

Deconvolution Tools for Seismic Signals

W. Stefan, E. Garnero, and R. A. Renaut

June 27, 2006

Abstract

This note provides complete details of the numerical algorithm implemented in the software package for signal restoration of seismic signals [5]. The complete software package is available electronically [6].

1 Introduction

In [5] we present a method of signal restoration to improve the signal to noise ratio, sharpen seismic arrival onset, and act as an empirical source deconvolution of specific seismic arrivals. Observed time series g_i are modeled as a convolution of a simpler time series f_i , and an invariant point spread function (PSF) h that attempts to account for the earthquake source process. The method is used on the shear wave time window containing SKS and S , whereby using a Gaussian PSF produces more impulsive, narrower, signals in the wave train. The resulting restored time series facilitates more accurate and objective relative travel time estimation of the individual seismic arrivals. We demonstrate the accuracy of the reconstruction method on synthetic seismograms generated by the reflectivity method. Clean and sharp reconstructions are obtained with real data, even for signals with relatively high noise content. Reconstructed signals are simpler, more impulsive, and narrower, which allows highlighting of some details of arrivals that are not readily apparent in raw waveforms. In particular, phases nearly coincident in time are separately identified after processing. This is demonstrated for two seismic wave pairs used to probe deep mantle and core-mantle boundary structure: (1) the S_{ab} and S_{cd} arrivals, which travel above and within, respectively, a 200-300 km thick higher than average shear wave velocity layer at the base of the mantle, observable in the 88 – 92 deg epicentral distance range; and (2) SKS and $SP_{diff}KS$, which are core waves with the latter having short arcs of P wave diffraction, and are nearly identical in timing near 108 – 110 deg in distance. Here we describe in detail the method for the Java/Matlab package which was developed for this signal

restoration. It is available for download, along with example data and synthetic seismograms.

2 Algorithm

We give a complete description of the steps necessary to compute the regularized deconvolution of a seismogram as described in [5].

2.0.1 L-BFGS method for minimization

The solution of the inverse problem to

$$g = f * h + n, \quad (1)$$

i.e. given the observed signal g and an estimate for the PSF h , we wish to find f is found by minimizing the convex objective function [8]

$$J(f) = \|g - f * h\|_2^2 + \lambda \text{TV}_\beta(f). \quad (2)$$

Here we consider an implementation based on Newton's method, in which we find the zero of the linear approximation to the gradient of the objective function. This corresponds to first replacing the objective function itself by a quadratic Taylor approximation and then finding its minimum. When the Hessian matrix $\nabla^2 J(f)$ is positive definite and Lipschitz continuous in the neighborhood of the minimizer f^* , Newton's method converges quadratically. The quadratic approximation is obtained by the Taylor expansion:

$$\begin{aligned} M_k(p) &:= J(f_k + p) \\ &\simeq J(f_k) + p^T \nabla J(f_k) + \frac{1}{2} p^T \nabla^2 J(f_k) p, \end{aligned} \quad (3)$$

where the Hessian matrix $M_k(p)$ is a function of the Newton search direction p at step k . Setting the derivative of $M_k(p)$ to zero and solving for p gives the formula for p at the k th step

$$p_k = -(\nabla^2 J(f_k))^{-1} \nabla J(f_k), \quad (4)$$

from which we obtain the Newton step

$$f_{k+1} = f_k + \alpha_k p_k. \quad (5)$$

The scalar α_k is the line search parameter which is used as a relaxation parameter. For a full Newton step $\alpha_k = 1$, but in practice $\alpha_k < 1$ can increase the convergence

radius of the method. In case of a non quadratic $J(f)$, as is the case here, α_k has to be chosen such that $J(f_k + \alpha_k p_k)$ satisfies the *sufficient descent condition* and the *curvature condition*, also known as the *Wolfe conditions* [2]. Usually this is done by performing a line search in which one finds the α_k that minimizes the scalar objective function

$$L(\alpha) = J(f_k + \alpha p_k), \quad (6)$$

under the constraints of the Wolfe conditions.

