Reduced Order Model using Graph Neural Networks

Generated by Doxygen 1.9.4

Chapter 1

Solving PDEs using a Graph Neural Network

1.1 PACS Project of Andrea Bonifacio and Sara Gazzoni

1.1.1 Introduction

This repository contains the code for a Python-based library for solving PDEs using a Graph Neural Network. The code is based on the paper `Learning Reduced-Order Models for Cardiovascular Simulations with Graph Neural Networks`. We built the library with the aim of creating a fully working pipeline from the generation of the data to the training of the model and the evaluation of the results. We divided the code into three main folders:

- `scripts`: contains the scripts for the generation of the data;
- `gNN`: contains the code for the implementation of the GNN;
- `notebooks`: contains some example notebooks to test the library.

1.1.2 Prerequisites

- Python 3.x
- Required packages: FEniCS, numpy, matplotlib, torch, dgl, gmsh, meshio, scipy, tqdm, jupyter

1.1.3 Installation

This installation procedure assumes that the user has already installed FEniCS in an Anaconda environment. If this is not the case, please refer to the `FEniCS installation guide`. To install the library, please follow these steps:

1. Clone the repository
2. Activate the FEniCS environment
3. Install the required packages using `pip install -r requirements.txt`
4. Check if the installation was successful by running the script `python scripts/test_installation.py`

1.1.4 Usage

The library is able to create meshes, solve variational problems, save them in a suitable format and train a GNN on the data. The user can choose which part of the pipeline to run. We will now describe each step separately. If the user wants to run the whole pipeline, please refer to the subsection **Notebooks**.

1.1.4.1 Mesh generation

The user can generate a mesh using the script `scripts/MeshUtils.py`. Inside the script, one can modify the variables `filename` and `output_dir` to choose the name of the mesh and the directory where to save it. Then, the user can run the command `python scripts/MeshUtils.py --args` where `args` are the various parameters that can be modified to generate the meshes. The parameters are:

- `--nmesh`: number of meshes to generate;
- `--nodes`: number of the interfaces inside the mesh;
- `--seed`: seed for the random generation of the meshes;
- `--hmax`: maximum length of interfaces;
- `--hmin`: minimum length of interfaces;
- `--wmax`: maximum spacing between interfaces;
- `--wmin`: minimum spacing between interfaces;
- `--lc`: characteristic length of the mesh;
- `--spacing`: boolean variable to choose if the interfaces are equally spaced or not (True = equally spaced).

1.1.4.2 Variational problem solver and data generation

The library is built to solve the following problems:

- Heat diffusion
- Stokes problem

There is an abstract class `Solver` that the user can extend to solve other variational problems. We prepared two scripts to generate the data for the two problems mentioned above. The scripts are `scripts/HeatDataset↩ Gen.py` and `scripts/StokesDatasetGen.py`. These two scripts solve the two problems and create the dataset. As for the section above, it is possible to modify the output directory which is stored in the variable `output_dir`, the mesh directory modifying the variable `mesh_dir` and the number of samples to generate which is stored in the variable `ngraphs`. In the same cell the user can modify the parameters of the problem. To run the script, the user can run the command

```
python scripts/HeatDatasetGen.py
```

or

```
python scripts/StokesDatasetGen.py
```

The class `MeshLoader` has a method `plot_mesh` that the user can use to see the mesh that is used to solve the problem.

1.1.4.3 GNN training

For further information, please refer to the `README.md` file inside the `gNN` folder.

To train a gNN, the user can run the command

```
python gNN/network1d/training.py --args
```

Inside the `main` function, the user can modify the following variables:

- `graphs_folder`: path to the folder containing the graphs;
- `target_features`: features to predict;
- `nodes_features`: features of the nodes;
- `edges_features`: features of the edges;

The features chosen must correspond to the features that were used during the graph generation. For example, if the user wants to predict the heat flux, the target features must be '['flux']`.

The user can modify the parameters of the training in the `main` function by adding them as `--args`. The parameters are:

- `--latent_size_gnn`: size of the latent space of the GNN;
- `--latent_size_mlp`: size of the latent space of the MLP;
- `--process_iterations`: number of iterations of the GNN;
- `--number_hidden_layers_mlp`: number of hidden layers of the MLP;
- `--learning_rate`: learning rate of the optimizer;
- `--batch_size`: batch size;
- `--lr_decay`: learning rate decay;
- `--nepochs`: number of epochs;
- `--weight_decay`: weight decay;
- `--rate_noise`: rate of noise to add to the target features;
- `--rate_noise_features`: rate of noise to add to the other features;
- `--stride`: stride of the time steps (how many time steps to consider);
- `--nout`: number of output features;
- `--bc_type`: type of boundary conditions;
- `--optimizer`: optimizer to use.

1.1.4.4 GNN testing

To test a GNN, the user can run the command

```
python gNN/network1d/tester.py $MODELPATH
```

where `$MODELPATH` is the path to the folder containing the trained model. The script will compute the errors for all the train and test geometries.

1.1.5 Notebooks

To facilitate the use of the library, we prepared some notebooks that show how to use the library. The notebooks are:

- `notebooks/HeatDatasetGen.ipynb`: notebook that generates the data for the heat diffusion problem and creates the graphs;
- `notebooks/StokesDatasetGen.ipynb`: notebook that generates the data for the Stokes problem and creates the graphs;
- `notebooks/ModelTester.ipynb`: notebook that, given a model already trained, shows how to test it on the train and test geometries;

These notebooks can be used out of the box, without any modification by simply running all the cells. The dataset generation notebooks work the same as the scripts already presented, so the interested reader can refer to the previous sections for further information.

The model tester notebook shows how to test a model on the train and test geometries. The user can modify the variable `path` to choose the model to test. The notebook will compute the errors for all the train and test geometries. It is also possible to test the network on a single geometry by modifying the variable `graphs_folder` to the path of the folder containing the graphs of the geometry to test and the variable `new_graph` to the name of the graph to test.

