awivc

December 13, 2022

1 Adaptive Waveform Inversion with Vector Classes

1.1 Overview

Adaptive Waveform Inversion (AWI) is (in some ways) similar to the approach we worked out to the single trace transmission problem. It is based on an extension that's like one single trace transmission for every source-receiver pair: that is, every such pair gets its own wavelet, and the goal is to concentrate the wavelet near $t = 0$ as much as possible. There are several notable differences. First, AWI presumes that a "true" wavelet, that generates the data, is known. So it's possible to focus on an "adaptive" kernel that produces the extended wavelet by convolution with the known "true" wavelet. This adaptive kernel really should be $\delta(t)$ at the correct model, so multiplication by t is really an annihilator, without having to make a "small support" assumption. We could have done this as well, but of course we got an estimate of the wavelet for our trouble, without having to assume a known "true" wavelet. Second, AWI deals with multidimensional models, and many traces, not just one.

There is some understanding to be gained by applying the methodology we developed to this larger scale problem. First, I'll review AWI as it is presented in the literature, along with a computational approach that makes it practical and an example. Next comes recasting AWI into the penalty form that we used in studying the single trace transmission problem, and a re-interpretation of a normalization as a preconditioner. Applied to transmission problems, I believe that AWI is not immune from stagnation at useless estimates - that was the motivation for Surface Source Extension inversion, which is a generalization. Finally, I do not believe that any of these techniques are any better than FWI when applied to reflection data. So there are two interesting negative results to be had. However we will also be able to explain exatly why AWI works, when it does work.

1.2 Warner-Guasch formulation of AWI

The version of AWI introduced by Michael Warner and Lluis Guasch (SEG 2014 and *Geophysics* 2016) assumes that seismic waves are governed by linear acoustics, and that each shot is associated to an isotropic point source with known location and wavelet. That is, the pressure and velocity fields $p(\mathbf{x}, t; \mathbf{x}_s)$, $\mathbf{v}(\mathbf{x}, t; \mathbf{x}_s)$ for the shot location $\mathbf{x}_s$ depend on the bulk modulus $\kappa(\mathbf{x})$, buoyancy $\beta(\mathbf{x})$ (reciprocal of the density $\rho(\mathbf{x})$), and wavelet $w(t; \mathbf{x}_s)$ through the acoustic system

$$\frac{\partial p}{\partial t} = -\kappa \nabla \cdot \mathbf{v} + w(t; \mathbf{x}_s)\delta(\mathbf{x} - \mathbf{x}_s); \frac{\partial \mathbf{v}}{\partial t} = -\beta \nabla p; p, \mathbf{v} = 0 \text{ for } t \ll 0.$$

For sake of brevity, define the model vector $m = (\kappa, \rho)$. The forward map or *modeling operator* is $F[m]w = \{p(\mathbf{x}_r, t; \mathbf{x}_s)\}$, in which shot and receiver positions $\mathbf{x}_s, \mathbf{x}_r$ define the acquisition geometry.

For now, assume that the data $d(\mathbf{x}_r, t; \mathbf{x}_s)$ is the output of the modeling operator for "true" model m_*, that is, the "true" bulk modulus, buoyancy, and wavelet κ_*, β_*, w_*: that is, $d = F[m_*]w_*$.

The extended modeling operator $\bar{F}$ maps extended sources $\bar{w}(\mathbf{x}_r, t; \mathbf{x}_s)$ to the same sampling of the pressure field. That is, the extended source depends on the receiver location as well as the source location - so there is one acoustic system for each source *and* receiver position - a lot of wave equations! If all of the wavelets for each source are the same, that is, $\bar{w}(\mathbf{x}_r, t; \mathbf{x}_s) = w(t; \mathbf{x}_s)$ is independent of receiver position, then $\bar{F}[m]\bar{w} = F[m]w$. That is, F is a special case, or restriction, of $\bar{F}$, so $\bar{F}$ is an extension of F. This is the *source-recever* extension, in the terminology of Huang et al. 2015.

AWI assumes that the extended sources are time convolutions of the (known) exact source with a kernel $\bar{u}(\mathbf{x}_r, t; \mathbf{x}_s)$: $\bar{w} = \bar{u} * w_*$ - the asterisk denotes convolution in time. Since linear acoustics is time-translation invariant, its solution commutes with time convolution, that is,

$$\bar{F}[m]\bar{w} = \bar{F}[m](\bar{u} * w_*).$$

With this set-up, the object of inversion can be formulated as:

Given d, find $m = (\kappa, \beta)$ and $\bar{w}$ so that $u(\mathbf{x}_r, t; \mathbf{x}_s) = \delta(t)$ (so $\bar{w} = w_*$) and $\bar{F}[m]\bar{w} = F[m]w_* \approx d$.

Warner and Guasch assume (implicitly) that there is always a (near-)zero-residual solution of the extended inversion problem. That is, they assume that for any m, there is a $\bar{w}$ for which $\bar{F}[m]\bar{w} \approx d$. If $\kappa \approx \kappa_*, \beta \approx \beta_*$, then $\bar{w}$ should be $\approx w_*$, so the adaptive kernel $\bar{u}$ should be approximately $\delta(t)$ and independent of $\mathbf{x}_s, \mathbf{x}_r$. Such a $\bar{u}$ is (approximately) in the null space of multiplication by t. Thus a first version of the AWI algorithm:

1. Given m, solve the problem $\bar{F}[m]\bar{w} = d$ for $\bar{w}$.
2. Deconvolve w_* from $\bar{w}$ to obtain $\bar{u}$ for which $\bar{w} = u * w_*$.
3. Compute the objective
$$J_0[m] = \int dx_s dx_r \left(\frac{\int dt |t\bar{u}|^2}{\int dt |\bar{u}|^2} \right)$$
and its gradient, then update κ, β by some gradient descent method. (The role of the normalization per source and receiver by the L^2 norm of $\bar{u}$ will be explained later.)

There are two things wrong with this algorithm. First, it involves way too many wave equation solves. Second, it depends on the assumption that you can fit the data exactly (which may or may not be the case, see examples to follow). Warner and Guasch fix the first problem. The second is fundamental, as will be shown below.

To solve the first problem, observe that the extended pressure field $\bar{p}$ is the convolution of the Green's function $G[m]$ (solution of the acoustic system above with $w(t, \mathbf{x}_s) = \delta(t)$) with the source function $w(t, \mathbf{x}_s)$. Since this source function is independent of the receiver position, only one wave equation needs to be solved for each source position, same as for non-extended modeling. Accordingly,

$$\bar{F}[m]\bar{w} = G * \bar{w}$$

Convolution is far cheaper than wave equation solve: thus the cost of computing the forward map is reduced to one wave equation solve per source, and one convolution per source and receiver.

On top of that, denote the convolution inverse of G by $\check{G}$, and the convolution inverse of w_* by $\check{w}_*$. Neither of these things may actually have a convolution inverse, strictly speaking. A solution of

a regularized least-squares problem can play the same role. However these things are also cheaply approximated via discrete Fourier transform, which is what Warner and Guasch do.

