Permeatus

Release 1.0.0

Andrew Angus, Lukasz Figiel

Mar 26, 2024

CONTENTS:

Permeation modelling tools for the Abaqus FEM software, specifically with the application of hydrogen permeation in polymer and polymer composite pipeline infrastructure and related experiments.

See *Usage* section for information on basic use, including how to *install* the project..

See *Documentation* for instructions on updating and compiling documentation.

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

1.1 Contents

1.1.1 Usage

Cloning Repository

To acquire the code, clone from GitHub:

```
$ git clone https://github.com/lwfig/permeatus.git
```

Note: For now, the GitHub repository is private, and access must be requested by sending an email to: l.w.figiel@warwick.ac.uk

Installation

To use permeatus, first ensure you are in the root directory of the repository (`cd permeatus` after cloning), and install it using pip:

```
$ pip install .
```

Basic Use

For a demonstration of how to use the software, a tutorial Jupyter notebook is available at `./tutorial/tutorial.ipynb`, with respect to the package root directory.

Development

If it is desired to simulate a 2D system, only the mesh function must be added, in the style of those in the homogenisation module. The rest of the codes functionality will then be utilised automatically.

1.1.2 Documentation

Editing

Documentation source files sit in `./docs/source`, with respect to the package root directory. Inside is the Python configuration file `conf.py`, and several `.rst` files which, represent separate pages of the documentation. The bibilography file `Hydrogen.bib` is also contained here, and can be updated if new citations are desired.

Compiling

The documentation requires the Sphinx package, and the bibtex and autoapi extensions, which can be installed with:

```
$ pip install sphinxcontrib-bibtex
$ pip install sphinx-autoapi
```

Compiling the documentation to HTML is then achieved by first ensuring that you are in the `./docs` folder, and running:

```
$ make html
```

The resulting HTML files are located in `./docs/build/html`, and opening `./docs/build/html/index.html` with a browser will open the documentation home page.

Similarly, if Latex is installed on the device, then compiling to PDF is achieved with:

```
$ make latexpdf
```

The resulting PDF will be located in `./docs/build/latex/permeatus.pdf`.

1.1.3 Bibliography

1.1.4 API Reference

This page contains auto-generated API reference documentation[1].

`permeatus`

Submodules

`permeatus.homogenisation`

Module for homogenisation of permeation in inhomogeneous systems.

The main component of this module is the homogenisation class, which inherits, and builds on, functionality from the *`permeatus.layered1D`* class. The parent class contains routines for running ABAQUS simulations and post-processing into effective permeation coefficients. This class then contains methods to construct various finite element

[1] Created with sphinx-autoapi

meshes representing inhomogenous permeation problems. It also contains analytical homogenisation models (effective medium theories).

Module Contents

Classes

homogenisation	Class for extracting effective properties from inhomogeneous systems.

class permeatus.homogenisation.**homogenisation**(*materials: int*, *vFrac: ArrayLike*, *D: ArrayLike | None = None*, *S: ArrayLike | None = None*, *P: ArrayLike | None = None*, *C0: float | None = None*, *C1: float | None = None*, *p0: float | None = None*, *p1: float | None = None*, *touts: ArrayLike | None = None*, *tstep: float | None = None*, *ncpu: int = 1*, *jobname: str = 'job'*, *verbose: bool = True*, *AR: float | None = None*)

Bases: *permeatus.layered1D.layered1D*

Class for extracting effective properties from inhomogeneous systems.

materials

Number of materials in the system.

vFrac

Volume fraction of each material

D

Diffusion coefficients for each material, with suggested units: [mm^2/hr].

S

Solubility coefficients for each material, with suggested units: [nmol/mm^3.MPa].

P

Permeability coefficients for each material, with suggested units: [nmol/mm.hr.MPa].

C0

Concentration source boundary condition at bottom boundary, with suggested units: [nmol/mm^3].

C1

Concentration sink boundary condition at top boundary, with suggested units: [nmol/mm^3].

p0

Pressure source boundary condition at bottom boundary, with suggested units: [MPa].

p1

Pressure sink boundary condition at top boundary, with suggested units: [MPa].

touts

Solution output times, if using ABAQUS, with suggested units: [hr].

tstep

Simulation timestep, if using ABAQUS, with suggested units: [hr].

`ncpu`

Number of CPUs to utilise, if using ABAQUS.

`jobname`

Name of job, if using ABAQUS.

`verbose`

Boolean flag which switches verbose output on or off.

`AR`

Aspect ratio of dispersed phase, if using a model in which aspect ratio is accounted for.

`field`

Dictionary of solution data at integration points, which can contain pressure, concentration, molar flux, and pressure/concentration gradient data for each timeframe. Additionally, integration point volume data can be stored.

