Sparse inversion **simplified**

Ning Tu, Navid Ghadermarzy, Rajiv Kumar

SLIM

University of British Columbia

Motivation

- Many techniques developed in this group use our in-house *sophisticated* and *complex* l1 solver -- SPGl1.

- Implementation of SPGl1 in low-level languages, e.g., C or Fortran, can be a daunting task to many *geoscientists*.

SLIM

Solution

Recent developments in Compressive Sensing (CS) using Linearized Bregman Projection (LBP).

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \lambda\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{x}\|_2 \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned}$$

For λ large enough, it converges to the solution of the Basis Pursuit (BP) problem.

Benefits

- **Easy implementation** (details in examples)
- Framework provided also for matrix completion

Preliminary applications

- Fast compressive imaging (by Ning Tu)
- Seismic data interpolation (by Rajiv Kumar)

SLIM

Fast compressive imaging using LBP

leveraging the sparse randomized block-Kaczmarz solver:

obtain submatrix $\underline{\mathbf{A}}^k$ and data $\underline{\mathbf{b}}^k$ at the k^{th} iteration

compute residual of the previous step $\mathbf{r}^k = \underline{\mathbf{A}}^k \mathbf{x}^k - \underline{\mathbf{b}}$

compute gradient $\mathbf{g}^k = \underline{\mathbf{A}}^{k'} \mathbf{r}^k$

compute steplength $t^k = \dfrac{\|\mathbf{r}^k\|_2^2}{\|\mathbf{g}^k\|_2^2}$

gradient descent $\mathbf{z}^{k+1} = \mathbf{z}^k - t^k \mathbf{g}^k$

soft thresholding $\mathbf{x}^{k+1} = \mathcal{S}_\lambda(\mathbf{z}^{k+1})$

LBP vs. SPGl1: computer codes

SLIM

LBP vs. SPGl1: computer codes

Start your stopwatch!

SLIM

LBP vs. SPGl1: computer codes

Start your stopwatch!

```
r = b_sub - A_sub*x;
g = A_sub'*r;
rnorm = norm(r,2);
gnorm = norm(g,2);
sl = (rnorm/gnorm)^2;
z = z+sl*g;
x = sign(x).*max(0,abs(x)-lambda);
```

LBP

E. van den Berg and M. Friedlander, 2008

LBP vs. SPGl1: computer codes

Start your stopwatch!

```
r = b_sub - A_sub*x;
g = A_sub'*r;
rnorm = norm(r,2);
gnorm = norm(g,2);
sl = (rnorm/gnorm)^2;
z = z+sl*g;
x = sign(x).*max(0,abs(x)-lambda);
```

LBP

```
function [x,r,g,info] = spgl1( A, b, tau, sigma, x, options )

m = length(b);

%----------------------------------------------------------------------
% Check arguments.
%----------------------------------------------------------------------
if ~exist('options','var'), options = []; end
if ~exist('x','var'), x = []; end
if ~exist('sigma','var'), sigma = []; end
if ~exist('tau','var'), tau = []; end

if nargin < 2 || isempty(b) || isempty(A)
   error('At least two arguments are required');
elseif isempty(tau) && isempty(sigma)
   tau = 0;
   sigma = 0;
   singleTau = false;
elseif isempty(sigma) % && ~isempty(tau)  <-- implied
   singleTau = true;
else
   if isempty(tau)
      tau = 0;
   end
   singleTau = false;
end

%----------------------------------------------------------------------
% Grab input options and set defaults where needed.
%----------------------------------------------------------------------
defaultopts = spgSetParms(...
'fid'        ,      1 , ... % File ID for output
'verbosity'  ,      2 , ... % Verbosity level
'iterations' ,   10*m , ... % Max number of iterations
'nPrevVals'  ,      3 , ... % Number previous func values for linesearch
'bpTol'      ,  1e-06 , ... % Tolerance for basis pursuit solution
'optTol'     ,  1e-04 , ... % Optimality tolerance
'decTol'     ,  1e-04 , ... % Req'd rel. change in primal obj. for Newton
'stepMin'    ,  1e-16 , ... % Minimum spectral step
'stepMax'    ,  1e+05 , ... % Maximum spectral step
'rootMethod' ,      2 , ... % Root finding method: 2=quad,1=linear (not used).
'activeSetIt',    Inf , ... % Exit with EXIT_ACTIVE_SET if nnz same for # its.
'subspaceMin',      0 , ... % Use subspace minimization
'iscomplex'  ,    NaN , ... % Flag set to indicate complex problem
'maxMatvec'  ,    Inf , ... % Maximum matrix-vector multiplies allowed
'weights'    ,      1 , ... % Weights W in ||Wx||_1
'Kaczmarz'   ,      0 , ... % Toggles whether Kaczmarz mode is on (experimental)
'KaczScale'  ,      1 , ... % Scaling factor for Tau when using Kaczmarz-type submatrices
'quitPareto' ,      0 , ... % Exits when pareto curve is reached
'minPareto'  ,      3 , ... % If quitPareto is on, the minimum number of iterations before checking for quitPareto conditions
'lineSrchIt' ,      1 , ... % Maximum number of line search iterations for spgLineCurvy, originally 10 ...
'feasSrchIt' ,  10000 , ... % Maximum number of feasible direction line search iteraitons, originally 10 ...
'ignorePErr' ,      0 , ... % Ignores projections error by issuing a warning instead of an error ...
'project'    , @NormL1_project , ...
'primal_norm', @NormL1_primal  , ...
'dual_norm'  , @NormL1_dual      ...
   );
options = spgSetParms(defaultopts, options);

fid           = options.fid;
logLevel      = options.verbosity;
maxIts        = options.iterations;
nPrevVals     = options.nPrevVals;
bpTol         = options.bpTol;
optTol        = options.optTol;
decTol        = options.decTol;
stepMin       = options.stepMin;
stepMax       = options.stepMax;
activeSetIt   = options.activeSetIt;
subspaceMin   = options.subspaceMin;
maxMatvec     = max(3,options.maxMatvec);
weights       = options.weights;
quitPareto    = options.quitPareto;
minPareto     = options.minPareto;
Kaczmarz      = options.Kaczmarz;
lineSrchIt    = options.lineSrchIt;
feasSrchIt    = options.feasSrchIt;
ignorePErr    = options.ignorePErr;

% maxLineErrors TEMPORARILY DISABLED to prevent very lar[illegible]g issue[illegible]ff a[illegible]orithm
maxLineErrors = Inf;     % Maximum number of line-search failures.
pivTol        = 1e-12;  % Threshold for significant Newton step.

%----------------------------------------------------------------------
% Initialize local variables.
%----------------------------------------------------------------------
iter          = 0;  itnTotLSQR = 0; % Total SPGL1 and LSQR iterations.
nProdA        = 0;  nProdAt    = 0;
lastFv        = -inf(nPrevVals,1);  % Last m function values.
nLineTot      = 0;                  % Total no. of linesearch steps.
printTau      = false;
nNewton       = 0;
bNorm         = norm(b,2);
stat          = false;
timeProject   = 0;
timeMatProd   = 0;
nnzIter       = 0;                  % No. of its with fixed pattern.
nnzIdx        = [];                 % Active-set indicator.
subspace      = false;              % Flag if did subspace min in current itn.
stepG         = 1;                  % Step length for projected gradient.
testUpdateTau = 0;                  % Previous step did not update tau

% Determine initial x, vector length n, and see if problem is complex
explicit = ~(isa(A,'function_handle'));
```

SPGl1

E. van den Berg and M. Friedlander, 2008

SLIM

LBP vs. SPGl1: computer codes

Start your stopwatch!

```
r = b_sub - A_sub*x;
g = A_sub'*r;
rnorm = norm(r,2);
gnorm = norm(g,2);
sl = (rnorm/gnorm)^2;
z = z+sl*g;
x = sign(x).*max(0,abs(x)-lambda);
```

LBP

SPGl1

```
function [x,r,g,info] = spgl1( A, b, tau, sigma, x, options )

m = length(b);

%----------------------------------------------------------------------
% Check arguments.
%----------------------------------------------------------------------
if ~exist('options','var'), options = []; end
if ~exist('x','var'), x = []; end
if ~exist('sigma','var'), sigma = []; end
if ~exist('tau','var'), tau = []; end

if nargin < 2 || isempty(b) || isempty(A)
   error('At least two arguments are required');
elseif isempty(tau) && isempty(sigma)
   tau = 0;
   sigma = 0;
   singleTau = false;
elseif isempty(sigma) % && ~isempty(tau)  <-- implied
   singleTau = true;
else
   if isempty(tau)
      tau = 0;
   end
   singleTau = false;
end

%----------------------------------------------------------------------
% Grab input options and set defaults where needed.
%----------------------------------------------------------------------
defaultopts = spgSetParms(...
'fid'        ,      1 , ... % File ID for output
'verbosity'  ,      2 , ... % Verbosity level
'iterations' ,   10*m , ... % Max number of iterations
'nPrevVals'  ,      3 , ... % Number previous func values for linesearch
'bpTol'      ,  1e-06 , ... % Tolerance for basis pursuit solution
'optTol'     ,  1e-04 , ... % Optimality tolerance
'decTol'     ,  1e-04 , ... % Req'd rel. change in primal obj. for Newton
'stepMin'    ,  1e-16 , ... % Minimum spectral step
'stepMax'    ,  1e+05 , ... % Maximum spectral step
'rootMethod' ,      2 , ... % Root finding method: 2=quad,1=linear (not used).
'activeSetIt',    Inf , ... % Exit with EXIT_ACTIVE_SET if nnz same for # its.
'subspaceMin',      0 , ... % Use subspace minimization
'iscomplex'  ,    NaN , ... % Flag set to indicate complex problem
'maxMatvec'  ,    Inf , ... % Maximum matrix-vector multiplies allowed
'weights'    ,      1 , ... % Weights W in ||Wx||_1
'Kaczmarz'   ,      0 , ... % Toggles whether Kaczmarz mode is on (experimental)
'KaczScale'  ,      1 , ... % Scaling factor for Tau when using Kaczmarz-type submatrices
'quitPareto' ,      0 , ... % Exits when pareto curve is reached
'minPareto'  ,      3 , ... % If quitPareto is on, the minimum number of iterations before checking for quitPareto conditions
'lineSrchIt' ,      1 , ... % Maximum number of line search iterations for spgLineCurvy, originally 10 ...
'feasSrchIt' ,  10000 , ... % Maximum number of feasible direction line search iteraitons, originally 10 ...
'ignorePErr' ,      0 , ... % Ignores projections error by issuing a warning instead of an error ...
'project'    , @NormL1_project , ...
'primal_norm', @NormL1_primal  , ...
'dual_norm'  , @NormL1_dual      ...
   );
options = spgSetParms(defaultopts, options);

fid           = options.fid;
logLevel      = options.verbosity;
maxIts        = options.iterations;
nPrevVals     = options.nPrevVals;
bpTol         = options.bpTol;
optTol        = options.optTol;
decTol        = options.decTol;
stepMin       = options.stepMin;
stepMax       = options.stepMax;
activeSetIt   = options.activeSetIt;
subspaceMin   = options.subspaceMin;
maxMatvec     = max(3,options.maxMatvec);
weights       = options.weights;
quitPareto    = options.quitPareto;
minPareto     = options.minPareto;
Kaczmarz      = options.Kaczmarz;
lineSrchIt    = options.lineSrchIt;
feasSrchIt    = options.feasSrchIt;
ignorePErr    = options.ignorePErr;

% maxLineErrors TEMPORARILY DISABLED to prevent very lar[illegible] issue[illegible] algorithm
maxLineErrors = Inf;     % Maximum number of line-search failures.
pivTol        = 1e-12;  % Threshold for significant Newton step.

%----------------------------------------------------------------------
% Initialize local variables.
%----------------------------------------------------------------------
iter          = 0;  [illegible]tnTotLSQR = 0; % Total SPGL1 and LSQR iterations.
[illegible]dAt    = 0;
[illegible]Vals,1);  % Last m function values.
nLineTot      = 0;                % Total no. of linesearch steps.
printTau      = false;
nNewton       = 0;
bNorm         = norm(b,2);
stat          = false;
timeProject   = 0;
timeMatProd   = 0;
nnzIter       = 0;                % No. of its with fixed pattern.
nnzIdx        = [];               % Active-set indicator.
subspace      = false;            % Flag if did subspace min in current itn.
stepG         = 1;                % Step length for projected gradient.
testUpdateTau = 0;                % Previous step did not update tau

% Determine initial x, vector length n, and see if problem is complex
explicit = ~(isa(A,'function_handle'));
```

