The next releases of Seis packages

Breno Bahia and Mauricio Sacchi

SAIG Annual Meeting

January 2022

Goals

- Seis Packages

- Machine Learning in Julia
    - Model adaptation
        - ✓ Perturb & Parametrize
        - ✓ Reuse & Regularize

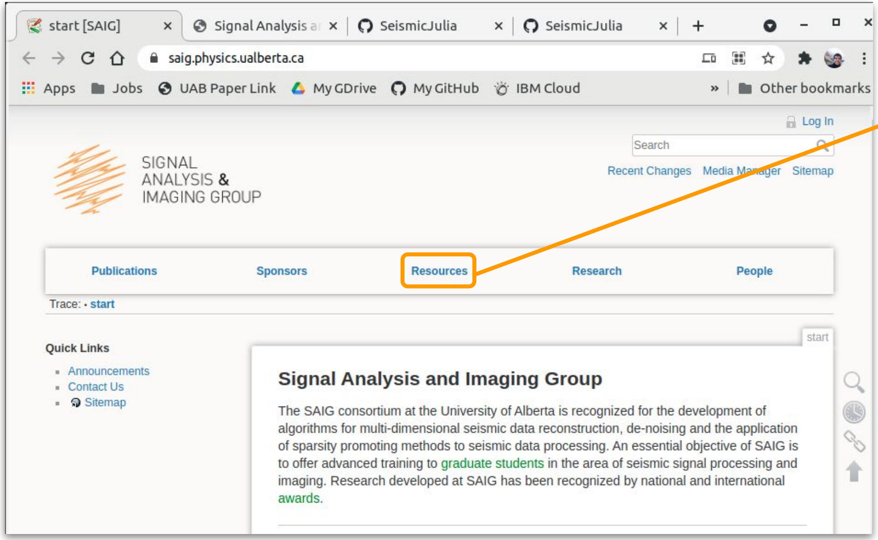
- Upcoming changes
    - Optimization & Operators
        - ✓ Regularization by denoising

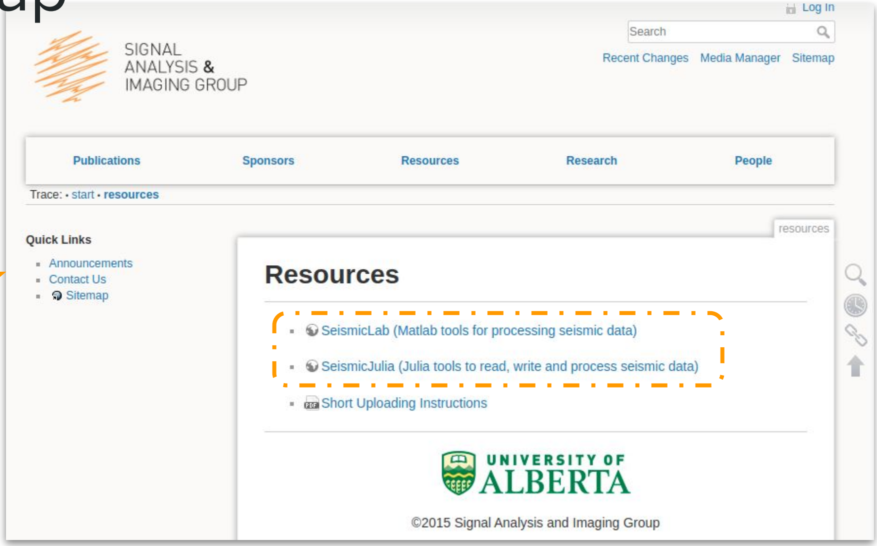
    - SeisProcessing.jl
        - ✓ SeisReconstruction.jl
        - ✓ SeisDenoise.jl
        - ✓ SeisDeblend.jl

    - SeisAcoustic.jl
        - ✓ Docs

Signal Analysis and Imaging Group

Group website:



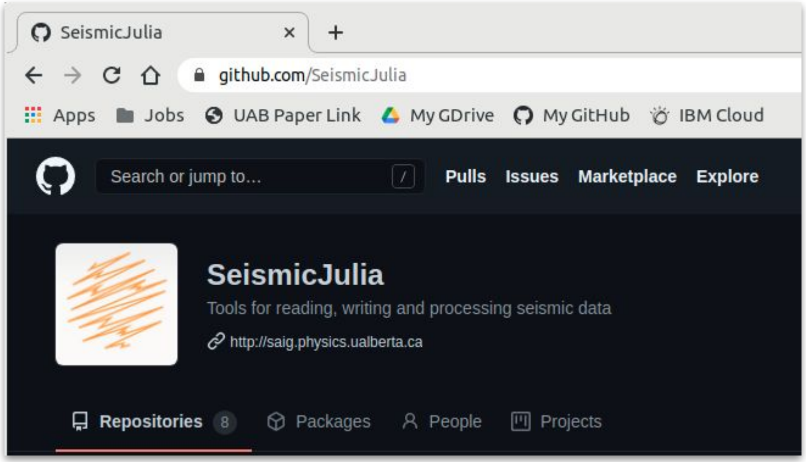


We can find the resources available here:

