ASKI

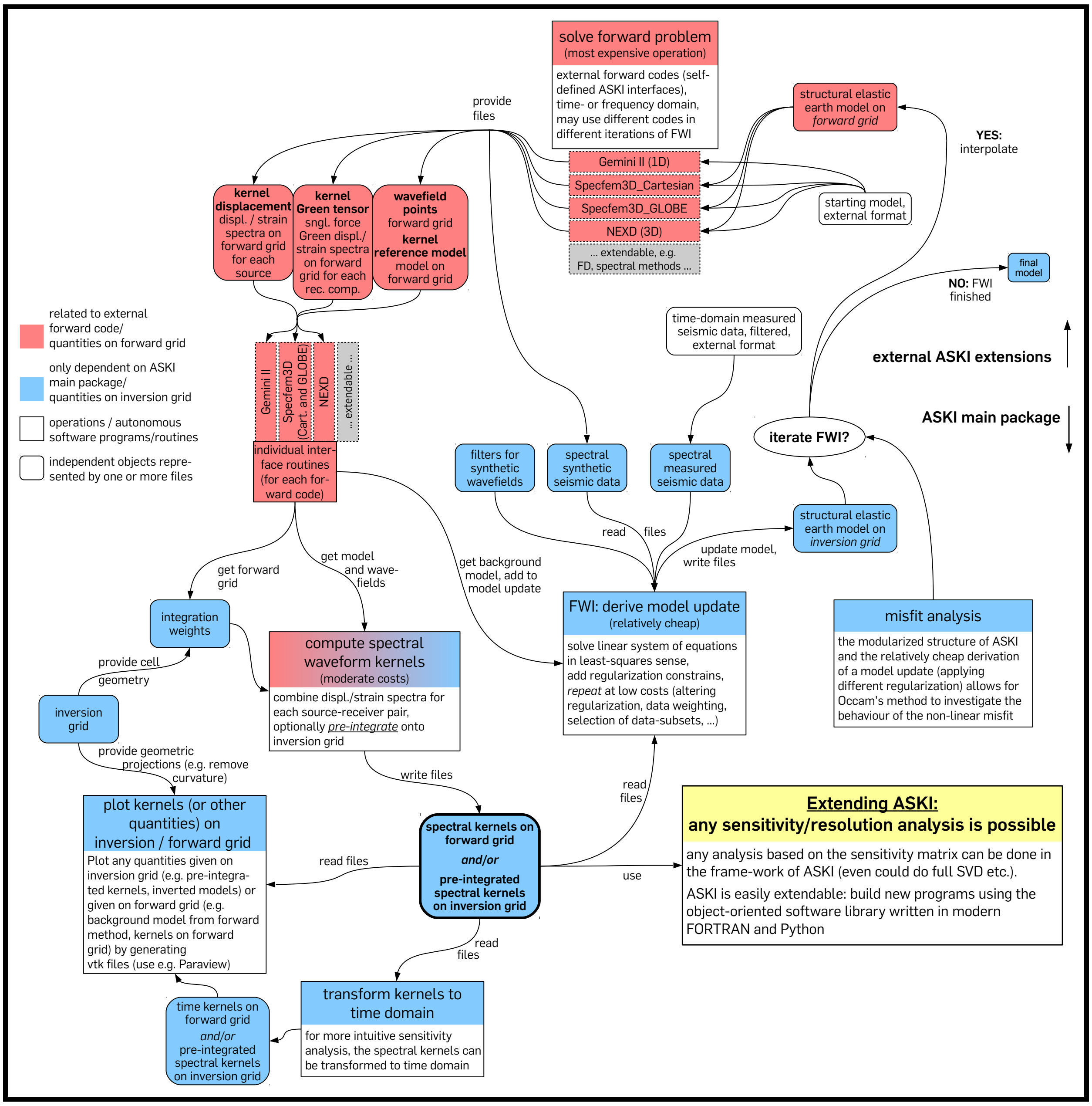
Analysis of
Sensitivity and
Kernel
Inversion

Developer's Manual

ASKI – version 1.2

Dec 2016

Florian Schumacher
Ruhr-Universität Bochum, Germany


solve forward problem
(most expensive operation)
external forward codes (self-defined ASKI interfaces), time- or frequency domain, may use different codes in different iterations of FWI
provide files
structural elastic earth model on *forward grid*
YES: interpolate
Gemini II (1D)
Specfem3D_Cartesian
Specfem3D_GLOBE
NEXD (3D)
... extendable, e.g. FD, spectral methods ...
starting model, external format
kernel displacement
displ. / strain spectra on forward grid for each source
kernel Green tensor
sngl. force Green displ./ strain spectra on forward grid for each rec. comp.
wavefield points
forward grid
kernel reference model
model on forward grid
final model
NO: FWI finished
related to external forward code/ quantities on forward grid
only dependent on ASKI main package/ quantities on inversion grid
operations / autonomous software programs/routines
independent objects represented by one or more files
Gemini II
Specfem3D (Cart. and GLOBE)
NEXD
... extendable ...
time-domain measured seismic data, filtered, external format
external ASKI extensions
ASKI main package
iterate FWI?
individual interface routines (for each forward code)
filters for synthetic wavefields
spectral synthetic seismic data
spectral measured seismic data
structural elastic earth model on *inversion grid*
read files
update model, write files
get forward grid
get model and wavefields
get background model, add to model update
integration weights
compute spectral waveform kernels
(moderate costs)
combine displ./strain spectra for each source-receiver pair, optionally *pre-integrate* onto inversion grid
FWI: derive model update
(relatively cheap)
solve linear system of equations in least-squares sense, add regularization constrains, *repeat* at low costs (altering regularization, data weighting, selection of data-subsets, ...)
misfit analysis
the modularized structure of ASKI and the relatively cheap derivation of a model update (applying different regularization) allows for Occam's method to investigate the behaviour of the non-linear misfit
provide cell geometry
inversion grid
provide geometric projections (e.g. remove curvature)
plot kernels (or other quantities) on inversion / forward grid
Plot any quantities given on inversion grid (e.g. pre-integrated kernels, inverted models) or given on forward grid (e.g. background model from forward method, kernels on forward grid) by generating vtk files (use e.g. Paraview)
write files
read files
spectral kernels on forward grid
and/or
pre-integrated spectral kernels on inversion grid
read files
use
Extending ASKI:
any sensitivity/resolution analysis is possible
any analysis based on the sensitivity matrix can be done in the frame-work of ASKI (even could do full SVD etc.).
ASKI is easily extendable: build new programs using the object-oriented software library written in modern FORTRAN and Python
read files
transform kernels to time domain
for more intuitive sensitivity analysis, the spectral kernels can be transformed to time domain
time kernels on forward grid
and/or
pre-integrated spectral kernels on inversion grid

If you use `ASKI` for your own research, please cite one of our papers [SFL16], or [SF16], as appropriate:

F. Schumacher, W. Friederich and S. Lamara,
"A flexible, extendable, modular and computationally efficient approach to scattering-integral-based seismic full waveform inversion",
Geophysical Journal International, (February, 2016) 204 (2): 1100-1119
`http://dx.doi.org/10.1093/gji/ggv505`

Schumacher F, Friederich W.
"ASKI: A modular toolbox for scattering-integral-based seismic full waveform inversion and sensitivity analysis utilizing external forward codes".
SoftwareX 5 (2016) 252–259,
`http://dx.doi.org/10.1016/j.softx.2016.10.005`

This documentation was written in the hope that it will be useful to the user, but it *cannot be assured* that it is accurate in every respect or complete in any sense. In fact, at some places *this manual is work in progress*.
Please do not hesitate to report any inconsistencies by opening (or adding to) an "issues" topic on `https://github.com/seismology-RUB/ASKI` or to improve this documentation by incorporating your experiences with `ASKI` and your personal experience of getting used to it (at best by modifying the source and issuing a pull request on gitHub, in any case let us know about it! Thanks). When you have developed new `ASKI` components or have modified existing once, please extend / modify the `ASKI` documentation accoringly.