So apparent that you do not really need a stopwatch.

SLIM

LBP vs. SPGl1: where are the complexities

SPGl1

- automatic calculates the sparsity parameter
  - determining whether the Pareto curve is reached
  - computing local slope of the Pareto curve
- non-deterministic line-search

LBP

- so far uses more "**heuristic**" threshold
  - easy implementation comes at a price
- uses deterministic line search
  - also the least costly among all gradient descent methods

Herrmann and Li, 2011; Tu and Herrmann, 2012; Tu and Herrmann, 2013

SLIM

LBP vs. SPGl1: rerandomization benefits

Rerandomization

- **redraw** source experiments/frequencies
  - speedup of convergence in terms of model errors
  - improved robustness to linearization errors
  - ***no overhead*** on simulation cost

SPGl1

- redraw **occasionally**
  - once the Pareto curve is reached

LBP

- redraw **every iteration**
  - leveraging the sparse Kaczmarz solver

Herrmann and Li, 2011; Tu and Herrmann, 2012

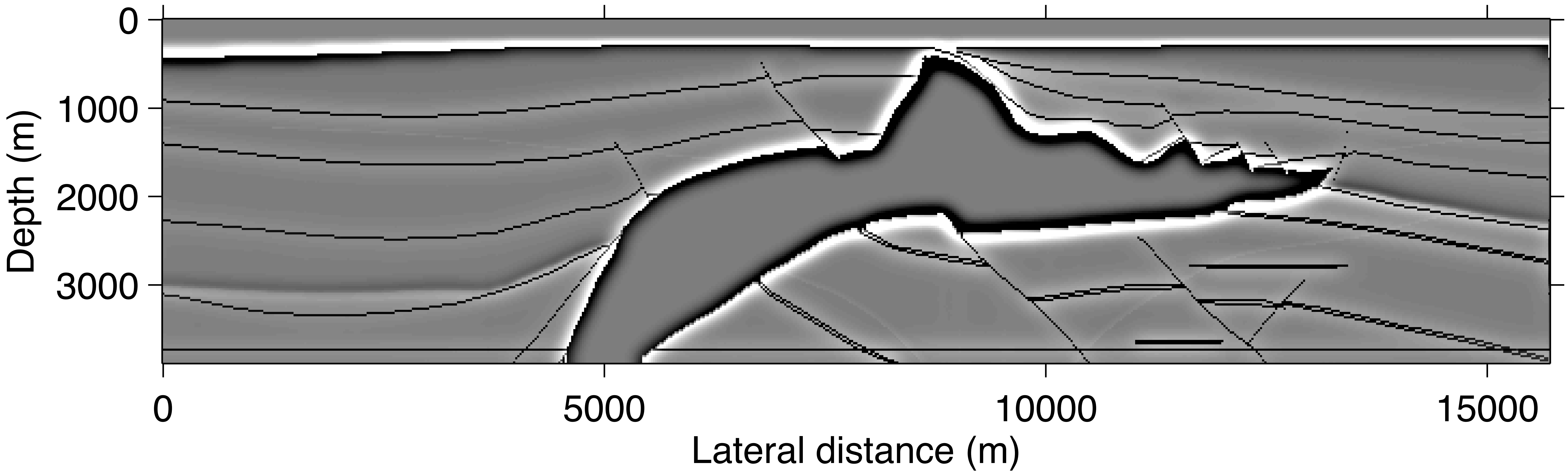
SLIM

LBP vs. SPGl1: imaging examples

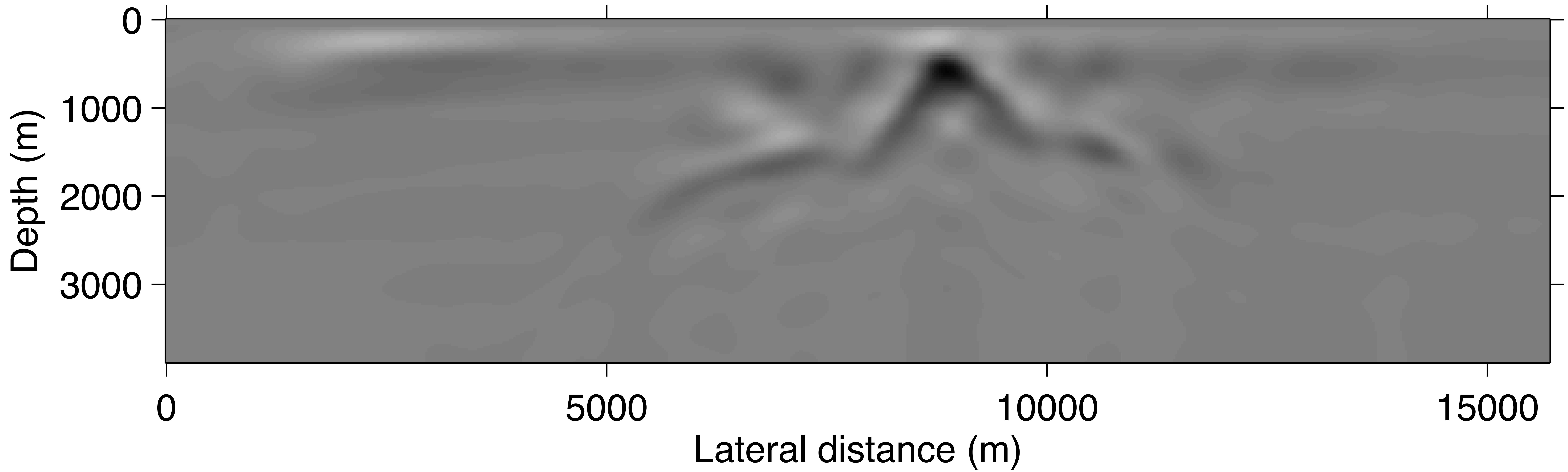
fast least-squares migrations of
- primaries
- primaries + surface multiples

- simulation cost ~1 RTM of all the data
  - ***very cheap*** (50 fold reduction) per-iteration cost by source/freq. subsampling
- input data generated by linearized modelling
- thresholding **heuristics**
  - keep 1% of curvelet coefficients at first
  - gradually relaxed to 5%

