1 Banana-doughnut kernel software package

Though ‘onset’ times of P and S waves are minimum travel time paths, cross-correlation travel times are often preferred because they are more accurate in the presence of noise, certainly if the onset is emergent. But cross-correlations involve integration over a time window. Scattered or multipathed energy arriving past the minimum arrival time can thus influence the measurement. Finite-frequency theory takes this into account. An additional advantage of finite-frequency theory is that it allows for stable amplitude inversions that model variations due to focusing or 3D structure in attenuation. Most importantly, one can extract more information out of a P or S wave by measuring its dispersion, cross-correlating over different frequency bands. Waveform inversion would provide a similar gain in resolution, but waveform inversions are much more nonlinear than delay time- or amplitude inversions.

The programs in this package will allow you to construct 3D finite-frequency kernels in complicated crustal models, write a sparse matrix file, and invert for a tomographic model. As an option, one can add onset times and have matrix3D compute the corresponding (ray-theoretical) matrix rows.

The structure of the package is as follows:

1. Use ray theory to calculate wavefields from source and receiver locations using grafbdyn
2. Calculate the matrix rows for one or more datasets using matrix3D
3. Combine all matrix rows and add correction parameters with assemblematrix3D
4. Invert for a model using mpisolvetomo3D

grafbdyn, matrix3D and assemblematrix3D have been put in the public domain, together with some simple plotting programs. The parallel (MPI) code mpisolvetomo3D which is still subject to frequent changes, as well as β-versions of programs to do cross-correlation, may be made available to Globalseis collaborators.

2 Quick install and test

If you have succesfully de-tarred the file:
```
tar -xvf BD3D.tar
```
you may first wish to edit compile and replace the GNU Fortran compile gfortran with your own Fortran compiler in case you do not have gfortran. Then run:
```
compile.
```
Also, if you wish to use the plotting software and scripts, you must have the GMT plotting programs installed on your machine. Though the computational tools may be of use to you even if you do not have GMT, the output is designed such that it can be plotted with GMT scripts. If you prefer Matlab for plotting, you’re on your own.
GMT can be downloaded for a number of platforms from: gmt.soest.hawaii.edu/gmt/gmt_download.html. Mac users can also use Fink to install GMT and any dependent software automatically (http://sourceforge.net/index.php). Windows users may have some trouble if they do not first download a free Unix emulation package Cygwin from http://sourceware.org/cygwin/ so they can install GMT just as if they run Linux or Unix.

To test if all works, I recommend you run the cook-book script: runexample1

This:

- Creates and plots a model (Fig 1).
- Creates and plots travel times (in sec) from a source (s1) and receiver (s2) node, as well as geometrical spreading (in km). This represents the ray-theoretical wavefield (Figs 2-5).
- Computes a 2D slice through the “banana-doughnut” sensitivity kernels and plots both the travel time and amplitude kernel (Fig 6-7).

The example scripts works fast because we limited the model to 3 nodes in the y-direction, thus creating a quasi 2D set-up. 3D application require much more time to compute the wavefield. A typical run on a smooth model of 100 × 100 × 50 nodes may take half an hour to complete, depending on the speed of your machine. If it takes much longer, it may be time to increase the memory, or to inspect if your model is really smooth.

3 How to use these programs

If you wish to adapt your own tomography software to use finite-frequency kernels rather than ray theory, the two tools you’ll need are the program grafbdyn and matrix3D. grafbdyn traces rays from one source

For those who wish to import kernels into existing inversion software, there is the subroutine bdsub, a more basic building block that uses the output from grafbdyn to compute the value of a finite-frequency sensitivity kernel at a point in the model for a given...
Figure 2: The travel time field from source s1, plotted with gmtt.

Figure 3: The travel time field from source s2, plotted with gmtt.

Figure 4: The geometrical spreading from source s1, plotted with gmtg.
to all nodes in the model (or from one receiver to compute the so-called adjoint field). matrix3D is a program that computes the sensitivity kernel and integrates it over the model voxels to produce the (sparse) rows of a matrix. For collaborators to the GLOBALSEIS project we also provide a \( \beta \)-version of mpisolveto3D to solve the linear system.

4 Running Grafbdyn.f

grafbdyn.f is a program that computes travel times and geometrical spreading values on the grid of a Cartesian box for a given source location. These can be used to compute finite-frequency kernels for delay time or amplitude sensitivity in local, 3D models, with bdsup or computebd. See Appendix A in case your source-receiver pair. A wrapper program computebd.f is provided to illustrate the use of the routine bdsup.
region is so large that the sphericity of the Earth plays a role.

The idea is to compute Green’s functions (wavefields for point sources) located in each hypocentre and in each receiver location. Because of reciprocity, the receiver Green functions from the receivers can be used to compute the wavefield from a scatterer anywhere in the model to that receiver. This scattered (or ‘adjoint’) field is needed to compute the sensitivity of the medium. We model these wavefields using ray theory in 3D media, which requires the models to be smooth. At present, the software cannot yet handle discontinuities such as the Moho, so these have to be represented by a gradient.

grafbdyn.f is a program to produce the travel times and geometrical spreading factors $R$ in a 3D cartesian box. It uses graph theory to produce starting rays, which are subsequently bent – see section 3.2 in Nolet (2008) We use the dynamic ray tracing routine para3d provided by Jean Virieux (Virieux, 1996) to compute the geometrical spreading $R$. Please cite these sources when using the results of this software in your publications.

The files created by this program can be used by program computebd to compute banana-doughnut kernels in the 3D box for those who like do-it-yourself programming.

