How incoherent can we be? Phase encoding with random boundaries

Chris Leader

SEP147 - p149

Tuesday, May 22nd

Motivation / research goals

Create a flexible, robust linearised inversion scheme that minimises I/O and favours computation

- Phase encoding
  - Inversion needed to remove crosstalk artifacts
- Random boundaries
  - Sufficient shots / iterations needed to stack out artifacts

Motivation / research goals

Create a flexible, robust linearised inversion scheme that minimises I/O and favours computation

- Phase encoding
  - Inversion needed to remove crosstalk artifacts
- Random boundaries
  - Sufficient shots / iterations needed to stack out artifacts

Can these schemes be effectively augmented? How is convergence changed? Shot sampling vs iteration count?

Table of contents

Linearised inversion

Inverting for the Born scattering potential

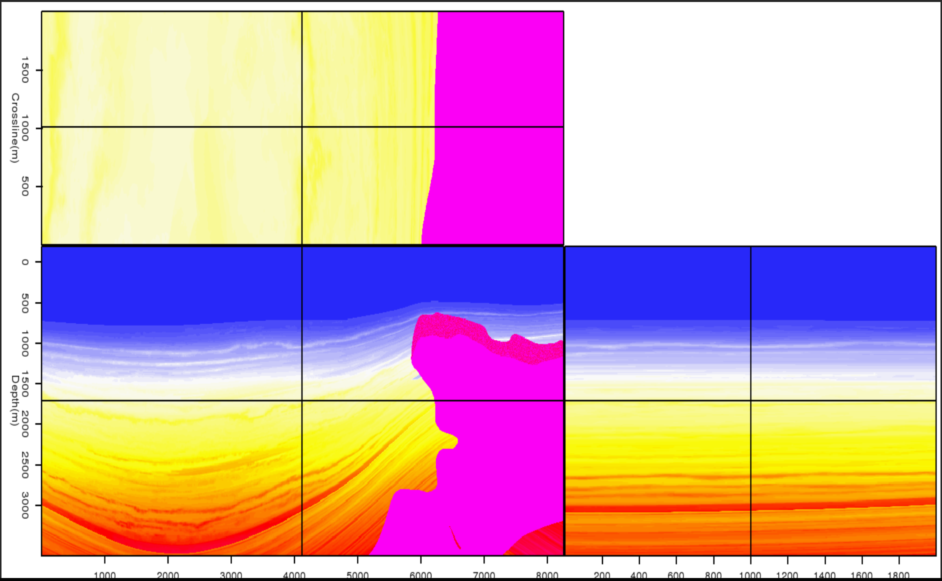
Assume we know the background velocity (kinematic model)

- $s^2(z,x,y) = b(z,x,y) + m(z,x,y)$

Two-way wave solution

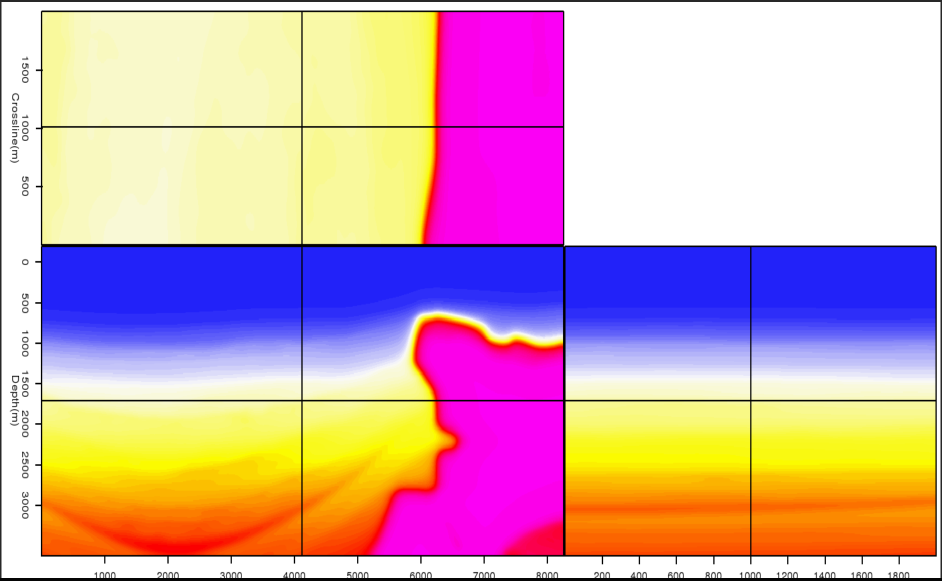
- Reverse Time Migration (RTM)
- Frist order Born scattering approximation
- Linearised inverion / Least Squares Reverse Time Migration (LSRTM)