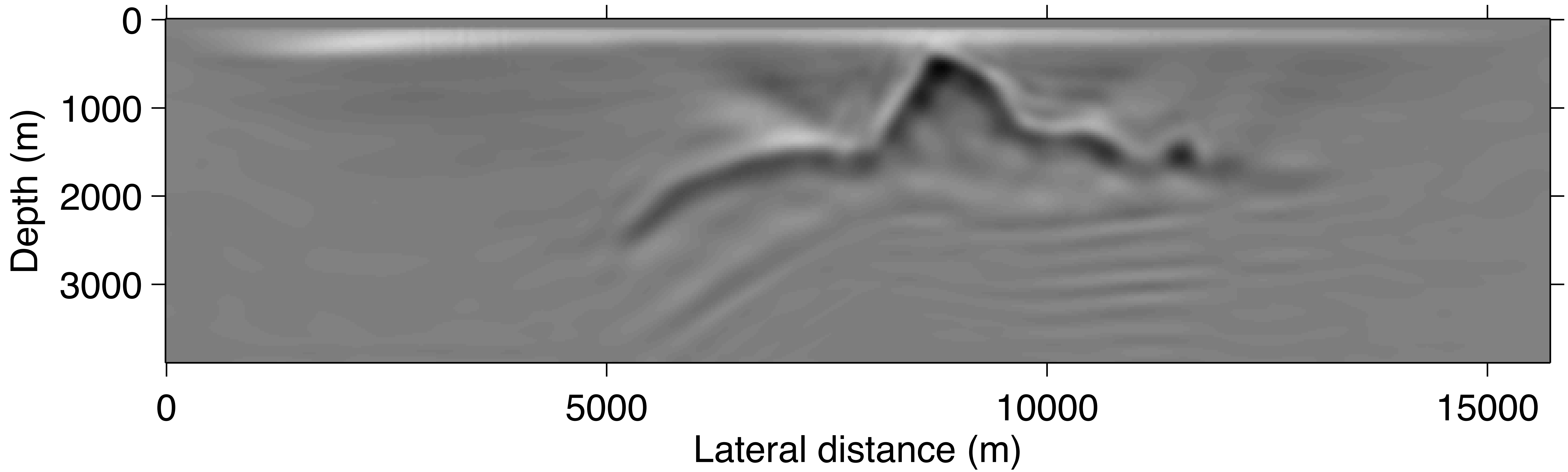
Fast imaging of primaries: true image



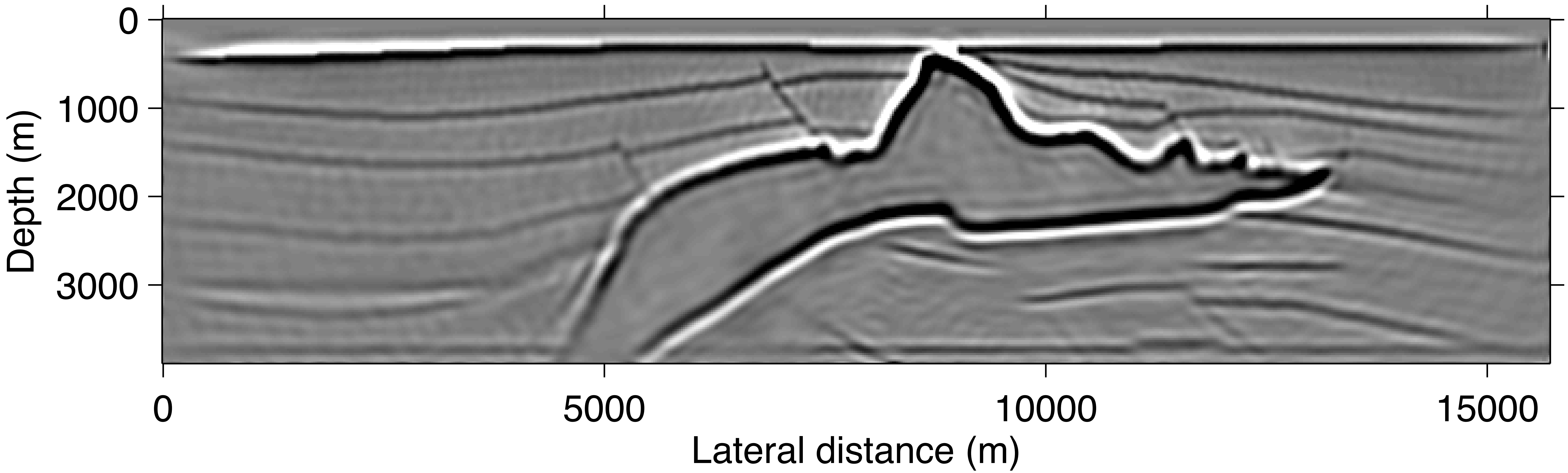
Fast imaging of primaries: by SPGl1, 1st iteration



Fast imaging of primaries: by LBP, 1st iteration

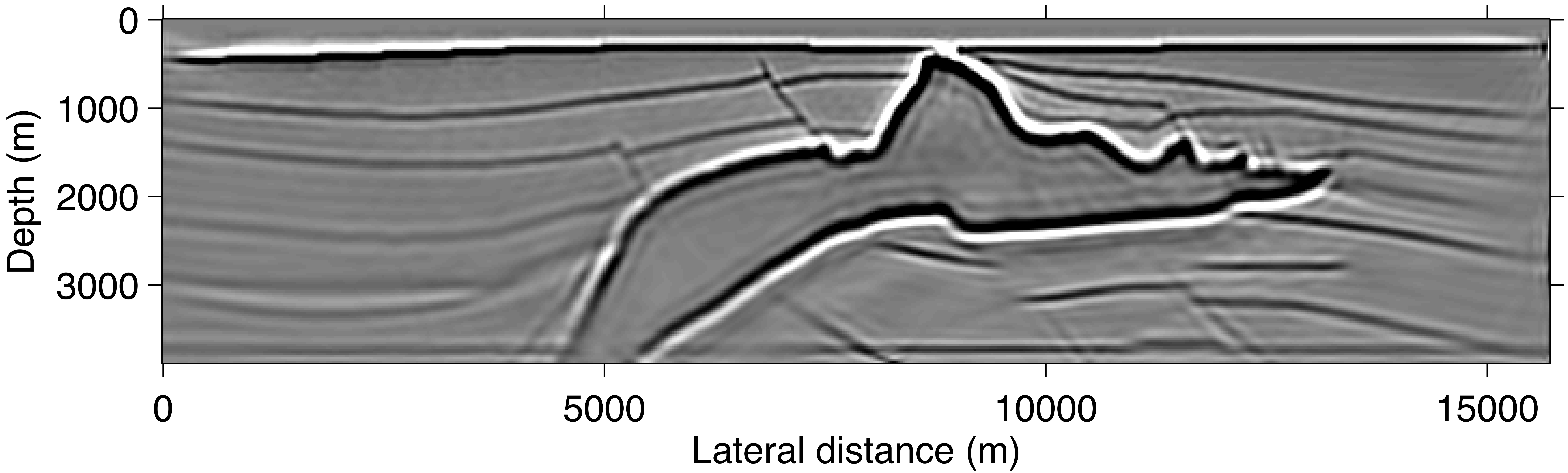


Fast imaging of primaries: by SPGl1, final image



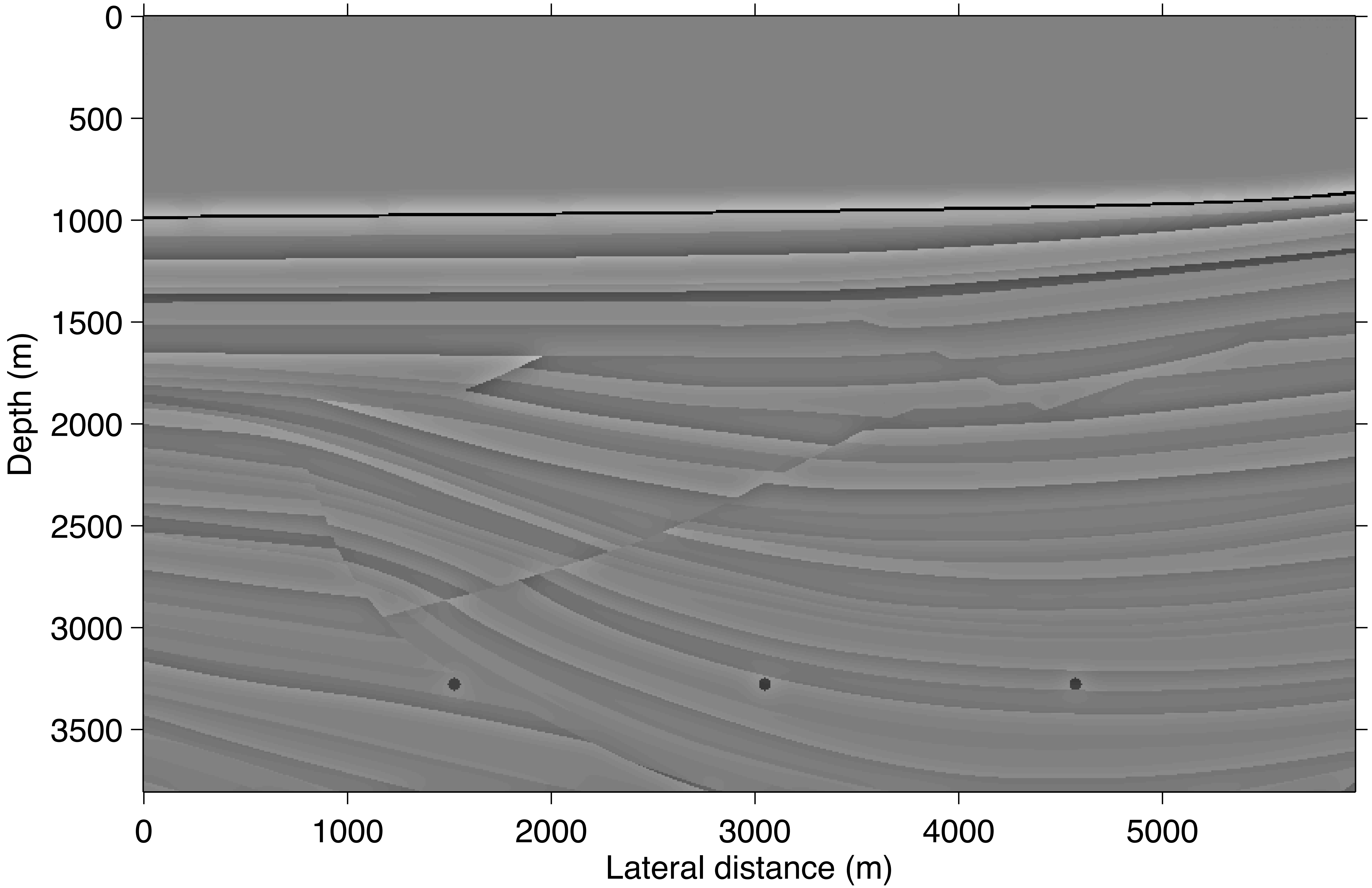
Simulation cost **~1 RTM** using all the data

