

SpecSWD

Generated by Doxygen 1.9.8



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Namespace Documentation</b>	<b>7</b>
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
<b>5 Class Documentation</b>	<b>15</b>
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
<b>6 File Documentation . . . . .</b>	<b>27</b>
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
<b>Index . . . . .</b>	<b>37</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">specswd</a>	
Derivative operators: . . . . .	<a href="#">7</a>



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">specswd::Mesh</a>	15
<a href="#">specswd::SolverLove</a>	20
<a href="#">specswd::SolverRayl</a>	23





## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">precision.hpp</a>	27
<a href="#">mesh.hpp</a>	27
<a href="#">attenuation.hpp</a>	28
<a href="#">GQTable.hpp</a>	29
<a href="#">iofunc.hpp</a>	29
<a href="#">quadrature.hpp</a>	30
<a href="#">frechet_op.hpp</a>	30
<a href="#">vti.hpp</a>	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  $coef * y^H @ d((w^2 M - E) - k^2 K)/dm_i @ x dm_i$*

- `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)`  

$$\text{compute coef} * y^{\wedge} \text{dag} @ d( (w^{\wedge} 2 M - E) - k^{\wedge} 2 K) / dm_i @ x dm_i$$
- `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)`  

$$\text{get SLS } Q \text{ terms on the elastic modulus}$$
- `void get_sls_Q_derivative (float freq, float Q, crealw &s, crealw &dspdqi)`  

$$\text{Get the } Q \text{ factor and derivative for SLS model.}$$
- `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 > &dspdqi)`
- `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

<i>dcdm, Frechet</i>	kernel for complex phase velocity, rst m
<i>c</i>	complex phase velocity
<i>dcLdm, dQiLdm</i>	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

<i>freq</i>	current frequency
<i>Qa</i>	Qvalue
<i>xA</i>	

**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

<i>freq</i>	frequency
<i>Q</i>	current Q
<i>s</i>	modulus factor mu = mu * s
<i>dsdqi</i>	$Q^{-1}$ 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 \*  $y^H @ d((w^2 M - E) - k^2 K)/dm_i @ x dm_i$

#### Parameters

<i>freq</i>	current frequency
<i>c</i>	current phase velocity
<i>coef</i>	scaling coefs
<i>egn</i>	eigen vector, shape(nglob_el)
<i>nspec_el/nglob_el</i>	mesh nelemts/unique points for elastic
<i>ibool_el</i>	elastic connectivity matrix, shape(nspec_el*NGLL+NGRL)
<i>jaco</i>	jacobian matrix, shape (nspec_el + 1)
<i>xN/xL/xQN/xQL/rho</i>	model parameters, shape(nspec_el*NGLL+NGRL)
<i>frekl_c</i>	dc/d(N/L/rho) (elastic) or dc/d(N/L/QN/QL/rho) (anelstic)
<i>frekl_q</i>	nullptr or dq/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 \*  $y^{\text{dag}}$  @  $d((w^2 M - E) - k^2 K)/dm_i$  @  $x$   $dm_i$

#### Parameters

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

<i>ng</i>	rows/cols of A, B
<i>A,B</i>	two matrices, type = COMMTP
<i>Q,Z,S,SP</i>	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

<i>phi</i>	direction angle, in rad
<i>c21</i>	c21 tensor, shape(21)
<i>cmin/cmax</i>	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, 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](#) (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

<i>freq</i>	current frequency
<i>phi</i>	directional angle

### 5.1.1.2 `create_db_aniso_()`

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

create database for Love wave

#### Parameters

<i>freq</i>	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

<i>freq</i>	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

<i>freq</i>	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

<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)

#### 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

<i>frekl</i>	derivatives, shape(nspec*NGLL+NGRL)
<i>kl_out</i>	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

<i>filename</i>	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

<i>filename</i>	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, attenuation flag, attenuation model flag