Full model

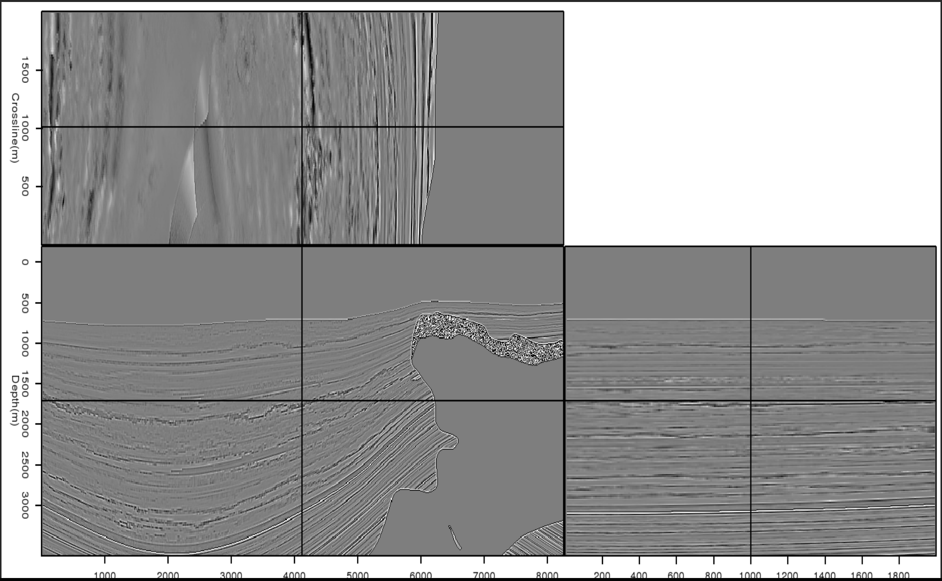


1500
1000
500
Crossline(m)
0
500
1000
1500
2000
2500
3000
Depth(m)
1000
2000
3000
4000
5000
6000
7000
8000
200
400
600
800
1000
1200
1400
1600
1800

Background (kinematic model)



'Reflectivity' (perturbation)



LI: Conventional algorithm

Using absorbing boundaries

1. Initialise
2. Forward model and save 4D source wavefields
3. $r = Fm - d_{obs}$
4. Iterate
   - $gg = F' r$
   - $rr = Fgg$
   - $(m, r) = \text{linear\_stepper}\ (m, r, gg, rr)$
5. Output m

LI: Conventional algorithm

Using absorbing boundaries

1. Initialise
2. Forward model and save 4D source wavefields
3. $r = Fm - d_{obs}$
4. Iterate
   - $gg = F'r$
   - $rr = Fgg$
   - $(m, r) = \text{linear\_stepper}\,(m, r, gg, rr)$
5. Output m

Let's look closer at $gg = F'r$

The time reversal problem

Our forward process:

$rr = Fgg$

$$d(\mathbf{x}_r, \mathbf{x}_s, \omega) = \sum_{\mathbf{x},\omega} f(\omega) G_0(\mathbf{x}, \mathbf{x}_s, \omega) m(\mathbf{x}) \sum_{\mathbf{x}} G_0(\mathbf{x}, \mathbf{x}_r, \omega)$$

Our adjoint process:

$gg = F'r$

$$m(\mathbf{x}) = \sum_{\mathbf{x}_s,\omega} f(\omega) G_0(\mathbf{x}, \mathbf{x}_s, \omega) \sum_{\mathbf{x}_r} G_0(\mathbf{x}, \mathbf{x}_r, \omega)\ d^*(\mathbf{x}_r, \mathbf{x}_s, \omega)$$

The time reversal problem

Our forward process:

$$rr = Fgg$$

$$d(\mathbf{x}_r, \mathbf{x}_s, \omega) = \sum_{\mathbf{x},\omega} f(\omega) G_0(\mathbf{x}, \mathbf{x}_s, \omega) m(\mathbf{x}) \sum_{\mathbf{x}} G_0(\mathbf{x}, \mathbf{x}_r, \omega)$$

Our adjoint process:

$$gg = F'r$$

$$m(\mathbf{x}) = \sum_{\mathbf{x}_s,\omega} f(\omega) G_0(\mathbf{x}, \mathbf{x}_s, \omega) \sum_{\mathbf{x}_r} G_0(\mathbf{x}, \mathbf{x}_r, \omega)\ d^*(\mathbf{x}_r, \mathbf{x}_s, \omega)$$

Opposite sense of time to source

What does this mean?

Practically, RTM needs two processes:

- Forward propagate the source wavefield
  - Save wavefield to disk (z,x,y,t)
- Back propagate the recevier wavefield
  - At imaging time step?
    - Read the source relevant source wavefield snapshot
    - Multiply source and receiver wavefields
    - Sum result to image

What does this mean?

Practically, RTM needs two processes:

- Forward propagate the source wavefield
  - Save wavefield to disk (z,x,y,t)
- Back propagate the recevier wavefield
  - At imaging time step?
    - Read the source relevant source wavefield snapshot
    - Multiply source and receiver wavefields
    - Sum result to image

I/O bottleneck

Table of contents

Random boundaries

Remove IO from propagation

- Make source wavefield time reversible
- We propagate an extra wavefield, but no disk access needed during the RTM time loop

However:

- Ensure boundaries are set up correctly
- Sufficient fold / iterations needed to stack out residual artifacts

Random boundaries

Remove IO from propagation

- Make source wavefield time reversible
- We propagate an extra wavefield, but no disk access needed during the RTM time loop

However:

- Ensure boundaries are set up correctly
- Sufficient fold / iterations needed to stack out residual artifacts

We can extend this to changing our boundaries between iterations

Static random boundaries

1. Initialise
2. Construct random boundaries
   - Calculate final wavefield snapshots
3. $r = Fm - d_{obs}$
4. Iterate
   - $gg = F'r$
   - $rr = Fgg$
   - $(m, r) = \text{linear-stepper}\ (m, r, gg, rr)$
5. Output m

Dynamic random boundaries

1. Initialise
2. $r = Fm - d_{obs}$
3. Iterate
   - Construct random boundaries
     - Calculate final wavefield snapshots
   - $gg = F'r$
   - $rr = Fgg$
   - $(m, r) =$ non-linear-stepper (m, r, gg, rr)
4. Output m

When is this useful?