1.1.6 Authors

- Andrea Bonifacio
- Sara Gazzoni

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 datagen Namespace Reference

Variables

- int ngraphs = 2
- string output_dir = "data/graphs_test/"
- string mesh_dir = "data/mesh_test/"
- mesh_info = json.load(open(mesh_dir + 'mesh_info.json'))
- mesh_name = mesh_info['mesh_name']
- nmesh = mesh_info['nmesh']
- nodes = mesh_info['nodes']
- f = Constant(0.0)
- g = Expression('a∗exp(-(t-b)∗(t-b)/c/c)',degree=2,a=5,b=2.5,c=1,t=0)
- u0 = Expression('0.0',degree=0)
- int T = 5
- int timesteps = 50
- int dt = T/timesteps
- int kmax = 100
- int kmin = 1
- imesh = np.random.randint(0,nmesh)
- mesh_load = mutil.MeshLoader(mesh_dir + mesh_name + f"_{imesh}")
- mesh = mesh_load.mesh
- bounds = mesh_load.bounds
- face = mesh_load.face
- V = FunctionSpace(mesh_load.mesh,"DG",1)
- k = round(np.random.uniform(kmin, kmax),2)
- heat_gaussian = gd.Heat(mesh_load,V,k,f,u0,dt,T,g)
- data = gd.DataHeat(heat_gaussian,mesh_load)

6.1.1 Variable Documentation

6.1.1.1 bounds

```
datagen.bounds = mesh_load.bounds
```

6.1.1.2 data

```
datagen.data = gd.DataHeat(heat_gaussian,mesh_load)
```

6.1.1.3 dt

```
int datagen.dt = T/timesteps
```

6.1.1.4 f

```
datagen.f = Constant(0.0)
```

6.1.1.5 face

```
datagen.face = mesh_load.face
```

6.1.1.6 g

```
datagen.g = Expression('a*exp(-(t-b)*(t-b)/c/c)',degree=2,a=5,b=2.5,c=1,t=0)
```

6.1.1.7 heat_gaussian

```
datagen.heat_gaussian = gd.Heat(mesh_load,V,k,f,u0,dt,T,g)
```

6.1.1.8 imesh

```
datagen.imesh = np.random.randint(0,nmesh)
```

6.1.1.9 k

```
datagen.k = round(np.random.uniform(kmin, kmax),2)
```

6.1.1.10 kmax

```
int datagen.kmax = 100
```

6.1.1.11 kmin

```
int datagen.kmin = 1
```

6.1.1.12 mesh

```
datagen.mesh = mesh_load.mesh
```

6.1.1.13 mesh_dir

```
string datagen.mesh_dir = "data/mesh_test/"
```

6.1.1.14 mesh_info

```
datagen.mesh_info = json.load(open(mesh_dir + 'mesh_info.json'))
```

6.1.1.15 mesh_load

```
datagen.mesh_load = mutil.MeshLoader(mesh_dir + mesh_name + f"_{imesh}")
```

6.1.1.16 mesh_name

```
datagen.mesh_name = mesh_info['mesh_name']
```

6.1.1.17 ngraphs

```
int datagen.ngraphs = 2
```

6.1.1.18 nmesh

```
datagen.nmesh = mesh_info['nmesh']
```

6.1.1.19 nodes

```
datagen.nodes = mesh_info['nodes']
```

6.1.1.20 output_dir

```
string datagen.output_dir = "data/graphs_test/"
```

6.1.1.21 T

```
int datagen.T = 5
```

6.1.1.22 timesteps

```
int datagen.timesteps = 50
```

6.1.1.23 u0

```
datagen.u0 = Expression('0.0',degree=0)
```

6.1.1.24 V

```
datagen.V = FunctionSpace(mesh_load.mesh,"DG",1)
```

6.2 GenerateData Namespace Reference

This file implements a solver class to solve a variational problem and a data generator class to store the data and the solutions of the problem solved in a dgl graph.

Classes

- class DataGenerator
- class DataHeat
- class DataNS
- class Heat
- class Solver
- class Stokes

6.2.1 Detailed Description

This file implements a solver class to solve a variational problem and a data generator class to store the data and the solutions of the problem solved in a dgl graph.

This file contains the implementation of two abstract classes: Solver and DataGenerator. The Solver class is used to solve a variational problem on a given mesh, while the DataGenerator class is used to store the data and the solutions of the problem at each time step in a dgl graph. The Solver class is then inherited by two subclasses: Stokes and Heat, which are used to solve the Stokes equation and the heat equation, respectively. The DataGenerator class is also inherited by two subclasses: DataNS and DataHeat, which are used to generate the data for the Stokes equations and the heat equation, respectively. This class needs to be initialized with the proper solver object and mesh object.

Authors

Andrea Bonifacio and Sara Gazzoni

6.3 GenerateGraph Namespace Reference

This file contains functions to generate DGL graphs from data generated by the GenerateData script.

Functions

- def generate_graph (point_data, points, edges_data, edges1, edges2)
- def add_field (graph, field, field_name, offset=0)
- def save (graph, filename, output_dir="../data/graphs/")

6.3.1 Detailed Description

This file contains functions to generate DGL graphs from data generated by the GenerateData script.

The GenerateGraph script contains some utility functions to generate a DGL graph. The graph is generated by the geenrate_graph function, which takes as input the data generated by the GenerateData script. It is possible to add time-dependent fields to the graph by using the add_field function. The graph can be saved in a specified directory by using the save function.

Authors

Andrea Bonifacio and Sara Gazzoni

6.3.2 Function Documentation

6.3.2.1 add_field()

```
def GenerateGraph.add_field (
              graph,
              field,
              field_name,
              offset = 0 )
```

```
Add time-dependent fields to a DGL graph.
Add time-dependent scalar fields as graph node features. The time-dependent
fields are stored as n x 1 x m Pytorch tensors, where n is the number of
graph nodes and m the number of timesteps.

Arguments:
    graph: DGL graph
    field: dictionary containing the time-dependent data (key: timestep, value: field value)
    field_name (string): name of the field
    offset (int): number of timesteps to skip.
                  Default: 0 -> keep all timesteps
```

6.3.2.2 generate_graph()

```
def GenerateGraph.generate_graph (
              point_data,
              points,
              edges_data,
              edges1,
              edges2 )
```

```
Generate DGL graph.

Arguments:
    point_data: dictionary containing nodes data (key: name, value: data)
    points: n x 2 numpy array of nodes coordinates
    edges_data: dictionary containing edge data (key: name, value: data)
    edges1: numpy array containing indices of source nodes for every edge
    edges2: numpy array containing indices of dest nodes for every edge

Returns:
    DGL graph
```

6.3.2.3 save()

```
def GenerateGraph.save (
              graph,
              filename,
              output_dir = "../data/graphs/" )
```

```
Save graph to disk as a DGL graph.

Arguments:
    graph: DGL graph
    filename (string): name of the file
    output_dir (string): path to output directory
```

6.4 MeshUtils Namespace Reference

This file contains utilities for creating and loading meshes.

