Material

Classes

- builtins.object
  - Material
    - Plate

class **Material**(builtins.object)

Material(thickness: float, longitudinal_wave_velocity: float, shear_wave_velocity: float, *, rayleigh_wave_velocity: float = None, name: str = 'no_material') - > None

Represents a material used in dispersion simulations.

This class defines the physical properties of a material, including its wave velocities
and optional attributes. It is used to characterize materials in simulations where
wave propagation through different media is analyzed.

Attributes:
  thickness (float): The thickness of the material plate, measured in millimeters (mm).
  longitudinal_wave_velocity (float): The velocity of longitudinal waves in the material, measured in meters per second (m/s).
  shear_wave_velocity (float): The velocity of shear waves in the material, measured in meters per second (m/s).
  rayleigh_wave_velocity (float, optional): The velocity of Rayleigh waves in the material, measured in meters per second (m/s). Defaults to None if not provided.
  name (str, optional): The name of the material. Defaults to "no_material" if not provided.

Methods:
  None

Methods defined here:

__eq__(self, other)
  Return self==value.

__init__(self, thickness: float, longitudinal_wave_velocity: float, shear_wave_velocity: float, *, rayleigh_wave_velocity: float = None, name: str = 'no_material') -> None
  Initialize self. See help(type(self)) for accurate signature.

__repr__(self)
  Return repr(self).

Data descriptors defined here:

__dict__
  dictionary for instance variables (if defined)

__weakref__
  list of weak references to the object (if defined)

Data and other attributes defined here:

__annotations__ = {'longitudinal_wave_velocity': <class 'float'>, 'name': <class 'str'>, 'rayleigh_wave_velocity': <class 'float'>, 'shear_wave_velocity': <class 'float'>, 'thickness': <class 'float'>}

__dataclass_fields__ = {'longitudinal_wave_velocity': Field(name='longitudinal_wave_velocity',type=<cl...appingproxy({}),kw_only=False,_field_type=_FIELD), 'name': Field(name='name',type=<class 'str'>,default='no...mappingproxy({}),kw_only=True,_field_type=_FIELD), 'rayleigh_wave_velocity': Field(name='rayleigh_wave_velocity',type=<class ...mappingproxy({}),kw_only=True,_field_type=_FIELD), 'shear_wave_velocity': Field(name='shear_wave_velocity',type=<class 'fl...appingproxy({}),kw_only=False,_field_type=_FIELD), 'thickness': Field(name='thickness',type=<class 'float'>,defa...appingproxy({}),kw_only=False,_field_type=_FIELD)}

__dataclass_params__ = _DataclassParams(init=True,repr=True,eq=True,order=False,unsafe_hash=False,frozen=False)

__hash__ = None

__match_args__ = ('thickness', 'longitudinal_wave_velocity', 'shear_wave_velocity')

name = 'no_material'

rayleigh_wave_velocity = None

class **Plate**(Material)

Plate(thickness: float, longitudinal_wave_velocity: float, shear_wave_velocity: float, *, rayleigh_wave_velocity: float = None, name: str = 'no_material') - > None

Represents a plate used in dispersion simulations.

This class defines the physical properties of a plate.
It is inheriting all material properties from the Material class.

Attributes:
  half_thickness (float) : Plate thickness divided by 2, in m

Methods:
  __post_init__ : changes units of thickness mm -> mm, initializes half_thickness

Method resolution order:
  Plate
  Material
  builtins.object

Methods defined here:

__eq__(self, other)
  Return self==value.

__init__(self, thickness: float, longitudinal_wave_velocity: float, shear_wave_velocity: float, *, rayleigh_wave_velocity: float = None, name: str = 'no_material') -

> None
Initialize self. See help(type(self)) for accurate signature.

__post_init__(self)
Post-initialization method that converts the thickness from millimeters to meters and calculates the half-thickness of the plate.

This method is automatically called after the dataclass instance is initialized. It converts the thickness attribute from millimeters to meters and computes the half-thickness of the plate.

__repr__(self)
Return repr(self).

Data and other attributes defined here:

__annotations__ = {'half_thickness': <class 'float'>}

__dataclass_fields__ = {'half_thickness': Field(name='half_thickness',type=<class 'float'>...appingproxy({}),kw_only=False,_field_type=_FIELD), 'longitudinal_wave_velocity': Field(name='longitudinal_wave_velocity',type=<cl...appingproxy({}),kw_only=False,_field_type=_FIELD), 'name': Field(name='name',type=<class 'str'>,default='no...mappingproxy({}),kw_only=True,_field_type=_FIELD), 'rayleigh_wave_velocity': Field(name='rayleigh_wave_velocity',type=<class ...mappingproxy({}),kw_only=True,_field_type=_FIELD), 'shear_wave_velocity': Field(name='shear_wave_velocity',type=<class 'fl...appingproxy({}),kw_only=False,_field_type=_FIELD), 'thickness': Field(name='thickness',type=<class 'float'>,defa...appingproxy({}),kw_only=False,_field_type=_FIELD)}

__dataclass_params__ = _DataclassParams(init=True,repr=True,eq=True,order=False,unsafe_hash=False,frozen=False)

__hash__ = None

__match_args__ = ('thickness', 'longitudinal_wave_velocity', 'shear_wave_velocity')

Data descriptors inherited from Material:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data and other attributes inherited from Material:

name = 'no_material'

rayleigh_wave_velocity = None

Wave

Modules

[numpy]

Classes

builtins.object

Wave

Lambwave
Shearwave

class **Lambwave**(Wave)

Lambwave(material, modes_nums, freq_thickness_max, cp_max, structure_mode=None, structure_freq=None, rows=None, columns=None, freq_thickness_points=None, cp_step=None)

A class representing a Lambwave type.

Dataclass inheriting Wave class, providing results and parameters specific to Lambwaves.
Class has different methods for calculation of dispersion and wavestructure for Lambwaves.

Methods:
__init__(material, modes_nums, freq_thickness_max, cp_max, structure_mode, structure_freq, rows, columns, freq_thickness_points, cp_step):
Initialization method that calls function calculating results.
__post_init__():
Post-initialization method that calls function calculating results.
calculate_symmetric(phase_velocity, freq_thickness):
Calcultes symmetric modes dispersion equation.
calculate_antisymmetric(phase_velocity, freq_thickness):
Calcultes antisymmetric modes dispersion equation.
get_group_velocity_eq(cp, fd, cp_prime, key):
Calculates group velocity from current phase velocity and fd.
calculate_wavestructure_components(x, cp, fd):
Calculates wavestructure components u and w.
solve_freq_equations(mode_type):
Solves frequency equations and returns function of cp with respect to fd.

Method resolution order:
Lambwave
Wave
builtins.object

Methods defined here:

__eq__(self, other)
Return self==value.

__init__(self, material, modes_nums, freq_thickness_max, cp_max, structure_mode=None, structure_freq=None, rows=None, columns=None, freq_thickness_points=None, cp_step=Nor
Initialization method that calls function calculating results.

This method is automatically called when the dataclass instance is initialized.
It calls init method of the base class Wave and sets additional parameters.

__post_init__(self)
Post-initialization method that calls function calculating results.

This method is automatically called after the dataclass instance is initialized.
It solves dispersion equation for symmetric and antisymmetric modes and calculates wavestructure.

__repr__(self)
Return repr(self).

calculate_antisymmetric(self, phase_velocity: float, freq_thickness: float) -> float
Calcultes antisymmetric modes dispersion equation.

Uses formula for antisymmetric modes dispersion equation and returns its real component.

Parameters:
phase_velocity (float): Current phase velocity value.
freq_thickness (float): Current frequency and thickness product value.

Returns:
float: Dispersion result.

calculate_symmetric(self, phase_velocity: float, freq_thickness: float) -> float
Calcultes symmetric modes dispersion equation.

Uses formula for symmetric modes dispersion equation and returns its real component.

Parameters:
phase_velocity (float): Current phase velocity value.
freq_thickness (float): Current frequency and thickness product value.

Returns:
float: Dispersion result.

calculate_wavestructure_components(self, x: numpy.ndarray, cp: float, fd: float) -> float
Calculates wavestructure components u and w.

Uses formula for wavestructure to obtain in-plane and out-of-plane components.

Parameters:
x (np.ndarray) : Array of points between -thickness/2 and thickness/2.
phase_velocity (float): Current phase velocity value.
freq_thickness (float): Current frequency and thickness product value.

Returns:
float: u-component
float: w-component

get_group_velocity_eq(self, cp: float, fd: float, cp_prime: float) -> float
Calculates group velocity from current phase velocity and fd.

Parameters:
phase_velocity (float): Current phase velocity value.
freq_thickness (float): Current frequency and thickness product value.
cp_prime (float): Current phase velocity derivative.

Returns:
float: Group velocity

solve_freq_equations(self, mode_type: str) -> dict
Solves frequency equations and returns a function of cp with respect to fd.

The algorithm is based on the one presented in J.L. Rose's "Ultrasonic Guided Waves in Solid Media," Chapter 6. The algorithm proceeds as follows:

1. Choose a frequency-thickness product fd_0.
2. Make an initial estimate of the phase velocity cp_0.
3. Evaluate the signs of the left-hand sides of the frequency equations.
4. Choose another phase velocity cp_1 > cp_0 and re-evaluate the signs of the frequency equations.
5. Repeat steps (3) and (4) until a sign change occurs, assuming this happens between cp_n and cp_n+1.
6. Use bisection to precisely locate the phase velocity in the interval cp_n < cp < cp_n+1
   where the left-hand side of the equation is close enough to zero.
7. After finding the root, continue searching at this fd for other roots according to steps (2) through (6).
8. Choose another fd product and repeat steps (2) through (7).

Parameters:
    mode_type (str): The type of modes shown on the plot: 'sym', 'anti', or 'both'.

Returns:
    dict: The results of the frequency equation solutions, indexed by mode type.

Data and other attributes defined here:

__annotations__ = {}

__dataclass_fields__ = {'columns': Field(name='columns',type=<class 'int'>,default=...appingproxy({}),kw_only=False,_field_type=_FIELD), 'cp_max': Field(name='cp_max',type=<cla 'int'>,default=<...appingproxy({}),kw_only=False,_field_type=_FIELD), 'cp_step': Field(name='cp_step',type=<class 'int'>,default=...appingproxy({}),kw_only=False,_field_type=_FIEL 'freq_thickness_max': Field(name='freq_thickness_max',type=<class 'int...appingproxy({}),kw_only=False,_field_type=_FIELD), 'freq_thickness_points': Field(name='freq_thickness_poi <class '...appingproxy({}),kw_only=False,_field_type=_FIELD), 'increasing_mode': Field(name='increasing_mode',type=<class 'str'>,...appingproxy({}),kw_only=False,_field_type=_FIE 'material': Field(name='material',type=<class 'wavedispersio...appingproxy({}),kw_only=False,_field_type=_FIELD), 'modes_functions': Field(name='modes_functions',type=<class 'dict'>...appingproxy({}),kw_only=False,_field_type=_FIELD), 'rows': Field(name='rows',type=<class 'int'>,default=<da...appingproxy({}),kw_only=False,_field_type=_FIELD), 'structu Field(name='structure_freq',type=<class 'array.a...appingproxy({}),kw_only=False,_field_type=_FIELD), ...}

__dataclass_params__ = _DataclassParams(init=True,repr=True,eq=True,order=False,unsafe_hash=False,frozen=False)

__hash__ = None

__match_args__ = ('material', 'freq_thickness_max', 'cp_max', 'structure_mode', 'structure_freq', 'rows', 'columns', 'freq_thickness_points', 'cp_step')

Methods inherited from Wave:

calculate_dispersion_components(self, phase_velocity: float, freq_thickness: float) -> float
    Calculates dispersion componetns k, p and q.

    Parameters:
        phase_velocity (float): Current phase velocity value.
        freq_thickness (float): Current frequency and thickness product value.

    Returns:
        float: wavenumber
        float: p-component
        float: q-component

calculate_group_wavenumber(self, result: dict) -> dict
    Calculates group velocity and wave number and updates the dictionary

    Estimates the derivative of phase velocity with respect to fd using interpolation,
    then calculates group_velocity using the estimated derivative.
    Adds cg_val as the third value in each tuple and returns updated dict.

    Parameters:
        result (dict) : Dictionary with dispersion results.

    Returns:
        dict: Dictionary with extended results

calculate_wave_structure(self, samples_x: int = 100) -> dict
    Calculates wave_structure in and out of plane components.

    Creates array between -thickness/2 and thickness/2.
    Uses the formula for wave structure to calculate u and w compontents.
    Depending on the cases appends results to array.

    Parameters:
        samples_x (int, optional) : Number of samples for the thickness array.

    Returns:
        list: List of in and out of plane components

get_converted_mode *lambda* _, key

interpolate_result(self, result: dict) -> dict
    Interpolate result and update the dictionary

    Performs interpolation using cubic spline, generates finer fd values for smoother plot
    and stores interpolated data in the new dictionary. Additionaly cp is also stored
    as function of frequency and thickness product for wave structure.

    Parameters:
        result (dict) : Dictionary with results to interpolate.

    Returns:
        dict: interpolated result

set_converted_mode *lambda* x

Data descriptors inherited from Wave:

__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)

Data and other attributes inherited from Wave:

kind = 3

modes_nums = {'antisymmetric': None, 'symmetric': None}

smoothen = 0

structure_cp = {}

structure_result = None

tolerance = 0.001

class **Shearwave**(Wave)

Shearwave(material, modes_nums, freq_thickness_max, cp_max, structure_mode=None, structure_freq=None, rows=None, columns=None, freq_thickness_points=None, cp_step=None)

A class representing a Shearwave type.

Dataclass inheriting Wave class, providing results and parameters specific to Shearwaves.
Class has different methods for calculation of dispersion and wavestructure for Shearwaves.

Attributes:
set_converted_key (lambda) : sets key from Shearwave to Lambwave style key.
get_converted_mode (lambda) : gets mode number for Shearwave from Lambwave style key.

Methods:
__init__(material, modes_nums, freq_thickness_max, cp_max, structure_mode, structure_freq, rows, columns, freq_thickness_points, cp_step):
    Initialization method that calls function calculating results.
__post_init__():
    Post-initialization method that calls function calculating results.
calculate_symmetric(phase_velocity, freq_thickness):
    Calcultes symmetric modes dispersion equation.
calculate_antisymmetric(phase_velocity, freq_thickness):
    Calcultes antisymmetric modes dispersion equation.
get_group_velocity_eq(cp, fd, cp_prime, key):
    Calculates group velocity from current phase velocity and fd.
calculate_wavestructure_components(x, cp, fd):
    Calculates wavestructure components u and w.
solve_freq_equations(mode_type):
    Solves frequency equations and returns function of cp with respect to fd.

Method resolution order:
- Shearwave
- Wave
- builtins.object

Methods defined here:

__eq__(self, other)
    Return self==value.

__init__(self, material, modes_nums, freq_thickness_max, cp_max, structure_mode=None, structure_freq=None, rows=None, columns=None, freq_thickness_points=None, cp_step=Nor
    Initialization method that calls function calculating results.

    This method is automatically called when the dataclass instance is initialized.
    It calls init method of the base class Wave and sets additional parameters.

__post_init__(self)
    Post-initialization method that calls function calculating results.

    This method is automatically called after the dataclass instance is initialized.
    It solves dispersion equation for symmetric and antisymmetric modes and calculates wavestructure.

__repr__(self)
    Return repr(self).

calculate_antisymmetric(self, phase_velocity, freq_thickness)
    Calcultes antisymmetric modes dispersion equation.

    Uses formula for antisymmetric modes dispersion equation and returns its real component.

    Parameters:
        phase_velocity (float): Current phase velocity value.
        freq_thickness (float): Current frequency and thickness product value.

    Returns:
        float: Dispersion result.

calculate_symmetric(self, phase_velocity, freq_thickness) -> float
    Calcultes symmetric modes dispersion equation.

    Uses formula for symmetric modes dispersion equation and returns its real component.

    Parameters:
        phase_velocity (float): Current phase velocity value.
        freq_thickness (float): Current frequency and thickness product value.

    Returns:
        float: Dispersion result.

calculate_wavestructure_components(self, x: numpy.ndarray) -> float
    Calculates wavestructure components u and w.

    Uses formula for wavestructure to obtain in-plane and out-of-plane components.

    Parameters:
        x (np.ndarray) : Array of points between -thickness/2 and thickness/2.

    Returns:
        float: u-component
        float: w-component

get_converted_mode *lambda* _, key

get_group_velocity_eq(self, fd: float, key: str) -> float
    Calculates group velocity from current phase velocity and fd.

    Parameters:
        freq_thickness (float): Current frequency and thickness product value.
        key (str, not used): Full mode name

    Returns:
        float: group velocity

set_converted_key *lambda* _, x

solve_freq_equations(self, mode_type: str) -> dict
    Solves frequency equations and returns function of cp with respect to fd.
    The algorithm is based on the one presented in J.L.Roses Ultrasonic Guided Waves in Solid Media - chapter 12.
    Use relation between material's shear_wave_velocity and frequency x thickness to obtain the phase velocity

    Parameters:
        mode_type (str) : Types of the modes shown on the plot: sym, anti or both.

    Returns:
        dict: Dictionary with dispersion results.

Data and other attributes defined here:

__annotations__ = {}

__dataclass_fields__ = {'columns': Field(name='columns',type=<class 'int'>,default=...appingproxy({}),kw_only=False,_field_type=_FIELD), 'cp_max': Field(name='cp_max',type=<cla
'int'>,default=<...appingproxy({}),kw_only=False,_field_type=_FIELD), 'cp_step': Field(name='cp_step',type=<class 'int'>,default=...appingproxy({}),kw_only=False,_field_type=_FIEL

'freq_thickness_max': Field(name='freq_thickness_max',type=<class 'int...appingproxy({}),kw_only=False,_field_type=_FIELD), 'freq_thickness_points': Field(name='freq_thickness_poi
<class '...appingproxy({}),kw_only=False,_field_type=_FIELD), 'increasing_mode': Field(name='increasing_mode',type=<class 'str'>,...appingproxy({}),kw_only=False,_field_type=_FIE
'material': Field(name='material',type=<class 'wavedispersio...appingproxy({}),kw_only=False,_field_type=_FIELD), 'modes_functions': Field(name='modes_functions',type=<class
'dict'>...appingproxy({}),kw_only=False,_field_type=_FIELD), 'rows': Field(name='rows',type=<class 'int'>,default=<da...appingproxy({}),kw_only=False,_field_type=_FIELD), 'structu
Field(name='structure_freq',type=<class 'array.a...appingproxy({}),kw_only=False,_field_type=_FIELD), ...}

__dataclass_params__ = _DataclassParams(init=True,repr=True,eq=True,order=False,unsafe_hash=False,frozen=False)

__hash__ = None

__match_args__ = ('material', 'freq_thickness_max', 'cp_max', 'structure_mode', 'structure_freq', 'rows', 'columns', 'freq_thickness_points', 'cp_step')

Methods inherited from Wave:

calculate_dispersion_components(self, phase_velocity: float, freq_thickness: float) -> float
    Calculates dispersion componetns k, p and q.

    Parameters:
        phase_velocity (float): Current phase velocity value.
        freq_thickness (float): Current frequency and thickness product value.

    Returns:
        float: wavenumber
        float: p-component
        float: q-component

calculate_group_wavenumber(self, result: dict) -> dict
    Calculates group velocity and wave number and updates the dictionary

    Estimates the derivative of phase velocity with respect to fd using interpolation,
    then calculates group_velocity using the estimated derivative.
    Adds cg_val as the third value in each tuple and returns updated dict.

    Parameters:
        result (dict) : Dictionary with dispersion results.

    Returns:
        dict: Dictionary with extended results

calculate_wave_structure(self, samples_x: int = 100) -> dict
    Calculates wave_structure in and out of plane components.

    Creates array between -thickness/2 and thickness/2.
    Uses the formula for wave structure to calculate u and w compontents.
    Depending on the cases appends results to array.

    Parameters:
        samples_x (int, optional) : Number of samples for the thickness array.

    Returns:
        list: List of in and out of plane components

interpolate_result(self, result: dict) -> dict
    Interpolate result and update the dictionary

    Performs interpolation using cubic spline, generates finer fd values for smoother plot
    and stores interpolated data in the new dictionary. Additionaly cp is also stored
    as function of frequency and thickness product for wave structure.

    Parameters:
        result (dict) : Dictionary with results to interpolate.

    Returns:
        dict: interpolated result

set_converted_mode *lambda* x

Data descriptors inherited from Wave:

__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)

Data and other attributes inherited from Wave:

kind = 3

modes_nums = {'antisymmetric': None, 'symmetric': None}

smoothen = 0

structure_cp = {}

structure_result = None

tolerance = 0.001

class **Wave**(builtins.object)
Wave(material: wavedispersion.Material.Material, freq_thickness_max: int, cp_max: int, structure_mode: str, structure_freq: array.array, rows: int, columns: int, freq_thickness_points: int
> None

A class representing a base wave type.

Dataclass storing parameters connected to general dispersion of the waves,
like max value of phase velocity and frequency times thickness,step between the values and
mode amount for symmetric and antisymmetric modes. It also has all the dictionaries for
storing the results, for velocity and wave structure.

Attributes:
material (Material) : Material class object providing the material information.
modes_nums (dict) : Number of symmetric and antisymmetric modes to find.
freq_thickness_max (int) : Max value of Frequency x Thickness [kHz x mm].
cp_max (int) : Max value of Frequency x Thickness [m/s].
structure_mode (str) : Which mode should be used for wavestructure plot.
structure_freq (array) : Frequencies at which to check Wavestructure.
rows (int) : Number of rows for Wavestructure plot.
colsumns (int) : Number of columns for Wavestructure plot.
freq_thickness_points (int) : Number of frequency x thickness points to find.
cp_step (int) : Step between phase velocity points checked.
kind (int) : Kind of interpolation, 3 is used for cubic.
smoothen (int) : Order of smoothening used for interpolation.
tolerance (float) : Tolerance for the root-finding.
modes_functions (dict) : Maps mode_type with functions calculating symmetric and antisymmetric modes.
velocities_dict (dict) : Maps material bulk velocities.
increasing_mode (str) : Unique mode that increases instead of decreasing with frequency x thickness.

structure_cp (dict) : Phase velocity used for wavestructure calculation.
structure_result (dict) : Result obtained for wavestructure.

Methods:
  __post_init__():
    Post-initialization method that sets default step values and plate velocites values.
  calculate_dispersion_components(phase_velocity, freq_thickness):
    Calculates dispersion componetns k, p and q.
  interpolate_result(result):
    Interpolate result and update the dictionary
  calculate_group_wavenumber(result):
    Calculates group velocity and wave number and updates the dictionary
  calculate_wave_structure(samples_x):
    Calculates wavestructure components u and w.

Methods defined here:

__eq__(self, other)
Return self==value.

__init__(self, material: wavedispersion.Material.Material, freq_thickness_max: int, cp_max: int, structure_mode: str, structure_freq: array.array, rows: int, columns: int, freq_thickness_po
cp_step: int) -> None
Initialize self. See help(type(self)) for accurate signature.

__post_init__(self)
Post-initialization method that sets default step values and plate velocites values.

This method is automatically called after the dataclass instance is initialized.
It sets default step values and plate velocites values.

__repr__(self)
Return repr(self).

calculate_dispersion_components(self, phase_velocity: float, freq_thickness: float) -> float
Calculates dispersion componetns k, p and q.

Parameters:
    phase_velocity (float): Current phase velocity value.
    freq_thickness (float): Current frequency and thickness product value.

Returns:
    float: wavenumber
    float: p-component
    float: q-component

calculate_group_wavenumber(self, result: dict) -> dict
Calculates group velocity and wave number and updates the dictionary

Estimates the derivative of phase velocity with respect to fd using interpolation,
then calculates group_velocity using the estimated derivative.
Adds cg_val as the third value in each tuple and returns updated dict.

Parameters:
    result (dict) : Dictionary with dispersion results.

Returns:
    dict: Dictionary with extended results

calculate_wave_structure(self, samples_x: int = 100) -> dict
Calculates wave_structure in and out of plane components.

Creates array between -thickness/2 and thickness/2.
Uses the formula for wave structure to calculate u and w compontents.
Depending on the cases appends results to array.

Parameters:
    samples_x (int, optional) : Number of samples for the thickness array.

Returns:
    list: List of in and out of plane components

get_converted_mode *lambda* _, key

interpolate_result(self, result: dict) -> dict
Interpolate result and update the dictionary

Performs interpolation using cubic spline, generates finer fd values for smoother plot
and stores interpolated data in the new dictionary. Additionaly cp is also stored
as function of frequency and thickness product for wave structure.

Parameters:
    result (dict) : Dictionary with results to interpolate.

Returns:
    dict: interpolated result

set_converted_mode *lambda* x

Data descriptors defined here:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data and other attributes defined here:

__annotations__ = {'columns': <class 'int'>, 'cp_max': <class 'int'>, 'cp_step': <class 'int'>, 'freq_thickness_max': <class 'int'>, 'freq_thickness_points': <class 'int'>, 'increasing_mode':
'material': <class 'wavedispersion.Material.Material'>, 'modes_functions': <class 'dict'>, 'rows': <class 'int'>, 'structure_freq': <class 'array.array'>, ...}

__dataclass_fields__ = {'columns': Field(name='columns',type=<class 'int'>,default=...appingproxy({}),kw_only=False,_field_type=_FIELD), 'cp_max': Field(name='cp_max',type=<cla
'int'>,default=<...appingproxy({}),kw_only=False,_field_type=_FIELD), 'cp_step': Field(name='cp_step',type=<class 'int'>,default=...appingproxy({}),kw_only=False,_field_type=_FIEL
'freq_thickness_max': Field(name='freq_thickness_max',type=<class 'int...appingproxy({}),kw_only=False,_field_type=_FIELD), 'freq_thickness_points': Field(name='freq_thickness_poi
<class '...appingproxy({}),kw_only=False,_field_type=_FIELD), 'increasing_mode': Field(name='increasing_mode',type=<class 'str'>,...appingproxy({}),kw_only=False,_field_type=_FIE
'material': Field(name='material',type=<class 'wavedispersio...appingproxy({}),kw_only=False,_field_type=_FIELD), 'modes_functions': Field(name='modes_functions',type=<class
'dict'>...appingproxy({}),kw_only=False,_field_type=_FIELD), 'rows': Field(name='rows',type=<class 'int'>,default=<da...appingproxy({}),kw_only=False,_field_type=_FIELD), 'structu
Field(name='structure_freq',type=<class 'array.a...appingproxy({}),kw_only=False,_field_type=_FIELD), ...}

__dataclass_params__ = _DataclassParams(init=True,repr=True,eq=True,order=False,unsafe_hash=False,frozen=False)

__hash__ = None

__match_args__ = ('material', 'freq_thickness_max', 'cp_max', 'structure_mode', 'structure_freq', 'rows', 'columns', 'freq_thickness_points', 'cp_step')

kind = 3

modes_nums = {'antisymmetric': None, 'symmetric': None}

smoothen = 0

structure_cp = {}

structure_result = None

tolerance = 0.001

Plot

Modules

numpy os matplotlib.pyplot

Classes

builtins.object

Plot

class **Plot**(builtins.object)

Plot(wave: wavedispersion.Wave.Wave, mode_type: str, cutoff_frequencies: bool = True, add_velocities: bool = True, path: str = 'results', symmetric_style: dict = <factory>, antisym
> None

A class to manage and manipulate Matplotlib plots.

This class is used for managing result display in both text and plot forms.
Supports creating, displaying, and managing plots in a variety of scenarios.
Provides methods for generating LaTeX formatted strings for mathematical
expressions, switching the Matplotlib backend to a non-interactive mode and saving,
showing and closing all open plot windows.

Attributes:
wave (Wave) : Wave class object providing the results to be plotted.
mode_type (str) : Types of the modes shown on the plot: sym, anti or both.
cutoff_frequencies (bool, optional) : Show cutoff frequencies on the plot.
add_velocities (bool, optional) : Show plate velocities on the plot.
path (str, optional) : Path to save the result files.
symmetric_style (dict, optional) : Dict with style kwargs for symmetric modes.
antisymmetric_style (dict, optional) : Dict with style kwargs for antisymmetric modes.
torsional_style (dict, optional) : Dict with style kwargs for torsional modes.
longitudinal_style (dict, optional) : Dict with style kwargs for longitudinal modes.
flexural_style (dict, optional) : Dict with style kwargs for flexural modes.
dashed_line_style (dict, optional) : Dict with style kwargs for all dashed lines on plots.
continuous_line_style (dict, optional) : Dict with style kwargs for all continous lines on plots.
in_plane_style (dict, optional) : Dict with style kwargs for the in_plane component on wavestructure plots.
out_of_plane_style (dict, optional) : Dict with style kwargs for the out_of_plane component on wavestructure plots.
velocity_style (dict, optional) : Dict with style kwargs for plate velocity option.
padding_factor (float, optional) : Padding thickness for plots.
get_figures (lambda) : Gets all plots in an array

Methods:
_generate_latex(string):
Converts a string into LaTeX format for displaying mathematical expressions.
switch_backend():
Switches Matplotlib's backend to 'agg' for non-interactive use.
close_all_plots():
Closes all open Matplotlib plot windows.
_find_max_value(index):
Find max value for the given wave parameter.
_draw_arrow(arrow):
Draws the arrow on the plot.
_add_cutoff_frequencies(mode, max_value, plot_type):
Add cutoff frequencies to the plot.
_add_plate_velocities(plot_type):
Add plate velocities to the plot.
_plot_velocity(plot_type):
Plot the given velocity or wavenumber.
_plot_wave_structure(title):
Plot the given velocity or wavenumber.
add_plot(plot_type):
Add plot velocity.
save_plots(format, transparent, **kwargs):
Save plots in specified format.
save_txt_results(date):
Save results in text file.
show_plots():
Displays all currently active Matplotlib plots.

Methods defined here:

__eq__(self, other)
Return self==value.

__init__(self, wave: wavedispersion.Wave.Wave, mode_type: str, cutoff_frequencies: bool = True, add_velocities: bool = True, path: str = 'results', symmetric_style: dict = <factory>, ant
Initialize self. See help(type(self)) for accurate signature.

__repr__(self)
Return repr(self).

add_plot(self, plot_type: str)
Add plot velocity.

This method is used for adding desired plots to simulation.
It checks the plot_type parameter and calls the appropriate method that
generates the plot.

Parameters:
plot_type (str): Type of the plot, ex. Phase velocity plot.

Returns:
None

close_all_plots(self)
Closes all open Matplotlib plots.

This method calls `plt.close('all')` to close all figure windows. It is useful
for cleaning up after generating plots to free up resources or to start a new
plotting session without interference from previous figures.