`frames`

Integer number of frames, corresponding to number of touts.

`D_eff`

Effective diffusion coefficient, derived from solution.

`S_eff`

Effective solubility coefficient, derived from solution.

`P_eff`

Effective permeability coefficient, derived from solution.

`ebberman_mesh`(*r: float*, *lc: float*, *showMesh: bool = True*)

Create mesh from Ebermann et. al paper

Create microstructure RVE mesh given in Ebermann *et al.* [EBKGluge22].

Parameters

- `r` – Particle radius, suggested units: [mm]
- `lc` – Mesh size control.
- `showMesh` – Control whether to launch Gmsh GUI and show created mesh.

`cross_section_mesh`(*nc: int*, *r: float*, *minSpaceFac: float = 0.1*, *maxMeshFac: float = 0.4*, *algorithm: str = 'LS'*, *showMesh: bool = True*, *seed: int | None = None*)

Create 2D fibre-reinforced composite perpendicular cross-section mesh

Create fibrous composite microstructure mesh as 2D perpendicular cross-section. Available algorithms for microstructure creation are random insertion or Lubachevsky-Stillinger [LS90].

Parameters

- `nc` – Number of circles, representing fibre cross-sections.
- `r` – Fibre radius, suggested units: [mm]
- `minSpaceFac` – Minimum spacing between fibres, as a factor of the fibre radius, to avoid meshing issues.
- `maxMeshFac` – Maximum mesh size, as a factor of the fibre radius.
- `algorithm` – Choose which algorithm to use, must be one of 'random' representing random insertion with acceptance-rejection, or 'LS' representing Lubachevsky-Stillinger.

- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.
- **seed** – Integer seed for random number generator, which can allow reproduction of the same random mesh for testing.

reuss_mesh(*Nx: int = 2*, *Ny: int = 80*, *showMesh: bool = True*)

Create mesh whose analytical solution is the Reuss bound.

Create a mesh of parallel material layers in the direction of flux. For detail see Auriault *et al.* [ABG10].

Parameters

- **Nx** – Number of cells in the x-direction.
- **Ny** – Number of cells in the y-direction.
- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.

voigt_mesh(*Nx: int = 20*, *Ny: int = 40*, *showMesh: bool = True*)

Create mesh whose analytical solution is the Voigt bound.

Create a mesh of material layers in series in the direction of flux. For detail see Auriault *et al.* [ABG10].

Parameters

- **Nx** – Number of cells in the x-direction.
- **Ny** – Number of cells in the y-direction.
- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.

get_eff_coeffs()

Get effective coefficients of system by numerical averaging

Get effective coefficients by numerical averaging. Simulation output at integration points is averaged by weighting by the volume of the integration point. The results are stored in the class attributes P_eff, D_eff, S_eff.

reuss_bound()

Calculate analytical Reuss bound

For detail see Auriault *et al.* [ABG10].

voigt_bound()

Calculate analytical Voigt bound

For detail see Auriault *et al.* [ABG10].

HS_upper_bound()

Calculate analytical Hashin-Strikman upper bound

For detail see Auriault *et al.* [ABG10].

HS_lower_bound()

Calculate analytical Hashin-Strikman lower bound

For detail see Auriault *et al.* [ABG10].

nielsen()

Calculate homogenised coefficients by the Nielsen model.

This model requires setting of the AR class attribute, and assumes an impermeable dispersed phase. See Prasad *et al.* [PNS21] for detail.

maxwell_eucken()

Calculate homogenised coefficients by the Maxwell-Eucken model.

See Wei *et al.* [WZRB18] for detail.

bruggeman()

Calculate homogenised coefficients by the Bruggeman model.

See Wei *et al.* [WZRB18] for detail.

chen()

Calculate homogenised coefficients by the Chen model.

See Chen *et al.* [CBJM02] for detail.

abstract steady_state()

Obsolete inherited method; not implemented.

abstract plot_1d()

Obsolete inherited method; not implemented.

permeatus.layered1D

Module for modelling of permeation in 1D layered systems.

The main component of this module is the layered1D class, which contains methods and attributes for modelling 1D layered systems using finite element analysis. Functionality includes an interface with ABAQUS, as well as an alternative steady-state solver. It also includes post-processing, with plotting of solutions and extracton of effective homogenised coefficients.

Module Contents

Classes

layered1D	Class for modelling 1D layered systems.

class permeatus.layered1D.**layered1D**(*materials: int*, *L: ArrayLike*, *D: ArrayLike | None = None*, *S: ArrayLike | None = None*, *P: ArrayLike | None = None*, *C0: float | None = None*, *C1: float | None = 0.0*, *p0: float | None = None*, *p1: float | None = 0.0*, *touts: ArrayLike | None = None*, *tstep: float | None = None*, *ncpu: int | None = 1*, *jobname: str | None = 'job'*, *verbose: bool = True*)

Class for modelling 1D layered systems.