Furthermore, I am aware of the poor LaTeXcoding of this document (at the moment, `\sloppy` is used at the beginning of the document to avoid overfull hboxes in many places). There is a lot of potential to improve the document style, hence the readability of the manual as a whole, as well as the coding style of the particular `.tex` files. *Please do not hesitate to improve!*

The LaTeXsource files and all related components of this document are available via `https://github.com/seismology-RUB/ASKI` , subdirectory `devel/doc/ASKI_developers_manual/` of the repository.

Florian Schumacher, Dec 2016

Contents

1 Introduction and guidelines for `ASKI` developers **4**

2 `ASKI` versions **5**

3 `ASKI` tools for developers **7**

3.1 `adapt_GPL_headers.py` 7

3.1.1 Options 7

3.1.2 Positional arguments 8

3.2 `create_Makefile_rule.py` 8

3.3 `create_rules_mk.py` 8

4 Add support for another external forward code **8**

4.1 What functionality / information must the forward code provide / produce for use with `ASKI` ? 8

4.1.1 forward grid 9

4.1.2 elastic model on wavefield points 9

4.1.3 spectral displacement wavefield propagating from seismic sources into medium as well as its strains 9

4.1.4 spectral synthetic data in format required by `ASKI` 10

4.1.5 spectral Green's functions originating at receiver positions as well as its strains 10

4.1.6 some script running all simulations in a FWI iteration 10

4.2 Make `ASKI` understand the forward code: `ASKI` interface sub-modules 11

4.3 Import updated model to the forward code for next iteration of FWI 12

4.4 As an option: use forward-code-specific quadrature rules for kernel pre-integration 13

4.5 Time-domain forward methods: Choosing a method of on-the-fly Fourier transform 13

5 Create new executables of the `ASKI` toolbox based on its software library **16**

6 Inversion grids **16**

6.1 Add another type of inversion grid 16

6.2 Using element grid of element-based forward method as inversion grid 17

6.3 `chunksInversionGrid` : implement a method of cell refinement 18

7 Things "to do" for the `ASKI` developers **19**

7.1 Todos that are straightforward to realize 19

7.1.1 use function `nextPathDataModelSpaceInfo` everywhere 19

7.1.2 finish implementing module `parameterCorrelation` 19

7.2 Todos that might require some more work 19

7.2.1 completion of module `ecartInversionGrid` 19

7.2.2 improvement of the `ASKI` user manual, section entitled “What is `ASKI` ?” 20

7.3 Ideas for new `ASKI` functionality 20

7.3.1 file-interface to module `dataModelSpaceInfo` 20

7.3.2 forward-method interface modules for synthetic data 20

7.3.3 support integer list of formats like “3-5,9,13,19-21” 21

7.3.4 other methods of smoothing kernel inverted models 21

References **22**

History **23**

GNU Free Documentation License **24**

1 Introduction and guidelines for `ASKI` developers

The development of (what became) the software package `ASKI` started in 2010 and from the beginning, auxilliary Fortran modules were utilized (such as e.g. `errorMessage` and many more) which were developed by Wolfgang Friederich (in many cases a lot earlier than 2010) and applied in the seismology group of Ruhr-University Bochum (RUB) in a lot of different software projects (e.g. also in Gemini) via a separate "include" repository. As a result, `ASKI` modules originating from this "auxilliary software library" most likely provide more functionality than `ASKI` requires. After moving the `ASKI` developers repository to `gitHub.com` in 2016, the `ASKI` modules from "include" were permanently forked from the RUB-internal code repository and may now be optimized for used with `ASKI` (not intended to merge them back into the original "include" repository at RUB, except for bug-fixes and fundamental enhancement of general functionality). So, if you wonder why you come accross subroutines that are not used in `ASKI` (so far!!), this might be a reason why.

Guidlines w.r.t. the `git` repository

At the moment, there are two branches on `gitHub.com` : `master` and `develop`.

The `master` branch is intended to contain the latest stable version of the software package (as also communicated in publications as well as the `ASKI` user manual). *No regular commits* should be performed to the `master` branch, changes should only be merged from the `develop` branch!

The `develop` branch is intended for development of new functionality (as its name says):

- The current state of the `develop` branch may be unstable or incomplete (e.g. in terms of documentation etc.).
- *Pull requests* should be issued to the `develop` branch only (and *not* to the `master` branch)!
- When a new portion of functionality is fully implemented and documented (thus, a "stable" state is reached), you may immediately merge the `develop` branch into the `master` branch. You don't need to wait until there is a new official "version" or something, you sould provide the latest stable version as quickly as possible to the user community.
- On the `develop` branch, *do not* commit `pdfs` of the LaTeX documents (like the manuals, etc.) each time you modify the documents! *Only* before merging into the `master` branch commit a re-compiled `pdf` for each document (so that in the `master` branch the LaTeX source corresponds to the repsective `pdf`).
- Locally, you may use yet more branches for development (like "feature" branches etc.), but for now the sparsely manned development team does not consider it necessary to have these kinds of branches also available on `gitHub.com`.

How is the code documented in general?

The code files are commented extensively. Furthermore, almost all Fortran modules, derived types, subroutines and functions are documented using doxygen syntax, even though doxygen output is not really optimal for Fortran code. doxygen documentation can be produced using the doxygen config file `doc/doxygen.conf`: After installing doxygen (and possibly modifying file `doxygen.conf` according to your needs), you should be able to produce doxygen html and latex documents executing `doxygen doxygen.conf` from path `doc/`, which should create subfolder `doc/doxygen_output` containing directories `html` and `latex`. For the html documentation, open `doc/doxygen_output/html/index.html` by a web-browser of your choice. For producing a latex pdf document, execute `make all` from `doc/doxygen_output/latex`, which should document `doc/doxygen_output/latex/refman.pdf`

Please preserve your experience!

If you struggled with the existing ASKI documentation (user manual, comments in code, doxygen, developers manual) because it was inconsistent, incomplete or simply wrong and you invested time to find out how it works, *please let future generations of users and developers benefit from your gained knowledge!* Everybody knows that documenting code and writing manuals consumes a lot of time, but correct documentation is essential for everyone using and developing software, and I'm sure you know that from your own experience. So, please invest a bit more time in correcting/extending the ASKI documentation (where applicable: user manual, comments in code, doxygen, developers manual), otherwise your knowledge is lost forever (you might even lose your knowledge yourself after some while, so please write it down!).

Thank you! (on behalf of everybody)

2 ASKI versions

ASKI 's release version numbering does not follow any standard of version numbering. Since releases were not very frequent so far and the source code was not publically available under version control, it was considered sufficient to have only a simple numbering for the purpose of distinguishing release versions. In case that developments and releases become more frequent in the future, the ASKI developers might consider to follow some standard for future release version numbering.

ASKI 0.3

ASKI 's first release version was numbered `0.3`, with `0` indicating a pre-release that was not very well tested at that point and `3` being the third version of internal development when porting from another internal versioning repository.