Fast imaging of primaries: by **LBP**, final image

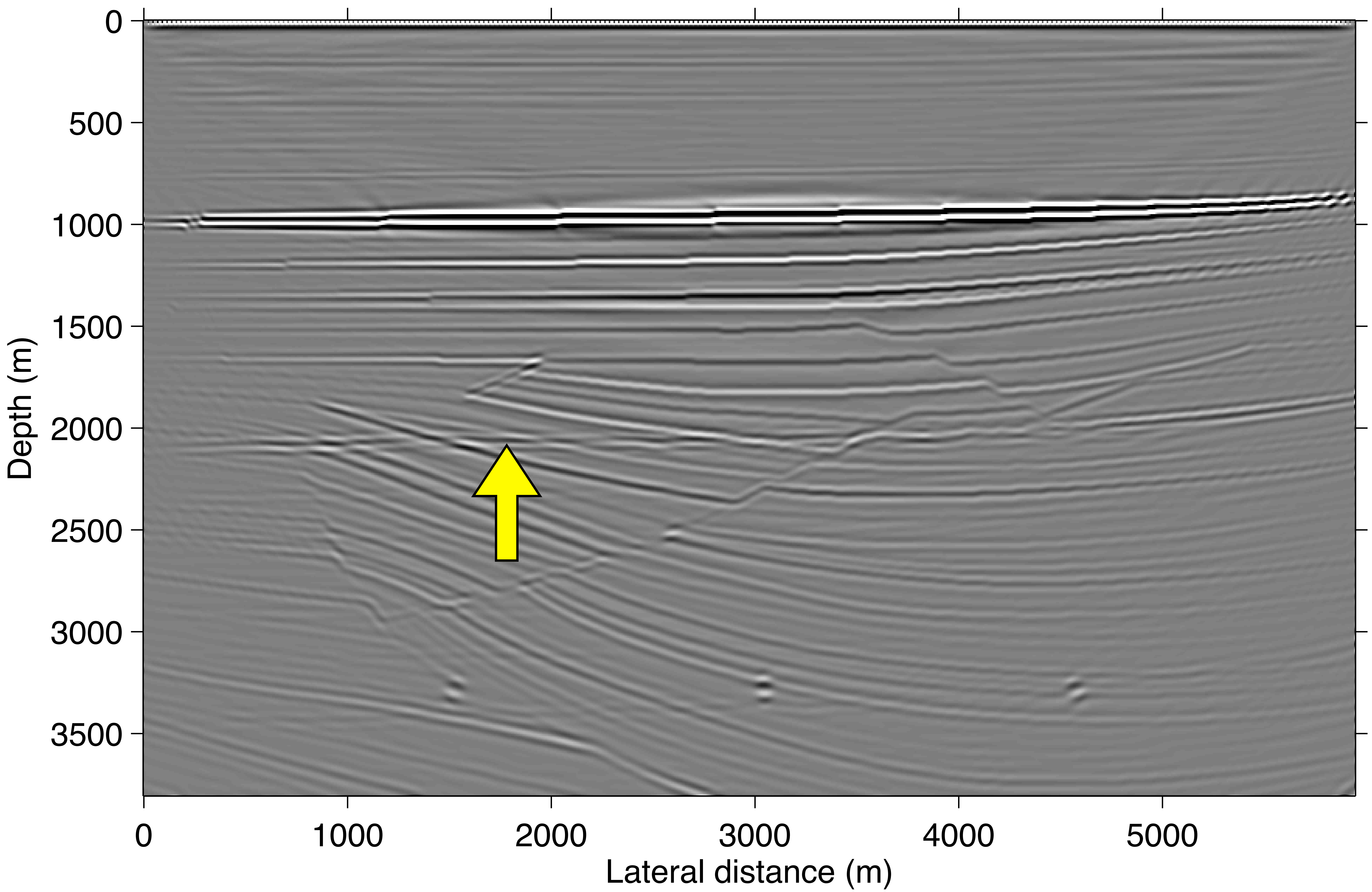


Simulation cost **~1 RTM** using all the data

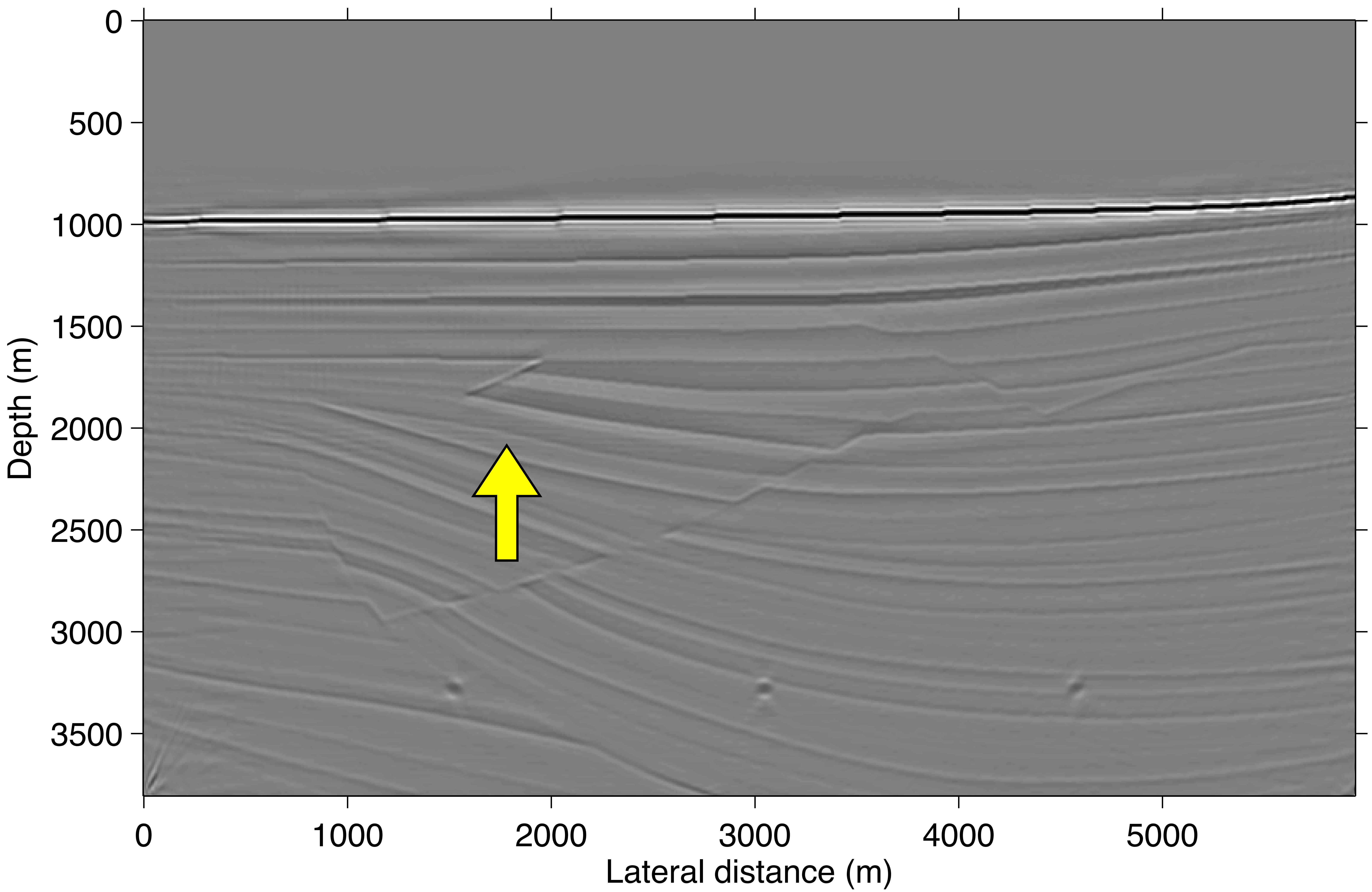
Fast imaging with multiples: true image



RTM with multiples [more details in Tue talk @ 4:30PM]

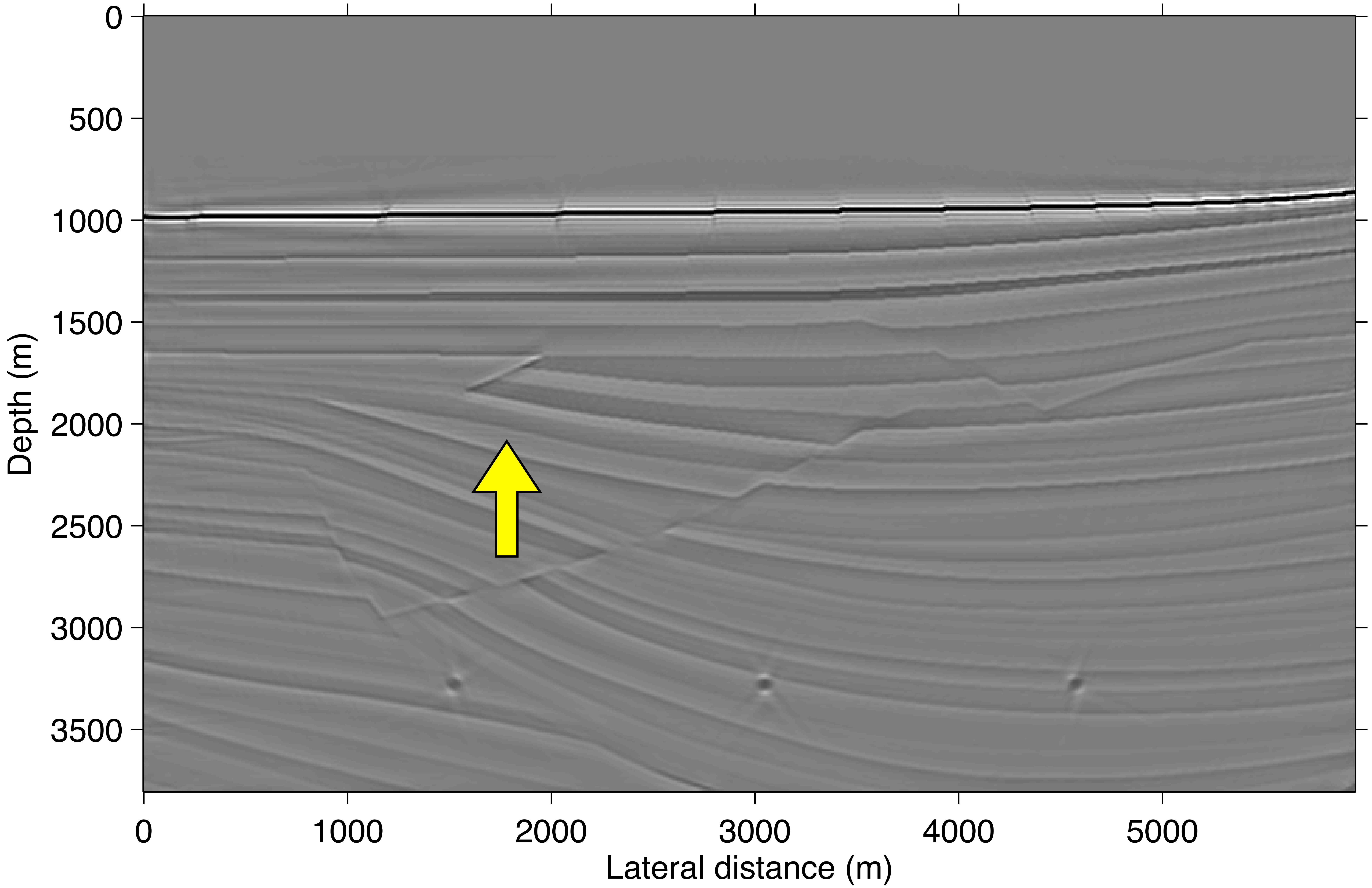


Fast imaging with multiples: by SPGl1, final image



Simulation cost **~1 RTM** using all the data

Fast imaging with multiples: by LBP, final image



Simulation cost **~1 RTM** using all the data

Data interpolation via matrix completion

$$\min_{\mathbf{X}} \quad \|\mathbf{X}\|_* \quad \text{s.t. } \|\mathcal{A}(X) - \mathbf{b}\|_2^2 \leq \sigma$$

where

$$\mathcal{A}(.) = \mathbf{M}\mathcal{S}^H(.)$$

$\mathbf{M}$ time-jittered operator

$\mathcal{S}^H$ transform operator

Algorithm

Huang et. al. '11

$$\text{Input} : \mathbf{X}^o = \hat{\mathbf{X}}^o = \mathbf{P}^o = \hat{\mathbf{P}}^o = 0, \mu = 5n, \tau = 1/\mu, \alpha = 1$$

