

SpecSWD

Generated by Doxygen 1.13.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 specswd::Mesh Struct Reference	5
3.1.1 Detailed Description	7
3.1.2 Member Function Documentation	7
3.1.2.1 create_database()	7
3.1.2.2 interp_model()	7
3.1.2.3 project_kl()	7
3.1.2.4 read_model()	8
3.1.2.5 read_model_full_aniso_()	8
3.1.2.6 read_model_header_()	8
3.1.2.7 read_model_love_()	8
3.1.2.8 read_model_rayl_()	9
3.2 specswd::SolverLove Class Reference	9
3.2.1 Member Function Documentation	10
3.2.1.1 compute_egn()	10
3.2.1.2 compute_egn_att()	10
3.2.1.3 compute_group_kl()	10
3.2.1.4 compute_group_kl_att()	11
3.2.1.5 compute_phase_kl()	11
3.2.1.6 compute_phase_kl_att()	11
3.2.1.7 egn2displ()	12
3.2.1.8 egn2displ_att()	12
3.2.1.9 transform_kernels()	12
3.3 specswd::SolverRayl Class Reference	13
3.3.1 Member Function Documentation	14
3.3.1.1 compute_egn()	14
3.3.1.2 compute_egn_att()	14
3.3.1.3 compute_group_kl()	14
3.3.1.4 compute_group_kl_att()	15
3.3.1.5 compute_phase_kl()	15
3.3.1.6 compute_phase_kl_att()	16
3.3.1.7 egn2displ()	16
3.3.1.8 egn2displ_att()	16
3.3.1.9 prepare_matrices()	17
3.3.1.10 transform_kernels()	17
4 File Documentation	19

4.1 global.hpp	19
4.2 specswd.hpp	19
4.3 mesh.hpp	20
4.4 attenuation.hpp	21
4.5 GQTable.hpp	22
4.6 iofunc.hpp	22
4.7 schur.hpp	22
4.8 frechet_op.hpp	24
4.9 vti.hpp	30
Index	33

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

specswd::Mesh		
SEM mesh class	5
specswd::SolverLove	9
specswd::SolverRayl	13

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

global.hpp	19
specswd.hpp	19
mesh.hpp	20
attenuation.hpp	21
GQTable.hpp	22
iofunc.hpp	22
schur.hpp	22
frechet_op.hpp	24
vti.hpp	30

Chapter 3

Class Documentation

3.1 specswd::Mesh Struct Reference

SEM mesh class.

```
#include <mesh.hpp>
```

Public Member Functions

- void [read_model](#) (const char *filename)
read 1D model
- void [create_database](#) (float freq, float phi)
Create SEM database by using input model info.
- void **print_model** () const
print 1-D model information
- void **print_database** () const
- void **allocate_1D_model** (int nz0, int swd_type, int has_att)
- void **create_model_attributes** ()
create attributes for elastic/acoustic regions
- void [interp_model](#) (const float *param, const std::vector< int > &elmnts, std::vector< float > &md) const
interpolate elastic/acoustic model by using coordinates
- void [project_kl](#) (const float *frekl, float *kl_out) const
project kernels to original 1-D model
- void **create_material_info** ()
- void [read_model_header](#) (const char *filename)
read header of 1D model, including wave type, attenuation flag, attenuation model flag
- void [read_model_love](#) (const char *filename)
read 1D VTI model for Love wave
- void [read_model_rayl](#) (const char *filename)
read 1D VTI model for Rayleigh wave
- void [read_model_full_aniso](#) (const char *filename)
read 1D full anisotropy model model for Rayleigh wave
- void **compute_minmax_veloc** (float phi, std::vector< float > &vmin, std::vector< float > &vmax)
- void **create_db_love** ()
create database for Love wave
- void **create_db_rayl** ()
create database for Love wave
- void **create_db_aniso** ()
create database for Love wave

Public Attributes

- int **nspec**
- int **nspec_grl**
- int **nglob**
- std::vector< int > **ibool**
- std::vector< float > **skel**
- std::vector< float > **znodes**
- std::vector< float > **jaco**
- std::vector< float > **zstore**
- int **nspec_ac**
- int **nspec_el**
- int **nspec_ac_grl**
- int **nspec_el_grl**
- std::vector< char > **is_elastic**
- std::vector< char > **is_acoustic**
- std::vector< int > **el_elmnts**
- std::vector< int > **ac_elmnts**
- int **nglob_ac**
- int **nglob_el**
- std::vector< int > **ibool_el**
- std::vector< int > **ibool_ac**
- std::vector< float > **xrho_ac**
- std::vector< float > **xrho_el**
- bool **HAS_ATT**
- int **SWD_TYPE**
- std::vector< float > **xA**
- std::vector< float > **xC**
- std::vector< float > **XL**
- std::vector< float > **xeta**
- std::vector< float > **xN**
- std::vector< float > **xQA**
- std::vector< float > **xQC**
- std::vector< float > **xQL**
- std::vector< float > **xQN**
- int **nQmodel_ani**
- std::vector< float > **xC21**
- std::vector< float > **xQani**
- std::vector< float > **xkappa_ac**
- std::vector< float > **xQk_ac**
- int **nfaces_bdry**
- std::vector< int > **ispec_bdry**
- std::vector< char > **bdry_norm_direc**
- int **nz_tomo**
- int **nregions**
- std::vector< float > **rho_tomo**
- std::vector< float > **vpv_tomo**
- std::vector< float > **vph_tomo**
- std::vector< float > **vsv_tomo**
- std::vector< float > **vsh_tomo**
- std::vector< float > **eta_tomo**
- std::vector< float > **QC_tomo**
- std::vector< float > **QA_tomo**
- std::vector< float > **QL_tomo**
- std::vector< float > **QN_tomo**

- `std::vector< float > c21_tomo`
- `std::vector< float > Qani_tomo`
- `std::vector< float > depth_tomo`
- `std::vector< int > region_bdry`
- `std::vector< int > iregion_flag`
- `std::vector< char > is_el_reg`
- `std::vector< char > is_ac_reg`
- `float PHASE_VELOC_MIN`
- `float PHASE_VELOC_MAX`
- `float freq`

3.1.1 Detailed Description

SEM mesh class.

3.1.2 Member Function Documentation

3.1.2.1 `create_database()`

```
void specswd::Mesh::create_database (
    float freq0,
    float phi)
```

Create SEM database by using input model info.

Parameters

<i>freq0</i>	current frequency, in Hz
<i>phi</i>	directional angle

3.1.2.2 `interp_model()`

```
void specswd::Mesh::interp_model (
    const float * param,
    const std::vector< int > & elmnts,
    std::vector< float > & md) const
```

interpolate elastic/acoustic model by using coordinates

Parameters

<i>param</i>	input model parameter, shape(nz_tomo)
<i>elmnts</i>	all elements used, ispec = elmnts[i]
<i>md</i>	model required to interpolate, shape(nspec_el*NGLL + nspec_el_grl * NGRL)

3.1.2.3 `project_kl()`

```
void specswd::Mesh::project_kl (
    const float * frekl,
    float * kl_out) const
```

project kernels to original 1-D model

Parameters

<i>frekl</i>	derivatives, shape(nspec*NGLL+NGRL)
<i>kl_out</i>	derivatives on original 1-Dmodel, shape(nz_)

3.1.2.4 read_model()

```
void specswd::Mesh::read_model (
    const char * filename)
```

read 1D model

Parameters

<i>filename</i>	1D model file
-----------------	---------------

3.1.2.5 read_model_full_aniso_()

```
void specswd::Mesh::read_model_full_aniso_ (
    const char * filename)
```

read 1D full anisotropy model model for Rayleigh wave

Parameters

<i>filename</i>	1D model file
-----------------	---------------

3.1.2.6 read_model_header_()

```
void specswd::Mesh::read_model_header_ (
    const char * filename)
```

read header of 1D model, including wave type, attenuation flag, attenuation model flag

Parameters

<i>filename</i>	model filename
-----------------	----------------

3.1.2.7 read_model_love_()

```
void specswd::Mesh::read_model_love_ (
    const char * filename)
```

read 1D VTI model for Love wave

Parameters

<i>filename</i>	1D model file
-----------------	---------------

3.1.2.8 read_model_rayl_()

```
void specswwd::Mesh::read_model_rayl_ (
    const char * filename)
```

read 1D VTI model for Rayleigh wave

Parameters

<i>filename</i>	1D model file
-----------------	---------------

The documentation for this struct was generated from the following files:

- mesh.hpp
- database.cpp
- initialize.cpp
- interpolate.cpp
- io.cpp

3.2 specswwd::SolverLove Class Reference

Public Member Functions

- void **prepare_matrices** (const [Mesh](#) &M)
prepare M/K/E matrices for Love wave, an/elastic case
- void **compute_egn** (const [Mesh](#) &M, std::vector< float > &c, std::vector< float > &egn, bool use_qz=false)
compute Love wave dispersion and eigenfunctions, elastic case
- void **compute_egn_att** (const [Mesh](#) &M, std::vector< scmplx > &c, std::vector< scmplx > &egn, bool use_qz=false)
compute rayleigh wave dispersion and eigenfunctions, visco-elastic case
- float **group_vel** (const [Mesh](#) &M, float c, const float *egn) const
compute velocity of love wave, elastic case
- scmplx **group_vel_att** (const [Mesh](#) &M, scmplx c, const scmplx *egn) const
compute velocity of love wave, anelastic case
- void **compute_phase_kl** (const [Mesh](#) &M, float c, const float *egn, std::vector< float > &frekl) const
compute love wave phase velocity kernels, elastic case
- void **compute_phase_kl_att** (const [Mesh](#) &M, scmplx c, const scmplx *egn, std::vector< float > &frekl_c, std::vector< float > &frekl_q) const
compute love wave phase velocity kernels, visco-elastic case
- void **compute_group_kl** (const [Mesh](#) &M, float c, const float *egn, std::vector< float > &frekl) const
compute group velocity and kernels for love wave group velocity, elastic case
- void **compute_group_kl_att** (const [Mesh](#) &M, scmplx c, scmplx u, const scmplx *egn, std::vector< float > &frekl_u, std::vector< float > &frekl_q) const
compute love wave group velocity kernels, visco-elastic case
- void **egn2displ** (const [Mesh](#) &M, float c, const float *egn, float *__restrict displ) const
convert eigenvector to displacement, elastic case
- void **egn2displ_att** (const [Mesh](#) &M, scmplx c, const scmplx *egn, scmplx *__restrict displ) const
convert eigenvector to displacement, visco-elastic case
- void **transform_kernels** (const [Mesh](#) &M, std::vector< float > &frekl) const
transform modulus kernel to velocity kernel, Love wave case

3.2.1 Member Function Documentation

3.2.1.1 compute_egn()

```
void specsrd::SolverLove::compute_egn (
    const Mesh & mesh,
    std::vector< float > & c,
    std::vector< float > & egn,
    bool use_qz = false)
```

compute Love wave dispersion and eigenfunctions, elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	dispersion, shape(nc)
<i>egn</i>	eigen functions(displ at y direction), shape(nc,nglob_el)
<i>use_qz</i>	if false, only compute phase velocities

3.2.1.2 compute_egn_att()

```
void specsrd::SolverLove::compute_egn_att (
    const Mesh & mesh,
    std::vector< scmplx > & c,
    std::vector< scmplx > & displ,
    bool use_qz = false)
```

compute rayleigh wave dispersion and eigenfunctions, visco-elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	dispersion, shape(nc) $c = c_0(1 + iQL^{-1})$
<i>displ</i>	eigen functions(displ at y direction), shape(nc,nglob_el)
<i>use_qz</i>	if true, save QZ matrix

3.2.1.3 compute_group_kl()

```
void specsrd::SolverLove::compute_group_kl (
    const Mesh & mesh,
    float c,
    const float * egn,
    std::vector< float > & frekl) const
```

compute group velocity and kernels for love wave group velocity, elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	current phase velocity
<i>egn</i>	eigen function, shape(nglob_el)
<i>frekl</i>	Frechet kernels (N/L/rho) for elastic parameters, shape(3,nspec*NGLL + NGRL)

3.2.1.4 compute_group_kl_att()

```
void specs wd::SolverLove::compute_group_kl_att (
    const Mesh & mesh,
    scmplx c,
    scmplx u,
    const scmplx * egn,
    std::vector< float > & frekl_u,
    std::vector< float > & frekl_q) const
```

compute love wave group velocity kernels, visco-elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	current complex phase velocity
<i>u</i>	current complex group velocity
<i>egn</i>	eigen function, shape(nglob_el)
<i>frekl_u</i>	$d\text{Re}(u)/d(N/L/QN/QL/\rho)$ shape(5,nspec*NGLL + NGRL)
<i>frekl_q</i>	$d(q_i)/d(N/L/QN/QL/\rho)$ shape(5,nspec*NGLL + NGRL)

3.2.1.5 compute_phase_kl()

```
void specs wd::SolverLove::compute_phase_kl (
    const Mesh & M,
    float c,
    const float * egn,
    std::vector< float > & frekl) const
```

compute love wave phase velocity kernels, elastic case

Parameters

<i>M</i>	Mesh clas
<i>c</i>	current phase velocity
<i>egn</i>	eigen function, shape(nglob_el)
<i>frekl</i>	Frechet kernels (N/L/rho) for elastic parameters, shape(3,nspec*NGLL + NGRL)

3.2.1.6 compute_phase_kl_att()

```
void specs wd::SolverLove::compute_phase_kl_att (
    const Mesh & M,
    scmplx c,
    const scmplx * egn,
    std::vector< float > & frekl_c,
    std::vector< float > & frekl_q) const
```

compute love wave phase velocity kernels, visco-elastic case

Parameters

<i>M</i>	mesh class
<i>c</i>	current complex phase velocity
<i>egn</i>	eigen function, shape(nglob_el)
<i>frekl</i> ↔ <i>_c</i>	$d\text{Re}(c)/d(N/L/QN/QL/\rho)$ shape(5,nspec*NGLL + NGRL)
<i>frekl</i> ↔ <i>_q</i>	$d(q_i)/d(N/L/QN/QL/\rho)$ shape(5,nspec*NGLL + NGRL)

3.2.1.7 egn2displ()

```
void specswd::SolverLove::egn2displ (
    const Mesh & M,
    float c,
    const float * egn,
    float *__restrict displ) const
```

convert eigenvector to displacement,elastic case

Parameters

<i>M</i>	mesh class
<i>c</i>	current phase velocity
<i>egn</i>	eigenvector
<i>displ</i>	output displacement, shape(nspec*NGLL+NGRL)

3.2.1.8 egn2displ_att()

```
void specswd::SolverLove::egn2displ_att (
    const Mesh & M,
    scmplx c,
    const scmplx * egn,
    scmplx *__restrict displ) const
```

convert eigenvector to displacement, visco-elastic case

Parameters

<i>M</i>	mesh class
<i>c</i>	current phase velocity
<i>egn</i>	eigenvector
<i>displ</i>	output displacement, shape(nspec*NGLL+NGRL)

3.2.1.9 transform_kernels()

```
void specswd::SolverLove::transform_kernels (
    const Mesh & M,
    std::vector< float > & frekl) const
```

transform modulus kernel to velocity kernel, Love wave case

Parameters

<i>M</i>	mesh class
<i>frekl</i>	frechet kernels, the shape depends on: <ul style="list-style-type: none"> • 1: elastic love wave: $N/L/\rho \rightarrow vsh/vsv/\rho$ • 2: anelastic love wave: $N/L/QNi/QLi/\rho \rightarrow vsh/vsv/QNi/QLi/\rho$

The documentation for this class was generated from the following files:

- vti.hpp
- eigenvalues.cpp
- frechet.cpp
- frechet_group.cpp
- group_velocity.cpp
- sem.cpp
- transform.cpp

3.3 specswd::SolverRayl Class Reference

Public Member Functions

- void [prepare_matrices](#) (const [Mesh](#) &M)
preparing M/K/E matrices for Rayleigh wave
- void [compute_egn](#) (const [Mesh](#) &M, std::vector< float > &c, std::vector< float > &ur, std::vector< float > &ul, bool use_qz=false)
compute rayleigh wave dispersion and eigenfunctions, elastic case
- void [compute_egn_att](#) (const [Mesh](#) &M, std::vector< scmplx > &c, std::vector< scmplx > &ur, std::vector< scmplx > &ul, bool use_qz=false)
compute rayleigh wave dispersion and eigenfunctions, visco-elastic case
- float [group_vel](#) (const [Mesh](#) &M, float c, const float *ur, const float *ul) const
compute velocity of love wave, elastic case
- scmplx [group_vel_att](#) (const [Mesh](#) &M, scmplx c, const scmplx *ur, const scmplx *ul) const
compute velocity of love wave, visco-elastic case
- void [compute_phase_kl](#) (const [Mesh](#) &M, float c, const float *ur, const float *ul, std::vector< float > &frekl) const
compute Rayleigh wave phase kernels, elastic case
- void [compute_phase_kl_att](#) (const [Mesh](#) &M, scmplx c, const scmplx *ur, const scmplx *ul, std::vector< float > &frekl_c, std::vector< float > &frekl_q) const
compute Rayleigh wave phase kernels, visco-elastic case
- void [compute_group_kl](#) (const [Mesh](#) &M, float c, const float *ur, const float *ul, std::vector< float > &frekl) const
compute group velocity and kernels for rayleigh wave group velocity, elastic case
- void [compute_group_kl_att](#) (const [Mesh](#) &M, scmplx c, scmplx u, const scmplx *ur, const scmplx *ul, std::vector< float > &frekl_u, std::vector< float > &frekl_q) const
compute love wave group velocity kernels, visco-elastic case
- void [egn2displ](#) (const [Mesh](#) &M, float c, const float *egn, float *__restrict displ) const
convert right eigenfunction to displacement, elastic case
- void [egn2displ_att](#) (const [Mesh](#) &M, scmplx c, const scmplx *egn, scmplx *__restrict displ) const
convert right eigenfunction to displacement, visco-elastic case
- void [transform_kernels](#) (const [Mesh](#) &M, std::vector< float > &frekl) const
transform modulus kernel to velocity kernel, Rayleigh wave case