ASKI 1.0

After some while of intensive testing and application of ASKI to synthetic and real-world cases, as well as development of a lot of ASKI tools, version `1.0` was released as the first ready-to-use version of ASKI .

ASKI 1.1

In general, version 1.1 should be compatible with 1.0 in terms of file formats and general use of the software. The main reason for this release was the fix of a pointer problem with gfortran. Compared with the previous release, some more tools are available (addSpikeCheckerToKim, createSpectralFilters, create_ASKI_evstat_filters.py) and forward-code-specific definition of complex frequencies is available (e.g. for use with Gemini). Some bugs were fixed.

ASKI 1.2

Version 1.2 is just an intermediate version number and denotes the status of the code when moving the development repository permanently to git and providing the current working ready-to-use version of ASKI by the master branch of repository https://github.com/seismology-RUB/ASKI.

Significant changes to version 1.1:

- There is an additional subdirectory devel/ containing developer tools and developer documentation.
- The source files of the ASKI user manual as well as a compiled pdf of the manual are provided now in subdirectory doc/
- Inversion grids of type chunksInversionGrid now provide base cell refinement capabilities (at the moment a random "toy" method is implmemented for illustration, but serious refinement method can now be implemented easily into module chunksInversionGrid.
- New/renamed/removed tools:

  chunksInvgrid2vtk for special vtk files related to chunks and refined cells

  createShoreLines (Fortran executable) and create_shore_lines.py (Python program utilizing the f2py interface generator) for generating shore line vtk files from native binary GSHHS shore line data files

  Executable createMeasuredData was renamed in transformMeasuredData

  Python script create_ASKI_evstat_filters.py is removed (functionality not required anymore)

The gitHub master branch

After porting ASKI to https://github.com/seismology-RUB/ASKI, the current version of its master branch should serve as a stable version to use, along with the current versions of the forward code packages supported by ASKI .

3 `ASKI` tools for developers

3.1 `adapt_GPL_headers.py`

Script `devel/adapt_GPL_headers.py` adapts GPL (and GFDL) headers of given files. Executing `adapt_GPL_headers.py -h` will print a usage message.

For the `ASKI` source files given as the positional arguments list (at least 1 must be given), the license headers are adapted by year and/or `ASKI` version, as specified by options `-y, --year` and `-ver, --version`, respectively (at least one such option must be given, as otherwise there is nothing to do).

The program detects file headers of specific form, having the following properties:

max length: The headers are expected within the first `NLINE` lines of a file (default: 20)

start line: The headers start at the first line containing the following characters (possibly preceded by some commetary characters): 76 `-` characters.

end line: The headers end by the first line after start line containing the following characters (possibly preceded by some commetary characters): 76 `-` characters.

text segment: there must be at least one line (between start and end line) that contains one of the following text samples:

```
under the terms of the GNU General Public License
under the terms of the GNU Free Documentation License
```

assumption: first line after start line does *not* contain “ASKI version “ (if so, it is ignored) but contains “Copyright ” followed by the year (year = the word following the *first* occurence of “Copyright ” in that line). The year will be replaced by a new value (if `-y, --year` is indicated). If in the original header, the year ends a sentence, i.e. ends on a period “.”, this period will be preserved. *Nowhere else in the header will the year be searched for and adapted!*

3.1.1 Options

`-h, --help` print usage message

`-y YEAR, --year YEAR` If set, then the copyright year (i.e. the word following the first occurence of “Copyright “ in the first line of the header text body) will be set to the given value `YEAR` (character string, not checked if it is a number).

`-ver VERSION, --version VERSION` If set, then the `ASKI` version will be set to the given value `VERSION` (character string, not checked if it contains numbers). Give the version here by numbers only, e.g. `1.3` for `ASKI` version 1.3

`-n NLINES, --nlines NLINES` Optional: if set, then the header is expected to be contained within the first `NLINE` lines of each indicated file (if not set, default value of 20 is used).

3.1.2 Positional arguments

`ASKI_src_files ...` List of filenames (relative or absolute) indicating all files for which the header should be adapted. You may as well use regular expressions on the command line such as
`f90/* py/* doc/ASKI_manual/*.tex devel/* devel/*/*`
The program generates warnings if particular files cannot be adapted (because there is no header found as expected) and continues to process the rest of the files.

3.2 `create_Makefile_rule.py`

Calling
`devel/create_Makefile_rule.py newTool`
from the `ASKI` installation directory will print a complete Makefile rule for target `newTool` to screen (output should be re-directed into a file by something like `> output.txt`). Any required libraries (`BLAS`, `LAPACK` etc.) should be accounted for correctly. Otherwise you should update script `create_makefile_rule.py`, in particular routine `required_libs_for_depobs`: please add any library requirements for new/modified modules that you have developed.

3.3 `create_rules_mk.py`

Script `create_rules_mk.py` (contained in `devel/`) can be utilized to re-create file `rules.mk`. From the `ASKI` installation directory, call
`devel/create_rules_mk.py > rules.mk`
(you might want to backup the existing `rules.mk` before you overwrite it, just in case).

4 Add support for another external forward code

`ASKI` does not implement an intrinsic code for simulation of seismic wave propagation in order to solve the forward problem, but instead provides generalized interfaces to different external wave propagation codes. At the moment, the 1D semi-analytical code Gemini [FD95] and the 3D spectral element code SPECFEM3D [TKL08] are supported in *both*, Cartesian and spherical framework. Additionally `ASKI` supports the 3D nodal discontinuous Galerkin code NEXD [Lam15] and we plan to implement support for the Finite Difference code SOFI [Boh02] in the future.

Here is a conceptual recipe describing the steps that need to be done to add another forward code to the ASKI software package:

4.1 What functionality / information must the forward code provide / produce for use with `ASKI` ?

The forward code must be able to provide information/definitions of the quantities listed in the following. Therefore, the forward code most likely need to be extended / modified for use with `ASKI` . However, the forward code is absolutely free in the way it provides this information, i.e. any standard (parameter) files can be used to provide information (the code is not to be modified in this case) or any files of some own file format can be produced (even parallel output can used, provided the knowledge of how to read it/locate the files), or any assumptions can be made on the quantities (assumptions on filenames, (sub)grids, model structure (e.g. 1D), etc.). Information on the following quantities just needs to exist somehow (either by standard functionality of the program, by assumptions or by files of arbitrary (new) format. How this information is communicated to `ASKI` is explained below in sec. 4.2.

4.1.1 forward grid

The forward code must provide point coordinates of the grid on which it produces wavefield output (see below displacements / Green functions). For grid-based forward codes, it is efficient (thus recommended) to use a (sub)grid of the simulation grid. An independent point grid can be chosen just as well, but this requires additional interpolation of wavefields (and, thus, additional costs). For modal methods (that do not have a simulation point grid), a suitable point grid must be chosen on which the wavefields are evaluated.

In `ASKI` , these forward grid points (on which the wavefields are provided) are referred to as *wavefield points* and they are completely independent of the chosen inversion grid, except that they should be contained in the inversion grid cells and they cover the domain sufficiently dense such that each inversion grid cell contains enough wavefield points ("enough" depends on the method of pre-integration).

4.1.2 elastic model on wavefield points

The forward code must provide model values on the wavefield points in the model prametrization as chosen in the `ASKI` main paramter file (at the moment only elastic parametrizations, see user manual, chapter "Files", section on main parameter file), e.g. if model parametrization `isoLameSI` is chosen, values of isotropic Lamé paramters and densitiy in SI units must be available.