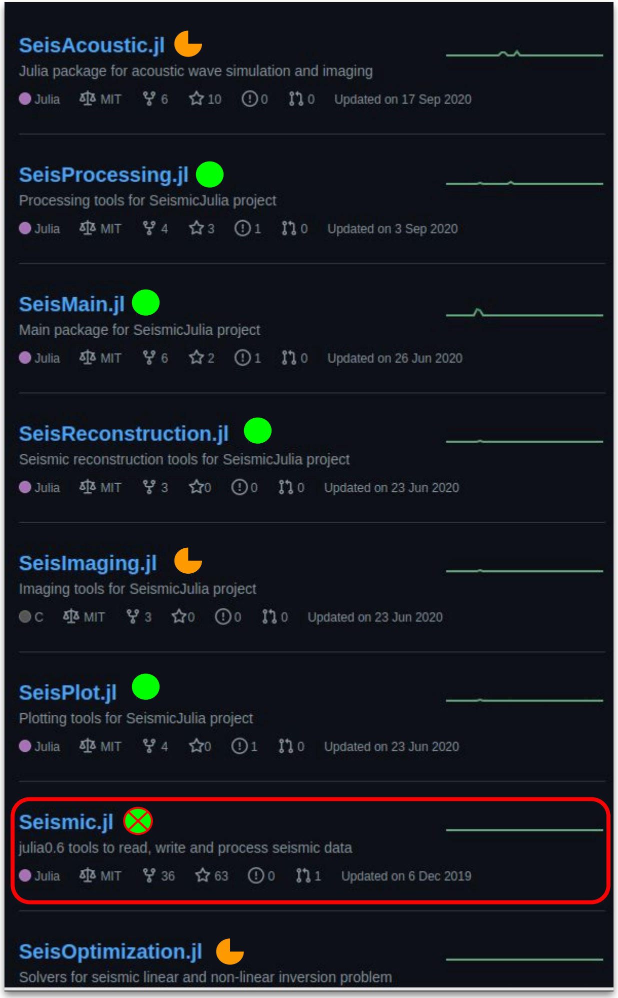
- SeismicLab
- Seismic.jl

SeismicJulia

SeismicJulia GitHub:



Registered
Not maintained
Not registered yet



Scientific Machine Learning (SciML)

SciML tries to go beyond the early attempts to bring machine learning into scientific computing.

1) Less data is required
2) Prevents overfitting
3) Exploits existing knowledge and tools

Revisits the scientific computing theory and envisions where an **universal approximator, such as a neural network,** might fit well.

- Automatic differentiation framework is key
  - Zygote.jl
  - Diffractor.jl

flux

SciML

Open Source Software for Scientific Machine Learning

https://sciml.ai

Specialized packages:

✓ DiffEqFlux.jl

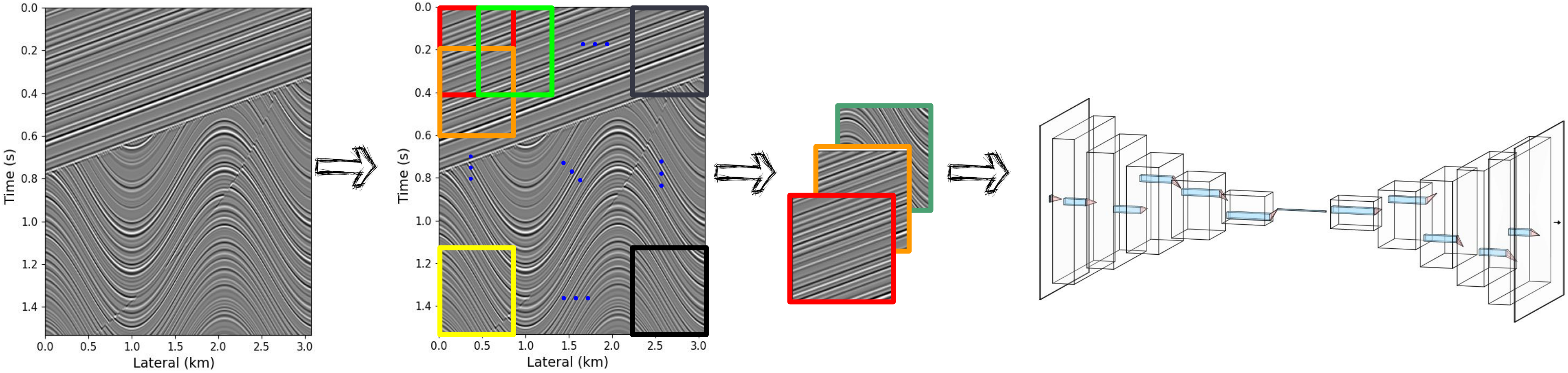
✓ NeuralPDE.jl

Deep learning for inverse problems

Deep learning for Inverse Problems

$$J(\theta) = \sum_i \left(\sum_{ix} \sum_{it} \left(\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}^{(i)} \mathbf{U}_t^{(i)} \right)^2 + \lambda_x \|\mathbf{D}_x \hat{\mathbf{P}}^{(i)}\|_2^2 + \lambda_t \|\mathbf{D}_t \hat{\mathbf{P}}^{(i)}\|_2^2 \right)$$

$$\mathbf{P}_\theta^{(i)} = \mathcal{N}_\theta(\mathbf{U}^{(i)}) = \mathcal{D}_\gamma(\mathcal{E}_\phi(\mathbf{U}^{(i)}))$$



Deep learning for Inverse Problems

Deep Image Prior (DIP) (Ulyanov et al, 2018)

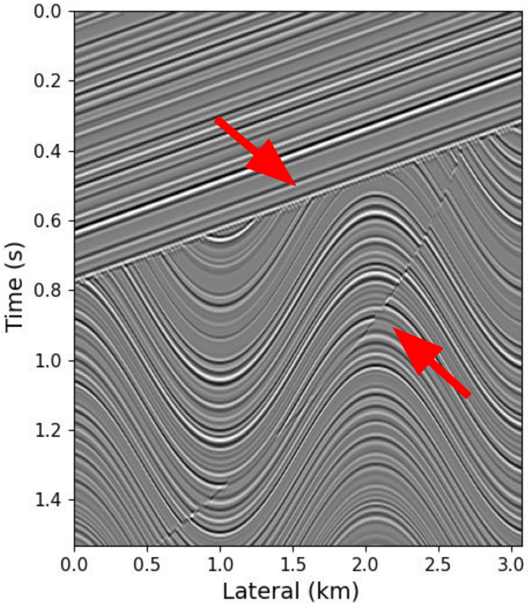
$$J(\theta) = \|\mathbf{y} - \mathbf{A} f_\theta(\mathbf{z})\|_2^2 + \lambda \mathcal{R}(f_\theta(\mathbf{z}))$$

