Frugal FWI

Felix J. Herrmann and Tristan van Leeuwen

SLIM
Seismic Laboratory for Imaging and Modeling
the University of British Columbia

Motivation

FWI is a relatively *immature* technology in need of

- less *reliance* on *accurate* starting models
- *automated* & *versatile* workflows
- better *theoretical* understanding

Today's focus is the development of *resourceful* FWI

- *small* subsets & *dynamic* accuracy
- *turnaround* times that may allow for QC & UQ

Today's agenda

Goals are

- introduction of *fast* simulation & optimization techniques
- *integration* into *versatile* FWI framework that *spends your computational resources only when needed*...

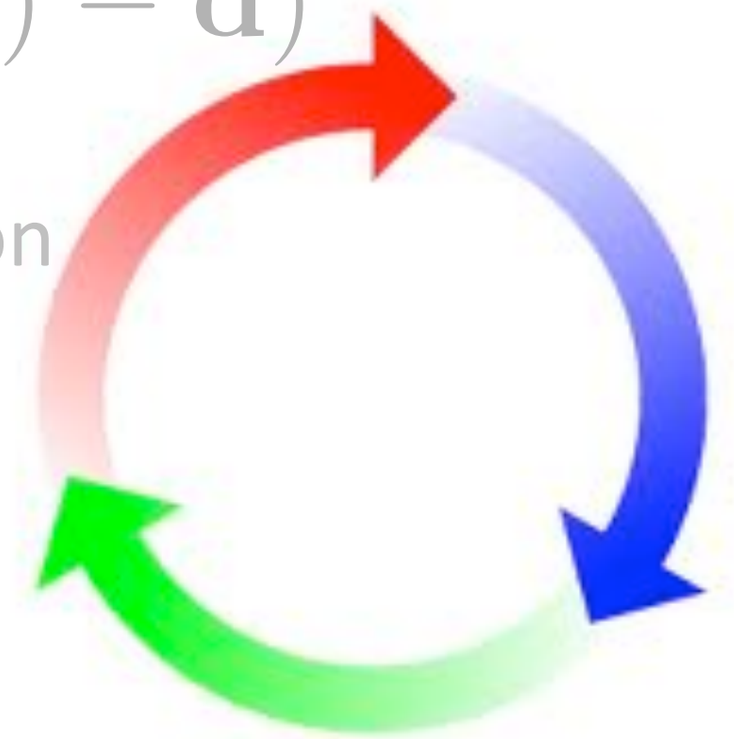
Challenges & opportunities

Fast FWI

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

robust

formulation



$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

versatile

modelling

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

***fast* optimization strategies**

computational framework

Fast optimization

Strategy:

- *reduce* costs by working w/ random *subsets* of sources
- allow for *inaccurate* physics (e.g., PDE solves)
- *convergence* guarantees via *dynamic* accuracy *control*
  - *dynamic* increase *size* subsets & *accuracy* PDE solves

Outcome:

- computationally *affordable* scheme for 2-D & 3-D FWI

Fast optimization

separable structure

misfit per source

$$\min_{\mathbf{m}} \Phi(\mathbf{m}) = \frac{1}{M} \sum_{i=1}^{M} \phi_i(\mathbf{m})$$

solution with *steepest descent*

$$\mathbf{m}_{k+1} = \mathbf{m}_k - \lambda_k \nabla \Phi(\mathbf{m}_k)$$

requires evaluation of *full* misfit and is *very* expensive

[Bertsekas '96,'08; Nemirovski '00]

Fast optimization
with *errors*

Allow *errors* in *gradients*–i.e.,

$$\widetilde{\nabla\Phi}_k = \nabla\Phi_k + \mathbf{e}_k$$

error in gradient

- *draw independent* source *aggregates* (*sup*ershots) or *sub*sets of *sequential* sources after each *model* update
- *stochastic*/incremental gradients

Leads to *sub*linear *convergence* & to *instabilities* due to *noise*

[Friedlander & Schmidt '12, Aravkin et.al. '12]

Fast optimization
with *convergence* guarantees

Approximate gradients by *sample averages*–i.e.,

$$\nabla\Phi \approx \nabla\widetilde{\Phi} = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\nabla\phi_i$$

and *bounding errors* (B_k) by *growing* the *sample size* of *subsets*.

Linear convergence if *error* at *iteration k* bounded by

$$||\mathbf{e}_k|| \leq B_k = \mathcal{O}(\gamma^k)$$

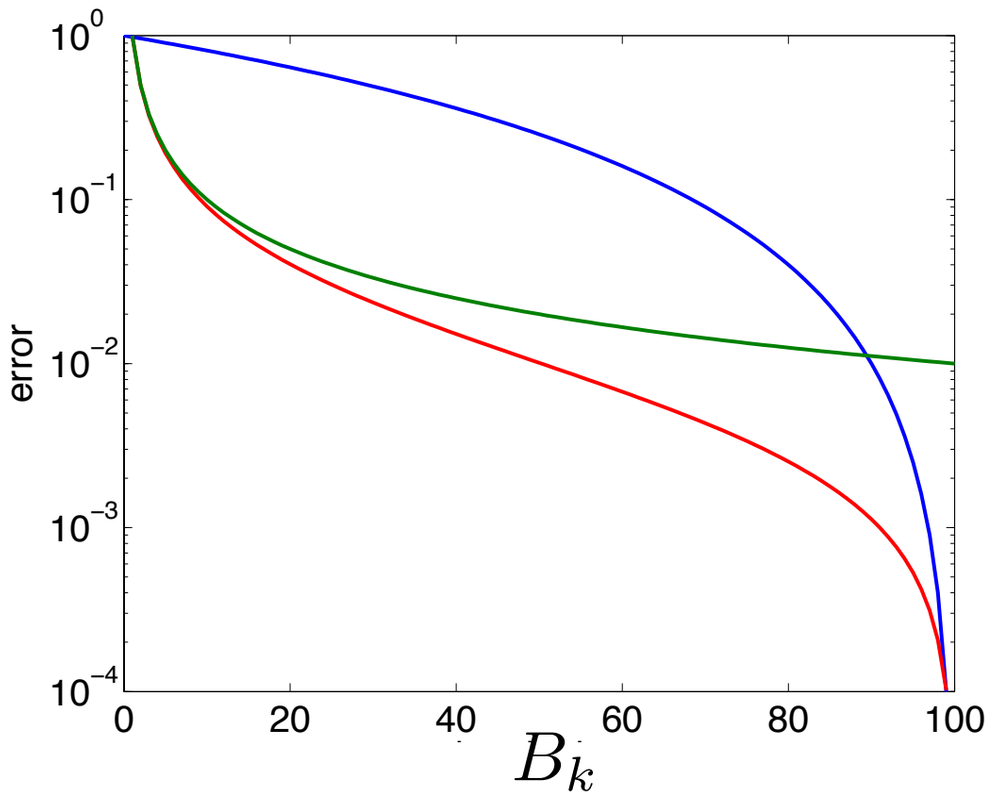
with γ the (unknown) *convergence* rate.

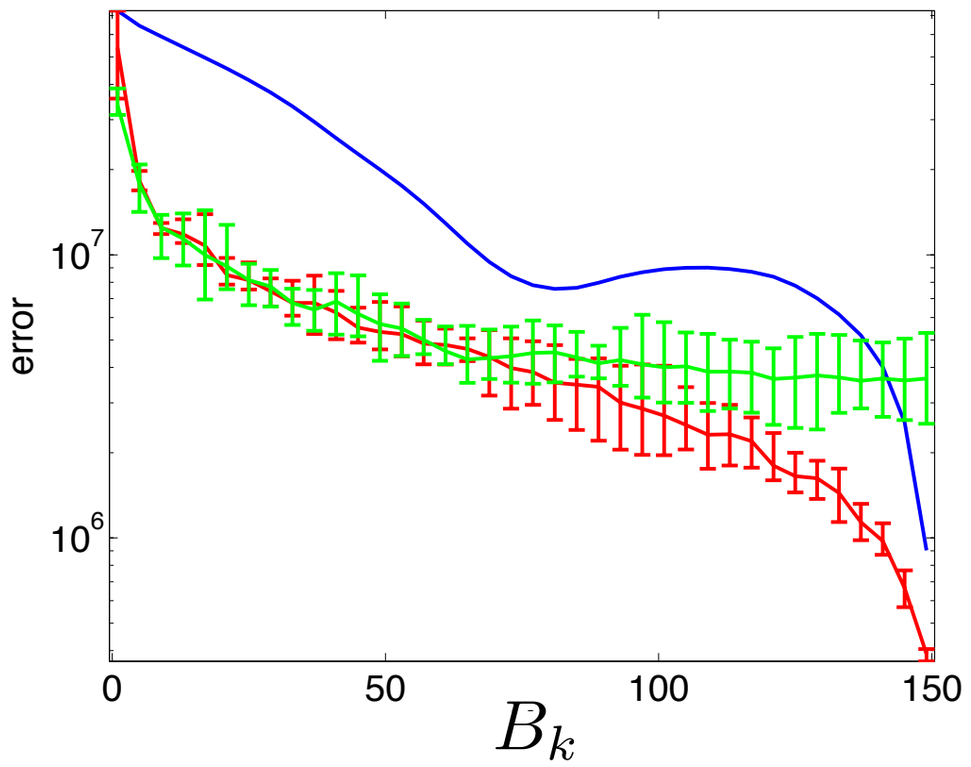
Fast optimization

increase sample size

Select sources

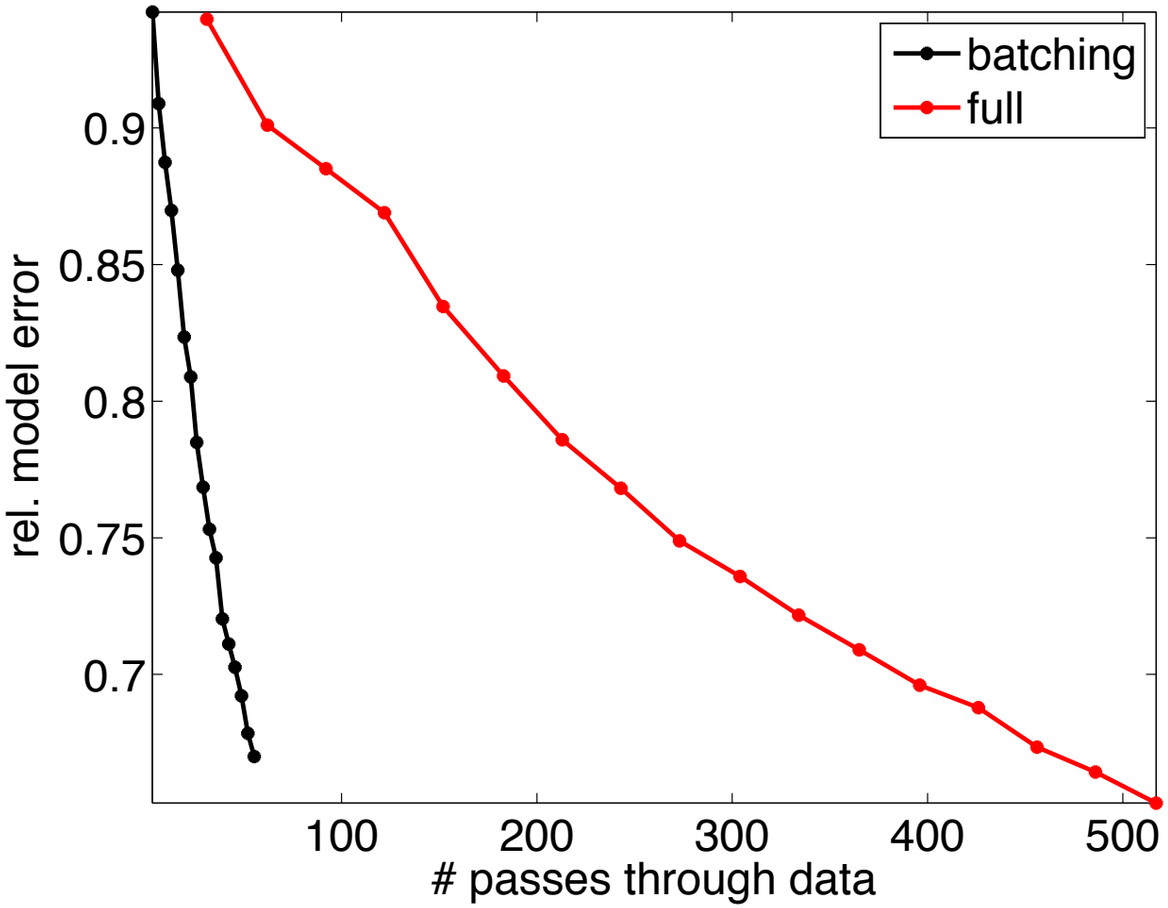
- in a pre-scribed order
- random *without* replacement
- random-*amplitude* source encoding

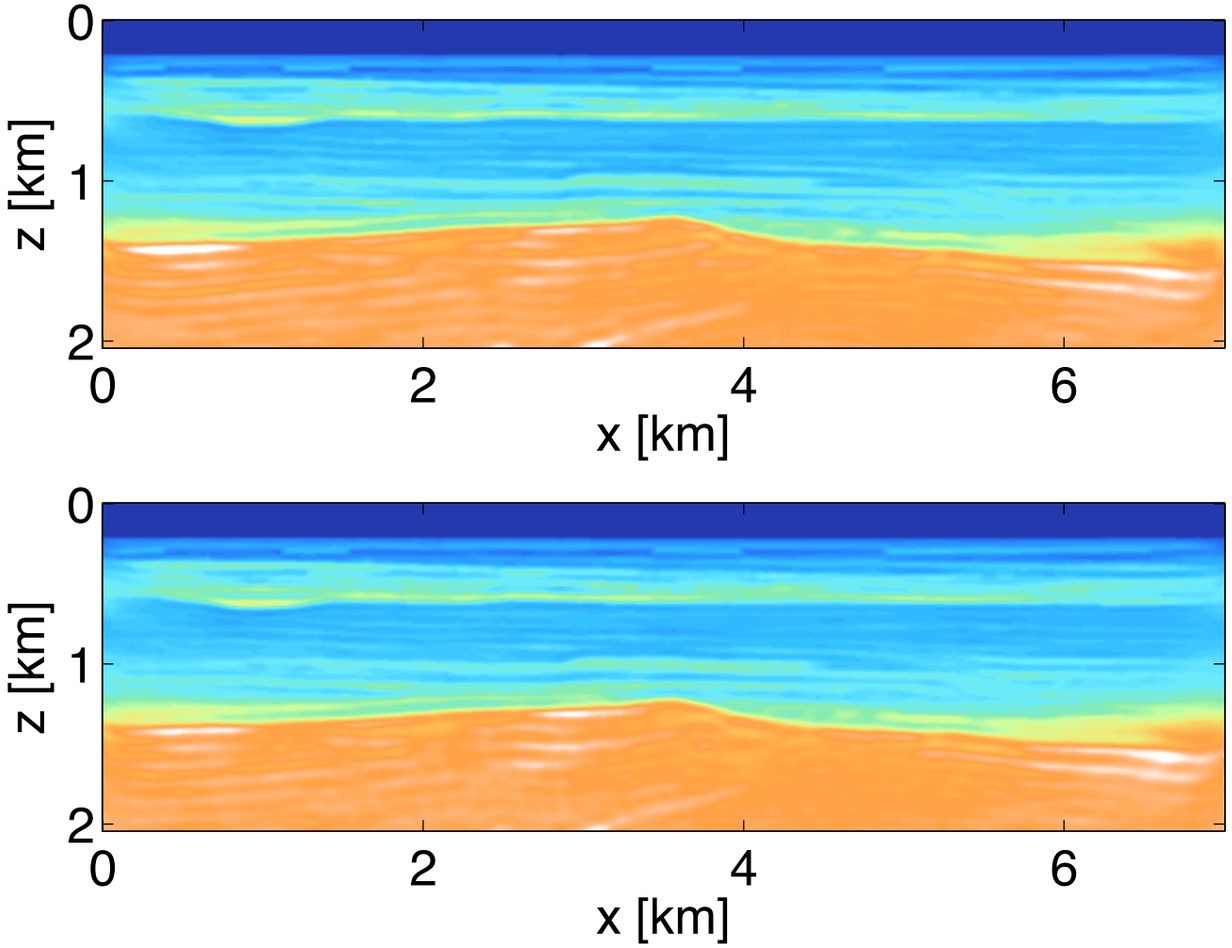




Fast optimization

10 x speedup





[van Leeuwen & FJH ’11]

Fast optimization

w/ *approximate* misfits & gradients

Frame work:

- allows for errors in misfit & gradient calculations
  - *limited* sample size and/or *imprecision* wave simulator
- *convergence* when error *bounded* by convergence *rate*
  - *inaccurate* calculations in *beginning* / when problem *ill-posed*

Challenge:

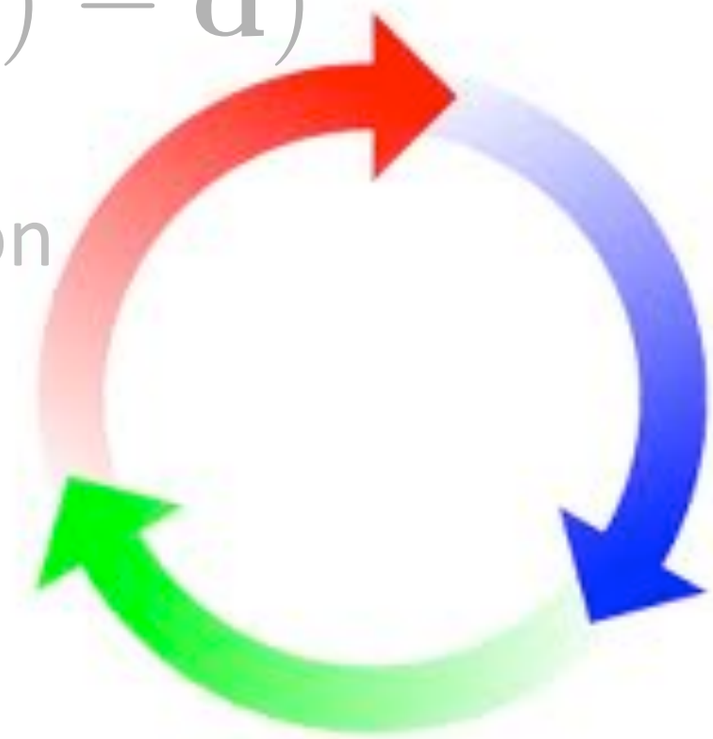
- *translate* into *practice* when *accuracy* & *convergence* unknown

Fast FWI

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

robust

formulation



$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

versatile

modelling

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

fast optimization strategies

computational framework

Versatile modelling

Strategy:

- avoid large *setup, memory* costs & *tuning* parameters
- offer *control* on *precision* wave simulations
  - by *increasing* number of *iterations* indirect solvers

Outcomes:

- *scalable* parallel wave simulations w/ prescribed *tolerance*
- simple *preconditioner* that works for different WE's

