A Unified 2D/3D Software Environment for Large Scale Time-Harmonic Full Waveform Inversion

Curt Da Silva & Felix Herrmann

SLIM
University of British Columbia

3D Full Waveform Inversion

Complicated process

- computationally intensive
- requires lots of memory, time
- large amount of programmer effort to get things *fast*
- often speed is the trade off for *correctness*

3D Full Waveform Inversion

Industry codebases, while fast

- are *inflexible* - hard to integrate new changes
- are *incorrect* - no ‘true derivatives’ of the underlying modelling code
- are *poorly maintained* - a new hire will have no idea what’s going on

3D Full Waveform Inversion

As a result

- codes are disconnected from mathematical underpinnings
- bugs are hard to diagnose
- difficult to incorporate new ideas from academia, research labs in to production-level codes

Software organization

Software hierarchy manages complexity

- human brains have very limited working memory
- if a particular part of a program only has one function, people using/debugging it only have to think about that one function
- if software is easier to reason about -> it’s easier to work with, easier to test

Software organization

Software hierarchy manages complexity

- we don't have to sacrifice performance
  - lowest level operations implemented in C w/multithreading

- hiding irrelevant details at each level
  - higher level functions don't have any idea about C/fortran/that gross stuff

Software organization

Anything that we do that isn’t solving PDEs is essentially irrelevant, computation time-wise

Software organization

Anything that we do that isn't solving PDEs is essentially irrelevant, computation time-wise

- advantageous for software design -> any overhead introduced is negligible compared to solving PDEs

  - if a single wavefield can be stored in RAM - true for low frequency time harmonic FWI

Software organization

PDEs are the computational bottleneck

- design our software for maximum ease of use + “plug and play” components
- speedups made to solving PDEs propagate to whole framework

FWI Problem

$$\min_{m} \frac{1}{2N_s} \sum_{i=1}^{N_s} \|P_r H(m)^{-1} q_i - d_i\|_2^2$$

m - discrete model vector
N_s - number of shots
P_r - receiver restriction operator
$H(m)u_i = q_i$ - monochromatic Helmholtz system for shot i
d_i - measured data for shot

New way to organize FWI Software

opAbstractMatrix

Modeling matrix : multiplication/division

opAbstractMatrix

A SPOT operator

- linear operator class - behaves like a matrix
- knows how to multiply, divide itself
- can handle matrix-free operations or form sparse matrix for 2D problems

Extensions

- Kaczmarz sweeps
- Jacobi iterations

opAbstractMatrix

Particular matrix-vector products specified at construction

discrete_helmholtz - constructs Helmholtz operator with particular parameters

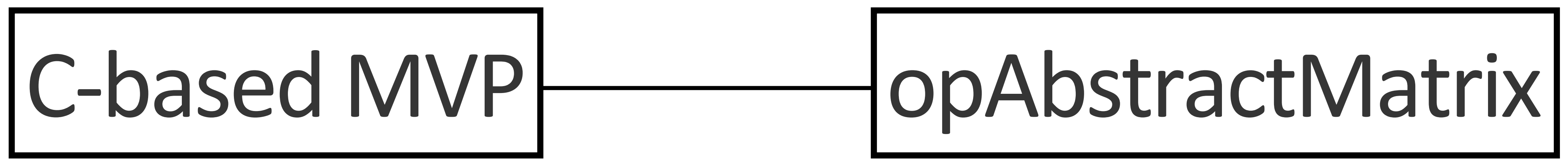
- can swap between stencils
- construct multigrid preconditioner

Excerpt from discrete_helmholtz

```
wn = param2wavenum(v_pml,freq,model.unit);
switch scheme
  case PDEopts.HELM3D_STD7
    Hmvp = FuncObj(@helm3d_7pt_mvp_mex,{vec(wn),vec(dt),vec(nt),npml,freq,[],[]});
    jacobi = FuncObj(@helm3d_7pt_jacobi_mex,{vec(wn),vec(dt),vec(nt),npml,freq,[],[],[],[]});
    kacz_sweep = [];
  case PDEopts.HELM3D_OPERTO27
    Hmvp = FuncObj(@helm3d_operto27_mvp,{wn,dt,nt,npml,[],n_threads,[],false});
    jacobi = [];
    kacz_sweep = FuncObj(@helm3d_operto27_kaczswp,{wn,dt,nt,npml,[],[],[],[],[]});
    if nargout >= 3
        [~,wn] = param2wavenum(v_pml,freq,model.unit);
        dHmvp = FuncObj(@helm3d_operto27_mvp,{wn,dt,nt,npml,[],n_threads,[],true});
        [~,~,wn] = param2wavenum(v_pml,freq,model.unit);
        ddHmvp = FuncObj(@helm3d_operto27_mvp,{wn,dt,nt,npml,[],n_threads,[],true});
    end
end
helm_params = struct;
helm_params.multiply = Hmvp;
helm_params.jacobi = jacobi;
helm_params.kacz_sweep = kacz_sweep;
helm_params.N = prod(nt);
helm_params.iscomplex = true;
H = opAbstractMatrix(mat_mode,helm_params,opts.solve_opts);
```

New way to organize FWI Software

Multithreaded
Mat-vec multiply



Modeling matrix :
multiplication/division

SLIM

Te

[1] Operto et. al. “3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study”, Geophysics 2007

C-based Matrix Vector Product

Implementation of 27-pt compact stencil [1]

Multi-threaded along the z-coordinate with openMP

Forward, adjoint modes

Te

[1] Chen, et. al. “An Optimal 9-Point Finite Difference Scheme For The Helmholtz Equation With PML.”, 2013

Helmholtz matrix

In 2D, we can afford to use explicit sparse matrices + fast direct solvers

- implementation of [1]

Explicit matrices VS implicit matrices is opaque to the user

- interface remains the same

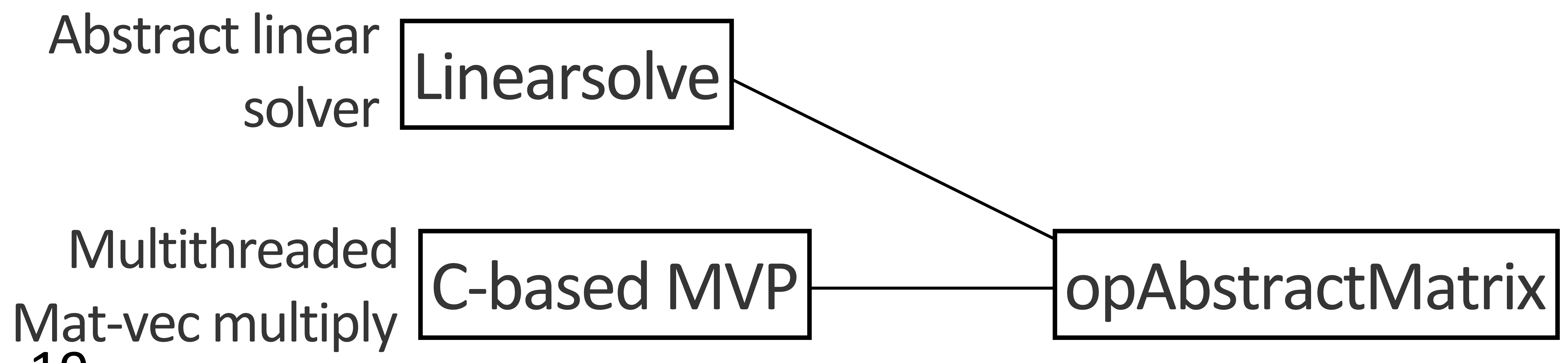
C-based Matrix Vector Product

Matlab Compiler

- write stencil-based code in Matlab -> C code with openMP multithreading

- nearly as fast as native C code, much easier to develop

New way to organize FWI Software



Modeling matrix : multiplication/division

Linearsolve

Abstract interface for “Solve Ax = b with a specified method”

- encourages code reuse - smoothers for multigrid, preconditioner applications

- calls the specified method (GMRES,CG, etc.) with the prescribed number of iterations, right hand side, initial guess, tolerance, and preconditioner

LinSolveOpts

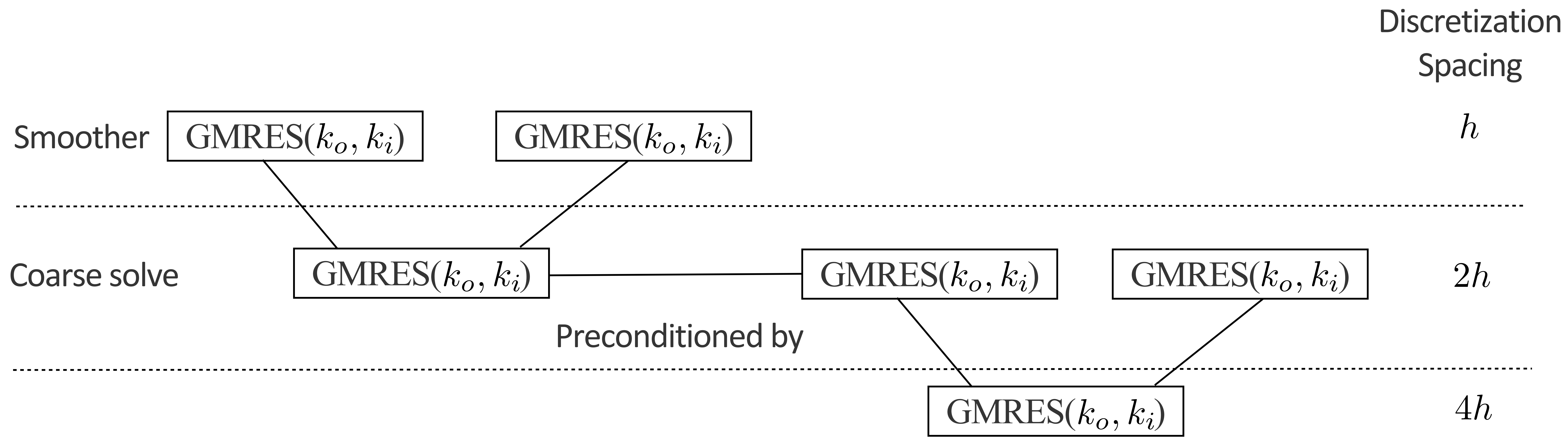
Object for storing

- linear solver method
- maximum outer iterations
- maximum inner iterations (for some solvers)
- tolerance
- preconditioner

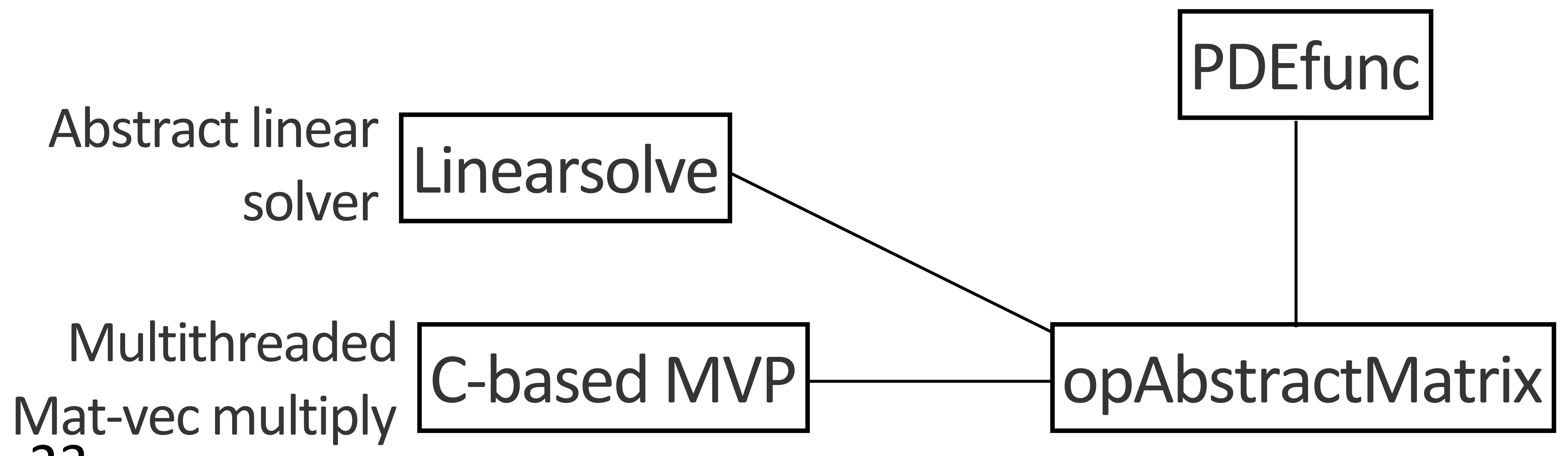
As well as default options for these

- Solvers: CG, FGMRES, LU, etc.
- Preconditioners: ML-GMRES, Shifted Laplacian, etc.

Multilevel-GMRES



New way to organize FWI Software



PDE-related quantities
Serial version

Modeling matrix : multiplication/division

PDEfunc

Main workhorse function

For each source index

- solve the Helmholtz equation - don't care how
- use solution to compute objective + gradient, demigration/ migration, hessian/GN hessian matrix vector product - whatever the user requests

Serial code, implicitly multithreaded

Excerpt from PDEfunc

```
Uk = Hk \ Qk_i;
switch func
   case OBJ
      [phi,dphi] = misfit(Pr*Uk,Dobs(:,data_idx),current_src_idx,freq_idx);
      f = f + phi;
      if nargout >= 2
        Vk = Hk' \ ( -Pr'* dphi);
          g = g + sum(real(conj(Uk) .* (dH'*Vk)),2);
      end

    case FORW_MODEL
      output(:,data_idx) = Pr*Uk;

   case JACOB_FORW
      dUk = Hk\(dHdm*(-Uk));
      output(:,data_idx) = Pr*dUk;

   case JACOB_ADJ
      Vk = Hk'\( -Pr'* input(:,data_idx) );
      output = output + sum(real(conj(Uk) .* (dH'*Vk)),2);
end
```

SLIM

PDEfunc

Extensions to Wave-equation Reconstruction Inversion

Standard FWI

$$\min_m \frac{1}{2}\|P_r u(m) - d\|_2^2$$
$$\text{s.t.} H(m)u(m) = q$$

SLIM

PDEfunc

Extensions to Wave-equation Reconstruction Inversion

$$\min_{m} \frac{1}{2}\|P_r u(m) - d\|_2^2 + \frac{\lambda}{2}\|H(m)u(m) - q\|_2^2$$

$$u(m) = \arg\min_{u} \left\| \begin{bmatrix} Pr \\ \lambda H(m) \end{bmatrix} u - \begin{bmatrix} d \\ \lambda q \end{bmatrix} \right\|_2$$

Song and Williamson,“Frequency-domain acoustic-wave modeling and inversion of crosshole data: Part I-2.5-D modeling method.” Geophysics (1995)

PDEfunc

Extensions to 2.5D FWI

- When the velocity is y-invariant

$$v(x,y,z) = h(x,z)$$

- After a Fourier transform in y-, the Helmholtz equation reads as

$$(\partial_x^2 + \partial_z^2 + \omega^2 h(x,z) - k_y^2)u_{k_y}(x,z) = S(\omega)\delta(x-x_s)\delta(z-z_s)$$

Song and Williamson,"Frequency-domain acoustic-wave modeling and inversion of crosshole data: Part I-2.5-D modeling method." Geophysics (1995)

SLIM

PDEfunc

Extensions to 2.5D FWI

- we can reconstruct the 3D wavefield $u(x,y,z)$ as

$$u(x,y,z) = \frac{1}{\pi}\int_0^{k_{nyq}} \tilde{u}_{k_y}(x,z)\cos(k_y(y-y_s))dk_y$$

$$= \sum_{j=1}^{N} w_j u_{k_y^j}(x,z)$$

Song and Williamson,“Frequency-domain acoustic-wave modeling and inversion of crosshole data: Part I-2.5-D modeling method.” Geophysics (1995)

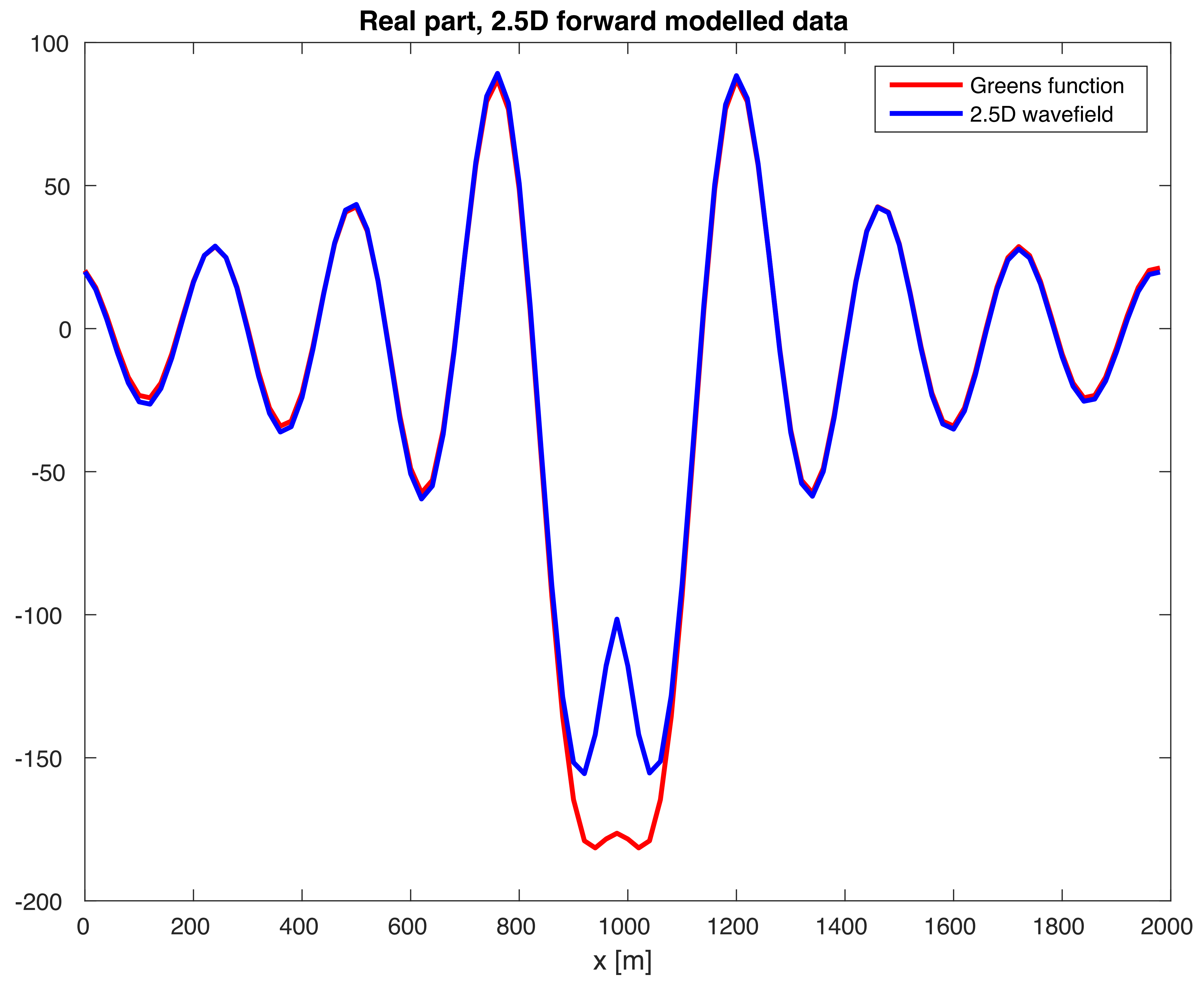
PDEfunc

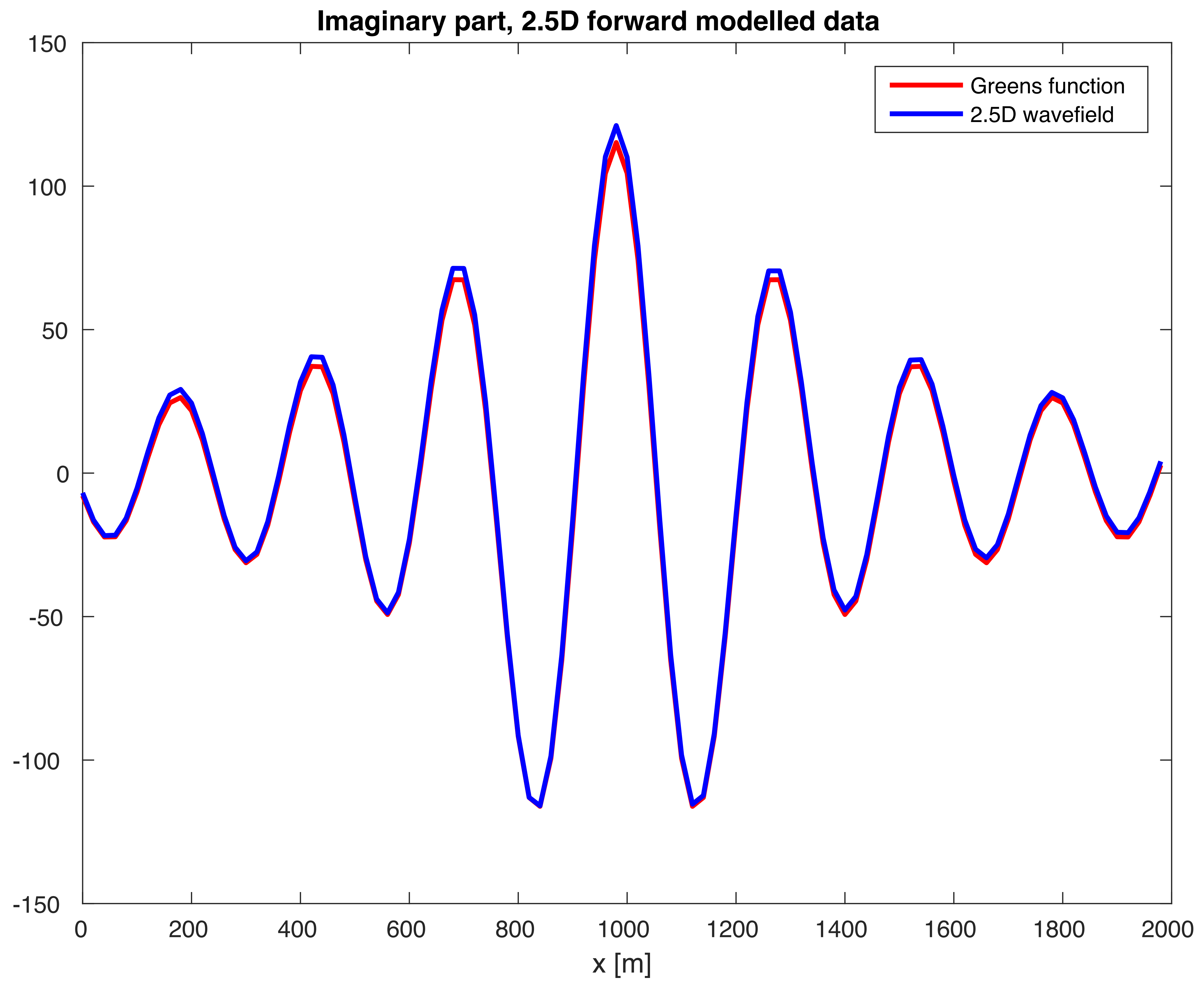
Extensions to 2.5D FWI

- weighted sum structure of the wavefield
  -> weighted sum structure for gradient, hessian, etc.