In `ASKI` , these model values are referred to as *kernel reference model.*

4.1.3 spectral displacement wavefield propagating from seismic sources into medium as well as its strains

On the wavefield points, the forward code must provide values of spectral displacement excited by each of the seismic sources involved in an inversion. The displpacement components (direction of particle movement) must be provided in directions of global Cartesian coordinates (even for spherical media). Otherwise the implemented kernel formulae do not work. For time-domain forward methods, the forward code must be extended by an on-the-fly Fourier transform in order to provide this quantity (transformation of the wavefields to frequency domain *after* the complete simulation i.e. not on-the-fly is of course also possible, but in case of SPECFEM was considered unsuitable due to huge memory requirements, also refer to 4.5 (page 13)).

In the same way, `ASKI` requires strains on the wavefield points, i.e. quantities

$$\frac{1}{2}\left(\frac{\partial u_k}{\partial x_\ell} + \frac{\partial u_\ell}{\partial x_k}\right)$$

for all global Cartesian directions k and ℓ, where u denotes particle displacement.

In `ASKI` , these spectral displacment and strain values are referred to as *kernel displacement.*

4.1.4 spectral synthetic data in format required by **ASKI**

Essentially the same values as for kernel displacement, but evaluated at the receiver positions in directions of the used station components (by which the measured data is provided, see user manual chapter “Basic steps”, section on data in `ASKI`). Also these values must be provided in the file format of synthetic data (see user manual, chapter “Files”, section on synthetic data files).

In the future, it would be better to have a set of interface modules for synthetic data (just as for wavefield points, kernel reference model, kernel displacement, kernel Green tensor $\rightarrow$ 7.3 (page 20))

4.1.5 spectral Green’s functions originating at receiver positions as well as its strains

“Green’s functions originating at receiver positions” here means the spectral displacement field (on wavefiled points) excited by a unit point force at the receiver position (pointing into the direction of a particular station component, i.e. there is such a Green’s function for each station component) using an impulsive source-time function (Dirac-delta impulse). For time-domain forward methods that need to use an approximation to the Dirac-delta impulse (e.g. a very thin Gaussian source-time-function), it is highly recommended to deconvolve it from the resulting wavefield: In case of SPECFEM this was found out to be necessary, because the inversion process fitted phase-shifted seismograms when just using a thin Gaussian. Moreover (just as for kernel displacements), in time-domain forward methods an on-the-fly Fourier transform must be applied to produce spectral output $\rightarrow$ 4.5 (page 13)

In the same way, `ASKI` requires the strains of Green’s functions on the wavefield points, i.e. quantities

$$\frac{1}{2}\left(\frac{\partial g_k}{\partial x_\ell} + \frac{\partial g_\ell}{\partial x_k}\right)$$

for all global Cartesian directions k and ℓ, where g denotes the particle displacement of a particular Green’s function.

In `ASKI` , these spectral displacment and strain values are referred to as *kernel Green tensor*.

4.1.6 some script running all simulations in a FWI iteration

For iterative FWI with `ASKI` , in each iteration the kernel displacement wavefields must be produced for each event in the event list and kernel Green tensor wavefields must be produced for each station in the station list at each station component involved in the inverted data set. In order to conduct all required simulations conveniently (and consistently!), it is highly recommended to have some kind of script that gets information from `ASKI` parameter files (about event, station locations, paths where to write the file output etc), sets all required parameter files of the forward code and conducts the simulation (by issuing a system call or something). For SPECFEM this is done by a (huge) python script `run_specfem3d(Globe/Cartesian) ForASKI_simulations.py`

4.2 Make `ASKI` understand the forward code: `ASKI` interface submodules

In order to make `ASKI` understand a new forward code, several things have to be extended / modified in `ASKI` 's source code. First of all, a forward code is denoted by a character string (without spaces) that is read from key `FORWARD_METHOD` of the main parameter file. All modules and routines for which any action depends on the forward code will check the value of this character string.

In function `validTypeInversionGrid` of module `inversionGrid`, it should be implemented whether the new forward method is incompatible with some inversion grids (e.g. Cartesian methods should be incompatible with spherical inversion grids and vice verca) or integration weight types (e.g. "external" integration weights, type 6, are only supported by special methods).

In functions `methodHasComplexKernelFrequency` and `getComplexKernelFrequency` of module `complexKernelFrequency`, please define whether the new forward code uses complex frequencies (such as Gemini) and define how they compute from frequency index and frequency step.

Furthermore, for all modules `wavefieldPoints`, `kernelReferenceModel`, `kernelDisplacement`, `kernelGreenTensor` the following steps need to be done, *exemplarily* described here for `kernelDisplacement` and new forward code name NEWCODE

1. Carefully study the functionality of module `kernelDisplacement` in terms of what routines can be called from outside and what they provide (e.g. strains and displacement fields are provided on all wavefield points for one frequency that was read in before by `readFrequencyKernelDisplacement`). It is helpful to compare the submodules of existing forward methods (e.g. `specfem3dKernelDisplacement`). This way, you should find out how a submodule `newcodeKernelDisplacement` for code NEWCODE could be organized best and it can even give you a hint how method NEWCODE could best provide any file output that is to be newly implemented.
2. Create the new module `newcodeKernelDisplacement` (can be of different name, could be more than one module, but please keep it simple!):
   (a) Declare derived type `newcode_kernel_displacement` defining the kernel displacement object of the new forward method (e.g. arrays and scalars for values and meta data required to manage the wavefield output provided by the forward code).
   (b) For each function or subroutine in module `kernelDisplacement` that forks to a particular forward method, implement a realization for your new module (either as a full routine in module `newcode_kernel_displacement` or as a short piece of code in the if-statements of module `kernelDisplacement` if the respective feature is not supported or has a trivial/general implementation). This assures that the new forward code is allowed to define its own file formats or to re-use standard (parameter)files (providing here the knowledge of how to access these files or any required information). *Make sure* that the respective routine in `kernelDisplacement` works correctly when it is called, refer to the (doxygen) commentary of the `kernelDisplacement` routine: e.g. usually only pointers to values are passed to module `kernelDisplacement` and deallcation has to be done

3. Add the new forward method to module `kernelDisplacement`:
    (a) Use the module by `use newcodeKernelDisplacement`.
    (b) For each function or subroutine in module `kernelDisplacement` that forks to a particular forward method, add an `else if` branch and call the respective routine in module `newcodeKernelDisplacement` (or implement trivial solution directly).
4. Update `rules.mk`: If module `newcodeKernelDisplacement` uses modules `module1`, `module2`, `module3` (etc.) add a new line of form
`newcodeKernelDisplacement.o: module1.o module2.o module3.o`
Otherwise, `rules.mk` does not need to be modified. Alternatively, use `create_rules_mk.py` → 3.3 (page 8)
5. Update `Makefile`: Every rule (to compile an `ASKI` executable) that depends on `kernelDisplacement.o` must now additionally depend on `newcodeKernelDisplacement.o` . As an alternative to expanding the rules manually, you may utilize script `py/create_Makefile_rule.py` → 3.2 (page 8)

4.3 Import updated model to the forward code for next iteration of FWI

In order to do iterative FWI with `ASKI` , the model derived in an iteration of FWI must be used as background Earth model for the next iteration of FWI. Hence, it must be transferred back into the forward solver. This process can be very different, dependent on the type of forward method (modal method or grid-based) and dependent on whether only a sub grid was used as wavefield points for `ASKI` and whether the simulation domain of the forward code exceeds the `ASKI` inversion domain.