Classes

- class MeshCreator
- class MeshLoader

Variables

- parser = argparse.ArgumentParser(description='Mesh creation')
- help
- type
- int
- default
- float
- bool
- args = parser.parse_args()
- string filename = "test_mesh"
- string output_dir = "data/mesh_test/"
- mesh_creator = MeshCreator(args)

6.4.1 Detailed Description

This file contains utilities for creating and loading meshes.

The MeshUtils file contains two classes: MeshCreator and MeshLoader. The MeshCreator class is used to generate meshes using Gmsh and to convert meshes from msh to xml format. The MeshLoader class is used to load meshes from xml files to be used in FEniCS. The file also contains a main function to call the MeshCreator class. The mesh parameters can be set in the main function or using the command line.

Authors

Andrea Bonifacio and Sara Gazzoni

6.4.2 Variable Documentation

6.4.2.1 args

```
MeshUtils.args = parser.parse_args()
```

6.4.2.2 bool

```
MeshUtils.bool
```

6.4.2.3 default

```
MeshUtils.default
```

6.4.2.4 filename

```
string MeshUtils.filename = "test_mesh"
```

6.4.2.5 float

```
MeshUtils.float
```

6.4.2.6 help

```
MeshUtils.help
```

6.4.2.7 int

```
MeshUtils.int
```

6.4.2.8 mesh_creator

```
MeshUtils.mesh_creator = MeshCreator(args)
```

6.4.2.9 output_dir

```
string MeshUtils.output_dir = "data/mesh_test/"
```

6.4.2.10 parser

```
MeshUtils.parser = argparse.ArgumentParser(description='Mesh creation')
```

6.4.2.11 type

```
MeshUtils.type
```

6.5 test_installation Namespace Reference

Variables

- list packages = ['numpy', 'matplotlib', 'torch', 'dolfin', 'meshio', 'dgl', 'scipy', 'tqdm','jupyter']
- bool flag = False

6.5.1 Variable Documentation

6.5.1.1 flag

```
bool test_installation.flag = False
```

6.5.1.2 packages

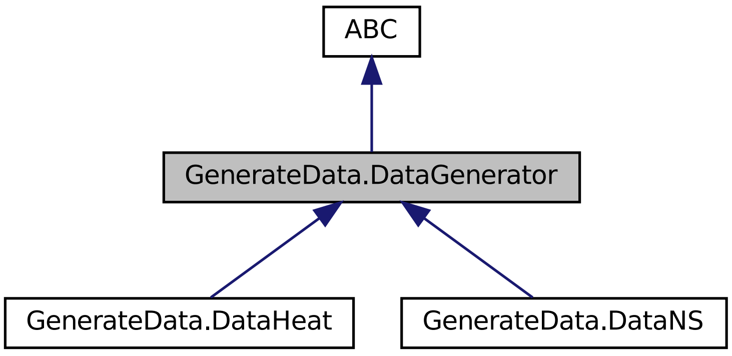
```
list test_installation.packages = ['numpy', 'matplotlib', 'torch', 'dolfin', 'meshio', 'dgl',
'scipy', 'tqdm','jupyter']
```

Chapter 7

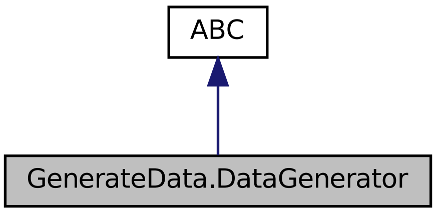
Class Documentation

7.1 GenerateData.DataGenerator Class Reference

Inheritance diagram for GenerateData.DataGenerator:



Collaboration diagram for GenerateData.DataGenerator:



Public Member Functions

- def __init__ (self, solver, mesh)
- def flux (self)
- def inlet_flux (self, tag, u)
- def area (self, tag)
- def create_edges (self)
- def edges_data (self)
- def nodes_data (self)
- def td_nodes_data (self)
- def centerline (self)
- def save_graph (self, output_dir, fields_names)
- def generate_json (self, output_dir, model_type)

Public Attributes

- solver
- mesh
- NNodes
- edges1
- edges2
- EdgesData
- NodesData
- center_line
- graph

7.1.1 Detailed Description

```
This class represents a data generator given a mesh and a solver object.
It stores the data and the solutions of a variational problem in a dgl graph.

This class is an abstract base class (ABC) and cannot be instantiated.

Attributes:
    solver: The solver object.
    mesh: The mesh object.
    NNodes: The number of interfaces in the mesh.
    edges1 (numpy.ndarray): Source nodes of the edges.
    edges2 (numpy.ndarray): Destination nodes of the edges.
    center_line (numpy.ndarray): The centerline coordinates of the inlet,
                                 interfaces and outlet of the mesh.
    NodesData (dict): A dictionary containing the nodes features.
    EdgesData (dict): A dictionary containing the edges features.
```

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
def GenerateData.DataGenerator.__init__ (
              self,
              solver,
              mesh )
```

```
Initializes the DataGenerator object.

Args:
    solver: The solver object.
    mesh: The mesh object.
```

Reimplemented in GenerateData.DataNS, and GenerateData.DataHeat.

7.1.3 Member Function Documentation

7.1.3.1 area()

```
def GenerateData.DataGenerator.area (
              self,
              tag )
```

```
Calculates the area of the face with the specified tag.

Args:
    tag: The tag of the face.

