwi/11_constrained_fwi_pqn.ipynb

Mored advanced interplay

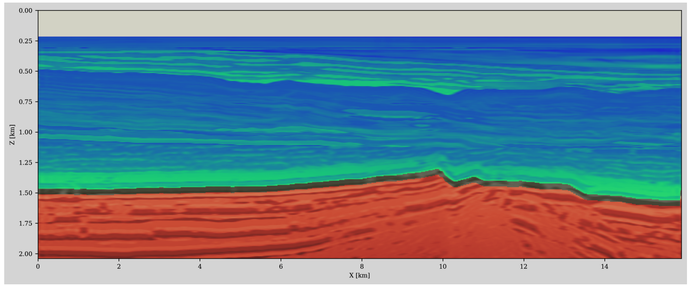


Seismic4CCS – Simulation-based seismic monitoring for CCS workflow



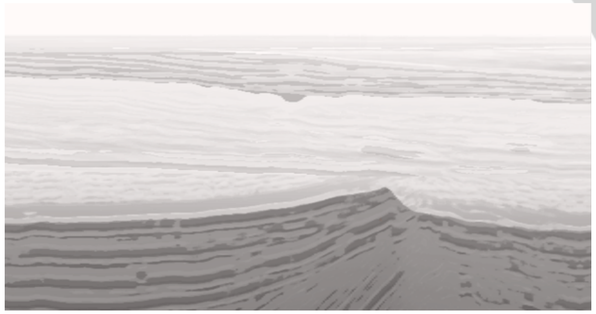
Seismic model
wave speed (c_p), density (ρ)

Conversion
$(\rho, c_p) \longrightarrow (\kappa, \phi)$



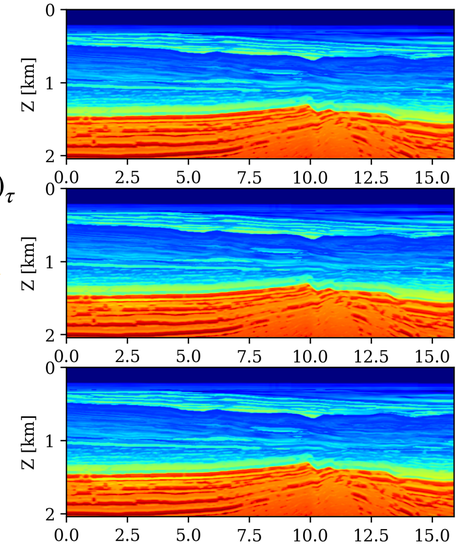
Fluid model
permeability (κ), porosity (ϕ)

Proxy Flow
$(\kappa, \phi) \longrightarrow S_\tau$



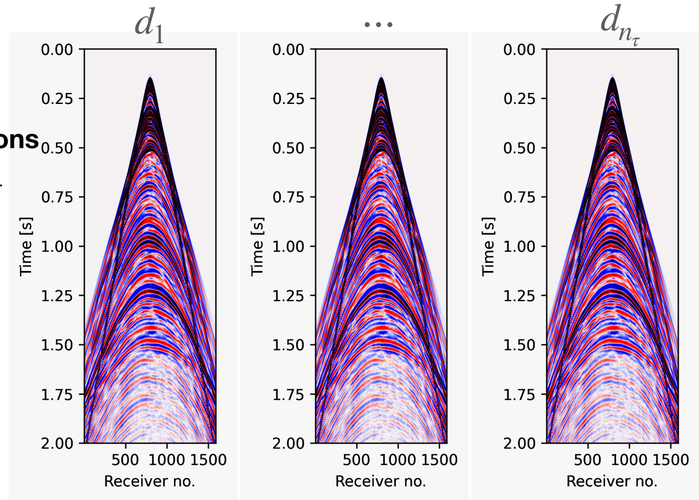
CO_2 dynamics
saturation, pressure

Conversion
$S_\tau \longrightarrow (\rho, c_p)_\tau$



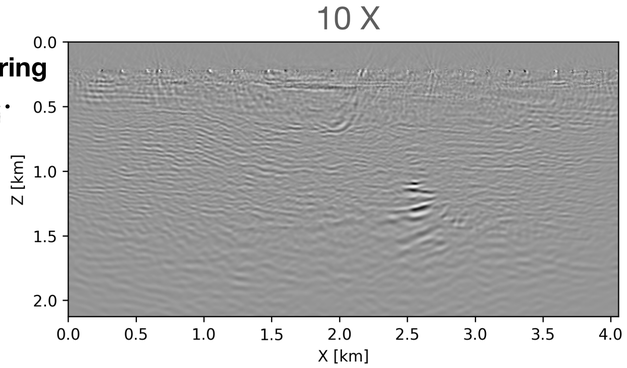
Time-lapse seismic model
wave speed, density