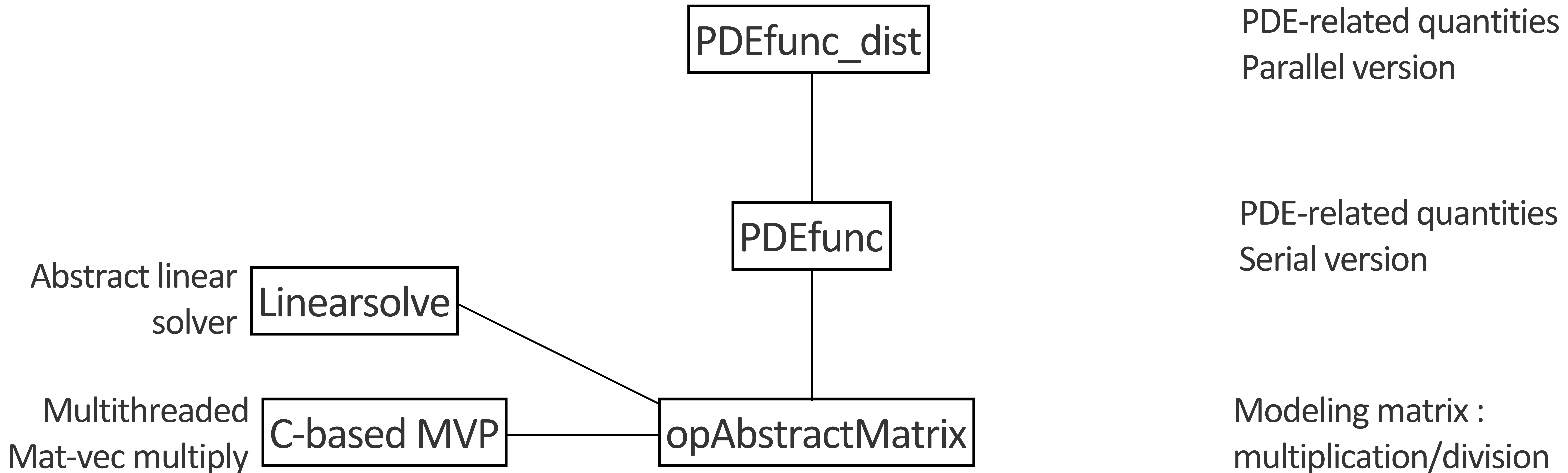
- correct 3D physics without full 3D costs

2.5D Modeling





New way to organize FWI Software



SLIM

Separable objective function

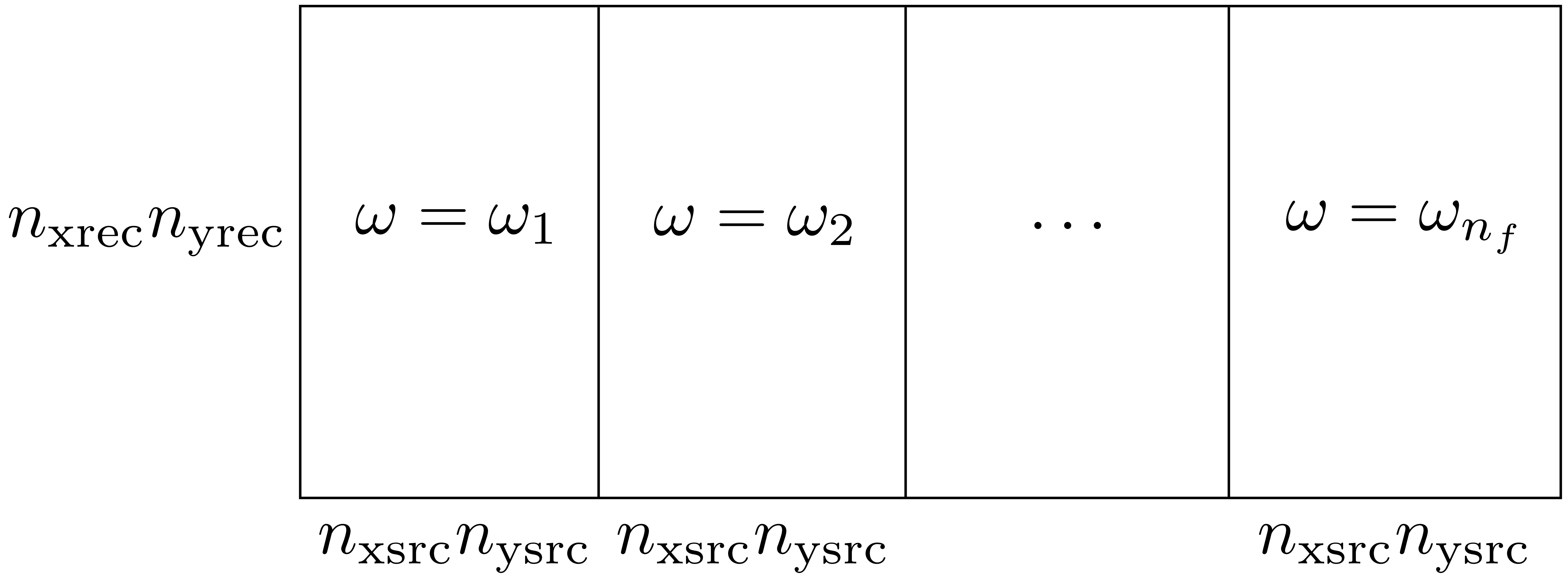
$$f_I(m) = \frac{1}{2|I|} \sum_{i \in I} \|P_r H(m)^{-1} q_i - d_i\|_2^2$$
$$= \frac{1}{2|I|} \sum_{i \in I} f_i(m)$$

The objective function is *separable* over shots/frequencies
- distribute indices to parallel workers

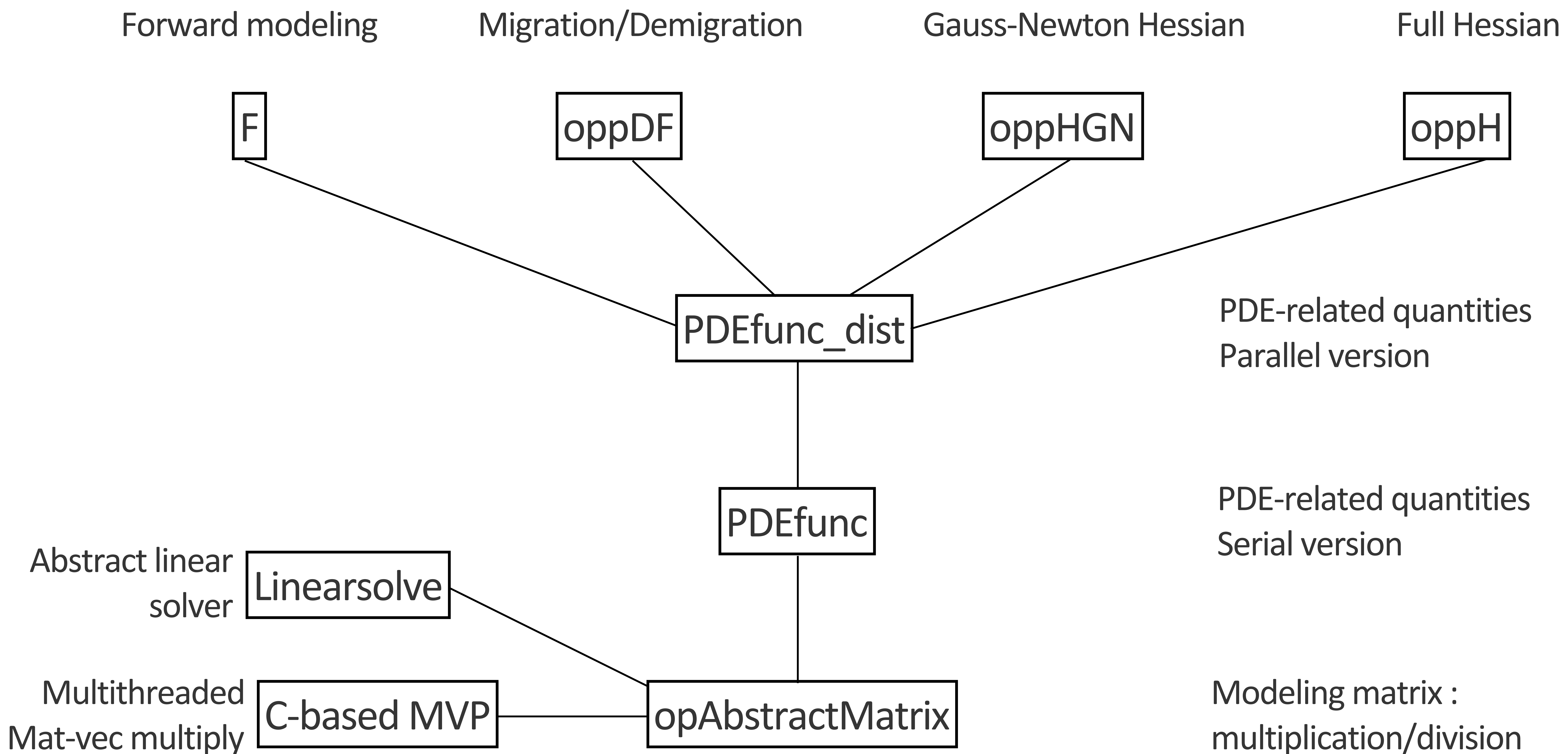
Objective separable -> gradient, GN Hessian, Hessian are separable

PDEfunc_dist does no computation, just parallel distribution + summation
- separate computation from parallelization
- easiest component to 'swap out' with your own parallelization scheme

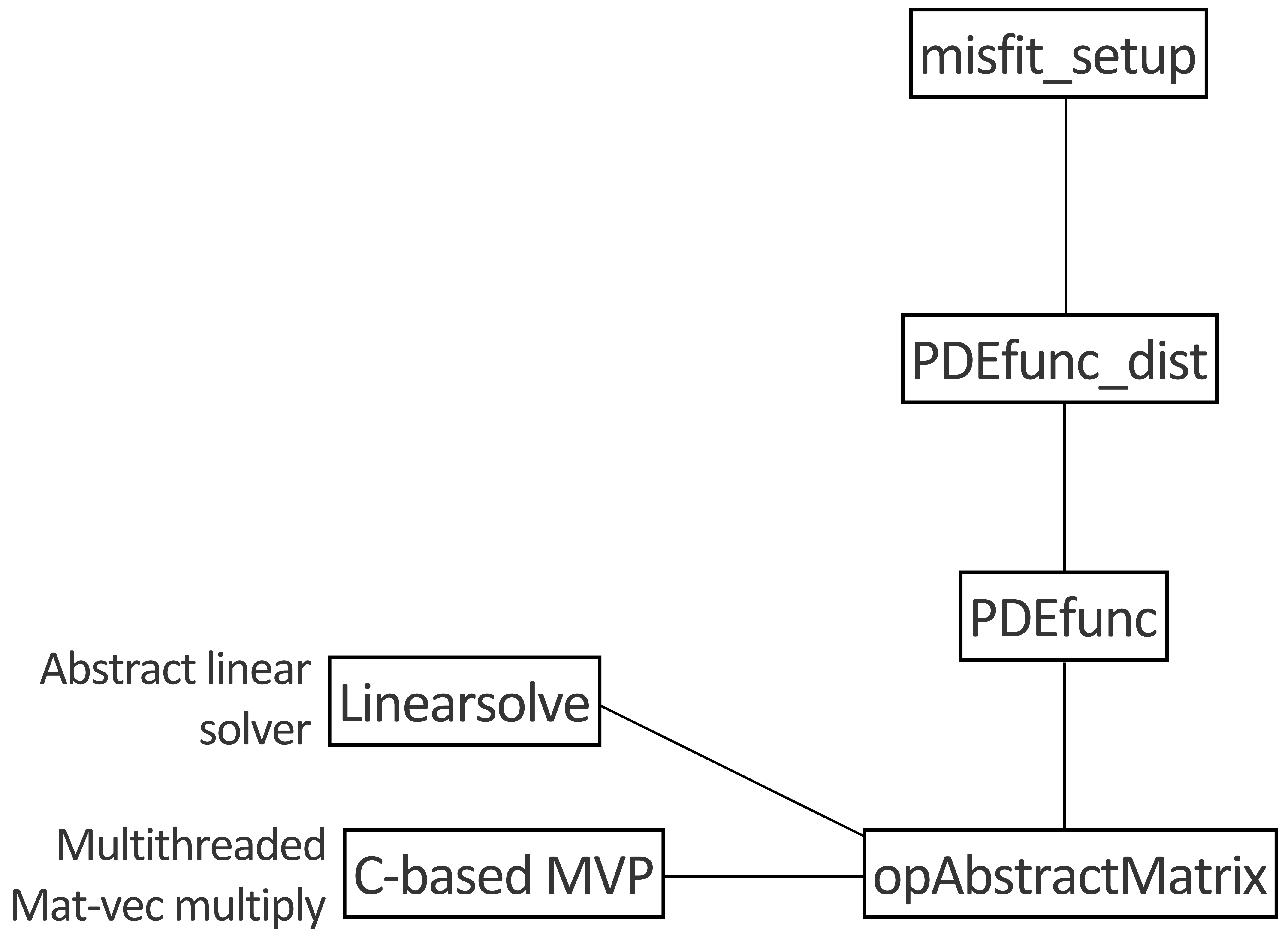
Data volume


$n_{\text{xrec}}n_{\text{yrec}}$
$\omega = \omega_1$
$\omega = \omega_2$
$\dots$
$\omega = \omega_{n_f}$
$n_{\text{xsrc}}n_{\text{ysrc}}$
$n_{\text{xsrc}}n_{\text{ysrc}}$
$n_{\text{xsrc}}n_{\text{ysrc}}$

New way to organize FWI Software



New way to organize FWI Software



FWI objective setup

PDE-related quantities
Parallel version

PDE-related quantities
Serial version

Modeling matrix : multiplication/division

misfit_setup

Constructs function handle for objective

- velocity subsampling
- frequency slice distribution

Batch mode interface to the objective

- stochastic inversion algorithm can specify which source indices to use

Fancy wrapper around PDEfunc_dist

PDEopts

Options for specifying

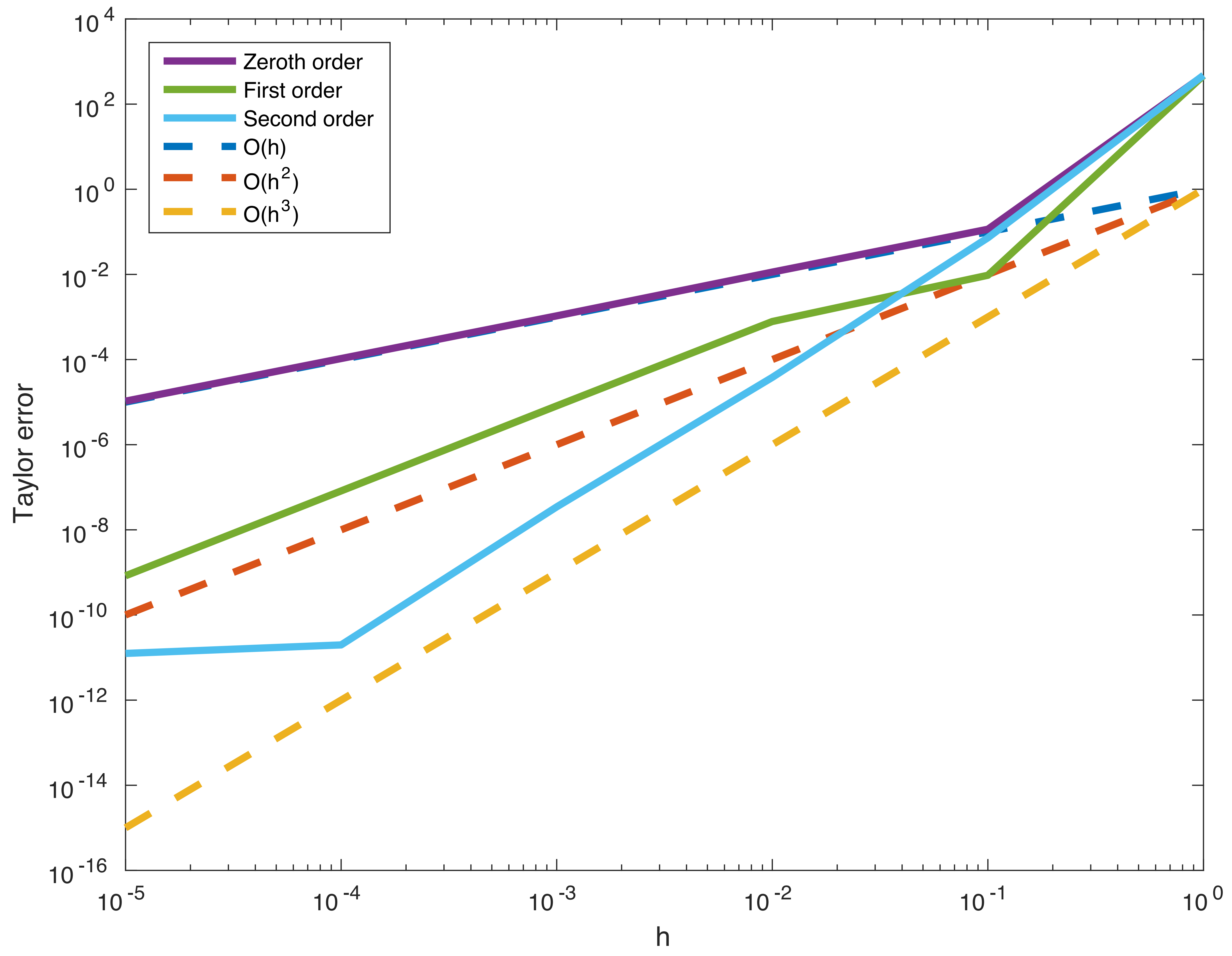
- PDE stencil
- PML width/layout
- preconditioner
- source/receiver interpolation
- source estimation
- ...

Taylor error test

$$f(m + h\delta m) - f(m) = O(h)$$

$$f(m + h\delta m) - f(m) - h\nabla f(m)^T \delta m = O(h^2)$$

$$f(m + h\delta m) - f(m) - h\nabla f(m)^T \delta m - \frac{h^2}{2}\delta m^T \nabla^2 f(m) \delta m = O(h^3)$$



SLIM

Adjoint Test

	$\langle Ax, y \rangle$	$\langle x, A^H y \rangle$	Relative Difference
Helmholtz system matrix	$1.903020 + 2.087502i \cdot 10^1$	$1.903020 + 2.087502i \cdot 10^1$	$1.51 \cdot 10^{-15}$
Jacobian	$-6.204229 \cdot 10^{-2}$	$-6.204229 \cdot 10^{-2}$	$6.8525 \cdot 10^{-10}$
Hessian	$-5.842717 \cdot 10^{-3}$	$-5.842717 \cdot 10^{-3}$	$7.9767 \cdot 10^{-11}$

Results

Algorithm

$$\min_m \frac{1}{N_s}\sum_{i=1}^{N_s} f_i(m)$$
$$\text{s.t.} \quad m_L \leq m \leq m_U$$

m - discrete model vector

m_L, m_U - point-wise model bounds (water layer + constant min/max velocities)

$f_i(m) = \frac{1}{2}\|P_r H(m)^{-1} q_i - d_i\|_2^2$ - per-shot misfit function

P_r - receiver restriction operator

$H(m)u_i = q_i$ - discrete Helmholtz system for shot i

d_i - measured data for shot

Algorithm

$$\min_m \frac{1}{N_s} \sum_{i=1}^{N_s} f_i(m)$$
$$\text{s.t.} \quad m_L \leq m \leq m_U$$

We have too many shots to process at once

- Can process p shots at a time when we have p Matlab workers

Typically $N_s \gg p$

Schmidt, et. al., “Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm”, 2009

Algorithm

$$m_k = \arg\min_m \frac{1}{|I_k|} \sum_{i \in I_k} f_i(m)$$

$$\text{s.t.} \quad m_L \leq m \leq m_U$$

At the kth iteration, randomly draw a subset of sources $I_k \subset \{1, \ldots, N_s\}$ with $|I_k| = p$

Approximately solve the above problem with constrained LBFGS or spectral projected gradient

Repeat for T iterations

Algorithm

Inner subproblem

- solved with $\frac{N_s}{p}$ function evaluations
- each subproblem is equivalent to one pass over the full data

We use three outer iterations

- equivalent to three gradient steps with all the shots

3D FWI Example

Overthrust model

- 20 km x 20 km x 4.6 km - 50 m spacing, 500m water layer
- 50 x 50 sources, 200m spacing - 2500 shots
- 401 x 401 receivers, 50m spacing
- 3Hz - 6Hz frequency range, single freq. inverted at a time

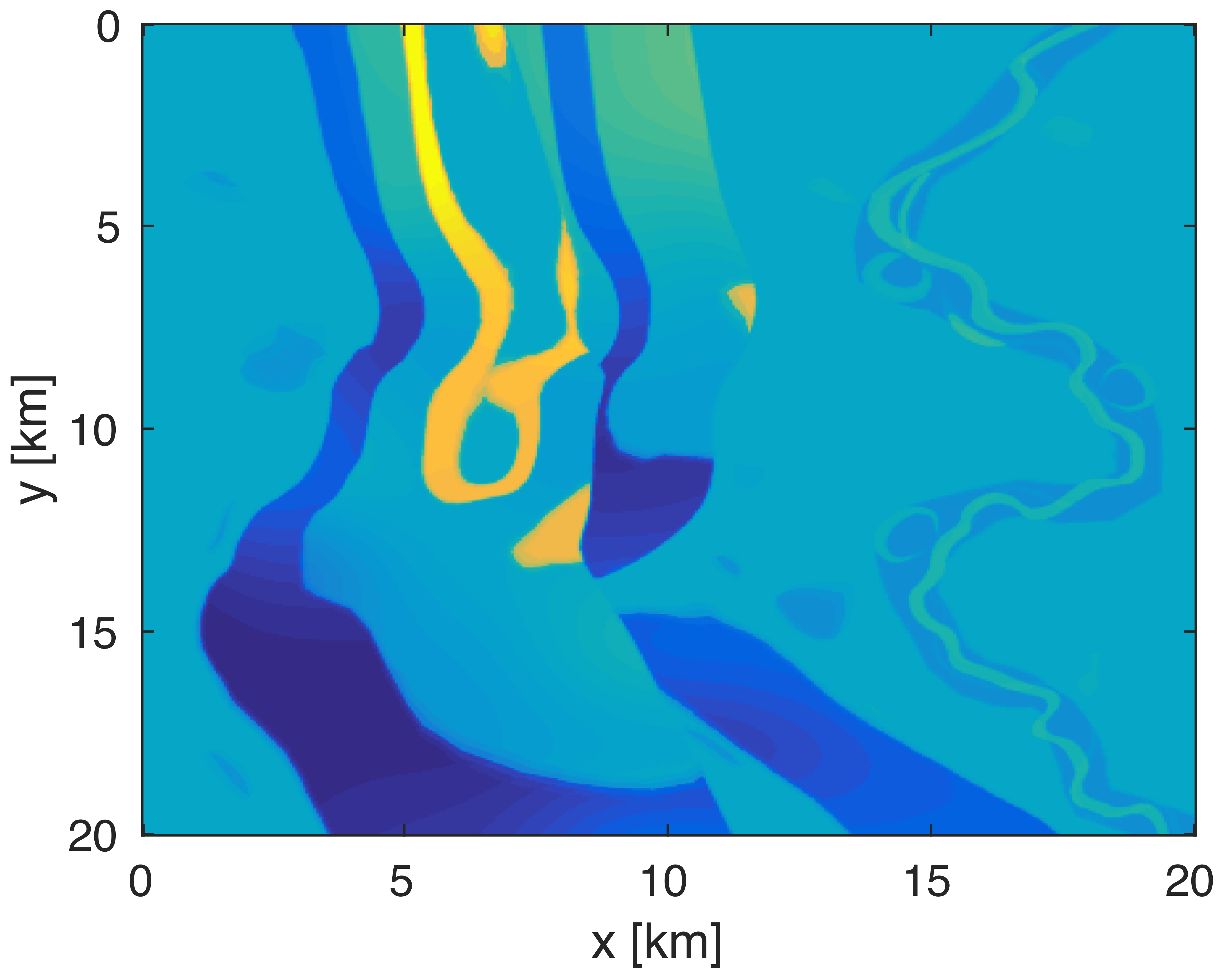
Computational Environment

SENAI Yemoja cluster

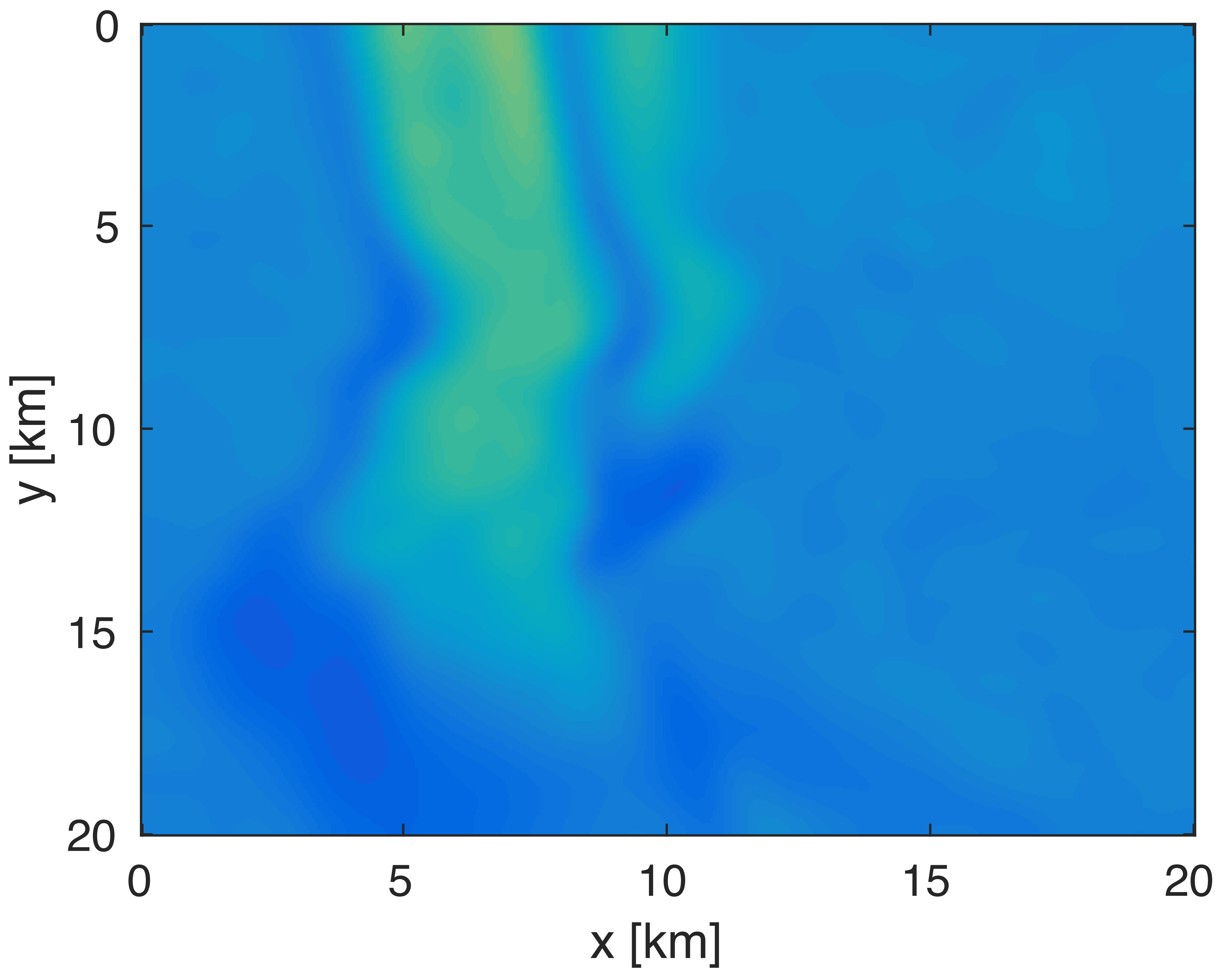
- 100 nodes, 128 GB RAM each, 20-core processors

