SpecSWD

Generated by Doxygen 1.9.8

1 Namespace Index 1

1.1 Namespace List 1

2 Class Index 3

2.1 Class List 3

3 File Index 5

3.1 File List 5

4 Namespace Documentation 7

4.1 specswd Namespace Reference 7

4.1.1 Detailed Description 8

4.1.2 Function Documentation 9

4.1.2.1 get_cQ_kl() 9

4.1.2.2 get_sls_modulus_factor() 10

4.1.2.3 get_sls_Q_derivative() 10

4.1.2.4 love_deriv_op_() 10

4.1.2.5 rayl_deriv_op_() 11

4.1.2.6 schur_qz() 12

4.1.2.7 solve_christoffel() 13

5 Class Documentation 15

5.1 specswd::Mesh Struct Reference 15

5.1.1 Member Function Documentation 17

5.1.1.1 create_database() 17

5.1.1.2 create_db_aniso_() 17

5.1.1.3 create_db_love_() 17

5.1.1.4 create_db_rayl_() 18

5.1.1.5 interp_model() 18

5.1.1.6 project_kl() 18

5.1.1.7 read_model() 18

5.1.1.8 read_model_full_aniso_() 19

5.1.1.9 read_model_header_() 19

5.1.1.10 read_model_love_() 19

5.1.1.11 read_model_rayl_() 19

5.2 specswd::SolverLove Class Reference 20

5.2.1 Member Function Documentation 20

5.2.1.1 compute_egn() 20

5.2.1.2 compute_egn_att() 21

5.2.1.3 compute_group_kl() 21

5.2.1.4 compute_group_kl_att() 21

5.2.1.5 compute_phase_kl() 22

5.2.1.6 compute_phase_kl_att() 22

5.2.1.7 transform_kernels() 23

5.3 specswd::SolverRayl Class Reference 23
5.3.1 Member Function Documentation 24
5.3.1.1 compute_egn() 24
5.3.1.2 compute_egn_att() 24
5.3.1.3 compute_phase_kl() 25
5.3.1.4 compute_phase_kl_att() 25
5.3.1.5 transform_kernels() 26

6 File Documentation **27**
6.1 precision.hpp 27
6.2 mesh.hpp 27
6.3 attenuation.hpp 28
6.4 GQTable.hpp 29
6.5 iofunc.hpp 29
6.6 quadrature.hpp 30
6.7 frechet_op.hpp 30
6.8 vti.hpp 34

Index **37**

Chapter 1

Namespace Index

1.1 Namespace List

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

specswd
Derivative operators: 7

Chapter 2

Class Index

2.1 Class List

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

specswd::Mesh 15
specswd::SolverLove 20
specswd::SolverRayl 23

Chapter 3

File Index

3.1 File List

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

precision.hpp 27
mesh.hpp 27
attenuation.hpp 28
GQTable.hpp 29
iofunc.hpp 29
quadrature.hpp 30
frechet_op.hpp 30
vti.hpp 34

Chapter 4

Namespace Documentation

4.1 specswd Namespace Reference

derivative operators:

Classes

- struct Mesh
- class SolverLove
- class SolverRayl

Typedefs

- typedef std::complex< float > **scmplx**

Functions

- void solve_christoffel (float phi, const float ∗c21, float &cmin, float &cmax)

    find the min/max phase velocity by solving Christoffel equations G_{ik} g_k = v^2 g_i
- template<typename T , typename ... Args>
  void **allocate** (int n, T &vec1, Args &...args)
- template<typename COMMTP = double, typename SAVETP = float>
  void schur_qz (int ng, Eigen::MatrixX< COMMTP > &A, Eigen::MatrixX< COMMTP > &B, std::vector< SAVETP > &Qmat_, std::vector< SAVETP > &Zmat_, std::vector< SAVETP > &Smat_, std::vector< SAVETP > &Spmat_)

    QZ decomposition of matrix A and B.
- template<typename T >
  void get_cQ_kl (T &dcdm, T c, float &dcLdm, float &dQiLdm)

    convert d\tilde{c}/dm to dcL/dm, dQiL/dm, where \tilde{c} = c(1 + 1/2 i Qi)
- template<typename T = float>
  void love_deriv_op_ (float freq, T c, T coef, const T ∗y, const T ∗x, int nspec_el, int nglob_el, const int ∗ibool←_el, const float ∗jaco, const float ∗xN, const float ∗xL, const float ∗xQN, const float ∗xQL, float ∗__restrict frekl_c, float ∗__restrict frekl_q)

    compute coef ∗ y^H @ d((w^2 M -E) - k^ K)/dm_i @ x dm_i

- template< typename T = float >
  void rayl_deriv_op_ (float freq, T c, T coef, const T *y, const T *x, int nspec_el, int nspec_ac, int nspec_el_grl, int nspec_ac_grl, int nglob_el, int nglob_ac, const int *el_elmnts, const int *ac_elmnts, const int *ibool_↩el, const int *ibool_ac, const float *jaco, const float *xrho_el, const float *xrho_ac, const float *xA, const float *xC, const float *xL, const float *xeta, const float *xQA, const float *xQC, const float *xQL, const float *xkappa_ac, const float *xQk_ac, float *__restrict frekl_c, float *__restrict frekl_q)

  *compute coef * y^dag @ d((w^2 M -E) - k^2 K)/dm_i @ x dm_i*
- template< typename T >
  T **get_love_group_vel** (int ng, float freq, T c, const T *egn, const float *Mmat, const T *Kmat)
- template< typename T >
  T **get_rayl_group_vel** (int ng, float freq, T c, const T *ur, const T *ul, const T *Mmat, const T *Kmat)
- template< typename T = float >
  void **egn2displ_love_** (int nspec, const int *ibool_el, const T *egn, T *__restrict displ)
- template< typename T = float >
  void **egn2displ_rayl_** (int nspec_el, int nspec_ac, int nspec_el_grl, int nspec_ac_grl, int nglob_el, int nglob↩_ac, const float *jaco, const int *ibool_el, const int *ibool_ac, const int *el_elmnts, const int *ac_elmnts, const float *xrho_ac, const T *egn, float freq, T c, T *__restrict displ)
- crealw get_sls_modulus_factor (float freq, float Q)

  get SLS Q terms on the elastic modulus
- void get_sls_Q_derivative (float freq, float Q, crealw &s, crealw &dsdqi)