## Parameters

<i>filename</i>	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

<i>filename</i>	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

<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

## 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

<i>freq</i>	current frequency
<i>c</i>	dispersion, shape(nc)
<i>egn</i>	eigen functions(displ at y direction), shape(nc,nglob_el)
<i>save_qz</i>	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

<i>freq</i>	current frequency
<i>c</i>	dispersion, shape(nc) $c = c_0(1 + iQL^{-1})$
<i>egn</i>	eigen functions(displ at y direction), shape(nc,nglob_el)
<i>save_qz</i>	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

<i>freq</i>	current frequency
<i>c</i>	current phase velocity
<i>displ</i>	eigen function, shape(nglob_el)
<i>frekl</i>	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

<i>freq</i>	current frequency
<i>c</i>	current complex phase velocity
<i>displ</i>	eigen function, shape(nglob_el)
<i>frekl</i> ↔ <i>_c</i>	$d\text{Re}(u)/d(N/L/QN/QL/\rho)$ shape(5,nspec*NGLL + NGRL)
<i>frekl</i> ↔ <i>_q</i>	$d(q)/d(N/L/QN/QL/\rho)$ 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

<i>freq</i>	current frequency
<i>c</i>	current phase velocity
<i>displ</i>	eigen function, shape(nglob_el)
<i>frekl</i>	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

<i>freq</i>	current frequency
-------------	-------------------

## Parameters

<i>c</i>	current complex phase velocity
<i>displ</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>	$d(q)/d(N/L/QN/QL/\rho)$ 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

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

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 specsrd::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

<i>freq</i>	current frequency
<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>save_qz</i>	if true, save QZ matrix

### 5.3.1.2 compute\_egn\_att()

```
void specsrd::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

<i>freq</i>	current frequency
<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>save_qz</i>	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

<i>freq</i>	current frequency
<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)

## 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

<i>freq</i>	current frequency
<i>c</i>	current phase velocity
<i>ur/ul</i>	right/left eigen function, shape(nglob_el*2+nglob_ac)
<i>frekl</i> ↔ <i>_c</i>	$d\text{Re}(c)/d(A/C/L/\eta/Q_a/Q_c/Q_l/\kappa/Q_k/\rho)$ kernels for elastic parameters, shape(10,nspec*NGLL + NGRL)
<i>frekl</i> ↔ <i>_q</i>	$d\text{Re}(Q_R)/d(A/C/L/\eta/Q_a/Q_c/Q_l/\kappa/Q_k/\rho)$ kernels for elastic parameters, shape(10,nspec*NGLL + NGRL)

### 5.3.1.5 transform\_kernels()

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

transform modulus kernel to velocity kernel, Rayleigh wave case

#### Parameters

<i>frekl</i>	frechet kernels, the shape depends on: <ul style="list-style-type: none"><li>• 1: elastic rayleigh wave: <math>A/C/L/\eta/\kappa/\rho \rightarrow v_{ph}/v_{pv}/v_{sv}/\eta/v_p/\rho</math></li><li>• 2 anelastic rayleigh wave: <math>A/C/L/\eta/QA_i/QC_i/QL_i/\kappa/Q_k/\rho \rightarrow v_{ph}/v_{pv}/v_{sv}/\eta/QA_i/QC_i/QL_i/v_p/Q_k/\rho</math></li></ul>
--------------	---

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