Then

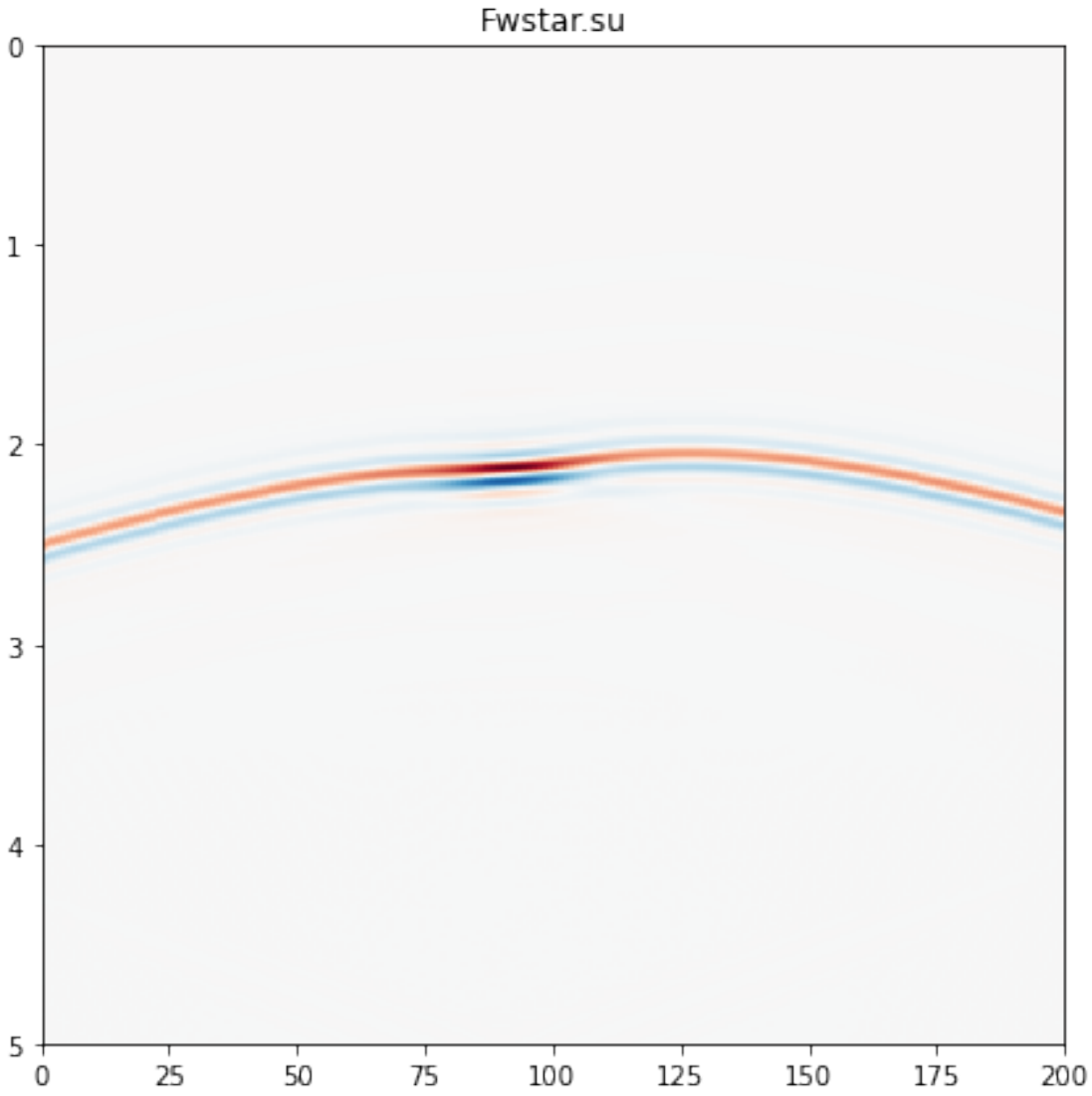
$$\bar{w} = \check{G} * d, \bar{u} = \check{w}_* * \bar{w} = \check{w}_* * \check{G} * d$$

and the first two steps in the above AWI algorithm can be replaced by inexpensive convolutions. This is the second version of the AWI algorithm. It is inexpensive enough to be employed on 3D data at field scale.

1.3 Example, part 1

To begin with, let's show that the source-receiver extension, really is an extension. First create model and data as in the "Python interface to IWAVE" notebook. The bulk modulus has a somewhat weaker acoustic lens than the example in that notebook. The same point-source bandpass wavelet generates a pressure gather over 201 receivers spaced 20 m apart.

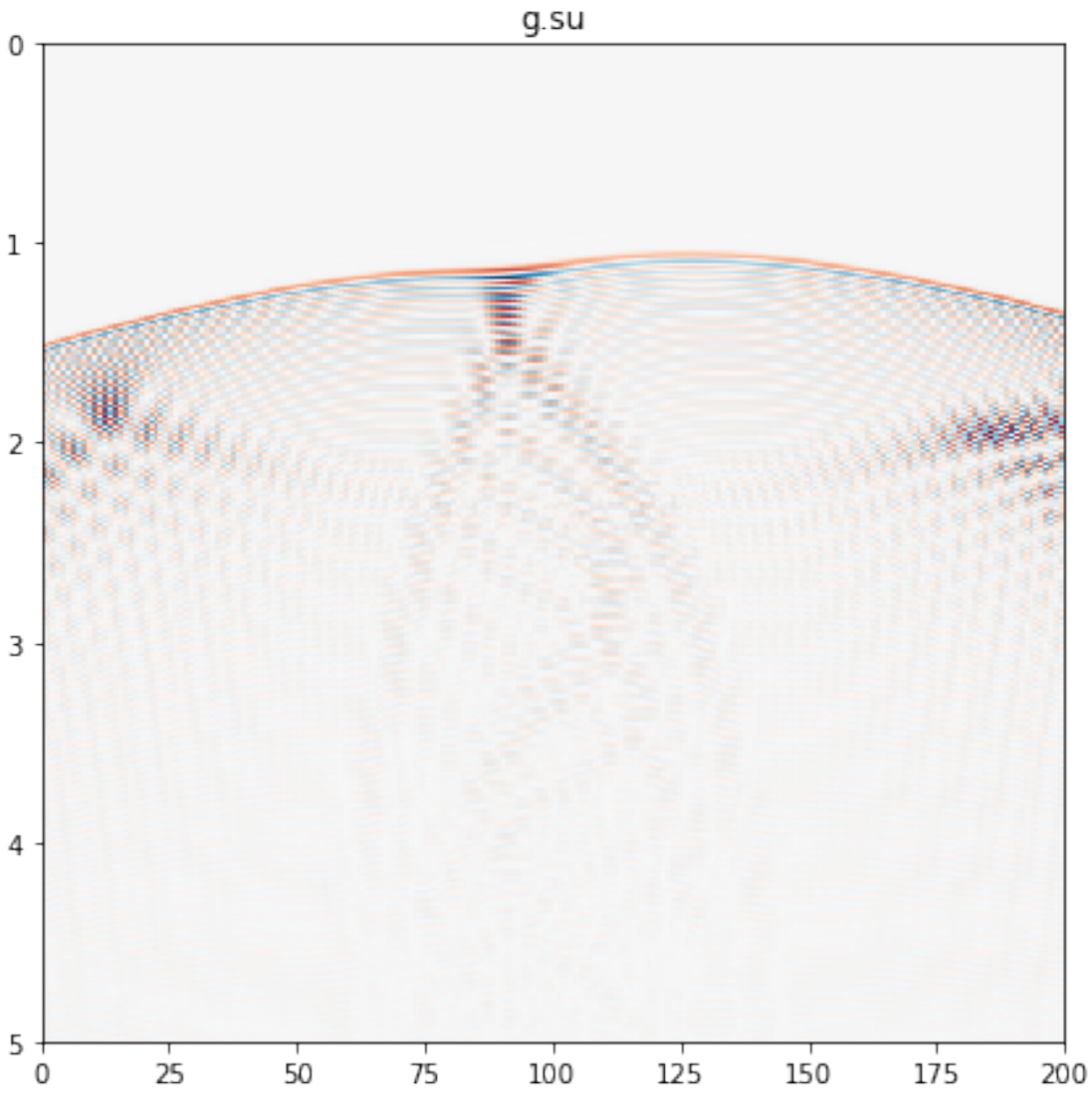
[1]:
```
import linalg
import op
import data
import vcl
import segyvc
import cg

# bulk modulus with less focussing lens
data.model(bulkfile='m.rsf', bulk=4.0, nx=401, nz=201, dx=20, dz=20, lensfac=0.
 ↪7)

# bandpass filter source at sx=4200, sz=3000 (single trace)
data.bpfilt(file='wstar.su',nt=251,dt=8.0,s=1.0,f1=1.0,f2=2.5,f3=7.
 ↪5,f4=12,sx=4200,sz=3000)

# create zero data file with same source position, rz=500, rx=[2000,6000]
data.rechdr(file='g.su',nt=626,dt=8.
 ↪0,rxmin=2000,rxmax=6000,ntr=201,rz=1000,sx=4200,sz=3000)

# create delta function at sx=4200, sz=3000
data.delta(file='delta.su', nt=201, dt=8.0, sx=4200, sz=3000)

# copy rechdr onto file to store simulation output
linalg.copy('g.su','Fmwstar.su')

# apply fd operator - not VC object yet
op.fdop(m='m.rsf', w='wstar.su', d='Fmwstar.su')

# plot 'Fwstar.su'
linalg.simplot('Fwstar.su')
```




```
simplot: data min = -7.3255e-06, data max = 8.6175e-06
```