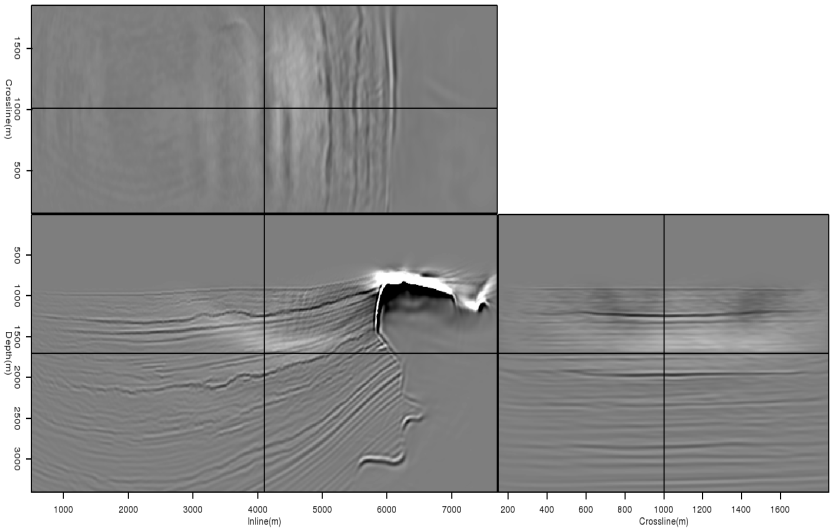
Dynamic random boundaries require more computation, typically 12% longer

- Theoretically, a non-linear solver should be used
- Similar results seen with linear solver, however

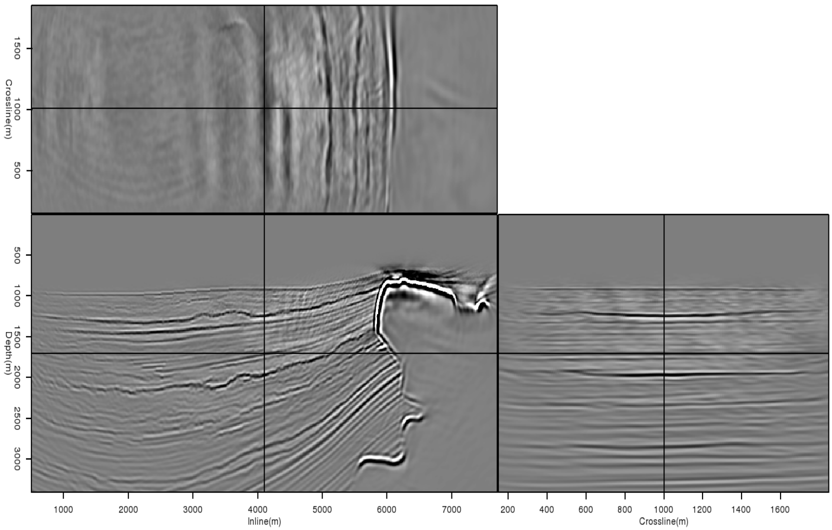
Advantage seen in areas of poor shot sampling

- Can also vary boundary depth
- Artifacts still seem to stack out at around $\sqrt{n}$