Computation of the Newton direction using (4) requires the inversion of the Hessian matrix which is, in many cases, impractical even if a closed form is available. Alternatively, in quasi Newton methods the Hessian at step k (i.e. $\nabla^2 J(f_k)$) is replaced by an approximation B_k and p_k is calculated to satisfy

$$p_k = -B_k^{-1} \nabla J(f_k). \quad (7)$$

Here, to save computation, we use the BFGS method [2] which uses a rank-two update for the update of the approximate Hessian

$$B_{k+1} = (I - \gamma_k y_k s_k^T) B_k (I - \gamma_k s_k y_k^T) + \gamma_k y_k y_k^T, \quad (8)$$

where

$$\gamma_k = \frac{1}{y_k^T s_k}, \quad (9)$$

$$s_k = f_{k+1} - f_k, \quad y_k = \nabla J(f_{k+1}) - \nabla J(f_k). \quad (10)$$

and $B_0 = I$ is assumed. Note s_k is proportional to the search direction via $s_k = \alpha_k p_k$.

The inverse $H_k := B_k^{-1}$ can be obtained by the Sherman-Morrison-Woodbury formula [2]:

$$H_{k+1} = (I - \gamma_k s_k y_k^T) H_k (I - \gamma_k y_k s_k^T) + \gamma_k s_k s_k^T. \quad (11)$$

While this expression can be used in (7), H_k may be very large. Specifically, for a signal of length m H_k is of size $m \times m$. Therefore, rather than maintaining the entire matrix we reduce memory consumption by using the L-BFGS method in which only a limited number of vectors s_k and y_k are maintained. Good convergence results can be achieved by keeping between $l = 5$ and 10 vectors. At iteration k , H_k can be computed using the vector pairs s_i, y_i and an initial Hessian approximation

$$H_k^0 = \rho_k I$$

where

$$\rho_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \quad (12)$$

An efficient method to compute $H_k \nabla J(f_k)$ needed for the search direction (7) is given by

Algorithm 1 *L-BFGS two-loop recursion* [2]

```

 $q := \nabla J(f_k)$ 
For  $i := k - 1, k - 2, \dots, k - l$ 
     $\eta_i := \gamma_i s_i^T q$ 
     $q := q - \eta_i y_i$ 
End For
 $\xi := H_k^0 q$ 
For  $i := k - l, k - l + 1, \dots, k - 1$ 
     $\nu := \gamma_i y_i^T \xi$ 
     $\xi := \xi + s_i(\eta_i - \nu)$ 
End For
Stop with result  $H_k \nabla J(f_k) == \xi$ 

```

2.0.2 Calculation of the Convolution

For two vectors $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, the convolution can be written as

$$c_k = (a * b)_k = \sum_{i \in \Omega(k)} a_i b_{k-i+1}, \quad (13)$$

where $\Omega(k)$ is the set of integers over which the sum is built. There are different ways to choose $\Omega(k)$. There is a *full* mode and a *valid* mode. For the *full* mode $\Omega(k)$ is the subset of all integers such that the indices of a and b are valid. The resulting vector is of length $n + m - 1$. Let $\Omega_a = \{1, 2, \dots, n\}$ and $\Omega_b = \{1, 2, \dots, m\}$, and assume, without loss of generality, $n > m$. With this notation we have

$$\Omega(k) = \{i \in \mathbb{N} | i \in \Omega_a \wedge k - i + 1 \in \Omega_b\}, \quad (14)$$

where \wedge denotes the logical *and* operator. This can also be written as

$$\Omega(k) = \{\max(1, k + 1 - m), \dots, \min(k, n)\}, \quad (15)$$

or

$$\begin{aligned}
\Omega(1) &= \{1\} \\
\Omega(2) &= \{1, 2\} \\
&\vdots \\
\Omega(m) &= \{1, \dots, m\} \\
\Omega(m+1) &= \{2, \dots, m+1\} \\
&\vdots \\
\Omega(n) &= \{n-m, \dots, n\} \\
\Omega(n+1) &= \{n-m+1, \dots, n\} \\
&\vdots \\
\Omega(n+m-2) &= \{n-1, n\} \\
\Omega(n+m-1) &= \{n\}.
\end{aligned} \tag{16}$$

Note that the middle part i.e. $\Omega(m), \dots, \Omega(n)$ consists of m elements which means that in this part all values of b are used. Outside of this region some of the values of b are omitted because they correspond to invalid indices of a , which is equivalent to padding the sequence a with zeros.

