SWDTTI

Generated by Doxygen 1.12.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

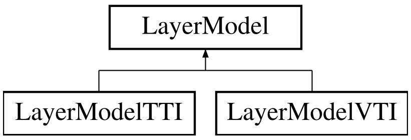
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 LayerModel Class Reference

Inheritance diagram for LayerModel:



Public Member Functions

- void **initialize** ()

  initialize GLL/GRL nodes/weights
- void interp_model (const float ∗z, const float ∗param, std::vector< double > &md) const

  interpolate a model by using coordinates
- void create_mesh (const int ∗nel, const float ∗thk, const float ∗zlist, int nlayer, double scale)

  Create SEM mesh by using input model info.
- void project_kl (const float ∗z, const double ∗param_kl, double ∗kl_out) const

  project SEM-type kernels to original model

Public Attributes

- std::array< double, NGLL > **xgll**
- std::array< double, NGLL > **wgll**
- std::array< double, NGRL > **xgrl**
- std::array< double, NGRL > **wgrl**
- std::array< double, NGLL ∗NGLL > **hprimeT**
- std::array< double, NGLL ∗NGLL > **hprime**
- std::array< double, NGRL ∗NGRL > **hprimeT_grl**
- std::array< double, NGRL ∗NGRL > **hprime_grl**
- int **nspec**
- int **nspec_grl**
- int **nglob**

- std::vector< int > **ibool**
- std::vector< float > **skel**
- std::vector< double > **znodes**
- std::vector< double > **jaco**
- std::vector< double > **zstore**
- bool **IS_DICON_MODEL**
- std::vector< int > **ilayer_flag**
- double **PHASE_VELOC_MIN**
- double **PHASE_VELOC_MAX**

Static Public Attributes

- static const int **NGLL** = 7
- static const int **NGRL** = 20

4.1.1 Member Function Documentation

4.1.1.1 create_mesh()

```
void LayerModel::create_mesh (
             const int * nel,
             const float * thk,
             const float * zlist,
             int nlayer,
             double scale)
```

Create SEM mesh by using input model info.

Parameters

nel	no. of elements for each layer, shape(nlayer - 1)
thk	thickness of each layer, shape(nlayer)
zlist	cumsum(thk)
nlayer	no. of layers
scale	scale factor for GRL layer, zbot = sum(thk) + xgrl[-1] ∗ scale

4.1.1.2 interp_model()

```
void LayerModel::interp_model (
             const float * z,
             const float * param,
             std::vector< double > & md) const
```

interpolate a model by using coordinates

Parameters

z	input model z coordinates, shape(nlayer)
param	input model parameter, shape(nlayer)
md	model required to interpolate, shape(nspec∗NGLL + NGRL)

Returns

float

4.1.1.3 project_kl()

```
void LayerModel::project_kl (
            const float * z,
            const double * param_kl,
            double * kl_out) const
```

project SEM-type kernels to original model

Parameters

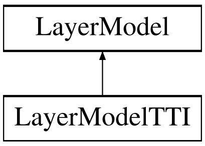
z	z coordiantes of previous model, shape(nlayer)
param↩ _kl	SEM typed kernel, shape(nspec ∗ NGLL + NGRL)
kl_out	kernels in original model, shape(nlayer)

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

- swdlayer.hpp
- initialize.cpp

4.2 LayerModelTTI Class Reference

Inheritance diagram for LayerModelTTI:



Public Member Functions

- void create_database (double freq, int nlayer, const float ∗vph, const float ∗vpv, const float ∗vsh, const float ∗vsv, const float ∗eta, const float ∗theta0, const float ∗phi0, const float ∗rho, const float ∗thk, bool is_layer)

  initalize SEM mesh and create a TTI database from a layered model
- void prepare_matrices (double phi)

  prepare M/K1/K2/E matrices for TTI model
- void compute_egnfun (double freq, double phi, std::vector< double > &c, std::vector< dcmplx > &displ) const

  compute phase velocity and eigen displacements for a given direction
- std::array< double, 2 > compute_kernels (double freq, double c, double phi, const std::vector< dcmplx > &displ, std::vector< double > &frekl) const

  compute group velocity and kernels for tti model
- void transform_kernels (std::vector< double > &frekl) const

  transform kernels from base to vp/vs/eta/rho/phi/theta

Public Member Functions inherited from LayerModel

- void **initialize** ()

  initialize GLL/GRL nodes/weights
- void interp_model (const float ∗z, const float ∗param, std::vector< double > &md) const

  interpolate a model by using coordinates
- void create_mesh (const int ∗nel, const float ∗thk, const float ∗zlist, int nlayer, double scale)