This class contains attributes and methods for modelling 1D layered systems, with either ABAQUS for time-dependent solutions, or with a steady-state finite element solver. Capablities include modelling of both pressure and concentration driven problems, as well as plotting of solutions, and extraction of effective properties.

materials

Number of material layers in the system.

L

1D lengths of each layer in direction of permeation, with suggested units: [mm].

`D`

Diffusion coefficients for each layer, with suggested units: [mm^2/hr].

`S`

Solubility coefficients for each layer, with suggested units: [nmol/mm^3.MPa].

`P`

Permeability coefficients for each layer, with suggested units: [nmol/mm.hr.MPa].

`C0`

Concentration source boundary condition at first layer, with suggested units: [nmol/mm^3].

`C1`

Concentration sink boundary condition at last layer, with suggested units: [nmol/mm^3].

`p0`

Pressure source boundary condition at first layer, with suggested units: [MPa].

`p1`

Pressure sink boundary condition at last layer, with suggested units: [MPa].

`touts`

Solution output times, if using ABAQUS, with suggested units: [hr].

`tstep`

Simulation timestep, if using ABAQUS, with suggested units: [hr].

`ncpu`

Number of CPUs to utilise, if using ABAQUS.

`jobname`

Name of job, if using ABAQUS.

`verbose`

Boolean flag which switches verbose output on or off.

`totL`

Total length of layered system.

`field`

Dictionary of solution data at integration points, which can contain pressure, concentration, molar flux, and pressure/concentration gradient data for each timeframe. Additionally, integration point volume data can be stored.

`frames`

Integer number of frames, corresponding to number of touts.

`D_eff`

Effective diffusion coefficient, derived from solution.

`S_eff`

Effective solubility coefficient, derived from solution.

`P_eff`

Effective permeability coefficient, derived from solution.

`layered_mesh`(*N: ArrayLike*)

Create mesh using Gmsh for solving layered system in Abaqus.

Parameters
`N` – Integer number of computational cells assigned to each layer, if modelling with ABAQUS.

`write_abaqus_diffusion`(*dx: float*, *dy: float*, *PBC: bool = True*)

Write Abaqus diffusion simulation input file from Gmsh/permeatus data.

Gmsh has built-in capability to write nodesets and element sets to an Abaqus-style input file. This function then extends that capability to set up a diffusion/permeation simulation by writing details of boundary conditions, material properties, and step details to the input file.

`dx`

Bounding box x dimensions.

`dy`

Bounding box y dimensions.

`PBC`

Boolean flag for whether to apply periodic boundary conditions on the left and right boundaries.

`submit_job`()

Submit Abaqus job.

Mesh creation should be conducted prior to job submission.

The output files C.csv, J.csv, and V.csv are produced, with concentration, flux, and integration point volumes respectively. These are produced from a python script submitted to Abaqus cae which processes the .odb output database. The script template is located in permeatus/data/abaqus_postscript.py, relative to the package root directory.

`read_field`(*target: str*)

Read field data output from Abaqus csv file

Process Abaqus output csv file into class attribute field dictionary. Field dictionary has the following layout: field[<field>][<frame>] = {'x': <x co-ordinates>, 'y': <y co-ordinates>, 'data': <field data>, 'material': <material number>}

The integration volume field is only stored for a single frame 0, as it will be unchanged through time.

Must have run submit_job() to produce the required csv files.

Parameters
`target` – Specify target field; must be one of 'C', 'J', or 'V'.

`get_eff_coeffs`(*method='numerical'*)

Get effective coefficients of system by desired method

Get effective coefficients by either numerical averaging or by analytical solution to layered system (Reuss bound solution). The results are stored in the class attributes P_eff, D_eff, S_eff.

Parameters
`method` – Desired method; one of 'numerical' or 'analytical'

`get_P_eff`()

Get effective permeability of system by numerical averaging.

The result is stored in the P_eff class attribute.

V_mean(*field: ArrayLike*) → float

Return integration point volume weighted mean of field.

Parameters
`field` – The field to be averaged by integration point volume weighting.

Returns
Volume-weighted average of field

Return type
float

get_gradC()

Calculate concentration gradient at integration points

Calculate concentration gradient field from flux and concentration solutions, via Fick's first law. Store result in dictionary under 'grad' key.

get_gradp()

Calculate pressure gradient at integration points

Calculate pressure gradient field from flux and pressure solutions, via Darcy's law. Store result in dictionary under 'grad' key.

get_p()

Calculate pressure field at integration points

Calculate pressure field from concentration solution and solubilities. Store result in field attribute.

plot_1d(*plotTarget: str = 'C'*, *showPlot: bool = True*, *timemask: ArrayLike | None = None*)

Plot 1D layered solution from numerical data

Numerical fields for the target should be read previous to calling this method.