3.3.1 Member Function Documentation

3.3.1.1 compute_egn()

```
void specsud::SolverRayl::compute_egn (
    const Mesh & mesh,
    std::vector< float > & c,
    std::vector< float > & ur,
    std::vector< float > & ul,
    bool use_qz = false)
```

compute rayleigh wave dispersion and eigenfunctions, elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	dispersion, shape(nc) $c = c_0(1 + iQL^{-1})$
<i>ur/ul</i>	left/right eigenvectors, shape(nc,nglob_el*2+nglob_ac)
<i>use_qz</i>	if true, save QZ matrix

3.3.1.2 compute_egn_att()

```
void specsud::SolverRayl::compute_egn_att (
    const Mesh & mesh,
    std::vector< scmplx > & c,
    std::vector< scmplx > & ur,
    std::vector< scmplx > & ul,
    bool use_qz = false)
```

compute rayleigh wave dispersion and eigenfunctions, visco-elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	dispersion, shape(nc) $c = c_0(1 + iQL^{-1})$
<i>ur/ul</i>	left/right eigenvectors, shape(nc,nglob_el*2+nglob_ac)
<i>use_qz</i>	if true, save QZ matrix

3.3.1.3 compute_group_kl()

```
void specsud::SolverRayl::compute_group_kl (
    const Mesh & mesh,
    float c,
    const float * ur,
    const float * ul,
    std::vector< float > & frekl) const
```

compute group velocity and kernels for rayleigh wave group velocity, elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	current phase velocity
<i>ur/ul</i>	right/left eigen function, shape(nglob_el*2+nglob_ac)
<i>frekl</i>	Frechet kernels (A/C/L/kappa/rho) for elastic parameters, shape(5,nspec*NGLL + NGRL)

3.3.1.4 compute_group_kl_att()

```
void specswd::SolverRayl::compute_group_kl_att (
    const Mesh & mesh,
    scmplx c,
    scmplx u,
    const scmplx * ur,
    const scmplx * ul,
    std::vector< float > & frekl_u,
    std::vector< float > & frekl_q) const
```

compute love wave group velocity kernels, visco-elastic case

Parameters

<i>mesh</i>	Mesh class
<i>c</i>	current complex phase velocity
<i>u</i>	current complex group velocity
<i>ur/ul</i>	right/left eigen function, shape(nglob_el*2+nglob_ac)
<i>frekl_u</i>	dRe(u)/d(N/L/QN/QL/rho) shape(5,nspec*NGLL + NGRL)
<i>frekl_q</i>	d(qi)/d(N/L/QN/QL/rho) shape(5,nspec*NGLL + NGRL)

3.3.1.5 compute_phase_kl()

```
void specswd::SolverRayl::compute_phase_kl (
    const Mesh & M,
    float c,
    const float * ur,
    const float * ul,
    std::vector< float > & frekl) const
```

compute Rayleigh wave phase kernels, elastic case

Parameters

<i>M</i>	mesh class
<i>c</i>	current phase velocity
<i>ur/ul</i>	right/left eigen function, shape(nglob_el*2+nglob_ac)
<i>frekl</i>	Frechet kernels A/C/L/eta/kappa/rho_kl kernels for elastic parameters, shape(6,nspec*NGLL + NGRL)

3.3.1.6 compute_phase_kl_att()

```
void specs wd::SolverRayl::compute_phase_kl_att (
    const Mesh & M,
    scmplx c,
    const scmplx * ur,
    const scmplx * ul,
    std::vector< float > & frekl_c,
    std::vector< float > & frekl_q) const
```

compute Rayleigh wave phase kernels, visco-elastic case

Parameters

<i>M</i>	mesh class
<i>c</i>	current phase velocity
<i>ur/ul</i>	right/left eigen function, shape(nglob_el*2+nglob_ac)
<i>frekl_c</i>	dRe(c)/d(A/C/L/eta/Qa/Qc/ql/kappa/Qk/rho) kernels for elastic parameters, shape(10,nspec*NGLL + NGRL)
<i>frekl_q</i>	dRe(Q_R)/d(A/C/L/eta/Qa/Qc/ql/kappa/Qk/rho) kernels for elastic parameters, shape(10,nspec*NGLL + NGRL)

3.3.1.7 egn2displ()

```
void specs wd::SolverRayl::egn2displ (
    const Mesh & M,
    float c,
    const float * egn,
    float *__restrict displ) const
```

convert right eigenfunction to displacement, elastic case

Parameters

<i>M</i>	Mesh class
<i>c</i>	current phase velocity
<i>egn</i>	eigenfunction, shape(nglob_el*2+nglob_ac)
<i>displ</i>	displacement, shape(2,npts)

3.3.1.8 egn2displ_att()

```
void specs wd::SolverRayl::egn2displ_att (
    const Mesh & M,
    scmplx c,
    const scmplx * egn,
    scmplx *__restrict displ) const
```

convert right eigenfunction to displacement, visco-elastic case

Parameters

<i>M</i>	Mesh class
<i>c</i>	current phase velocity
<i>egn</i>	eigenfunction, shape($\text{nglob_el} \times 2 + \text{nglob_ac}$)
<i>displ</i>	displacement, shape(2, npts)

3.3.1.9 prepare_matrices()

```
void specswd::SolverRayl::prepare_matrices (
    const Mesh & M)
```

preparing M/K/E matrices for Rayleigh wave

Parameters

<i>M</i>	Mesh class
----------	----------------------------

3.3.1.10 transform_kernels()

```
void specswd::SolverRayl::transform_kernels (
    const Mesh & M,
    std::vector< float > & frekl) const
```

transform modulus kernel to velocity kernel, Rayleigh wave case

Parameters

<i>M</i>	mesh class
<i>frekl</i>	frechet kernels, the shape depends on: <ul style="list-style-type: none"> • 1: elastic rayleigh wave: $A/C/L/\eta/\kappa/\rho \rightarrow vph/vpv/vsv/\eta/vp/\rho$ • 2 anelastic rayleigh wave: $A/C/L/\eta/QAi/QCi/QLi/\kappa/Qki/\rho \rightarrow vph/vpv/vsv/\eta/QAi/QCi/QLi/vp/Qki/\rho$

The documentation for this class was generated from the following files:

- vti.hpp
- eigenvalues.cpp
- frechet.cpp
- frechet_group.cpp
- group_velocity.cpp
- sem.cpp
- transform.cpp

Chapter 4

File Documentation

4.1 global.hpp

```
00001 #ifndef SPECSWD_LIB_GLOB_H_
00002 #define SPECSWD_LIB_GLOB_H_
00003
00004 #include "mesh/mesh.hpp"
00005 #include "vti/vti.hpp"
00006
00007 #include <memory>
00008 #include <complex>
00009
00010 #define CNAME(a) extern "C" void a
00011
00012 // global vars for solver/mesh
00013
00014 namespace specswd_pylib
00015 {
00016
00017 extern std::unique_ptr<specswd::Mesh> M_;
00018 extern std::unique_ptr<specswd::SolverLove> LoveSol_;
00019 extern std::unique_ptr<specswd::SolverRayl> RaylSol_;
00020
00021 // global vars for eigenvalues/eigenvectors
00022 extern std::vector<float> egnr_, egnl_, c_, u_;
00023 extern std::vector<specswd::scmplx> cegnr_, cegnl_, cc_, cu_;
00024
00025 } // namespace specswd_pylib
00026
00027 #endif
```

4.2 specswd.hpp

```
00001 #ifndef SPECSWD_LIB_UTILS_H_
00002 #define SPECSWD_LIB_UTILS_H_
00003
00004 #ifdef __cplusplus
00005 extern "C" {
00006 #endif /* __cplusplus */
00007
00008 void
00009 specswd_init_GQTable();
00010
00011 void
00012 specswd_init_mesh_love(
00013     int nz, const float *z, const float *rho, const float *vsh,
00014     const float *vsv, const float *QN, const float *QL,
00015     bool HAS_ATT, bool print_tomo_info
00016 );
00017
00018 void
00019 specswd_init_mesh_rayl(
00020     int nz, const float *z, const float *rho,
00021     const float *vph, const float *vpv, const float *vsv,
00022     const float *QA, const float *QC, const float *QL,
00023     bool HAS_ATT, bool print_tomo_info
00024 );
```

```

00024 );
00025
00026 void
00027 specswd_egn_love(float freq,bool use_qz);
00028
00029 void
00030 specswd_egn_rayl(float freq,bool use_qz);
00031
00032 void
00033 specswd_group_love();
00034
00035 void
00036 specswd_group_rayl();
00037
00038
00039 #ifdef __cplusplus
00040 }
00041 #endif /* __cplusplus */
00042
00043 #endif