  Get the Q factor and derivative for SLS model.
- void **set_C21_att_model** (float freq, const float *Qm, int nQmodel, crealw __restrict *c21, int funcid, bool do_deriv)
- void **get_sls_Q_derivative** (float freq, float Qm, std::complex< float > &s, std::complex< float > &dsdqi)
- void **set_C21_att_model** (float freq, const float *Qm, int nQmodel, std::complex< float > *__restrict c21, int funcid=0, bool do_deriv=false)
- void **__myfwrite** (const void *__ptr, size_t __size, size_t __nitems, FILE *__stream)
- template< typename T >
  void **write_binary_f** (FILE *fp, const T *data, size_t n)

Variables

- const int **NSLS** = 5
- const std::array< double, NSLS > **y_sls_ref** = {1.93044501, 1.64217132, 1.73606189, 1.42826439, 1.↩66934129}
- const std::array< double, NSLS > **w_sls_ref** = {4.71238898e-02, 6.63370885e-01, 9.42477796e+00, 1.↩14672436e+02,1.05597079e+03}

4.1.1 Detailed Description

derivative operators:

Note

y.H @ (dA / dm - alpha dB / dm) @ x

4.1.2 Function Documentation

4.1.2.1 get_cQ_kl()

```
template<typename T >
void specswd::get_cQ_kl (
            T & dcdm,
            T c,
            float & dcLdm,
            float & dQiLdm )
```

convert d\tilde{c}/dm to dcL/dm, dQiL/dm, where \tilde{c} = c(1 + 1/2 i Qi)

Parameters

dcdm,Frechet	kernel for complex phase velocity, rst m
c	complex phase velocity
dcLdm,dQiLdm	dc / dm and dQi / dm

4.1.2.2 get_sls_modulus_factor()

```
std::complex< float > specswd::get_sls_modulus_factor (
            float freq,
            float Q )
```

get SLS Q terms on the elastic modulus

Parameters

freq	current frequency
Qa	Qvalue
xA	

4.1.2.3 get_sls_Q_derivative()

```
void specswd::get_sls_Q_derivative (
            float freq,
            float Q,
            crealw & s,
            crealw & dsdqi )
```

Get the Q factor and derivative for SLS model.

Parameters

freq	frequency
Q	current Q
s	modulus factor mu = mu ∗ s
dsdqi	Q^{-1} derivative ds / dQi

4.1.2.4 love_deriv_op_()

```
template<typename T = float>
void specswd::love_deriv_op_ (
            float freq,
            T c,
            T coef,
            const T * y,
            const T * x,
            int nspec_el,
```

```
            int nglob_el,
            const int * ibool_el,
            const float * jaco,
            const float * xN,
            const float * xL,
            const float * xQN,
            const float * xQL,
            float *__restrict frekl_c,
            float *__restrict frekl_q )
```

compute coef * y^H @ d((w^2 M -E) - k^ K)/dm_i @ x dm_i

Parameters

freq	current frequency
c	current phase velocity
coef	scaling coefs
egn	eigen vector,shape(nglob_el)
nspec_el/nglob_el	mesh nelemnts/unique points for elastic
ibool_el	elastic connectivity matrix, shape(nspec_el*NGLL+NGRL)
jaco	jacobian matrix, shape (nspec_el + 1)
xN/xL/xQN/xQL/rho	model parameters, shape(nspec_el*NGLL+NGRL)
frekl_c	dc/d(N/L/rho) (elastic) or dc/d(N/L/QN/QL/rho) (anelstic)
frekl_q	nullptr or dqc/d(N/L/QN/QL/rho) (anelstic)

Note

frekl_c and frekl_q should be set to 0 before calling this routine

4.1.2.5 rayl_deriv_op_()

```
template<typename T = float>
void specswd::rayl_deriv_op_ (
            float freq,
            T c,
            T coef,
            const T * y,
            const T * x,
            int nspec_el,
            int nspec_ac,
            int nspec_el_grl,
            int nspec_ac_grl,
            int nglob_el,
            int nglob_ac,
            const int * el_elmnts,
            const int * ac_elmnts,
            const int * ibool_el,
            const int * ibool_ac,
            const float * jaco,
            const float * xrho_el,
            const float * xrho_ac,
            const float * xA,
            const float * xC,
```

```
            const float * xL,
            const float * xeta,
            const float * xQA,
            const float * xQC,
            const float * xQL,
            const float * xkappa_ac,
            const float * xQk_ac,
            float *__restrict frekl_c,
            float *__restrict frekl_q )
```

compute coef ∗ y^dag @ d((w^2 M -E) - k^2 K)/dm_i @ x dm_i

Parameters

freq	current frequency
c	current phase velocity
coef	derivative scaling coefs
y/x	dot vector,shape(nglob_el∗2+nglob_ac)
nspec_el/nglob_el	mesh nelemnts/unique points for elastic
nspec_ac/nglob_ac	mesh nelemnts/unique points for acoustic
nspec_el/ac_grl	no. of GRL elements
ibool_el	elastic connectivity matrix, shape(nspec_el∗NGLL+nspec_el_grl∗NGRL)
ibool_ac	elastic connectivity matrix, shape(nspec_ac∗NGLL+nspec_ac_grl∗NGRL)
jaco	jacobian matrix, shape (nspec_el + 1)
xA/xC/xL/xeta/xQA/xQC/xQL/xrho	elastic model parameters,ibool_el.shape
xkappa_ac/xQk_ac/xrho_ac	acoustic model parameters, ibool_ac.shape
frekl_c	dc/d(A/C/L/kappa/rho) (elastic) or dc/d(A/C/L/QA/QC/QL/kappa/Qk/rho) (anelstic)
frekl_q	nullptr or dqc/d(A/C/L/QA/QC/QL/kappa/Qk/rho) (anelstic)

Note

frekl_c and frekl_q should be set to 0 before calling this routine

4.1.2.6 schur_qz()

```
template<typename COMMTP = double, typename SAVETP = float>
void specswd::schur_qz (
            int ng,
            Eigen::MatrixX< COMMTP > & A,
            Eigen::MatrixX< COMMTP > & B,
            std::vector< SAVETP > & Qmat_,
            std::vector< SAVETP > & Zmat_,
            std::vector< SAVETP > & Smat_,
            std::vector< SAVETP > & Spmat_ )
```

QZ decomposition of matrix A and B.