$$\hat{\mathbf{x}} = f_{\theta*}(\mathbf{z})$$

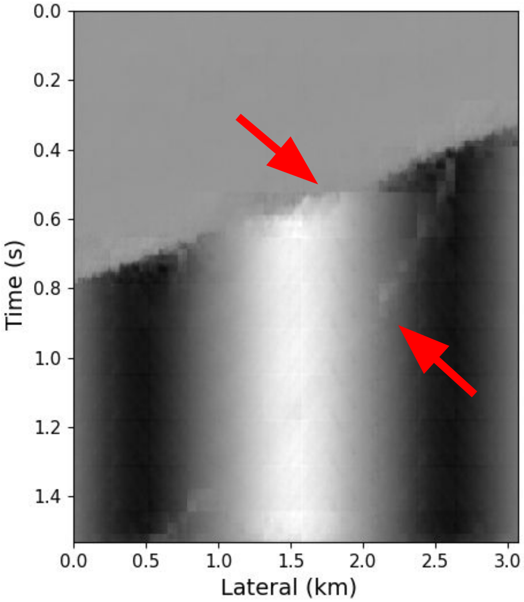
$$J(\theta) = \sum_i \left(\sum_{ix} \sum_{it} \left(\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}^{(i)} \mathbf{U}_t^{(i)} \right)^2 + \lambda_x \|\mathbf{D}_x \hat{\mathbf{P}}^{(i)}\|_2^2 + \lambda_t \|\mathbf{D}_t \hat{\mathbf{P}}^{(i)}\|_2^2 \right)$$

(Ulyanov et al (2018): https://arxiv.org/abs/1711.10925)

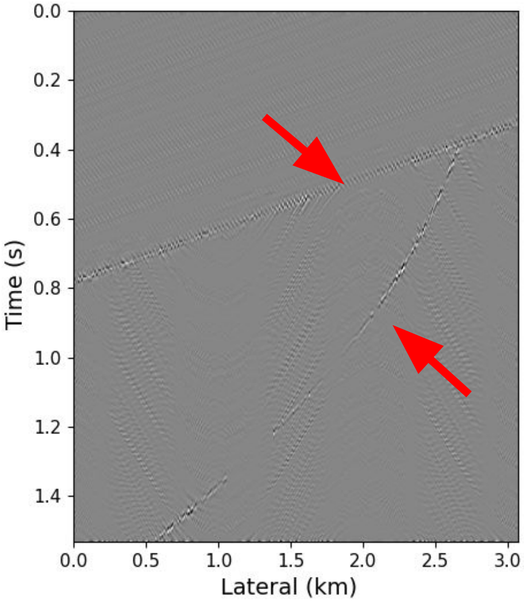
Deep learning for Inverse Problems



Input section

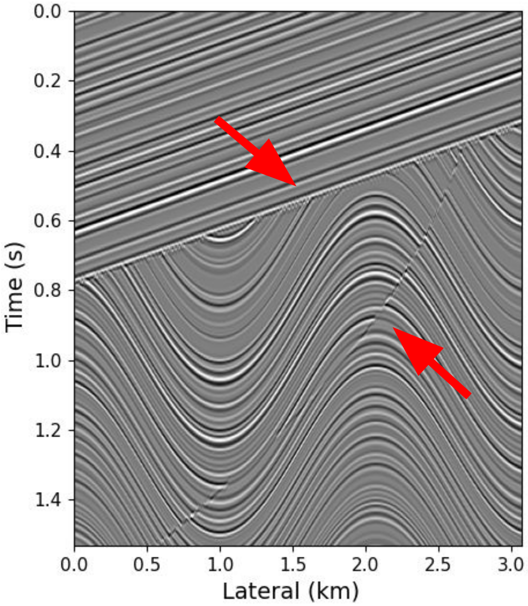


Estimate slope field after 1000 epochs of ADAM

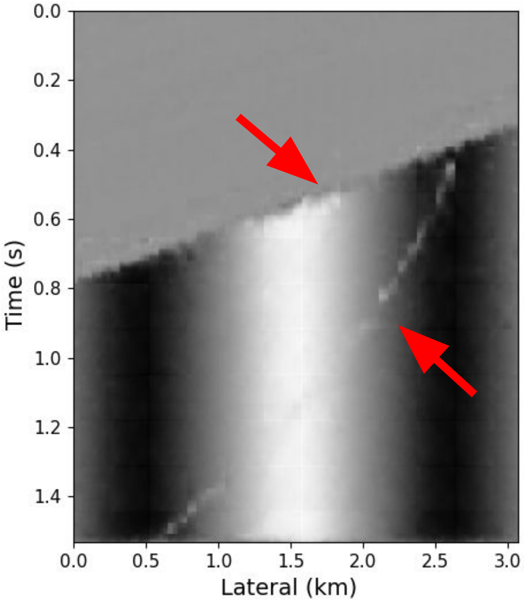


PWD result with obtained slope field

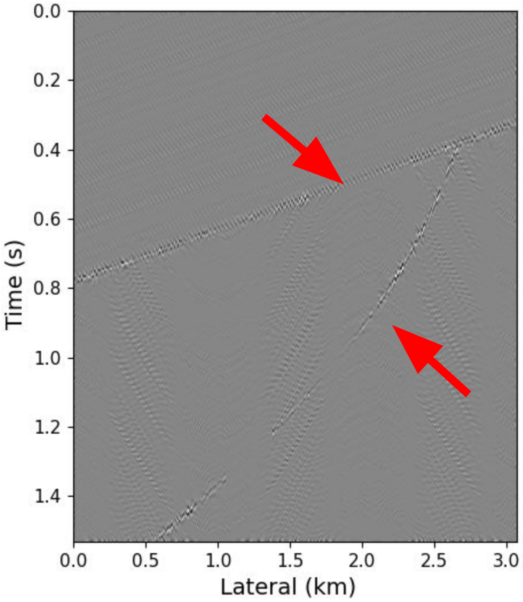
Deep learning for Inverse Problems



Input section



Estimate slope field after 500 epochs of ADAM



PWD result with obtained slope field

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) *- On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) *- On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

$$\mathbf{y}_0 = \mathbf{A}_0\mathbf{x} + \varepsilon$$

Trained solver

$$\mathbf{x}^* = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_0, \hat{\theta})$$