- 400 Parallel Matlab workers (4 per node), Helmholtz MVP uses 5 threads - full core utilization

z=1000m slice

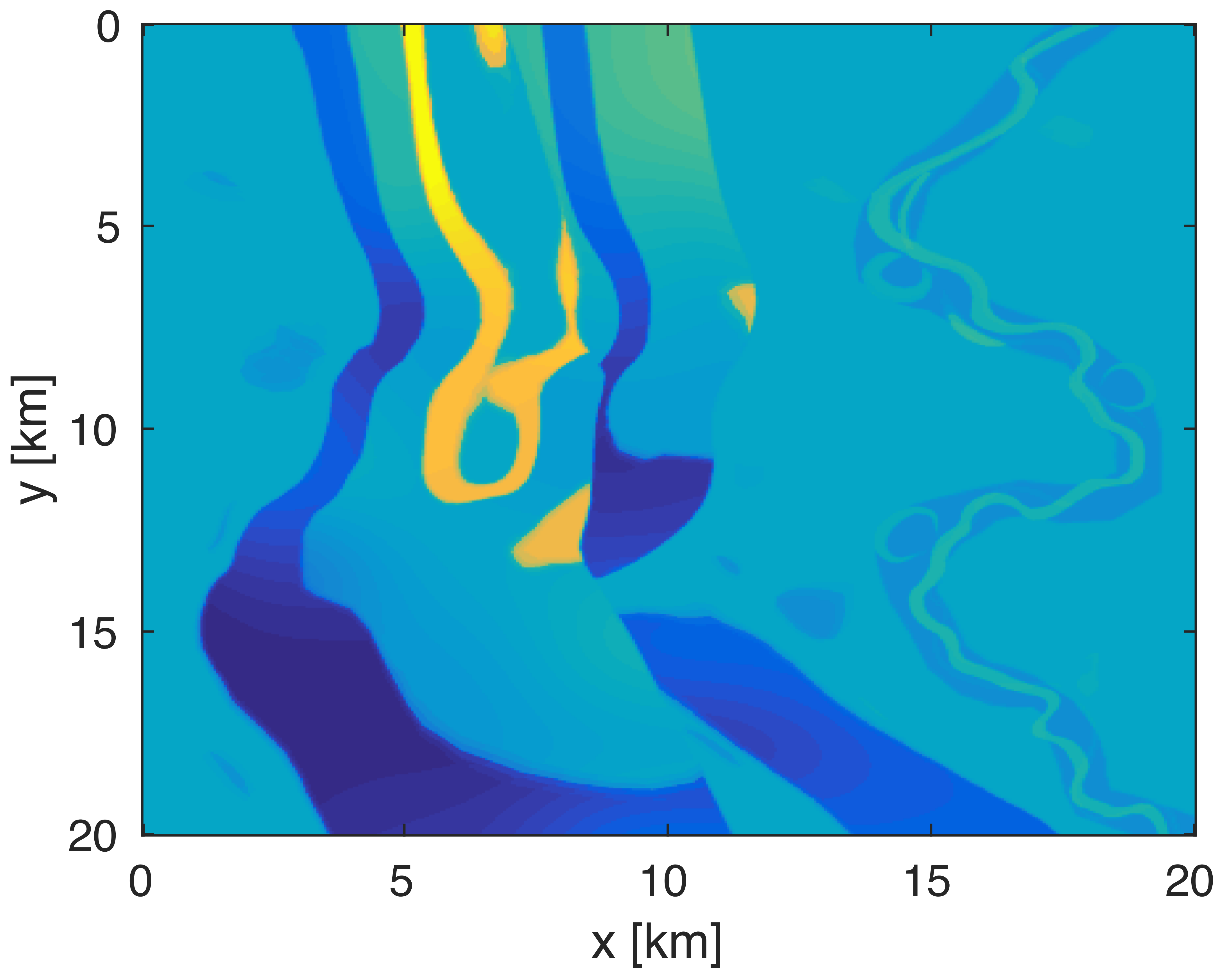


True model

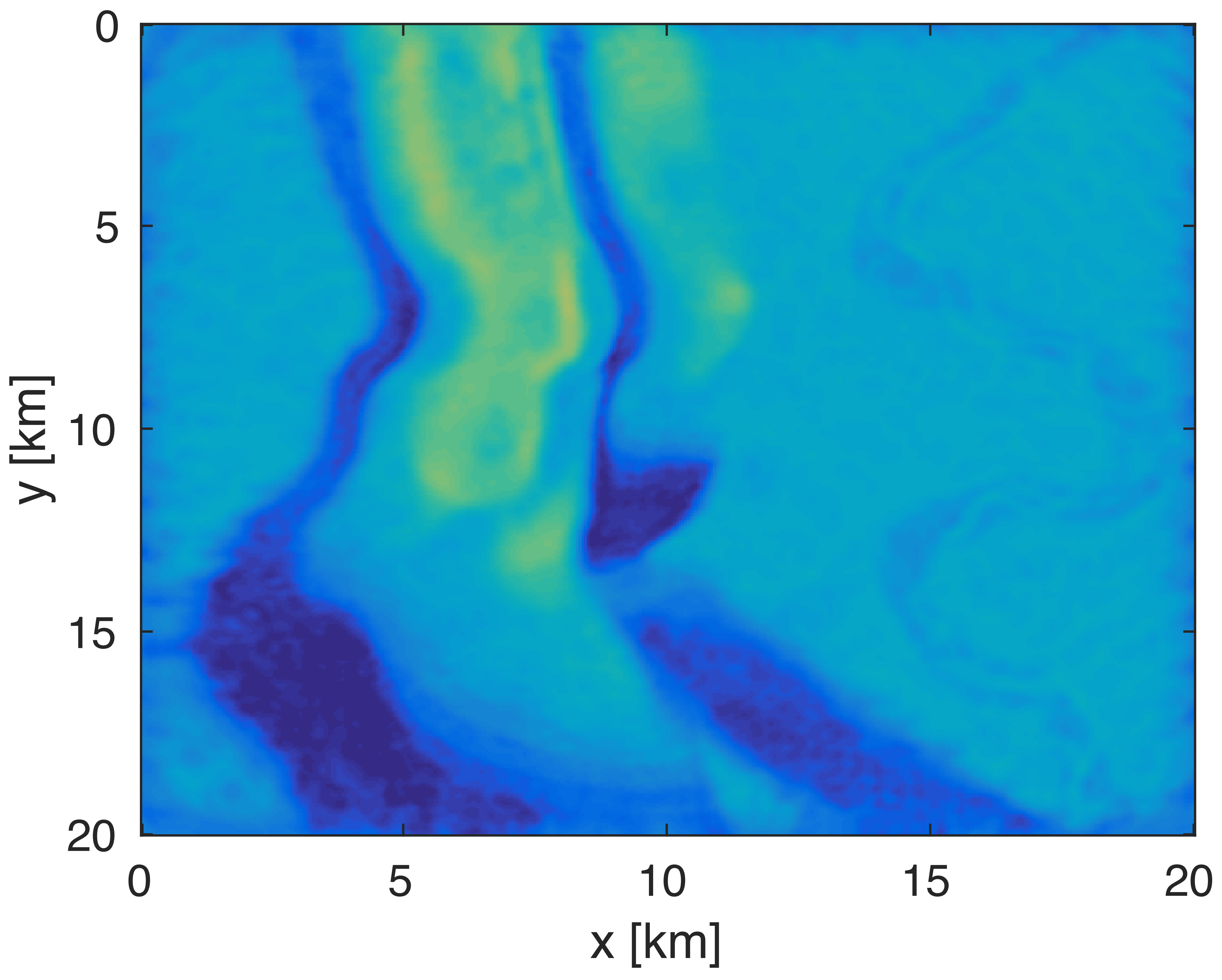


Initial model

z=1000m slice

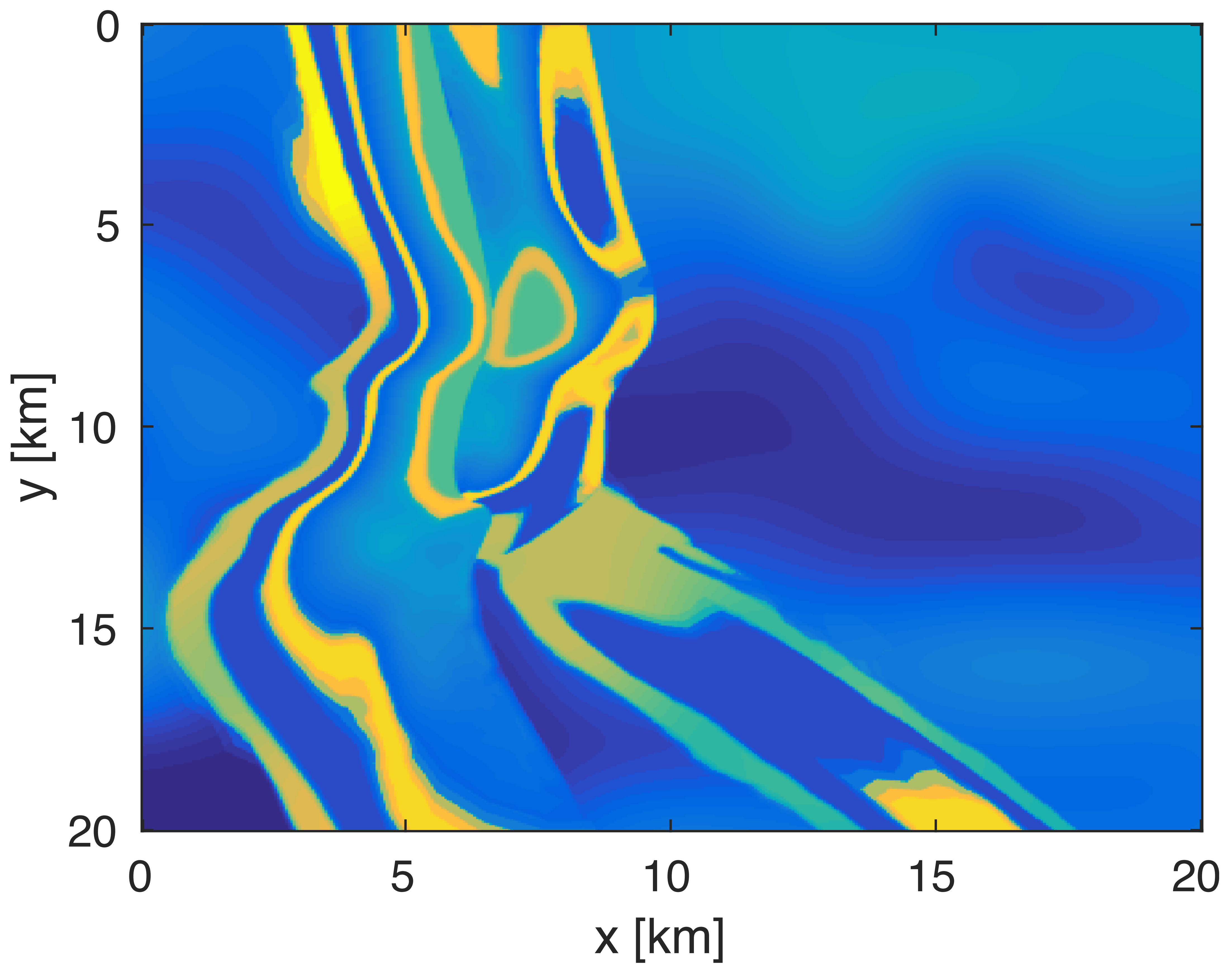


True model

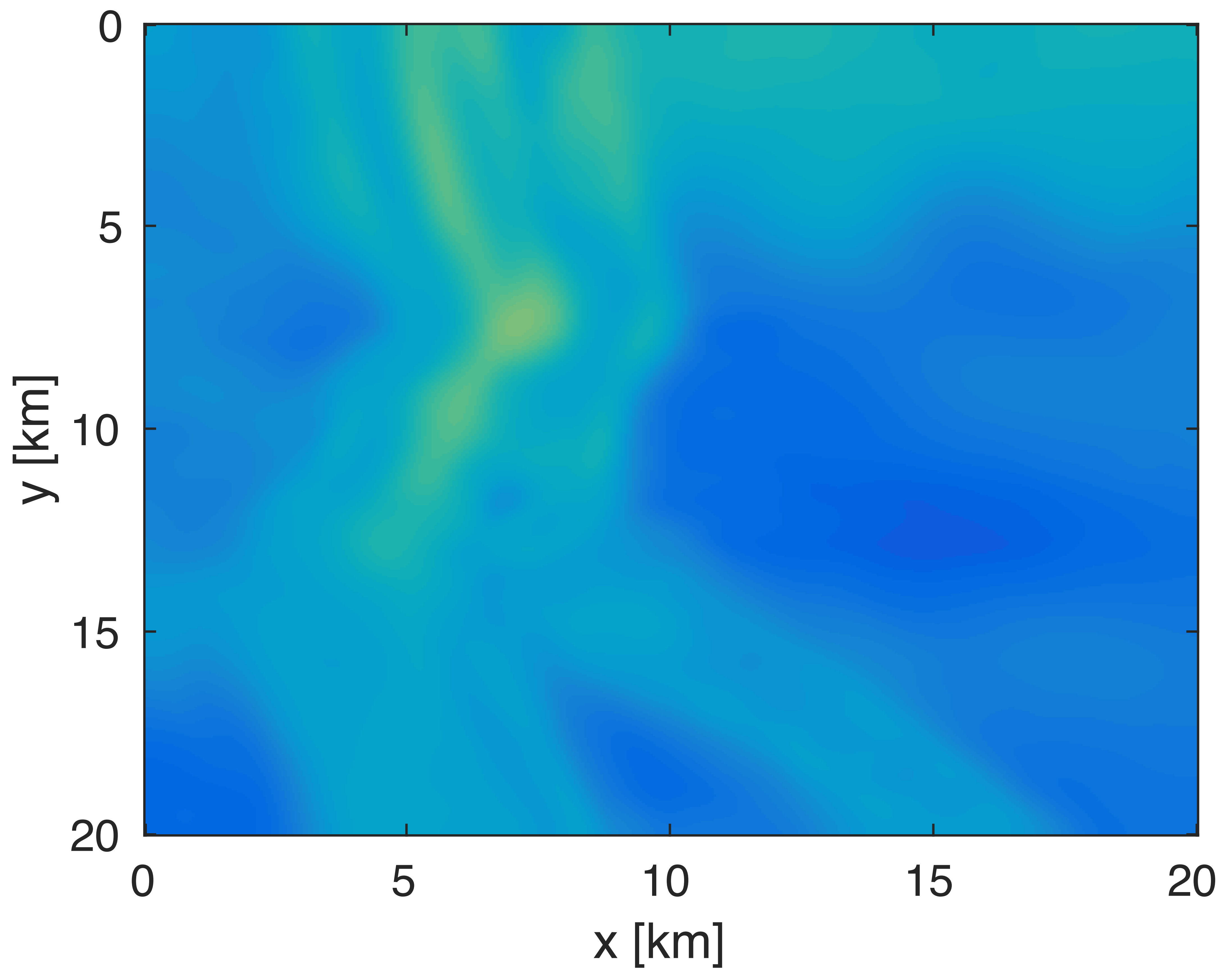


Stochastic LBFGS

z=2000m slice

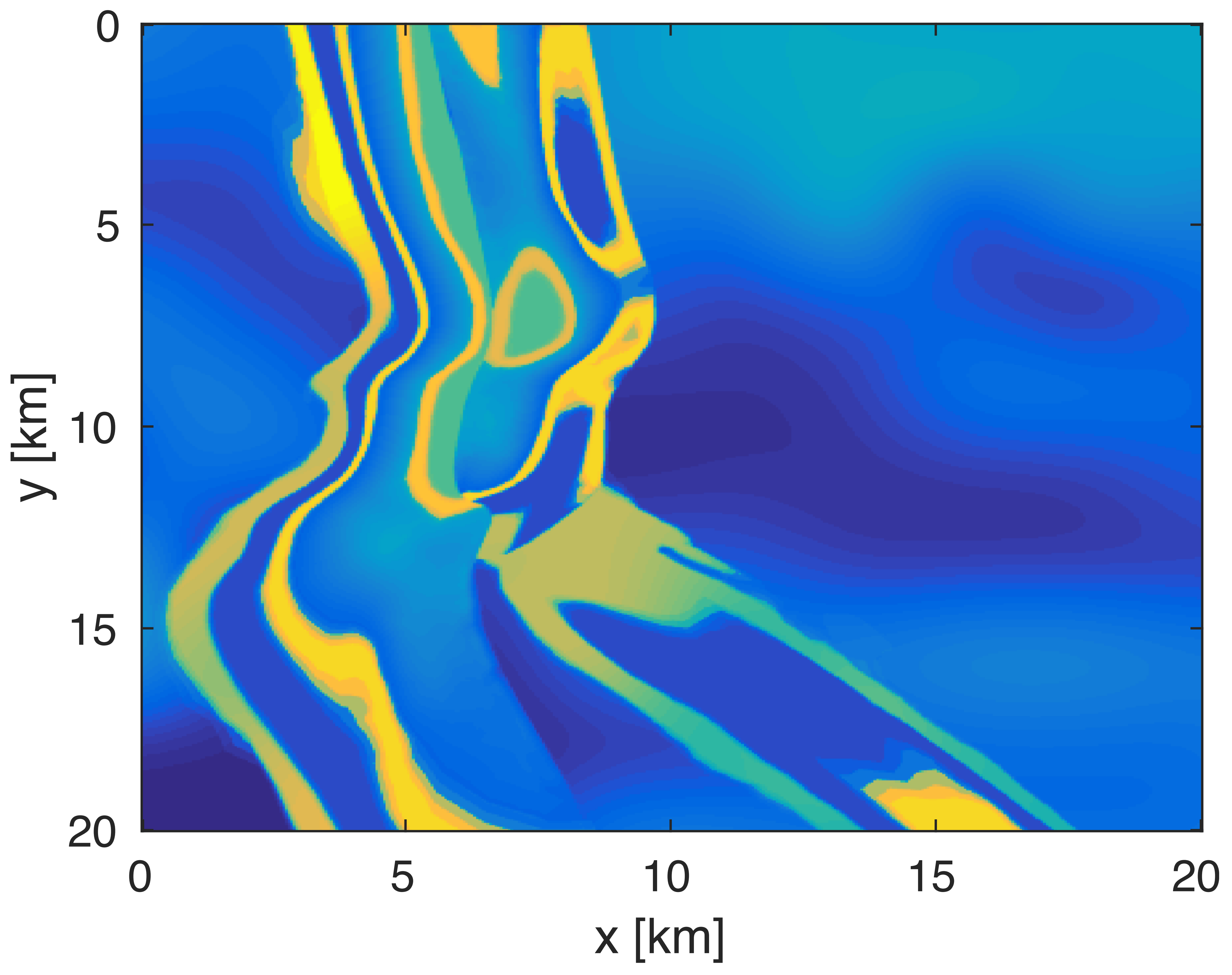


True model

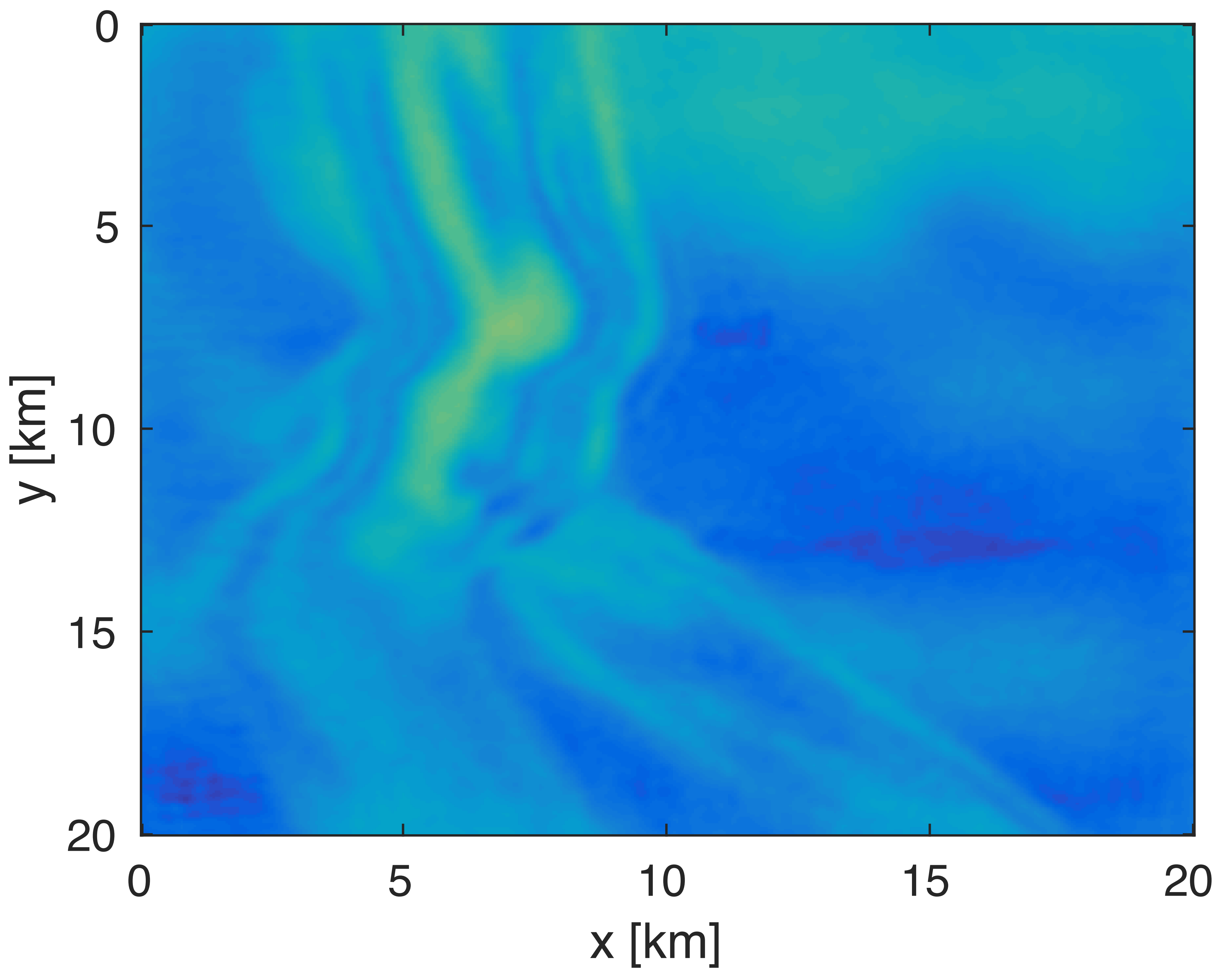