```

4.3 mesh.hpp

```

00001 #ifndef SPECSWD_MESH_H_
00002 #define SPECSWD_MESH_H_
00003
00004 #include <complex>
00005 #include <vector>
00006 #include <array>
00007
00008 namespace specswd
00009 {
00010
00015 struct Mesh {
00016
00017     // SEM Mesh
00018     int nspec,nspec_grl; // no. of elements for gll/grl layer
00019     int nglob; // no. of unique points
00020     std::vector<int> ibool; // connectivity matrix, shape(nspec * NGLL + NGRL)
00021     std::vector<float> skel; // skeleton, shape(nspec * 2 + 2)
00022     std::vector<float> znodes; // shape(nspec * NGLL + NGRL)
00023     std::vector<float> jaco; // jacobian for GLL, shape(nspec + 1) dz / dxi
00024     std::vector<float> zstore; // shape(nglob)
00025
00026     // element type for each medium
00027     int nspec_ac,nspec_el;
00028     int nspec_ac_grl,nspec_el_grl;
00029     std::vector<char> is_elastic, is_acoustic;
00030     std::vector<int> el_elmnts,ac_elmnts; // elements for each media, shape(nspec_? + nspec_?_grl)
00031
00032     // unique array for acoustic/elastic
00033     int nglob_ac, nglob_el;
00034     std::vector<int> ibool_el, ibool_ac; // connectivity matrix, shape shape(nspec_? + nspec_?_grl)
00035
00036     // density and elastic parameters
00037     std::vector<float> xrho_ac; // shape(nspec_ac * NGLL + nspec_ac_grl * NGRL)
00038     std::vector<float> xrho_el; // shape (nsepc_el * NGLL + nspec_el_grl * NGRL)
00039
00040     // attenuation/type flag
00041     bool HAS_ATT;
00042     int SWD_TYPE; // =0 Love wave, = 1 for Rayleigh = 2 full aniso
00043
00044     // vti media
00045     std::vector<float> xA,xC,xL,xeta,xN; // shape(nspec_el * NGLL+ nspec_el_grl * NGRL)
00046     std::vector<float> xQA,xQC,xQL,xQN; // shape(nspec_el * NGLL+ nspec_el_grl * NGRL), Q model
00047
00048     // full anisotropy
00049     int nQmodel_ani; // no. of Q used for anisotropy
00050     std::vector<float> xC2l; // shape(2l,nspec_el * NGLL+ nspec_el_grl * NGRL)
00051     std::vector<float> xQani; // shape(nQmodel_ani,nspec_el * NGLL+ nspec_el_grl * NGRL)
00052
00053     // fluid vti
00054     std::vector<float> xkappa_ac,xQk_ac;
00055
00056     // fluid-elastic boundary
00057     int nfaces_bdry;
00058     std::vector<int> ispec_bdry; // shape(nfaces_bdry,2) (i,:) = [ispec_ac,ispec_el]
00059     std::vector<char> bdry_norm_direct; // shape(nfaces_bdry), = 1 point from acoustic -> z direc
00060     elastic
00061     int nz_tomo, nregions;
00062     std::vector<float> rho_tomo;
00063     std::vector<float> vpv_tomo,vph_tomo,vsv_tomo,vsh_tomo,eta_tomo;

```



```

00064     std::vector<float> QC_tomo,QA_tomo,QL_tomo,QN_tomo;
00065     std::vector<float> c2l_tomo,Qani_tomo;
00066     std::vector<float> depth_tomo;
00067     std::vector<int> region_bdry; // shape(nregions,2)
00068     std::vector<int> iregion_flag; // shape(nspec + 1), return region flag
00069
00070     // interface with layered model
00071     std::vector<char> is_el_reg, is_ac_reg; // shape(nregions)
00072
00073     float PHASE_VELOC_MIN,PHASE_VELOC_MAX;
00074
00075     // current frequency
00076     float freq;
00077
00078     // public functions
00079     void read_model(const char *filename);
00080     void create_database(float freq,float phi);
00081     void print_model() const;
00082     void print_database() const;
00083     void allocate_1D_model(int nz0,int swd_type,int has_att);
00084     void create_model_attributes();
00085
00086     // interpolate model
00087     void interp_model(const float *param,const std::vector<int> &elmnts,std::vector<float> &md) const;
00088     void project_kl(const float *frekl, float *kl_out) const;
00089
00090     // private functions below
00091     // =====
00092     //
00093     void create_material_info_();
00094
00095     // 1-D model
00096     void read_model_header_(const char *filename);
00097     void read_model_love_(const char *filename);
00098     void read_model_rayl_(const char *filename);
00099     void read_model_full_aniso_(const char *filename);
00100
00101     // create SEM database
00102     void compute_minmax_veloc_(float phi,std::vector<float> &vmin,std::vector<float> &vmax);
00103     void create_db_love_();
00104     void create_db_rayl_();
00105     void create_db_aniso_();
00106 };
00107
00108 } // namespace specswd
00109
00110
00111
00112
00113 #endif

```

4.4 attenuation.hpp

```

00001
00002 #ifndef SPECSWD_ATT_TABLE_H_
00003 #define SPECSWD_ATT_TABLE_H_
00004
00005 #include <complex>
00006
00007 namespace specswd
00008 {
00009
00010     const int NSLS = 5;
00011
00012     std::complex<float> get_sls_modulus_factor(float freq,float Q);
00013     void
00014     get_sls_Q_derivative(float freq,float Qm,std::complex<float> &s,
00015                         std::complex<float> &dstdq);
00016
00017     void get_C2l_att(float freq,const float *Qm,int nQmodel,
00018                    std::complex<float> * c2l,
00019                    int funcid=1);
00020
00021
00022 }
00023
00024 #endif

```

4.5 GQTable.hpp

```

00001 #ifndef SPECSWD_GQTABLE_H_
00002 #define SPECSWD_GQTABLE_H_
00003
00004 #include <array>
00005
00006 namespace GQTable
00007 {
00008
00009     const int NGLL = 7, NGRL = 20;
00010     extern std::array<float,NGLL> xgl1,wgl1;
00011     extern std::array<float,NGRL> xgr1,wgr1;
00012     extern std::array<float,NGLL*NGLL> hprimeT,hprime; // hprimeT(i,j) = l'_i(xi_j)
00013     extern std::array<float,NGRL*NGRL> hprimeT_grl,hprime_grl;
00014
00015     void initialize();
00016
00017 } // GQTable
00018
00019
00020 #endif

```

4.6 iofunc.hpp

```

00001 #ifndef SPECSWD_IOFUNC_H_
00002 #define SPECSWD_IOFUNC_H_
00003
00004 #include <iostream>
00005
00006 namespace specswwd
00007 {
00008
00009
00010     inline void __myfwrite(const void *__ptr, size_t __size, size_t __nitems, FILE *__stream)
00011     {
00012         size_t size = fwrite(__ptr,__size,__nitems,__stream);
00013         if(size != __nitems) {
00014             printf("cannot write to binary!\n");
00015             exit(1);
00016         }
00017     }
00018
00019
00020     template<typename T>
00021     void
00022     write_binary_f(FILE *fp, const T *data, size_t n)
00023     {
00024         // write integers of the size
00025         int size = (int)(n * sizeof(T));
00026
00027         // integer front
00028         __myfwrite(&size,sizeof(int),1,fp);
00029
00030         // data
00031         __myfwrite(data,sizeof(T),n,fp);
00032
00033         // integer back
00034         __myfwrite(&size,sizeof(int),1,fp);
00035     }
00036
00037 } // namespace specswwd
00038
00039
00040 #endif

```

4.7 schur.hpp

```

00001 #ifndef SPECSWD_SCHUR_H_
00002 #define SPECSWD_SCHUR_H_
00003
00004 #include <Eigen/Core>
00005 #include <Eigen/Eigenvalues>
00006
00007 #ifdef SPECSWD_EGN_DOUBLE
00008     typedef double realw;
00009     #define LAPACKE_REAL(name) LAPACKE_d ## name

```

```

00010 #define LAPACKE_CMPLX(name) LAPACKE_z ## name
00011 #define LCREALW lapack_complex_double
00012
00013 #else
00014 typedef float realw;
00015 #define LAPACKE_REAL(name) LAPACKE_s ## name
00016 #define LAPACKE_CMPLX(name) LAPACKE_c ## name
00017 #define LCREALW lapack_complex_float
00018 #endif
00019
00020 typedef std::complex<realw> crealw;
00021
00022
00023 namespace specsrd {
00024
00031 template<typename COMMTP=double,typename SAVETP=float> void
00032 schur_qz(int ng,
00033         Eigen::MatrixX<COMMTP> &A,
00034         Eigen::MatrixX<COMMTP> &B,
00035         std::vector<SAVETP> &Qmat, std::vector<SAVETP> &Zmat,
00036         std::vector<SAVETP> &Smat, std::vector<SAVETP> &Spmat
00037 )
00038 {
00039     static_assert(std::is_same_v<SAVETP, float> ||
00040                 std::is_same_v<SAVETP, scmplx>);
00041
00042     // resize all matrices
00043     Qmat.resize(ng*ng); Zmat.resize(ng*ng);
00044     Smat.resize(ng*ng); Spmat.resize(ng*ng);
00045
00046     // run QZ
00047     int sdim = 0;
00048     if constexpr (std::is_same_v<SAVETP, float>) {
00049         Eigen::VectorX<COMMTP> alphas(ng), alphas_i(ng), betas(ng);
00050         if constexpr (std::is_same_v<realw, float>) {
00051             LAPACKE_REAL(gges) (
00052                 LAPACK_COL_MAJOR, 'V', 'V', 'N', nullptr,
00053                 ng, A.data(), ng, B.data(), ng, &sdim, alphas.data(),
00054                 alphas_i.data(), betas.data(), Qmat.data(), ng,
00055                 Zmat.data(), ng
00056             );
00057             memcpy(Smat.data(), A.data(), A.size()*sizeof(A(0,0)));
00058             memcpy(Spmat.data(), B.data(), B.size()*sizeof(B(0,0)));
00059         }
00060         else {
00061             Eigen::MatrixX<COMMTP> Q1(ng,ng), Z1(ng,ng);
00062             LAPACKE_REAL(gges) (
00063                 LAPACK_COL_MAJOR, 'V', 'V', 'N', nullptr,
00064                 ng, A.data(), ng, B.data(), ng, &sdim, alphas.data(),
00065                 alphas_i.data(), betas.data(), Q1.data(), ng,
00066                 Z1.data(), ng
00067             );
00068
00069             // copy to Smat/Spmat
00070             for(int j = 0; j < ng; j++) {
00071                 for(int i = 0; i < ng; i++) {
00072                     int idx = j * ng + i;
00073                     Smat[idx] = A(i,j);
00074                     Spmat[idx] = B(i,j);
00075                     Qmat[idx] = Q1(i,j);
00076                     Zmat[idx] = Z1(i,j);
00077                 }
00078             }
00079         }
00080         else { // save type is scmplx
00081             Eigen::VectorX<COMMTP> alpha(ng), beta(ng);
00082
00083             if constexpr (std::is_same_v<realw, float>) {
00084                 LAPACKE_CMPLX(gges) (
00085                     LAPACK_COL_MAJOR, 'V', 'V', 'N', nullptr,
00086                     ng, (LCREALW*)A.data(), ng, (LCREALW*)B.data(),
00087                     ng, &sdim, (LCREALW*)alpha.data(), (LCREALW*)beta.data(),
00088                     (LCREALW*)Qmat.data(), ng,
00089                     (LCREALW*)Zmat.data(), ng
00090                 );
00091                 memcpy(Smat.data(), A.data(), A.size()*sizeof(A(0,0)));
00092                 memcpy(Spmat.data(), B.data(), B.size()*sizeof(B(0,0)));
00093             }
00094             else {
00095                 Eigen::MatrixX<COMMTP> Q1(ng,ng), Z1(ng,ng);
00096                 LAPACKE_CMPLX(gges) (
00097                     LAPACK_COL_MAJOR, 'V', 'V', 'N', nullptr,
00098                     ng, (LCREALW*)A.data(), ng, (LCREALW*)B.data(),
00099                     ng, &sdim, (LCREALW*)alpha.data(), (LCREALW*)beta.data(),
00100                     (LCREALW*)Q1.data(), ng,
00101                     (LCREALW*)Z1.data(), ng
00102                 );