Gordon & Gordon '11-'12
van Leeuwen et. al. '12

CGMN & CARP-BCG

Use *simple* Kaczmarz *row* projections

$$\mathbf{x} := \mathbf{x} + \frac{\lambda}{||\mathbf{a}_i||_2^2} \left(b_i - \mathbf{a}_i^T \mathbf{x}\right) \mathbf{a}_i,$$

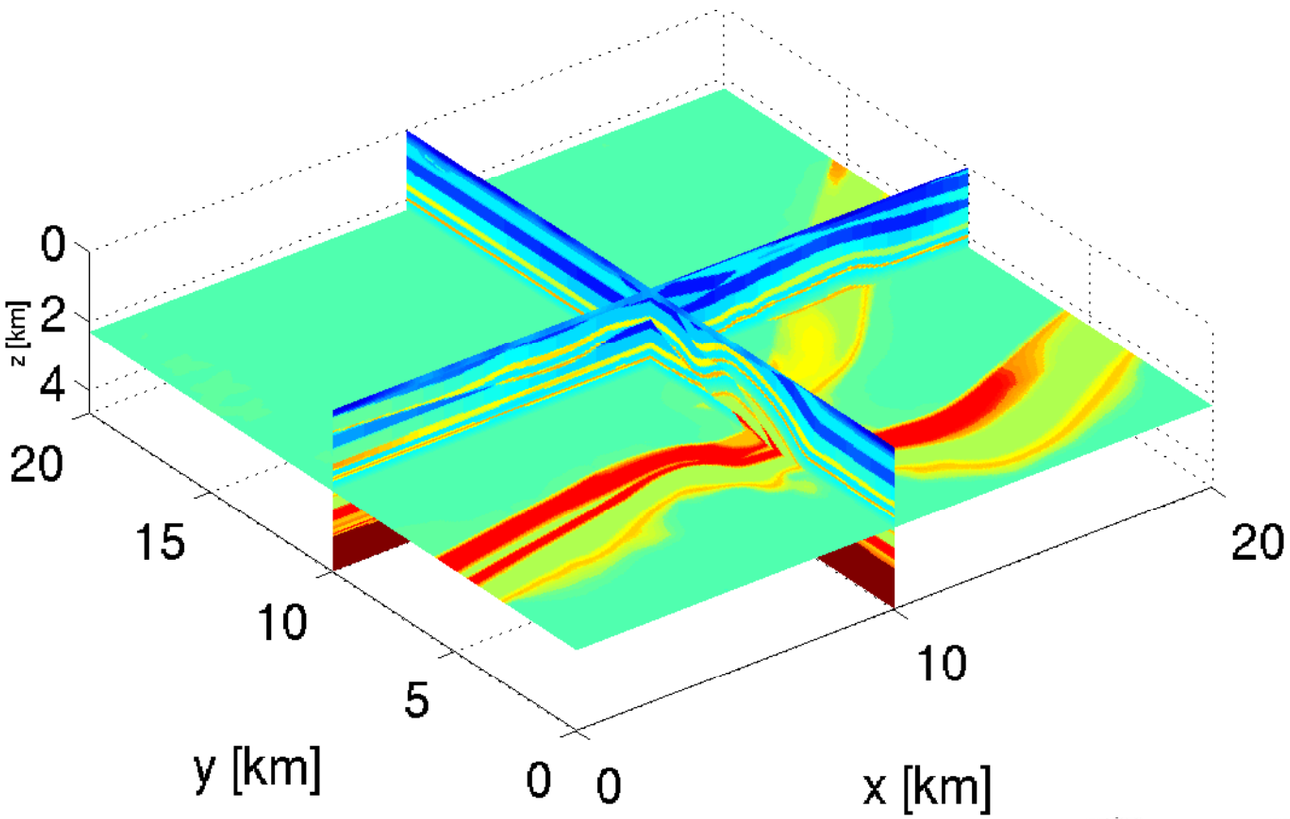
to form a *preconditioner* with *double* sweeps that

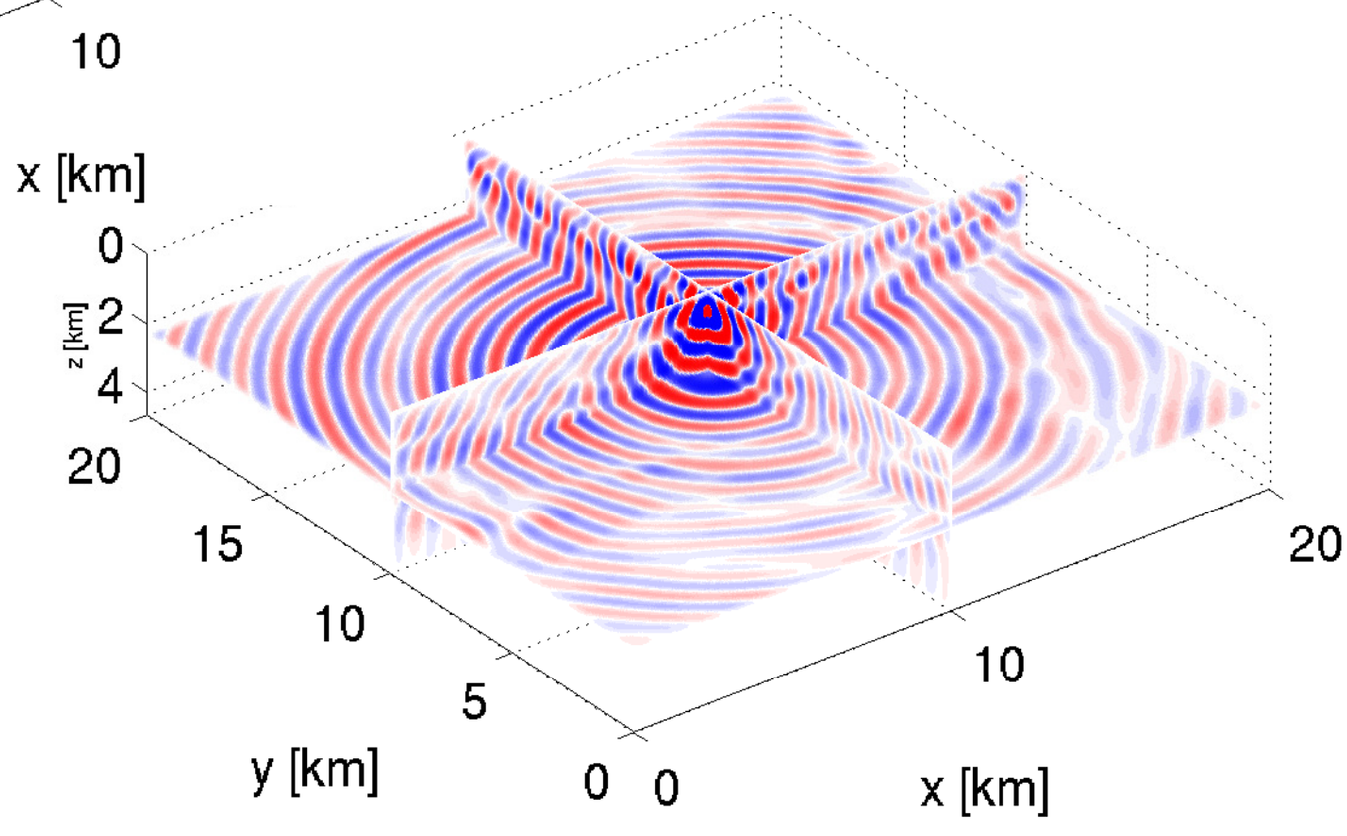
- deals with *multiple* right-hand-sides *simultaneously*
- is *parallelizible* by *projecting* row blocks *independently*
- can be *accelerated* by CG

Simple *scalable* algorithm with *controllable* accuracy...

Overtrust model

4.5 Hz in 1633s





27 point stencil

10 pts per wavelength

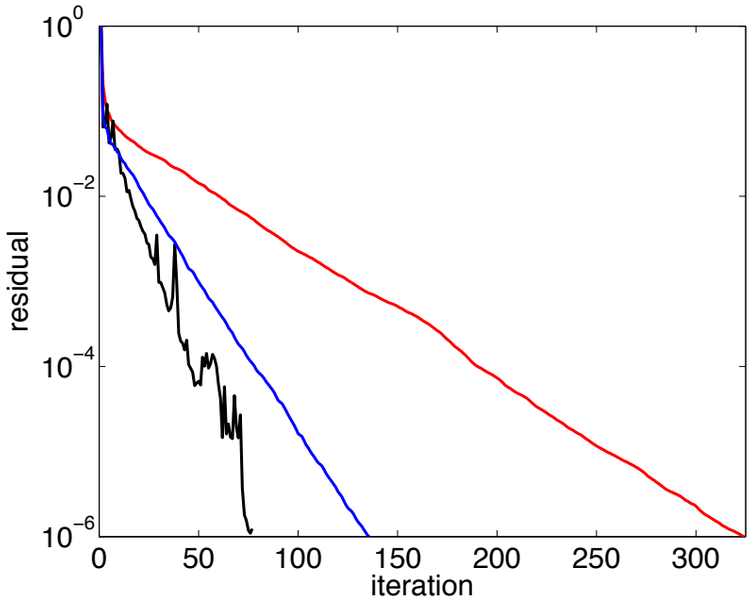
PML

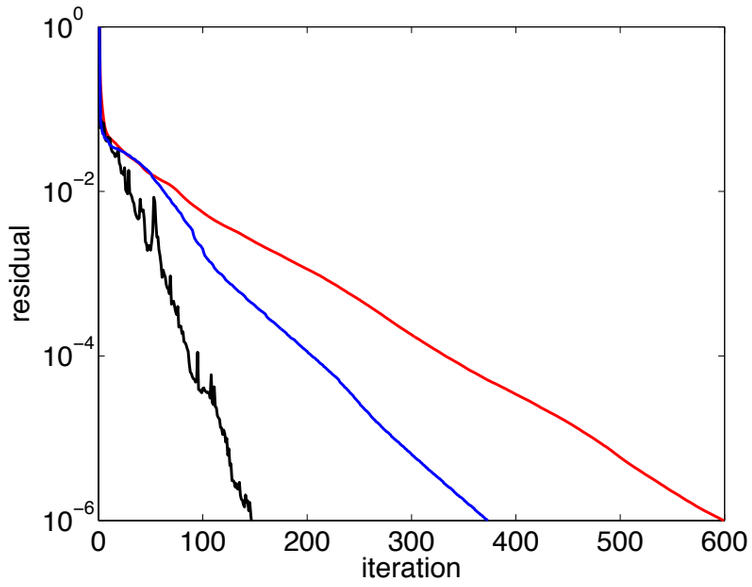
5km X 5km X 2.5Km

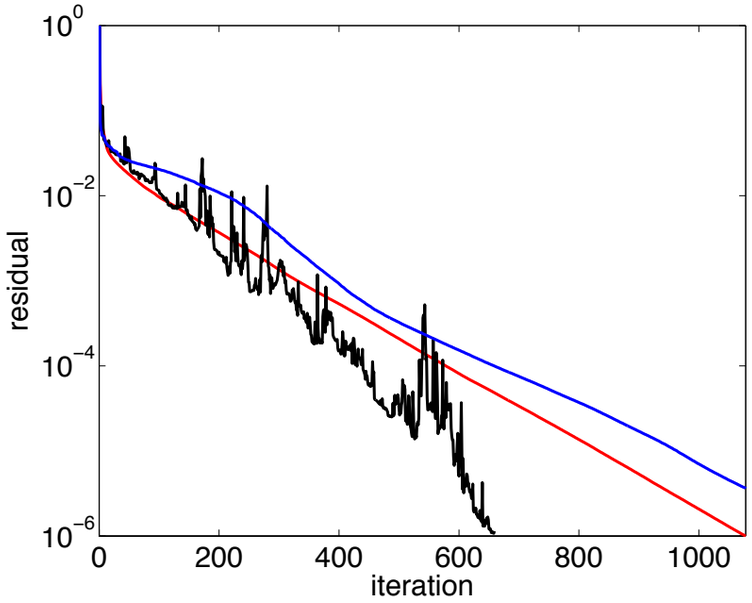
CGMN

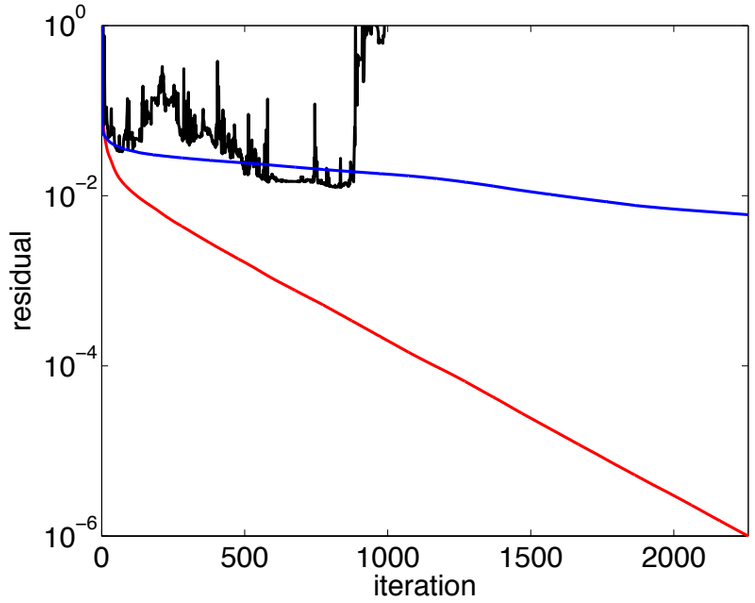
0.5, 1, 2, 4.5 Hz
non-optimized

comparison Bigstab, GMRES(5), CGMN









CGMN

0.5, 1, 2, 4.5 Hz

non-optimized

		CGMN		BiCGstab		GMRES(5)	
f [Hz]	N	iter	time [s]	iter	time [s]	iter	time [s]
0.5	31212	324	7.3	77	1.4	135	1.8
1.0	244824	599	117.5	146	26.9	150	43.0
2.0	1898847	1077	1575.7	659	848.2	747	1048.3
4.5	15115294	2259	28220.7	817*	12174.9	5000*	38340.8

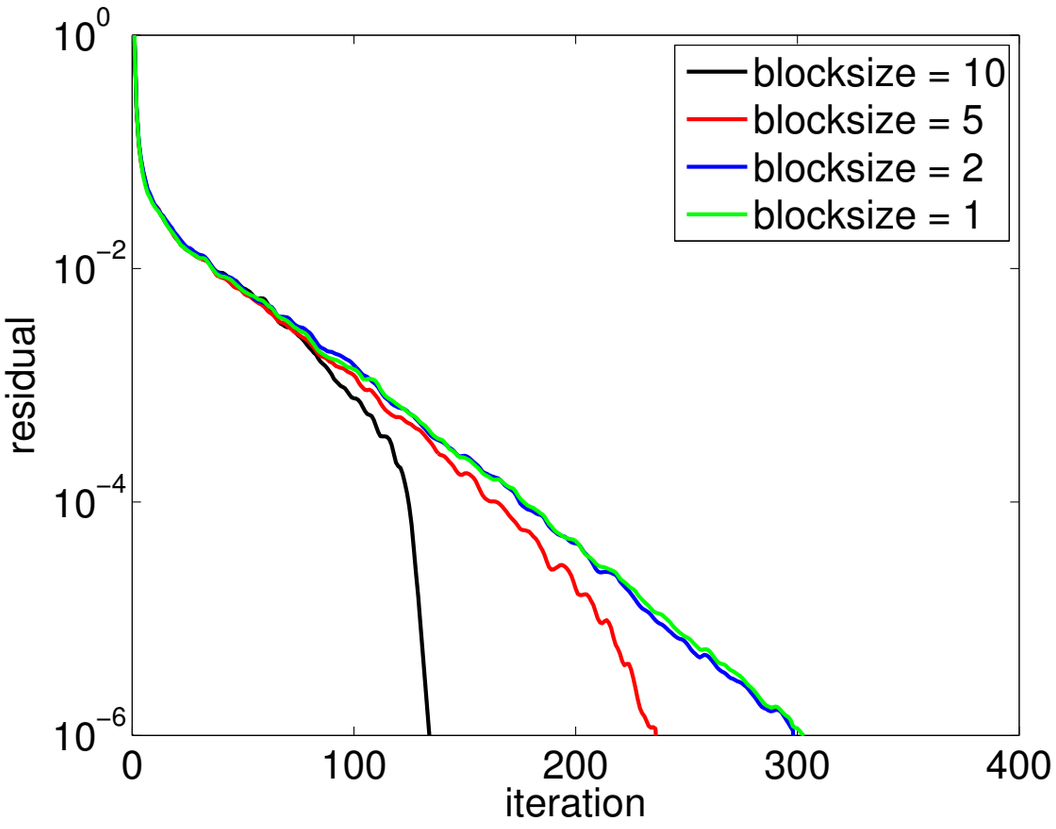
Experiments were done with Matlab 2012 on a Dual-Core SuperMicro system with 2 Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz and 128 GB RAM