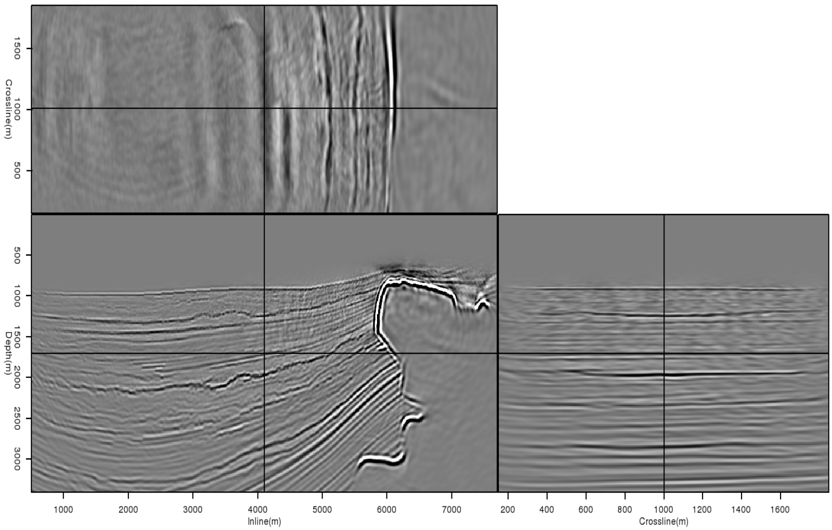
LI: Iteration 1



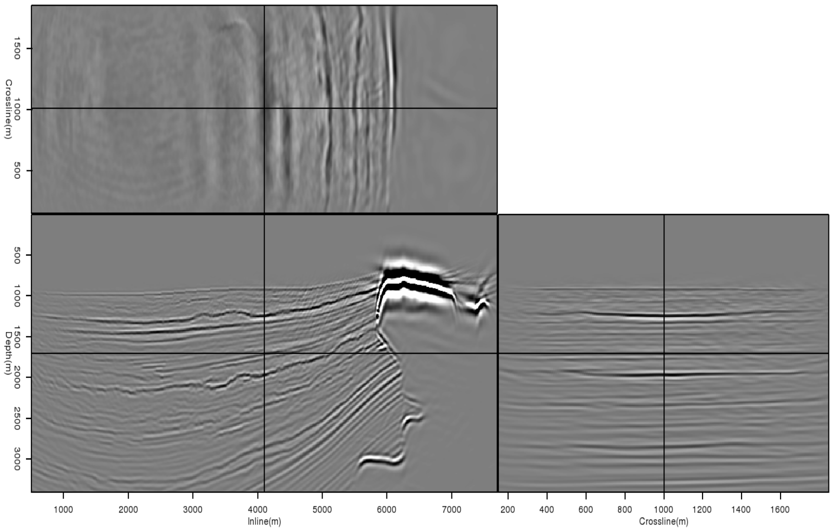
LI: Iteration 5



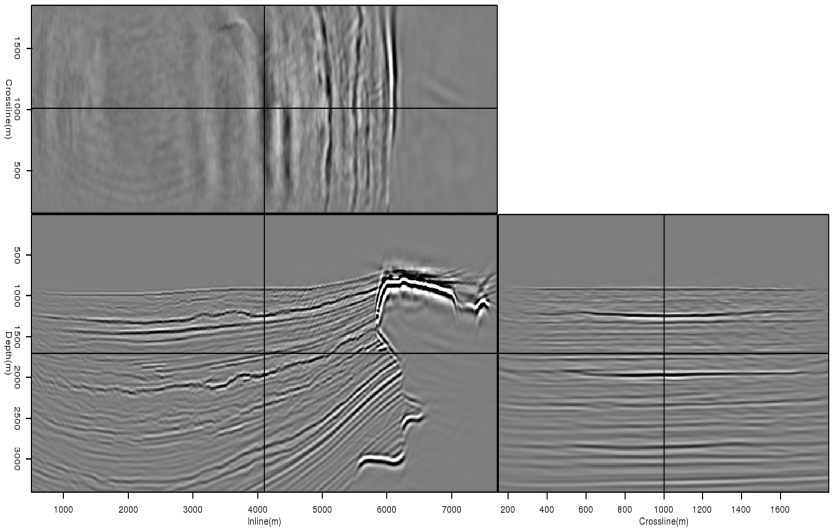
LI: Iteration 10



LI: Iteration 1, cut low wavenumbers



LI: Iteration 5, cut low wavenumbers



LI: Iteration 10, cut low wavenumbers

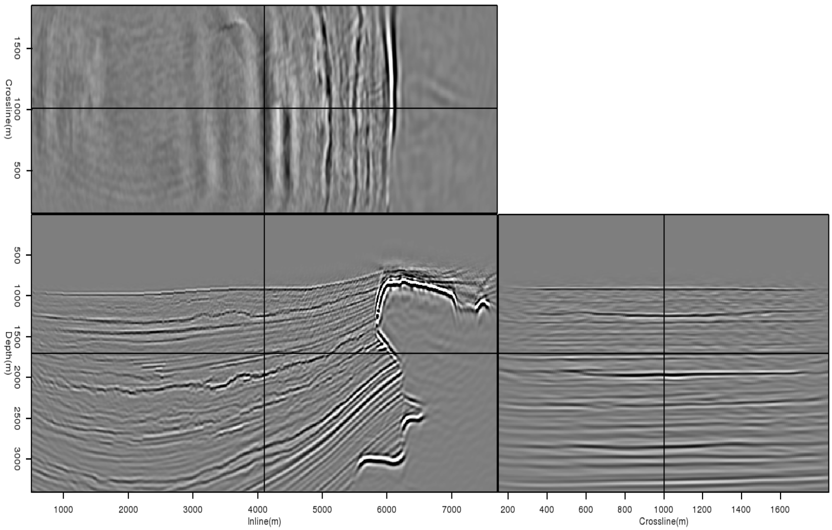


Table of contents

Phase encoding

Aim is to reduce the quantity of data we are migrating and modelling

Weight, shift and sum shots together

- Create one, or a series of, super-shot(s)
- Extra computation needed to attenuate crosstalk
  - Balance of data-size vs computation
- Cost can approach independence from the number of sources

$$\widetilde{d}(\mathbf{x}_r, p_s, \omega) = \sum_{\mathbf{x}_s} \alpha(\mathbf{x}_s, p_s) d(\mathbf{x}_r, \mathbf{x}_s, \omega)$$

$$\widetilde{f}(\mathbf{x}_r, p_s, \omega) = \sum_{\mathbf{x}_s} \alpha(\mathbf{x}_s, p_s) f(\omega)$$