Parameters

ng	rows/cols of A, B
A,B	two matrices, type = COMMTP
Q,Z,S,SP	QZ matrix, where A = Q @ S @ Z.H, B = Q @ S' @ Z.H

4.1.2.7 solve_christoffel()

```
void specswd::solve_christoffel (
            float phi,
            const float * c21,
            float & cmin,
            float & cmax )
```

find the min/max phase velocity by solving Christoffel equations G_{ik} g_k = v^2 g_i

Parameters

phi	direction angle, in rad
c21	c21 tensor, shape(21)
cmin/cmax	min/max phase velocity

Chapter 5

Class Documentation

5.1 specswd::Mesh Struct Reference

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
- void **print_database** () const
- void **allocate_1D_model** (int nz0, int swd_type, int has_att)
- void **create_model_attributes** ()
- 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, attenutation 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_ (float freq)
  create database for Love wave
- void create_db_rayl_ (float freq)
  create database for Love wave
- void create_db_aniso_ (float freq)
  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**

5.1.1 Member Function Documentation

5.1.1.1 create_database()

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

Create SEM database by using input model info.

Parameters

freq	current frequency
phi	directional angle

5.1.1.2 create_db_aniso_()

```
void specswd::Mesh::create_db_aniso_ (
            float freq )
```

create database for Love wave

Parameters

freq	current frequency,in Hz

5.1.1.3 create_db_love_()

```
void specswd::Mesh::create_db_love_ (
            float freq )
```

create database for Love wave

Parameters

freq	current frequency,in Hz

5.1.1.4 create_db_rayl_()

```
void specswd::Mesh::create_db_rayl_ (
            float freq )
```

create database for Love wave

Parameters

freq	current frequency,in Hz

5.1.1.5 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

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

5.1.1.6 project_kl()

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

project kernels to original 1-D model

Parameters

frekl	derivatives, shape(nspec∗NGLL+NGRL)
kl_out	derivatives on original 1-Dmodel, shape(nz_)

5.1.1.7 read_model()

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

read 1D model

Parameters

filename	1D model file

5.1.1.8 read_model_full_aniso_()

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

read 1D full anisotropy model model for Rayleigh wave

Parameters

filename	1D model file

5.1.1.9 read_model_header_()

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

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

Parameters

filename	model filename

5.1.1.10 read_model_love_()

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

read 1D VTI model for Love wave

Parameters

filename	1D model file

5.1.1.11 read_model_rayl_()

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

read 1D VTI model for Rayleigh wave

Parameters

filename	1D model file

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

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

5.2 specswd::SolverLove Class Reference

Public Member Functions

- void **prepare_matrices** (float freq, const Mesh &M)
  prepare M/K/E matrices for Love wave, an/elastic case
- void compute_egn (const Mesh &M, float freq, std::vector< float > &c, std::vector< float > &egn, bool save↩_qz=false)
  compute Love wave dispersion and eigenfunctions, elastic case
- void compute_egn_att (const Mesh &M, float freq, std::vector< scmplx > &c, std::vector< scmplx > &egn, bool save_qz=false)
  compute rayleigh wave dispersion and eigenfunctions, visco-elastic case
- float **group_vel** (const Mesh &M, float freq, float c, const float ∗egn) const
  compute velocity of love wave, elastic case
- scmplx **group_vel_att** (const Mesh &M, float freq, scmplx c, const scmplx ∗egn) const
  compute velocity of love wave, anelastic case
- void compute_phase_kl (const Mesh &M, float freq, 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, float freq, 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 freq, float c, const float ∗egn, std::vector< float > &frekl) const
  compute group velocity and kernels for love wave phase velocity, elastic case
- void compute_group_kl_att (const Mesh &M, float freq, scmplx c, const scmplx ∗egn, std::vector< float > &frekl_c, std::vector< float > &frekl_q) const
  compute love wave group velocity kernels, visco-elastic case
- void **egn2displ** (const Mesh &M, float freq, float c, const float ∗egn, float ∗__restrict displ) const
- void **egn2displ_att** (const Mesh &M, float freq, scmplx c, const scmplx ∗egn, scmplx ∗__restrict displ) const
- void transform_kernels (const Mesh &M, std::vector< float > &frekl) const
  transform modulus kernel to velocity kernel, Love wave case

5.2.1 Member Function Documentation

5.2.1.1 compute_egn()

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

compute Love wave dispersion and eigenfunctions, elastic case

Parameters

freq	current frequency
c	dispersion, shape(nc)
egn	eigen functions(displ at y direction), shape(nc,nglob_el)
save_qz	if true, save QZ matrix

5.2.1.2 compute_egn_att()

```
void specswd::SolverLove::compute_egn_att (
            const Mesh & mesh,
            float freq,
            std::vector< scmplx > & c,
            std::vector< scmplx > & displ,
            bool save_qz = false )
```

compute rayleigh wave dispersion and eigenfunctions, visco-elastic case

Parameters

freq	current frequency
c	dispersion, shape(nc) c = c0(1 + iQL^{-1})
egn	eigen functions(displ at y direction), shape(nc,nglob_el)
save_qz	if true, save QZ matrix

5.2.1.3 compute_group_kl()

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

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

Parameters

freq	current frequency
c	current phase velocity
displ	eigen function, shape(nglob_el)
frekl	Frechet kernels (N/L/rho) for elastic parameters, shape(3,nspec*NGLL + NGRL)

5.2.1.4 compute_group_kl_att()

```
void specswd::SolverLove::compute_group_kl_att (
            const Mesh & mesh,
```

```
            float freq,
            scmplx c,
            const scmplx * egn,
            std::vector< float > & frekl_c,
            std::vector< float > & frekl_q ) const
```

compute love wave group velocity kernels, visco-elastic case

Parameters

freq	current frequency
c	current complex phase velocity
displ	eigen function, shape(nglob_el)
frekl←_c	dRe(u)/d(N/L/QN/QL/rho) shape(5,nspec∗NGLL + NGRL)
frekl←_q	d(qi)/d(N/L/QN/QL/rho) shape(5,nspec∗NGLL + NGRL)

5.2.1.5 compute_phase_kl()

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

compute love wave phase velocity kernels, elastic case

Parameters

freq	current frequency
c	current phase velocity
displ	eigen function, shape(nglob_el)
frekl	Frechet kernels (N/L/rho) for elastic parameters, shape(3,nspec∗NGLL + NGRL)