An early version of the program was initially tested on a 1.4GHz Mac Powerbook using the g77 Fortran compiler. More recently it was tested for a very large model on a single node of a cluster with bi-pro quad core Opteron 2.7 Ghz processors, using the gfortran compiler. A smooth 3D model with $256 \times 256 \times 128 = 8,388,608$ nodes, using default settings, took 10 hours to compute rays from one source to all nodes, or about 4 ms per ray; with jssw8 set negative (see below) the CPU time was reduced to 4,100 seconds at the expense of errors of up to 10% in geometrical spreading, with time errors of about 0.5 ms per ray.

The program has undergone some limited testing against ground truth: on a homogeneous model as well as on a constant gradient model with $v=6.25*(1+0.1z)$; the latter gave errors up to 2% in geometrical spreading, the most error prone quantity, using default settings (all jssw flags 0). However, geometrical spreading is likely more in error for very strong gradients that replace discontinuities (which cannot be handled by this program). One is therefore advised to stick to smoothly varying reference models. If nonlinearity forces one to iterate, smoothing between iterations is good policy.

There is also a man page available (type \texttt{man grafbdyn}). To be able to use this with the Unix \texttt{man} command, move it to the area where your man pages are located. Ask your system administrator to do this.
for you. You’ll probably have to use **sudo** to create the necessary permission if you do this on your own laptop:

```
sudo mv grafbdyn.1 /sw/share/man/man1/
```

### 4.1 How to run grafbdyn

#### 4.1.1 Step 1: create the model

Before you start, create your slowness (1/velocity) model in a box that has at most \(NX \times NY \times NZ\) nodes, with a constant grid spacing of \(h\) km that is the same in all three directions.

If your area is large, you can gain precision by taking the sphericity of the Earth into account using a flattening transformation. See Appendix A.

Since the same format is used for model file as well as for the travel time- and geometrical spreading fields, lines 2 and 3 have numbers that are not used (you can set them to 0 or to store other information) The model file is in ascii format:

- rec 1: \(nx,ny,nz,x0,y0,z0,hh\) (box dimensions \(nx,ny,nz\), upper left corner \(x0,y0,z0\), cell size \(hh\))
- rec 2: \(s1,s2,s3\) (not used, but it is often convenient to give lon,lat and depth \(z\) at model origin)
- rec 3: \(xs,ys,zs\) (not used, you can set them to 0)
- rec 4ff: slowness if a model file, else time or geometrical spreading (\(x\) moves fastest, then \(y\), then \(z\))

An example of a model file is:

```
128 128 64 0.000000 0.000000 0.000000 2.000000
-73.5000 -32.5000 0.000000
.000000 0.000000 0.000000
0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667
0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667
0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667
0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667
...
0.118329 0.118429 0.118529 0.118629 0.118729 0.118700 0.118700 0.118700
0.118329 0.118429 0.118500 0.118600 0.118600 0.118600 0.118500 0.118329
0.118329 0.118350 0.118760 0.118670 0.118560 0.118380 0.118350 0.118310
```

#### 4.1.2 Step 2: prepare station and event coordinates

Compile two separate lists, one for receivers (stations) and one for sources. Give each of them an identification. The *idents* help you to organize the data structure. Number your events in some way, and have names for your stations. The runs of **grafbdyn** will result in (say) \(N\) station fields and \(M\) source fields, which would be able to generate \(M \times N\) banana-doughnut kernels in a 3D model. The source idsents do not have to be consecutive, you could for example use dates like 2006293 or, for more than one event per day, 2006293a, 2006293b etc. I advise against using dots (.) in the ident, since in **computebd.f** dots are already used to create file names for the kernels, such as Kt.2006293a.NE24 for the kernel from source 2006293a to station NE24. In fact, a very simple but effective way of identing is to name your sources s001, s002, s003,... and your receivers r001, r002, r003,...
4.1.3 Step 3: write a runscript

You have many different wavefields to compute but each runs separately, even if you run it on a cluster. This calls for the use of scripts. The workhorse that submits every source or receiver to a cluster is called subgraf. On the THERA cluster of Geoazur, which uses the OAR scheduler, the script is:

```csh
#!/bin/csh
# runs grafbdyn on thera
if($#argv < 4) then
    echo Usage rungraf ident x y z
    exit
endif
rm -f dum.$argv[1]
echo "#! /bin/bash" > dum.$argv[1]
echo "#OAR -n grafbdyn" >> dum.$argv[1]
echo "#OAR -O %jobid%.out" >> dum.$argv[1]
echo "#OAR -E %jobid%.out" >> dum.$argv[1]
echo "#OAR -l walltime=04:00:00" >> dum.$argv[1]
echo "#OAR -p ib='DDR'" >> dum.$argv[1]
echo "source /Softs/env_gnu.sh" >> dum.$argv[1]
chmod a+x ./dum.$argv[1]
oarsub -S ./dum.$argv[1]
```

One calls subgraf for each source and receiver, with their coordinates, e.g.:

```
subgraf r012 27.4 98.5 0.
subgraf s008 12.1 123.7 7.4
```

for one receiver at \(x = 27.4, y = 98.5\) and one earthquake at \(x = 12.1, y = 123.7\) and depth \(z = 7.4\).