`ASKI` handles inverted Earth models (model updates as well as new model values) by different files that are not self-consistent (values in `.kim` files correspond to inversion grid cells, which themselves are defined elsewhere, namely in an individual way by the respective inversion grid sub module). This storage of information is intended for internal `ASKI` use only. Therefore, `ASKI` offers to export such inverted models in a self-consistent format that may be used by the individual fortward code to communicate the model information (`ASKI` executable `exportKim`, option `-otxt`, see `ASKI` user manual). The resulting text file contains coordinates of inversion grid cell centers as well as the model values which are assigned to these points. The cell centers act as some kind of unstructured grid of control nodes on which the model values are given. Furthermore, information about neighbouring cells (i.e. neighbouring control nodes) as well as approximate average spatial extend of the inversion grid cells (i.e. the influence radius of the control nodes) are given in the file. The `ASKI` user manual dedicates a section in chapter “Files” to the exact format of this text file.

Since in `ASKI` the forward and inversion grid are in general independent of each other (and forward codes do not even need to be grid-based), you will require an individual transference of `ASKI` inverted models to the model description of the new forward code. Modal methods will have to approximate expansion coefficients from the model values on the control nodes (inversion grid cell centers) and grid-based forward codes using structured grids may realize much simpler interpolation methods than those using unstructured grids. E.g. for the `SPECFEM` and `NEXD` forward methods, an unstructured 3D interpolation after Shepard [She68] was implemented, which is founded on inverse-distance weighting and accounts for issues of nearby points, direction and slope. For sure, the current implementation of subroutine `shepard_interpolation_model_ASKI` in module `model_ASKI` of the `SPECFEM3D_Cartesian_for_ASKI` extension package (file `model_external_values.f90`) can be improved w.r.t. computational performance, but it may well serve you as an illustration how to implement Shepard's agorithm.

4.4 As an option: use forward-code-specific quadrature rules for kernel pre-integration

Using integration weights of type 6 ("external integration weights") is only supported along with a suitable type of inversion grid, at the moment only `specfem3dInversionGrid`. This type of integration weights is potentially useful for element-based (weak-form) forward methods: Their own quadrature rules for integration onto their elements may be re-used by `ASKI` for kernel pre-integration. Also this may be beneficial for special kinds of forward grids in combination with specific inversion grids, where you have certain pre-knowledge of how to integrate over the cells with high precision.

In order to realize the use of these special kinds of integration rules, the inversion grid module needs to provide the integration weights: routine `transformToStandardCellInversionGrid` must be called with `type_standard_cell` equal to -1 on input in order to request the inversion grid to return full integration weights instead of jacobian values in return variable `jacobian`. Hence, this is only possible for specific types of inversion grids that support this input value (at the moment only `specfem3dInversionGrid`). So you might need to add another inversion grid type to `ASKI` that works with your forward code (see sec. 6.2) or modify an existing inversion grid submodule accordingly. In particular, routine `validTypeInversionGrid` must accept the use of type 6 integration weights along with your inversion grid type.

4.5 Time-domain forward methods: Choosing a method of on-the-fly Fourier transform

For time-domain forward methods, the spectral wavefields need to be produced by Fourier transform from the time-domain wavefield *at every wavefield point*. Doing this by the fast Fourier algorithm (FFT) has several issues/drawbacks:

- since the time-series will be heavily oversampled (due to Courant stability criteria), the frequency-domain spectra will be oversampled, too (you compute more than you actually need, usually you require output at very few frequencies only)
- the frequency discretization of the output spectra is pre-defined by the time sampling of the time series and cannot be chosen independently (as requested by the `ASKI` user and defined in the `ASKI` main/iter parameter files)
- for realistic applications, you usually cannot hold the complete wavefield (at all wavefield points) in memory until you can apply the FFT at the very end of the simulation (for FFT you need to have the complete time series available)

In order to evaluate the Fourier transform of the waveifields at the particular set of requested frequencies $f_k = \omega_k/2\pi$, $k = 1, N_F$, the sums

$$S(\omega_k) = S_k = \Delta t \sum_{j=1}^{N_T} s(t_j)\, e^{-i\omega_k t_j} = \Delta t \Big[\sum_{j=1}^{N_T} s(t_j) \cos(-\omega_k t_j) \; + \; i \sum_{j=1}^{N_T} s(t_j) \sin(-\omega_k t_j) \Big] \quad , \tag{1}$$

have to be computed for each k, where the $s(t_j)$ represent samples of displacement (or strain) components at one location of a wavefield point (or the receiver position in case of producing spectral synthetic data as described in 4.1.4 (page 10)). Note that eq. (1) is a time-discretized simple approximation of the Fourier transform of the continuous signal $s(t)$:

$$S(\omega) = \mathcal{F}[s(t)](\omega) = \int_{-\infty}^{\infty} s(t)\, e^{-i\omega t} \, \mathrm{d}t \; . \tag{2}$$

In discretized setting, time t_j is represeted as $(j-1)\Delta t$ (time starts at $t_1 = 0$) and angular frequency ω_k as $2\pi f_k = 2\pi \ell_k \Delta f$, where ℓ_k is the k^{th} `ASKI` frequency index (normally denoted as `jf` or so, here j is already used so please excuse this different notation ℓ here).

A straight-forward way of evaluating the left-hand-side representation of (1) on the fly (i.e. *without* having all samples $s(t_j)$ simultaneously available but only the latest sample at a time) would be to add summand $s(t_j)\, e^{-i\omega_k t_j}$ to a complex-valued variable S_k (initialized by 0) in iteration j of the forward simulation. This requires (in each iteration of the time loop, for each wavefield/strain component, for each wavefield point and for each frequency) 2 real-valued multiplications (multiplying the complex number $e^{-i\omega_k t_j}$ by the real number $s(t_j)$ and two real-valued additions (adding the complex number $s(t_j)\, e^{-i\omega_k t_j}$ to the current complex value in S_k). Note that the right-hand-side form of (1) has the very same requirements. Multiplying the resulting S_k by Δt at the very end (or incorporating Δt in pre-computed constants $e^{-i\omega_k t_j}\, \Delta t$, $\cos(-\omega_k t_j)\, \Delta t$ or $\sin(-\omega_k t_j)\, \Delta t$ that are computed once before the time loop) is neglected in this argumentation of performance.

A computationally more efficient way of computing sum (1) on the fly is to apply an algorithm first described by Gerald Goertzel in 1958 [Goe58] which for given x computes sums of the form

$$\sum_{j=0}^{N} a_j \cos(j\, x) \quad \text{and} \quad \sum_{j=1}^{N} a_j \sin(j\, x)$$

and can be adapted here to compute S_k by the righ-hand side of (1) choosing $x = -\omega_k\, \Delta t$ and $a_j = s(t_j)$:

$$W_r = 2\cos(x) \tag{3}$$

$$W_i = \sin(x) \tag{4}$$

$$U_1 = 0 \tag{5}$$

$$U_2 = 0 \tag{6}$$

$$\text{for } j = N_T, \ldots 2 \text{ do:} \tag{7}$$

$$U_0 = s(t_j) + W_r\, U_1 - U_2 \tag{8}$$

$$U_2 = U_1 \tag{9}$$

$$U_1 = U_0 \tag{10}$$

after iterating (backwards in time) down to t_2, the spectral value S_k computes as:

$$\mathfrak{Re}(S_k) = \Delta t \; (s(t_1) + 0.5\, W_r\, U_1 - U_2) \tag{11}$$

$$\mathfrak{Im}(S_k) = \Delta t\, W_i\, U_1 \tag{12}$$

In each iteration of the time loop (i.e. lines (8) to (10)), for each wavefield/strain component, for each wavefield point and for each frequency here only 1 real-valued multiplication is required (in line (8)) and also two real-valued additions (in the same line). Compared with the explicit summation described above, we save one multiplication here. This sounds not so much, but since a real-valued multiplication is usually much more expensive than a real-valued summation, this algorithm may be close to twice as fast as the explicit summation above.

