Hierarchical Tucker Tensor Optimization - Applications to 4D Seismic Data Interpolation

Curt Da Silva and Felix J. Hermann

SLIM
University of British Columbia

SLIM

High Dimensional Interpolation

Motivation:

- High dimensional data volumes have dependencies between dimensions that we can exploit in order to interpolate missing data
- Full source data is useful in the context of generating simultaneous sources for FWI, quality control

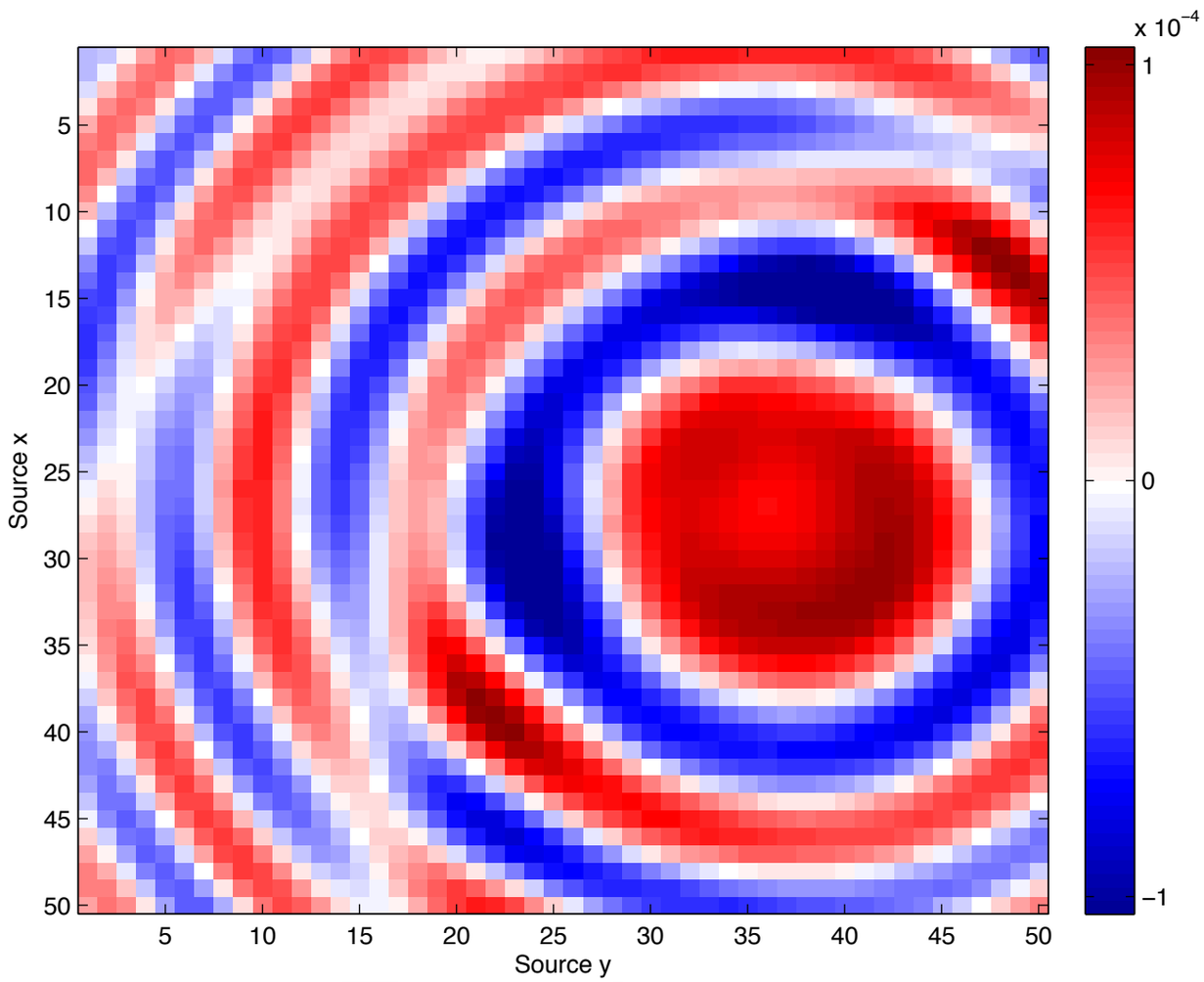
SLIM

High Dimensional Interpolation

Goal:

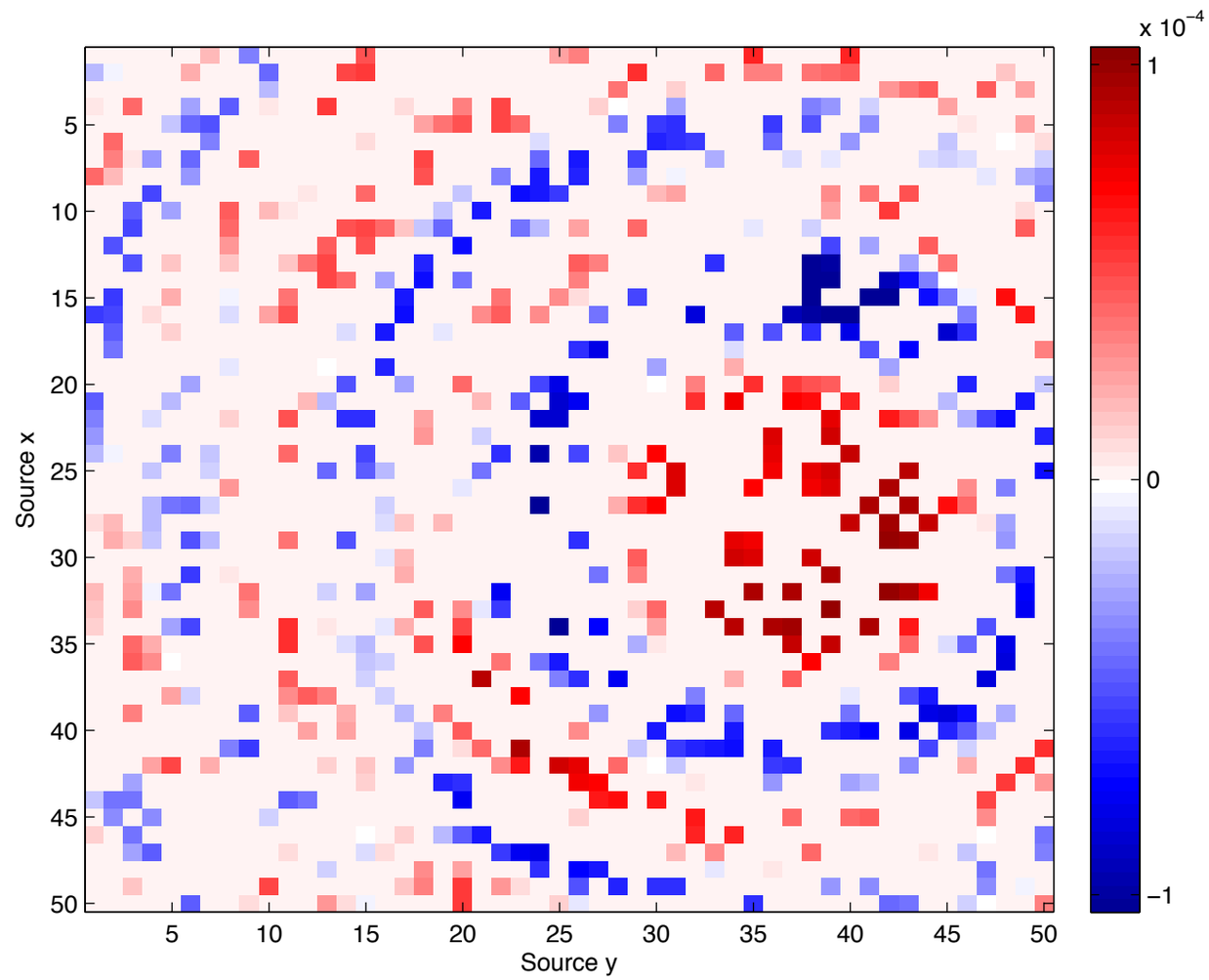
- Use our knowledge about favourable sampling conditions in 1D to design an optimization paradigm that favours missing data recovery
- Using randomized source acquisition to reduce costs of acquiring full source data

75% Missing Source Pairs



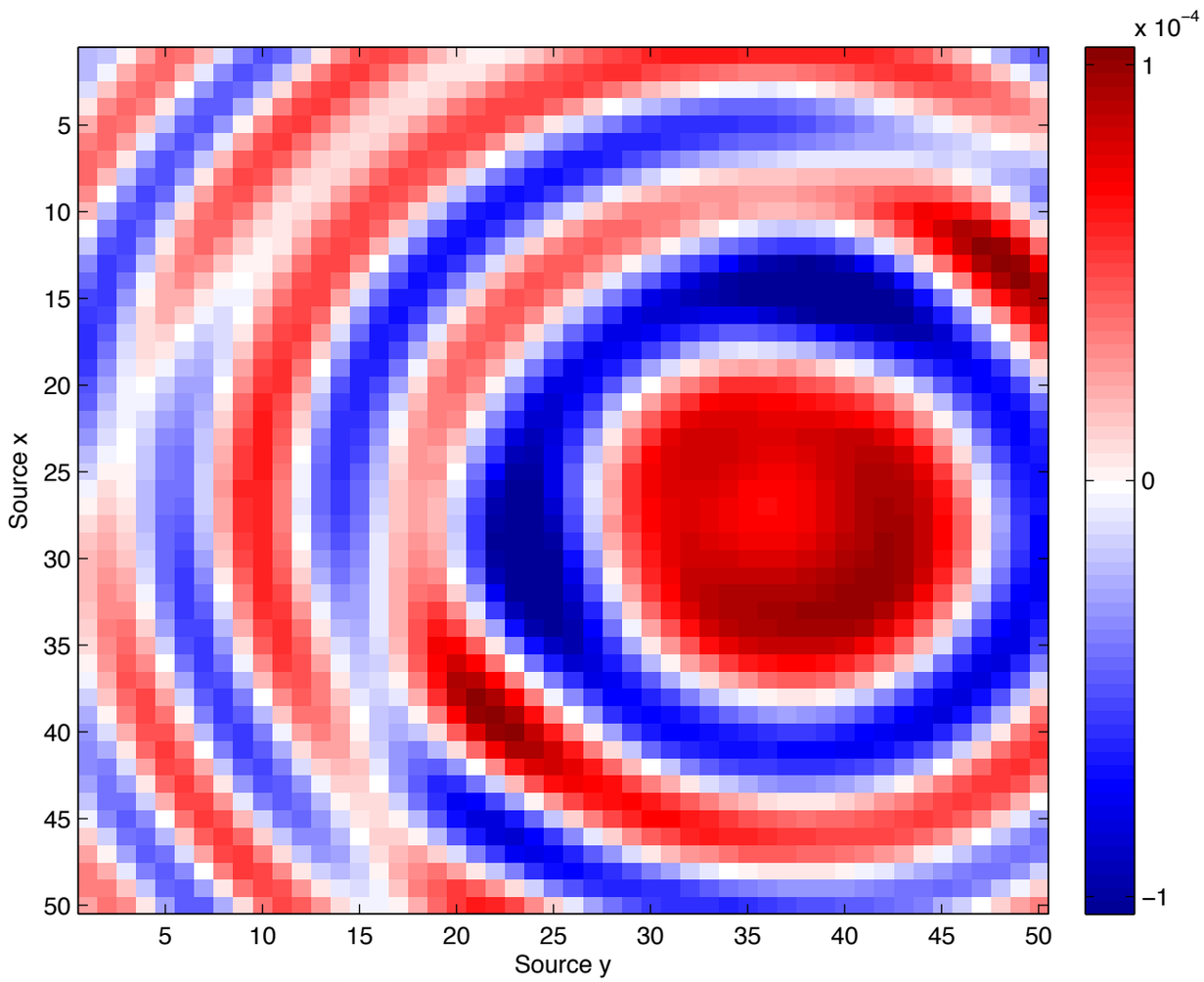
True data

(Rec x, Rec y) = (30,39)



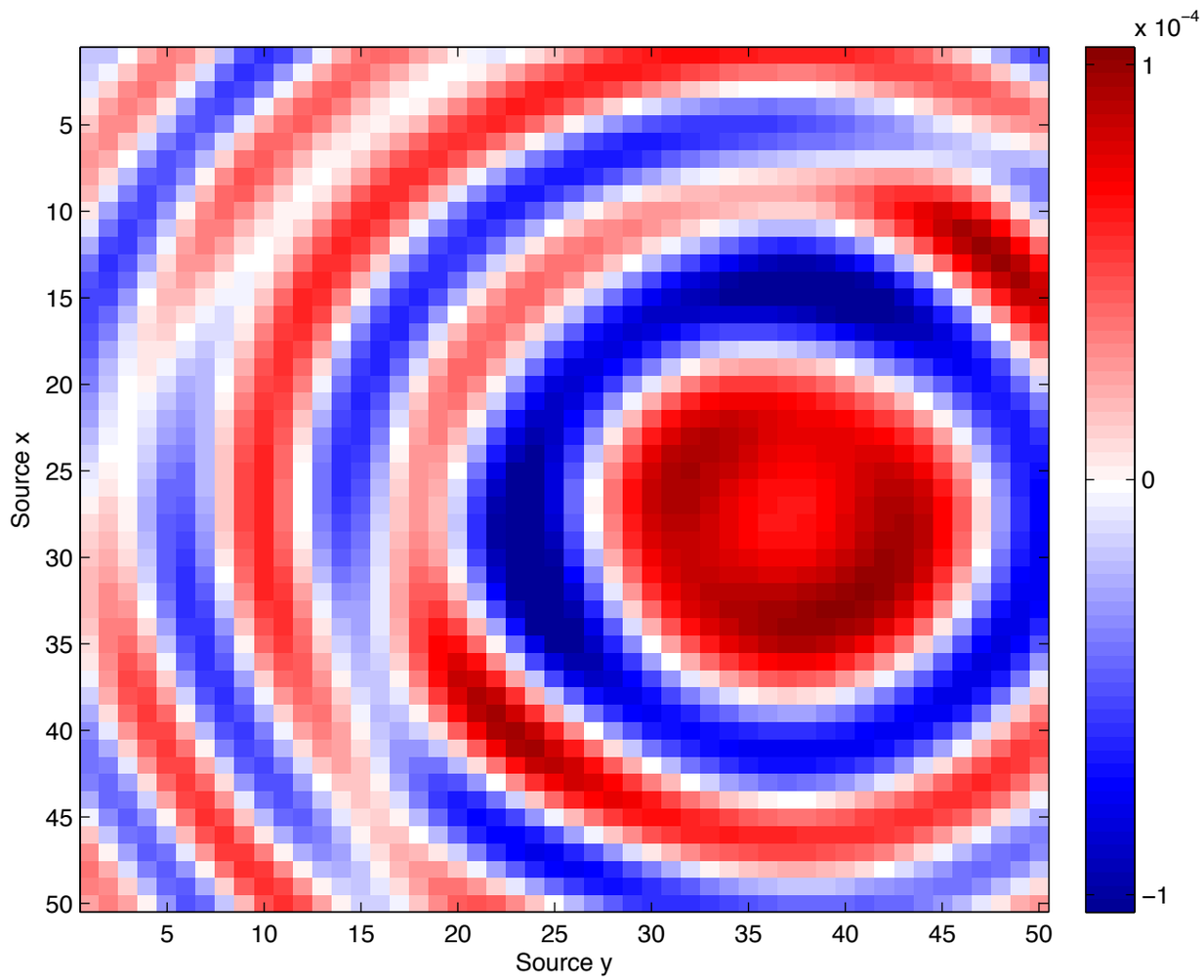
Subsampled data

75% Missing Source Pairs



True data

(Rec x, Rec y) = (30,39)



Recovered data

SNR - 18.7 dB

High Dimensional Interpolation

Goal:

- Generalize our notions of rank for matrices to higher-dimensional objects (tensors) and develop an efficient representation for these objects
- Avoid SVDs on full data

SLIM

Hierarchical Tucker Format

Matricization

- The *matricization* of a tensor X with dimensions $1, \ldots, d$ along the dimensions $t = (t_1, \ldots, t_r)$ is the matrix formed by placing the dimensions t along the rows and dimensions t^c along the columns
- Denoted $X^{(t)}$

Example in Matlab

```
n1 = 20; n2 = 20; n3 = 20; n4 = 20;
% Tensor
x = randn(n1,n2,n3,n4);
% Matricization along dimensions 1 and 2
x12 = reshape(x,n1 * n2, n3 * n4);
% Matricization along dimensions 3 and 4
y34 = permute(x,[3 4 1 2]);
x34 = reshape(x, n3 * n4, n1 * n2);
% Matricization along dimensions 1 and 3
y13 = permute(x,[1 3 2 4]);
x13 = reshape(x,n1 * n3, n2 * n4);
```

$X^{(1,2)}$

$X^{(3,4)}$

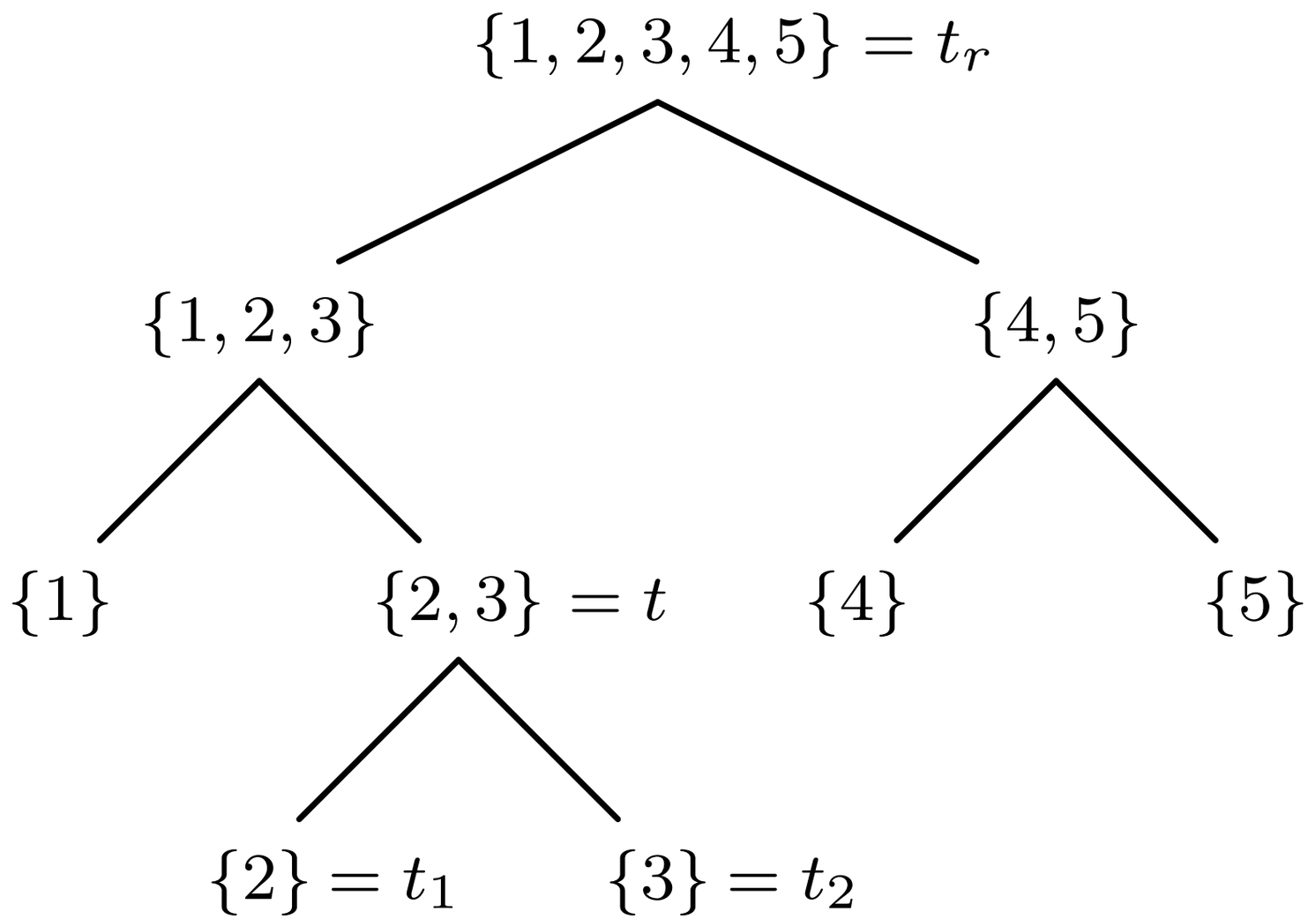
$X^{(1,3)}$

Hierarchical Tucker Format

- A *dimension tree* T for dimensions $\{1, \ldots, d\}$ is a non-trivial binary tree such that
  - the root, t_{root}, is has the label $\{1, \ldots, d\}$
  - each non-leaf node, t, can be written as $t = t_l \cup t_r$ where t_l is the left child of t, t_r is the right child of t $\quad t_l \cap t_r = \emptyset$

A. Uschmajew, B. Vandereycken. The geometry of algorithms using hierarchical tensors. 2012

Example



Hierarchical Tucker Format

- A tensor X can be written in the *Hierarchical Tucker format* corresponding to a dimension tree T and a vector of hierarchical ranks

$$(k_t)_{t \in T}, k_{\text{root}} = 1$$

if it can be written as

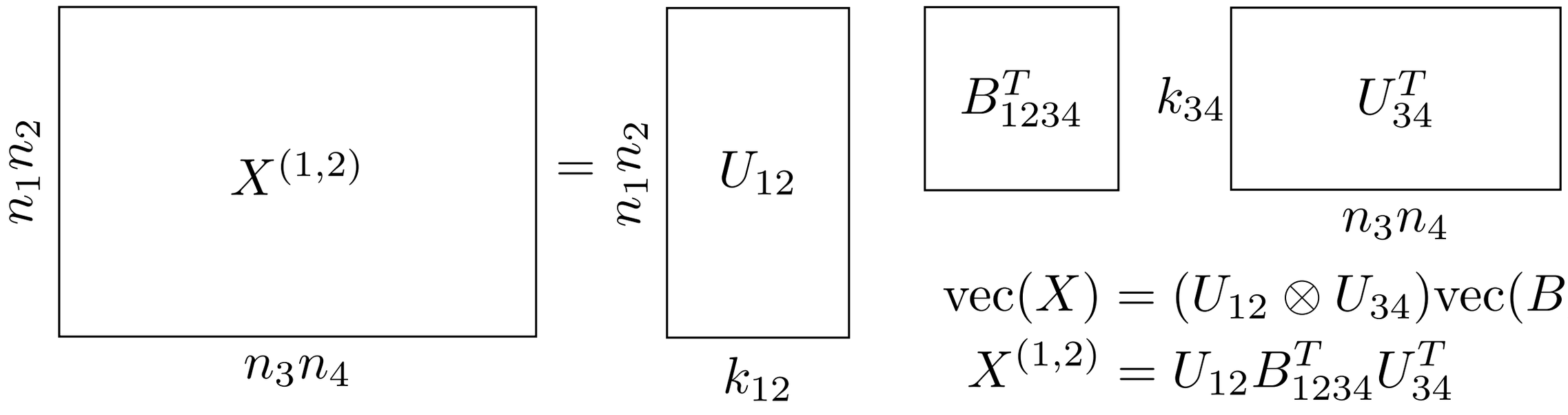
Hierarchical Tucker Format

$$\mathrm{vec}(X) = (U_{t_l} \otimes U_{t_r}) B_{t_{\mathrm{root}}}^{(1,2)} \quad t = t_{\mathrm{root}}$$

$$U_t = (U_{t_l} \otimes U_{t_r}) B_t^{(1,2)} \quad t \text{ not a leaf}$$

$$U_t \in \mathbb{C}^{n_t \times k_t} \qquad B_t \in \mathbb{C}^{k_l \times k_r \times k_t}$$

Example $X - n_1 \times n_2 \times n_3 \times n_4$ tensor

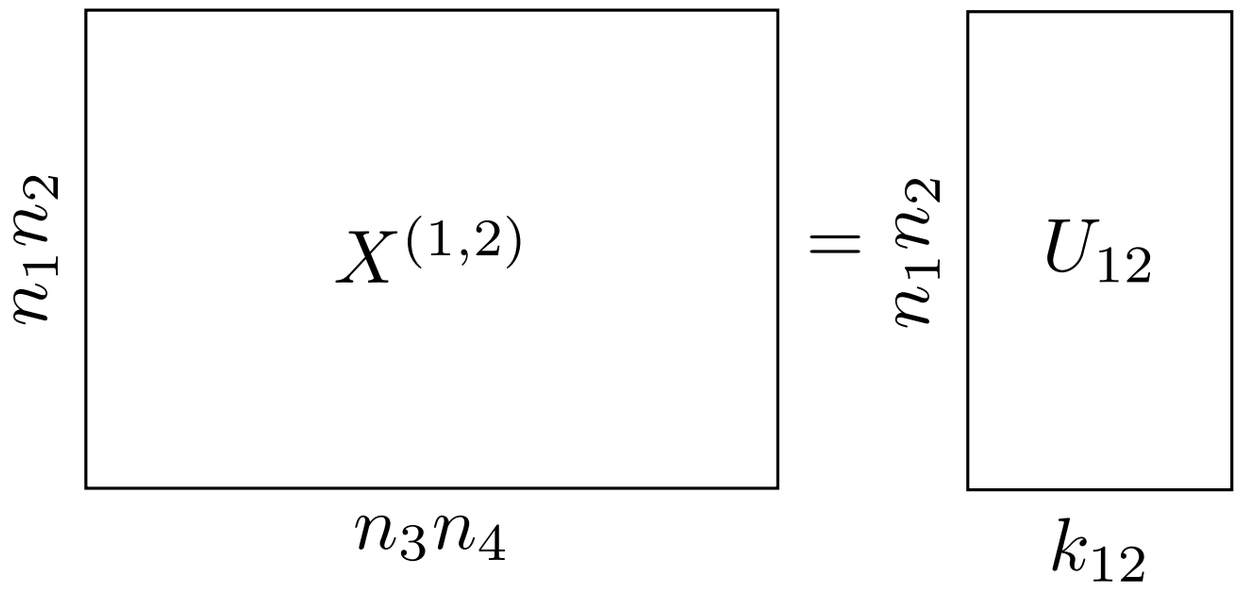


$$\text{vec}(X) = (U_{12} \otimes U_{34})\text{vec}(B_{1234})$$

$$X^{(1,2)} = U_{12} B^T_{1234} U^T_{34}$$

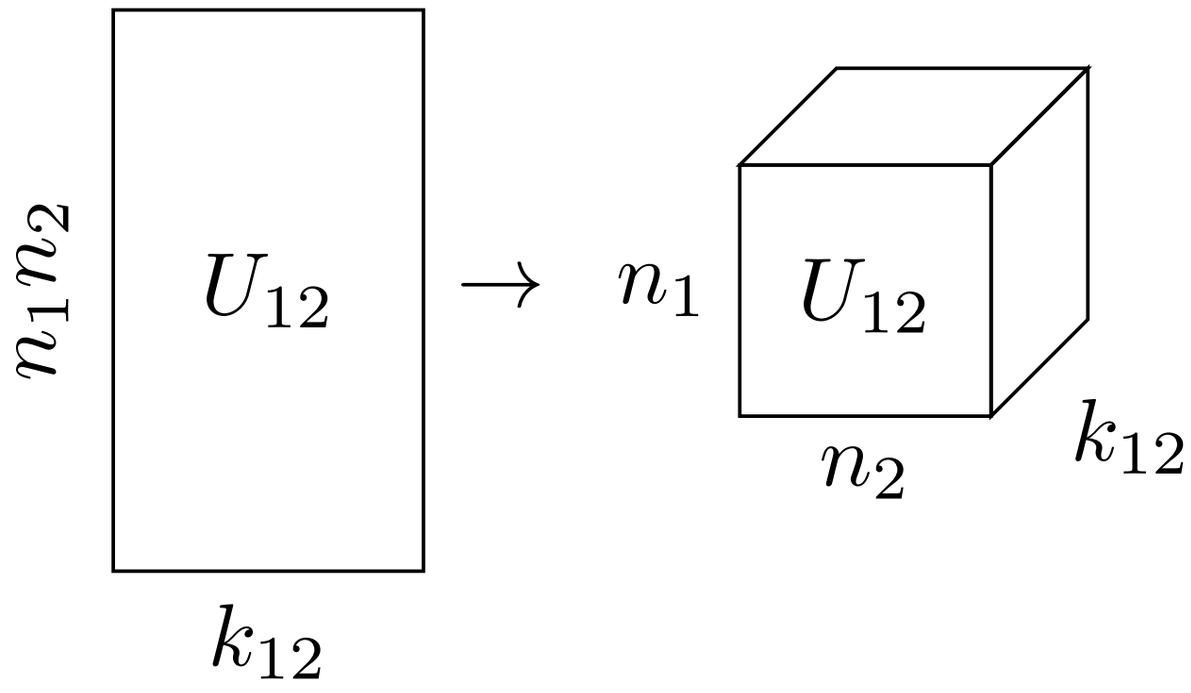
Example

$X - n_1 \times n_2 \times n_3 \times n_4$ tensor

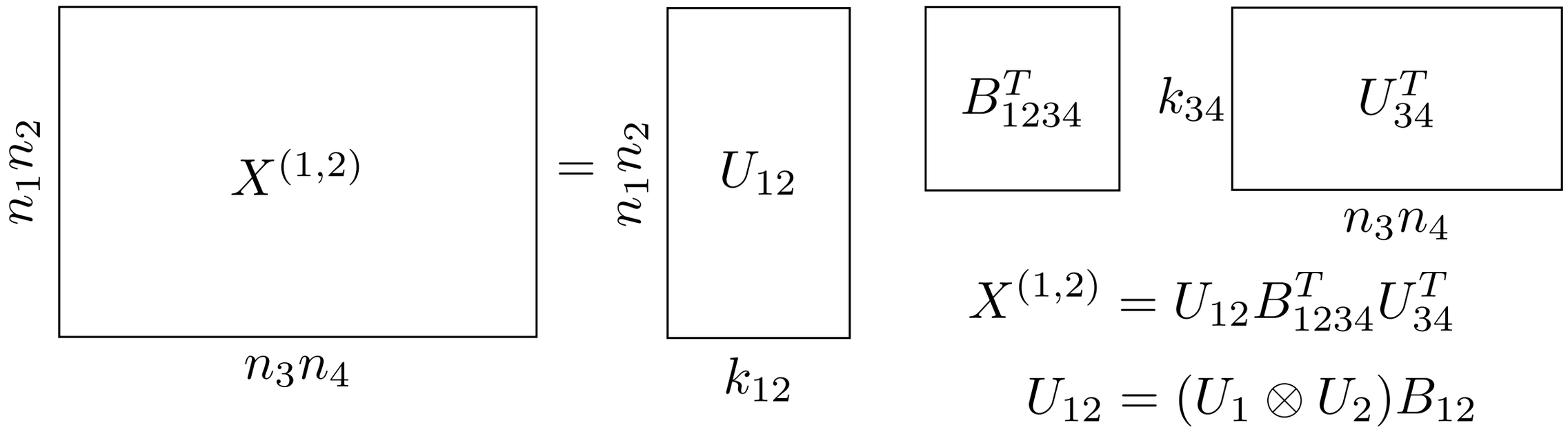


$$X^{(1,2)} = U_{12} B^T_{1234} U^T_{34}$$

$$U_{12} = (U_1 \otimes U_2) B_{12}$$

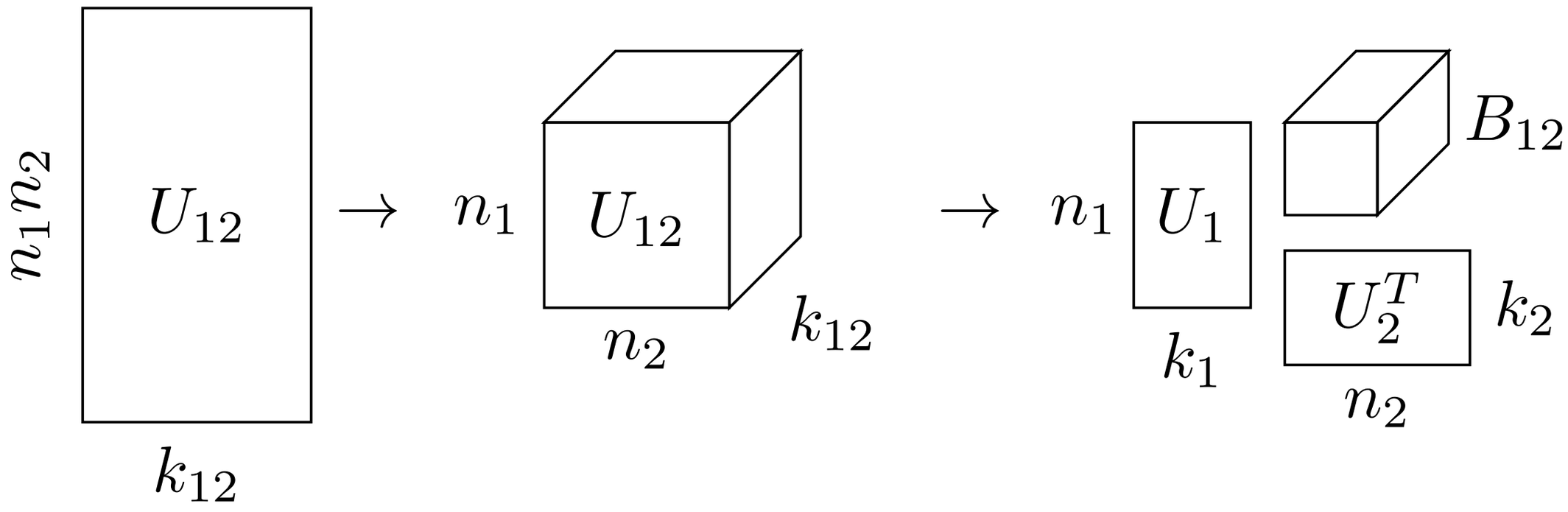


Example $X - n_1 \times n_2 \times n_3 \times n_4$ tensor



$$X^{(1,2)} = U_{12} B^T_{1234} U^T_{34}$$

$$U_{12} = (U_1 \otimes U_2) B_{12}$$

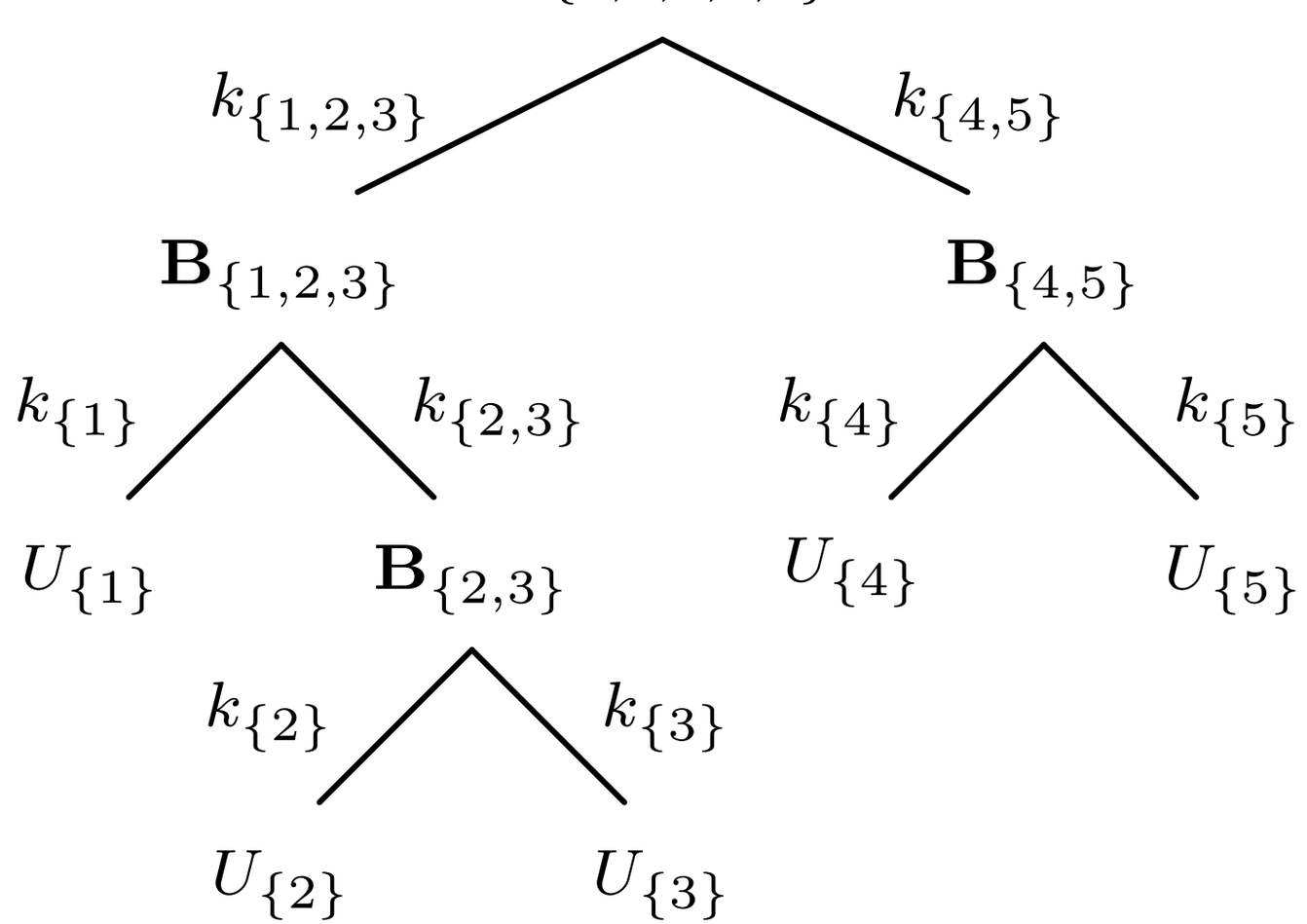


A. Uschmajew, B. Vandereycken. The geometry of algorithms using hierarchical tensors. 2012

Example

$$\mathrm{vec}(X) = (U_{\{1,2,3\}} \otimes U_{\{4,5\}})\mathbf{B}_{\{1,2,3,4,5\}}$$

$$k_{t_r} = 1$$



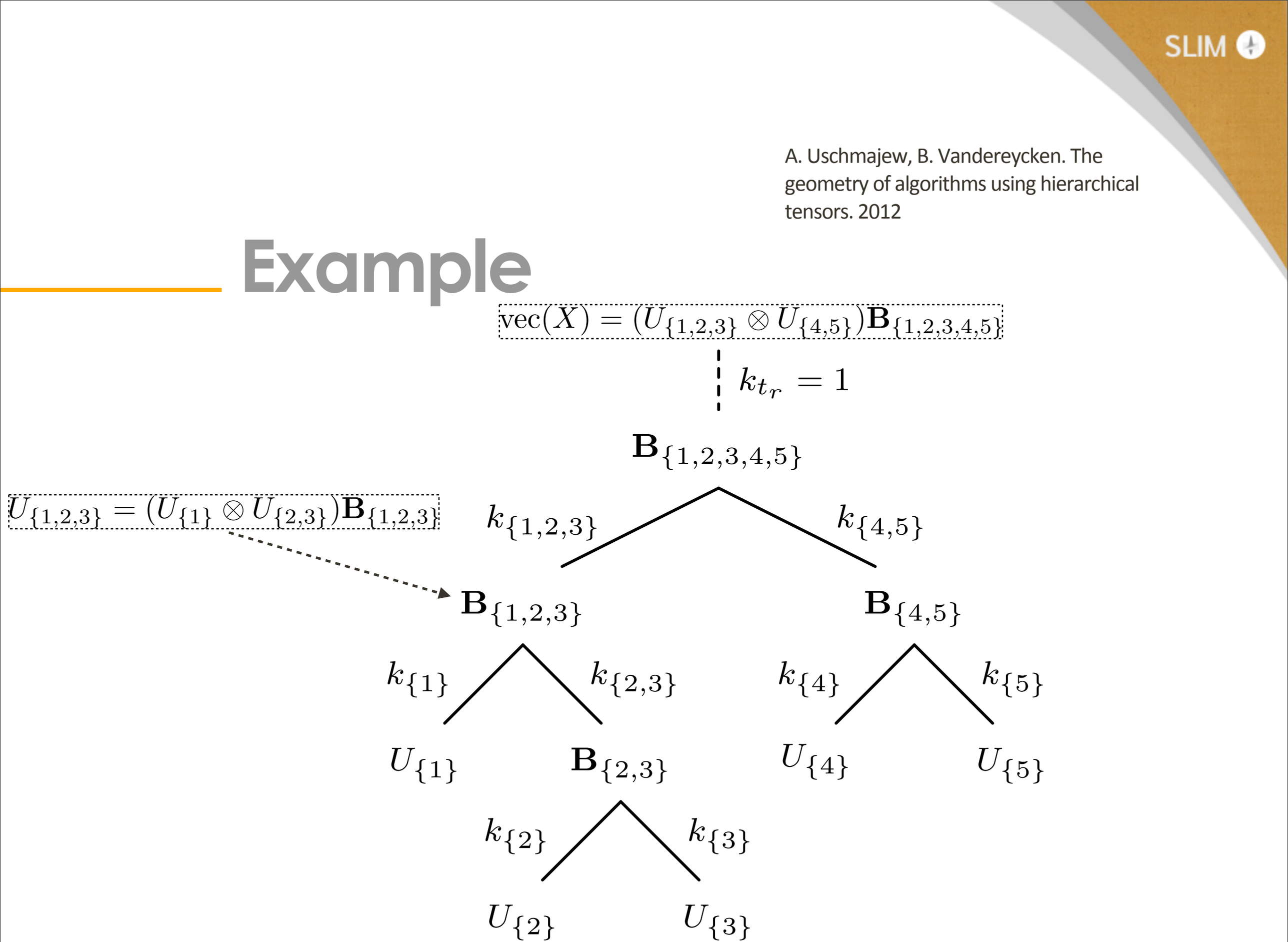
A. Uschmajew, B. Vandereycken. The geometry of algorithms using hierarchical tensors. 2012

Example

$$\mathrm{vec}(X) = (U_{\{1,2,3\}} \otimes U_{\{4,5\}})\mathbf{B}_{\{1,2,3,4,5\}}$$

$$k_{t_r} = 1$$

$$U_{\{1,2,3\}} = (U_{\{1\}} \otimes U_{\{2,3\}})\mathbf{B}_{\{1,2,3\}}$$



A. Uschmajew, B. Vandereycken. The geometry of algorithms using hierarchical tensors. 2012

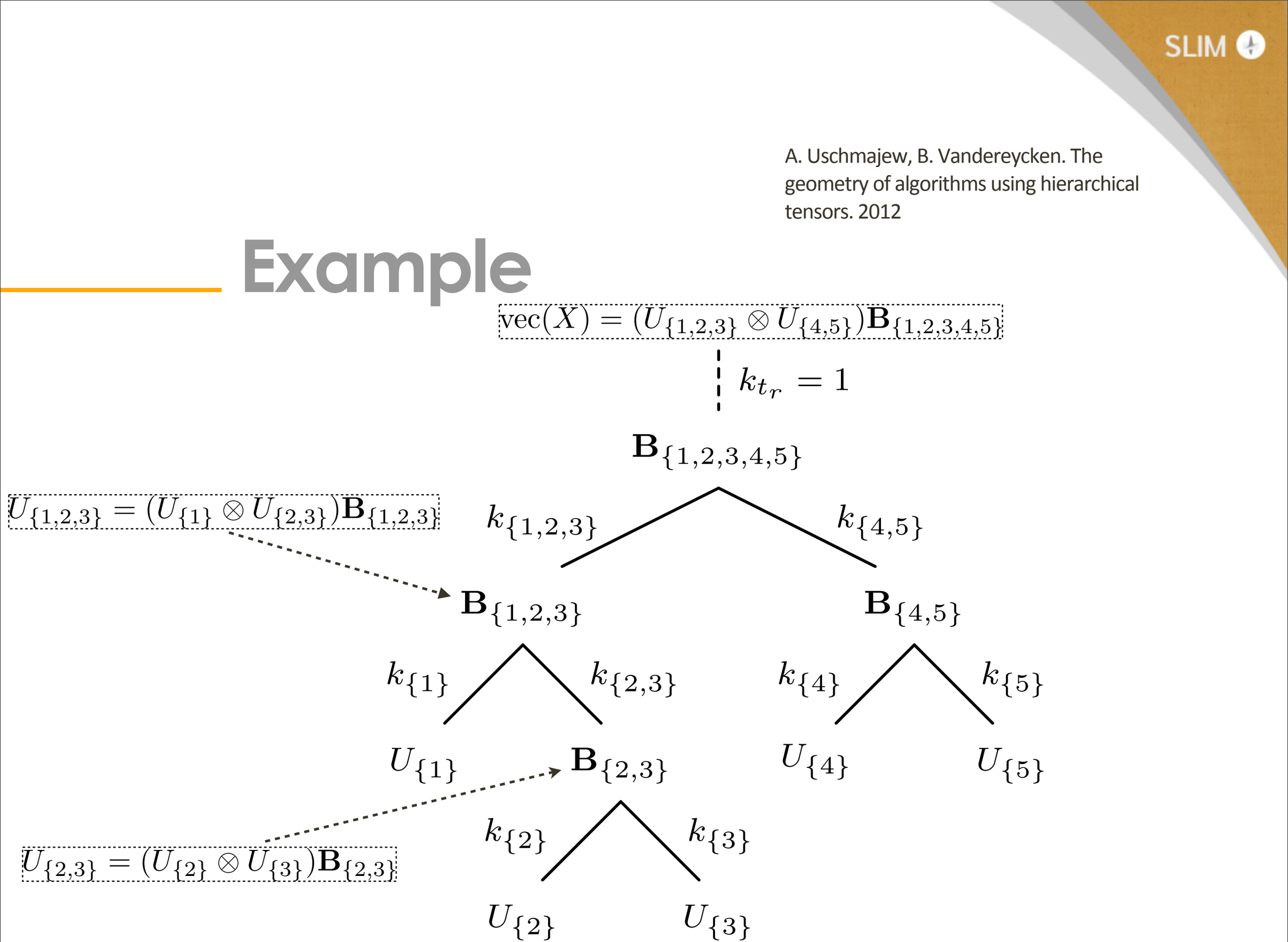
Example

$$\mathrm{vec}(X) = (U_{\{1,2,3\}} \otimes U_{\{4,5\}})\mathbf{B}_{\{1,2,3,4,5\}}$$

$$k_{t_r} = 1$$

$$\mathbf{B}_{\{1,2,3,4,5\}}$$

$$U_{\{1,2,3\}} = (U_{\{1\}} \otimes U_{\{2,3\}})\mathbf{B}_{\{1,2,3\}}$$



$$U_{\{2,3\}} = (U_{\{2\}} \otimes U_{\{3\}})\mathbf{B}_{\{2,3\}}$$

A. Uschmajew, B. Vandereycken. The geometry of algorithms using hierarchical tensors. 2012

Example

$$\mathrm{vec}(X) = (U_{\{1,2,3\}} \otimes U_{\{4,5\}})\mathbf{B}_{\{1,2,3,4,5\}}$$

$$k_{t_r} = 1$$

$$\mathbf{B}_{\{1,2,3,4,5\}} \qquad U_{\{4,5\}} = (U_{\{4\}} \otimes U_{\{5\}})\mathbf{B}_{\{4,5\}}$$

$$U_{\{1,2,3\}} = (U_{\{1\}} \otimes U_{\{2,3\}})\mathbf{B}_{\{1,2,3\}}$$

$k_{\{1,2,3\}}$ $k_{\{4,5\}}$

$\mathbf{B}_{\{1,2,3\}}$ $\mathbf{B}_{\{4,5\}}$

$k_{\{1\}}$ $k_{\{2,3\}}$ $k_{\{4\}}$ $k_{\{5\}}$

$U_{\{1\}}$ $\mathbf{B}_{\{2,3\}}$ $U_{\{4\}}$ $U_{\{5\}}$

$$U_{\{2,3\}} = (U_{\{2\}} \otimes U_{\{3\}})\mathbf{B}_{\{2,3\}}$$

$k_{\{2\}}$ $k_{\{3\}}$

$U_{\{2\}}$ $U_{\{3\}}$

Hierarchical Tucker Format

- We don't need to store the matrices U_t when t is not a leaf node
  - We only need to store U_t for the leaves, B_t for the internal nodes
  - We don't need to store the (full) tensor

Hierarchical Tucker Format

- Storage $\leq dNK + (d-2)K^3 + K^2$

  where $K = \max_{t \in T} k_t$, $N = \max_{i=1,\ldots,d} n_i$

- Compare to N^d storage for the full tensor
- Effectively breaking the curse of dimensionality when $K \ll N$

Hierarchical Tucker Format

- E.g. for a 100 x 100 x 100 x 100 cube with max rank 20, $N = 100 \quad d = 4 \quad k = 20$

  Naive storage: $N^d = 10^8$ parameters

  HTucker storage:

  $$dNK + (d-2)K^3 + K^2$$

  $$= 24400 \text{ parameters}$$

  Compression of a factor of 99.97%

www.sam.math.ethz.ch/NLAgroup/
htucker_toolbox.html

Hierarchical Tucker Format

- Existing tools for using this format

hTucker toolbox

- Matlab code for performing operations in this framework
- truncation, addition/subtraction, tensor-tensor contraction, etc.

Hierarchical Tucker Format

- Truncation of a full tensor X to a rank-k hTucker tensor requires knowledge of *all* of the entries of X
- Requires SVDs of underlying matricizations, which are expensive
- Problematic in seismic data cases, where not all of the data is available

Hierarchical Tucker Format

- We can use knowledge of the multidimensional structure of, say, a frequency slice of data with missing sources/receivers in order to recover the full data via solving an optimization problem
- Analogous to $l1$ regularized curvelet interpolation in the 2D seismic case

Optimization Format

- Given data D with missing sources and/or receivers, subsampling operator A, full tensor expansion operator

$$\phi : (U_t, B_t) \rightarrow \mathbb{C}^{n_1 \times \cdots \times n_d}$$

solve

$$\min_{(U_t, B_t)} \|A\phi(x) - D\|_2^2$$

Derivatives

- Derivatives of a particular node with respect to its children can be computed efficiently
- The chain rule gives the total derivative of the function ϕ

SLIM

Derivatives

- Messy painful derivation, available on request
- After much simplification, boils down to about 20 lines of matlab code, using the SPOT framework + code from the hTucker toolbox

Derivative code

```
function [dUdL,dUdR,dUdB] = dHTuck(Uleft, Uright, B)

    [kr,kl,k] = size(B);
    nl = size(Uleft,1); nr = size(Uright,1);
    Bhat = matricize(B,1,[2 3]);
    Bhat = dematricize(Uright * Bhat,[nr,kl,k],1,[2,3]);

    Bhat = matricize(Bhat,[1 3],2);
    dUdL = opFunction(nl*nr*k, nl*kl, @(x,mode) dUdL_func(x,mode, Bhat,nr,nl,k,kl));
    BhatR = dematricize( Uleft*matricize(permute(B,[2 1 3]),1, [2 3]), [nl,kr,k],1,[2 3]);
    dUdR = opKron(matricize(BhatR,[1 3],2),opDirac(nr));
    dUdB = opKron(opDirac(k),Uleft,Uright);

end

function y = dUdL_func(x,mode,Bhat,nr,nl,k,kl)
    if mode == 1
        %Forward mode
        v = reshape(x,nl,kl);
        q = Bhat * v';

        result = reshape(q,nr,k,nl);
        result = permute(result,[1 3 2]);
        y = vec(result);
    else
        %Adjoint mode
        v = reshape(x,nr,nl,k);
        q = Bhat' * matricize(v,[1 3],2);

        %Block transposition
        q = reshape(q,kl,nl);
        y = vec(q');
    end
end
```

Derivatives

- Only involve matrix-matrix multiplication of small matrices compared to the ambient, full-tensor space
- Immediately parallelizable
  - Replace “opKron” with “oppKron2Lo”

Differential Geometry

- The function

$$\phi : (U_t, B_t) \rightarrow \mathbb{C}^{n_1 \times \cdots \times n_d}$$

  is not injective
- There are sets of distinct matrices $(U_t, B_t), (V_t, C_t)$ where $\phi(U_t, B_t) = \phi(V_t, C_t)$

Differential Geometry

- This non-uniqueness (somewhat artificially) leads to *multiple minima* in the optimization program
- Can be resolved by characterizing the differential geometric structure of this format
- This gives us a *smooth manifold* (i.e. curved surface)

[1] A. Uschmajew, B. Vandereycken. The geometry of algorithms using hierarchical tensors. 2012

Differential Geometry

- The authors in [1] study the non-uniqueness of the general, non-orthogonalized hTucker format
- Characterization of the horizontal space (tangent vectors which change ϕ) and the vertical space (tangent vectors which do not change ϕ)

Differential Geometry

- We restrict ourselves to the *orthogonalized Hierarchical Tucker format*
- The matrices (U_t, B_t) not at the root are constrained to be orthogonal

Differential Geometry

- Different geometric structure, but still within a workable framework
- Projections become much better behaved with these constraints

P.A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton Univ Press, 2008.

Differential Geometry

- With these orthogonality constraints, the standard matrix trace turns this format into a *Riemannian manifold*
- Large amount of existing research dealing with solving optimization problems on Riemannian manifolds
- Standard algorithms such as Steepest Descent translatable into this framework

Sampling

- With 2D volumes, we know several things
  - Low-rank structure can be recovered through solving a rank-minimizing optimization problem
  - Random removal of points increases the rank of the underlying matrix, which favours recovery by rank-minimization

Sampling

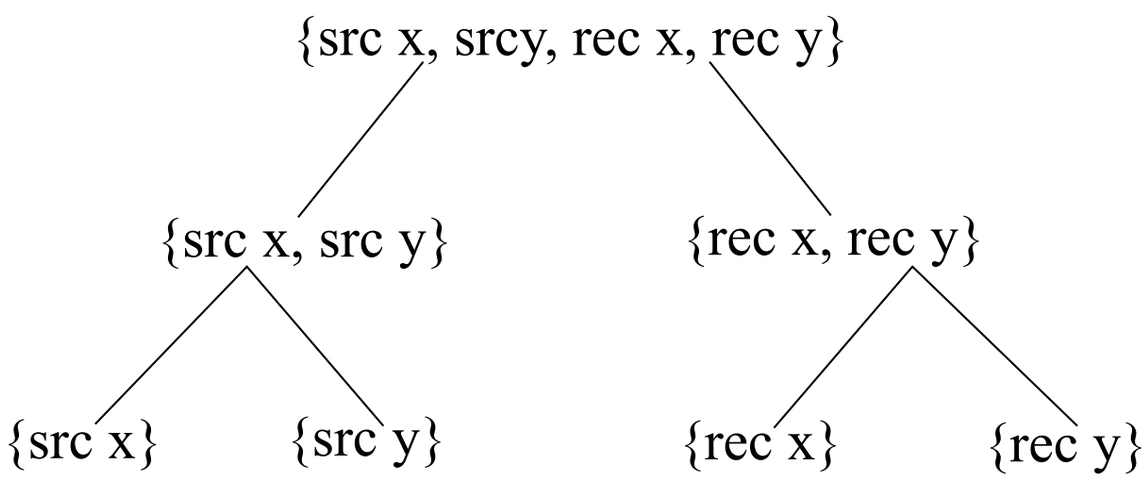
- In higher dimensions, there are multiple ways to ‘separate’ different dimensions
- No unique notion of rank, as in the matrix case

Sampling - Seismic Example

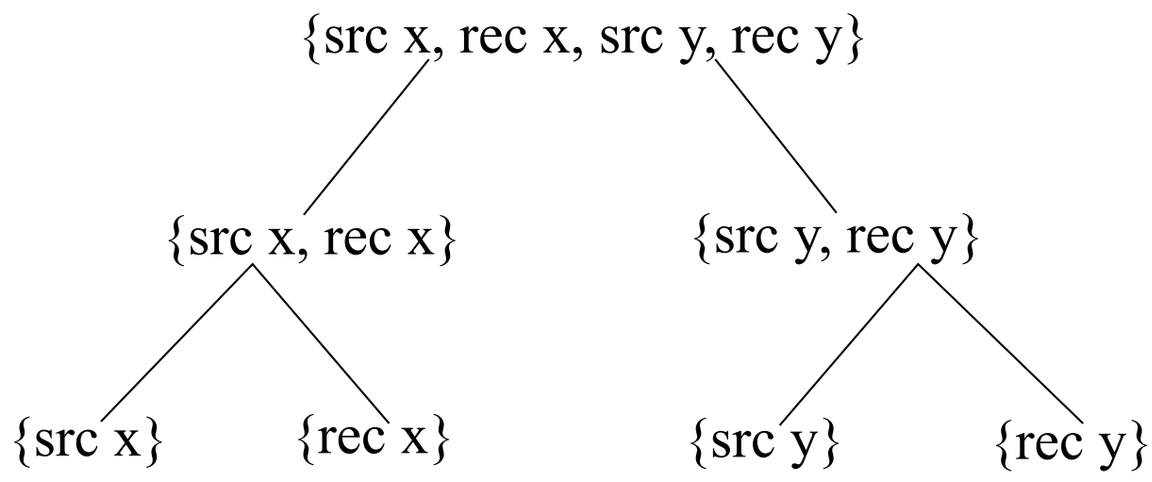
- We consider a 3D seismic survey with coordinates (src x, src y, rec x, rec y, time)
- We take a Fourier transform in time and restrict ourselves to a single frequency slice

Sampling

- For a frequency slice with coordinates (src x, src y, rec x, rec y), there are essentially two choices of dimension tree (by reciprocity)