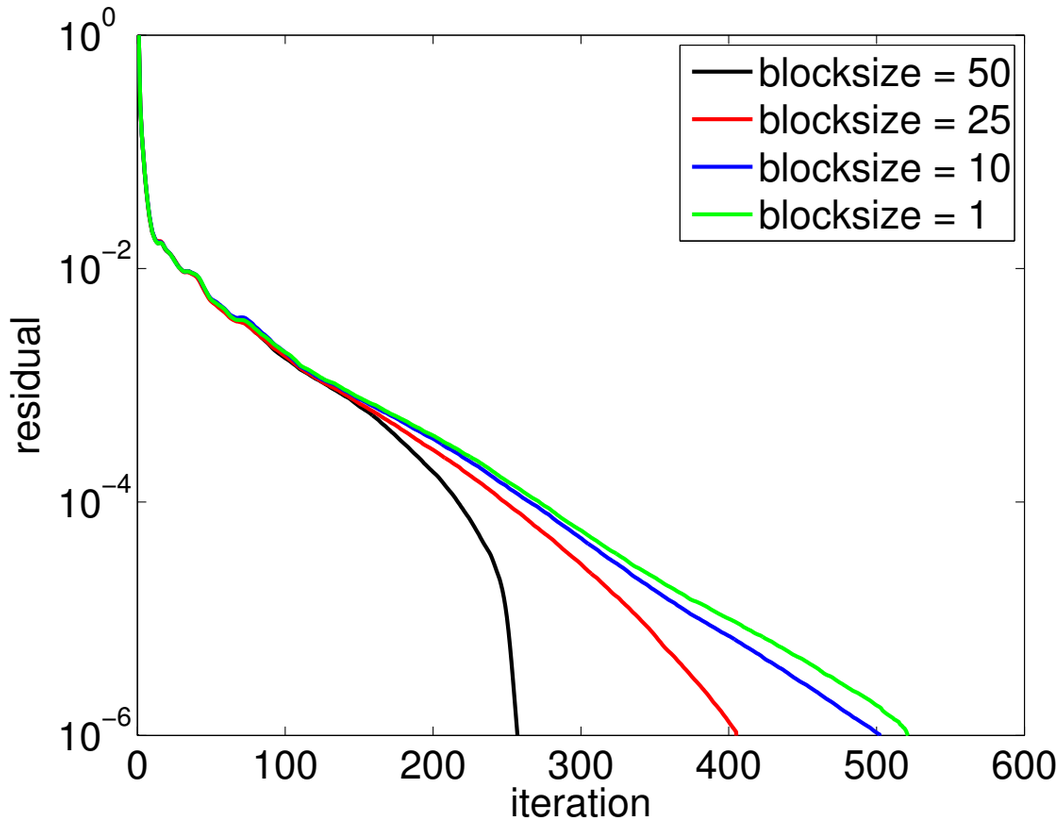
Block CG

0.5,1 Hz

sources selected *randomly*

multiple right-hand-sides





Block CG

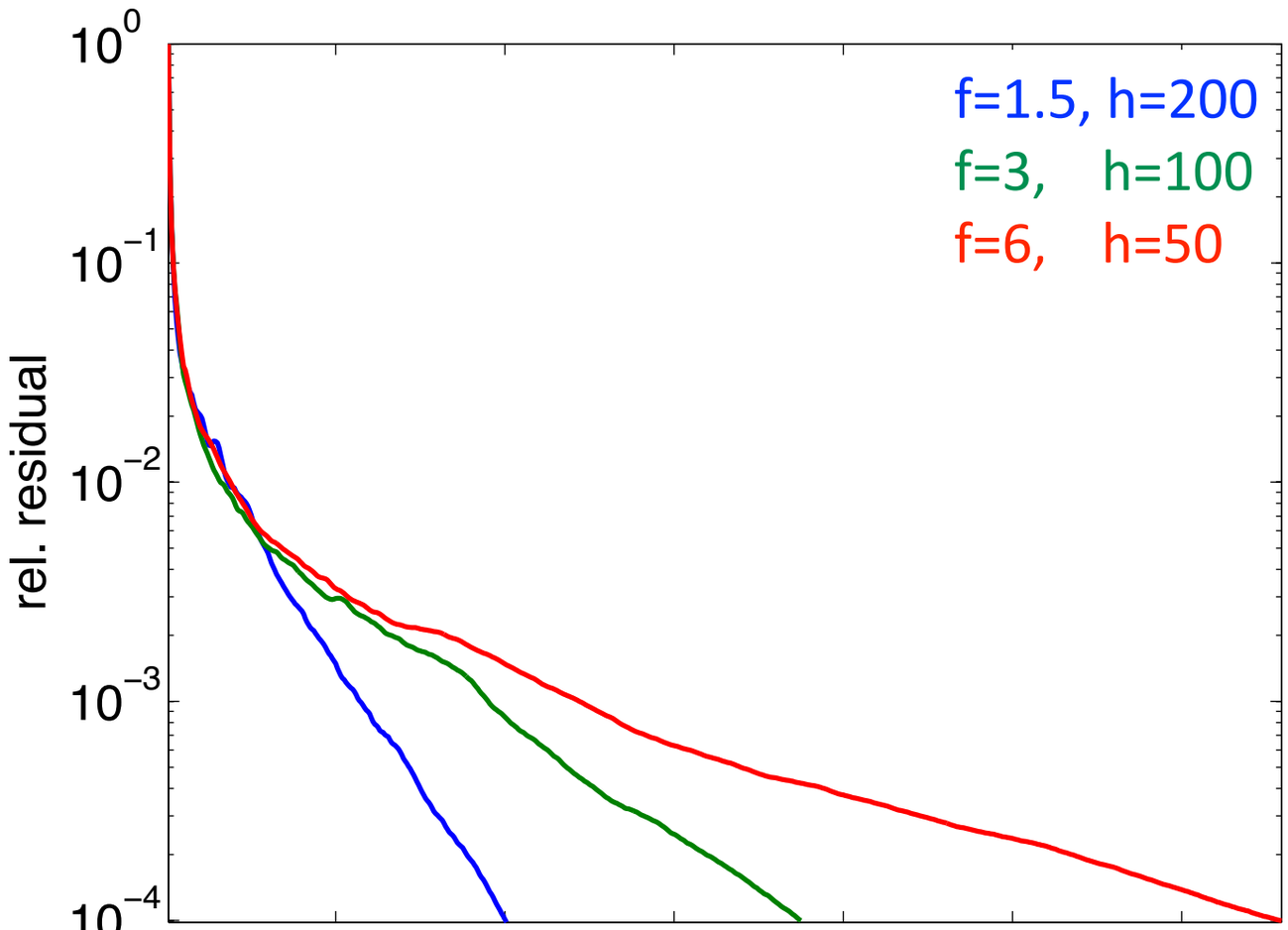
0.5,1 Hz

sources selected *randomly*

f [Hz]	N	blocksize	iter	time [s]
0.5	31212	10	135	21.5
0.5	31212	5	237	37.4
0.5	31212	2	301	62.3
0.5	31212	1	305	47.6
1.0	244824	50	258.0	1622.5
1.0	244824	25	408	2550.0
1.0	244824	10	503	3226.1
1.0	244824	1	531	4233.7

Example

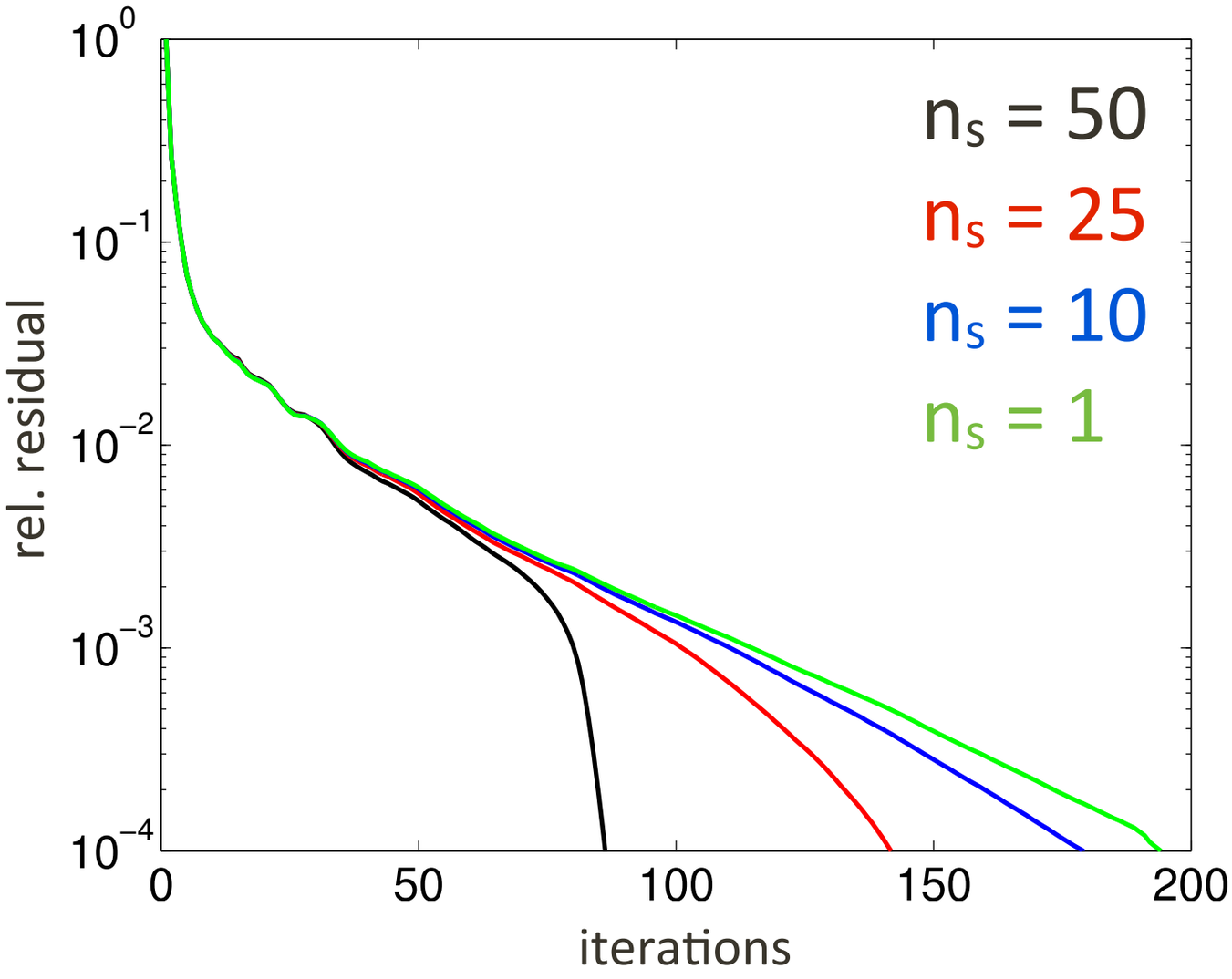
one source



on 1 processor

Example

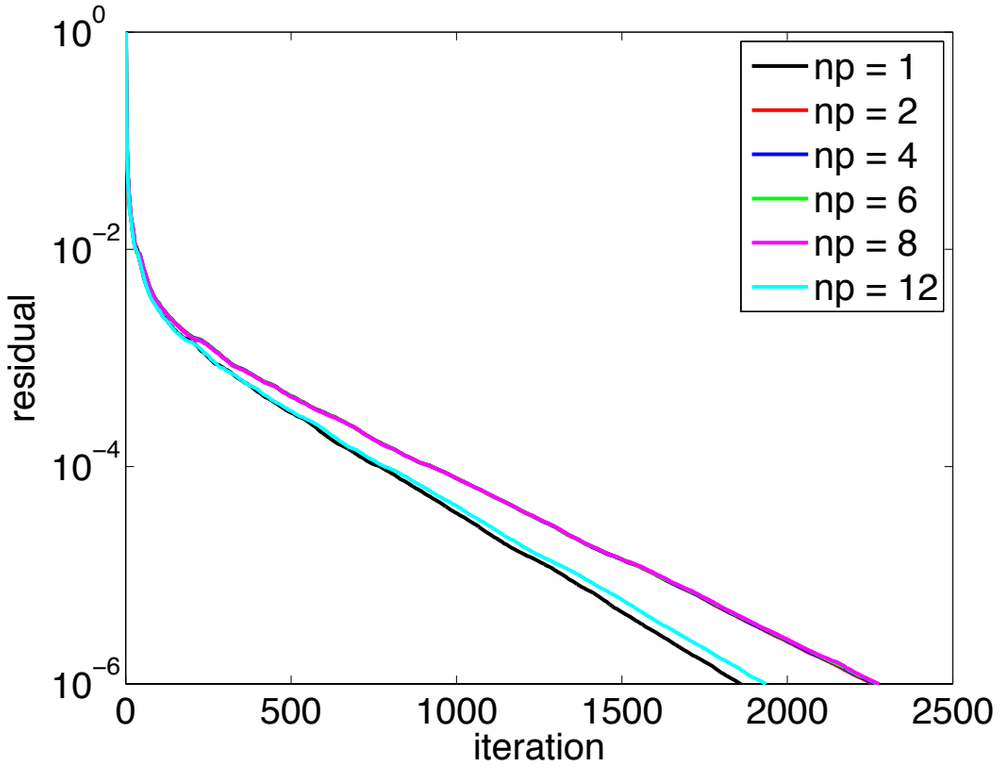
multiple sources



CARP-CG

parallel over *blocks* of *rows*
averaging guarantees *convergence*

multiple cores



np	iter	time [s]	efficiency
1.0	1859	13834.8	1.0
2.0	2256	9141.7	0.9
4.0	2265	4649.0	0.9
6.0	2271	3326.6	0.8
8.0	2276	2598.9	0.8
12.0	1934	1633.0	0.7

Gordon & Gordon ’11-’12

Versatile modelling
w/ *approximate* PDE solves

Framework:

- *smooth* errors as a function of # of *iterations*
  - allows for *dynamic* precision *control*
- multiple right-hand-sides & easily parallelizable
  - scale to 3D FWI

Challenge:

- *translate* into *practice* when *accuracy* & *convergence* unknown

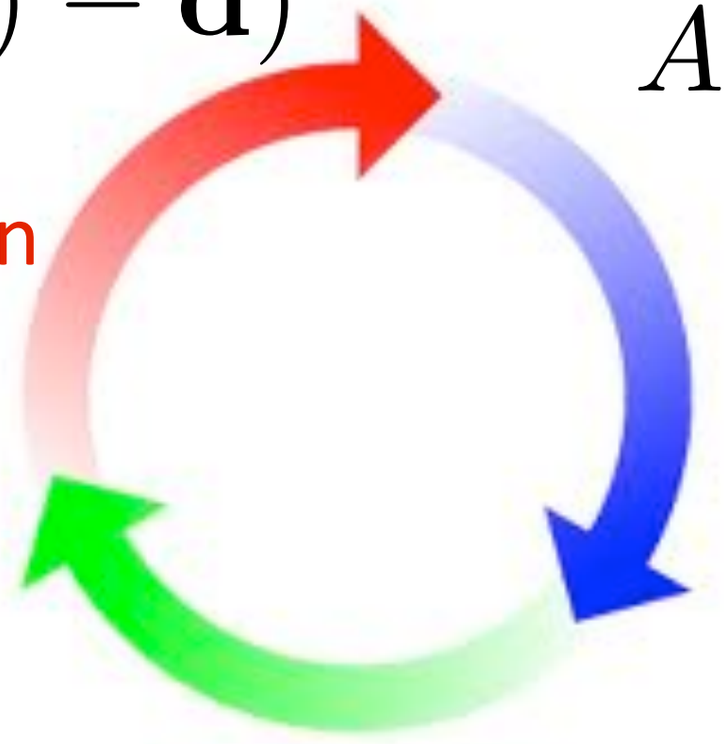
FWI

w/ controlled *sloppiness*

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

robust

formulation



$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

versatile

modelling

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

fast optimization strategies

computational framework

Frugal misfit
w/ approximate PDE solves

Heuristic based on *behavior* of the *misfit* as a function of ϵ

$$\phi_i(\mathbf{m}, \epsilon) = \rho\Big(P_i \mathbf{u}_i(\epsilon) - \mathbf{d}_i\Big),$$

by solving PDEs to *tolerance* ϵ.

Ideally find ϵ by guaranteeing

true solution

$$|\phi_i(\mathbf{m}, \epsilon) - \phi_i(\mathbf{m}, 0)| \leq \eta \phi_i(\mathbf{m}, 0)$$

for some fraction η.

[van Leeuwen & FJH '13]

Frugal misfit

w/ approximate PDE solves

Instead find k such that

$$|\phi_i(\mathbf{m}, \alpha^k \epsilon) - \phi_i(\mathbf{m}, \alpha^{k+1} \epsilon)| \leq \eta \phi_i(\mathbf{m}, \alpha^{k+1} \epsilon) \quad 0 < \alpha < 1$$

by increasing the precision, i.e., $\epsilon \mapsto \alpha\epsilon$, if this *inequality* does ***not*** hold.

Frugal misfit

Algorithm 1 $\{f, \mathbf{g}\} = \mathsf{misfit}(\mathbf{m}, \mathcal{I}, \eta)$

$\epsilon = 10^{-2}$, $\alpha = 0.5$// Initialization
for $i \in \mathcal{I}$ **do**
  for $k = 0 \rightarrow 10$ **do**
    solve $A(\mathbf{m})\mathbf{u} = \mathbf{s}_i$ up to ϵ // solve forward equation
    $r_k = \rho(P_i\mathbf{u} - \mathbf{d}_i)$ // compute residual
    if $|r_k - r_{k-1}| \leq \eta r_k$ **then**
      break
    else
      $\epsilon = \alpha\epsilon$
    end if
  end for
  solve $A(\mathbf{m})^*\mathbf{v} = {P_i}^*\nabla\rho(P_i\mathbf{u} - \mathbf{d}_i)$ up to ϵ
  $f = f + |\mathcal{I}|^{-1}\rho(P_i\mathbf{u} - \mathbf{d}_i)$ // misfit
  $\mathbf{g} = \mathbf{g} + |\mathcal{I}|^{-1}G(\mathbf{m}, \mathbf{u})^*\mathbf{v}$ // gradient
end for

Stochastic
Quasi-Newton

Final algorithm has the following key ingredients:

- draws *independent* random *subsets* for *each* misfit & gradient calculation
- decreases *fraction* $\eta \mapsto \eta/2$ when *linesearch* fails
- increases *sample* size when *average* objective does *not* decrease–i.e, if $(f_{k+1} + f'_{k+1}) \geq (f_k + f'_k)$
- Quasi-Newton Hessian w/ lBFGS & a single *extra* gradient calculation for the same sample

Stochastic
Quasi-Newton

Algorithm 1 Stochastic L-BFGS method

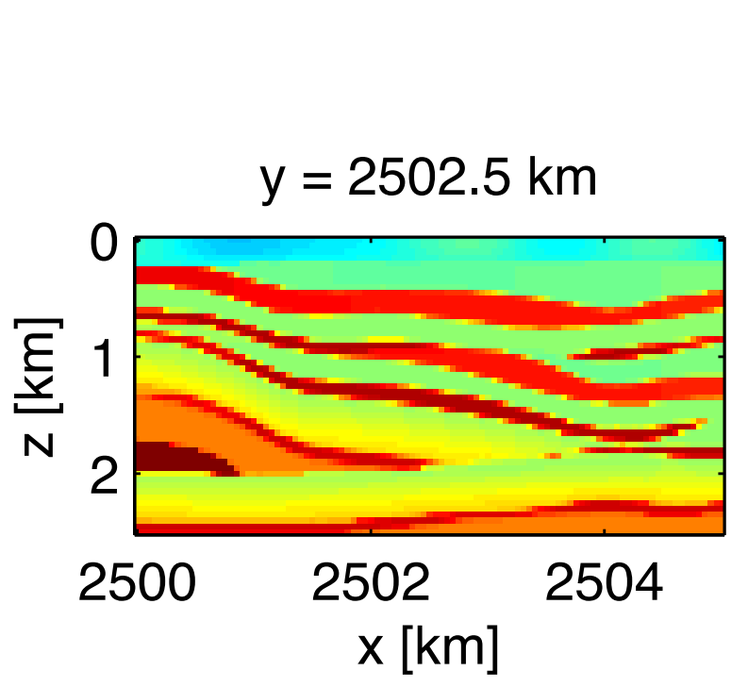
$\eta = 0.1$, $b = 1$, $\beta = 1$, $b_{\max} = M$ // Initialize
choose $\mathcal{I}_0 \subseteq \{1, 2, \ldots, M\}$ s.t. $|\mathcal{I}_0| = b$
$\{f_0, \mathbf{g}_0\} = \mathsf{misfit}(\mathbf{m}_0, \mathcal{I}_0, \eta)$ // *frugal* misfit & gradient at initial guess
while not converged **do**
  $\delta\mathbf{m}_k = \mathsf{lbfgs}(-\mathbf{g}_k, \{\mathbf{t}_l\}_{l=k-m}^{k}, \{\mathbf{y}_l\}_{l=k-m}^{k})$ // low-rank inverse Hessian
  $\{\mathbf{m}_{k+1}, f_{k+1}, \mathbf{g}_{k+1}\} = \mathsf{linesearch}(f_k, \mathbf{g}_k, \delta\mathbf{m}_k)$
  if linesearch successfull **then**
    $\mathbf{t}_{k+1} = \mathbf{m}_{k+1} - \mathbf{m}_k$, $\mathbf{y}_{k+1} = \mathbf{g}_{k+1} - \mathbf{g}_k$ // update L-BFGS vectors
    choose $\mathcal{I}_{k+1} \subseteq \{1, 2, \ldots, M\}$ s.t. $|\mathcal{I}_{k+1}| = b$ // draw new sample
    $\{f'_{k+1}, \mathbf{g}'_{k+1}\} = \mathsf{misfit}(\mathbf{m}_{k+1}, \mathcal{I}_{k+1}, \eta)$ // misfit & gradient new sample
    if $(f_{k+1} + f'_{k+1}) \geq (f_k + f'_k)$ **then**
      $b = \min(b + \beta, b_{\max})$ // increase batch
    end if
    $f_{k+1} = f'_{k+1}$, $\mathbf{g}_{k+1} = \mathbf{g}'_{k+1}$, $k = k + 1$ // Use new misfit & gradient
  else
    $\eta = \eta/2$ // narrow tolerenance
  end if
end while