Encoding function α

The consensus has been that randomly selecting $+1$ or -1 gives the best convergence properties (Romerero et al., 2000; Krebs et al., 2009)

However:

- Changing α inherently changes our oberserved data, d_{obs}
- The first step of each iteration recalculates the 'initial' residual
    - One more forward process per iteration
    - Cost increase by (roughly) 1.5x

PELI: Conventional

1. Initialise
2. Iterate
   - Create α
   - $d = \alpha d_{obs}$
   - $r = Fm - d$
     - Create and save 4D source wavefields
   - $gg = F'r$
   - $rr = Fgg$
   - $(m, r) = \text{non-linear-stepper}\,(m, r, gg, rr)$
3. Output m

PELI: Cost considerations

Separated linearised inversion:

- About 2x the operator cost per iteration
- Use a conjugate direction solver

Phase encoded linearised inversion

- About 3x the operator cost per iteration
- Use a non-linear solver

PELI: Cost considerations

Separated linearised inversion:

- About 2x the operator cost per iteration
- Use a conjugate direction solver

Phase encoded linearised inversion

- About 3x the operator cost per iteration
- Use a non-linear solver

How does this extend to random boundaries?

PELI: Conventional

1. Initialise
2. Iterate
   - Create α
   - $d = \alpha d_{obs}$
   - $r = Fm - d$
     - Create final source wavefield snapshots
   - $gg = F'r$
   - $rr = Fgg$
   - $(m, r) = \text{non-linear-stepper}\ (m, r, gg, rr)$
3. Output m

PELI with random boundaries

Algorithm extension is obvious

We get dynamic random boundaries for free

Both techniques rely on certain wavefields being more coherent than others

- Does their combination violate any of their individual assumptions?
- Would this slow convergence significantly?

We find similar convergence characteristics, but with an asymptote towards greater misfit error

PELI with random boundaries

Let us propagate 100 combined shots through the same random boundary

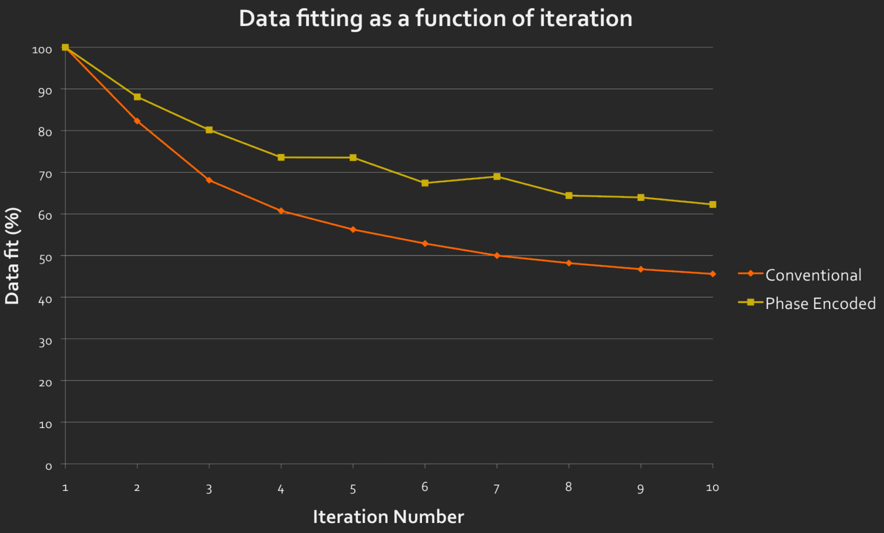
- Different incident angle $\implies$ different scattering

We will have correlation between:

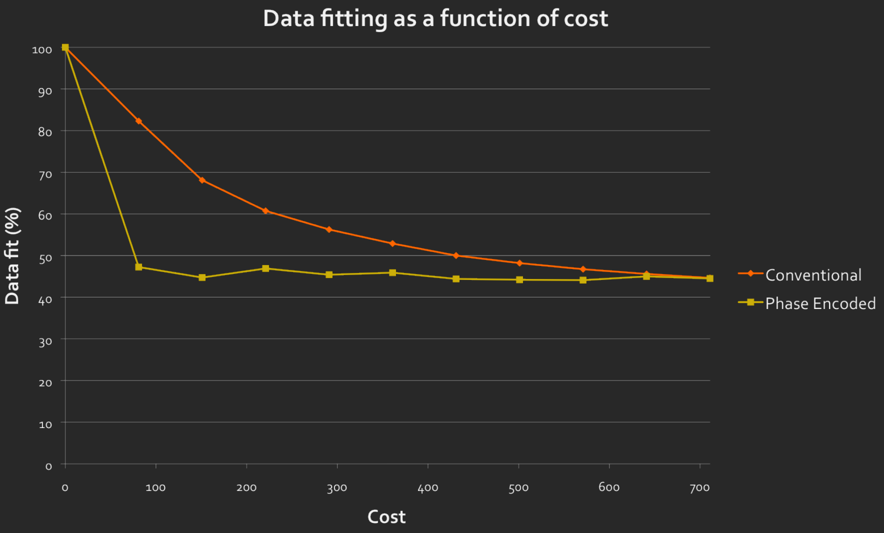
- Scattered fields with scattered fields
- Scattered fields with coherent fields
- Coherent fields with non-matching coherent fields
- Coherent fields with matching coherent fields