The problem of the algorithm in this form, however, is that it requires *reverse* knowledge of the imput time series $s(t_j)$, first requiring the value at last time and iterating backwards in time. This, of course, does not allow to do an on-the-fly Fourier transform in simulations of seismic waves (propagating *forward* in time).

Looking at the proof of the algorithm as given in [Goe58], I (Florian Schumacher, Sep 2016) tried to find an analogous recursion relation that allows to step through the time series in *forward* direction from $s(t_1)$ up to last sample $s(t_{N_T})$) but was not successful. However, there is a trivial way to adapt the above algorithm for our purposes, making use of properties of the Fourier transform: Inserting the time series $s(t_j)$ into the algorithm in *forward* order (starting with $s(t_1)$, $s(t_2)$ ...) means mathematically to compute the Fourier transform of $s(-t)$. This can be compensated by negating the frequency, i.e. choosing $-\omega_k$ instead of ω_k everywhere, due to the time-reversal property of the Fourier transform:

$$\mathcal{F}[s(-t)] = S(-\omega)$$

However, this only fixes the time-reversal of the time series. Since Goertzel's algorithm assumes the last sample of the time to come first, technically the time series is assumed to start at time $-T = -(N_T - 1)\Delta t$ and to end at time 0. Therefore, additionally to negating the frequency, we need to compensate for a time shift (resulting in a constant phase shift of the output spectrum) applying the time-shift property of the Fourier transform:

$$\mathcal{F}[s(t-T)] = S(\omega)e^{i\omega\,(-T)}$$

Taking all considerations into account, we need to adapt the above original algorithm by Goertzel for our purposes in the following way:

- Choose $x = +\omega_k\,\Delta t$
- Step forward through the time series, i.e. modify line (7) as

$$\text{for } j = 1, \dots (N_T - 1) \text{ do:} \tag{7-b}$$

- Correct the computation of the real part of the spectral value by modifying line (11) as

$$\mathfrak{Re}(S_k) = \Delta t\ \left(s(t_{N_T}) + 0.5\,W_r\,U_1 - U_2\right) \tag{11-b}$$

- After the algorithm is completed, additionally correct for the time shift by

$$S_k = S_k\,e^{-\omega_k(N_T-1)\Delta t} \tag{13}$$

Demonstrating the significant performance improvement when using Goertzel's algorithm, compared with the explicit summation, these two algorithms were implemented in the example program `goertzel.f90`, provided in directory `devel/Goertzel/` of the `ASKI` repository. Please refer to `devel/Goertzel/README.md` on any information regarding this example program, in particular on how to compile and run it. *Please note*, however, that e.g. in case of forward code `SPECFEM3D_Cartesian`, the overall performance improvement of using Goertzel's algorithm (compared with explicit summation) is *not* noticeable, since these operations comprise only a part of the additional costs for producing output for `ASKI` and other things like array/memory access etc. also play an important role w.r.t. overall performance.

5 Create new executables of the `ASKI` toolbox based on its software library

New Fortran executables can be created using existing `ASKI` Fortran modules (or creating new ones, of course). A list of all `ASKI` Fortran modules along with short descriptions about what functionality they provide, is provided by the doxygen documentation (chapter 1 in the LaTex document and "Modules" (`namespaces.html`) in the html document). In order to make life easier for all `ASKI` users, the command line behaviour should be very similar to all other Fortran executables of `ASKI` (i.e. using module `argumentParser` and printing usage message if some input is incorrect and, if required, setting `main_parfile` as last positional argument).

In order to correctly compile the new executable, here having exemplary name `newTool`, you need to:

1. Update `rules.mk`: If program `newTool` uses modules `module1`, `module2`, `module3` (etc.) add a new line of form
   `newTool.o: module1.o module2.o module3g.o`
   If any of the modules, e.g. `module2` was newly created and itself depends on other modules, e.g. `module4`, `module5`, add a new line of form
   `module2.o: module4.o module5.o`

   If `newTool` does not use any modules and is not dependent on other files, `rules.mk` does not need to be modified.

   As an alternative to modifying `rules.mk` manulally (especially when extending existing modules that are used in many other modules), you can ultilize `create_rules_mk.py` → 3.3 (page 8)

2. Add a new rule for `newTool` to `Makefile`:

   (a) Run script
   `devel/create_Makefile_rule.py newTool > rule.txt`
   from the `ASKI` installation directory → 3.2 (page 8)

   (b) Add content of `rule.txt` to `Makefile` (GNU-Make rule to compile executable `newTool`). The required libraries (`BLAS`, `LAPACK` etc.) should have been accounted for correctly (otherwise you should update script `create_makefile_rule.py`, in particular routine `required_libs_for_depobs`; please add any library requirements for `newTool`).

   (c) All general `ASKI` executables from which all `ASKI` users benefit, should be part of the rule of target `all`, so please add `newTool` there unless `newTool` is a very special executable that should be compiled explicitely by experienced users only (then, please stress this in the documentation).

6 Inversion grids

6.1 Add another type of inversion grid

1. Create a new module (here using exemplary name `newInversionGrid`):

   (a) Declare derived type `new_inversion_grid` defining the new inversion grid.
   (b) For each function or subroutine in module `inversionGrid` implement a realization for inversion grids of type `new_inversion_grid` (either as a full routine in module `newInversionGrid` or as a short piece of code in `select` statements of module `inversionGrid` if the respective feature is not supported or has a trivial/general implementation).

2. Add the new inversion grid type (here using exemplary name `newInversionGrid`) to module `inversionGrid`:
   (a) Use the module by `use newInversionGrid`.
   (b) Increase `character_length_type_inversion_grid` if necessary.
   (c) Extend string `all_valid_types_inversion_grid`.
   (d) Add pointer of type `new_inversion_grid` to derived type `inversion_grid`.
   (e) For each function or subroutine, add a `case` in the `select` clauses and call the inversion-grid-specific routine (or implement trivial solution).
3. Update `rules.mk`: If module `newInversionGrid` uses modules `module1`, `module2`, `module3` (etc.) add a new line of form
   `newInversionGrid.o: module1.o module2.o module3.o`
   Otherwise, add a new line of the form
   `newInversionGrid.o:`
4. Update `Makefile`: Every rule (to compile an `ASKI` executable) that depends on `inversionGrid.o` must now additionally depend on `newInversionGrid.o`

6.2 Using element grid of element-based forward method as inversion grid

For element-based forward methods (such as e.g. SPECFEM3D), it can be beneficial to use the volumetric elements as inversion grid cells along with using the contained forward grid points as wavefield points: The localization of wavefield points inside the inversion grid cells is known in this case and, optionally, also quadrature rules for numerical integration can be either known or should be computable (semi-)analytically. If you would like to add this kind of functionality for your own newly added element-based forward method, you are advised to have a close look at the realization of module `specfem3dInversionGrid`.

By SPECFEM3D, for instance, all forward grid points in a certain spatial subdomain are stored element-wise in a known order in some file, so that it is possible for module `specfem3dInversionGrid` to extract the 8 corner points of the spectral elements from that set of coordinates (which are required for the 8-corner hexahedral vtk cell output) and to trivially locate the wavefield points inside the elements (and give their position inside the hexahedral reference element $[-1,1] \times [-1,1] \times [-1,1] \subset \mathbb{R}^3$).