This script only deals with the submission to a cluster, you would not need it if you run the program on your laptop. The running of grafbdyn with the needed input is done by the script named rungrafbdyn – and this is what you would call if you are working directly on your laptop:

```csh
#!/bin/csh
if($#argv < 5) then
    echo "Usage rungrafbdyn fmodel ident x y z"
    echo "This makes grafbdyn search for a file named fmodel with the model"
    echo "locate a source at x,y,z and tag an ident of max 40 characters"
    echo "to the resulting time and geometrical spreading files"
    exit
endif
/projet/globalseis/bin/grafbdyn << eof
1
0 0 0 -1 0 0 0 -20 0 0
$argv[1]
```

1
If you do not run this locally at Geoazur, you must replace the executable /projet/globalseis/bin/grafbdyn with its correct location on your own machine (we shall later discuss what the best input parameters are on the second line of input). It is most efficient to store the calls to subgraf also in a script. Here is an example that computes the (adjoint) wavefield from 8 receivers in a borehole at $x = 10, y = 30$

```
subgraf r501 10 30 10  
subgraf r502 10 30 15  
subgraf r503 10 30 20  
subgraf r504 10 30 25  
subgraf r505 10 30 30  
subgraf r506 10 30 35  
subgraf r507 10 30 40  
subgraf r508 10 30 45
```

### 4.1.4 Step 4: decide on the best running parameters

The input shown in the grafbdyn script consists of the following lines (the input is also prompted from screen if you prefer to run the program manually):

1. default switch indicator (0 if program should run with default parameters, 1 if you wish to change some)
2. 10 sense switches (in case default switch set to 1)
3. model file name
4. source coordinates $(x, y, z)$
5. output file ident (results in output files t.ident, g.ident)

Example of an input file to compute rays from source $s001$ in model CorsicaVp is:

```
0 (use default settings)
CorsicaVp
12.3 294.1 12.5 (depth 12.5 km)
s001
```

This input will create binary files $t.s001$ and $g.s001$ with travel-time and geometrical spreading fields, respectively. The default parameters used for grafbdyn usually give good results. You run with default settings if you give a 0 on the first input line (and omit the second line with ten sense switches). In some cases you may wish to speed up computations by allowing the program to be less careful. In other cases, the program emits a warning that tells you to set a sense switch for better results. The way to find out what is best for your model is to run the program for a number of sources and inspect the output.

Example of an input file to compute rays from source $s001$ in model CorsicaVp is:
The same computation, but with some defaults overridden, has as input:

1 (read sense switches next)
0 0 0 0 0 2 0 -20 0 0
CorsicaVp
12.3 294.1 12.5 (depth 12.5 km)
s001

Normally, the program does not output the rays themselves, as this would require an incredible amount of disk space. Instead, it specifies the travel time and geometrical spreading values at each of the model nodes\(^2\). A sense switch jssw3 is provided to output ray trajectories themselves, but this should be used with caution.

Output files:
- `t.ident`: travel times
- `g.ident`: geometrical spreading factors
- `out.ident`: diagnostic output file
- `ray_graph.xyz`: plottable ray file if jssw(3)>0
- `tasc.ident`: ascii travel time file if jdebug>0 in the main program
- `debugbd.ident`: useful for debugging if something went wrong

The "idents" help you to organize the data structure. The model file and the output files have the same structure. The output files are binary, single precision, but the model file is ascii. We treat only 3D models. As in the test example, 2D may be accommodated in a way by specifying ny=3, ys=-hh and ignoring y unequal to 0. The geometrical spreading computed in the y=0 plane is true 3D, but you cannot use the output to let bdsbf or matrix3D.f compute banana-doughnut kernels outside that plane, so it is advisable to use a larger model width in any case, even though that increases computing time.

The program does not care in what direction positive x, y or z points. It is good practice, though, to stick to:
- positive x is East
- positive y is North
- positive z is Down (lefthanded...)

units: km,s,s/km

The program is oblivious about units but km, s/km are recommended as standard practice, certainly if you wish to use some of the GMT plotting scripts that I (GN) have left scattered around. Of course, if you work with models of much smaller scale you can simply replace km with m and s with ms (or even mm and µs).

Topography can be accommodated by setting the slowness larger than 999 s/km. Travel times and geometrical spreading to these nodes will simply be set to 0. The high slowness deters rays from entering this space, hopefully. [DOES NOT WORK YET]

Only first arrivals are computed, a limitation fundamental to graph theory. This ray may be a headwave if that is the fastest. Rays are supposed to have straight segments in between te nodes at which they are specified.

\(^2\)Rays can be found by reconstructing the gradients to the time field or search for minimum time trajectory in the summed field of direct and adjoint times; though this is not needed for finite-frequency tomography, if you use the option in matrix3D to accept onset times it does reconstruct rays that way.
After exiting the graph theory, rays are regridded with a node spacing that is of the order of the node spacing in the model (this can be reduced by changing switch 6). Before the dynamic ray-tracing step, the node spacing is again reduced by a factor of ten (unless jssw8 is negative). By default, velocities are extrapolated outside the box if needed. This may be changed by setting jssw(2)=1, in which case all rays are confined to the box space, i.e. a headwave along a side surface will be selected if it is the first arriving one. Geometrical spreading is not correctly computed for headwaves, since routine para3d assumes it is a diving ray. A ray that exits the box becomes very likely a headwave along the side of the box.

4.1.5 The sense switches.

The working of the program can be influenced by changing the switches in jssw. Default operation is with all switches 0, while a switch usually activates a change when set to 1, unless noted otherwise below. The meaning of the switches is as follows:

1. 0=write time and geometrical spreading files, (t.ident, g.ident), 1= write maslov index file m.ident as well, 2=write t,g and p.ident file (graph node precedents), 4=write only t.ident and p.indent and skip dynamic ray tracing.
2. 1=force ray to stay in box (usually not needed)
3. save ray output in file ray_graph.xyz. If 1: only the rays to surface (z=0) nodes, if 2: all rays (use program selectray to get GMT plottable files from ray_graph.xyz)
4. 1=do a search in the gradient direction, rather than use a step size as predicted by the linearization (very robust but slow). Note that even the linear step size is limited to avoid ray excursions far away from the graph theoretical solution, so extra robustness is rarely needed. If jssw(4)=-1 you can keep it fully linear (this is faster but only safe for very smooth models).
5. if jssw5=2 suppress some screen output, 1 suppress out.ident, 3 suppresses both.
6. ray node spacing is refined by a factor equal to jssw6. The default ray node spacing for bending is equal to the model node spacing, the integration step size used to solve the dynamic ray tracing equations is ten times smaller (the latter refinement can be suppressed using jssw(8)<0 for faster – but less accurate – computations).
7. set convergence tolerance for bending to 1/10**jssw7 seconds (the default is 1/10**4)
8. set max of bending iterations to abs(jssw8)(default is 20). If jssw8<0 the ray node refinement for dynamic ray tracing is suppressed. This may speed up convergence considerably .
9. set forward star to xyz (e.g. jssw9=423 has length 4 in x, 2 in y, and 3 in z direction). Default is 444. Lengths > 9 not allowed.
10. taper of topography. Normally, a slowness ≥ 999 (velocity <0.001) indicates 'air’. To ease the burden on the bending routine we may allow the ray to venture into the air by adding two air nodes with 1/3 resp 1/9 the surface node velocity. [DO NOT USE, STILL PROBLEMATIC]