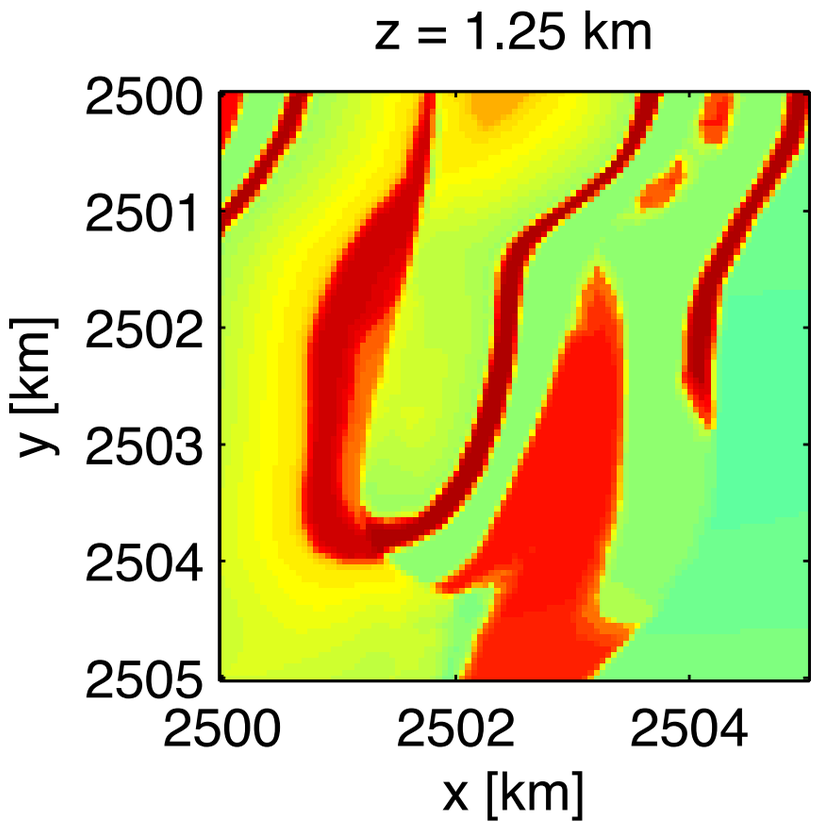
Overthrust model

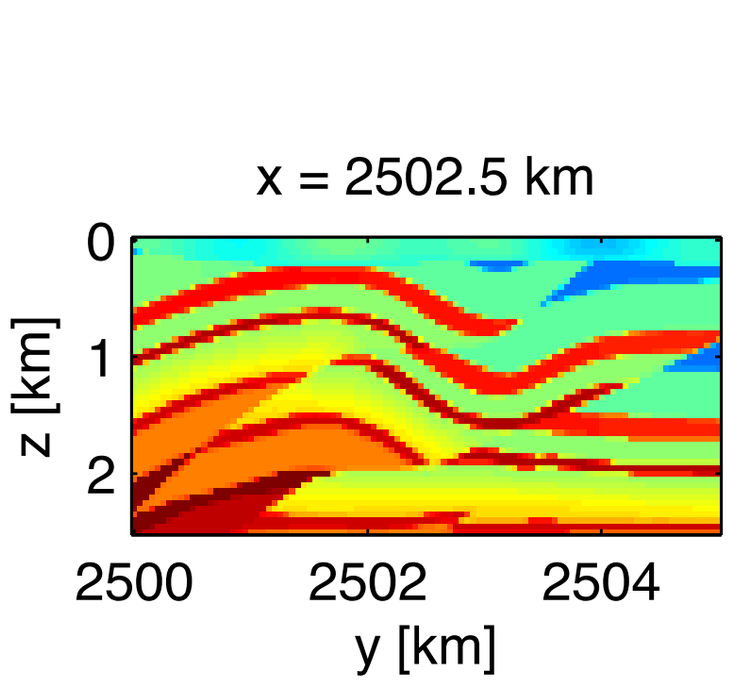
true model

5km X 5km X 2.5Km

121 sources & 2601 receivers

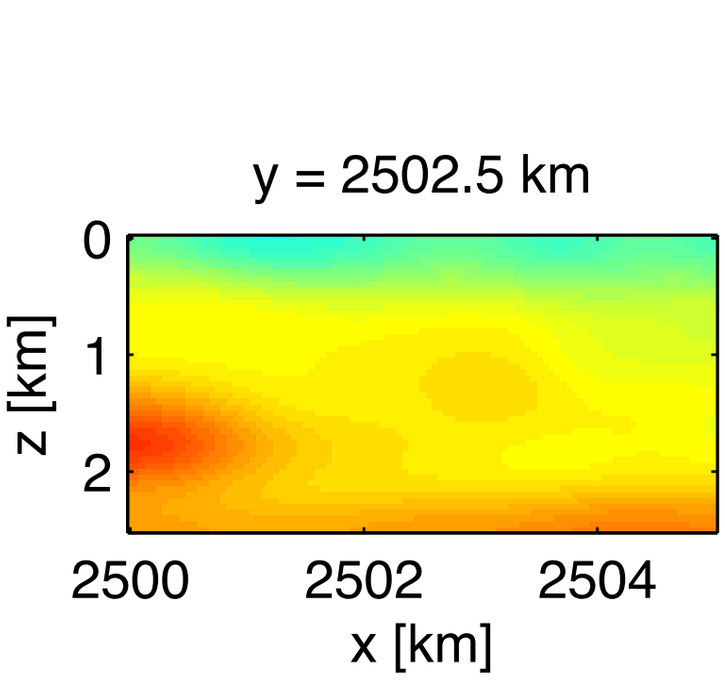


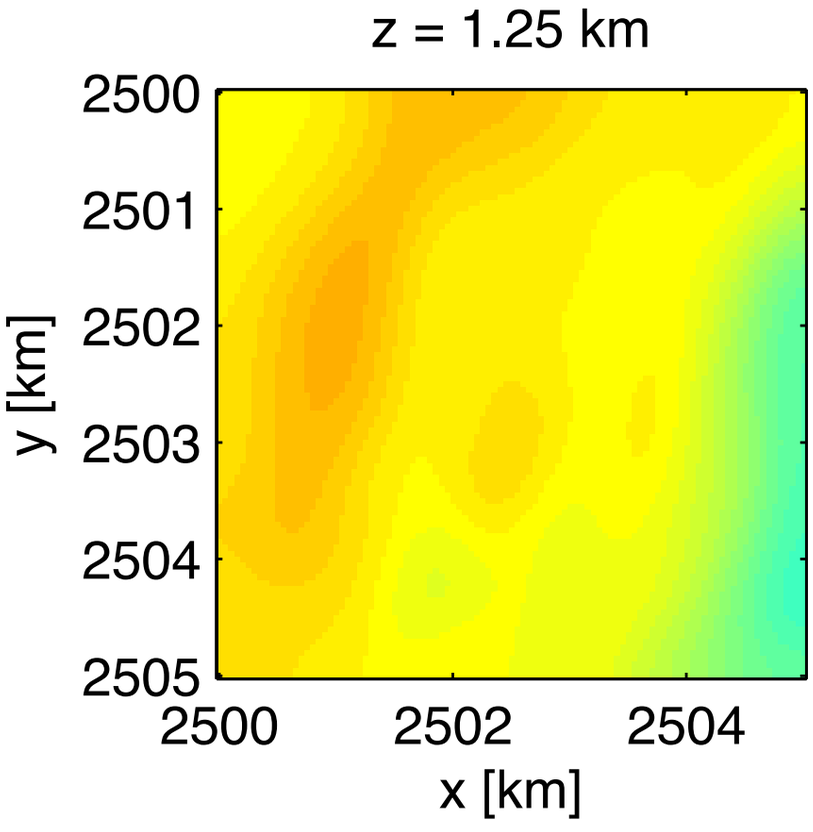


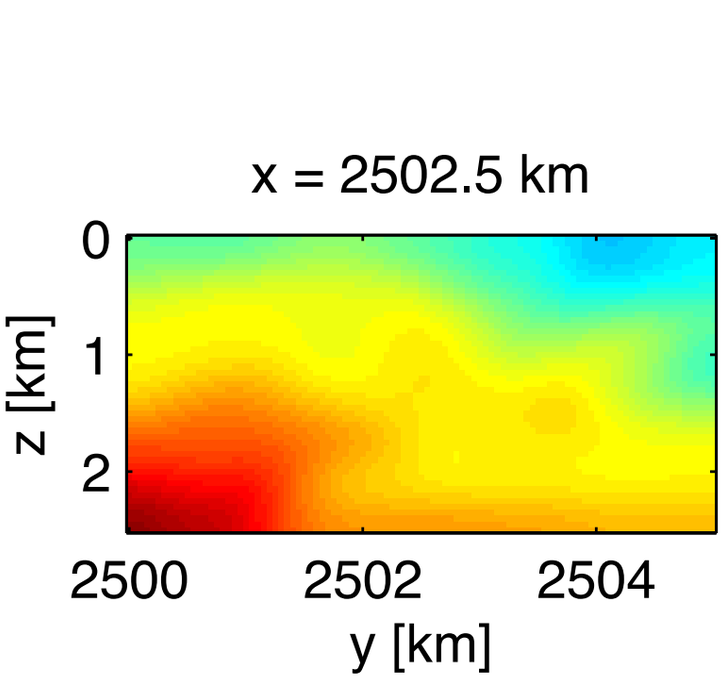


Overthrust model

initial model



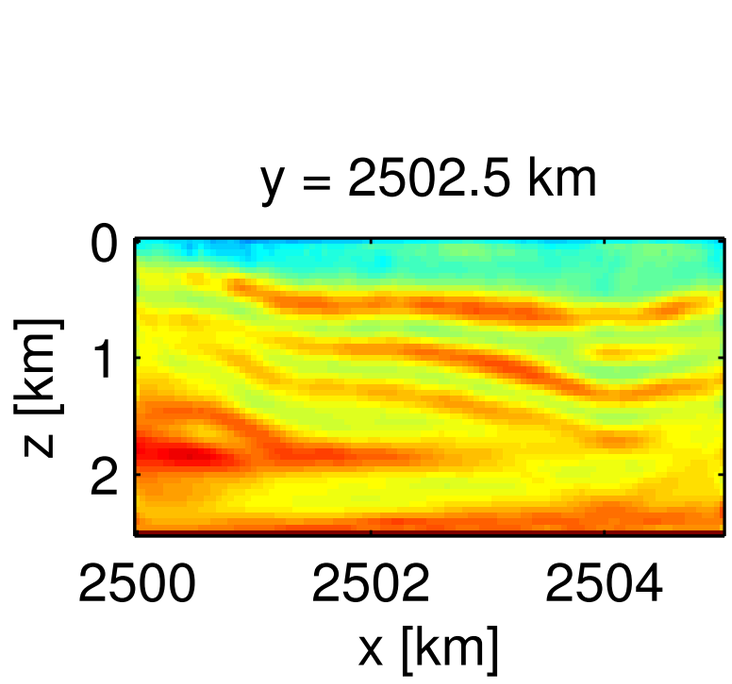


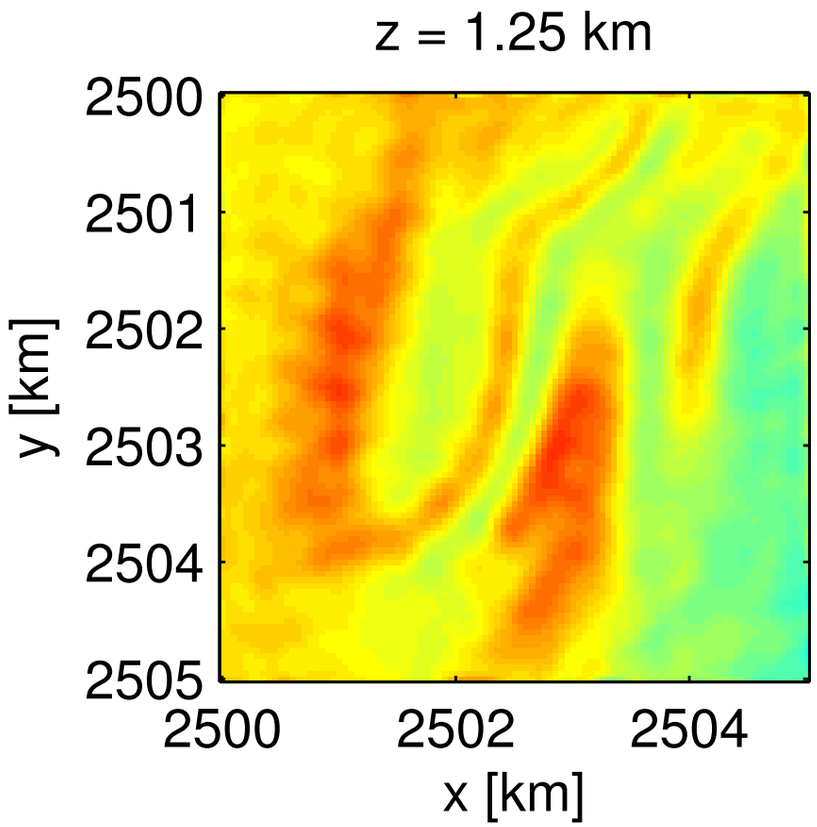


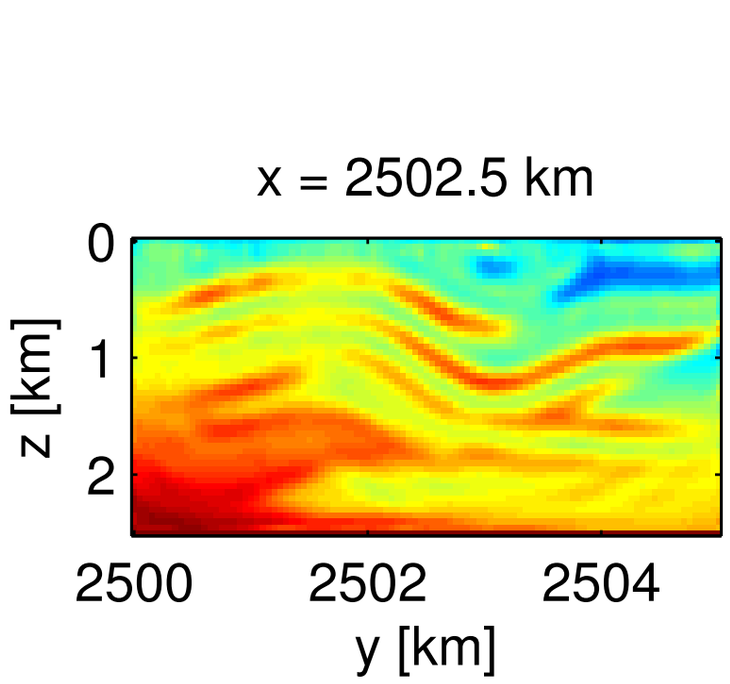
Overthrust model

recovered model *w/ b=1*

2 passes through *data* for each *(4,6,8) Hz*



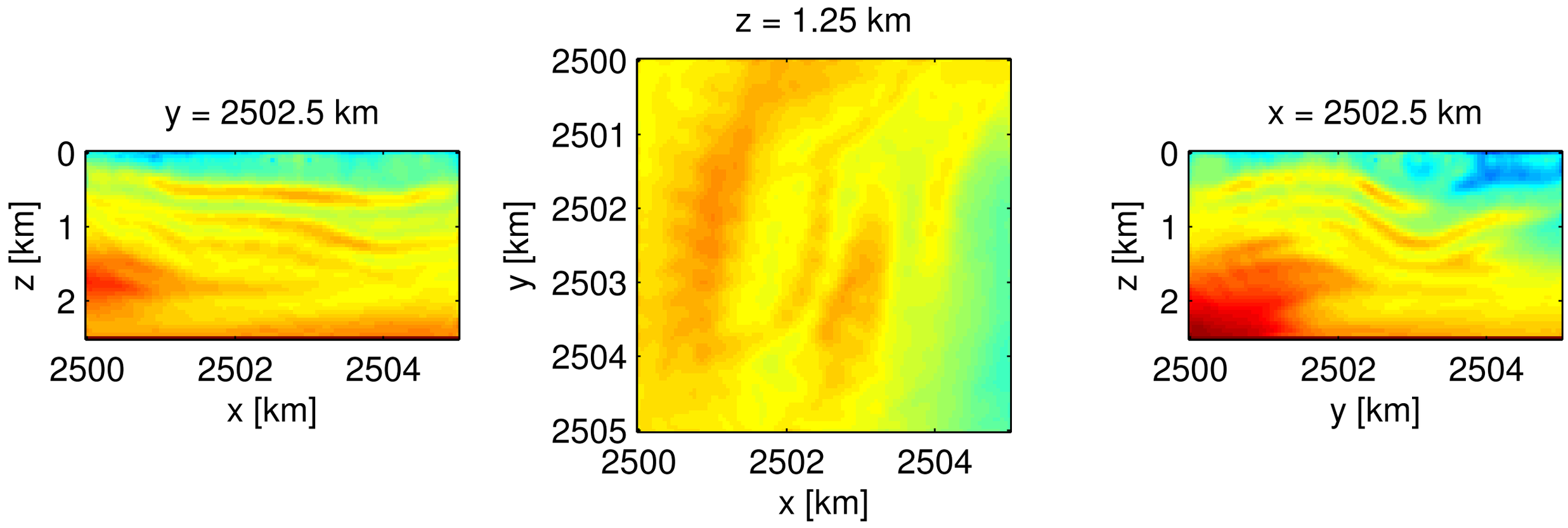




Overthrust model

recovered model *w/ b=121*

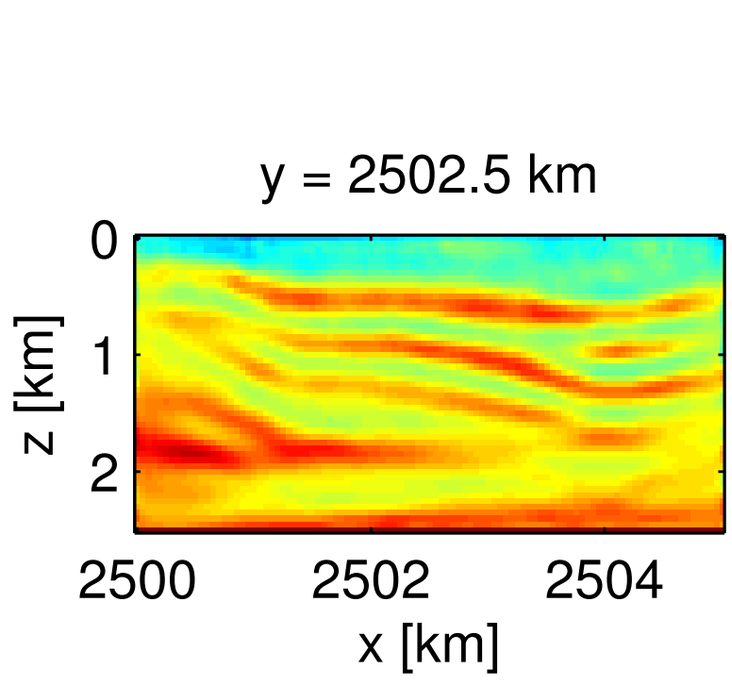
2 passes through *data* for each *(4,6,8) Hz*