To see that $F[m]w_*$ is also the result of extended modeling, create a Green's function for the source and receiver positions - this requires a zero-phase delta function source at the correct location.

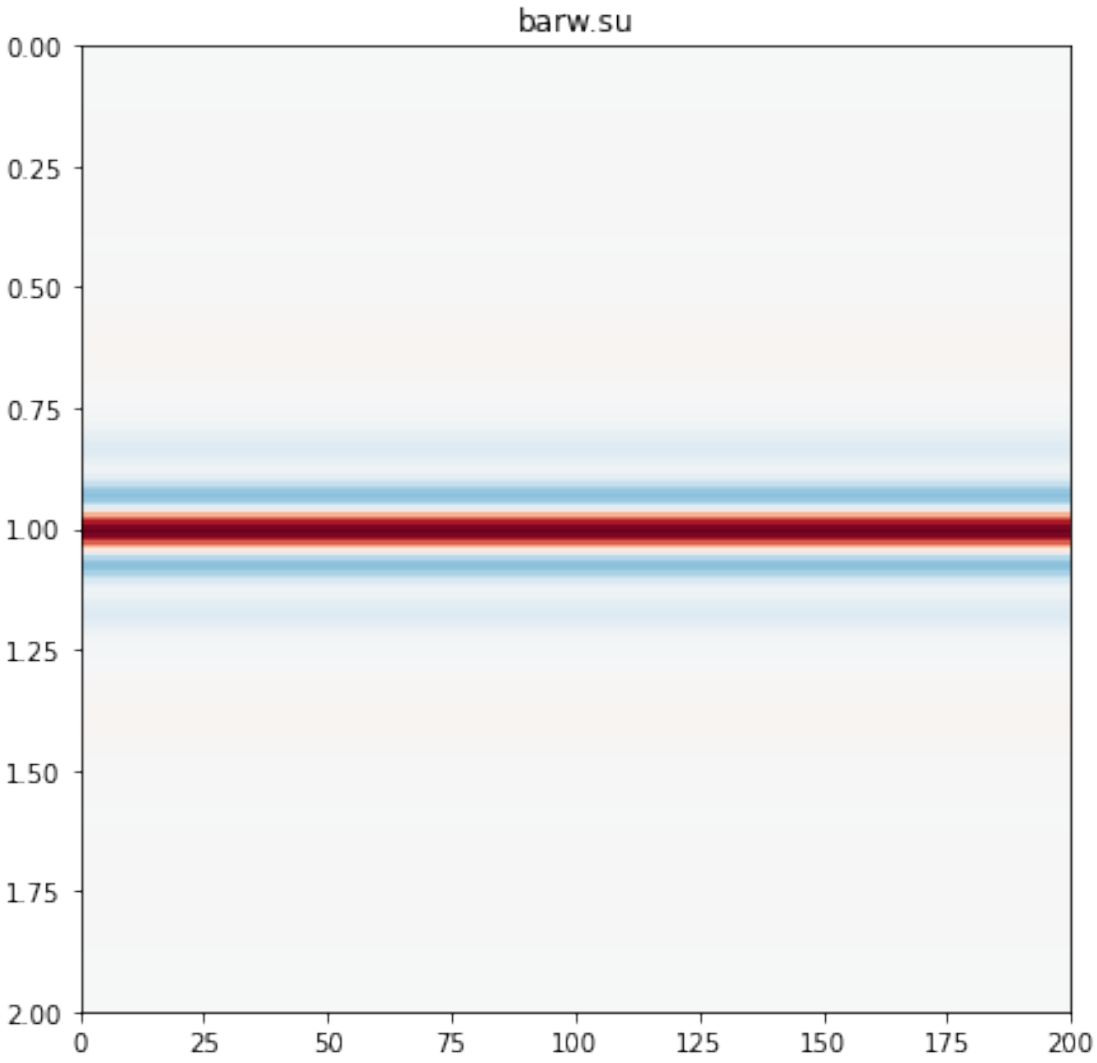
[2]:
```
# create delta function at sx=4200, sz=3000
data.delta(file='delta.su', nt=201, dt=8.0, sx=4200, sz=3000)

# apply fd op to obtain green's fucntion
op.fdop(m='m.rsf',w='delta.su',d='g.su')

# take a look
linalg.simplot('g.su')
```




```
simplot: data min = -3.3352e-05, data max = 3.0076e-05
```

Create an extended source, replicating the bandpass filter for each trace (receiver position) by creating an extended source gather with the same source/receiver locations as the data, replicating the single-trace bandpass filter in every trace.

[3]:
```
# extended source headers
data.bpfiltgather(file='barw.su',nt=251,dt=8.0,s=1.0,f1=1.0,f2=2.5,f3=7.
↪5,f4=12,ntr=201,sxstart=2000,szstart=6000,dsx=20,dsz=0)

# plot
linalg.simplot('barw.su')
```



