High-performance seismic applications of DEVITO

Mathias Louboutin

Tuesday October 25
SINBAD Consortium Meeting 2016

SLIM
University of British Columbia

Continuation of

02:30—03:15 PM

Gerard Gorman

Open Performance portablE SeismiC Imaging —OPESCI

Stay tuned for

04:00—04:30 PM

Philipp A. Witte

A large-scale time-domain modeling and inversion workflow in Julia

Motivations

Symbolic representation of PDEs

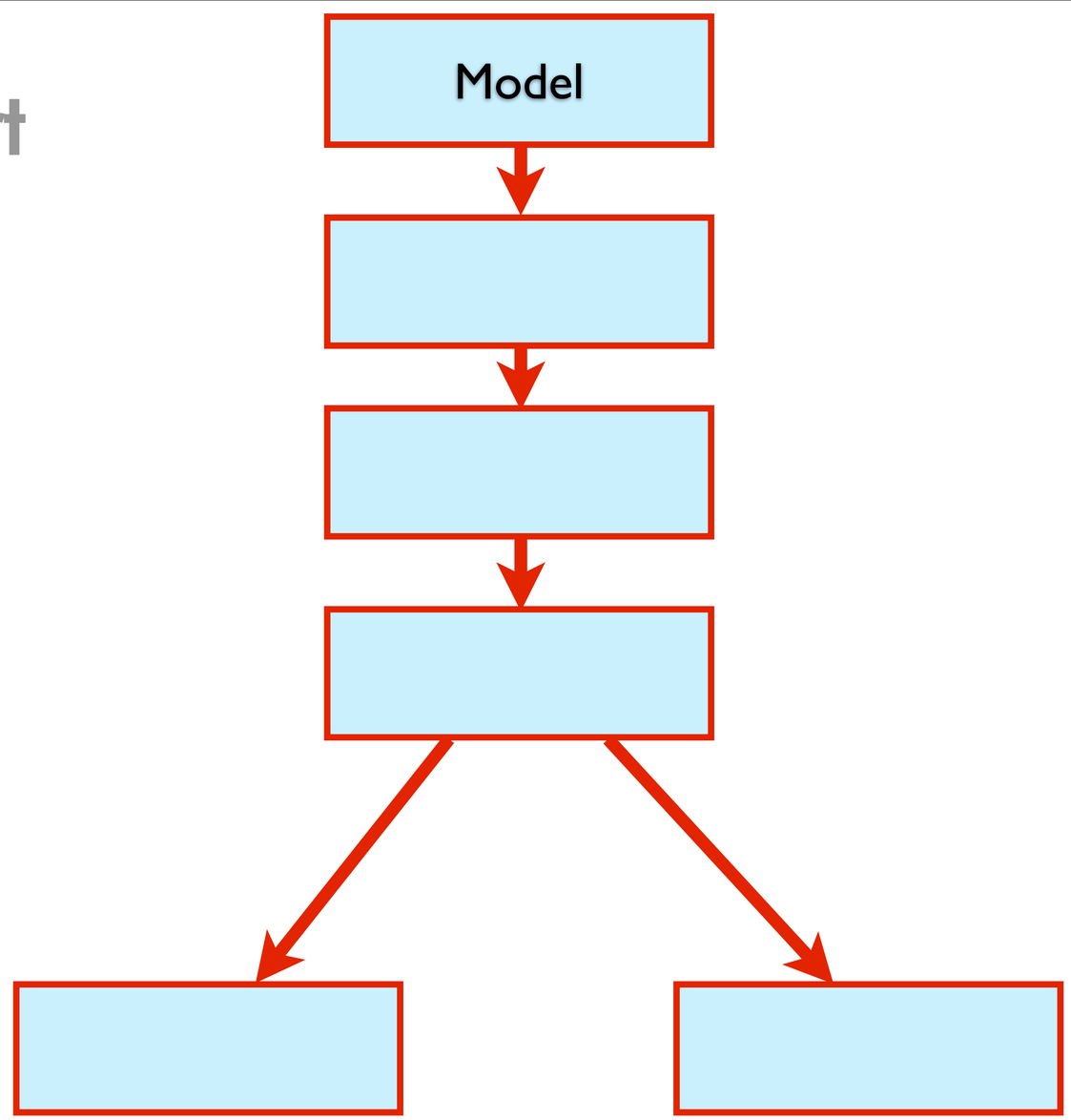
Automatic code generation

Flexible discretization

Flexible physical representation

Simple example step by step walkthrough

Devito flowchart



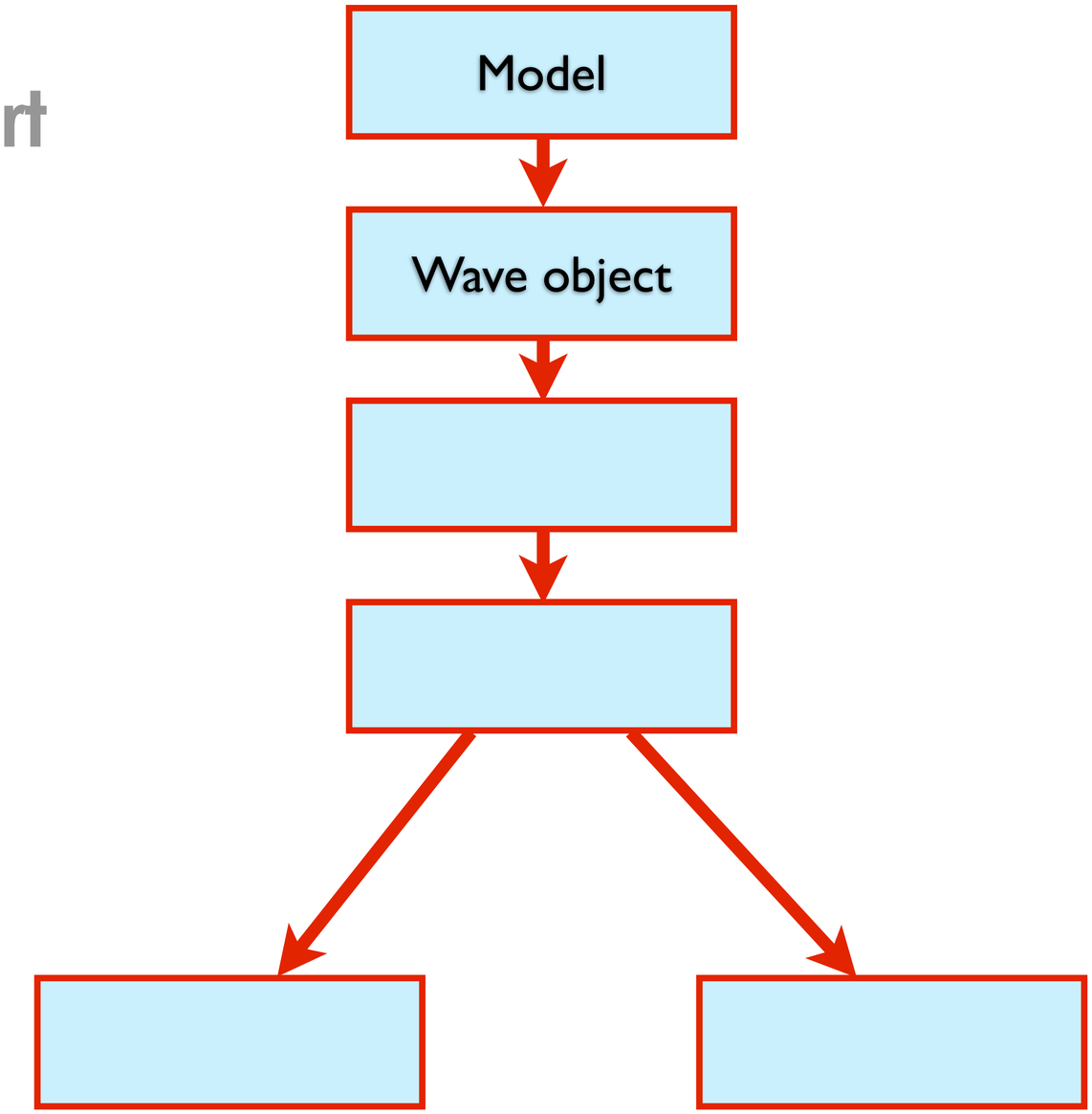
Grid & physical properties

```
# Grid definition
dimensions = (200, 200)
origin = (0., 0.)
spacing = (20., 20.)

# Velocity model

true_vp = np.ones(dimensions) + .5
true_vp[:, int(dimensions[0] /2):int(dimensions[0])] = 2.5

# Model object for operators
model = IGrid()
model.create_model(origin, spacing, true_vp)
```

Acquisition parameters

```
data = IShot()
# Source parameters
f0 = .010
dt = model.get_critical_dt()
t0 = 0.0
nt = int(1+(tn-t0)/dt)

time_series = source(np.linspace(t0, tn, nt), f0)
location = (origin[0] + dimensions[0] * spacing[0] * 0.5,
            origin[1] + dimensions[1] * spacing[1] * 0.5,
            origin[2] + 2 * spacing[2])
data.set_source(time_series, dt, location)
# Receiver geometry
receiver_coords = np.zeros((101, 3))
receiver_coords[:, 0] = np.linspace(2 * spacing[0],
                                    origin[0] + (dimensions[0] - 2) * spacing[0],
                                    num=101)
receiver_coords[:, 1] = 500
receiver_coords[:, 2] = location[2]
data.set_receiver_pos(receiver_coords)
data.set_shape(nt, 101)
```

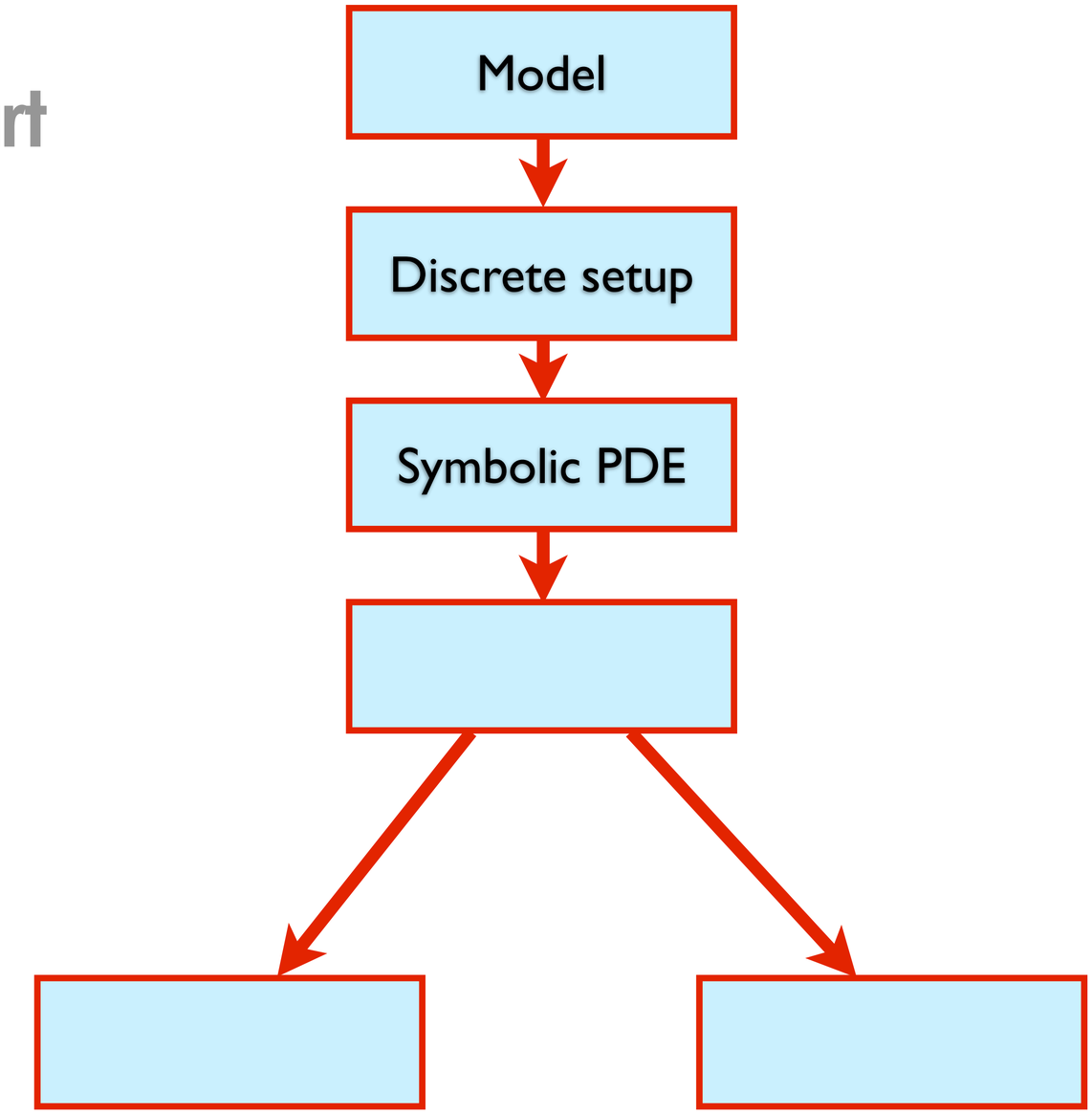
Devito flowchart



Generate acoustic object w/ default discretization orders

```
Acoustic = Acoustic_cg(model, data)
```