We have $\approx$ twice the noise of each method independently

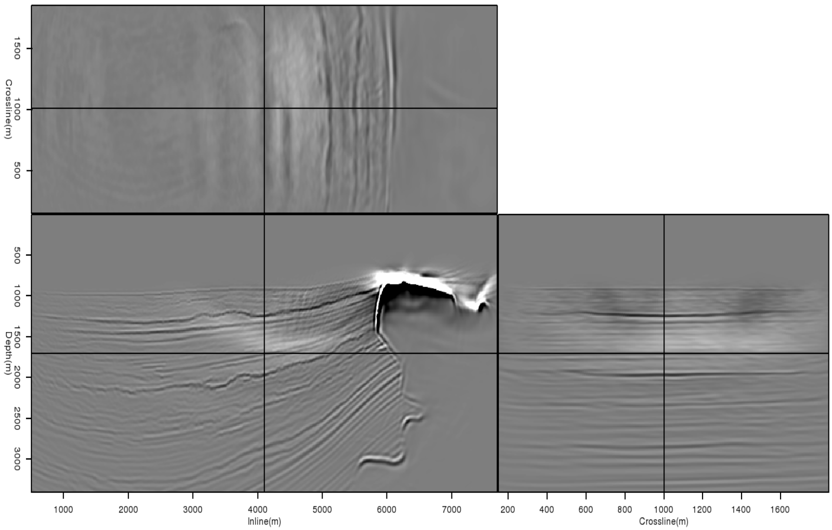
Convergence with iterations



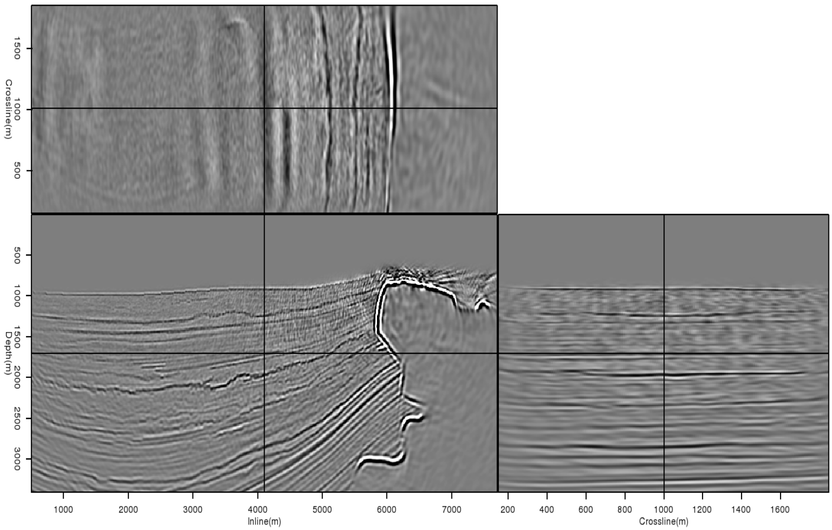
Convergence with cost



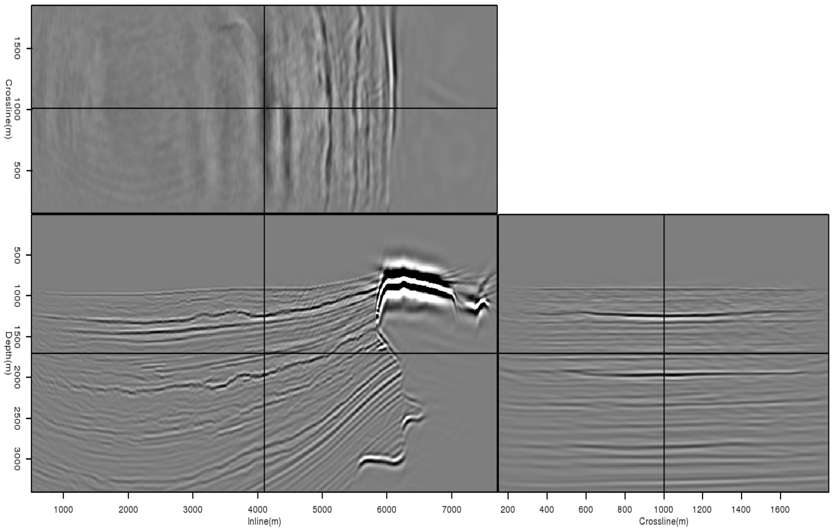
LI: Iteration 1



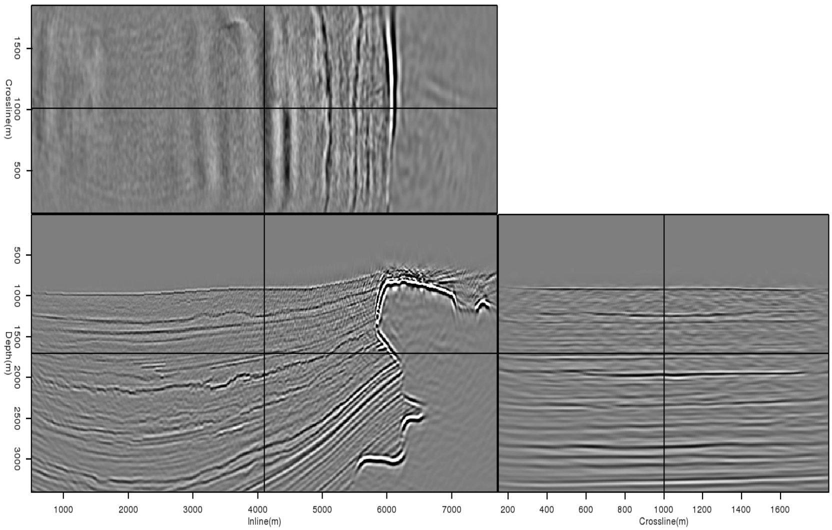
PELI: Equivalent cost



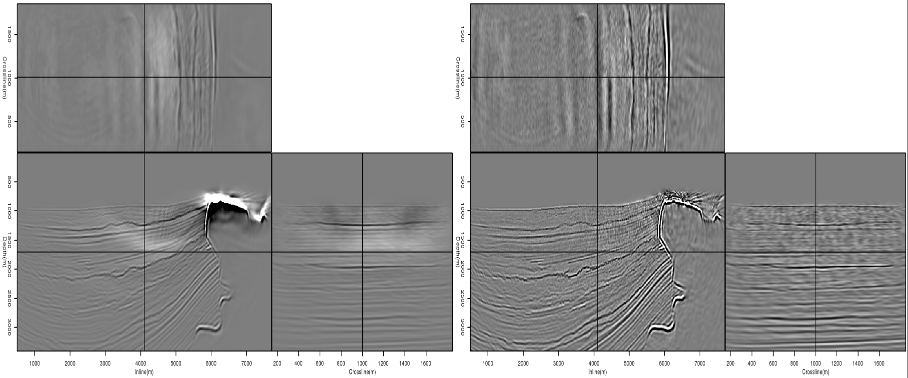
LI: Iteration 1, with filter



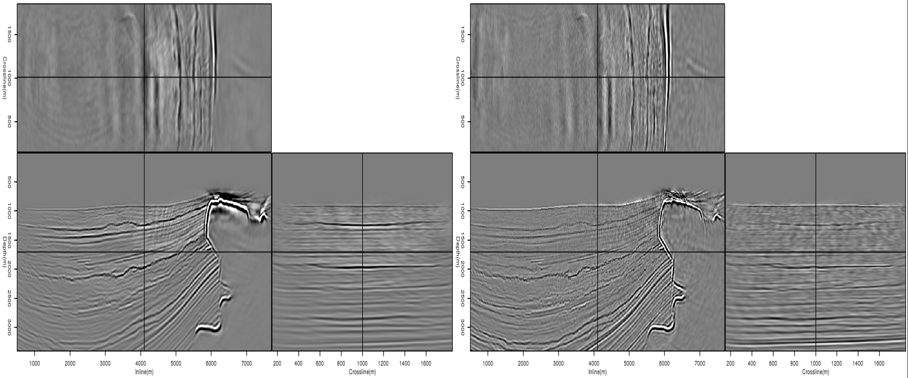
PELI: Equivalent cost, with filter



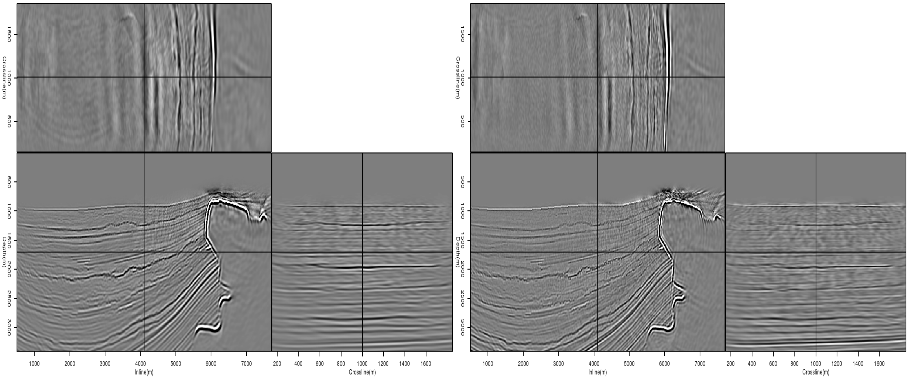
Equivalent cost comparison, raw: 1



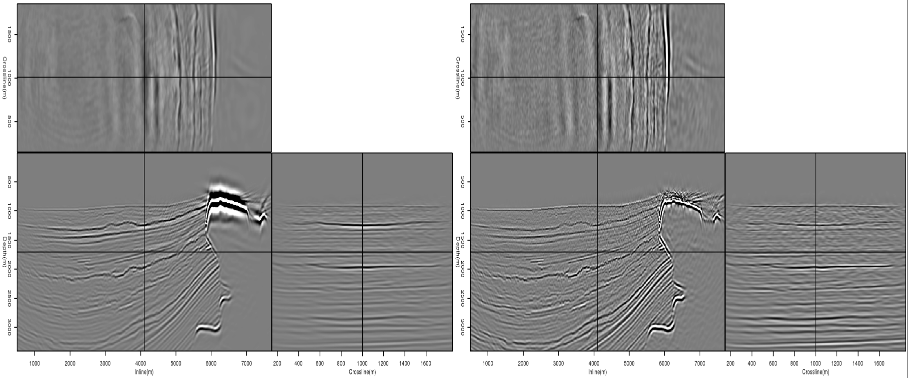