  Create SEM mesh by using input model info.
- void project_kl (const float ∗z, const double ∗param_kl, double ∗kl_out) const

  project SEM-type kernels to original model

Public Attributes

- std::vector< double > **xrho**
- std::vector< double > **xA**
- std::vector< double > **xC**
- std::vector< double > **xL**
- std::vector< double > **xF**
- std::vector< double > **xN**
- std::vector< double > **xT**
- std::vector< double > **xP**

Public Attributes inherited from LayerModel

- std::array< double, NGLL > **xgll**
- std::array< double, NGLL > **wgll**
- std::array< double, NGRL > **xgrl**
- std::array< double, NGRL > **wgrl**
- std::array< double, NGLL ∗NGLL > **hprimeT**
- std::array< double, NGLL ∗NGLL > **hprime**
- std::array< double, NGRL ∗NGRL > **hprimeT_grl**
- std::array< double, NGRL ∗NGRL > **hprime_grl**
- int **nspec**
- int **nspec_grl**
- int **nglob**
- std::vector< int > **ibool**
- std::vector< float > **skel**
- std::vector< double > **znodes**
- std::vector< double > **jaco**
- std::vector< double > **zstore**
- bool **IS_DICON_MODEL**
- std::vector< int > **ilayer_flag**
- double **PHASE_VELOC_MIN**
- double **PHASE_VELOC_MAX**

Additional Inherited Members

Static Public Attributes inherited from LayerModel

- static const int **NGLL** = 7
- static const int **NGRL** = 20

4.2.1 Member Function Documentation

4.2.1.1 compute_egnfun()

```
void LayerModelTTI::compute_egnfun (
            double freq,
            double phi,
            std::vector< double > & c,
            std::vector< dcmplx > & displ) const
```

compute phase velocity and eigen displacements for a given direction

Parameters

freq	current frequency
phi	current direction, in deg
c	phase velocity
displ	displacement

4.2.1.2 compute_kernels()

```
std::array< double, 2 > LayerModelTTI::compute_kernels (
            double freq,
            double c,
            double phi,
            const std::vector< dcmplx > & displ,
            std::vector< double > & frekl) const
```

compute group velocity and kernels for tti model

Parameters

freq	current frequency
c	phase velocity at this frequency
phi	azimuthal angle of c
displ	eigen function, shape(nglob $*$ 3)
frekl	Frechet kernels A/C/F/L/N/T/P/rho_kl kernels for elastic parameters, shape(8,nspec$*$NGLL + NGRL)

Returns

double u group velocity and it's azimthual angle

4.2.1.3 create_database()

```
void LayerModelTTI::create_database (
            double freq,
            int nlayer,
            const float * vph,
            const float * vpv,
            const float * vsh,
            const float * vsv,
            const float * eta,
            const float * theta0,
            const float * phi0,
            const float * rho,
            const float * thk,
            bool is_layer)
```

initalize SEM mesh and create a TTI database from a layered model

Parameters

freq	current frequency
nlayer	# of nlayers
vpv/vph/vsv/vsh/eta/rho	layer model vti parameters, shape(nlayer)
theta0/phi0	axis direction

4.2.1.4 prepare_matrices()

```
void LayerModelTTI::prepare_matrices (
            double phi)
```

prepare M/K1/K2/E matrices for TTI model

Parameters

phi	polar angle of k vector, in deg

4.2.1.5 transform_kernels()

```
void LayerModelTTI::transform_kernels (
            std::vector< double > & frekl) const
```

transform kernels from base to vp/vs/eta/rho/phi/theta

Parameters

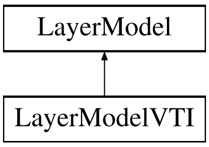
frekl	base Frechet kernels, shape(8,nspec*NGLL+NGRL)

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

- swdlayertti.hpp
- sem_tti.cpp
- swdlayertti.cpp
- tti_kernels.cpp

4.3 LayerModelVTI Class Reference

Inheritance diagram for LayerModelVTI:



Public Member Functions

- void create_database (double freq, int nlayer, const float ∗vph, const float ∗vpv, const float ∗vsh, const float ∗vsv, const float ∗eta, const float ∗rho, const float ∗thk, bool is_layer)

  initalize SEM mesh and create a VTI database from a layered model
- void prepare_matrices (int wavetype)

  prepare M/K/E matrices
- void compute_slegn (double freq, std::vector< double > &c, std::vector< double > &displ) const

  compute Love wave dispersion and eigenfunctions
- void compute_sregn (double freq, std::vector< double > &c, std::vector< double > &displ) const

  compute Love wave dispersion and eigenfunctions
- double compute_love_kl (double freq, double c, const double ∗displ, std::vector< double > &frekl) const

  compute group velocity and kernels for love wave
- double compute_rayl_kl (double freq, double c, const double ∗displ, std::vector< double > &frekl) const

  compute group velocity and kernels for love wave
- void transform_kernels (std::vector< double > &frekl) const

  transform kernels from base to vp/vs/eta/rho

Public Member Functions inherited from LayerModel

- void **initialize** ()

  initialize GLL/GRL nodes/weights
- void interp_model (const float ∗z, const float ∗param, std::vector< double > &md) const

  interpolate a model by using coordinates
- void create_mesh (const int ∗nel, const float ∗thk, const float ∗zlist, int nlayer, double scale)