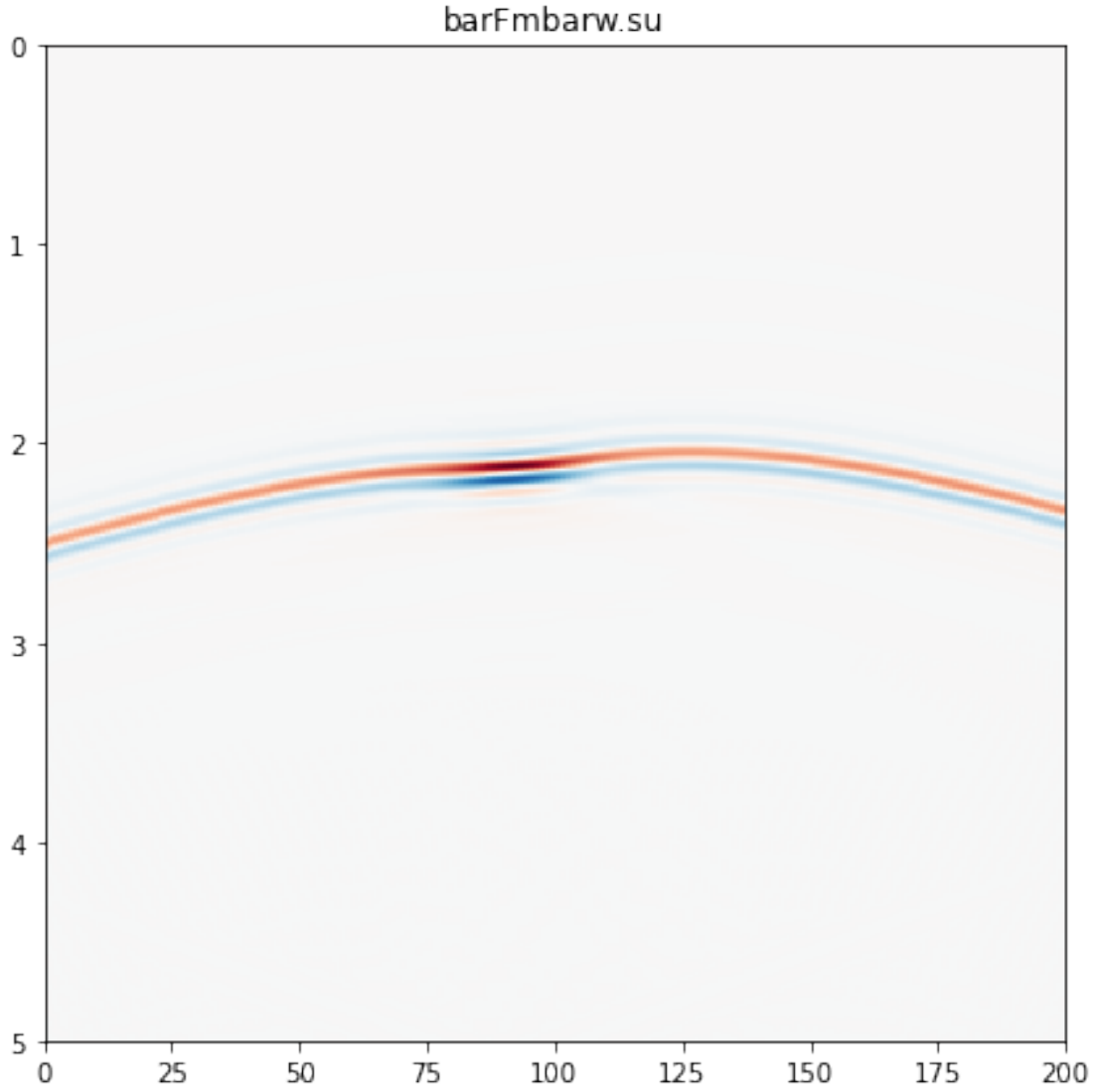

```
simplot: data min = -6.6359e-03, data max = 1.5873e-02
```

To apply $\bar{F}[m]$, simply use *the same* Green's function as used in convolutional modeling, but with an extended source, that is, a gather of source traces, one per receiver position. barw.su is such a gather. The function op.convop(g,w,d,adj) is set up to recognize automatically that there is one input trace per receiver (that is, one input trace per trace in the Green's function) and perform the correct convolution, writing the result to the output trace

In this case, 'barw.su' is the extended source that produces the same output as the single-trace source 'wstar.su', illustrating that $\bar{F}[m]$ is an extension of $F[m]$:

[4]:
```
linalg.copy('g.su','barFmbarw.su')
op.convop(g='g.su',w='barw.su',d='barFmbarw.su',adj=0)
linalg.simplot('barFmbarw.su')
linalg.copy('barFmbarw.su','diffbarFmbarw.su')
```

```
linalg.lincomb(-1.0,'Fmwstar.su','diffbarFmbarw.su')
dnorm = linalg.norm('Fmwstar.su')
diffnorm = linalg.norm('diffbarFmbarw.su')
print('norm of barFbarw.su - Fmwstar.su = %10.4e' % (diffnorm))
print('relative error = %10.4e' % (diffnorm/dnorm))
```




```
simplot: data min = -7.3256e-06, data max = 8.6175e-06
norm of barFbarw.su - Fmwstar.su = 2.9438e-07
relative error = 5.8399e-04
```

1.4 Least squares formulation

Recall the convolutional representation of $\bar{F}$:

$$\bar{F}[m]\bar{w} = G[m] * \bar{w}$$

also the relation between the adaptive kernel $\bar{u}$, (known) source wavelet w_*, and extended source $\bar{w}$:

$$\bar{w} = \bar{u} * w_*$$

Putting these together and using the commutativity of convolution,

$$\bar{F}[m]\bar{w} = (G[m] * w_*) * \bar{u} = (F[m]w_*) * \bar{u}$$

In other words, the predicted *non-extended* data $F[m]w_*$ is the convolution kernel mapping the adaptive kernel $\bar{u}$ to predicted *extended* data $\bar{F}[m]\bar{w}$.

This observation suggests reformulating the inverse problem directly in terms of the adaptive kernel $\bar{u}$: that is, find $\bar{u}$ so that

$$\bar{S}[m, w_*]\bar{u} \equiv (F[m]w_*) * \bar{u} = \bar{F}[m](\bar{u} * w_*)$$

approximates the data. Here $\bar{S}[m, w_*]$ is the operator of convolution with the *non-extended* predicted data $F[m]w_*$.

The straightforward least squares formulation would be: given m, w_*, d, find $\bar{u}$ to minimize (approximately)

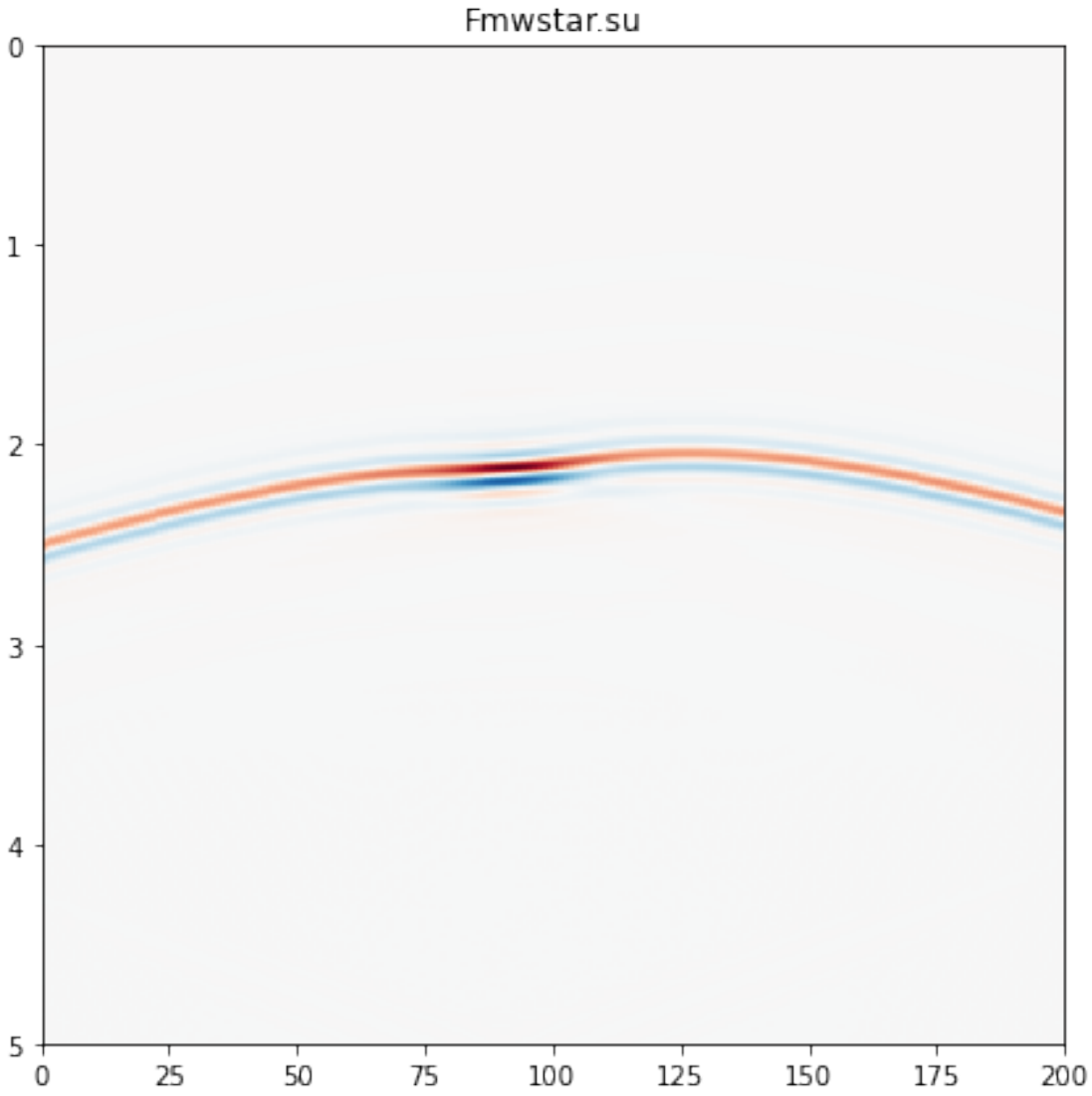
$$J_0[m, \bar{u}; w_*, d] = \|\bar{S}[m, w_*]\bar{u} - d\|^2$$