Equivalent cost comparison, raw: 5



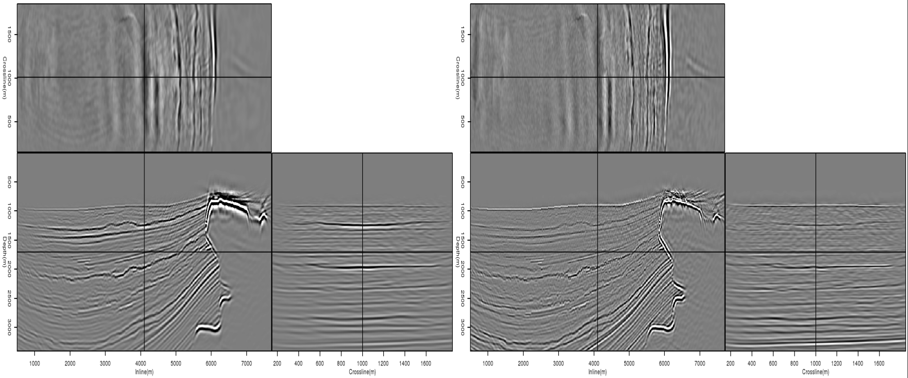
Equivalent cost comparison, raw: 10



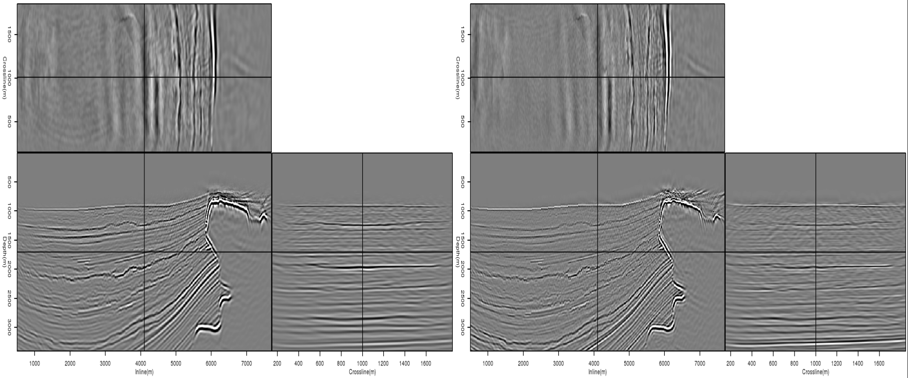
Equivalent cost comparison, filtered: 1



Equivalent cost comparison, filtered: 5



Equivalent cost comparison, filtered: 10



Cleaning up our gradients

Using one supershot, we have 7 other GPUs sitting on our node

We can use these to perform multiple realisations per iteration

- Use 8 different encoding schemes
- Stack and normalise the gradient
- We see slight convergence improvement
- Also, slight data-fit improvement

Note: same residual