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) *- On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

$$\mathbf{y}_0 = \mathbf{A}_0\mathbf{x} + \varepsilon$$

Trained solver

$$\mathbf{x}^* = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_0, \hat{\theta})$$

$$\mathbf{y}_1 = \mathbf{A}_1\mathbf{x} + \varepsilon'$$

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) *- On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

$$\mathbf{y}_0 = \mathbf{A}_0\mathbf{x} + \varepsilon$$

Trained solver

$$\mathbf{x}^* = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_0, \hat{\theta})$$

$$\mathbf{y}_1 = \mathbf{A}_1\mathbf{x} + \varepsilon'$$

$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_1, \hat{\theta})$$

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) - *On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

$$\mathbf{y}_0 = \mathbf{A}_0\mathbf{x} + \varepsilon$$

Trained solver

$$\mathbf{x}^* = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_0, \hat{\theta})$$

$$\mathbf{y}_1 = \mathbf{A}_1\mathbf{x} + \varepsilon'$$

$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_1, \hat{\theta})$$

$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_1, \mathbf{y}_1, \hat{\theta})$$

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) *- On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

$$\mathbf{y}_0 = \mathbf{A}_0\mathbf{x} + \varepsilon$$

Trained solver

$$\mathbf{x}^* = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_0, \hat{\theta})$$

Bad performance:

$$\mathbf{y}_1 = \mathbf{A}_1\mathbf{x} + \varepsilon'$$

$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_1, \hat{\theta})$$

$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_1, \mathbf{y}_1, \hat{\theta})$$

Model drift

Deep learning for Inverse Problems

Deep learning algorithms typically yield *unstable* methods for inverse problems...

Antun et al. (2020) *- On instabilities of deep learning in image reconstruction and the potential costs of AI. PNAS.*

… but it can *complement and improve* existing methods in *scientific computing and inverse problems*.

Gilton et al. (2021) *- Model adaptation for inverse problems in imaging. IEEE.*

Network parameters are still useful

Perturb & Parametrize **(P&P)** → Requires retraining

Reuse & Regularize **(R&R)** → No retraining

$$N(\mathbf{A}_0) \approx N(\mathbf{A}_1)$$

(similar null spaces)

$$\mathbf{y}_0 = \mathbf{A}_0\mathbf{x} + \varepsilon$$

Trained solver

$$\mathbf{x}^* = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_0, \hat{\theta})$$

Bad performance:

$$\mathbf{y}_1 = \mathbf{A}_1\mathbf{x} + \varepsilon'$$

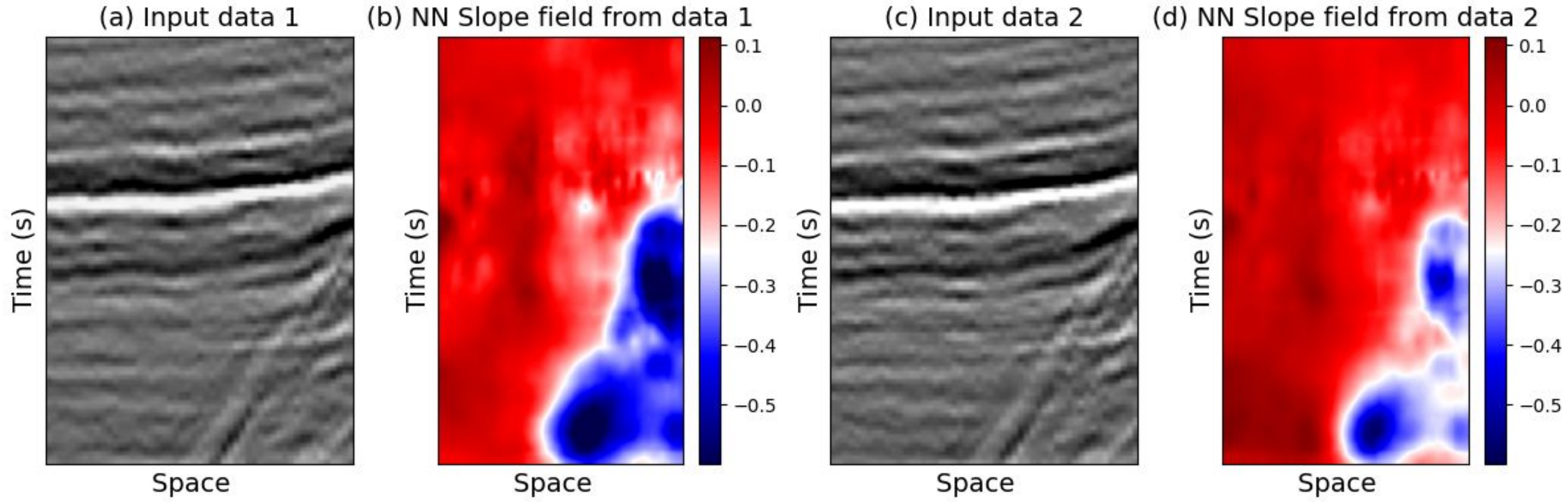
$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_0, \mathbf{y}_1, \hat{\theta})$$

$$\hat{\mathbf{x}} = \mathcal{N}(\mathbf{A}_1, \mathbf{y}_1, \hat{\theta})$$

Model drift

Model Adaptation: Perturb & Parametrize

$$J(\theta) = \sum_i \left(\sum_{ix} \sum_{it} \left(\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}^{(i)} \mathbf{U}_t^{(i)} \right)^2 + \lambda_x \| \mathbf{D}_x \hat{\mathbf{P}}^{(i)} \|_2^2 + \lambda_t \| \mathbf{D}_t \hat{\mathbf{P}}^{(i)} \|_2^2 \right)$$