Seismic simulations
$(\rho, c_p)_\tau \longrightarrow d_\tau$



Time-lapse seismic
pressure data

Seismic monitoring
$\delta m_2 - \delta m_1, \delta m_3 - \delta m_1, \cdots$



Time-lapse seismic images
impedance contrast

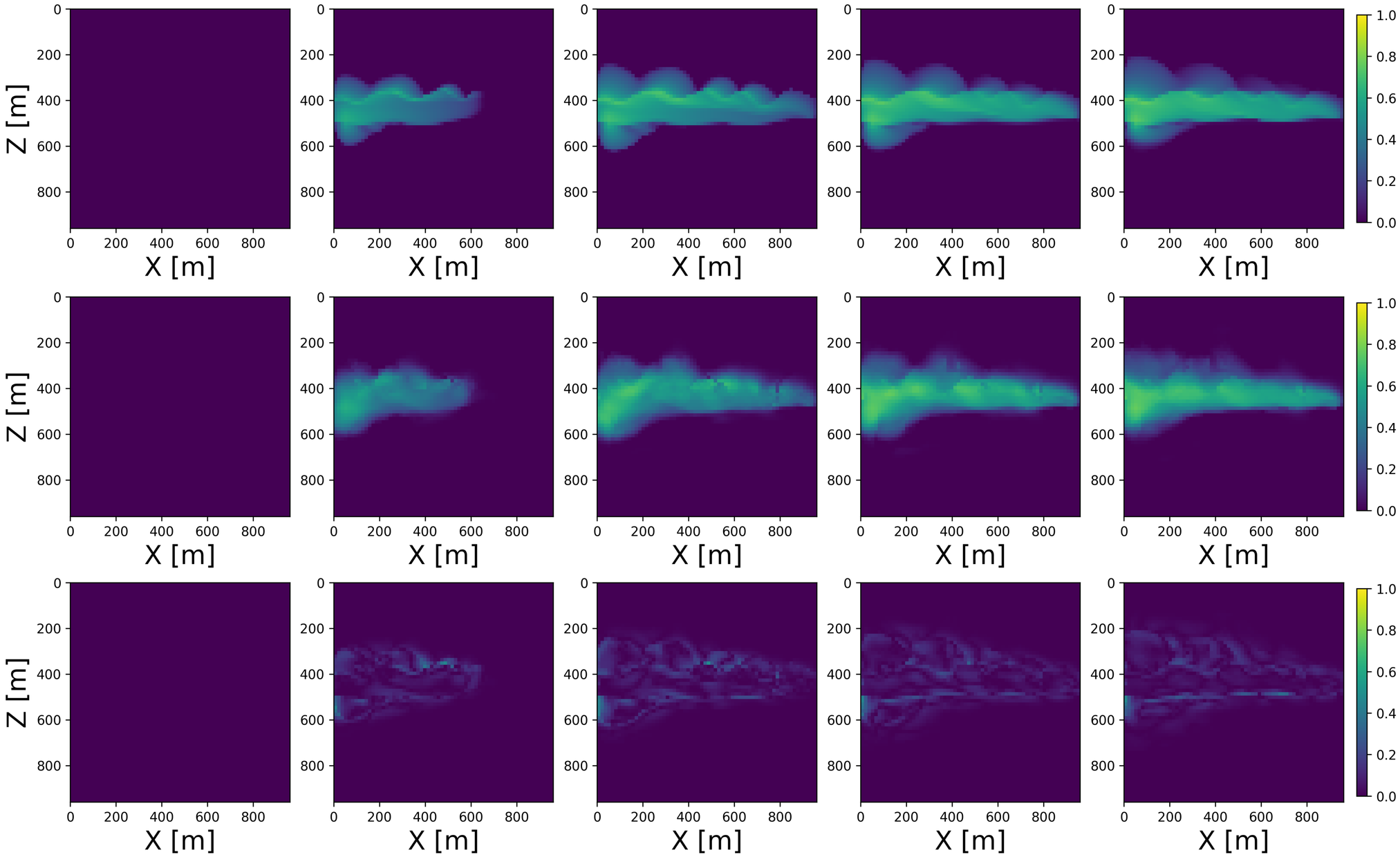
Inversion

predicted CO_2 plumes from inverted $\mathbf{K}$

Ground truth

Predicted

Difference



Observations

Abstractions control complexity

Natural coupling of physics, optimization, ML, Cloud, …

Allows for extendable fast agnostic automatic differentiation (AD)

Enables inter-operability and collaborations