Parameters

- **plotTarget** – One of 'C' or 'p', to determine whether concentration or pressure solutions are plotted.
- **showPlot** – Control whether to show plot immediately or delay (useful to compile multiple plots in one figure).
- **timemask** – Boolean array the same length as the number of frames, which controls whether to plot that frame.

steady_state(*plot: bool = False*, *plotTarget: str = 'C'*, *showPlot: bool = True*) → Tuple[permeatus.utils.np.ndarray, permeatus.utils.np.ndarray, permeatus.utils.np.ndarray, permeatus.utils.np.ndarray, float]

1D finite element steady state solution

Get steady-state solution with native 1D finite element solver. The method calculates the pressure and concentration solutions at layer boundaries, as well as the scalar flux solution. The concentration solution will be dual-valued at internal boundaries. Optional plotting is controlled by function arguments.

Parameters

- **plot** – Control whether to plot solution.
- **plotTarget** – One of 'C' or 'p', to determine whether concentration or pressure solutions are plotted.
- **showPlot** – Control whether to show plot immediately or delay (useful to compile multiple plots in one figure).

Returns

- *x* – Spatial points of pressure solution
- *xc* – Spatial points of concentration solution
- *p* – Pressure solution
- *C* – Concentration solution
- *J* – Scalar flux solution

`permeatus.utils`

Utility functions for creating meshes and setting up ABAQUS permeation simulations.

These utility functions mostly utilise gmsh objects, and introduce functionality to either aid in mesh creation, or to set up ABAQUS permeation simulations.

Module Contents

Functions

boundary_nodes_2d(→ Tuple[numpy.ndarray[float], ...)	Get boundary node sets in 2D box setups
bound_proximity_check_2d(→ bool)	Check for disk proximities to 2D bounding box boundaries.
periodic_copy(→ int)	Add periodic copies of disks which are over bounding box boundaries.
periodic_disks(→ Tuple[Tuple[int, int], int])	Periodic geometry for disks of specified centers and radius.
periodic_mesh(m, dx, eps)	Enforce periodic mesh on x-bounds
nodeset(f, nodes)	Function to write node sets in Abaqus input file

permeatus.utils.**boundary_nodes_2d**(*m: gmsh.model*, *dx: float*, *dy: float*) → Tuple[numpy.ndarray[float], numpy.ndarray[float], numpy.ndarray[float], numpy.ndarray[float]]

Get boundary node sets in 2D box setups

Given a Gmsh model, and bounding box x and y dimensions, extract the node sets for each boundary.

Parameters

- **m** – Gmsh model; shortcut for gmsh.model.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

Returns

- *bottomnodes* – Node set for bottom boundary.
- *topnodes* – Node set for top boundary.
- *leftnodes* – Node set for left boundary.
- *rightnodes* – Node set for right boundary.

permeatus.utils.**bound_proximity_check_2d**(*c: numpy.ndarray[float]*, *r: float*, *eps: float*, *dx: float*, *dy: float*) → bool

Check for disk proximities to 2D bounding box boundaries.

Check given disks outer edges are a minimum distance from bounding box boundaries.

Parameters

- **c** – x and y coordinates of disks.
- **r** – Disk radius.
- **eps** – Minimum spacing between disks and boundary.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

Returns
Boolean which flags whether all disks edges are above the minimum distance from bounding box boundaries.

Return type
bool

permeatus.utils.**periodic_copy**(*m: gmsh.model*, *c: numpy.ndarray[float]*, *r: float*, *dx: float*, *dy: float*, *maxtag: int*) → int

Add periodic copies of disks which are over bounding box boundaries.

Parameters

- **m** – Gmsh model; shortcut for gmsh.model.
- **c** – x and y coordinates of disks.
- **r** – Disk radius.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

Returns
Highest Gmsh 2D object tag, after adding periodic disks.

Return type
int

permeatus.utils.**periodic_disks**(*nc: int*, *c: numpy.ndarray[float]*, *m: gmsh.model*, *dx: float*, *dy: float*, *r: float*, *eps: float*) → Tuple[Tuple[int, int], int]

Periodic geometry for disks of specified centers and radius.

Takes circle centers and creates periodically wrapped geometry, with respect to bounding box.

Parameters

- **nc** – Number of circles
- **m** – Gmsh model; shortcut for gmsh.model.
- **c** – x and y coordinates of disks.
- **r** – Disk radius.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

- **eps** – Minimum spacing between circles.