If only the middle part of the convolution is to be used, namely no padding is introduced, the convolution is called a *valid* convolution

$$\begin{aligned}
\Omega(1) &= \{1, \dots, m\} \\
\Omega(2) &= \{2, \dots, m+1\} \\
&\vdots \\
\Omega(n-m+1) &= \{n-m, \dots, n\}.
\end{aligned} \tag{17}$$

The convolution is computed by the following algorithm.

Algorithm 2 FREQUENCYDOMAIN CALCULATION OF CONVOLUTION:
Given two signals $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, calculate c given by (13). The valid convolution can be found as a subset of the vector which results from full convolution.

1. Pad both signals with zeros to appropriate length $n + m - 1$.
2. Transform both to frequency domain by using a fast Fourier transform (FFT).
3. Point wise multiply the signals with each other.
4. Transform the resulting signal back into the physical domain by an inverse FFT (iFFT).
5. If the FFT has not been performed using a real FFT, ignore imaginary part of the signal which occurs only due to round off errors.

2.0.3 The TV term

For the discretization of the total variation we replace the derivative in the TV regularization term

$$\text{TV}_\beta(f) = \int_{-\infty}^{\infty} \sqrt{f'(t)^2 + \beta} dt \quad (18)$$

by a first order finite difference approximation,

$$TV_\beta(f) = \sum_{i=2}^m \sqrt{(f_i - f_{i-1})^2 + \beta}, \quad (19)$$

which can also be written in terms of a differentiation matrix D acting on f

$$TV_\beta(f) = \sum_{i=1}^{m-1} (\Phi(Df))_i. \quad (20)$$

Here $\Phi(x)$ is the vector valued function

$$\Phi(x) = \begin{pmatrix} \sqrt{x_1^2 + \beta} \\ \vdots \\ \sqrt{x_n^2 + \beta} \end{pmatrix},$$

and D is the first order finite difference approximation of the derivative, [3]

$$D = \begin{pmatrix} -1 & 1 & 0 & \dots & \\ 0 & -1 & 1 & 0 & \dots \\ \vdots & \ddots & \ddots & \ddots & \\ & & & & \end{pmatrix}.$$

Note that the resulting matrix is not square but of size $n - 1 \times n$. In particular, this arises because D must have the appropriate null space such that derivatives of constant vectors are zero

$$D \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = 0.$$

2.0.4 Gradient of the objective function

The computation of the gradient of the objective function is both cheap and easy. The derivative of the k -th component of r which occurs in the fit-to-data term in (2), is

$$\begin{aligned}\frac{\partial r_k}{\partial f_j} &= - \sum_{l \in \Omega(k)} \frac{\partial}{\partial f_j} f_l h_{k-l+1} \\ &= - \sum_{l \in \Omega(k)} h_{k-l+1} \delta_{j-l} \\ &= -h_{k-j+1}.\end{aligned}$$

Thus

$$\begin{aligned}\frac{\partial}{\partial f_j} \|r\|_2^2 &= \sum_k \frac{\partial}{\partial f_j} (r_k)^2 = \sum_k 2r_k \frac{\partial r_k}{\partial f_j} \\ &= -2 \sum_k r_k h_{k-j+1},\end{aligned}$$

which requires the computation of the valid convolution $(r * \bar{h})$, where $\bar{h}_j = h_{n-j+1}$. In case of a valid convolution in the fit-to-data term, $(r * \bar{h})$ uses a full convolution. This makes the computation very cost effective because FFTs can be used to compute the gradient.

The gradient of the regularization term in (2) is computed as

$$\nabla TV_\beta(f) = D^T \Phi'(Df),$$

where the derivative of Φ is to be understood component wise. Hence

$$\nabla J(f) = -2r * \bar{h} + \lambda D^T \Phi'(Df). \quad (21)$$

Note that $*$ is either a full or valid convolution depending on the convolution in (2).