  Create SEM mesh by using input model info.
- void project_kl (const float ∗z, const double ∗param_kl, double ∗kl_out) const

  project SEM-type kernels to original model

Public Attributes

- std::vector< double > **xrho**
- std::vector< double > **xA**
- std::vector< double > **xC**
- std::vector< double > **xL**
- std::vector< double > **xF**
- std::vector< double > **xN**

Public Attributes inherited from LayerModel

- std::array< double, NGLL > **xgll**
- std::array< double, NGLL > **wgll**
- std::array< double, NGRL > **xgrl**
- std::array< double, NGRL > **wgrl**
- std::array< double, NGLL *NGLL > **hprimeT**
- std::array< double, NGLL *NGLL > **hprime**
- std::array< double, NGRL *NGRL > **hprimeT_grl**
- std::array< double, NGRL *NGRL > **hprime_grl**
- int **nspec**
- int **nspec_grl**
- int **nglob**
- std::vector< int > **ibool**
- std::vector< float > **skel**
- std::vector< double > **znodes**
- std::vector< double > **jaco**
- std::vector< double > **zstore**
- bool **IS_DICON_MODEL**
- std::vector< int > **ilayer_flag**
- double **PHASE_VELOC_MIN**
- double **PHASE_VELOC_MAX**

Additional Inherited Members

Static Public Attributes inherited from LayerModel

- static const int **NGLL** = 7
- static const int **NGRL** = 20

4.3.1 Member Function Documentation

4.3.1.1 compute_love_kl()

```
double LayerModelVTI::compute_love_kl (
            double freq,
            double c,
            const double * displ,
            std::vector< double > & frekl) const
```

compute group velocity and kernels for love wave

Parameters

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

Returns

double u group velocity

4.3.1.2 compute_rayl_kl()

```
double LayerModelVTI::compute_rayl_kl (
             double freq,
             double c,
             const double * displ,
             std::vector< double > & frekl) const
```

compute group velocity and kernels for love wave

Parameters

freq	current frequency
c	current phase velocity
displ	eigen function, shape(nglob ∗ 2)
frekl	Frechet kernels A/C/L/F/rho_kl kernels for elastic parameters, shape(5,nspec∗NGLL + NGRL)

Returns

double u group velocity

4.3.1.3 compute_slegn()

```
void LayerModelVTI::compute_slegn (
             double freq,
             std::vector< double > & c,
             std::vector< double > & displ) const
```

compute Love wave dispersion and eigenfunctions

Parameters

freq	current frequency
vmin,vmax	min/max velocity for your model
c	dispersion, shape(nc)
displ	eigen functions(displ at y direction), shape(nc,nglob)

4.3.1.4 compute_sregn()

```
void LayerModelVTI::compute_sregn (
             double freq,
             std::vector< double > & c,
             std::vector< double > & displ) const
```

compute Love wave dispersion and eigenfunctions

Parameters

freq	current frequency
vmin,vmax	min/max velocity for your model
c	dispersion, shape(nc)
displ	eigen functions(displ at x/z direction), shape(nc,2,nglob)

4.3.1.5 create_database()

```
void LayerModelVTI::create_database (
            double freq,
            int nlayer,
            const float * vph,
            const float * vpv,
            const float * vsh,
            const float * vsv,
            const float * eta,
            const float * rho,
            const float * thk,
            bool is_layer)
```

initalize SEM mesh and create a VTI database from a layered model

Parameters

freq	current frequency
nlayer	# of nlayers
vpv/vph/vsv/vsh/eta/rho	layer model vti parameters, shape(nlayer)
is_layer	if the input model is a layered (discontinuous) model

4.3.1.6 prepare_matrices()

```
void LayerModelVTI::prepare_matrices (
            int wavetype)
```

prepare M/K/E matrices

Parameters

wavetype	= 1 for Love = 2 for Rayleigh

4.3.1.7 transform_kernels()

```
void LayerModelVTI::transform_kernels (
            std::vector< double > & frekl) const
```

transform kernels from base to vp/vs/eta/rho

Parameters

frekl	base Frechet kernels, shape(3/5,nspec∗NGLL+NGRL)

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

- swdlayervti.hpp
- sem_vti.cpp
- swdlayervti.cpp
- vti_kernels.cpp

Chapter 5

File Documentation

5.1 quadrature.hpp

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

//GLL
void gauss_legendre_lobatto(double* knots, double* weights, int length);
void lagrange_poly(double xi,int 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
```

5.2 swdio.hpp