```

```

00103
00104         // copy to Smat/Spmat
00105         for(int j = 0; j < ng; j++) {
00106             for(int i = 0; i < ng; i++) {
00107                 int idx = j * ng + i;
00108                 Smat[idx] = A(i,j);
00109                 Spmat[idx] = B(i,j);
00110                 Qmat[idx] = Q1(i,j);
00111                 Zmat[idx] = Z1(i,j);
00112             }
00113         }
00114     }
00115 }
00116
00117 }
00118
00119 #endif

```

4.8 frechet_op.hpp

```

00001 #ifndef SPECSWD_FRECHET_OP_H_
00002 #define SPECSWD_FRECHET_OP_H_
00003
00008
00009 #include "shared/attenuation.hpp"
00010 #include "shared/GQTable.hpp"
00011
00012 #define GET_REAL(dc_dm,loc) frekl_r[loc*size+id1] = dc_dm.real(); \
00013                             frekl_i[loc*size+id1] = dc_dm.imag();
00014
00015 namespace specswwd
00016 {
00017
00024 void inline
00025 get_fQ_kl(int npts,std::complex<float> f_cmplx,
00026           const float *frekl_r,
00027           float *__restrict frekl_i)
00028 {
00029     float f_real = f_cmplx.real();
00030     float f_imag = f_cmplx.imag();
00031     float fQi = 2. * f_imag / f_real;
00032
00033     for(int ipt = 0; ipt < npts; ipt++) {
00034         float dQidm = (frekl_i[ipt] * 2. - fQi * frekl_r[ipt]) / f_real;
00035         frekl_i[ipt] = dQidm;
00036     }
00037 }
00038
00054 template<typename T = float > void
00055 love_op_matrix(float freq,T c_M, T c_K,T c_E, const T *y,const T *x,
00056               int nspec_el,int nglob_el, const int *ibool_el,
00057               const float *jaco, const float *xN,
00058               const float *xL,const float *xQN,
00059               const float *xQL,float * __restrict frekl_r,
00060               float * __restrict frekl_i)
00061 {
00062     // check template type
00063     static_assert(std::is_same_v<float,T> || std::is_same_v<std::complex<float>,T>);
00064
00065     using namespace GQTable;
00066     std::array<T,NGRL> rW,lW;
00067     size_t size = nspec_el*NGLL + NGRL;
00068     for(int ispec = 0; ispec < nspec_el + 1; ispec++) {
00069         int id = ispec * NGLL;
00070         float J = jaco[ispec]; // jacobians in this layers
00071
00072         const bool is_gll = (ispec != nspec_el);
00073         const float *w = is_gll? wgl.data(): wgrl.data();
00074         const float *hp = is_gll? hprime.data(): hprime_grl.data();
00075         const int NGL = is_gll? NGLL : NGRL;
00076
00077         // cache displ in a element
00078         for(int i = 0; i < NGL; i++) {
00079             int iglob = ibool_el[id+i];
00080             rW[i] = x[iglob];
00081             lW[i] = y[iglob];
00082             if constexpr (std::is_same_v<T,std::complex<float>) {
00083                 lW[i] = std::conj(lW[i]);
00084             }
00085         }
00086
00087         // compute kernels
00088         T dc_drho{}, dc_dN{}, dc_dL{};

```

```

00089     T dc_dqni{}, dc_dqli{};
00090     T sn = 1., sl = 1.;
00091     T dsdqni{}, dsdqqli{};
00092     for(int m = 0; m < NGL; m++) {
00093         dc_drho = w[m] * J * rW[m] * lW[m] * c_M;
00094
00095         // get sls derivative if required
00096         if constexpr (std::is_same_v<T, std::complex<float>>) {
00097             get_sls_Q_derivative(freq, xQN[id+m], sn, dsdqni);
00098             get_sls_Q_derivative(freq, xQL[id+m], sl, dsdqqli);
00099             dsdqni *= xN[id+m];
00100             dsdqqli *= xL[id+m];
00101         }
00102
00103         // N kernel
00104         T temp = rW[m] * lW[m] * J * w[m] * c_K;
00105         dc_dN = temp * sn;
00106         dc_dqni = temp * dsdqni;
00107
00108         // L kernel
00109         T sx{}, sy{};
00110         for(int i = 0; i < NGL; i++) {
00111             sx += hp[m*NGL+i] * rW[i];
00112             sy += hp[m*NGL+i] * lW[i];
00113         }
00114         temp = sx * sy * w[m] / J * c_E;
00115         dc_dL = temp * sl;
00116         dc_dqli = temp * dsdqqli;
00117
00118         // copy to frekl
00119         int idl = id + m;
00120         if constexpr (std::is_same_v<T, std::complex<float>>) {
00121             GET_REAL(dc_dN, 0); GET_REAL(dc_dL, 1);
00122             GET_REAL(dc_dqni, 2); GET_REAL(dc_dqli, 3);
00123             GET_REAL(dc_drho, 4);
00124         }
00125         else {
00126             frekl_r[0*size+idl] = dc_dN;
00127             frekl_r[1*size+idl] = dc_dL;
00128             frekl_r[2*size+idl] = dc_drho;
00129         }
00130     }
00131 }
00132 }
00133
00165 template<typename T = float >
00166 void
00167 rayl_op_matrix(float freq, T c_M, T c_K, T c_E, const T *y, const T *x,
00168               int nspec_el, int nspec_ac, int nspec_el_grl, int nspec_ac_grl, int nglob_el,
00169               int nglob_ac, const int *el_elmnts, const int *ac_elmnts,
00170               const int *ibool_el, const int *ibool_ac,
00171               const float *jaco, const float *xrho_el, const float *xrho_ac,
00172               const float *xA, const float *xC, const float *xL, const float *xeta,
00173               const float *xQA, const float *xQC, const float *xQL,
00174               const float *xkappa_ac, const float *xQk_ac,
00175               float *__restrict frekl_r,
00176               float *__restrict frekl_i)
00177 {
00178     // check template type
00179     static_assert(std::is_same_v<float, T> || std::is_same_v<std::complex<float>, T>);
00180
00181     // constants
00182     using namespace GQTable;
00183     size_t size = nspec_el * NGLL + nspec_el_grl * NGRL +
00184                 nspec_ac * NGLL + nspec_ac_grl * NGRL;
00185
00186     // loop elastic elements
00187     std::array<T, NGRL> U, V, lU, lV; //left/right eigenvectors in on element
00188     for(int ispec = 0; ispec < nspec_el + nspec_el_grl; ispec++) {
00189         int iel = el_elmnts[ispec];
00190         int id = ispec * NGLL;
00191
00192         // jacobian
00193         float J = jaco[iel];
00194
00195         // get const arrays
00196         const bool is_gll = (ispec != nspec_el);
00197         const float *weight = is_gll? wgll.data(): wgrl.data();
00198         const float *hp = is_gll? hprime.data(): hprime_grl.data();
00199         const int NGL = is_gll? NGLL : NGRL;
00200
00201         // cache U, V and lU, lV
00202         for(int i = 0; i < NGL; i++) {
00203             int iglob = ibool_el[id + i];
00204             U[i] = x[iglob];
00205             V[i] = x[iglob + nglob_el];
00206             lU[i] = y[iglob];