for $k = 0, 1, \ldots$ **do**

$$\mathbf{X}^{k+1} := \text{Shrink}(\mathbf{X}^k - \mu(\tau \mathcal{A}'(\mathcal{A}(X) - \mathbf{b}) - \mathbf{P^k}), \mu)$$

$$\mathbf{P}^{k+1} := \hat{\mathbf{P}}^k - (\tau \mathcal{A}'(\mathcal{A}(\hat{\mathbf{X}}^k) - \mathbf{b}) - (\mathbf{X^{k+1}} - \hat{\mathbf{X}}^{\mathbf{k}})/\mu$$

$$\hat{\mathbf{X}}^{k+1} := \alpha_k \mathbf{X}^{k+1} + (1 - \alpha_k)\mathbf{X}^k$$

$$\hat{\mathbf{P}}^{k+1} := \alpha_k \mathbf{P}^{k+1} + (1 - \alpha_k)\mathbf{P}^k$$

$$\alpha_k := 1 + 2/(k+2) * ((k+2)/2 - 1)$$

end for

Algorithm

Huang et. al. '11

$$\mathrm{Shrink}(Y, \gamma) := U \mathrm{diag}(\max(\sigma - \gamma, 0)) V^H$$

where

$$Y = U \mathrm{diag}(\sigma) V^H$$

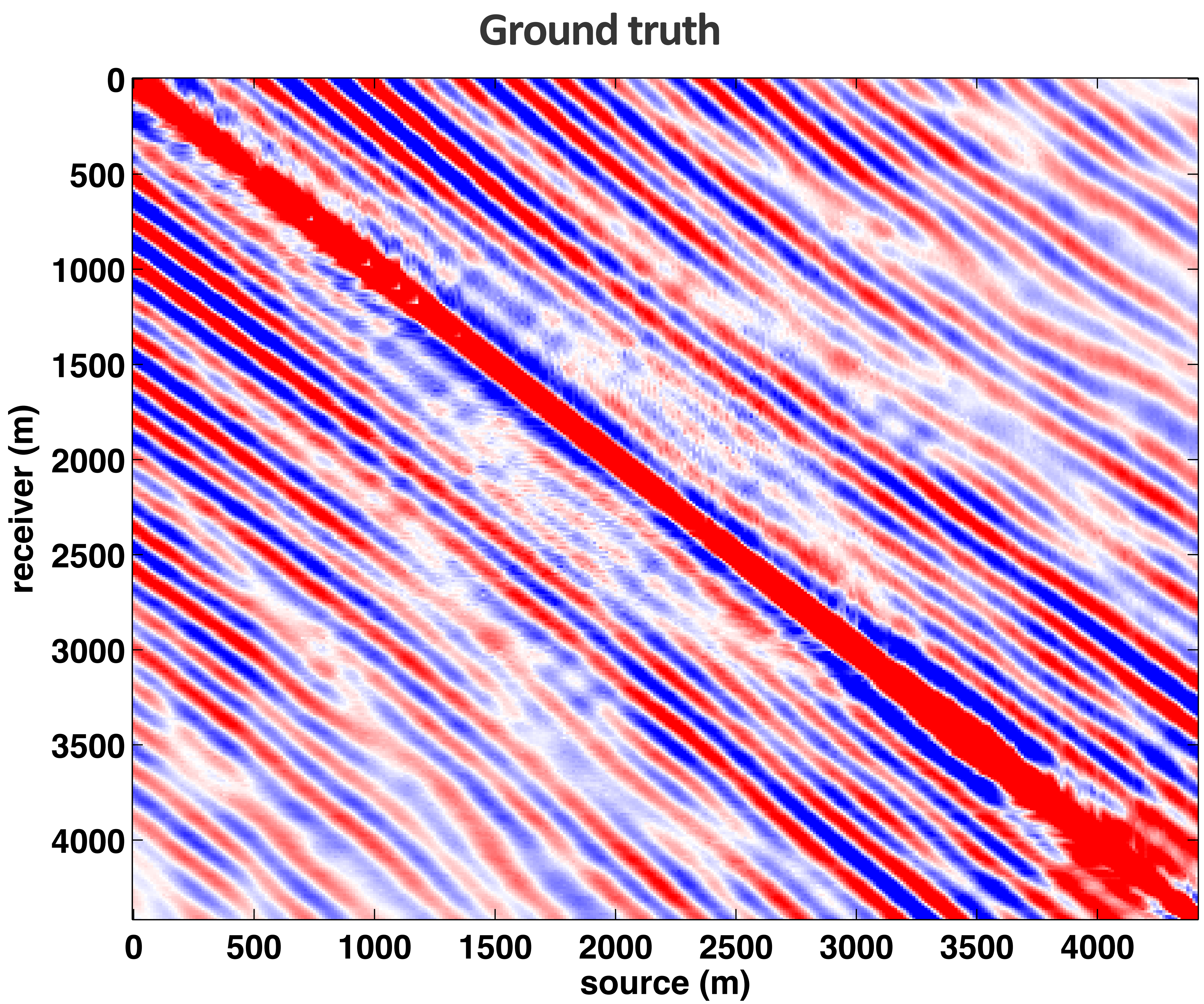
Matlab Code

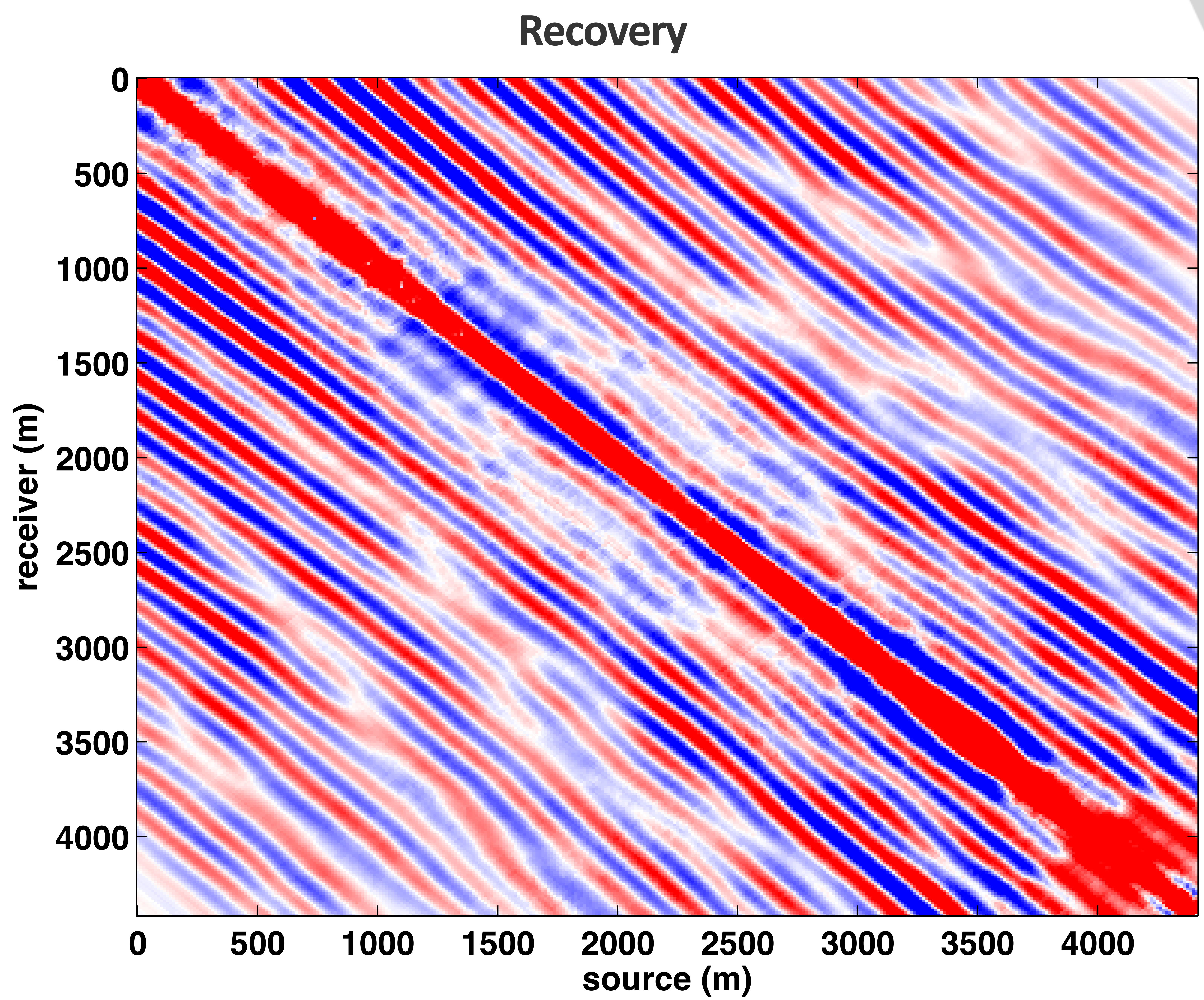
```
% as per Theorem 3.3
alpha = 1 + (2/(k+2)) * (1./theta - 1);
% As per algorithm 4 on page no 446
res   = A(X) - M;
X     = reshape(X - mu*(tau*(Adj(res) - P)),params.mhnumr, params.mhnumc);
X     = shrink( X , mu);
P     = Phat - tau*(Adj(A(Xhat) - M)) - (X - Xhat)./mu;
Xhat  = alpha*X + (1- alpha) * Xprev;
Phat  = alpha*P + (1- alpha) * Pprev;
Xprev = X;
Pprev = P;
% update theta
theta = 2/(k+2);
obj   = norm(A(X) - M,'fro')/norm(M,'fro');
k     = k+1;
```