5.2.1.6 compute_phase_kl_att()

```
void specswd::SolverLove::compute_phase_kl_att (
            const Mesh & M,
            float freq,
            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

freq	current frequency

Parameters

c	current complex phase velocity
displ	eigen function, shape(nglob_el)
frekl↩ _c	dRe(c)/d(N/L/QN/QL/rho) shape(5,nspec∗NGLL + NGRL)
frekl↩ _q	d(qi)/d(N/L/QN/QL/rho) shape(5,nspec∗NGLL + NGRL)

5.2.1.7 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

frekl	frechet kernels, the shape depends on: • 1: elastic love wave: N/L/rho -> vsh/vsv/rho • 2: anelastic love wave: N/L/QNi/QLi/rho -> 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

5.3 specswd::SolverRayl Class Reference

Public Member Functions

- void **prepare_matrices** (float freq, const Mesh &M)
- void compute_egn (const Mesh &M, float freq, std::vector< float > &c, std::vector< float > &ur, std::vector< float > &ul, bool save_qz=false)

  compute rayleigh wave dispersion and eigenfunctions, elastic case
- void compute_egn_att (const Mesh &M, float freq, std::vector< scmplx > &c, std::vector< scmplx > &ur, std::vector< scmplx > &ul, bool save_qz=false)

  compute rayleigh wave dispersion and eigenfunctions, visco-elastic case
- float **group_vel** (const Mesh &M, float freq, float c, const float ∗ur, const float ∗ul) const

compute velocity of love wave, elastic case

- scmplx **group_vel_att** (const Mesh &M, float freq, scmplx c, const scmplx ∗ur, const scmplx ∗ul) const

  compute velocity of love wave, elastic case
- void compute_phase_kl (const Mesh &M, float freq, 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, float freq, 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 freq, float c, const float ∗ur, const float ∗ul, std::vector< float > &frekl) const
- void **compute_group_kl_att** (const Mesh &M, float freq, scmplx c, const scmplx ∗ur, const scmplx ∗ul, std::vector< float > &frekl_c, std::vector< float > &frekl_q) const
- void **egn2displ** (const Mesh &M, float freq, float c, const float ∗egn, float ∗__restrict displ) const
- void **egn2displ_att** (const Mesh &M, float freq, scmplx c, const scmplx ∗egn, scmplx ∗__restrict displ) const
- void transform_kernels (const Mesh &M, std::vector< float > &frekl) const

  transform modulus kernel to velocity kernel, Rayleigh wave case

5.3.1 Member Function Documentation

5.3.1.1 compute_egn()

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

compute rayleigh wave dispersion and eigenfunctions, elastic case

Parameters

freq	current frequency
c	dispersion, shape(nc) c = c0(1 + iQL^{-1})
ur/ul	left/right eigenvectors, shape(nc,nglob_el∗2+nglob_ac)
save_qz	if true, save QZ matrix

5.3.1.2 compute_egn_att()

```
void specswd::SolverRayl::compute_egn_att (
            const Mesh & mesh,
            float freq,
            std::vector< scmplx > & c,
            std::vector< scmplx > & ur,
            std::vector< scmplx > & ul,
            bool save_qz = false )
```

compute rayleigh wave dispersion and eigenfunctions, visco-elastic case

Parameters

freq	current frequency
c	dispersion, shape(nc) c = c0(1 + iQL^{-1})
ur/ul	left/right eigenvectors, shape(nc,nglob_el∗2+nglob_ac)
save_qz	if true, save QZ matrix

5.3.1.3 compute_phase_kl()

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

compute Rayleigh wave phase kernels, elastic case

Parameters

freq	current frequency
c	current phase velocity
ur/ul	right/left eigen function, shape(nglob_el∗2+nglob_ac)
frekl	Frechet kernels A/C/L/eta/kappa/rho_kl kernels for elastic parameters, shape(6,nspec∗NGLL + NGRL)

5.3.1.4 compute_phase_kl_att()

```
void specswd::SolverRayl::compute_phase_kl_att (
            const Mesh & M,
            float freq,
            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

freq	current frequency
c	current phase velocity
ur/ul	right/left eigen function, shape(nglob_el∗2+nglob_ac)
frekl↩_c	dRe(c)/d(A/C/L/eta/Qa/Qc/Ql/kappa/Qk/rho) kernels for elastic parameters, shape(10,nspec∗NGLL + NGRL)
frekl↩_q	dRe(Q_R)/d(A/C/L/eta/Qa/Qc/Ql/kappa/Qk/rho) kernels for elastic parameters, shape(10,nspec∗NGLL + NGRL)

5.3.1.5 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

frekl	frechet kernels, the shape depends on: • `1`: elastic rayleigh wave: A/C/L/eta/kappa/rho -> vph/vpv/vsv/eta/vp/rho • `2` anelastic rayleigh wave: A/C/L/eta/QAi/QCi/QLi/kappa/Qki/rho -> 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
- group_velocity.cpp
- sem.cpp
- transform.cpp

Chapter 6

File Documentation

6.1 precision.hpp

```
#ifndef SPECSWD_PRECISION_H_
#define SPECSWD_PRECISION_H_

#ifdef SPECSWD_EGN_DOUBLE
typedef double realw;
#define LAPACKE_REAL(name) LAPACKE_d ## name
#define LAPACKE_CMPLX(name) LAPACKE_z ## name
#define LCREALW lapack_complex_double

#else
typedef float realw;
#define LAPACKE_REAL(name) LAPACKE_s ## name
#define LAPACKE_CMPLX(name) LAPACKE_c ## name
#define LCREALW lapack_complex_float
#endif

typedef std::complex<realw> crealw;


#endif
```

6.2 mesh.hpp

```
#ifndef SPECSWD_MESH_H_
#define SPECSWD_MESH_H_

#include <complex>
#include <vector>
#include <array>

namespace specswd
{

struct Mesh {

    // SEM Mesh
    int nspec,nspec_grl; // no. of elements for gll/grl layer
    int nglob; // no. of unique points
    std::vector<int> ibool; // connectivity matrix, shape(nspec * NGLL + NGRL)
    std::vector<float> skel;  // skeleton, shape(nspec * 2 + 2)
    std::vector<float> znodes; // shape(nspec * NGLL + NGRL)
    std::vector<float> jaco; // jacobian for GLL, shape(nspec + 1) dz / dxi
    std::vector<float> zstore; // shape(nglob)

    // element type for each medium
    int nspec_ac,nspec_el;
    int nspec_ac_grl,nspec_el_grl;
    std::vector<char> is_elastic, is_acoustic;
    std::vector<int> el_elmnts,ac_elmnts; // elements for each media, shape(nspec_? + nspec_?_grl)

    // unique array for acoustic/elastic
    int nglob_ac, nglob_el;
    std::vector<int> ibool_el, ibool_ac; // connectivity matrix, shape shape(nspec_? + nspec_?_grl)