```
#include <iostream>

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) {
        fprintf(stderr,"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);
}
```

5.3 swdlayer.hpp

```
#ifndef SWDLAYER_MODEL
```

```
#define SWDLAYER_MODEL

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

class LayerModel {

public:
    // GLL/GRL nodes and weights
    static const int NGLL = 7, NGRL = 20;
    std::array<double,NGLL> xgll,wgll;
    std::array<double,NGRL> xgrl,wgrl;
    std::array<double,NGLL*NGLL> hprimeT,hprime; // hprimeT(i,j) = l'_i(xi_j)
    std::array<double,NGRL*NGRL> hprimeT_grl,hprime_grl;

private:
    void initialize_nodes();

public:
    // SEM Mesh
    int nspec,nspec_grl; // # of elements for gll/grl layer
    int nglob; // # of unique points
    std::vector<int> ibool; // connectivity matrix, shape(nspec * NGLL + NGRL)
    std::vector<float> skel;  // skeleton, shape(nspec * 2 + 2)
    std::vector<double> znodes; // shape(nspec * NGLL + NGRL)
    std::vector<double> jaco; // jacobian for GLL, shape(nspec + 1) dz / dxi
    std::vector<double> zstore; // shape(nglob)

public:
    bool IS_DICON_MODEL;
    std::vector<int> ilayer_flag; // shape(nspec + 1), return layer flag
    double PHASE_VELOC_MIN,PHASE_VELOC_MAX;

//functions
public:
    LayerModel(){};
    void initialize();
    void interp_model(const float *z,const float *param,std::vector<double> &md) const;
    void create_mesh(const int *nel, const float *thk,const float *zlist,int nlayer,double scale);
    void project_kl(const float *z, const double *param_kl, double *kl_out) const;
};

#endif
```

5.4 swdlayertti.hpp

```
#ifndef SWD_LAYER_TTI_MODEL
#define SWD_LAYER_TTI_MODEL

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

#include "swdlayer.hpp"

class LayerModelTTI : public LayerModel{

typedef std::complex<double> dcmplx;
public:

    LayerModelTTI(){};

private:
    std::vector<dcmplx> Mmat,Emat,K1mat,K2mat; // matrices for SEM,shape(3*nglob,3*nglob) om^2 M = k^2 K_2 + k K_1 + E

public:

    // density
    std::vector<double> xrho;

    // tti Love parameters A,C,L,F,N, theta,phi
    std::vector<double> xA,xC,xL,xF,xN; // shape(nspec * NGLL + NGRL)
    std::vector<double> xT,xP; // theta/phi, shape(nspec *NGLL + NGRL), in rad


    // VTI model
    void create_database(double freq,int nlayer, const float *vph, const float* vpv,
                         const float *vsh, const float *vsv, const float *eta,
                         const float *theta0, const float *phi0,
                         const float *rho,const float *thk,bool is_layer);

    void prepare_matrices(double phi);

    void compute_egnfun(double freq, double phi, std::vector<double> &c, std::vector<dcmplx> &displ) const;
    std::array<double,2>
    compute_kernels(double freq, double c,double phi,
                    const std::vector<dcmplx> &displ,
                    std::vector<double> &frekl) const;

    void transform_kernels(std::vector<double> &frekl) const;
};

#endif
```

5.5 swdlayervti.hpp

```
#ifndef SWDLAYER_VTI_MODEL
#define SWDLAYER_VTI_MODEL

#include "swdlayer.hpp"

class LayerModelVTI: public LayerModel  {

public:
    LayerModelVTI(){};

private:
    std::vector<double> Mmat,Emat,Kmat; // matrices for SEM, om^2 M = k^2 K + E

public:

    // density
    std::vector<double> xrho;

    // vti Love parameters
    std::vector<double> xA,xC,xL,xF,xN; // shape(nspec * NGLL + NGRL)

    // VTI model
    void create_database(double freq,int nlayer, const float *vph, const float* vpv,
                        const float *vsh, const float *vsv, const float *eta,
                        const float *rho,const float *thk, bool is_layer);

    void prepare_matrices(int wavetype);

    void compute_slegn(double freq,std::vector<double> &c,
                         std::vector<double> &displ) const;
    void compute_sregn(double freq,std::vector<double> &c,
                        std::vector<double> &displ) const;

    double compute_love_kl(double freq,double c,const double *displ, std::vector<double> &frekl) const;
    double compute_rayl_kl(double freq,double c,const double *displ, std::vector<double> &frekl) const;

    void transform_kernels(std::vector<double> &frekl) const;

private:
    void prepare_matrices_love();
    void prepare_matrices_rayl();
};

#endif
```

Index