Returns

- *boxdimtag* – Tuple containing the dimension and tag number of the bounding box.
- *boxtag* – Tag number of the bounding box.

permeatus.utils.**periodic_mesh**(*m*, *dx*, *eps*)

Enforce periodic mesh on x-bounds

Use Gmsh functionality to make mesh periodically consistent across bounding box x-dimension.

Parameters

- **m** – Gmsh model; shortcut for gmsh.model.
- **dx** – Bounding box x dimensions.
- **eps** – Mesh minimum spacing.

permeatus.utils.**nodeset**(*f: fileinput.input*, *nodes: numpy.ndarray[int]*)

Function to write node sets in Abaqus input file

Parameters

- **f** – Input file object.
- **nodes** – Node set

Package Contents

Classes

layered1D	Class for modelling 1D layered systems.
homogenisation	Class for extracting effective properties from inhomogeneous systems.

Functions

boundary_nodes_2d(→ Tuple[numpy.ndarray[float], ...)	Get boundary node sets in 2D box setups
bound_proximity_check_2d(→ bool)	Check for disk proximities to 2D bounding box boundaries.
periodic_copy(→ int)	Add periodic copies of disks which are over bounding box boundaries.
periodic_disks(→ Tuple[Tuple[int, int], int])	Periodic geometry for disks of specified centers and radius.
periodic_mesh(m, dx, eps)	Enforce periodic mesh on x-bounds
nodeset(f, nodes)	Function to write node sets in Abaqus input file

class permeatus.**layered1D**(*materials: int*, *L: ArrayLike*, *D: ArrayLike | None = None*, *S: ArrayLike | None = None*, *P: ArrayLike | None = None*, *C0: float | None = None*, *C1: float | None = 0.0*, *p0: float | None = None*, *p1: float | None = 0.0*, *touts: ArrayLike | None = None*, *tstep: float | None = None*, *ncpu: int | None = 1*, *jobname: str | None = 'job'*, *verbose: bool = True*)

Class for modelling 1D layered systems.

This class contains attributes and methods for modelling 1D layered systems, with either ABAQUS for time-dependent solutions, or with a steady-state finite element solver. Capablities include modelling of both pressure and concentration driven problems, as well as plotting of solutions, and extraction of effective properties.

`materials`
: Number of material layers in the system.

`L`
: 1D lengths of each layer in direction of permeation, with suggested units: [mm].

`D`
: Diffusion coefficients for each layer, with suggested units: [mm^2/hr].

`S`
: Solubility coefficients for each layer, with suggested units: [nmol/mm^3.MPa].

`P`
: Permeability coefficients for each layer, with suggested units: [nmol/mm.hr.MPa].

`C0`
: Concentration source boundary condition at first layer, with suggested units: [nmol/mm^3].

`C1`
: Concentration sink boundary condition at last layer, with suggested units: [nmol/mm^3].

`p0`
: Pressure source boundary condition at first layer, with suggested units: [MPa].

`p1`
: Pressure sink boundary condition at last layer, with suggested units: [MPa].

`touts`
: Solution output times, if using ABAQUS, with suggested units: [hr].

`tstep`
: Simulation timestep, if using ABAQUS, with suggested units: [hr].

`ncpu`
: Number of CPUs to utilise, if using ABAQUS.

`jobname`
: Name of job, if using ABAQUS.

`verbose`
: Boolean flag which switches verbose output on or off.

`totL`
: Total length of layered system.

`field`
: Dictionary of solution data at integration points, which can contain pressure, concentration, molar flux, and pressure/concentration gradient data for each timeframe. Additionally, integration point volume data can be stored.

`frames`
: Integer number of frames, corresponding to number of touts.

`D_eff`

Effective diffusion coefficient, derived from solution.

`S_eff`

Effective solubility coefficient, derived from solution.

`P_eff`

Effective permeability coefficient, derived from solution.

`layered_mesh`(*N: ArrayLike*)

Create mesh using Gmsh for solving layered system in Abaqus.

Parameters

`N` – Integer number of computational cells assigned to each layer, if modelling with ABAQUS.

`write_abaqus_diffusion`(*dx: float*, *dy: float*, *PBC: bool = True*)

Write Abaqus diffusion simulation input file from Gmsh/permeatus data.

Gmsh has built-in capability to write nodesets and element sets to an Abaqus-style input file. This function then extends that capability to set up a diffusion/permeation simulation by writing details of boundary conditions, material properties, and step details to the input file.

`dx`

Bounding box x dimensions.

`dy`

Bounding box y dimensions.

`PBC`

Boolean flag for whether to apply periodic boundary conditions on the left and right boundaries.

`submit_job`()

Submit Abaqus job.