Initial model

z=2000m slice

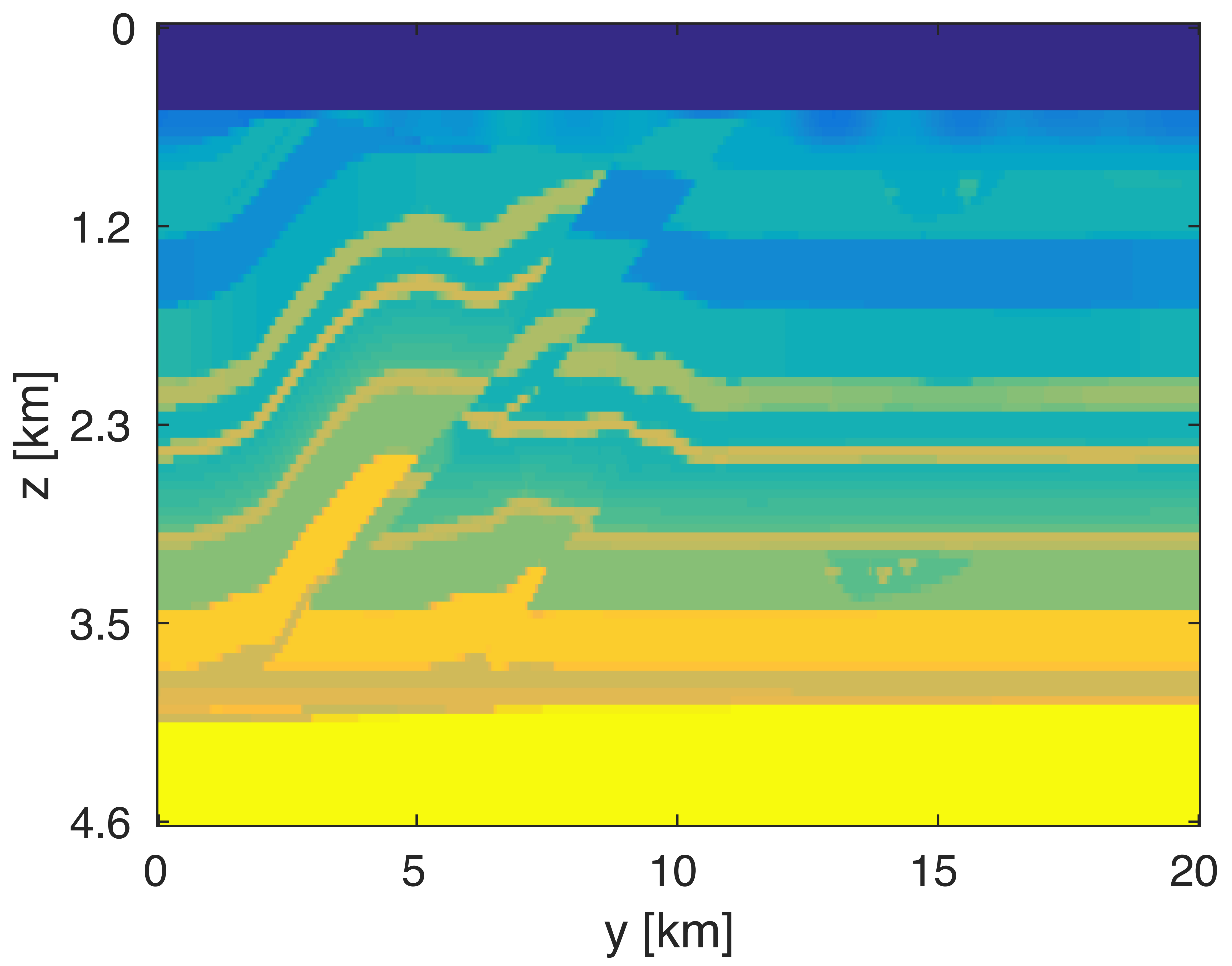


True model

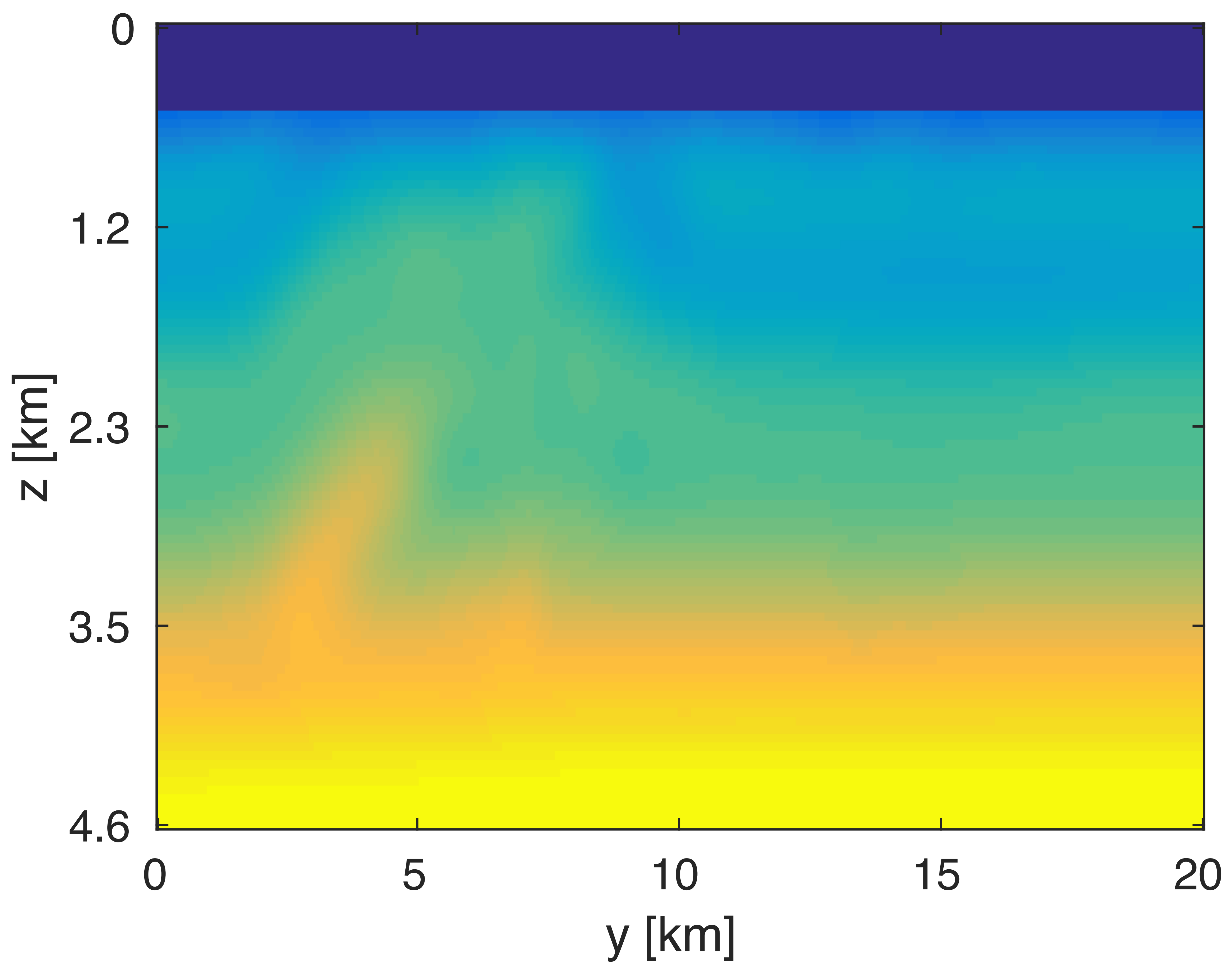


Stochastic LBFGS

x=12.5km slice

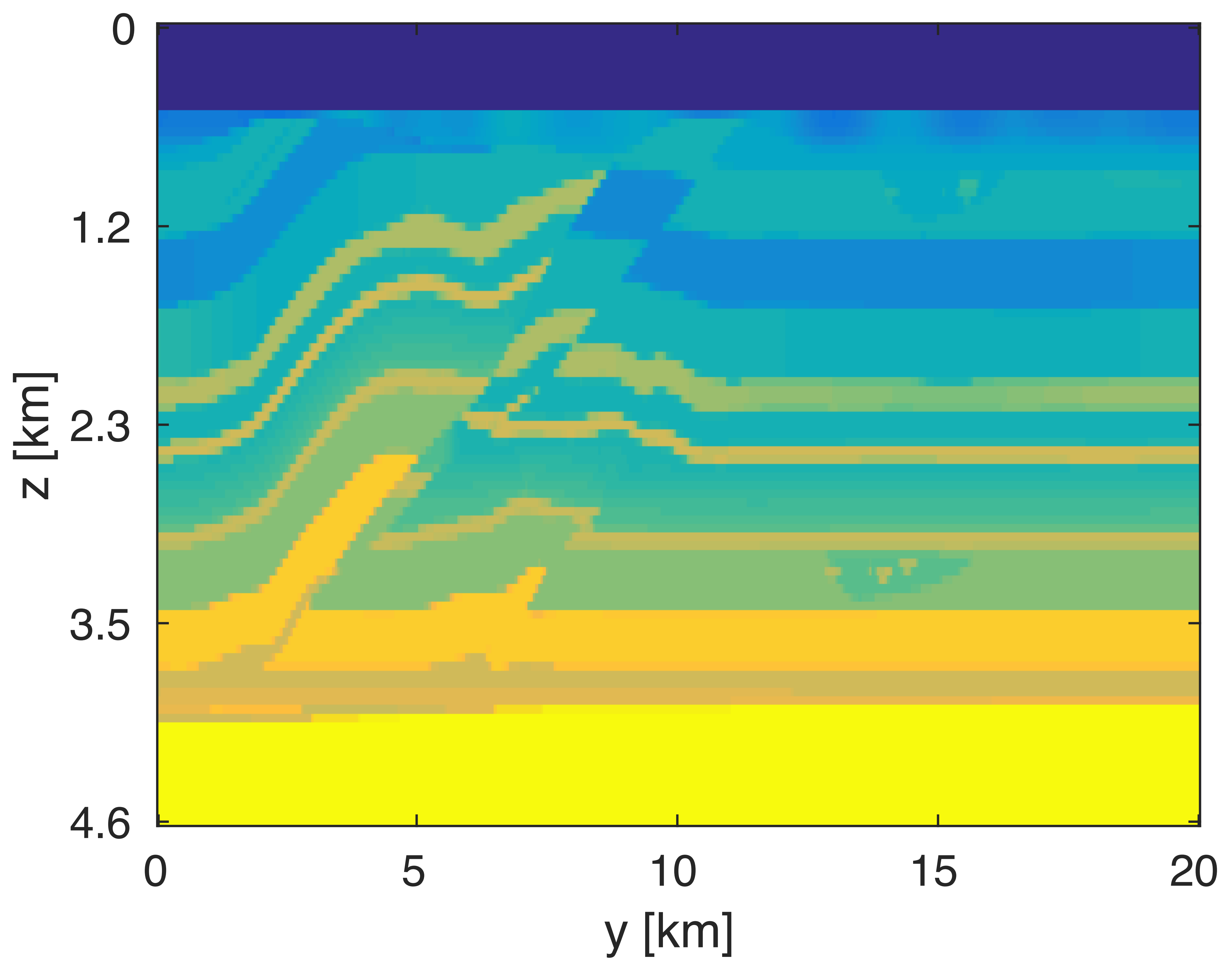


True model

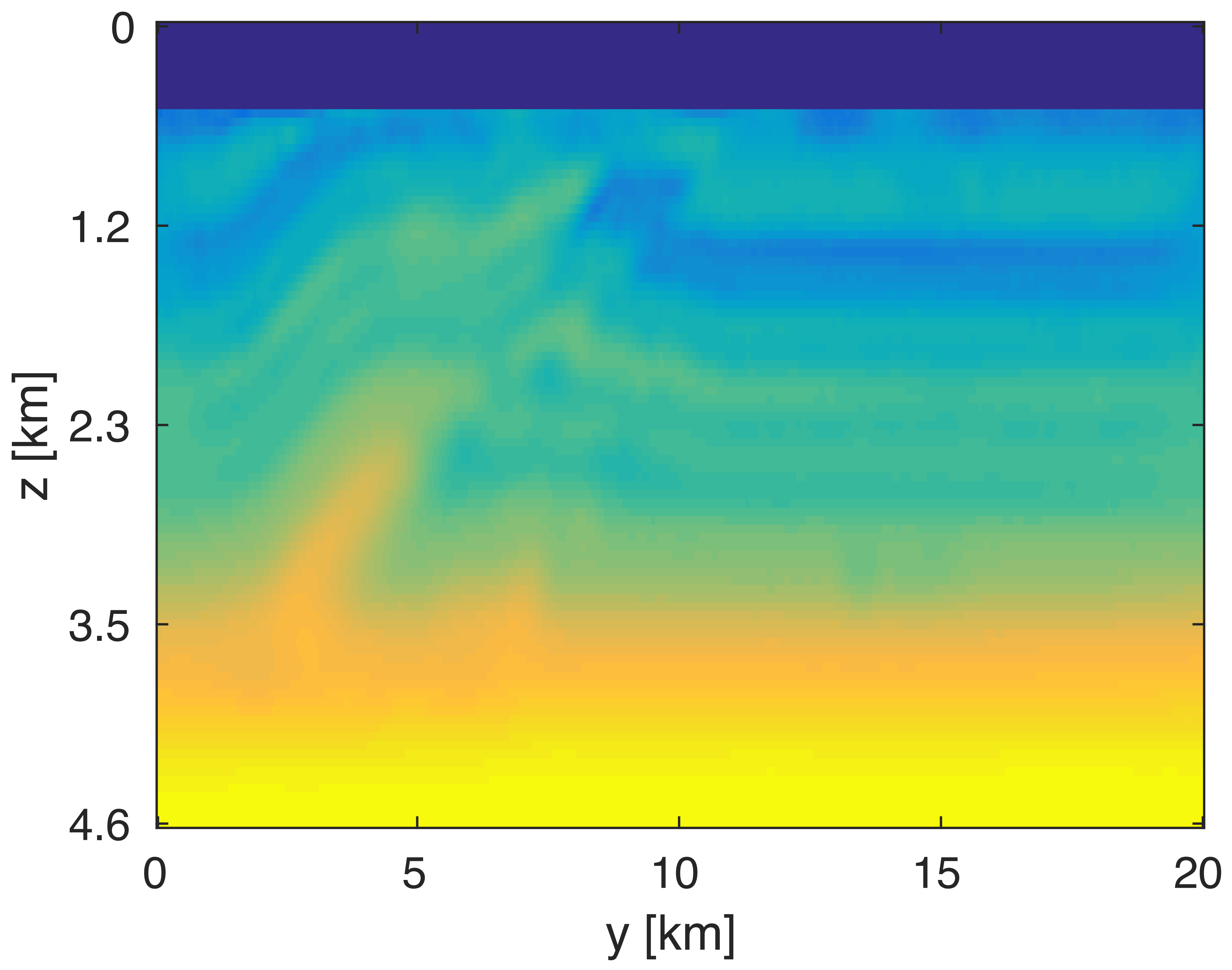


Initial model

x=12.5km slice

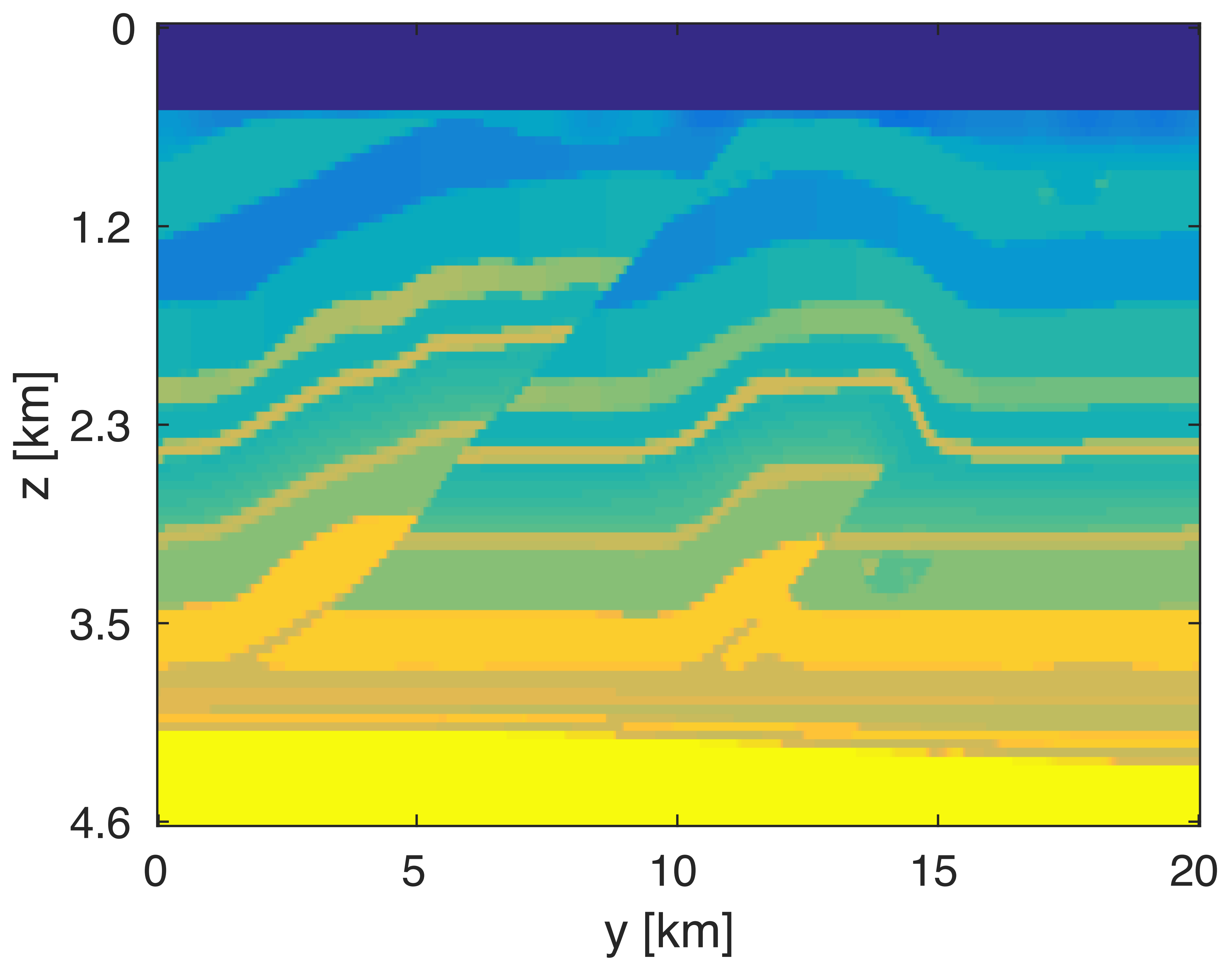


True model

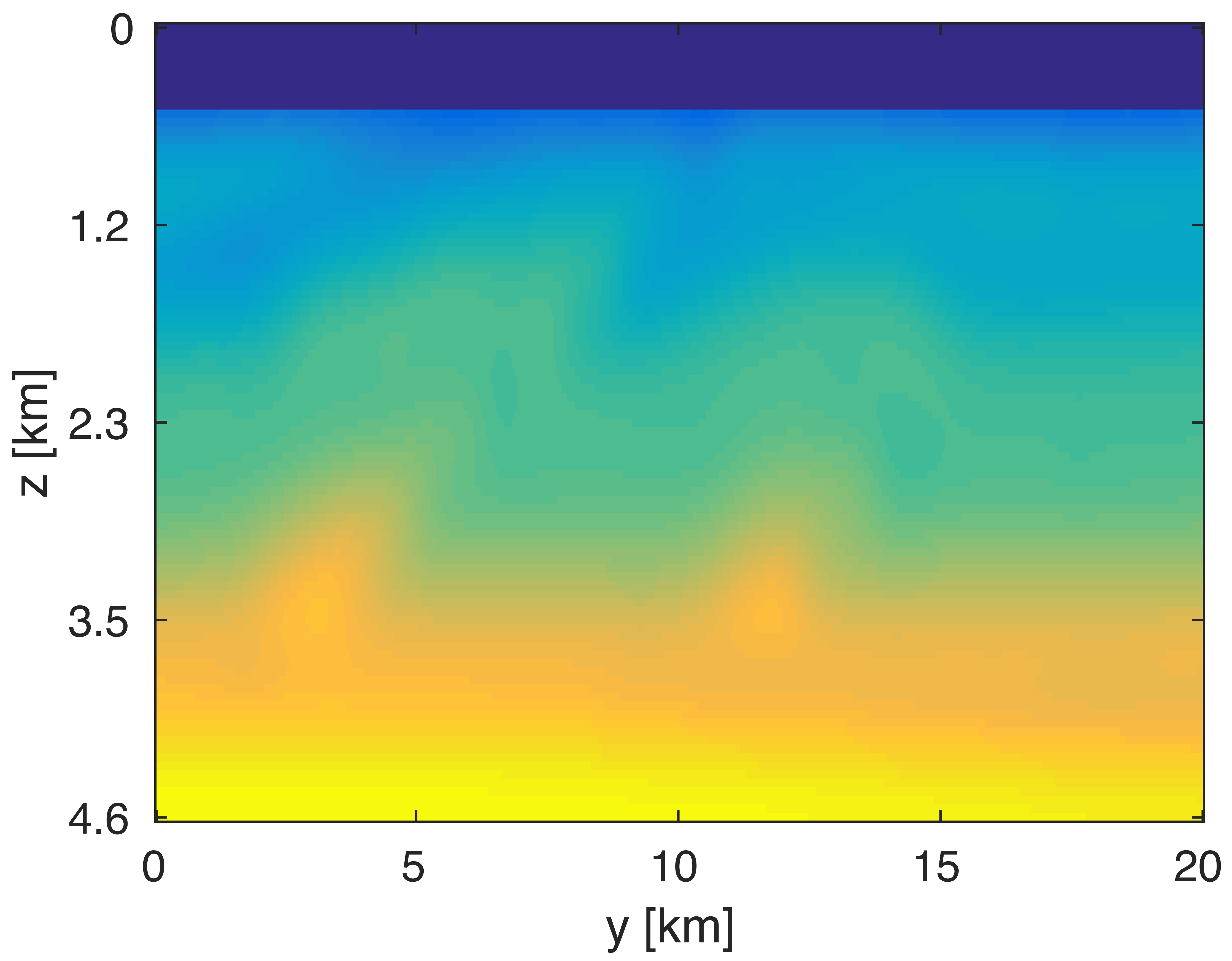


Stochastic LBFGS

x=17.5km slice

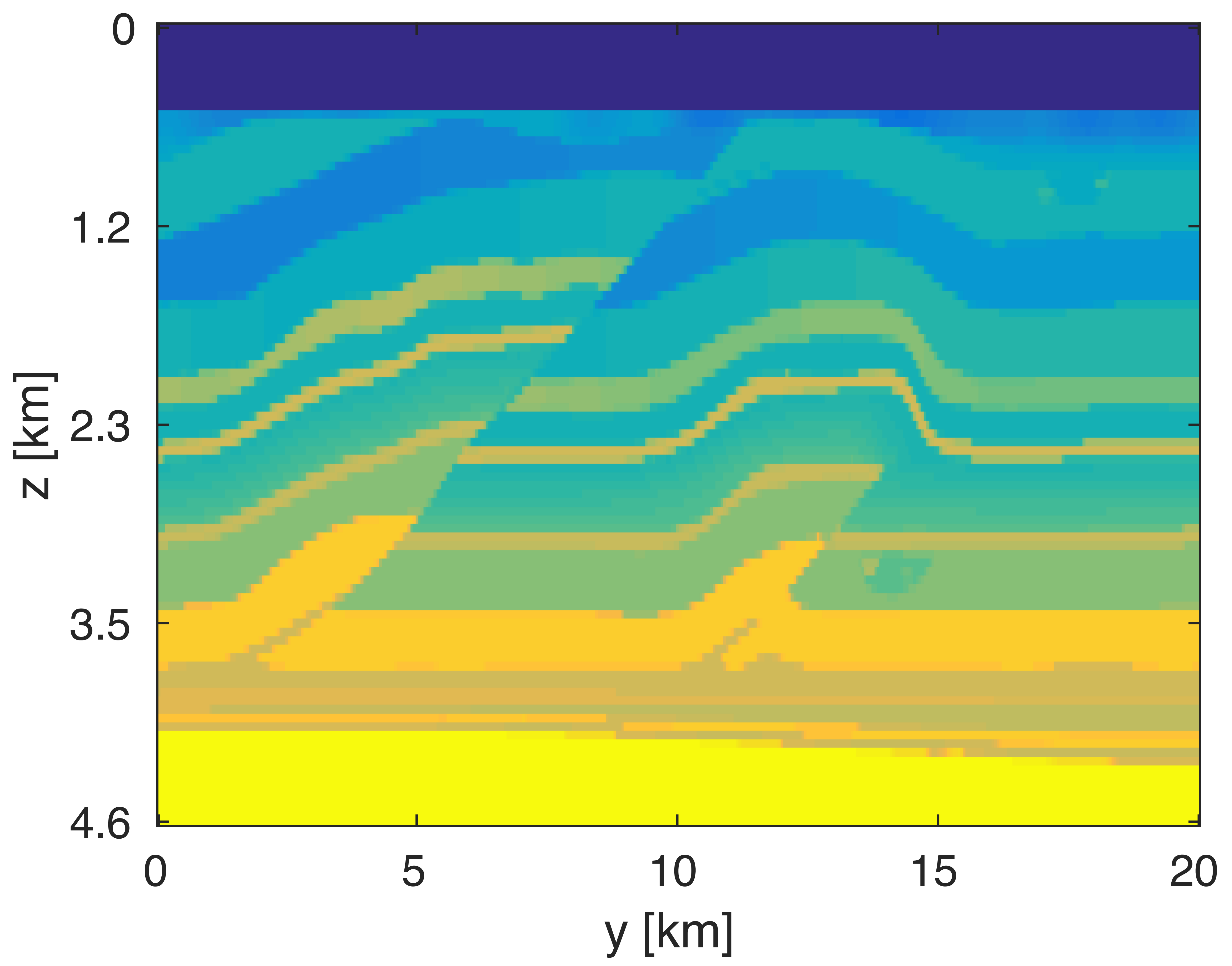


True model

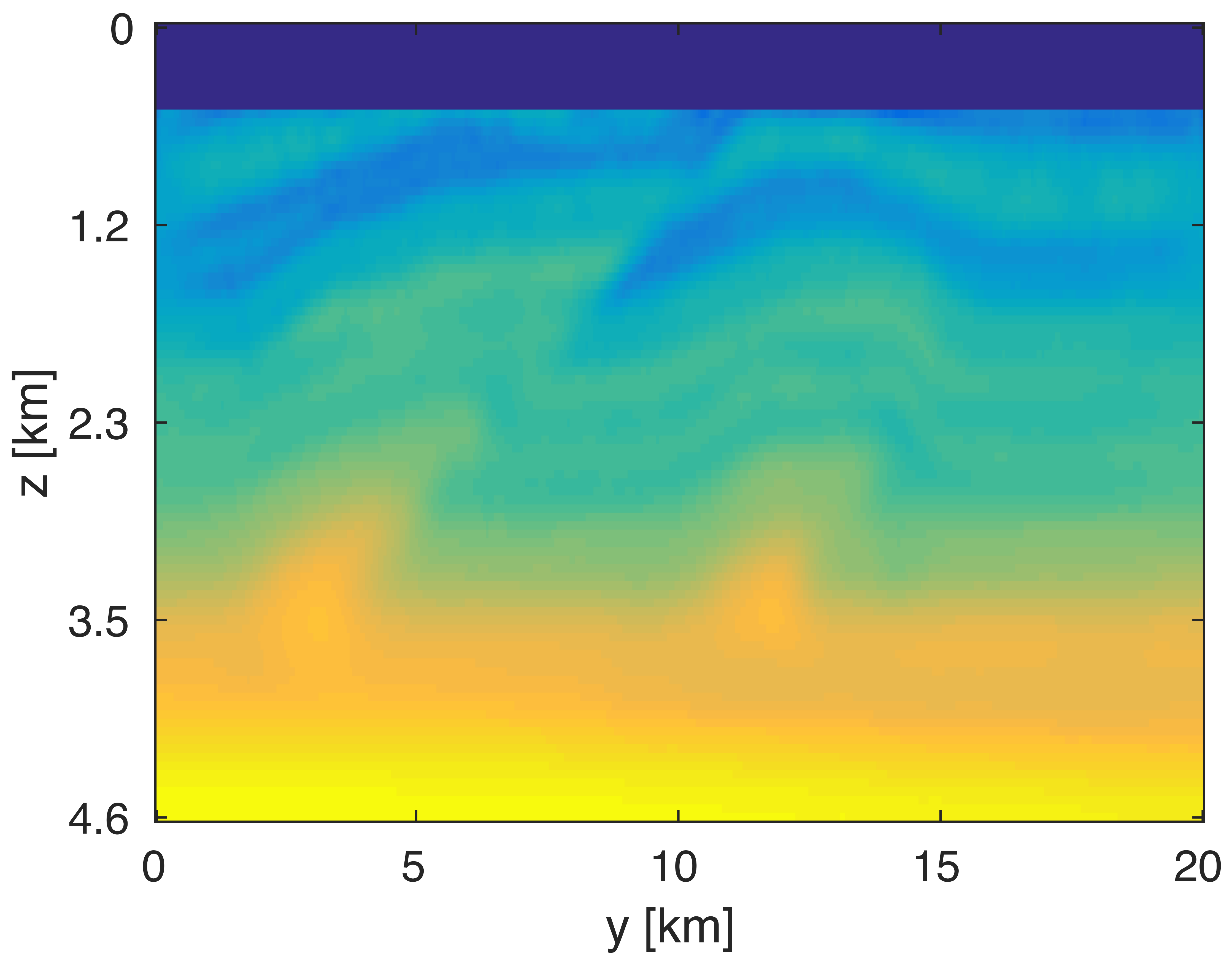


Initial model

x=17.5km slice

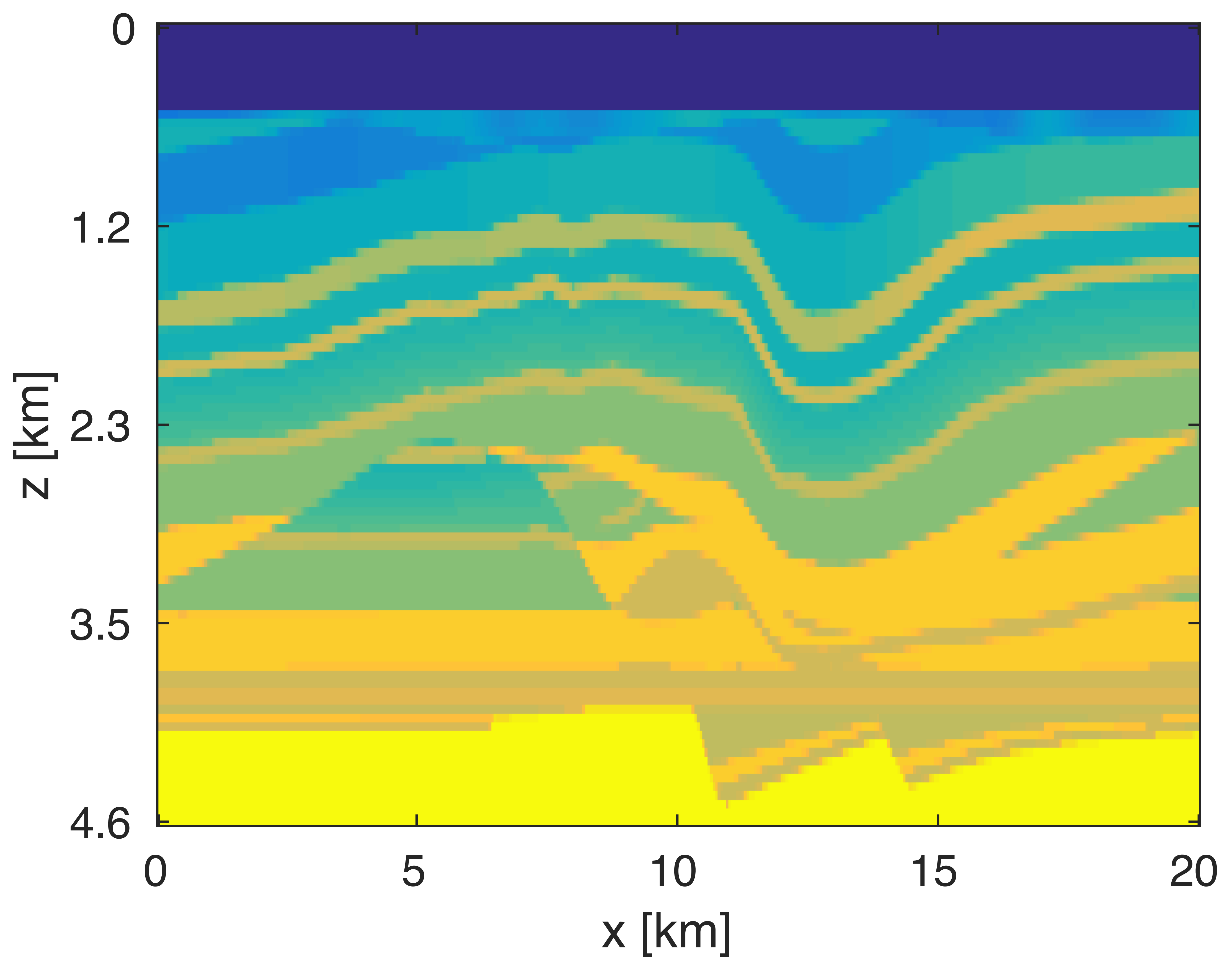


True model

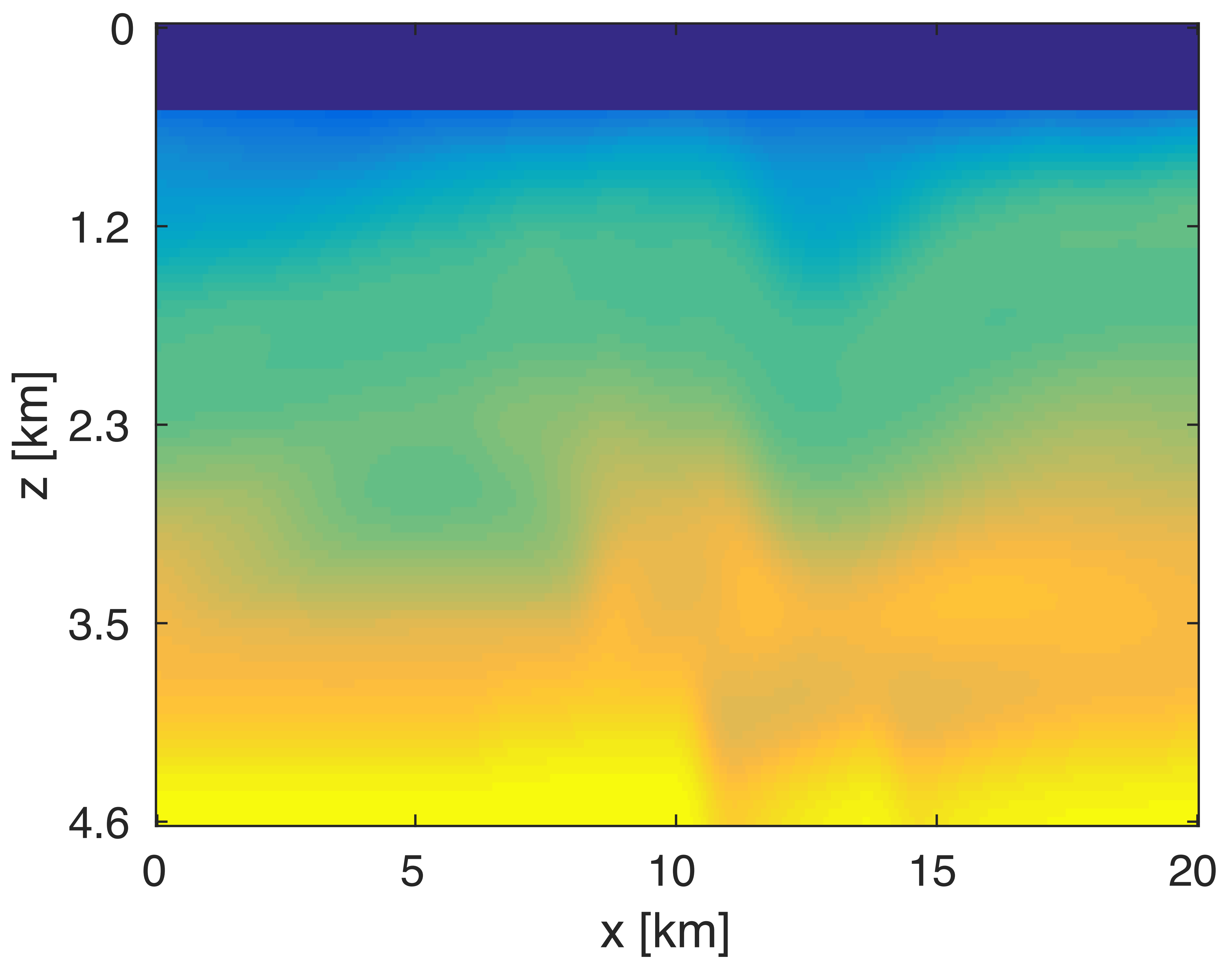


Stochastic LBFGS

y=5km slice

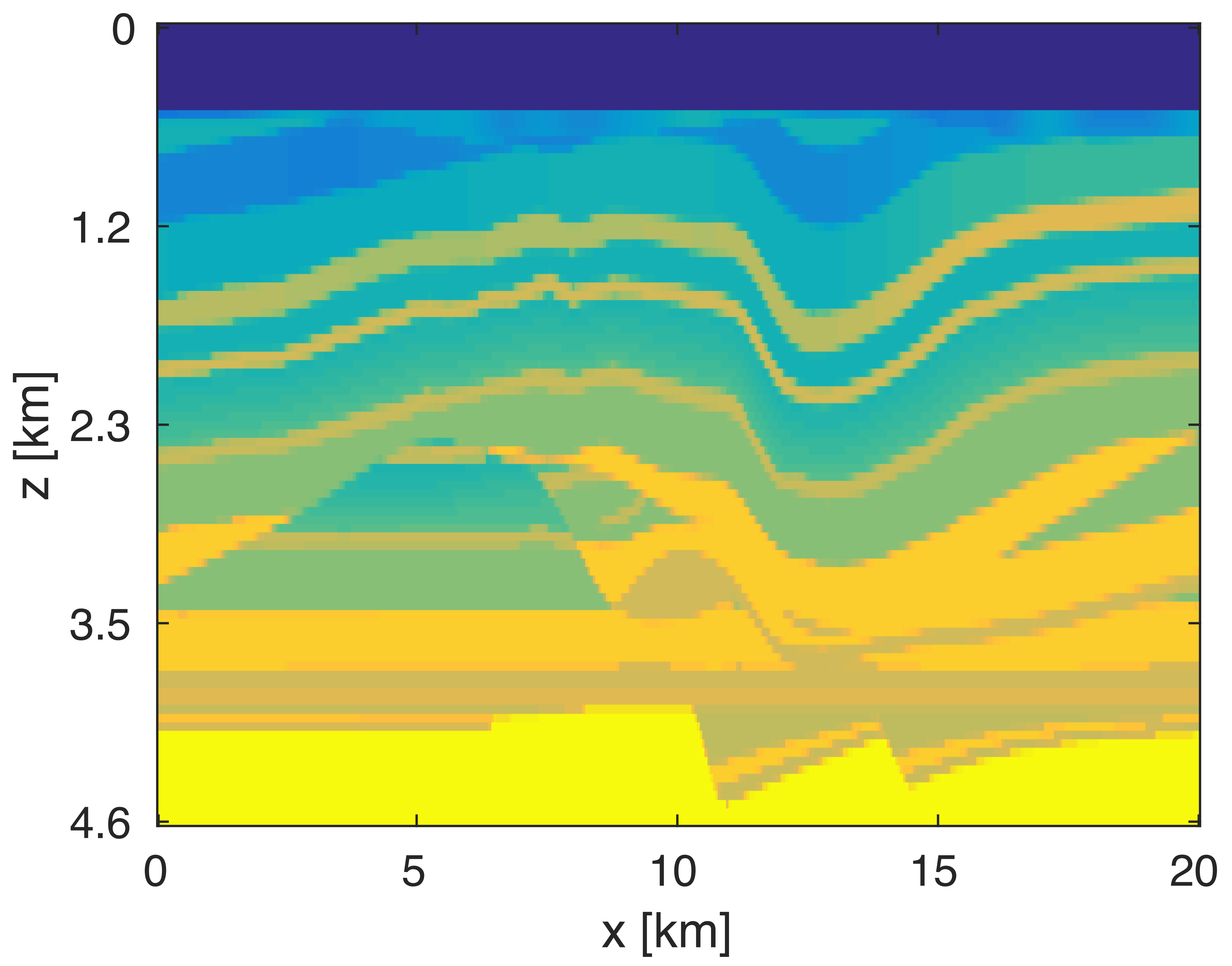


True model

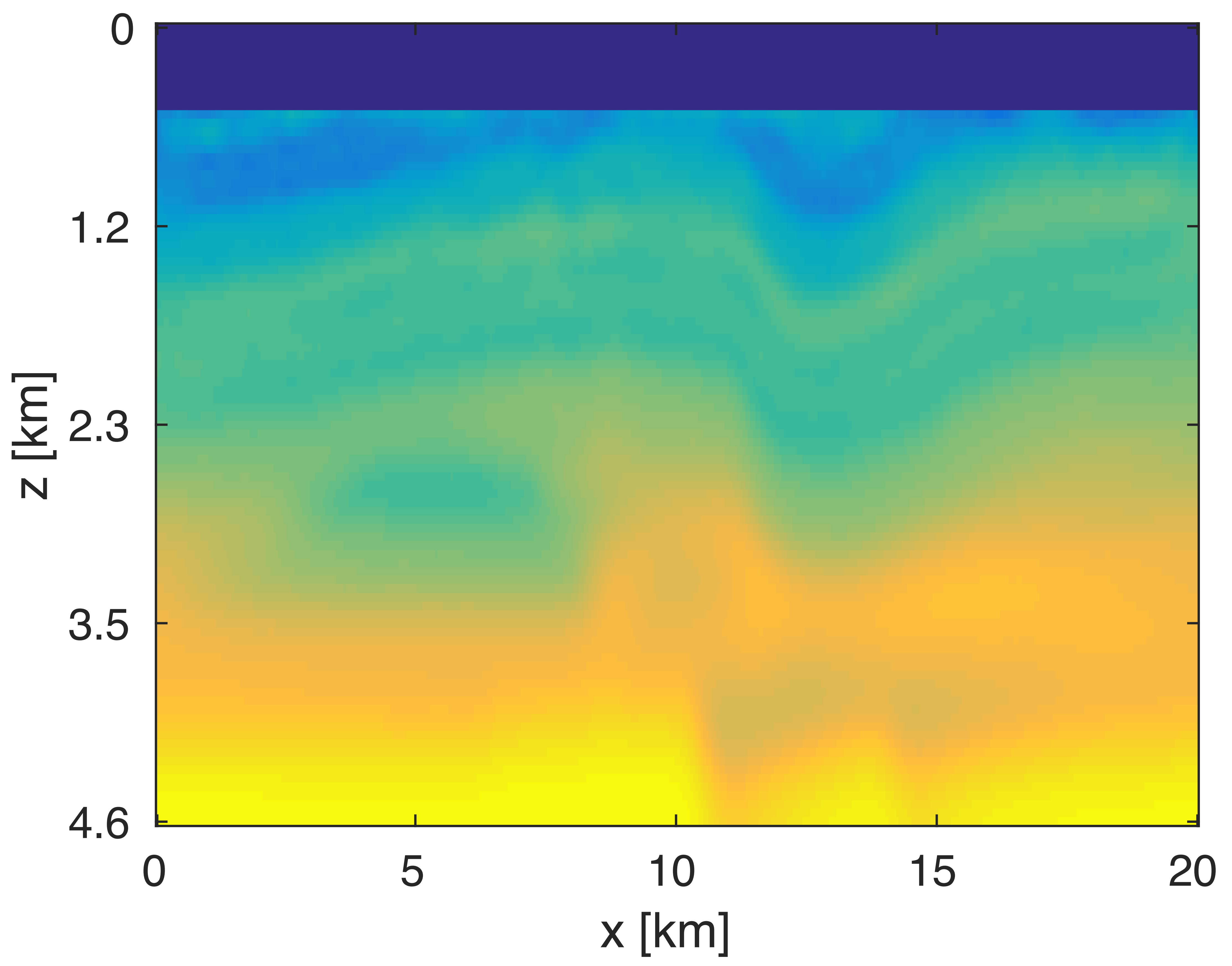


Initial model

y=5km slice

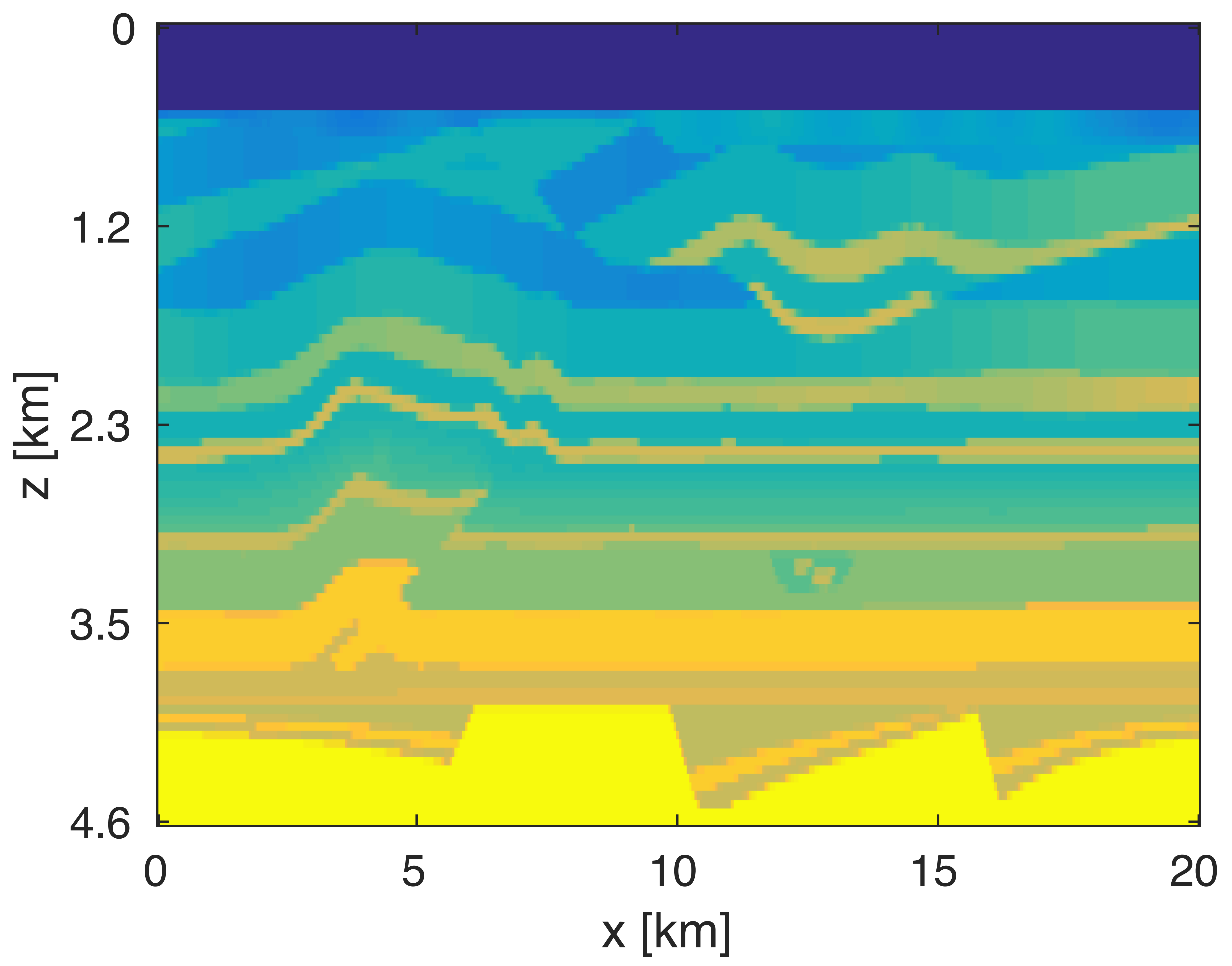


True model

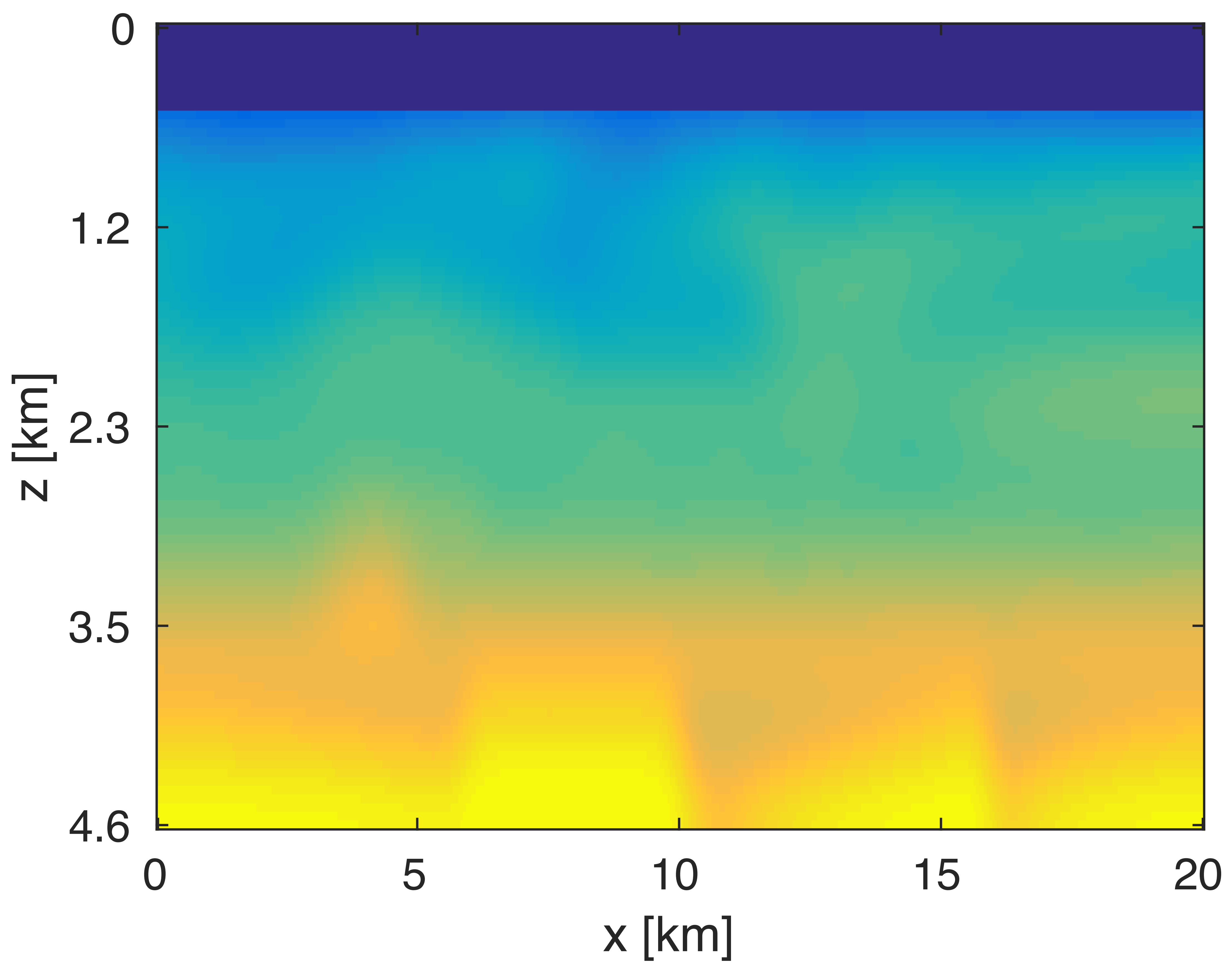


Stochastic LBFGS