```
00001 #ifndef SPECSWD_PRECISION_H_
00002 #define SPECSWD_PRECISION_H_
00003
00004 #ifdef SPECSWD_EGN_DOUBLE
00005 typedef double realw;
00006 #define LAPACKE_REAL(name) LAPACKE_d ## name
00007 #define LAPACKE_CMPLX(name) LAPACKE_z ## name
00008 #define LCREALW lapack_complex_double
00009
00010 #else
00011 typedef float realw;
00012 #define LAPACKE_REAL(name) LAPACKE_s ## name
00013 #define LAPACKE_CMPLX(name) LAPACKE_c ## name
00014 #define LCREALW lapack_complex_float
00015 #endif
00016
00017 typedef std::complex<realw> crealw;
00018
00019
00020 #endif
```

### 6.2 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 specswwd
00009 {
00010
00011 struct Mesh {
00012
00013     // SEM Mesh
00014     int nspec,nspec_grl; // no. of elements for gll/grl layer
00015     int nglob; // no. of unique points
00016     std::vector<int> ibool; // connectivity matrix, shape(nspec * NGLL + NGRL)
00017     std::vector<float> skel; // skeleton, shape(nspec * 2 + 2)
00018     std::vector<float> znodes; // shape(nspec * NGLL + NGRL)
00019     std::vector<float> jaco; // jacobian for GLL, shape(nspec + 1) dz / dxi
00020     std::vector<float> zstore; // shape(nglob)
00021
00022     // element type for each medium
00023     int nspec_ac,nspec_el;
00024     int nspec_ac_grl,nspec_el_grl;
00025     std::vector<char> is_elastic, is_acoustic;
00026     std::vector<int> el_elmnts,ac_elmnts; // elements for each media, shape(nspec_? + nspec_?_grl)
00027
00028     // unique array for acoustic/elastic
00029     int nglob_ac, nglob_el;
00030     std::vector<int> ibool_el, ibool_ac; // connectivity matrix, shape shape(nspec_? + nspec_?_grl)
00031 }
```

```

00032 // density and elastic parameters
00033 std::vector<float> xrho_ac; // shape(nspec_ac * NGLL + nspec_ac_grl * NGRL)
00034 std::vector<float> xrho_el; // shape (nsepc_el * NGLL + nspec_el_grl * NGRL)
00035
00036 // attenuation/type flag
00037 bool HAS_ATT;
00038 int SWD_TYPE; // =0 Love wave, = 1 for Rayleigh = 2 full aniso
00039
00040 // vti media
00041 std::vector<float> xA,xC,xL,xeta,xN; // shape(nspec_el * NGLL+ nspec_el_grl * NGRL)
00042 std::vector<float> xQA,xQC,xQL,xQN; // shape(nspec_el * NGLL+ nspec_el_grl * NGRL), Q model
00043
00044 // full anisotropy
00045 int nQmodel_ani; // no. of Q used for anisotropy
00046 std::vector<float> xC2l; // shape(2l,nspec_el * NGLL+ nspec_el_grl * NGRL)
00047 std::vector<float> xQani; // shape(nQmodel_ani,nspec_el * NGLL+ nspec_el_grl * NGRL)
00048
00049 // fluid vti
00050 std::vector<float> xkappa_ac,xQk_ac;
00051
00052 // fluid-elastic boundary
00053 int nfaces_bdry;
00054 std::vector<int> ispec_bdry; // shape(nfaces_bdry,2) (i,:) = [ispec_ac,ispec_el]
00055 std::vector<char> bdry_norm_direc; // shape(nfaces_bdry), = 1 point from acoustic -> z direc
elastic
00056
00057 int nz_tomo, nregions;
00058 std::vector<float> rho_tomo;
00059 std::vector<float> vpv_tomo,vph_tomo,vsv_tomo,vsh_tomo,eta_tomo;
00060 std::vector<float> QC_tomo,QA_tomo,QL_tomo,QN_tomo;
00061 std::vector<float> c2l_tomo,Qani_tomo;
00062 std::vector<float> depth_tomo;
00063 std::vector<int> region_bdry; // shape(nregions,2)
00064 std::vector<int> iregion_flag; // shape(nspec + 1), return region flag
00065
00066 // interface with layered model
00067 std::vector<char> is_el_reg, is_ac_reg; // shape(nregions)
00068
00069 float PHASE_VELOC_MIN,PHASE_VELOC_MAX;
00070
00071 // public functions
00072 void read_model(const char *filename);
00073 void create_database(float freq,float phi);
00074 void print_model() const;
00075 void print_database() const;
00076 void allocate_1D_model(int nz0,int swd_type,int has_att);
00077 void create_model_attributes();
00078
00079 // interpolate model
00080 void interp_model(const float *param,const std::vector<int> &elmnts,std::vector<float> &md) const;
00081 void project_kl(const float *frekl, float *kl_out) const;
00082
00083 // private functions below
00084 // =====
00085 //
00086 void create_material_info_();
00087
00088 // 1-D model
00089 void read_model_header_(const char *filename);
00090 void read_model_love_(const char *filename);
00091 void read_model_rayl_(const char *filename);
00092 void read_model_full_aniso_(const char *filename);
00093
00094 // create SEM database
00095 void compute_minmax_veloc(float phi,std::vector<float> &vmin,std::vector<float> &vmax);
00096 void create_db_love_(float freq);
00097 void create_db_rayl_(float freq);
00098 void create_db_aniso_(float freq);
00099 };
00100
00101 } // namespace specswd
00102
00103
00104
00105
00106 #endif