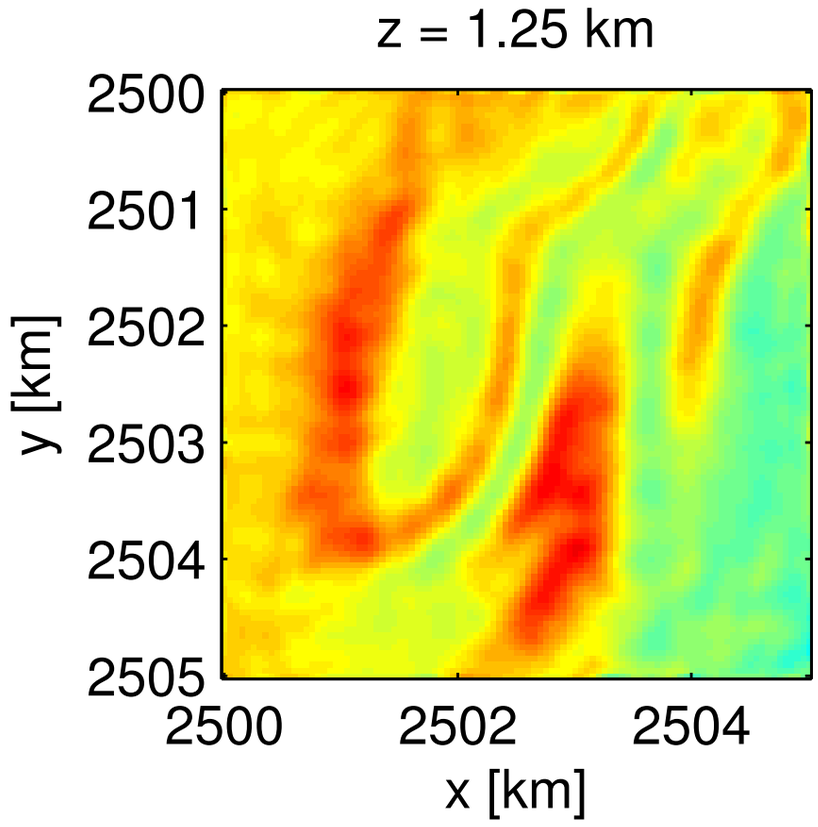


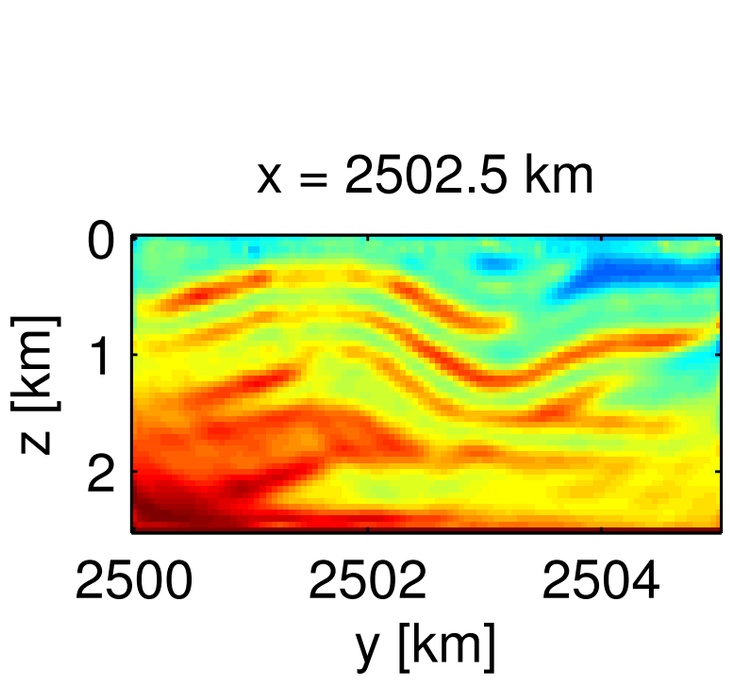
Overthrust model

growing sample size

2 passes through *data* for each *(4,6,8) Hz*



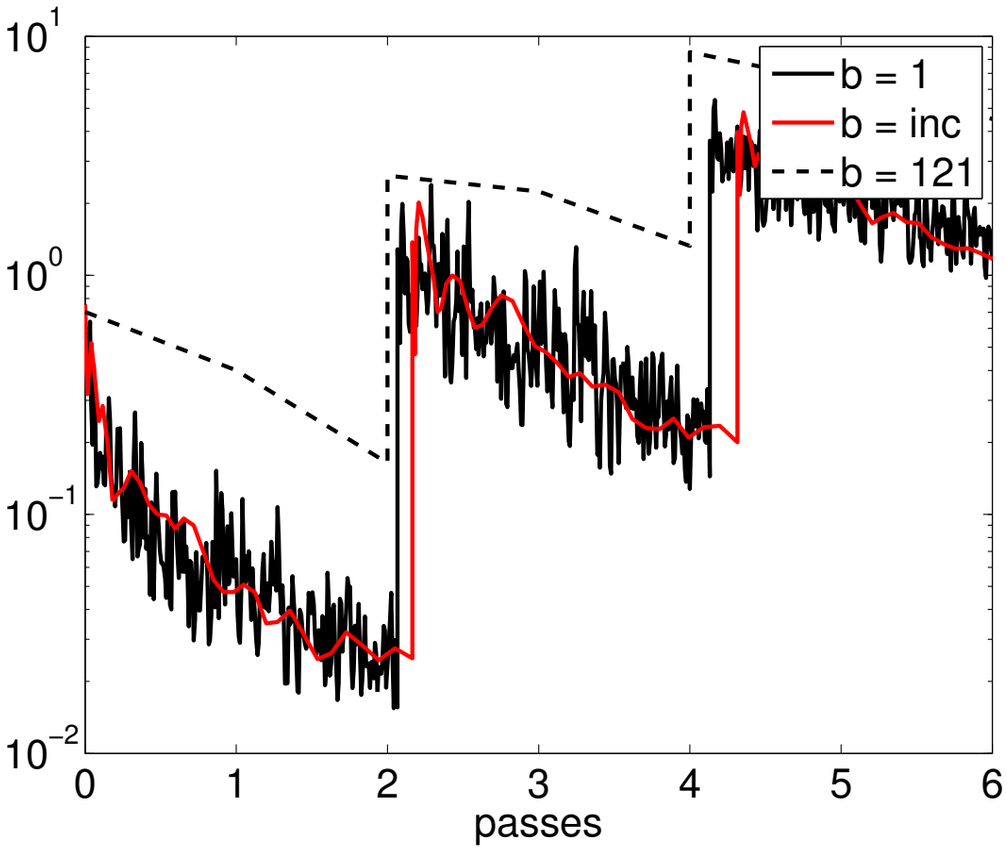




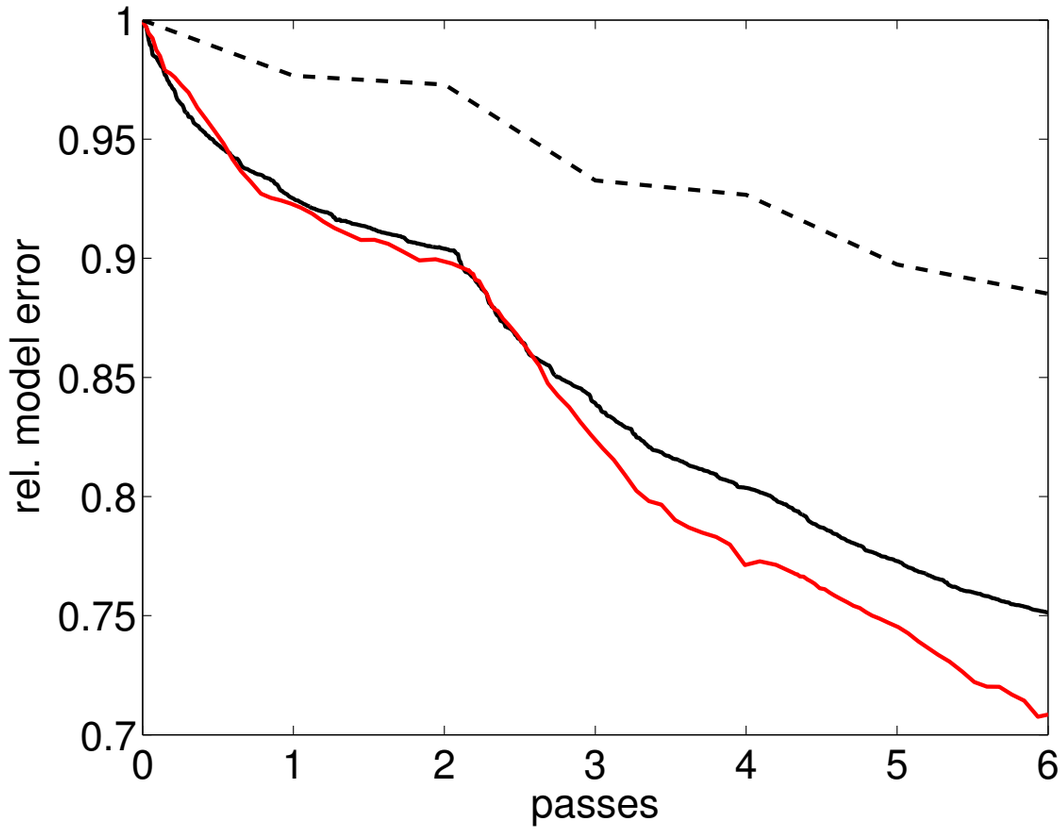
Performance

misfit & relative model error

misfit



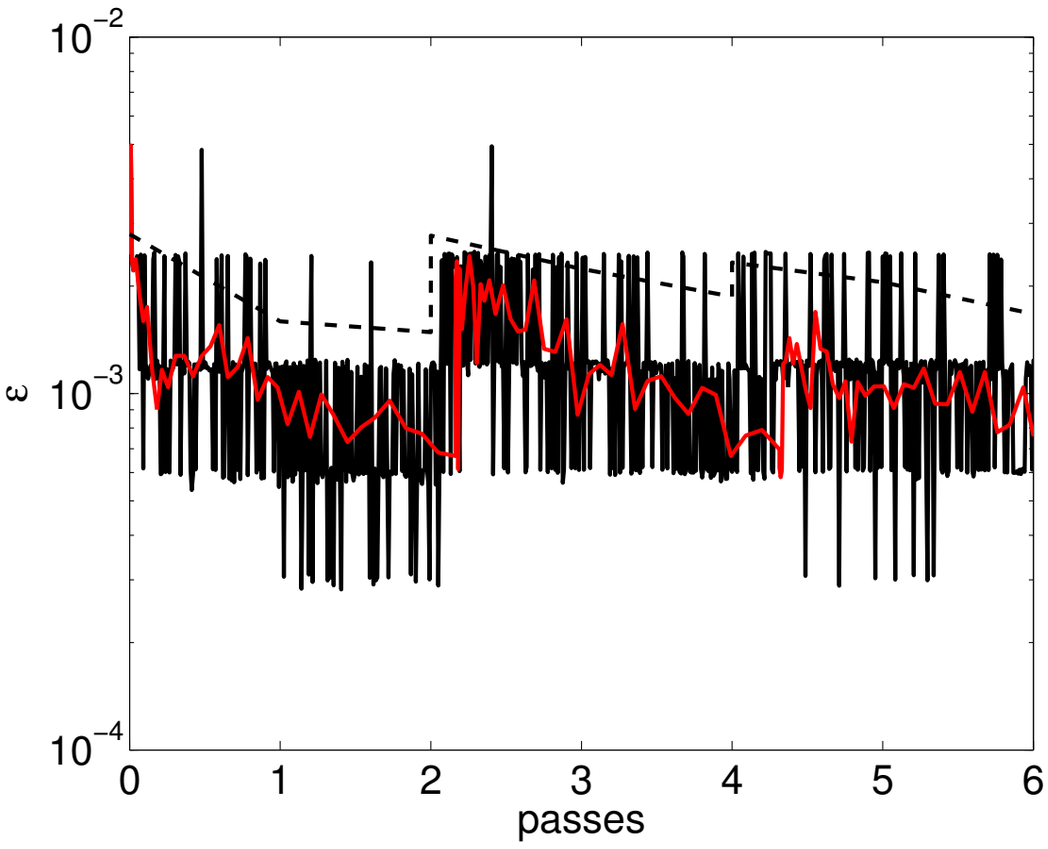
model error



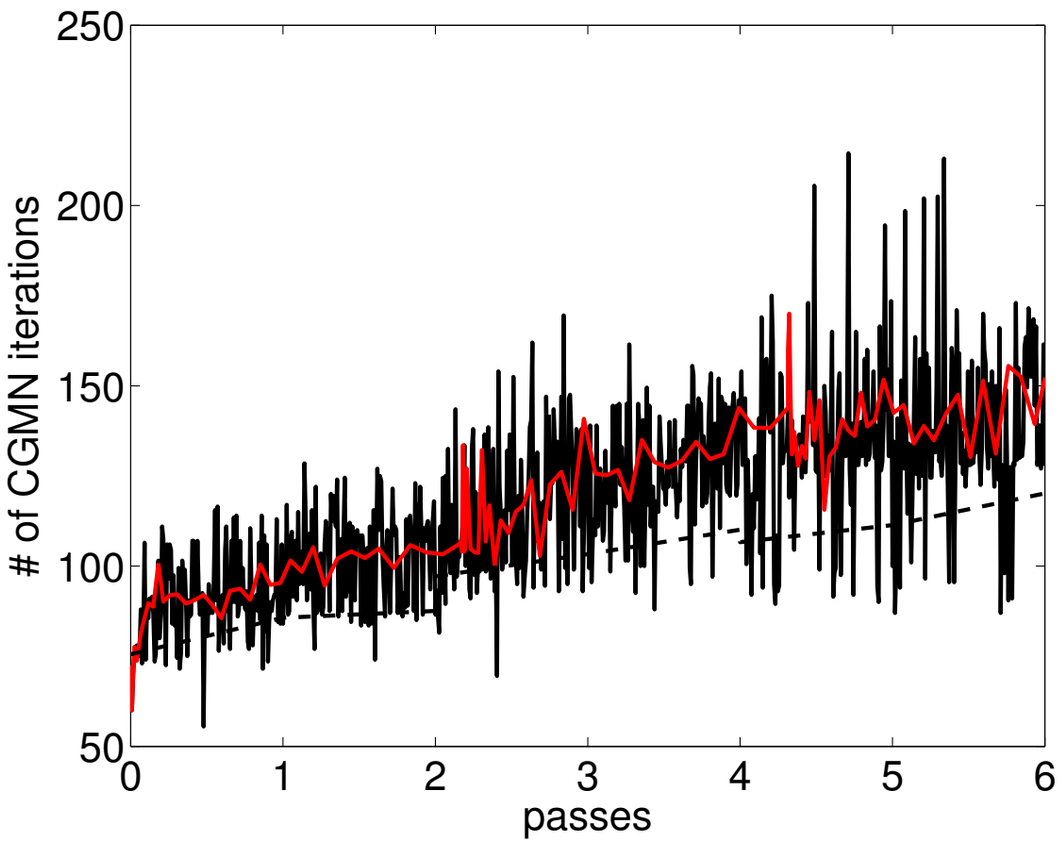
Performance

tolerance & # CARP-CG iterations

accuracy

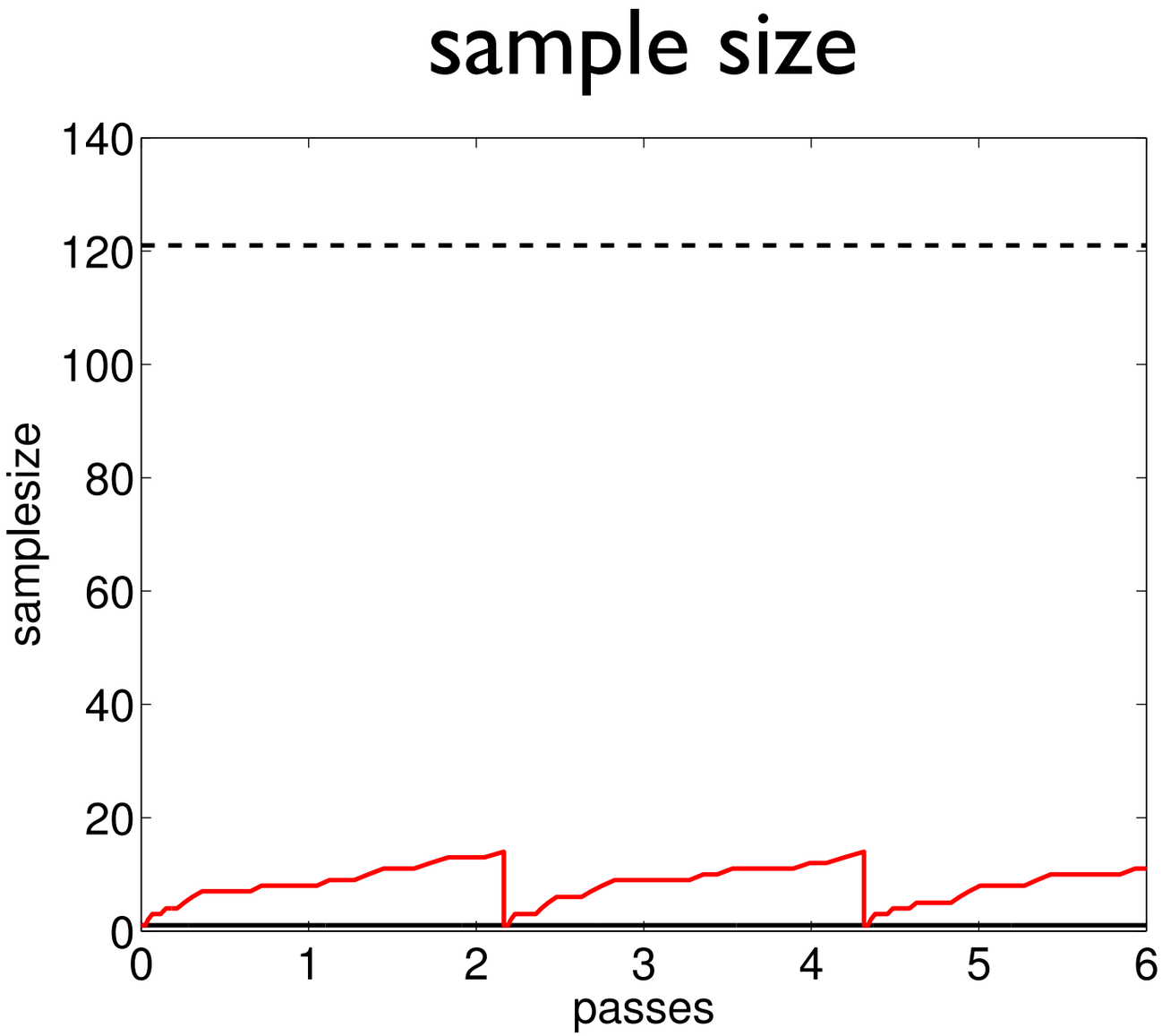


of iterations



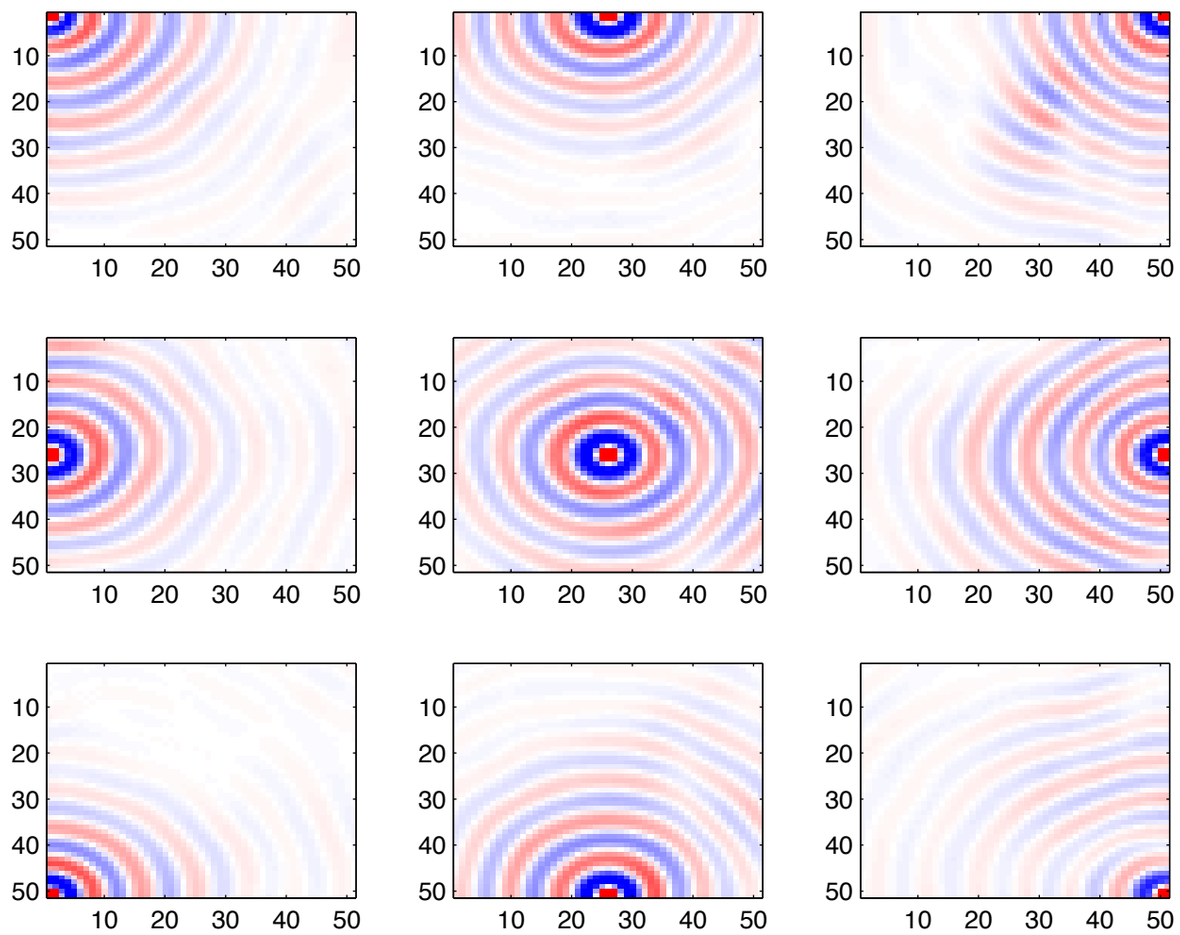
Performance

sample size



Performance

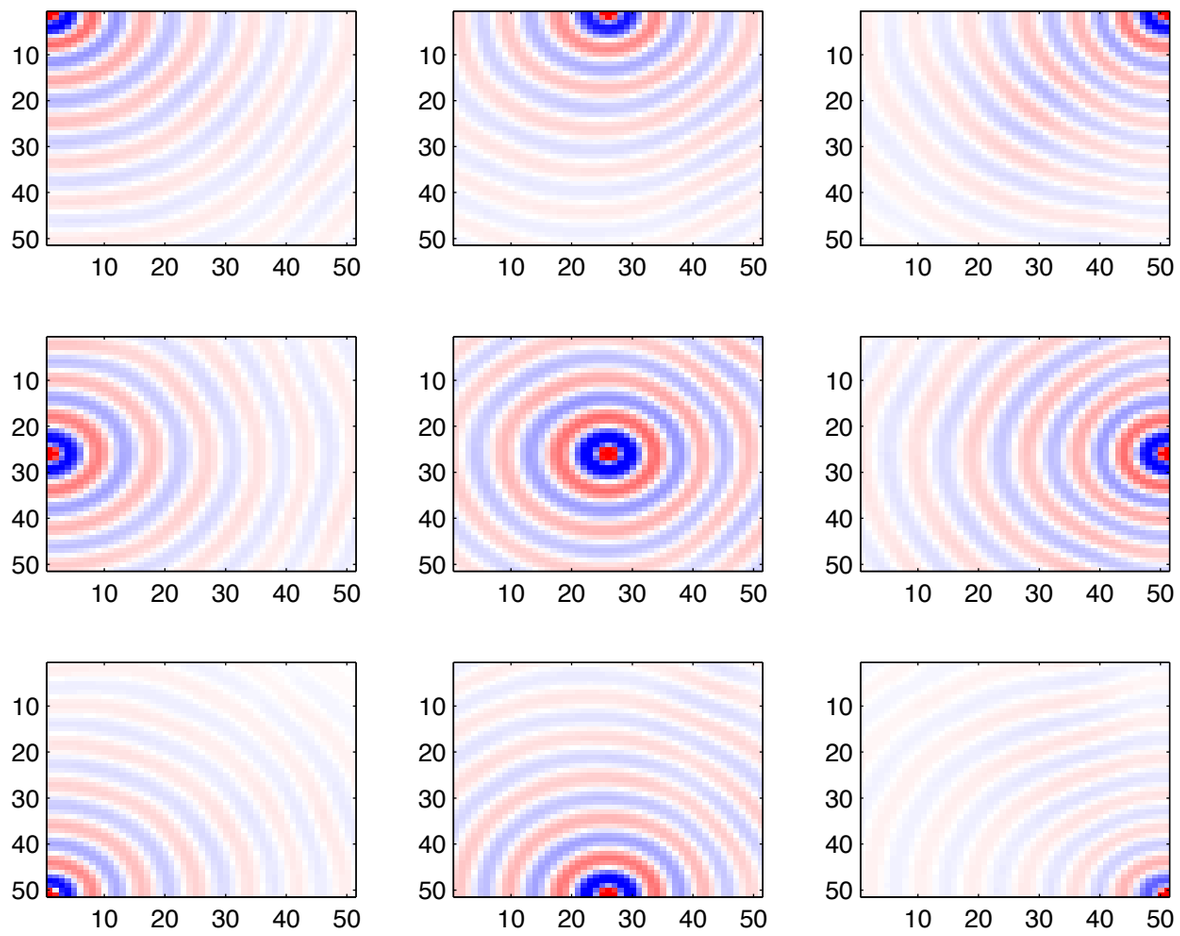
input data @ 4Hz