```

```
    // density and elastic parameters
    std::vector<float> xrho_ac; // shape(nspec_ac * NGLL + nspec_ac_grl * NGRL)
    std::vector<float> xrho_el; // shape (nsepc_el * NGLL + nspec_el_grl * NGRL)

    // attenuation/type flag
    bool HAS_ATT;
    int SWD_TYPE; // =0 Love wave, = 1 for Rayleigh = 2 full aniso

    // vti media
    std::vector<float> xA,xC,xL,xeta,xN; // shape(nspec_el * NGLL+ nspec_el_grl * NGRL)
    std::vector<float> xQA,xQC,xQL,xQN; // shape(nspec_el * NGLL+ nspec_el_grl * NGRL), Q model

    // full anisotropy
    int nQmodel_ani; // no. of Q used for anisotropy
    std::vector<float> xC21; // shape(21,nspec_el * NGLL+ nspec_el_grl * NGRL)
    std::vector<float> xQani; // shape(nQmodel_ani,nspec_el * NGLL+ nspec_el_grl * NGRL)

    // fluid vti
    std::vector<float> xkappa_ac,xQk_ac;

    // fluid-elastic boundary
    int nfaces_bdry;
    std::vector<int> ispec_bdry; // shape(nfaces_bdry,2) (i,:) = [ispec_ac,ispec_el]
    std::vector<char> bdry_norm_direc; //  shape(nfaces_bdry), = 1 point from acoustic -> z direc elastic

    int nz_tomo, nregions;
    std::vector<float> rho_tomo;
    std::vector<float> vpv_tomo,vph_tomo,vsv_tomo,vsh_tomo,eta_tomo;
    std::vector<float> QC_tomo,QA_tomo,QL_tomo,QN_tomo;
    std::vector<float> c21_tomo,Qani_tomo;
    std::vector<float> depth_tomo;
    std::vector<int> region_bdry; // shape(nregions,2)
    std::vector<int> iregion_flag; // shape(nspec + 1), return region flag

    // interface with layered model
    std::vector<char> is_el_reg, is_ac_reg; // shape(nregions)

    float PHASE_VELOC_MIN,PHASE_VELOC_MAX;

    // public functions
    void read_model(const char *filename);
    void create_database(float freq,float phi);
    void print_model() const;
    void print_database() const;
    void allocate_1D_model(int nz0,int swd_type,int has_att);
    void create_model_attributes();

    // interpolate model
    void interp_model(const float *param,const std::vector<int> &elmnts,std::vector<float> &md) const;
    void project_kl(const float *frekl, float *kl_out) const;

    // private functions below
    // ==============================
    //
    void create_material_info_();

    // 1-D model
    void read_model_header_(const char *filename);
    void read_model_love_(const char *filename);
    void read_model_rayl_(const char *filename);
    void read_model_full_aniso_(const char *filename);

    // create SEM database
    void compute_minmax_veloc_(float phi,std::vector<float> &vmin,std::vector<float> &vmax);
    void create_db_love_(float freq);
    void create_db_rayl_(float freq);
    void create_db_aniso_(float freq);
};

} // namespace specswd




#endif
```

6.3 attenuation.hpp

```

#ifndef SPECSWD_ATT_TABLE_H_
#define SPECSWD_ATT_TABLE_H_

```

```
#include <complex>

namespace specswd
{

const int NSLS = 5;

std::complex<float> get_sls_modulus_factor(float freq,float Q);
void
get_sls_Q_derivative(float freq,float Qm,std::complex<float> &s,
                     std::complex<float> &dsdqi);

void set_C21_att_model(float freq,const float *Qm,int nQmodel,
                        std::complex<float>* __restrict c21,
                        int funcid=0,bool do_deriv=false);


}

#endif
```

6.4 GQTable.hpp

```
#ifndef SPECSWD_GQTABLE_H_
#define SPECSWD_GQTABLE_H_

#include <array>

namespace GQTable
{

const int NGLL = 7, NGRL = 20;
extern std::array<float,NGLL> xgll,wgll;
extern std::array<float,NGRL> xgrl,wgrl;
extern std::array<float,NGLL*NGLL> hprimeT,hprime; // hprimeT(i,j) = l'_i(xi_j)
extern std::array<float,NGRL*NGRL> hprimeT_grl,hprime_grl;

void initialize();

} // GQTable


#endif
```

6.5 iofunc.hpp

```
#ifndef SPECSWD_IOFUNC_H_
#define SPECSWD_IOFUNC_H_

#include <iostream>

namespace specswd
{


inline void __myfwrite(const void *__ptr, size_t __size, size_t __nitems, FILE *__stream)
{
    size_t size = fwrite(__ptr,__size,__nitems,__stream);
    if(size != __nitems) {
        printf("cannot write to binary!\n");
        exit(1);
    }
}


template<typename T>
void
write_binary_f(FILE *fp, const T *data, size_t n)
{
    // write integers of the size
    int size = (int)(n * sizeof(T));

    // integer front
    __myfwrite(&size,sizeof(int),1,fp);

    // data
    __myfwrite(data,sizeof(T),n,fp);

    // integer back
```

```
    __myfwrite(&size,sizeof(int),1,fp);
}

} // namespace specswd


#endif
```

6.6 quadrature.hpp

```
#ifndef SPECSWD_QUADRATURE_H_
#define SPECSWD_QUADRATURE_H_
#include <cmath>

//GLL
void gauss_legendre_lobatto(double* knots, double* weights, size_t length);
void lagrange_poly(double xi,size_t nctrl,const double *xctrl,
                double *h,double*  hprime);

// GRL
void gauss_radau_laguerre(double *xgrl,double *wgrl,size_t length);
double laguerre_func(size_t n, double x);

#endif
```