Depending on the complexity of the model, some rays may fail to converge sufficiently within the number of iterations allowed (20 or jssw8). If this is a small number of the total, it may not be serious since grafbdyn will nevertheless include the travel time in the output file t.ident and compute a (possibly very wrong) geometrical spreading. In the worst cases, the graph theoretical ray may be substituted.
4.1.6 Dimensions

Dimensions are set in the include file eno.inc:

- rays, during bending: \( L_{RAY}/10 \) (1600) nodes, unless refinement for dynamic ray tracing is suppressed (jssw8 < 0), in which case \( L_{RAY} \) (16000) nodes.
- rays, during dynamic tracing: 16000 nodes (> 10 \times \) bending size). This is only needed for \( x_r, y_r, z_r \) and their duplicates in regrid.

Maximum model dimensions: see (or edit) eno.inc. Currently 256 \times 256 \times 128.

The file out.ident should be inspected to see if anything irregular has occurred. Here is an example of an output:

Start run at: 35721.05

Model model3D
Dimensions 128 128 64
Origin 0.000000 0.000000 0.000000
Step size 2.000000
Limits 254.0000 254.0000 126.0000
Sense switches: 0 0 0 0 0 0 0 0 0 0
jsrefine= 1
Tolerance 1.000000E-03 sec
Max bending iterations 20
Ray node spacing 2.000000
Dynamic ray tracing step size 0.2000000
Source at 26.00000 19.00000 17.00000
Node nearest to source is 14 10 9
iptsource,vsource= 3 6.139866

\[
\begin{array}{cccccccccc}
\text{ii} & x & y & z & N_{ray} & \text{tgraf} & \text{tbend} & \text{tdynrt} & N_{par} & \text{dmax} & R & M & \text{iter} & \text{ier} \\
1 & 0.0 & 0.0 & 0.0 & 19 & 6.0697 & 6.0127 & 6.0134 & 183 & 0.0 & 35.5 & 0 & 3 & 0 \\
2 & 2.0 & 0.0 & 0.0 & 19 & 5.8372 & 5.7816 & 5.7823 & 176 & 0.0 & 34.2 & 0 & 3 & 0 \\
\end{array}
\]

where ii is the node, located at \( x, y, z \). During bending, the ray is specified at \( N_{ray} \) nodes. tgraf, tbend and tdynrt are the travel times computed during the graphing, bending and dynamic ray tracing step, respectively (tbend and tdynrt may differ slightly because the ray is gridded more finely). \( \text{dmax} \) is the largest depth reached by the ray to this node. \( N_{par} \) is the number of ray nodes used during the dynamic ray tracing. The Maslow index \( M \) should normally be 0 – if it is not, the bending may have converged upon a ray that is close but not the absolute minimum travel time (see however the section FAQ). \text{iter} is the number of bending iterations, and \text{ier} is an error indicator (consult the source code in the rare occasion that it is nonzero).

4.2 FAQ: Frequently asked questions

1. How do I find out where to put the man pages? Try: man -w pscoast.1 – this will tell you where the man pages for GMT are stored. You can also directly give the file path: man ./grafbdyn.1 which may be needed if you do not have superuser permission and your system manager is on vacation.

2. The sense switches offer so many options, how do I know what to use? The defaults usually work fine. Unless you see that something goes really wrong, do not use anything else (except perhaps setting jssw8 negative if your model is smooth enough).
3. How can I speed up computations? It may not be a good idea to reduce the size of the model, since then the sensitivity kernels you compute are truncated at the model edges. But the program uses a conservative step size in dynamic ray tracing to guard against inaccuracies introduced by steep gradients, and you may give up some accuracy for speed. Choose a source location such that the rays will graze the steepest gradient, and run grafbdyn twice, once with jssw8=0 and once with jssw8=-20. Then use slice3d and gmtgdz to make plots of geometrical spreading or compare the values of $R$ in the out.ident file; if they differ little, it is safe to run with jssw8<0 (e.g. -20).

4. I have a model with a Moho discontinuity, how do I specify that? Sorry, discontinuities are spread out over at least one grid distance, since the paraxial ray tracing cannot handle them.

5. My computations give quite a few nonzero Maslow indices! First arrivals should have Maslow (or ‘KMAH’) index zero, since they have not passed a caustic (focal line where $R = 0$). The subroutine para3d returns sometimes nonzero indices without any clear indication in the geometrical spreading field that $R \to 0$ (passage of a caustic); we suspect these are artefacts introduced by spline interpolating a steep gradient, and we do not worry too much. In particular, headwaves seem to confuse the Maslow index computation (dmax close to 0). The difference between the travel time after bending (tbend) and (tdynr) is also a good indicator if the linear interpolation used in bending, and the spline interpolation used in the dynamic ray tracing differ very much (dynamic ray tracing is forced to use splines because it needs second derivatives of velocity). Since the program assumed ray theory is valid to compute scattered energy, the background program should always be smooth enough that these differences do not matter (i.e. they remain well below the data uncertainty).