Returns:
    The area of the face with the specified tag.
```

7.1.3.2 centerline()

```
def GenerateData.DataGenerator.centerline (
              self )
```

```
Computes the centerline coordinates of the mesh corresponding
to the inlet, interfaces and outlet.

Returns:
    A n x 2 numpy array with the centerline coordinates of the mesh,
    where n is the number of nodes.
```

7.1.3.3 create_edges()

```
def GenerateData.DataGenerator.create_edges (
              self )
```

```
Creates the edges of the mesh.

Returns:
    edges1 (numpy.ndarray): Source nodes of the edges.
    edges2 (numpy.ndarray): Destination nodes of the edges.
```

7.1.3.4 edges_data()

```
def GenerateData.DataGenerator.edges_data (
              self )
```

```
Stores the edges data in a dictionary.

The data stored for each edge are the edge ID, the area of the face
where the edge is located and the length of the edge.

Returns:
    The dictionary containing the edges data.
```

7.1.3.5 flux()

```
def GenerateData.DataGenerator.flux (
              self )
```

Reimplemented in GenerateData.DataHeat, and GenerateData.DataNS.

7.1.3.6 generate_json()

```
def GenerateData.DataGenerator.generate_json (
              self,
              output_dir,
              model_type )
```

```
Generate JSON file containing information about the dataset.

Arguments:
    output_dir (string): path to output directory
    model_type (string): equation type (e.g. "heat")
```

Reimplemented in GenerateData.DataNS, and GenerateData.DataHeat.

7.1.3.7 inlet_flux()

```
def GenerateData.DataGenerator.inlet_flux (
              self,
              tag,
              u )
```

Reimplemented in GenerateData.DataNS, and GenerateData.DataHeat.

7.1.3.8 nodes_data()

```
def GenerateData.DataGenerator.nodes_data (
              self )
```

```
Stores the data of the nodes in a dictionary.

The data stored are the thermal conductivity, the node IDs,
the inlet mask, the outlet mask and the length of the interface
where the node is located.

Returns:
    The dictionary containing the nodes data.
```

7.1.3.9 save_graph()

```
def GenerateData.DataGenerator.save_graph (
              self,
              output_dir,
              fields_names )
```

```
Saves the graph with the specified fields.

Args:
    fields_names: The names of the fields.
    output_dir: The output directory to save the graph.

Returns:
    The saved graph.
```

Reimplemented in GenerateData.DataNS, and GenerateData.DataHeat.

7.1.3.10 td_nodes_data()

```
def GenerateData.DataGenerator.td_nodes_data (
              self )
```

Reimplemented in GenerateData.DataNS, and GenerateData.DataHeat.

7.1.4 Member Data Documentation

7.1.4.1 center_line

```
GenerateData.DataGenerator.center_line
```

7.1.4.2 edges1

```
GenerateData.DataGenerator.edges1
```

7.1.4.3 edges2

```
GenerateData.DataGenerator.edges2
```

7.1.4.4 EdgesData

```
GenerateData.DataGenerator.EdgesData
```

7.1.4.5 graph

```
GenerateData.DataGenerator.graph
```

7.1.4.6 mesh

```
GenerateData.DataGenerator.mesh
```

7.1.4.7 NNodes

```
GenerateData.DataGenerator.NNodes
```

7.1.4.8 NodesData

`GenerateData.DataGenerator.NodesData`

7.1.4.9 solver

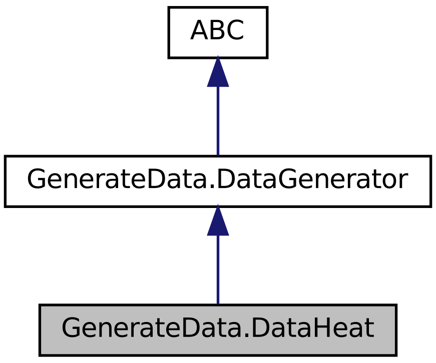
`GenerateData.DataGenerator.solver`

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

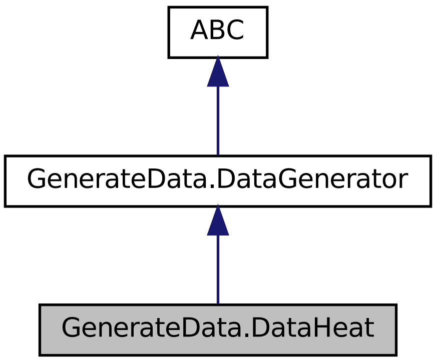
- scripts/GenerateData.py

7.2 GenerateData.DataHeat Class Reference

Inheritance diagram for GenerateData.DataHeat:



Collaboration diagram for GenerateData.DataHeat:



Public Member Functions

- def __init__ (self, solver, mesh)
- def flux (self, interface, u)
- def inlet_flux (self, tag, u)
- def td_nodes_data (self)
- def save_graph (self, output_dir)
- def generate_json (self, output_dir)

Public Attributes

- model_type
- target_fields
- TDNodesData

7.2.1 Detailed Description

```
This class represents a data generator for a heat solver.
It inherits from the DataGenerator class.

Attributes:
    solver: The solver object for heat equation.
    mesh: The mesh object representing the computational domain.
    model_type (string): The type of model (heat).
    fields_names (string): The fields that will be predicted by the graph neural network (flux).
    TDNodesData (dict): A dictionary containing the time-dependent data at each node.
```

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def GenerateData.DataHeat.__init__ (
              self,
              solver,
              mesh )
```

```
Constructor for the DataHeat class.

Args:
    solver: The solver object used to solve the heat equation.
    mesh: The mesh object representing the computational domain.
```

Reimplemented from GenerateData.DataGenerator.

7.2.3 Member Function Documentation

7.2.3.1 flux()

```
def GenerateData.DataHeat.flux (
              self,
              interface,
              u )
```

```
Calculates the heat flux at a specified interface.

Args:
    interface: The tag representing the interface.
    u: The temperature field.

Returns:
    the heat flux at the specified interface.
```

Reimplemented from GenerateData.DataGenerator.

7.2.3.2 generate_json()

```
def GenerateData.DataHeat.generate_json (
              self,
              output_dir )
```

```
Generate JSON file containing information about the dataset.

The function calls the generate_json method of the super class,
passing the model_type attributes.

Arguments:
    output_dir (string): path to output directory
```

Reimplemented from GenerateData.DataGenerator.