```

```

00207         lV[i] = y[iglob + nglob_el];
00208         if constexpr (std::is_same_v<T, std::complex<float>)> {
00209             lU[i] = std::conj(lU[i]);
00210             lV[i] = std::conj(lV[i]);
00211         }
00212     }
00213
00214     // compute kernel
00215     T dc_drho{}, dc_dA{}, dc_dC{}, dc_dL{};
00216     T dc_deta{}, dc_dQci{}, dc_dQai{}, dc_dQli{};
00217     const T two = 2.;
00218     for(int m = 0; m < NGL; m++) {
00219         T temp = weight[m] * J * c_M;
00220         dc_drho = temp * (U[m] * lU[m] + V[m] * lV[m]);
00221
00222         // get sls factor if required
00223         T sa = 1., sl = 1., sc = 1.;
00224         T dsdqai{}, dsdqci{}, dsdqli{};
00225         T C = xC[id+m], A = xA[id+m], L = xL[id+m], eta = xeta[m];
00226         if constexpr (std::is_same_v<T, std::complex<float>)> {
00227             get_sls_Q_derivative(freq, xQA[id+m], sa, dsdqai);
00228             get_sls_Q_derivative(freq, xQC[id+m], sc, dsdqci);
00229             get_sls_Q_derivative(freq, xQL[id+m], sl, dsdqli);
00230             dsdqai *= A;
00231             dsdqci *= C;
00232             dsdqli *= L;
00233         }
00234
00235         // K matrix
00236         // dc_dA
00237         temp = weight[m] * J * U[m] * lU[m] * c_K;
00238         dc_dA = temp * sa; dc_dQai = temp * dsdqai;
00239
00240         // dc_dL
00241         temp = weight[m] * J * V[m] * lV[m] * c_K;
00242         dc_dL = temp * sl; dc_dQli = temp * dsdqli;
00243
00244         // Ematrix
00245         T sx{}, sy{}, lsx{}, lsy{};
00246         for(int i = 0; i < NGL; i++) {
00247             sx += hp[m*NGL+i] * U[i];
00248             sy += hp[m*NGL+i] * V[i];
00249             lsx += hp[m*NGL+i] * lU[i];
00250             lsy += hp[m*NGL+i] * lV[i];
00251         }
00252
00253         // E1
00254         temp = weight[m] / J * sx * lsx * c_E;
00255         dc_dL += temp * sl; dc_dQli += temp * dsdqli;
00256
00257         // E3
00258         temp = weight[m] / J * sy * lsy * c_E;
00259         dc_dC = temp * sc; dc_dQci = temp * dsdqci;
00260
00261         // K2, d / dm_k sum_{ij} w_j F_j hpT(i,j) U_j lV_i
00262         // = - \sum_{ij} w_k dF/dm_k hpT(i,k) U_k lV_i = lsy * w_k * U_k * dF/dm_k
00263         temp = weight[m] * U[m] * lsy * c_K;
00264         dc_deta = temp * (A*sa - two*L*sl);
00265         temp *= eta;
00266         dc_dA += temp * sa; dc_dQai += temp * dsdqai;
00267         dc_dL += -temp * two * sl; dc_dQli += -temp * two * dsdqli;
00268
00269         // K2, -d / dm_k sum_{ij} w_i L_i hp(i,j) U_j lV_i
00270         // = - \sum_{ij} w_k dL/dm_k hp(j,k) U_j lV_k = -sx * w_k * lV_k dL/dm_k
00271         temp = -weight[m] * lV[m] * sx * c_K;
00272         dc_dL += -temp * two * sl; dc_dQli += -temp * two * dsdqli;
00273
00274         //E2 \sum_{ij} w_k dF/dm_k hp(j,k) V_j lU_k = -sx * w_k * lV_k dF/dm_k
00275         temp = weight[m] * lU[m] * sy * c_E;
00276         dc_deta = temp * (A*sa - two*L*sl);
00277         temp *= eta;
00278         dc_dA += temp * sa; dc_dQai += temp * dsdqai;
00279         dc_dL += -temp * two * sl; dc_dQli += -temp * two * dsdqli;
00280
00281         // E2 -lsx * w_k * V_k * dL/dm_k
00282         temp = -weight[m] * V[m] * lsx * c_E;
00283         dc_dL += -temp * two * sl; dc_dQli += -temp * two * dsdqli;
00284
00285         // copy them to frekl
00286         int idl = iel * NGLL + m;
00287         if constexpr (std::is_same_v<T, float>)> {
00288             frekl_r[0*size+idl] = dc_dA;
00289             frekl_r[1*size+idl] = dc_dC;
00290             frekl_r[2*size+idl] = dc_dL;
00291             frekl_r[3*size+idl] = dc_deta;
00292             frekl_r[5*size+idl] = dc_drho;
00293         }

```

```

00294         else {
00295             GET_REAL(dc_dA,0);
00296             GET_REAL(dc_dC,1);
00297             GET_REAL(dc_dL,2);
00298             GET_REAL(dc_deta,3);
00299             GET_REAL(dc_dQai,4);
00300             GET_REAL(dc_dQci,5);
00301             GET_REAL(dc_dQli,6);
00302             GET_REAL(dc_drho,9);
00303         }
00304     }
00305 }
00306
00307 // acoustic eleemnts
00308 std::array<T,NGRL> chi,lchi;
00309 for(int ispec = 0; ispec < nspec_ac + nspec_ac_grl; ispec++) {
00310     int iel = ac_elmnts[ispec];
00311     int id = ispec * NGLL;
00312
00313     // const arrays
00314     const bool is_gll = (ispec != nspec_ac);
00315     const float *weight = is_gll? wgll.data(): wgrl.data();
00316     const float *hp = is_gll? hprime.data(): hprime_grl.data();
00317     const int NGL = is_gll? NGLL : NGRL;
00318
00319     // jacobians
00320     float J = jaco[iel];
00321
00322     // cache chi and lchi in one element
00323     for(int i = 0; i < NGL; i++) {
00324         int iglob = ibool_ac[id + i];
00325         chi[i] = (iglob == -1) ? 0: x[iglob+nglob_el*2];
00326         lchi[i] = (iglob == -1) ? 0.: y[iglob+nglob_el*2];
00327         if constexpr (std::is_same_v<T,std::complex<float>){
00328             lchi[i] = std::conj(lchi[i]);
00329         }
00330     }
00331
00332     // derivatives
00333     T dc_dkappa{},dc_drho{}, dc_dqki{};
00334     T sk = 1., dskdqi = 0.;
00335     for(int m = 0; m < NGL; m++){
00336         // copy material
00337         float rho = xrho_ac[id+m];
00338         float kappa = xkappa_ac[id+m];
00339         if constexpr (std::is_same_v<T,std::complex<float>){
00340             get_sls_Q_derivative(freq,xQk_ac[id+m],sk,dskdqi);
00341             dskdqi *= kappa;
00342         }
00343
00344         // kappa kernel
00345         T temp = -c_M * weight[m] * J*chi[m] * lchi[m] / (sk * kappa) / (sk * kappa);
00346         dc_dkappa = temp * sk;
00347         dc_dqki = temp * dskdqi;
00348
00349         dc_drho = -c_K * weight[m] * J * chi[m] * lchi[m] / rho / rho;
00350
00351         T sx{},sy{};
00352         for(int i = 0; i < NGL; i++) {
00353             sx += hp[m*NGL+i] * chi[i];
00354             sy += hp[m*NGL+i] * lchi[i];
00355         }
00356         dc_drho += -c_E * weight[m] / J / (rho*rho) * sx * sy;
00357
00358         // copy to frekl
00359         int idl = iel * NGLL + m;
00360         if constexpr (std::is_same_v<T,float>){
00361             frekl_r[4*size+idl] = dc_dkappa;
00362             frekl_r[5*size+idl] = dc_drho;
00363         }
00364         else {
00365             GET_REAL(dc_dkappa,7);
00366             GET_REAL(dc_dqki,8);
00367             GET_REAL(dc_drho,9);
00368         }
00369     }
00370 }
00371 }
00372
00373 // /**
00374 //  * @brief compute coef * y^dag @ d( (w^2 M -E) - k^2 K)/dm_i @ x dm_i
00375 //  * @param freq current frequency
00376 //  * @param c current phase velocity
00377 //  * @param coef derivative scaling coeffs
00378 //  * @param y/x dot vector,shape(nglob_el*2+nglob_ac)
00379 //  * @param nspec_el/nglob_el mesh nelemnts/unique points for elastic
00380 //  * @param nspec_ac/nglob_ac mesh nelemnts/unique points for acoustic