Devito flowchart



PDE

$$m(x,y,z)\frac{d^2u(x,y,z,t)}{dt^2} - \nabla^2 u(x,y,z,t) = q$$

$$u(.,.,.,0) = 0$$

$$\frac{du(x,y,z,t)}{dt}|_{t=0} = 0$$

$u(x,y,z,t)$: pressure wavefield

$m(x,y,z)$: square slowness

q : pressure source

∇^2 : Laplacian operator

Discretization

$$\mathbf{m}(\mathbf{x})\frac{\mathbf{u}(\mathbf{x},\mathbf{t}+dt)-2\mathbf{u}(\mathbf{x},t)+\mathbf{x},\mathbf{t}-dt)}{dt^2}-\Delta\mathbf{u}(\mathbf{x},\mathbf{t})=0$$

```
equation = m * u.dt2 - u.laplace + damp * u.dt
```

Absorbing boundary condition

$\mathbf{u}$: discretized wavefield

$\mathbf{m}$: discretized square slowness

Δ : discretized Laplacian

```
u = TimeData(name="u", shape=model.get_shape_comp(), time_dim=nt
             time_order=time_order, space_order=spc_order,
             save=save,dtype=damp.dtype)

m = DenseData(name="m", shape=model.get_shape_comp(),
              dtype=damp.dtype)


Lap = u.laplace
```

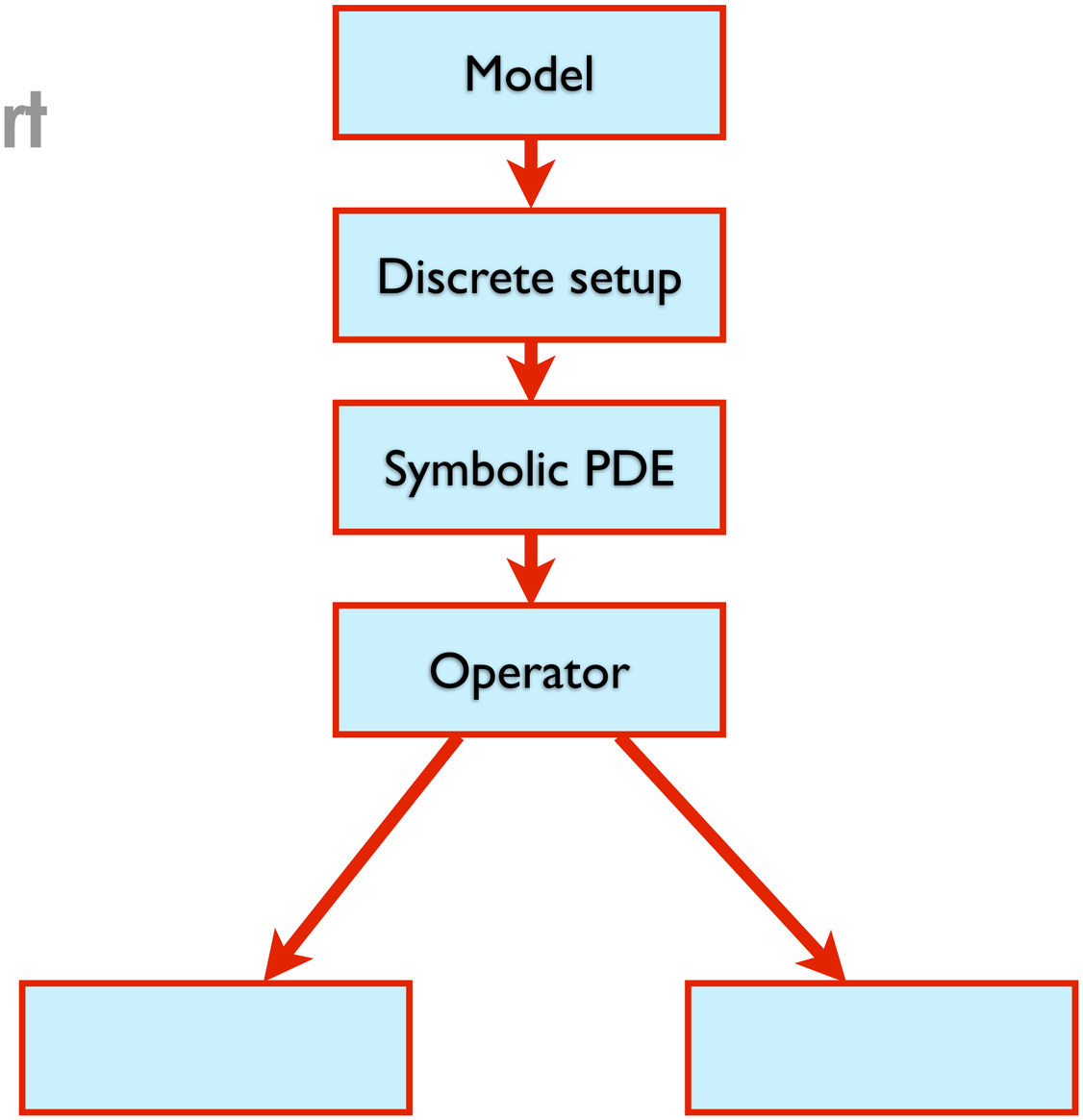
Stencil

$$\mathbf{u}(\mathbf{x}, \mathbf{t} + dt) = \frac{dt^2}{\mathbf{m}(\mathbf{x})}(2\mathbf{u}(\mathbf{x}, t) - \mathbf{u}(\mathbf{x}, \mathbf{t} - dt) + \Delta\mathbf{u}(\mathbf{x}, \mathbf{t}))$$

u.forward = solve(equation, u.forward)

```
stencil = Eq(u.forward, solve(equation, u.forward))
```

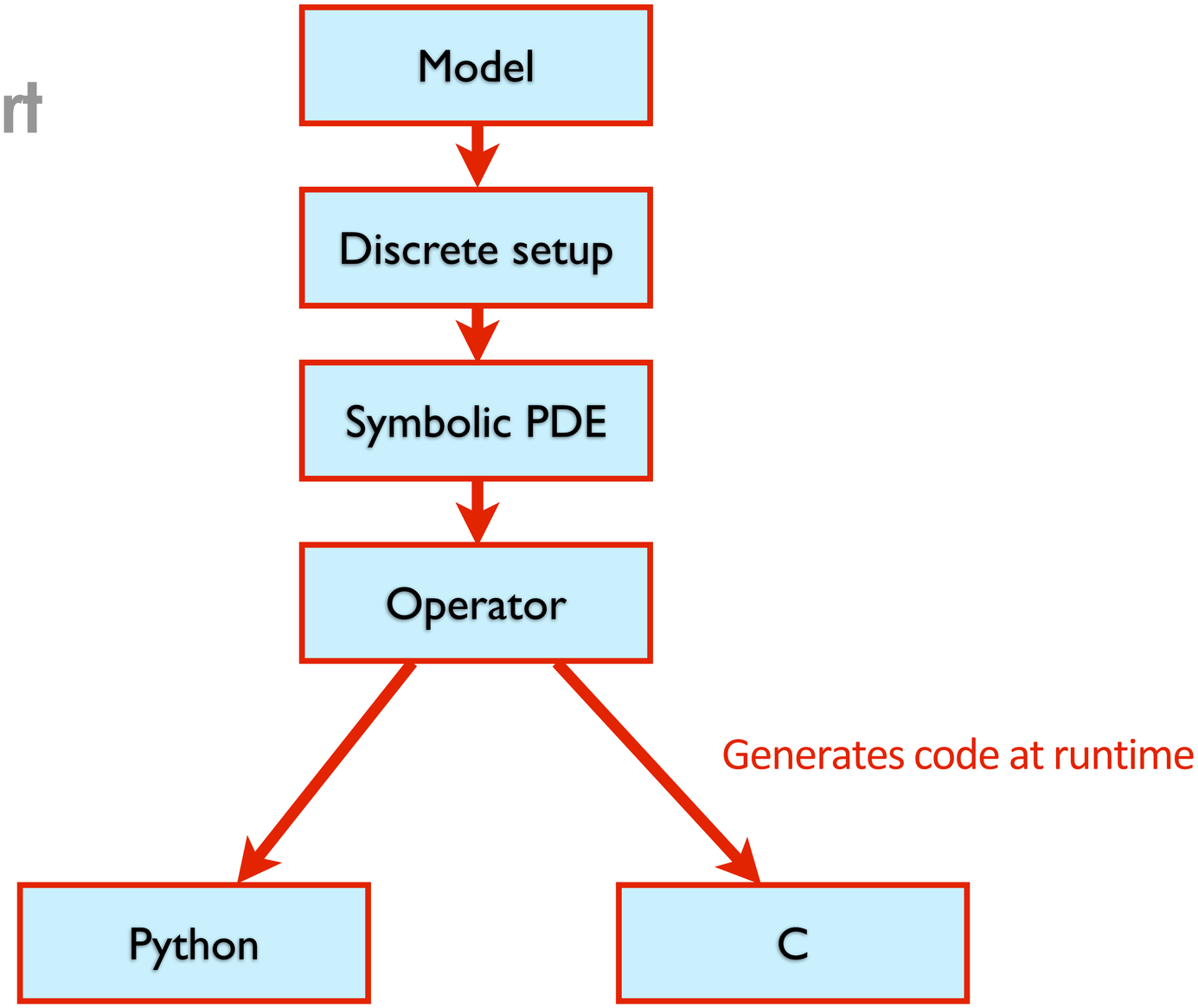
Devito Flowchart



Forward operator

```
# Create a forward operator
super(ForwardOperator, self).__init__(nt, m.shape,
                                      stencils=Eq(p.forward,stencil),
                                      subs=subs,
                                      spc_border=spc_order/2,
                                      time_order=time_order,
                                      forward=True,
                                      dtype=m.dtype,
                                      **kwargs)

# Insert source and receiver terms post-hoc
self.propagator.time_loop_stencils_a = source.add(m, u) + rec.read(u)
```

Devito Flowchart



Python

```
(rec, u) = Acoustic.Forward()
```

```
def Forward(self):
    xmin, ymin = self.model.origin
    nt = self.nt
    nx, ny = self.model.get_shape()
    m = self.model.vp**(-2)
    dt = self.dt
    h = self.h

    u = np.zeros((nt, nx, ny))
    damp = self.damp_boundary()
    rec = np.zeros((nt, self.nrec))

    for ti in range(0, nt):
        for a in range(1, nx-1):
            for b in range(1, ny-1):
                if ti == 0:
                    u[ti, a, b] = self.ts(0, 0, 0, 0, 0, 0,
                                          m[a, b], dt, h, damp[a, b])
                elif ti == 1:
                    u[ti, a, b] = self.ts(0, u[ti - 1, a - 1, b],
                                          u[ti - 1, a, b],
                                          u[ti - 1, a + 1, b],
                                          u[ti - 1, a, b - 1],
                                          u[ti - 1, a, b + 1],
                                          m[a, b], dt, h, damp[a, b])
                else:
                    u[ti, a, b] = self.ts(u[ti - 2, a, b],
                                          u[ti - 1, a - 1, b],
                                          u[ti - 1, a, b],
                                          u[ti - 1, a + 1, b],
                                          u[ti - 1, a, b - 1],
                                          u[ti - 1, a, b + 1],
                                          m[a, b], dt, h, damp[a, b])

        self.add_source(self.data.get_source(ti), m, dt, u[ti, :, :])
        self.read_rec(rec[ti, :], u[ti, :, :])

    return rec, u
```

