Classes for transient field calculation

"TraFiC" software written in Python

1 – Principles

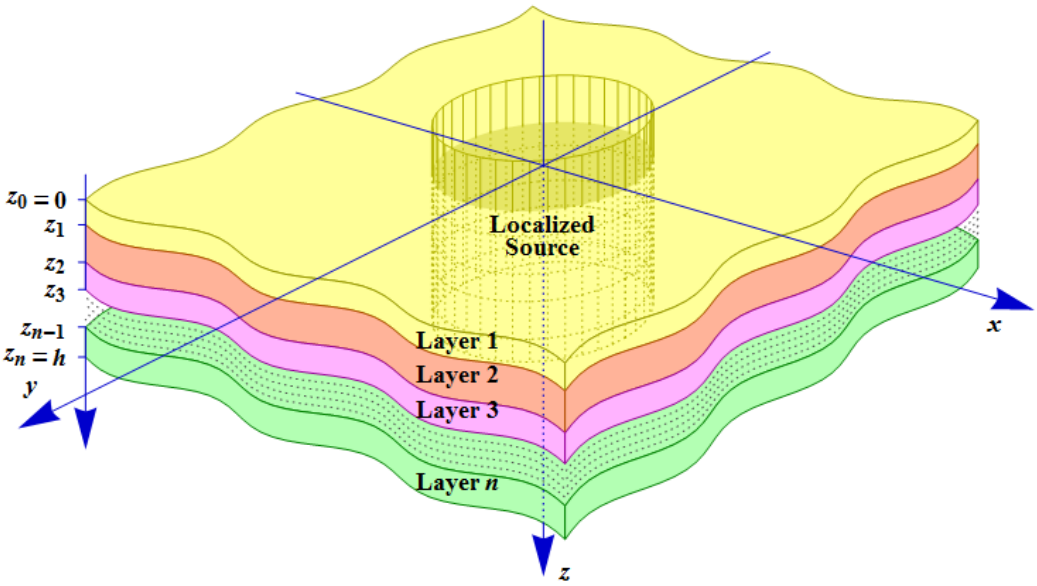

Figure 1: A multilayered infinite plate of thickness h with n layers (from [2])

1.1 Six partial waves in each elastic layer

After a Laplace transform with respect to time t and a Fourier transform with respect to the horizontal position $\mathbf{x}$ (see Table 1), the following ordinary differential equation is solved in the $(\mathbf{k}, z, s)$-domain (see [6] for more details) in each layer:

$$\begin{cases} (\mathbf{n}\diamond\mathbf{n})\,\tilde{\mathbf{U}}''(z) - \mathbb{i}\,[(\mathbf{n}\diamond\mathbf{k}) + (\mathbf{k}\diamond\mathbf{n})]\,\tilde{\mathbf{U}}'(z) - \left[(\mathbf{k}\diamond\mathbf{k}) + \rho\,s^2\,\mathbb{I}\right]\,\tilde{\mathbf{U}}(z) = -\tilde{\mathbf{F}}(z) \\ \tilde{\mathbf{\Sigma}}_z(z) = (\mathbf{n}\diamond\mathbf{n})\,\tilde{\mathbf{U}}'(z) - \mathbb{i}\,(\mathbf{n}\diamond\mathbf{k})\,\tilde{\mathbf{U}}(z) \end{cases} \tag{1}$$

where $\tilde{\mathbf{U}}(z)$ and $\tilde{\mathbf{\Sigma}}_z(z)$ denote the displacement vector and the stress vector in the z-direction.

The volume force source $\tilde{\mathbf{F}}(z)$ is assumed to be zero in the current version of the code because sources are first surface sources located at interfaces (and boundaries considered as interfaces with vacuum).

The solution of Eq. (1) in each layer is the sum of six partial waves:

$$\begin{bmatrix} \tilde{\mathbf{U}}(z) \\ \hline \tilde{\mathbf{\Sigma}}_z(z) \end{bmatrix} = \mathbf{\Xi}\,\mathcal{E}(z)\,\mathbf{a} = \sum_{i=1}^{6} a_i\,\exp[-\mathbb{i}\,\kappa_i\,(z - z_i)]\,\boldsymbol{\xi}_i\,, \quad \text{where} \begin{cases} \mathcal{E}(z) = \underset{1\leqslant i\leqslant 6}{\operatorname{diag}}\{\exp[-\mathbb{i}\,\kappa_i\,(z - z_i)]\} \\ \mathbf{\Xi} \quad = [\,\boldsymbol{\xi}_i\,]_{1\leqslant i\leqslant 6} \end{cases}, \tag{2}$$

κ_i and $\boldsymbol{\xi}_i$ being respectively the eigenvalues and the eigenvectors of the following so-called *Stroh matrix* $\mathcal{N}$[12]:

$$\mathcal{N} = \left[\begin{array}{c|c} -(\mathbf{n}\diamond\mathbf{n})^{-1}\,(\mathbf{n}\diamond\mathbf{k}) & \mathbb{i}\,(\mathbf{n}\diamond\mathbf{n})^{-1} \\ \hline \mathbb{i}\left(\rho\,s^2\,\mathbb{I} + (\mathbf{k}\diamond\mathbf{k}) - (\mathbf{k}\diamond\mathbf{n})(\mathbf{n}\diamond\mathbf{n})^{-1}(\mathbf{n}\diamond\mathbf{k})\right) & -(\mathbf{k}\diamond\mathbf{n})(\mathbf{n}\diamond\mathbf{n})^{-1} \end{array}\right]. \tag{3}$$

The six partial waves are sorted such that $\mathsf{Im}(\kappa_i) > 0$, $i = 1, 2, 3$ (upgoing waves propagating in the negative z direction, see Fig. 1) and $\mathsf{Im}(\kappa_i) < 0$, $i = 4, 5, 6$ (downgoing waves propagating in the positive z direction)

	Time-domain t	notation	Laplace-domain s	notation
Physical space $\mathbf{x}, z$	$(\mathbf{x}, z, t)$	$\mathbf{u}$	$(\mathbf{x}, z, s)$	$\mathbf{U}$
Horizontal wave-vector $\mathbf{k}$ Vertical position z	$(\mathbf{k}, z, t)$	$\tilde{\mathbf{u}}$	$(\mathbf{k}, z, s)$	$\tilde{\mathbf{U}}$
Horizontal wave-vector $\mathbf{k}$ Vertical wavenumber k_z	$(\mathbf{k}, k_z, t)$	$\hat{\mathbf{u}}$	$(\mathbf{k}, k_z, s)$	$\hat{\mathbf{U}}$

Table 1: Notations (from [6])

2 – Material classes

- File: `MaterialClasses.py`
- Last version: `Version 3.33 - 2017, October, 13`

Material

Homogeneous material

Properties:

`+ rho`	`float`, mass density.
`+ name`	`str`, name.
`+ mtype`	`str`, material type.
`+ type`	`str`, class name.

Methods:

`+ Material(param, name)`	
	`'param'` is a dictionary giving the parameters values, one key containing the words `"rho"`, `"density"` and/or `"mass"`; `'name'` is a string.
`+ check()`	returns a `bool` indicating if the material is well-defined.
`+ prt_attributes(nbdec=3)`	
	checking tool: printing all attributes; `'nbdec'` is the number of decimal digits.
`+ save(filePath)`	save the material in a text file.
`+ setName(newname)`	
`+ setRho(newrho)`	
`+ tosave()`	returns a `str` to write in a text file.

Private static methods:

`- addsubclass(className)`	
`- attrNames()`	returns a `list` of `str`.
`- subclasses()`	returns a `list` of `str`.

Tools:

- `ImportMaterialFromText(text)`: inverse of the `tosave()` method.
- `ImportMaterialFromFile(file_path)`: inverse of the `save(file_path)` method.

Fluid (Material)

Homogeneous fluid

Properties:

+ c	float, speed of sound.

Methods:

+ Fluid(param, name)	the 'param' dictionary has to contain the "c" key or one key containing the words "celerity", "speed" or "velocity".

Overloaded methods:

+ check(self)
+ prt_attributes(nbdec=3)
+ tosave(self)

AnisotropicElasticSolid (Material)

Anisotropic elastic medium (the most general case)

Properties:

+ lol	3-by-3 numpy.ndarray, $(\mathbf{l} \diamond \mathbf{l})$.
+ lom	3-by-3 numpy.ndarray, $(\mathbf{l} \diamond \mathbf{m})$.
+ lon	3-by-3 numpy.ndarray, $(\mathbf{l} \diamond \mathbf{n})$.
+ mol	3-by-3 numpy.ndarray, $(\mathbf{m} \diamond \mathbf{l})$.
+ mom	3-by-3 numpy.ndarray, $(\mathbf{m} \diamond \mathbf{m})$.
+ mon	3-by-3 numpy.ndarray, $(\mathbf{m} \diamond \mathbf{n})$.
+ nol	3-by-3 numpy.ndarray, $(\mathbf{n} \diamond \mathbf{l})$.
+ nom	3-by-3 numpy.ndarray, $(\mathbf{n} \diamond \mathbf{m})$.
+ non	3-by-3 numpy.ndarray, $(\mathbf{n} \diamond \mathbf{n})$.

Methods:

+ AnisotropicElasticSolid(param, name)	the 'param' dictionary has to contain the 21 following keys: "c11", "c12", "c13", "c14", "c15", "c16", "c22", "c23", "c24", "c25", "c26", "c33", "c34", "c35", "c36", "c44", "c45", "c46", "c55", "c56", "c66".
+ c(i,j)	returns the stiffness (float) c_{ij} (Voigt notation), where i and j are int between 1 and 6.
+ setC(i,j,newCij)	
+ diamond(u,v)	returns the 3-by-3 numpy.ndarray $(\mathbf{u} \diamond \mathbf{v})$.
+ newC(P)	returns the dictionary of stiffnesses in a new basis. P is the change-of-basis orthonal matrix containing the column-vectors of the new basis expressed in the old basis.
+ rotate(phi,theta=None,alpha=None)	returns a new material obtained by rotation of the crystallographic axes (angles are given in degrees, see Fig. 2).

Overloaded methods:

+ check()	Consistency check
+ prt_attributes(nbdec=3)	
+ tosave()	

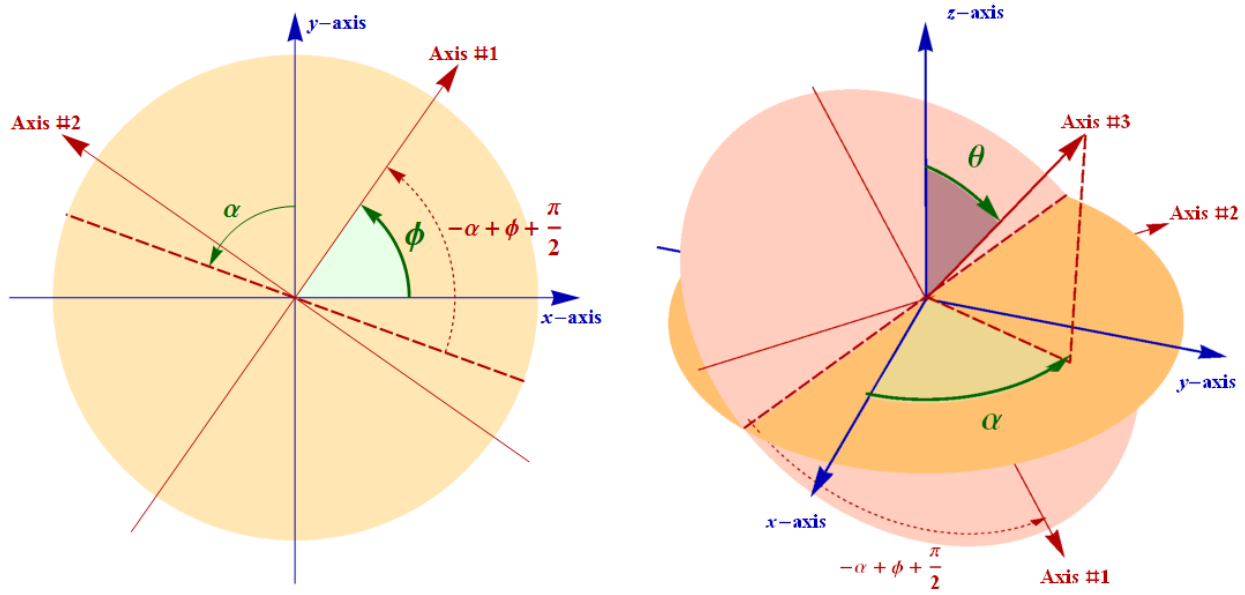

Figure 2: Angles of rotation of the crystallographic axes ϕ, θ, α,

$$\mathbf{u}_x = \begin{bmatrix} c_\alpha\, c_{\alpha-\phi}\, c_\theta + s_\alpha\, s_{\alpha-\phi} \\ c_\alpha\, s_{\alpha-\phi}\, c_\theta - s_\alpha\, c_{\alpha-\phi} \\ c_\alpha\, s_\theta \end{bmatrix}, \mathbf{u}_y = \begin{bmatrix} s_\alpha\, c_{\alpha-\phi}\, c_\theta - c_\alpha\, s_{\alpha-\phi} \\ s_\alpha\, s_{\alpha-\phi}\, c_\theta + c_\alpha\, c_{\alpha-\phi} \\ s_\alpha\, s_\theta \end{bmatrix}, \mathbf{u}_z = \begin{bmatrix} -c_{\alpha-\phi}\, s_\theta \\ -s_{\alpha-\phi}\, s_\theta \\ c_\theta \end{bmatrix}, \text{ with } \begin{cases} c_a = \cos a \\ s_a = \sin a \end{cases}.$$

TransverselyIsotropicElasticSolid (Material)

Transversely isotropic elastic medium (axis #3 is the axis of symmetry))

Properties:

+ c11	Stiffness, =c12+2*c66.
+ c12	Stiffness, =c11-2*c66.
+ c13	Stiffness.
+ c33	Stiffness.
+ c44	Stiffness.
+ c66	Stiffness, =(c11-c12)/2.
+ cPH	Speed of the pressure horizontal wave, $\sqrt{c_{11}/\rho}$.
+ cPV	Speed of the pressure vertical wave, $\sqrt{c_{33}/\rho}$.
+ cSH	Speed of the shear horizontal wave, $\sqrt{c_{66}/\rho}$.
+ cSV	Speed of the shear vertical wave, $\sqrt{c_{44}/\rho}$.

Methods:

+ TransverselyIsotropicElasticSolid(param, name)	the 'param' dictionary has to contain the 3 keys "c13", "c33", "c44", and 2 of the 3 keys "c11", "c12", "c66".
+ export()	returns a 'AnisotropicElasticSolid' instance with the same stiffnesses.
+ rotate(theta,alpha)	returns a new (anisotropic) material obtained by rotation of the crystallographic axes (angles are given in degrees).

Overloaded methods:

+ check()	Calculation of missing parameters and consistency check.
+ prt_attributes(nbdec=3)	
+ tosave()	

IsotropicElasticSolid (Material)

Isotropic elastic medium

Properties:

`+ c11`	Stiffness, `=c12+2*c44`.
`+ c12`	Stiffness, `=c11-2*c44`, first Lamé coefficient λ.
`+ c44`	Stiffness, `=(c11-c12)/2`, second Lamé coefficient μ.
`+ cL`	Speed of the pressure wave, $\sqrt{c_{11}/\rho}$.
`+ cT`	Speed of the shear wave, $\sqrt{c_{44}/\rho}$.
`+ E`	Young's Modulus.
`+ nu`	Poisson's ratio.

Methods:

`+ IsotropicElasticSolid(param, name)`	the `'param'` dictionary has to contain 2 of the 3 keys `"c11"`, `"c12"`, `"c66"`, or the 2 speeds `"cL"`, `"cT"`.
`+ export()`	returns a `'TransverselyIsotropicElasticSolid'` instance with the same stiffnesses.

Overloaded methods:

`+ check()`	Calculation of missing parameters and consistency check .
`+ prt_attributes(nbdec=3)`	
`+ tosave()`	

3 – US Material classes

- File: `USMaterialClasses.py`
- Last version: `Version 1.1 - 2017, October, 13`

The aim of « *US Material* » classes is to provide an oriented material where the « *partial waves* » (elastodynamic/acoustic, see above) are defined by their vertical wavenumbers κ_i and polarizations $\boldsymbol{\xi}_i$, with respect to the Laplace complex variable s and to the horizontal wave-vector $\mathbf{k}$. If several layers are made of the same oriented material, the calculation of the partial waves will be done only once.

We define the following numbers: n_s of s values, n_x of k_x values, n_y of k_y values.

A function `'USMaterial'` with two arguments, a material (instance of a subclass of the `'Material'` class) and an optional set of angles in degrees, is available such it returns an instance of the convenient « *US Material* » class (subclass of the virtual `'USMat'` class). An optional third argument named `'USset'` serves to manage simultaneously several plates for parallel computations.

If the material is isotropic, no angle is needed; for the moment, two classes are defined in this case: `'Fluid'` and `'USTIE'` (transversely isotropic elastic media with a symmetry axis in the z-direction, including isotropic media).

If the material is transversely isotropic (with a symmetry axis in the z-direction), the tilt angle θ of the symmetry axis has to be given; if $\theta = 0$, an instance of the `'USTIE'` class is created; if $\theta \neq 0$, the azimuthal (*i.e.* in the xy-plane) direction α of the symmetry axis is needed and an instance of the `'USAniE'` (anisotropic elastic) class is created.

If the material is anisotropic, the angles are ϕ, θ, α as detailed above and in the `'USAniE'` class below.

USMat

Virtual mother class of all « US Material » classes

Properties:

`+ initial_material`	the (non-oriented) material.
`+ angles`	angles which define the position of the crystallographic axes in the xyz-space.
`+ mat`	oriented material.
`+ nb_pw`	number of partial waves in each direction (3 for a solid, 1 for a fluid).
`+ Kz`	array of wavenumbers κ_i, of shape $n_s \times n_x [\times n_y] \times d$, where $d = 2\,$`nb_pw`.
`+ P`	array of polarizations Ξ, of shape $n_s \times n_x [\times n_y] \times 6 \times d$.
`+ MMprod`	for matrix-by-matrix product by calling **`numpy.einsum`**: `'ijklm,ijkmn->ijkln'` (3D) or `'ijkl,ijklm->ijkm'` (2D).
`+ MVprod`	for matrix-by-vector product by calling **`numpy.einsum`**: `'ijklm,ijkm->ijkl'` (3D) or `'ijkl,ijl->ijk'` (2D).
`+ nMBytes`	An estimation of the in-memory size in megabytes. It corresponds to `Kz.nbytes` + `P.nbytes`.

Methods:

`+ USMat(material, angles,USset=None)`

`+ clearSK()`	Clears **`Ks`** and **`P`**; useful to release memory.
`+ fieldComponents(dzup,dzdown)`	returns the $(n_s \times n_x [\times n_y] \times 6 \times d)$-array containing the columns $\exp(-\mathbb{i}\,\kappa_i\,$`dzup`$)\ \xi_i$, $0 \leqslant i <$ `nb_pw` (upgoing waves), and $\exp(-\mathbb{i}\,\kappa_i\,$`dzdown`$)\ \xi_i$, `nb_pw` $\leqslant i < d$ (downgoing waves).
`+ update(ones,S2,Kx,Kx2,Ky=None,Ky2=None,Kxy=None)`	has to be redefined for each derived class to update **`Kz`** and **`P`**. In 2D, **`ones`**, **`S2`**, **`Kx`** and **`Kx2`** are $(n_s \times n_x)$ arrays; In 3D, they are $(n_s \times n_x \times n_y)$ arrays; **`S2`** is the array of s^2; **`Kx2`** contains the squares of the elements of **`Kx`**; **`Ky`**, **`Kxy`** (=**`Kx*Ky`**) and **`Ky2`** are defined only in 3D.

Static properties:

`+ ListOfOrientedMaterials`	Dictionary of lists of existing oriented materials for preventing the duplication of the same US Material in a given plate ("*US set*").

Static methods:

`+ sort_eig(M)`	returns the arrays of eigenvalues and eigenvectors of an array M of square matrices, such that the eigenvalues with positive imaginary part are at the beginning (vertical wavenumbers of upgoing partial waves) and the eigenvalues with negative imaginary part are at the end (downgoing waves); in each of the two groups, the eigenvalues are sorted by increasing absolute values of their imaginary parts.

USFluid (USMat)

Methods:

`+ USFluid(material,USset=None)` **material** is an instance of the `'Fluid'` class.

Overloaded methods:

`+ update(ones,S2,K,K2,Q=None)` `K` is an array of either the wavenumbers in 2D or the norms of the horizontal wave-vectors in 3D; `K2` contains the squares of the elements of `K`; `Q` is an array of the azimuthal angles of the horizontal wave-vectors in 3D (in radians).

USTIE (USMat)

US Transversely Isotropic Elastic Material with the symmetry axis in the z-direction

Methods:

`+ USTIE(material,USset=None)` **material** is an instance of one the following two classes: `'TransverselyIsotropicElasticSolid'` or `'IsotropicElasticSolid'`.

Overloaded methods:

`+ update(ones,S2,K,K2,Q=None)` `K` is an array of either the wavenumbers in 2D or the norms of the horizontal wave-vectors in 3D; `K2` contains the squares of the elements of `K`; `Q` is an array of the azimuthal angles of the horizontal wave-vectors in 3D (in radians).

USAniE (USMat)

US Anisotropic Elastic Material

Methods:

`+ USAniE(material,angles,USset=None)`
material is an instance of one of the following two classes: `'AnisotropicElasticSolid'`, with **angles**$=(\phi,\theta,\alpha)$, or `'TransverselyIsotropicElasticSolid'`, with **angles**$=(\theta,\alpha)$.

Overloaded methods:

`+ update(ones,S2,Kx,Kx2,Ky=None,Ky2=None,Kxy=None)`
the only case where `Kx`, `Ky` and their combinations are considered.

4 – Multilayered plate class

- File: `USLayeredPlate.py`
- Last version: `Version 1.2 - 2017, October 8`

4.1 A basic 'USLayer' class

USLayer	
A basic class with 3 attributes: US material, thickness, number of partial waves	
Properties:	
`+ h` `+ thickness`	thickness (in meter).
`+ usm` `+ us_material`	US material.
`+ npw`	total number of partial waves.
Method:	
`+ USLayer(thickness, us_material)`	
Overloaded method:	
`+ __str__(number=None)`	printed by `print(object)`; the optional argument `number` is the number of the layer.

4.2 The 'USMultilayeredPlate' class

A multilayered plate is characterized by m layers rounded by two half-spaces, each of them being vacuum, a fluid or a solid.

In each layer $\#m$ of thickness h_m, the six-dimensional vector containing the displacement and the vertical stress is expressed with respect to the coefficients of the partial waves [see Eq (2) for elastic solids]:

$$\forall z\,,\ z_{m-1} < z < z_m\,,\ \left[\begin{array}{c}\tilde{\mathbf{U}}(z)\\ \hdashline \tilde{\boldsymbol{\Sigma}}_z(z)\end{array}\right] = \left[\begin{array}{c:c}\mathbf{P}_m^{[\mathrm{up}]} & \mathbf{P}_m^{[\mathrm{down}]}\end{array}\right] \left[\begin{array}{c:c}\mathcal{E}_m^{[\mathrm{up}]}(z-z_m) & \mathbb{O}_3\\ \hdashline \mathbb{O}_3 & \mathcal{E}_m^{[\mathrm{down}]}(z-z_{m-1})\end{array}\right] \left[\begin{array}{c}\mathbf{a}_m^{[\mathrm{up}]}\\ \hdashline \mathbf{a}_m^{[\mathrm{down}]}\end{array}\right] . \tag{4}$$

The dimension of the vectors $\mathbf{a}_m^{[\mathrm{up}]}$ and $\mathbf{a}_m^{[\mathrm{down}]}$ is 3 for solids, 1 for fluids, and 0 for vacuum. It is also the number of partial waves in each direction, the number of columns of the polarizations matrices $\mathbf{P}_m^{[\mathrm{up}]}$ and $\mathbf{P}_m^{[\mathrm{down}]}$, and the half of the total number d_m of partial waves in layer $\#m$.

At interface $\#m$, between layer $\#m$ and layer $\#(m+1)$, we write the e_m (dis)continuity equations:

$$-\left[\begin{array}{c:c}\mathbf{M}_m^{[\mathrm{up}]}(h_m) & \mathbf{M}_m^{[\mathrm{down}]}(h_m)\end{array}\right] \left[\begin{array}{c}\mathbf{a}_m^{[\mathrm{up}]}\\ \hdashline \mathbf{a}_m^{[\mathrm{down}]}\end{array}\right] + \left[\begin{array}{c:c}\mathbf{M}_{m+1}^{[\mathrm{up}]}(0) & \mathbf{M}_{m+1}^{[\mathrm{down}]}(0)\end{array}\right] \left[\begin{array}{c}\mathbf{a}_{m+1}^{[\mathrm{up}]}\\ \hdashline \mathbf{a}_m^{[\mathrm{down}]}\end{array}\right] = \Delta\mathbf{V}_m\,, \tag{5}$$

where

$$\mathbf{M}_m^{[\mathrm{down}]}(\delta z) = \mathbf{Q}_m^{[\mathrm{down}]}\,\mathcal{E}_m^{[\mathrm{down}]}(\delta z)\ ,\ \mathbf{M}_m^{[\mathrm{up}]}(\delta z) = \mathbf{Q}_m^{[\mathrm{up}]}\,\mathcal{E}_m^{[\mathrm{up}]}(\delta z - h_m)\ , \tag{6}$$

the polarizations matrix $\mathbf{Q}$ and the vector $\Delta\mathbf{V}$ of jumps across the interface being defined in Table 2. These jumps correspond to "*interface sources*" and are zeros if there is no source.

At interface #0 (top interface):

$$-\,\mathbf{Q}_0^{[\mathrm{up}]}\ \ \mathbf{a}_0^{[\mathrm{up}]} + \left[\begin{array}{c:c}\mathbf{M}_1^{[\mathrm{up}]}(0) & \mathbf{M}_1^{[\mathrm{down}]}(0)\end{array}\right] \left[\begin{array}{c}\mathbf{a}_1^{[\mathrm{up}]}\\ \hdashline \mathbf{a}_1^{[\mathrm{down}]}\end{array}\right] = \Delta\mathbf{V}_0\,, \tag{7}$$

Type	*Number of lines*	*Polarizations* $\mathbf{Q}$	*Jumps*
Solid/Solid	6	$\mathbf{Q} = \mathbf{P}$	$\Delta\mathbf{V} = \begin{bmatrix} \Delta\tilde{\mathbf{U}} \\ \hdashline \Delta\tilde{\mathbf{\Sigma}}_z \end{bmatrix}$
Fluid/Solid	4	last four lines of $\mathbf{P}$	$\Delta\mathbf{V} = \begin{bmatrix} \Delta\tilde{U}_z \\ \hdashline \Delta\tilde{\mathbf{\Sigma}}_z \end{bmatrix}$
Vacuum/Solid (free)	3	last three lines of $\mathbf{P}$	$\Delta\mathbf{V} = \Delta\tilde{\mathbf{\Sigma}}_z$
Fluid/Fluid	2	3rd and 6th lines of $\mathbf{P}$	$\Delta\mathbf{V} = \begin{bmatrix} \Delta\tilde{U}_z \\ \Delta\tilde{\Sigma}_{zz} \end{bmatrix}$
Vacuum/Fluid (rigid wall)	1	3rd line of $\mathbf{P}$	$\Delta\mathbf{V} = \Delta\tilde{U}_z$
Vacuum/Fluid (free)	1	6th line of $\mathbf{P}$	$\Delta\mathbf{V} = \Delta\tilde{\Sigma}_{zz}$

Table 2: Polarizations and jumps to be considered at interfaces.

the dimension of the coefficient vector $\mathbf{a}_0^{[\text{up}]}$ in the top half-space being denoted by d_0.

At interface $\#n$ (bottom interface):

$$-\left[\, \mathbf{M}_n^{[\text{up}]}(h_n) \,\vdots\, \mathbf{M}_n^{[\text{down}]}(h_n) \,\right] \begin{bmatrix} \mathbf{a}_n^{[\text{up}]} \\ \hdashline \mathbf{a}_n^{[\text{down}]} \end{bmatrix} + \mathbf{Q}_{n+1}^{[\text{down}]} \;\; \mathbf{a}_{n+1}^{[\text{down}]} = \Delta\mathbf{V}_n \;, \tag{8}$$

the dimension of the coefficient vector $\mathbf{a}_{n+1}^{[\text{down}]}$ in the bottom half-space being denoted by d_{n+1}.

All these equations give together the following d-by-d linear system:

$$\mathbf{M}\,\mathbf{A} = \Delta\mathbf{U} \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} \mathbf{a}_0^{[\text{up}]} \} d_0 \text{ coef.} \\ \hdashline \mathbf{a}_1^{[\text{up}]} \\ \mathbf{a}_1^{[\text{down}]} \} d_1 \text{ coef.} \\ \hdashline \vdots \\ \hdashline \mathbf{a}_n^{[\text{up}]} \\ \mathbf{a}_n^{[\text{down}]} \} d_n \text{ coef.} \\ \hdashline \mathbf{a}_{n+1}^{[\text{down}]} \} d_{n+1} \text{ coef.} \end{bmatrix}, \Delta\mathbf{U} = \begin{bmatrix} \Delta\mathbf{U}_0 \} e_0 \text{ coef.} \\ \hdashline \vdots \\ \hdashline \Delta\mathbf{U}_m \} e_m \text{ coef.} \\ \hdashline \vdots \\ \hdashline \Delta\mathbf{U}_n \} e_n \text{ coef.} \end{bmatrix}, \tag{9}$$

and the d-by-d matrix $\mathbf{M}$ is built as follows:

$$\left[\begin{array}{c|c|c|c|cc|cc|c|cc|c} & \overbrace{\quad}^{d_0} & \overbrace{\quad\quad}^{d_1} & & \overbrace{\quad\quad}^{d_m} & & \overbrace{\quad\quad}^{d_{m+1}} & & & \overbrace{\quad\quad}^{d_n} & & \overbrace{\quad}^{d_{n+1}} \\ e_0 & -\mathbf{Q}_0[\text{up}] & \mathbf{M}_1^{[\text{up}]}(0) & \mathbf{M}_1^{[\text{down}]}(0) & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hdashline & \ddots & \ddots & \ddots & \ddots & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \hdashline e_m & 0 & 0 & 0 & 0 & -\mathbf{M}_m^{[\text{down}]}(h_m) & -\mathbf{M}_m^{[\text{up}]}(h_m) & \mathbf{M}_{m+1}^{[\text{down}]}(0) & \mathbf{M}_{m+1}^{[\text{up}]}(0) & 0 & 0 & 0 & 0 \\ \hdashline & \ddots & \ddots & \ddots & \ddots & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \hdashline e_n & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & -\mathbf{M}_n^{[\text{down}]}(h_n) & -\mathbf{M}_n^{[\text{up}]}(h_n) & \mathbf{Q}_{n+1}^{[\text{down}]} \end{array}\right] \tag{10}$$

One can demonstrate that $d = \sum_{m=0}^{n+1} d_m = \sum_{m=0}^{n} e_m$.

USMultilayeredPlate (1/2)

US multilayered plate containing layers, each layer characterized by a thickness and an oriented material

Properties:

+ layers	**list** of n US layers.
+ topUSmat	US material in the top half-space (can be **None**)
+ bottomUSmat	US material in the bottom half-space (can be **None**)
+ Z	**list** of vertical positions z_i.
+ dim	d, the **number of unknown coefficients**.
+ C	**list** of numbers: **C[0]** is the number d_0 of upgoing partial waves in the top half-space ($z < 0$); **C[m]-C[$m-1$]** is the number d_m of partial waves in layer $\#m$; **dim-C[n]** is the number d_{m+1} of downgoing partial waves in the bottom half-space ($z > z_{\max}$=**Z[-1]**).
+ R	**list** of numbers: **R[0]** is the number e_0 of equations at interface $\#0$; **R[m]-R[$m-1$]** is the number e_m of equations at interface $\#m$ ($0 < m < n$); **dim-R[$n-1$]** is the number e_n of equations at interface $\#n$.
+ Vs	Vector (one-dimensional **numpy.ndarray**) containing the values of the Laplace variable s.
+ Vkx	Vector containing the values of the wavenumbers k_x in the x-direction.
+ Vky	**None** (in 2D), or vector containing the values of the wavenumbers k_y in the y-direction (in 3D).
+ M	$(n_s \times n_x [\times n_y] \times d \times d)$ array of the global matrices M containing the equations at the interfaces (see above), for all s, k_x and k_y values.
+ nMBytes	An estimation of the in-memory size in megabytes.
+ nbPartialWaves	Total number of partial waves in the US materials of the plate.

Methods:

+ USMultilayeredPlate(thickness,material,angles=None)
builds a monolayered plate in vacuum.

+ appendLayer(thickness,material,angles=None)
adds a layer under the existing plate.

+ setTopHalfSpace(material,angles=None)
changes the material of the top half-space.

+ setBottomHalfSpace(material,angles=None)
changes the material of the bottom half-space.

+ update(Vs,Vkx,Vky=None,usmat='All',buildM=True)
updates both the US materials and the global M matrix. The vectors **Vs**, **Vkx** and, in 3D, **Vky**, contain the values of the Laplace variable s and of the components of the horizontal wave-vector **k**; if these vectors are unchanged, the optional argument **usmat** can be None or only a specified US material, such that the update is limited to the latter material. The global M matrix is built only if the optional argument **buildM** is True.

. . . / . . .

USMultilayeredPlate (2/2)

Methods:

`+ clearSK()`	Clears all US parameters; useful to release memory.
`+ clearM()`	Clears only the M array; useful to release memory before field calculation.
`+ rebuildM(forced=False)`	Rebuilds the M array. Does nothing if `forced` is false and M is not `None`.
`+ field(z,C)`	z is the vertical position, C is a $(n_s \times n_x[\times n_y] \times d)$ array containing the partial-waves coefficients. Returns the $(n_s \times n_x[\times n_y] \times 6)$ array giving both the displacement and the vertical stress at vertical position z.
`+ copy()`	Creates a copy of the multilayer plate without US parameters.

Overloaded method:

`+ __str__()`	printed by `print(object)`.

References

[1] Ducasse E., Deschamps M., *A nonstandard wave decomposition to ensure the convergence of Debye series for modeling wave propagation in an immersed anisotropic elastic plate*, Wave Motion **49** (2012) 745-764. DOI 10.1016/j.wavemoti.2012.05.001

[2] Ducasse E., Deschamps M., *Time-domain computation of the response of composite layered anisotropic plates to a localized source*, Wave Motion **51** (2014) 1364-1381. DOI 10.1016/j.wavemoti.2014.08.003

[3] Hayashi T., Inoue D., *Calculation of leaky Lamb waves with a semi-analytical finite element method*, Ultrasonics **54**(6) (2014) 1460-1469. DOI 10.1016/j.ultras.2014.04.021

[4] Hosten B., Deschamps M., Tittmann B. R., *Inhomogeneous wave generation and propagation in lossy anisotropic solids. Application to the characterization of viscoelastic composite materials*, J. Acoust. Soc. Am. **82**(5) (1987) 1763-1770. DOI 10.1121/1.395170

[5] Kausel E., *Thin-layer method: formulation in the time domain*, Int. J. Numer. Meth. Eng. **37**(6) (1994) 927-941. DOI 10.1002/nme.1620370604

[6] Mora P., Ducasse E., Deschamps M., *Transient 3D elastodynamic field in an embedded multilayered anisotropic plate*, Ultrasonics **69** (2016) 106-115. DOI 10.1016/j.ultras.2016.03.020

[7] Mora P., Ducasse E., Deschamps M., *Interaction of a guided wave with a crack in an embedded multilayered anisotropic plate: global matrix with Laplace transform formalism*, Proceedings of ICU 2015, Physics Procedia **70** (2015) 326-329. DOI 10.1016/j.phpro.2015.08.217

[8] Mora P., Elastodynamic response of a layered anisotropic plate, comparative approaches. Towards the development of hybrid methods (Ph.D. thesis, *in French*), Université de Bordeaux, 2015. HAL tel-01290710

[9] Park J., Wave motion in finite and infinite media using the Thin-Layer Method (Ph.D. thesis), Massachusetts Institute of Technology, 2002.

[10] Park J., Kausel E., *Response of layered half-space obtained directly in the time domain, part 1: SH sources*, Bull. Seismol. Soc. Am. **96**(5) (2006) 1795-1809. DOI 10.1785/0120050247

[11] Phinney R. A., *Theoretical calculation of the spectrum of first arrivals in layered elastic mediums*, J. Geophys. Res. **70**(20) (1965) 5107-5123. DOI 10.1029/JZ070i020p05107

[12] Stroh A.N., *Steady state problems in anisotropic elasticity*, J. Math. Phys. **41** (1962) 77–103. DOI 10.1002/sapm196241177