7.2.3.3 inlet_flux()

```
def GenerateData.DataHeat.inlet_flux (
              self,
              tag,
              u )
```

```
Calculates the heat flux at the inlet.

Args:
    tag: The tag representing the inlet.
    u: The temperature field.

Returns:
    the heat flux at the inlet.
```

Reimplemented from GenerateData.DataGenerator.

7.2.3.4 save_graph()

```
def GenerateData.DataHeat.save_graph (
              self,
              output_dir )
```

```
Saves the graph in a specified directory.

This function calls the save_graph method of the super class,
passing the model_type and fields_names attributes.

Args:
    output_dir: The directory to save the graph in.
```

Reimplemented from GenerateData.DataGenerator.

7.2.3.5 td_nodes_data()

```
def GenerateData.DataHeat.td_nodes_data (
              self )
```

```
Stores the time-dependent data in a dictionary.

The data stored are the heat flux at each node and at each time step.

Returns:
    A dictionary containing the heat flux at each node and at each time step.
```

Reimplemented from GenerateData.DataGenerator.

7.2.4 Member Data Documentation

7.2.4.1 model_type

```
GenerateData.DataHeat.model_type
```

7.2.4.2 target_fields

```
GenerateData.DataHeat.target_fields
```

7.2.4.3 TDNodesData

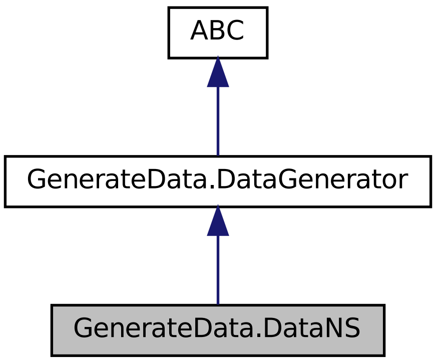
```
GenerateData.DataHeat.TDNodesData
```

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

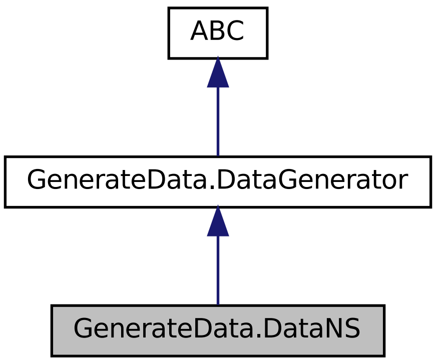
- scripts/GenerateData.py

7.3 GenerateData.DataNS Class Reference

Inheritance diagram for GenerateData.DataNS:



Collaboration diagram for GenerateData.DataNS:



Public Member Functions

- def __init__ (self, solver, mesh)
- def flux (self, tag, u)
- def inlet_flux (self, tag, u)
- def outlet_flux (self, tag, u)
- def mean_pressure_interface (self, tag, p)
- def mean_pressure_boundaries (self, tag, p)
- def td_nodes_data (self)
- def save_graph (self, output_dir)
- def generate_json (self, output_dir)

Public Attributes

- model_type
- target_fields
- TDNodesData

7.3.1 Detailed Description

```
This class represents a data generator for a Stokes solver.
It inherits from the DataGenerator class.

Attributes:
    solver (Solver): The solver object used for solving the Stokes equations.
    mesh (Mesh): The mesh object representing the computational domain.
    model_type (string): The type of model (stokes).
    target_fields (string): The fields that will be predicted
                            by the graph neural network (flowrate and pressure).
    TDNodesData (dict): A dictionary containing the time-dependent data
                        at each node.
```

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def GenerateData.DataNS.__init__ (
              self,
              solver,
              mesh )
```

```
Constructor for the DataNS class.

Args:
    solver (Solver): The solver object used to solve the Stokes equations.
    mesh (Mesh): The mesh object representing the computational domain.
```

Reimplemented from GenerateData.DataGenerator.

7.3.3 Member Function Documentation

7.3.3.1 flux()

```
def GenerateData.DataNS.flux (
              self,
              tag,
              u )
```

```
Computes the flowrate at a given interface.

Args:
    tag: The tag representing the interface.
    u: The velocity variable.
```

Reimplemented from GenerateData.DataGenerator.

7.3.3.2 generate_json()

```
def GenerateData.DataNS.generate_json (
              self,
              output_dir )
```

```
Generate JSON file containing information about the dataset.

The function calls the generate_json method of the super class,
passing the model_type attributes.

Arguments:
    output_dir (string): path to output directory
```

Reimplemented from GenerateData.DataGenerator.

7.3.3.3 inlet_flux()

```
def GenerateData.DataNS.inlet_flux (
              self,
              tag,
              u )
```

```
Computes the flowrate at the inlet.

Args:
    tag: The tag representing the inlet.
    u: The velocity variable.
```

Reimplemented from GenerateData.DataGenerator.

7.3.3.4 mean_pressure_boundaries()

```
def GenerateData.DataNS.mean_pressure_boundaries (
              self,
              tag,
              p )
```

```
Calculates the mean pressure on a specified boundary.

Args:
    tag: The tag representing the boundary (inlet or outlet).
    p: The pressure variable.
```

7.3.3.5 mean_pressure_interface()

```
def GenerateData.DataNS.mean_pressure_interface (
              self,
              tag,
              p )
```

```
Calculates the mean pressure on a specified interface.

Args:
    tag: The tag representing the interface.
    p: The pressure variable.
```

7.3.3.6 outlet_flux()

```
def GenerateData.DataNS.outlet_flux (
              self,
              tag,
              u )
```

```
Computes the flowrate at the outlet.

Args:
    tag: The tag representing the outlet.
    u: The velocity variable.
```

7.3.3.7 save_graph()

```
def GenerateData.DataNS.save_graph (
              self,
              output_dir )
```

```
Saves the graph in a specified directory.

This function calls the save_graph method of the super class,
passing the model_type and target_fields attributes.

