Notes for the Normal-Mode Codes

A C Birch

May 27, 2002

Contents

1 Introduction

The purpose of this document is provide an outline of some of my normal-mode based MATLAB codes for computations related to time-distance helioseismology. In particular this document describes code for computing cross-correlations (sec. 3) and travel-time kernels (sec. 4), as well a number of utility codes (sec. 5). The inputs and outputs for each function in this collection of codes are described in the individual MATLAB `.m` files, available at http://soi.stanford.edu/papers/dissertations/birch. In this document I have tried to provide the big picture: the way all of the codes fit together and the basic purpose of each function. Before describing the codes I will first describe the system of units and conventions that I use (sec. 2). The last section (sec. 6) describes the basic data structures that appear throughout this document.

2 Units and Normalization

Unless otherwise specifically noted, all variables are in cgs units. The exception to this rule is distances, which are measured in heliocentric great circle angle, in radians. Frequencies and damping rates are in angular frequency in s^{-1}. I don't use real units for source strength, as a result cross-correlations are not in any real units. This is fine for the computation of travel-time kernels, as the overall amplitude of the source function does not effect the kernel. The normalization of the eigenfunctions is

$$\int_{\odot} \rho(\boldsymbol{r}) \mathrm{d}\boldsymbol{r}\; \boldsymbol{\xi}^{nlm}(\boldsymbol{r}) \cdot \boldsymbol{\xi}^{nlm}(\boldsymbol{r}) = 1\,. \tag{1}$$

3 Cross-Correlations

This computation is described in some detail my dissertation (Birch, 2002). The starting point is a normal-mode expansion of the Green's function and a simple statistical model for the wave-sources. This set of codes models the cross-correlation as a function of time and distance for uncorrelated monopole sources at a particular depth. It would be very easy to generalize the codes to the correlated case.

3.1 `run_single_xcorr.m`

The main script for computing cross-correlations is `run_single_xcorr`. This script in turns calls the function `single_xcorr` which does the main calculation. There are a number of variables in `run_single_xcorr.m` that can be set by the user, they are

- `solar_model_name`: the name of the file containing the `solar_model` to be used

- `EXCLUDE_F`: set this to one to remove f modes from the calculation, if you want f modes set it to zero
- `OBS`: the radial grid point to observe at, this is an index into the radial grid in the `solar_model` structure
- `DEPTH`: the source location, this is an index into the radial grid in the `solar_model` structure
- `delta`: the distances, in radians, to compute at
- `T_MIN`: the smallest time lag to compute a cross-correlation at
- `T_MAX`: the largest time lag to compute a cross-correlation at
- `T_STEPS`: the number of point in the time grid

The result of running `run_single_xcorr` is the cross-correlation, `f`, which is a `length(delta)` $\times$ `T_STEPS` matrix.

3.2 `single_xcorr.m`

This is the code that does the actual computation. The sample code that I have distributed with these notes is currently set up to compute cross-correlations in the distributed-source model. By substituting the filter function for `amp` in `run_single_xcorr.m` instead of computing the mode amplitudes from the source depth and the eigenfunctions, the same code can be used in the single-source model.

4 Travel-Time Kernels

The full three-dimensional travel-time kernels connect travel-time perturbations to sound-speed perturbations by

$$\delta\tau(\mathbf{1},\mathbf{2}) = \int_{\odot} \mathrm{d}\boldsymbol{r}\; K(\mathbf{1},\mathbf{2};\boldsymbol{r}) \frac{\delta c^2(\boldsymbol{r})}{c^2(|\boldsymbol{r}|)}\,. \tag{2}$$

I define the one-dimensional travel-time kernels, which are relevant when the sound-speed perturbation depends only on radius, by

$$\delta\tau(\Delta) = \int_0^{r_{\mathrm{top}}} \mathrm{d}r\; K_1(\Delta, r) \frac{\delta c^2(r)}{c^2(r)} \tag{3}$$

where r is fractional radius and r_{top} is the upper limit of the fractional radius (somewhere above the photosphere, wherever the solar model runs out). A derivation of both of these kernels was done by Birch (2002).

This section describes the codes for computing three- and one-dimensional travel-time kernels. There is only one code that should be called from the

command line, `run_kernel`. This function takes as input the name of a parameter file. After reading the input file, `run_kernel` loads a solar model by calling `load_solar_model_pack`. The code then calls `calc_int_kernel` for one-dimensional kernels or `calc_kernel` for slices through three-dimensional kernels. Finally the results are saved to disk by `save_results`.

4.1 `run_kernel.m`

This is the only function for computing kernels that should be called from the command line. If `params.ONED=0` it will call `calc_kernel`, otherwise it will call `calc_int_kernel`, to do the computational work.