As an optional feature, the module `specfem3dInversionGrid` can provide the integration weights for numerical integration of the kernels on the spectral element that are also used by the spectral element method. These have a high numerical accuracy due to the special distribution of points inside the element (GLL points). In this case `ASKI` can benefit from this knowledge and does not need to compute its own integration weights. Alternatively, other types of `ASKI` integration rules can be used along with an inversion grid of type `specfem3dInversionGrid`.

6.3 `chunksInversionGrid` : implement a method of cell refinement

First, you should make yourself familiar with module `chunksInversionGrid`. Unfortunately, this module has kept growing and became some kind of monsterous construct. There is definitely a lot of commentary in the code, which mostly is formatted using doxygen syntax (regarding routines and variables descriptions), so it might be worth referring to the doxygen documentation. Additionally, fig. 1 shows the positioning of chunks in `LOCAL_FLAT` projection.

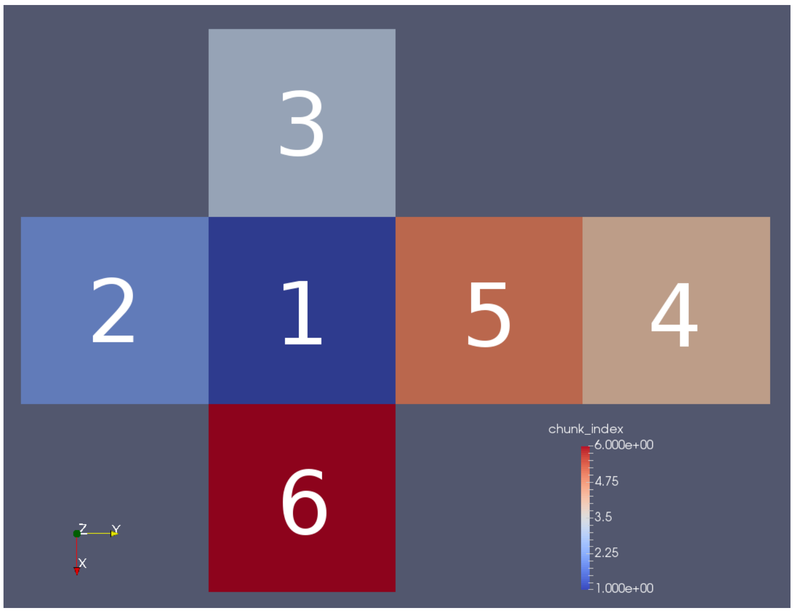


Figure 1: distribution of chunks of the chunksInversionGrid in `LOCAL_FLAT` projection

In order to make `ASKI` support a new method of base cell refinement for inversion grids of type `chunksInversionGrid`, module `chunksInversionGrid` must be extended, as indicated by comments

```
!####################################
! ADD YOUR REFINEMENT METHOD HERE
!####################################
```

in file `chunksInversionGrid.f90`. In particular,

- In routine `readCheckParFileChunksInversionGrid`, the new value of keyword `CHUNKS_INVGRID_CREF_METHOD` (name of refinement method) must pass all checks and (if required) any expected values must be read from keyword `CHUNKS_INVGRID_CREF_PARAMETERS`. Alternatively, you may need to introduce another keyword to read in some filenames or introduce some naming convention for files containing additional information.

- Routine `doCellRefinementChunksInversionGrid` must execute one or more new subroutines (to be implemented specifically for the new refinement method) which do the actual refinement, as described in the commented toy subroutine `requiredSubroutinesForNewRefinementMethod`.

7 Things "to do" for the `ASKI` developers

Please make sure to remove those aspects from this section of the `ASKI` developers manual which you have fixed.

7.1 Todos that are straightforward to realize

7.1.1 use function `nextPathDataModelSpaceInfo` everywhere

Originally, loops over event-station paths were done manually (by first getting the paths by `getPathsDataModelSpaceInfo` and then looping over them). Some of these implementations might be more clear if a construct like
`do while( nextPathDataModelSpaceInfo(...) )`
was used, *directly* accessing information on components and frequencies via function `nextPathDataModelSpaceInfo` (instead of using `getIndxDataSamples` inside the manual loop over the paths).

Especially in the module `dataModelSpaceInfo` itself, the old approach (looping over paths manually) is done thoroughly, as well as in `investigateDataResiduals` (sensible also for `computeDataFromKernelSystem`?!)

7.1.2 finish implementing module `parameterCorrelation`

So far, there is only a quick hack done for constructing objects (for reasons of testing): In the parameter correlation file as given by the main parameter file, assume only ONE line, containing for parameter vs the parameters of rho , vp (in that order, space separated). Please refer to comments in code file `parameterCorrelation.f90` containing `FS FS` (always referring to author Florian Schumacher).

7.2 Todos that might require some more work

7.2.1 completion of module `ecartInversionGrid`

The module `ecartInversionGrid` is exensively (!) documented in the code file `ecartInversionGrid.f90`, mostly *without* doxygen syntax (so you can only access the documentation by reading the code file's commentary sections).

The following subroutines/functionality must still be implemented for *hexahedral* (i.e. `hex8`) cells:

1. `locateWpInsideHex8CellEcartInversionGrid`
2. `transformToStandardHexEcartInversionGrid`
3. `getVolumeHex8EcartInversionGrid`
4. `createFaceNeighboursEcartInversionGrid`: only face neighbours are searched for among `tet4` cells, *which still is unstable, see below*. Still to do: include `hex8` cells in the search, i.e. search for neighbours among `hex8` cells *and* between `hex8` and `tet4` cells.

> *Alternatively*, a neighbour search (i.e. creation of the neighbours file) could be done outside this module by arbitrary software, e.g. already at the stage of exporting the grid from Trelis (if possible?!) by script `cubit2ASKIecartInversionGrid.py`.

Also, the neighbour search among the `tet4` cells in subroutine `createFaceNeighboursEcartInversionGrid` (following the approach of testing signed areas of triangles which are spanned by one point of the first tri face and an edge of the other tri face, as described in section 4 of paper [DG02]) turned out to be still unstable and must be fixed: please compare example files and `README` in
`doc/ASKI_developers_manual/TODO_ecartInversionGrid_cell_neighbours/`
(the inversion grid given there is also given as an example in the user manual and in `doc/example_ecartInversionGrid/`, as of August 2016). For some `tet4` cells not all neighbours are found correctly, for some cells not even one of the neighbours is found. The reason for that should be due to the numerical implementation of the search algorithm (possibly problems comparing floating point numbers).

7.2.2 improvement of the `ASKI` user manual, section entitled "What is `ASKI` ?"

We might include some pictures there: a set of points inside the volumes of the inversion grid, a sensitivity kernel for one specific datum (real part of some frequency sample of a seismogram (src,rec,comp)) explaining that positive high values at one scatterer mean that this very datum increases (significantly) if the current model parameter is increased at this scatterer, etc.