C

```
(rec, u) = Acoustic.Forward()
```

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
```

C

```
          timings->loop_body += (double)(end_loop_body.tv_sec-start_loop_body.tv_sec)+(double)(end_loop_body.tv_usec-
start_loop_body.tv_usec)/1000000;
        }
        {
          struct timeval start_loop_stencils_a, end_loop_stencils_a;
          gettimeofday(&start_loop_stencils_a, NULL);
          for (int p = 0; p < 2; p++)
          {
            u[t2][int(floor(5.0e-2F*src_coords[p][0])) + 40][int(floor(5.0e-2F*src_coords[p][1])) + 40] =
3.48444444444444F*(2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p][0])) + src_coords[p][0])*float(-2.0e
+1F*int(floor(5.0e-2F*src_coords[p][1])) + src_coords[p][1]) - 5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p]
[0])) + src_coords[p][0]) - 5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p][1])) + src_coords[p][1]) +
1)*src[i3][p]/m[int(floor(5.0e-2F*src_coords[p][0])) + 40][int(floor(5.0e-2F*src_coords[p][1])) + 40] + u[t2]
[int(floor(5.0e-2F*src_coords[p][0])) + 40][int(floor(5.0e-2F*src_coords[p][1])) + 40];
            u[t2][int(floor(5.0e-2F*src_coords[p][0])) + 40][int(floor(5.0e-2F*src_coords[p][1])) + 41] =
3.48444444444444F*(-2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p][0])) + src_coords[p][0])*float(-2.0e
+1F*int(floor(5.0e-2F*src_coords[p][1])) + src_coords[p][1]) + 5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p]
[1])) + src_coords[p][1]))*src[i3][p]/m[int(floor(5.0e-2F*src_coords[p][0])) + 40][int(floor(5.0e-2F*src_coords[p]
[1])) + 41] + u[t2][int(floor(5.0e-2F*src_coords[p][0])) + 40][int(floor(5.0e-2F*src_coords[p][1])) + 41];
            u[t2][int(floor(5.0e-2F*src_coords[p][0])) + 41][int(floor(5.0e-2F*src_coords[p][1])) + 40] =
3.48444444444444F*(-2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p][0])) + src_coords[p][0])*float(-2.0e
+1F*int(floor(5.0^ 2F*src_coords[p][1])) + src_coords[p][1]) + 5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p]
[0])) + src_coord                                                                    5.0e-2F*src_coords[p]
[1])) + 40] + u[t                                                                    ][1])) + 40];
            u[t2][int(floor(5.0e-2F*src_coords[p][0])) + 41][int(floor(5.0e-2F*src_coords[p][1])) + 41] =
8.71111111111111e-3F*float(-2.0e+1F*int(floor(5.0e-2F*src_coords[p][0])) + src_coords[p][0])*float(-2.0e
+1F*int(floor(5.0e-2F*src_coords[p][1])) + src_coords[p][1])*src[i3][p]/m[int(floor(5.0e-2F*src_coords[p][0])) + 41]
[int(floor(5.0e-2F*src_coords[p][1])) + 41] + u[t2][int(floor(5.0e-2F*src_coords[p][0])) + 41]
[int(floor(5.0e-2F*src_coords[p][1])) + 41];
          }
          for (int p = 0; p < 101; p++)
          {
            rec[i3][p] = (-2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p]
[0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) + 5.0e-2F*float(-2.0e
+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p][0]))*u[t2][int(floor(5.0e-2F*rec_coords[p][0])) + 41]
[int(floor(5.0e-2F*rec_coords[p][1])) + 40] + (-2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) +
rec_coords[p][0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) + 5.0e-2F*float(-2.0e
+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]))*u[t2][int(floor(5.0e-2F*rec_coords[p][0])) + 40]
[int(floor(5.0e-2F*rec_coords[p][1])) + 41] + (2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) +
rec_coords[p][0])*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1]) - 5.0e-2F*float(-2.0e
+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p][0]) - 5.0e-2F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p]
[1])) + rec_coords[p][1]) + 1)*u[t2][int(floor(5.0e-2F*rec_coords[p][0])) + 40][int(floor(5.0e-2F*rec_coords[p][1]))
+ 40] + 2.5e-3F*float(-2.0e+1F*int(floor(5.0e-2F*rec_coords[p][0])) + rec_coords[p][0])*float(-2.0e
+1F*int(floor(5.0e-2F*rec_coords[p][1])) + rec_coords[p][1])*u[t2][int(floor(5.0e-2F*rec_coords[p][0])) + 41]
[int(floor(5.0e-2F*rec_coords[p][1])) + 41];
          }
          {
            gettimeofday(&end_loop_stencils_a, NULL);
            timings->loop_stencils_a += (double)(end_loop_stencils_a.tv_sec-start_loop_stencils_a.tv_sec)+(double)
(end_loop_stencils_a.tv_usec-start_loop_stencils_a.tv_usec)/1000000;
          }
        }
      }
    }
    {
      gettimeofday(&end_kernel, NULL);
      timings->kernel += (double)(end_kernel.tv_sec-start_kernel.tv_sec)+(double)(end_kernel.tv_usec-
start_kernel.tv_usec)/1000000;
    }
  }
  return 0;
}
```

(rec, u) = Acoustic.Forward()

C

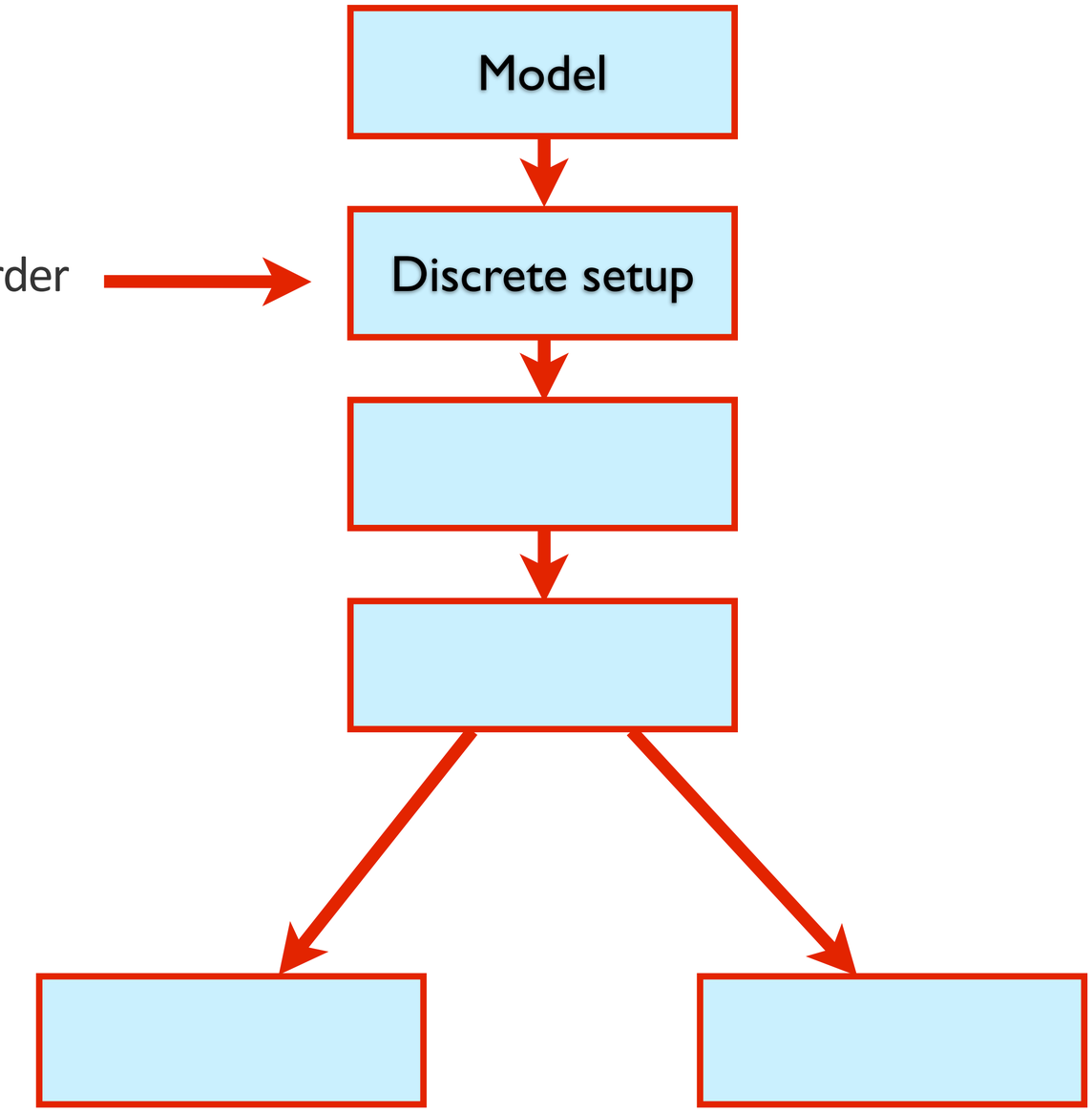
```
(rec, u) = Acoustic.Forward()
```

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
```

Flexibility

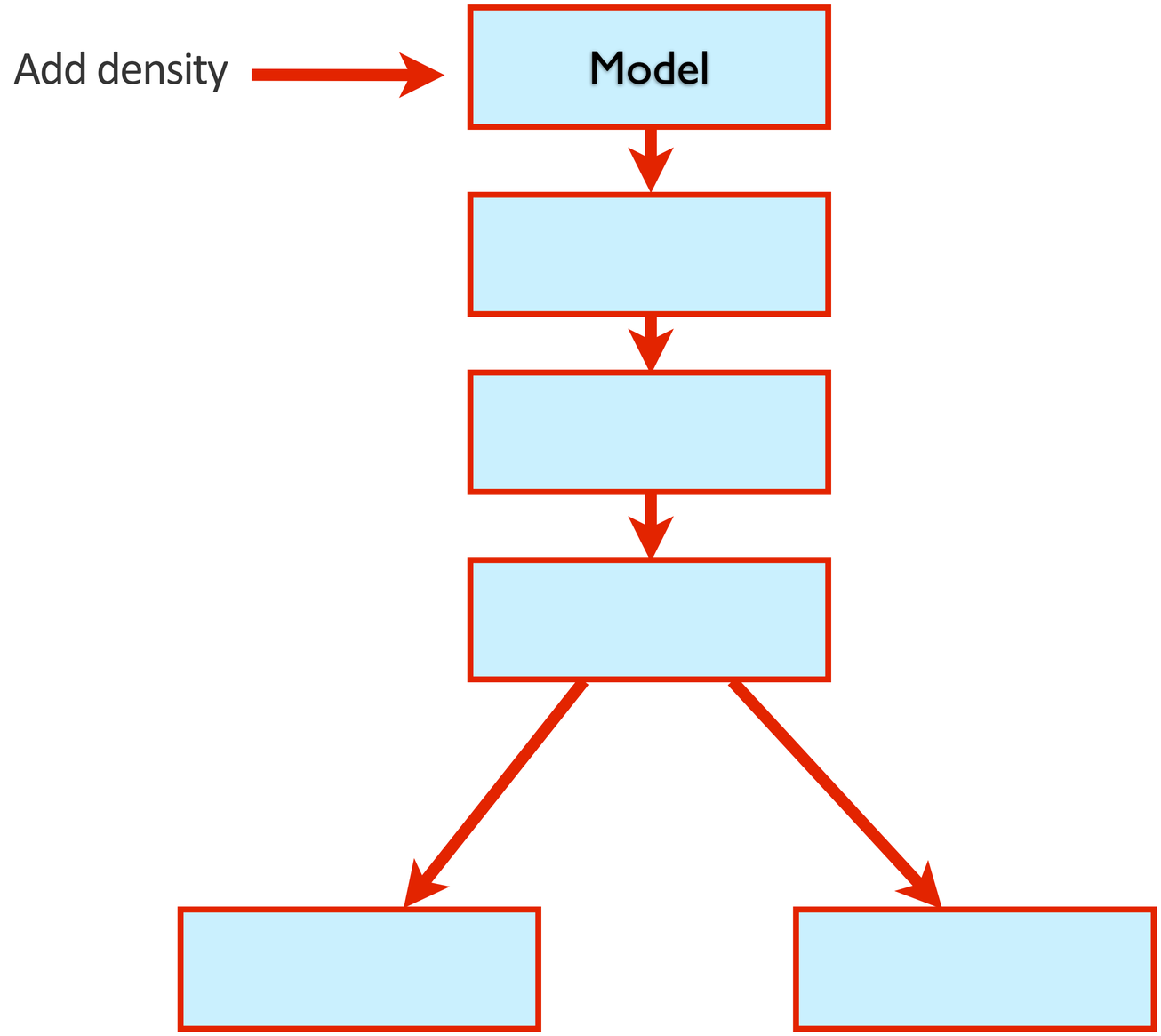
How to change discretization or physics?