```

## 6.3 attenuation.hpp

```

00001
00002 #ifndef SPECSWD_ATT_TABLE_H_
00003 #define SPECSWD_ATT_TABLE_H_
00004

```



```

00005 #include <complex>
00006
00007 namespace specswwd
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> &dsdq);
00016
00017 void set_C21_att_model(float freq, const float *Qm, int nQmodel,
00018                       std::complex<float>* __restrict c21,
00019                       int funcid=0, bool do_deriv=false);
00020
00021
00022 }
00023
00024 #endif

```

## 6.4 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> xgll, wgll;
00011 extern std::array<float, NGRL> xgrl, wgrl;
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

```

## 6.5 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 specsrd
00038
00039
00040 #endif

```

## 6.6 quadrature.hpp

```

00001 #ifndef SPECSWD_QUADRATURE_H_
00002 #define SPECSWD_QUADRATURE_H_
00003 #include <cmath>
00004
00005 //GLL
00006 void gauss_legendre_lobatto(double* knots, double* weights, size_t length);
00007 void lagrange_poly(double xi, size_t nctrl, const double *xctrl,
00008                 double *h, double* hprime);
00009
00010 // GRL
00011 void gauss_radau_laguerre(double *xgrl, double *wgrl, size_t length);
00012 double laguerre_func(size_t n, double x);
00013
00014 #endif