Mesh creation should be conducted prior to job submission.

The output files C.csv, J.csv, and V.csv are produced, with concentration, flux, and integration point volumes respectively. These are produced from a python script submitted to Abaqus cae which processes the .odb output database. The script template is located in permeatus/data/abaqus_postscript.py, relative to the package root directory.

`read_field`(*target: str*)

Read field data output from Abaqus csv file

Process Abaqus output csv file into class attribute field dictionary. Field dictionary has the following layout: field[<field>][<frame>] = {'x': <x co-ordinates>, 'y': <y co-ordinates>, 'data': <field data>, 'material': <material number>}

The integration volume field is only stored for a single frame 0, as it will be unchanged through time.

Must have run submit_job() to produce the required csv files.

Parameters

`target` – Specify target field; must be one of 'C', 'J', or 'V'.

`get_eff_coeffs`(*method='numerical'*)

Get effective coefficients of system by desired method

Get effective coefficients by either numerical averaging or by analytical solution to layered system (Reuss bound solution). The results are stored in the class attributes P_eff, D_eff, S_eff.

Parameters
`method` – Desired method; one of 'numerical' or 'analytical'

`get_P_eff()`

Get effective permeability of system by numerical averaging.

The result is stored in the P_eff class attribute.

`V_mean`(*field: ArrayLike*) → float

Return integration point volume weighted mean of field.

Parameters
`field` – The field to be averaged by integration point volume weighting.

Returns
Volume-weighted average of field

Return type
float

`get_gradC()`

Calculate concentration gradient at integration points

Calculate concentration gradient field from flux and concentration solutions, via Fick's first law. Store result in dictionary under 'grad' key.

`get_gradp()`

Calculate pressure gradient at integration points

Calculate pressure gradient field from flux and pressure solutions, via Darcy's law. Store result in dictionary under 'grad' key.

`get_p()`

Calculate pressure field at integration points

Calculate pressure field from concentration solution and solubilities. Store result in field attribute.

`plot_1d`(*plotTarget: str = 'C'*, *showPlot: bool = True*, *timemask: ArrayLike | None = None*)

Plot 1D layered solution from numerical data

Numerical fields for the target should be read previous to calling this method.

Parameters

- **plotTarget** – One of 'C' or 'p', to determine whether concentration or pressure solutions are plotted.
- **showPlot** – Control whether to show plot immediately or delay (useful to compile multiple plots in one figure).
- **timemask** – Boolean array the same length as the number of frames, which controls whether to plot that frame.

`steady_state`(*plot: bool = False*, *plotTarget: str = 'C'*, *showPlot: bool = True*) → Tuple[permeatus.utils.np.ndarray, permeatus.utils.np.ndarray, permeatus.utils.np.ndarray, permeatus.utils.np.ndarray, float]

1D finite element steady state solution

Get steady-state solution with native 1D finite element solver. The method calculates the pressure and concentration solutions at layer boundaries, as well as the scalar flux solution. The concentration solution will be dual-valued at internal boundaries. Optional plotting is controlled by function arguments.

Parameters

- **plot** – Control whether to plot solution.
- **plotTarget** – One of 'C' or 'p', to determine whether concentration or pressure solutions are plotted.
- **showPlot** – Control whether to show plot immediately or delay (useful to compile multiple plots in one figure).

Returns

- *x* – Spatial points of pressure solution
- *xc* – Spatial points of concentration solution
- *p* – Pressure solution
- *C* – Concentration solution
- *J* – Scalar flux solution

class permeatus.**homogenisation**(*materials: int*, *vFrac: ArrayLike*, *D: ArrayLike | None = None*, *S: ArrayLike | None = None*, *P: ArrayLike | None = None*, *C0: float | None = None*, *C1: float | None = None*, *p0: float | None = None*, *p1: float | None = None*, *touts: ArrayLike | None = None*, *tstep: float | None = None*, *ncpu: int = 1*, *jobname: str = 'job'*, *verbose: bool = True*, *AR: float | None = None*)

Bases: `permeatus.layered1D.layered1D`

Class for extracting effective properties from inhomogeneous systems.

materials

Number of materials in the system.

vFrac

Volume fraction of each material

D

Diffusion coefficients for each material, with suggested units: [mm^2/hr].

S

Solubility coefficients for each material, with suggested units: [nmol/mm^3.MPa].

P

Permeability coefficients for each material, with suggested units: [nmol/mm.hr.MPa].

C0

Concentration source boundary condition at bottom boundary, with suggested units: [nmol/mm^3].