Args:
    output_dir: The directory to save the graph in.
```

Reimplemented from GenerateData.DataGenerator.

7.3.3.8 td_nodes_data()

```
def GenerateData.DataNS.td_nodes_data (
              self )
```

```
Stores the time-dependent data in a dictionary.

The data stored are the flowrate and the pressure at each node and at each time step.

Returns:
    A dictionary containing the flowrate and a dictionary containing
    the pressure, both at each node and at each time step.
```

Reimplemented from GenerateData.DataGenerator.

7.3.4 Member Data Documentation

7.3.4.1 model_type

```
GenerateData.DataNS.model_type
```

7.3.4.2 target_fields

```
GenerateData.DataNS.target_fields
```

7.3.4.3 TDNodesData

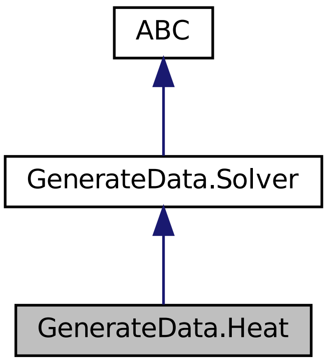
```
GenerateData.DataNS.TDNodesData
```

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

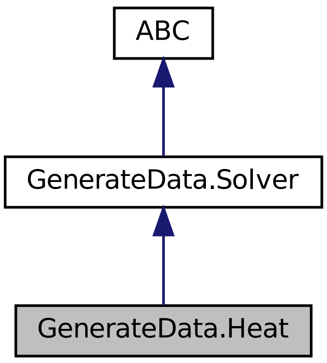
- scripts/GenerateData.py

7.4 GenerateData.Heat Class Reference

Inheritance diagram for GenerateData.Heat:



Collaboration diagram for GenerateData.Heat:



Public Member Functions

- def __init__ (self, mesh, V, k, f, u0, dt, T, g, doplot=False)
- def set_parameters (self, V, k, f, u0, dt, T, g)
- def solve (self)
- def plot_solution (self, u)

Public Attributes

- V
- k
- f
- u0
- dt
- T
- g
- doplot
- ts
- ut

7.4.1 Detailed Description

```
Class representing a heat solver.

This class inherits from the Solver class and provides methods to solve the heat equation
using the Discontinuous Galerkin method with non-homogeneous Neumann boundary conditions.

Attributes:
    V (FunctionSpace): Function space for the solution.
    k (float): Thermal conductivity.
    f (Expression): Source term.
    u0 (Expression): Initial condition.
    dt (float): Time step.
    T (float): Final time.
    g (Expression): Neumann boundary condition at the inlet.
    doplot (bool): Flag indicating whether to plot the solution at each time step.
    ts (numpy.ndarray): Array of time steps.
    ut (numpy.ndarray): Array of solutions at each time step.
```

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def GenerateData.Heat.__init__ (
              self,
              mesh,
              V,
              k,
              f,
              u0,
              dt,
              T,
              g,
              doplot = False )
```

```
Initialize the Heat class.

Args:
    mesh (Mesh): Mesh object.
    V (FunctionSpace): Function space for the solution.
    k (float): Thermal conductivity.
    f (Expression): Source term.
    u0 (Expression): Initial condition.
    dt (float): Time step.
    T (float): Final time.
    g (Expression): Neumann boundary condition at the inlet.
    doplot (bool): Flag indicating whether to plot the solution at each time step.
```

Reimplemented from GenerateData.Solver.

7.4.3 Member Function Documentation

7.4.3.1 plot_solution()

```
def GenerateData.Heat.plot_solution (
              self,
              u )
```

```
Plot the solution.

Args:
    u: The solution to be plotted.
```

Reimplemented from GenerateData.Solver.

7.4.3.2 set_parameters()

```
def GenerateData.Heat.set_parameters (
              self,
              V,
              k,
              f,
              u0,
              dt,
              T,
              g )
```

```
Method to set different parameters for the heat solver.

Args:
    V (FunctionSpace): Function space for the solution.
    k (float): Thermal conductivity.
    f (Expression): Source term.
    u0 (Expression): Initial condition.
    dt (float): Time step.
    T (float): Final time.
    g (Expression): Neumann boundary condition at the inlet.
```

Reimplemented from GenerateData.Solver.

7.4.3.3 solve()

```
def GenerateData.Heat.solve (
              self )
```

```
Method to solve the heat equation.

The problem is solved using the Discontinuous Galerkin method
and imposing non-homogeneous Neumann boundary condition at the inlet
and homogeneous Neumann boundary condition at the outlet and walls.

Returns:
    numpy.ndarray: Array of solutions at each time step.
```

Reimplemented from GenerateData.Solver.

7.4.4 Member Data Documentation

7.4.4.1 doplot

```
GenerateData.Heat.doplot
```

7.4.4.2 dt

```
GenerateData.Heat.dt
```

7.4.4.3 f

```
GenerateData.Heat.f
```

7.4.4.4 g

```
GenerateData.Heat.g
```

7.4.4.5 k

```
GenerateData.Heat.k
```

7.4.4.6 T

```
GenerateData.Heat.T
```

7.4.4.7 ts

```
GenerateData.Heat.ts
```

7.4.4.8 u0

```
GenerateData.Heat.u0
```

7.4.4.9 ut

```
GenerateData.Heat.ut
```

7.4.4.10 V

```
GenerateData.Heat.V
```

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

- scripts/GenerateData.py

7.5 MeshUtils.MeshCreator Class Reference

Public Member Functions

- def __init__ (self, args)
- def create_mesh (self, filename, output_dir)
- def convert_mesh (self, output_dir)
- def create_info_file (self, output_dir, meshname)

Public Attributes

- nmesh
- seed
- hmax
- hmin
- lc
- wmax
- wmin
- spacing
- nodes

7.5.1 Detailed Description

```
A class for creating mesh using Gmsh.

Attributes:
    nmesh: Number of meshes to create.
    seed: Seed for random number generation.
    hmax: Maximum value for the interfaces height.
    hmin: Minimum value for the interfaces height.
    lc: Characteristic length.
    wmax: Maximum value for the distance between nodes.
    wmin: Minimum value for the distance between nodes.
    spacing: Flag indicating whether the nodes are equispaced or not.
    nodes: Number of nodes.
```