1.5 Example, part 2

Up to this point it's been convenient to use simple functions that perform linear algebra and mapping operations, as in the linalg and op modules, without enforcing strict parallelism with the corresponding mathematics. From here on, this ad hoc approach starts to obscure the program logic. Use of vector calculus classes leads to much a much clearer transformation from mathematics to code.

Begin by treating the simulator output 'Fwstar.su' as a vector. The module *segyvc.py* defines a vector interface for SEGY files, following the vector calculus classes in *vcl.py*. To wrap 'Fwstar.su' as a vector in the sense of these classes, create a *segyvc.Space*, using a SEGY file having the same headers (time samples, source and receiver positions, etc. - the same metadata). Then instantiate a vector in this space, and use the link method of the *vcl.Vector* class to wrap the simulator output as this vector.

[5]:
```
dsp = segyvc.Space('g.su')
Fmwstar = vcl.Vector(dsp)
Fmwstar.link('Fmwstar.su')

# plot data member of Fwstar - which is 'Fmwstar.su'
linalg.simplot(Fmwstar.data)
```




```
simplot: data min = -7.3255e-06, data max = 8.6175e-06
```

The domain space of $\bar{S}[m, w_*]$ consists of as many traces as the data or the Green's function, but presumably with a time range containing $t = 0$. Since the ideal input is $\bar{u} = \delta(t)$, construct a gather consisting of 201 traces of length 2 s with the firast sample at $t = -1$ s to serve as the data of a *segyvc.SEGYSpace*

[6]:
```
data.rechdr(file='baru.su',nt=251,dt=8.
↪0,rxmin=2000,rxmax=6000,ntr=201,rz=1000,sx=4200,sz=3000,delrt=-1000)
```

Then build an input *segyvc.Space* based on this file, and a *vcl.Vector* in it. For economy of storage, use the link method to wrap 'baru.su' in this Vector:

[7]:
```
usp=segyvc.Space('baru.su')
baru=vcl.Vector(usp)
baru.link('baru.su')
```

The VCL version of the conjugate gradient algorithm accepts only VCL objects - Vectors, Operators, scalars - as input. It differs from the version laid out in the *cg0* module particularly in that it is passed an Operator created externally, whereas the *cg0* version uses its filename arguments to call the *op.convop* function - which therefore limits *cg0* to solving least squares convolution problems using SEGY files as data. The *cg* version, on the other hand, is not constrained in that way. As shown in the *vcl.ipynb* notebook, it is solves matrix-vector least squares problems set up in a VCL wrapper around NumPy, and indeed any such problem formulated with the VCL abstractions.

So the next step is to construct an Operator class (*segyvc.ConvolutionOperator*) wrapper around *op.convop*, to simulate $\bar{S}[m, w_*]$. Note that a *vcl.Operator* "knows" its domain and range on construction, before its action on a vector is required. That's different from a function like *op.convop*, and requires a few more lines of code and some sanity checking (internal to *segyvc.ConvolutionOperator*).

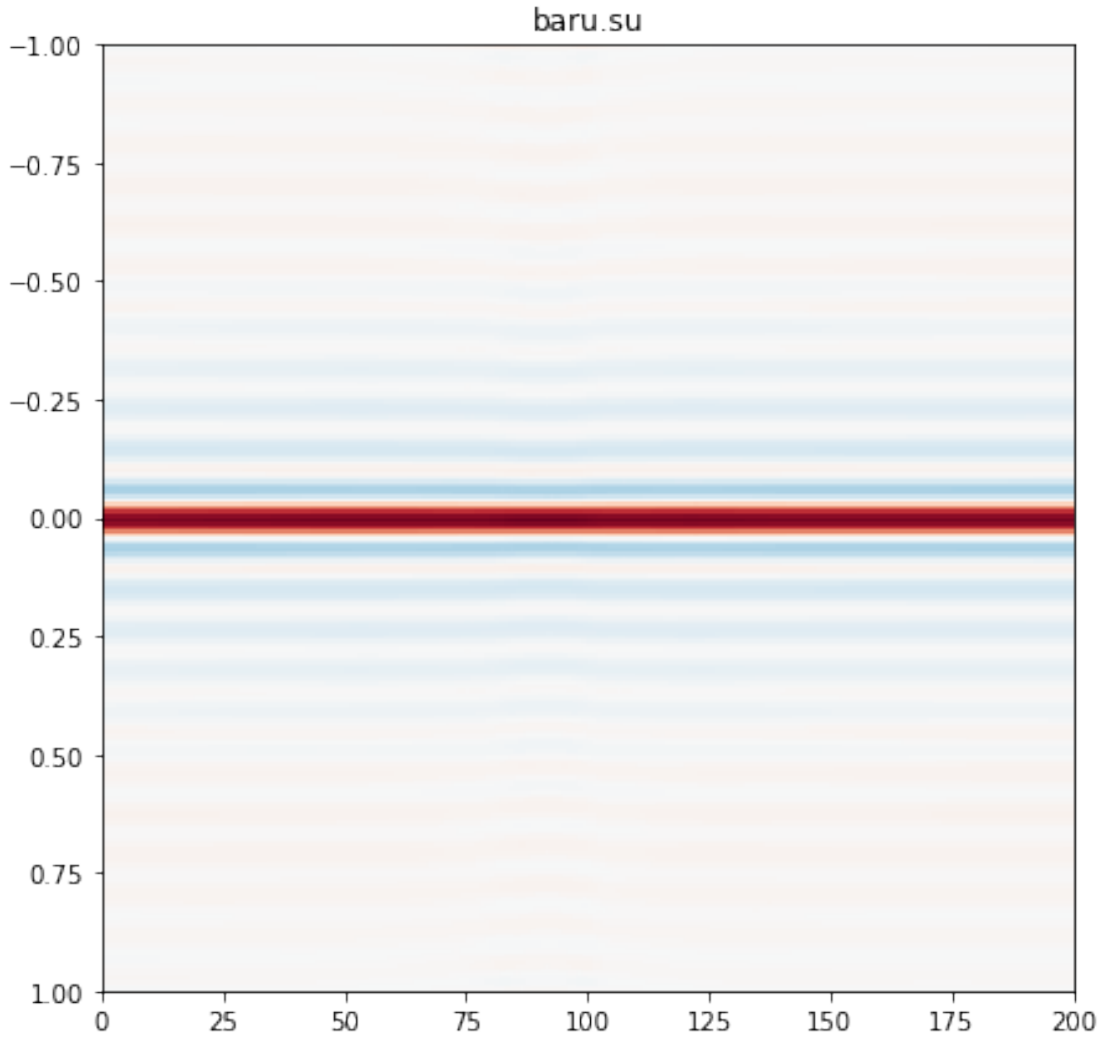
[8]:
```
Smwstar = segyvc.ConvolutionOperator(dom=usp, rng=dsp, green=Fmwstar.data)
```

Note that the last argument is the file name of the non-extended predicted data 'Fmwstar.su', which is the data object managed by *Fwstar*. This is typical of operator construction: some inputs are VCL class objects (like the domain and range spaces), while others will be other types of objects, whatever is necessary to carry out the internal processes of the operator.