NN on training data

NN on unseen data

Model Adaptation: Perturb & Parametrize

P0:

$$J(\theta) = \sum_i \left(\sum_{ix} \sum_{it} \left(\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}^{(i)} \mathbf{U}_t^{(i)} \right)^2 + \lambda_x \|\mathbf{D}_x \hat{\mathbf{P}}^{(i)}\|_2^2 + \lambda_t \|\mathbf{D}_t \hat{\mathbf{P}}^{(i)}\|_2^2 \right)$$

$$\hat{\mathbf{P}}_{\theta_0}^{(i)} = \mathcal{N}_{\theta_0}(\mathbf{U}_0^{(i)})$$

Model Adaptation: Perturb & Parametrize

P0:

$$J(\theta) = \sum_i \left(\sum_{ix} \sum_{it} \left(\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}^{(i)} \mathbf{U}_t^{(i)} \right)^2 + \lambda_x \|\mathbf{D}_x \hat{\mathbf{P}}^{(i)}\|_2^2 + \lambda_t \|\mathbf{D}_t \hat{\mathbf{P}}^{(i)}\|_2^2 \right)$$

$$\hat{\mathbf{P}}_{\theta_0}^{(i)} = \mathcal{N}_{\theta_0}(\mathbf{U}_0^{(i)})$$

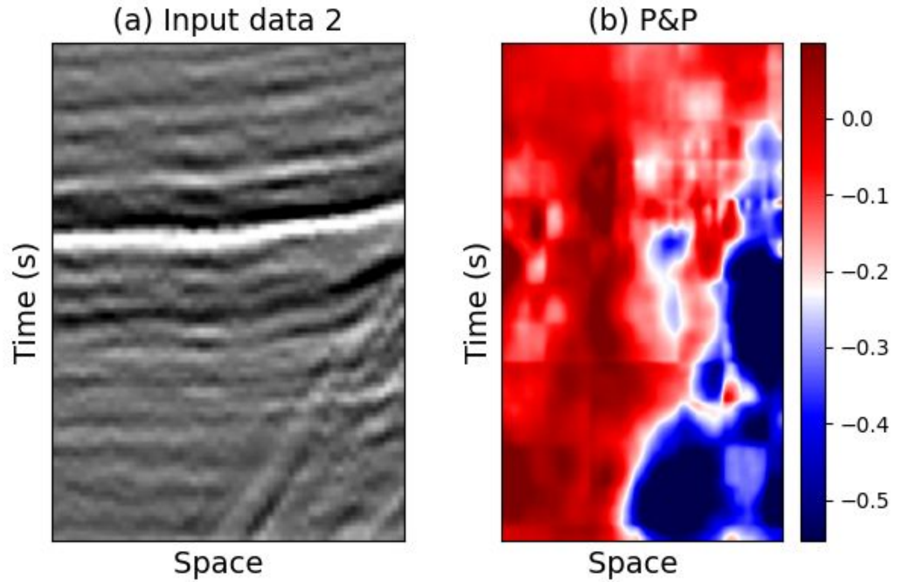
P1:

$$\hat{\mathbf{P}}_{\theta_1}^{(i)} = \mathcal{N}_{\theta_1}(\mathbf{U}_1^{(i)})$$

$$J(\theta) = \|\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}_{\theta}^{(i)} \mathbf{U}_t^{(i)}\|_2^2 + \mu \|\theta - \theta_0\|_2^2$$

Model Adaptation: Perturb & Parametrize

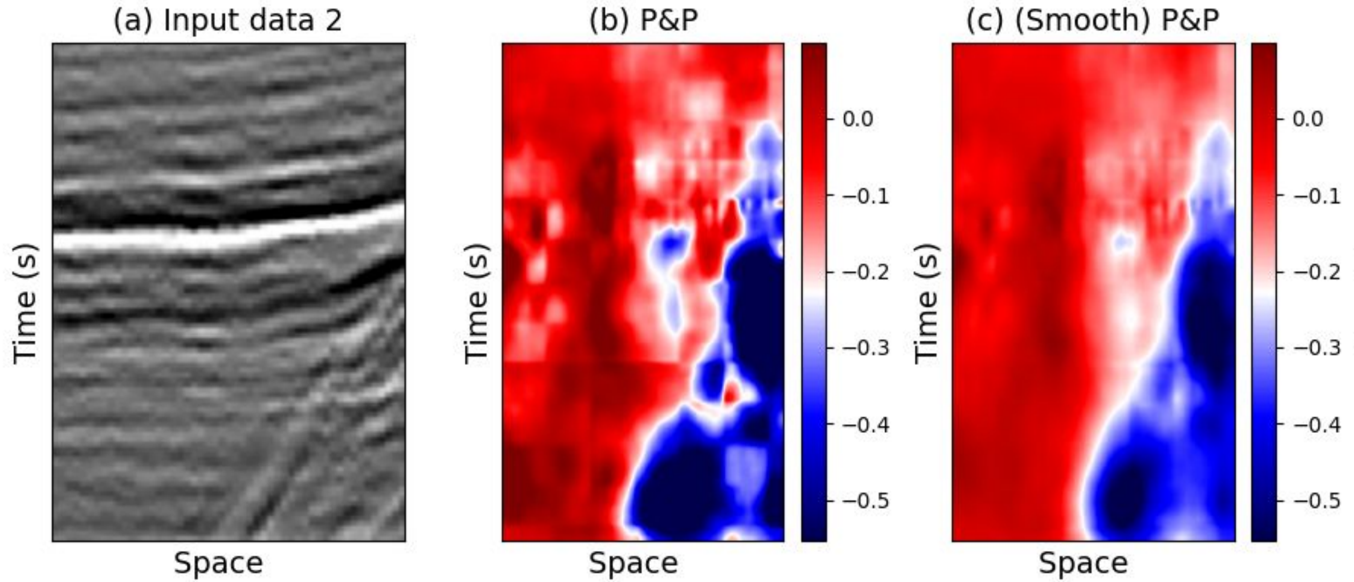
$$J(\theta) = \|\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}_\theta^{(i)}\mathbf{U}_t^{(i)}\|_2^2 + \mu\|\theta - \theta_0\|_2^2$$