Canonical Decomposition

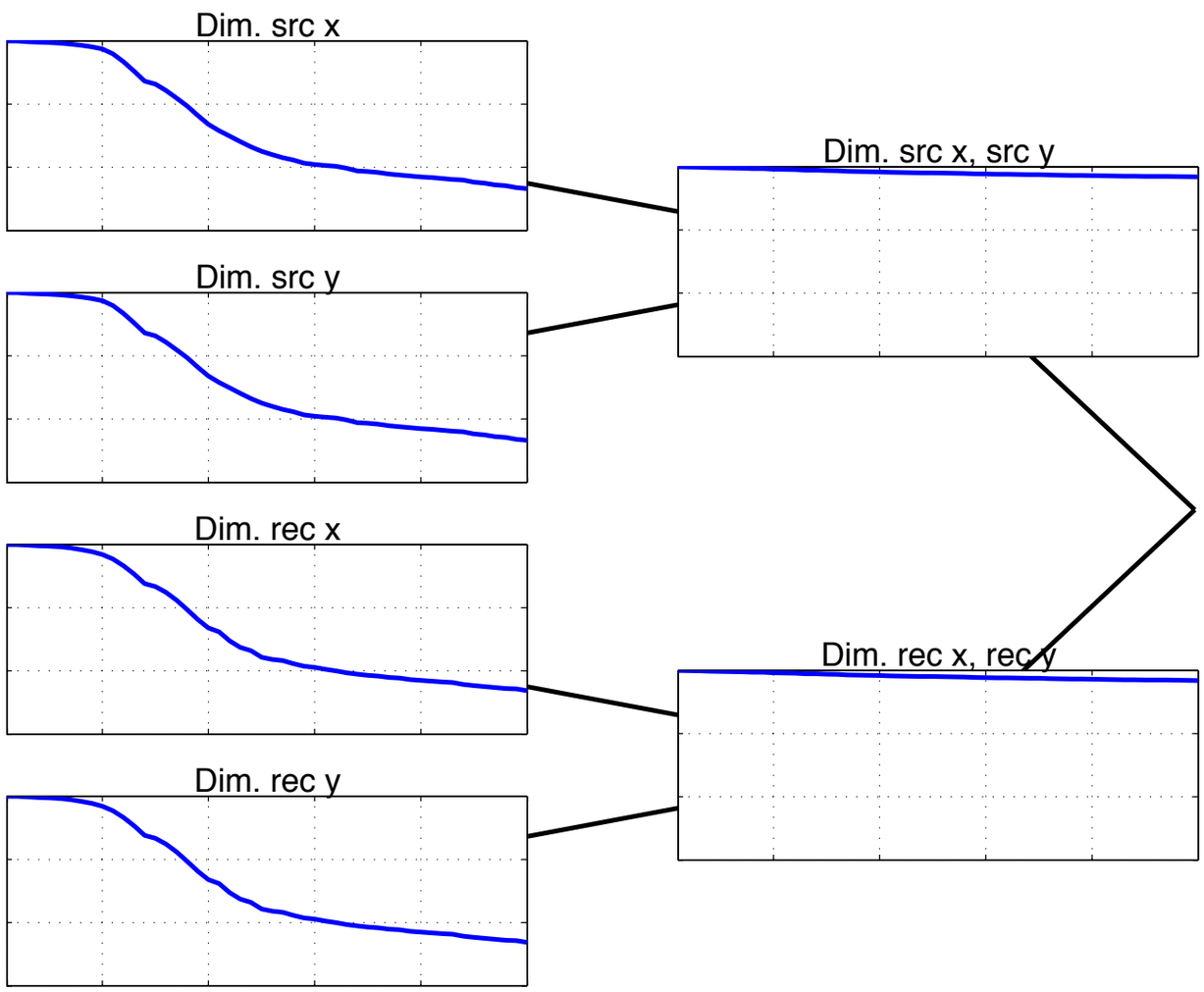


Non-canonical Decomposition

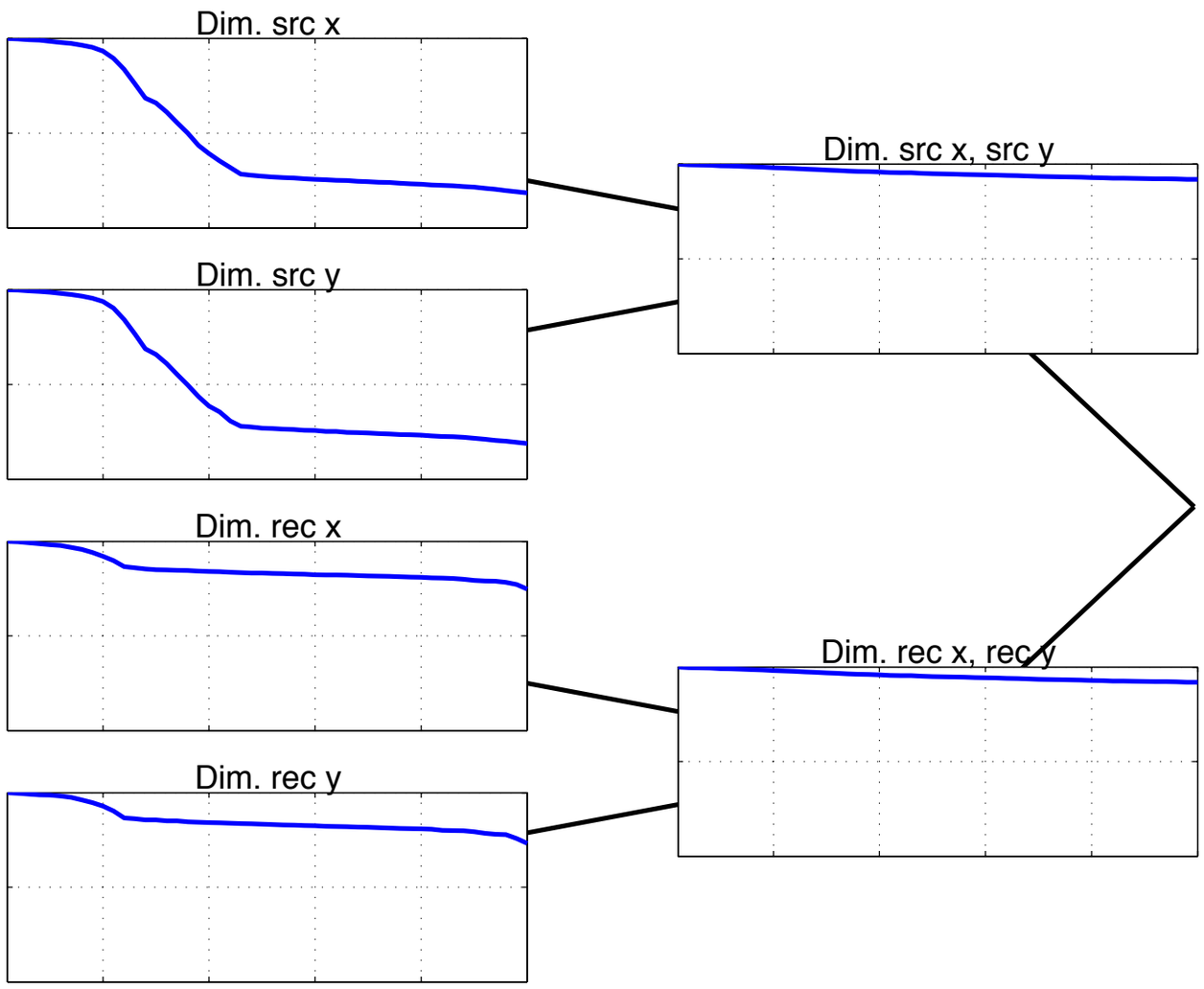
Effects of sampling

25% random receiver pairs removed

- (src x, src y), (rec x, rec y) pairing



No subsampling

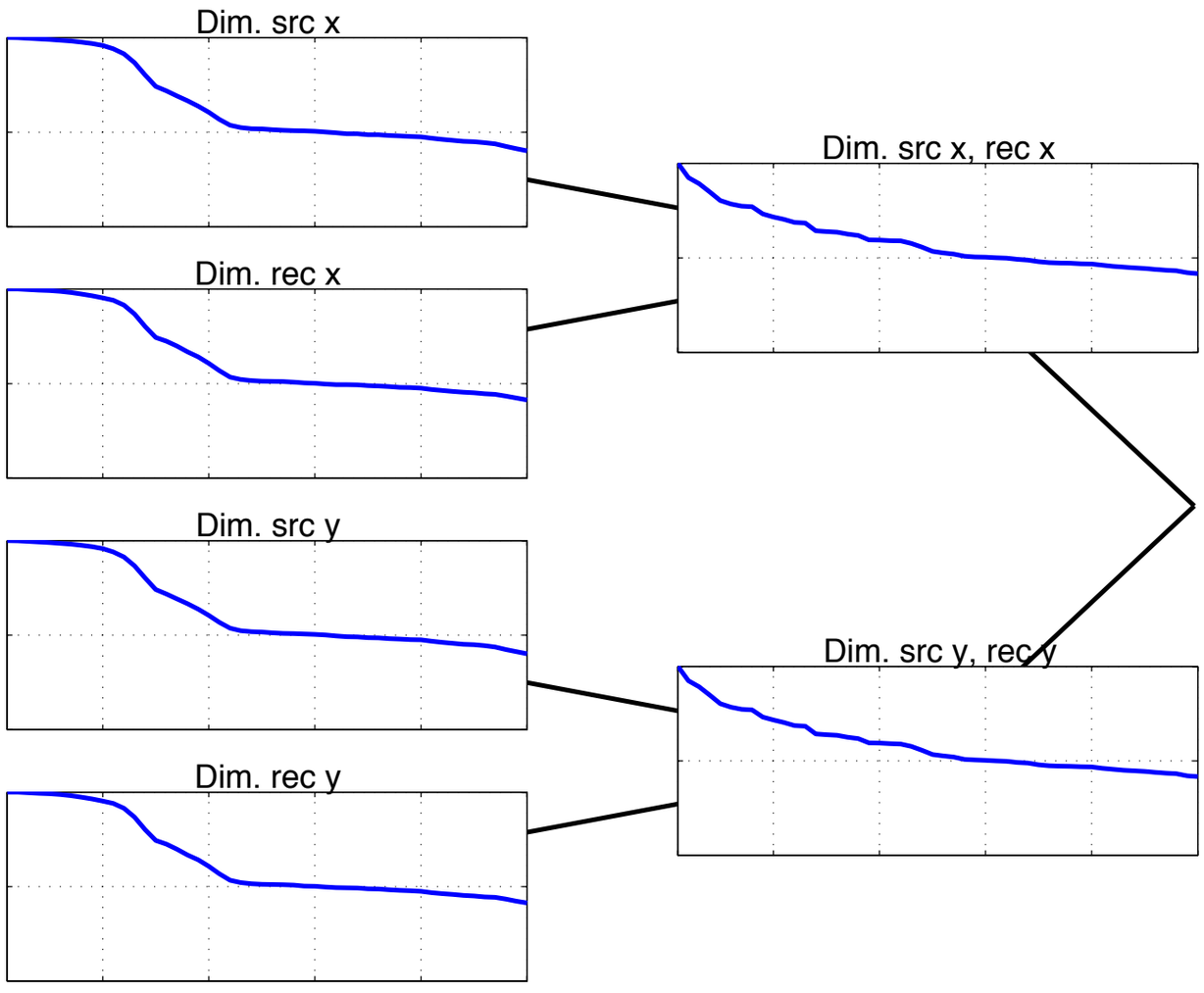


Rec subsampling

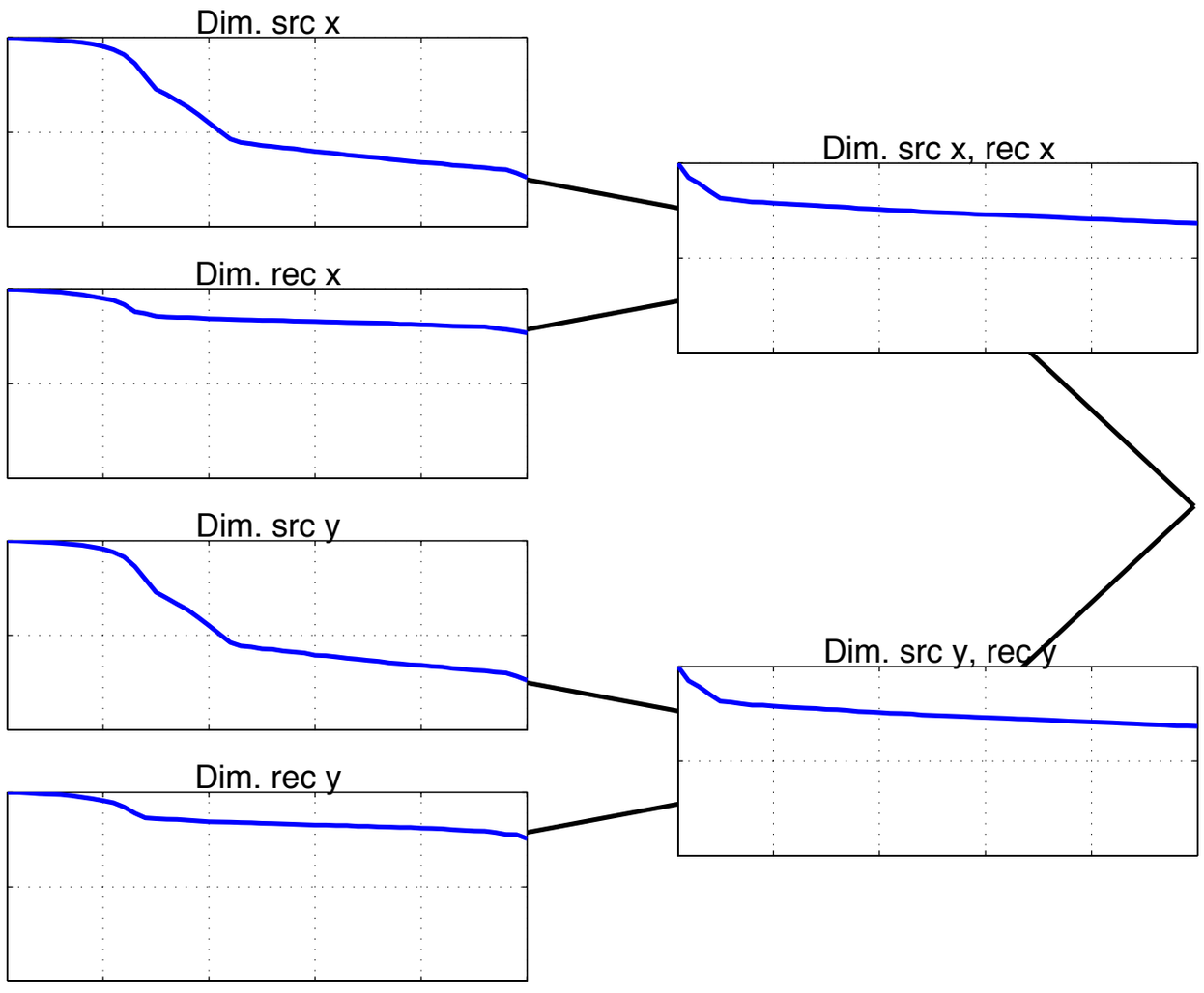
Effects of sampling

25% random receiver pairs removed

- (src x, rec x), (src y, rec y) pairing



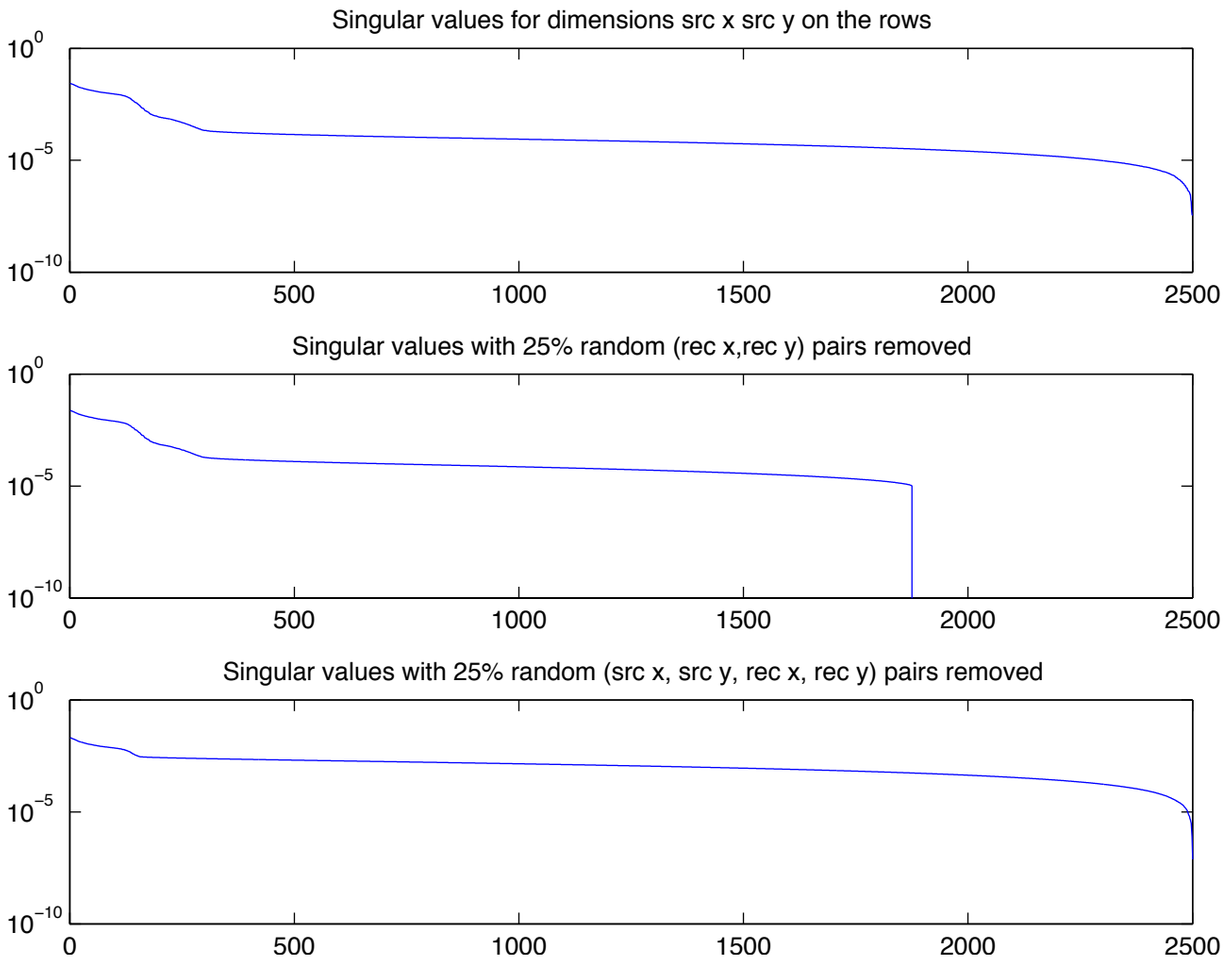
No subsampling



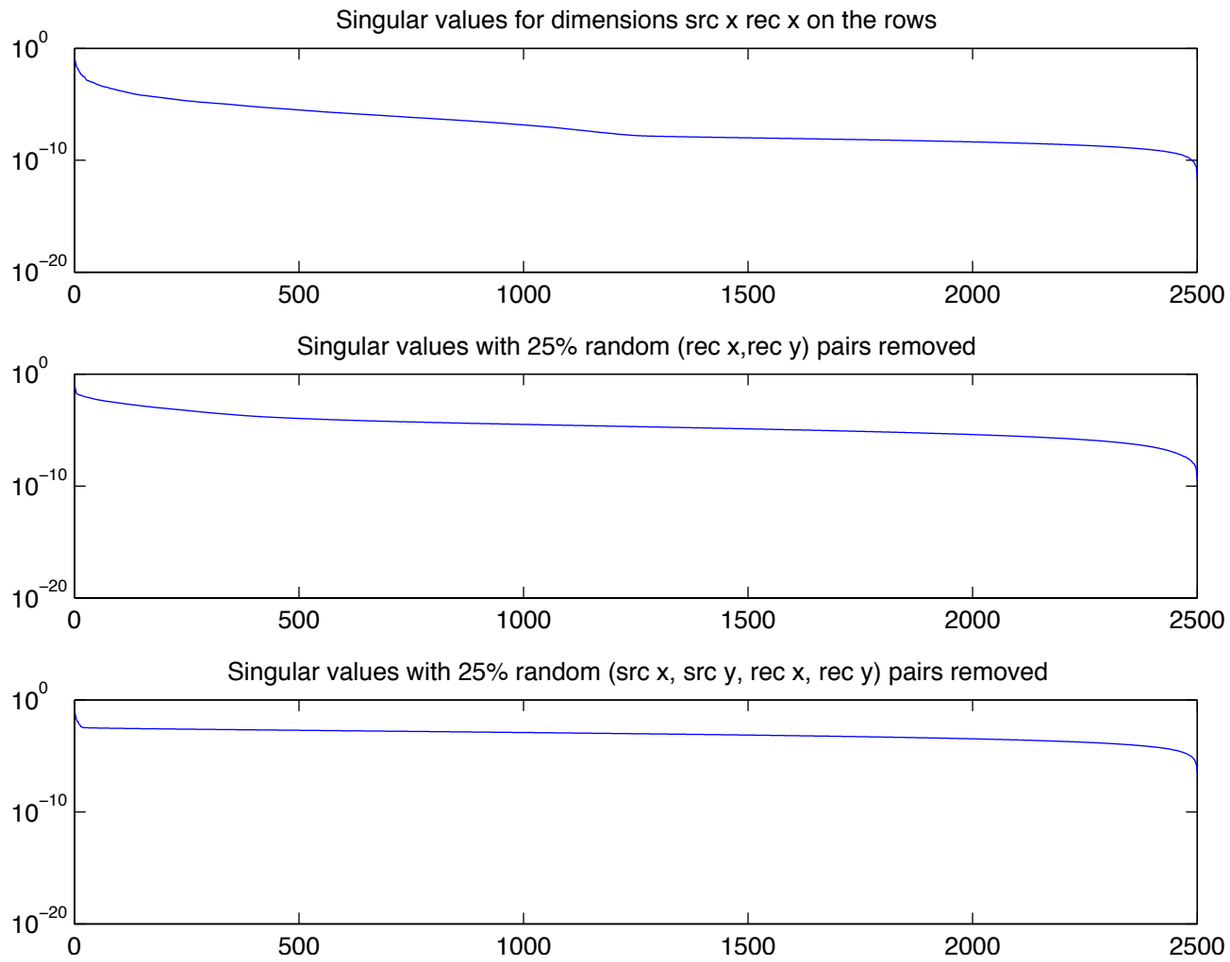
Rec subsampling

Effects of sampling

This phenomenon can also be understood in the context of different matricizations of the tensor

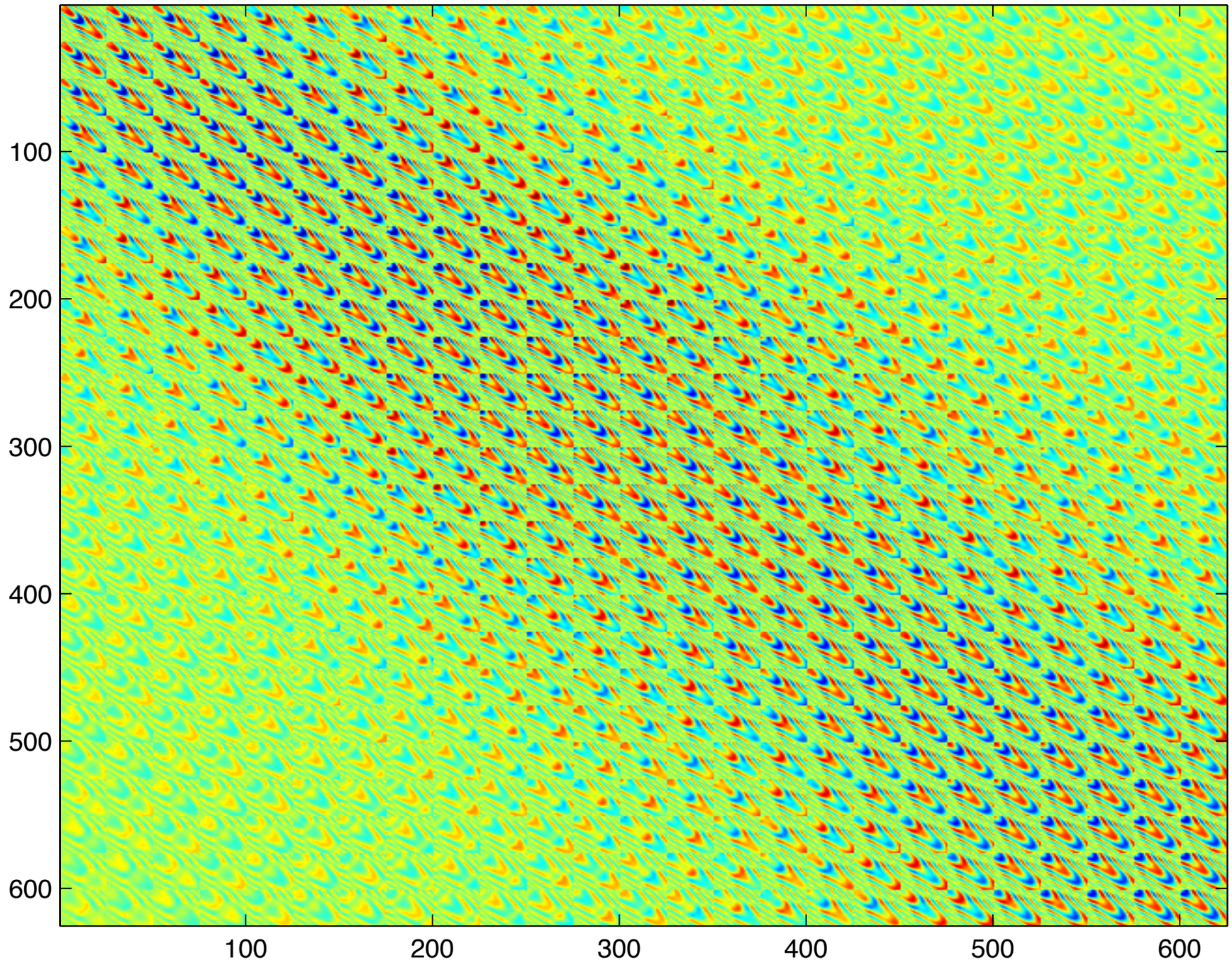


Canonical Grouping



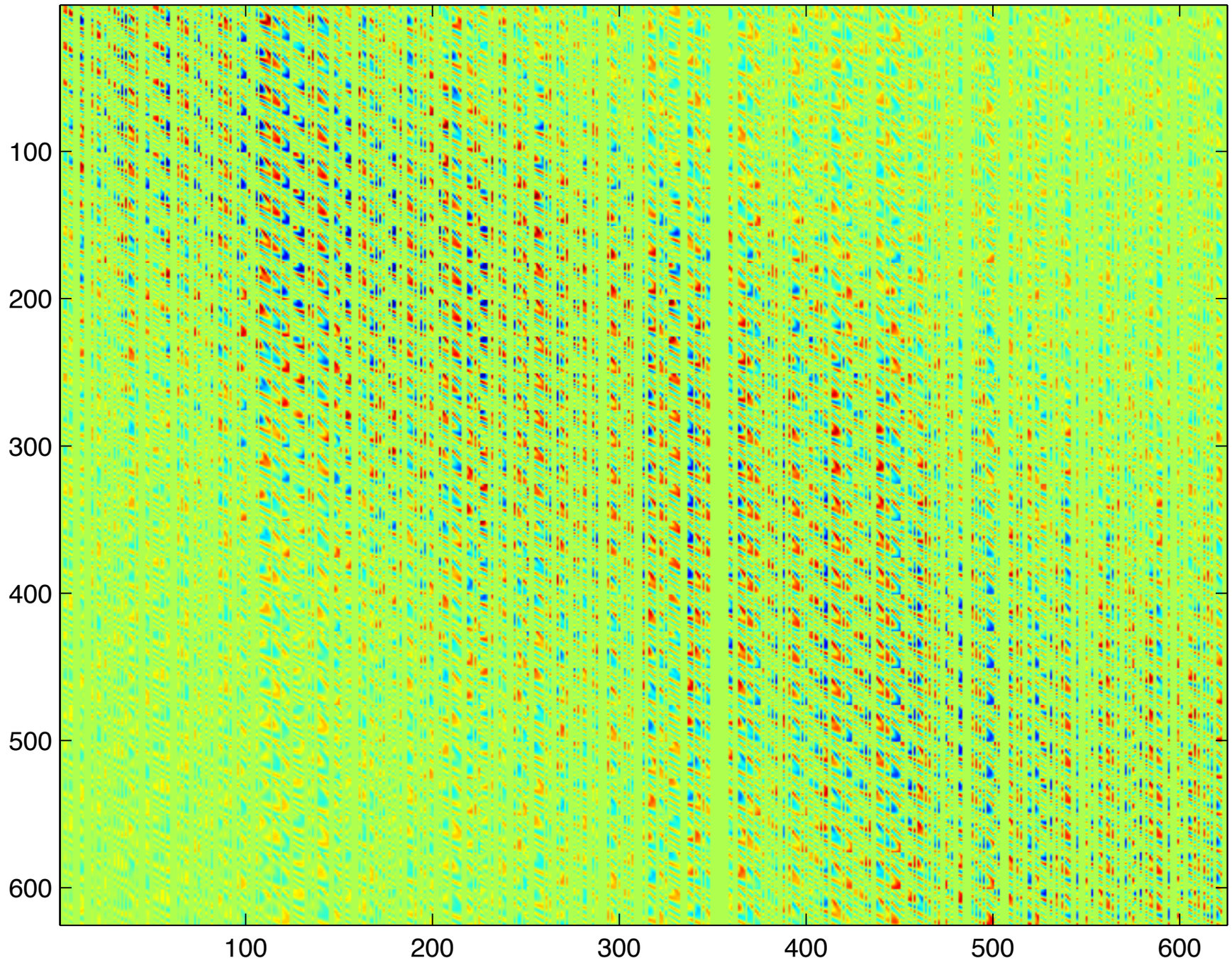
Non-canonical Grouping

Matricizations + Sampling



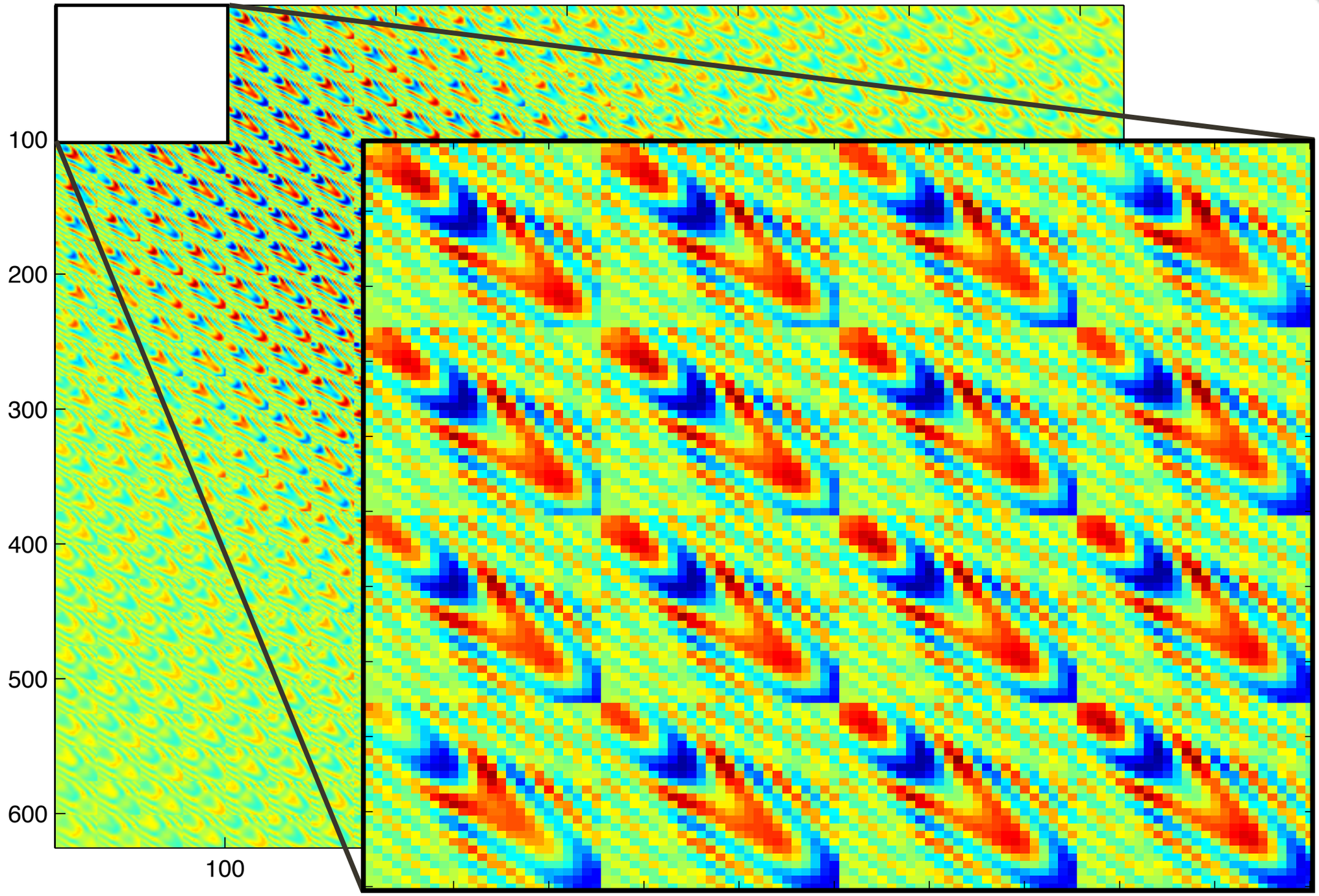
(Src x, Src y) matricization - full data

Matricizations + Sampling



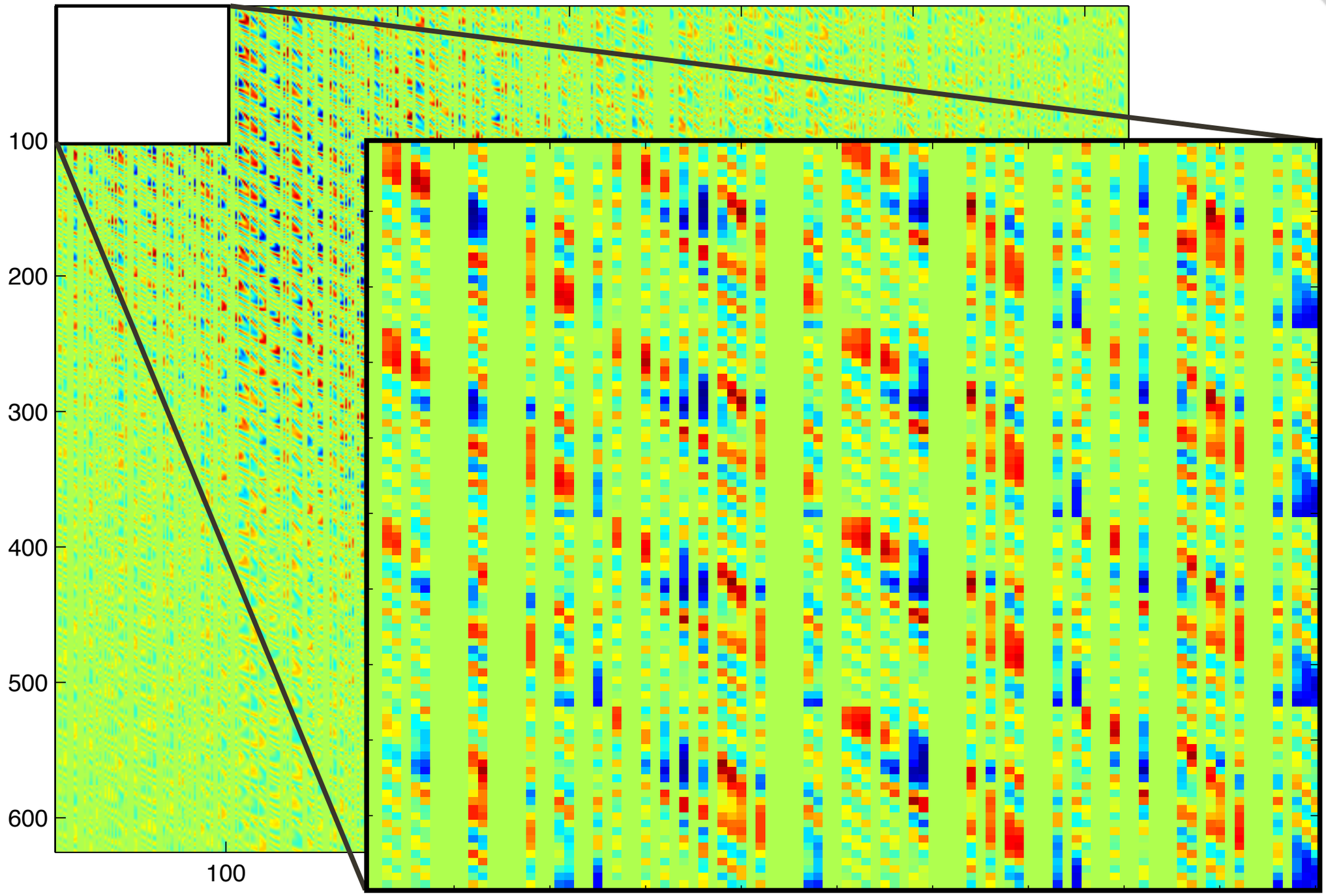
(Src x, Src y) matricization - 50% missing receivers

Matricizations + Sampling



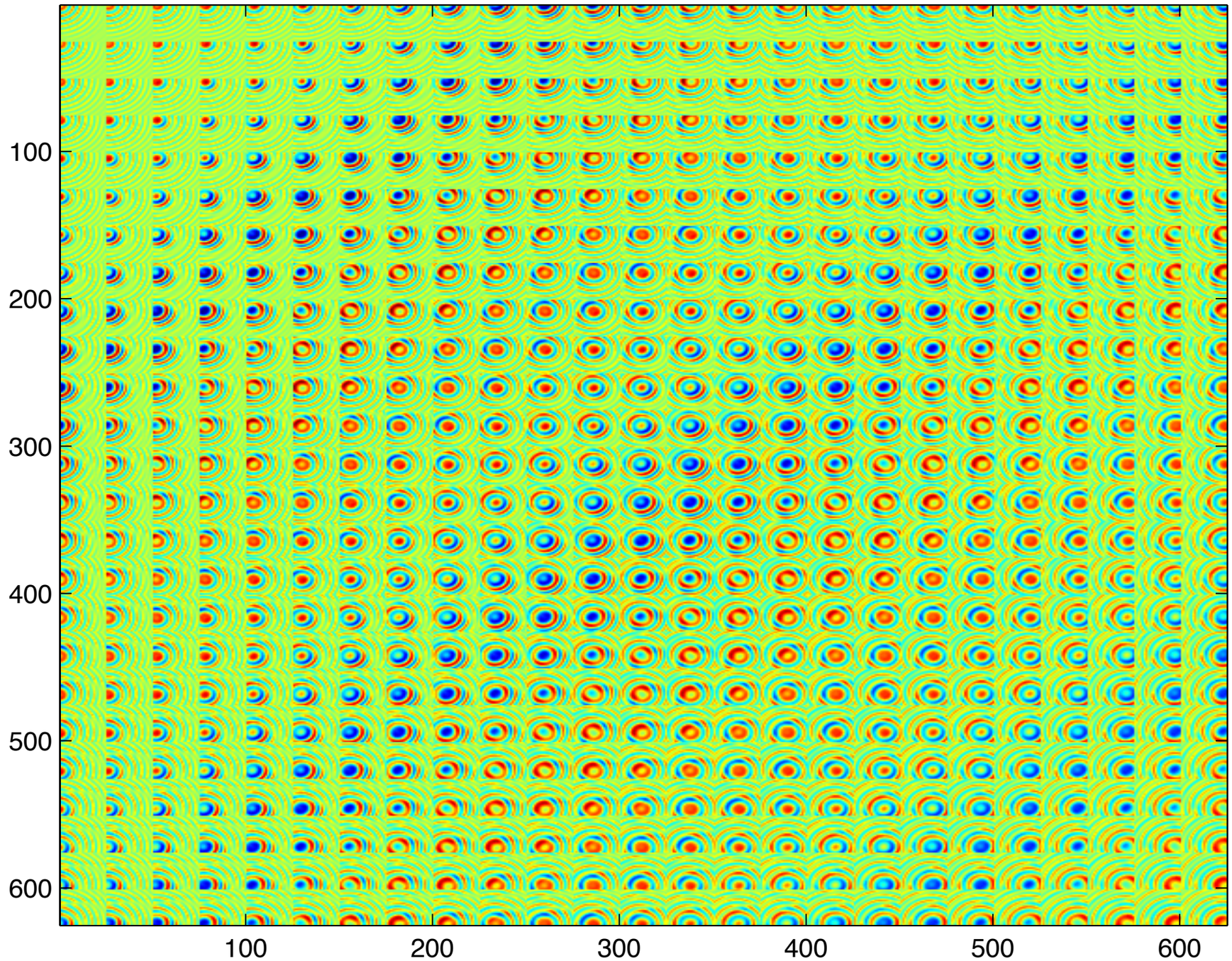
(Src x, Src y) matricization - full data

Matricizations + Sampling



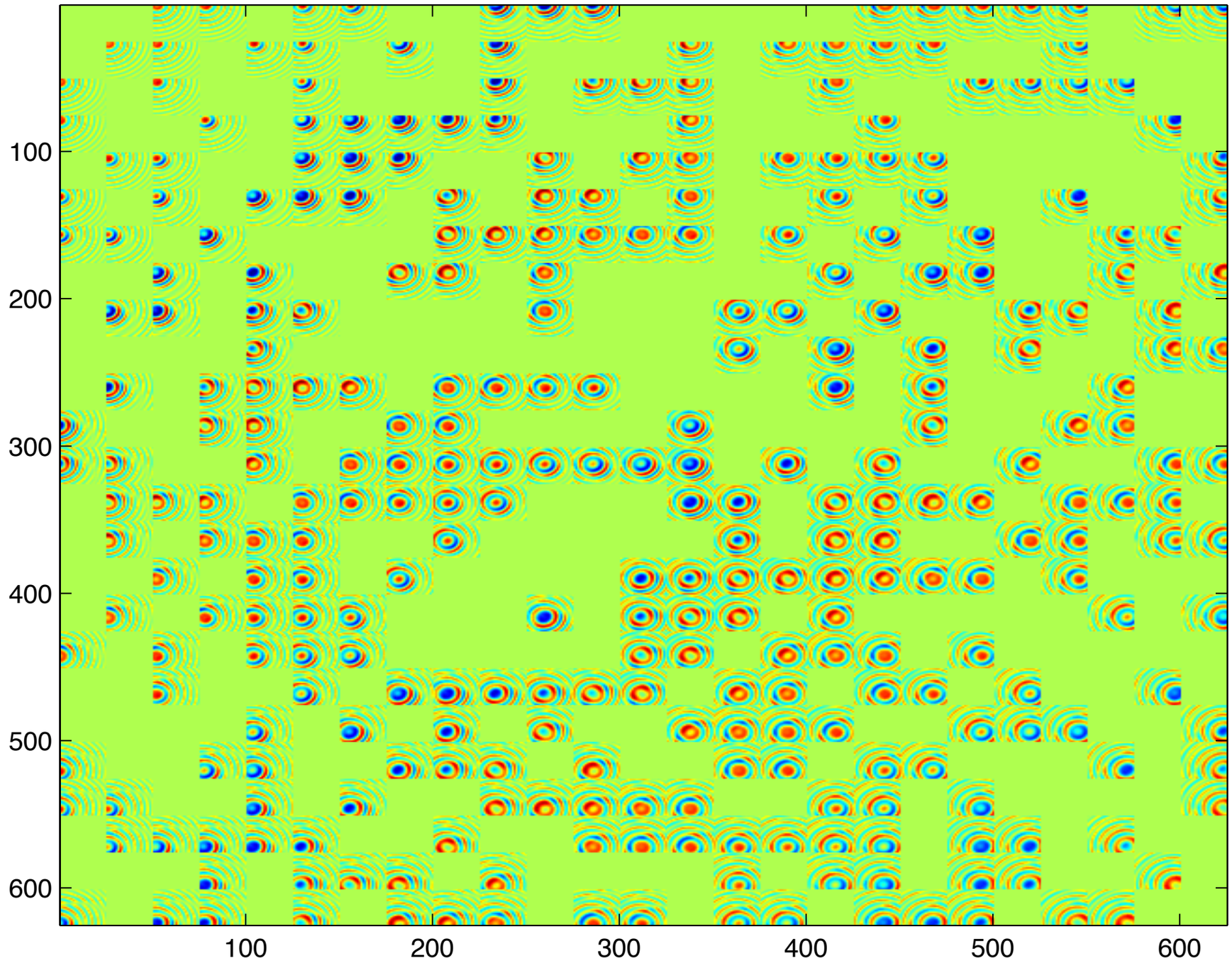
(Src x, Src y) matricization - 50% missing receivers

Matricizations + Sampling



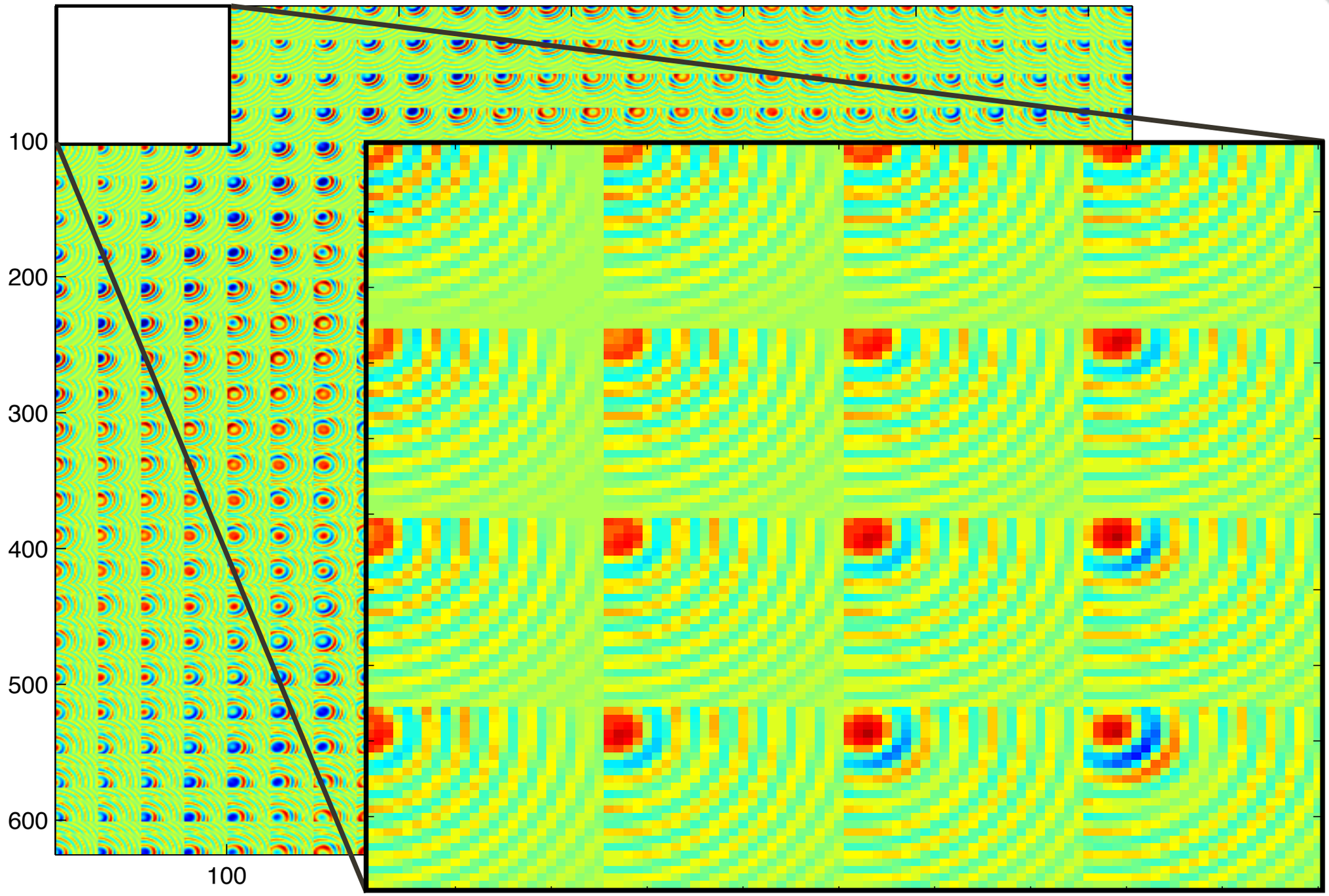
(Src x, Rec x) matricization - full data

Matricizations + Sampling



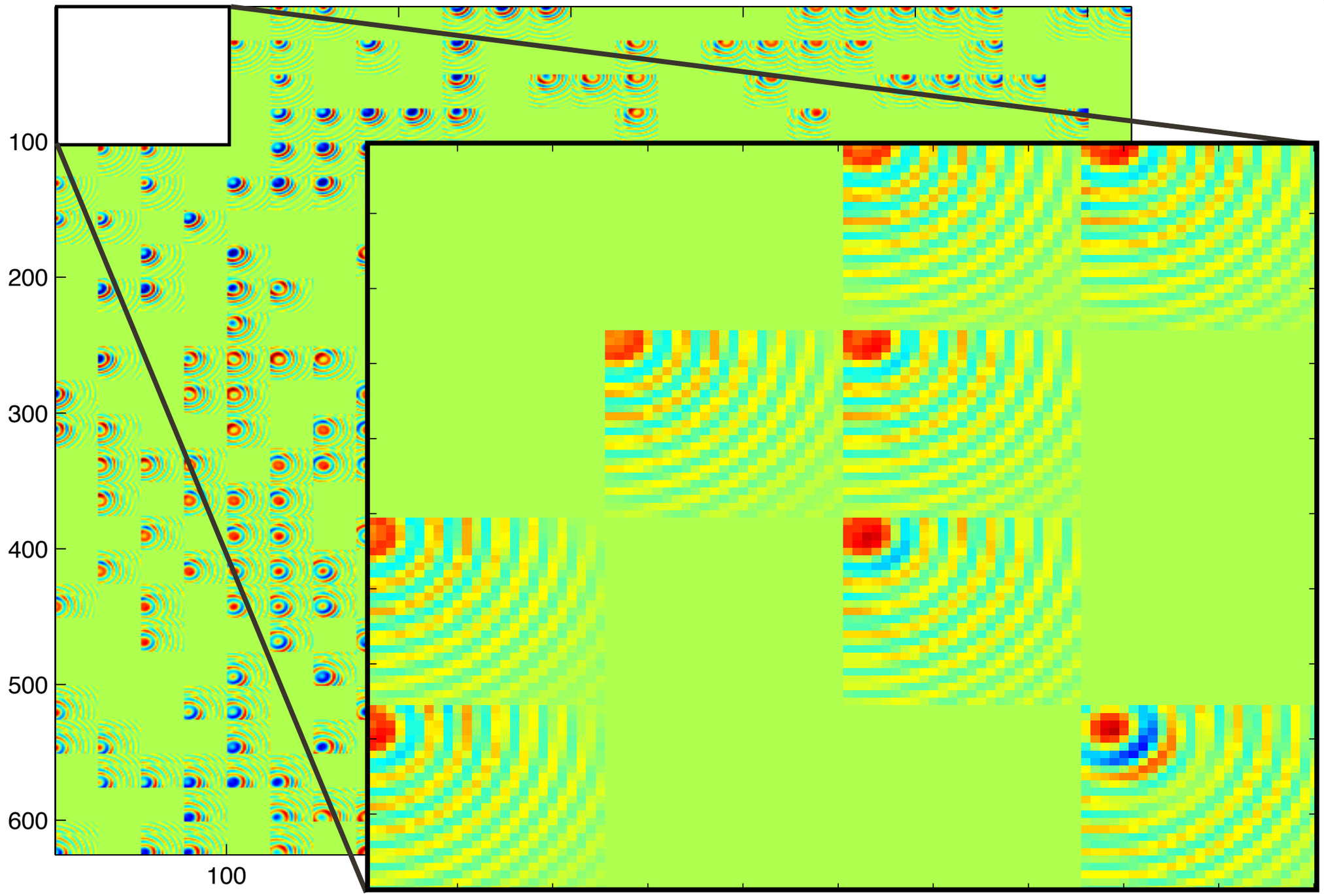
(Src x, Rec x) matricization - 50% missing receivers

Matricizations + Sampling



(Src x, Rec x) matricization - full data

Matricizations + Sampling



(Src x, Rec x) matricization - 50% missing receivers

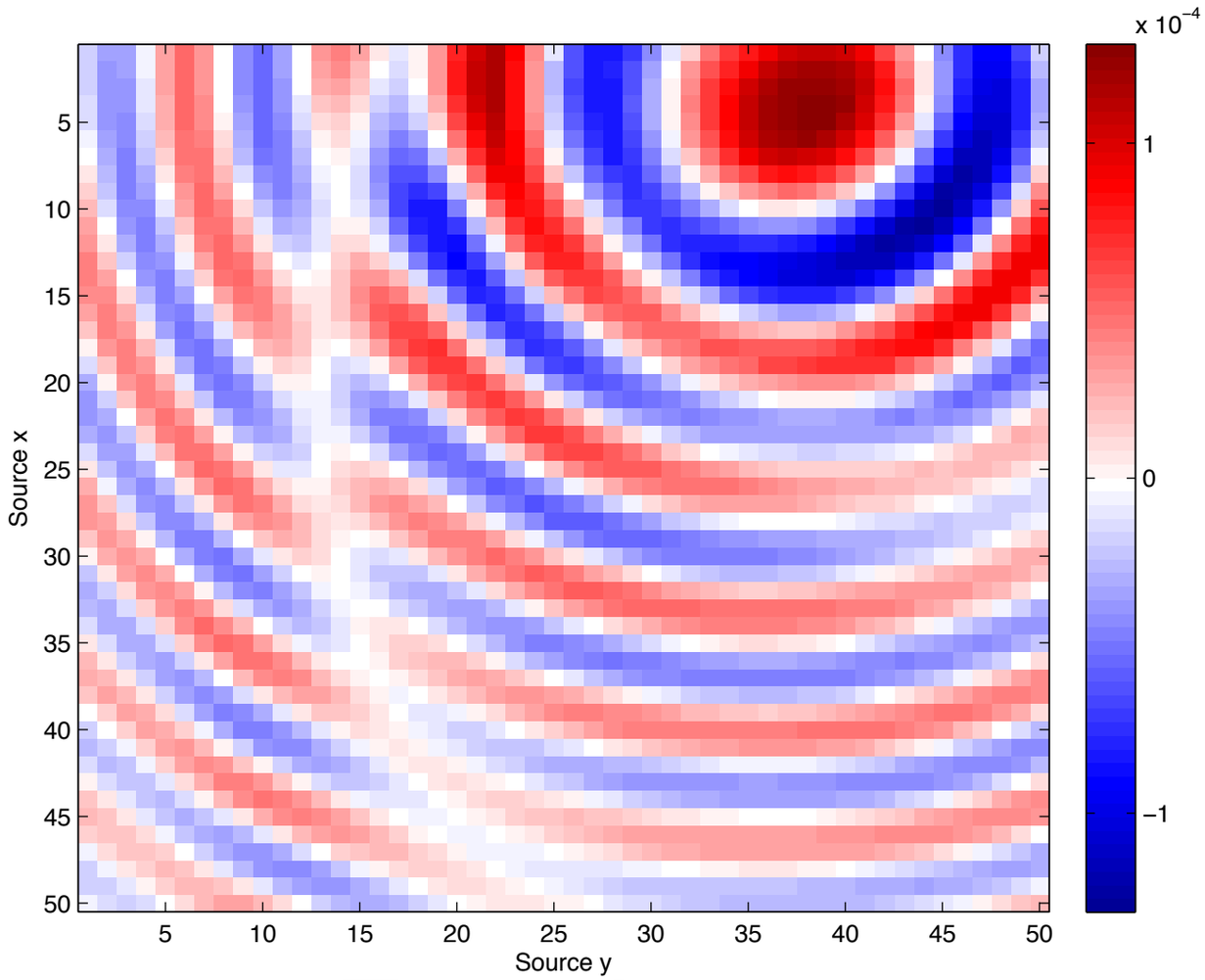
SLIM

Results

Synthetic 4D Data

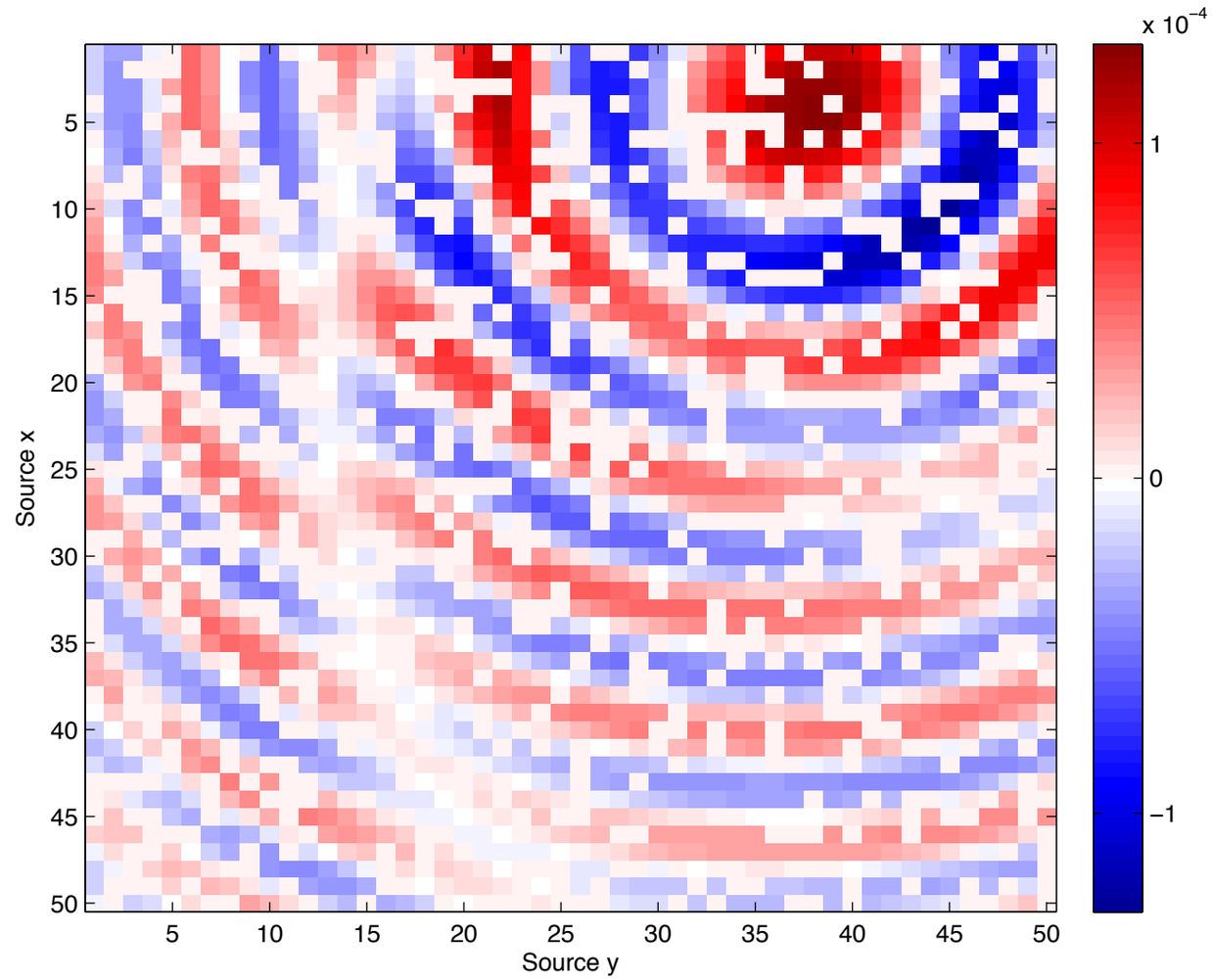
- Single reflector model
- Single frequency slice (real part)
- 50 x 50 sources
- 50 x 50 receivers
- Nonlinear CG in this format - 200 iterations
- Hierarchical rank 20 (internal + individual dimension rank)

25% Missing Source Pairs



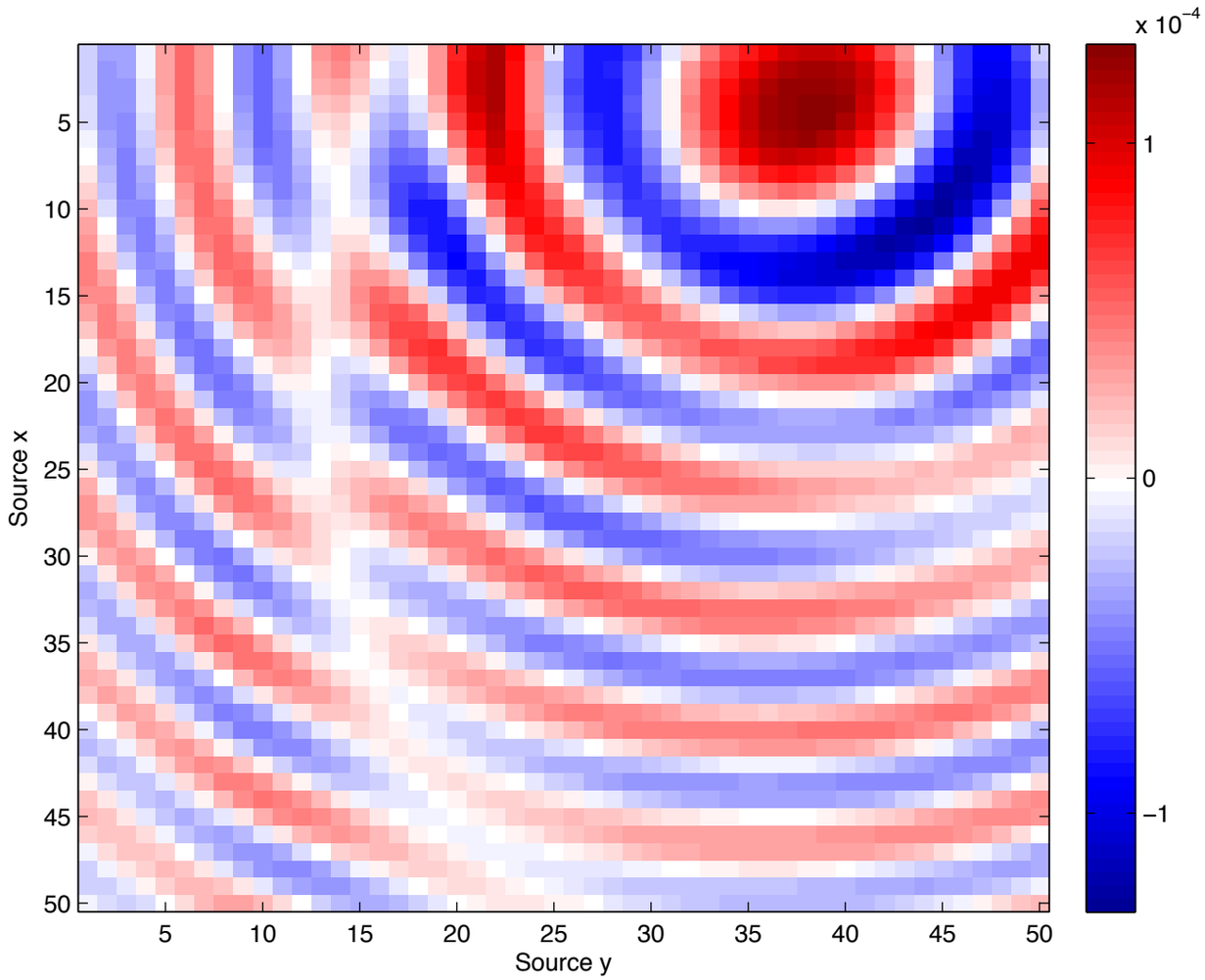
True data

(Rec x, Rec y) = (5,39)



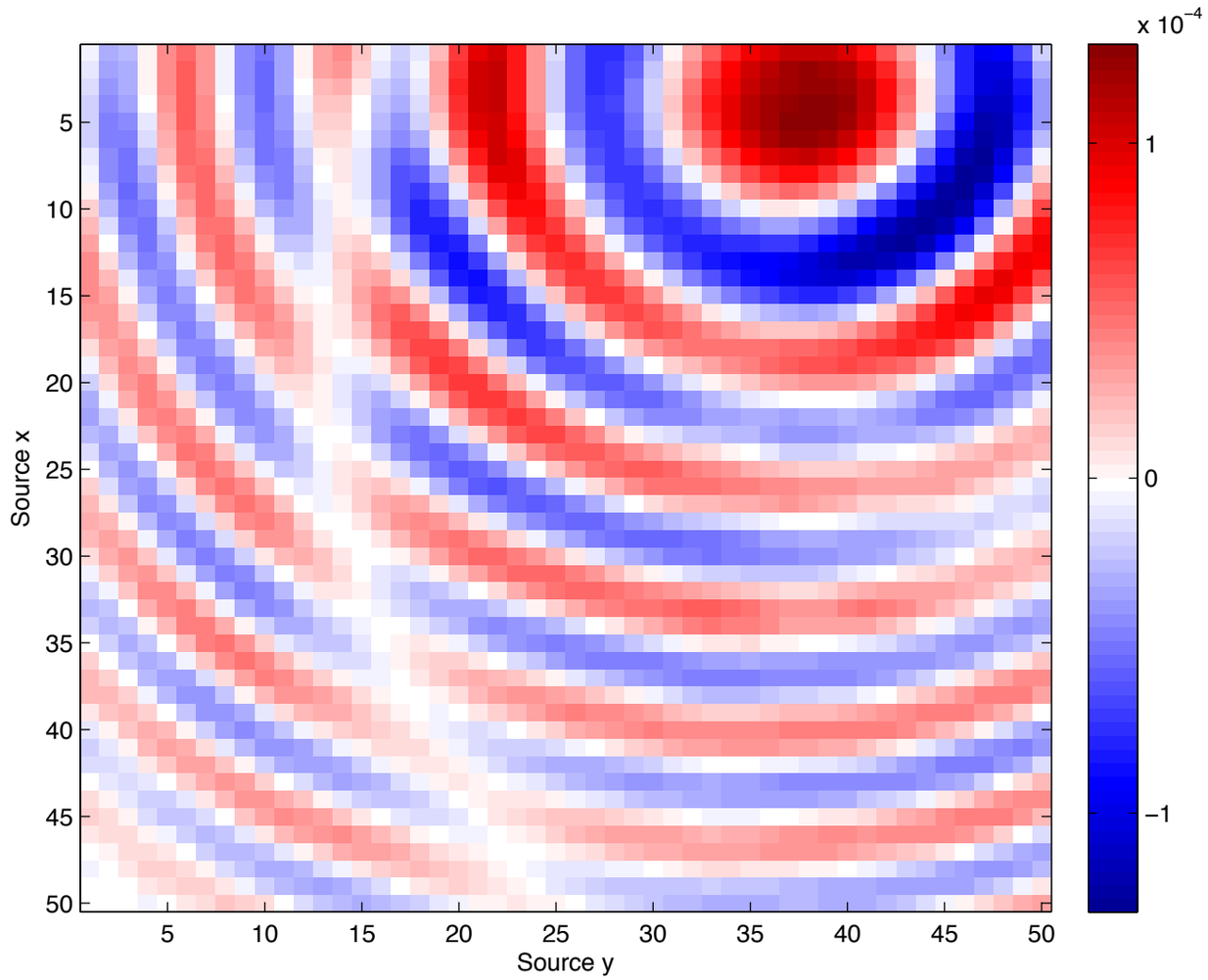
Subsampled data

25% Missing Source Pairs



True data

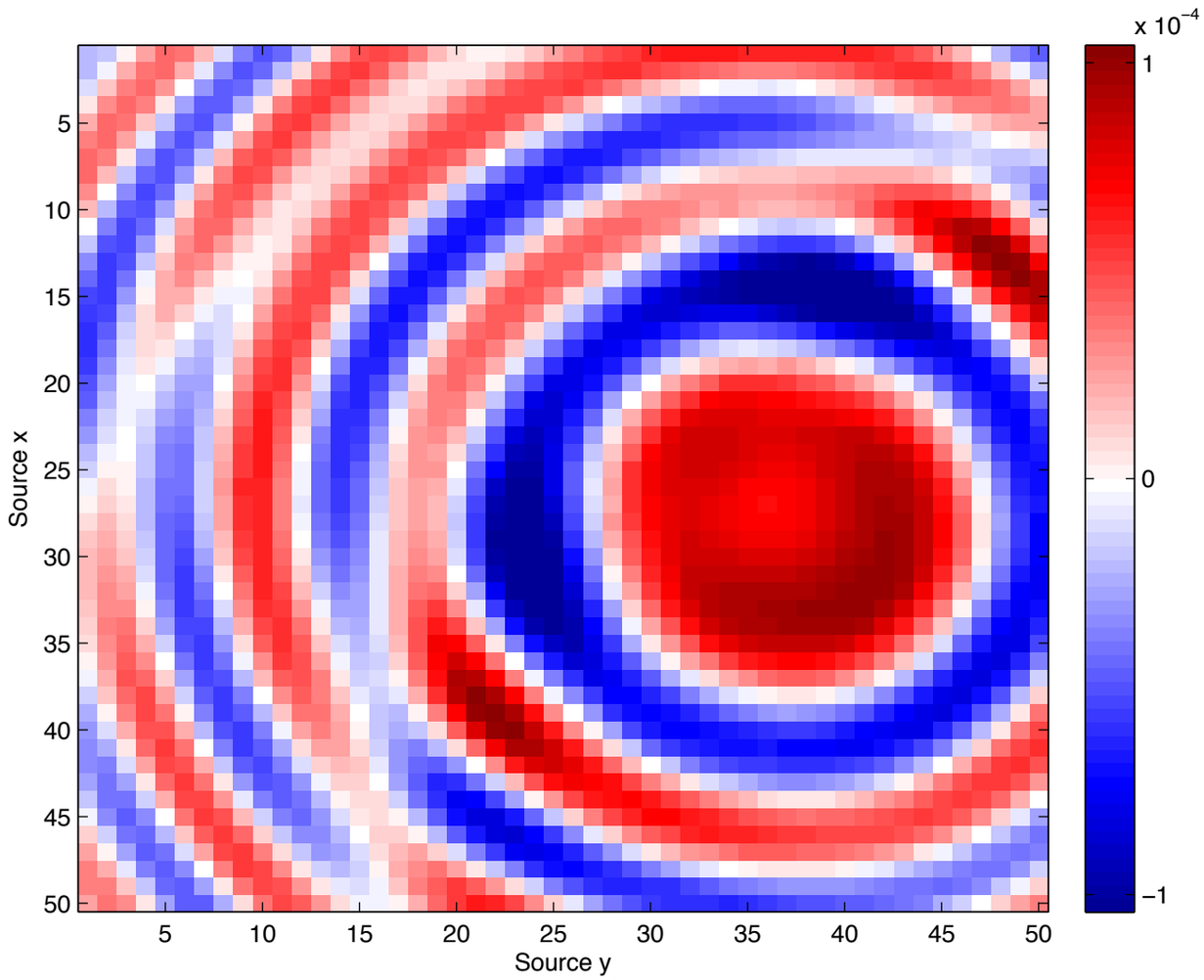
(Rec x, Rec y) = (5,39)