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__()

```
def MeshUtils.MeshCreator.__init__ (
              self,
              args )
```

```
Constructor for the MeshCreator class.

Args:
    args: The arguments for the mesh creation.
```

7.5.3 Member Function Documentation

7.5.3.1 convert_mesh()

```
def MeshUtils.MeshCreator.convert_mesh (
              self,
              output_dir )
```

```
Convert the meshes in a given directory from msh to xml format.

Args:
    output_dir: The directory containing the meshes to convert.
```

7.5.3.2 create_info_file()

```
def MeshUtils.MeshCreator.create_info_file (
              self,
              output_dir,
              meshname )
```

```
Create a json file containing the mesh information.

Args:
    output_dir: The directory to save the json file.
```

7.5.3.3 create_mesh()

```
def MeshUtils.MeshCreator.create_mesh (
              self,
              filename,
              output_dir )
```

```
Create a mesh using Gmsh.

Args:
    filename: The name of the output file.
    output_dir: The directory to save the output file.
    plot: A flag indicating whether to plot the mesh.
```

7.5.4 Member Data Documentation

7.5.4.1 hmax

```
MeshUtils.MeshCreator.hmax
```

7.5.4.2 hmin

```
MeshUtils.MeshCreator.hmin
```

7.5.4.3 lc

```
MeshUtils.MeshCreator.lc
```

7.5.4.4 nmesh

```
MeshUtils.MeshCreator.nmesh
```

7.5.4.5 nodes

```
MeshUtils.MeshCreator.nodes
```

7.5.4.6 seed

```
MeshUtils.MeshCreator.seed
```

7.5.4.7 spacing

```
MeshUtils.MeshCreator.spacing
```

7.5.4.8 wmax

```
MeshUtils.MeshCreator.wmax
```

7.5.4.9 wmin

```
MeshUtils.MeshCreator.wmin
```

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

- scripts/MeshUtils.py

7.6 MeshUtils.MeshLoader Class Reference

Public Member Functions

- def __init__ (self, filename)
- def update_tags (self, tags={}, nodes=-1)
- def measure_definition (self)
- def plot_mesh (self)

Public Attributes

- meshfile
- mesh
- bounds
- face
- n
- h
- tags
- rename_boundaries
- rename_faces
- dS
- ds
- dx

7.6.1 Detailed Description

```
A class for loading mesh from a xml file to be used in FEniCS.

Attributes:
    meshfile: The name of the mesh file.
    mesh: The FEniCS mesh.
    bounds: FEniCS MeshFunction for the boundaries of the mesh.
    face: FEniCS MeshFunction for the faces of the mesh.
    n: The normal vector of the mesh.
    h: The characteristic length of the mesh.
    tags: A dictionary containing the tags of the mesh.
    rename_boundaries: FEniCS MeshFunction for the boundaries of the mesh with the tags.
    rename_faces: FEniCS MeshFunction for the faces of the mesh with the tags.
    dS: Measure for integration over external boundaries (inlet and outlet).
    ds: Measure for integration over internal boundaries (interface).
    dx: Measure for integration over faces.
```

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
def MeshUtils.MeshLoader.__init__ (
              self,
              filename )
```

```
Constructor for the MeshLoader class.

Args:
    filename: The name of the mesh file without extension.
```

7.6.3 Member Function Documentation

7.6.3.1 measure_definition()

```
def MeshUtils.MeshLoader.measure_definition (
              self )
```

```
Method to define the measures for the integration.
```

7.6.3.2 plot_mesh()

```
def MeshUtils.MeshLoader.plot_mesh (
              self )
```

```
Method to plot the mesh.
```

7.6.3.3 update_tags()

```
def MeshUtils.MeshLoader.update_tags (
              self,
              tags = {},
              nodes = -1 )
```

```
Method to save the tags of the boundaries and faces of the mesh.
If the number of nodes are provided, the tags are created automatically.
If the number of nodes are not provided, the tags must be provided.

Args:
    tags: A dictionary containing the tags of the mesh.
    nodes: The number of nodes of the mesh.

Returns:
    A dictionary containing the tags of the mesh.
```

7.6.4 Member Data Documentation

7.6.4.1 bounds

```
MeshUtils.MeshLoader.bounds
```

7.6.4.2 dS

```
MeshUtils.MeshLoader.dS
```

7.6.4.3 ds

```
MeshUtils.MeshLoader.ds
```

7.6.4.4 dx

```
MeshUtils.MeshLoader.dx
```

7.6.4.5 face

```
MeshUtils.MeshLoader.face
```

7.6.4.6 h

```
MeshUtils.MeshLoader.h
```

7.6.4.7 mesh

```
MeshUtils.MeshLoader.mesh
```

7.6.4.8 meshfile

```
MeshUtils.MeshLoader.meshfile
```

7.6.4.9 n

```
MeshUtils.MeshLoader.n
```

7.6.4.10 rename_boundaries

```
MeshUtils.MeshLoader.rename_boundaries
```

7.6.4.11 rename_faces

```
MeshUtils.MeshLoader.rename_faces
```

7.6.4.12 tags

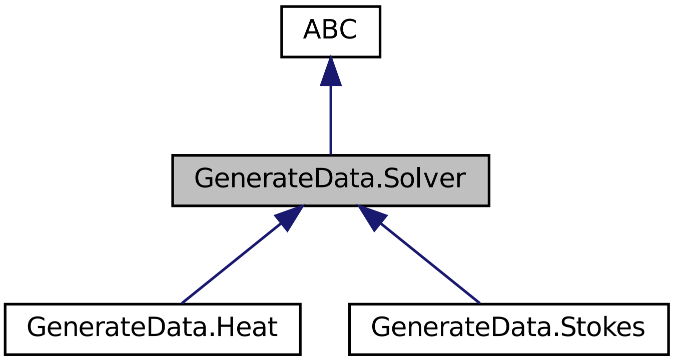
```
MeshUtils.MeshLoader.tags
```

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

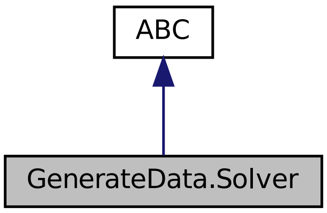
- scripts/MeshUtils.py

7.7 GenerateData.Solver Class Reference

Inheritance diagram for GenerateData.Solver:



Collaboration diagram for GenerateData.Solver:



Public Member Functions

- def __init__ (self, mesh)
- def set_parameters (self)
- def solve (self)
- def plot_solution (self)

Public Attributes

- mesh

7.7.1 Detailed Description

```
This class represents a solver for a variational problem on a given mesh.