We trust that the bending converges to the absolute minimum travel time because the graph theoretical solution is the absolute minimum time for rays that are constrained to follow graph nodes – but this is not guaranteed. In matrix3D we check if the time reciprocity is satisfied, and reject data where opposite directions result in convergence to very different rays (default if travel times differ more than 0.5%). This provides a crude check against wrongly converged ray trajectories. In any case, matrix3D will set the index to 0, assuming you invert direct arrivals $P$ or $S$.

6. My model dimension exceeds 256? Edit the dimensions specified in file eno.inc, recompile grafbdyn, and have LOTS of patience...

7. Is there a parallel version of grafbdyn? No, but since the program is usually run for many sources and receivers, it can be efficient to use a cluster, with separate runs for each source or receiver on each node.

5 Bdsfub

The most complete and user friendly tool to compute a matrix of banana-doughnut sensitivity kernels in local models is matrix3D. But kernels can also be computed with subroutine BDSUB. Since each kernel is essentially the row of the matrix to be used in a tomographic inversion, you shall wish to use this routine to replace the ray tracing routine that constructs your tomographic matrix in your own software – or use computebd.f if you are just trying things out (see next).
6 Computebd.f

This is a wrapper program for bdsub.f that computes kernels from the ray field files constructed with grafbdyn.f.

7 Matrix3D.f

This program is rather easy to run. One calls matrix3D once for every dataset. Normally, datasets are arranged per source, but this is not a requirement. All data fed to one run must be of the same wavetype (P or S), but arrival times and amplitudes may be mixed (\(t^*\) will be accommodated in the next release). One may invert for velocity, attenuation, or both. The output is a matrix segment with as many rows as there are data in the file data3D.ident, where ident is the same source ident used by grafbdyn. Its output files t.ident and g.ident must be present in the same directory from where you run matrix3D; the model used by grafbdyn should also be in this directory.

matrix3D asks input from the screen, but an easy way to run it for many sources is to adapt the following script named runmatrix3D, which uses an arrangement of data per source:

```bash
#!/bin/csh
echo "This program is called by submat - do not call directly"
if($#argv < 3) then
    echo "Usage runmatrix3D datafile-ident minband maxband"
    exit
endif
/projet/globalseis/bin/matrix3D << eof
CorsicaVp
1 0 0 # Vp,Vs,Qs
66.0 150.0 # tolerance
0.9 # min corr coefficient
y
eof
```

This example script is hardwired to run on a model named CorsicaVp for kernels of P velocity. The third line of input is again an ident attached to the file name of the matrix segment. Since we arranged the data per source, we simply use the same source ident used earlier; you are free to choose a new ident here, however. Computations are fast and normally there is no need to divert this to a cluster, unless you are already doing everything on a cluster. If this cluster is there, the following script named submat will submit your job to the queue named ‘DDR’:

```bash
#!/bin/csh
# Move data3D.* files into the working directory
# before calling submat (or suballm)
if($#argv < 3) then
    echo "Usage submat ident minband maxband"
    exit
endif
rm -f dumm.$argv[1]
echo "#!/bin/bash" > dumm.$argv[1]
echo "#OAR -n matrix3D" >> dumm.$argv[1]
echo "#OAR -O %jobid%.out" >> dumm.$argv[1]
echo "#!/bin/bash" > dumm.$argv[1]
echo "#OAR -n matrix3D" >> dumm.$argv[1]
echo "#OAR -O %jobid%.out" >> dumm.$argv[1]
```

13
echo "#OAR -E %jobid%.err" >> dumm.$argv[1]
echo "#OAR -l walltime=04:00:00" >> dumm.$argv[1]
echo "#OAR -p ib='DDR'" >> dumm.$argv[1]
# echo "#OAR -p ib='none'" >> dumm.$argv[1]
echo "source /Softs/env_gnu.sh" >> dumm.$argv[1]
echo "./runmatrix3D $argv[*]" >> dumm.$argv[1]
chmod a+x ./dumm.$argv[1]
oarsub -S ./dumm.$argv[1]

7.1 Data files

If you run matrix3D with ident s023, the program looks for a data file named data3D.s023 that you have created earlier. The data file data3D. ident has the following structure:

1. background Q
2. frequency band information
3. source ident or stop
4. event number
5. receiver ident or source or stop
6. stationcode, networkcode, component code
7. (ff) datatype, datum-observed, datum-error, correlationcoefficient, frequencybandnumber, windowlength

The background Q – an average value for the quality factor of the whole model – is only used in case of attenuation tomography, but needs always be present even if only delay times are inverted for. The source and receiver ident(s) enable the program to find the output files t.ident and g.ident that were computed by grafbdyn. The event number is used to identify events for possible event corrections in the inversion. Similarly station codes are used to identify stations for station corrections.

The datatype is 1 for travel times, 2 for relative amplitude anomalies. Note that the times are the actual travel times, not the delays (the program computes the delays itself). If datatype=0, the program returns to line 5 to read a new receiver ident, or to go to line 3 if source is given as the receiver ident. It will halt halt if stop is given as a source or receiver ident.

The frequency band information has a format that depends on the type (digitized or analytical amplitude spectra). The first record gives nband, the number of frequency bands, but the rest of it can have one of two formats, depending on the sign of nband:

**Format 1** a format in which the filters are specified at nfreq angular frequencies $\omega_1, \omega_2, \ldots$ (usually normalized to max 1, but the normalization has no influence on the kernel computed). For each band:
line 1: nfreq, maximum correlation window length (in s) and average noise level in this band
line 2ff: $\omega$ (rad/s), $|\hat{m}(\omega)|$
where nfreq is the number of frequencies that specify the filter, and the max window length is the largest for that band in the data set. See eq (4.21) in Nolet (2008) for $\hat{m}(\omega)$. 