Recovered data

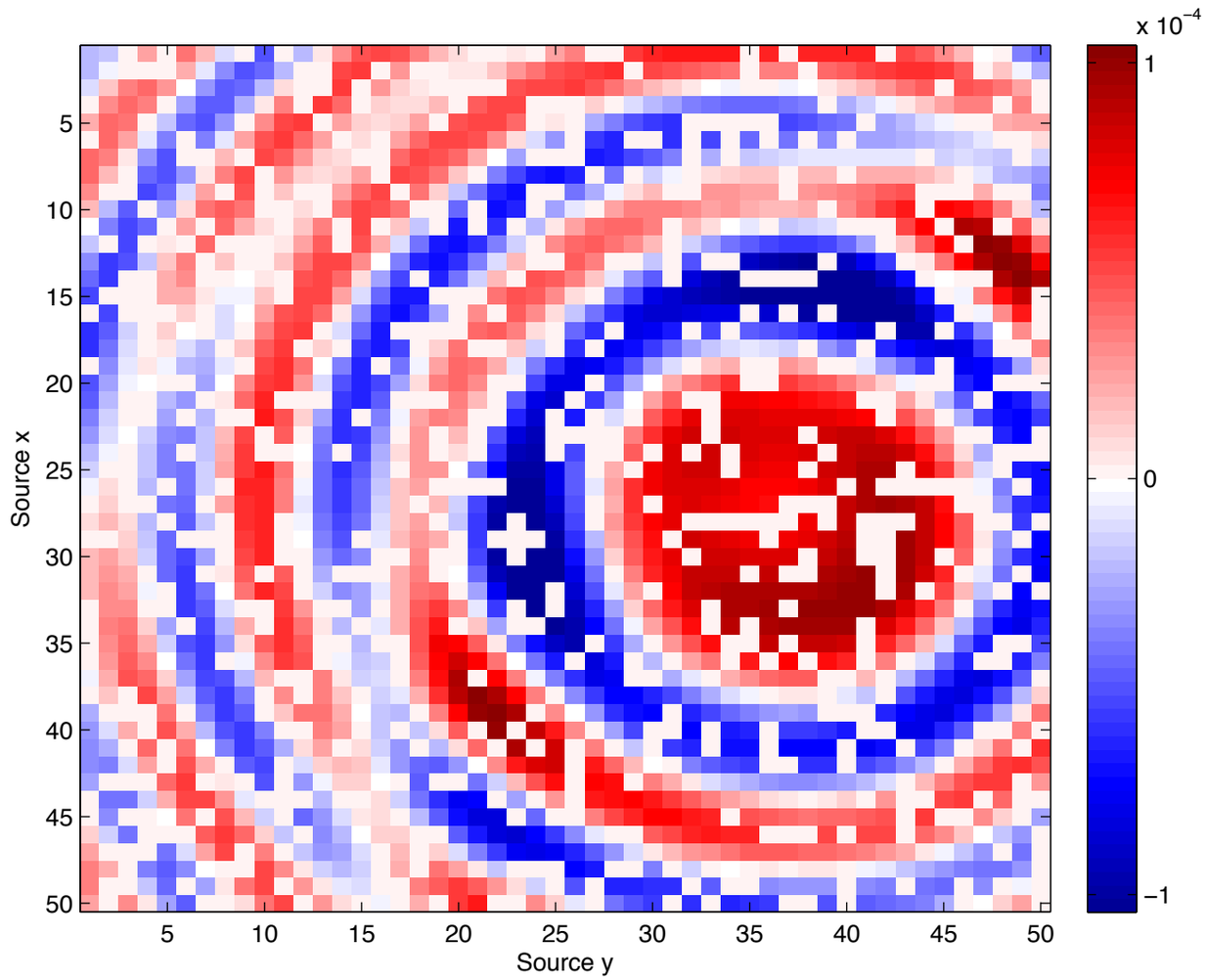
SNR - 16.8 dB

25% Missing Source Pairs



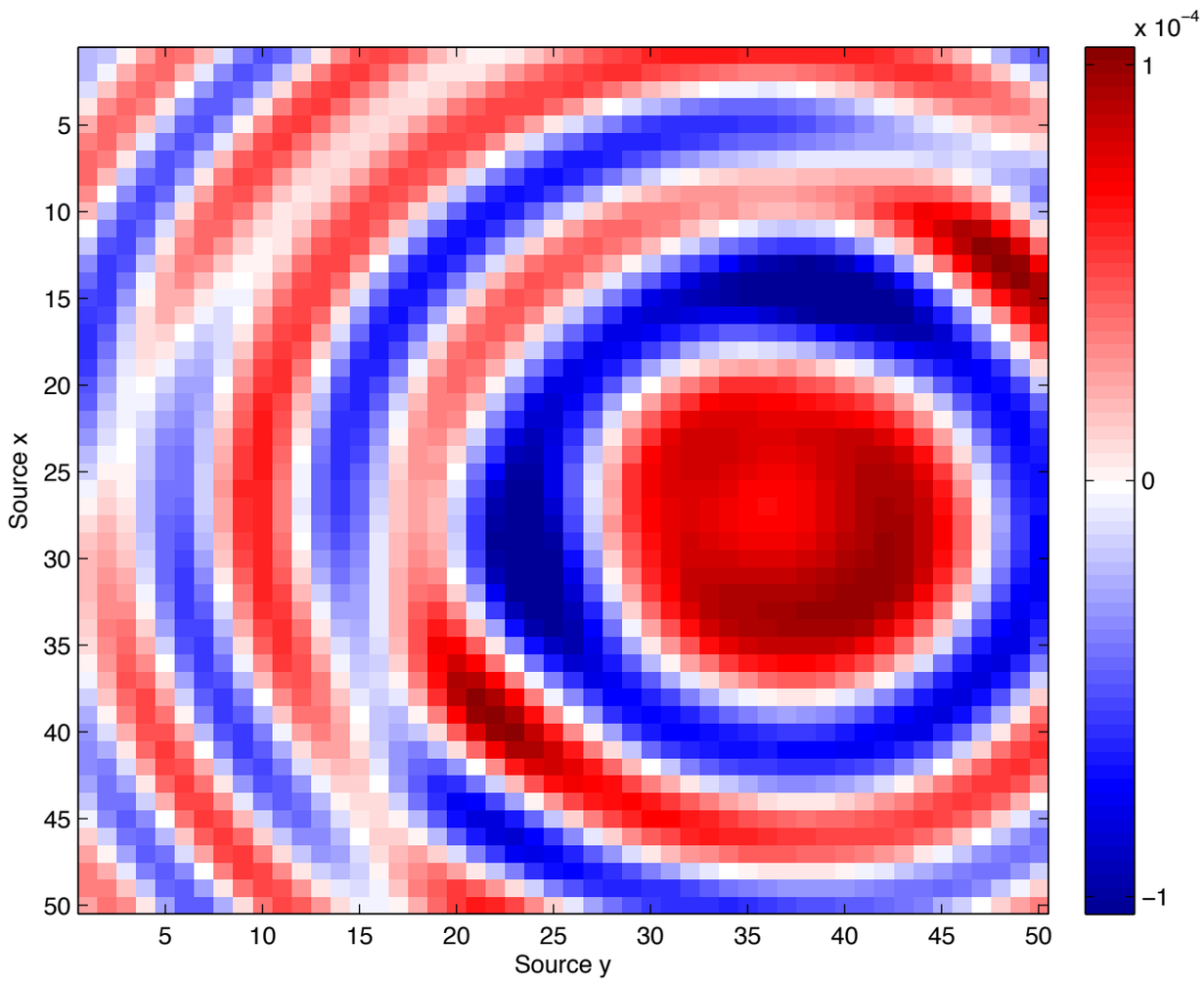
True data

(Rec x, Rec y) = (30,39)



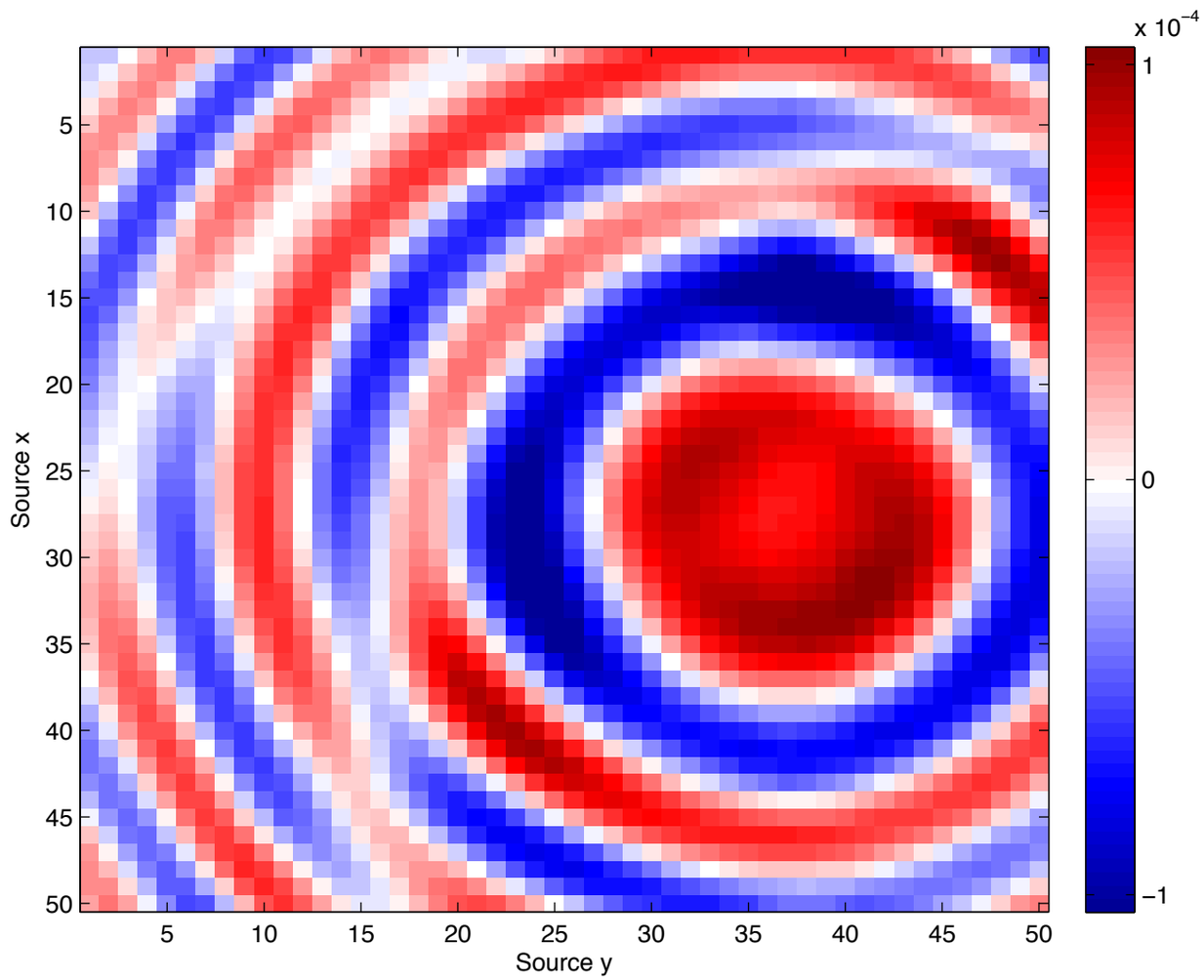
Subsampled data

25% Missing Source Pairs



True data

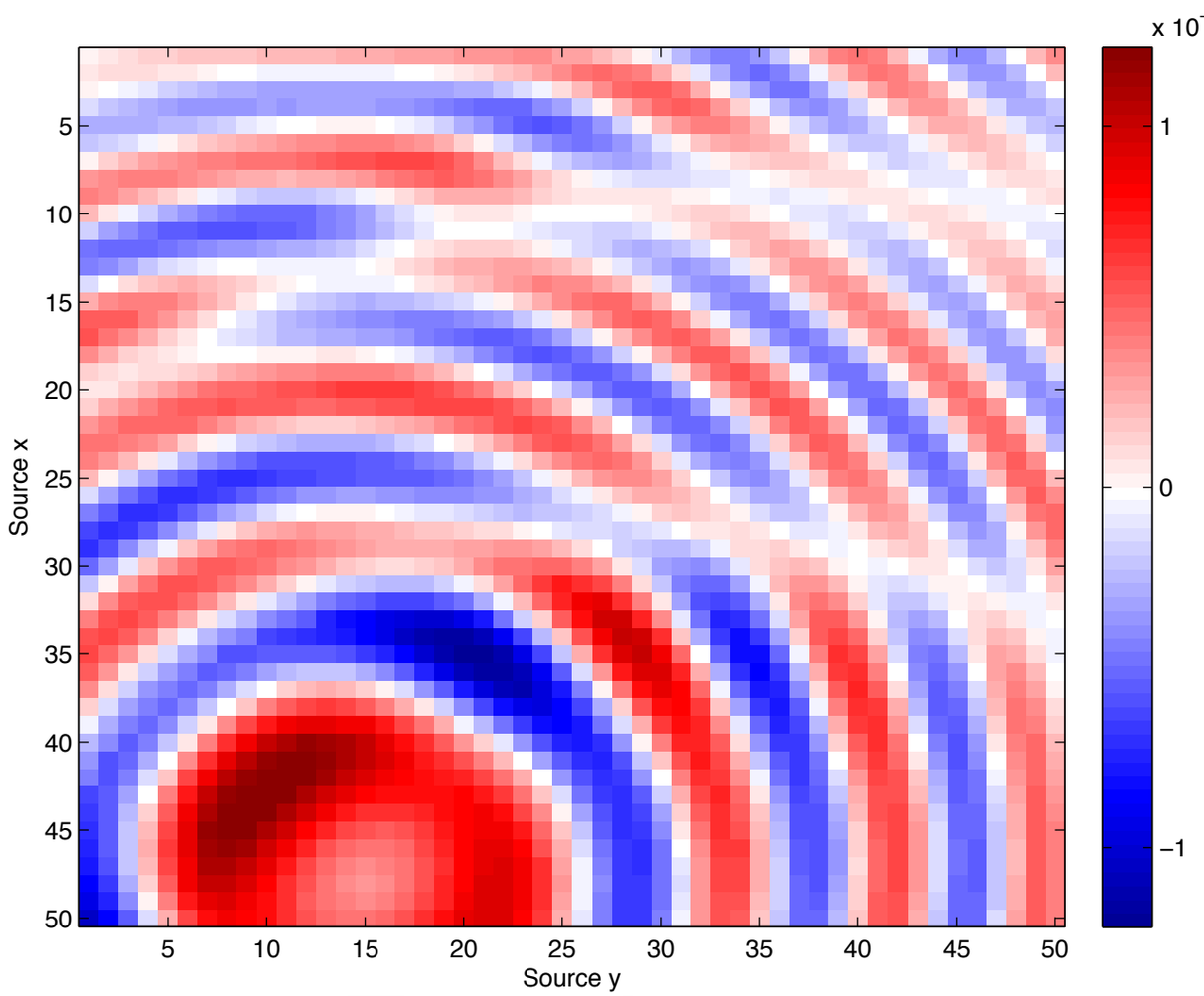
(Rec x, Rec y) = (30,39)



Recovered data

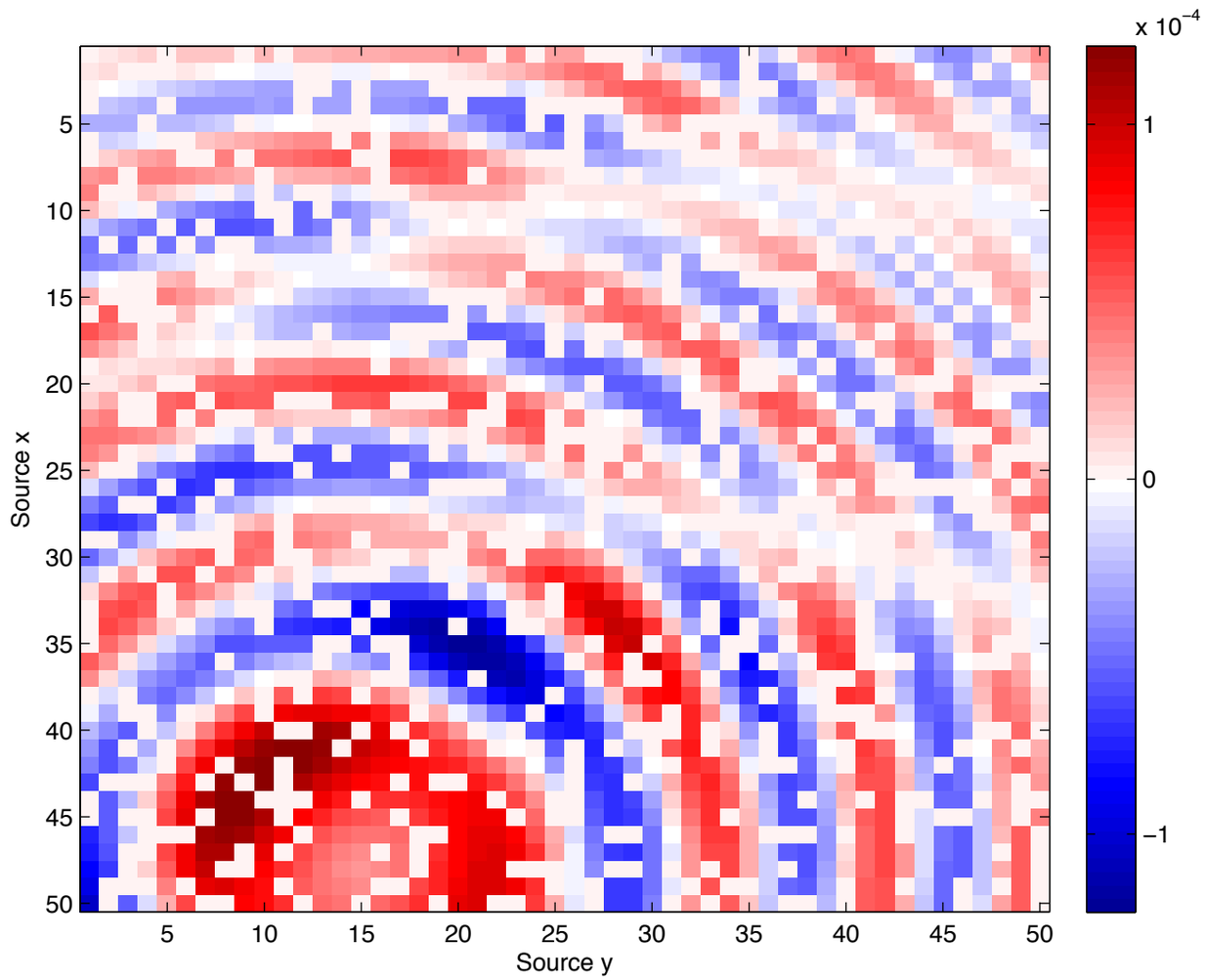
SNR - 18.7 dB

25% Missing Source Pairs



True data

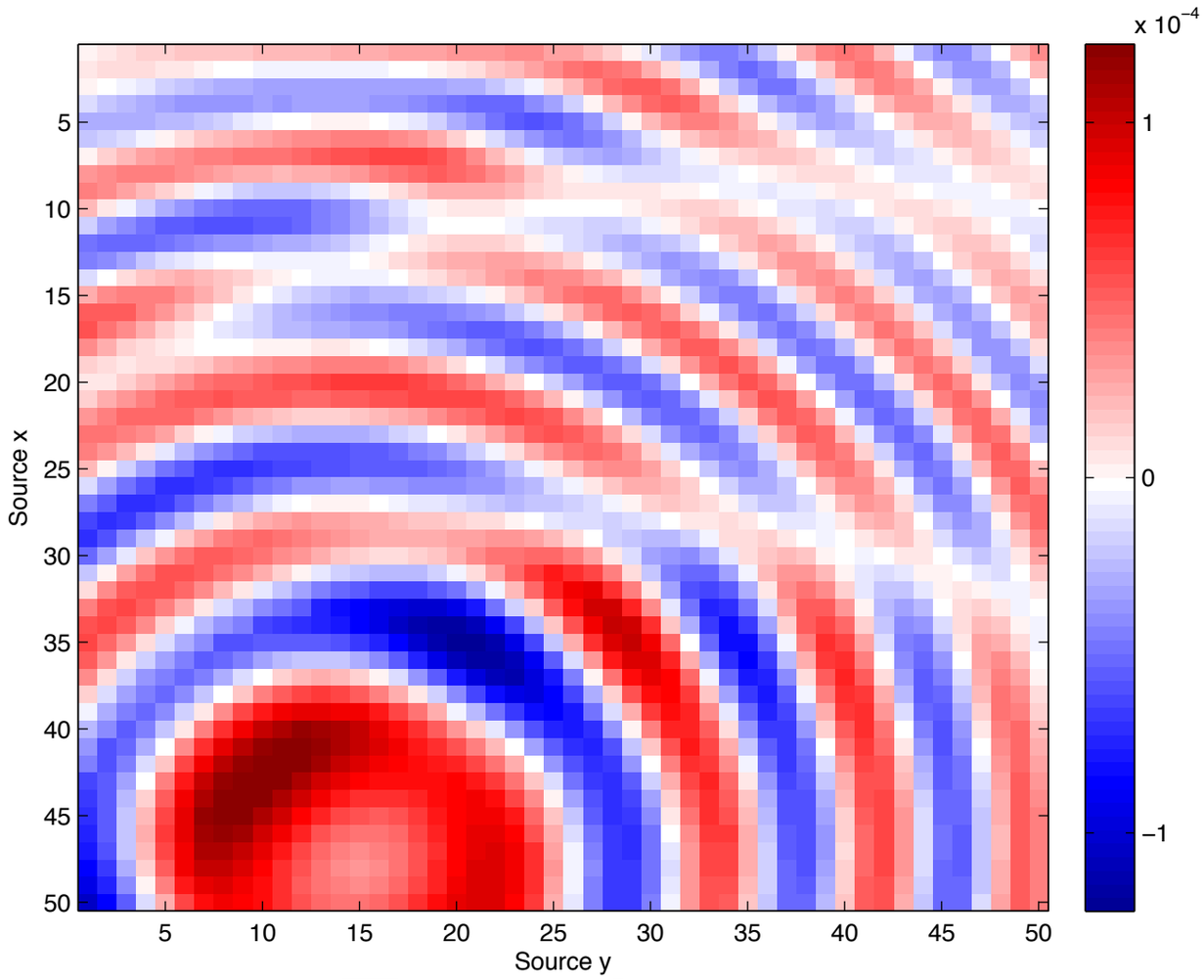
(Rec x, Rec y) = (50,17)



Subsampled data

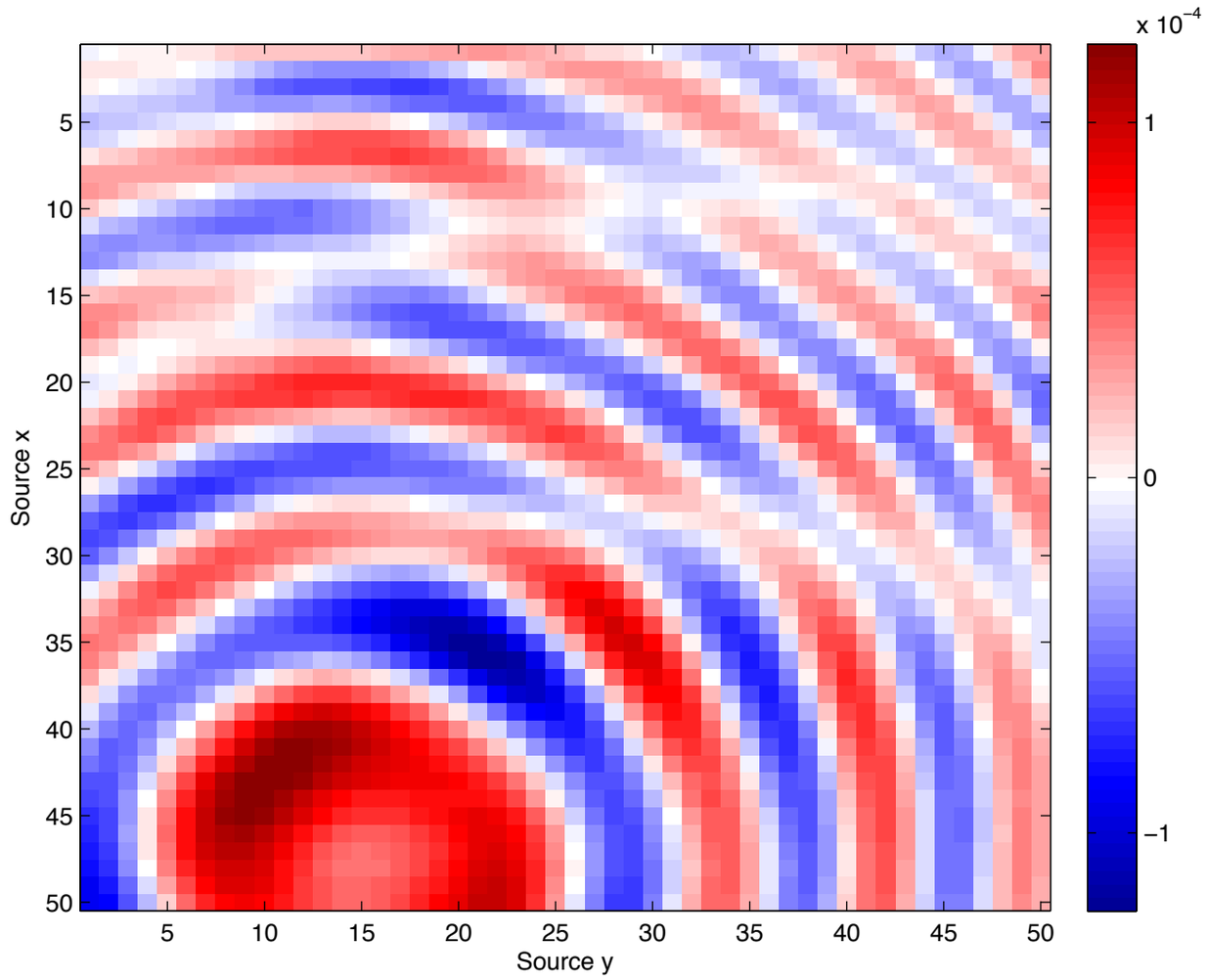
SLIM

25% Missing Source Pairs



True data

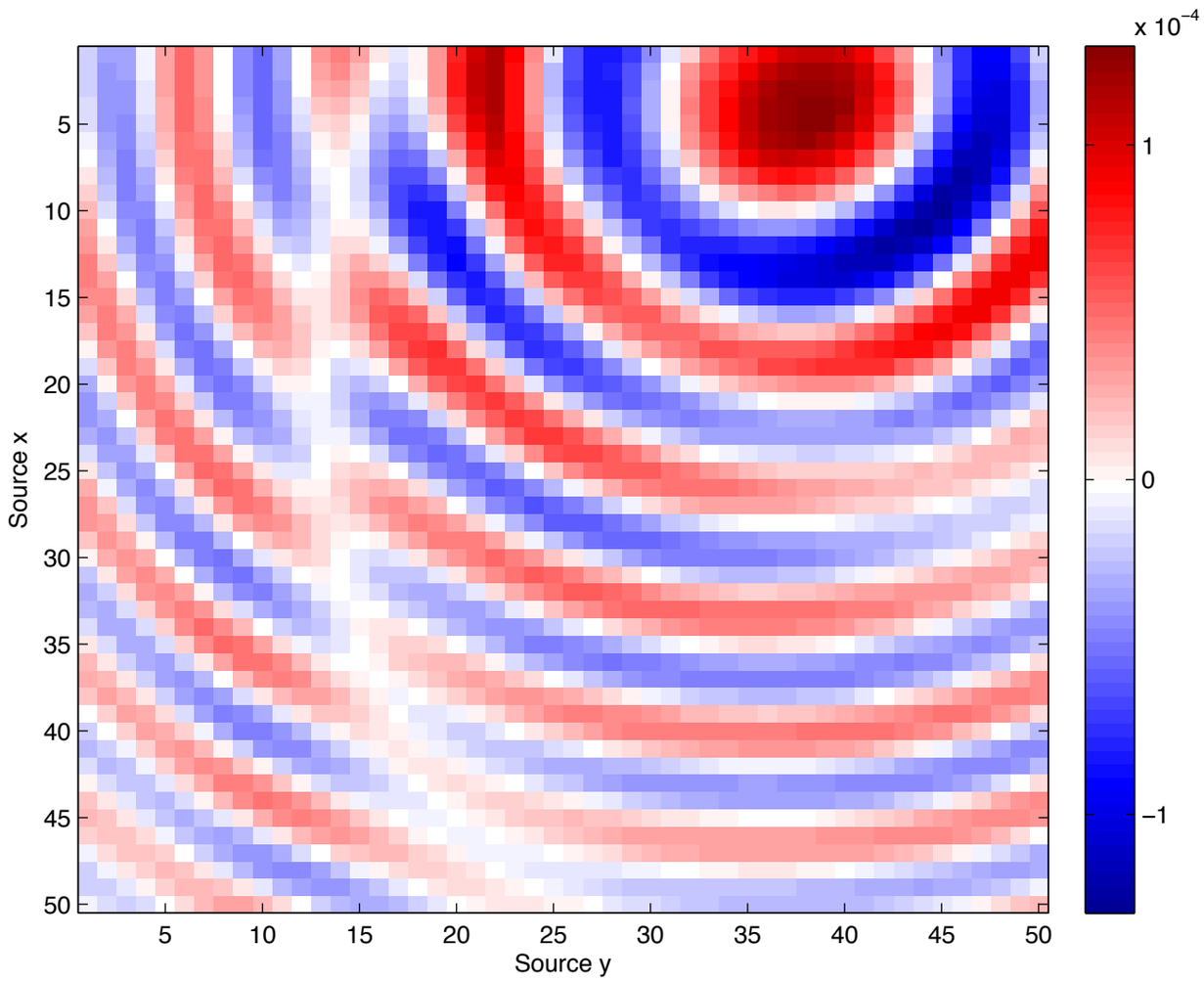
(Rec x, Rec y) = (50,17)



Recovered data

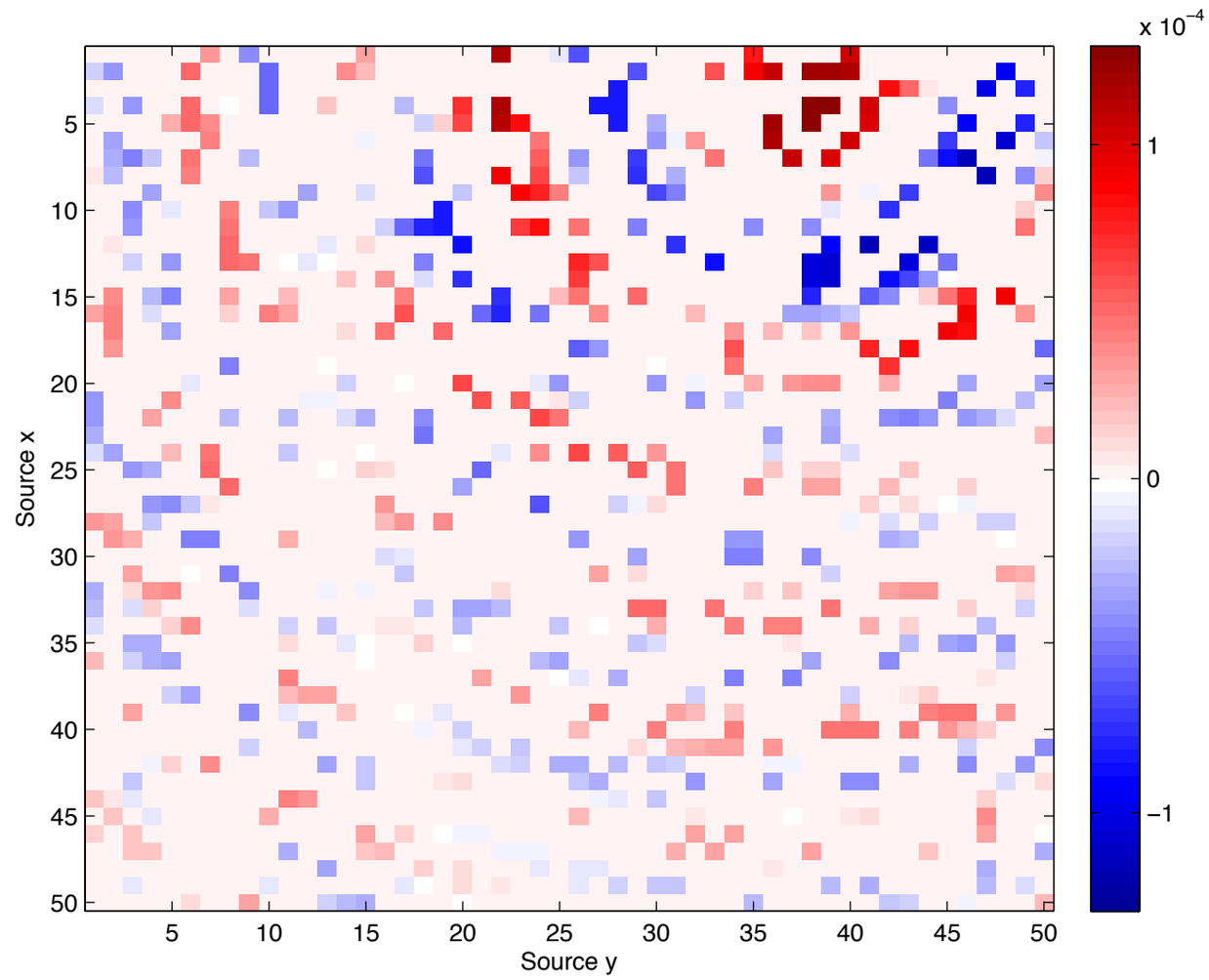
SNR - 14.8 dB

75% Missing Source Pairs



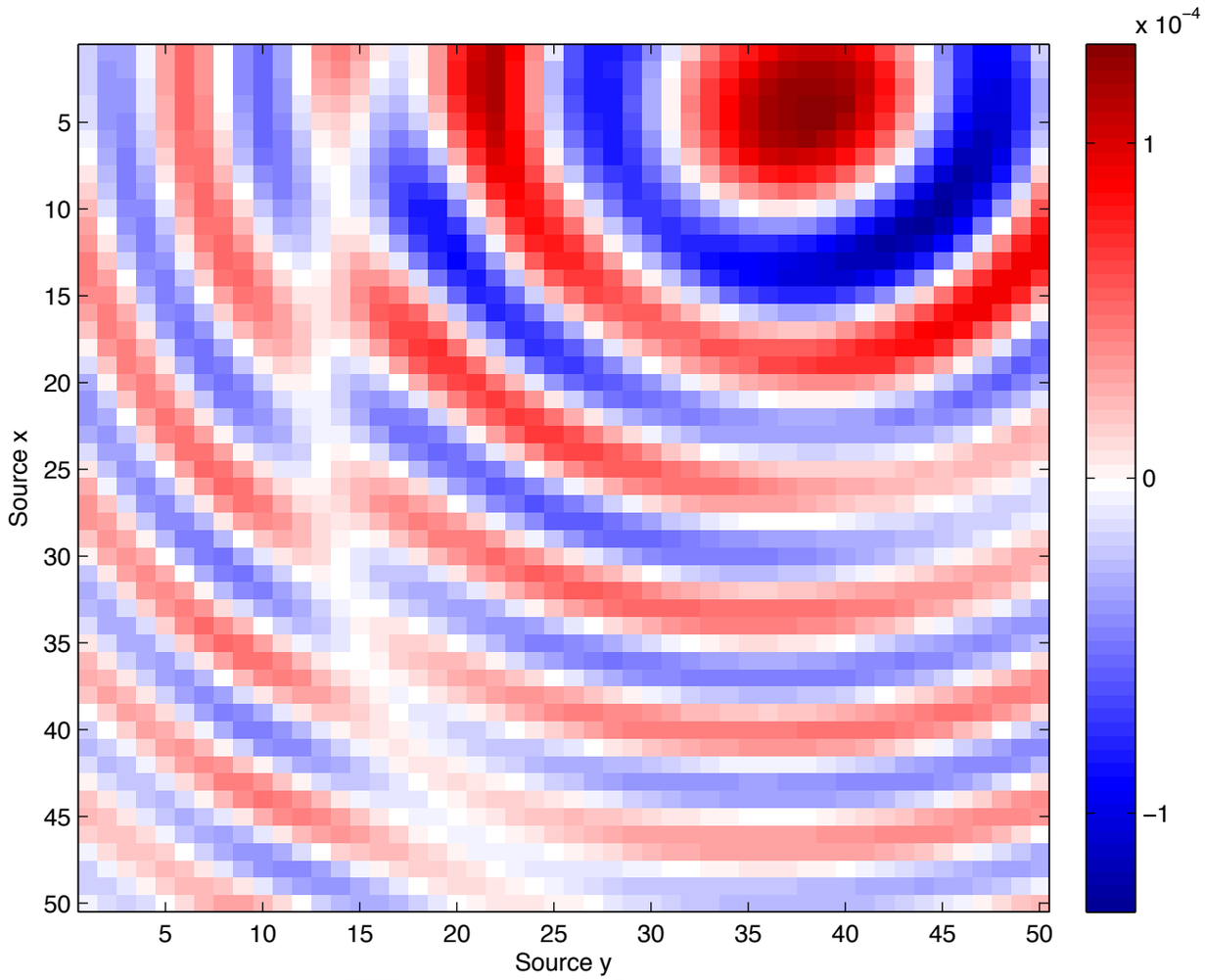
True data

(Rec x, Rec y) = (5,39)



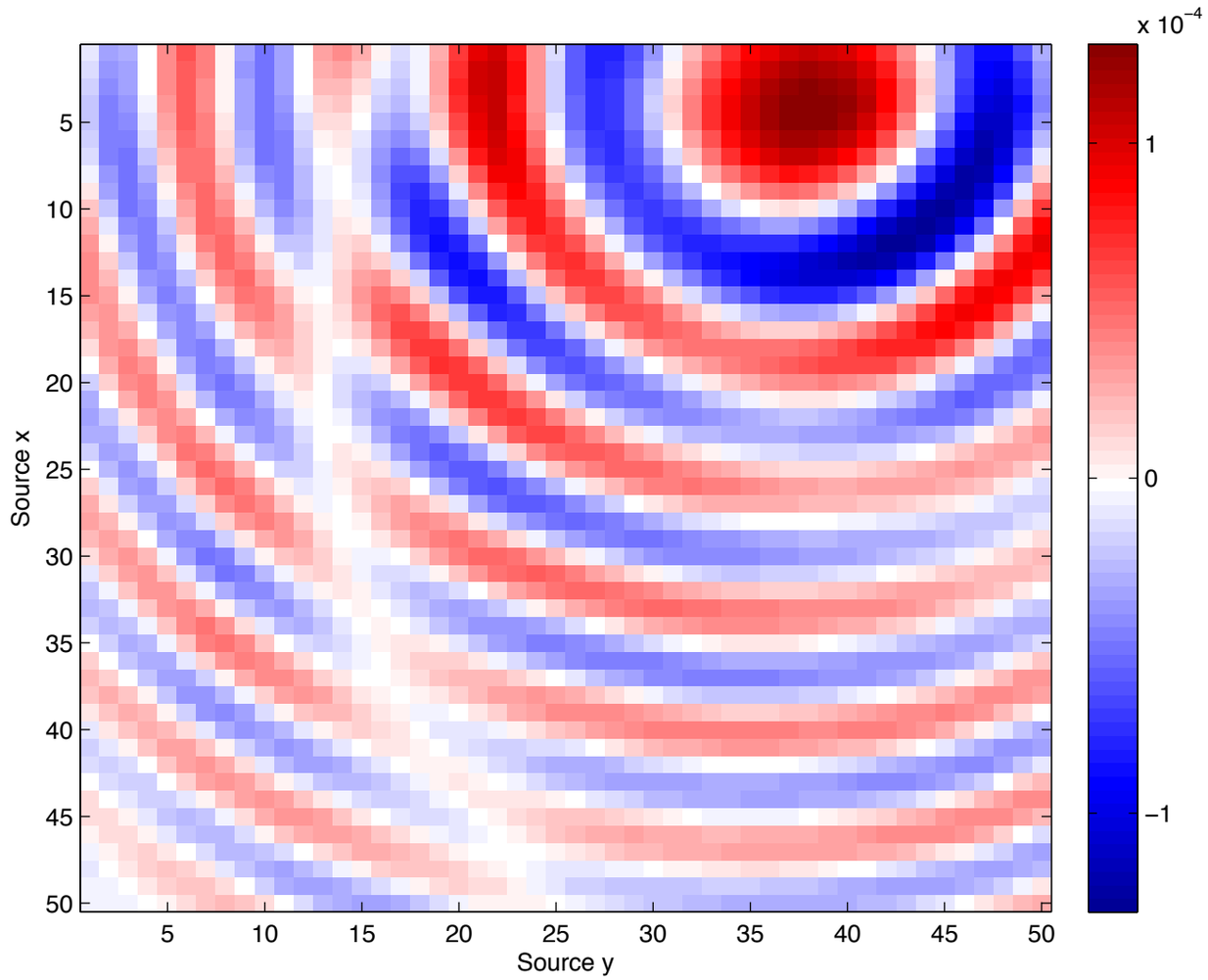
Subsampled data

75% Missing Source Pairs



True data

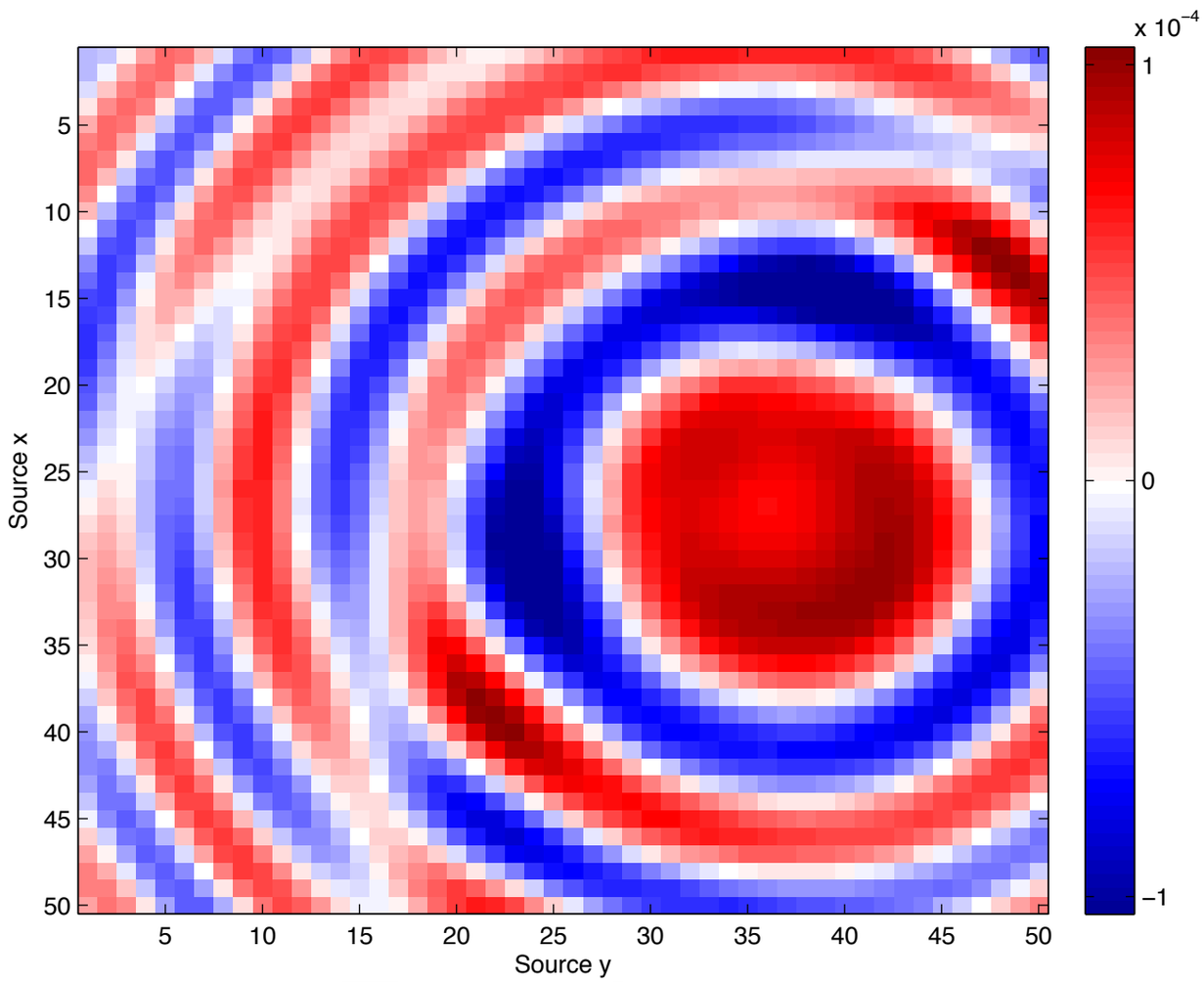
(Rec x, Rec y) = (5,39)



Recovered data

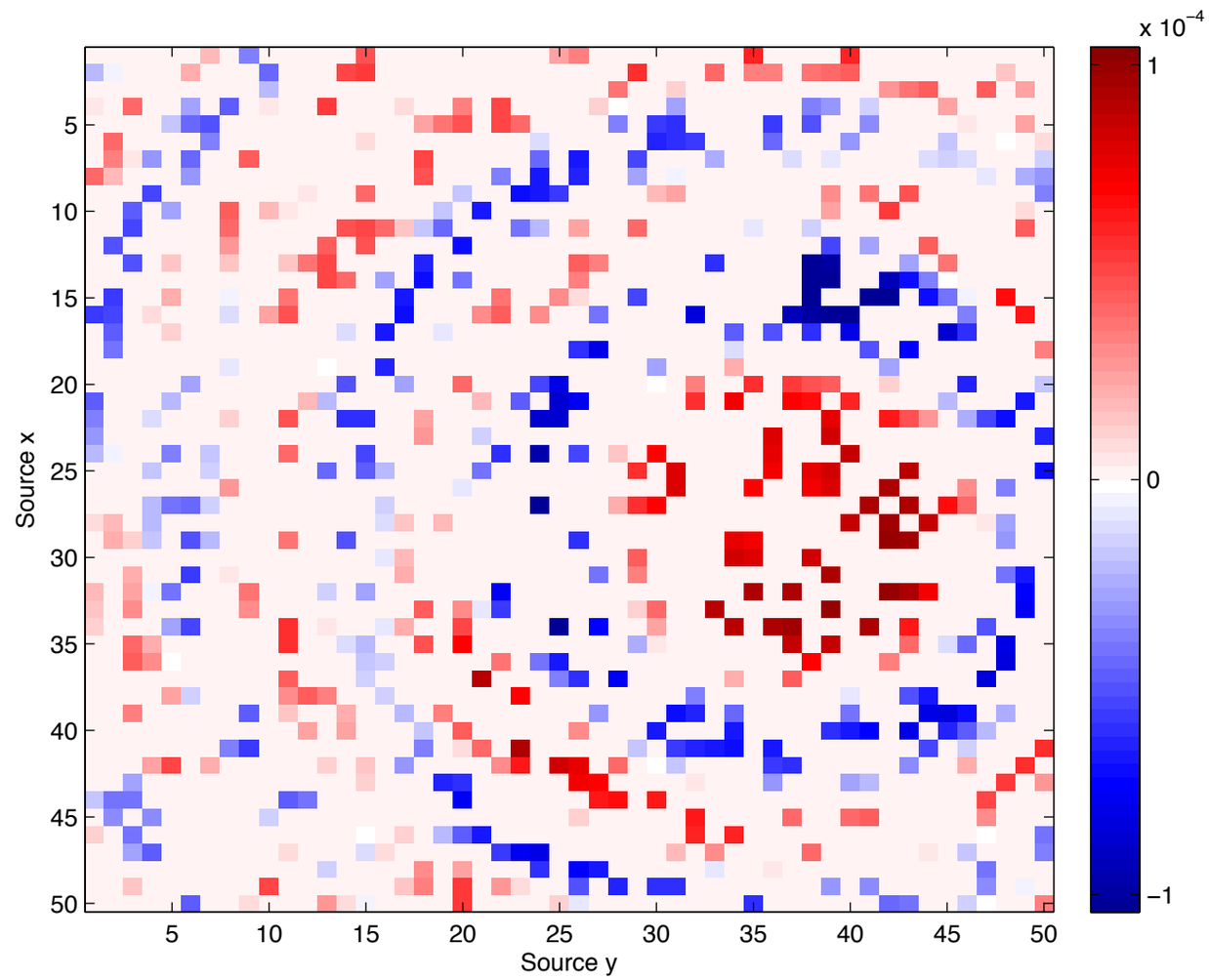
SNR - 18.5dB

75% Missing Source Pairs



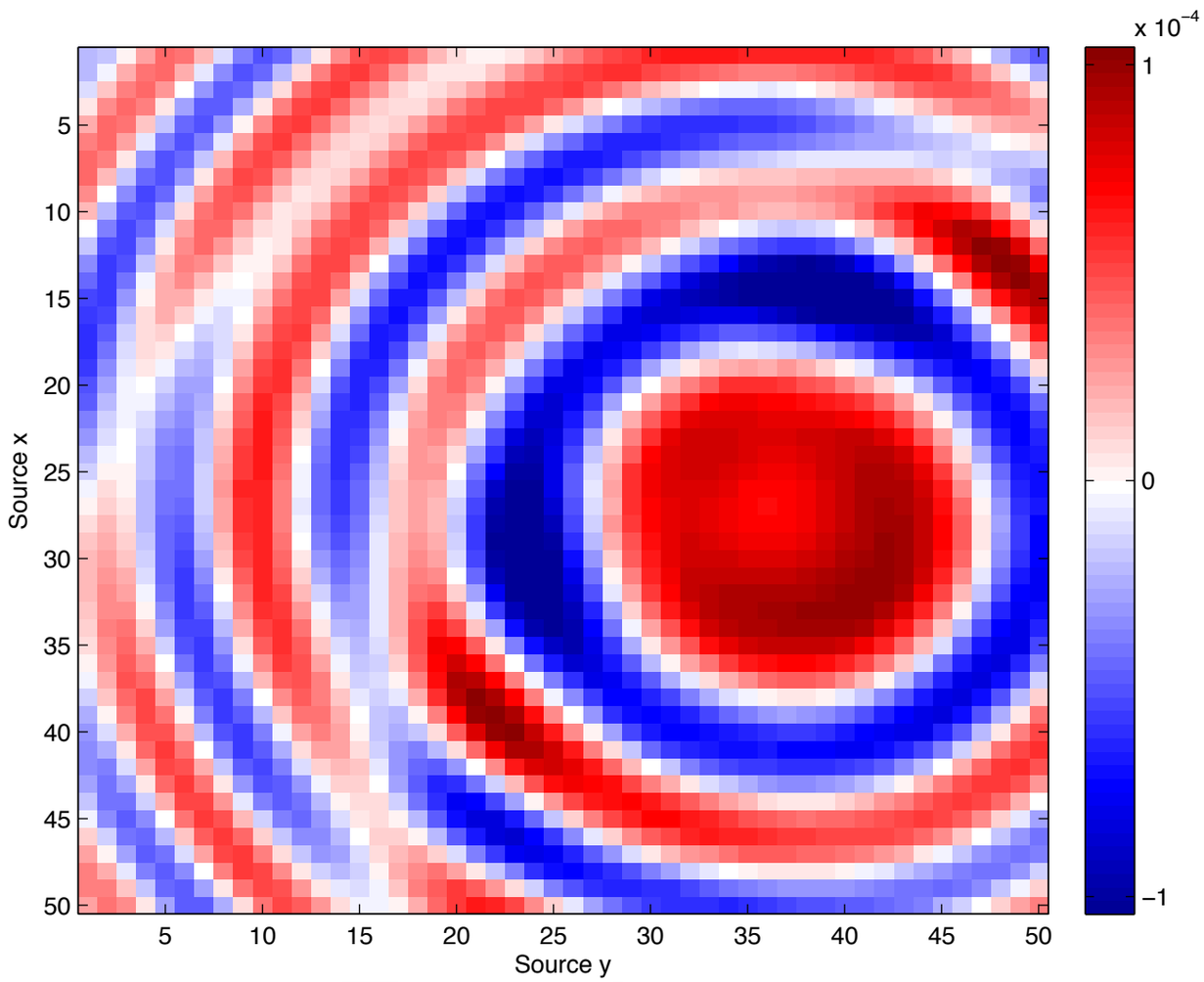
True data

(Rec x, Rec y) = (30,39)



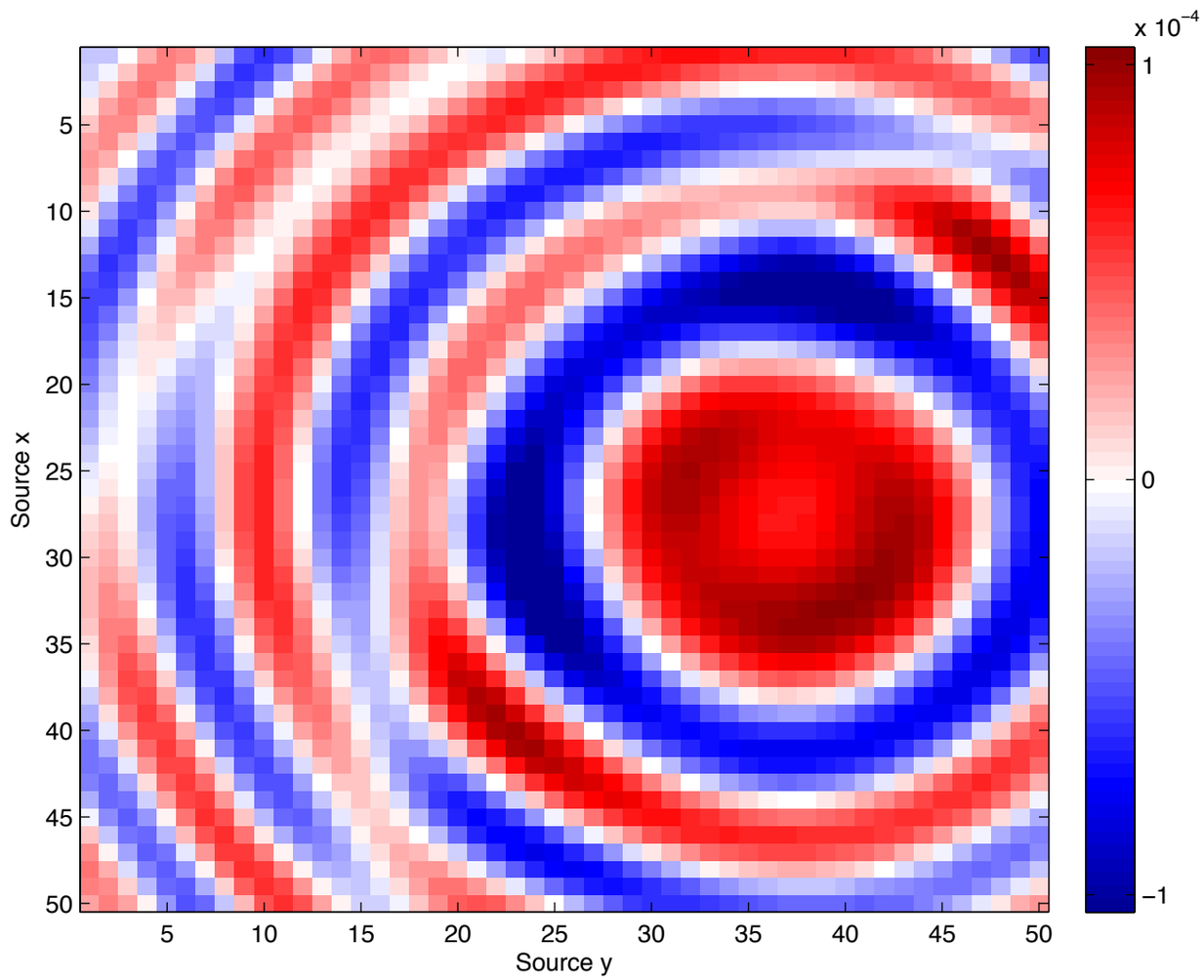
Subsampled data

75% Missing Source Pairs



True data

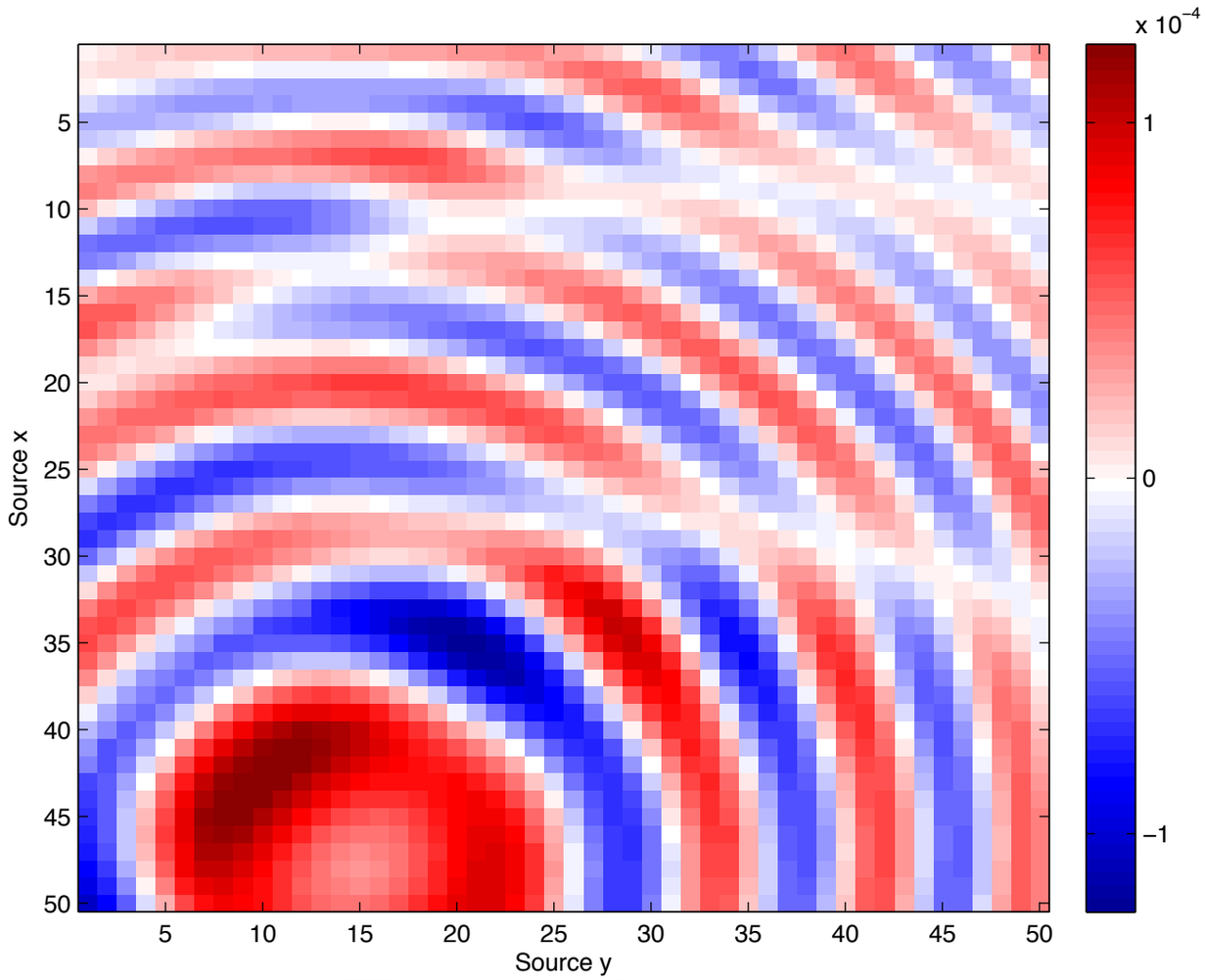
(Rec x, Rec y) = (30,39)



Recovered data

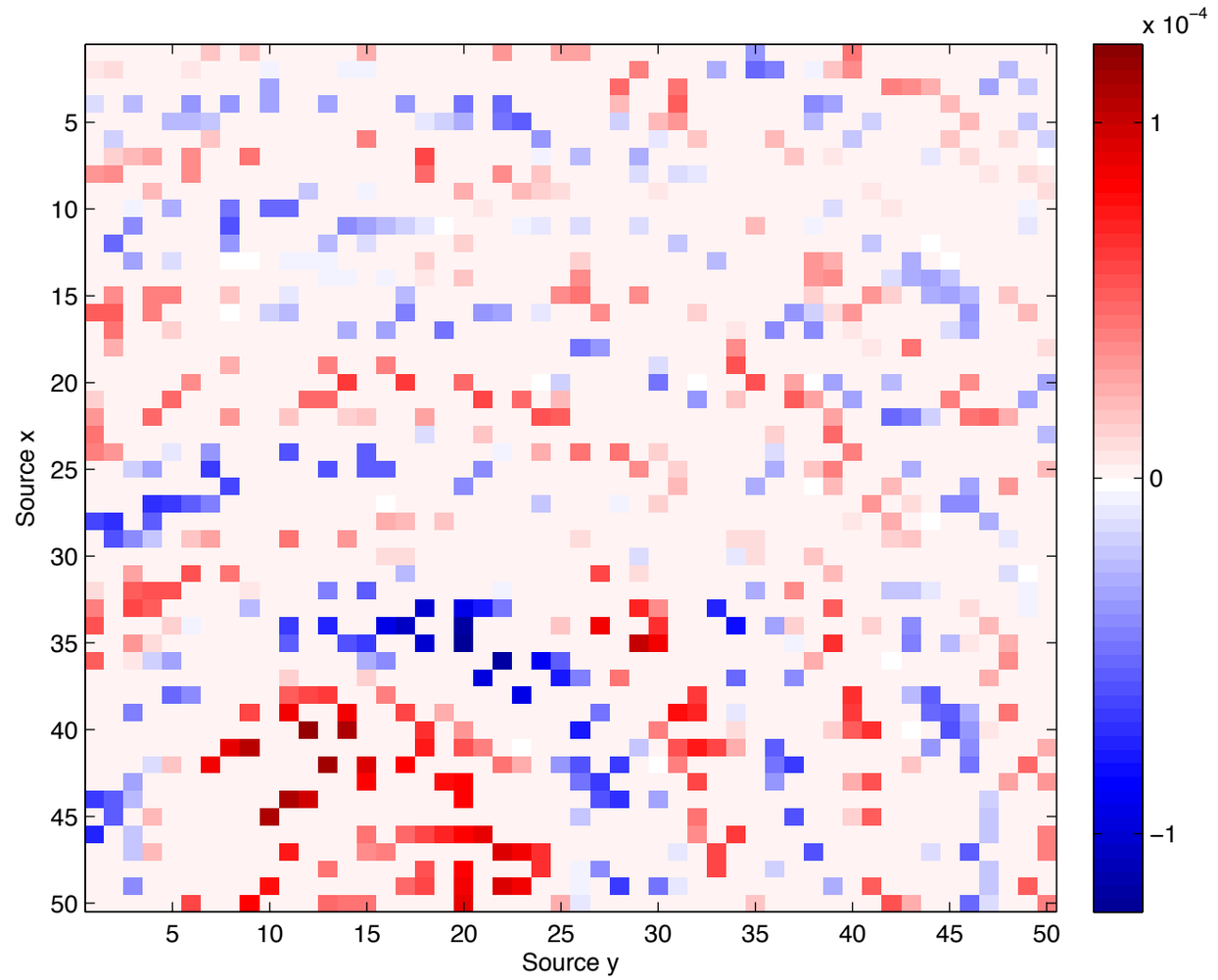
SNR - 18.7 dB

75% Missing Source Pairs



True data

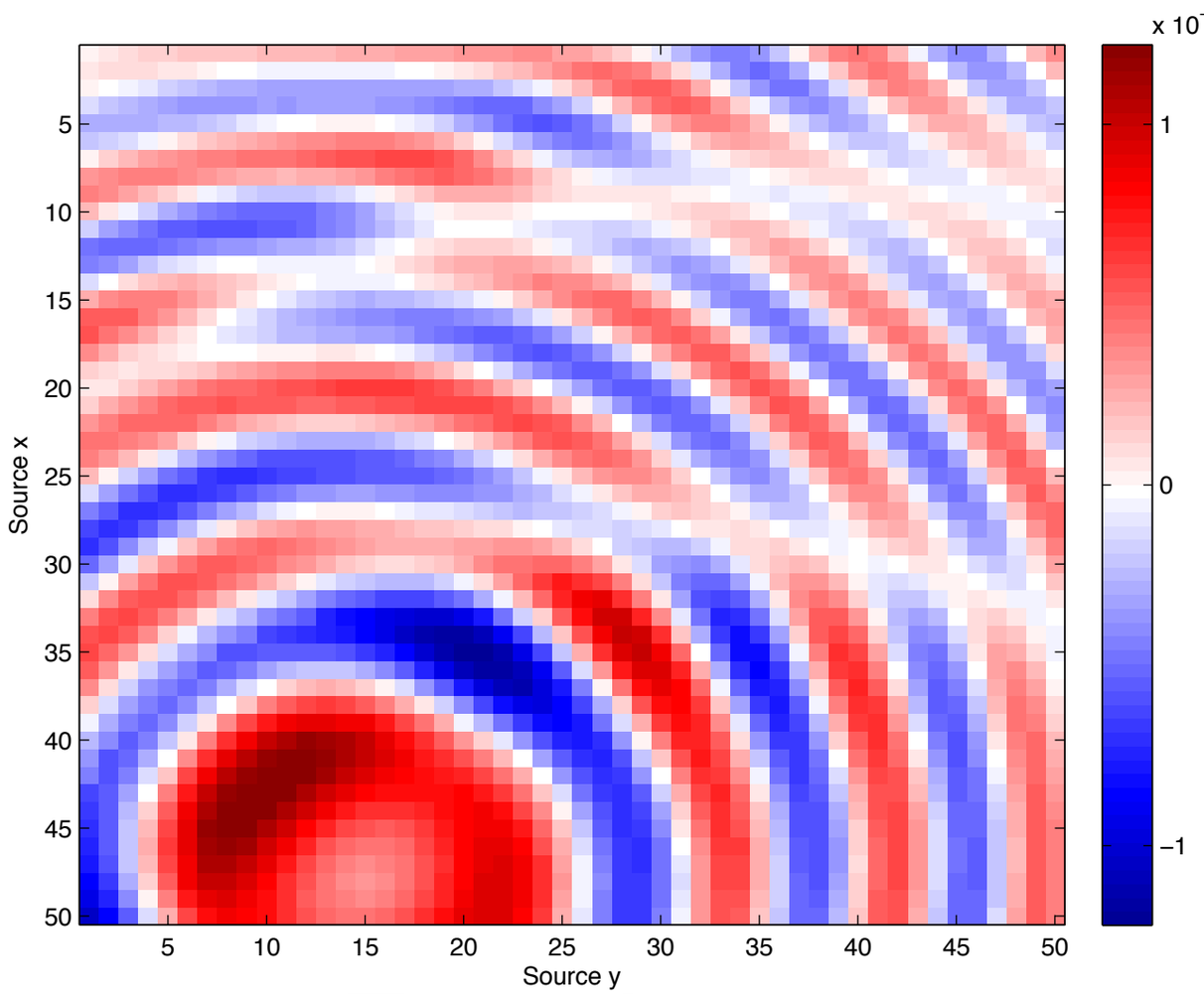
(Rec x, Rec y) = (50,17)