The module *cg.py* defines a VCL version of CG for normal equations, exactly analogous to *cg0.conjgrad*. As in that function, the residual and normal residual (*vcl.Vector* objects, now) are passed as arguments, to facilitate examination afterwards.

[9]:
```
# max allowed iterations
kmax = 50
# residual norm reduction for termination
eps  = 0.01
# normal residual norm reduction for termination
rho  = 0.001
# residual
e    = vcl.Vector(dsp)
# normal residual
r    = vcl.Vector(usp)

try:
    cg.conjgrad(x=baru, b=Fmwstar, A=Smwstar, kmax=kmax, eps=eps, rho=rho, e=e,␣
 ↪r=r, verbose=1)
except Exception as ex:
    print(ex)
else:
    linalg.simplot(baru.data)
```

```
 k        |e|      |e|/|e0|       |r|         |r|/r0|
50   5.0321e-06   9.9827e-03   3.9392e-10   2.1044e-03
```




```
simplot: data min = -7.1111e-03, data max = 2.2039e-02
```

The waveform is perfectly symmetric, independent of receiver location (trace index), and more focused at $t = 0$ than the extended source 'barw.su', plotted above. Convolving 'baru.su' with 'wstar.su' would produce a close approximation of 'barw.su'.

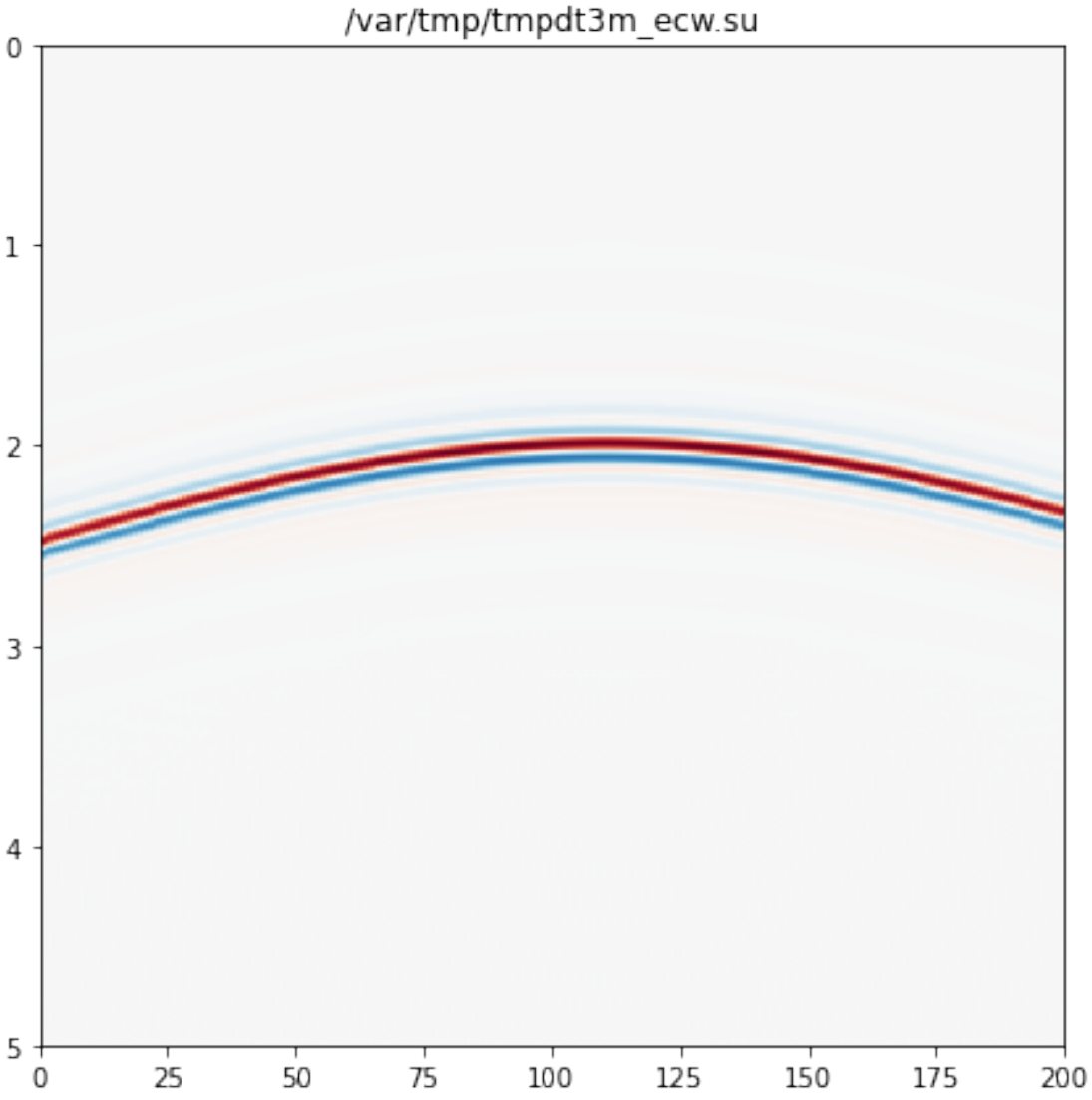
By sheer luck, the relative error has just dropped below 1% at 50 iterations. If you plot the residual offline on the grey scale of the data (beyond the current capability of *linalg.simplot*), you can't really see it in most renderings. This is excellent data fit.

The next hypothesis to test is whether the extended model can fit the data well for any model m, correct or not. For example, we could use a constant bulk modulus $m0$ with $\kappa = 4.0$, rather than the field with the lens that created the data 'Fmwstar.su'. A VCL implementation starts with FD computation of the non-extended simulation with inputs $m0$, represented by the RSF file 'm0.rsf',

and the point source 'wstar.su'. The FD operator is not (yet) implemented as a VCL operator object.

Note that since we have already defined the data space *dsp*, it's just as easy to create the non-extended predicted data *Fm0wstar* as a VCL Vector, accessing its data object (SEGY file name) as needed to pass to non-VCL functions.

[10]:
```
# homog bulk modulus = 4.0 GPa
data.model(bulkfile='m0.rsf', bulk=4.0, nx=401, nz=201, dx=20, dz=20, lensfac=1.
 ↪0)

# compute non-extended predicted data d0
Fm0wstar = vcl.Vector(dsp)
op.fdop(m='m0.rsf', w='wstar.su', d=Fm0wstar.data)

linalg.simplot(Fm0wstar.data)
```