7.3 Ideas for new `ASKI` functionality

7.3.1 file-interface to module `dataModelSpaceInfo`

The parsing of dataModelSpaceInfo files in module `dataModelSpaceInfo` (routine `createFromFileDataModelSpaceInfo`) is very inconvenient (as it is implemented in Fortran). Alternatively (as an *additional* feature!), module `dataModelSpaceInfo` can support to read in a text file that contains listed information of all data samples and/or model values (maybe two separate files for data samples and model values?!), i.e. basically arrays `evid`, `staname`, `comp`, `ifreq`, `imre`, `wdata`, `normalization_mdata`, `normalization_sdata`, `normalization_kernel`, as columns (for model values file: this means arrays `param`, `cell`, respectively). Also memorize values like `ndata`, `nmval`, `parametrization` in these files. *Additionally* on could remember in these files information like paths, number of paths, which receiver components, frequencies are contained in which paths, all different comp, all different ifreq, etc. The best thing would be to memorize all (or a lot of) information required by the programs, e.g. all data indices per path, etc. This could allow for quicker access in the application of the programs (although everything is working now).

7.3.2 forward-method interface modules for synthetic data

It would be much better to have a set of interface modules for synthetic data (just as for wavefield points, kernel reference model, kernel displacement, kernel Green tensor), because the forward code requires knowledge about the `ASKI` -internal file format of synthetic data. These interface modules of synthetic data could use the `discreteFourierTransform` module (if required) and know the location, name and format of any synthetic data files written by the forward code.

7.3.3 support integer list of formats like “3-5,9,13,19-21”

For some input on the command line (the `ASKI` Fortran executables usually use the module `argumentParser`), explicit vectors of integer values are required (e.g. `-subs` of executable `chunksInvgrid2vtk`). At the moment, all indices have to be given on the command line explicitely (space-separated). Usually one has to apply quotes like
`-subs "3 4 5 9 13 19 20 21"`
This can get very long and lengthy and unconvenient for a user that does not want to use option `-all_subs`.

Such integer vectors are also used in another respect in `ASKI` , namely for defining frequency indices in parameter files (main parameter file and iteration-step-specific parameter file). If an input line there becomes too long (because of so many space-separated items in the line), it might even not be possible to read this line properly by module `inputParameter`, since it uses a maximum character length for value strings of 500.

This functionality could be realized by implementing a new module (called something like `integerRanges`) which knows how to transform strings of form “3-5,9,13,19-21” to integer arrays `3 4 5 9 13 19 20 21`. This module could even support different kinds of string formatting which it automatically detects (if it does not detect a known format, it returns an error). It is thinkable to also support formats like `all-except:3-5,9,13,19-21`, but in this case it must be clear to module `integerRanges` what “all” means, i.e. to which entirety the numbers refer.

7.3.4 other methods of smoothing kernel inverted models

In module `linearModelRegularization` the only supported smoothing method at the moment is implemented in subroutine `addNeighbourAverageSmoothingLinearModelRegularization` and builds averages over direct face neighbours (possibly accounting for different boundary conditions on boundaries where there are no neighbours or boundaries that should not be smoothed over (last aspect not yet implemented but possible after somehow defining internal boundaries)).

An alternative method that *does not require neighbour information* and is more physically motivated is suggested in the following:

- Given a fixed radius r, detect for each inversion grid cell c all other cells c_n for which the distance between the center points of c and c_n is $\leq r$ (regard those cells as “neighbours” in this sense, i.e. “central-radius-neighbours”, although they do not need to be *face* neighbours in the above sence). This detection of “closeby” cells should be implemented in module `inversionGrid` and its containing submodules as an alternative “neighbour search” called something like `getIndicesCentralRadiusNeighboursInversionGrid`
- Then build smoothing conditions as averages on all central-radius-neighbours. Optionally, the weight w_n of neighbour c_n in the linear combination of this average may depend on the distance between center points of c and c_n (as a result: cell further away have a smaller influence on the smoothed value).

- As an alternative to searching for central-radius-neighbours by a fixed radius r (constant r throughout the inversion grid, regardless of the spatial extent of inversion grid cell c which constitutes the local spatial resolution of the inversion grid), the search may be performed by a dynamic radius r_c which depends on the particular inversion grid cell c for which neighbours should be found. r_c may e.g. be computed by multiplying the radius of cell c by a fixed given factor $f > 1$ (constand f throughout the inversion grid). Thereby, the influence of the smoothing operation would correspond to the local spatial resolution of the inversion grid (in regions of finer spatial resolution, the smoothing would have a smaller range of influence, in regions with coarser spatial resolution, the model values on the inversion grid cells would depend on values at larger distances). This variety of the central-radius neighbours search comes closes to the search of face neighbours.

References

[Boh02] Thomas Bohlen. Parallel 3-d viscoelastic finite difference seismic modelling. *Computers & Geosciences*, 28(8):887–899, 2002.

[DG02] Olivier Devillers and Philippe Guigue. Faster Triangle-Triangle Intersection Tests. Technical Report RR-4488, INRIA, June 2002.

[FD95] W. Friederich and Jörg Dalkolmo. Complete synthetic seismograms for a spherically symmetric earth by a numerical computation of Green's function in the frequency domain. *Geophys. J. Int.*, 122:537–550, 1995.

[Goe58] Gerald Goertzel. An algorithm for the evaluation of finite trigonometric series. *The American Mathematical Monthly*, 65(1):34–35, 1958. https://doi.org/10.2307/2310304.

[Lam15] Lasse Lambrecht. *Forward and inverse modeling of seismic waves for reconnaissance in mechanized tunneling*. doctoral dissertation, Institute of Geology, Mineralogy and Geophysics, Ruhr-Universität Bochum, 2015. urn:nbn:de:hbz:294-44164.

[SF16] Florian Schumacher and Wolfgang Friederich. ASKI: A modular toolbox for scattering-integral-based seismic full waveform inversion and sensitivity analysis utilizing external forward codes. *SoftwareX*, 5:252 – 259, 2016.

[SFL16] Florian Schumacher, Wolfgang Friederich, and Samir Lamara. A flexible, extendable, modular and computationally efficient approach to scattering-integral-based seismic full waveform inversion. *Geophysical Journal International*, 204(2):1100–1119, 2016.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. ACM.

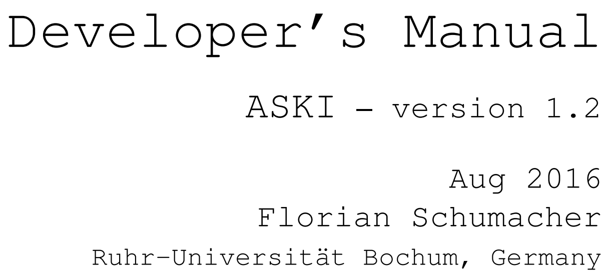
[TKL08] Jeroen Tromp, Dimitri Komatitsch, and Qinya Liu. Spectral-element and adjoint methods in seismology. *Communications in Computational Physics*, 3(1):1–32, 2008.

History

This is a section on the history of this document, i.e. its previously published versions, as referred to by the GNU Free Documentation License (version 1.3).

ASKI Developer's Manual, ASKI version 1.2, August 2016

Recognizable snippet from the title page (scaled):



Developer's Manual

ASKI - version 1.2

Aug 2016
Florian Schumacher
Ruhr-Universität Bochum, Germany

Title: ASKI Developer's Manual, ASKI version 1.2, August 2016
Year: 2016
Authors: Florian Schumacher (Ruhr-Universität Bochum, Germany)

This version of this document is provided for download (as of August 2016) at
`https://github.com/seismology-RUB/ASKI/releases/tag/v1.2`
being contained in the provided `.zip` code archive as file
`ASKI-1.2/devel/doc/ASKI_developers_manual_1-2_aug-2016.pdf`
Direct link (as of August 2016):
`https://github.com/seismology-RUB/ASKI/archive/v1.2.zip`

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this

License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you