2.0.5 Algorithm overview

The following is an overview of the complete restoration algorithm

Algorithm 3 SIGNAL RESTORATION: *Given blurred signal g and a PSF h find a restored signal f to accuracy as determined by a given stopping criterion ϵ . Initialize with $f_0 = g * \delta$, where $*$ is a full convolution and δ is the delta function i.e. 1 in the center and 0 elsewhere. l is the number of vector pairs (s_k, y_k) to be saved and should be between 5 and 10.*

```

 $k := 0$ 
While  $\|\nabla J(f_k)\| > \epsilon$ 
Do
  Compute  $J(f_k)$  by (2) using Algorithm 2 and (20).
  Compute  $\nabla J(f_k)$  by (21).
  Compute search direction  $p_k$  by (7) using Algorithm 1.
  Compute step length  $\alpha_k$  by performing a line search along  $p_k$ 
    until Wolfe conditions are satisfied.
  Update  $f_{k+1} := f_k + \alpha_k p_k$ .
  If  $k > l$ 
    Discard the vector pair  $\{s_{k-l}, y_{k-l}\}$  from storage.
  End If
  Compute and save  $s_k := f_{k+1} - f_k$ 
    and  $y_k := \nabla J(f_{k+1}) - \nabla J(f_k)$ .
  Test for convergence. If converged Break else  $k := k + 1$ .
End Do

```

2.0.6 Edge detection

Lastly, to find the arrival times in the deconvolved signal we use the following edge detection scheme. At any point in the signal four successive points are interpolated by a cubic polynomial $P(t)$. The position of the zero of the second derivative t_0 (i.e. $P''(t_0) = 0$) is found. If t_0 lies between point 2 and 3 and the absolute value of the first derivative at t_0 is larger than some threshold (i.e. $|P'(t_0)| > P_{der}$) then t_0 is the position of a jump. This edge detection method provides sub pixel measurements of the jump positions.

The complete code (Java and Matlab) including the given examples is available [6].

3 The Software Package

An interactive Java application demonstrates the method and can be used to explore the impact of different parameter choices, and two Matlab examples show how this software can be used in a Matlab environment.

3.1 Download and install instructions

- download the whole package including the sample data (about 5Mb) and uncompress [6]. This software requires Java VM $\geq 1.4.2$.

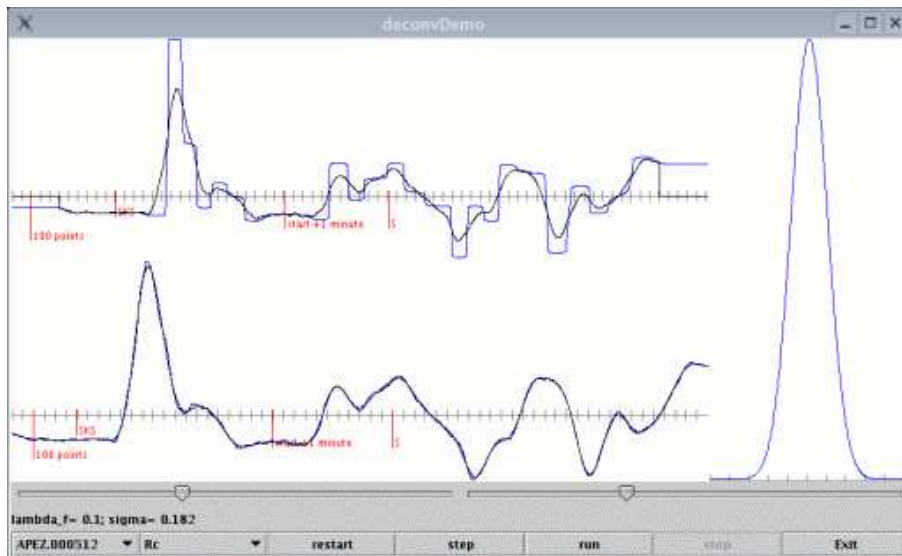


Figure 1: Screen shot of the Java demoapplication to illustrate the signal restoration.