6.7 frechet_op.hpp

```
#ifndef SPECSWD_FRECHET_OP_H_
#define SPECSWD_FRECHET_OP_H_

#include "shared/attenuation.hpp"
#include "shared/GQTable.hpp"

namespace specswd
{

template <typename T>  void
get_cQ_kl(T &dcdm,T c,
          float &dcLdm,float &dQiLdm)
{
    static_assert(std::is_same_v<std::complex<float>,T>);
    float cl = c.real();
    float Qi = 2. * c.imag() / cl;
    dcLdm = dcdm.real();
    dQiLdm = (dcdm.imag() * 2. - Qi * dcLdm) / cl;
}

template<typename T = float >
void love_deriv_op_(float freq, T c, T coef,const T *y,const T *x,
                    int nspec_el,int nglob_el, const int *ibool_el,
                    const float *jaco, const float *xN,
                    const float *xL,const float *xQN,
                    const float *xQL, float * __restrict frekl_c,
                    float * __restrict frekl_q)
{
    // check template type
    static_assert(std::is_same_v<float,T> || std::is_same_v<std::complex<float>,T>);

    using namespace GQTable;
    std::array<T,NGRL> rW,lW;
    size_t size = nspec_el*NGLL + NGRL;
    T om = 2 * M_PI * freq;
    T k2 = (om * om) / (c * c);
    for(int ispec = 0; ispec < nspec_el + 1; ispec ++) {
        const float *hp = &hprime[0];
        const float *w = &wgll[0];
        float J = jaco[ispec]; // jacobians in this layers
        int NGL = NGLL;
        int id = ispec * NGLL;

        // GRL layer
        if(ispec == nspec_el) {
            hp = &hprime_grl[0];
            w = &wgrl[0];
            NGL = NGRL;
        }

        // cache displ in a element
        for(int i = 0; i < NGL; i ++) {
```

```
            int iglob = ibool_el[id+i];
            rW[i] = x[iglob];
            lW[i] = y[iglob];
            if constexpr (std::is_same_v<T,std::complex<float») {
                lW[i] = std::conj(lW[i]);
            }
        }

        // compute kernels
        T dc_drho{}, dc_dN{}, dc_dL{};
        T dc_dqni{}, dc_dqli{};
        T sn = 1., sl = 1.;
        T dsdqni{}, dsdqli{};
        for(int m = 0; m < NGL; m ++) {
            dc_drho = w[m] * J * om * om * rW[m] * lW[m] * coef;

            // get sls derivative if required
            if constexpr (std::is_same_v<T,std::complex<float») {
                get_sls_Q_derivative(freq,xQN[id+m],sn,dsdqni);
                get_sls_Q_derivative(freq,xQL[id+m],sl,dsdqli);
                dsdqni *= xN[id+m];
                dsdqli *= xL[id+m];
            }

            // N kernel
            T temp = -k2 * rW[m] * lW[m]  * J * w[m] * coef;
            dc_dN = temp * sn;
            dc_dqni = temp * dsdqni;

            // L kernel
            T sx{},sy{};
            for(int i = 0; i < NGL; i ++) {
                sx += hp[m*NGL+i] * rW[i];
                sy += hp[m*NGL+i] * lW[i];
            }
            temp = -sx * sy * w[m] / J * coef;
            dc_dL = temp * sl;
            dc_dqli = temp * dsdqli;

            // copy to frekl
            int id1 = id + m;
            if constexpr (std::is_same_v<T,std::complex<float») {
                get_cQ_kl(dc_dN,c,frekl_c[0*size+id1],frekl_q[0*size+id1]);
                get_cQ_kl(dc_dL,c,frekl_c[1*size+id1],frekl_q[1*size+id1]);
                get_cQ_kl(dc_dqni,c,frekl_c[2*size+id1],frekl_q[2*size+id1]);
                get_cQ_kl(dc_dqli,c,frekl_c[3*size+id1],frekl_q[3*size+id1]);
                get_cQ_kl(dc_drho,c,frekl_c[4*size+id1],frekl_q[4*size+id1]);
            }
            else {
                frekl_c[0*size+id1] = dc_dN;
                frekl_c[1*size+id1] = dc_dL;
                frekl_c[2*size+id1] = dc_drho;
            }
        }
    }
}

template<typename T = float >
void
rayl_deriv_op_(float freq,T c,T coef,const T *y, const T *x,
                int nspec_el,int nspec_ac,int nspec_el_grl,int nspec_ac_grl,int nglob_el,
                int nglob_ac, const int *el_elmnts,const int *ac_elmnts,
                const int* ibool_el, const int* ibool_ac,
                const float *jaco,const float *xrho_el,const float *xrho_ac,
                const float *xA, const float *xC,const float *xL,const float *xeta,
                const float *xQA, const float *xQC,const float *xQL,
                const float *xkappa_ac, const float *xQk_ac,
                float *__restrict frekl_c,
                float *__restrict frekl_q)
{
    // check template type
    static_assert(std::is_same_v<float,T> || std::is_same_v<std::complex<float>,T>);

    // constants
    using namespace GQTable;
    size_t size = nspec_el *  NGLL + nspec_el_grl * NGRL +
                    nspec_ac * NGLL + nspec_ac_grl * NGRL;
    T om = 2 * M_PI * freq;
    T k2 = std::pow(om / c,2);

    // loop elastic elements
    std::array<T,NGRL> U,V,lU,lV;
    for(int ispec = 0; ispec < nspec_el + nspec_el_grl; ispec ++) {
        int iel = el_elmnts[ispec];
        int id = ispec * NGLL;

        const float *weight = wgll.data();
```

```
        const float *hp = hprime.data();
        int NGL = NGLL;

        // jacobian
        float J = jaco[iel];

        // grl case
        if(ispec == nspec_el) {
            weight = wgrl.data();
            hp = hprime_grl.data();
            NGL = NGRL;
        }

        // cache U,V and lU,lV
        for(int i = 0; i < NGL; i ++) {
            int iglob = ibool_el[id + i];
            U[i] = x[iglob];
            V[i] = x[iglob + nglob_el];
            lU[i] = y[iglob];
            lV[i] = y[iglob + nglob_el];
            if  constexpr (std::is_same_v<T,std::complex<float») {
                lU[i] = std::conj(lU[i]);
                lV[i] = std::conj(lV[i]);
            }
        }

       // compute kernel
        T dc_drho{}, dc_dA{}, dc_dC{}, dc_dL{};
        T dc_deta{}, dc_dQci{},dc_dQai{},dc_dQli{};
        const T two = 2.;
        for(int m = 0; m < NGL; m ++) {
            T temp = weight[m] * J * coef;
            dc_drho = temp * om * om *
                (U[m] * lU[m] + V[m] * lV[m]);

            // get sls factor if required
            T sa = 1.,sl = 1.,sc = 1.;
            T dsdqai{},dsdqci{},dsdqli{};
            float C = xC[id+m], A = xA[id+m],
                   L = xL[id+m], eta = xeta[m];
            if constexpr (std::is_same_v<T,std::complex<float») {
                get_sls_Q_derivative(freq,xQA[id+m],sa,dsdqai);
                get_sls_Q_derivative(freq,xQC[id+m],sc,dsdqai);
                get_sls_Q_derivative(freq,xQL[id+m],sl,dsdqai);
                dsdqai *= A;
                dsdqci *= C;
                dsdqli *= L;
            }

            // K matrix
            // dc_dA
            temp = -weight[m] * J * k2 * U[m] * lU[m] * coef;
            dc_dA = temp * sa; dc_dQai = temp * dsdqai;

            // dc_dL
            temp = -weight[m] * J * k2 * V[m] * lV[m] * coef;
            dc_dL = temp * sl; dc_dQli = temp * dsdqli;

            // Ematrix
            T sx{},sy{},lsx{},lsy{};
            for(int i = 0; i < NGL; i ++) {
                sx += hp[m*NGL+i] * U[i];
                sy += hp[m*NGL+i] * V[i];
                lsx += hp[m*NGL+i] * lU[i];
                lsy += hp[m*NGL+i] * lV[i];
            }
            temp = -weight[m] / J * sx * lsx * coef ;
            dc_dL += temp * sl; dc_dQli += temp * dsdqli;

            temp = - weight[m] / J * sy * lsy * coef;
            dc_dC = temp * sc; dc_dQci = temp * dsdqci;

            // eta
            temp = - (k2 * weight[m] * U[m] * lsy +
                        weight[m] * lU[m] * sy) * coef;
            dc_deta = temp * (A*sa - two*L*sl);

            temp *= eta;
            dc_dA += temp * sa; dc_dQai += temp * dsdqai;
            dc_dL += - temp * two * sl; dc_dQli += -temp * two * dsdqli;

            temp = k2 * weight[m] * lV[m] * sx + weight[m] * V[m] * lsx;
            temp *= coef;
            dc_dL +=  temp * sl;
            dc_dQli += temp * dsdqli;

            // copy them to frekl
```

```
            int id1 = iel * NGLL + m;
            if constexpr (std::is_same_v<T,float>) {
                frekl_c[0*size+id1] = dc_dA;
                frekl_c[1*size+id1] = dc_dC;
                frekl_c[2*size+id1] = dc_dL;
                frekl_c[3*size+id1] = dc_deta;
                frekl_c[5*size+id1] = dc_drho;
            }
            else {
                get_cQ_kl(dc_dA,c,frekl_c[0*size+id1],frekl_q[0*size+id1]);
                get_cQ_kl(dc_dC,c,frekl_c[1*size+id1],frekl_q[1*size+id1]);
                get_cQ_kl(dc_dL,c,frekl_c[2*size+id1],frekl_q[2*size+id1]);
                get_cQ_kl(dc_deta,c,frekl_c[3*size+id1],frekl_q[3*size+id1]);
                get_cQ_kl(dc_dQai,c,frekl_c[4*size+id1],frekl_q[4*size+id1]);
                get_cQ_kl(dc_dQci,c,frekl_c[5*size+id1],frekl_q[5*size+id1]);
                get_cQ_kl(dc_dQli,c,frekl_c[6*size+id1],frekl_q[6*size+id1]);
                get_cQ_kl(dc_drho,c,frekl_c[9*size+id1],frekl_q[9*size+id1]);
            }
        }
    }

    // acoustic eleemnts
    std::array<T,NGRL> chi,lchi;
    for(int ispec = 0; ispec < nspec_ac + nspec_ac_grl; ispec ++) {
        int iel = ac_elmnts[ispec];
        int id = ispec * NGLL;
        const float *weight = wgll.data();
        const float *hp = hprime.data();
        int NGL = NGLL;

        // jacobians
        float J = jaco[iel];

        // grl case
        if(ispec == nspec_ac) {
            weight = wgrl.data();
            hp = hprime_grl.data();
            NGL = NGRL;
        }

        // cache chi and lchi in one element
        for(int i = 0; i < NGL; i ++) {
            int iglob = ibool_ac[id + i];
            chi[i] = (iglob == -1) ? 0: x[iglob+nglob_el*2];
            lchi[i] = (iglob == -1) ? 0.: y[iglob+nglob_el*2];
            if  constexpr (std::is_same_v<T,std::complex<float>>) {
                lchi[i] = std::conj(lchi[i]);
            }
        }

        // derivatives
        T dc_dkappa{},dc_drho{}, dc_dqki{};
        T sk = 1., dskdqi = 0.;
        for(int m = 0; m < NGL; m ++ ){
            // copy material
            float rho = xrho_ac[id+m];
            float kappa = xkappa_ac[id+m];
            if constexpr (std::is_same_v<T,std::complex<float>>) {
                get_sls_Q_derivative(freq,xQk_ac[id+m],sk,dskdqi);
                dskdqi *= kappa;
            }

            // kappa kernel
            T temp = std::pow(om/(sk * kappa),2) *weight[m]* J*
                        chi[m] * lchi[m] * coef;
            dc_dkappa = temp * sk;
            dc_dqki =  temp * dskdqi;

            dc_drho = -k2 * std::pow(om/rho,2) *weight[m]* J*
                        chi[m] * lchi[m] * coef;

            T sx{},sy{};
            for(int i = 0; i < NGL; i ++) {
                sx += hp[m*NGL+i] * chi[i];
                sy += hp[m*NGL+i] * lchi[i];
            }
            dc_drho += weight[m] / J / (rho*rho) * sx * sy * coef;

            // copy to frekl
            int id1 = iel * NGLL + m;
            if constexpr (std::is_same_v<T,float>) {
                frekl_c[4*size+id1] = dc_dkappa;
                frekl_c[5*size+id1] = dc_drho;
            }
            else {
                get_cQ_kl(dc_dkappa,c,frekl_c[7*size+id1],frekl_q[7*size+id1]);
                get_cQ_kl(dc_dqki,c,frekl_c[8*size+id1],frekl_q[8*size+id1]);
```

```
                get_cQ_kl(dc_drho,c,frekl_c[9*size+id1],frekl_q[9*size+id1]);
            }

        }
    }
}


} // namespace specswd

#endif
```