4.2 `calc_kernel.m`

This function does the actual calculation of three-dimensional travel-time kernels. It should be used by running `run_kernel`. `calc_kernel` begins by calling `select_modes` to choose which modes to use in the calculation, this decision is based on the filter (described in `params`.) Then `isolation_filter` is used to compute the time derivative of the zero-order signal. Then `calcg` computes the coupling matrix $\mathbf{G}$. The Legendre polynomials are computed by `calc_legendre`. The computation of horizontal distances is done based on the value of `params.CUT` and is a little complicated. Finally, the kernel is computed in the matrix formalism described by Birch & Kosovichev (2000).

Currently the code only supports cuts in the plane of the ray path and the plane perpendicular to the ray path at the lower turning. It shouldn't be too painful to come put with a more general scheme though.

4.3 `calc_int_kernel.m`

This function does the calculation of one-dimensional travel-time kernels. It should be used by running `run_kernel`. The operation of `calc_int_kernel` is essentially identical to `calc_kernel`. It is simpler, however, in that modes with different angular degree are not coupled. As a result the coupling matrix $\mathbf{G}$ is only computed for modes with the same l, which speeds up the calculation significantly.

4.4 `calcg.m`

This code does a numerical integration to estimate the coupling matrix

$$G_{ij} = \int_{t_{\min}}^{t_{\max}} \mathrm{d}t \; \dot{f}(t) \frac{\cos\omega_i t - \cos\omega_j t}{\omega_i^2 - \omega_j^2} \,. \tag{4}$$

The time derivative of the zero order signal is $\dot{f}(t)$. The limit as $\omega_i \to \omega_j$ is used when $i = j$. There is probably a much better way to compute the coupling matrix, which shows interesting patterns, suggesting that some sort of

asymptotics would be very useful. Currently up to half of the computational time in obtaining three-dimensional kernels is taken in this subroutine.

4.5 `isolation_filter`

This function computes the zero-order signal by normal-mode summation. The time derivatives of the signal are obtained exactly, which is nice as it saves having to take time derivatives numerically, which allows a cruder grid.

4.6 `select_modes`

Given a filter, described in a `params` data structure, this function returns the modes that have largest amplitude when put through the filter. The idea is to limit the modes in the computation to only those which are not essentially destroyed by the filter.

4.7 `INPUT`

This is a template input file for `run_calc_kernel`. It is just a nice form for saving and editing `params` data structures (sec. 6).

5 Utility Functions

The functions in this section are not specific to any particular calculation, but rather generic code that is useful in a number of places.

5.1 `load_solar_model.m`

This function takes the name of a file and reads the solar_model structure from that file. This code can also remove the f modes from the model. It assumes that the eigenfunctions are normalized so that they have a norm equal to the solar mass, and converts to the normalization of norm one.

5.2 `load_solar_model_pack.m`

This function is essentially the same as `load_solar_model` except that it returns the solar model in a structure instead of in separate variables.

5.3 `extract_eigenfunctions.m`

This function extracts the radial displacement function at the observation depth and the divergence of the displacement at the source depth. This is useful in the computation of cross-correlations in the distributed source model.

5.4 OTF.m

This function gives the OTF as a function of angular degree. The current bit of code uses an approximation to the square root of the MTF measured by Giles (1999) for the MDI full-disk mode. A correction is made so that the OTF is appropriate to medium-l MDI data. In general you can plug in any OTF that depends only on angular degree.

5.5 calc_legendre.m

The routine does a recursive calculation of the Legendre polynomials, this type of calculation is described in any number of references.

5.6 AsympLegendreP.m

This uses a standard asymptotic expansion of the Legendre polynomials, see any standard reference.

5.7 read_params.m

A little function to read a set of parameters from disk and put them into a `params` structure.

5.8 save_results.m

This saves a `kernel` structure to disk. It chooses a name for the file based on `kernel.NMODES`, `kernel.delta`, and `kernel.CUT`. In order to avoid writing over existing files it also adds a version number of one greater than any existing kernel files with the same `kernel.NMODES`, `kernel.delta`, and `kernel.CUT`.

6 Data Structures

The three complicated data structures that this set of codes uses are `solar_model`, `params`, and `kernel`. The first, `solar_model`, contains the information about a solar model and the normal modes in that model. The second, `params`, contains the parameters for the calculation of a travel-time kernels. Finally `kernel` holds the results of a calculation of a kernel. The structures `params` and `kernel` are only used by the codes that compute travel-time kernels, while `solar_model` is used by nearly every main code in this collection. The next three subsections describe the elements of these structures.