The software is published under the GNU License. We ask everyone who uses our software to mention the reference [5]

3.2 Using the examples

In the main directory there is a Java program called `deconvDemo.java`. To run it

- set the `CLASSPATH` variable: e.g.
`export CLASSPATH=$CLASSPATH:.;jnt.jar;jfft.jar;riso.jar;deblurrseismo.jar`
(in Windows "set CLASSPATH=.;jnt.jar;jfft.jar;riso.jar;deblurrseismo.jar")
- run the program with: `java deconvDemo`

Figure 1 shows a screen shot of the Java demonstration. The top graph shows the original wave train and the deconvolved. The lower graph shows the original wave train and the forward projection of the deconvolved data i.e. the data fit. The labels are arrival time predictions from ray theory. Press the "run" button to start the reconstruction. The default settings should give a reconstruction similar to those in [1]. You can use the sliding controls to adjust the parameters while the algorithm is running. The left control changes the regularization parameter λ the right the PSF width σ .

- *lambda* controls the trade off between the smoothness of the reconstruction and the data fit. Very small values of *lambda* will have a very good data fit in the lower graph, but also high noise in the reconstruction i.e. large oscillations. Large *lambda* result in a smooth reconstruction. Very large *lambda* will result in over smoothing of the signal and the reconstruction loses details until it is almost constant.
- *sigma* governs the width of the PSF, thus the amount of deblurring. A small *sigma* will result in little deblurring, larger *sigma* in more deblurring, however if *sigma* is too large the forward projection loses the data fit, resulting in unwanted oscillations in the reconstruction.

Note you may have to press "run" again after a solution was found and the minimization is stopped.

3.3 Matlab

Two matlab programs, in the directory "matlab", demonstrate the usage of the code in Matlab. The file synth.m produces the graphs for the synthetic seismograms in [5], and reldata.m the graphs for the real data in [5].

Due to the Java integration in Matlab it is very easy to use the code in Matlab. Assuming there is a vector *g* (the blurred signal) and a vector *h* (the PSF) the following code deconvolves *g*:

```
javaaddpath('deblurrseismo.jar');
javaaddpath('jnt.jar');
javaaddpath('riso.jar');
import edu.asu.seismo.signals.oned.*;
f_hat=blurset.deconvf(doubleSignal1d(g),psf1d(h),
    'v',lambda,1,1e-6,0,1e-6);
plot(f_hat.vec());
```

4 Acknowledgment

This software uses parts of

- JNT (The FFT routine is by Dr. Bruce R. Miller the Stopwatch is taken from SciMark 2.0 by Roldan Pozo and Bruce Miller) [7].
- The RISO Project [4].

- A Java wrapper for FFTW [1] by Daniel Darabos. In order to use FFTW you also have to download the original wrapper and FFTW 1.2 from www.fftw.org and compile the FFTW library as well as the wrapper library (jfftwlib) and set an appropriate path (e.g. LD_LIBRARY_PATH). The deconvolution tools will also work without fftw using a the native java FFT implementation from JNT but it will be considerably slower.
- TauP.

This study was supported by the grant NSF CMG-02223. The Authors wish to thank Sebastian Rost and Matthew Fouch for discussions and data.

References

- [1] Frigo, M & Johnson, S. FFTW. <http://www.fftw.org/>
- [2] Nocedal, J. & Wright, S. J., 1999. Numerical Optimization, *Springer-Verlag New York*.
- [3] Hansen, P. C., 1994. Regularization Tools: A Matlab package for analysis and solution of discrete ill-posed problems, *Numer. Alg.* <http://www.imm.dtu.dk/~pch/Regutools/>
- [4] Dodier, R: The RISO Project. <http://riso.sourceforge.net/>
- [5] Stefan, W Garnero, E & Renaut R., 2005a. Signal restoration through deconvolution applied to deep mantle seismic probes. Preprint at <http://math.asu.edu/~rosie>
- [6] Stefan, W Garnero, E & Renaut R., 2005b. Deconvolution Tools for Seismic Signals <http://mathpost.la.asu.edu/~stefan/seismodeconv.html>
- [7] Pozo, R & Miller, B :SciMark 2.0 a Java benchmark for scientific and numerical computing. <http://math.nist.gov/scimark2/index.html>
- [8] Vogel, C. R., 2002. Computational Methods for Inverse Problems, *SIAM, Frontiers in Applied Mathematics*.