Parameters:
None

Returns:
None

get_figures *lambda* _

save_plots(self, format: str = 'png', transparent: bool = False, **kwargs) -> list[str]
Save plots in specified format.

Parameters:
format (str): Format to save the plots (e.g., 'png', 'pdf', 'svg', etc.).

transparent (bool): Whether to save the plots with a transparent background.
**kwargs: Additional keyword arguments to pass to the `savefig` method.

Returns:
plots (list[str])

save_txt_results(self, date=False) -> str
Save results in text file.

The name of the file is generated automatically.

Example:
filename = f"Shearwaves_in_10_mm_Aluminium_plate.txt"

Parameters:
date (bool): Whether to save the plots with a date to prevent overriding.

Returns:
filepath (str)

show_plots(self)
Displays all currently active Matplotlib plots.

This method calls the `plt.show()` function to render and display
any plots that have been created.

Parameters:
None

Returns:
None

switch_backend(self)
Switches Matplotlib's backend to 'agg'.

The 'agg' backend is a non-interactive backend that can be used for
generating plot images without displaying them. This is useful for
saving plots to files in a script or when working in environments where
graphical display is not available.

Parameters:
None

Returns:
None

Data descriptors defined here:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data and other attributes defined here:

__annotations__ = {'add_velocities': <class 'bool'>, 'antisymmetric_style': <class 'dict'>, 'continuous_line_style': <class 'dict'>, 'cutoff_frequencies': <class 'bool'>, 'dashed_line_style':

__dataclass_fields__ = {'add_velocities': Field(name='add_velocities',type=<class 'bool'>,...appingproxy({}),kw_only=False,_field_type=_FIELD), 'antisymmetric_style': Field(name='ar
Field(name='dashed_line_style',type=<class 'dict...appingproxy({}),kw_only=False,_field_type=_FIELD), 'flexural_style': Field(name='flexural_style',type=<class 'dict'>,...appingproxy(
'str'>,defaul...appingproxy({}),kw_only=False,_field_type=_FIELD), 'out_of_plane_style': Field(name='out_of_plane_style',type=<class 'dic...appingproxy({}),kw_only=False,_field_type=

__dataclass_params__ = _DataclassParams(init=True,repr=True,eq=True,order=False,unsafe_hash=False,frozen=False)

__hash__ = None

__match_args__ = ('wave', 'mode_type', 'cutoff_frequencies', 'add_velocities', 'path', 'symmetric_style', 'antisymmetric_style', 'torsional_style', 'longitudinal_style', 'flexural_style', 'dashed

add_velocities = True

cutoff_frequencies = True

path = 'results'

Tests

Modules

[matplotlib.pyplot]

Functions

load_data(file_path: str, thickness: float) -> dict

Load validation results from text file.

Loads data obtained from disperse software, which can then be used for comparison.

Parameters:
    file_path (str) = Path to the disperse software results file
    thickness (float) : Plate thickness used for simulations

Returns:
    data (dict)

plot_close_all()

Closes all open Matplotlib plots.

This method calls `plt.close('all')` to close all figure windows. It is useful
for cleaning up after generating plots to free up resources or to start a new
plotting session without interference from previous figures.

Parameters:
    None

Returns:
    None

plot_data(data, wave_model, type, title, save_path, background=False)

Plot validation results from both softwares.

Plots data obtained from disperse software and overlaps it with result from this library.

Parameters:
    wave_model (Wave) = Wave class object compared.
    type (str) : Type of the plot: group or phase velocity.
    title (str) : Title for the plot.
    save_path (str) : Path where validation result should be stored.
    background (bool) : Use non-interactive backend for plots.

Returns:
    None

plot_show()

Displays all currently active Matplotlib plots.

This method calls the `plt.show()` function to render and display
any plots that have been created.

Parameters:
    None

Returns:
    None

setup_lamb_wave(file_path) -> dict | wavedispersion.Wave.Lambwave

Setups lamb wave with parameters for testing.

This function is used for quick setup for the lamb wave with parameters
for testing and validation.

Parameters:
    file_path (str) = Path to the disperse software results file

Returns:
    data_lamb (dict)

lamb (Lambwave)

setup_shear_wave(file_path: str) -> dict | wavedispersion.Wave.Shearwave

Setups shear wave with parameters for testing.

This function is used for quick setup for the shear wave with parameters for testing and validation.

Parameters:

file_path (str) = Path to the disperse software results file

Returns:

data_shear (dict)
shear (Shearwave)