```

## 6.7 frechet\_op.hpp

```

00001 #ifndef SPECSWD_FRECHET_OP_H_
00002 #define SPECSWD_FRECHET_OP_H_
00003
00010 #include "shared/attenuation.hpp"
00011 #include "shared/GQTable.hpp"
00012
00013 namespace specsrd
00014 {
00015
00022 template <typename T> void
00023 get_cQ_kl(T &dcdm, T c,
00024          float &dcLdm, float &dQiLdm)
00025 {
00026     static_assert(std::is_same_v<std::complex<float>, T>);
00027     float cl = c.real();
00028     float Qi = 2. * c.imag() / cl;
00029     dcLdm = dcdm.real();
00030     dQiLdm = (dcdm.imag() * 2. - Qi * dcLdm) / cl;
00031 }
00032
00047 template<typename T = float >
00048 void love_deriv_op_(float freq, T c, T coef, const T *y, const T *x,
00049                   int nspec_el, int nglob_el, const int *ibool_el,
00050                   const float *jaco, const float *xN,
00051                   const float *xL, const float *xQN,
00052                   const float *xQL, float * __restrict frekl_c,
00053                   float * __restrict frekl_q)
00054 {
00055     // check template type
00056     static_assert(std::is_same_v<float, T> || std::is_same_v<std::complex<float>, T>);
00057
00058     using namespace GQTable;
00059     std::array<T, NGRL> rW, lW;
00060     size_t size = nspec_el * NGLL + NGRL;
00061     T om = 2 * M_PI * freq;
00062     T k2 = (om * om) / (c * c);
00063     for(int ispec = 0; ispec < nspec_el + 1; ispec++) {
00064         const float *hp = &hprime[0];
00065         const float *w = &wgl1[0];
00066         float J = jaco[ispec]; // jacobians in this layers
00067         int NGL = NGLL;
00068         int id = ispec * NGLL;
00069
00070         // GRL layer
00071         if(ispec == nspec_el) {
00072             hp = &hprime_grl[0];
00073             w = &wgrl[0];
00074             NGL = NGRL;
00075         }
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 * om * om * rW[m] * lW[m] * coef;
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 = -k2 * rW[m] * lW[m] * J * w[m] * coef;
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 * coef;
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_cQ_kl(dc_dN,c,frekl_c[0*size+idl],frekl_q[0*size+idl]);
00122             get_cQ_kl(dc_dL,c,frekl_c[1*size+idl],frekl_q[1*size+idl]);
00123             get_cQ_kl(dc_dqni,c,frekl_c[2*size+idl],frekl_q[2*size+idl]);
00124             get_cQ_kl(dc_dqli,c,frekl_c[3*size+idl],frekl_q[3*size+idl]);
00125             get_cQ_kl(dc_drho,c,frekl_c[4*size+idl],frekl_q[4*size+idl]);
00126         }
00127         else {
00128             frekl_c[0*size+idl] = dc_dN;
00129             frekl_c[1*size+idl] = dc_dL;
00130             frekl_c[2*size+idl] = dc_drho;
00131         }
00132     }
00133 }
00134 }
00135
00154 template<typename T = float >
00155 void
00156 rayl_deriv_op_(float freq,T c,T coef,const T *y, const T *x,
00157     int nspec_el,int nspec_ac,int nspec_el_grl,int nspec_ac_grl,int nglob_el,
00158     int nglob_ac, const int *el_elmnts,const int *ac_elmnts,
00159     const int* ibool_el, const int* ibool_ac,
00160     const float *jaco,const float *xrho_el,const float *xrho_ac,
00161     const float *xA, const float *xQ,const float *xL,const float *xeta,
00162     const float *xQA, const float *xQC,const float *xQL,
00163     const float *xkappa_ac, const float *xQk_ac,
00164     float *__restrict frekl_c,
00165     float *__restrict frekl_q)
00166 {
00167     // check template type
00168     static_assert(std::is_same_v<float,T> || std::is_same_v<std::complex<float>,T>);
00169
00170     // constants
00171     using namespace GQTable;
00172     size_t size = nspec_el * NGLL + nspec_el_grl * NGRL +
00173         nspec_ac * NGLL + nspec_ac_grl * NGRL;
00174     T om = 2 * M_PI * freq;
00175     T k2 = std::pow(om / c,2);
00176
00177     // loop elastic elements
00178     std::array<T,NGRL> U,V,lU,lV;
00179     for(int ispec = 0; ispec < nspec_el + nspec_el_grl; ispec++) {
00180         int iel = el_elmnts[ispec];
00181         int id = ispec * NGLL;
00182
00183         const float *weight = wgl1.data();