4 hours

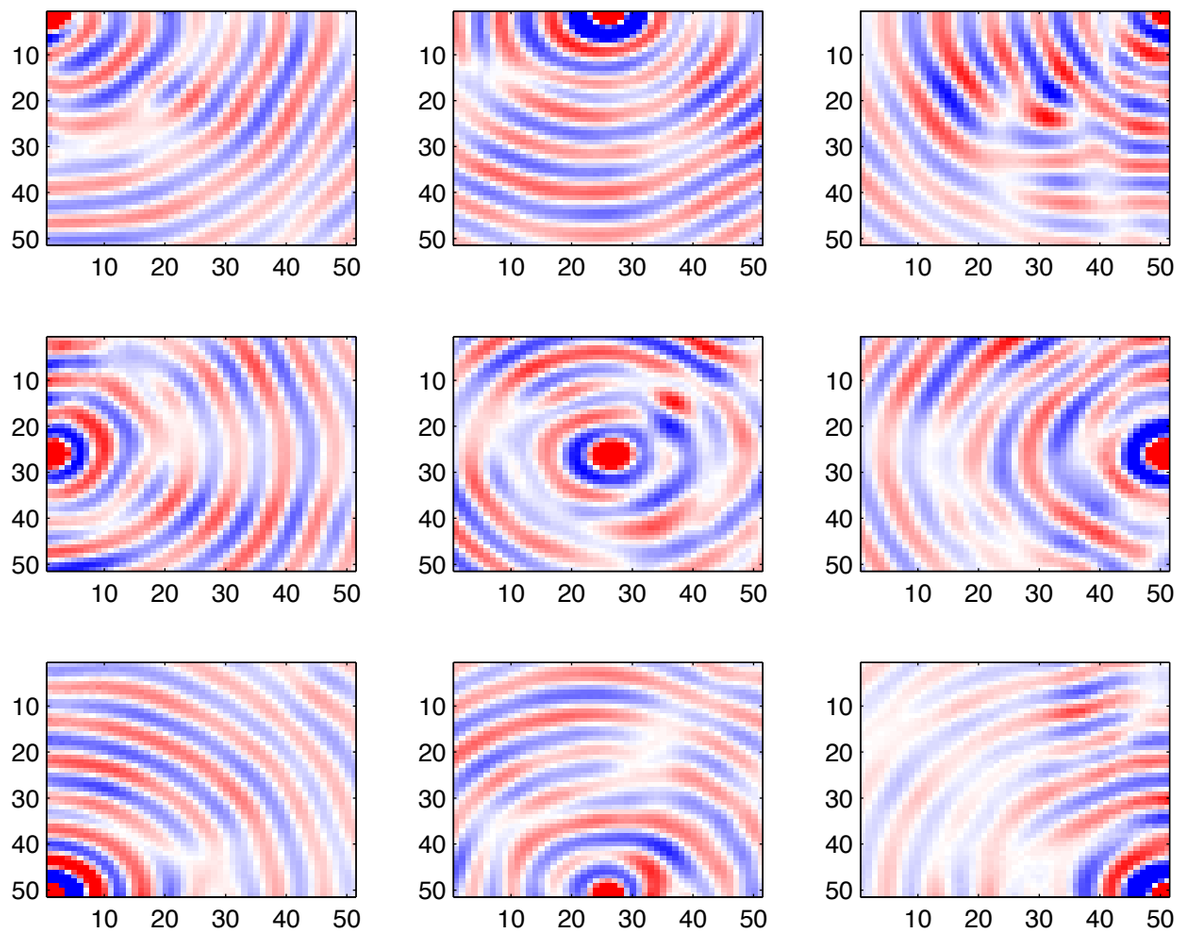
Performance

initial data @ 4Hz



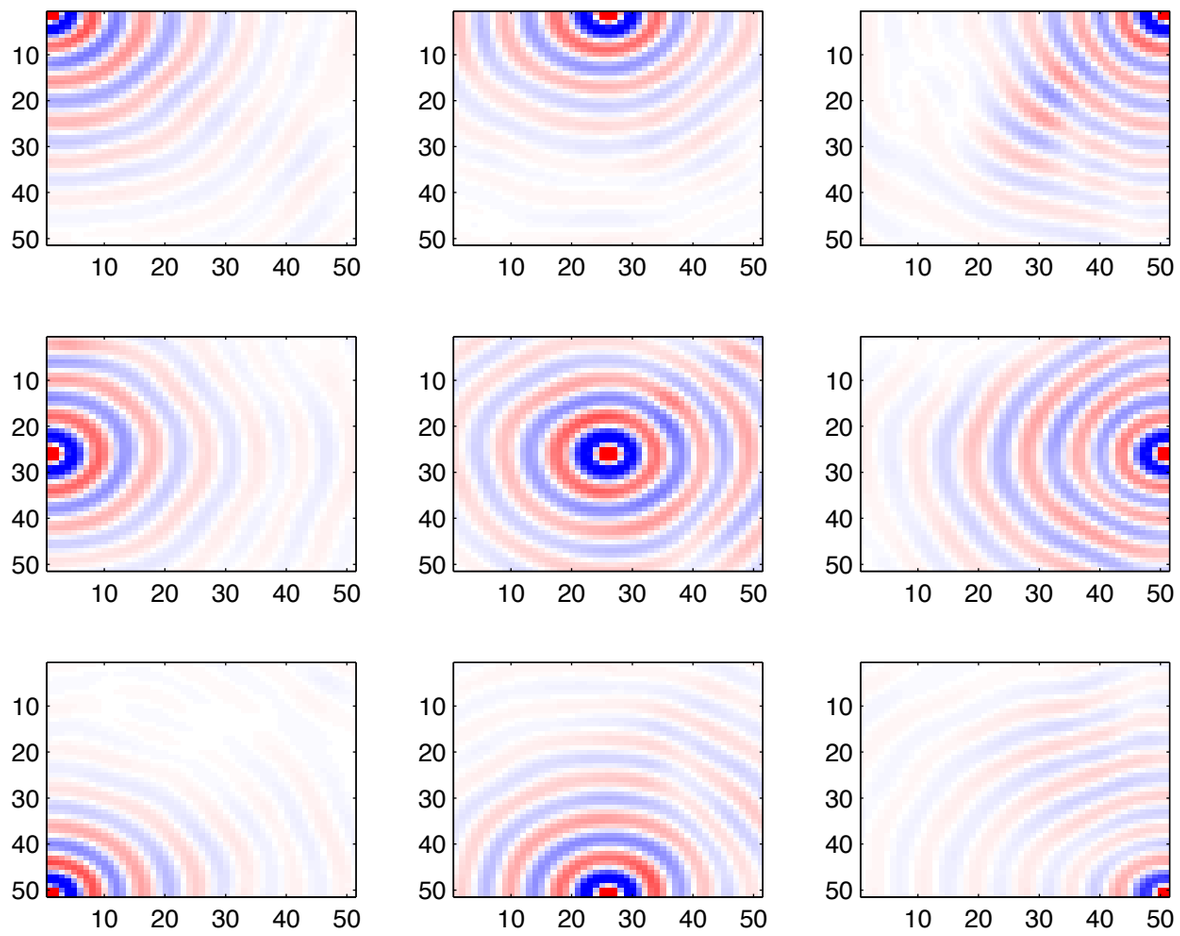
Performance

initial residual @ 4Hz



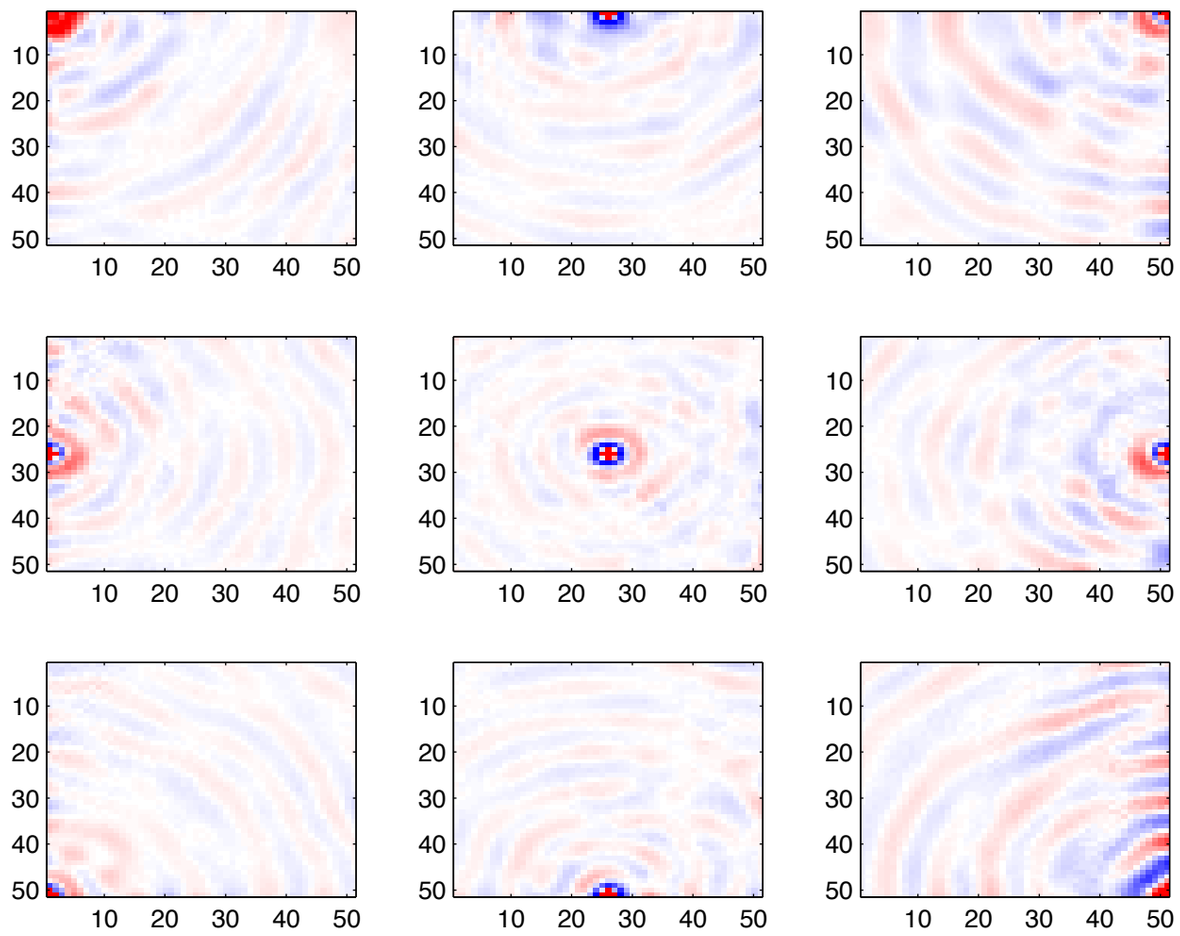
Performance

final data @ 4Hz



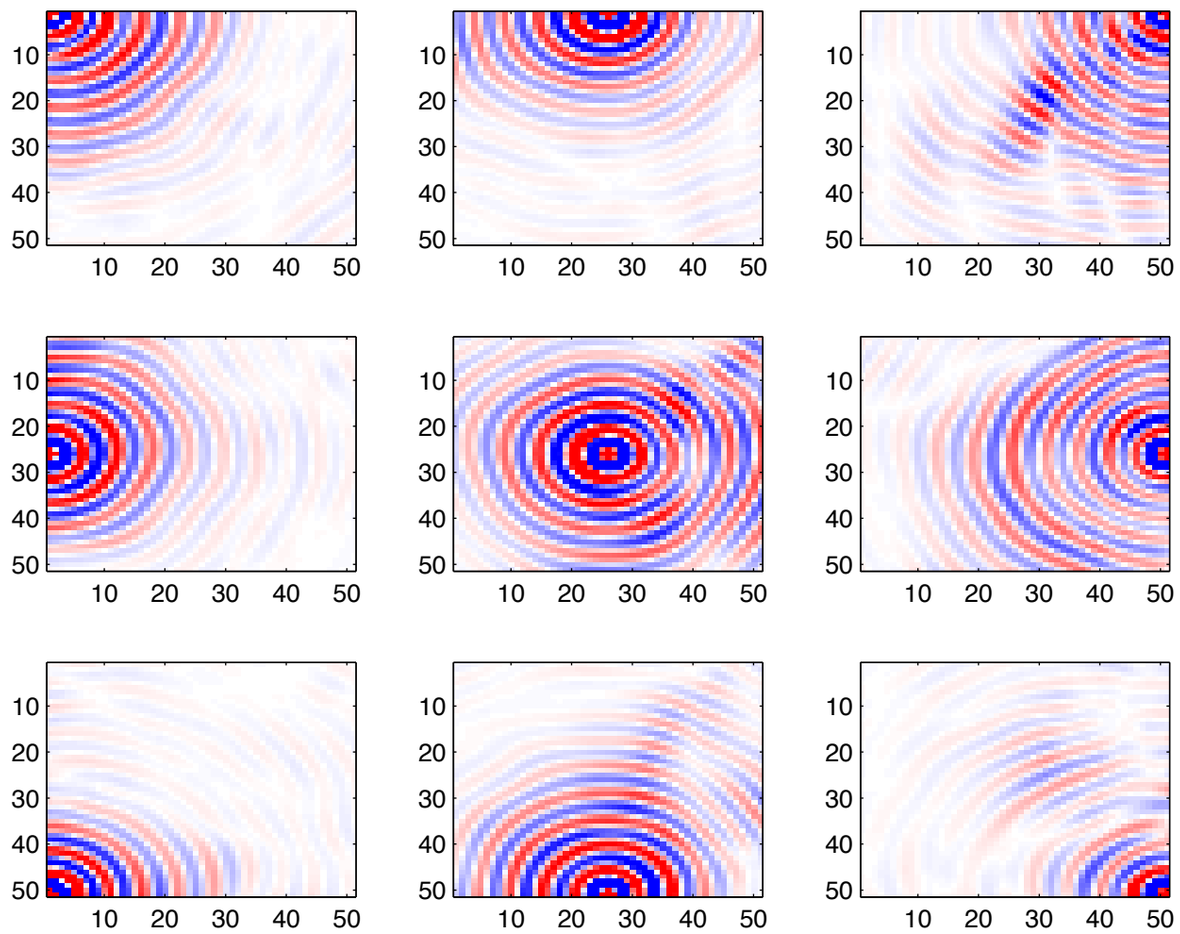
Performance

final residual @ 4Hz



Performance

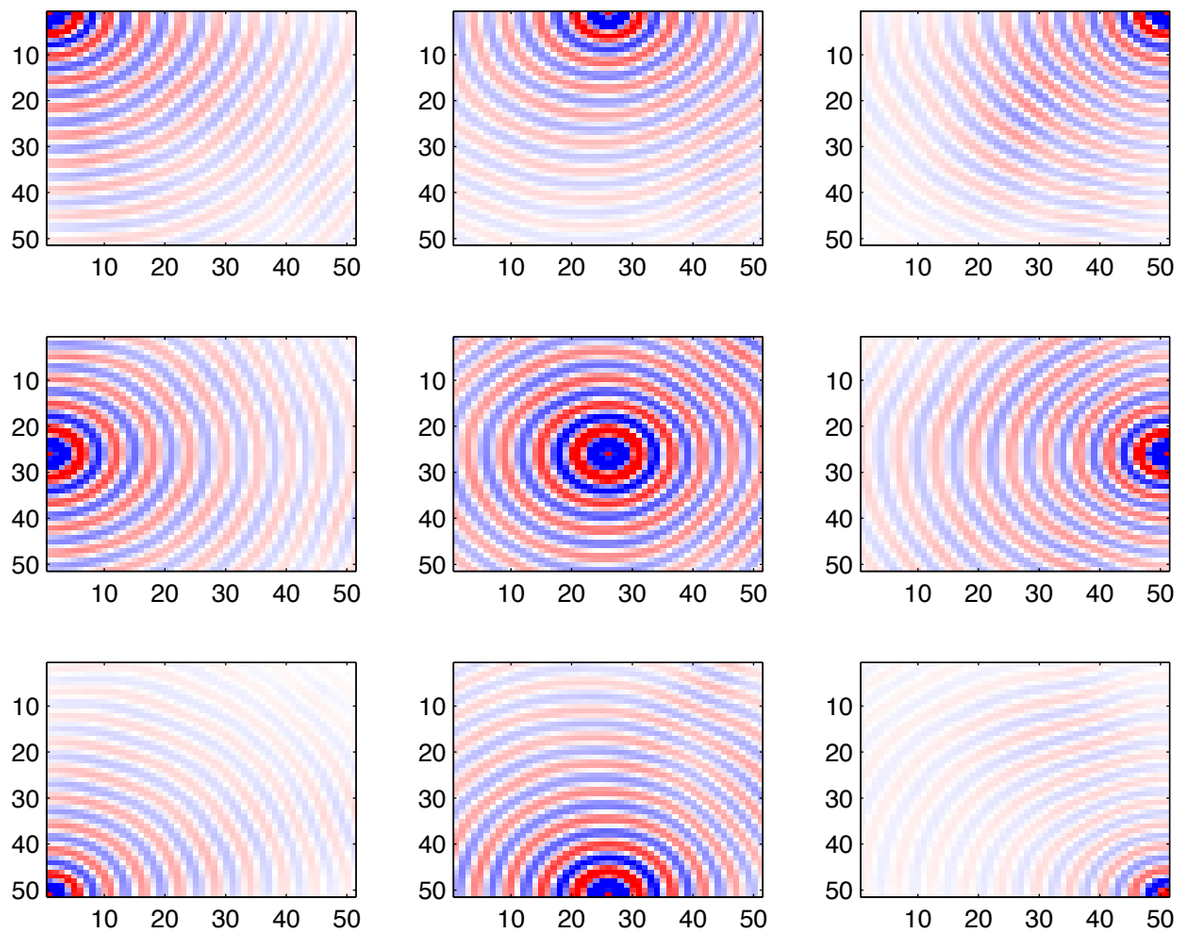
input data @ 6Hz



15 hours

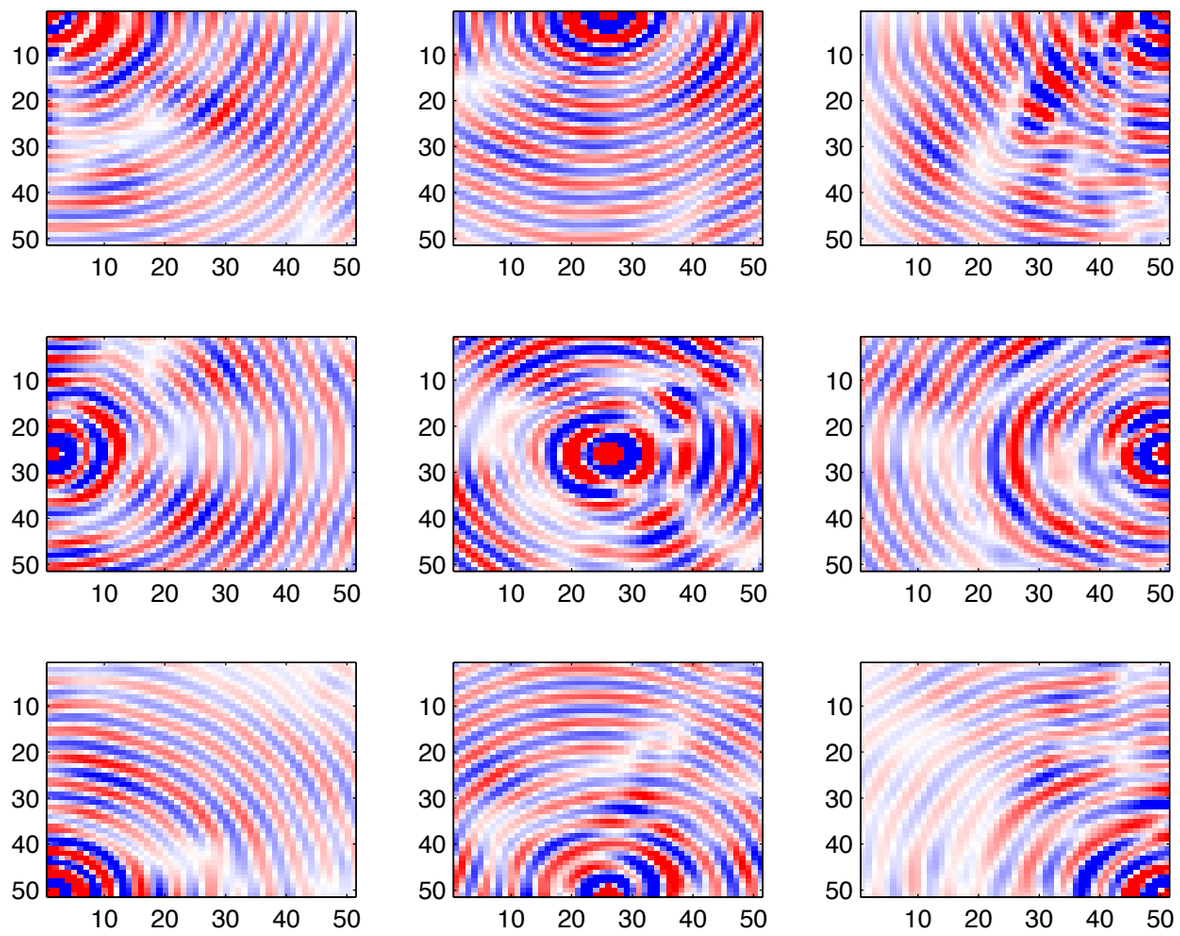
Performance

initial data @ 6Hz



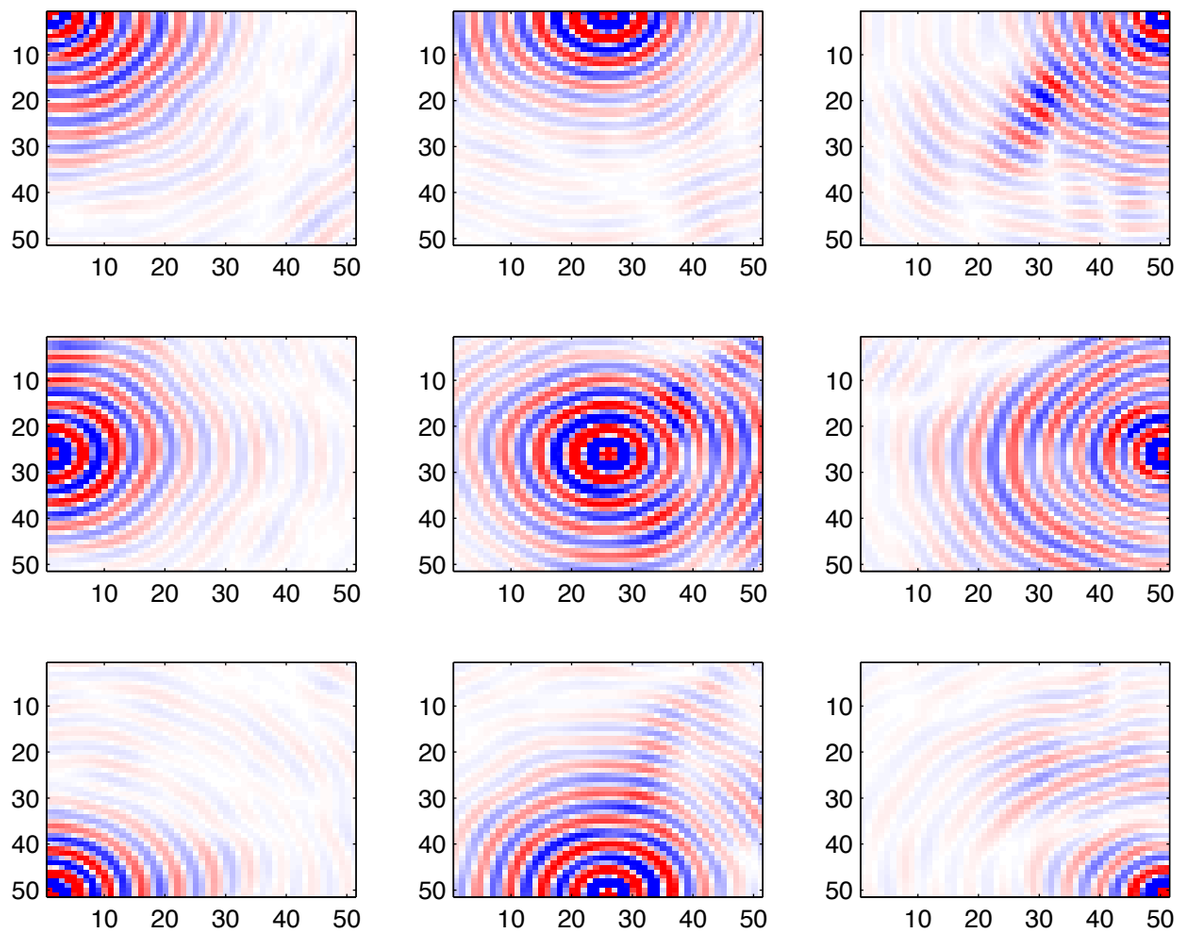
Performance

initial residual @ 6Hz



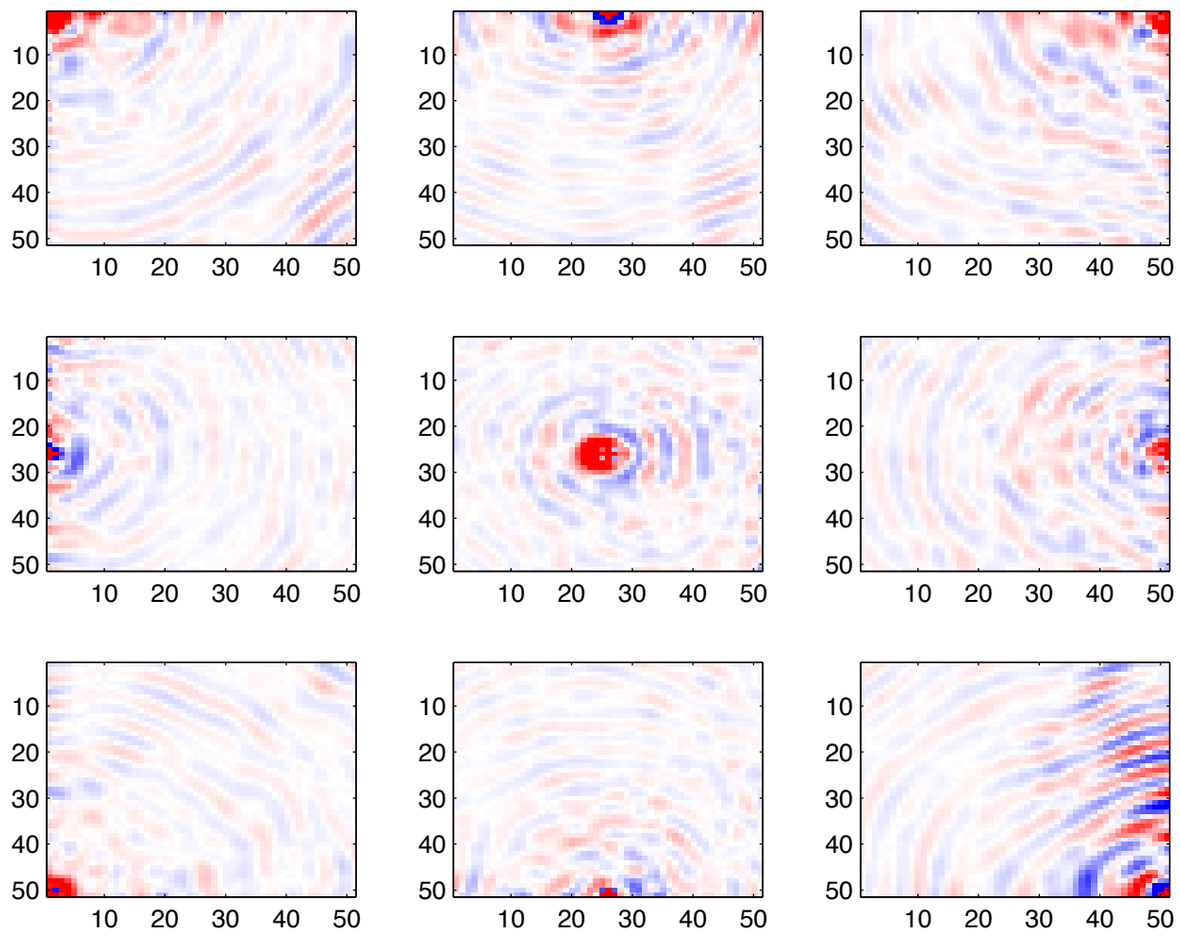
Performance

final data @ 6Hz



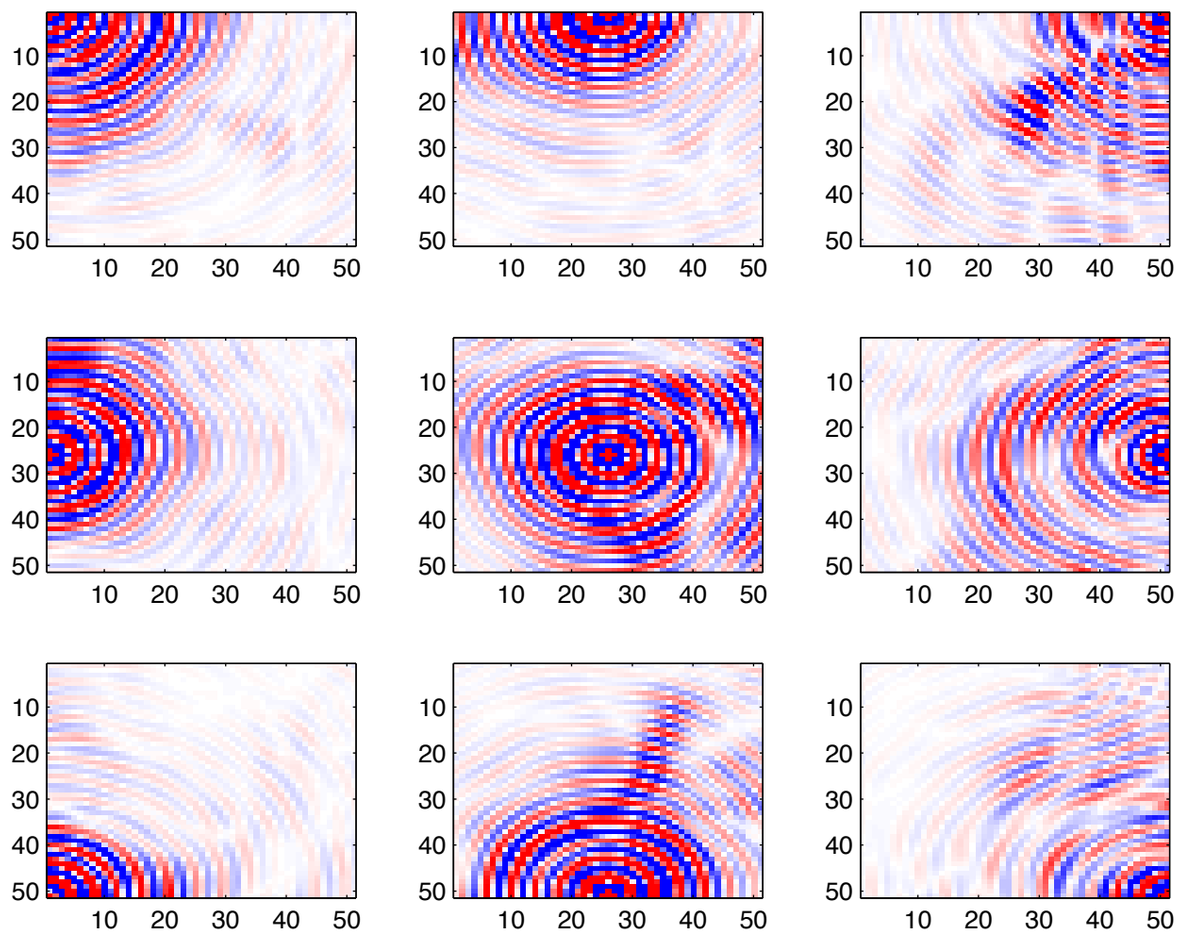