C1

Concentration sink boundary condition at top boundary, with suggested units: [nmol/mm^3].

p0

Pressure source boundary condition at bottom boundary, with suggested units: [MPa].

p1

Pressure sink boundary condition at top boundary, with suggested units: [MPa].

touts

Solution output times, if using ABAQUS, with suggested units: [hr].

tstep

Simulation timestep, if using ABAQUS, with suggested units: [hr].

ncpu
Number of CPUs to utilise, if using ABAQUS.

jobname
Name of job, if using ABAQUS.

verbose
Boolean flag which switches verbose output on or off.

AR
Aspect ratio of dispersed phase, if using a model in which aspect ratio is accounted for.

field
Dictionary of solution data at integration points, which can contain pressure, concentration, molar flux, and pressure/concentration gradient data for each timeframe. Additionally, integration point volume data can be stored.

frames
Integer number of frames, corresponding to number of touts.

D_eff
Effective diffusion coefficient, derived from solution.

S_eff
Effective solubility coefficient, derived from solution.

P_eff
Effective permeability coefficient, derived from solution.

ebberman_mesh(*r: float*, *lc: float*, *showMesh: bool = True*)

Create mesh from Ebermann et. al paper

Create microstructure RVE mesh given in Ebermann *et al.* [EBKGluge22].

Parameters

- **r** – Particle radius, suggested units: [mm]
- **lc** – Mesh size control.
- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.

cross_section_mesh(*nc: int*, *r: float*, *minSpaceFac: float = 0.1*, *maxMeshFac: float = 0.4*, *algorithm: str = 'LS'*, *showMesh: bool = True*, *seed: int | None = None*)

Create 2D fibre-reinforced composite perpendicular cross-section mesh

Create fibrous composite microstructure mesh as 2D perpendicular cross-section. Available algorithms for microstructure creation are random insertion or Lubachevsky-Stillinger [LS90].

Parameters

- **nc** – Number of circles, representing fibre cross-sections.
- **r** – Fibre radius, suggested units: [mm]
- **minSpaceFac** – Minimum spacing between fibres, as a factor of the fibre radius, to avoid meshing issues.
- **maxMeshFac** – Maximum mesh size, as a factor of the fibre radius.
- **algorithm** – Choose which algorithm to use, must be one of 'random' representing random insertion with acceptance-rejection, or 'LS' representing Lubachevsky-Stillinger.

- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.
- **seed** – Integer seed for random number generator, which can allow reproduction of the same random mesh for testing.

reuss_mesh(*Nx: int = 2*, *Ny: int = 80*, *showMesh: bool = True*)

Create mesh whose analytical solution is the Reuss bound.

Create a mesh of parallel material layers in the direction of flux. For detail see Auriault *et al.* [ABG10].

Parameters

- **Nx** – Number of cells in the x-direction.
- **Ny** – Number of cells in the y-direction.
- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.

voigt_mesh(*Nx: int = 20*, *Ny: int = 40*, *showMesh: bool = True*)

Create mesh whose analytical solution is the Voigt bound.

Create a mesh of material layers in series in the direction of flux. For detail see Auriault *et al.* [ABG10].

Parameters

- **Nx** – Number of cells in the x-direction.
- **Ny** – Number of cells in the y-direction.
- **showMesh** – Control whether to launch Gmsh GUI and show created mesh.

get_eff_coeffs()

Get effective coefficients of system by numerical averaging

Get effective coefficients by numerical averaging. Simulation output at integration points is averaged by weighting by the volume of the integration point. The results are stored in the class attributes P_eff, D_eff, S_eff.

reuss_bound()

Calculate analytical Reuss bound

For detail see Auriault *et al.* [ABG10].

voigt_bound()

Calculate analytical Voigt bound

For detail see Auriault *et al.* [ABG10].

HS_upper_bound()

Calculate analytical Hashin-Strikman upper bound

For detail see Auriault *et al.* [ABG10].

HS_lower_bound()

Calculate analytical Hashin-Strikman lower bound

For detail see Auriault *et al.* [ABG10].

nielsen()

Calculate homogenised coefficients by the Nielsen model.

This model requires setting of the AR class attribute, and assumes an impermeable dispersed phase. See Prasad *et al.* [PNS21] for detail.

maxwell_eucken()

Calculate homogenised coefficients by the Maxwell-Eucken model.

See Wei *et al.* [WZRB18] for detail.

bruggeman()

Calculate homogenised coefficients by the Bruggeman model.

See Wei *et al.* [WZRB18] for detail.

chen()

Calculate homogenised coefficients by the Chen model.

See Chen *et al.* [CBJM02] for detail.

abstract steady_state()

Obsolete inherited method; not implemented.

abstract plot_1d()

Obsolete inherited method; not implemented.

permeatus.**boundary_nodes_2d**(*m: gmsh.model*, *dx: float*, *dy: float*) → Tuple[numpy.ndarray[float], numpy.ndarray[float], numpy.ndarray[float], numpy.ndarray[float]]

Get boundary node sets in 2D box setups

Given a Gmsh model, and bounding box x and y dimensions, extract the node sets for each boundary.