100 epochs
lr=0.0001

Model Adaptation: Perturb & Parametrize

$$J(\theta) = \|\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}_\theta^{(i)}\mathbf{U}_t^{(i)}\|_2^2 + \mu\|\theta - \theta_0\|_2^2 + \lambda_x\|\mathbf{D}_x\mathbf{P}_\theta^{(i)}\|_2^2 + \lambda_t\|\mathbf{D}_t\mathbf{P}_\theta^{(i)}\|_2^2$$

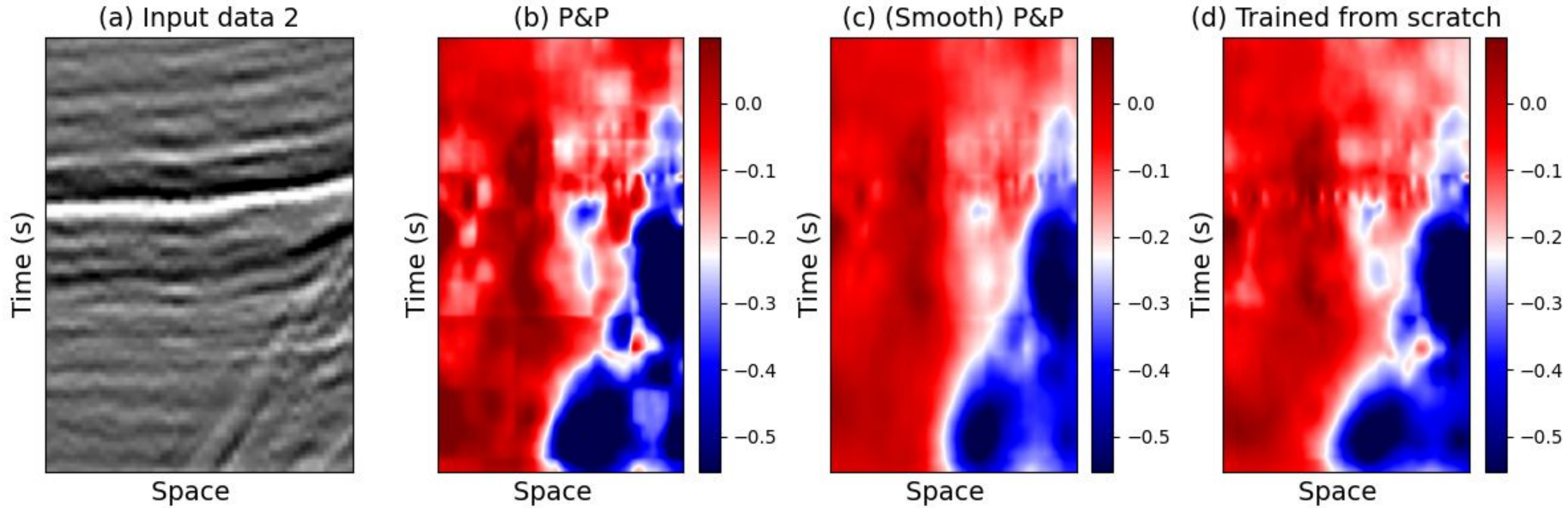


100 epochs
lr=0.0001

500 epochs
lr=0.0001

Model Adaptation: Perturb & Parametrize

$$J(\theta) = \|\mathbf{U}_x^{(i)} + \hat{\mathbf{P}}_\theta^{(i)} \mathbf{U}_t^{(i)}\|_2^2 + \lambda_x \|\mathbf{D}_x \mathbf{P}_\theta^{(i)}\|_2^2 + \lambda_t \|\mathbf{D}_t \mathbf{P}_\theta^{(i)}\|_2^2$$



100 epochs
lr=0.0001

500 epochs
lr=0.0001

500 epochs
lr=0.01

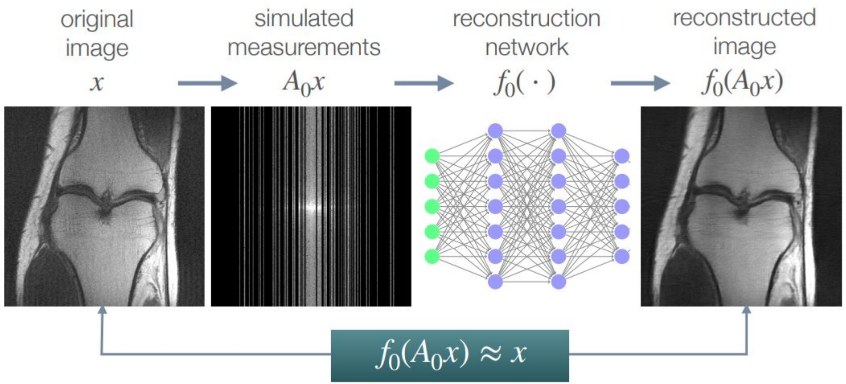
Model Adaptation: Reuse & Regularize

- P&P
  - Train for P0 and retrain for P1 (transfer learning)
  - Relies (too much) on the initial network for P0