Change discretization order



Generate acoustic object w/ default discretization orders

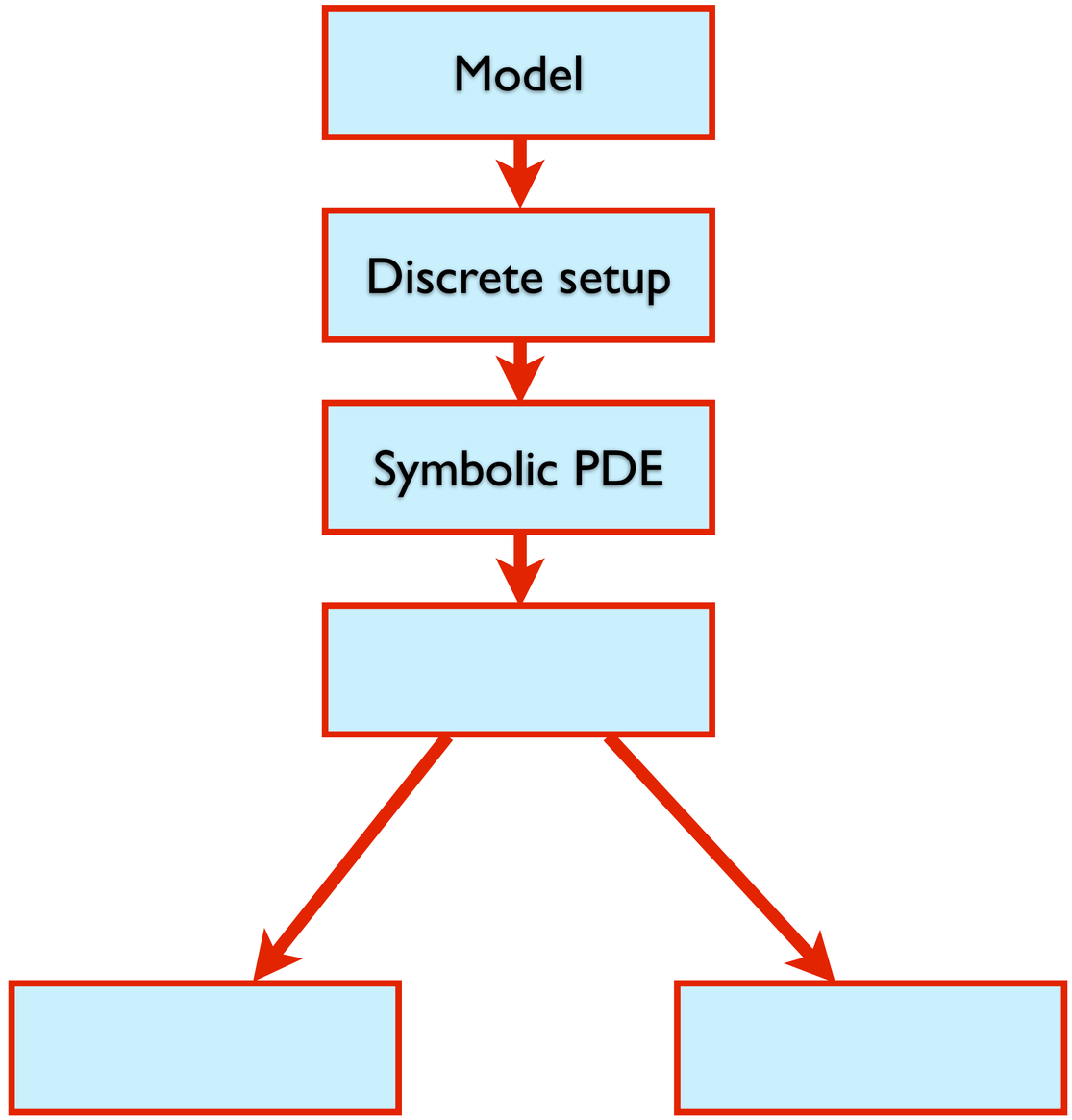
```
Acoustic = Acoustic_cg(model,data,t_order=(2,4) ,s_order=(2,4,6,8,10,12,...))
```


Add density
Model

Grid & physical properties

```
# Grid definition
dimensions = (200, 200)
origin = (0., 0.)
spacing = (20., 20.)

model = IGrid()

# True velocity (as an array)
true_vp = np.ones(dimensions) + .5
true_vp[:, int(dimensions[0] /2):int(dimensions[0])] = 2.5

density = true_vp - .5
# Smooth velocity
initial_vp = smooth10(true_vp, dimensions)

dm = true_vp**-2 - initial_vp**-2

dv = -true_vp + initial_vp

model.create_model(origin, spacing, true_vp, density)
```

Devito



PDE

$$\frac{m(x,y,z)}{\rho(x,y,z)}\frac{d^2u(x,y,z,t)}{dt^2} - \nabla.(\frac{1}{\rho(x,y,z)}\nabla u(x,y,z,t)) = q$$

$$u(.,.,.,0) = 0$$

$$\frac{du(x,y,z,t)}{dt}|_{t=0} = 0$$

$u(x,y,z,t)$: pressure wavefield

$m(x,y,z)$: square slowness

$rho(x,y,z)$: density

q : pressure source

∇ : gradient operator

$\nabla.$: divergence operator

Discretized equation

Set to 1 without density

Absorbing boundary condition

```
equation = m /rho * u.dt2 - Lap + damp * u.dt
```

```
u = TimeData(name="u", shape=model.get_shape_comp(), time_dim=nt,
             time_order=time_order, space_order=spc_order,
             save=save,dtype=damp.dtype)

m = DenseData(name="m", shape=model.get_shape_comp(),
              dtype=damp.dtype)

rho = DenseData(name="rho", shape=model.get_shape_comp(),
                dtype=damp.dtype)

Lap = (1/rho * u.dx2 - (1/rho)**2 * rho.dx * u.dx +
       1/rho * u.dy2 - (1/rho)**2 * rho.dy * u.dy +
       1/rho * u.dz2 - (1/rho)**2 * rho.dz * u.dz)
```

Testing framework

Rigorousness unit tests for code validation

Why testing

Rigorousness tests used as unit tests

Validate implementation

Allows stable continuous software integration with automated testing

Testing framework

Forward adjoint test:

for any random $\mathbf{x} \in \text{span}(\mathbf{P}_s\mathbf{A}^T\mathbf{P}_r^T), \mathbf{y} \in \text{span}(\mathbf{P}_r\mathbf{A}\mathbf{P}_s^T)$

$$< \mathbf{P}_r\mathbf{A}\mathbf{P}_s^T\mathbf{x}, \mathbf{y} > - < \mathbf{x}, \mathbf{P}_s\mathbf{A}^T\mathbf{P}_r^T\mathbf{y} >= 0$$

Gradient test:

for any small model perturbation $\mathbf{dm}$

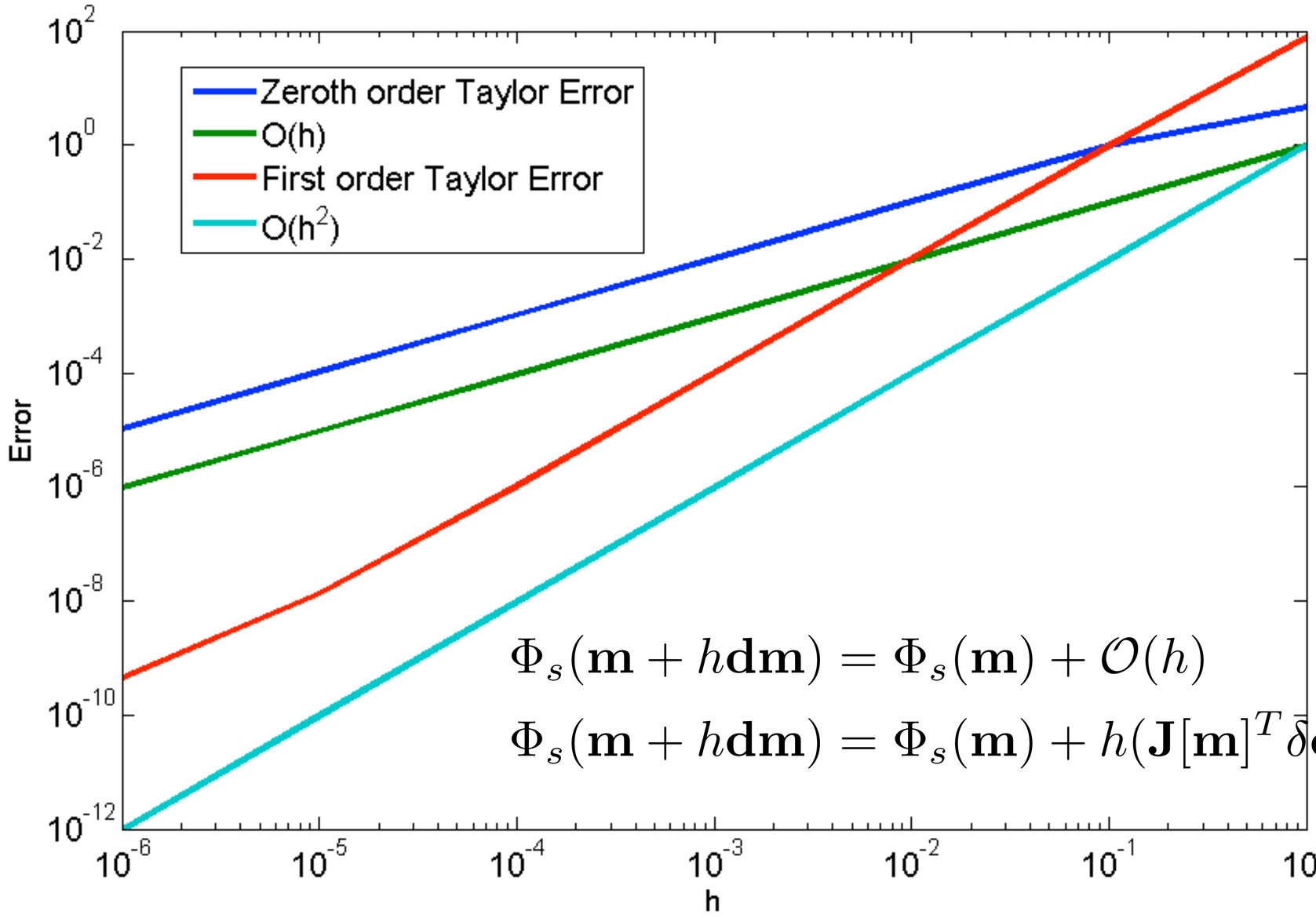
$$\Phi_s(\mathbf{m} + h\mathbf{dm}) = \Phi_s(\mathbf{m}) + \mathcal{O}(h)$$

$$\Phi_s(\mathbf{m} + h\mathbf{dm}) = \Phi_s(\mathbf{m}) + h(\mathbf{J}[\mathbf{m}]^T\delta\mathbf{d})\mathbf{dm} + \mathcal{O}(h^2)$$

Adjoint test

Order	Dimension	$< \mathbf{F}\vec{x}, \vec{y} >$	$< \vec{x}, \mathbf{F}^t\vec{y} >$	Difference	ratio
2nd order	2D:	373323.7976042	373323.79754348495	6.07169350e-05	1.0
4th order	2D:	340158.14865283	340158.14852526429	0.00012756	1.0
6th order	2D:	341557.39488282	341557.3947399481	0.00014287	1.0
8th order	2D:	358240.85136059	358240.85119318636	0.00016741	1.0
10th order	2D:	393488.55616541	393488.55592699442	0.00023841	1.0
12th order	2D:	439561.40056139	439561.40020344808	0.00035794	1.0
2nd order	3D:	2.17496552	2.174965534979078 2	-1.23030883e-08	0.99999999
4th order	3D:	3.64447937	3.644479393901626 9	-2.13132316e-08	0.99999999
6th order	3D:	3.78730372	3.787303745086803 5	-2.22477072e-08	0.99999999
8th order	3D:	3.80286229	3.802862312852404 2	-2.23545817e-08	0.99999999
10th order	3D:	3.80557957	3.80557596334308	-2.23736993e-08	0.99999999
12th order	3D:	3.80318675	3.803186773144855 9	-2.23587757e-08	0.99999999

Gradient test



$$\Phi_s(\mathbf{m}+h\mathbf{dm}) = \Phi_s(\mathbf{m}) + \mathcal{O}(h)$$

$$\Phi_s(\mathbf{m}+h\mathbf{dm}) = \Phi_s(\mathbf{m}) + h(\mathbf{J}[\mathbf{m}]^T\delta\mathbf{d})\mathbf{dm} + \mathcal{O}(h^2)$$

Accuracy validation

Wavefield accuracy check against a precise modelling kernel.