14
Format 2  the "visually" dominant period $p_{\text{max}}$ of a Gaussian filter as specified in Hung et al., GJI 141:175, 2000 (see also Hung et al., GJI 146:289-312, 2001, eqs 11-13). In this case the routine computes the filter values as if they were read as in case (1). For each band:

line 1: $p_{\text{max}}$ (in s), window length (in s) and average noise level in this band

Note than $\omega = 2\pi f$ if frequency $f$ is in Hz. The noise level is not used by the software but may be handy to remove the noisiest data if you suspect this can improve the imaging. If the max window length is specified as 0, a default length of 2.0 times the dominant period will be substituted. Mercerat and Nolet (2012) define optimal cross-correlation window boundaries $t_1$ and $t_2$ as follows:

$$t_1 = t_{\text{pred}} - \sigma - d_{\text{taper}}$$  \hspace{1cm} (1)
$$t_2 = t_{\text{pred}} + \sigma + f_c^{-1} + d_{\text{pulse}} + d_{\text{taper}}$$  \hspace{1cm} (2)

where $t_{\text{pred}}$ is the predicted arrival time with uncertainty $\sigma$, $d_{\text{pulse}}$ is the duration of the body wave pulse on the broadband record, $f_c$ is the central frequency of the passband filter and $d_{\text{taper}}$ is the duration of the windowing taper.

One chooses the second format by specifying nbands negative. Thus, if the first line of the data file is $+4$, the routine expects four bands, explicitly specified, whereas if nbands=-5, the five lines following define the dominant periods of each Gaussian filter band. The average noise level in each band is useful for selection purposes, but can be specified 0.

An example of a data file is data3D.s023 for a single source s023:

```
600 # background Q
-6
4.0 8.0 1.0 # period, largest windowlength, noise level
2.0 4.0 1.0
1.0 2.0 1.0
0.5 1.0 1.0
0.25 0.5 1.0
0.12 0.25 1.0
s023
023 # event number
r001
R001 XX BHZ
1 6.34 0.02 0.9 1 8.0 # type T, time, error, corr-coef, band nr, actual windowlength
1 6.34 0.01 0.9 2 4.0
1 6.323 0.005 0.9 3 2.0
1 6.306 0.005 0.9 4 1.0
1 6.271 0.002 0.9 5 0.5
1 6.243 0.006 0.9 6 0.25
0 0. 0. 0. 0. 0. # end of data for this station, go on to next
r002
R002 XX BHZ
1 2.76 0.02 0.9 1 8.0
1 2.77 0.01 0.9 2 4.0
1 2.777 0.005 0.9 3 2.0
1 2.783 0.005 0.9 4 1.0
1 2.779 0.002 0.9 5 0.5
1 2.794 0.006 0.9 6 0.25
0 0. 0. 0. 0. 0.
... (etcetera)
0 0. 0. 0. 0. 0.
r030
R030 XX BHZ
```

15
The output of `matrix3D` is:
1. `matrix3D.info.T.ident` and `matrix3D.info.A.ident`—used by `assemblematrix3D`,
2. `matrix3D.out.ident` readable file with diagnostics,
3. `matrixT.ident` and `matrixA.ident` with matrix segments for time and amplitude data, respectively,
4. (optionally) `kt00001.ident`, `kt00002.ident` etc. plottable kernel files.
5. `suspect_paths.ident` lists all paths with delay times larger than 10 standard deviations, to be able to spot errors in source or station coordinates etc.

### 7.2 Reading the output of `matrix3D`

The most relevant part of the out file lists all data, e.g.:

<table>
<thead>
<tr>
<th>Event</th>
<th>1 at</th>
<th>110.0</th>
<th>66.0</th>
<th>65.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Station</td>
<td>B</td>
<td>P</td>
<td>D</td>
<td>epidist</td>
</tr>
<tr>
<td>1 r510</td>
<td>Vp</td>
<td>T</td>
<td>100.00</td>
<td>760364</td>
</tr>
<tr>
<td>1 r510</td>
<td>Vp</td>
<td>T</td>
<td>100.00</td>
<td>755750</td>
</tr>
<tr>
<td>2 r511</td>
<td>Vp</td>
<td>T</td>
<td>100.00</td>
<td>751662</td>
</tr>
<tr>
<td>2 r511</td>
<td>Vp</td>
<td>T</td>
<td>100.00</td>
<td>478723</td>
</tr>
<tr>
<td>2 r511</td>
<td>Vp</td>
<td>T</td>
<td>100.00</td>
<td>294250</td>
</tr>
<tr>
<td>3 r512</td>
<td>Vp</td>
<td>T</td>
<td>100.00</td>
<td>754604</td>
</tr>
</tbody>
</table>

Following the event location, there are as many lines as data for this event. B is the frequency band, P the parameter type and D the data type (T or A), dist is the event-station distance, non0 is the number of nonzero elements in the matrix row, acc0 should be 100% if all nonzeroes sum to the ray-theoretical value of the kernel integral. However, we reject small matrix elements, and sparse indicated the percentage of nonzero elements, volume gives the accuracy of the sparse matrix row also in per cent: the fraction of the predicted delay time if the velocity in the whole model is increased by the same relative perturbation (ideally is 100%), delay is the observed delay with standard error sigma, R is the correlation coefficient; rcpT and rcpR test the accuracy of the ray tracing by comparing reciprocal values of travel time and geometrical spreading, respectively (ideally rcpT=rcpR=1).