Subsampled data

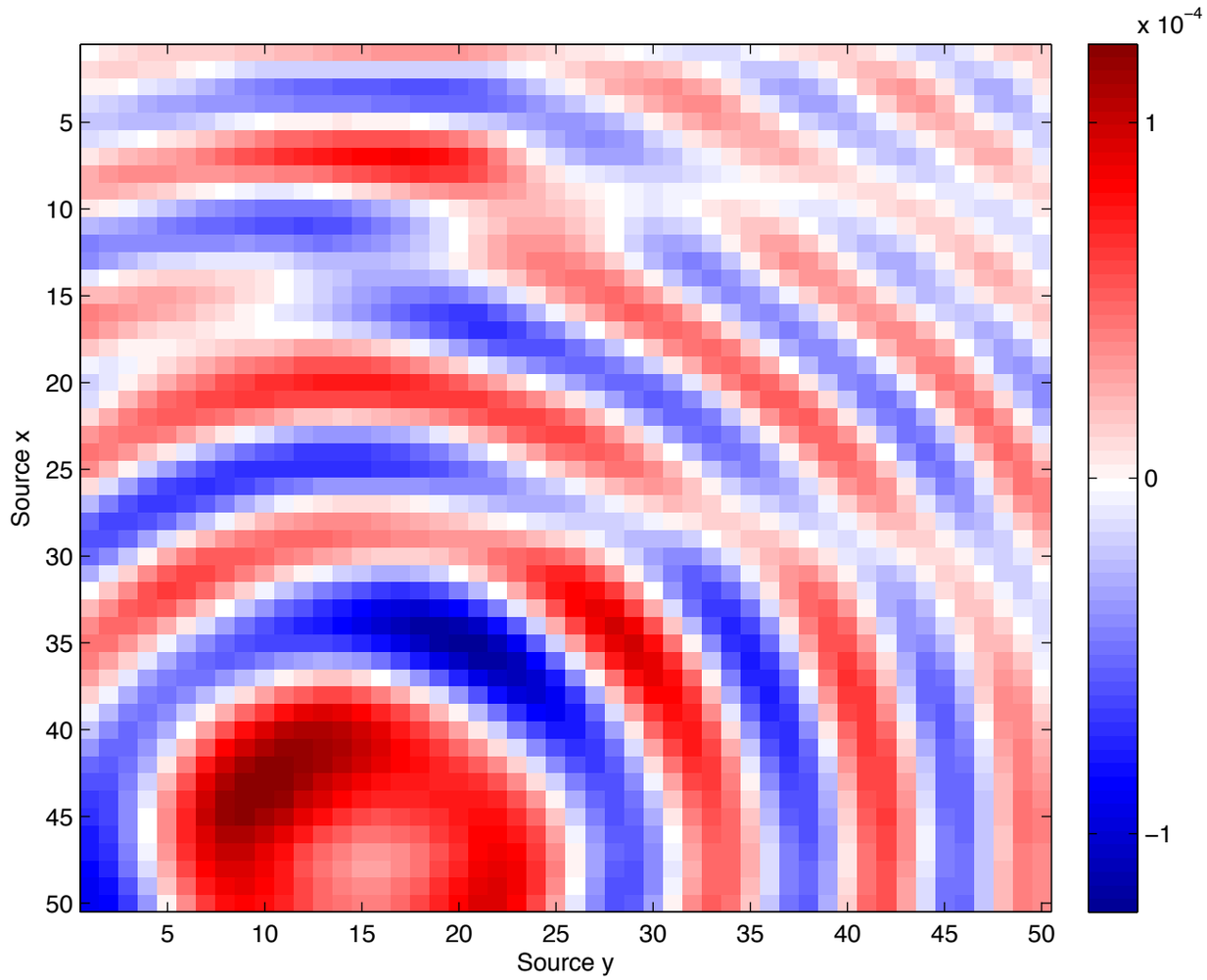
SLIM

75% Missing Source Pairs



True data

(Rec x, Rec y) = (50,17)



Recovered data

SNR - 13 dB

Synthetic BG Data

- Unknown model, unknown full data
- 200 shots (subsampled srcs) x 401 x 401 receivers data at 7.34 Hz
- Receivers subsampled to 101 x 101, fourier interpolated back to 401 x 401 to produce figures (due to low frequency content of slice)

Synthetic BG Data

- Single frequency slice (real part) - normalized
- 80 internal rank, 80 receiver rank, various source ranks

Synthetic BG Data

- minConf_PQN - optimization code by M. Schmidt, M. Friedlander, and K. Murphy (2009)
- L-BFGS code for optimization problems with costly objectives, cheap constraint projection
  - Exactly our regime

Synthetic BG Data

- Interpolating from 200 irregularly sampled shots -> 100 x 100 grid (10000 shots) - 98% missing data
- No free lunch here, illustrates the effects of overly aggressive upsampling
- Better results on a less-greedy grid
  - Can’t coarsen grid too much

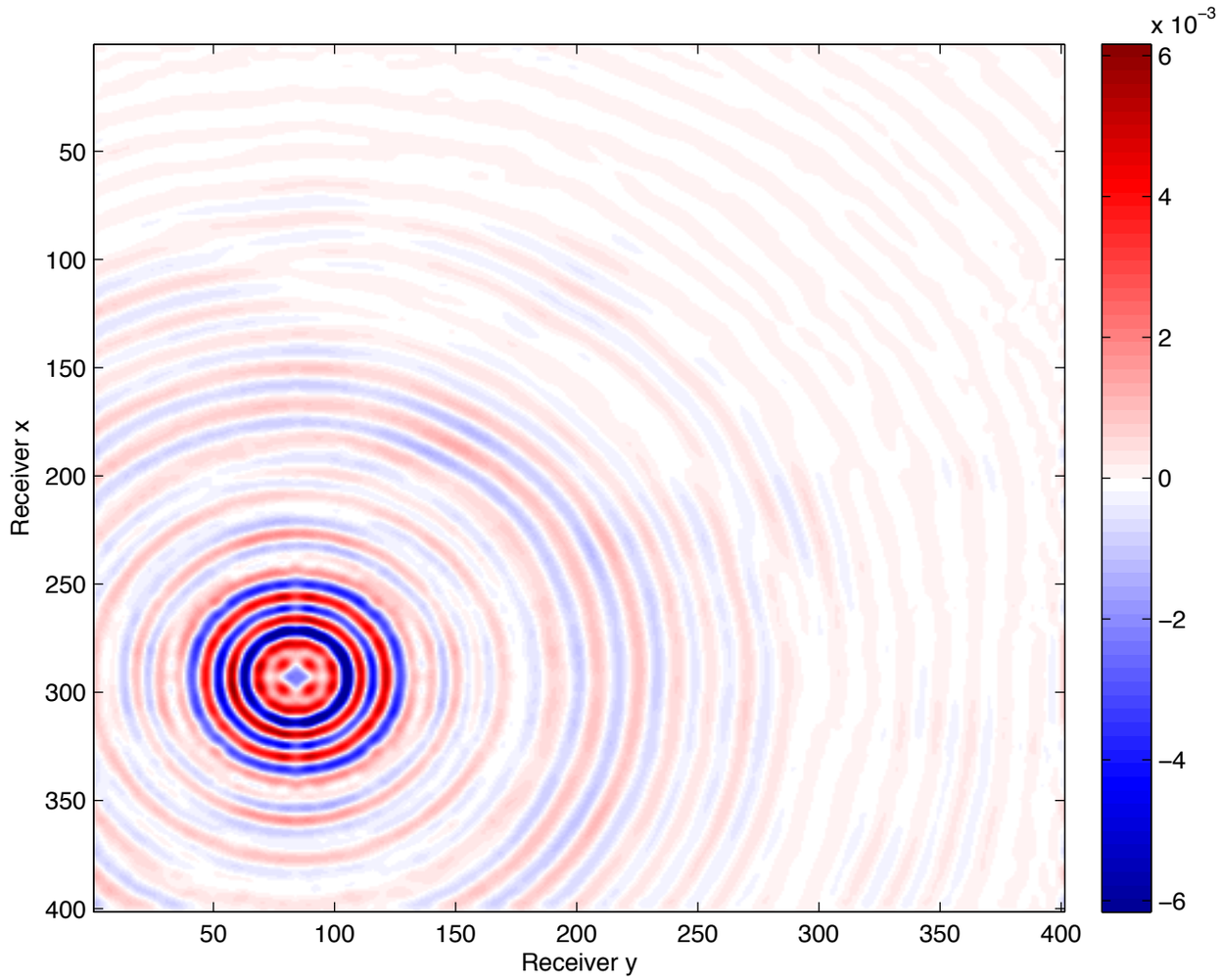
Synthetic BG Data

- Source rank 20
- Locations of poor results aren't structured
  - Doesn't really matter if we weren't going to obtain anything at those points anyways

Synthetic BG Data

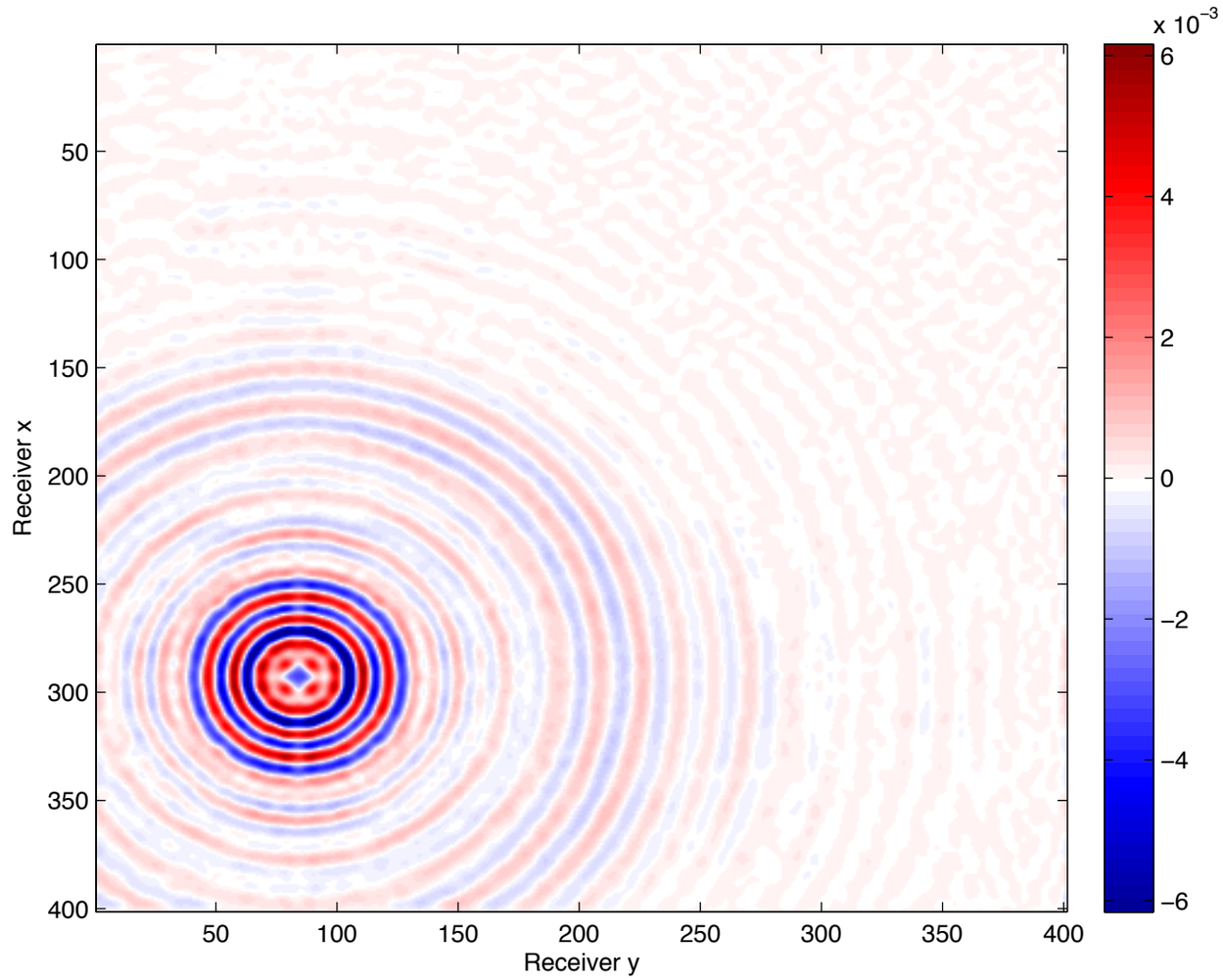
- Source rank 20 - good tradeoff between interpolation quality + SNR
- Number of parameters: 282560
- Compare to 102010000 for the full 100 x 100 x 101 x 101 cube
- Compression of a factor of 99.7%

Source rank 20 - 500 iterations



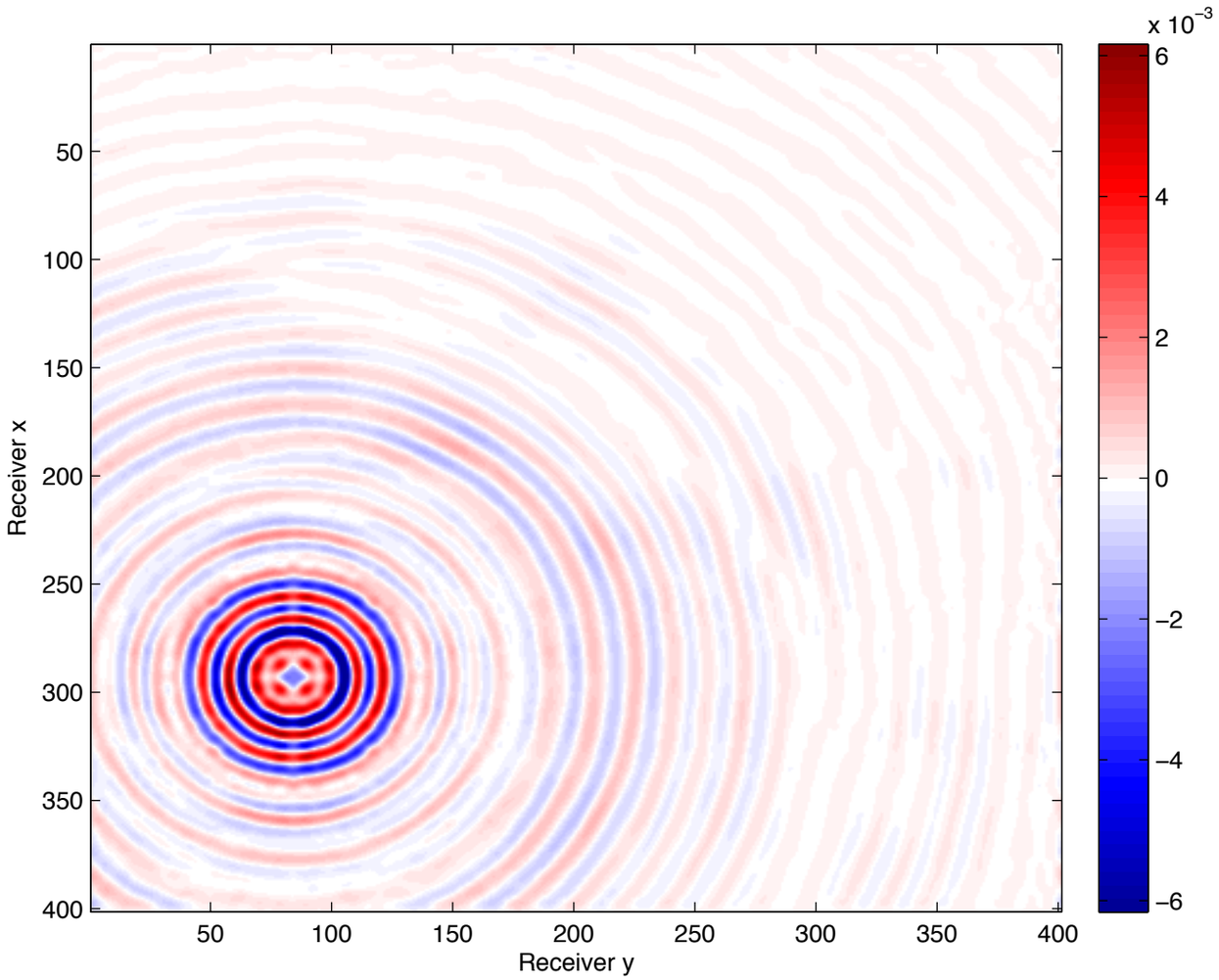
Known data

(Src x, Src y) = (50,15)



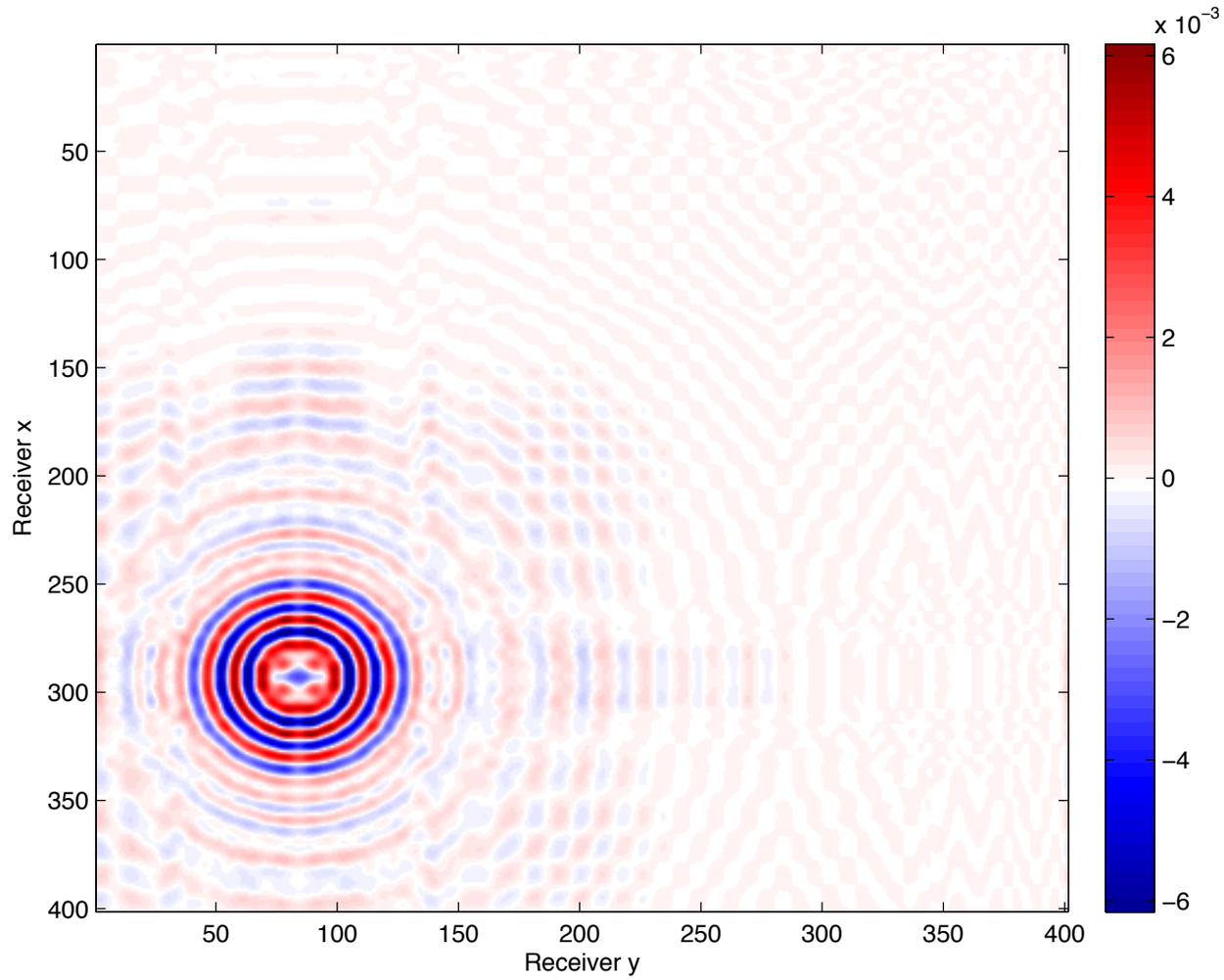
Interpolated data - SNR 13.6 dB

LR Matrix Completion



Known data

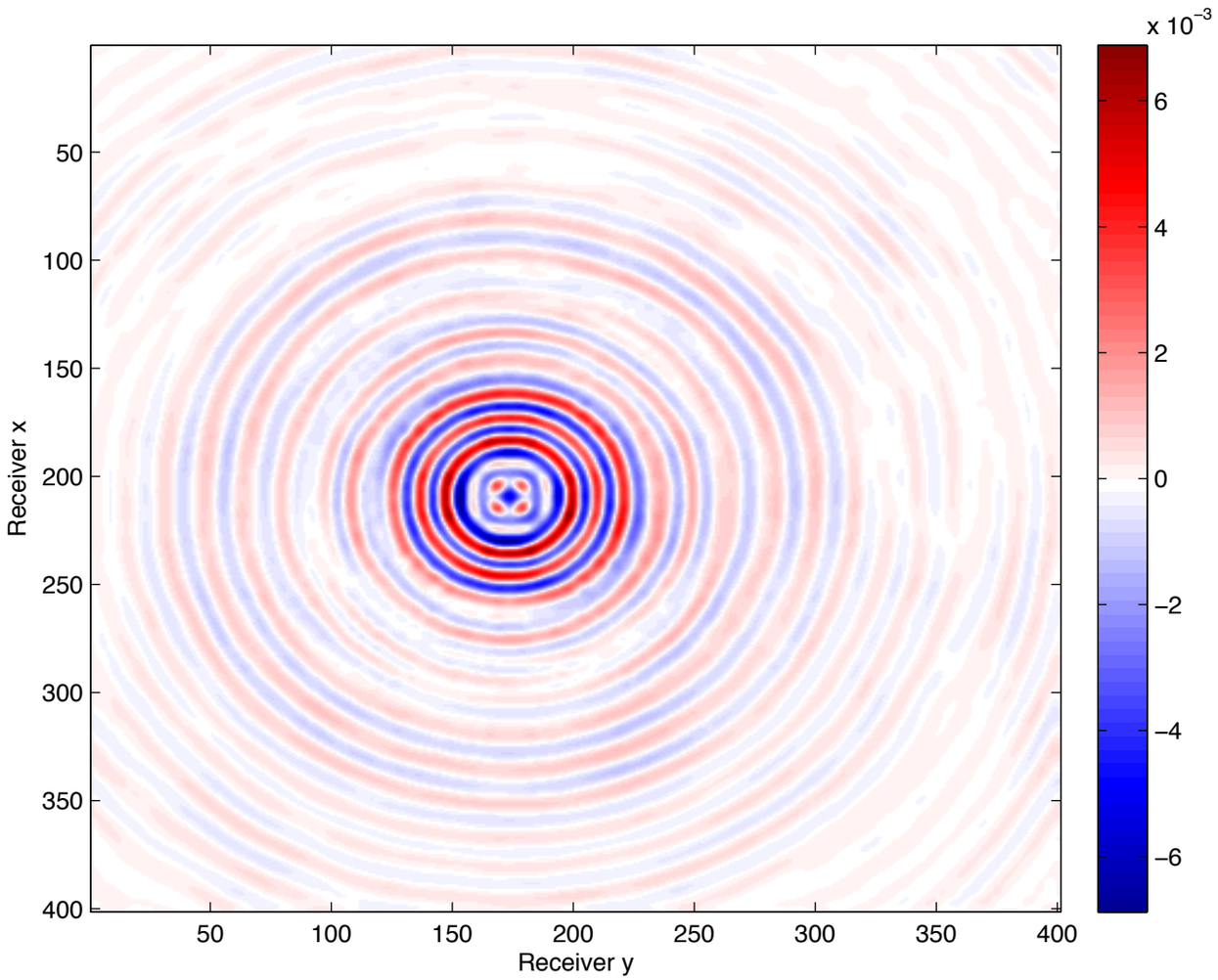
(Src x, Src y) = (50,15)



Interpolated data -

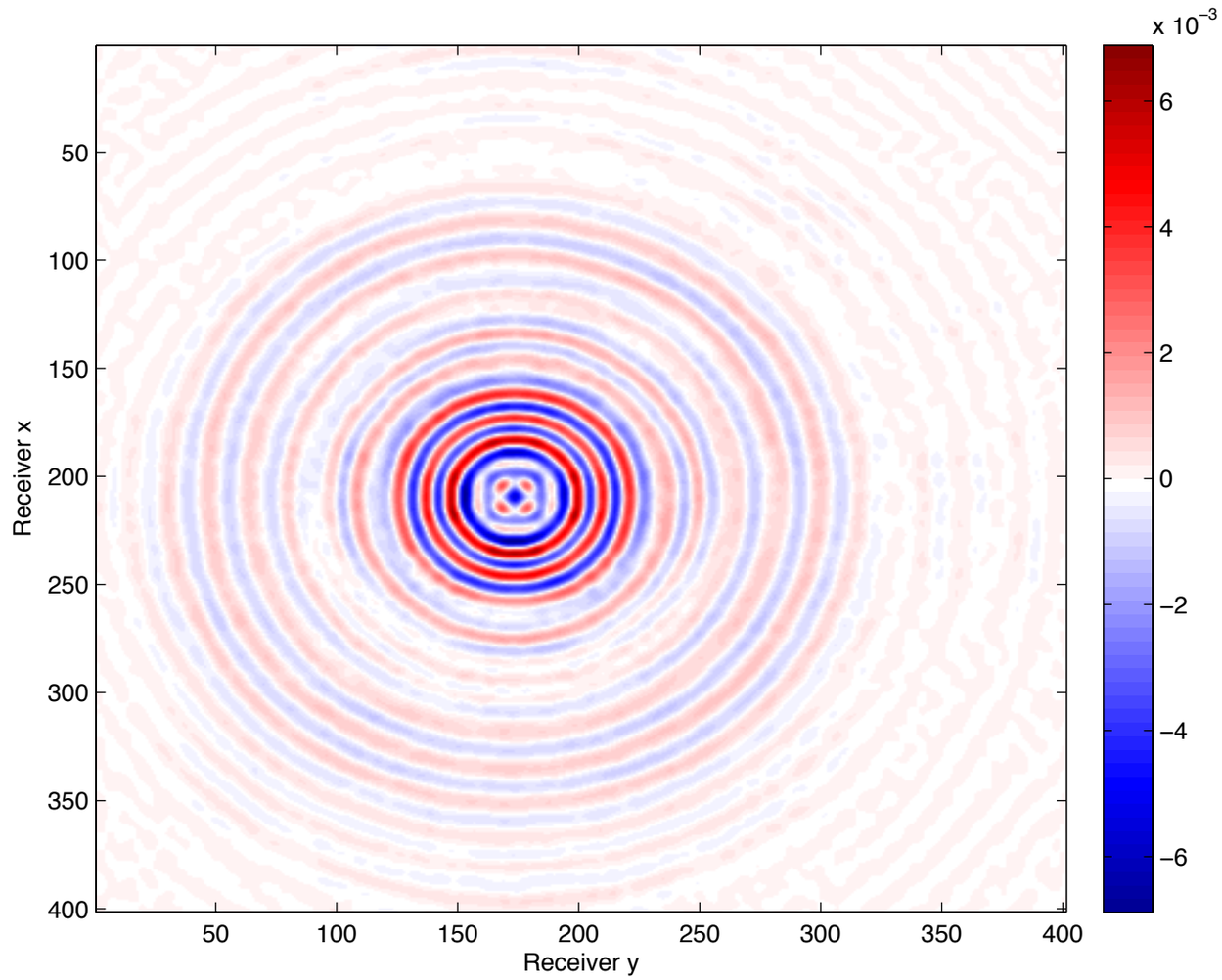
SNR 8.14 dB

Source rank 20 - 500 iterations



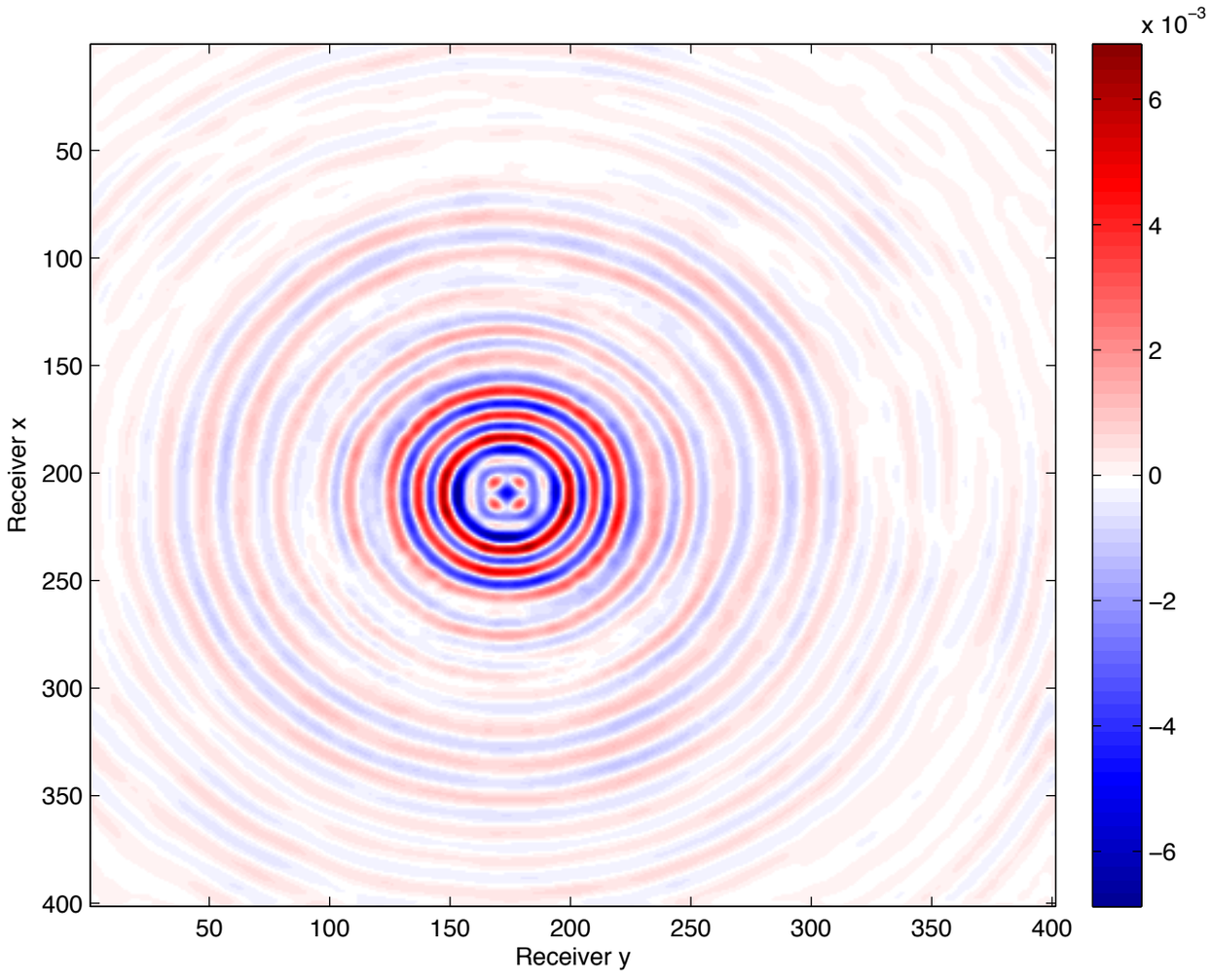
Known data

(Src x, Src y) = (36,30)



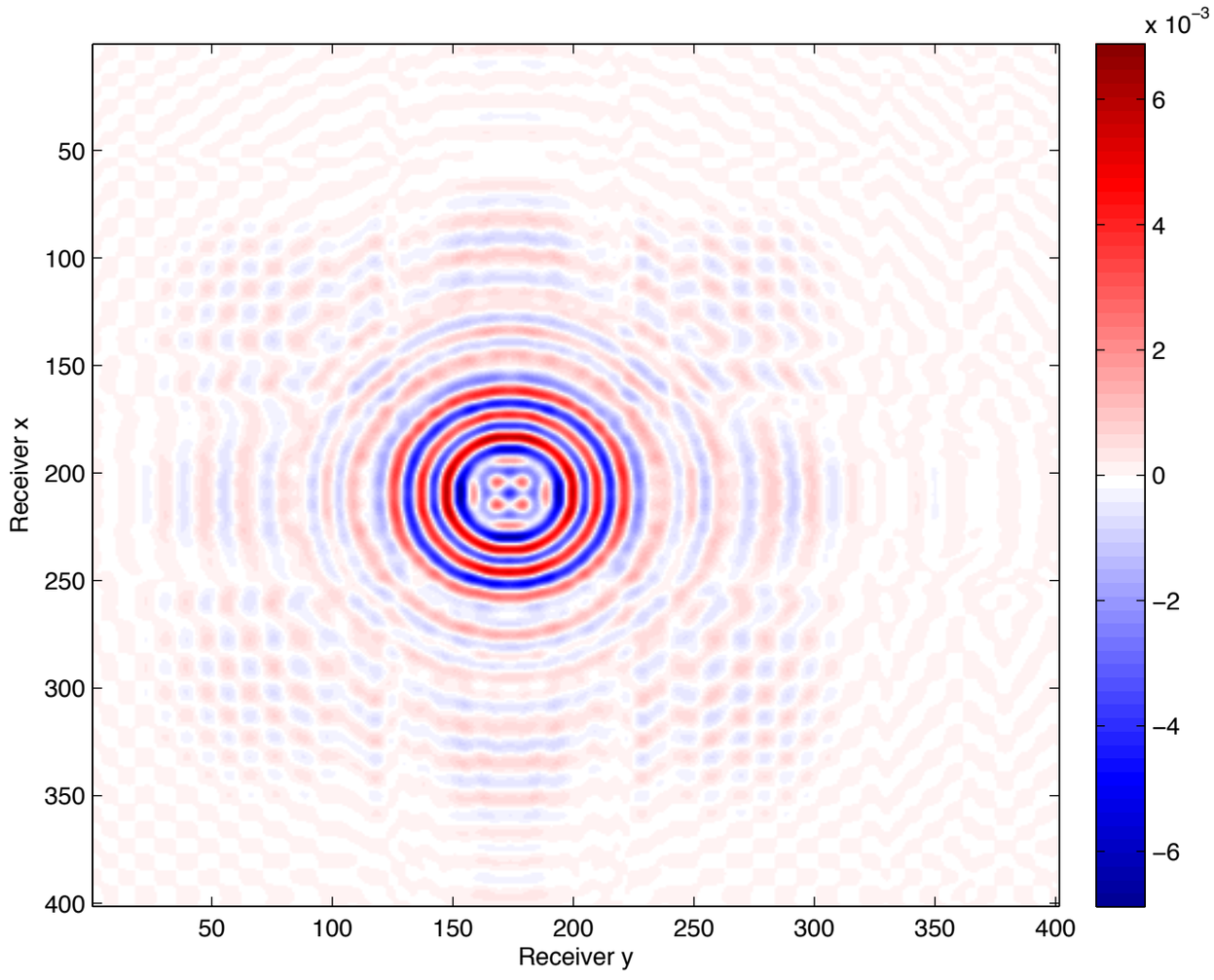
Interpolated data - SNR 12.6 dB

LR Matrix Completion



Known data

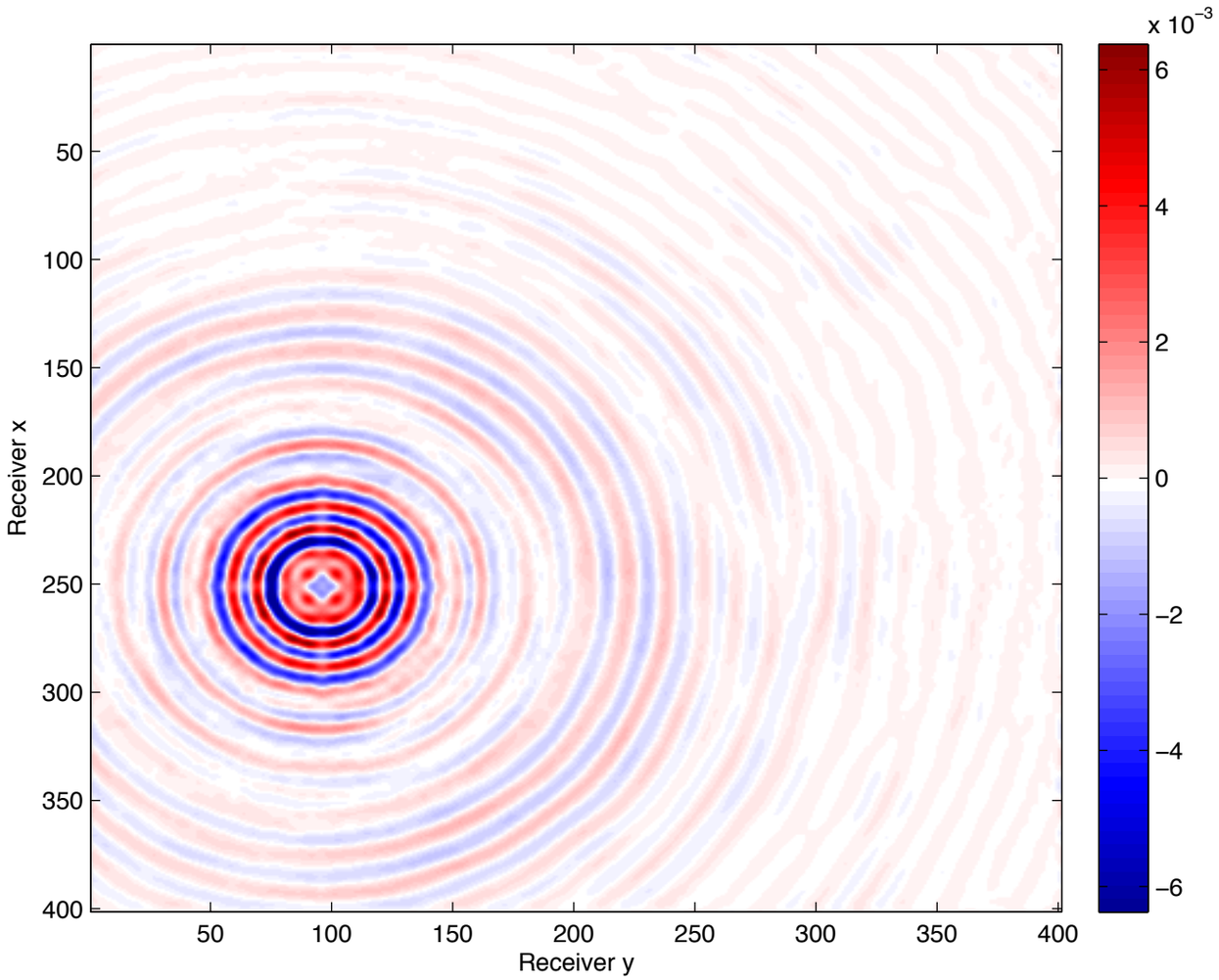
(Src x, Src y) = (36,30)



Interpolated data -

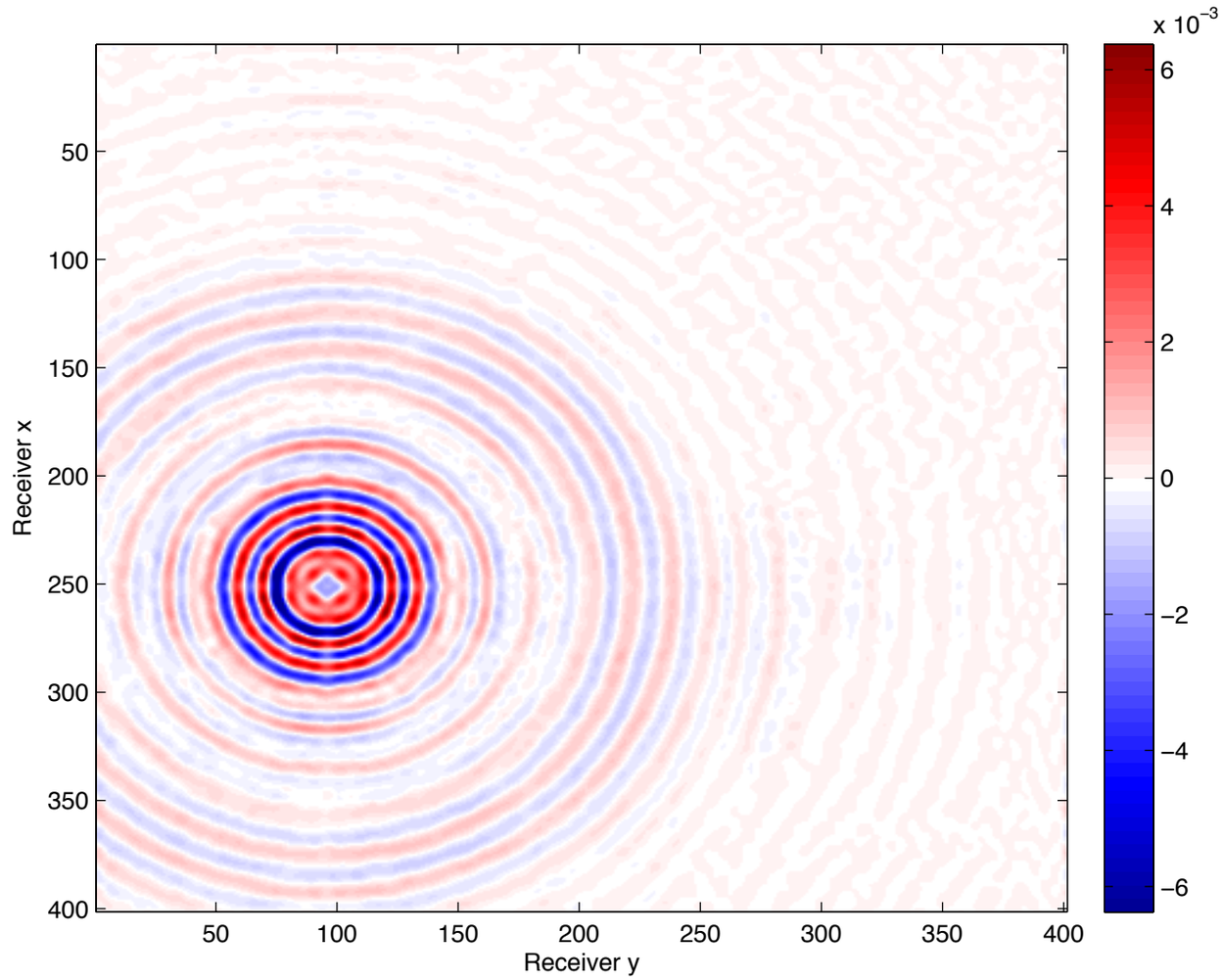
SNR 8.49 dB

Source rank 20 - 500 iterations



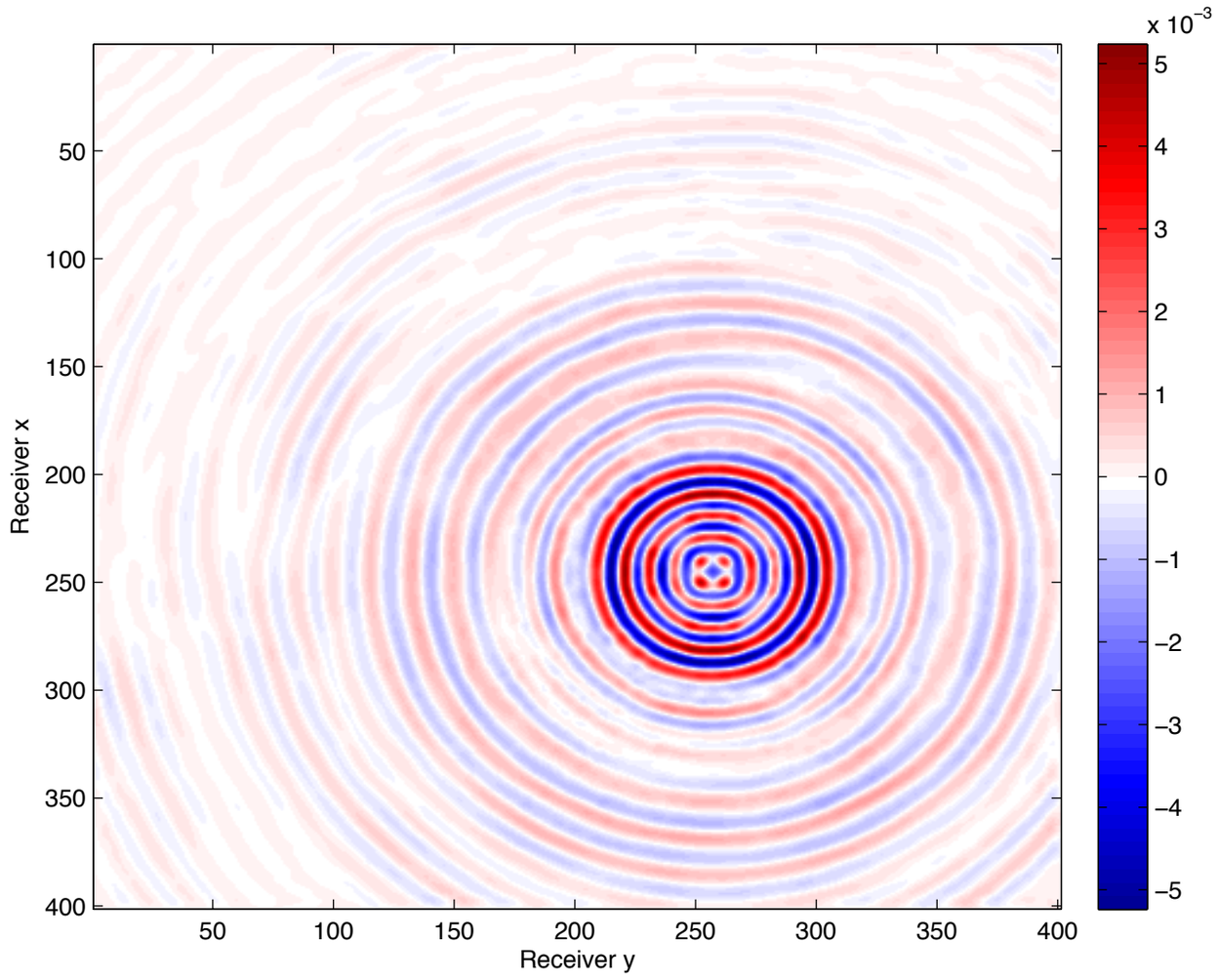
Known data

(Src x, Src y) = (43,17)



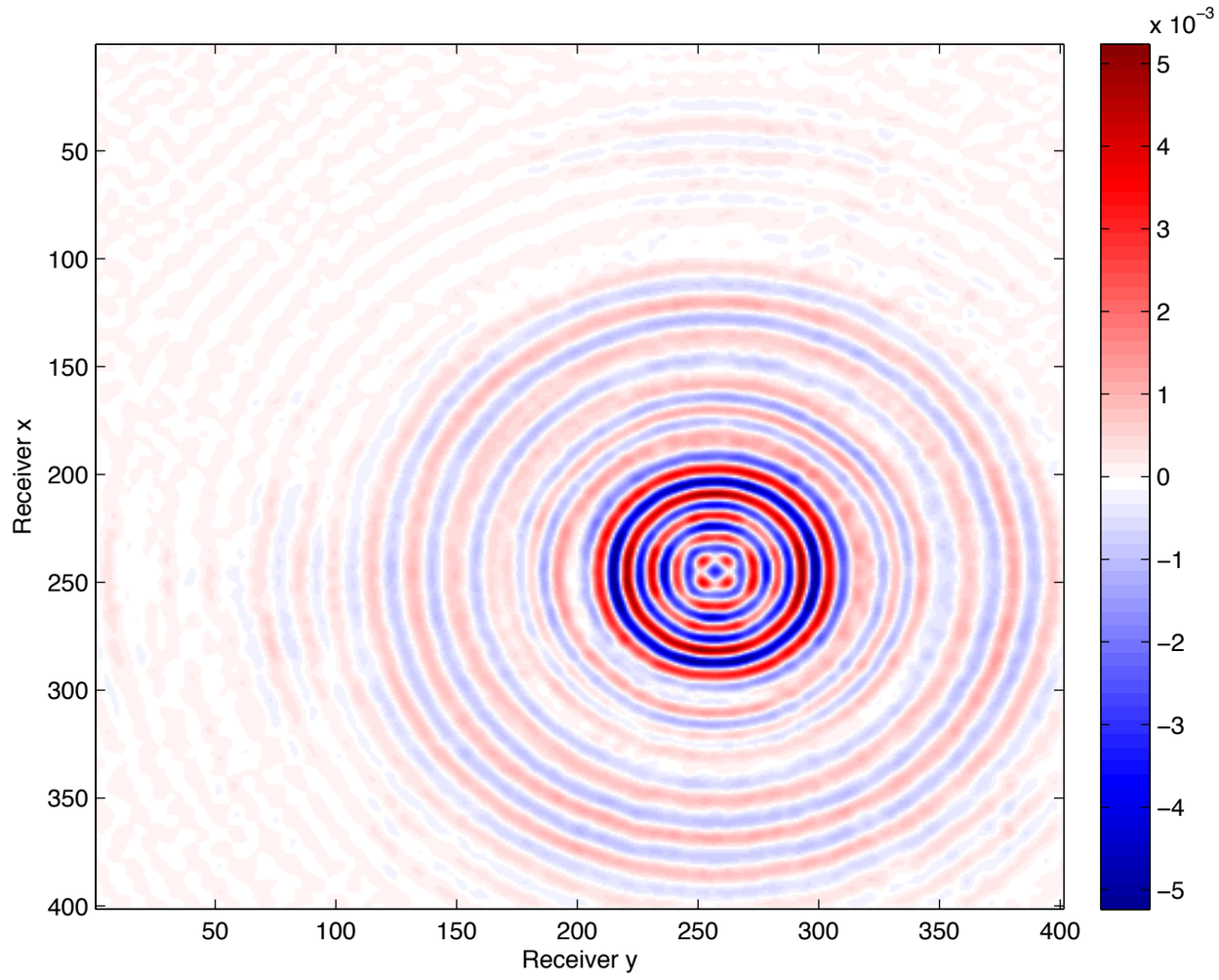
Interpolated data - SNR 13.9 dB

Source rank 20 - 500 iterations



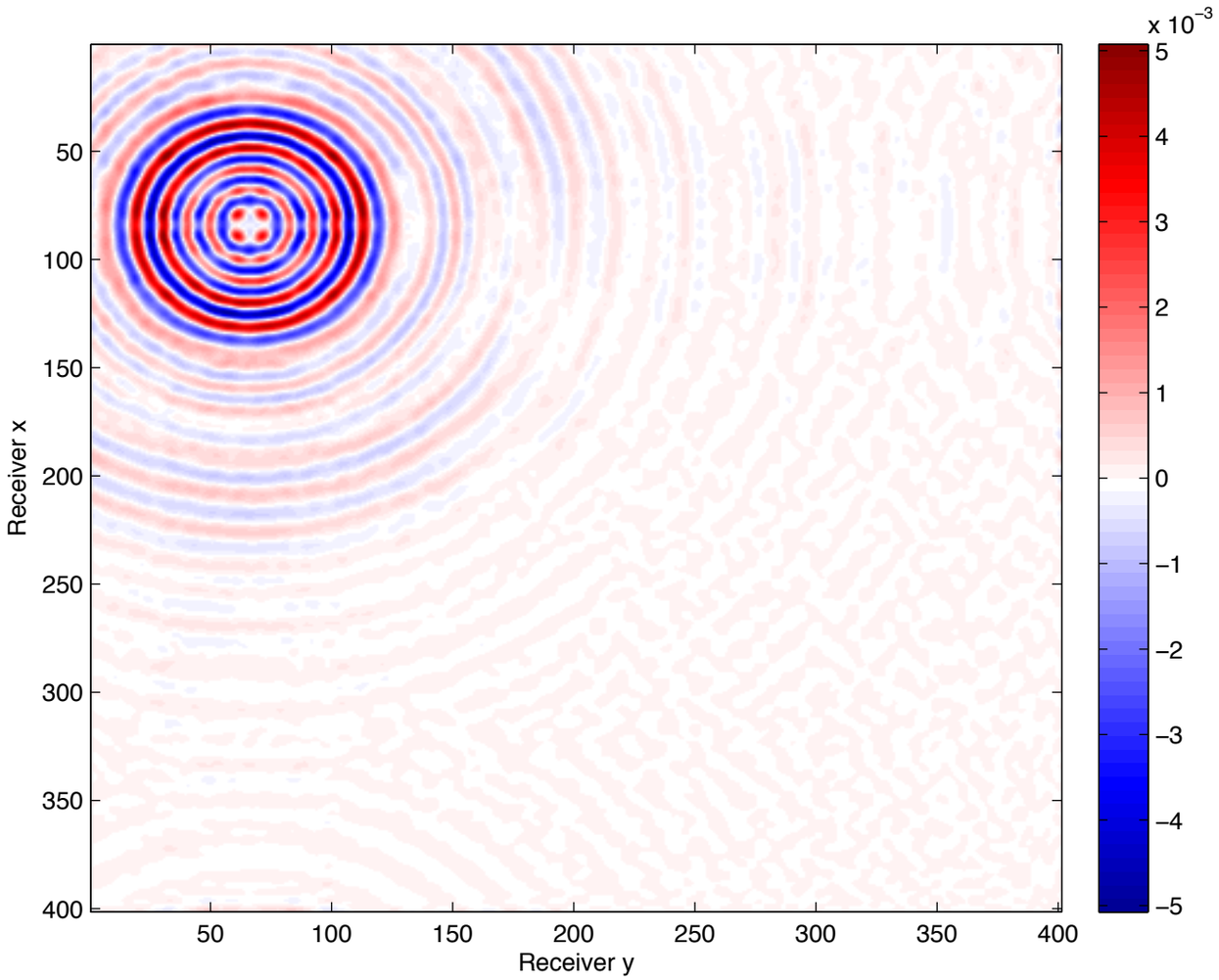
Known data

(Src x, Src y) = (42,44)



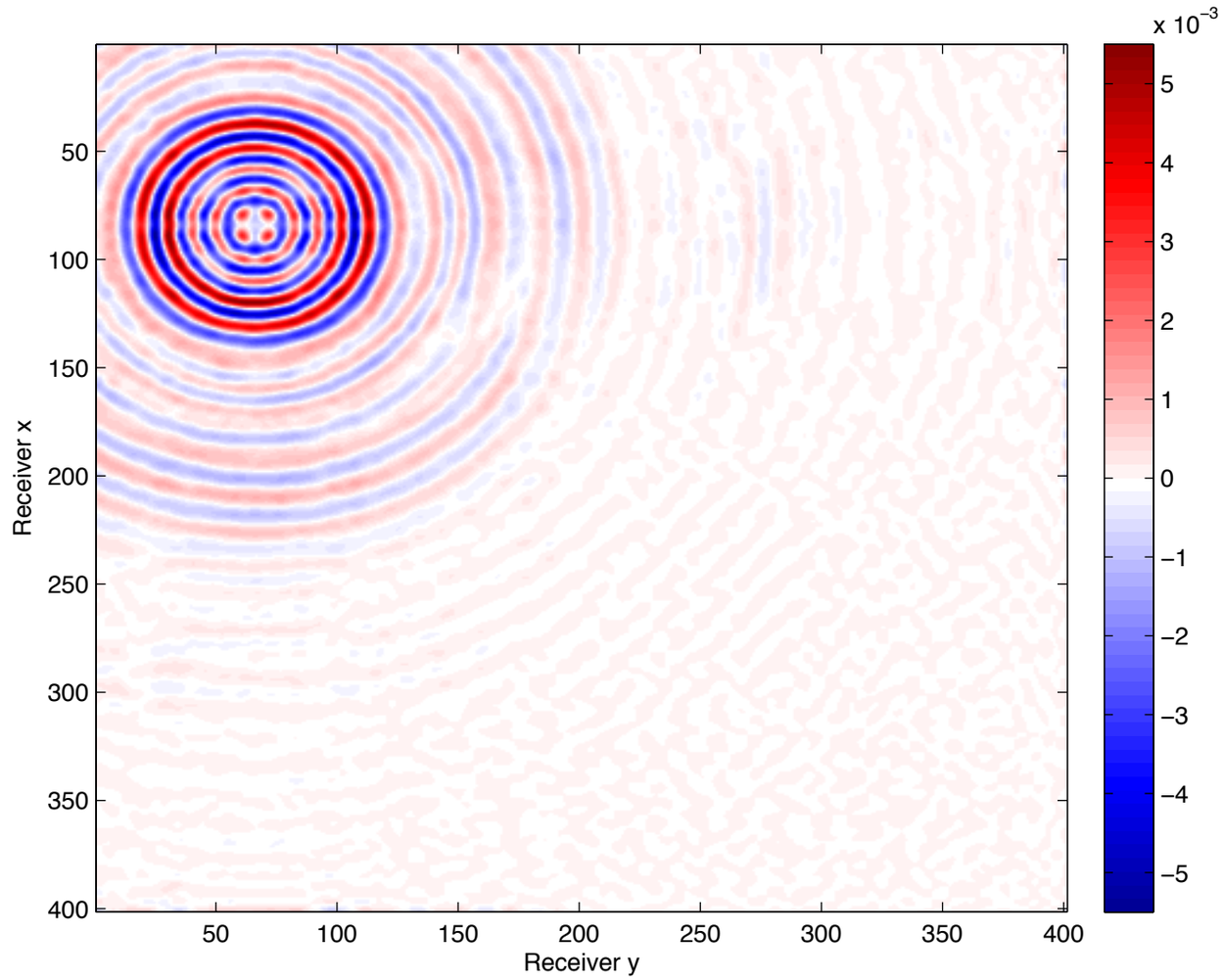
Interpolated data - SNR 13.2 dB

Source rank 5 vs source rank 20



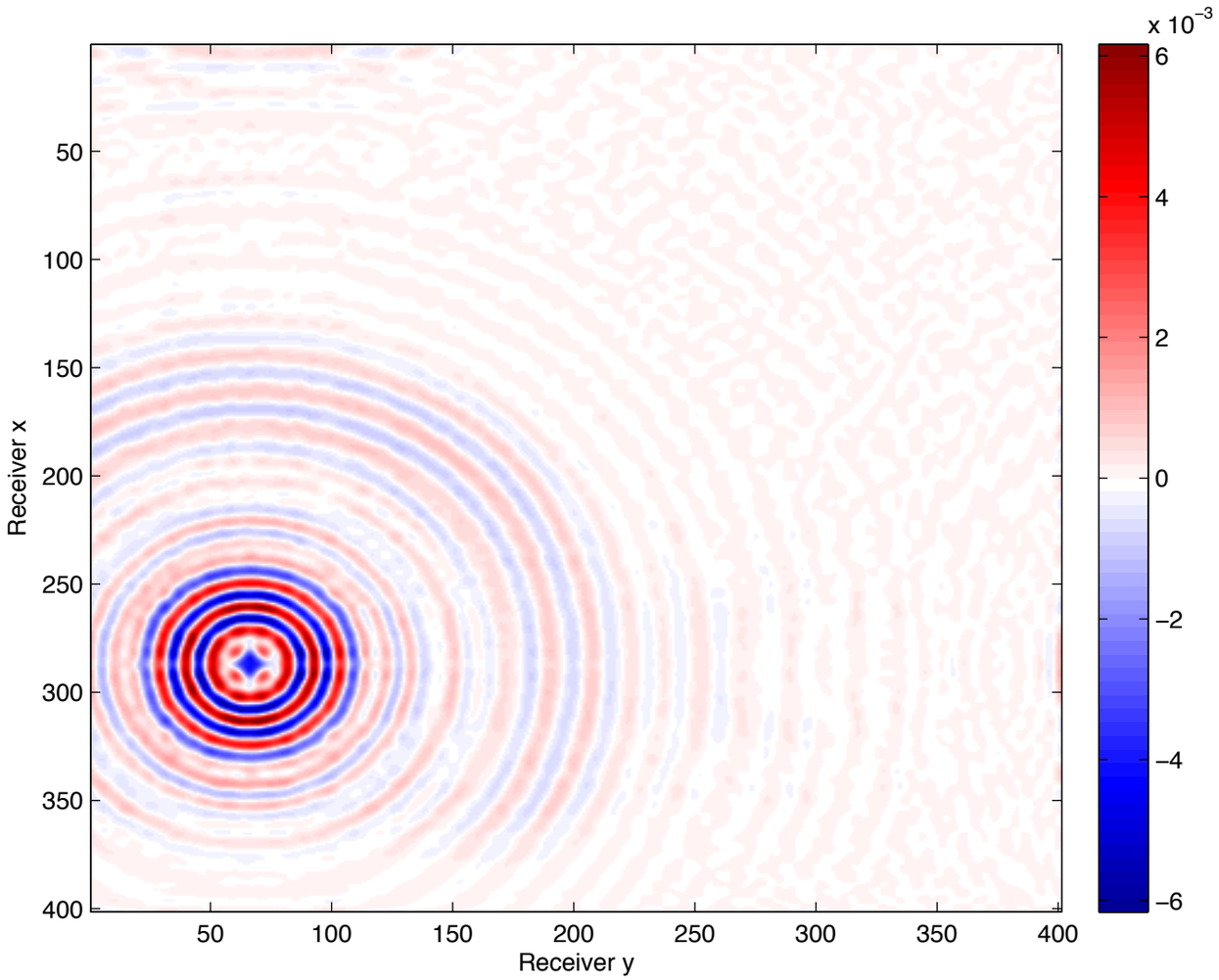
Rank 5

(Src x, Src y) = (15,12)