6.8 vti.hpp

```
#ifndef SPECSWD_SOLVER_H_
#define SPECSWD_SOLVER_H_

#include "mesh/mesh.hpp"

#include <complex>
#include <vector>


namespace specswd
{

typedef std::complex<float>  scmplx;

class SolverLove {

private:
    // solver matrices
    std::vector<float> Mmat,Emat,Kmat;
    std::vector<scmplx> CMmat,CEmat,CKmat;

    // QZ matrix all are column major
    std::vector<float> Qmat_,Zmat_,Smat_,Spmat_; // column major!
    std::vector<scmplx> cQmat_,cZmat_,cSmat_,cSpmat_;

public:

    // eigenfunctions/values
    void prepare_matrices(float freq,const Mesh &M);
    void compute_egn(const Mesh &M,float freq,
                    std::vector<float> &c,
                    std::vector<float> &egn,
                    bool save_qz=false);
    void compute_egn_att(const Mesh &M,float freq,
                        std::vector<scmplx> &c,
                        std::vector<scmplx> &egn,
                        bool save_qz=false);

    // group velocity
    float group_vel(const Mesh &M,float freq,
                    float c,const float *egn) const;
    scmplx group_vel_att(const Mesh &M,float freq,
                         scmplx c, const scmplx *egn) const ;

    // phase velocity kernels
    void compute_phase_kl(const Mesh &M,float freq,
                        float c,const float *egn,
                        std::vector<float> &frekl) const;
    void compute_phase_kl_att(const Mesh &M,float freq,
                        scmplx c, const scmplx *egn,
                        std::vector<float> &frekl_c,
                        std::vector<float> &frekl_q) const;
    // group kernel
    void compute_group_kl(const Mesh &M,float freq,
                        float c,const float *egn,
                        std::vector<float> &frekl) const;

    void compute_group_kl_att(const Mesh &M,float freq,
                        scmplx c, const scmplx *egn,
                        std::vector<float> &frekl_c,
                        std::vector<float> &frekl_q) const;
    // tranforms
    void egn2displ(const Mesh &M,float freq,float c,
                    const float*egn, float * __restrict displ) const;
    void egn2displ_att(const Mesh &M,float freq,scmplx c,const scmplx *egn,
                      scmplx * __restrict displ) const;
    void transform_kernels(const Mesh &M,std::vector<float> &frekl) const;
};