Performance

final residual @ 6Hz



Performance

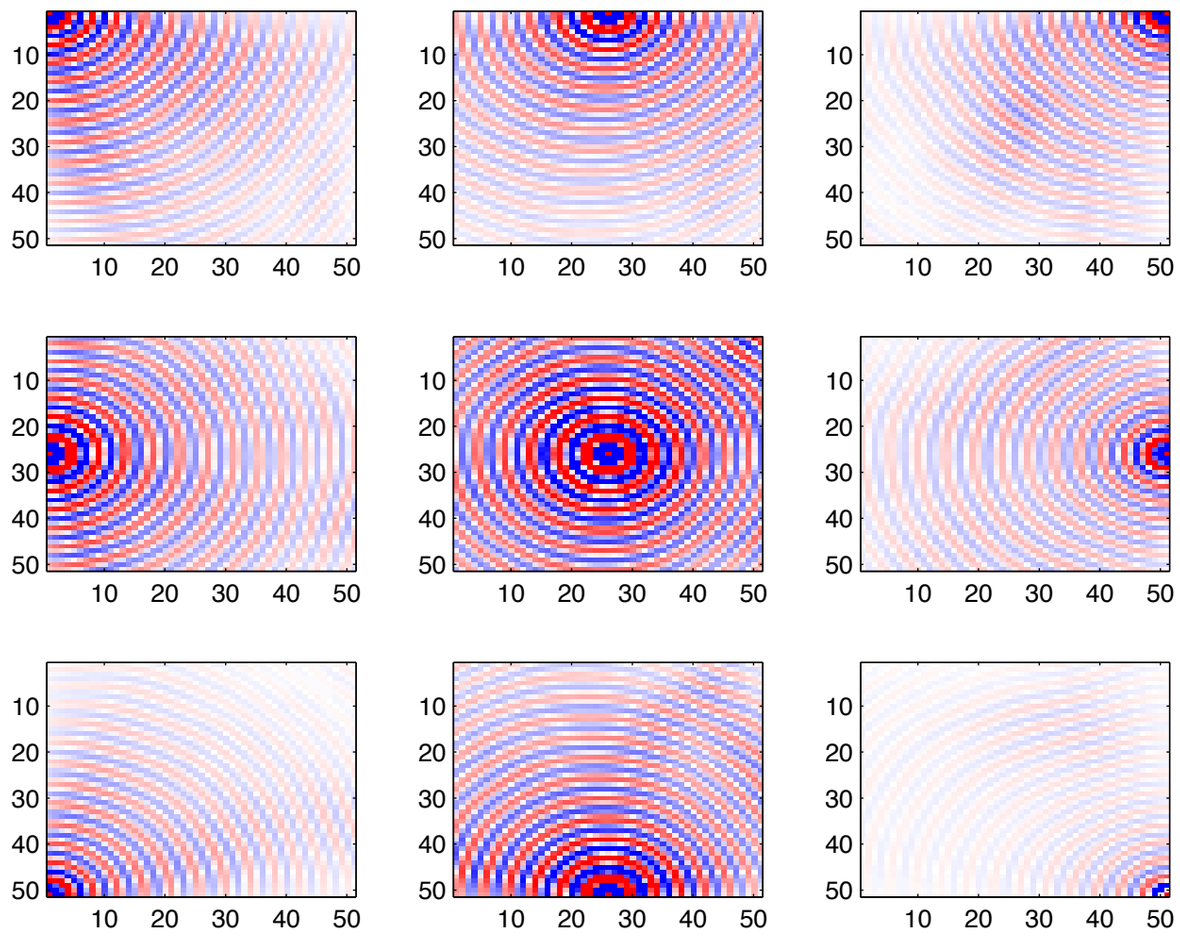
input data @ 8Hz



32 hours

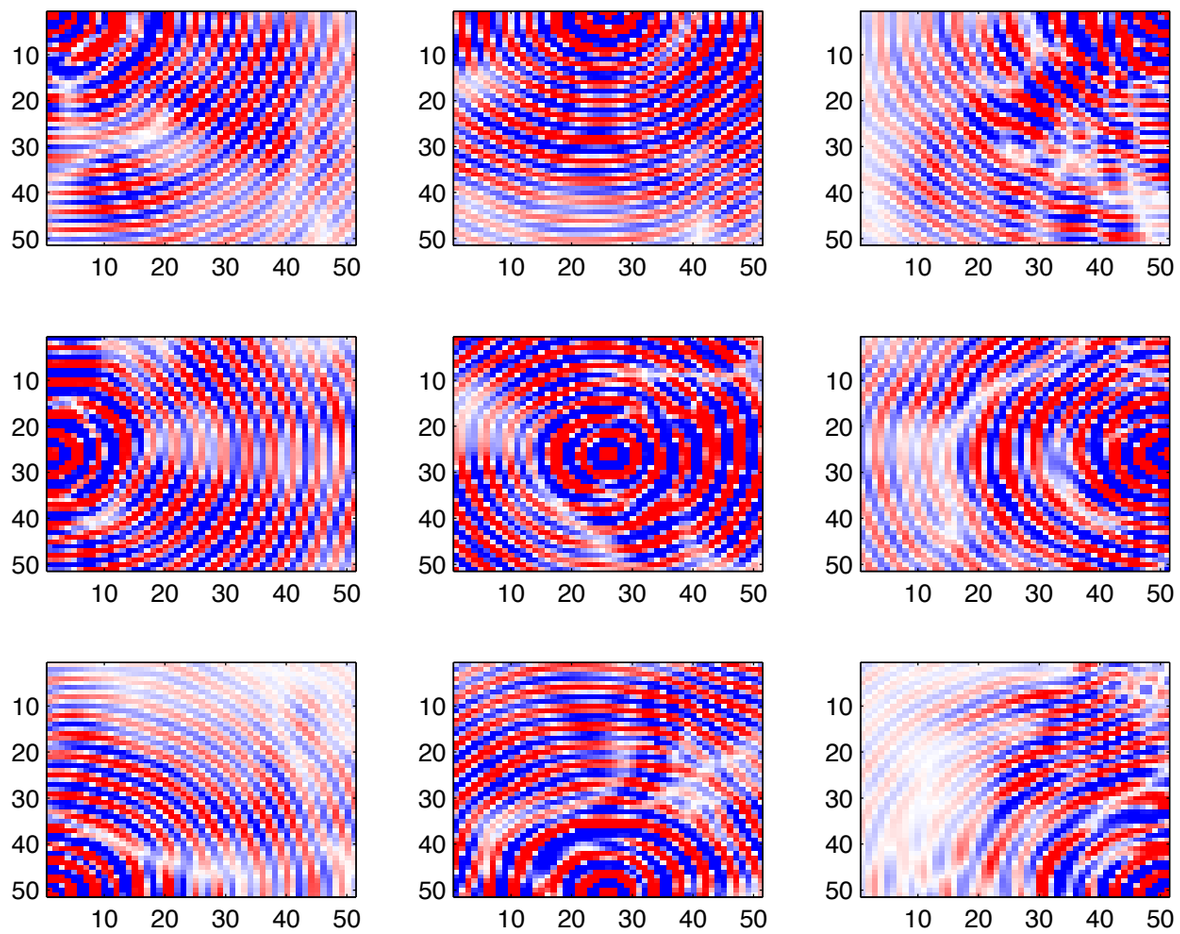
Performance

initial data @ 8Hz



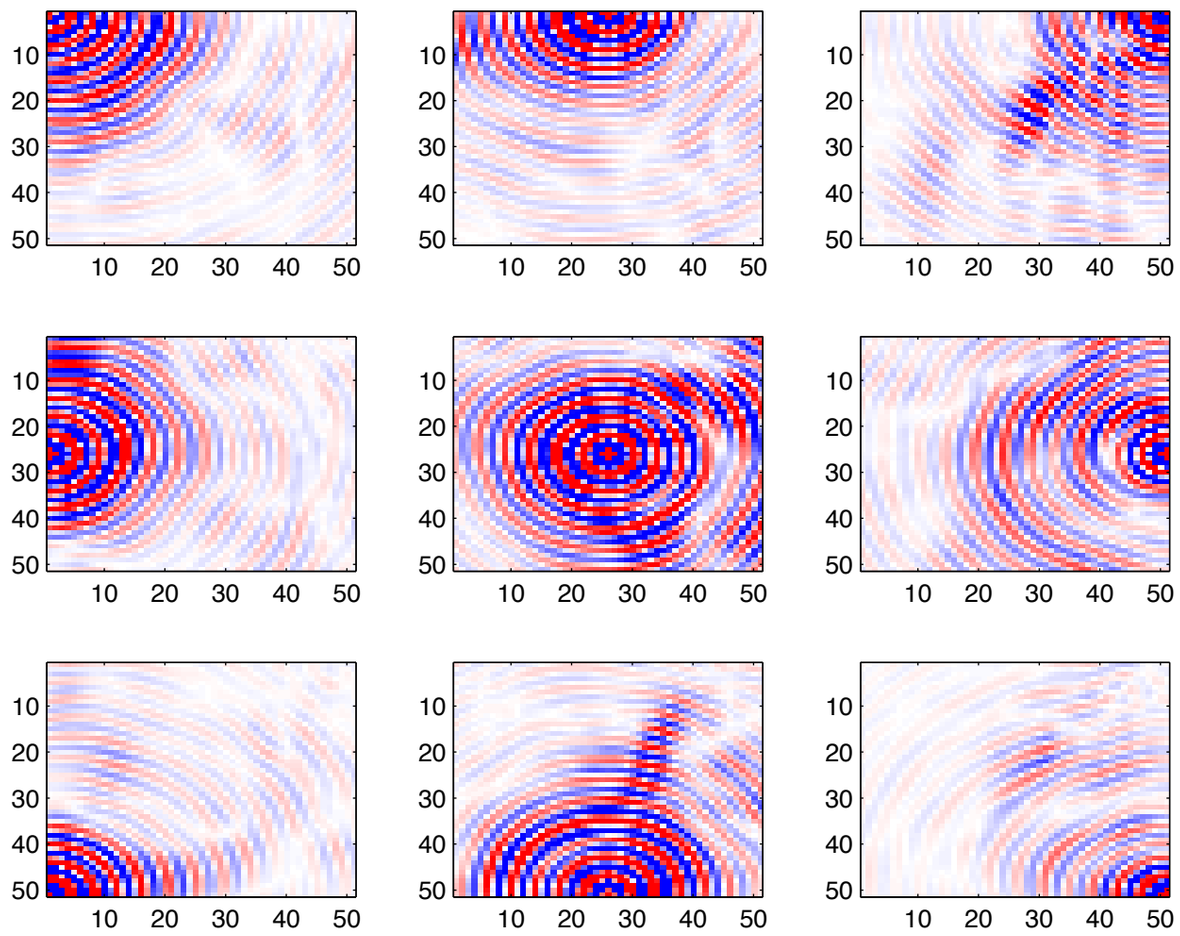
Performance

initial residual @ 8Hz



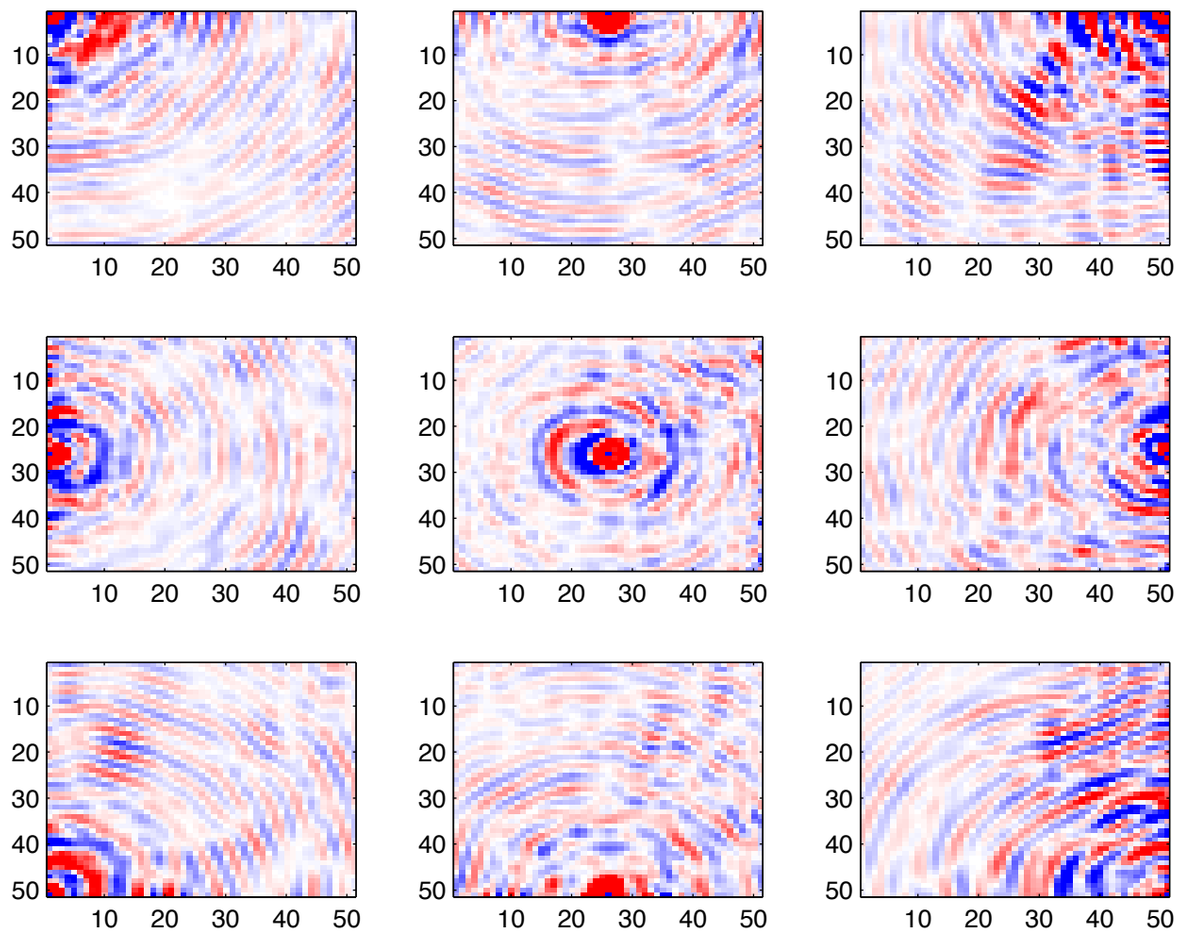
Performance

final data @ 8Hz



Performance

final residual @ 8Hz



Observations

Able to carry out 3-D FWI with *dynamic*

- growth of *sample* size
- *tolerance* PDE solves

Model error decays much *faster* compared to *working* with *all* data

Opens possibilities to use *sophisticated* regularizations

Summary

Main *ingredients* for a *scalable* approach to 3D FWI:

- *iterative* Helmholtz solver w/ *little* memory imprint, computational overhead, and model-dependent tuning
- practical *stopping* criterion for wave simulator
- (stochastic) optimization technique that exploits the *separable* structure of FWI by working w/ *small* subsets
- *strategy* to *increase* sample size and accuracy as needed

Carry home message

Insisting on working w/

- *all* data
- *full* accuracy

can be *detrimental* to *FWI*.

When *ill*-conditioned use *less* rather than *more* data & *accuracy*.

Better to *call* for *more* data & *accuracy* only when *strictly* needed.

Less is really more...