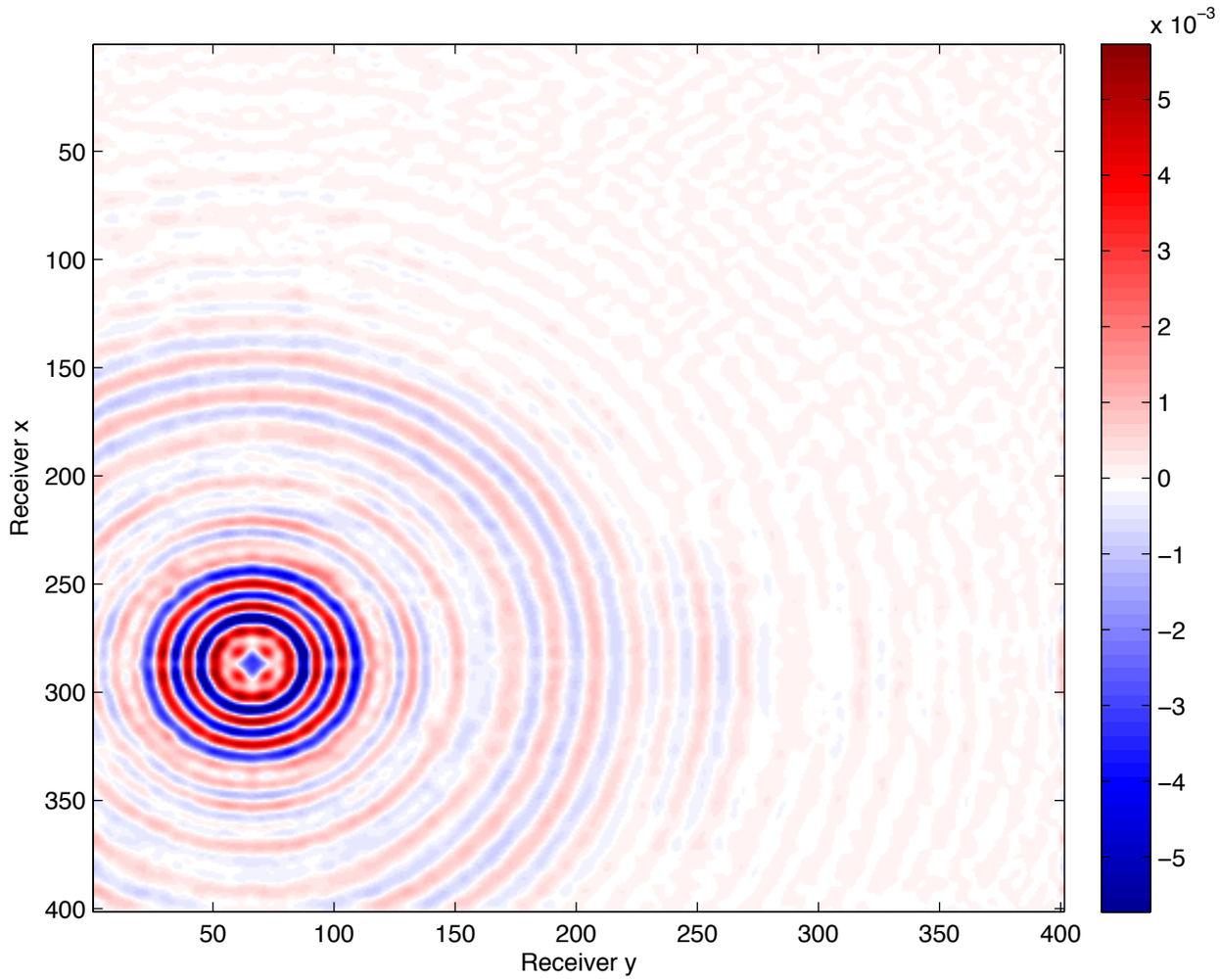
Rank 20

Source rank 5 vs source rank 20



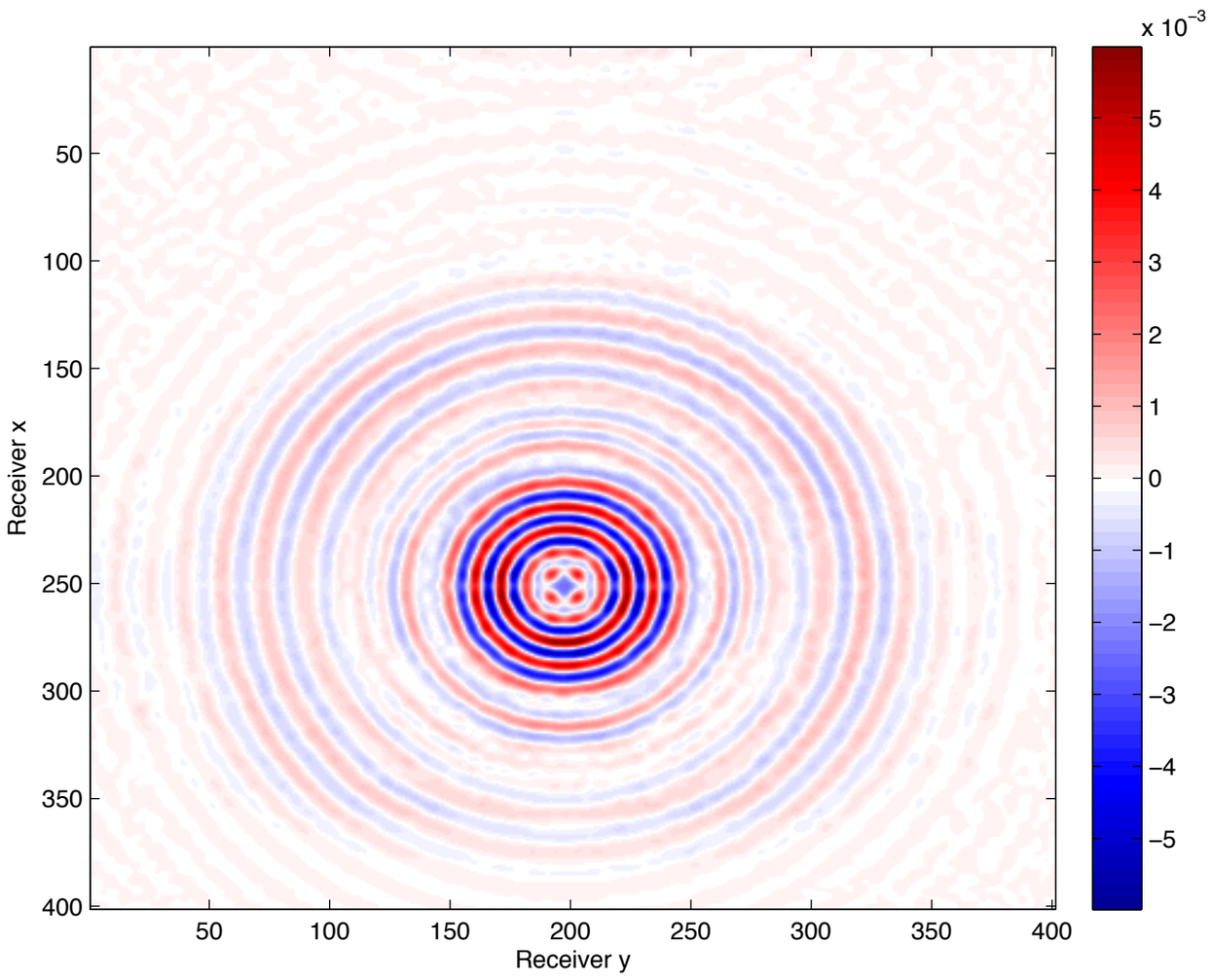
Rank 5

(Src x, Src y) = (49,12)



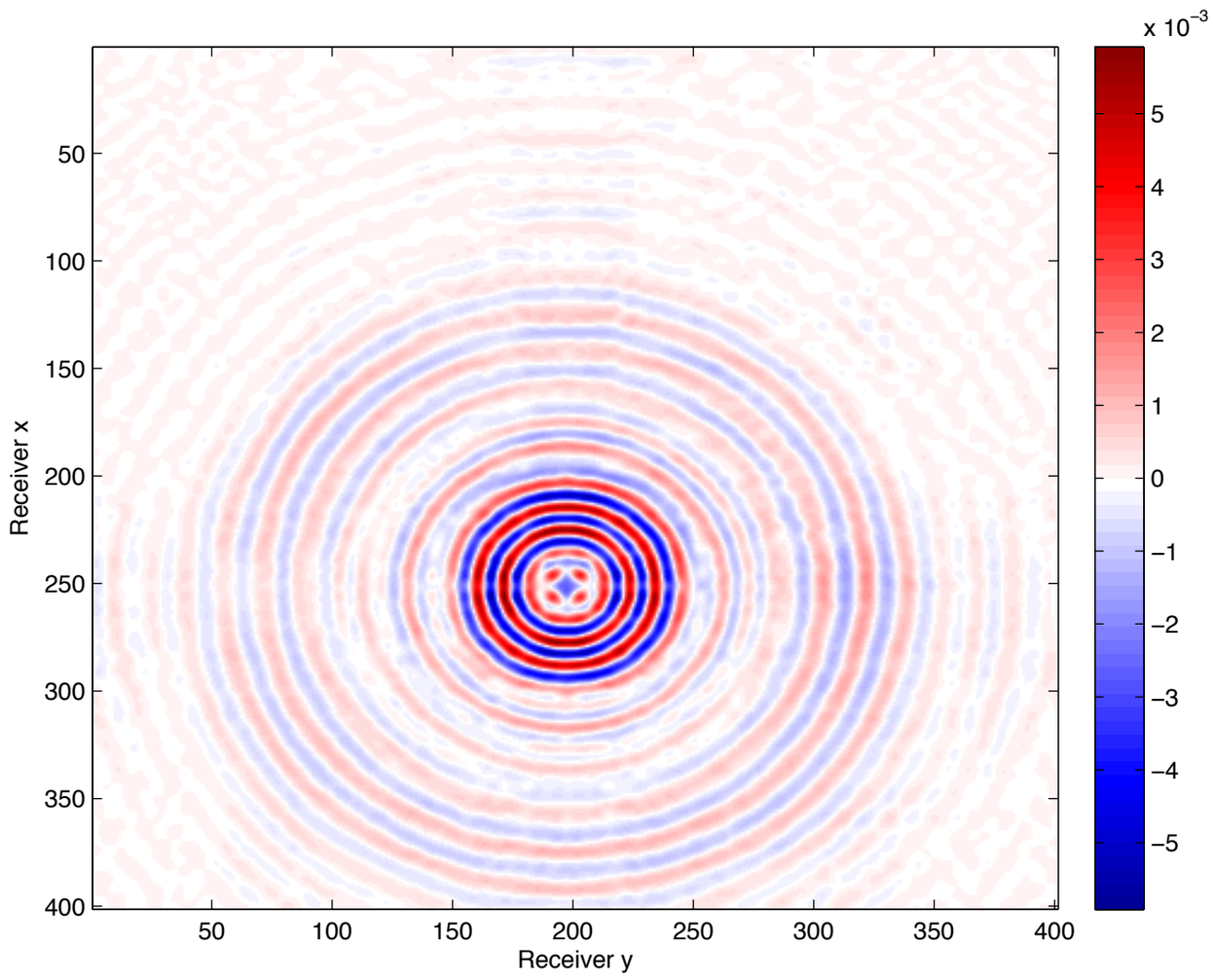
Rank 20

Source rank 5 vs source rank 20



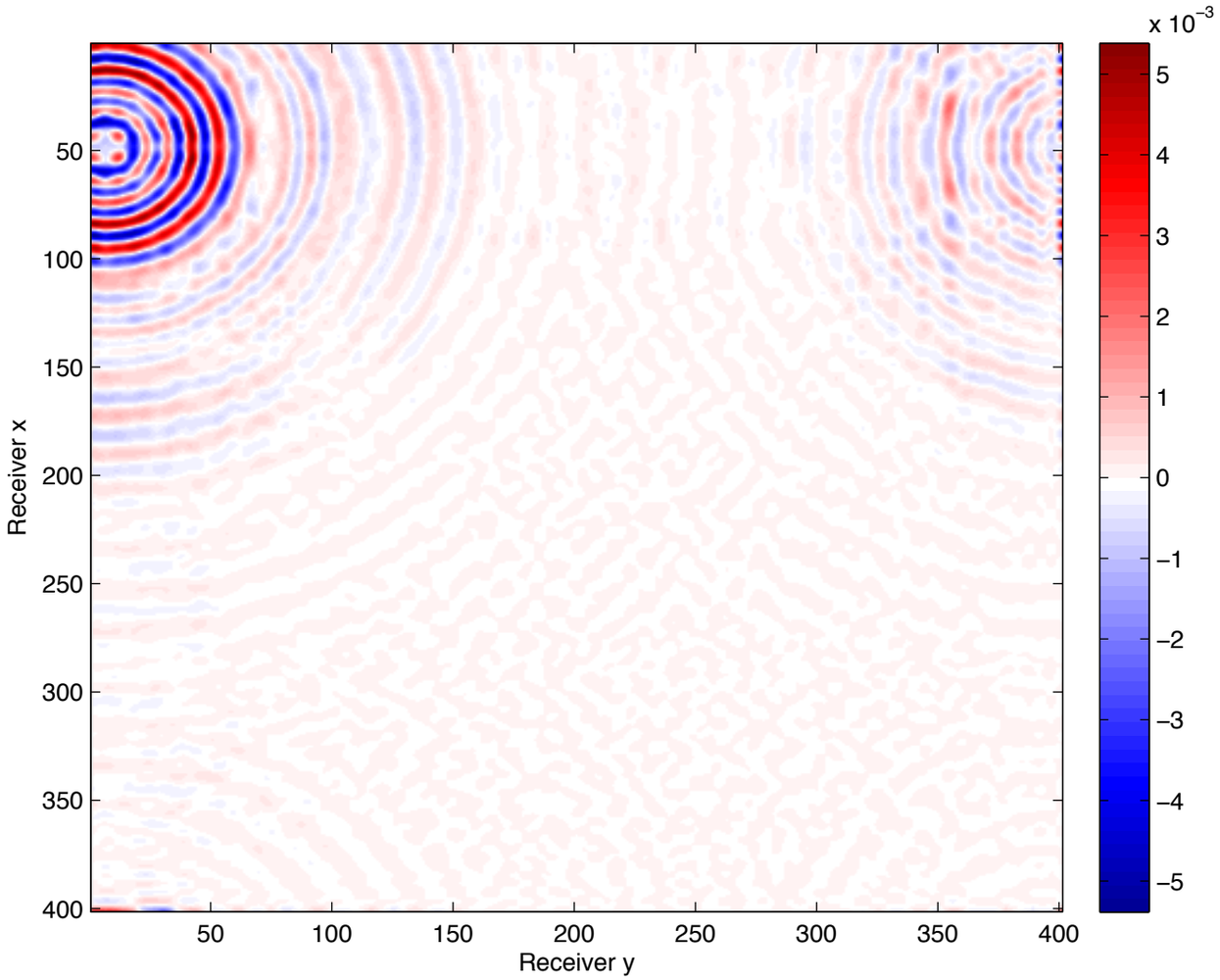
Rank 5

(Src x, Src y) = (43,34)



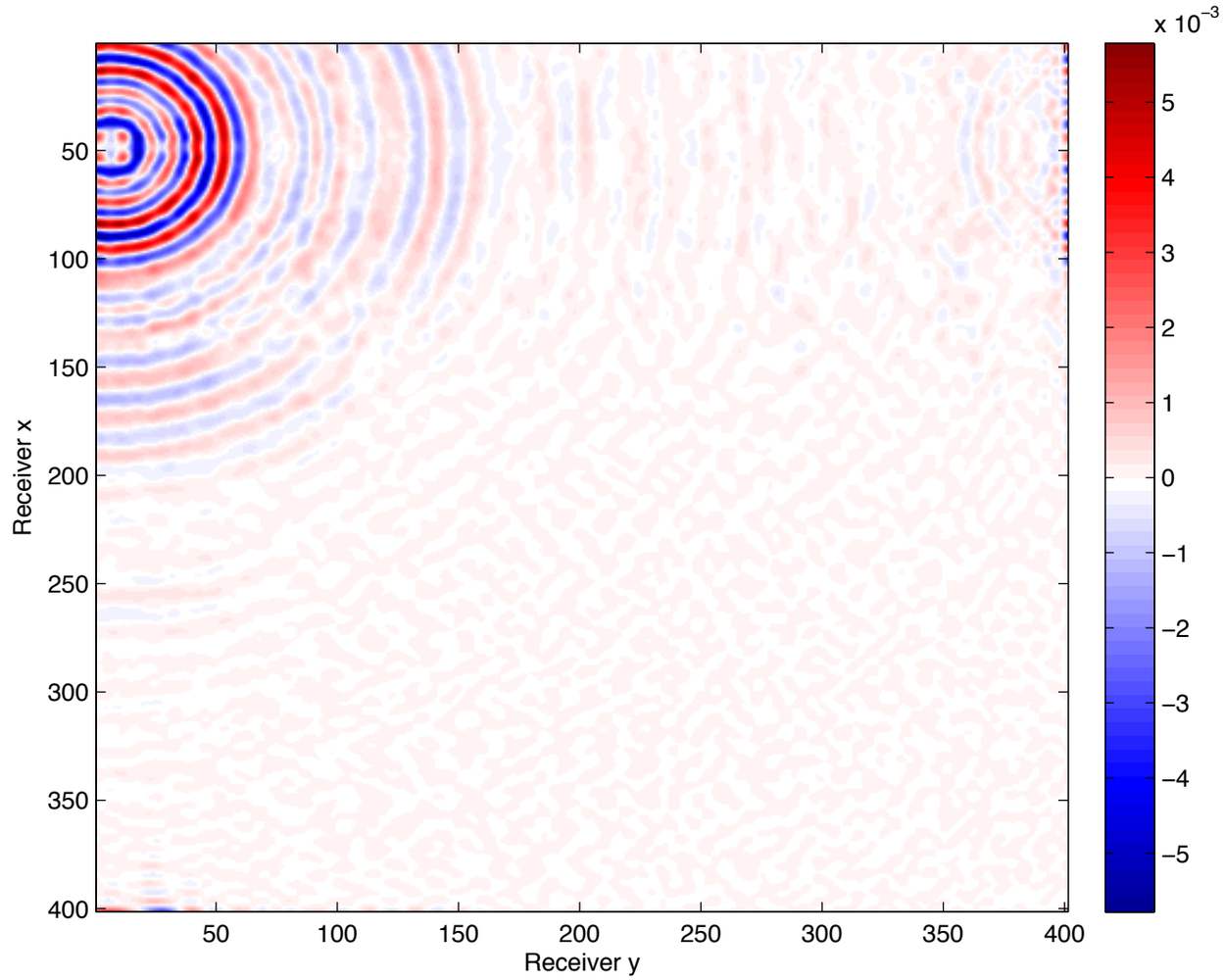
Rank 20

Source rank 5 vs source rank 20



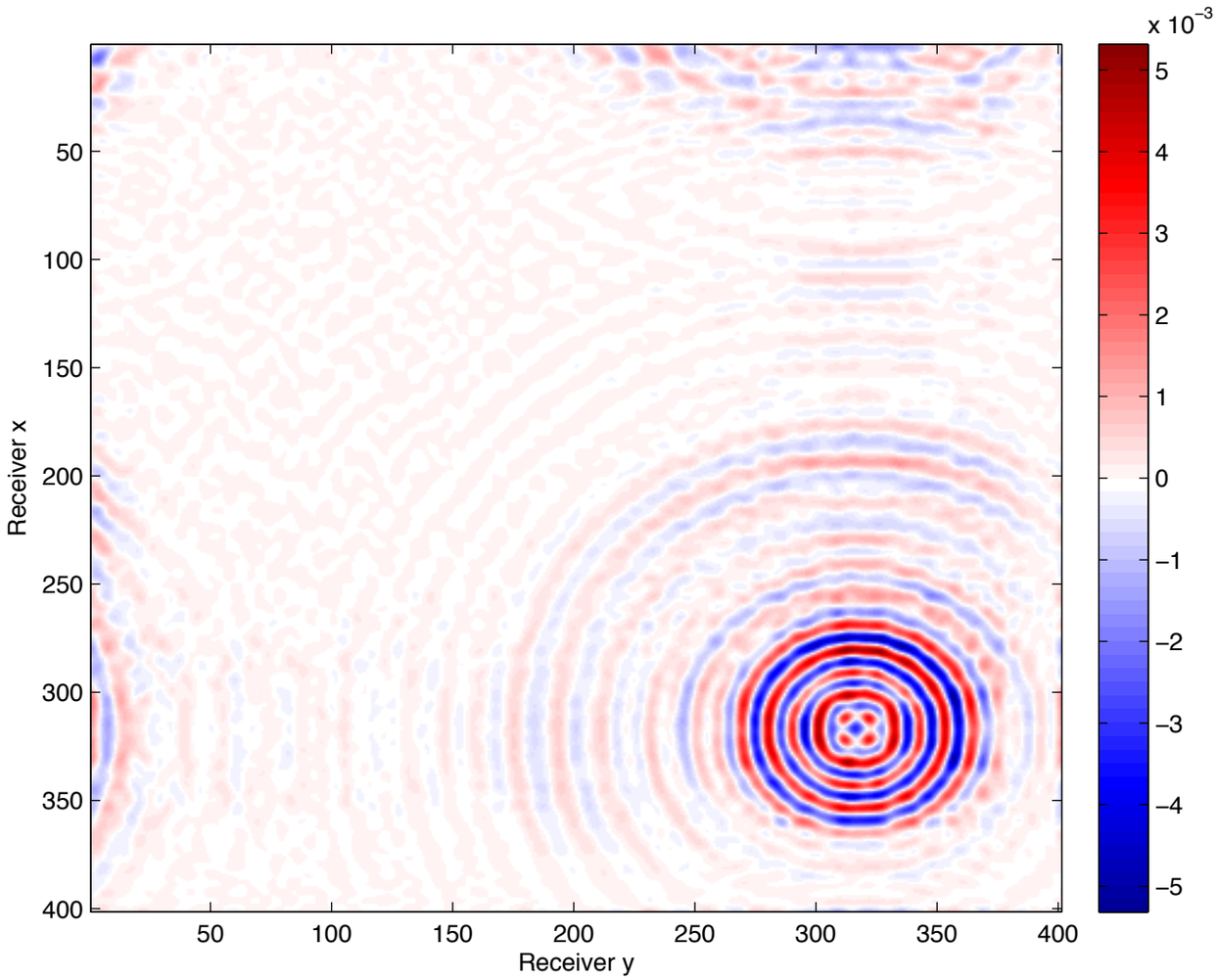
Rank 5

(Src x, Src y) = (9,2)



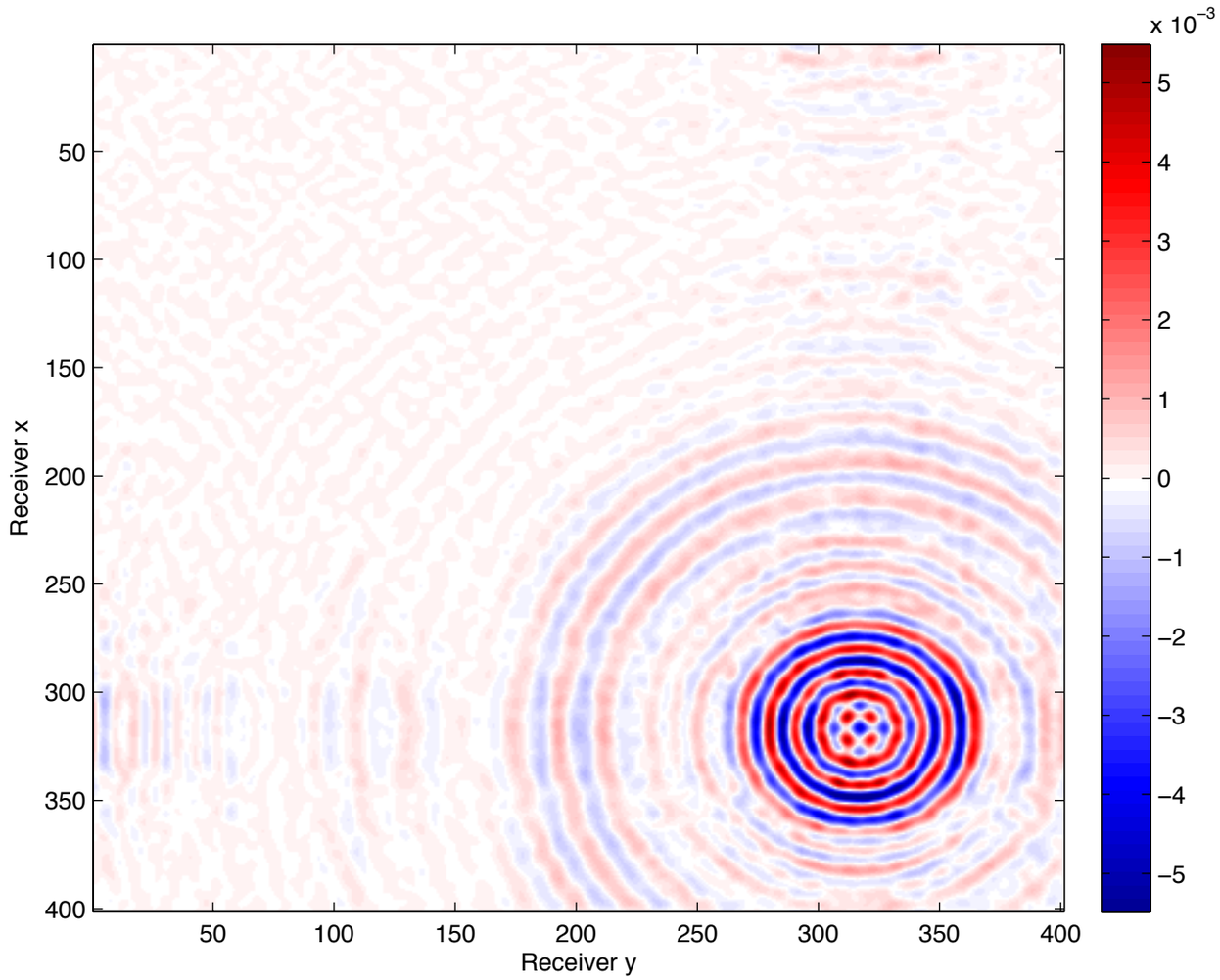
Rank 20

Source rank 5 vs source rank 20



Rank 5

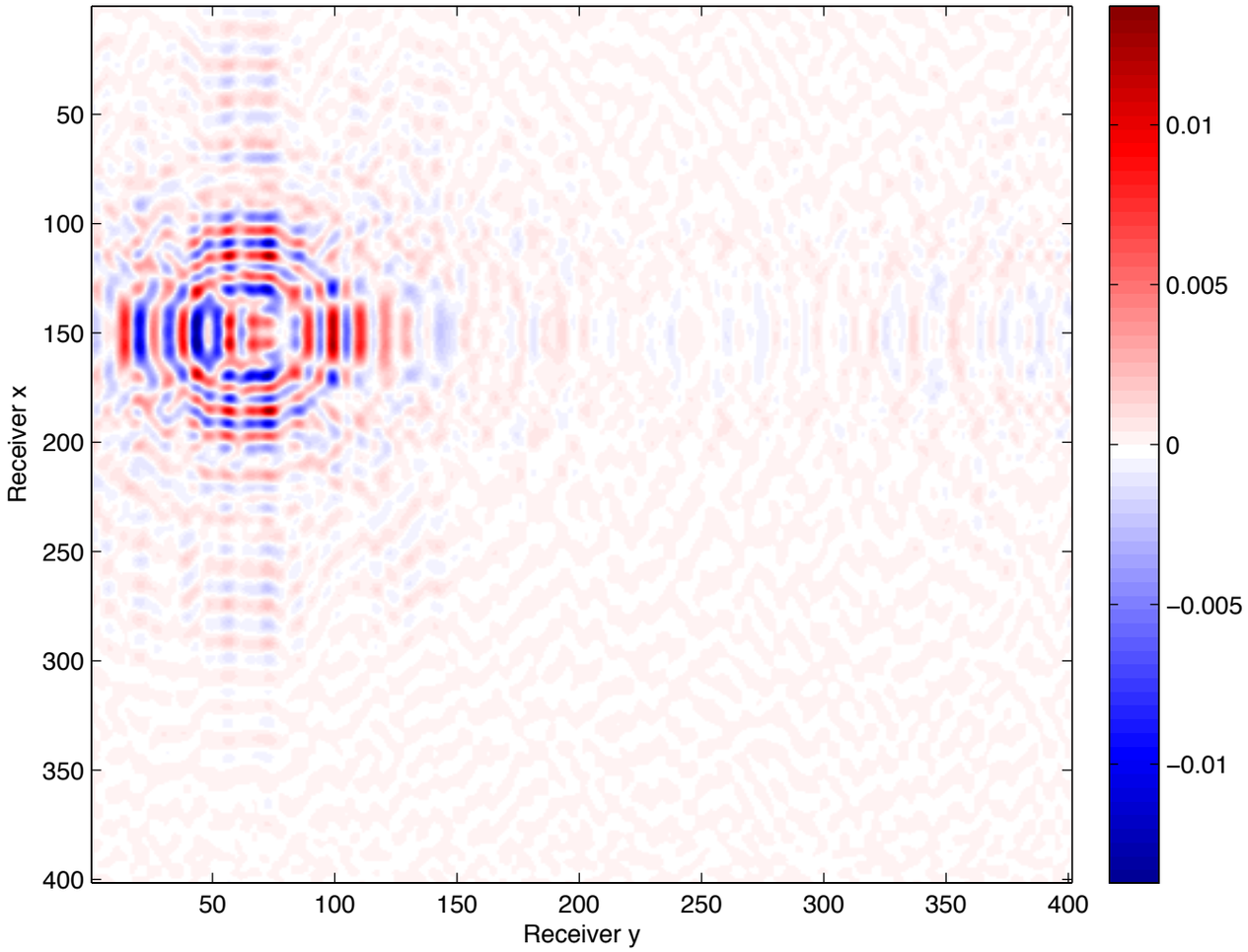
(Src x, Src y) = (54,54)



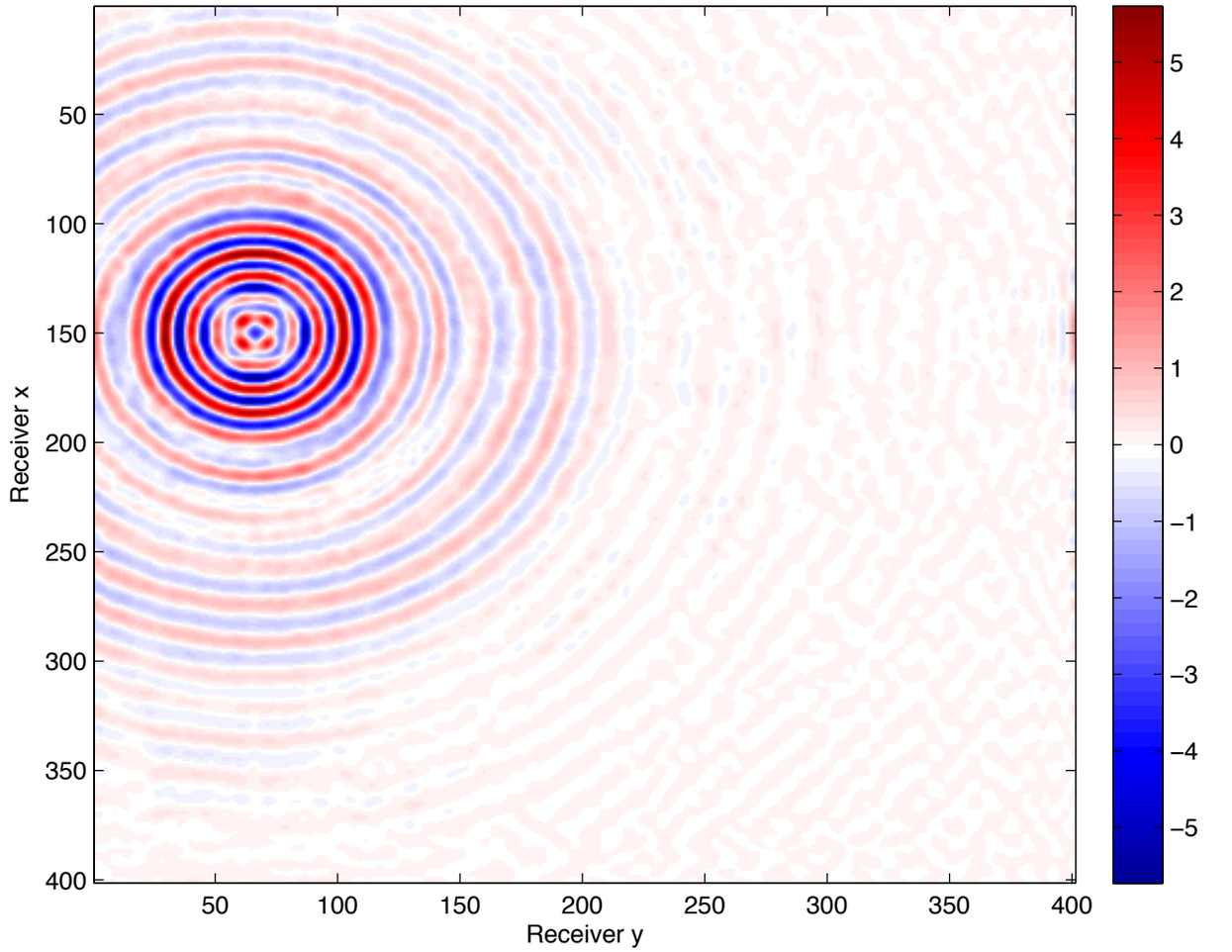
Rank 20

Source rank 20 - 500 iterations

Missing sufficient data spread



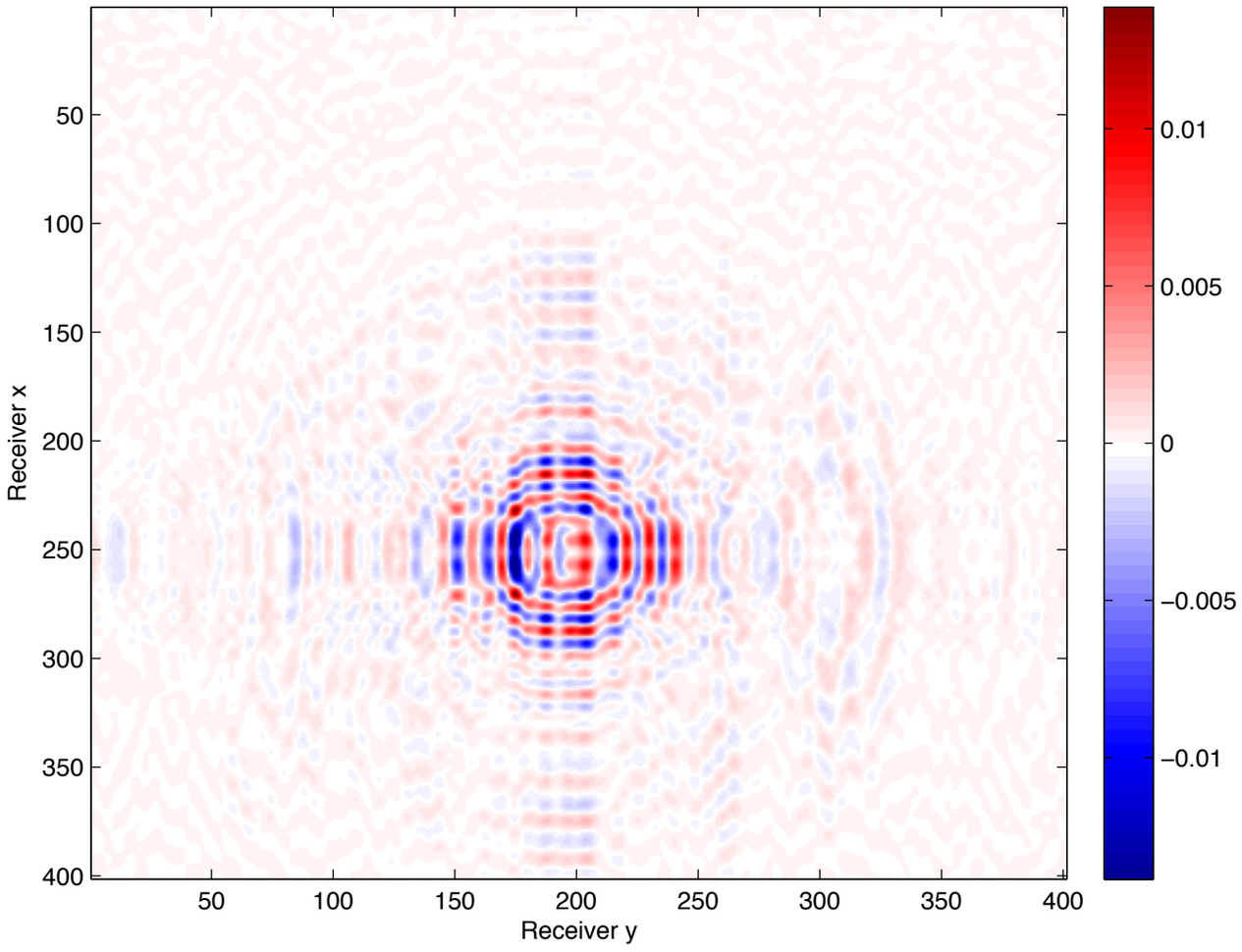
Interpolated data
(Src x, Src y) = (26,11)



Interpolated data
(Src x, Src y) = (26,12)

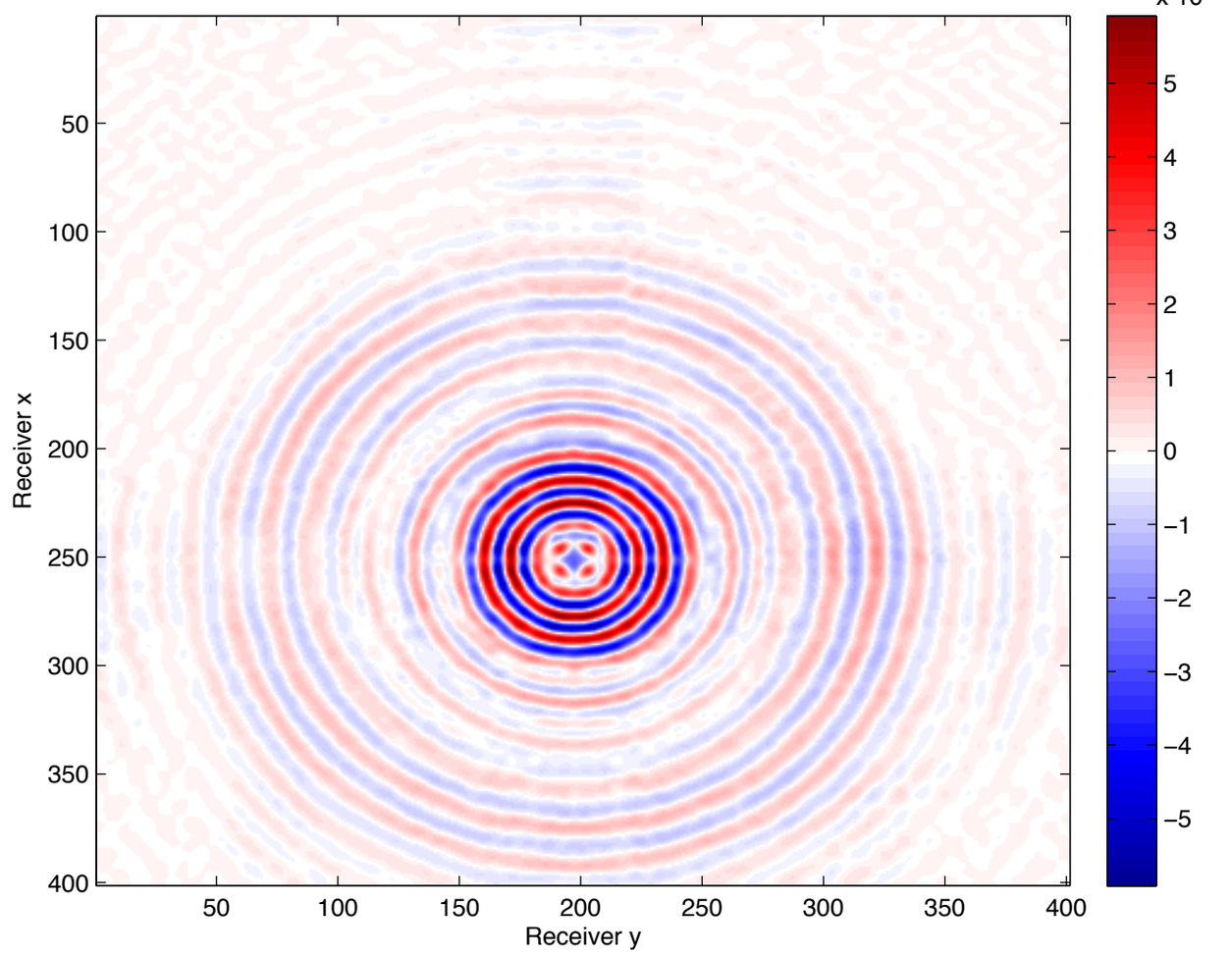
Source rank 20 - 500 iterations

Missing sufficient data spread



Interpolated data

(Src x, Src y) = (43,33)



Interpolated data

(Src x, Src y) = (43,34)

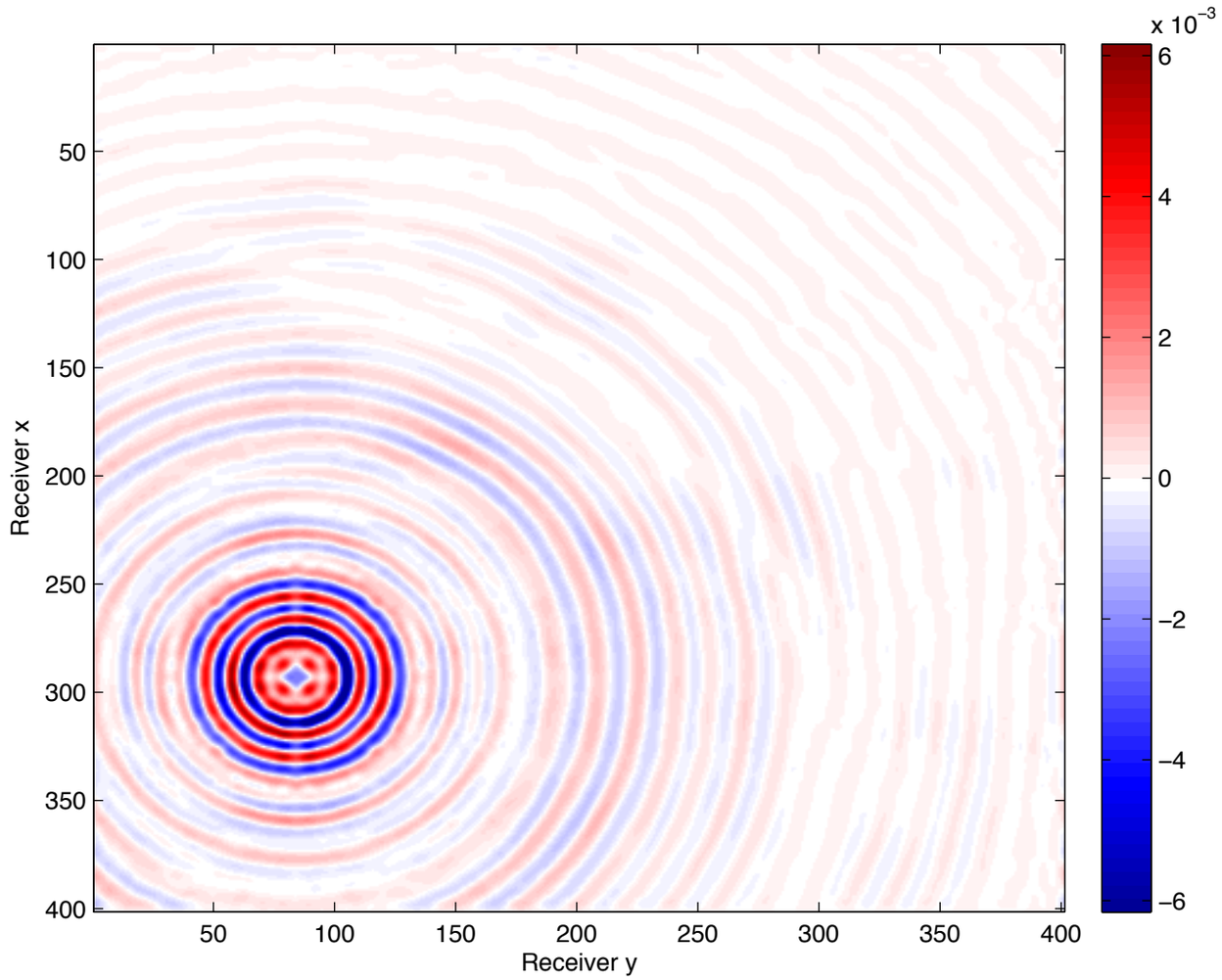
Synthetic BG Data

- Currently we are only trying to minimize the energy misfit between our model + the data we have (as well as choosing a rank for our underlying model)
- Our assumption is that the underlying tensor has *quickly*-decaying singular values in different matricizations

Synthetic BG Data

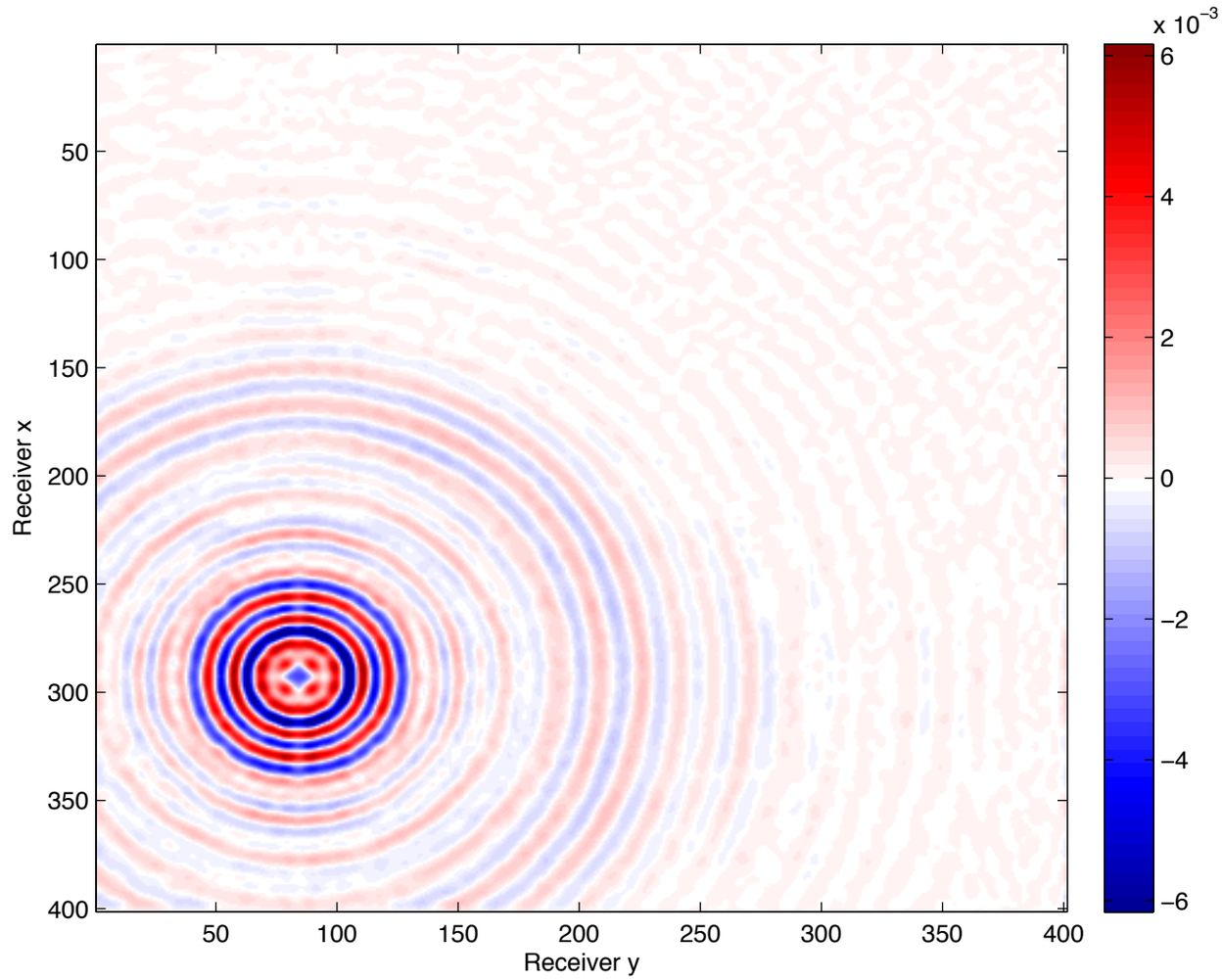
- We can impose a penalty on the decay of the singular values of these matricizations
  - This talk is far too long already, so we'll skip the details here
  - Suffice to say, it can be done, and inexpensively at that (full tensor not used, only interior parameter matrices)

Source rank 20 - 500 iterations



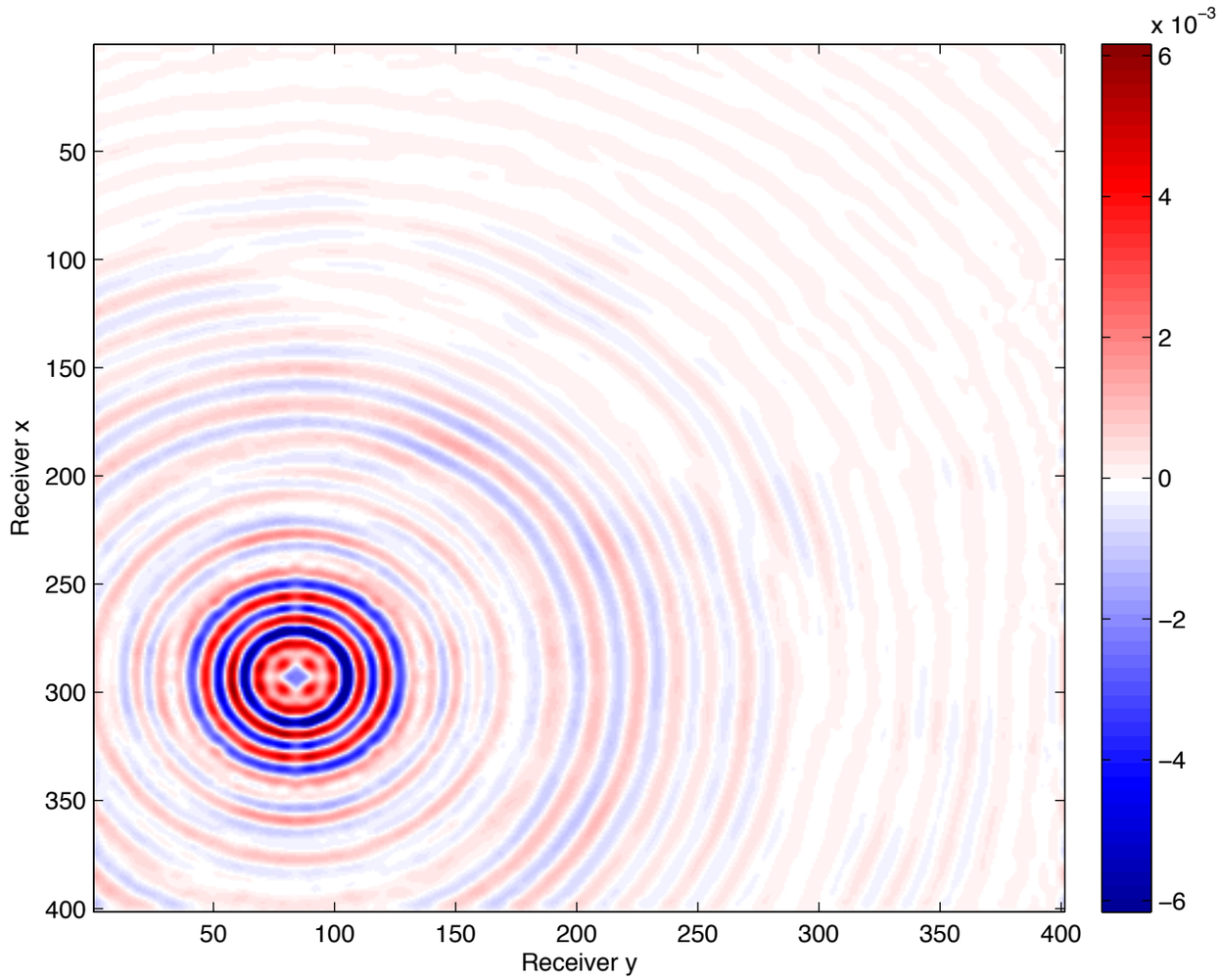
Known data

(Src x, Src y) = (50,15)



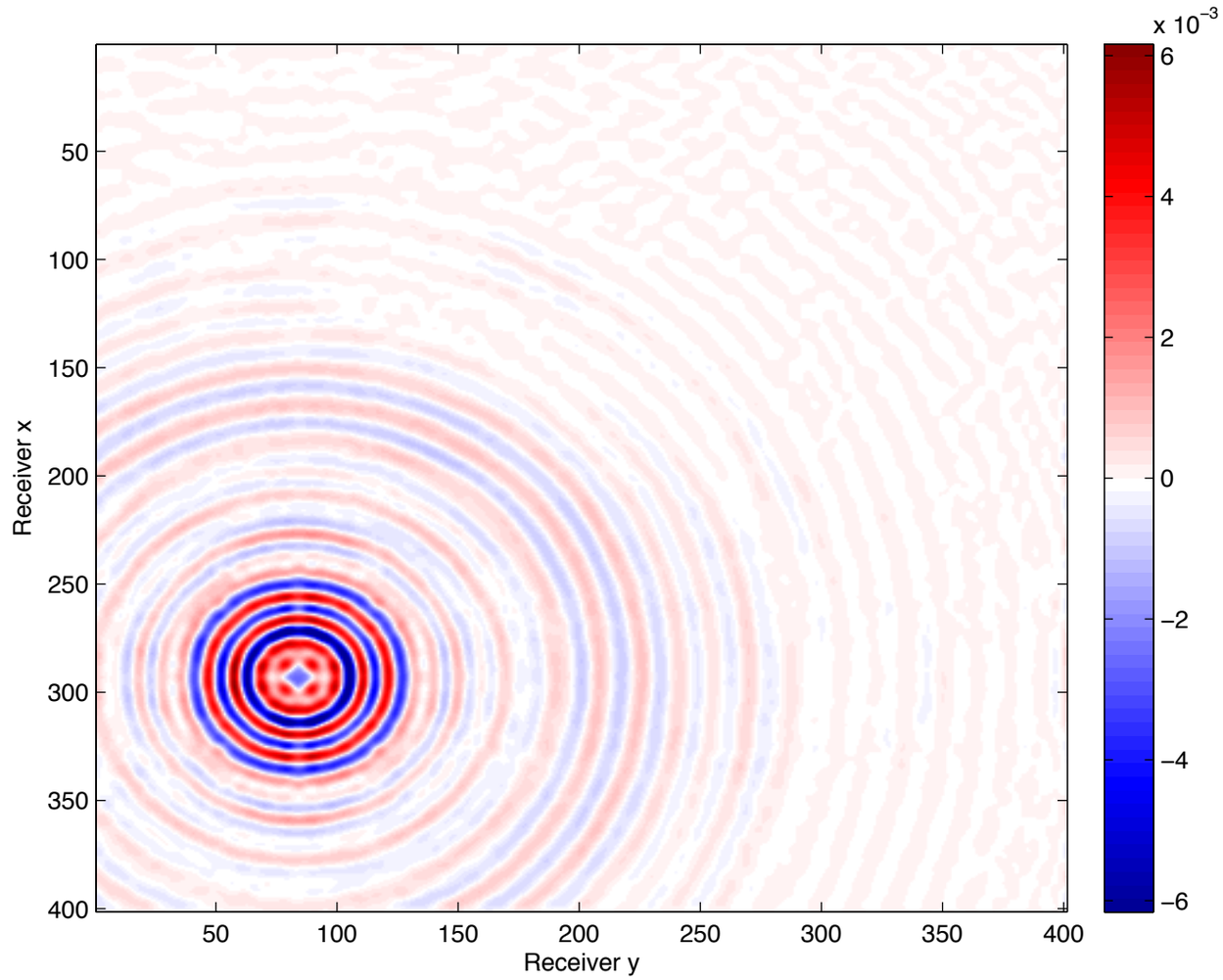
Interpolated data - SNR 13.6 dB

Source rank 20 - 500 iterations



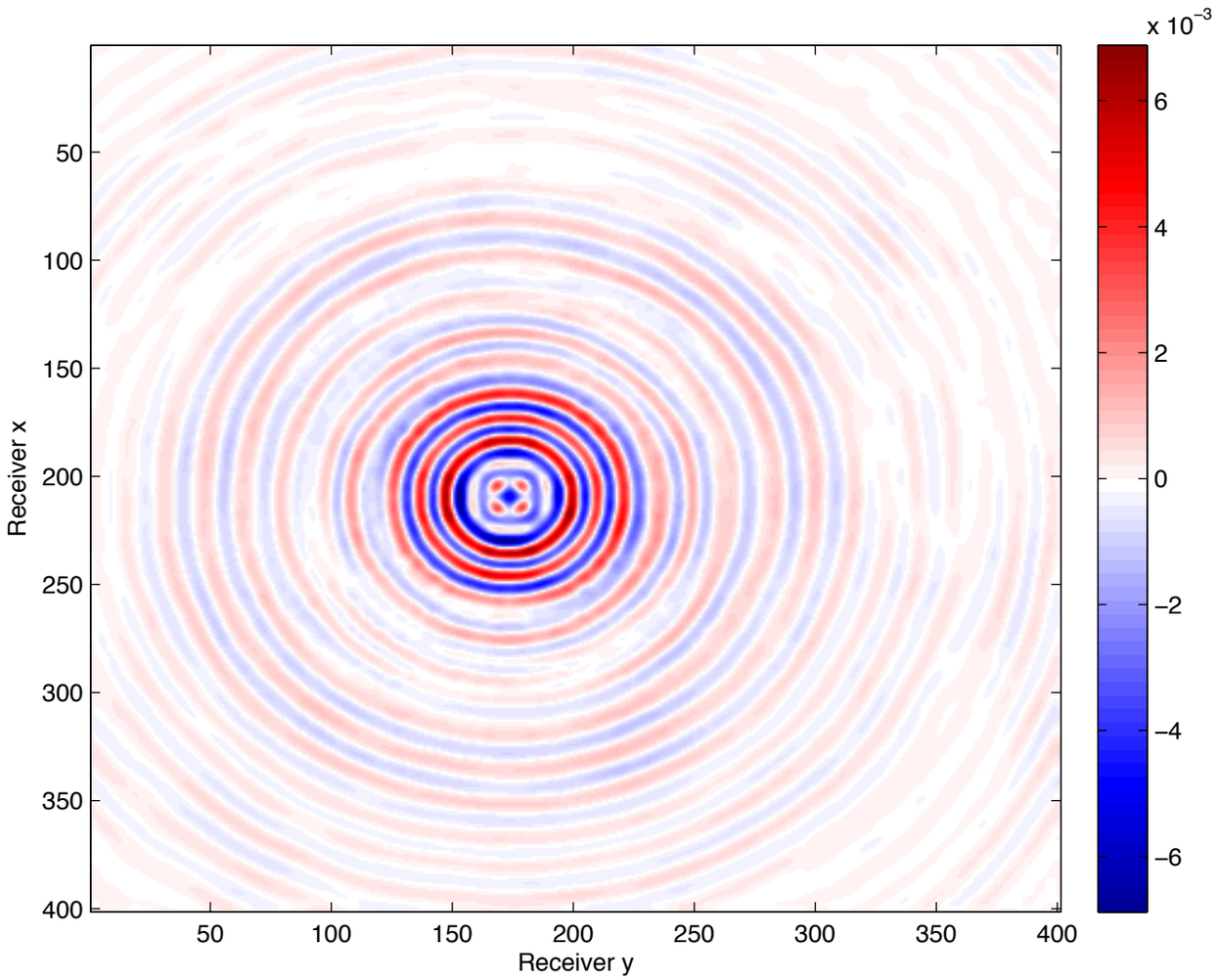
Known data

(Src x, Src y) = (50,15)



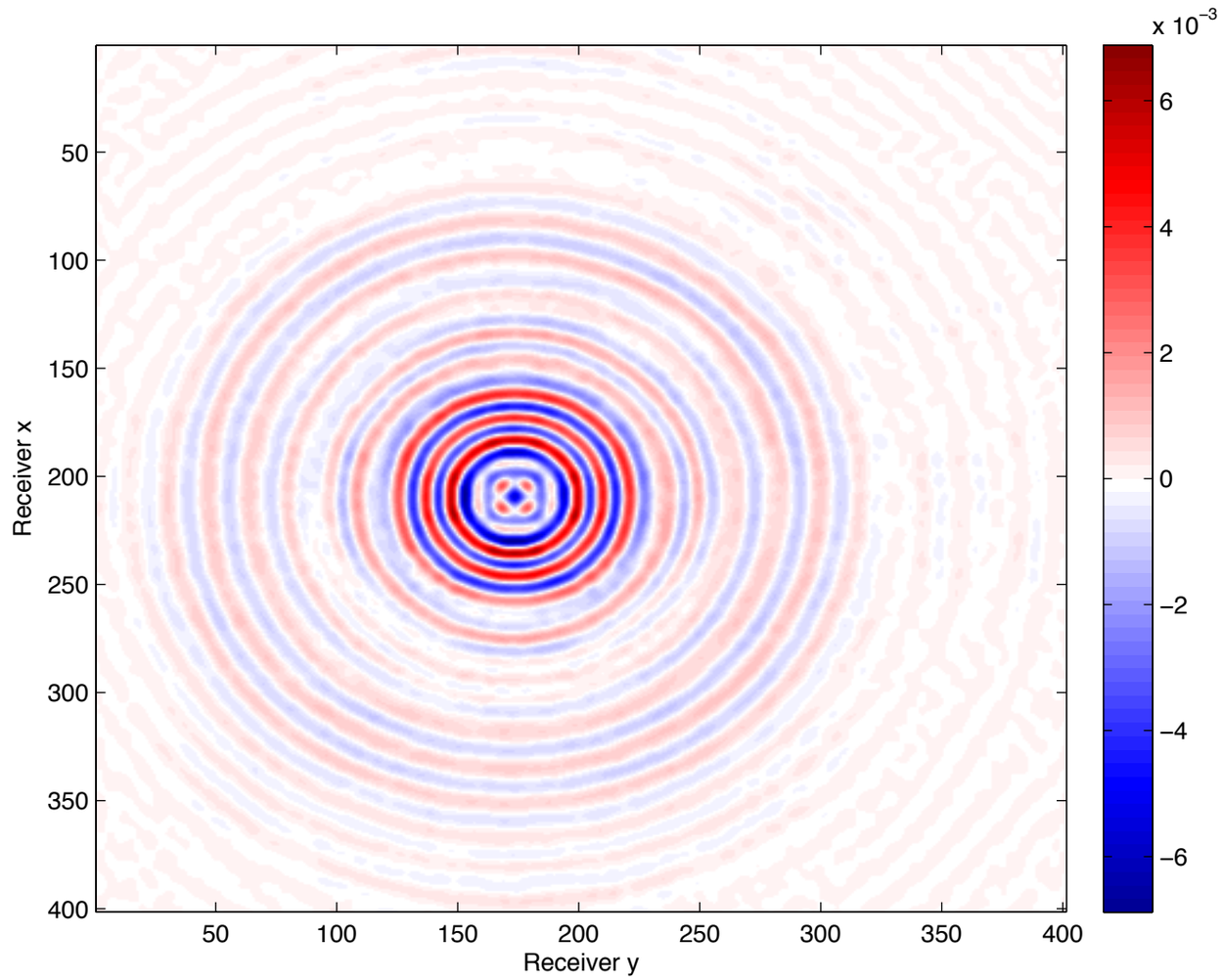
Interpolated data - SNR 15.2 dB

Source rank 20 - 500 iterations



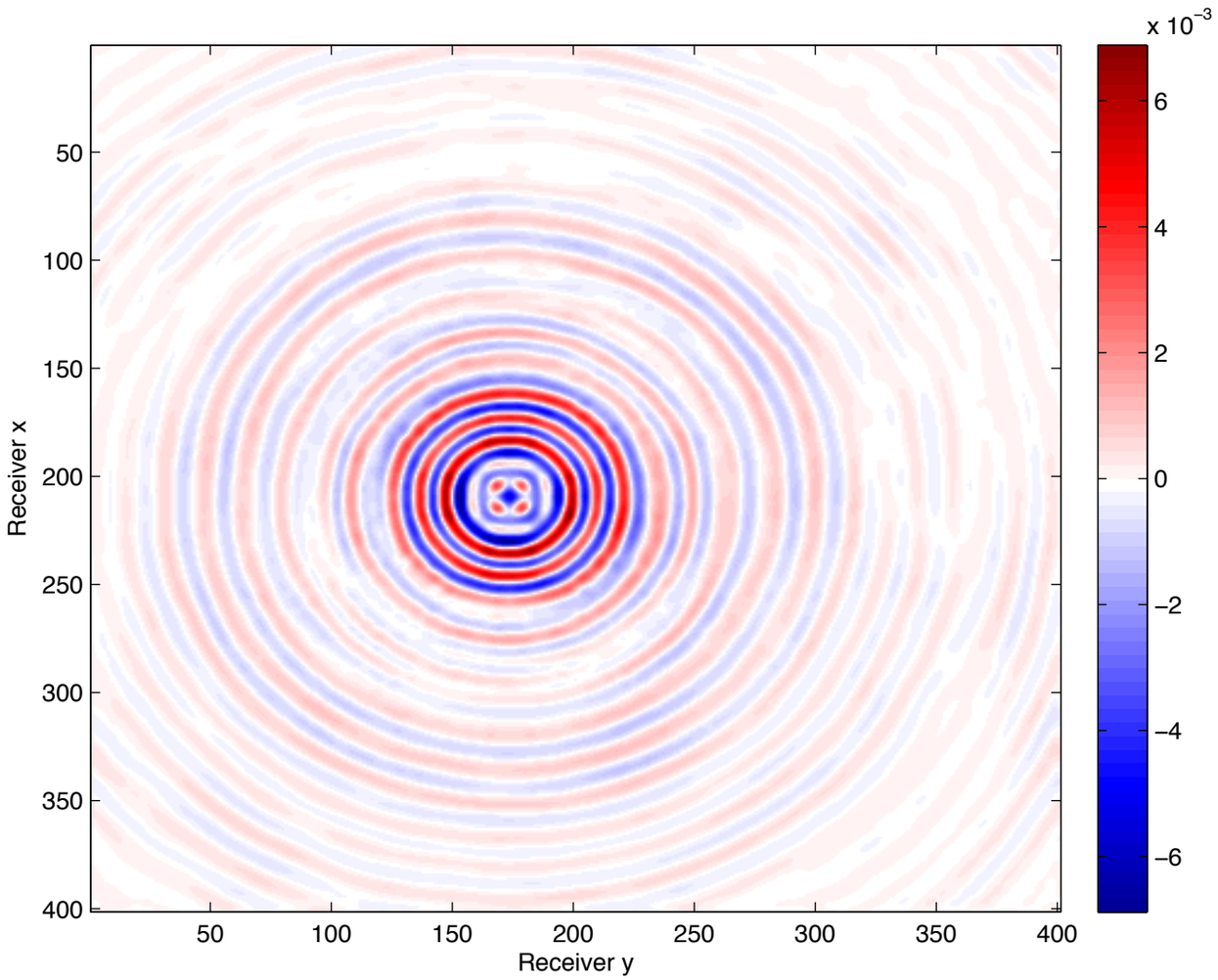
Known data

(Src x, Src y) = (36,30)