```

```
class SolverRayl {

private:
    // solver matrices
    std::vector<float> Mmat,Emat,Kmat;
    std::vector<scmplx> CMmat,CEmat,CKmat;

    // QZ matrix all are column major
    std::vector<float> Qmat_,Zmat_,Smat_,Spmat_; // column major!
    std::vector<scmplx> cQmat_,cZmat_,cSmat_,cSpmat_;

public:
    void prepare_matrices(float freq,const Mesh &M);
    void compute_egn(const Mesh &M,float freq,
                    std::vector<float> &c,
                    std::vector<float> &ur,
                    std::vector<float> &ul,
                    bool save_qz=false);
    void compute_egn_att(const Mesh &M,float freq,
                        std::vector<scmplx> &c,
                        std::vector<scmplx> &ur,
                        std::vector<scmplx> &ul,
                        bool save_qz=false);

    // group velocity
    float group_vel(const Mesh &M,float freq,
                    float c,const float *ur,
                    const float *ul) const;
    scmplx group_vel_att(const Mesh &M,float freq,
                        scmplx c, const scmplx *ur,
                        const scmplx *ul) const;

    // phase velocity kernels
    void compute_phase_kl(const Mesh &M,float freq,
                        float c,const float *ur,
                        const float *ul,
                        std::vector<float> &frekl) const;
    void compute_phase_kl_att(const Mesh &M,float freq,
                        scmplx c, const scmplx *ur,
                        const scmplx *ul,
                        std::vector<float> &frekl_c,
                        std::vector<float> &frekl_q) const;

    // group velocity kernels
    void compute_group_kl(const Mesh &M,float freq,
                        float c,const float *ur,
                        const float *ul,
                        std::vector<float> &frekl) const;
    void compute_group_kl_att(const Mesh &M,float freq,
                        scmplx c, const scmplx *ur,
                        const scmplx *ul,
                        std::vector<float> &frekl_c,
                        std::vector<float> &frekl_q) const;


    // transforms
    void egn2displ(const Mesh &M,float freq,float c,
                    const float*egn, float * __restrict displ) const;
    void egn2displ_att(const Mesh &M,float freq,scmplx c,const scmplx *egn,
                      scmplx * __restrict displ) const;
    void transform_kernels(const Mesh &M,std::vector<float> &frekl) const;
};


} // namespace specswd




#endif
```

Index

attenuation.hpp, 28

compute_egn
    specswd::SolverLove, 20
    specswd::SolverRayl, 24
compute_egn_att
    specswd::SolverLove, 21
    specswd::SolverRayl, 24
compute_group_kl
    specswd::SolverLove, 21
compute_group_kl_att
    specswd::SolverLove, 21
compute_phase_kl
    specswd::SolverLove, 22
    specswd::SolverRayl, 25
compute_phase_kl_att
    specswd::SolverLove, 22
    specswd::SolverRayl, 25
create_database
    specswd::Mesh, 17
create_db_aniso_
    specswd::Mesh, 17
create_db_love_
    specswd::Mesh, 17
create_db_rayl_
    specswd::Mesh, 17

frechet_op.hpp, 30

get_cQ_kl
    specswd, 9
get_sls_modulus_factor
    specswd, 10
get_sls_Q_derivative
    specswd, 10
GQTable.hpp, 29

interp_model
    specswd::Mesh, 18
iofunc.hpp, 29

love_deriv_op_
    specswd, 10

mesh.hpp, 27

precision.hpp, 27
project_kl
    specswd::Mesh, 18

quadrature.hpp, 30

rayl_deriv_op_
    specswd, 11
read_model
    specswd::Mesh, 18
read_model_full_aniso_
    specswd::Mesh, 19
read_model_header_
    specswd::Mesh, 19
read_model_love_
    specswd::Mesh, 19
read_model_rayl_
    specswd::Mesh, 19

schur_qz
    specswd, 12
solve_christoffel
    specswd, 13
specswd, 7
    get_cQ_kl, 9
    get_sls_modulus_factor, 10
    get_sls_Q_derivative, 10
    love_deriv_op_, 10
    rayl_deriv_op_, 11
    schur_qz, 12
    solve_christoffel, 13
specswd::Mesh, 15
    create_database, 17
    create_db_aniso_, 17
    create_db_love_, 17
    create_db_rayl_, 17
    interp_model, 18
    project_kl, 18
    read_model, 18
    read_model_full_aniso_, 19
    read_model_header_, 19
    read_model_love_, 19
    read_model_rayl_, 19
specswd::SolverLove, 20
    compute_egn, 20
    compute_egn_att, 21
    compute_group_kl, 21
    compute_group_kl_att, 21
    compute_phase_kl, 22
    compute_phase_kl_att, 22
    transform_kernels, 23
specswd::SolverRayl, 23
    compute_egn, 24
    compute_egn_att, 24
    compute_phase_kl, 25
    compute_phase_kl_att, 25

transform_kernels, 25

transform_kernels
specswd::SolverLove, 23
specswd::SolverRayl, 25

vti.hpp, 34