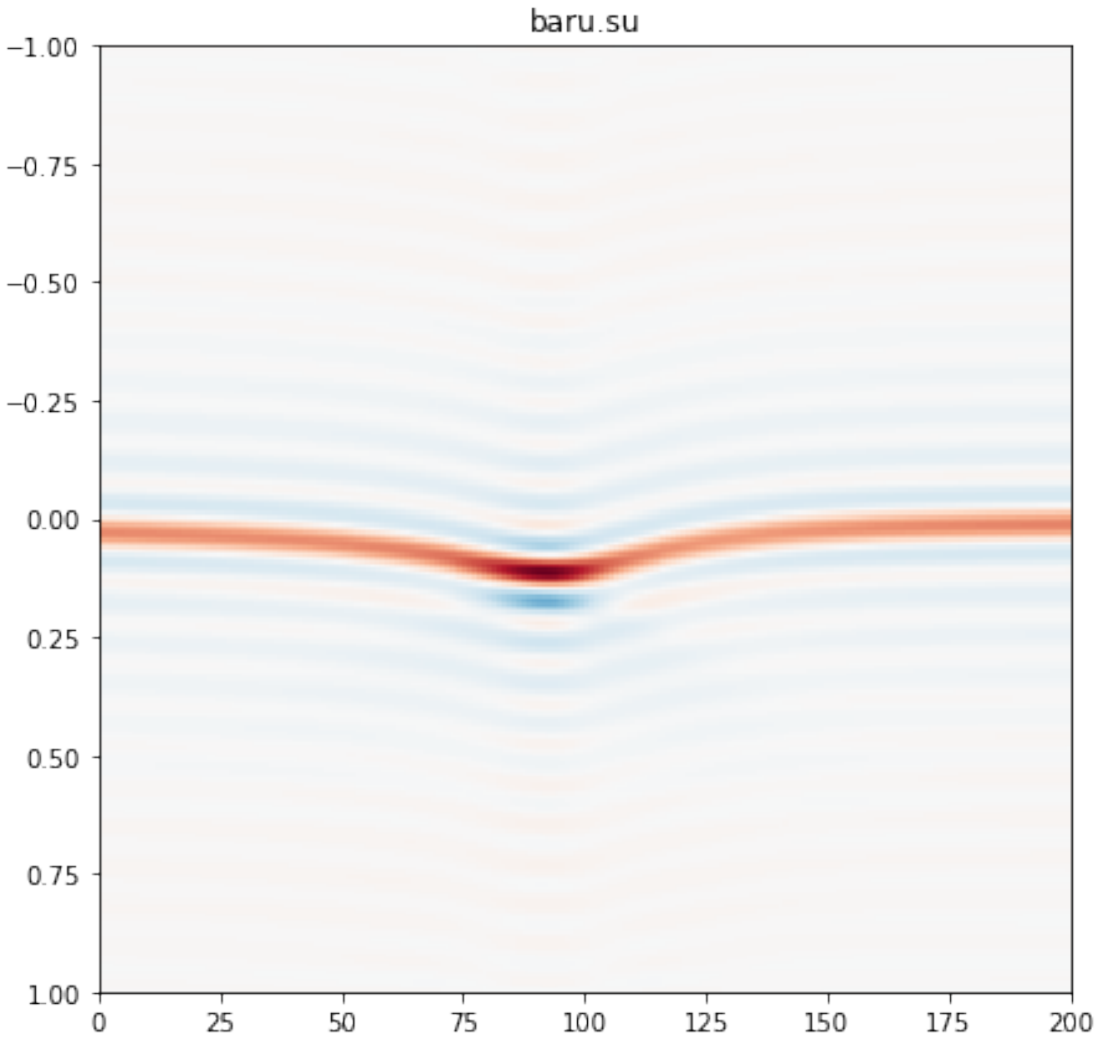
```
simplot: data min = -3.2286e-06, data max = 4.4665e-06
```

Note from the plot title that the file wrapped in the vector *Fm0wstar* is a temporary file, with a system-generated name. On exit from the Python process in which this notebook code is running, all such temporary files should be unlinked (deleted from the file system). I have not used the link method to replace this temporary file with a specified name, since there is no reason to make the data persistent beyond this process.

Now estimate an adaptive kernel *baru* that fits the data *Fmwstar* but with the *wrong* bulk modulus (hence wrong velocity or slowness). The modeling operator *Sm0wstar* emulates $\bar{S}[m_0, w_*]$. All of the auxiliary parameters of the CG process are the same, and I have simply overwritten the adaptive kernel vector *baru* with the new data.

[12]:
```
Sm0wstar = segyvc.ConvolutionOperator(dom=usp, rng=dsp, green=Fm0wstar.data)

cg.conjgrad(x=baru, b=Fmwstar, A=Sm0wstar, kmax=kmax, eps=eps, rho=rho, e=e,␣
 ↪r=r, verbose=1)

linalg.simplot(baru.data)
```

```
-----------------------------------------------------
  k       |e|      |e|/|e0|      |r|        |r|/r0|
 25  5.0108e-06  9.9405e-03  2.1586e-10  1.5757e-03
```




```
simplot: data min = -2.1221e-02, data max = 4.2119e-02
```

Two major things to notice: first, even though the bulk modulus (hence slowness, velocity) is incorrect, the data is fit. The relative error (enorm/enorm0 in the program output) is less than 0.01, at 25 iterations. This illustrates the assumption underlying AWI (and all other working extension methods: data can always be fit, even with the wrong model. The second thing to note is that the energy in the adaptive kernel baru.su computed with the homogeneous bulk modulus is delayed: it appears to arrive approximately 0.1 s late in the middle traces (80-120). This indicates that the model responsible for generating the data is slower in that region, which is correct (it has a slow lens in the middle). Measuring this delay by the norm-squared of the multiply-by-t operator applied to the adaptive kernel is the basic idea of AWI, and is very close in spirit to our treatment of the single-trace problem.

1.6 Relation with penalty functions

Recall the AWI objective function: with the adaptive kernel $\bar{u}$ chosen so that $\bar{S}[m, w_*]\bar{u} = d$,

$$J_0[m, w_*] = \sum_{x_s, x_r} \left(\frac{\int dt |t\bar{u}|^2}{\int dt |\bar{u}|^2} \right)$$

This function is the $\alpha \to 0$ limit of a reduced penalty function:

$$J_0[m, w_*] = \lim_{\alpha \to 0} \frac{1}{\alpha^2} \tilde{J}_\alpha[m, w_*]$$

where

$$\begin{aligned} \tilde{J}_\alpha[m, w_*] &= \min_u J_\alpha[m, w_*, u] \\ &= J_\alpha[m, w_*, \tilde{u}_\alpha[m, w_*]] \end{aligned}$$

with

$$J_\alpha[m, w_*, u] = \frac{1}{2}(\|\bar{S}[m, w_*]u - d\|^2 + \alpha^2 \|Tu\|_m^2)$$

Here T is the multiply-by-t operator, and $\| \cdot \|_m$ denotes a weighted norm:

$$\begin{aligned} \|u\|_m^2 &= \sum_{x_s, x_r} \left(\frac{\int dt |u|^2}{\int dt |\tilde{u}_0[m, w_*]|^2} \right) \\ &= \|W[m, w_*]u\|^2 \end{aligned}$$

The weight operator $W[m, w_*]$ multiplies the trace at x_s, x_r by the reciprocal norm of the corresponding trace of $\tilde{u}_0[m, w_*]$, the solution of

$$\tilde{u}_0[m, w_*] = \text{argmin}_u \|\bar{S}[m, w_*]u - d\|^2$$

Assuming that the minimum value on the RHS is actually zero, evidently the adaptive kernel $\bar{u}$ in the definition at the top of this paragraph is the same as $\tilde{u}_0[m, w_*]$. In general, some regularization is probably required for a proper definition of $\tilde{u}_0[m, w_*]$. It is the solution of a deconvolution problem. In the deconvolution context, (Tihonov) regularization is known as "pre-whitening". Warner and Guasch mention pre-whitening as part of their computations, though they do not give details.

1.7 Why AWI works, and why the weighted norm is important

To see why AWI works, in at least some settings, assume that the spatial dimension is three. The 2D case has a similar conclusion but with somewhat more technical difficulty.