Symes, W. W.: Acoustic Staggered Grid Modeling in IWAVE, THE RICE INVERSION PROJECT, p. 141, http://www.trip.caam.rice.edu/ reports/2014/book.pdf#page=145, 2015a.

Symes, W. W.: IWAVE structure and basic use cases, THE RICE INVERSION PROJECT, p. 85, http://www.trip.caam.rice.edu/reports/2014/ book.pdf#page=89, 2015b.

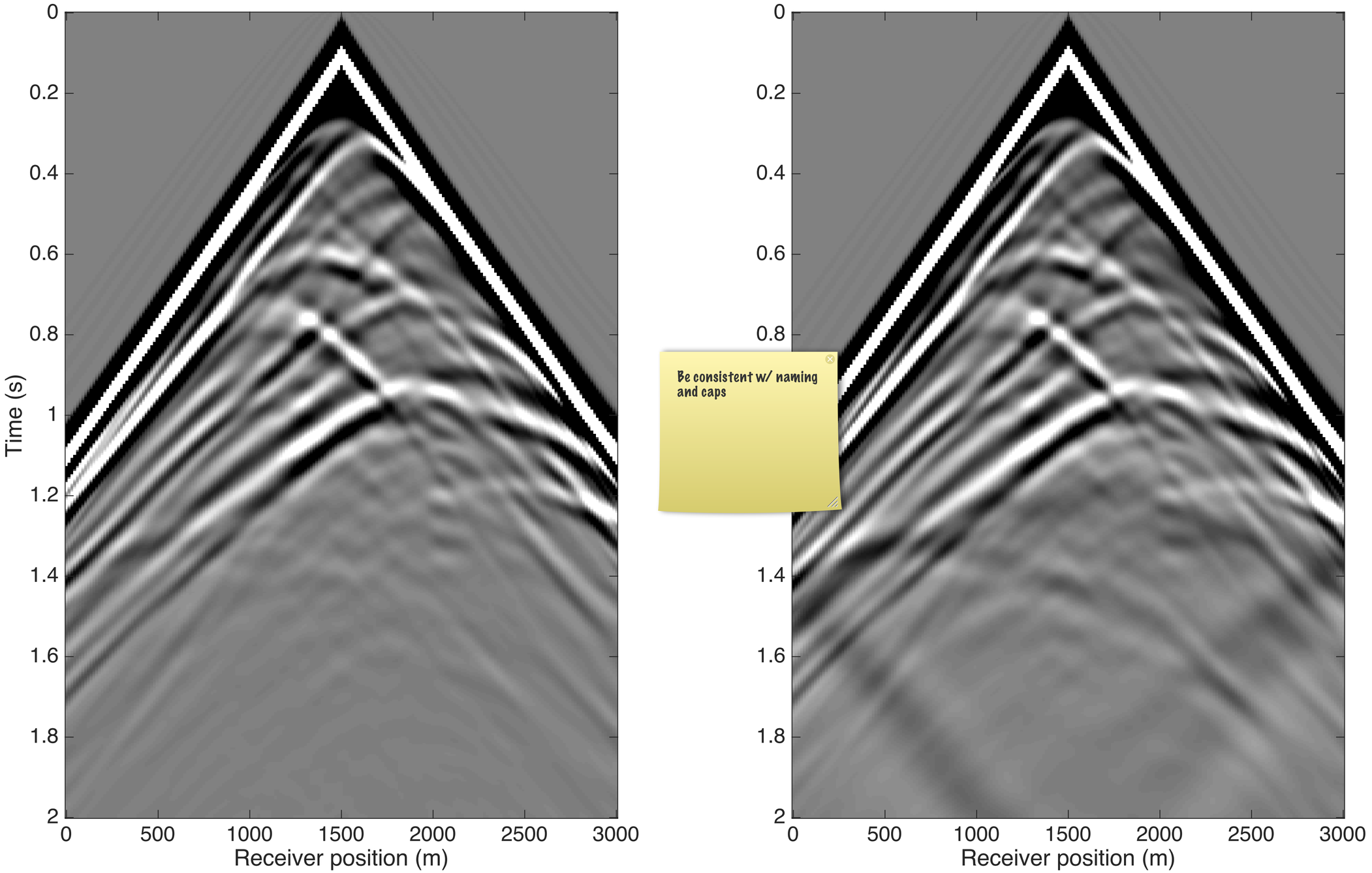
Reference

IWAVE as an precision reference

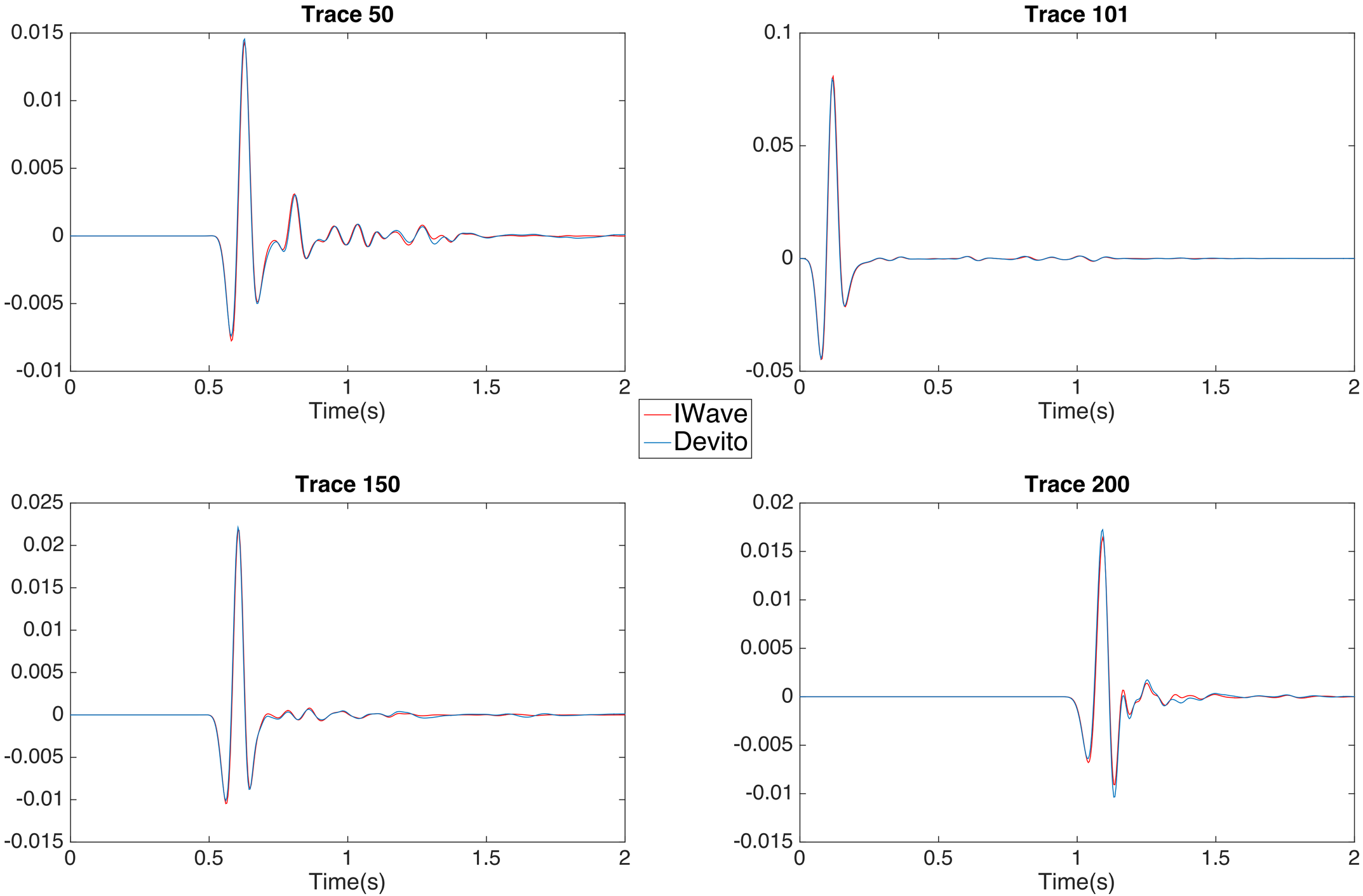
No time/performance comparison

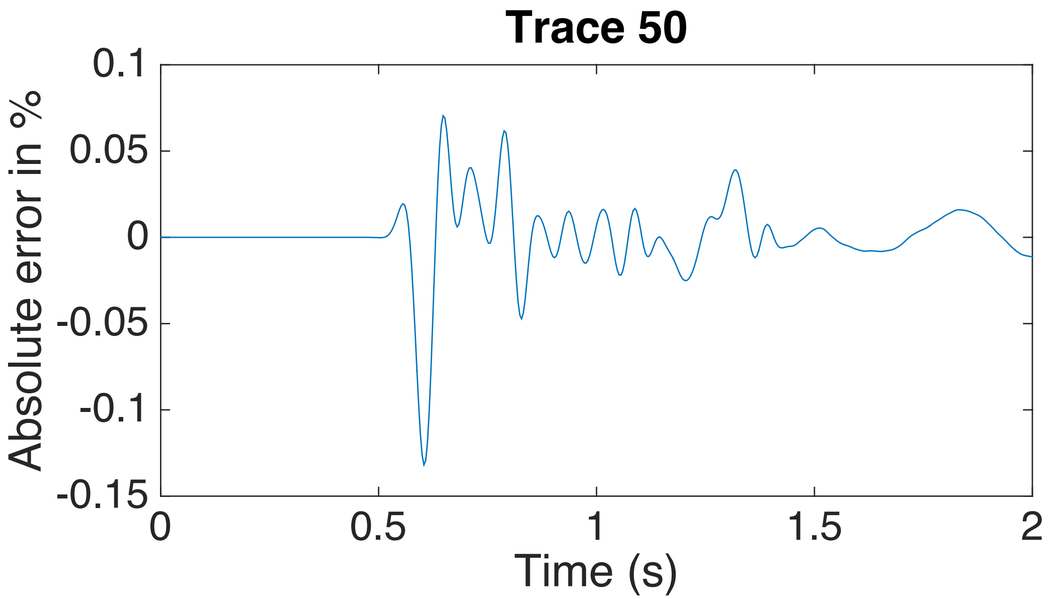
Marmousi 2D

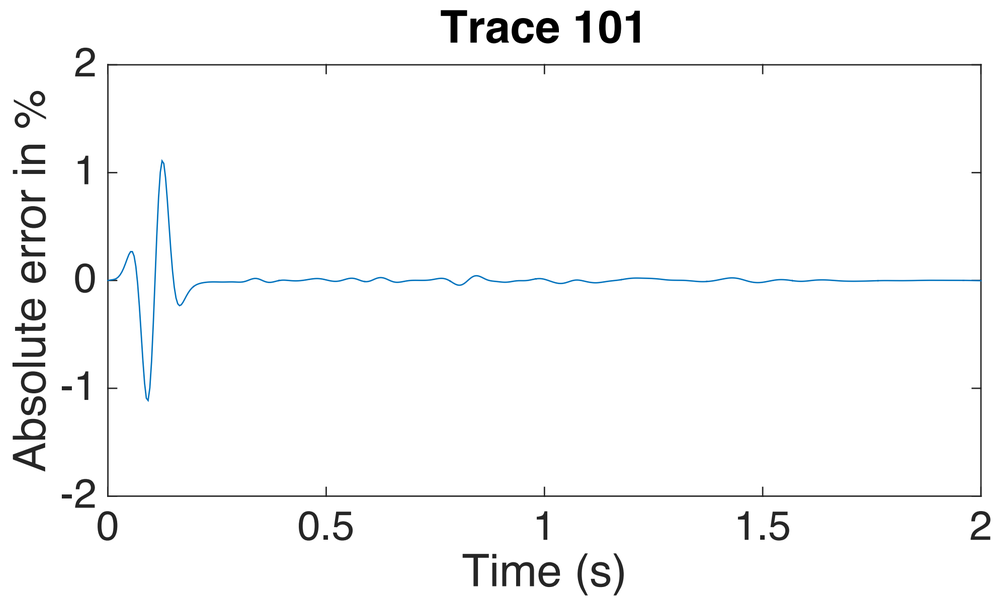
- 201x70 grid points
- 15m grid
- 10Hz Ricker wavelet and 2sec recording
- Source a x =1500m and z = 45m
- Receivers at every grid points at z = 45m