```

```

00184     const float *hp = hprime.data();
00185     int NGL = NGLL;
00186
00187     // jacobian
00188     float J = jaco[iel];
00189
00190     // grl case
00191     if(ispec == nspec_el) {
00192         weight = wgrl.data();
00193         hp = hprime_grl.data();
00194         NGL = NGRL;
00195     }
00196
00197     // cache U,V and lU,lV
00198     for(int i = 0; i < NGL; i++) {
00199         int iglob = ibool_el[id + i];
00200         U[i] = x[iglob];
00201         V[i] = x[iglob + nglob_el];
00202         lU[i] = y[iglob];
00203         lV[i] = y[iglob + nglob_el];
00204         if constexpr (std::is_same_v<T, std::complex<float> >) {
00205             lU[i] = std::conj(lU[i]);
00206             lV[i] = std::conj(lV[i]);
00207         }
00208     }
00209
00210     // compute kernel
00211     T dc_drho{}, dc_dA{}, dc_dC{}, dc_dL{};
00212     T dc_deta{}, dc_dQci{}, dc_dQai{}, dc_dQli{};
00213     const T two = 2.;
00214     for(int m = 0; m < NGL; m++) {
00215         T temp = weight[m] * J * coef;
00216         dc_drho = temp * om * om *
00217             (U[m] * lU[m] + V[m] * lV[m]);
00218
00219         // get sls factor if required
00220         T sa = 1., sl = 1., sc = 1.;
00221         T dsdqai{}, dsdqci{}, dsdqli{};
00222         float C = xC[id+m], A = xA[id+m],
00223             L = xL[id+m], eta = xeta[m];
00224         if constexpr (std::is_same_v<T, std::complex<float> >) {
00225             get_sls_Q_derivative(freq, xQA[id+m], sa, dsdqai);
00226             get_sls_Q_derivative(freq, xQC[id+m], sc, dsdqai);
00227             get_sls_Q_derivative(freq, xQL[id+m], sl, dsdqai);
00228             dsdqai *= A;
00229             dsdqci *= C;
00230             dsdqli *= L;
00231         }
00232
00233         // K matrix
00234         // dc_dA
00235         temp = -weight[m] * J * k2 * U[m] * lU[m] * coef;
00236         dc_dA = temp * sa; dc_dQai = temp * dsdqai;
00237
00238         // dc_dL
00239         temp = -weight[m] * J * k2 * V[m] * lV[m] * coef;
00240         dc_dL = temp * sl; dc_dQli = temp * dsdqli;
00241
00242         // Ematrix
00243         T sx{}, sy{}, lsx{}, lsy{};
00244         for(int i = 0; i < NGL; i++) {
00245             sx += hp[m*NGL+i] * U[i];
00246             sy += hp[m*NGL+i] * V[i];
00247             lsx += hp[m*NGL+i] * lU[i];
00248             lsy += hp[m*NGL+i] * lV[i];
00249         }
00250         temp = -weight[m] / J * sx * lsx * coef;
00251         dc_dL += temp * sl; dc_dQli += temp * dsdqli;
00252
00253         temp = -weight[m] / J * sy * lsy * coef;
00254         dc_dC = temp * sc; dc_dQci = temp * dsdqci;
00255
00256         // eta
00257         temp = - (k2 * weight[m] * U[m] * lsy +
00258             weight[m] * lU[m] * sy) * coef;
00259         dc_deta = temp * (A*sa - two*L*sl);
00260
00261         temp *= eta;
00262         dc_dA += temp * sa; dc_dQai += temp * dsdqai;
00263         dc_dL += -temp * two * sl; dc_dQli += -temp * two * dsdqli;
00264
00265         temp = k2 * weight[m] * lV[m] * sx + weight[m] * V[m] * lsx;
00266         temp *= coef;
00267         dc_dL += temp * sl;
00268         dc_dQli += temp * dsdqli;
00269
00270         // copy them to frekl