Gulf of Suez
70% missing source

LBP

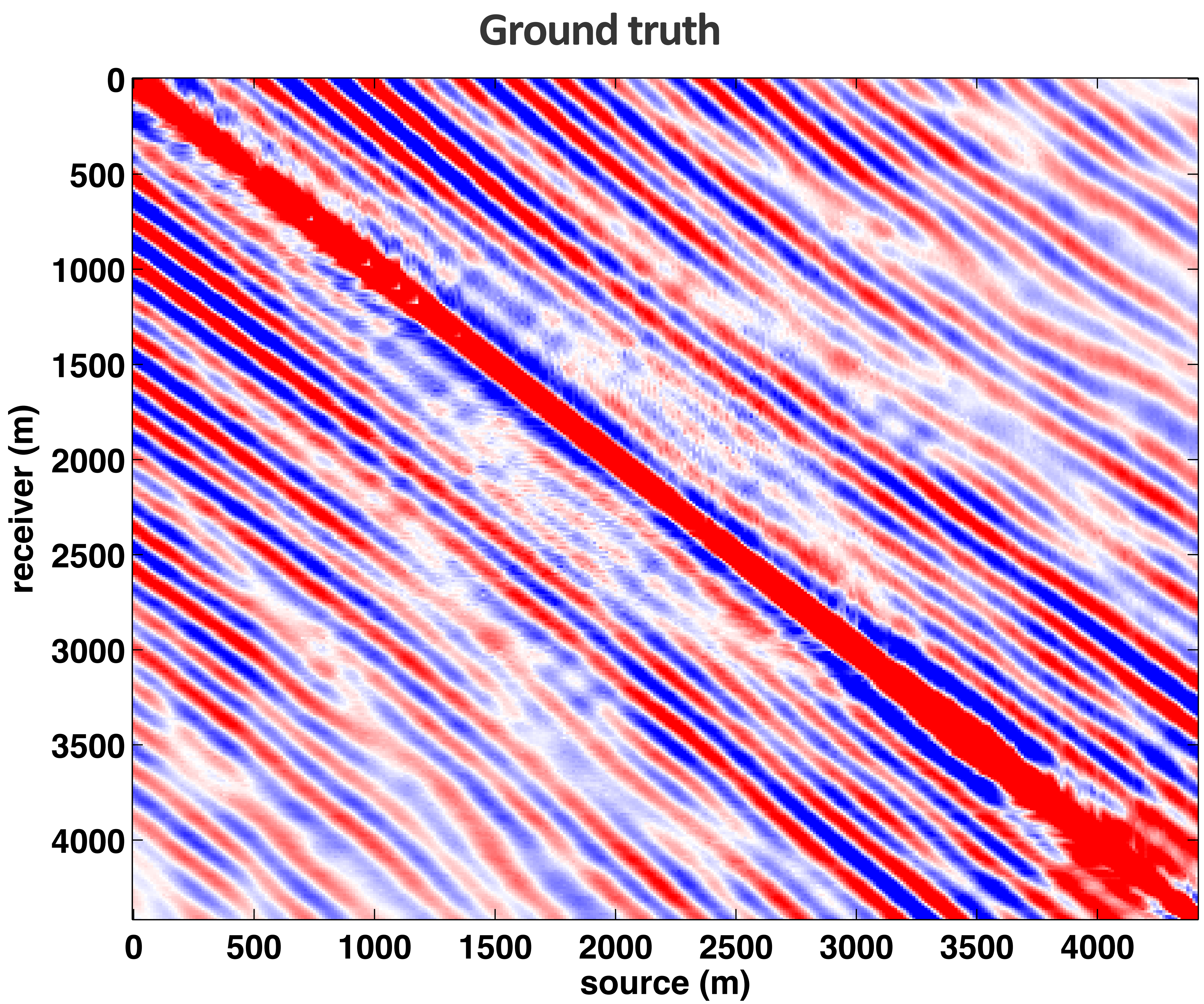
SNR = 15.9 dB

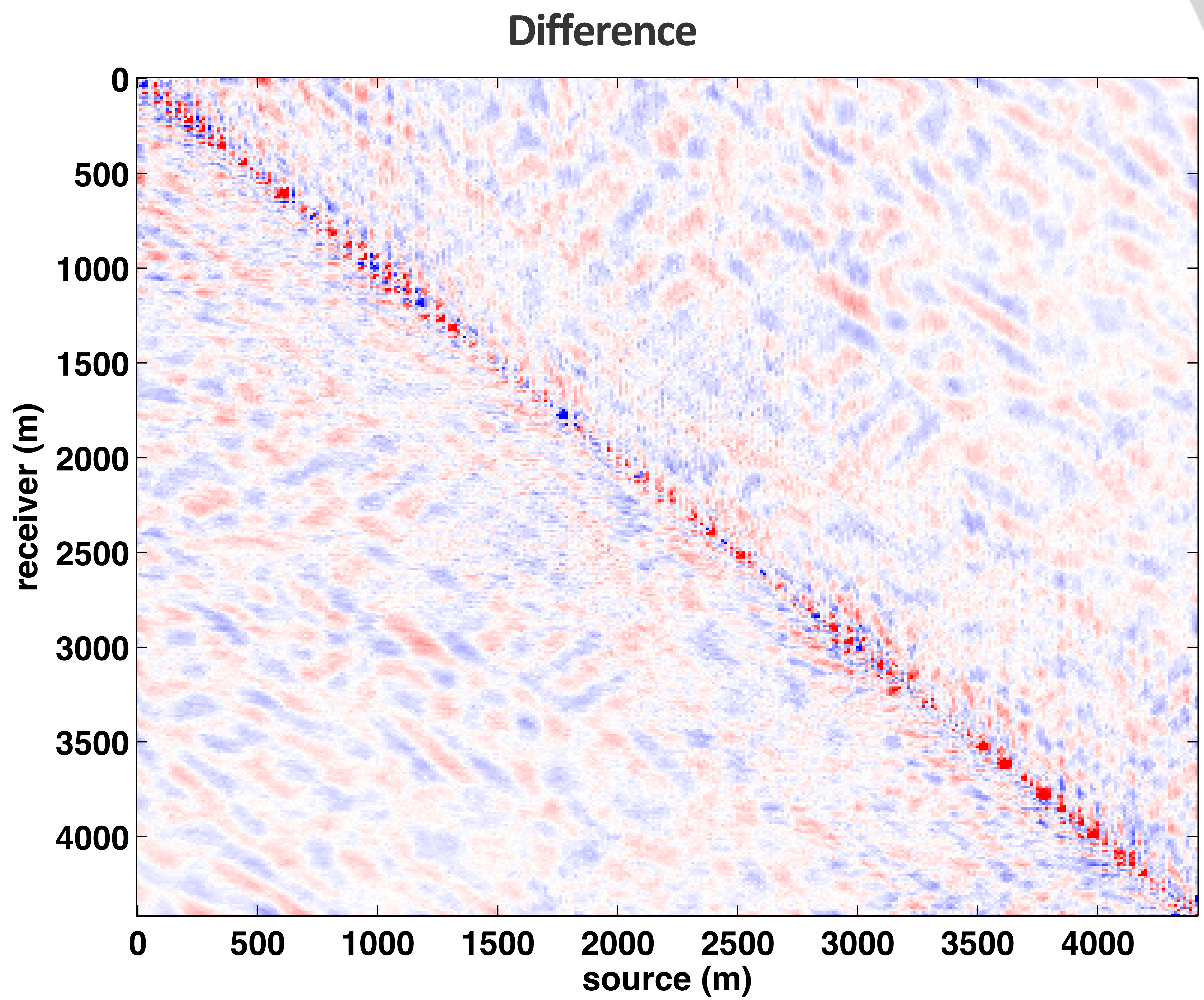




LBP

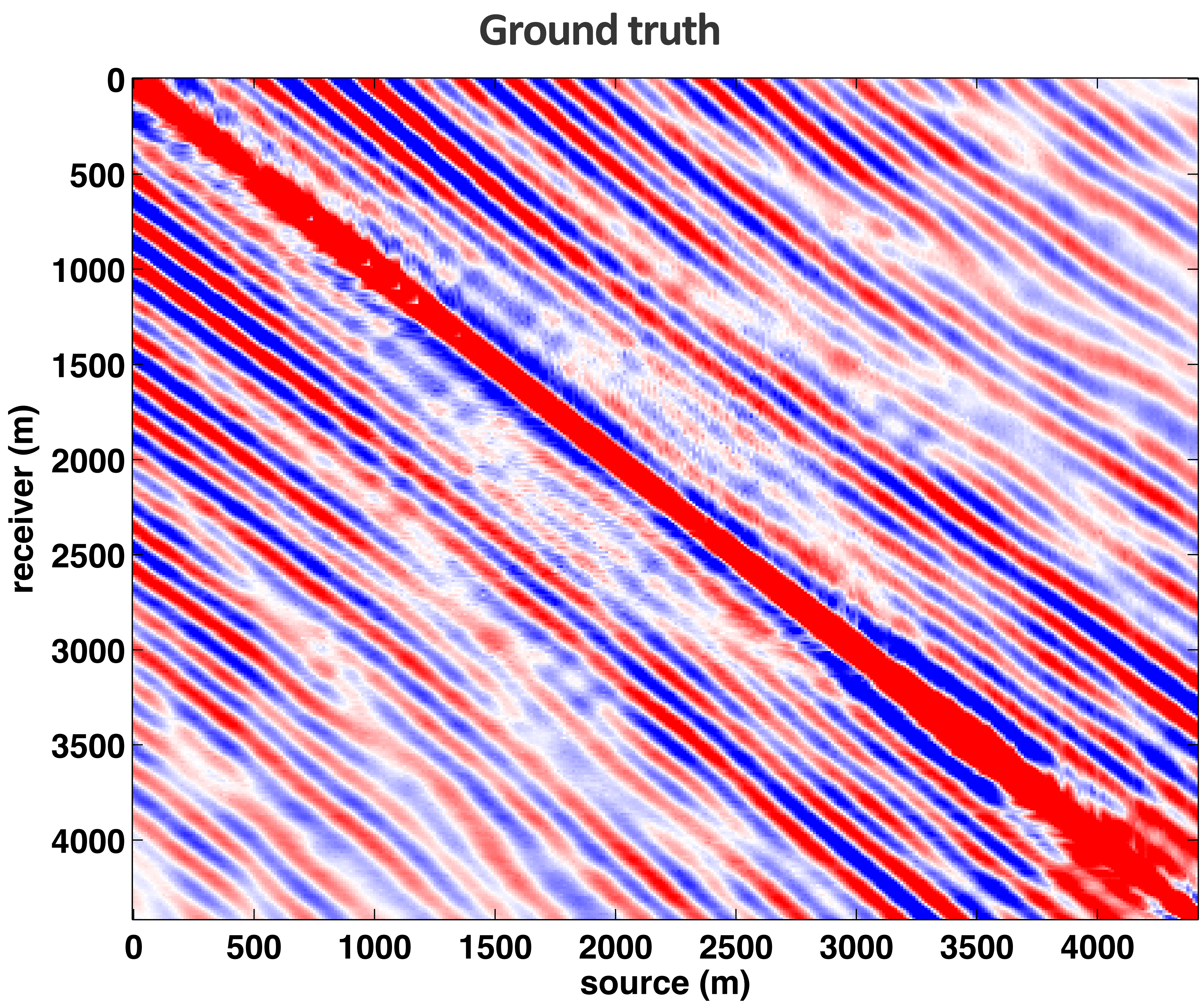
SNR = 15.9 dB

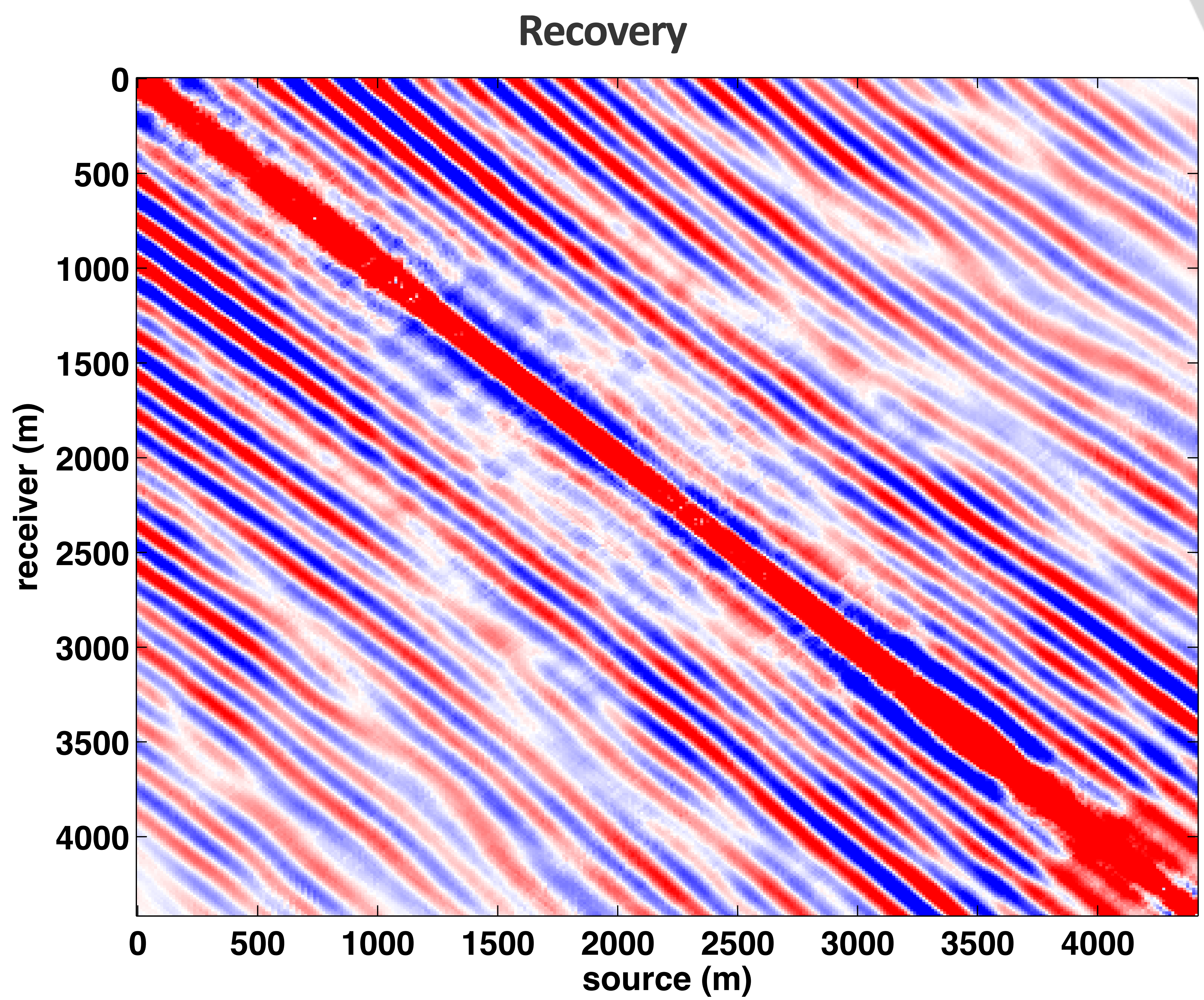




SPGL1

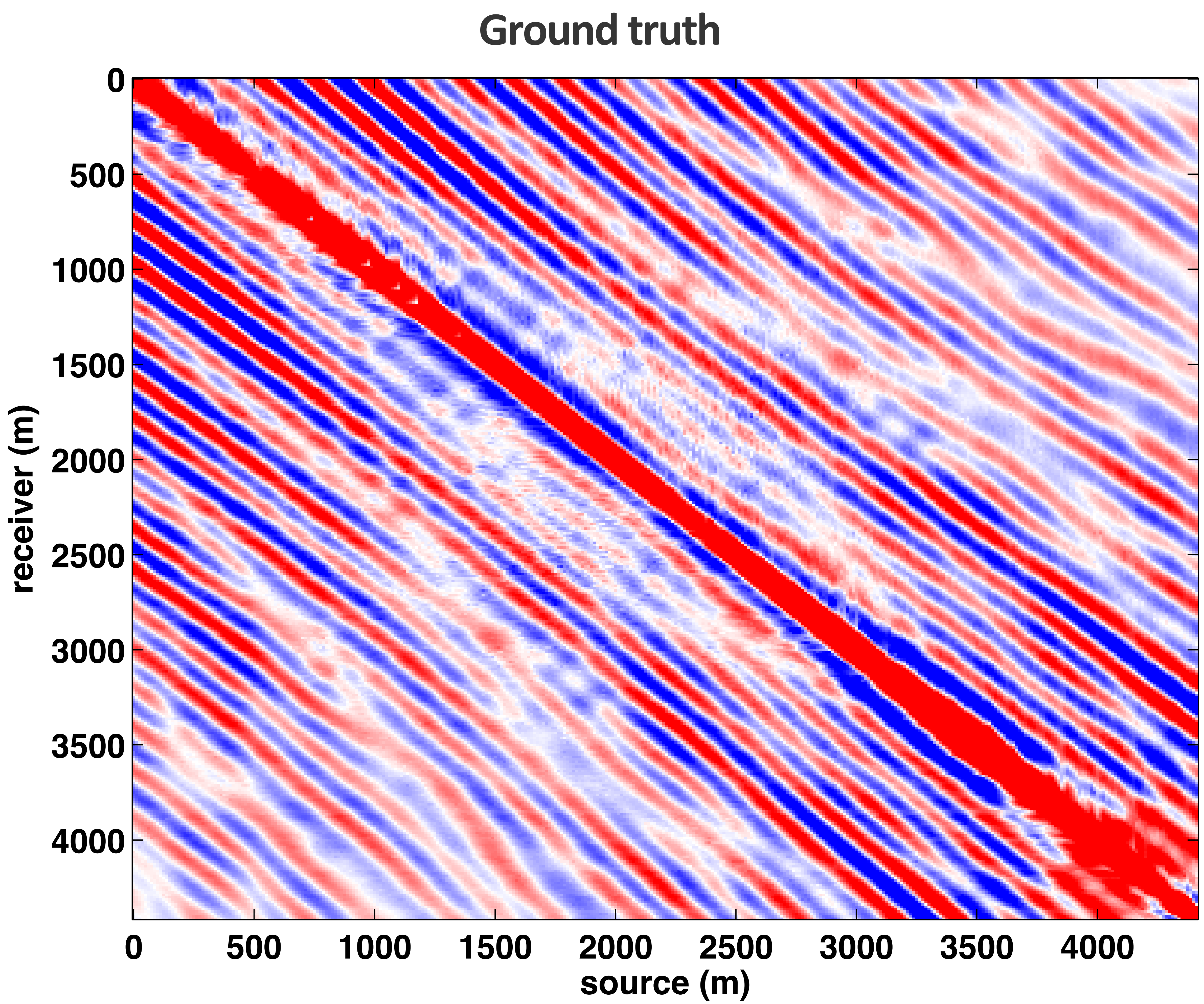
SNR = 15.3 dB

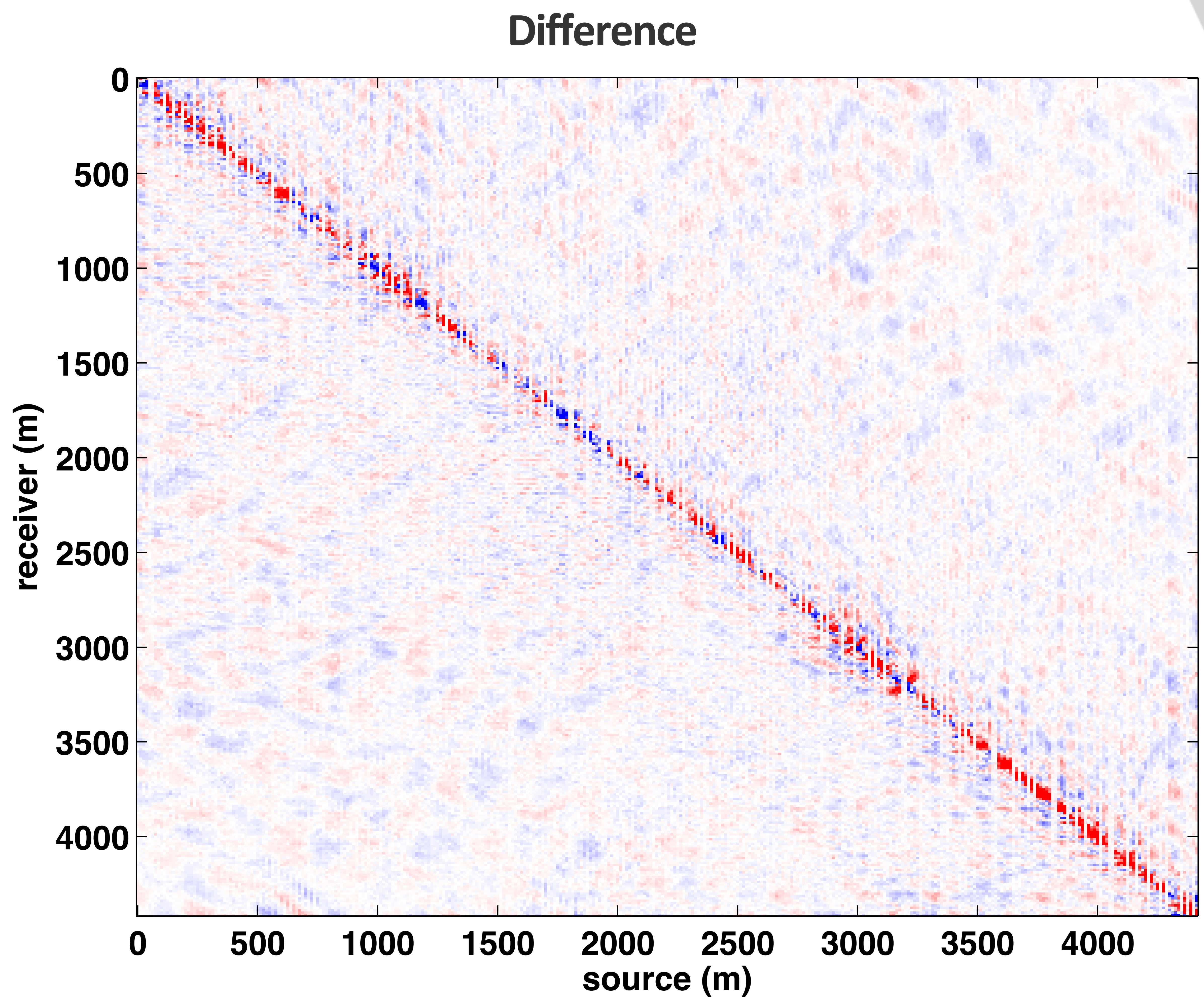




SPGL1

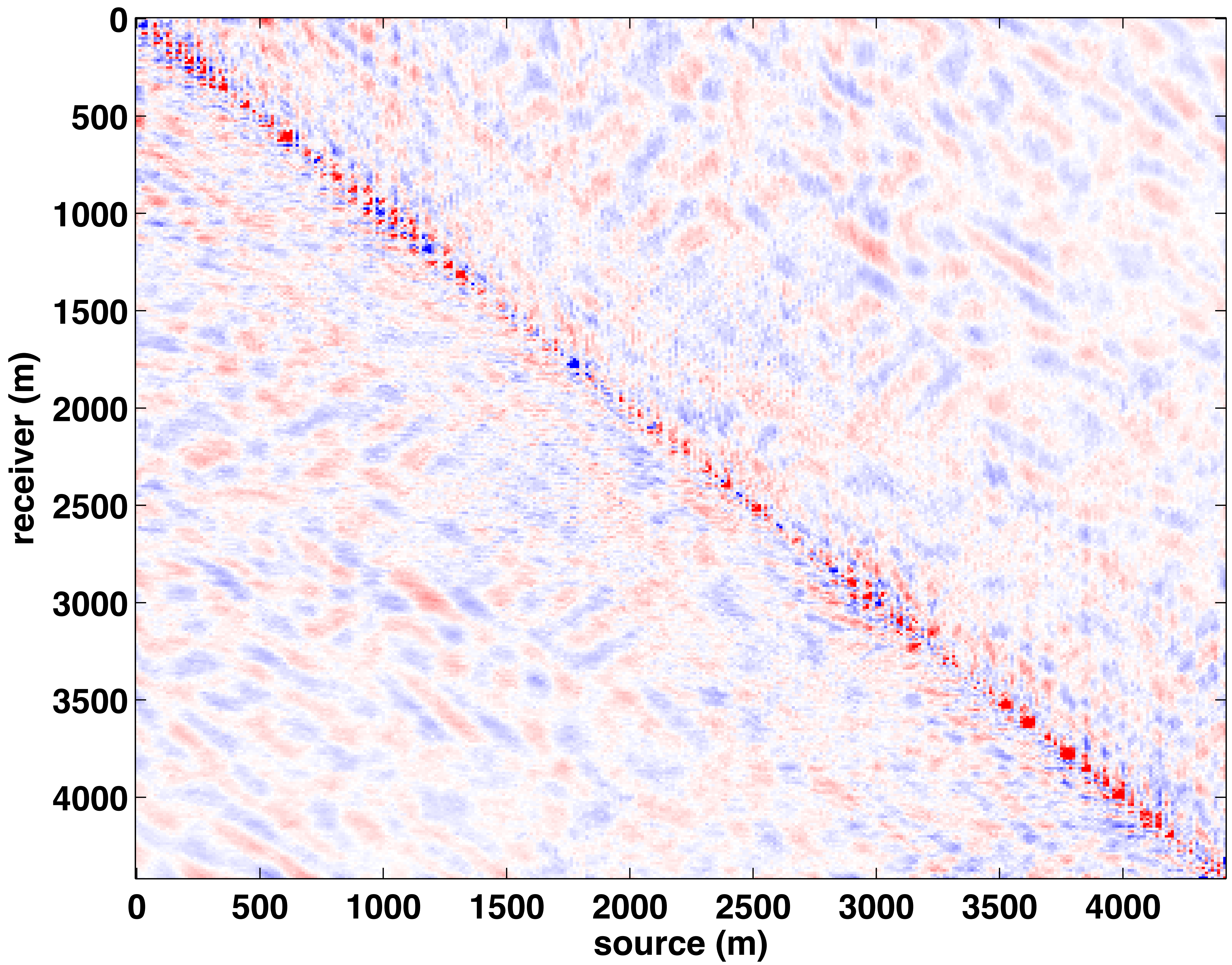
SNR = 15.3 dB



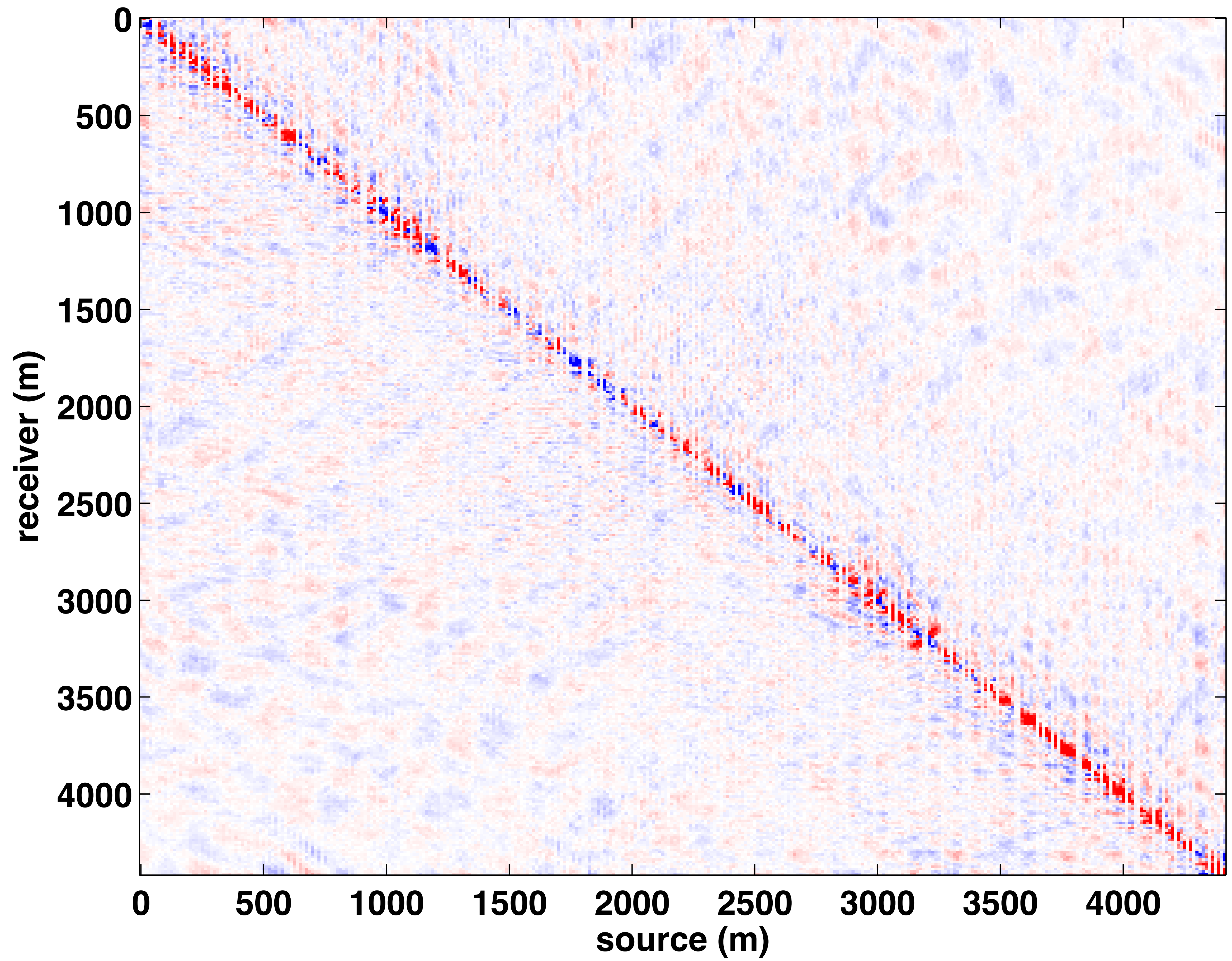


SPGL1 v/s LBP

LBP



SPGL1



Conclusion

- effectiveness of LBP-based sparse inversion approaches preliminarily verified

- **simple implementation**
  - to promote the uptake of SLIM technologies by industry

Future work

- more extensive testing
- more investigation into the heuristics
  - relating the heuristics to the solution path of SPGl1