Model Adaptation: Reuse & Regularize

- P&P
  - Train for P0 and retrain for P1 (transfer learning)
  - Relies (too much) on the initial network for P0
- R&R
  - Train for P0 but does not retrain for P1
  - The composition of the forward model and trained networks should act as an auto-encoder

$$\hat{\mathbf{x}} \approx g(\mathbf{x}) = \mathcal{N}(\mathbf{A}_0\mathbf{x})$$



Model Adaptation: Reuse & Regularize

- R&R
  - Auto-encoder intuition $\hat{\mathbf{x}} \approx g(\mathbf{x}) = \mathcal{N}(\mathbf{A}_0\mathbf{x})$
  - Use g(x) as a denoiser in **RED** as to propose a regularized model-based problem

$$\hat{\mathbf{x}} = \min_{\mathbf{x}} \|\mathbf{y} - \mathbf{A}_1\mathbf{x}\|_2^2 + \lambda \mathbf{x}^T(\mathbf{x} - g(\mathbf{x}))$$

$$\mathbf{x}_0 = \mathbf{A}_1^{\dagger}\mathbf{y}$$

for $k = 1, 2, \ldots, K$

$$\mathbf{z}_k = \mathcal{N}(\mathbf{A}_0\mathbf{x}_{k-1})$$

$$\mathbf{x}_k = (\mathbf{A}_1^T\mathbf{A}_1 + \lambda\mathbf{I})^{-1}(\mathbf{A}_1^T\mathbf{y} + \lambda\mathbf{z}_k)$$

Optimization & Operators

Optimization & Operators

- Proximal algorithms
    - Projected Gradient Descent
    - (Robust) Iterative Hard Thresholding
    - (Fast) Iterative Soft Thresholding
- RED
    - Gradient descent
    - Fixed-point iterations
    - ADMM

- Main inputs
  - Pairs of forward and adjoint operators
    - Matrix-free
      - Easy connection with Jets.jl

SIGNAL ANALYSIS & IMAGING GROUP

Optimization & Operators

- Proximal algorithms
    - Projected Gradient Descent
    - (Robust) Iterative Hard Thresholding
    - (Fast) Iterative Soft Thresholding
- ***RED***
    - ***Gradient descent***
    - ***Fixed-point iterations***
    - ***ADMM***

- Main inputs
    - Pairs of forward and adjoint operators
        - Matrix-free
            - Easy connection with Jets.jl

Deblend

<STATUS> <UNDER REVIEW> twitter DOI 10.1007/978-3-319-76207-4_15

This repository contains the workflow adopted to read, blend and deblend two different datasets: The Mississippi Canyon data & Valhall data.

This project is based on tools for reading, writing and processing seismic data offered by the SeismicJulia project such as SeisPlot, SeisProcessing and SeisAcoustic. These packages are tested and updated based on Julia 1.0.

If you use the SeismicJulia project, please cite the following paper

```
@article{stanton2016efficient,
  title={Efficient geophysical research in Julia},
  author={Stanton, Aaron and Sacchi, Mauricio D},
  journal={CSEG GeoConvention 2016},
  pages={1--3},
  year={2016}
}
```

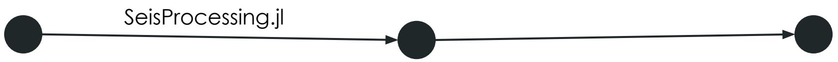
Basic usage

Not public yet.

- Tomography: Vargas (2020)
- Deblending: Bahia et al (2021)
- FWI: Anagaw and Sacchi (2022)

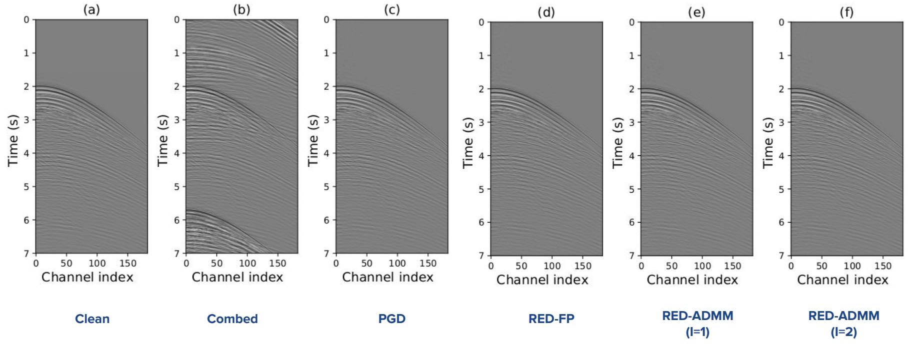
Processing

SeisProcessing.jl



- Revamp SeisProcessing.jl

Centralized package for seismic data processing

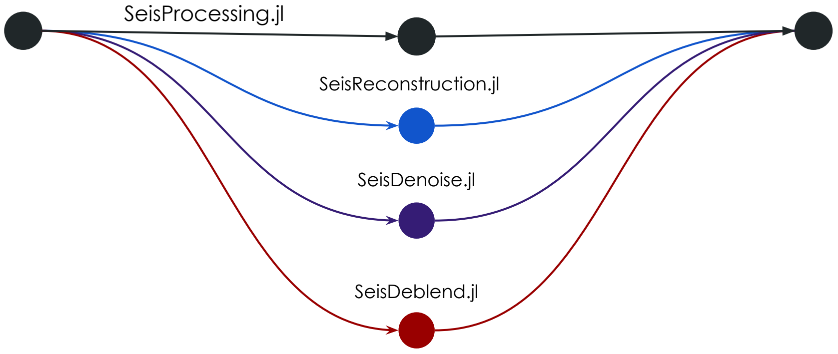


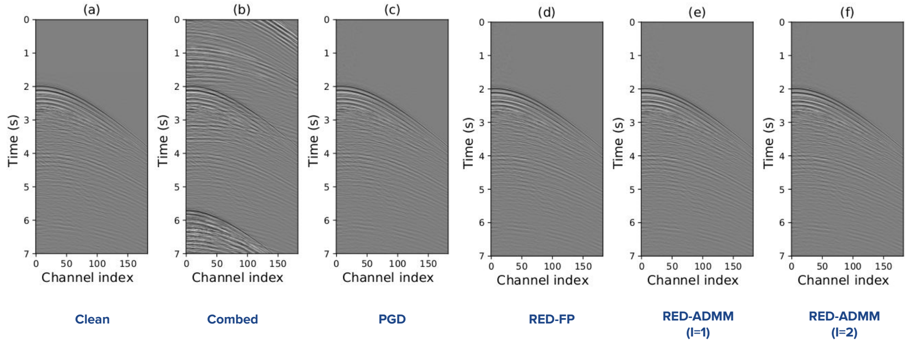
SeisProcessing.jl

- Revamp SeisProcessing.jl

  Centralized package for seismic data processing

  - SeisReconstruction.jl (Fernanda)
  - SeisDenoise.jl (Wenlei)
  - SeisDeblend.jl (Breno, Rongzhi, Ji Li)
  - Should contain its own optimization routines
    - CG, FISTA, RED, PGD, etc…