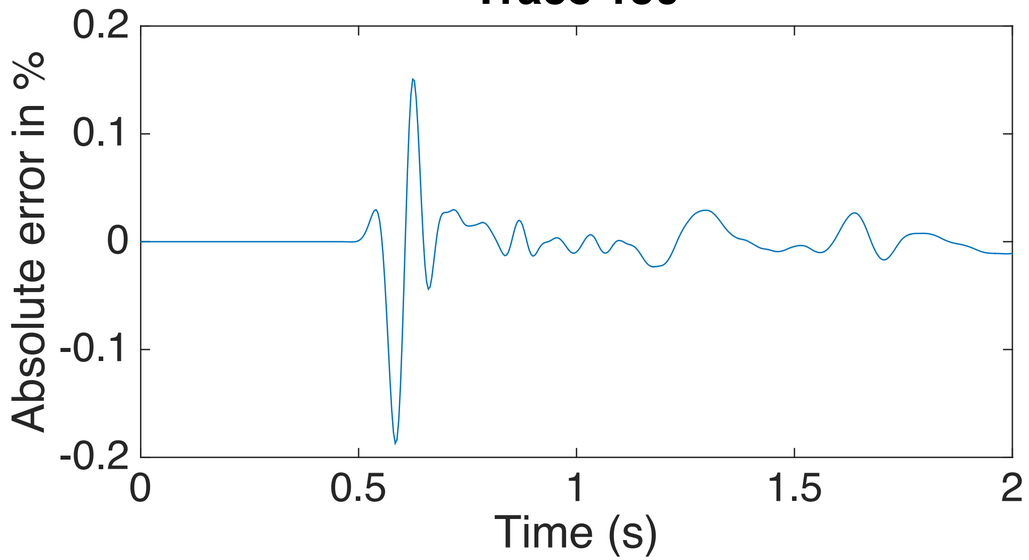


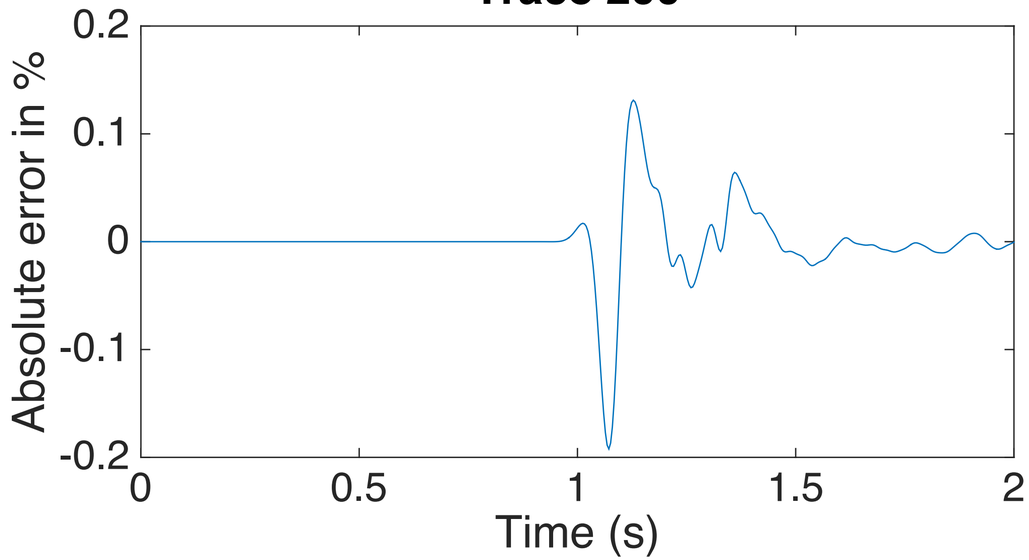
Be consistent w/ naming and caps


Trace 50
0.015
0.01
0.005
0
-0.005
-0.01
0
0.5
1
1.5
2
Time(s)
Trace 101
0.1
0.05
0
-0.05
0
0.5
1
1.5
2
Time(s)
IWave
Devito
Trace 150
0.025
0.02
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.5
1
1.5
2
Time(s)
Trace 200
0.02
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.5
1
1.5
2
Time(s)


Trace 50
Absolute error in %
0.1
0.05
0
-0.05
-0.1
-0.15
0
0.5
1
1.5
2
Time (s)

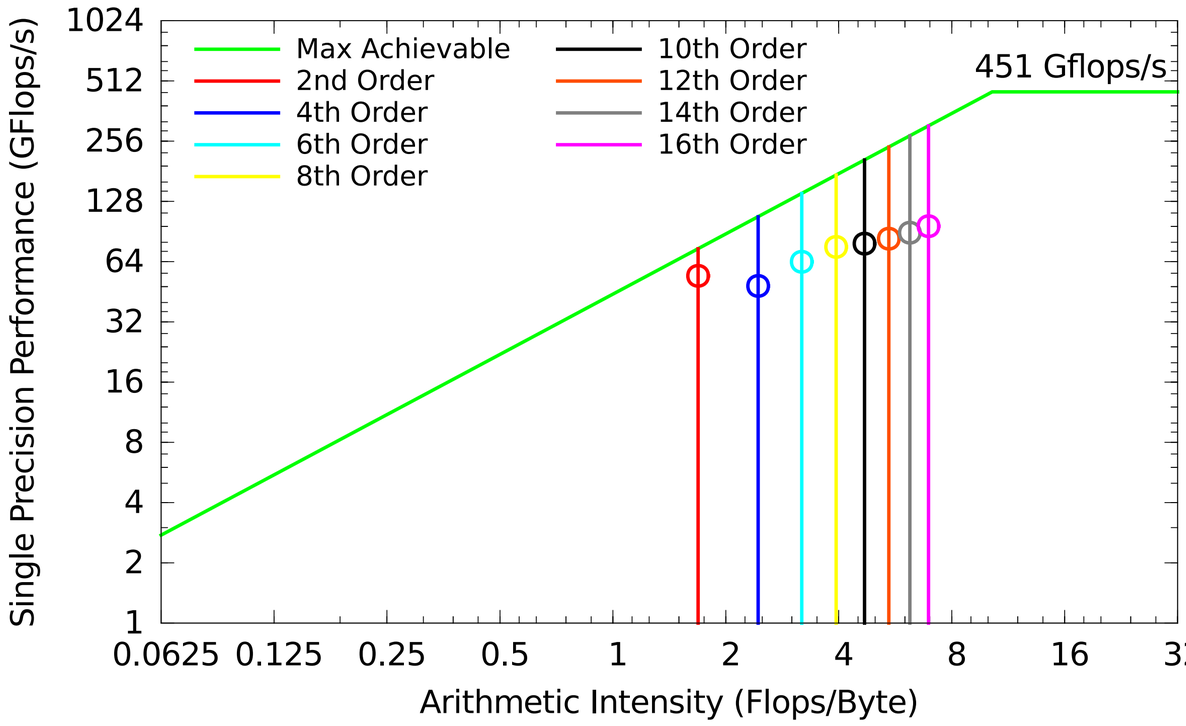

Trace 101
Absolute error in %
2
1
0
-1
-2
0
0.5
1
1.5
2
Time (s)

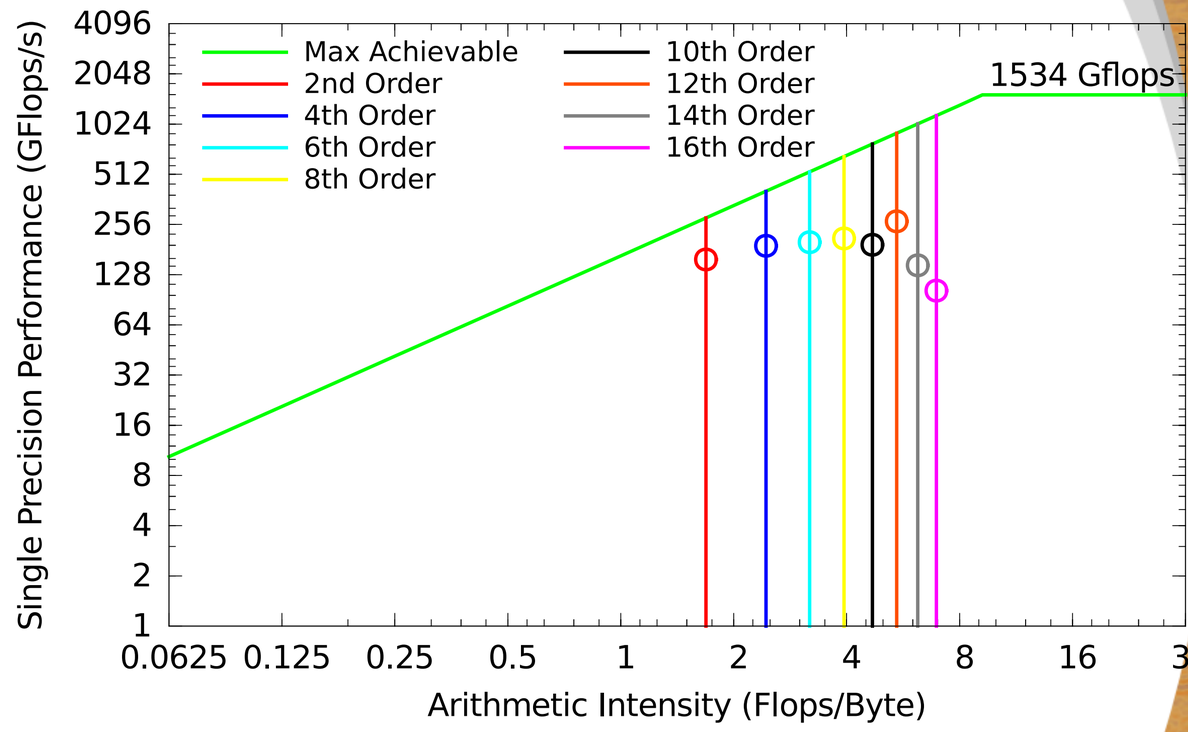

Trace 150
Absolute error in %
0.2
0.1
0
-0.1
-0.2
0
0.5
1
1.5
2
Time (s)


Trace 200
Absolute error in %
0.2
0.1
0
-0.1
-0.2
0
0.5
1
1.5
2
Time (s)