```

```

00271         int idl = iel * NGLL + m;
00272         if constexpr (std::is_same_v<T,float>) {
00273             frekl_c[0*size+idl] = dc_dA;
00274             frekl_c[1*size+idl] = dc_dC;
00275             frekl_c[2*size+idl] = dc_dL;
00276             frekl_c[3*size+idl] = dc_deta;
00277             frekl_c[5*size+idl] = dc_drho;
00278         }
00279         else {
00280             get_cQ_kl(dc_dA,c,frekl_c[0*size+idl],frekl_q[0*size+idl]);
00281             get_cQ_kl(dc_dC,c,frekl_c[1*size+idl],frekl_q[1*size+idl]);
00282             get_cQ_kl(dc_dL,c,frekl_c[2*size+idl],frekl_q[2*size+idl]);
00283             get_cQ_kl(dc_deta,c,frekl_c[3*size+idl],frekl_q[3*size+idl]);
00284             get_cQ_kl(dc_dQai,c,frekl_c[4*size+idl],frekl_q[4*size+idl]);
00285             get_cQ_kl(dc_dQci,c,frekl_c[5*size+idl],frekl_q[5*size+idl]);
00286             get_cQ_kl(dc_dQli,c,frekl_c[6*size+idl],frekl_q[6*size+idl]);
00287             get_cQ_kl(dc_drho,c,frekl_c[9*size+idl],frekl_q[9*size+idl]);
00288         }
00289     }
00290 }
00291
00292 // acoustic eleemnts
00293 std::array<T,NGRL> chi,lchi;
00294 for(int ispec = 0; ispec < nspec_ac + nspec_ac_grl; ispec++) {
00295     int iel = ac_elmnts[ispec];
00296     int id = ispec * NGLL;
00297     const float *weight = wgl1.data();
00298     const float *hp = hprime.data();
00299     int NGL = NGLL;
00300
00301     // jacobians
00302     float J = jaco[iel];
00303
00304     // grl case
00305     if(ispec == nspec_ac) {
00306         weight = wgrl.data();
00307         hp = hprime_grl.data();
00308         NGL = NGRL;
00309     }
00310
00311     // cache chi and lchi in one element
00312     for(int i = 0; i < NGL; i++) {
00313         int iglob = ibool_ac[id + i];
00314         chi[i] = (iglob == -1) ? 0: x[iglob+nglob_el*2];
00315         lchi[i] = (iglob == -1) ? 0.: y[iglob+nglob_el*2];
00316         if constexpr (std::is_same_v<T,std::complex<float>)> {
00317             lchi[i] = std::conj(lchi[i]);
00318         }
00319     }
00320
00321     // derivatives
00322     T dc_dkappa{},dc_drho{}, dc_dqki{};
00323     T sk = 1., dskdqi = 0.;
00324     for(int m = 0; m < NGL; m++) {
00325         // copy material
00326         float rho = xrho_ac[id+m];
00327         float kappa = xkappa_ac[id+m];
00328         if constexpr (std::is_same_v<T,std::complex<float>)> {
00329             get_sls_Q_derivative(freq,xQk_ac[id+m],sk,dskdqi);
00330             dskdqi *= kappa;
00331         }
00332
00333         // kappa kernel
00334         T temp = std::pow(om/(sk * kappa),2) * weight[m] * J *
00335             chi[m] * lchi[m] * coef;
00336         dc_dkappa = temp * sk;
00337         dc_dqki = temp * dskdqi;
00338
00339         dc_drho = -k2 * std::pow(om/rho,2) * weight[m] * J *
00340             chi[m] * lchi[m] * coef;
00341
00342         T sx{},sy{};
00343         for(int i = 0; i < NGL; i++) {
00344             sx += hp[m*NGL+i] * chi[i];
00345             sy += hp[m*NGL+i] * lchi[i];
00346         }
00347         dc_drho += weight[m] / J / (rho*rho) * sx * sy * coef;
00348
00349         // copy to frekl
00350         int idl = iel * NGLL + m;
00351         if constexpr (std::is_same_v<T,float>)> {
00352             frekl_c[4*size+idl] = dc_dkappa;
00353             frekl_c[5*size+idl] = dc_drho;
00354         }
00355         else {
00356             get_cQ_kl(dc_dkappa,c,frekl_c[7*size+idl],frekl_q[7*size+idl]);
00357             get_cQ_kl(dc_dqki,c,frekl_c[8*size+idl],frekl_q[8*size+idl]);

```

```

00358         get_cQ_kl(dc_drho,c,frekl_c[9*size+idl],frekl_q[9*size+idl]);
00359     }
00360
00361     }
00362 }
00363 }
00364
00365
00366 } // namespace specswwd
00367
00368 #endif