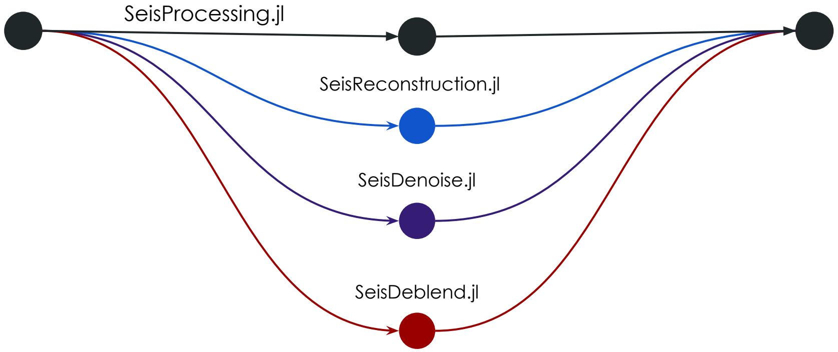
SeisProcessing.jl

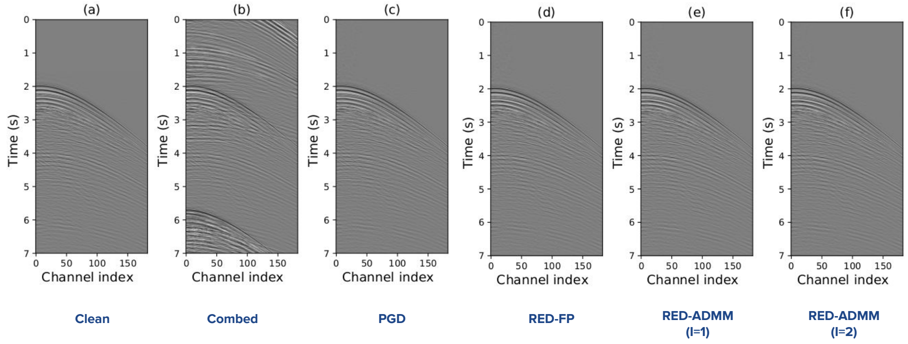
- Revamp SeisProcessing.jl

  Centralized package for seismic data processing

  - SeisReconstruction.jl (Fernanda)
  - SeisDenoise.jl (Wenlei)
  - SeisDeblend.jl (Breno, Rongzhi, Ji Li)
  - Should contain its own optimization routines
    - CG, FISTA, RED, PGD, etc…

  - Next steps:
    - Standardize code base
      - (Calls to) Forward and Adjoint operators
      - Type annotation and stability
    - Documentation guidelines





Imaging

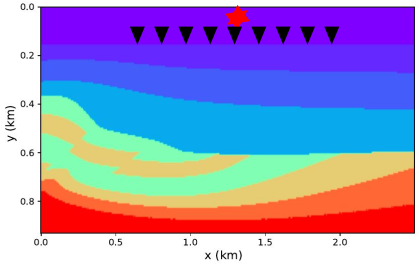
SeisAcoustic.jl

- Revamp SeisAcoustic.jl (Wenlei)

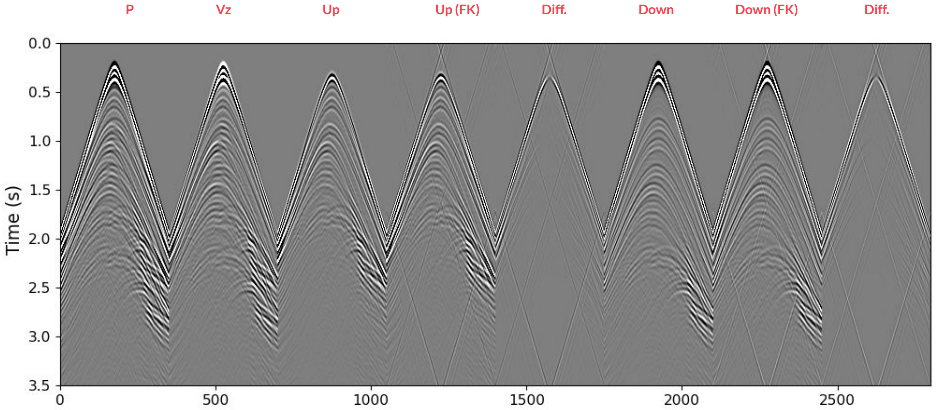
  Package for acoustic modeling and imaging

  - Should contain its own optimization routines
    - SD, CG, ML, etc …
  - Documentation
    - Rafael Manenti
    - Joaquin Acedo
    - Breno Bahia

Git & GitHub training



$$\begin{bmatrix} \mathbf{U} \\ \mathbf{D} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & -\frac{\rho\omega}{k_z} \\ 1 & \frac{\rho\omega}{k_z} \end{bmatrix} \begin{bmatrix} \mathbf{P} \\ \mathbf{V}_z \end{bmatrix}$$



Conclusions

- Seis packages
  - Centralizing SeisProcessing.jl
    - ✓ SeisReconstruction.jl
    - ✓ SeisDenoise.jl
    - ✓ SeisDeblend.jl
      - Regularization by denoising
      - Projected Gradient Descent
  - Documenting SeisAcoustic.jl
  - SeisLearn.jl? SeisML.jl?
    - ✓ Model adaptation
      - Perturb & Parametrize
        - Retraining (Transfer learning)
      - Reuse & Regularize
        - No retraining (**RED**)

SIGNAL
ANALYSIS &
IMAGING GROUP