6.1 solar_model

The elements of `solar_model` are

- `n`: radial orders

- `l`: angular degrees
- `w`: angular frequencies
- `gamma`: damping rates
- `r`: radial grid (fractional radius)
- `U`: radial displacement eigenfunctions
- `V`: horizontal displacement eigenfunctions
- `c`: sound speed as a function of radius
- `rho`: density as a function of radius
- `p`: pressure as a function of radius
- `R`: solar radius
- `mass`: solar mass

The horizontal and vertical eigenfunctions are defined by the convention that the full displacement eigenfunction is given by

$$\boldsymbol{\xi}^{nlm}(r,\theta,\phi) = U(r)\mathbf{P}_{lm}(\theta,\phi) + V(r)\mathbf{B}_{lm}(\theta,\phi) \tag{5}$$

where the $\mathbf{P}$ and $\mathbf{B}$ are the real vector spherical harmonics defined by Dahlen & Tromp (1998).

6.2 `params`

The elements of `params` are

- `ONED`: which type of kernel to compute
  - `ONED`=1 : compute a one dimensional kernel
  - `ONED`=0 : compute a slice through the three-dimensional kernel
- `MODEL`: the file containing the `solar_model` structure to use in this calculation
- `NMODES`: the number of modes to use in the calculation
- `SKIP`: the factor to reduce the number of modes by, e.g. SKIP=1 does nothing, SKIP=2 only computes with every other mode.
- `FILTER_TYPE`:
  - `FILTER_TYPE`=0 : Gaussian group speed filter
  - `FILTER_TYPE`=1 : Gaussian phase speed filter
  - `FILTER_TYPE`=2 : angular degree cutoff filter

- FILTER_PARAM1:
  - if FILTER_TYPE=0 : FILTER_PARAM1 is the central group speed
  - if FILTER_TYPE=1 : FILTER_PARAM1 is the central phase speed
  - if FILTER_TYPE=2 : FILTER_PARAM1 is the minimum angular degree to use
- FILTER_PARAM2:
  - if FILTER_TYPE=0 : FILTER_PARAM2 is the group speed width
  - if FILTER_TYPE=1 : FILTER_PARAM2 is the phase speed width
  - if FILTER_TYPE=2 : FILTER_PARAM2 is the maximum angular degree to use
- FILTER_PARAM3: central frequency of the Gaussian frequency filter
- FILTER_PARAM4: the width of the Gaussian frequency filter
- EXCLUDE_F: set to zero to use the f mode in the calculation
- T_MIN: minimum time for the time window
- T_MAX: maximum time for the time window
- T_STEPS: number of steps in time for integrations
- delta: the distance at which the travel time is computed
- CUT:
  - CUT=0 : compute in the plane of the ray
  - CUT=1 : compute in the plane perpendicular to the ray at the lower turning point
- NPHI: number of mesh points in the angular direction
- NR: number of mesh points in the radial direction
- MINR: minimum fractional radius for the radial grid
- VERBOSE: set to zero to make the code quiet
- PLOTS: set to zero to suppress all plots that are made during the computation
- CALC_F_ONLY: set to one to compute the cross-correlation only, skipping the computation of the kernel

The most complicated part of this structure is probably the filter parameters. The parameter `FILTER_TYPE` controls which type of filtering is done *in addition* to the frequency filter. If you only want to do frequency filtering you can set `FILTER_TYPE=2` (angular degree cut-off filter) and then set `FILTER_PARAM1` and `FILTER_PARAM2` so that the cutoff in angular degree doesn't actually do anything. Another subtlety is that you have to choose the size of the time step in the numerical integrations involved in computing the coupling matrix (by choosing the parameter `T_STEPS`.) I have not done an exhaustive study of this issue, but you certainly want at least ten grid points per period or so.

6.3 `kernel`

The elements of `kernel` are

- `c`: sounds speed as a function of `r`
- `rho`: density as function of `r`
- `k`: the kernel
- `n`: radial orders of the modes used in computing the kernel
- `l`: angular degrees of the modes used in computing the kernel
- `w`: frequencies of the modes used in computing the kernel
- `r`: the grid in radius
- `phi`: the grid in angle
- `CUT`: same as from `params`
- `f`: the zero signal or cross-correlation (depending on whether we are working in the single-source or distributed-source model)

References

Birch, A. C. 2002, PhD thesis, Stanford University

Birch, A. C. & Kosovichev, A. G. 2000, Solar Phys., 192, 193

Dahlen, F. A. & Tromp, J. 1998, Theoretical Global Seismology (Princeton, NJ: Princeton University Press)

Giles, P. M. 1999, PhD thesis, Stanford University