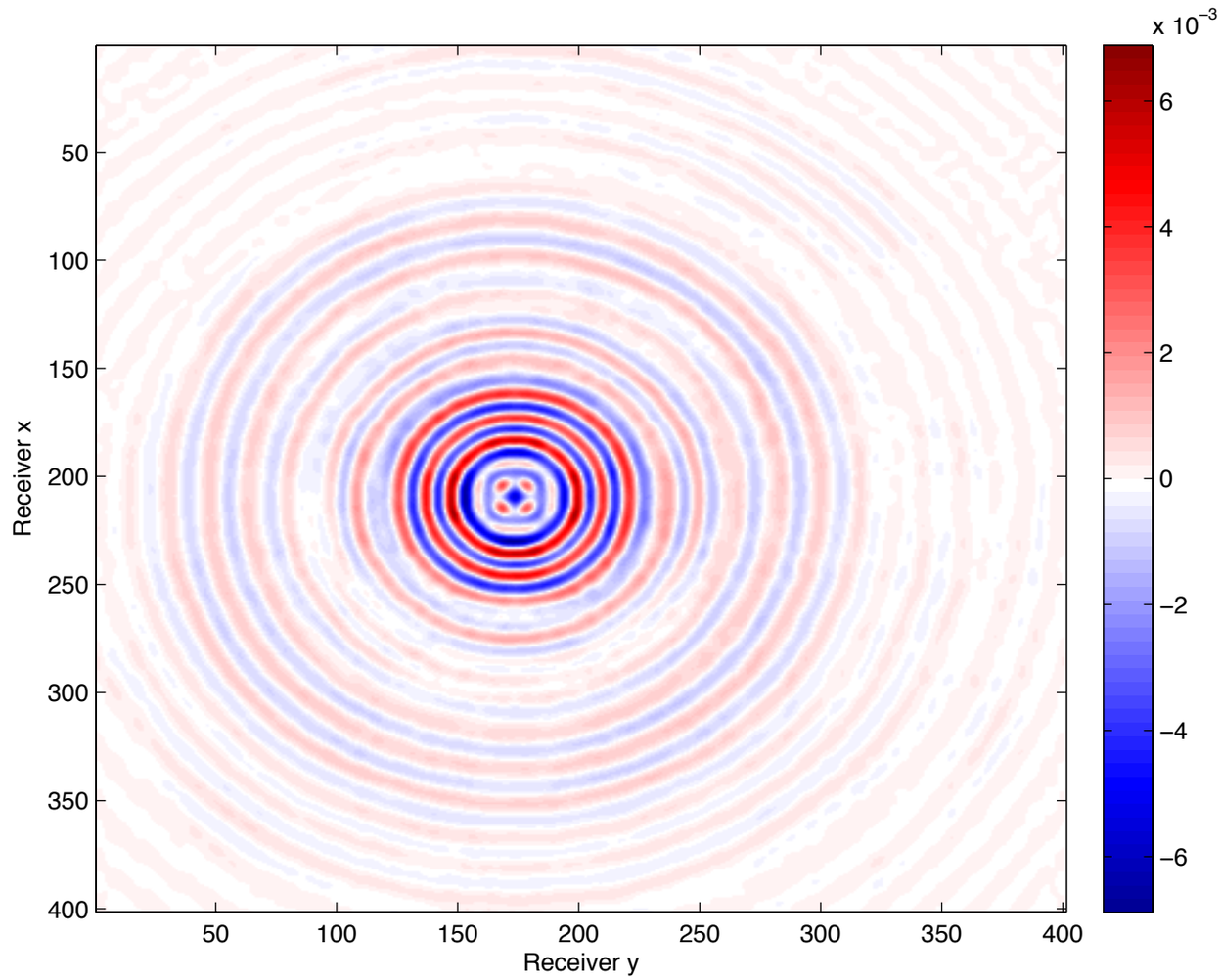
Interpolated data - SNR 12.6 dB

Source rank 20 - Regularization



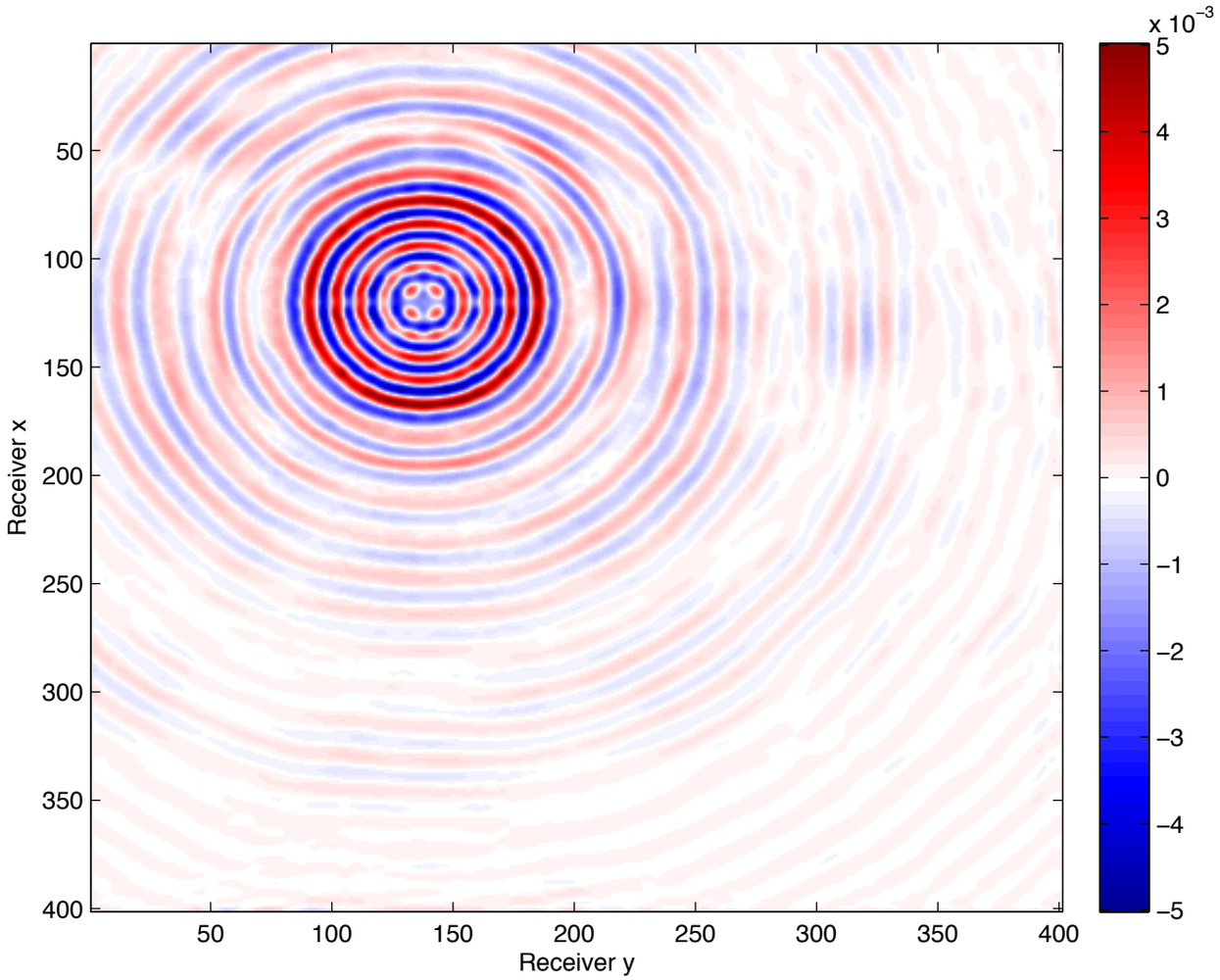
Known data

(Src x, Src y) = (36,30)



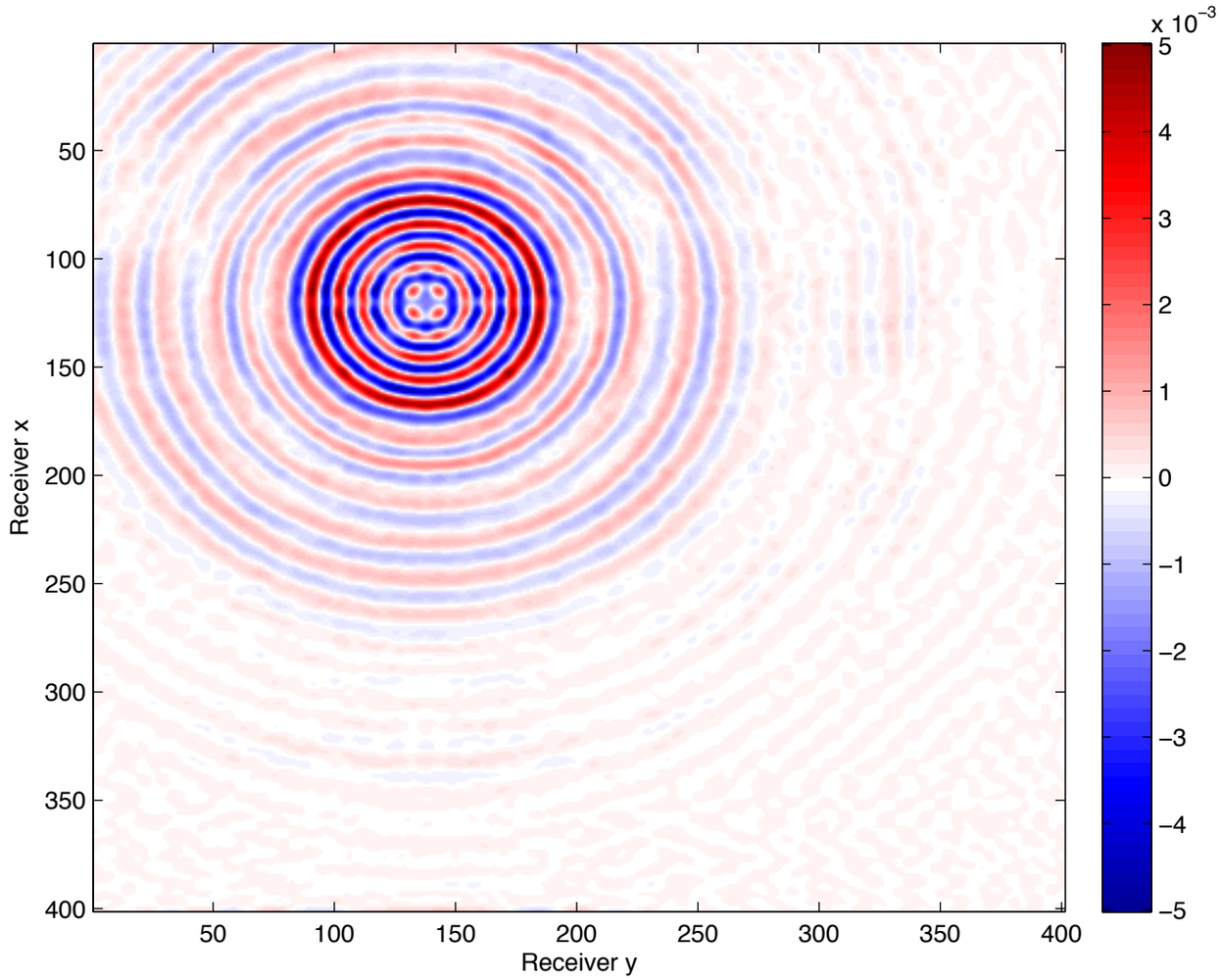
Interpolated data - SNR 14.2 dB

Source rank 20 - 500 iterations



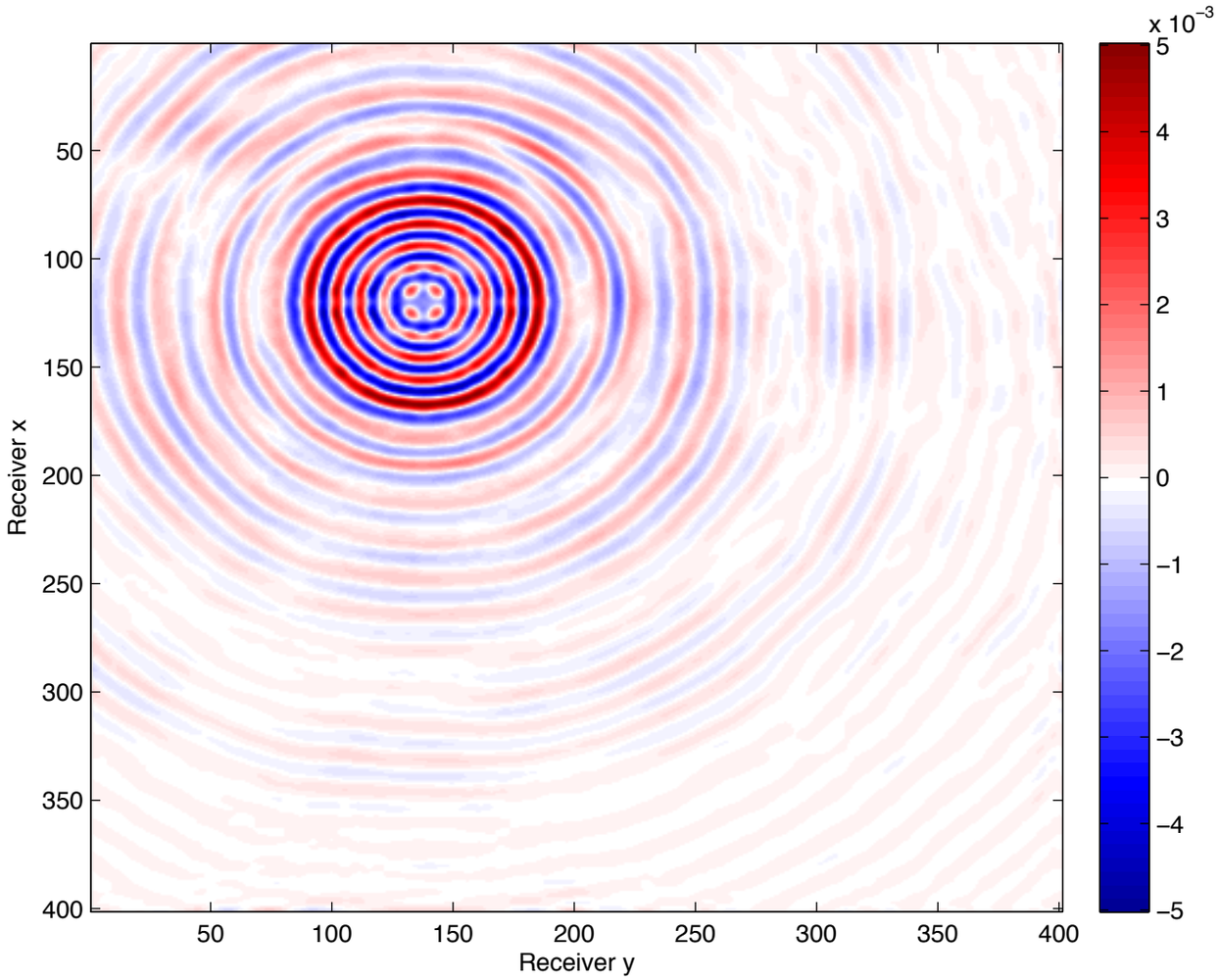
Known data

(Src x, Src y) = (21,24)



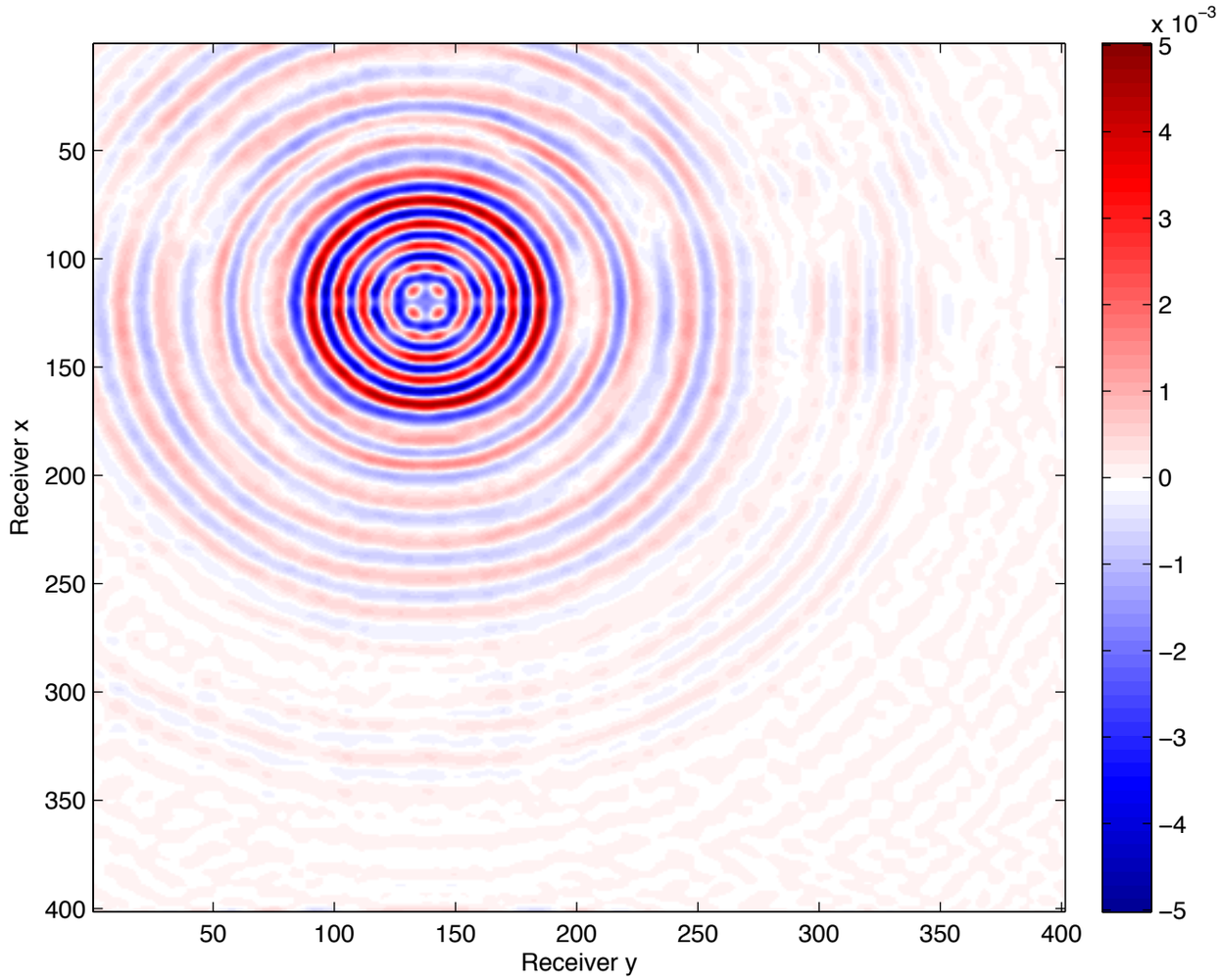
Interpolated data - SNR 12.5 dB

Source rank 20 - Regularization



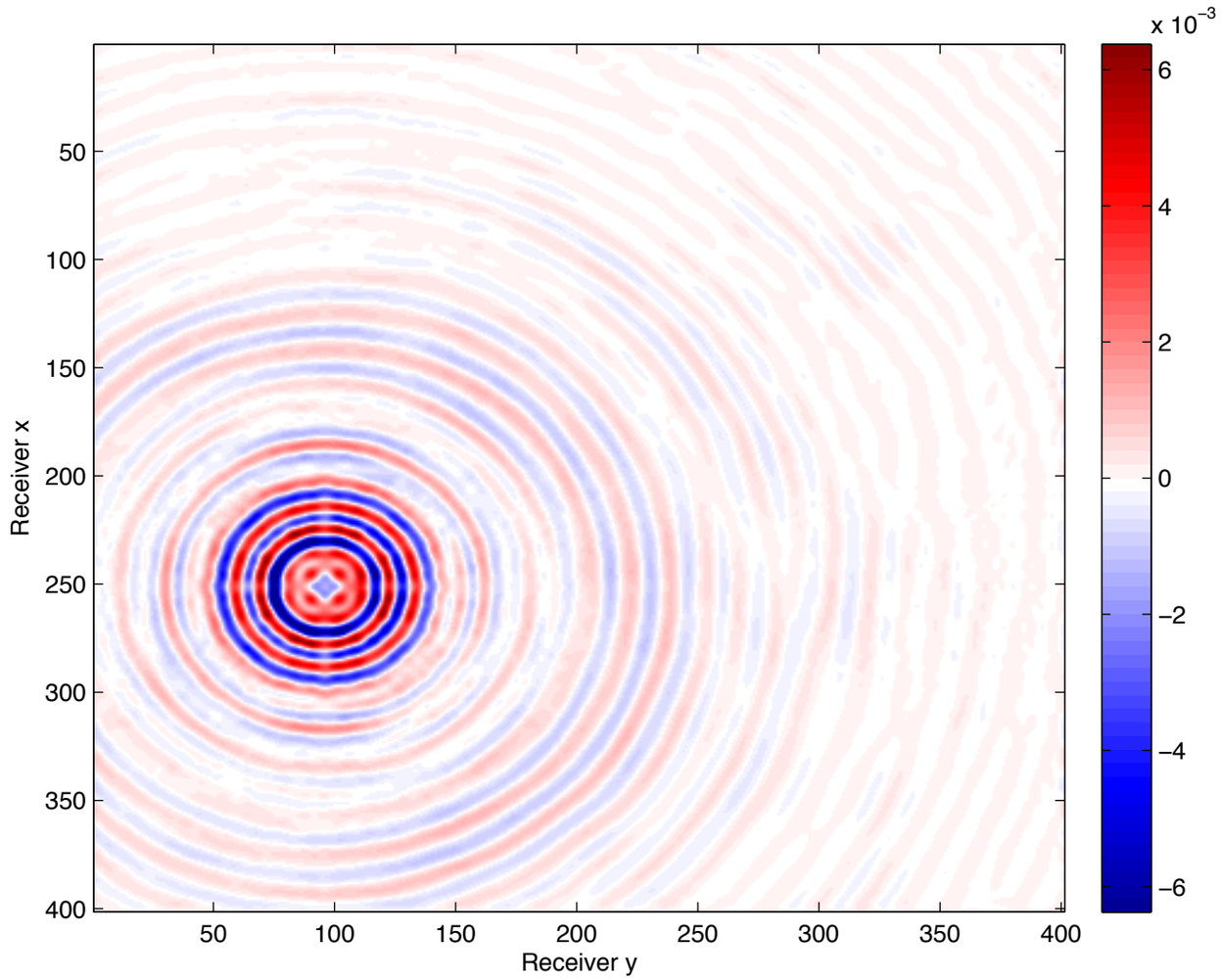
Known data

(Src x, Src y) = (21,24)



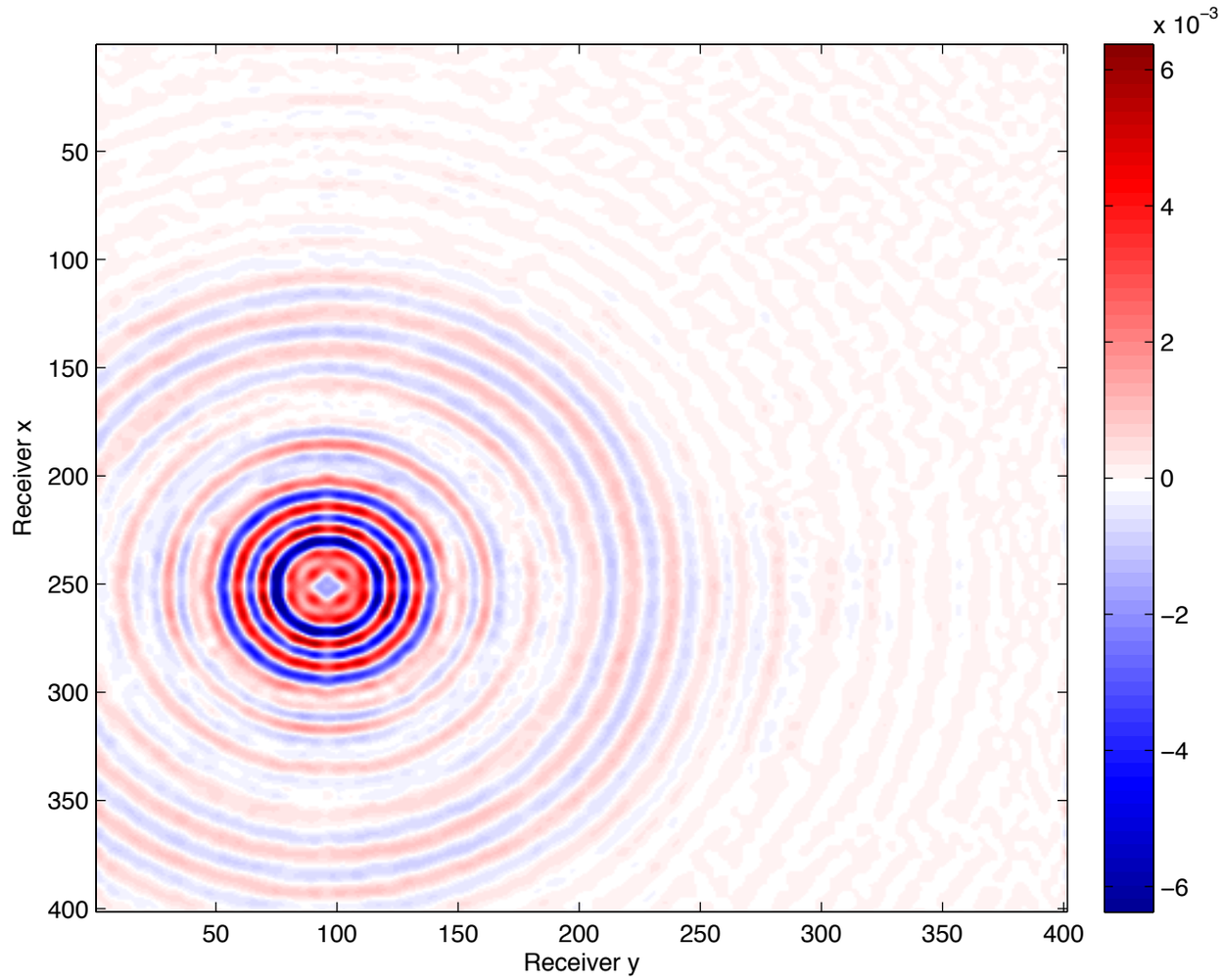
Interpolated data - SNR 14.2 dB

Source rank 20 - No Regularization



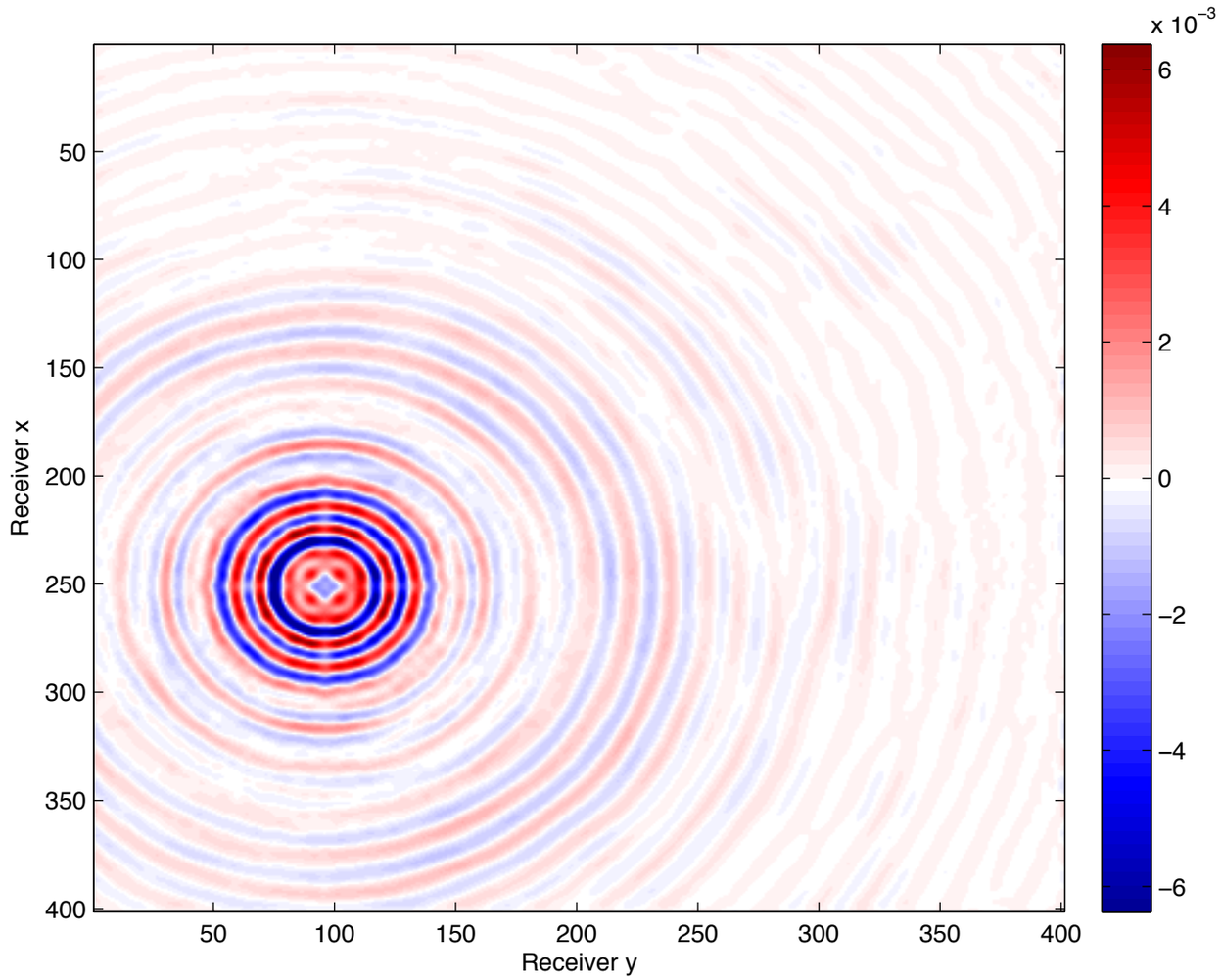
Known data

(Src x, Src y) = (64,25)



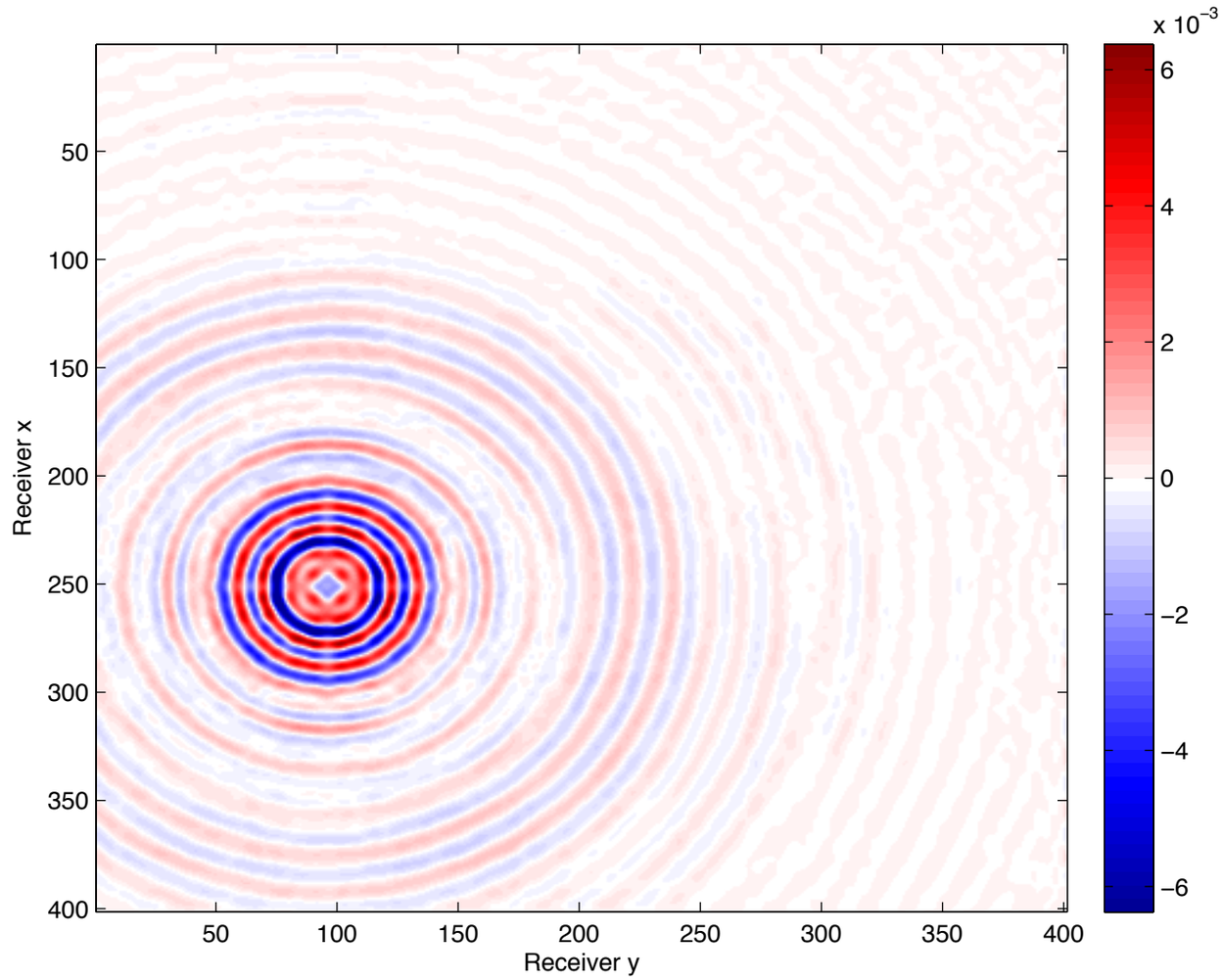
Interpolated data - SNR 13.9 dB

Source rank 20 - Regularization



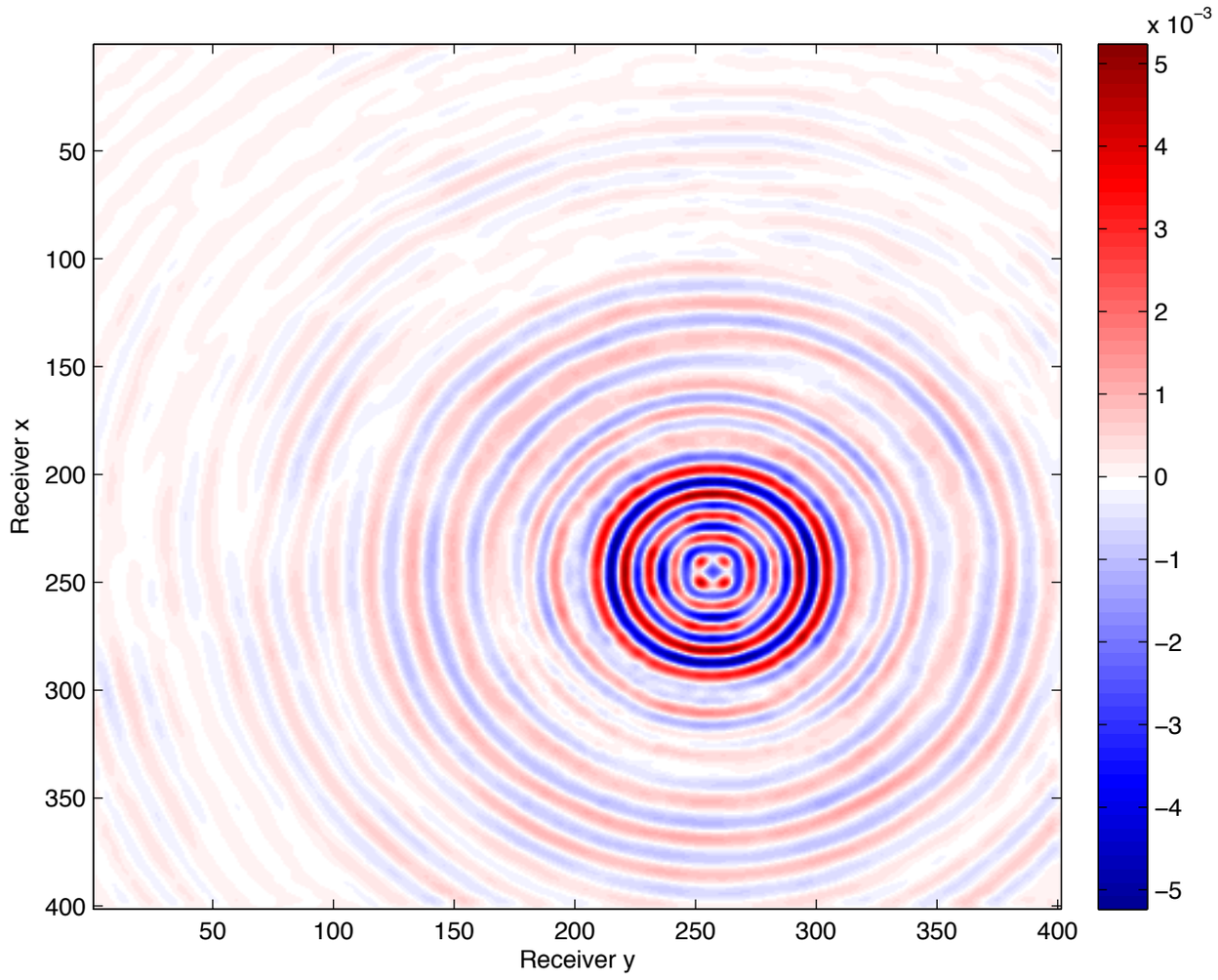
Known data

(Src x, Src y) = (64,25)



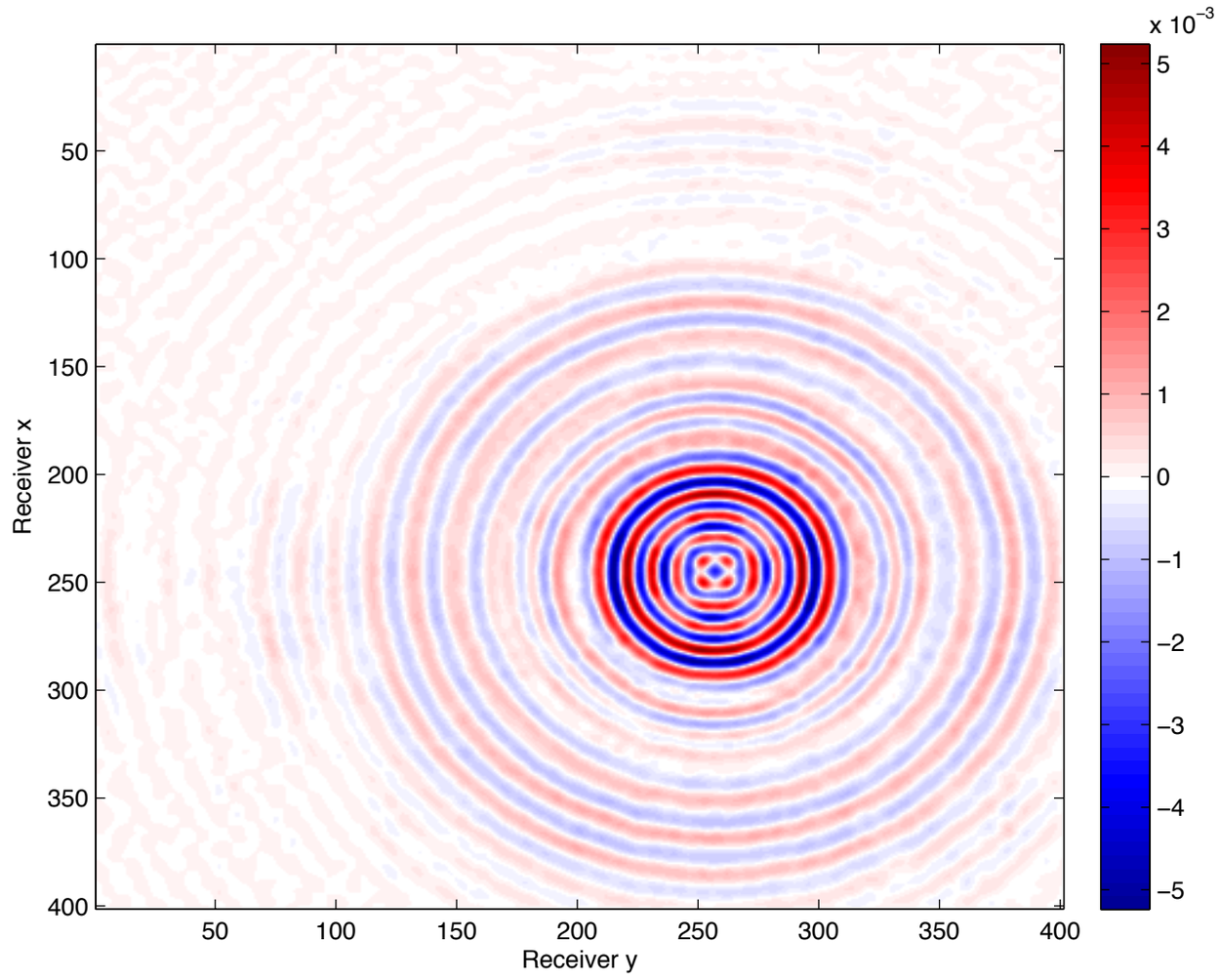
Interpolated data - SNR 15.4 dB

Source rank 20 - No Regularization



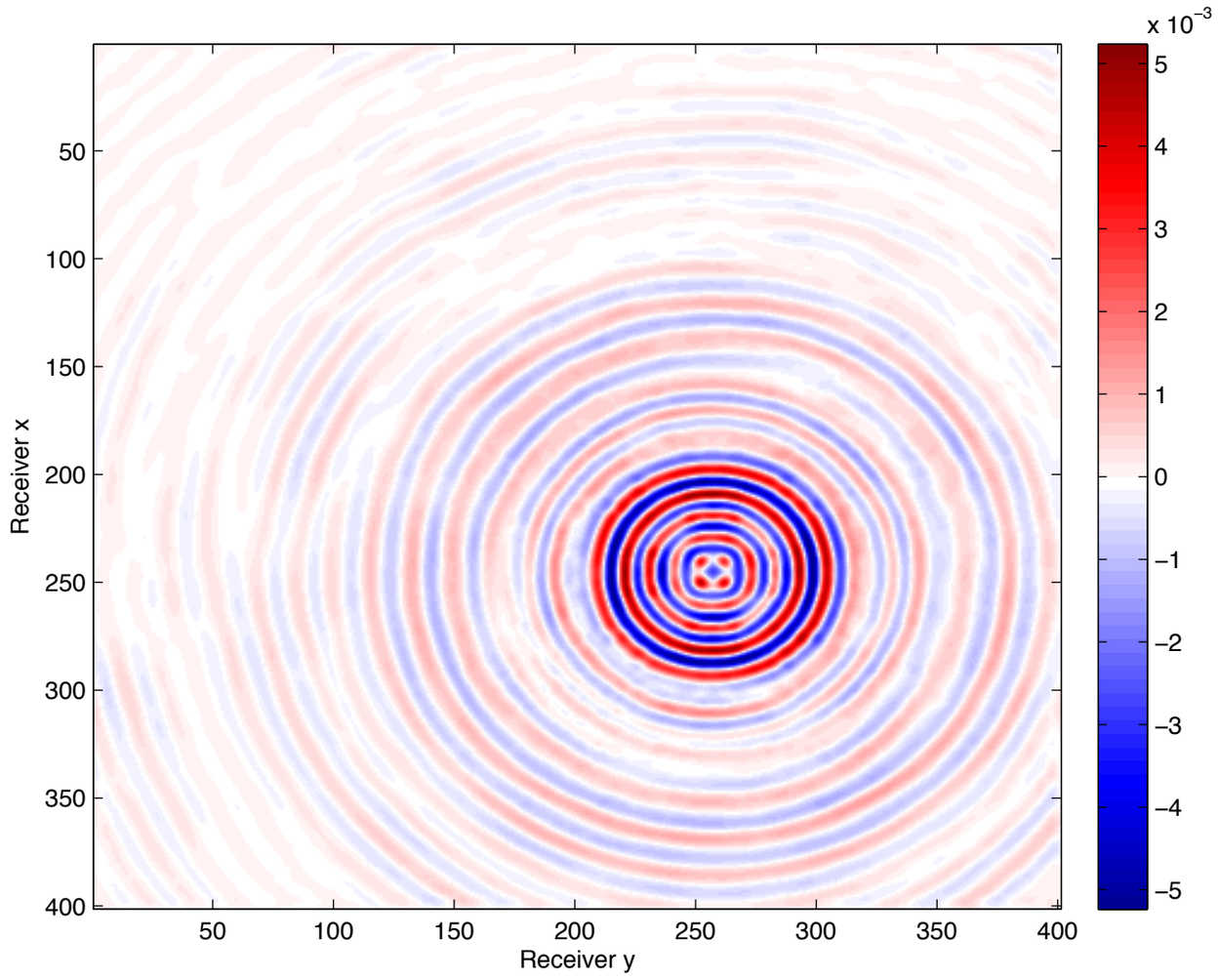
Known data

(Src x, Src y) = (63,66)



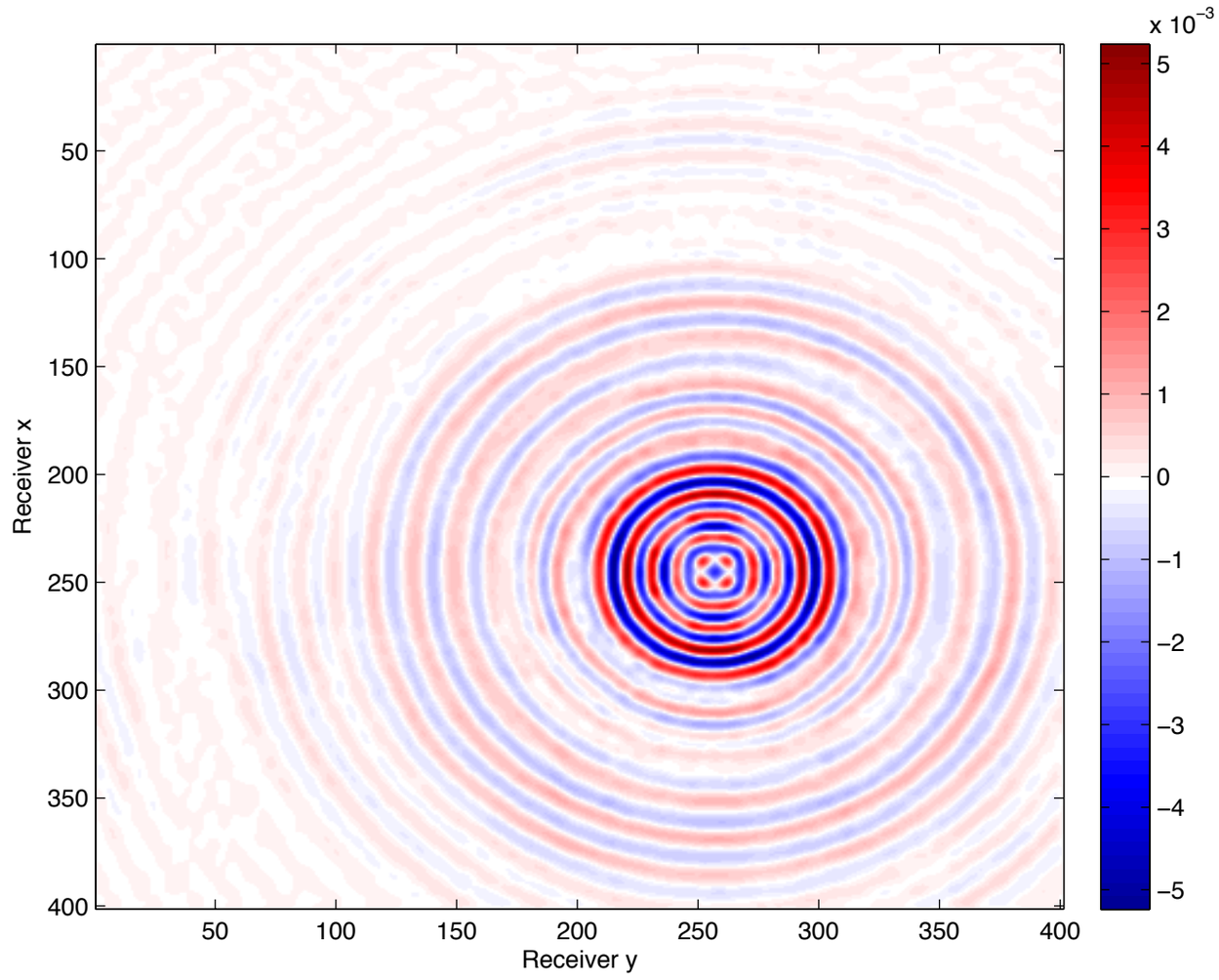
Interpolated data - SNR 13.2 dB

Source rank 20 - Regularization



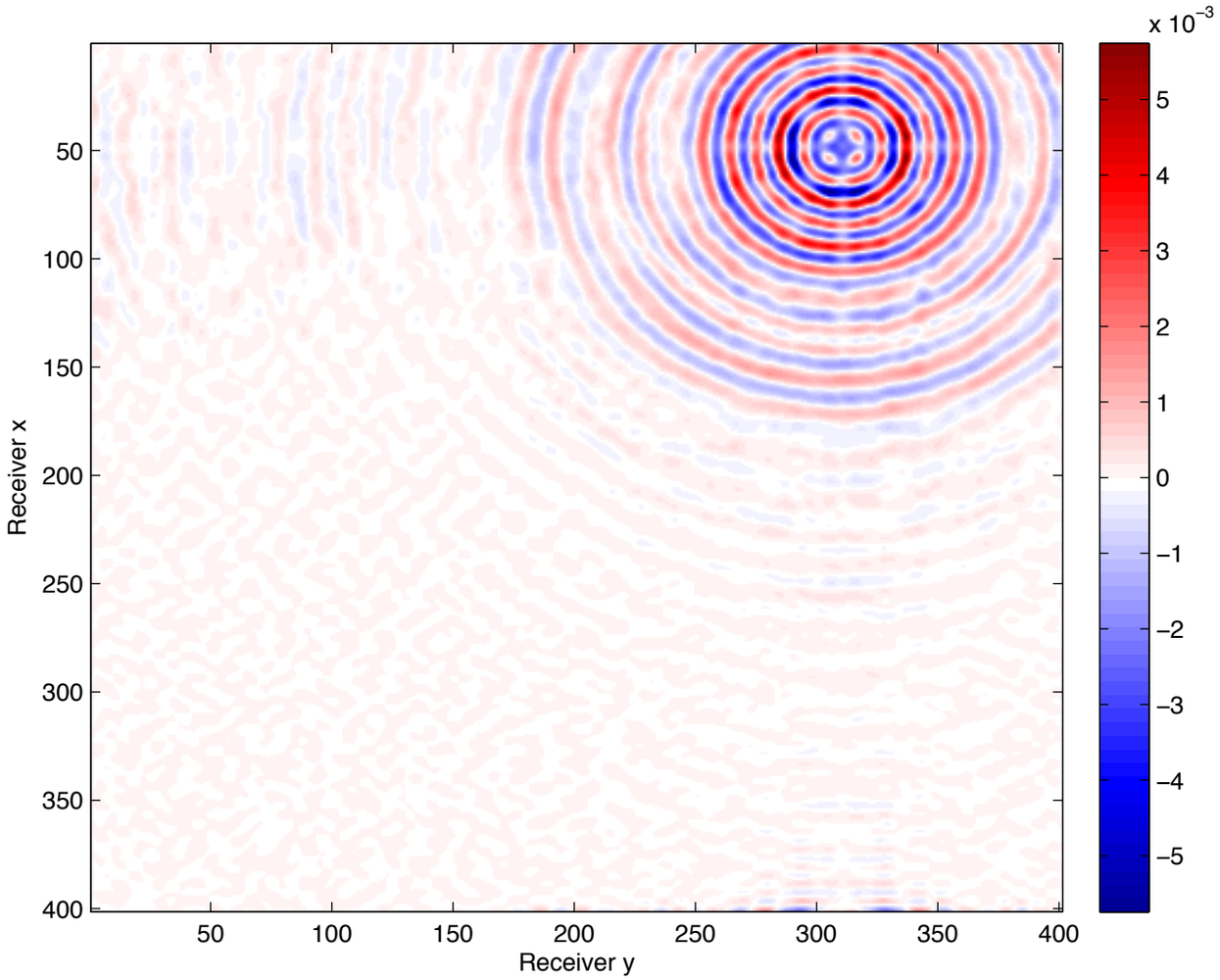
Known data

(Src x, Src y) = (63,66)



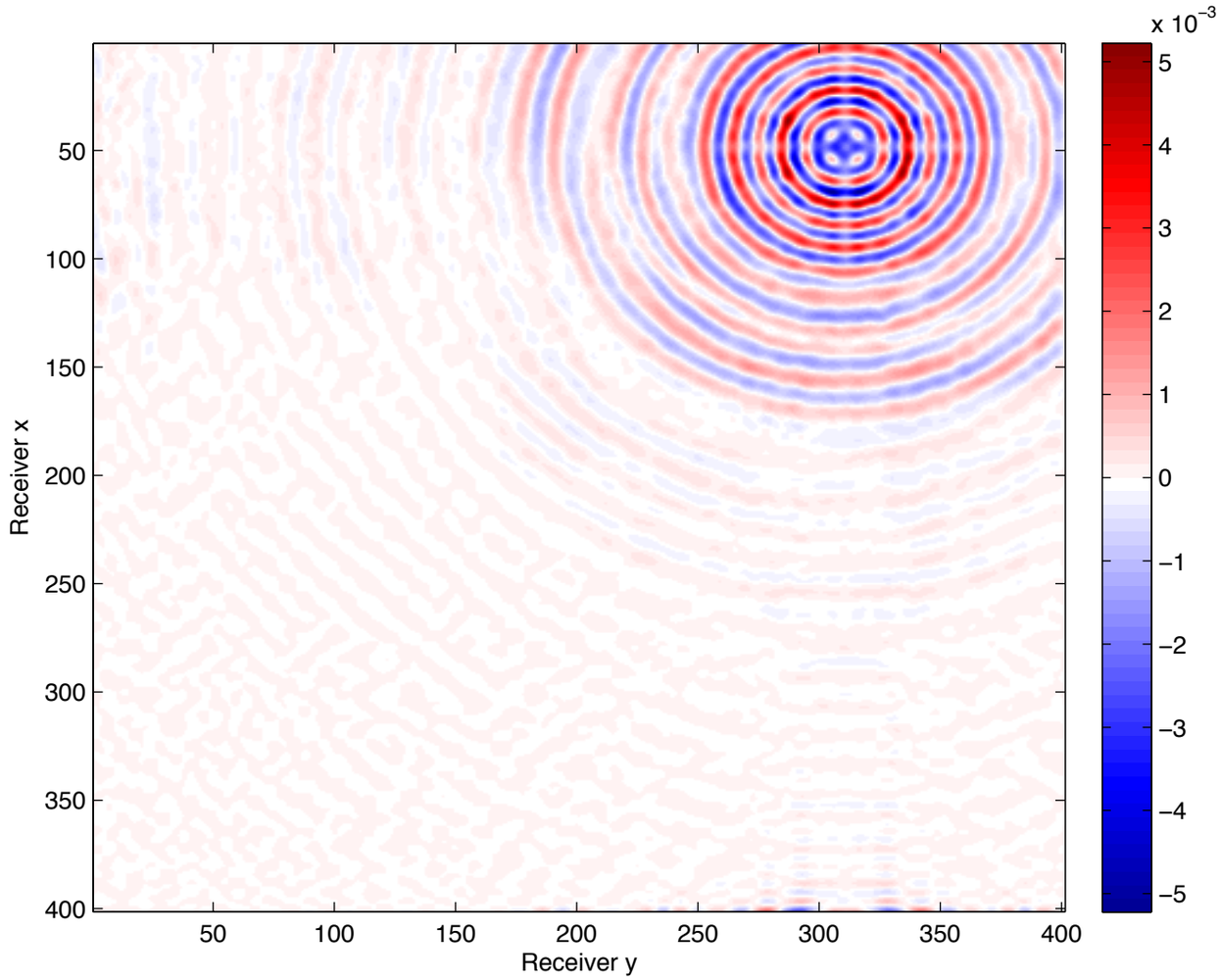
Interpolated data - SNR 15 dB

Interpolated Data



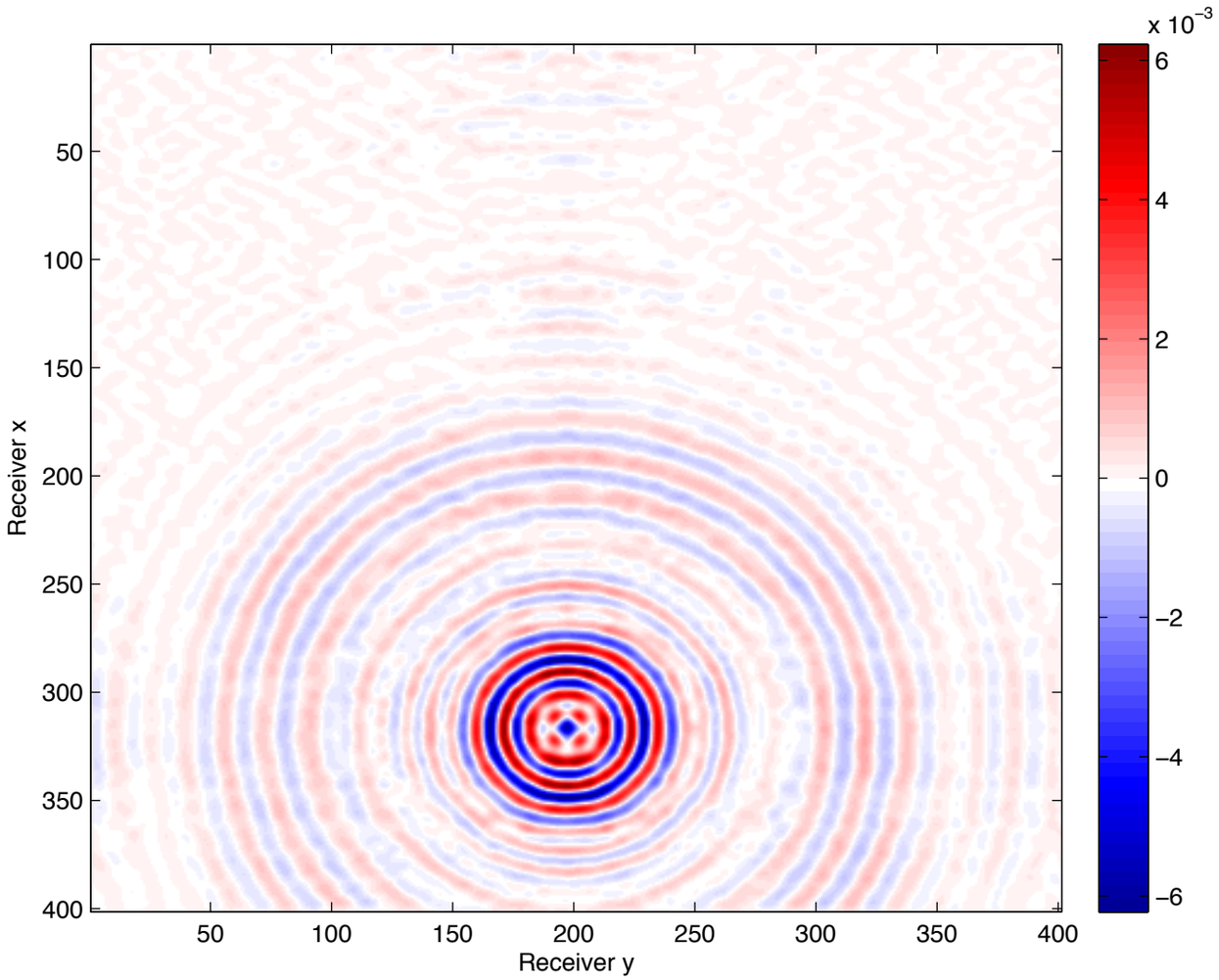
No Regularization

(Src x, Src y) = (9,53)