y=10km slice

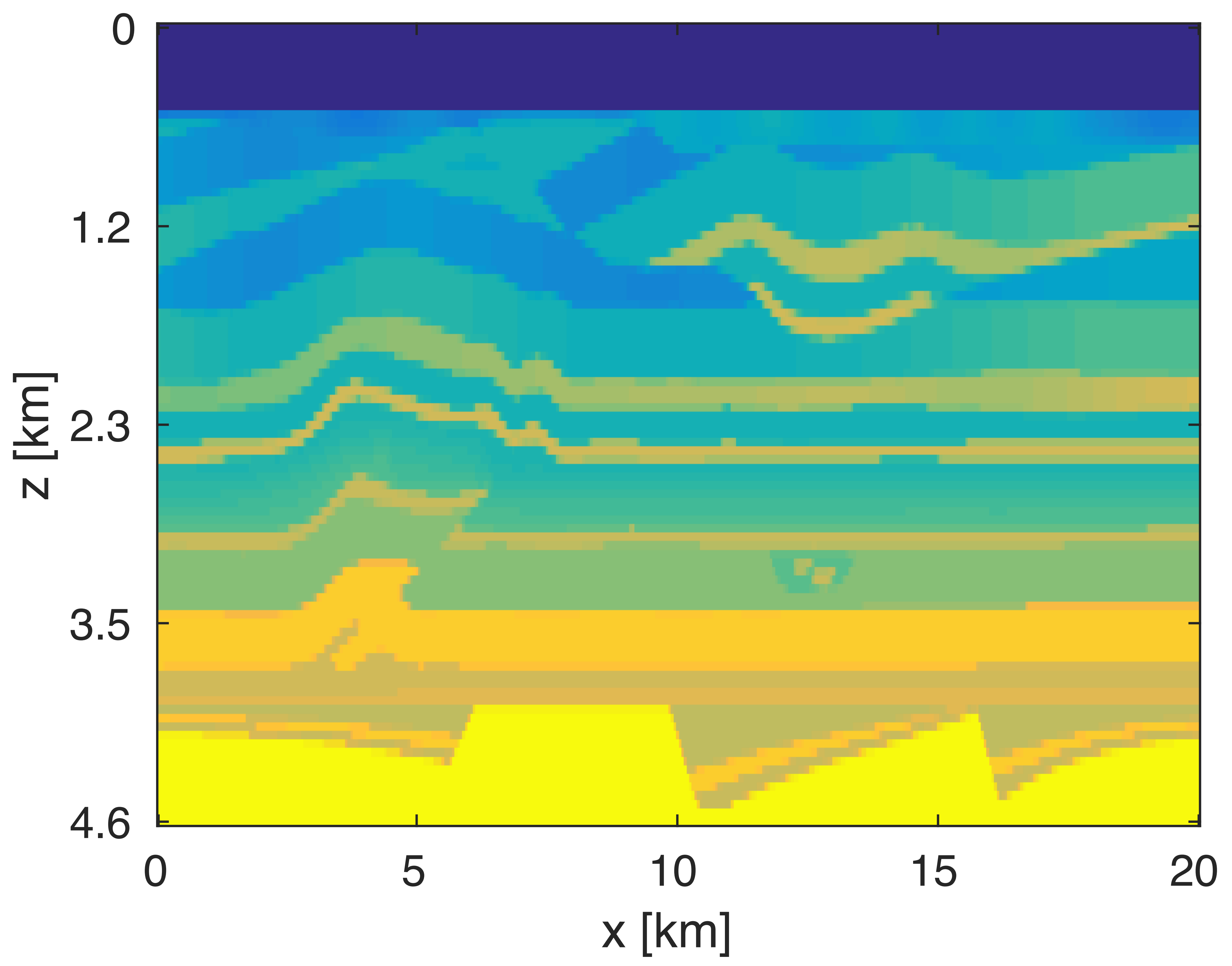


True model

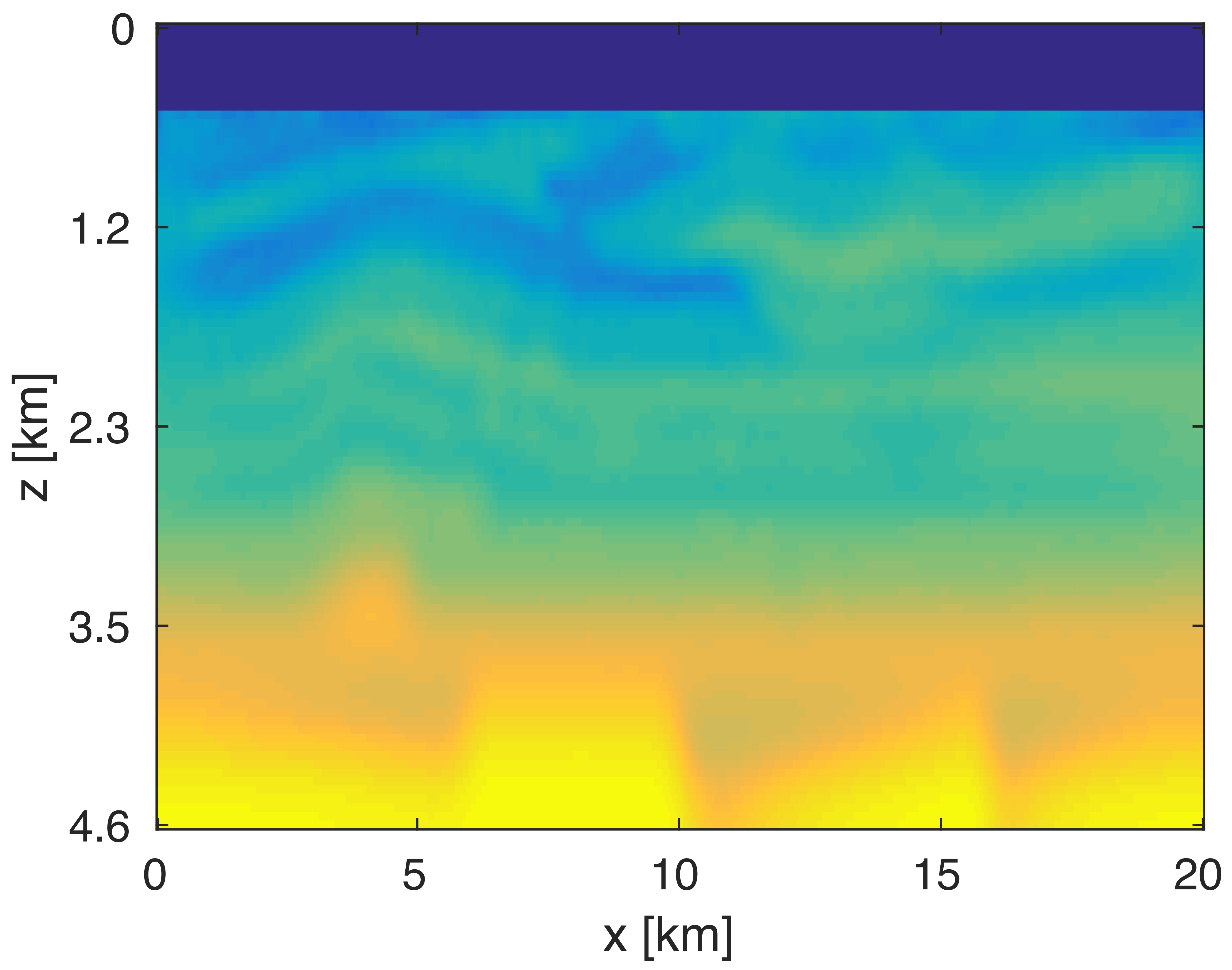


Initial model

y=10km slice



True model



Stochastic LBFGS

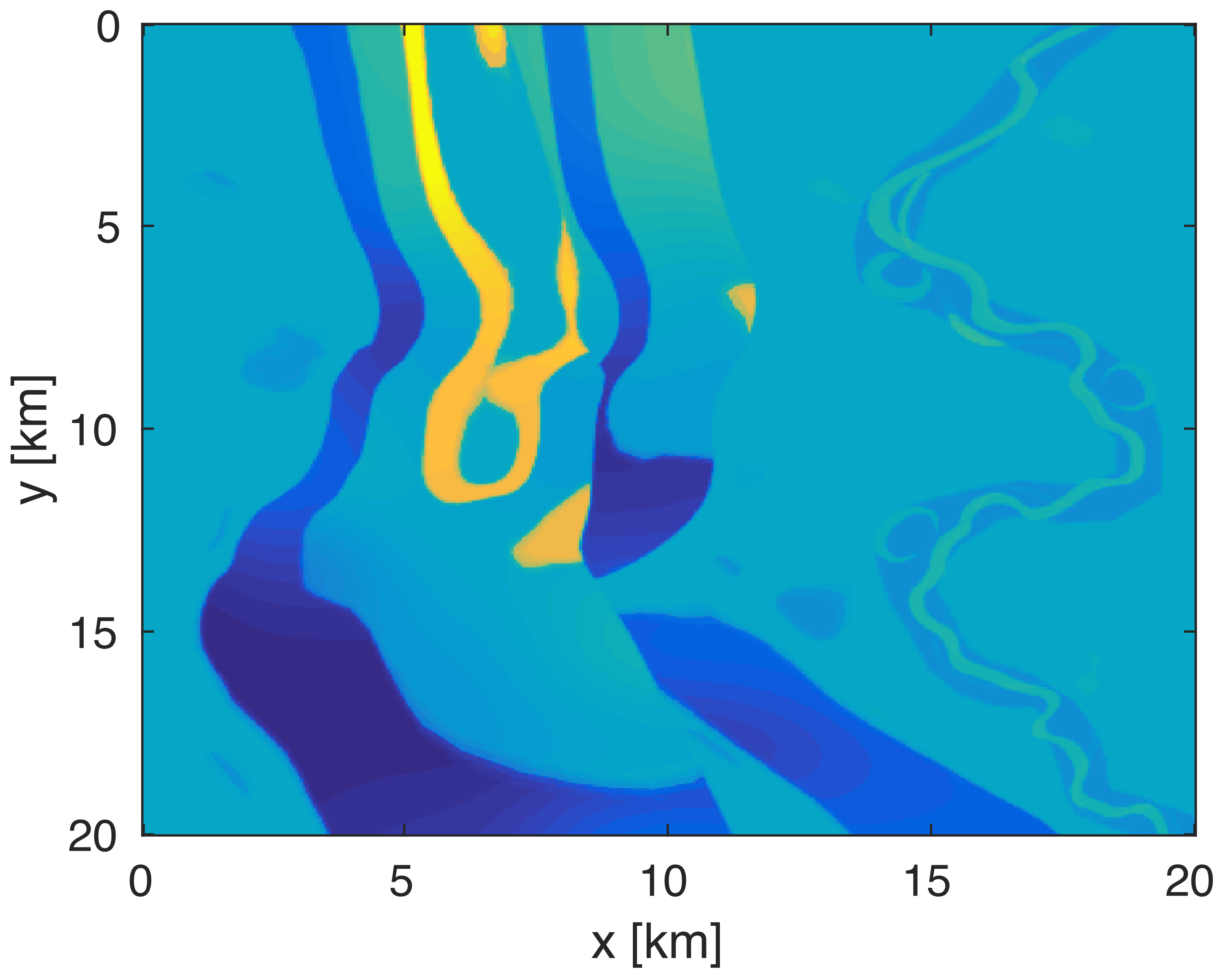
3D Overthrust Model

Same model as before, no water layer (SEG abstract results)

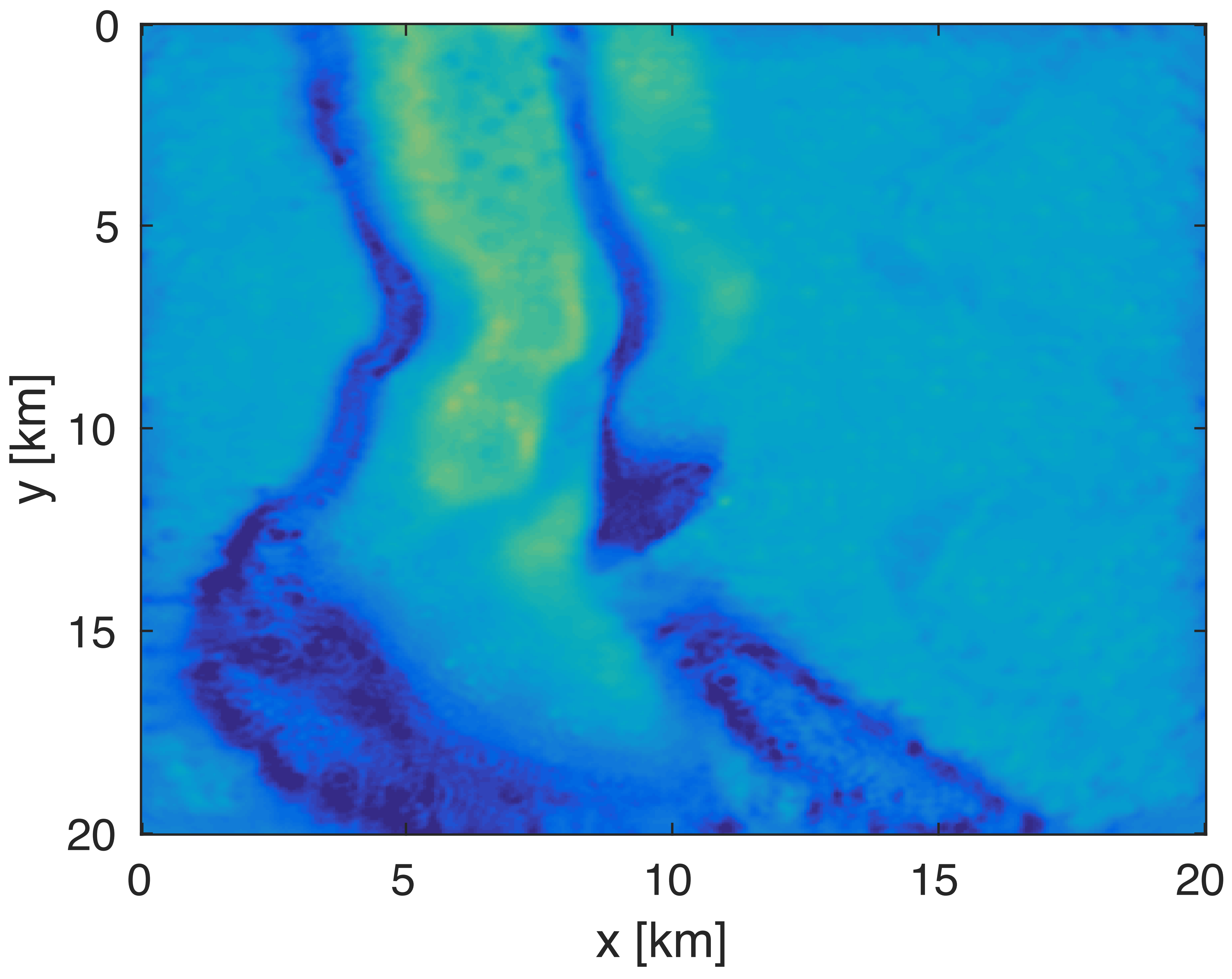
3Hz - 8Hz, inverted one frequency at a time

Compare the stochastic approach to the full-data approach (equivalent # of PDEs solved)