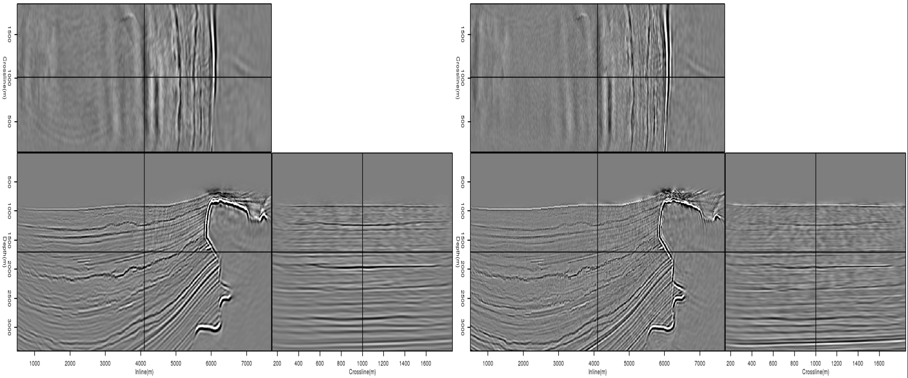


Table of contents

Conclusions

Inversion improves images created with random boundaries

Phase encoding and random boundaries can be combined

As a function of iteration number, we see better convergence with separated inversion (as expected)

As a function of cost, using an ℓ_2 norm, we see a significant benefit for phase encoding

Using multiple realisations per iteration, we can slightly improve convergence properties

We have created a 3D inversion scheme that requires minimal IO

Acnkowledgments

Ali Almomin, for help with image error interpretation

SEP sponsors