Performance benchmark

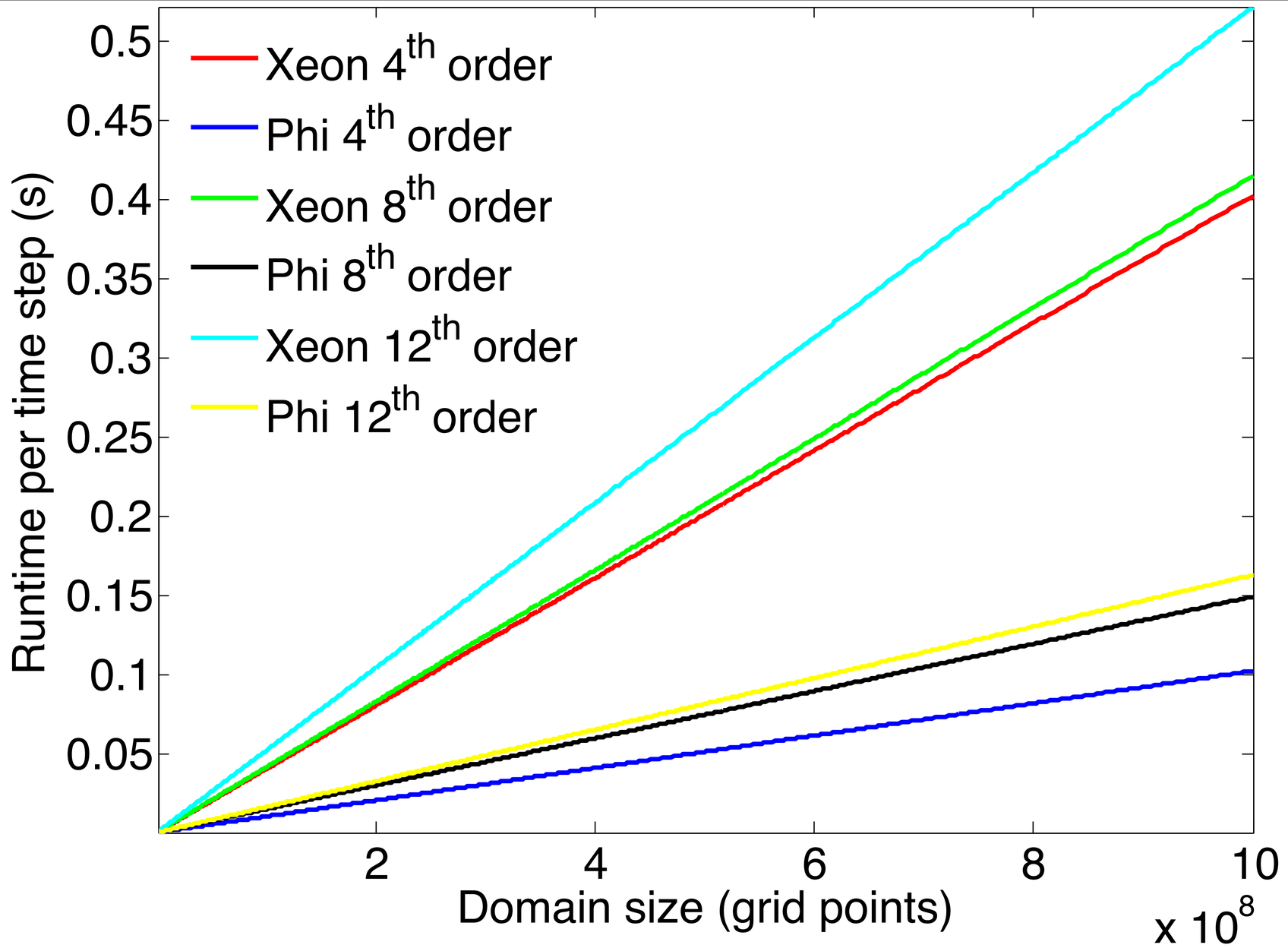
Roofline model on Xeon and Xeon Phi





Louboutin, M., Lange, M., Herrmann, F. J., Kukreja, N., and Gorman, G.: Performance prediction of finite-difference solvers for different computer architectures, submitted to the Computer and Geoscience Journal on September 16, 2016., 2016.

Patterson, D. A. and Hennessy, J. L.: Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann PublishersInc., San Francisco, CA, USA, 3rd edn., 2007.



Time to solution from the roofline results

Time vs. order

5km x 5km x 5km domain

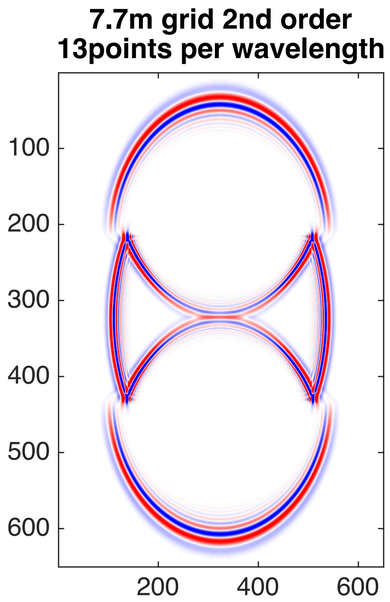
2nd to 20th order spacial discretization

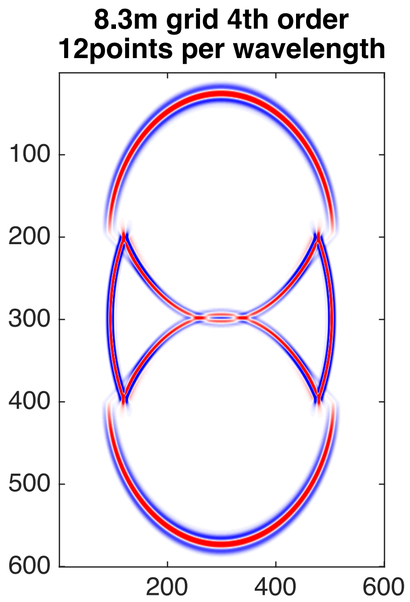
13 to 4 points per wavelength (10Hz, 100m wavelength)