```

```

00381 // * @param nspec_el/ac_grl no. of GRL elements
00382 // * @param ibool_el elastic connectivity matrix, shape(nspec_el*NGLL+nspec_el_grl*NGRL)
00383 // * @param ibool_ac elastic connectivity matrix, shape(nspec_ac*NGLL+nspec_ac_grl*NGRL)
00384 // * @param jaco jacobian matrix, shape (nspec_el + 1)
00385 // * @param xA/xC/xL/xeta/xQA/xQC/xQL/xrho elastic model parameters,ibool_el.shape
00386 // * @param xkappa_ac/xQk_ac/xrho_ac acoustic model parameters, ibool_ac.shape
00387 // * @param frekl_r dc/d(A/C/L/kappa/rho) (elastic) or dc/d(A/C/L/QA/QC/QL/kappa/Qk/rho) (anelstic)
00388 // * @param frekl_i nullptr or dqc/d(A/C/L/QA/QC/QL/kappa/Qk/rho) (anelstic)
00389 // * @note frekl_r and frekl_i should be set to 0 before calling this routine
00390 // */
00391 // template<typename T = float >
00392 // void
00393 // rayl_deriv_op(float freq,T c,T coef,const T *y, const T *x,
00394 //               int nspec_el,int nspec_ac,int nspec_el_grl,int nspec_ac_grl,int nglob_el,
00395 //               int nglob_ac, const int *el_elmnts,const int *ac_elmnts,
00396 //               const int* ibool_el, const int* ibool_ac,
00397 //               const float *jaco,const float *xrho_el,const float *xrho_ac,
00398 //               const float *xA, const float *xC,const float *xL,const float *xeta,
00399 //               const float *xQA, const float *xQC,const float *xQL,
00400 //               const float *xkappa_ac, const float *xQk_ac,
00401 //               float *__restrict frekl_r,
00402 //               float *__restrict frekl_i)
00403 // {
00404 //     // check template type
00405 //     static_assert(std::is_same_v<float,T> || std::is_same_v<std::complex<float>,T>);
00406 //
00407 //     // constants
00408 //     using namespace GQTable;
00409 //     size_t size = nspec_el * NGLL + nspec_el_grl * NGRL +
00410 //                  nspec_ac * NGLL + nspec_ac_grl * NGRL;
00411 //     T om = 2 * M_PI * freq;
00412 //     T k2 = std::pow(om / c,2);
00413 //
00414 //     // loop elastic elements
00415 //     std::array<T,NGRL> U,V,lU,lV;
00416 //     for(int ispec = 0; ispec < nspec_el + nspec_el_grl; ispec++) {
00417 //         int iel = el_elmnts[ispec];
00418 //         int id = ispec * NGLL;
00419 //
00420 //         const float *weight = wgl1.data();
00421 //         const float *hp = hprime.data();
00422 //         int NGL = NGLL;
00423 //
00424 //         // jacobian
00425 //         float J = jaco[iel];
00426 //
00427 //         // grl case
00428 //         if(ispec == nspec_el) {
00429 //             weight = wgrl.data();
00430 //             hp = hprime_grl.data();
00431 //             NGL = NGRL;
00432 //         }
00433 //
00434 //         // cache U,V and lU,lV
00435 //         for(int i = 0; i < NGL; i++) {
00436 //             int iglob = ibool_el[id + i];
00437 //             U[i] = x[iglob] * coef;
00438 //             V[i] = x[iglob + nglob_el] * coef;
00439 //             lU[i] = y[iglob];
00440 //             lV[i] = y[iglob + nglob_el];
00441 //             if constexpr (std::is_same_v<T,std::complex<float>)> {
00442 //                 lU[i] = std::conj(lU[i]);
00443 //                 lV[i] = std::conj(lV[i]);
00444 //             }
00445 //         }
00446 //
00447 //         // compute kernel
00448 //         T dc_drho{}, dc_dA{}, dc_dC{}, dc_dL{};
00449 //         T dc_deta{}, dc_dQci{},dc_dQai{},dc_dqli{};
00450 //         const T two = 2.;
00451 //         for(int m = 0; m < NGL; m++) {
00452 //             T temp = weight[m] * J;
00453 //             dc_drho = temp * om * om *
00454 //                 (U[m] * lU[m] + V[m] * lV[m]);
00455 //
00456 //             // get sls factor if required
00457 //             T sa = 1.,sl = 1.,sc = 1.;
00458 //             T dsdqai{},dsdqci{},dsdqli{};
00459 //             float C = xC[id+m], A = xA[id+m],
00460 //                 L = xL[id+m], eta = xeta[m];
00461 //             if constexpr (std::is_same_v<T,std::complex<float>)> {
00462 //                 get_sls_Q_derivative(freq,xQA[id+m],sa,dsdqai);
00463 //                 get_sls_Q_derivative(freq,xQC[id+m],sc,dsdqci);
00464 //                 get_sls_Q_derivative(freq,xQL[id+m],sl,dsdqli);
00465 //                 dsdqai *= A;
00466 //                 dsdqci *= C;
00467 //                 dsdqli *= L;

```



```

00468 //      }
00469
00470 //      // K matrix
00471 //      // dc_dA
00472 //      temp = -weight[m] * J * k2 * U[m] * lU[m];
00473 //      dc_dA = temp * sa; dc_dQai = temp * dsdqai;
00474
00475 //      // dc_dL
00476 //      temp = -weight[m] * J * k2 * V[m] * lV[m];
00477 //      dc_dL = temp * sl; dc_dQli = temp * dsdqli;
00478
00479 //      // Ematrix
00480 //      T sx{},sy{},lsx{},lsy{};
00481 //      for(int i = 0; i < NGL; i++) {
00482 //          sx += hp[m*NGL+i] * U[i];
00483 //          sy += hp[m*NGL+i] * V[i];
00484 //          lsx += hp[m*NGL+i] * lU[i];
00485 //          lsy += hp[m*NGL+i] * lV[i];
00486 //      }
00487 //      temp = -weight[m] / J * sx * lsx;
00488 //      dc_dL += temp * sl; dc_dQli += temp * dsdqli;
00489
00490 //      temp = - weight[m] / J * sy * lsy;
00491 //      dc_dC = temp * sc; dc_dQci = temp * dsdqci;
00492
00493 //      // eta
00494 //      temp = - (k2 * weight[m] * U[m] * lsy +
00495 //          weight[m] * lU[m] * sy);
00496 //      dc_deta = temp * (A*sa - two*L*sl);
00497
00498 //      temp *= eta;
00499 //      dc_dA += temp * sa; dc_dQai += temp * dsdqai;
00500 //      dc_dL += - temp * two * sl; dc_dQli += -temp * two * dsdqli;
00501
00502 //      temp = k2 * weight[m] * lV[m] * sx + weight[m] * V[m] * lsx;
00503 //      temp *= coef;
00504 //      dc_dL += temp * sl;
00505 //      dc_dQli += temp * dsdqli;
00506
00507 //      // copy them to frekl
00508 //      int idl = iel * NGLL + m;
00509 //      if constexpr (std::is_same_v<T,float>) {
00510 //          frekl_r[0*size+idl] = dc_dA;
00511 //          frekl_r[1*size+idl] = dc_dC;
00512 //          frekl_r[2*size+idl] = dc_dL;
00513 //          frekl_r[3*size+idl] = dc_deta;
00514 //          frekl_r[5*size+idl] = dc_drho;
00515 //      }
00516 //      else {
00517 //          GET_REAL(dc_dA,0);
00518 //          GET_REAL(dc_dC,1);
00519 //          GET_REAL(dc_dL,2);
00520 //          GET_REAL(dc_deta,3);
00521 //          GET_REAL(dc_dQai,4);
00522 //          GET_REAL(dc_dQci,5);
00523 //          GET_REAL(dc_dQli,6);
00524 //          GET_REAL(dc_drho,9);
00525 //      }
00526 //      }
00527 //  }
00528
00529 //  // acoustic elemnts
00530 //  std::array<T,NGL> chi,lchi;
00531 //  for(int ispec = 0; ispec < nspec_ac + nspec_ac_grl; ispec++) {
00532 //      int iel = ac_elmnts[ispec];
00533 //      int id = ispec * NGLL;
00534 //      const float *weight = wgrl.data();
00535 //      const float *hp = hprime.data();
00536 //      int NGL = NGLL;
00537
00538 //      // jacobians
00539 //      float J = jaco[iel];
00540
00541 //      // grl case
00542 //      if(ispec == nspec_ac) {
00543 //          weight = wgrl.data();
00544 //          hp = hprime_grl.data();
00545 //          NGL = NGLL;
00546 //      }
00547
00548 //      // cache chi and lchi in one element
00549 //      for(int i = 0; i < NGL; i++) {
00550 //          int iglob = ibool_ac[id + i];
00551 //          chi[i] = (iglob == -1) ? 0: x[iglob+nglob_el*2] * coef;
00552 //          lchi[i] = (iglob == -1) ? 0.: y[iglob+nglob_el*2];
00553 //          if constexpr (std::is_same_v<T,std::complex<float>)>) {
00554 //              lchi[i] = std::conj(lchi[i]);