z=500m slice

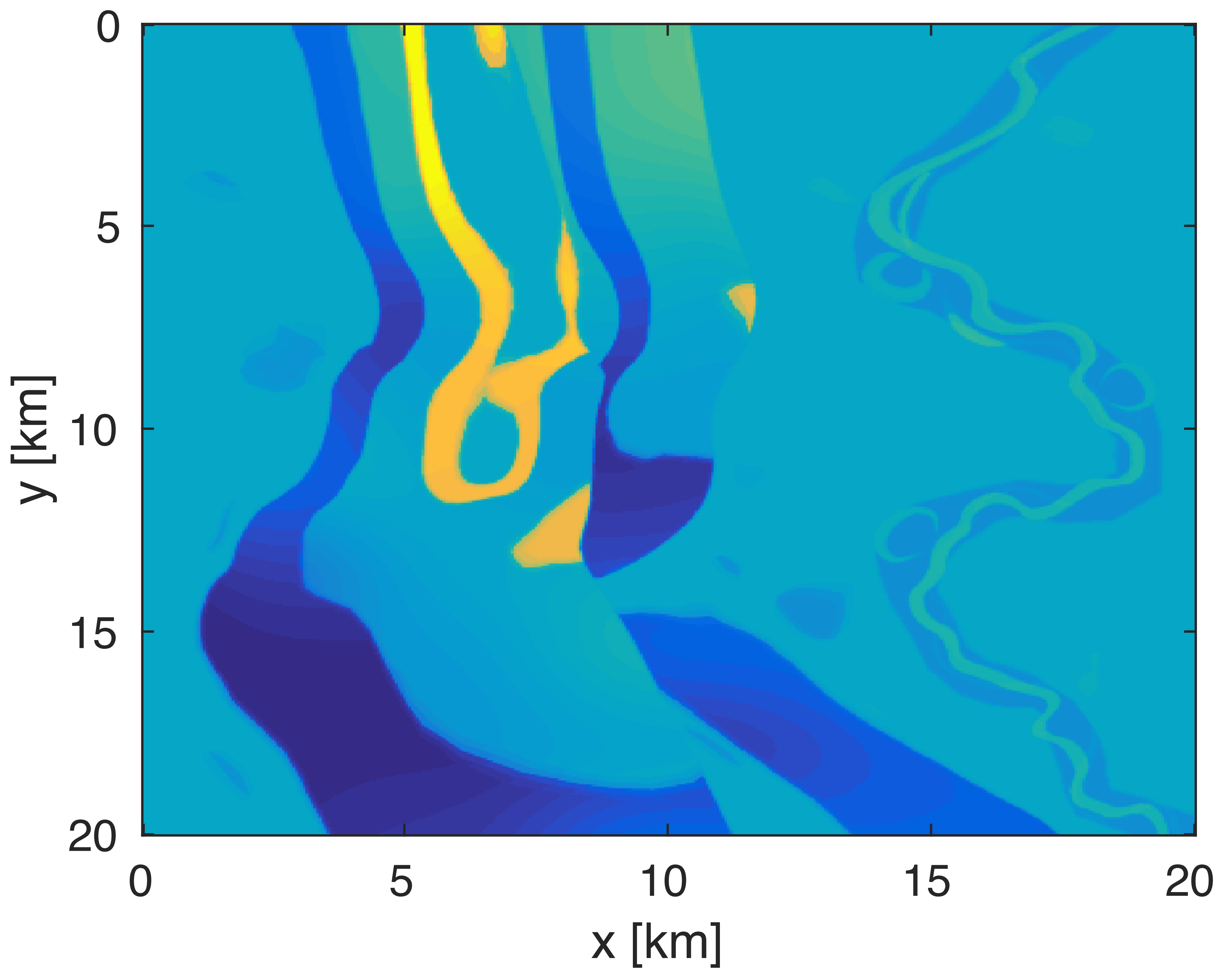


True model

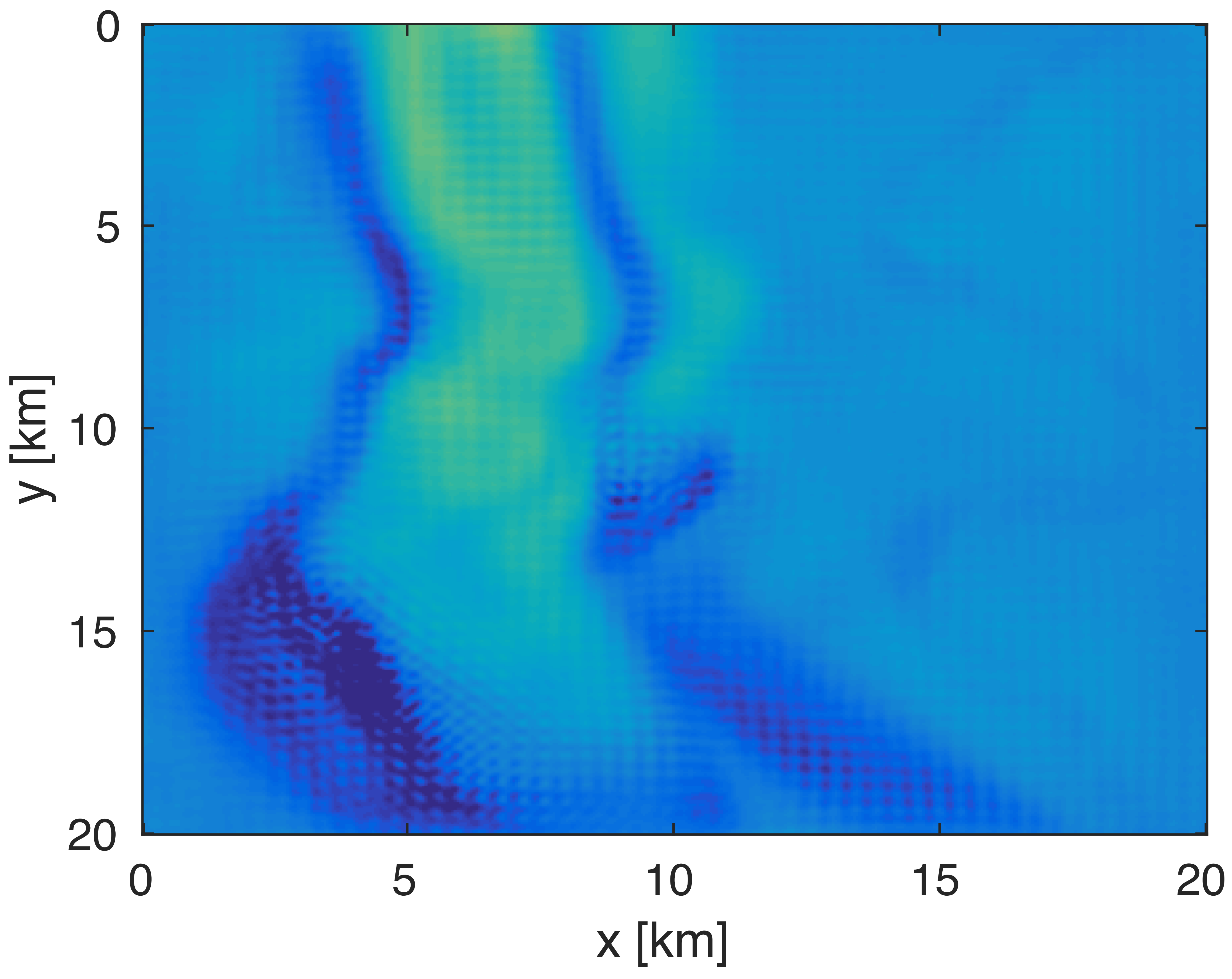


Stochastic LBFGS

z=500m slice

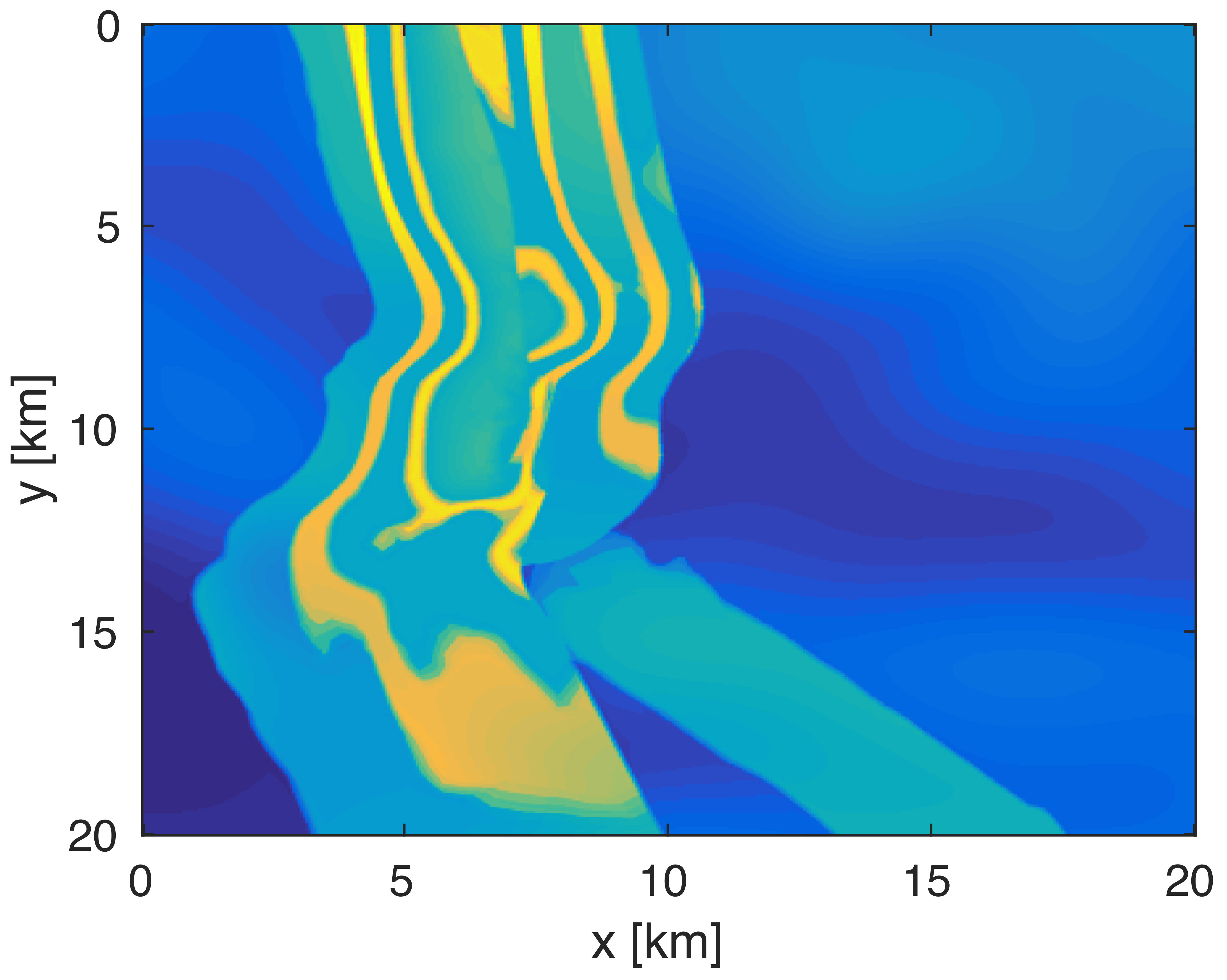


True model

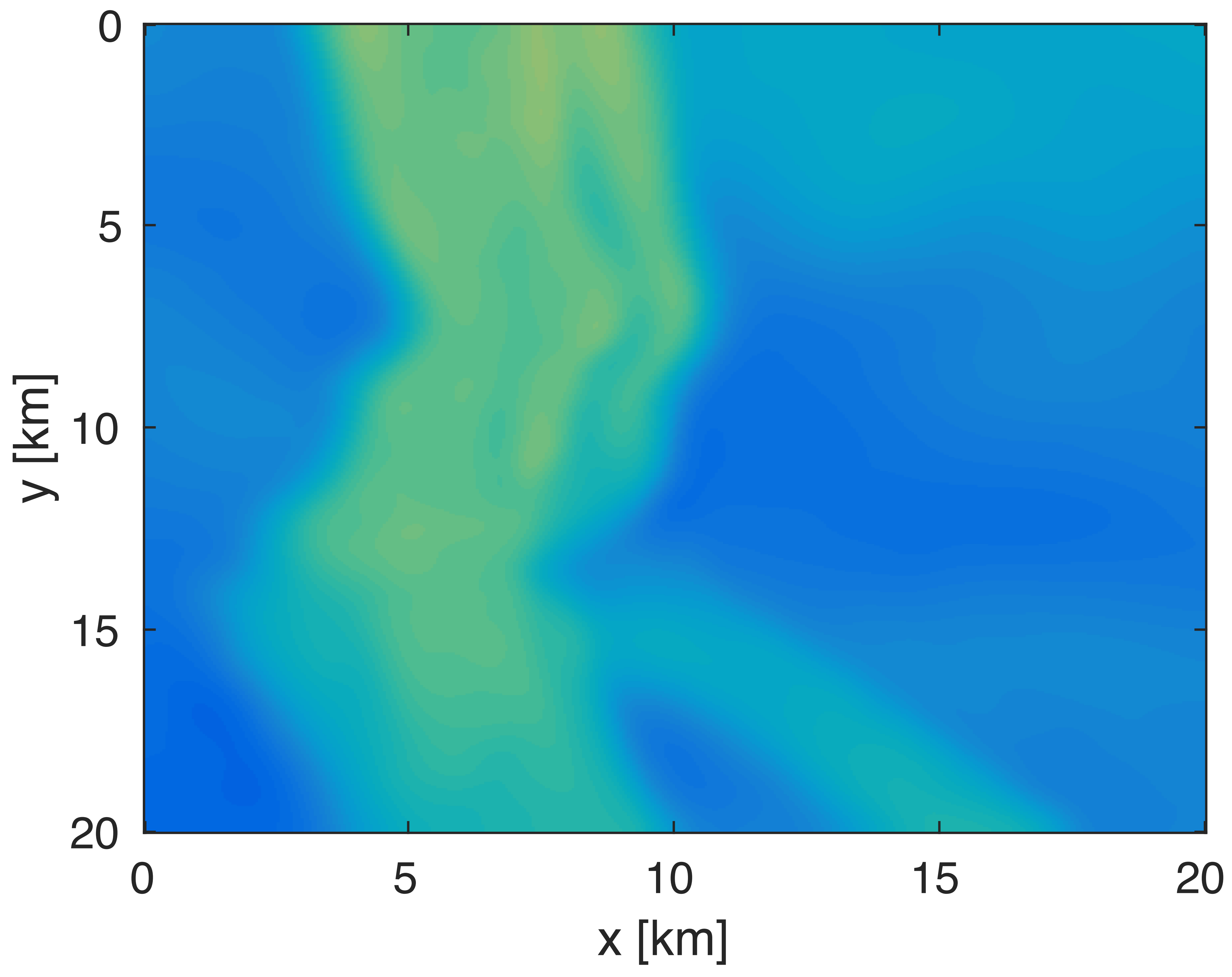


Full data

z=1000m slice

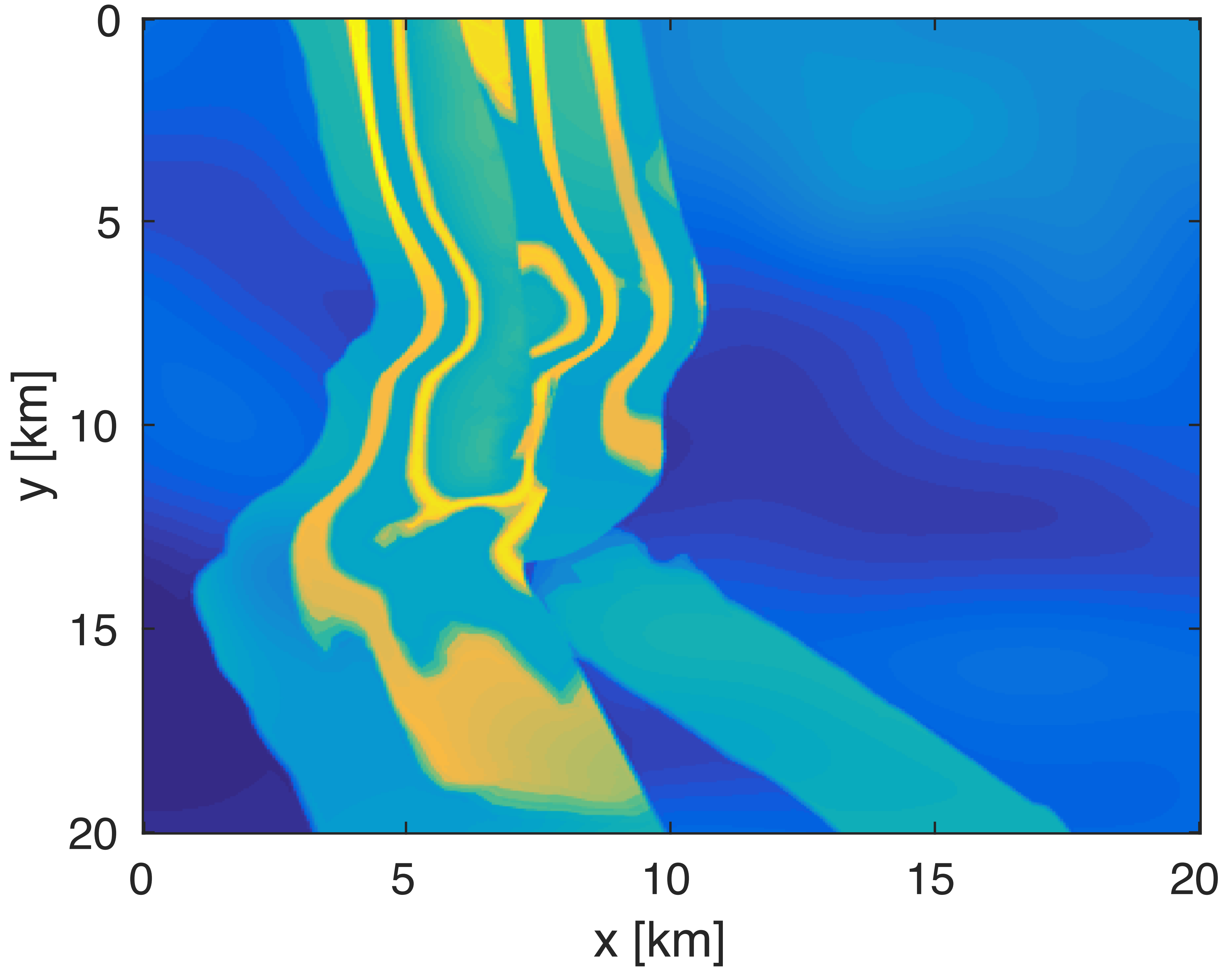


True model

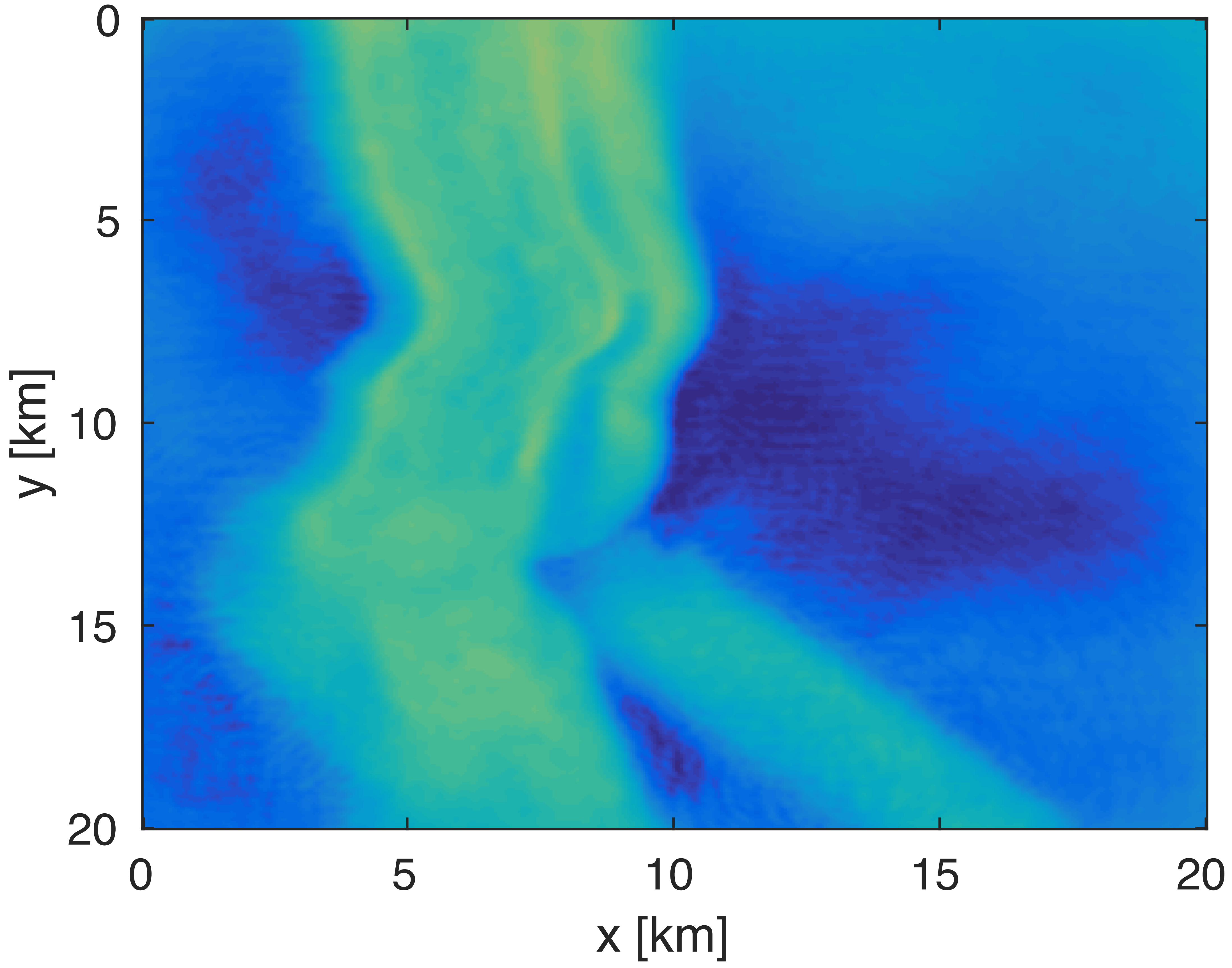


Initial Model

z=1000m slice

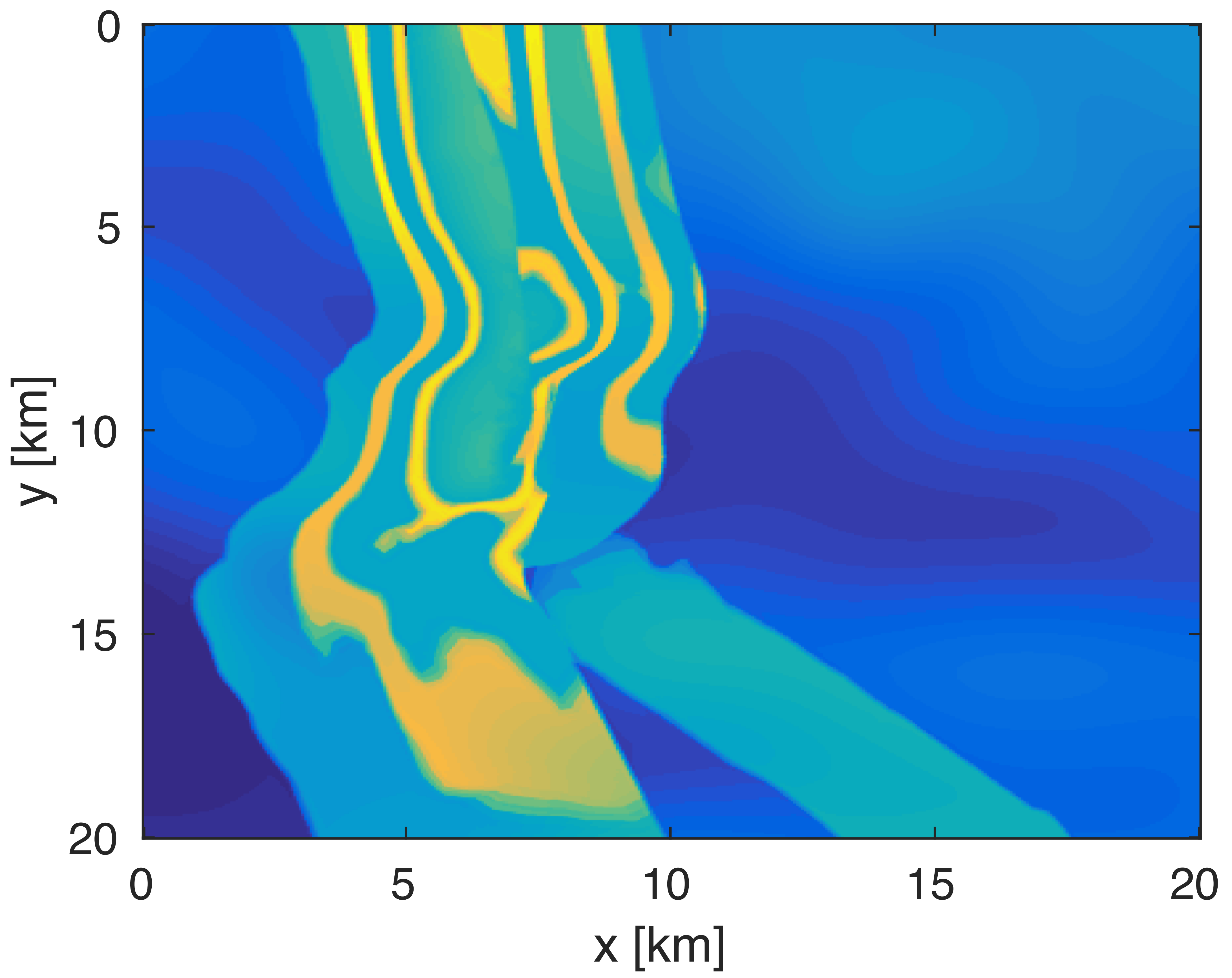


True model

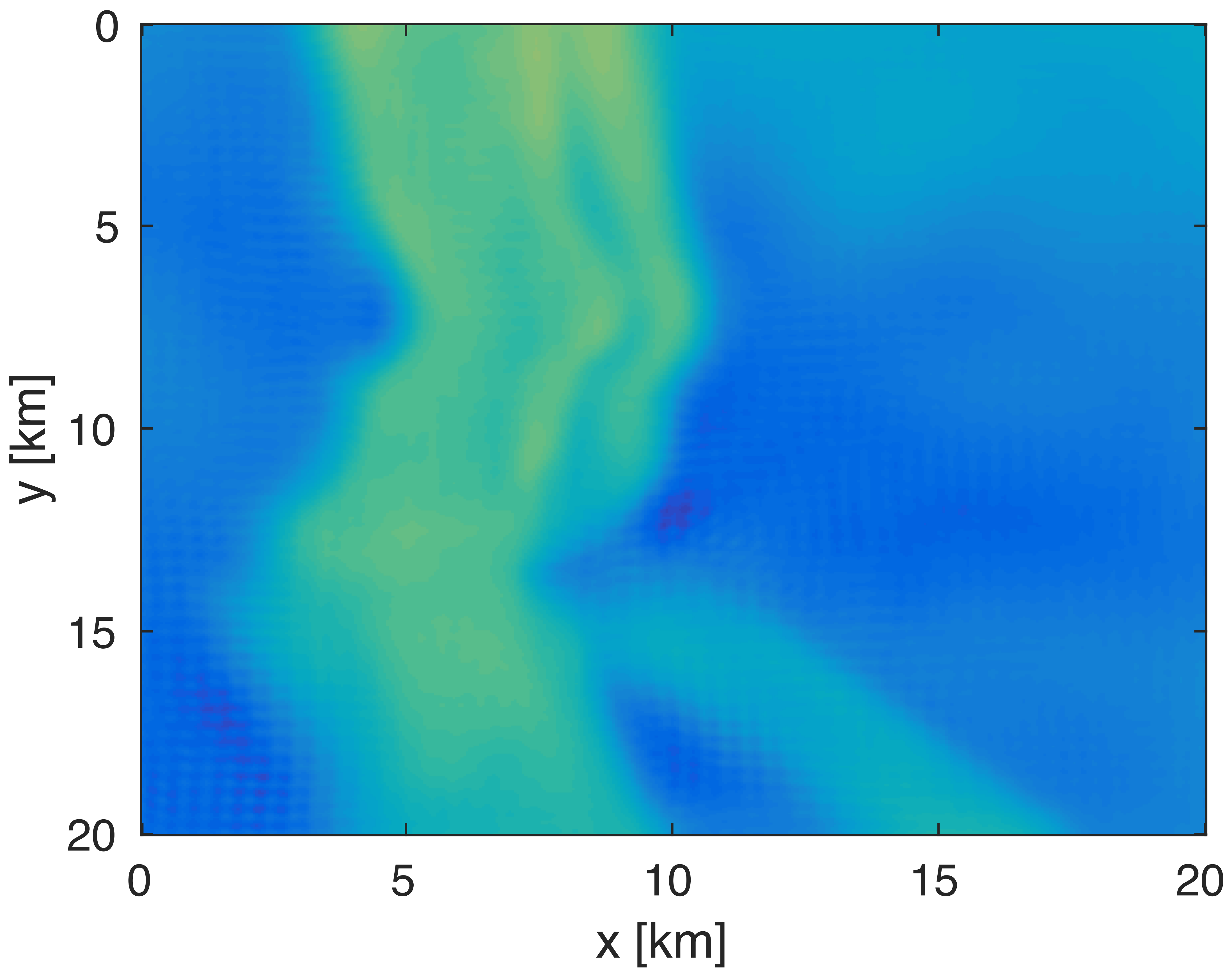


Stochastic LBFGS

z=1000m slice

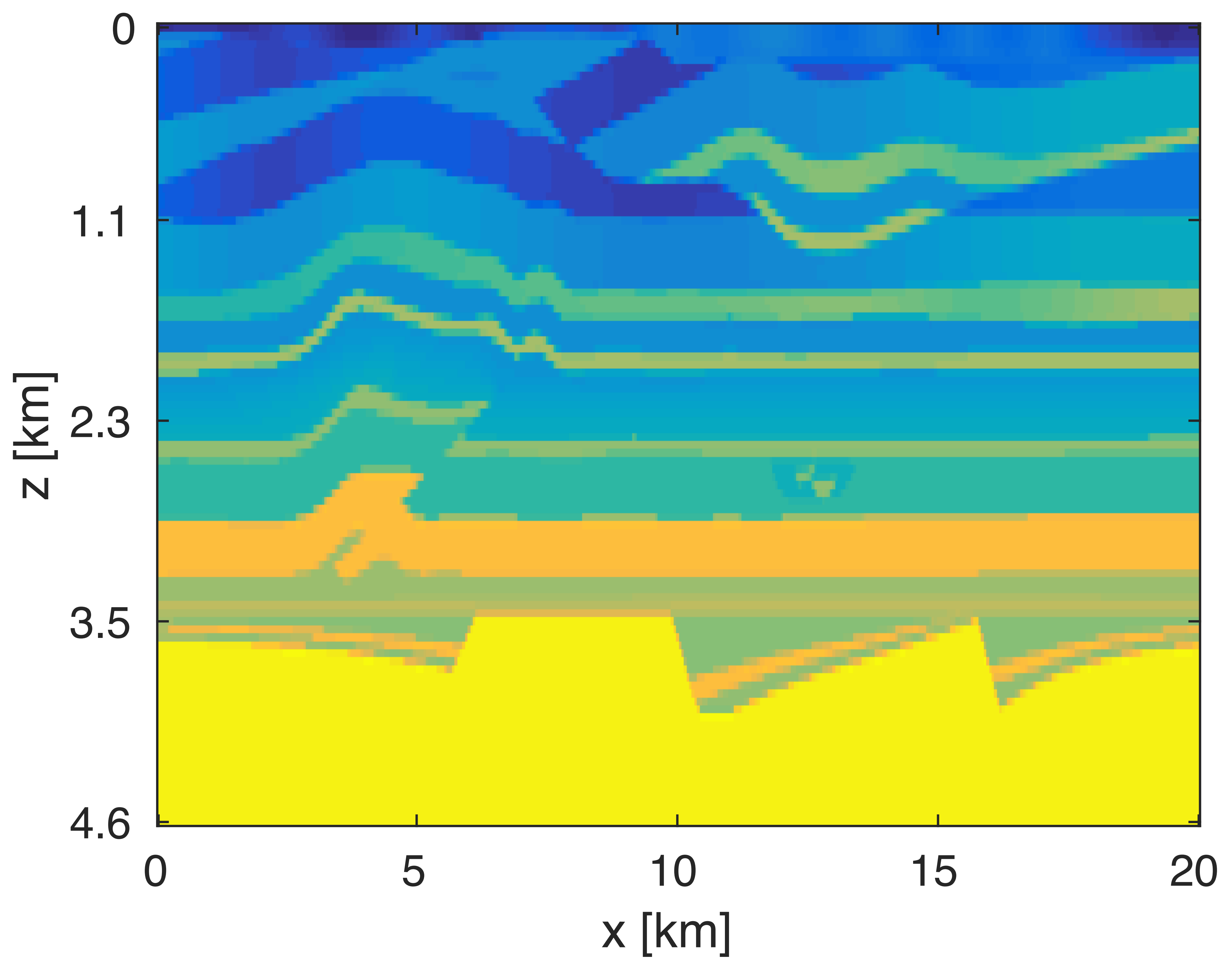


True model

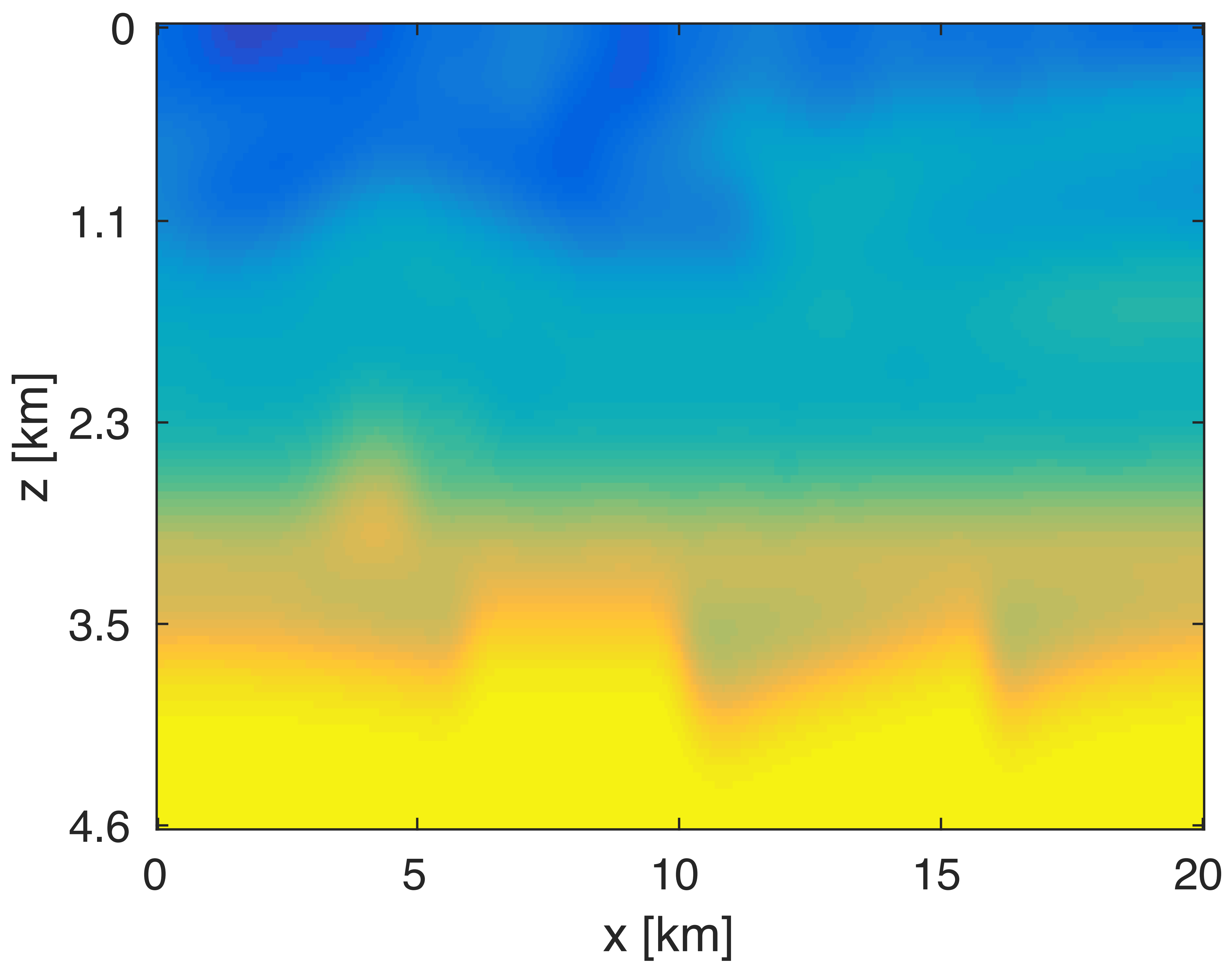


Full data

y=10000m slice

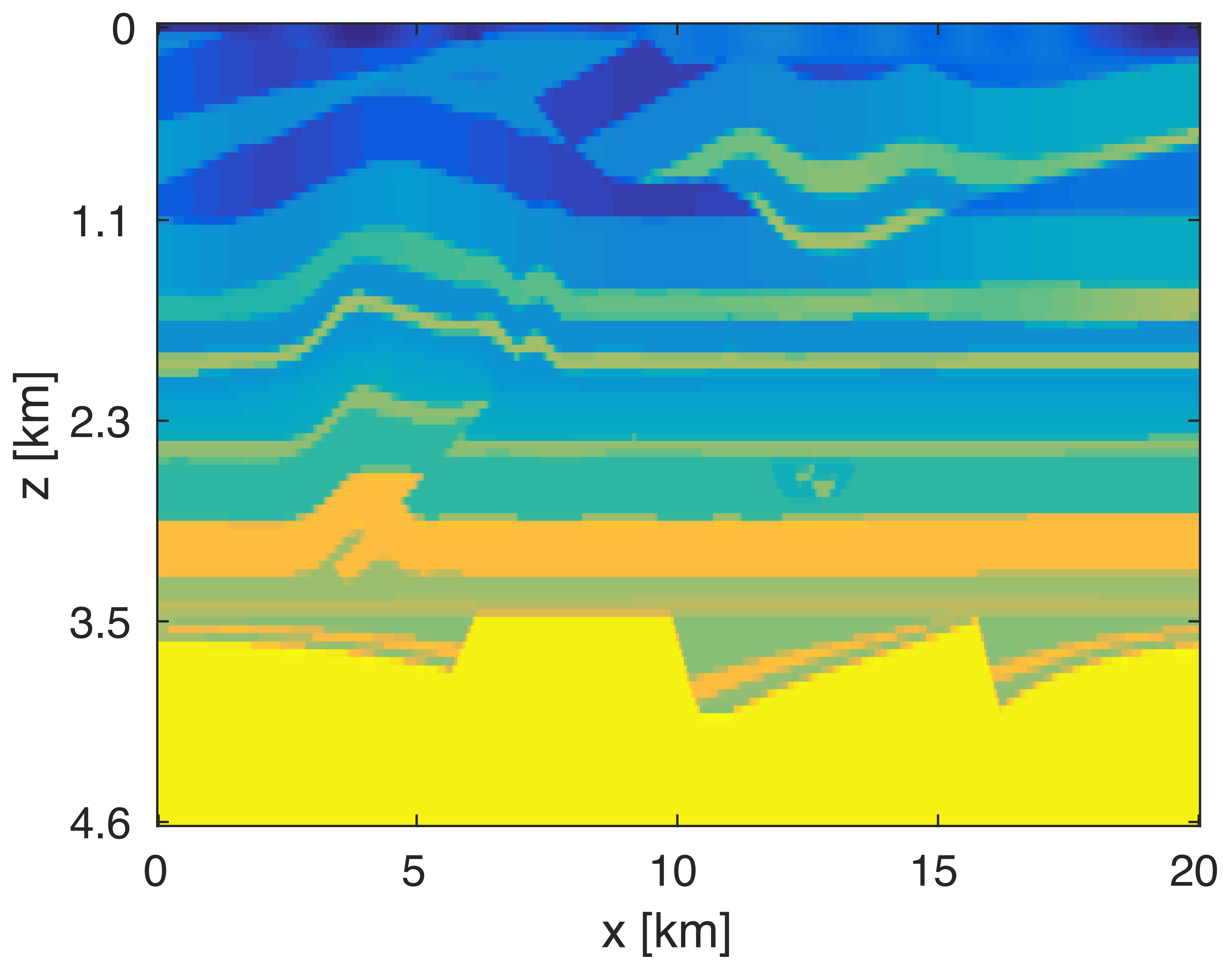


True model

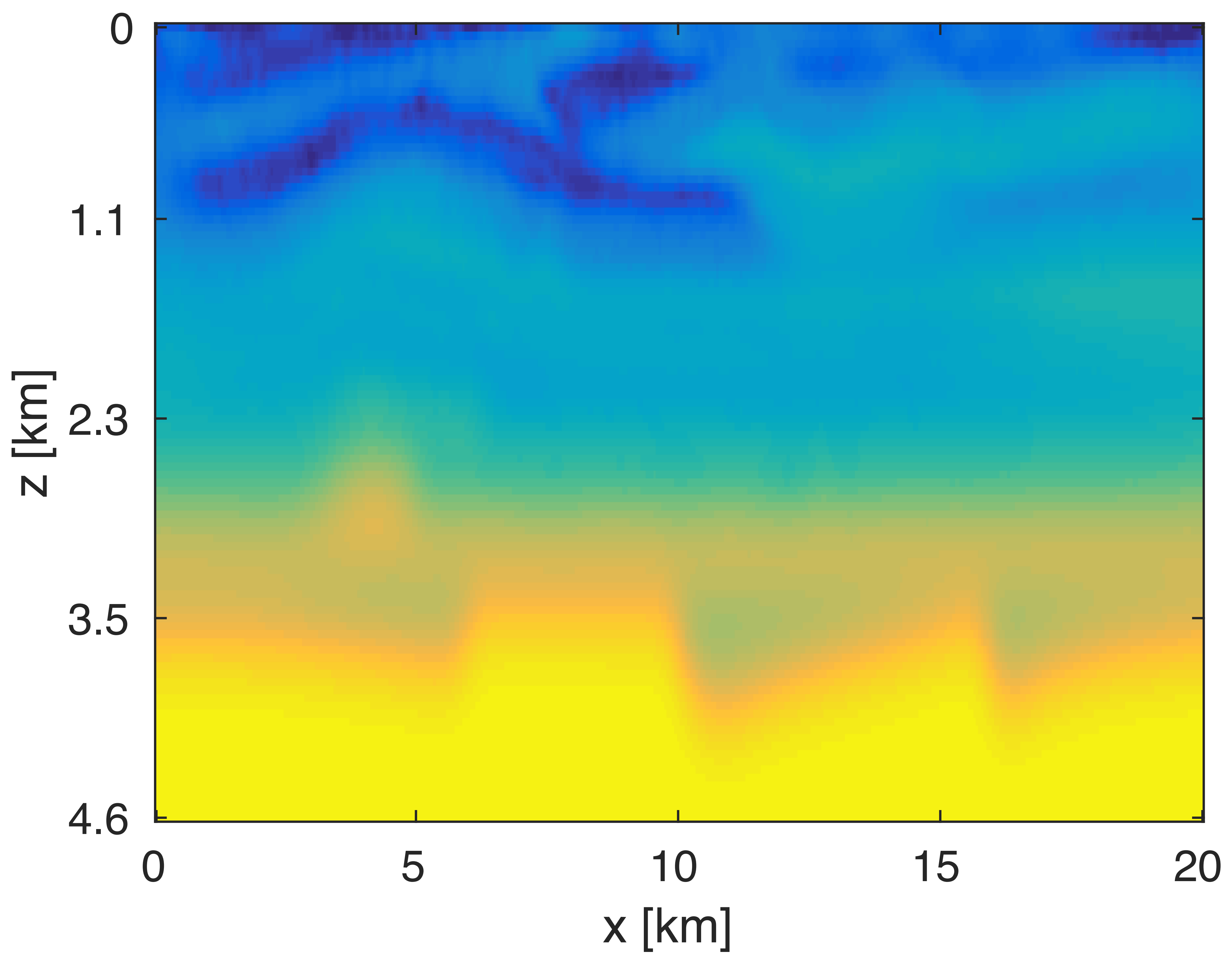


Initial model

y=10000m slice

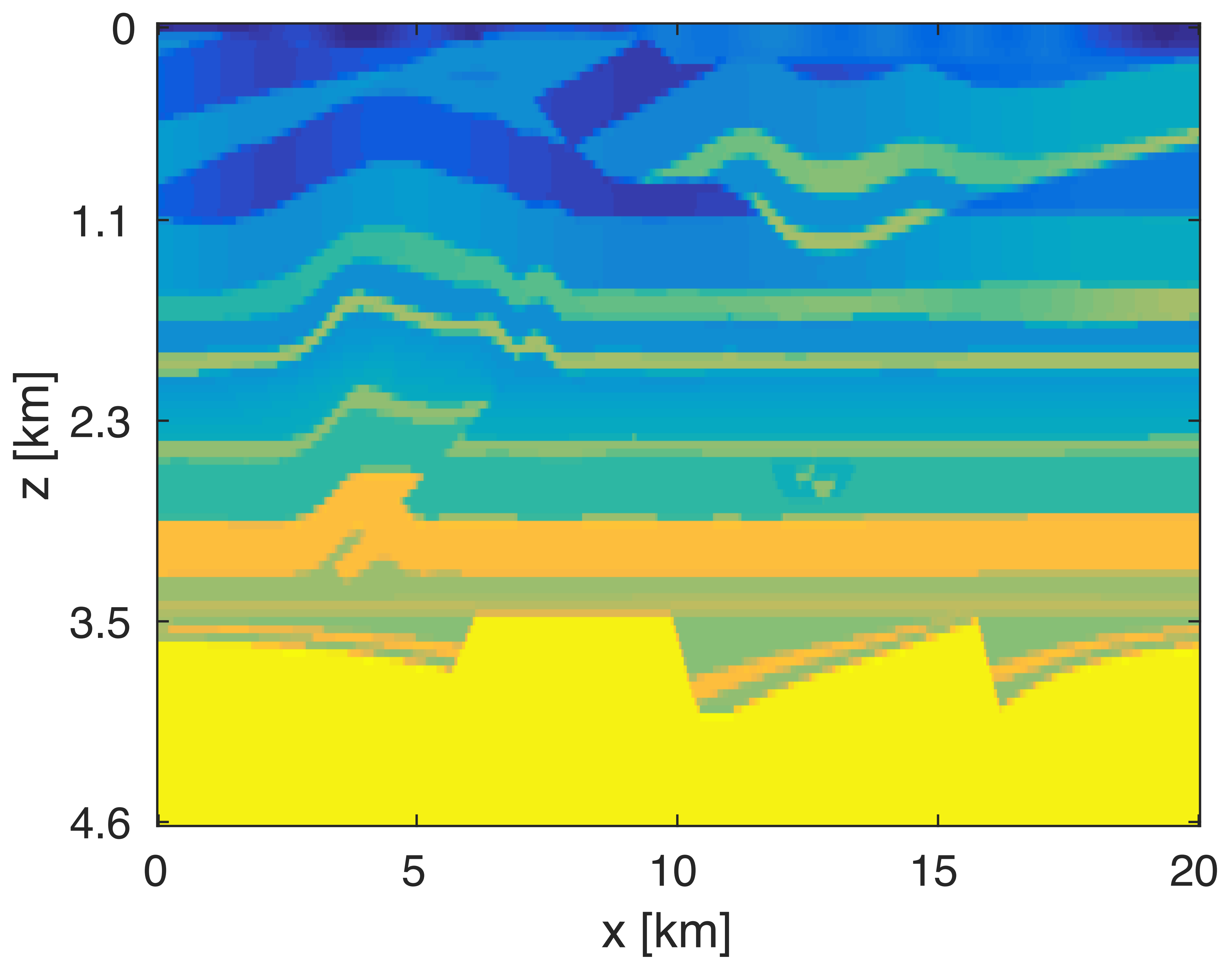


True model

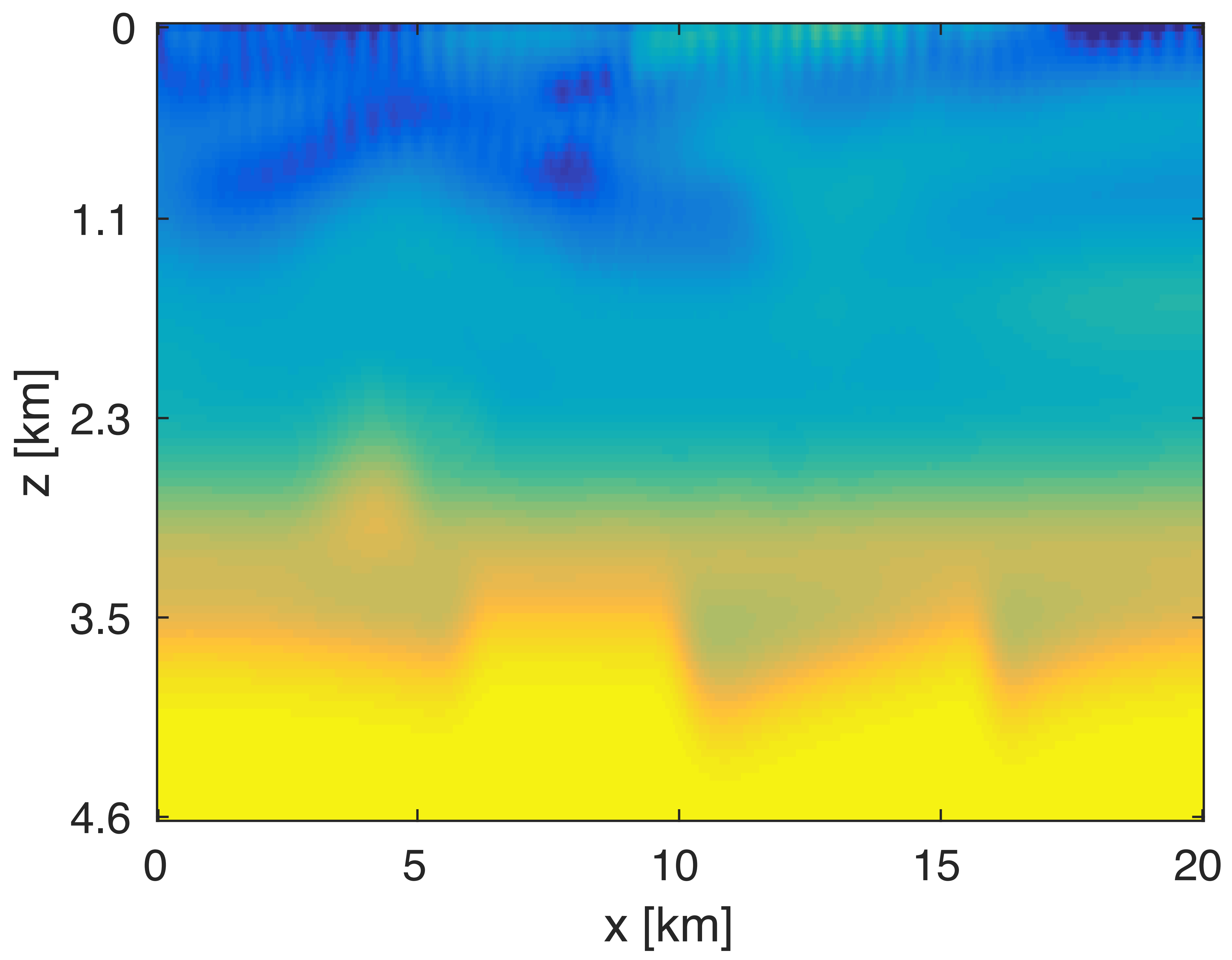


Stochastic LBFGS

y=10000m slice



True model



Full data

Summary

Performance and correctness don’t have to be mutually exclusive

- Design software in a modular, hierarchical way yields benefits of both

Modularity -> flexibility

- Very easy to swap out modules (PDE discretizations, preconditioners) without changing code

Summary

Modularity -> Easier to test

- Easier to test -> easier to get right

We can design code that is *demonstrably* correct

- Reduce scope of potential problems in FWI

Summary

Right abstractions for FWI ->

- ease of use
- computationally efficient
- flexible
- easy to extend, understand, optimize
- can prototype algorithms in 2D, run immediately in 3D

Acknowledgements

This research was carried out as part of the SINBAD project with the support of the member organizations of the SINBAD Consortium.

Acknowledgements

The authors wish to acknowledge the SENAI CIMATEC Supercomputing Center for Industrial Innovation, with support from BG Brasil, Shell, and the Brazilian Authority for Oil, Gas and Biofuels (ANP), for the provision and operation of computational facilities and the commitment to invest in Research & Development.

Thank you for your attention