- possible extension to other applications
- software release

Acknowledgements

Thank you all for your attention!

SINBAD

NSERC CRSNG

This work was in part financially supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant (22R81254) and the Collaborative Research and Development Grant DNOISE II (375142-08). This research was carried out as part of the SINBAD II project with support from the following organizations: BG Group, BGP, CGG, Chevron, ConocoPhillips, ION, Petrobras, PGS, Statoil, Total SA, Sub Salt Solutions, WesternGeco, and Woodside.

References

Friedlander, M.P., Tseng, P.: Exact regularization of convex programs. SIAM J. Optim. 18, 1326–1350 (2007)

Mangasarian, O.L., Meyer, R.R.: Nonlinear perturbation of linear programs. SIAM J. Control Optim. 17, 745–752 (1979)

Dirk Lorenz, Stephan Wenger, Frank Sch¨opfer, Marcus Magnor: A sparse Kaczmarz solver and a linearized Bregman method for online compressed sensing, arXiv 1403.7543v1

Ewout van den Berg and Michael P. Friedlander, "Probing the Pareto frontier for basis pursuit solutions", SIAM Journal on Scientific Computing, vol. 31, p. 890-912, 2008.

Felix J. Herrmann and Xiang Li, "Efficient least-squares imaging with sparsity promotion and compressive sensing", Geophysical Prospecting, vol. 60, p. 696-712, 2012.

Ning Tu and Felix J. Herrmann, "Imaging with multiples accelerated by message passing", SEG Technical Program Expanded Abstracts, 2012.

Tu, N., Li, X., & Herrmann, F. J., 2013a. Controlling linearization errors in l1 regularized inversion by rerandomization, in SEG Technical Program Expanded Abstracts, pp. 4640–4644, SEG.