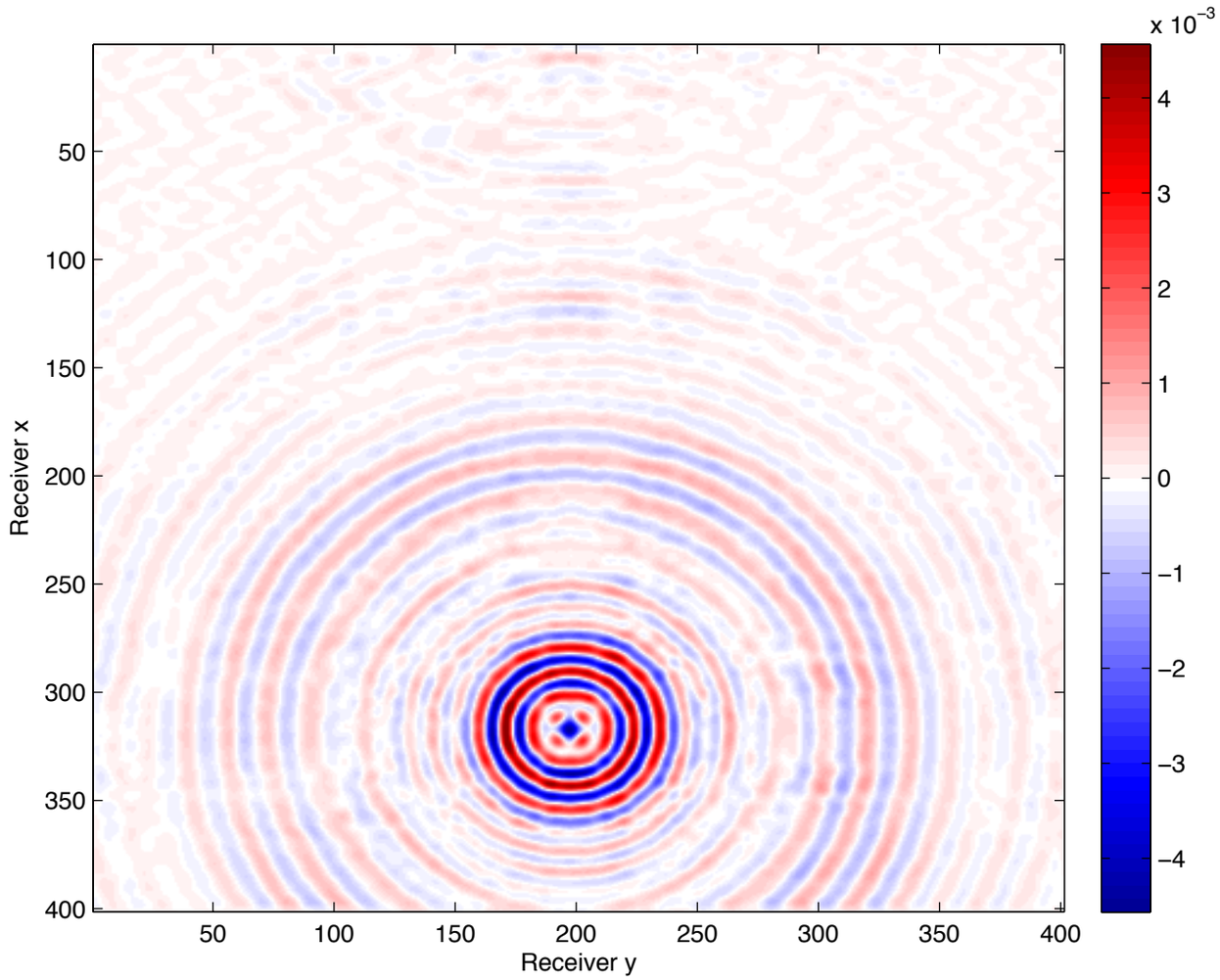
Regularization

Interpolated Data



No Regularization

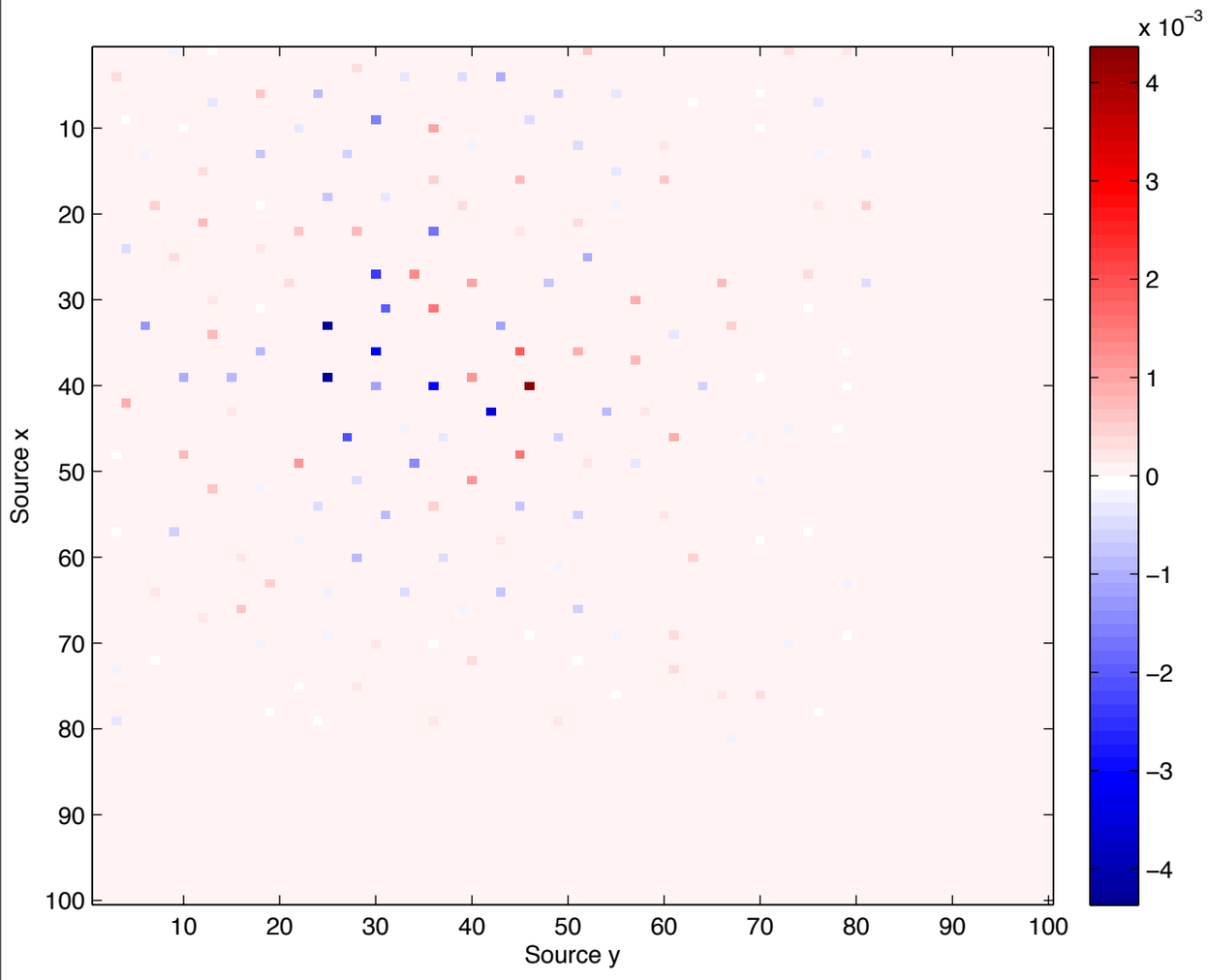
(Src x, Src y) = (54,34)