If the accuracy is very different from 100%, this usually means the sensitivity kernel extends outside of the box. Alternatively, the cross-correlation time window may be too short to accommodate all the scattered energy from the anomaly. In the above example where the accuracy is 112.8%, if Vp in the box is uniformly increased such that one would expect a change in travel time for the first datum of 0.1s, the predicted change will be 0.1128 seconds (in this case an overestimate because the second Fresnel zone is outside of the box).
8 Assemblematrix3D.f

This program is able to put the matrix rows together that were computed in the previous step. It also adds columns to the matrix if you decide to add data-dependent corrections as unknowns. For example, you may wish to allow for the origin time or the location of an earthquake to be shifted slightly such as to avoid a bias in the model caused by an error in the event catalogue. For an extensive discussion of corrections, see Chapter 13 in Nolet (2008). How to weigh such correction parameters against model parameters is a decision that can be taken in the next step when one solves the system. assemblematrix3D creates a very large binary matrix file, and is normally run on the cluster where enough disk space is available.

8.1 Input file

The input file to assemblematrix3D.f has the following lines:

1. model file name
2. 10 lines with parameter sense switches (0/1) and prior uncertainties
3. (only if sense switch 3 > 0): ratio of dln Qp to dln Qs
4. station elevation correction (1) or not (0)
5. correct delays for Q dispersion to 1 s (1), to other period (2), or not at all (0)
6. (only if last card is 2): period for Q correction (s)
7. ident for output files from this run
8. next matrix file name (eg matrixT.P) or errlim,demean,weigh,group,stop
9. repeat last card until the file name is ”stop” or, in case of weigh first give:
10. weight for last group of data

The sense switches in line 2-11 represent parameters and corrections to invert for (if \( \neq 0 \)) or not (if 0):

1. dln Vp
2. dln Vs
3. dln Qs
4. Hypoctr corr (km)
5. Station corr \( t_P \) (s)
6. Station corr dln \( A_P \) (if 2: correct for each cluster separately)
7. Origin time corr dTo(s) (if 2: correct for each cluster separately)
8. Event moment corr dlnA’,
9. Station corr \( t_S \) (s) (if \(-1\): set S correction equal to \( \sqrt{3} \times P\)-correction)
10. Station corr dln\(A_S\)

The errlim option is to put a lower limit on the standard errors at were originally assigned to the data. The erradd option adds a constant to each standard deviation in the data file. These two options provides a crude but simple way to ‘correct’ for possible error sources discovered after the data file was created. The line after these optional commands should give the numerical value of the limit or added constant. These commands only operate on the matrix files that will follow them. One disables the option by repeating the command followed by a zero on the next line.

The demean option is for the last group of data (i.e. since the last demean). This allows for separate demeaning of different data groups. Data that need no demeaning should be input last. Be careful with weighting of data groups, as it moves the solution away from the maximum likelihood solution.

The script subasm to submit is:

```bash
#!/bin/csh -x
if($#argv < 1) then
    echo Usage subasm ident
    echo For example subasm b17
    exit
endif
if(-e in.asm$argv[1]) then
    echo input file used is in.asm$argv[1]
else
    echo ERROR EXIT in.asm$argv[1] not found
    exit
endif
rm -f duma.$argv[1]
echo "#!/bin/bash" > duma.$argv[1]
echo "#OAR -n assemblematrix3D" >> duma.$argv[1]
echo "#OAR -O %jobid%.out" >> duma.$argv[1]
echo "#OAR -E %jobid%.err" >> duma.$argv[1]
echo "#OAR -l walltime=04:00:00" >> duma.$argv[1]
echo "#OAR -p ib='DDR'" >> duma.$argv[1]
# echo "#OAR -p ib='none'" >> duma.$argv[1]
echo "source /Softs/env_gnu.sh" >> duma.$argv[1]
echo "/projet/globalseis/bin/assemblematrix3D < in.asm$argv[1]" >> duma.$argv[1]
chmod a+x ./duma.$argv[1]
oarsub -S ./duma.$argv[1]
```

This script needs an input file in.asm.xxxx, e.g.:

```
modelChile0
1 0.05
0 0
0 0
1 5
0 0
0 0
```
In this example, a matrix file `mat.allP` is created from the data sets used to create the `matrixT.*` files, and one uses the option to set the mean of the data to 0 and to set a lower limit for standard errors to 0.1.

9 mpisolvetomo3D

This is parallel software to accommodate the large matrix fully into memory. It thus needs to run on a cluster with a large enough memory to be able to store the full matrix. Much of the theoretical background can be found in chapters 14 and 15 of Nolet (2008).

It is at this stage that decisions on regularization (damping) must be made, and on the level of misfit that condemns a datum to be an outlier and is rejected. Normally, several trials are needed to get a good idea what works best.

9.1 Input

The input file for mpisolvetomo has the following lines:

1. name of the working directory (with the output of assemblematrix)
2. ident of the matrix file
3. ident of the aux file (normally the same, unless synthetic data for resolution test)
4. ident for this run to distinguish output from other runs
5. chi2target, epsnorm, epssmooth, epscorr - where epsnorm and epssmooth are norm- and smooth damping for model parameters, respectively, and epscorr varies the damping for corrections. Their use is dependent on the value of chi2target, the target $\chi^2_{\text{rel}}$ (relative, i.e. divided by number of data). If you specify chi2target > 0, the program looks for the correct damping parameters to obtain the requested data fit (generally somewhere near $\chi^2_{\text{rel}} = 1$), keeping the ratios between epsnorm, epssmooth and
epscorr constant. However, if you specify chi2target < 0, the program will do the inversion for the specified damping parameters and ignore the value of chi2target. Finally, setting chi2target=0 results in the computation of the “L-curve” of data misfit versus model norm.

6. outlier limit (e.g. 3.0 rejects data that, after a first inversion with very low damping, remain 3 standard error away from their predicted values).