For spatial position $\mathbf{x}$ "not too far" from a source location $\mathbf{x}_s$, the Green's function has a useful decomposition:

$$G[m](\mathbf{x}, \mathbf{x}_s, t) = a[m](\mathbf{x}, \mathbf{x}_s)\delta(t - \tau[m[(\mathbf{x}, \mathbf{x}_s)) + R[m](\mathbf{x}, \mathbf{x}_s, t).$$

This is the *geometric optics* (or *progressing wave*) decomposition. Assuming that the coefficients in the wave equation are smooth, the *amplitude* $a[m]$ and *travel time* $\tau[m]$are smooth, positive and well-defined in a punctured neighborhood $0 < |\mathbf{x} - \mathbf{x}_s| < r$ of the source point $\mathbf{x}_s$. The *remainder* $R[m]$ is smooth except for a jump discontinuity at $t = \tau[m](\mathbf{x}, \mathbf{x}_s)$, hence less singular than the leading δ term. It is "causal", that is, $= 0$ for $t < \tau$. In this punctured ball of radius $r > 0$, the

decomposition of $G[m]$ stated above holds, but generally fails at distance $> r$ from the source. The radius r depends on the slowness field: it is $= \infty$ for constant slowness, and is smaller for more oscillatory slowness. For a complete treatment of geometric optics as stated here, see Friedlander, *The wave equation on curved space-time*, Cambridge University Press, 1975. It is also described in Courant and Hilbert, *Methods of Mathematical Physics*, v. 2, Ch. VI, section 4, and in many other texts.

I will start by investigating the behaviour of the Warner-Guasch objective using the geometric optics tool, then turn to the penalty function. The analysis that follows is formal, in that I do not explicitly bound the error contributions from the remainder term $R[m]$ in any of the following expressions, acknowledging them by using the symbol $\approx$ rather than equality. Also, I tacitly assume that the geometric optics decomposition holds at all of the spatial locations of interest, that is,

$$G[m](\mathbf{x}, \mathbf{x}_s, t) \approx a[m](\mathbf{x}, \mathbf{x}_s)\delta(t - \tau[m[(\mathbf{x}, \mathbf{x}_s)).$$

I will eventually revisit this assumption, as one of the main points to be made is what happens to AWI when it fails.

Start by examining a particular source-receiver pair $\mathbf{x}_s, \mathbf{x}_r$, suppress from the notation.

$$F[m]w_*(t) \approx a[m]w_*(t - \tau[m])$$

so

$$S[m, w_*]u(t) \approx a[m](w_* * u)(t - \tau[m])$$

Assume that the data is noise-free:

$$d(t) = F[m_*]w_*(t) \approx a[m_*]w_*(t - \tau[m_*])$$

The (approximate) solution of $S[m, w_*]u = d$ is obvious by inspection: $u(t) = a[m_*]/a[m]\delta(t - \tau[m_*] + \tau[m])$. There is immediately a big problem: this u is not square-integrable, and the AWI objective defined above is undefined. To fix this, it's necessary to treat u as the solution of a regularized least squares problem:

$$u_\epsilon = \text{argmin}_u(\|S[m, w_*]u - d\|^2 + \epsilon^2\|u\|^2)$$

$$\approx \text{argmin}_u(\|a[m](w_* * u)(\cdot - \tau[m]) - a[m_*]w_*(\cdot - \tau[m_*])\|^2 + \epsilon^2\|u\|^2)$$

Use the temporary abbreviations $a_* = a[m_*], a = a[m], \tau_* = \tau[m_*], \tau = \tau[m]$, and write in terms of the Fourier transforms $\hat{u}_\epsilon, \hat{w}_*$ of u_ϵ and w_*:

$$\hat{u}_\epsilon \approx \text{argmin}_{\hat{u}} \left(\int d\omega |a\hat{w}_*\hat{u}e^{i\omega\tau} - a_*\hat{w}_*e^{i\omega\tau_*}|^2 + \epsilon^2|\hat{u}|^2 \right)$$

The normal equation is

$$(a^2|\hat{w}_*|^2 + \epsilon^2)\hat{u} = aa_*|\hat{w}_*|^2e^{i\omega(\tau_* - \tau)}$$

the solution of which is

$$\hat{u}_\epsilon \approx \frac{a^*}{a}\hat{g}_{\frac{a^*}{a}\epsilon}e^{i\omega(\tau_* - \tau)}$$

where

$$\hat{g}_\epsilon = \frac{a_*^2|\hat{w}_*|^2}{a_*^2|\hat{w}_*|^2 + \epsilon^2}.$$

Since $w_* \in C_0^\infty$ (did I say that?), $\hat{w}_*(\omega) \to 0$ faster than any negative power of ω as $|\omega| \to \infty$ and is analytic, whence $\hat{g}_\epsilon : \epsilon > 0$ tends to 1 almost everywhere as $\epsilon \to 0$. Hence the inverse Fourier transform g_ϵ tends to δ in the sense of distributions, and

$$u_\epsilon(t) \approx \frac{a^*}{a} g_{\frac{a^*}{a}\epsilon}(t - (\tau_* - \tau))$$

to a multiple of a shifted δ, in fact exactly the distribution solution of $\bar{S}[m, w_*]u = d$. However for $\epsilon > 0$, u_ϵ is square integrable.

Analysis of the unweighted AWI objective (numerator of the formula above) with u replaced by u_ϵ goes back to Huang and Symes SEG 2015:

$$\int dt |t u_\epsilon|^2 \approx \frac{a_*^2}{a^2} \int dt\, t^2 |g_{\frac{a^*}{a}\epsilon}(t - (\tau_* - \tau))|^2$$

$$= \frac{a_*^2}{a^2} \int dt\, (t + (\tau_* - \tau))^2 |g_{\frac{a^*}{a}\epsilon}(t)|^2$$

The linear (in t) term vanishes, since g is necessarily even in time (since it Fourier transform is real). So this is

$$= \frac{a_*^2}{a^2} \int dt\, t^2 |g_{\frac{a^*}{a}\epsilon}(t)|^2 + (\tau_* - \tau)^2 \frac{a_*^2}{a^2} \|g_{\frac{a^*}{a}\epsilon}\|^2$$

The first term tends to zero as $\epsilon \to 0$, since g_ϵ is a delta-family, so you can make it negligible with proper choice of ϵ. The second term suggests that this function is related to the travel-time error $\tau_* - \tau$.

This is the key observation, since the travel-time error drives travel-time tomography and forcing it to zero forces m to approximate m_*, at least for well-behaved smooth models. In other words, AWI may be travel-time tomography in disguise! This would explaing why AWI works, since travel-time tomography (fitting of travel times, rather than wave data) works.

However the amplitude terms - the presence of a_*/a both as a multiplier and in the scaling of ϵ - may create stationary points for models differing in their travel time predictions. So stationary points of the unweighted AWI objective may not yield slowness models that match travel-times with the data, even in the noise-free case (even though the examples reported in Huang and Symes SEG 2015, and other works cited there, suggest that sometimes this is the case).

The scaling by trace norms of u_ϵ (the weight operator, in other words) eliminates the possibility of non-travel-time stationary points. Since the objective function is built up trace-by-trace, use the same approximation for the norm of u_ϵ:

$$\|u_\epsilon\| \approx \frac{a^*}{a} \|g_{\frac{a^*}{a}\epsilon}\|$$

So

$$J_0[m, w_*] \approx \sum_{\mathbf{x}_s, \mathbf{x}_r} (\tau[m_*](\mathbf{x}_s, \mathbf{x}_r) - \tau[m](\mathbf{x}_s, \mathbf{x}_r))^2$$

ignoring the $\|t g_{\ldots}\|^2$ term as being negligible for small enough ϵ.

In other words, the AWI objective is *exactly* an approximation to the travel-time tomography objective. And that's why it works - when geometric optics in the form given here is an accurate approximation to the Green's function.

It remains to see what geometric optics has to say about the penalty objective $\tilde{J}_\alpha[m, w_*]$, how geometric optics might fail, and what happens to AWI when it does.

[ ]:

[ ]:

[ ]: