

SWDTTI

Generated by Doxygen 1.12.0

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 LayerModel Class Reference	7
4.1.1 Member Function Documentation	8
4.1.1.1 create_mesh()	8
4.1.1.2 interp_model()	8
4.1.1.3 project_kl()	9
4.2 LayerModelTTI Class Reference	9
4.2.1 Member Function Documentation	11
4.2.1.1 compute_egnfun()	11
4.2.1.2 compute_kernels()	11
4.2.1.3 create_database()	12
4.2.1.4 prepare_matrices()	12
4.2.1.5 transform_kernels()	12
4.3 LayerModelVTI Class Reference	13
4.3.1 Member Function Documentation	14
4.3.1.1 compute_love_kl()	14
4.3.1.2 compute_rayl_kl()	15
4.3.1.3 compute_slegn()	15
4.3.1.4 compute_sregn()	15
4.3.1.5 create_database()	16
4.3.1.6 prepare_matrices()	16
4.3.1.7 transform_kernels()	16
5 File Documentation	17
5.1 quadrature.hpp	17
5.2 swdio.hpp	17
5.3 swdlayer.hpp	17
5.4 swdlayertti.hpp	18
5.5 swdlayervti.hpp	19
Index	21

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

LayerModel	7
LayerModelTTI	9
LayerModelVTI	13

Chapter 2

Class Index

2.1 Class List

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

LayerModel	7
LayerModelTTI	9
LayerModelVTI	13

Chapter 3

File Index

3.1 File List

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

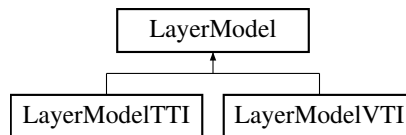
quadrature.hpp	17
swdio.hpp	17
swdlayer.hpp	17
swdlayertti.hpp	18
swdlayervti.hpp	19

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

<i>nel</i>	no. of elements for each layer, shape(nlayer - 1)
<i>thk</i>	thickness of each layer, shape(nlayer)
<i>zlist</i>	cumsum(thk)
<i>nlayer</i>	no. of layers
<i>scale</i>	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

<i>z</i>	input model z coordinates, shape(nlayer)
<i>param</i>	input model parameter, shape(nlayer)
<i>md</i>	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

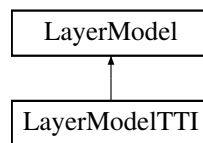
<i>z</i>	z coordiantes of previous model, shape(nlayer)
<i>param_kl</i>	SEM typed kernel, shape(nspec * NGLL + NGRL)
<i>kl_out</i>	kernels in original model, shape(nlayer)

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)
italize 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

<i>freq</i>	current frequency
<i>phi</i>	current direction, in deg
<i>c</i>	phase velocity
<i>displ</i>	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

<i>freq</i>	current frequency
<i>c</i>	phase velocity at this frequency
<i>phi</i>	azimuthal angle of c
<i>displ</i>	eigen function, shape(nglob * 3)
<i>frekl</i>	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 azimuthal 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)
```

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

Parameters

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

4.2.1.4 prepare_matrices()

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

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

Parameters

<i>phi</i>	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

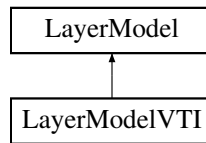
<i>frekl</i>	base Frechet kernels, shape(8,nspec*NGLL+NGRL)
--------------	--

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

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

<i>freq</i>	current frequency
<i>c</i>	current phase velocity
<i>displ</i>	eigen function, shape(nglob * 2)
<i>frekl</i>	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

<i>freq</i>	current frequency
<i>vmin,vmax</i>	min/max velocity for your model
<i>c</i>	dispersion, shape(nc)
<i>displ</i>	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

<i>freq</i>	current frequency
<i>vmin,vmax</i>	min/max velocity for your model
<i>c</i>	dispersion, shape(nc)
<i>displ</i>	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)
```

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

Parameters

<i>freq</i>	current frequency
<i>nlayer</i>	# of nlayers
<i>vpv/vph/vsv/vsh/eta/rho</i>	layer model vti parameters, shape(nlayer)
<i>is_layer</i>	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

<i>wavetype</i>	= 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

<i>frekl</i>	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

```
00001 #ifndef SWD_QUADRATURE
00002 #define SWD_QUADRATURE
00003 #include <cmath>
00004
00005 //GLL
00006 void gauss_legendre_lobatto(double* knots, double* weights, int length);
00007 void lagrange_poly(double xi,int 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
```

5.2 swdio.hpp

```
00001 #include <iostream>
00002
00003 inline void __myfwrite(const void *__ptr, size_t __size, size_t __nitems, FILE *__stream)
00004 {
00005     size_t size = fwrite(__ptr,__size,__nitems,__stream);
00006     if(size != __nitems) {
00007         fprintf(stderr,"cannot write to binary!\n");
00008         exit(1);
00009     }
00010 }
00011
00012
00013 template<typename T>
00014 void
00015 write_binary_f(FILE *fp, const T *data, size_t n)
00016 {
00017     // write integers of the size
00018     int size = (int)(n * sizeof(T));
00019
00020     // integer front
00021     __myfwrite(&size,sizeof(int),1,fp);
00022
00023     // data
00024     __myfwrite(data,sizeof(T),n,fp);
00025
00026     // integer back
00027     __myfwrite(&size,sizeof(int),1,fp);
00028 }
```

5.3 swdlayer.hpp

```
00001 #ifndef SWDLAYER_MODEL
```

```

00002 #define SWDLAYER_MODEL
00003
00004 #include <complex>
00005 #include <vector>
00006 #include <array>
00007
00008 class LayerModel {
00009 public:
00010     // GLL/GRL nodes and weights
00011     static const int NGLL = 7, NGRL = 20;
00012     std::array<double,NGLL> xgll,wgll;
00013     std::array<double,NGRL> xgrl,wgrl;
00014     std::array<double,NGLL*NGLL> hprimeT,hprime; // hprimeT(i,j) = l'_i(xi_j)
00015     std::array<double,NGRL*NGRL> hprimeT_grl,hprime_grl;
00016
00017 private:
00018     void initialize_nodes();
00019
00020 public:
00021     // SEM Mesh
00022     int nspec,nspec_grl; // # of elements for gll/grl layer
00023     int nglob; // # of unique points
00024     std::vector<int> ibool; // connectivity matrix, shape(nspec * NGLL + NGRL)
00025     std::vector<float> skel; // skeleton, shape(nspec * 2 + 2)
00026     std::vector<double> znodes; // shape(nspec * NGLL + NGRL)
00027     std::vector<double> jaco; // jacobian for GLL, shape(nspec + 1) dz / dxi
00028     std::vector<double> zstore; // shape(nglob)
00029
00030 public:
00031     bool IS_DICON_MODEL;
00032     std::vector<int> ilayer_flag; // shape(nspec + 1), return layer flag
00033     double PHASE_VELOC_MIN,PHASE_VELOC_MAX;
00034
00035 //functions
00036 public:
00037     LayerModel(){};
00038     void initialize();
00039     void interp_model(const float *z,const float *param,std::vector<double> &md) const;
00040     void create_mesh(const int *nel, const float *thk,const float *zlist,int nlayer,double scale);
00041     void project_kl(const float *z, const double *param_kl, double *kl_out) const;
00042 };
00043
00044 #endif

```

5.4 swdlayertti.hpp

```

00001 #ifndef SWD_LAYER_TTI_MODEL
00002 #define SWD_LAYER_TTI_MODEL
00003
00004 #include <complex>
00005 #include <vector>
00006 #include <array>
00007
00008 #include "swdlayer.hpp"
00009
00010 class LayerModelTTI : public LayerModel{
00011 public:
00012     typedef std::complex<double> dcmplx;
00013     public:
00014         LayerModelTTI(){};
00015
00016 private:
00017     std::vector<dcmplx> Mmat,Emat,Klmat,K2mat; // matrices for SEM,shape(3*nglob,3*nglob) om^2 M = k^2
00018     K_2 + k K_1 + E
00019
00020 public:
00021
00022     // density
00023     std::vector<double> xrho;
00024
00025     // tti Love parameters A,C,L,F,N, theta,phi
00026     std::vector<double> xA,xC,xL,xF,xN; // shape(nspec * NGLL + NGRL)
00027     std::vector<double> xT,xP; // theta/phi, shape(nspec *NGLL + NGRL), in rad
00028
00029 // VTI model
00030 void create_database(double freq,int nlayer, const float *vpv, const float* vpv,
00031                     const float *vsh, const float *vsv, const float *eta,
00032                     const float *theta0, const float *phi0,
00033                     const float *rho,const float *thk,bool is_layer);
00034
00035

```

```

00036     void prepare_matrices(double phi);
00037
00038     void compute_egnfun(double freq, double phi, std::vector<double> &c, std::vector<dcmplx> &displ)
        const;
00039     std::array<double,2>
00040     compute_kernels(double freq, double c, double phi,
00041                     const std::vector<dcmplx> &displ,
00042                     std::vector<double> &frekl) const;
00043
00044     void transform_kernels(std::vector<double> &frekl) const;
00045 };
00046
00047 #endif

```

5.5 swdlayervti.hpp

```

00001 #ifndef SWDLAYER_VTI_MODEL
00002 #define SWDLAYER_VTI_MODEL
00003
00004 #include "swdlayer.hpp"
00005
00006 class LayerModelVTI: public LayerModel {
00007
00008 public:
00009     LayerModelVTI(){};
00010
00011 private:
00012     std::vector<double> Mmat,Emat,Kmat; // matrices for SEM,  $\omega^2 M = k^2 K + E$ 
00013
00014 public:
00015
00016     // density
00017     std::vector<double> xrho;
00018
00019     // vti Love parameters
00020     std::vector<double> xA,xC,xL,xF,xN; // shape (nspec * NGLL + NGRL)
00021
00022     // VTI model
00023     void create_database(double freq,int nlayer, const float *vph, const float* vpv,
00024                         const float *vsh, const float *vsv, const float *eta,
00025                         const float *rho,const float *thk, bool is_layer);
00026
00027     void prepare_matrices(int wavetype);
00028
00029     void compute_slegn(double freq,std::vector<double> &c,
00030                       std::vector<double> &displ) const;
00031     void compute_sregl(double freq,std::vector<double> &c,
00032                       std::vector<double> &displ) const;
00033
00034     double compute_love_kl(double freq,double c,const double *displ, std::vector<double> &frekl)
        const;
00035     double compute_rayl_kl(double freq,double c,const double *displ, std::vector<double> &frekl)
        const;
00036
00037     void transform_kernels(std::vector<double> &frekl) const;
00038
00039 private:
00040     void prepare_matrices_love();
00041     void prepare_matrices_rayl();
00042 };
00043
00044 #endif

```


Index

- compute_egnfun
 - [LayerModelTTI, 11](#)
- compute_kernels
 - [LayerModelTTI, 11](#)
- compute_love_kl
 - [LayerModelVTI, 14](#)
- compute_rayl_kl
 - [LayerModelVTI, 14](#)
- compute_slegn
 - [LayerModelVTI, 15](#)
- compute_sregn
 - [LayerModelVTI, 15](#)
- create_database
 - [LayerModelTTI, 11](#)
 - [LayerModelVTI, 15](#)
- create_mesh
 - [LayerModel, 8](#)
- interp_model
 - [LayerModel, 8](#)
- [LayerModel, 7](#)
 - [create_mesh, 8](#)
 - [interp_model, 8](#)
 - [project_kl, 8](#)
- [LayerModelTTI, 9](#)
 - [compute_egnfun, 11](#)
 - [compute_kernels, 11](#)
 - [create_database, 11](#)
 - [prepare_matrices, 12](#)
 - [transform_kernels, 12](#)
- [LayerModelVTI, 13](#)
 - [compute_love_kl, 14](#)
 - [compute_rayl_kl, 14](#)
 - [compute_slegn, 15](#)
 - [compute_sregn, 15](#)
 - [create_database, 15](#)
 - [prepare_matrices, 16](#)
 - [transform_kernels, 16](#)
- prepare_matrices
 - [LayerModelTTI, 12](#)
 - [LayerModelVTI, 16](#)
- project_kl
 - [LayerModel, 8](#)
- [quadrature.hpp, 17](#)
- [swdio.hpp, 17](#)
- [swdlayer.hpp, 17](#)
- [swdlayertti.hpp, 18](#)
- [swdlayervti.hpp, 19](#)
- transform_kernels
 - [LayerModelTTI, 12](#)
 - [LayerModelVTI, 16](#)