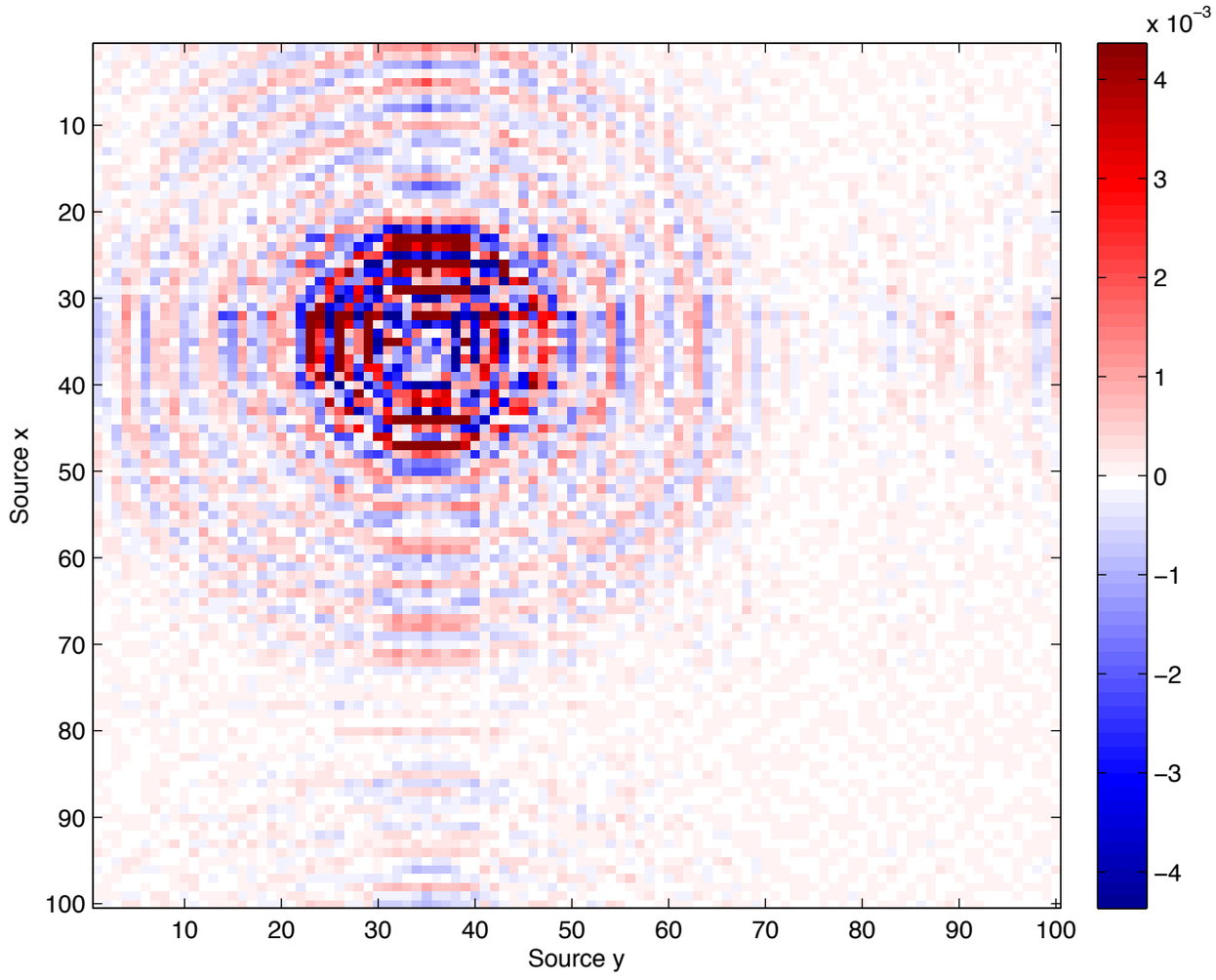
Regularization

Common Receiver Gathers

Rec x = 35, Rec y = 35



Known data

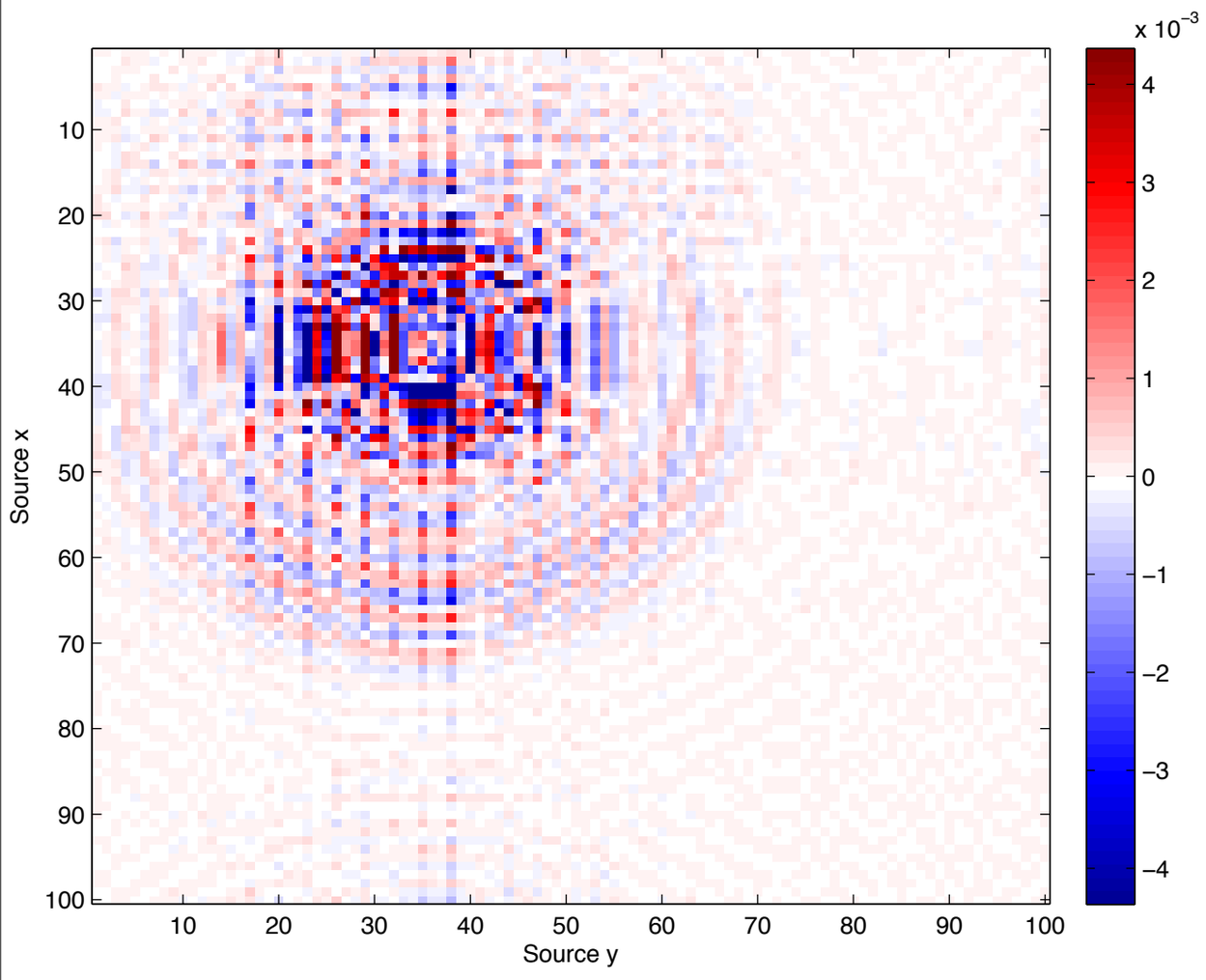


Interpolated data

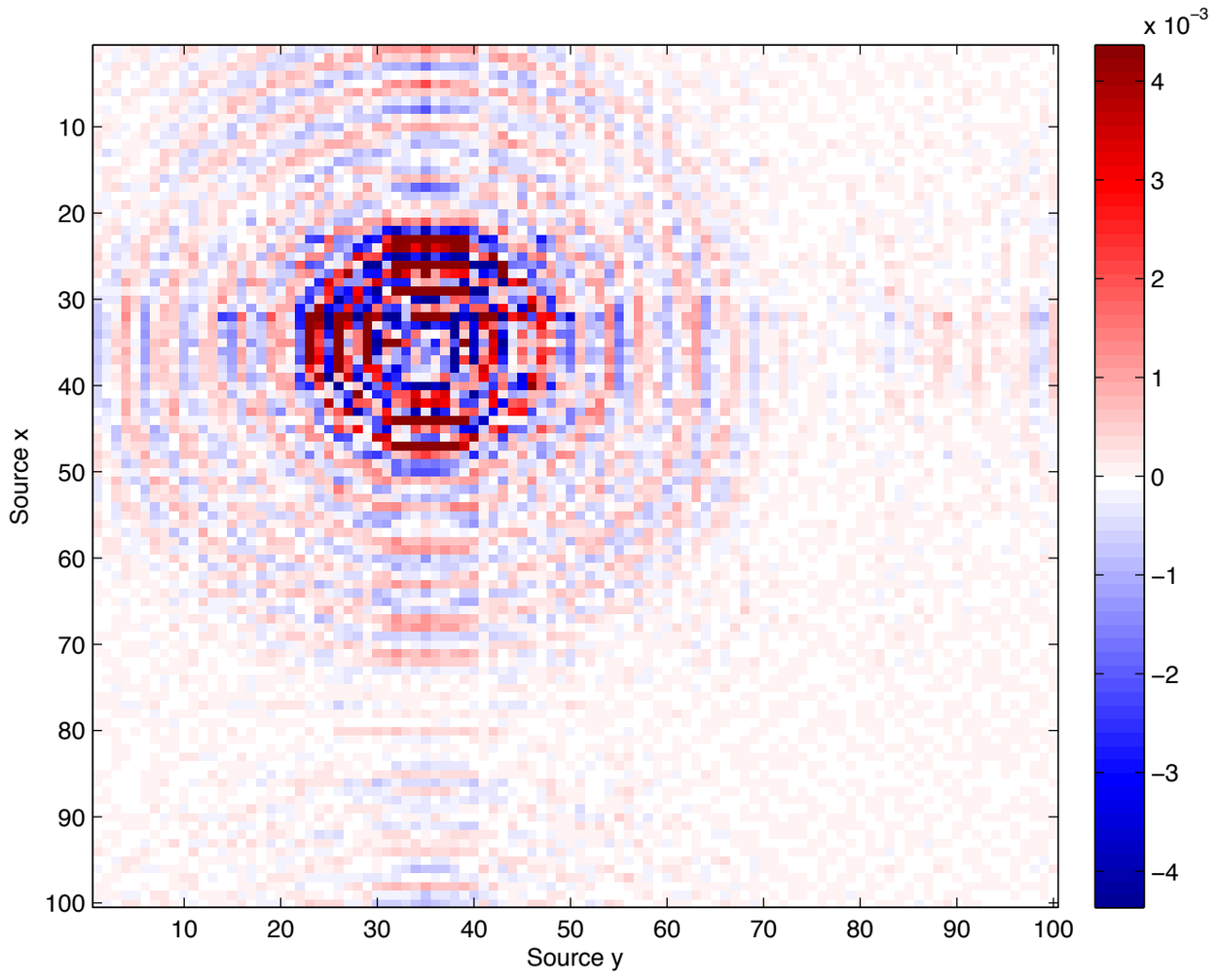
SLIM

Common Receiver Gathers

Rec x = 35, Rec y = 35



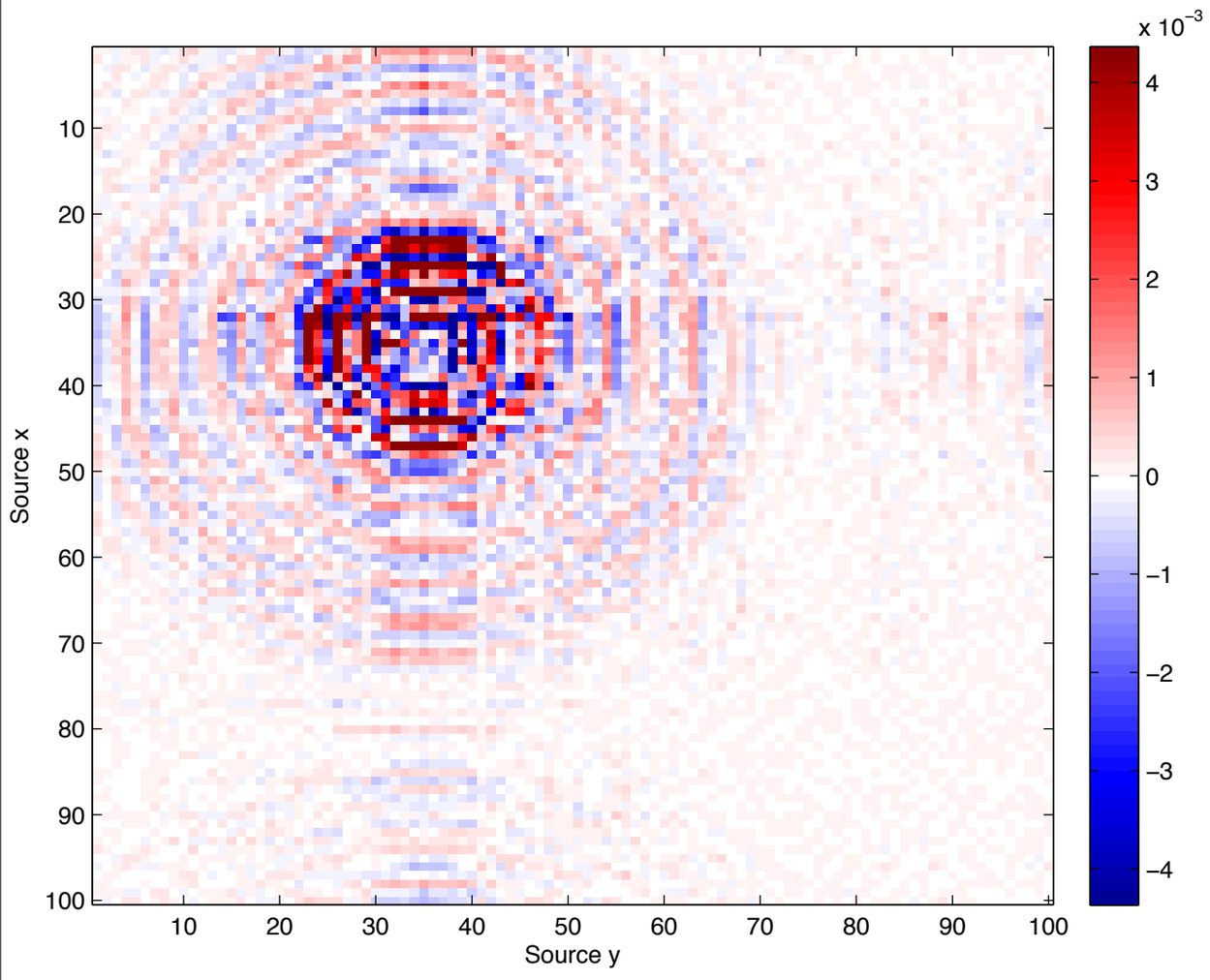
Source Rank 5



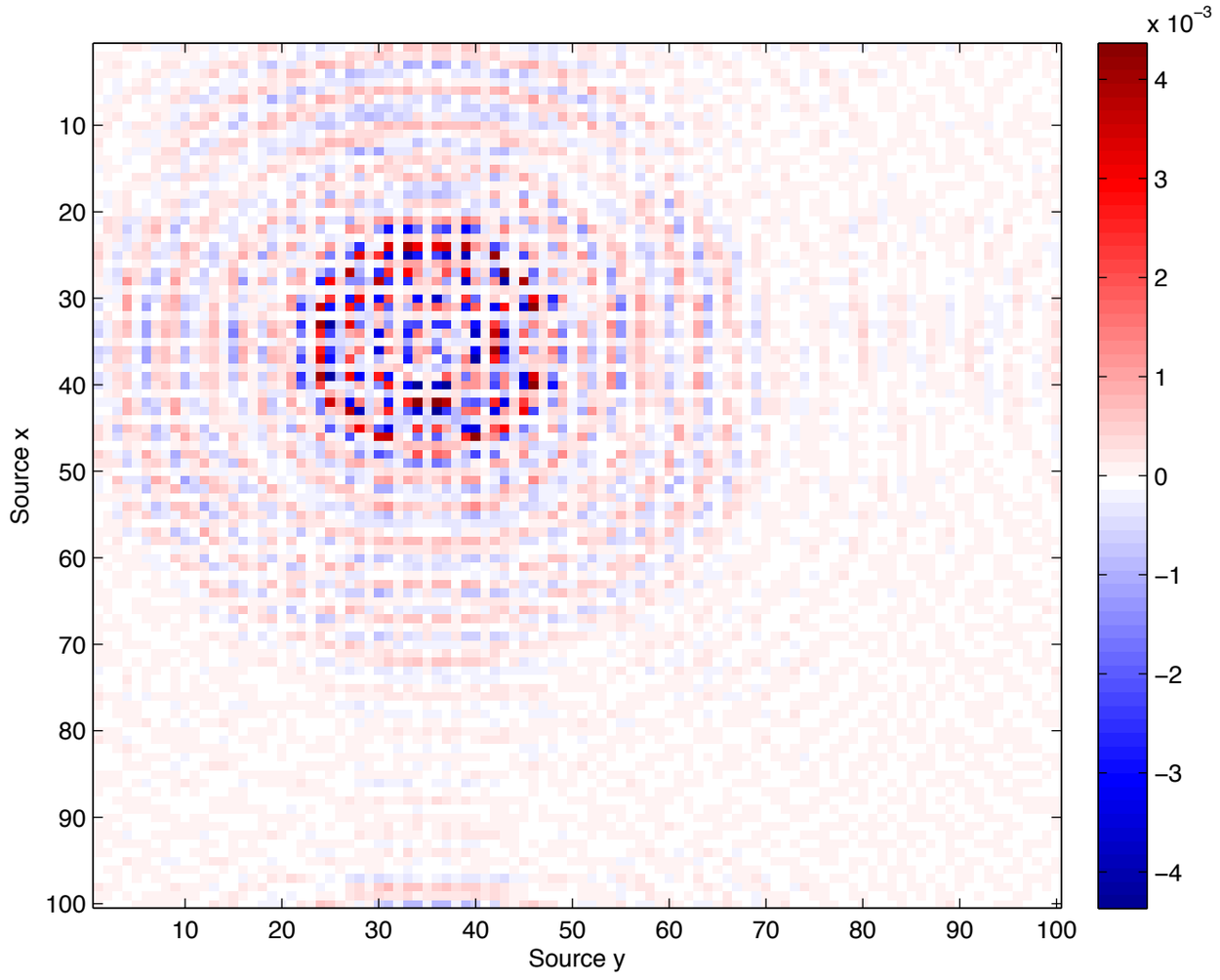
Source Rank 20

Common Receiver Gathers

Rec x = 35, Rec y = 35



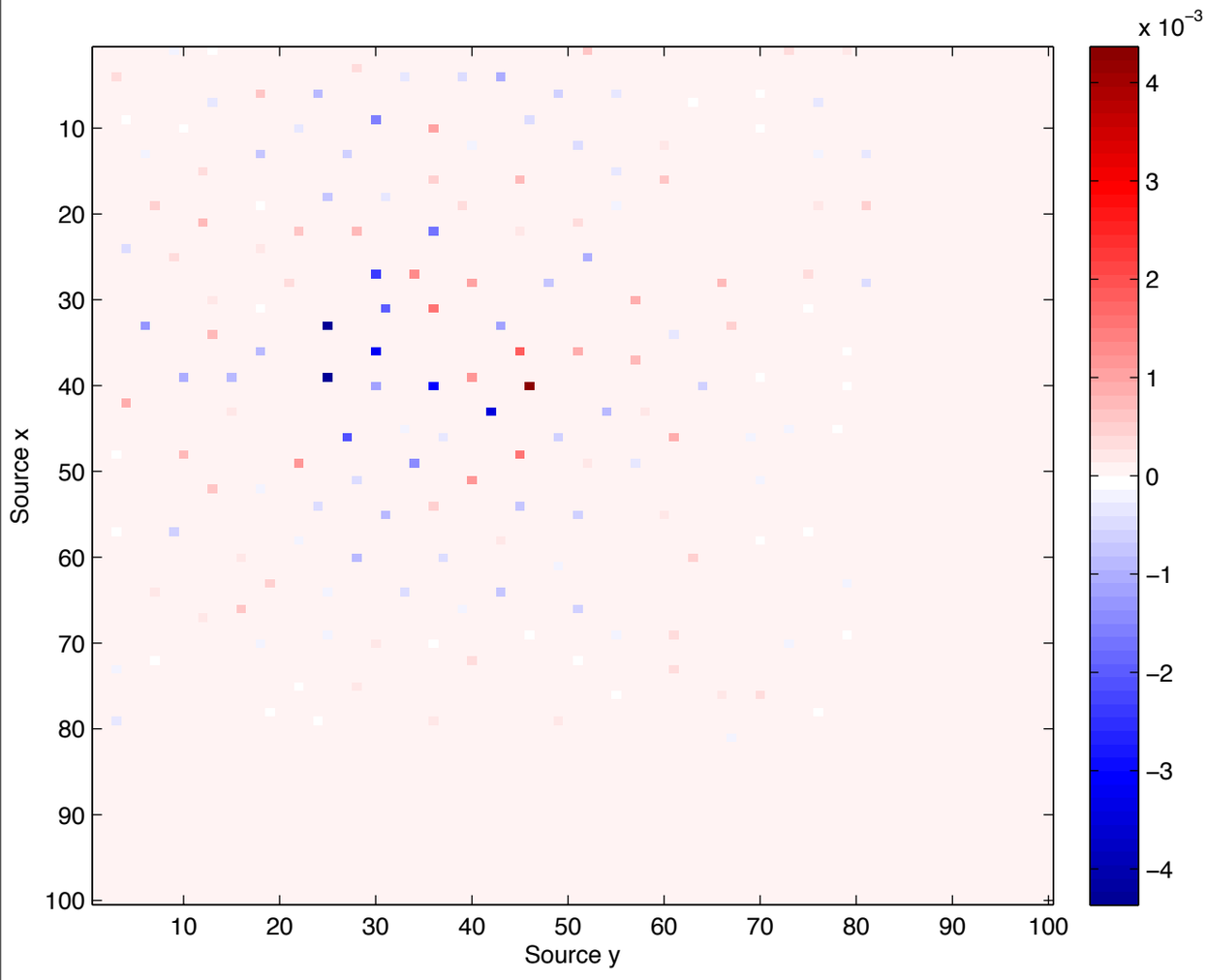
No Regularization



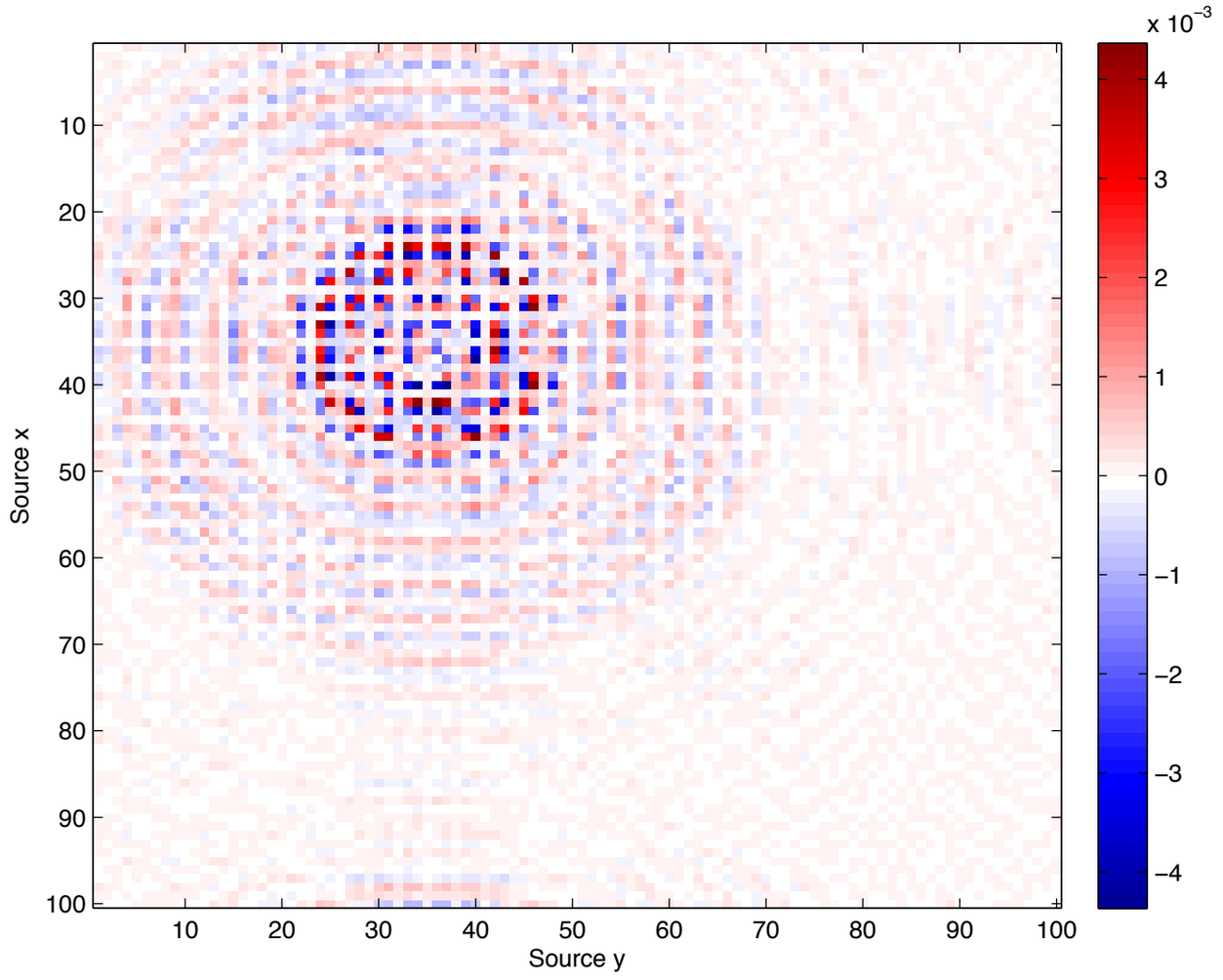
Regularization

Common Receiver Gathers

Rec x = 35, Rec y = 35



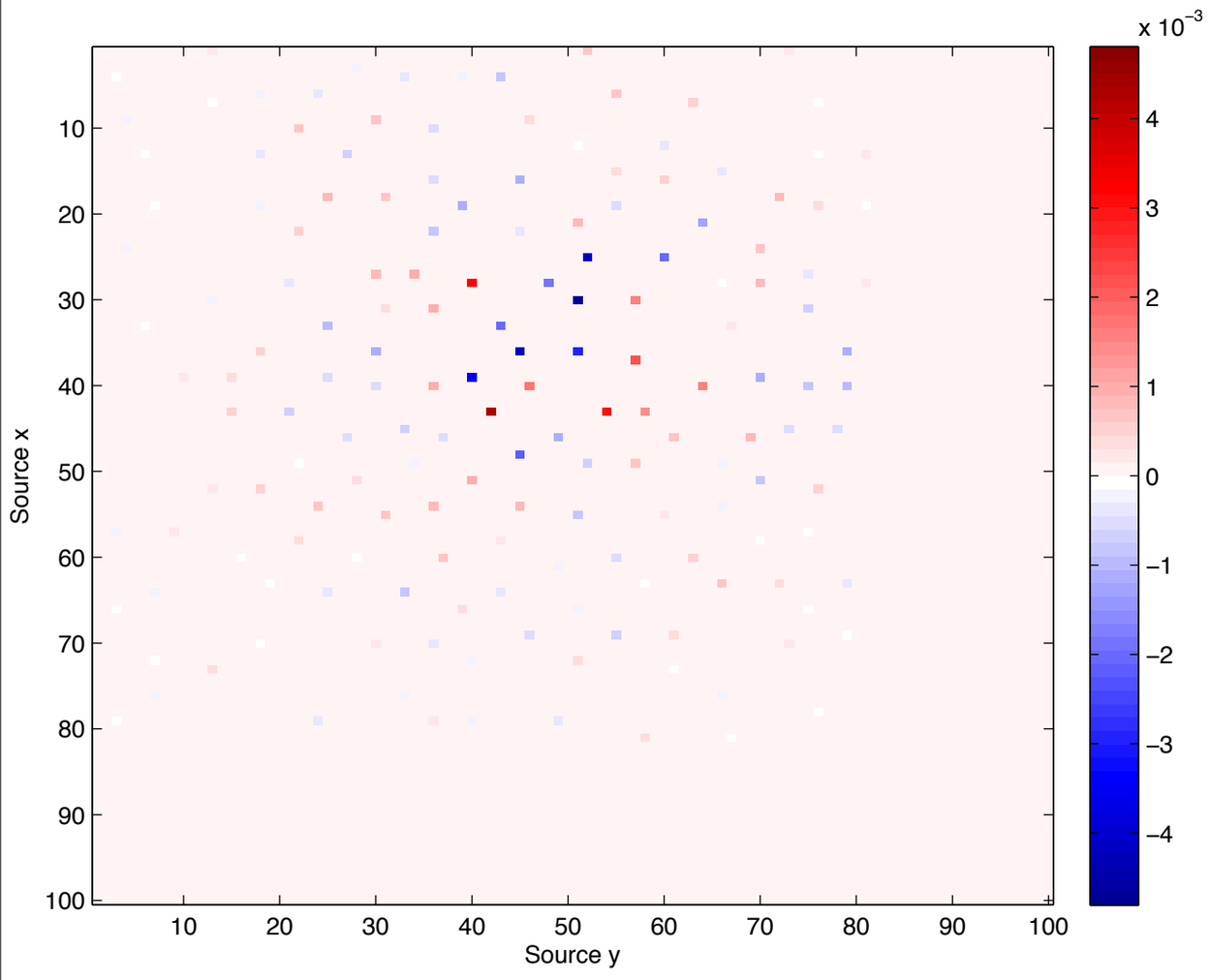
Known data



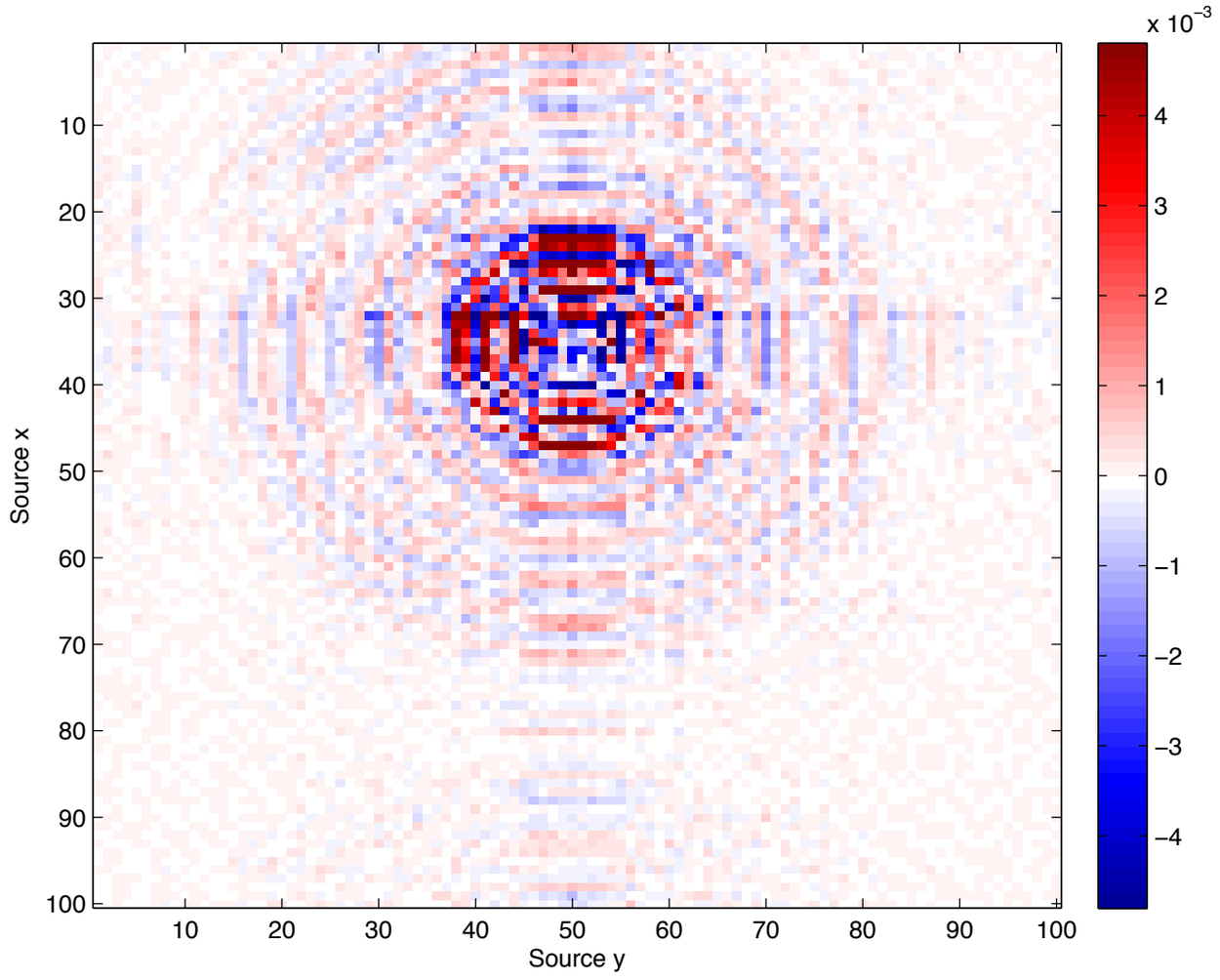
Regularized final result

Common Receiver Gathers

Rec x = 35, Rec y = 50



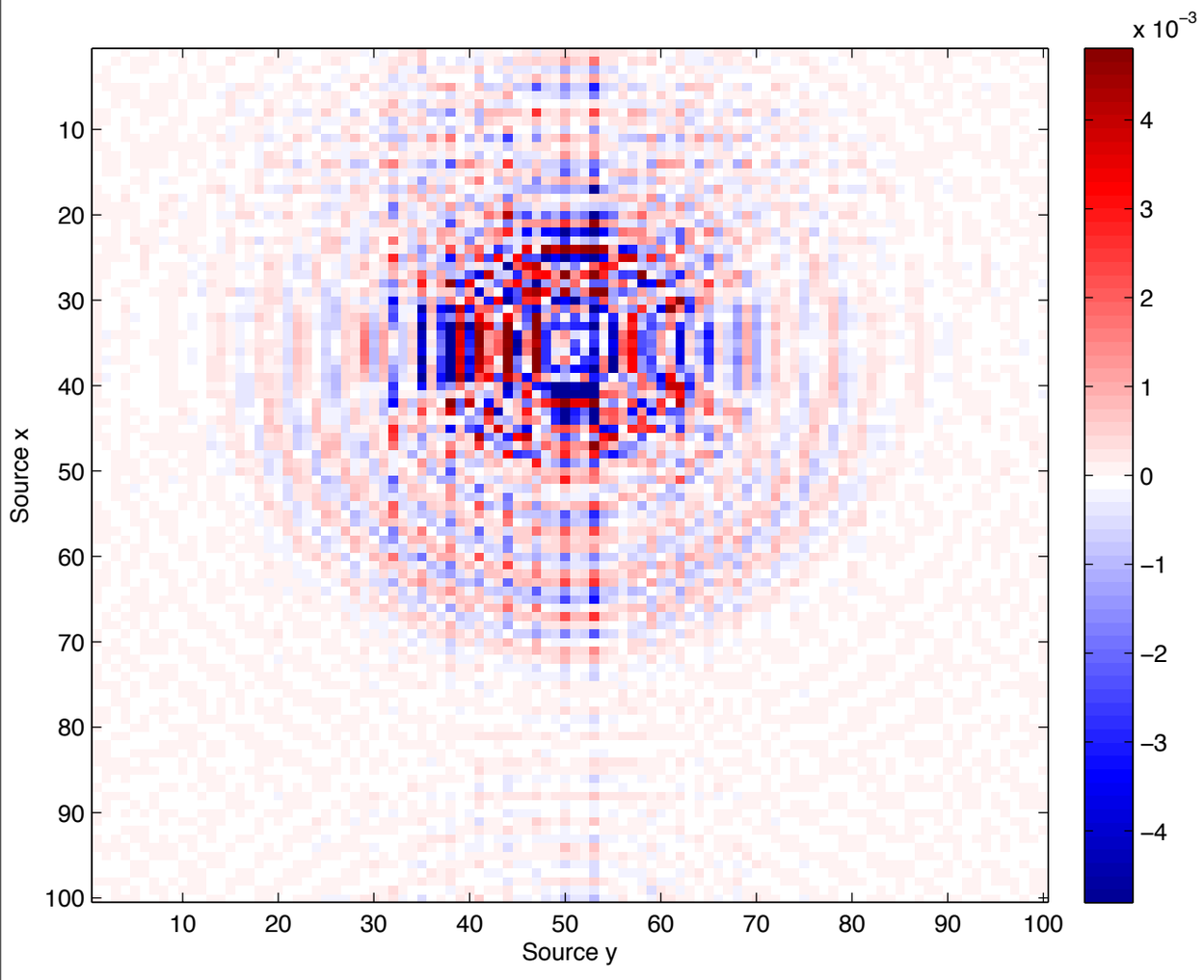
Known data



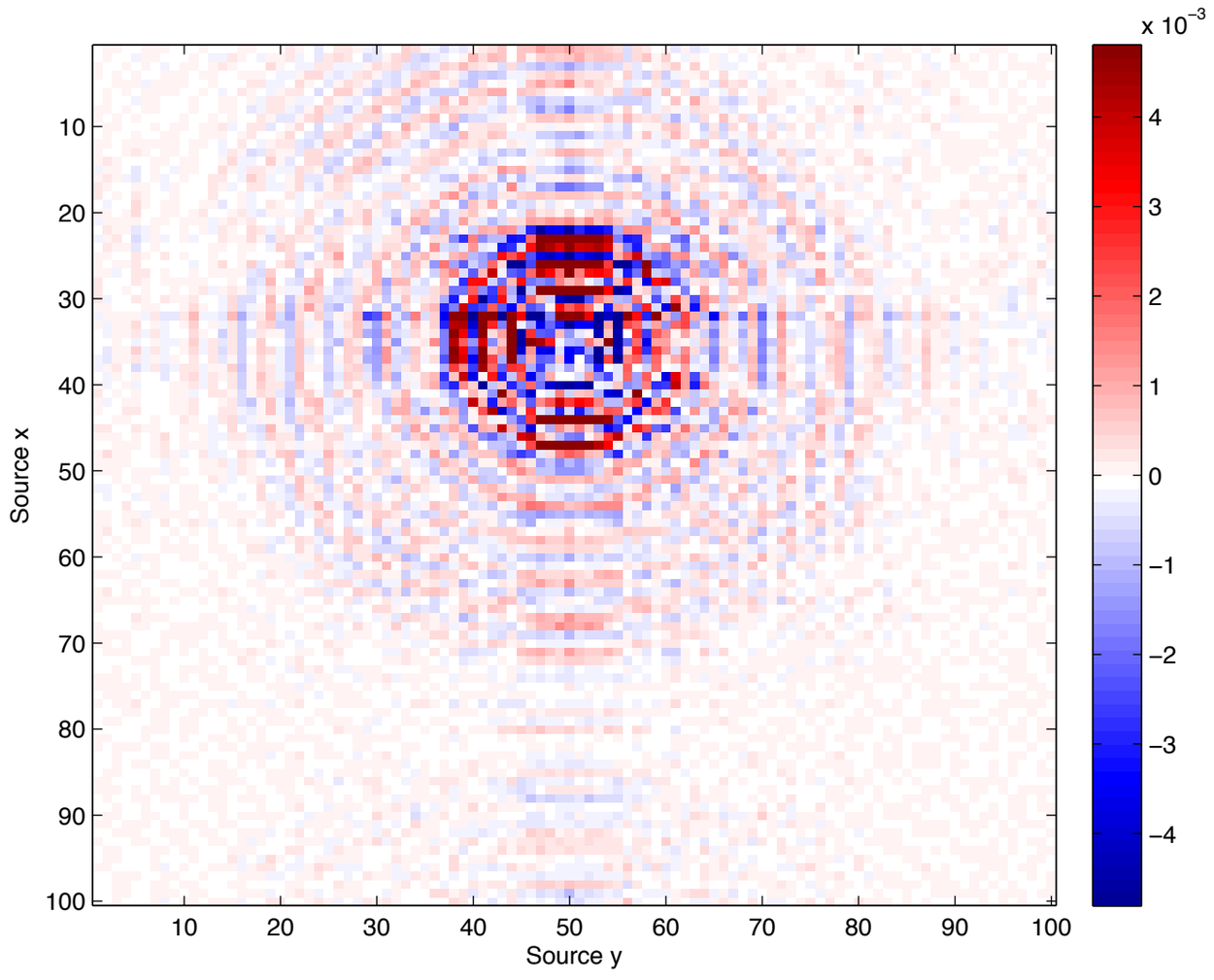
Interpolated data

Common Receiver Gathers

Rec x = 35, Rec y = 50



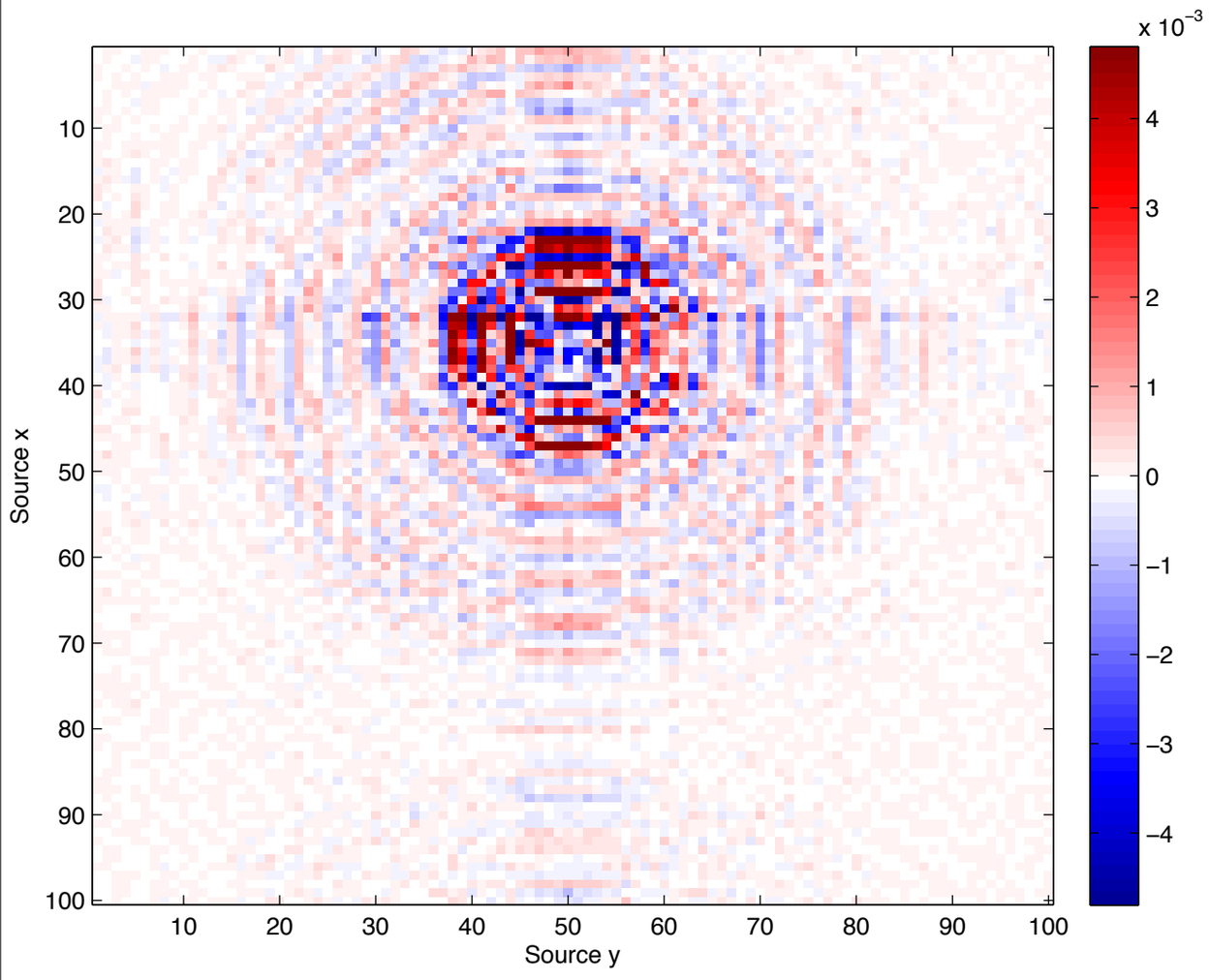
Source Rank 5



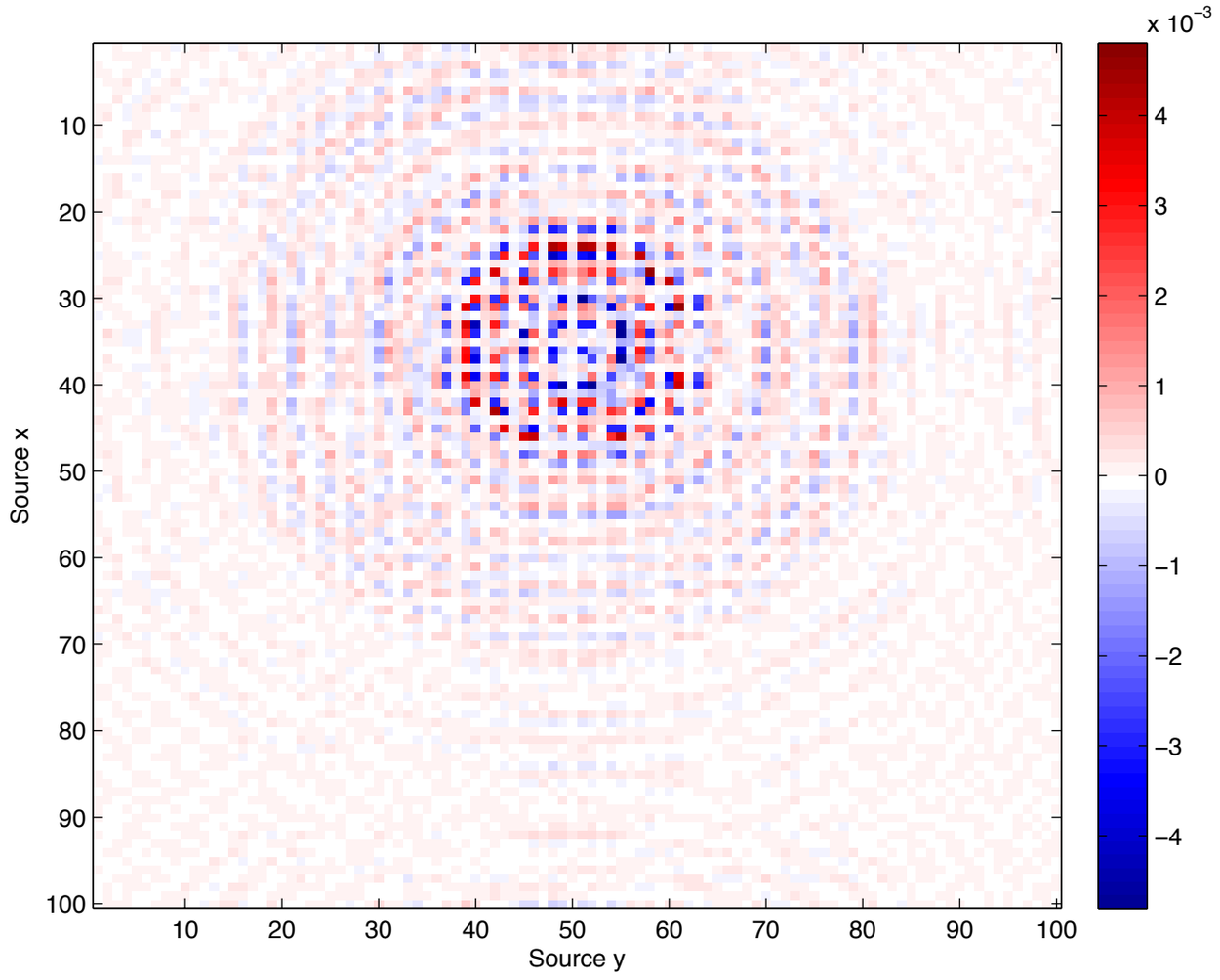
Source Rank 20

Common Receiver Gathers

Rec x = 35, Rec y = 50



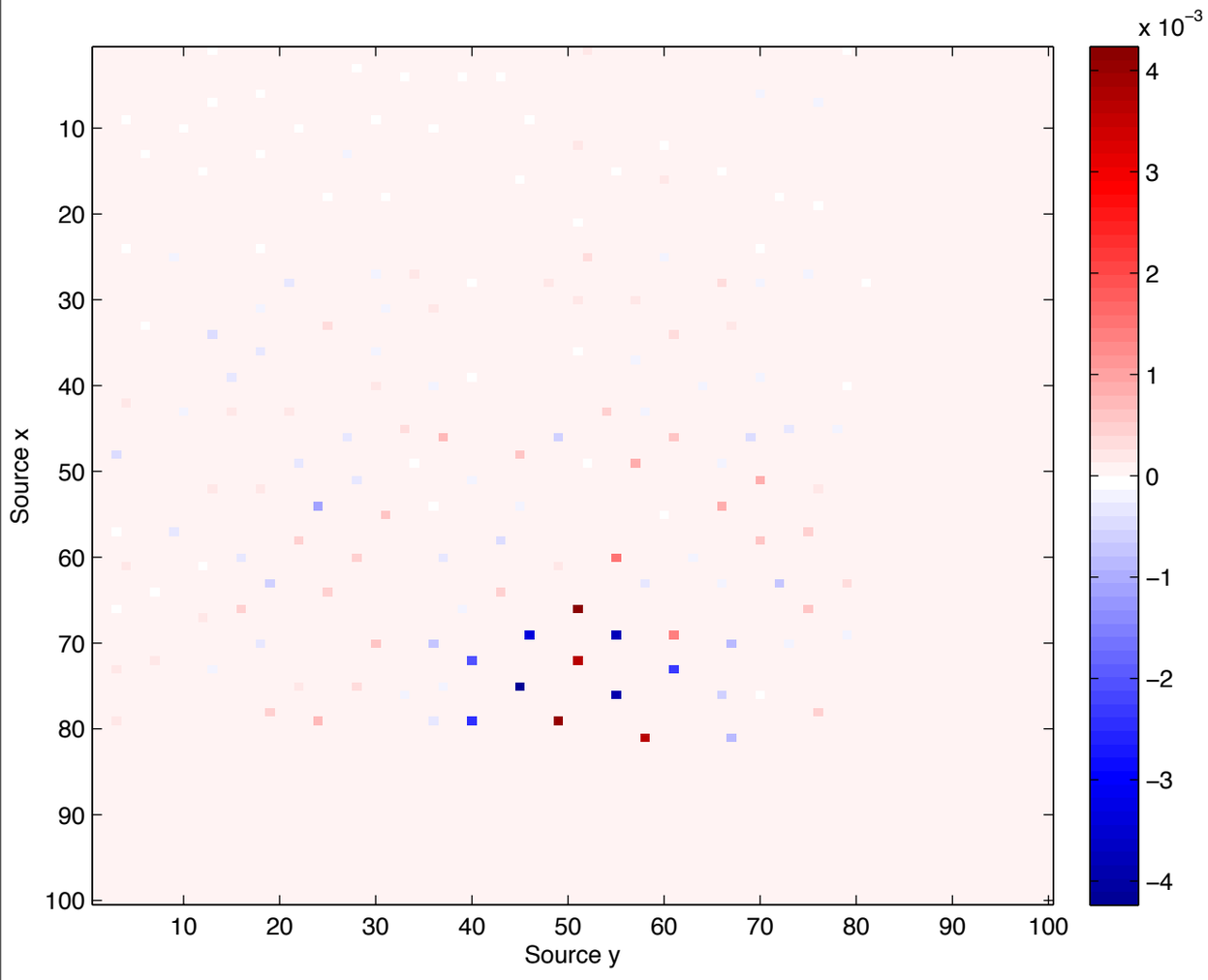
No Regularization



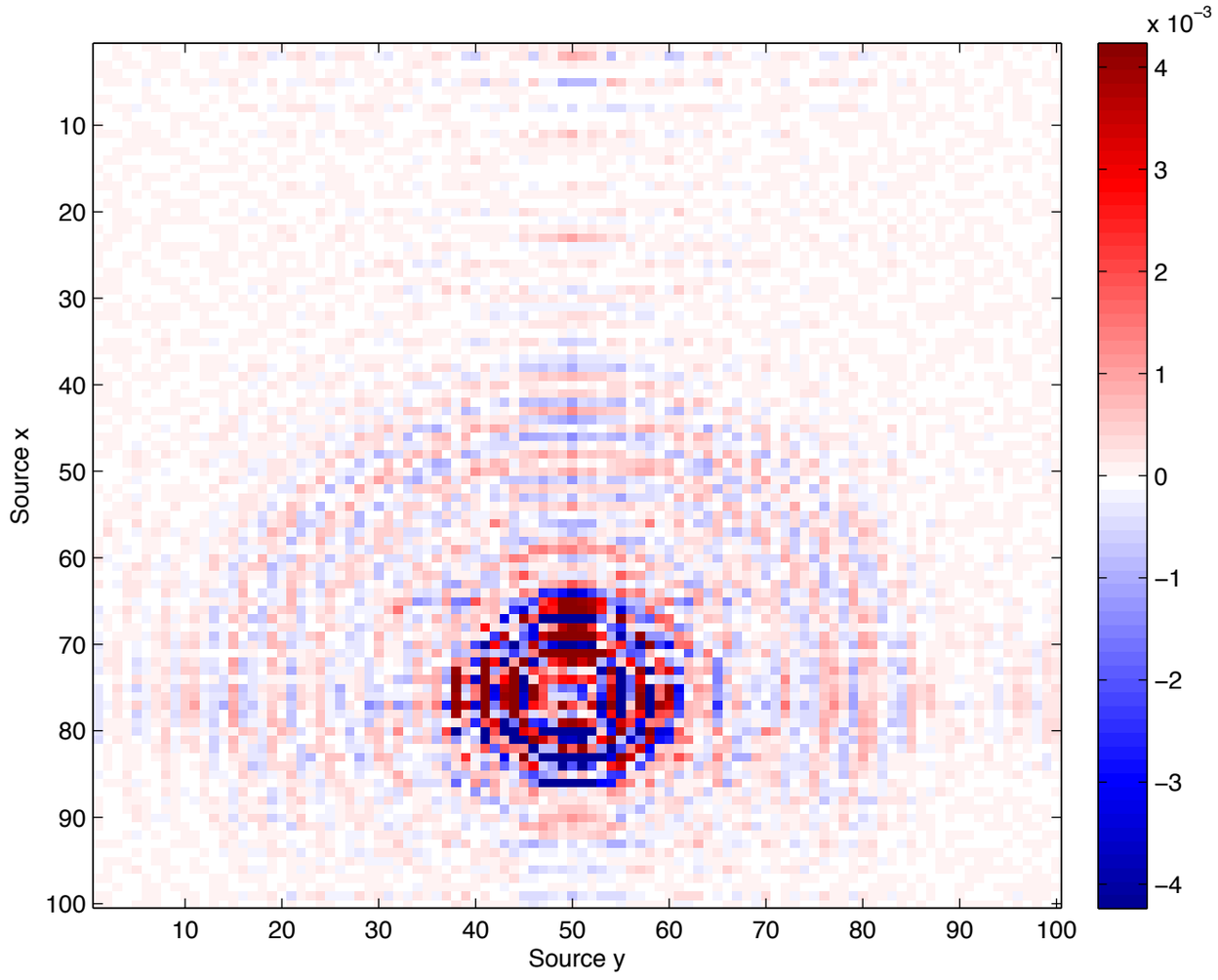
Regularization

Common Receiver Gathers

Rec x = 75, Rec y = 50



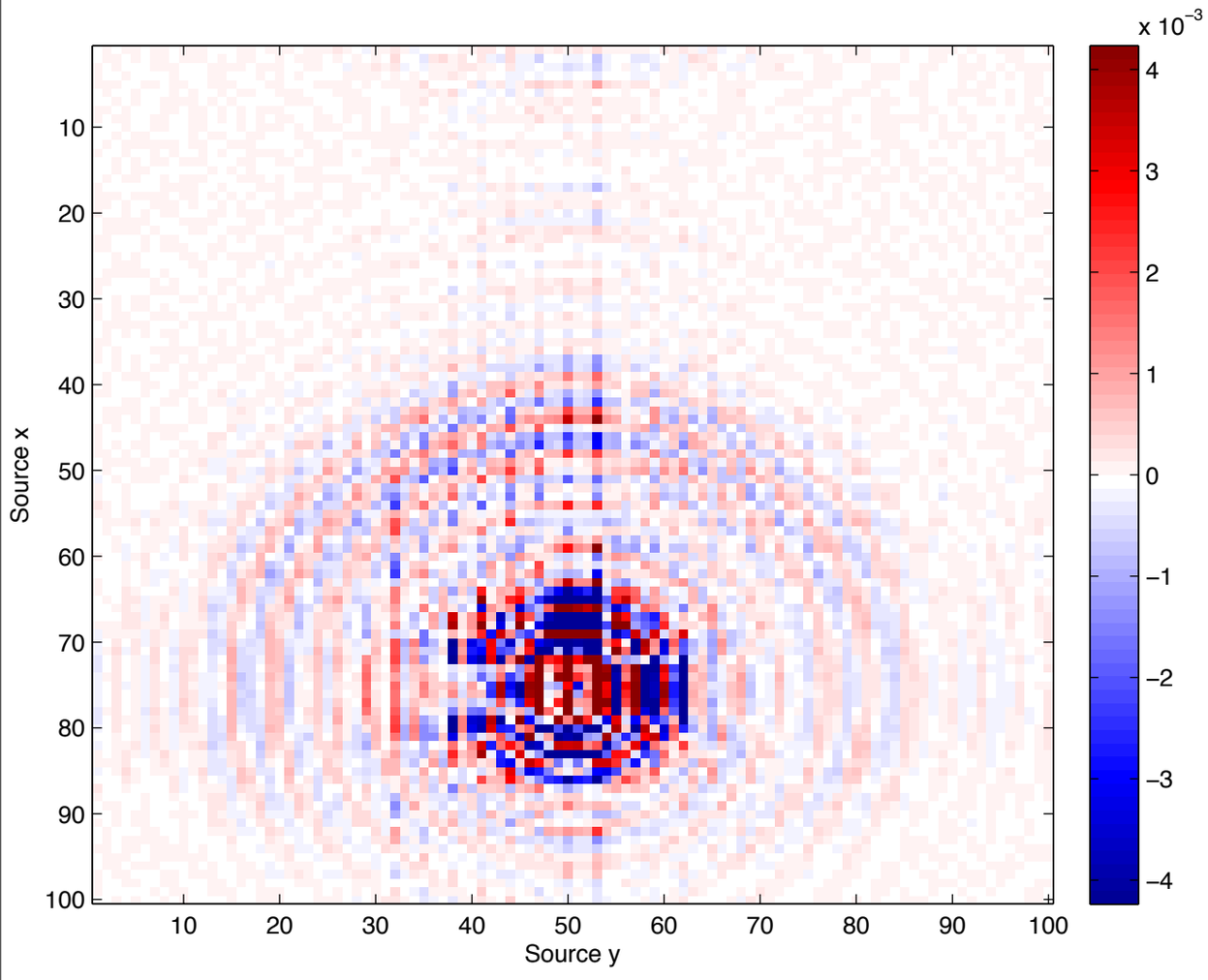
Known data



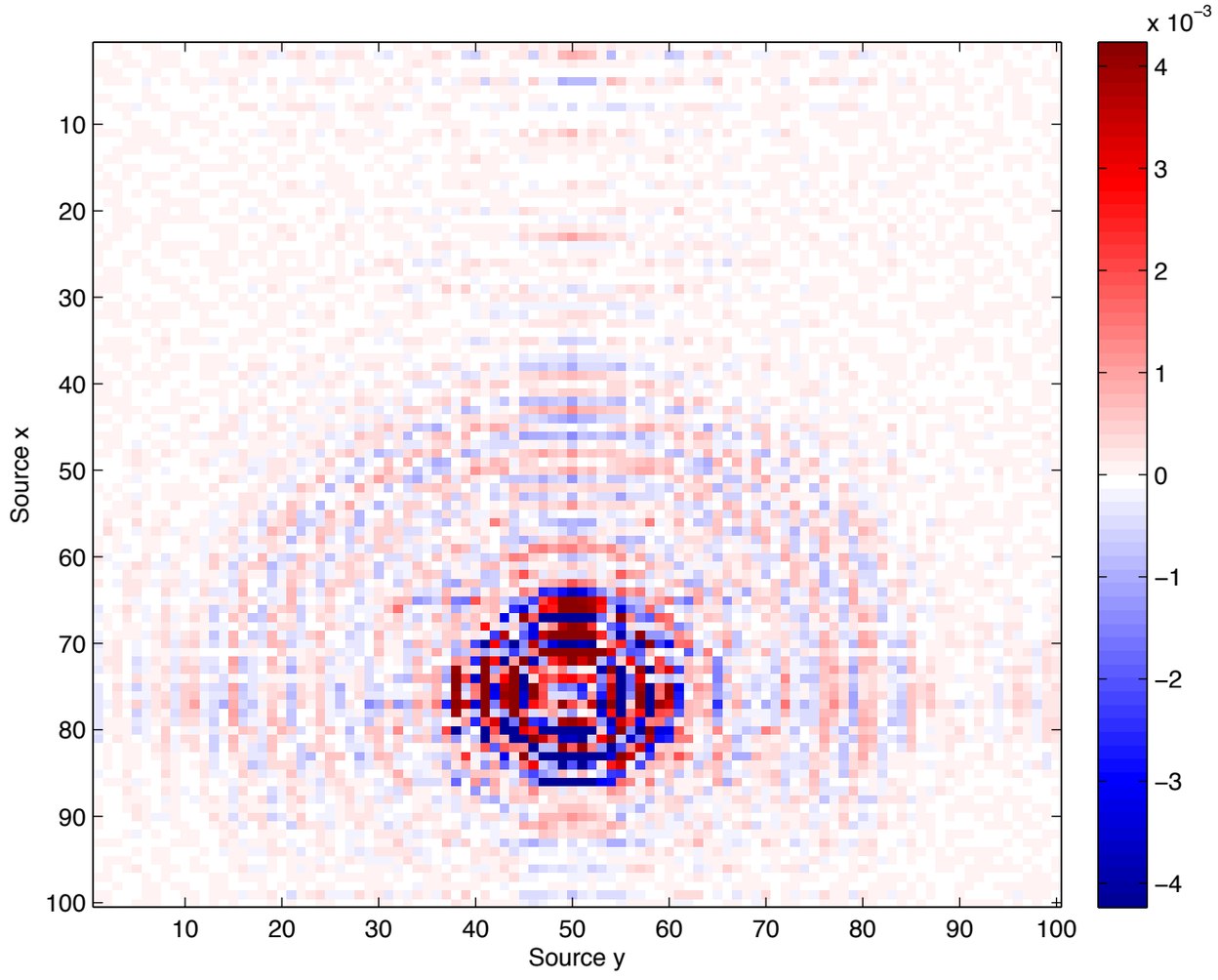
Interpolated data

Common Receiver Gathers

Rec x = 75, Rec y = 50



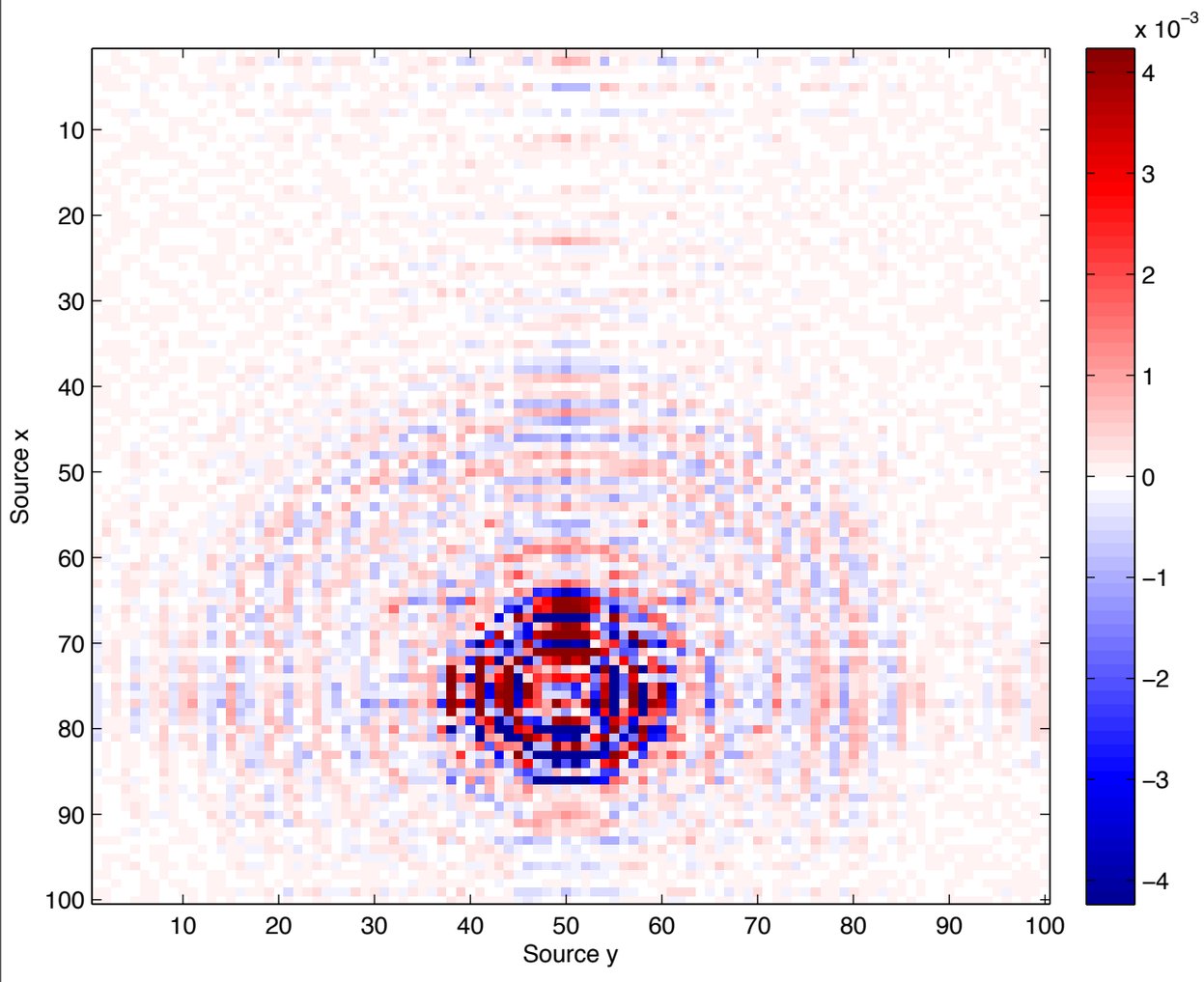
Source Rank 5



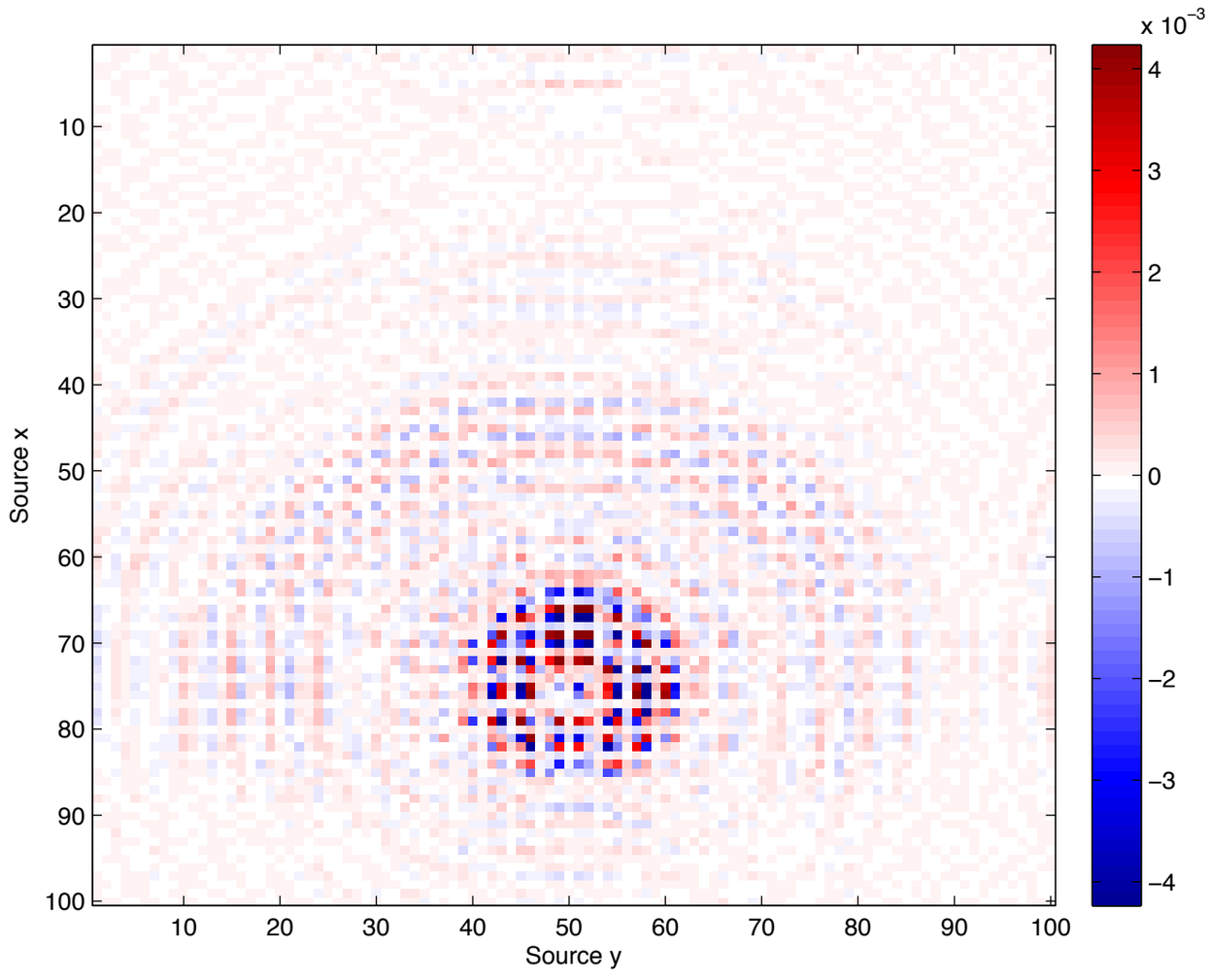
Source Rank 20

Common Receiver Gathers

Rec x = 75, Rec y = 50



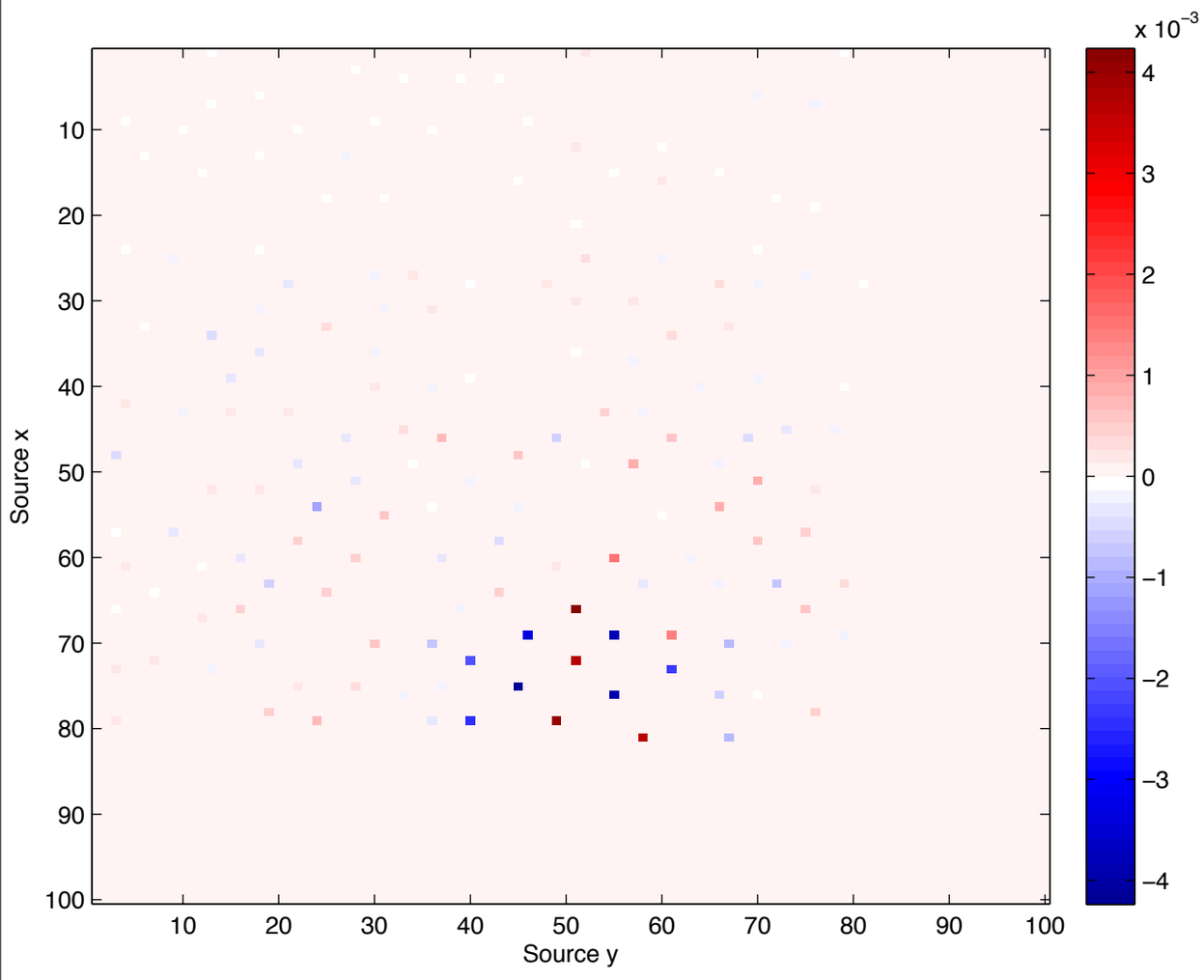
No Regularization



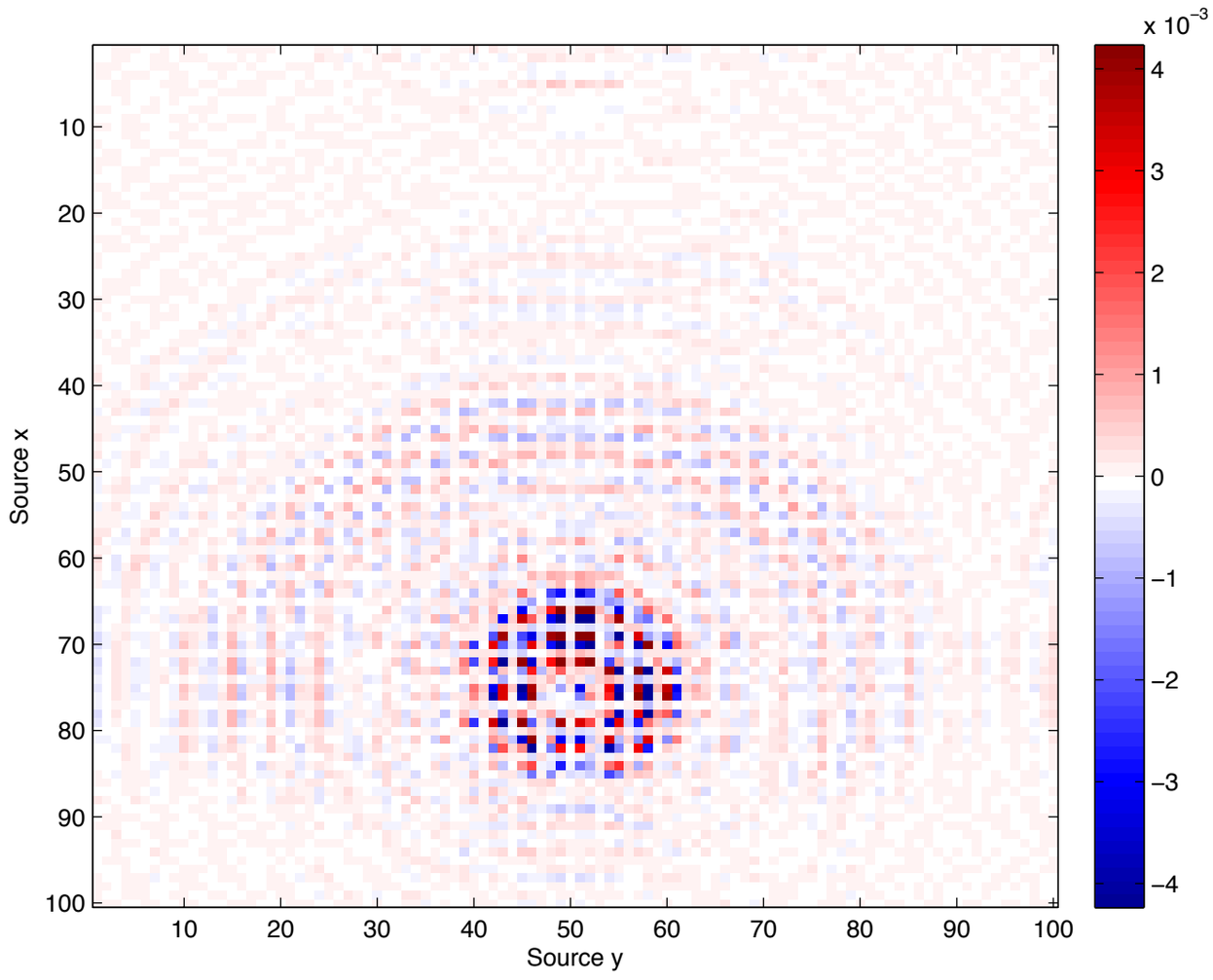
Regularization

Common Receiver Gathers

Rec x = 75, Rec y = 50



Known data

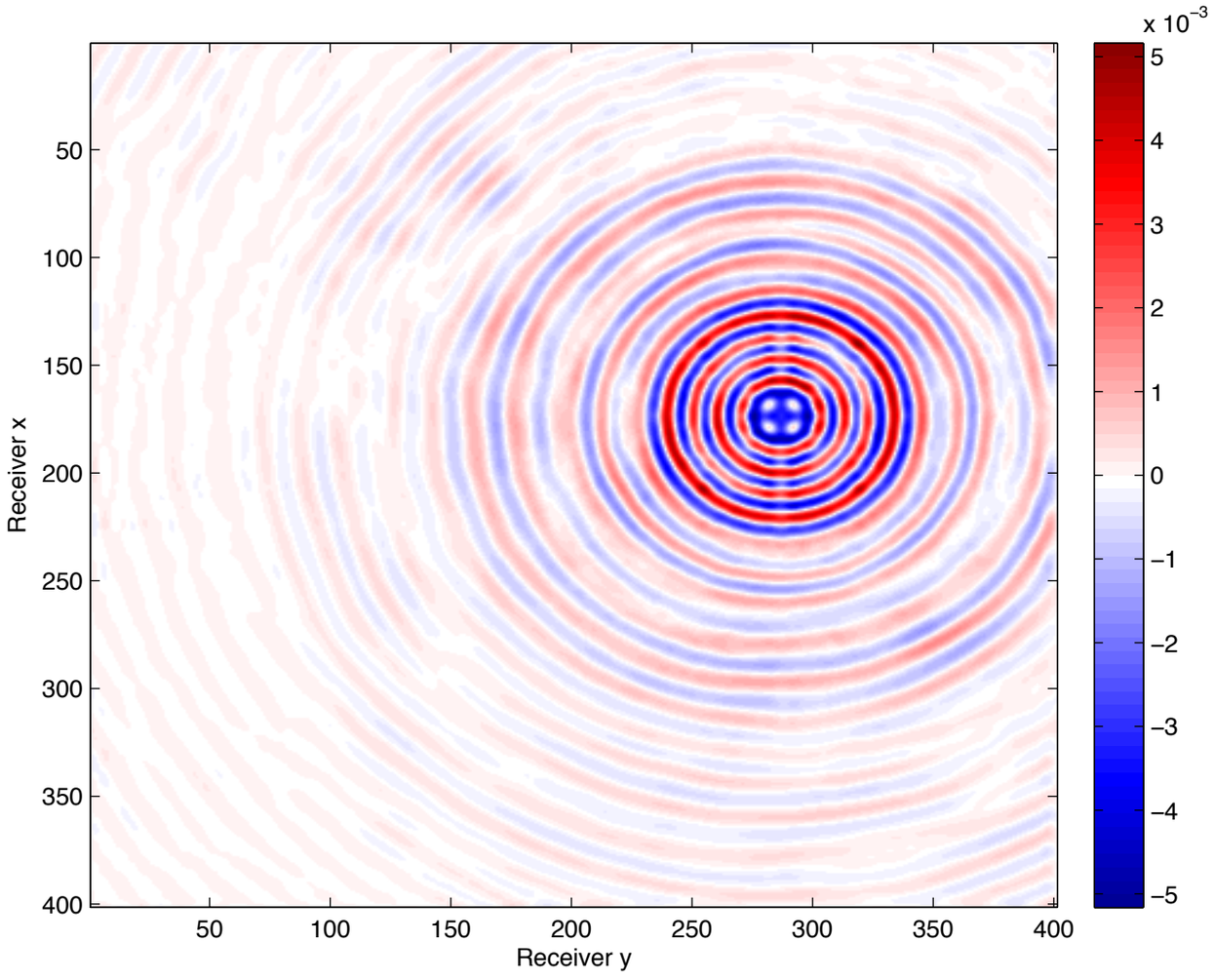


Regularized final result

Shot Reconstruction

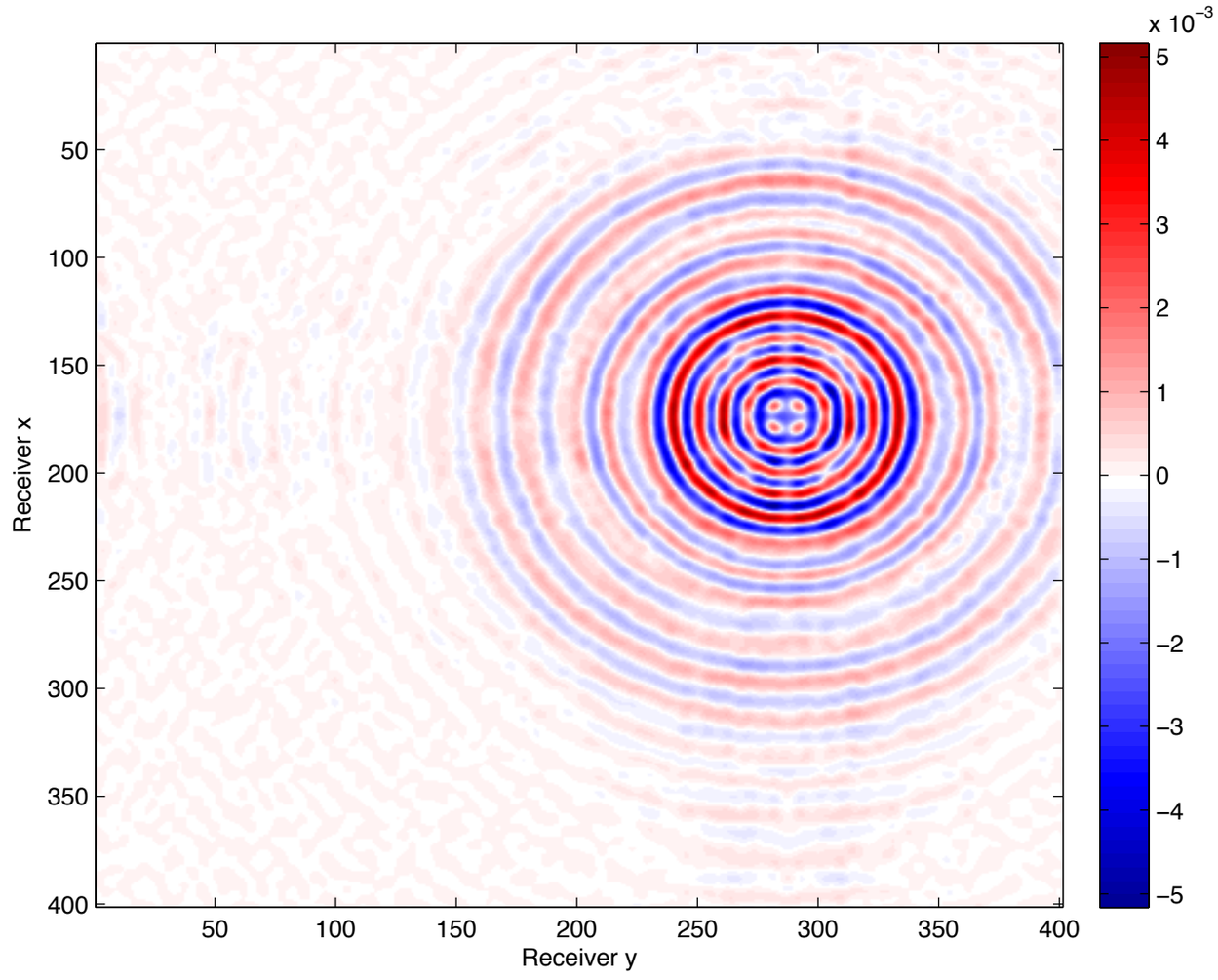
- We consider the reconstruction error of 3 shots on a reduced data set (using the remaining 197 shots)
- We obtain still reasonable recovery results

Shot Reconstruction



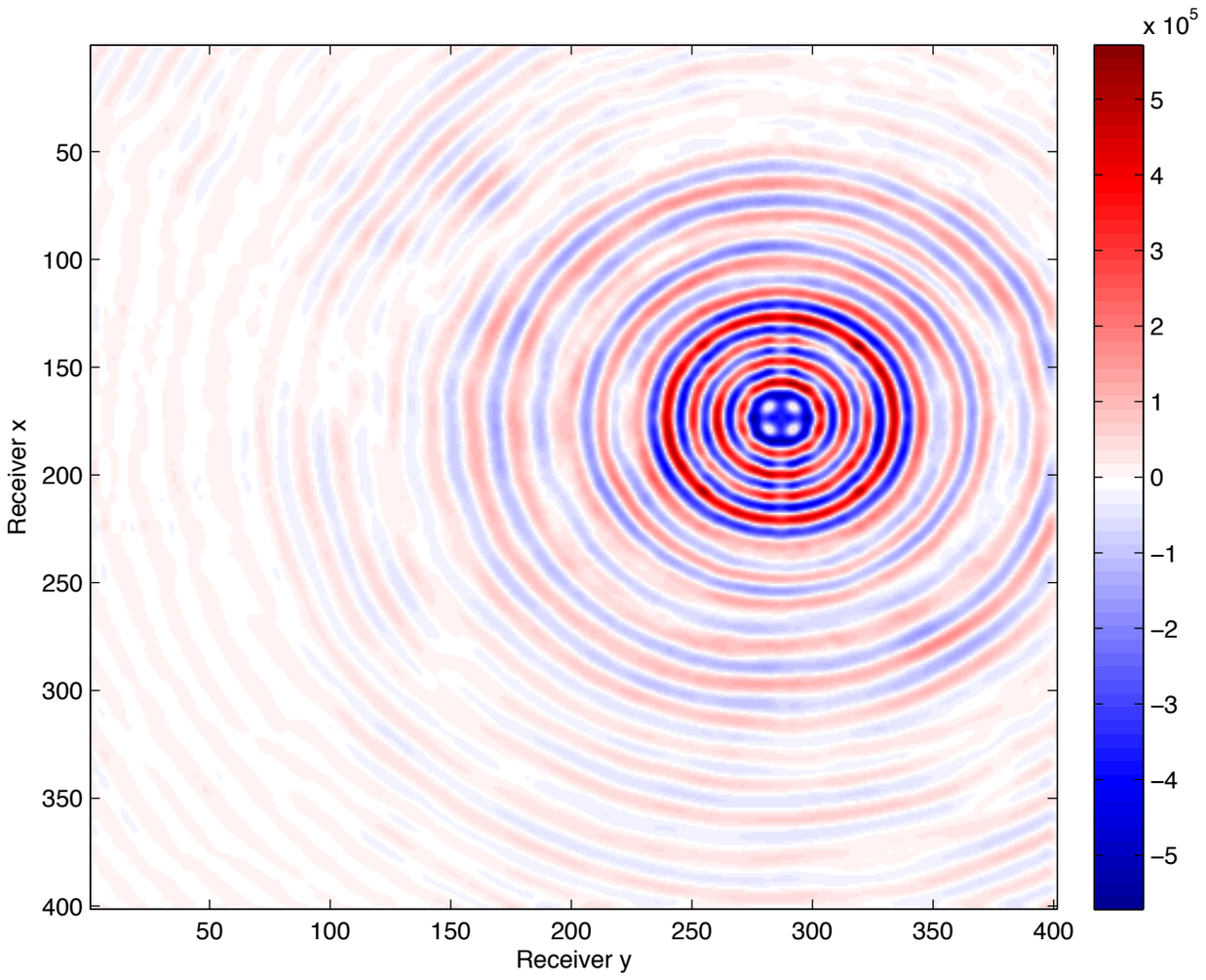
True data

(Src x, Src y) = (45,73)



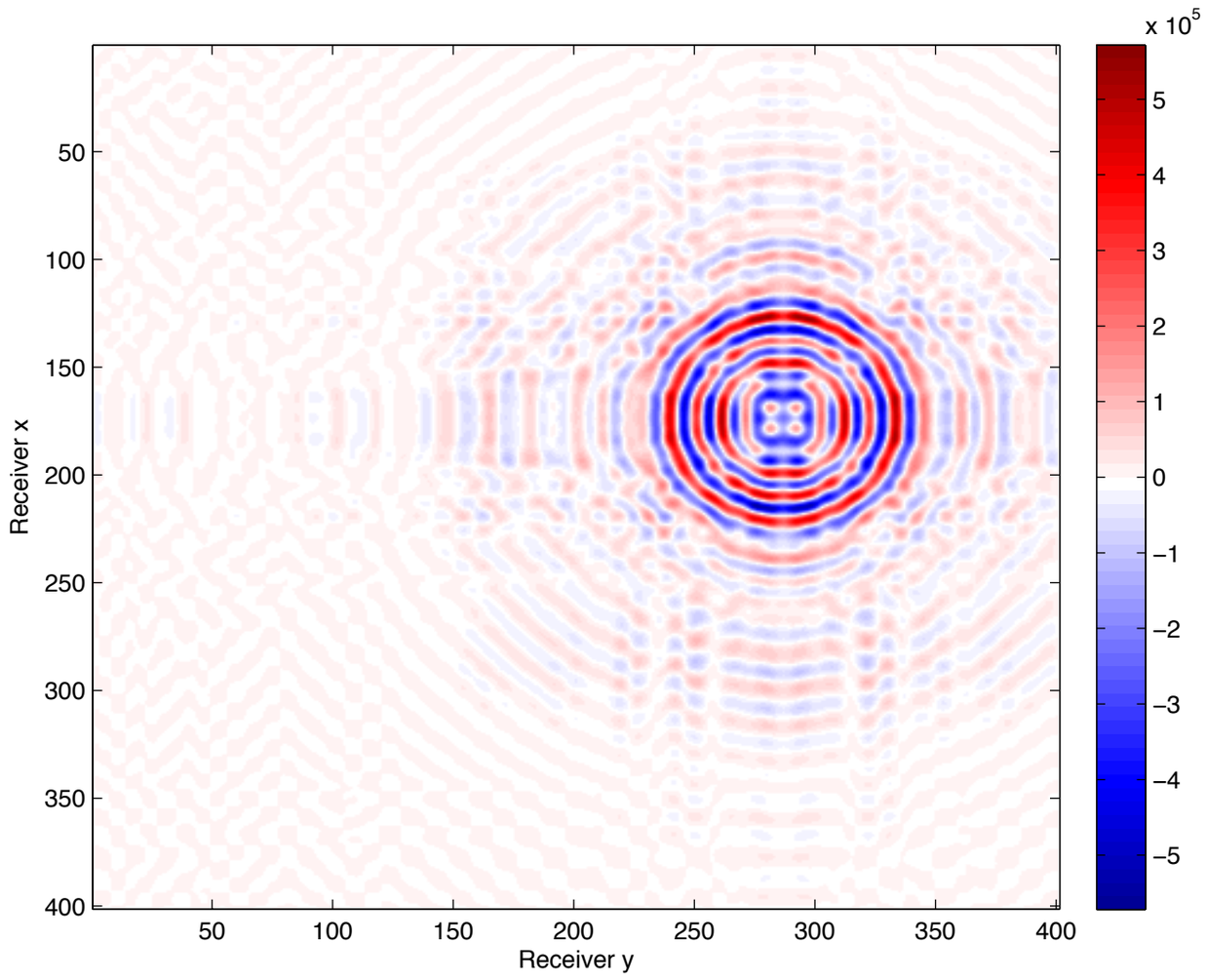
Interpolated data - SNR 8.46 dB

Shot Reconstruction - Jellyfish



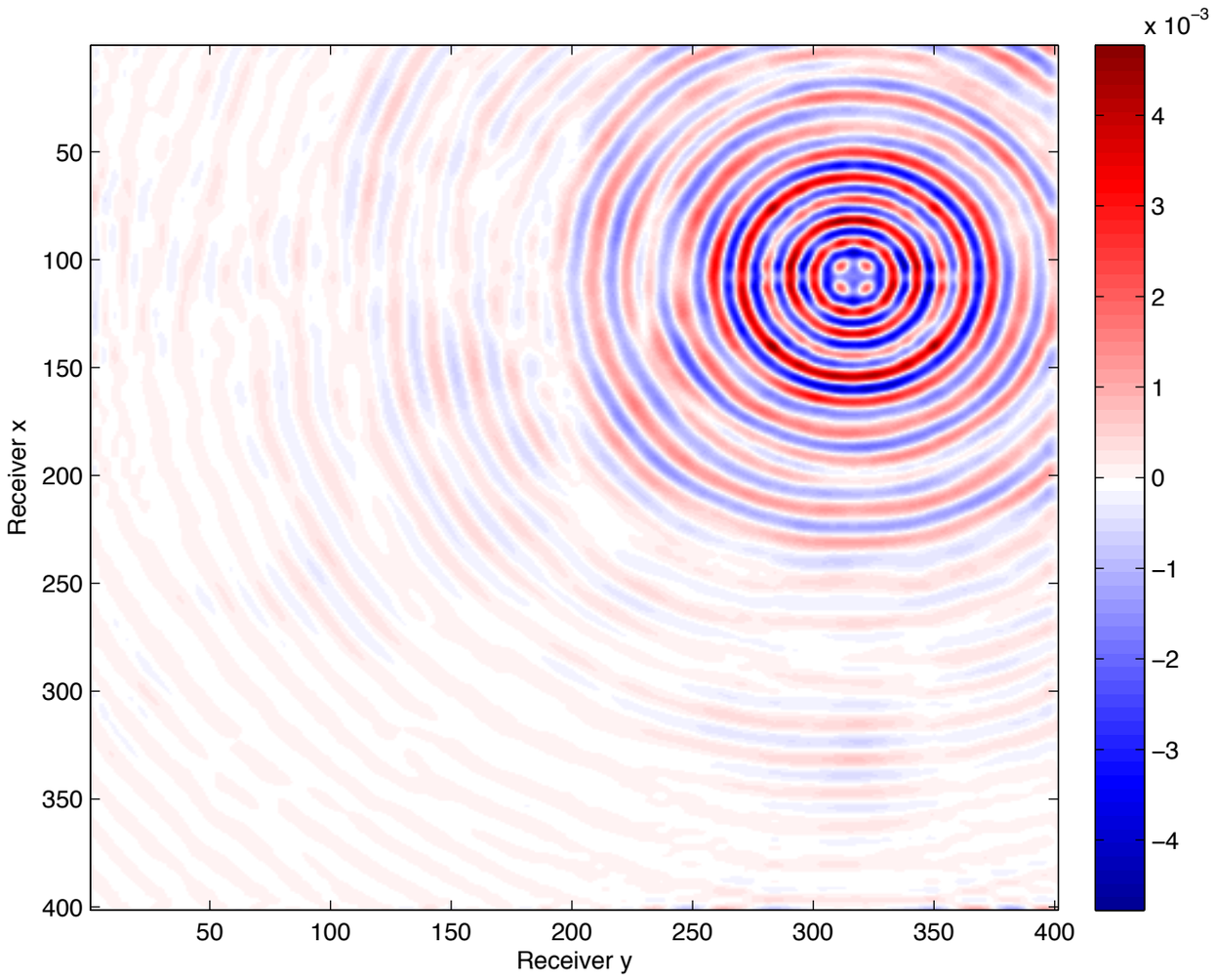
True data

(Src x, Src y) = (45,73)



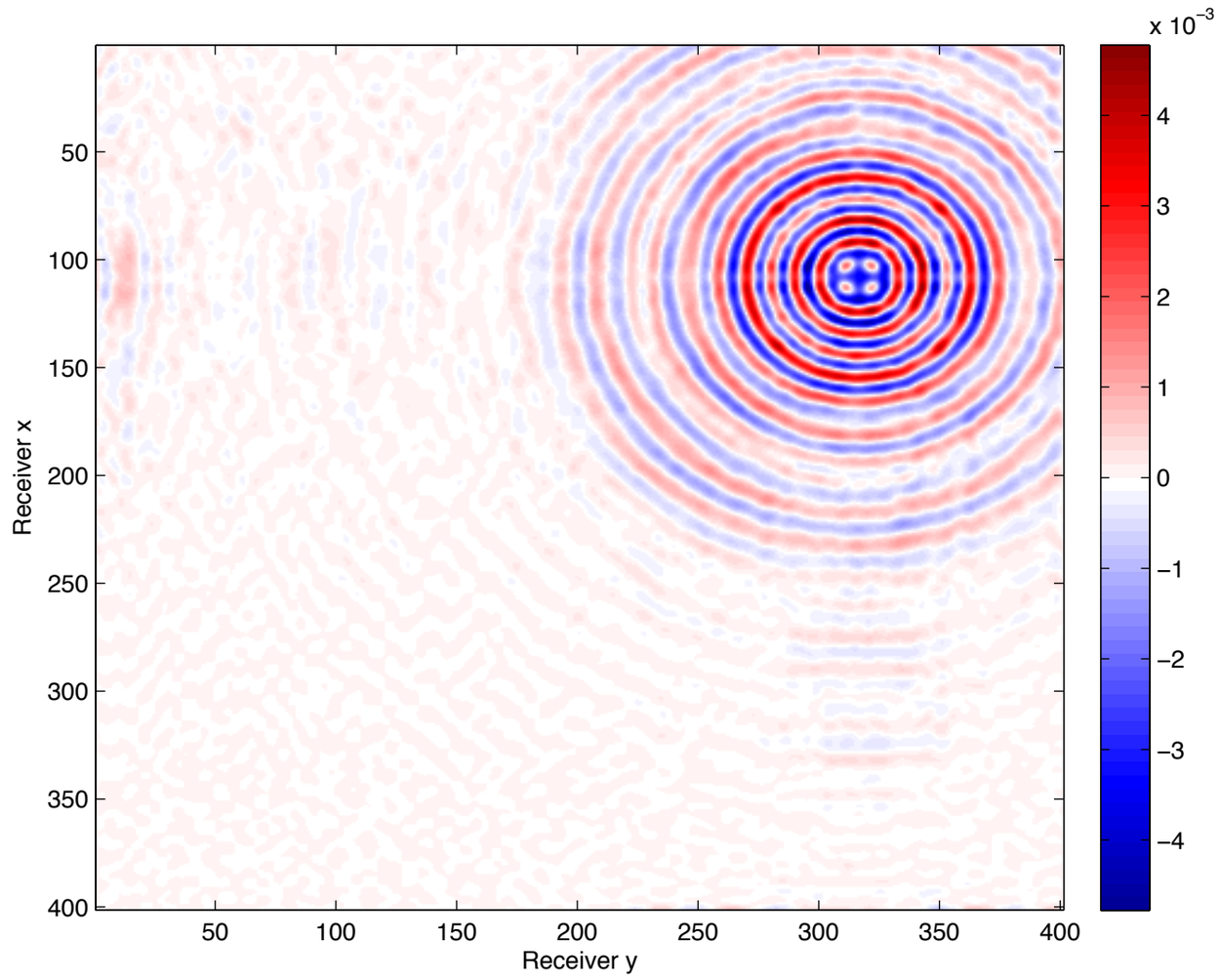
Interpolated data - SNR 3.86 dB

Shot Reconstruction



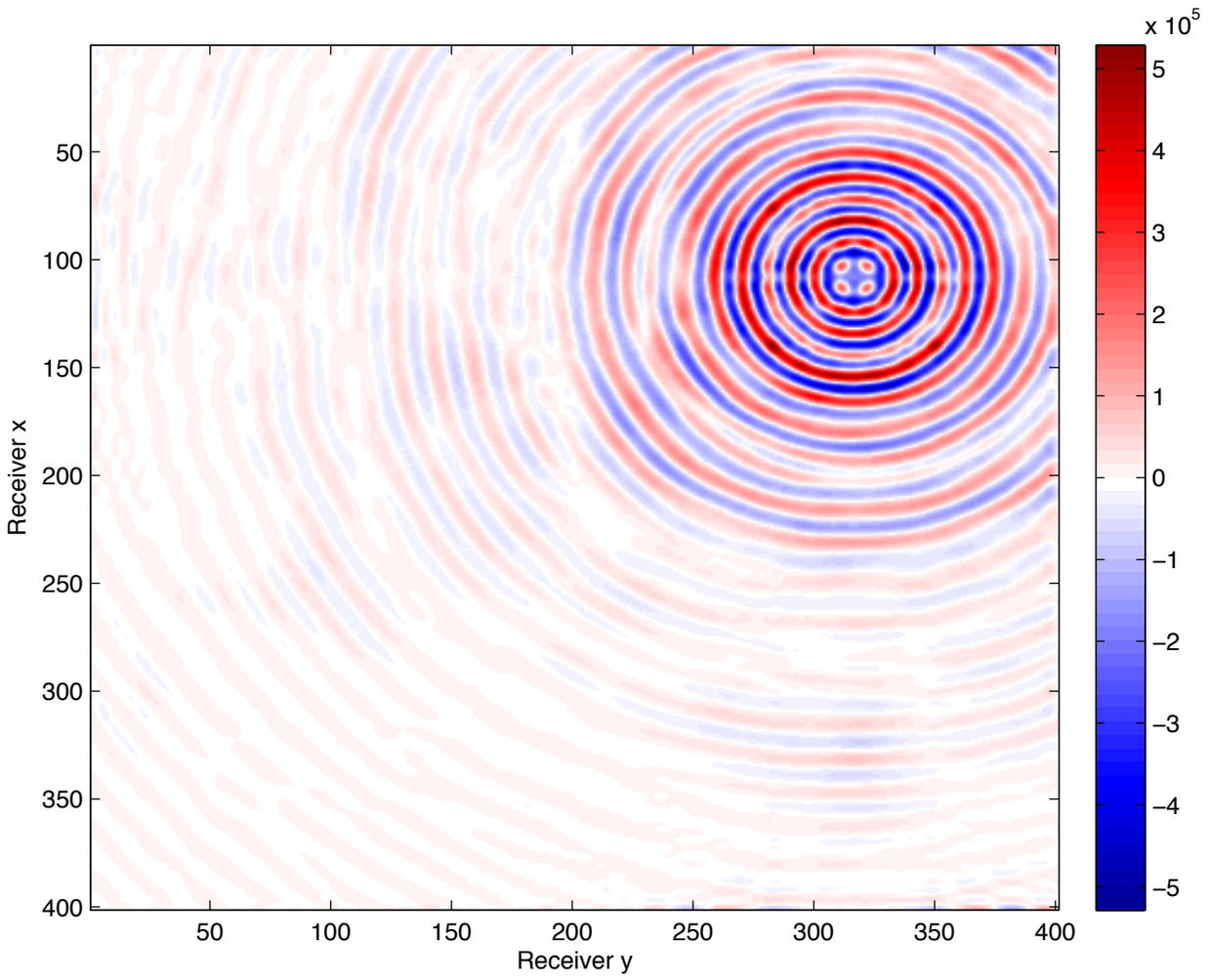
True data

(Src x, Src y) = (28,81)



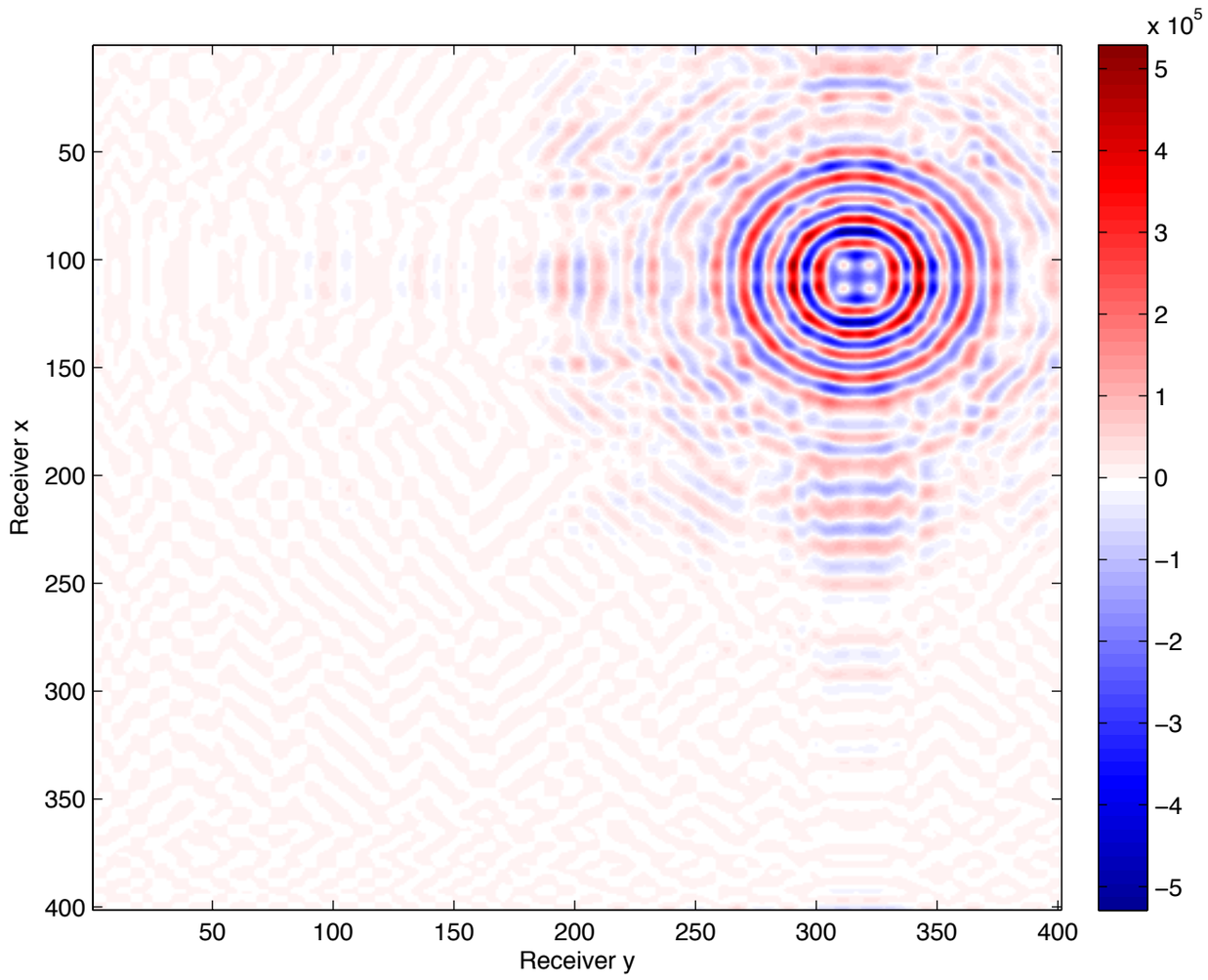
Interpolated data - SNR 7.98 dB

Shot Reconstruction - Jellyfish



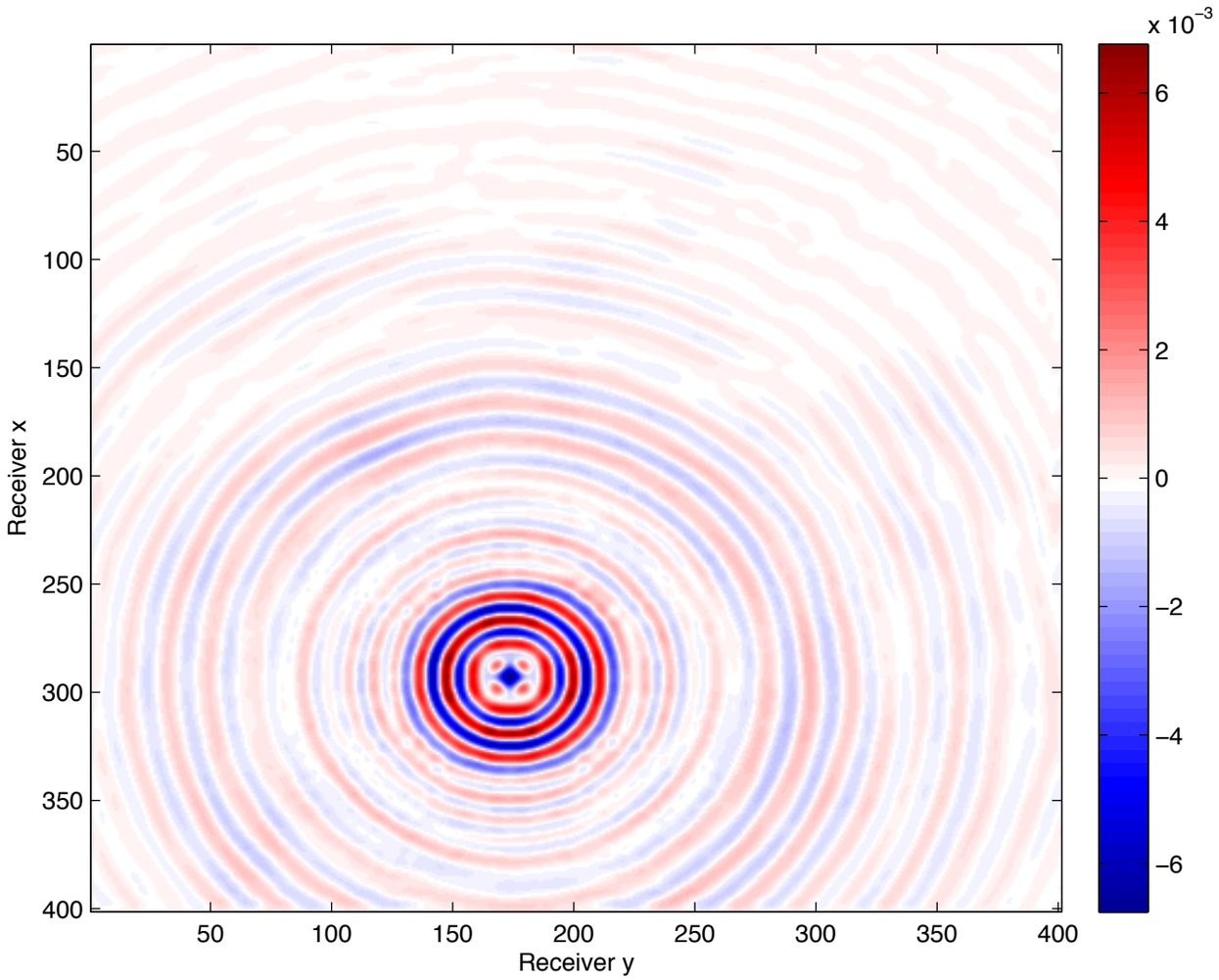
True data

(Src x, Src y) = (28,81)



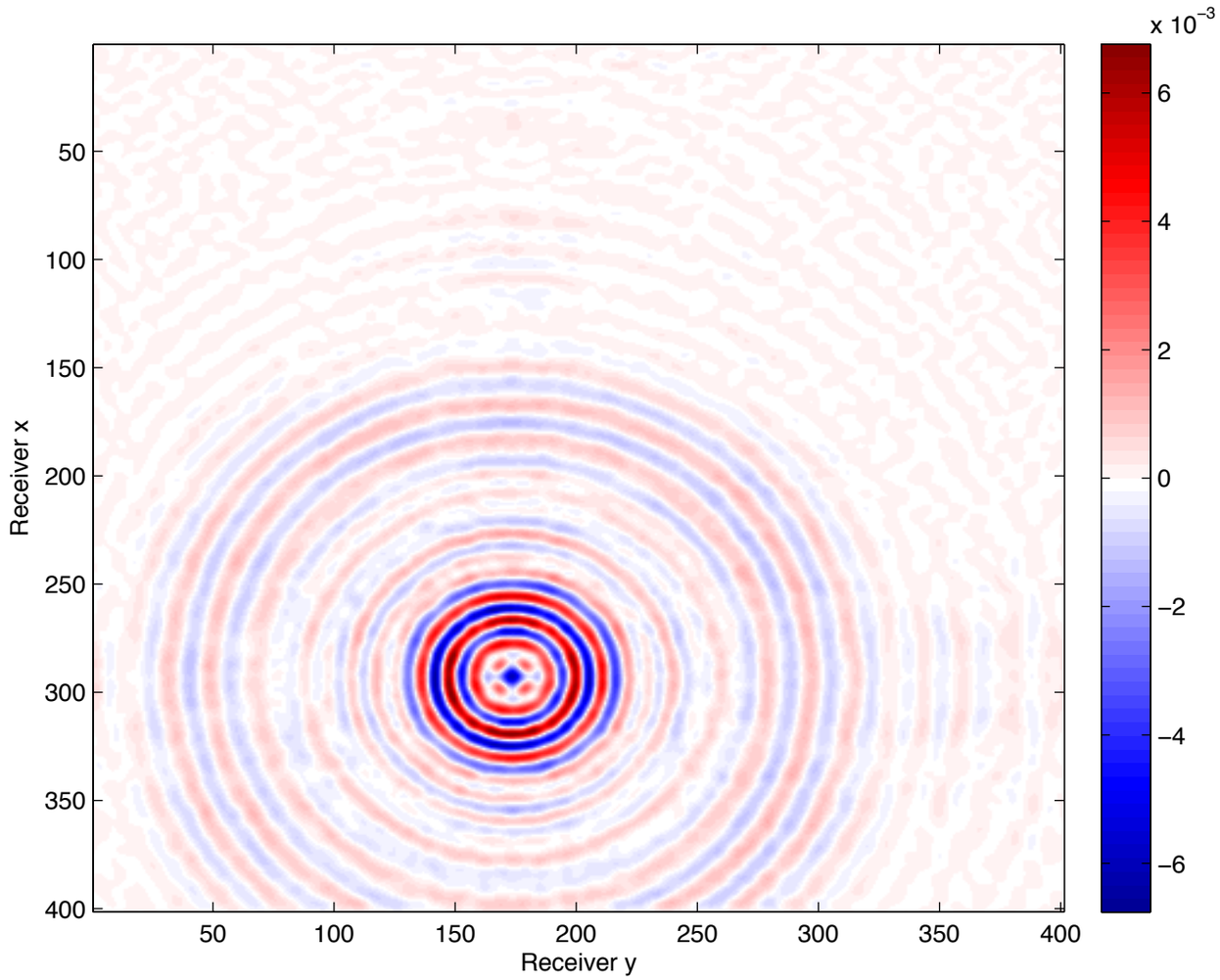
Interpolated data - SNR 3.89 dB

Shot Reconstruction



True data

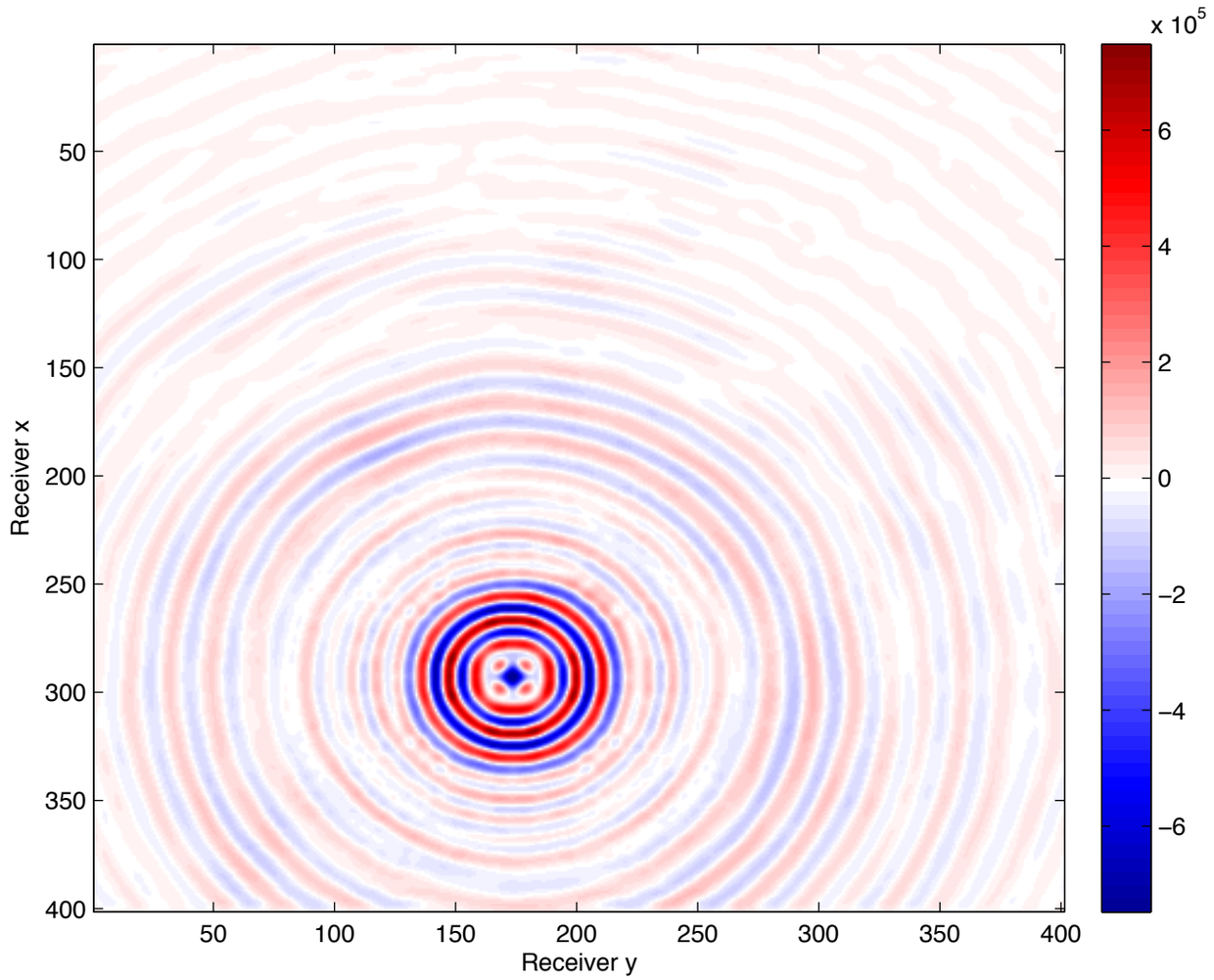
(Src x, Src y) = (75,45)



Interpolated data -

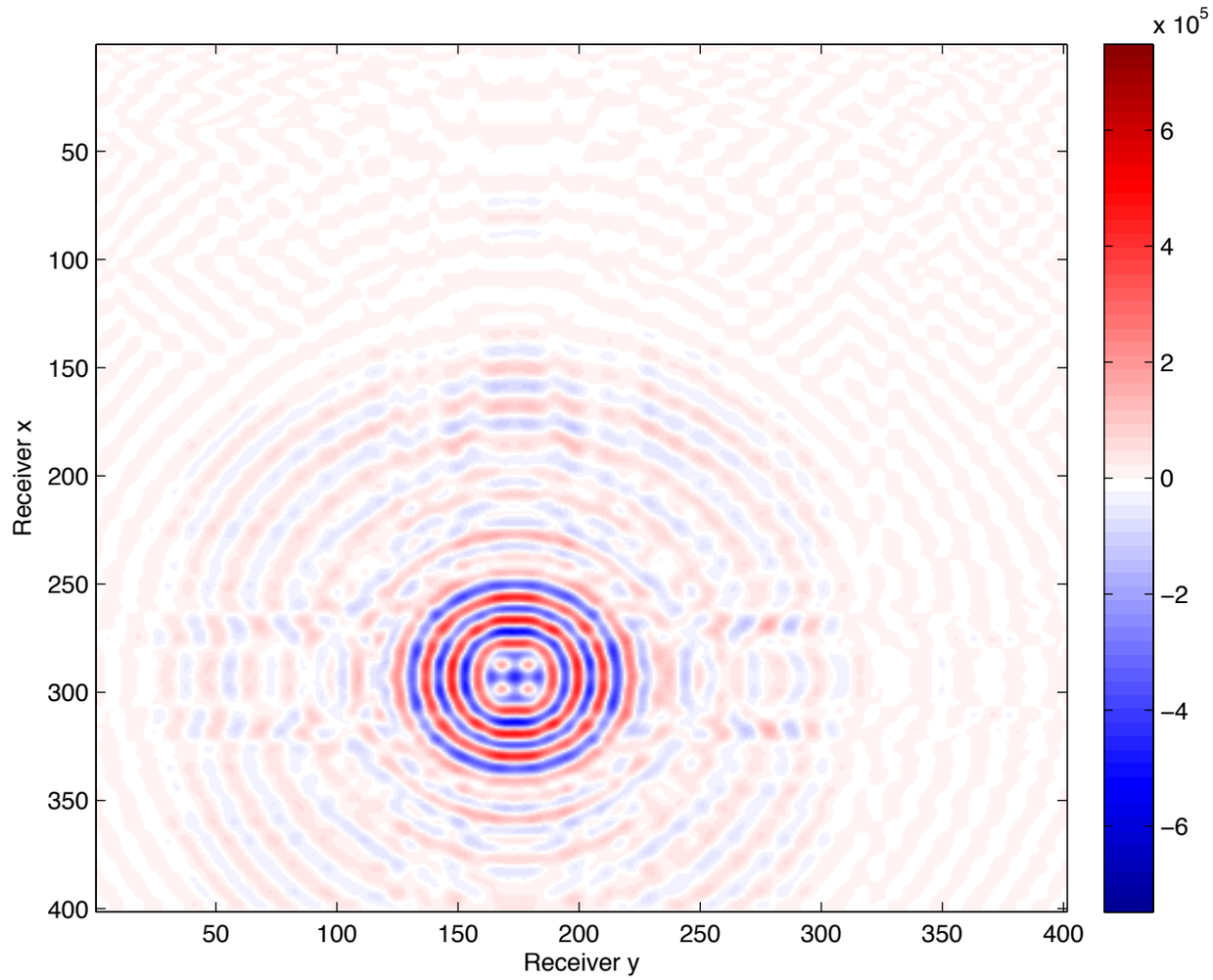
SNR 10.6 dB

Shot Reconstruction - Jellyfish



True data

(Src x, Src y) = (75,45)



Interpolated data - SNR 4.62 dB

Conclusion

- 3D seismic data has an underlying structure that we can exploit for interpolation (hierarchical tucker format)
- The choice of organization of this data has important implications in the success of the recovery

Conclusion

- We can impose low-rank regularization of the underlying matricizations of the tensor *without* having to form them explicitly/ compute SVDs
- Achieve reasonable results from hardly any data at all (2% of sources)

Future Work

- Using the Hierarchical Tucker format (or other tensor formats) to speed up multidimensional convolution
- Automatic parameter selection for regularization
- Adapt this approach to higher frequency data, which tend to be higher rank

Future Work

- Impose regularization based on an understanding the physics underlying these data sets
- E.g. use knowledge of the behaviour of wave-propagation in 3D to prevent spurious artifacts

Future Work

- Apply the Hierarchical Tucker format to high-dimensional image volumes for computing outer products efficiently
- Image focusing

Acknowledgements

Thank you for your attention

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, BP, Chevron, ConocoPhillips, Petrobras, PGS, Total SA, and WesternGeco.