```

```

00555 //      }
00556 //      }
00557
00558 //      // derivatives
00559 //      T dc_dkappa{},dc_drho{}, dc_dqki{};
00560 //      T sk = 1., dskdqi = 0.;
00561 //      for(int m = 0; m < NGL; m ++ ){
00562 //          // copy material
00563 //          float rho = xrho_ac[id+m];
00564 //          float kappa = xkappa_ac[id+m];
00565 //          if constexpr (std::is_same_v<T,std::complex<float>)> {
00566 //              get_sls_Q_derivative(freq,xQk_ac[id+m],sk,dskdqi);
00567 //              dskdqi *= kappa;
00568 //          }
00569
00570 //          // kappa kernel
00571 //          T temp = std::pow(om/(sk * kappa),2) *weight[m]* J*
00572 //              chi[m] * lchi[m];
00573 //          dc_dkappa = temp * sk;
00574 //          dc_dqki = temp * dskdqi;
00575
00576 //          dc_drho = -k2 * std::pow(om/rho,2) *weight[m]* J*
00577 //              chi[m] * lchi[m];
00578
00579 //          T sx{},sy{};
00580 //          for(int i = 0; i < NGL; i ++){
00581 //              sx += hp[m*NGL+i] * chi[i];
00582 //              sy += hp[m*NGL+i] * lchi[i];
00583 //          }
00584 //          dc_drho += weight[m] / J / (rho*rho) * sx * sy;
00585
00586 //          // copy to frekl
00587 //          int idl = iel * NGLL + m;
00588 //          if constexpr (std::is_same_v<T,float>)> {
00589 //              frekl_r[4*size+idl] = dc_dkappa;
00590 //              frekl_r[5*size+idl] = dc_drho;
00591 //          }
00592 //          else {
00593 //              GET_REAL(dc_dkappa,7);
00594 //              GET_REAL(dc_dqki,8);
00595 //              GET_REAL(dc_drho,9);
00596 //          }
00597 //      }
00598 //  }
00599 // }
00600
00601
00602 } // namespace specswwd
00603
00604 #undef GET_REAL
00605
00606 #endif

```

4.9 vti.hpp

```

00001 #ifndef SPECSWD_SOLVER_H_
00002 #define SPECSWD_SOLVER_H_
00003
00004 #include "mesh/mesh.hpp"
00005
00006 #include <complex>
00007 #include <vector>
00008
00009
00010 namespace specswwd
00011 {
00012
00013 typedef std::complex<float> scmplx;
00014
00015 class SolverLove {
00016 private:
00017     // solver matrices
00018     std::vector<float> Mmat,Emat,Kmat;
00019     std::vector<scmplx> CMmat,CEmat,CKmat;
00020
00021     // QZ matrix all are column major
00022     std::vector<float> Qmat_,Zmat_,Smat_,Spmat_; // column major!
00023     std::vector<scmplx> cQmat_,cZmat_,cSmat_,cSpmat_;
00024
00025 public:
00026
00027     // eigenfunctions/values

```

```

00029 void prepare_matrices(const Mesh &M);
00030 void compute_egn(const Mesh &M,
00031                 std::vector<float> &c,
00032                 std::vector<float> &egn,
00033                 bool use_qz=false);
00034 void compute_egn_att(const Mesh &M,
00035                     std::vector<scmplx> &c,
00036                     std::vector<scmplx> &egn,
00037                     bool use_qz=false);
00038
00039 // group velocity
00040 float group_vel(const Mesh &M, float c, const float *egn) const;
00041 scmplx group_vel_att(const Mesh &M, scmplx c, const scmplx *egn) const;
00042
00043 // phase velocity kernels
00044 void compute_phase_kl(const Mesh &M,
00045                      float c, const float *egn,
00046                      std::vector<float> &frekl) const;
00047 void compute_phase_kl_att(const Mesh &M,
00048                          scmplx c, const scmplx *egn,
00049                          std::vector<float> &frekl_c,
00050                          std::vector<float> &frekl_q) const;
00051
00052 // group kernel
00053 void compute_group_kl(const Mesh &M,
00054                      float c, const float *egn,
00055                      std::vector<float> &frekl) const;
00056
00057 void compute_group_kl_att(const Mesh &M,
00058                          scmplx c, scmplx u, const scmplx *egn,
00059                          std::vector<float> &frekl_u,
00060                          std::vector<float> &frekl_q) const;
00061
00062 // transforms
00063 void egn2displ(const Mesh &M, float c,
00064               const float *egn, float * __restrict displ) const;
00065 void egn2displ_att(const Mesh &M, scmplx c, const scmplx *egn,
00066                   scmplx * __restrict displ) const;
00067 void transform_kernels(const Mesh &M, std::vector<float> &frekl) const;
00068 };
00069
00070 class SolverRayl {
00071 private:
00072     // solver matrices
00073     std::vector<float> Mmat, Emat, Kmat;
00074     std::vector<scmplx> CMmat, CEmat, CKmat;
00075
00076     // QZ matrix all are column major
00077     std::vector<float> Qmat_, Zmat_, Smat_, Spmat_; // column major!
00078     std::vector<scmplx> cQmat_, cZmat_, cSmat_, cSpmat_;
00079 public:
00080     void prepare_matrices(const Mesh &M);
00081     void compute_egn(const Mesh &M,
00082                     std::vector<float> &c,
00083                     std::vector<float> &ur,
00084                     std::vector<float> &ul,
00085                     bool use_qz=false);
00086     void compute_egn_att(const Mesh &M,
00087                         std::vector<scmplx> &c,
00088                         std::vector<scmplx> &ur,
00089                         std::vector<scmplx> &ul,
00090                         bool use_qz=false);
00091
00092     // group velocity
00093     float group_vel(const Mesh &M,
00094                    float c, const float *ur,
00095                    const float *ul) const;
00096     scmplx group_vel_att(const Mesh &M,
00097                        scmplx c, const scmplx *ur,
00098                        const scmplx *ul) const;
00099
00100     // phase velocity kernels
00101     void compute_phase_kl(const Mesh &M,
00102                          float c, const float *ur,
00103                          const float *ul,
00104                          std::vector<float> &frekl) const;
00105     void compute_phase_kl_att(const Mesh &M,
00106                             scmplx c, const scmplx *ur,
00107                             const scmplx *ul,
00108                             std::vector<float> &frekl_c,
00109                             std::vector<float> &frekl_q) const;
00110
00111     // group velocity kernels
00112     void compute_group_kl(const Mesh &M,
00113                          float c, const float *ur,
00114                          const float *ul,
00115                          std::vector<float> &frekl) const;

```

```
00116     void compute_group_kl_att(const Mesh &M,
00117                               scmplx c, scmplx u,
00118                               const scmplx *ur, const scmplx *ul,
00119                               std::vector<float> &frekl_u,
00120                               std::vector<float> &frekl_q) const;
00121
00122
00123     // transforms
00124     void egn2displ(const Mesh &M, float c,
00125                   const float*egn, float * __restrict displ) const;
00126     void egn2displ_att(const Mesh &M, scmplx c, const scmplx *egn,
00127                       scmplx * __restrict displ) const;
00128     void transform_kernels(const Mesh &M, std::vector<float> &frekl) const;
00129 };
00130
00131
00132 } // namespace specs_wd
00133
00134
00135
00136
00137 #endif
```

Index

attenuation.hpp, [21](#)

compute_egn

 specswd::SolverLove, [10](#)

 specswd::SolverRayl, [14](#)

compute_egn_att

 specswd::SolverLove, [10](#)

 specswd::SolverRayl, [14](#)

compute_group_kl

 specswd::SolverLove, [10](#)

 specswd::SolverRayl, [14](#)

compute_group_kl_att

 specswd::SolverLove, [10](#)

 specswd::SolverRayl, [15](#)

compute_phase_kl

 specswd::SolverLove, [11](#)

 specswd::SolverRayl, [15](#)

compute_phase_kl_att

 specswd::SolverLove, [11](#)

 specswd::SolverRayl, [15](#)

create_database

 specswd::Mesh, [7](#)

egn2displ

 specswd::SolverLove, [12](#)

 specswd::SolverRayl, [16](#)

egn2displ_att

 specswd::SolverLove, [12](#)

 specswd::SolverRayl, [16](#)

frechet_op.hpp, [24](#)

global.hpp, [19](#)

GQTable.hpp, [22](#)

interp_model

 specswd::Mesh, [7](#)

iofunc.hpp, [22](#)

mesh.hpp, [20](#)

prepare_matrices

 specswd::SolverRayl, [17](#)

project_kl

 specswd::Mesh, [7](#)

read_model

 specswd::Mesh, [8](#)

read_model_full_aniso_

 specswd::Mesh, [8](#)

read_model_header_

 specswd::Mesh, [8](#)

read_model_love_

 specswd::Mesh, [8](#)

read_model_rayl_

 specswd::Mesh, [9](#)

schur.hpp, [22](#)

specswd.hpp, [19](#)

specswd::Mesh, [5](#)

 create_database, [7](#)

 interp_model, [7](#)

 project_kl, [7](#)

 read_model, [8](#)

 read_model_full_aniso_, [8](#)

 read_model_header_, [8](#)

 read_model_love_, [8](#)

 read_model_rayl_, [9](#)

specswd::SolverLove, [9](#)

 compute_egn, [10](#)

 compute_egn_att, [10](#)

 compute_group_kl, [10](#)

 compute_group_kl_att, [10](#)

 compute_phase_kl, [11](#)

 compute_phase_kl_att, [11](#)

 egn2displ, [12](#)

 egn2displ_att, [12](#)

 transform_kernels, [12](#)

specswd::SolverRayl, [13](#)

 compute_egn, [14](#)

 compute_egn_att, [14](#)

 compute_group_kl, [14](#)

 compute_group_kl_att, [15](#)

 compute_phase_kl, [15](#)

 compute_phase_kl_att, [15](#)

 egn2displ, [16](#)

 egn2displ_att, [16](#)

 prepare_matrices, [17](#)

 transform_kernels, [17](#)

transform_kernels

 specswd::SolverLove, [12](#)

 specswd::SolverRayl, [17](#)

vti.hpp, [30](#)