7. itmax1, itmax2: number of iterations to perform by LSQR while seeking the right damping (itmax1) and when zooming in on the target \( \chi^2_{\text{rel}} \) (itmax2). Setting itmax1 a bit lower helps getting a good starting value for damping quickly. Depending on the size of the matrix, you may need several hundred iterations to converge to the proper model as specified by the damping. However, during early, exploratory runs, ne often sets the number of iterations to some low value (e.g. 50) that brings the model close enough to convergence to judge the effect of regularization.

8. colscale. Ignored if 0. If 0 < colscale < 1 the inversion tries to correct for different coverage, the more so as colscale approaches 1. However, be careful as too much column scaling may result in unstable solutions.

9. file name for the background model. This must be the same model as used throughout (grafbdyn, matrix3D, assemblemetrix3D).

```
/globalseis/nolet/BD3Dtest/TEST2pct/ALLBANDS
b17
b17
b17iter1
1.0 0.1 1.0 0.0  # chi2target, epsnorm, epssmooth, epscorr
3.0              # outlim
100 200          # itmax1, itmax2 (2 for zoom-in)
0.1              # colscale
model0D
```

On theran, one submits mpisolvetomo3D with following script:

```
#!/bin/csh
if($#argv<2) then
    echo Usage submpi ident nr_of_processors nr_of_nodes
    echo choose 8 processors per node for DDR
    echo eg submpi b17 64 8
    exit
endif
rm -f solv3D$argv[1]
echo "#!/bin/bash" > solv3D$argv[1]
echo "#OAR -n mpisolvetomo3D" >> solv3D$argv[1]
echo "#OAR -O %jobid%.out" >> solv3D$argv[1]
echo "#OAR -E %jobid%.out" >> solv3D$argv[1]
echo "#OAR -l nodes=$argv[3], walltime=12:30:0" >> solv3D$argv[1]
echo "#OAR -p ib='DDR'" >> solv3D$argv[1]
echo "source /Softs/env_gnu.sh" >> solv3D$argv[1]
echo "/Softs/openmpi-1.3.2-gnu/bin/mpirun -np $argv[2] /projet/globalseis/bin/mpisolvetomo3D
```
10 Other programs

The following programs are not essential but may be useful at times:

**mkslw3d** Writes a 3D slowness model from a 1D velocity-depth model, given in a file with depth/velocity (starting from the surface), or given a 2D model constructed with mkslw2d by widening the y dimension to ny pixels so it can be used as 3D input to grafbdyn and bdsub etc.

**bdsub** See **computebd**

**bdexample** This is a simple program to illustrate use of **bdsub**

**slice3d** Reads the travel time or geometrical spreading file or a (ascii) kernel file computed in 3D models and computes a vertical slice at any angle or a slice at constant depth. To get ascii kernel files set jdebug=1 in **computebd**. Some screen output is generated for travel time files and makes no sense when used for BD kernels, but the plot will be OK.

**computebd** Computes the banana-doughnut kernel on the model grid and writes this to a file that is structured just as the model file. Contains (and uses) subroutine **bdsub**

**getkernel3D** Reads one kernel out of a matrixT or matrixA file and writes it in standard format

**tg2asc** Translate binary travel time and geom spreading file to ascii, useful for debugging

**gmtt,gmta,gmtm** GMT scripts to plot travel time, geometrical spreading, or model slice from **slice3d.f**

**gmtka,gmtkt** plots kernels using GMT

References


11 Appendix A: dealing with the sphericity of the Earth

The Earth is spherical, with a radius $R$ km, and for large regions we cannot ignore the spherical shape. Though one degree in latitude at the surface of a spherical Earth has the same length everywhere, the length of one degree in longitude varies as $\cos \phi$ with latitude $\phi$ and will be different for the northern and southern part of your region. Both lengths shorten as one goes to depth $z$ and are proportional to $r/R = (R - z)/R$.

We can deal with the latitude ‘shrinking’ by rotating the center of our region to the equator, more precisely to (0E,0N), where the shortening of longitudes is minimal. This is described in section 3.7 of Nolet (2008). We deal with the depth shrinking using the Earth Flattening Transformation or EFT (e.g. Shearer, 2009, section 5.7):

$$z = -R \ln \left( \frac{r}{R} \right)$$

and

$$v_{\text{flat}}(z) = \left( \frac{R}{r} \right) v_{\text{spher}}(r)$$

The depth of the source should of course also be changed with the EFT. The EFT is exact for rays in a perfectly layered Earth where velocity depends on $z$ only. By extending the same transformation to 3D models we make a small error, which however is second order with respect to the velocity perturbations introduced by heterogeneity.

The effect of ellipticity on short regional raypaths is rather small, and variations in the ellipticity corrections of the times themselves can probably be ignored – check with Brian Kennett’s program ellip.f, available from www.rses.anu.edu.au/seismology/ttsoft.html. Instead, for the local radius $R$ one can adopt the radius of the standard ellipsoid to correct for the systematic effect of ellipticity on raypath length in the region:

$$R = a + \epsilon \left( \frac{1}{3} - \sin^2 \phi \right)$$

where $a = 6371$ km and $\epsilon = 1/298.3$. Latitudes of stations and events are listed as geographic latitudes in catalogues but should be converted to geocentric latitudes:

$$\tan \phi_{\text{geocentric}} = 0.9933056 \tan \phi_{\text{geographic}}$$

Subroutines geo2xy and xy2geo can perform the transition from geographical longitude/latitude to East/North (x/y) coordinates with a defined origin (best is the center of the region that contains most raypaths). Accuracy is defined by the errors in the computed lengths of raypaths along the surface when compared to true epicentral distances. You can test this accuracy with program transforms.f. For raypaths of a few hundred km in size the largest errors are in a diagonal (NE, NW) direction but remain below 0.1% (do not run these programs in single precision).