Parameters

- **m** – Gmsh model; shortcut for gmsh.model.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

Returns

- *bottomnodes* – Node set for bottom boundary.
- *topnodes* – Node set for top boundary.
- *leftnodes* – Node set for left boundary.
- *rightnodes* – Node set for right boundary.

permeatus.**bound_proximity_check_2d**(*c: numpy.ndarray[float]*, *r: float*, *eps: float*, *dx: float*, *dy: float*) → bool

Check for disk proximities to 2D bounding box boundaries.

Check given disks outer edges are a minimum distance from bounding box boundaries.

Parameters

- **c** – x and y coordinates of disks.
- **r** – Disk radius.
- **eps** – Minimum spacing between disks and boundary.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

Returns

Boolean which flags whether all disks edges are above the minimum distance from bounding box boundaries.

Return type
bool

`permeatus.periodic_copy`(*m: gmsh.model*, *c: numpy.ndarray[float]*, *r: float*, *dx: float*, *dy: float*, *maxtag: int*) → int

Add periodic copies of disks which are over bounding box boundaries.

Parameters

- **m** – Gmsh model; shortcut for gmsh.model.
- **c** – x and y coordinates of disks.
- **r** – Disk radius.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.

Returns
Highest Gmsh 2D object tag, after adding periodic disks.

Return type
int

`permeatus.periodic_disks`(*nc: int*, *c: numpy.ndarray[float]*, *m: gmsh.model*, *dx: float*, *dy: float*, *r: float*, *eps: float*) → Tuple[Tuple[int, int], int]

Periodic geometry for disks of specified centers and radius.

Takes circle centers and creates periodically wrapped geometry, with respect to bounding box.

Parameters

- **nc** – Number of circles
- **m** – Gmsh model; shortcut for gmsh.model.
- **c** – x and y coordinates of disks.
- **r** – Disk radius.
- **dx** – Bounding box x dimensions.
- **dy** – Bounding box y dimensions.
- **eps** – Minimum spacing between circles.

Returns

- *boxdimtag* – Tuple containing the dimension and tag number of the bounding box.
- *boxtag* – Tag number of the bounding box.

`permeatus.periodic_mesh`(*m*, *dx*, *eps*)

Enforce periodic mesh on x-bounds

Use Gmsh functionality to make mesh periodically consistent across bounding box x-dimension.

Parameters

- **m** – Gmsh model; shortcut for gmsh.model.
- **dx** – Bounding box x dimensions.
- **eps** – Mesh minimum spacing.

`permeatus.nodeset`(*f: fileinput.input*, *nodes: numpy.ndarray[int]*)

Function to write node sets in Abaqus input file

Parameters

- **f** – Input file object.
- **nodes** – Node set

BIBLIOGRAPHY

[ABG10] Jean-Louis Auriault, Claude Boutin, and Christian Geindreau. *Homogenization of Coupled Phenomena in Heterogenous Media*. John Wiley & Sons, 2010.

[CBJM02] M. Chen, J. Buckmaster, T.L. Jackson, and L. Massa. Homogenization issues and the combustion of heterogeneous solid propellants. *Proceedings of the Combustion Institute*, 29(2):2923–2929, January 2002. doi:10.1016/S1540-7489(02)80357-1.

[EBKGluge22] Michael Ebermann, Raffael Bogenfeld, Janko Kreikemeier, and Rainer Glüge. Analytical and numerical approach to determine effective diffusion coefficients for composite pressure vessels. *Composite Structures*, 291:115616, July 2022. doi:10.1016/j.compstruct.2022.115616.

[LS90] Boris D. Lubachevsky and Frank H. Stillinger. Geometric properties of random disk packings. *Journal of Statistical Physics*, 60(5):561–583, September 1990. doi:10.1007/BF01025983.

[PNS21] K. Prasad, M. Nikzad, and I. Sbarski. Modeling Permeability in Multi-Phase Polymer Composites: A Critical Review of Semi-Empirical Approaches. *Polymer Reviews*, 61(1):194–237, January 2021. doi:10.1080/15583724.2020.1743306.

[WZRB18] Han Wei, Shuaishuai Zhao, Qingyuan Rong, and Hua Bao. Predicting the effective thermal conductivities of composite materials and porous media by machine learning methods. *International Journal of Heat and Mass Transfer*, 127:908–916, December 2018. doi:10.1016/j.ijheatmasstransfer.2018.08.082.

PYTHON MODULE INDEX

p

INDEX

H

J

L

M

N

P

R

S

T

V

W