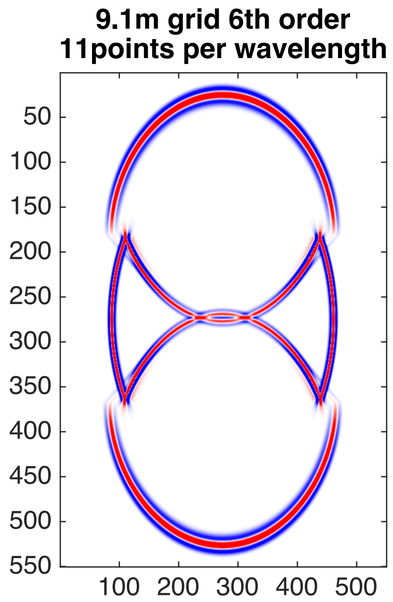
1.2sec modelling

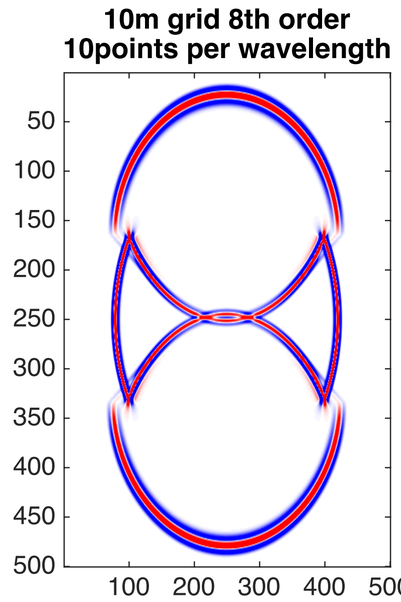
Three layer model

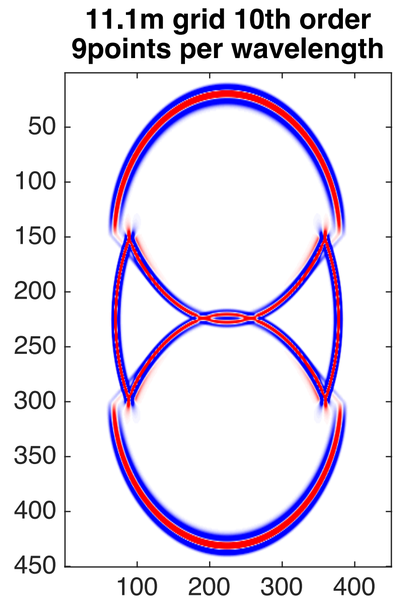
Source at the center

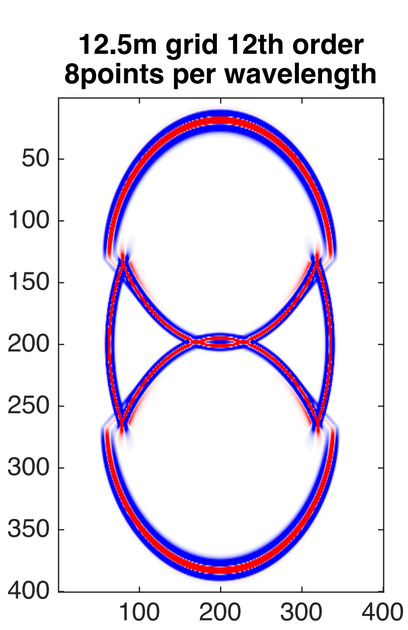

7.7m grid 2nd order
13points per wavelength

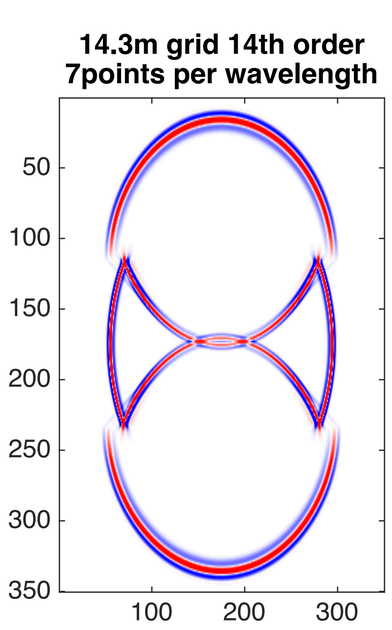

8.3m grid 4th order
12points per wavelength

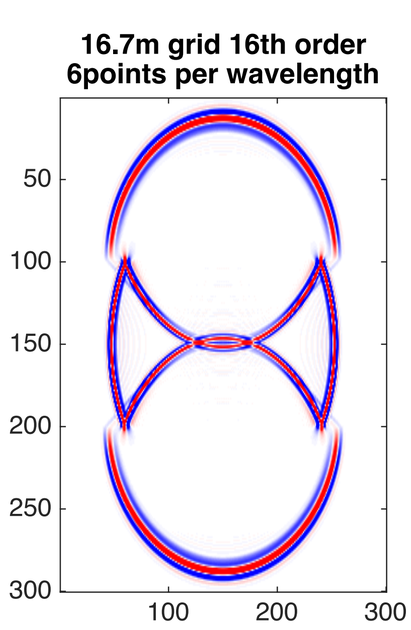

9.1m grid 6th order
11points per wavelength

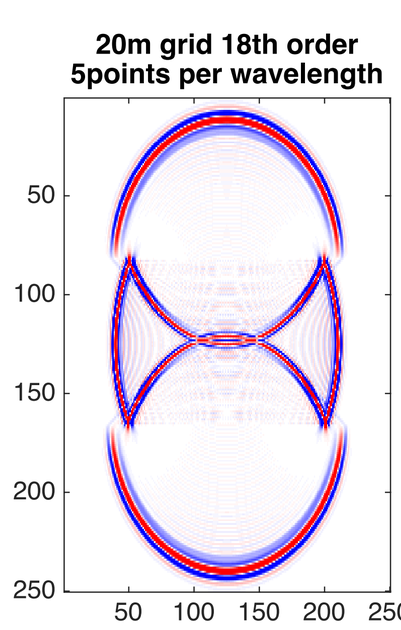

10m grid 8th order
10points per wavelength

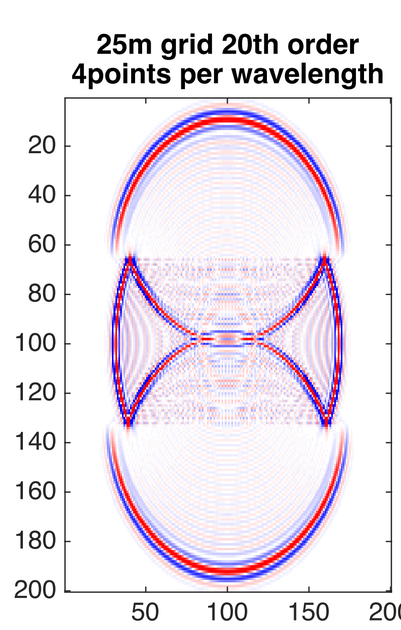

11.1m grid 10th order
9points per wavelength


12.5m grid 12th order
8points per wavelength

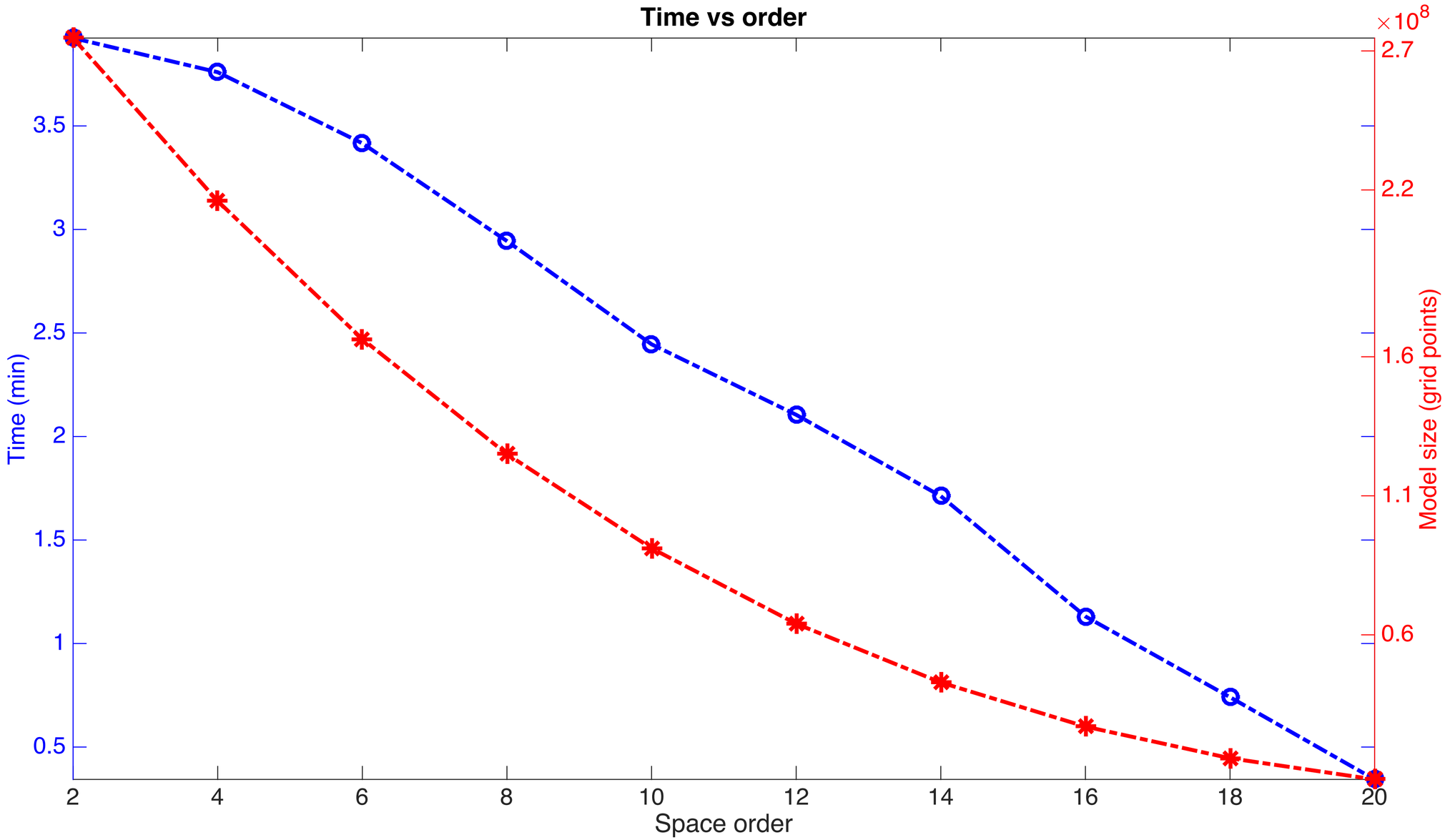

14.3m grid 14th order
7points per wavelength


16.7m grid 16th order
6points per wavelength


20m grid 18th order
5points per wavelength


25m grid 20th order
4points per wavelength

Time to solution



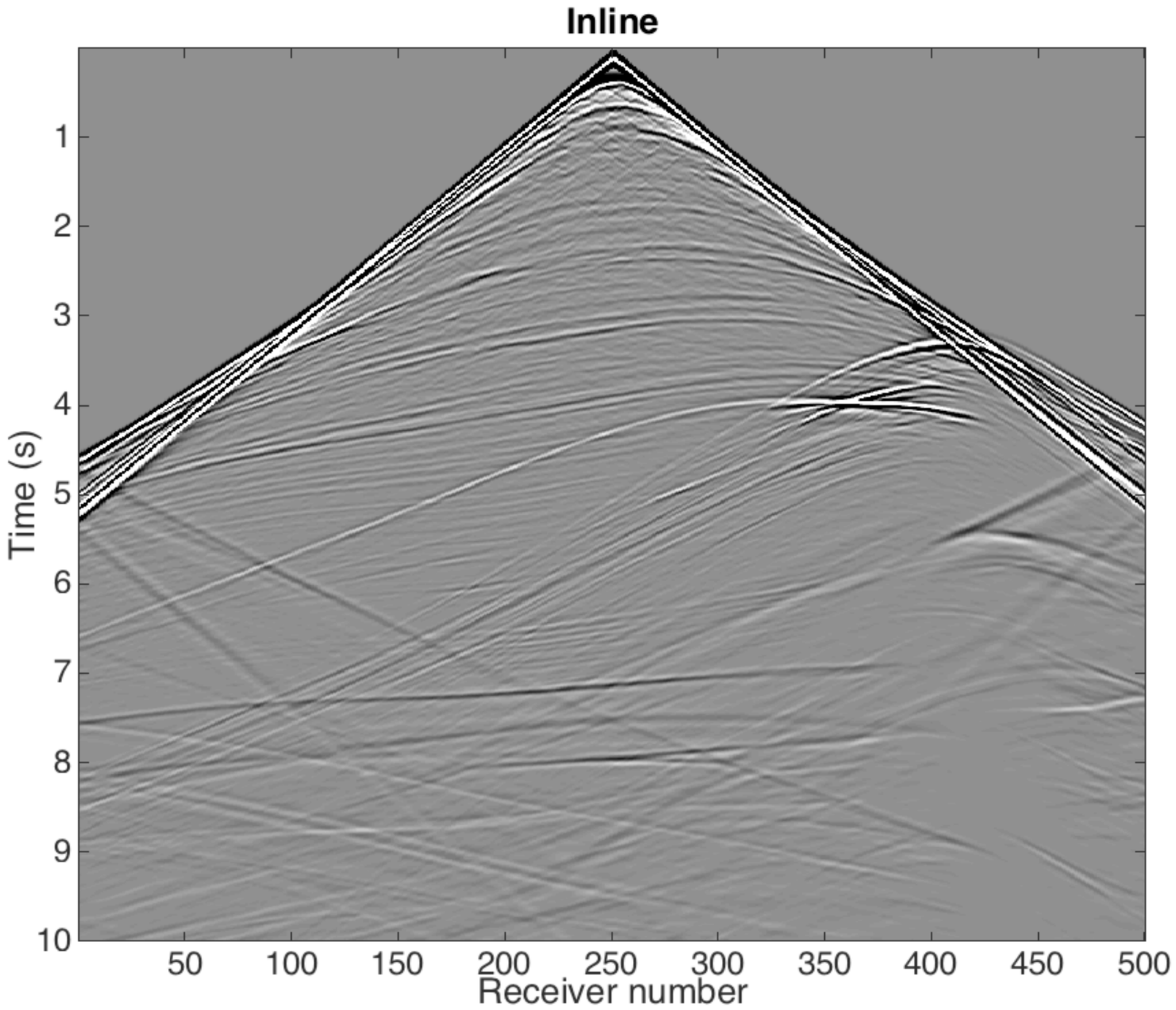
SEAM benchmark

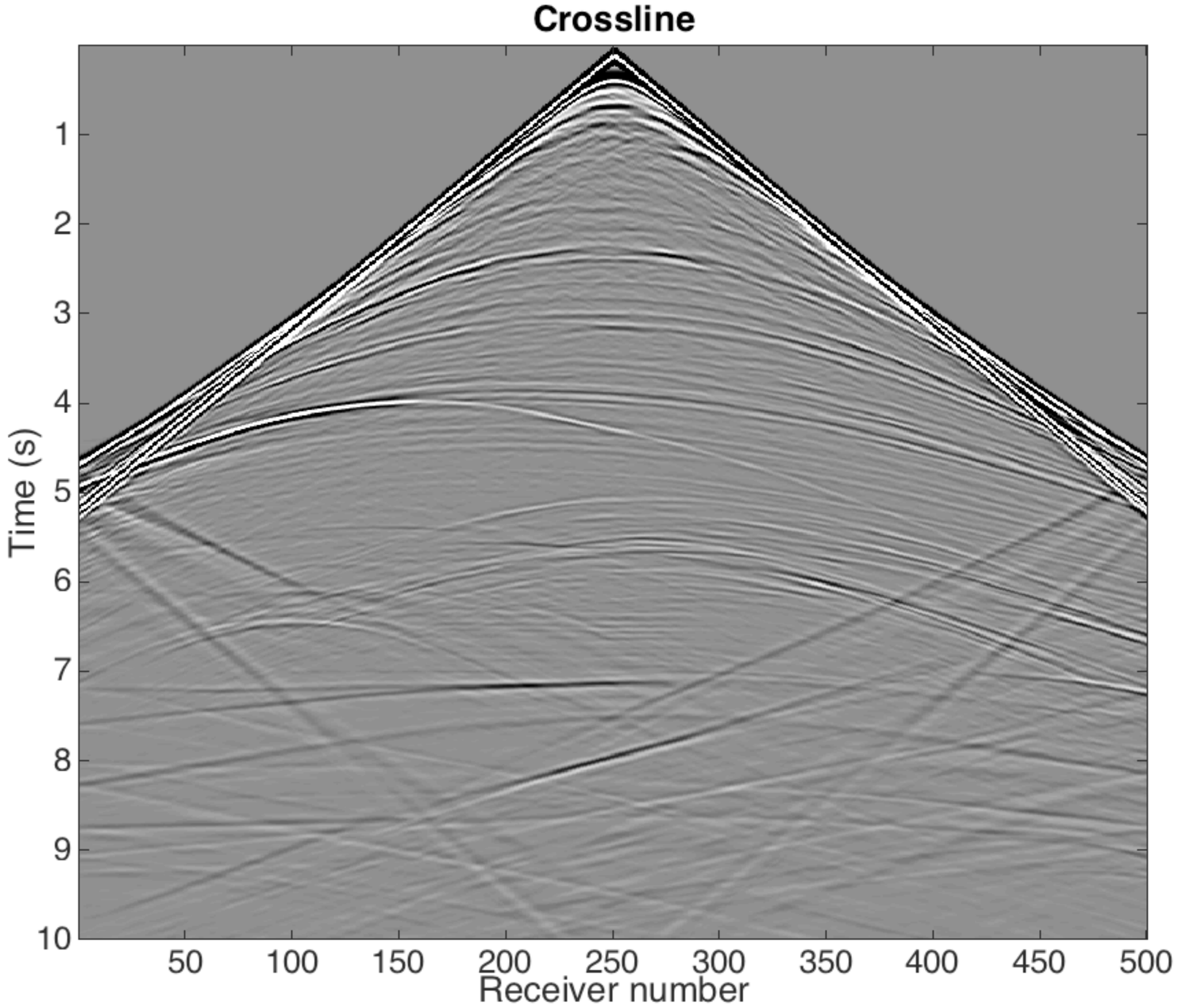
Model

- 1500x1500x1050 model (+80 ABC)
- 1480 to 4500 km/s velocity
- 10th order discretization
- 10Hz Ricker wavelet at (900, 900, 152)
- 10m Grid
- Receivers every 40m at 40m depth

Timings

- **Chevron code : 8h14**
- **OPESCI : 7h22 (10% faster)**


Inline
Time (s)
1
2
3
4
5
6
7
8
9
10
50
100
150
200
250
300
350
400
450
500
Receiver number


Crossline
Time (s)
1
2
3
4
5
6
7
8
9
10
50
100
150
200
250
300
350
400
450
500
Receiver number

Currently supported

Acoustic

1. Forward/Adjoint
2. Jacobian/Jacobian adjoint (demigration/migration)
3. Dipole sources Forward modelling
4. Applying the PDE/Adjoint PDE to a wavefield

TTI

5. Forward/Adjoint

Future work

Extend all acoustic operators to TTI

Continuous acquisition modelling

Better boundary conditions

A few links

http://www.opesci.org/

http://www.opesci.org/devito/

https://github.com/opesci/devito

Acknowledgements

This research was carried out as part of the SINBAD project with the support of the member organizations of the SINBAD Consortium and the Imperial College London Intel Parallel Computing Centre.