```

## 6.8 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
00028     void prepare_matrices(float freq,const Mesh &M);
00029     void compute_egn(const Mesh &M,float freq,
00030         std::vector<float> &c,
00031         std::vector<float> &egn,
00032         bool save_qz=false);
00033     void compute_egn_att(const Mesh &M,float freq,
00034         std::vector<scmplx> &c,
00035         std::vector<scmplx> &egn,
00036         bool save_qz=false);
00037
00038     // group velocity
00039     float group_vel(const Mesh &M,float freq,
00040         float c,const float *egn) const;
00041     scmplx group_vel_att(const Mesh &M,float freq,
00042         scmplx c, const scmplx *egn) const ;
00043
00044     // phase velocity kernels
00045     void compute_phase_kl(const Mesh &M,float freq,
00046         float c,const float *egn,
00047         std::vector<float> &frekl) const;
00048     void compute_phase_kl_att(const Mesh &M,float freq,
00049         scmplx c, const scmplx *egn,
00050         std::vector<float> &frekl_c,
00051         std::vector<float> &frekl_q) const;
00052
00053     // group kernel
00054     void compute_group_kl(const Mesh &M,float freq,
00055         float c,const float *egn,
00056         std::vector<float> &frekl) const;
00057
00058     void compute_group_kl_att(const Mesh &M,float freq,
00059         scmplx c, const scmplx *egn,
00060         std::vector<float> &frekl_c,
00061         std::vector<float> &frekl_q) const;
00062
00063     // transforms
00064     void egn2displ(const Mesh &M,float freq,float c,
00065         const float*egn, float * __restrict displ) const;
00066     void egn2displ_att(const Mesh &M,float freq,scmplx c,const scmplx *egn,
00067         scmplx * __restrict displ) const;
00068     void transform_kernels(const Mesh &M,std::vector<float> &frekl) const;
00069 };

```

```

00070 class SolverRayl {
00071
00072 private:
00073     // solver matrices
00074     std::vector<float> Mmat,Emat,Kmat;
00075     std::vector<scmplx> CMmat,CEmat,CKmat;
00076
00077     // QZ matrix all are column major
00078     std::vector<float> Qmat_,Zmat_,Smat_,Spmat_; // column major!
00079     std::vector<scmplx> cQmat_,cZmat_,cSmat_,cSpmat_;
00080
00081 public:
00082     void prepare_matrices(float freq,const Mesh &M);
00083     void compute_egn(const Mesh &M,float freq,
00084         std::vector<float> &c,
00085         std::vector<float> &ur,
00086         std::vector<float> &ul,
00087         bool save_qz=false);
00088     void compute_egn_att(const Mesh &M,float freq,
00089         std::vector<scmplx> &c,
00090         std::vector<scmplx> &ur,
00091         std::vector<scmplx> &ul,
00092         bool save_qz=false);
00093
00094     // group velocity
00095     float group_vel(const Mesh &M,float freq,
00096         float c,const float *ur,
00097         const float *ul) const;
00098     scmplx group_vel_att(const Mesh &M,float freq,
00099         scmplx c, const scmplx *ur,
00100         const scmplx *ul) const;
00101
00102     // phase velocity kernels
00103     void compute_phase_kl(const Mesh &M,float freq,
00104         float c,const float *ur,
00105         const float *ul,
00106         std::vector<float> &frekl) const;
00107     void compute_phase_kl_att(const Mesh &M,float freq,
00108         scmplx c, const scmplx *ur,
00109         const scmplx *ul,
00110         std::vector<float> &frekl_c,
00111         std::vector<float> &frekl_q) const;
00112
00113     // group velocity kernels
00114     void compute_group_kl(const Mesh &M,float freq,
00115         float c,const float *ur,
00116         const float *ul,
00117         std::vector<float> &frekl) const;
00118     void compute_group_kl_att(const Mesh &M,float freq,
00119         scmplx c, const scmplx *ur,
00120         const scmplx *ul,
00121         std::vector<float> &frekl_c,
00122         std::vector<float> &frekl_q) const;
00123
00124
00125     // transforms
00126     void egn2displ(const Mesh &M,float freq,float c,
00127         const float*egn, float * __restrict displ) const;
00128     void egn2displ_att(const Mesh &M,float freq,scmplx c,const scmplx *egn,
00129         scmplx * __restrict displ) const;
00130     void transform_kernels(const Mesh &M,std::vector<float> &frekl) const;
00131 };
00132
00133
00134 } // namespace specsrd
00135
00136
00137
00138
00139 #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](#)