This class is an abstract base class (ABC) and cannot be instantiated.

Attributes:
    mesh (Mesh): Mesh object.
```

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__()

```
def GenerateData.Solver.__init__ (
              self,
              mesh )
```

```
Initialize the Solver class.

Args:
    mesh (Mesh): Mesh object.
```

Reimplemented in GenerateData.Heat, and GenerateData.Stokes.

7.7.3 Member Function Documentation

7.7.3.1 plot_solution()

```
def GenerateData.Solver.plot_solution (
              self )
```

Reimplemented in GenerateData.Heat, and GenerateData.Stokes.

7.7.3.2 set_parameters()

```
def GenerateData.Solver.set_parameters (
              self )
```

Reimplemented in GenerateData.Heat, and GenerateData.Stokes.

7.7.3.3 solve()

```
def GenerateData.Solver.solve (
              self )
```

Reimplemented in GenerateData.Stokes, and GenerateData.Heat.

7.7.4 Member Data Documentation

7.7.4.1 mesh

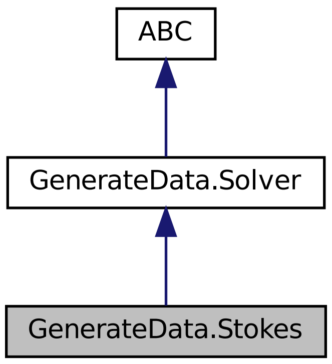
```
GenerateData.Solver.mesh
```

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

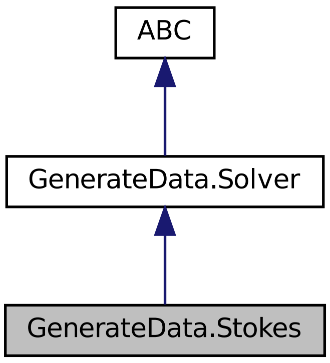
- scripts/GenerateData.py

7.8 GenerateData.Stokes Class Reference

Inheritance diagram for GenerateData.Stokes:



Collaboration diagram for GenerateData.Stokes:



Public Member Functions

- def __init__ (self, mesh, V, Q, rho, mu, U0, L0, inflow, f, dt, T, k, doplot=False)
- def set_parameters (self, V, Q, rho, mu, U0, L0, inflow, f, dt, T)
- def solve (self)
- def plot_solution (self, u, p)

Public Attributes

- V
- Q
- rho
- mu
- U0
- L0
- inflow
- f
- dt
- T
- doplot
- k
- ts
- ut
- pt
- u
- p

7.8.1 Detailed Description

```
Class representing a Stokes solver.

This class inherits from the Solver class and provides
methods to solve the Stokes equation.

Attributes:
    mesh (Mesh): Mesh object.
    V (FunctionSpace): Velocity function space.
    Q (FunctionSpace): Pressure function space.
    rho (float): Density of the fluid.
    mu (float): Dynamic viscosity of the fluid.
    U0 (float): Characteristic velocity of the fluid.
    L0 (float): Characteristic length of the fluid.
    inflow (float): Inflow rate of the fluid.
    f (Expression): Source term.
    dt (float): Time step.
    T (float): Final time.
    k (float): Reynolds number.
    doplot (bool): Flag indicating whether to
                   plot the solution at each time step.
    ts (numpy.ndarray): Array of time steps.
    ut (numpy.ndarray): Array of velocity solutions at each time step.
    pt (numpy.ndarray): Array of pressure solutions at each time step.
```

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
def GenerateData.Stokes.__init__ (
              self,
              mesh,
              V,
              Q,
              rho,
              mu,
              U0,
              L0,
              inflow,
              f,
              dt,
              T,
              k,
              doplot = False )
```

```
Initialize the GenerateData class.

Args:
    V (FunctionSpace): Velocity function space.
    Q (FunctionSpace): Pressure function space.
    rho (float): Density of the fluid.
    mu (float): Dynamic viscosity of the fluid.
    U0 (float): Characteristic velocity of the fluid.
    L0 (float): Characteristic length of the fluid.
    inflow (float): Inflow rate of the fluid.
    f (Expression): Source term.
    dt (float): Time step.
    T (float): Final time.
    doplot (bool): Flag indicating whether to plot
                   the solution at each time step.
```

Reimplemented from GenerateData.Solver.

7.8.3 Member Function Documentation

7.8.3.1 plot_solution()

```
def GenerateData.Stokes.plot_solution (
              self,
              u,
              p )
```

```
Plot the solution u and p.

Args:
    u: The velocity solution.
    p: The pressure solution.
```

Reimplemented from GenerateData.Solver.

7.8.3.2 set_parameters()

```
def GenerateData.Stokes.set_parameters (
              self,
              V,
              Q,
              rho,
              mu,
              U0,
              L0,
              inflow,
              f,
              dt,
              T )
```

```
Set the parameters for the simulation.

Args:
    V (float): Velocity of the fluid.
    Q (float): Flow rate of the fluid.
    rho (float): Density of the fluid.
    mu (float): Viscosity of the fluid.
    U0 (float): Initial velocity of the fluid.
    L0 (float): Initial length of the fluid.
    inflow (float): Inflow rate of the fluid.
    f (float): Force applied to the fluid.
    dt (float): Time step for the simulation.
    T (float): Total time for the simulation.
```

Reimplemented from GenerateData.Solver.

7.8.3.3 solve()

```
def GenerateData.Stokes.solve (
              self )
```

```
Solve the Stokes equation.
```

Reimplemented from GenerateData.Solver.

7.8.4 Member Data Documentation

7.8.4.1 doplot

```
GenerateData.Stokes.doplot
```

7.8.4.2 dt

```
GenerateData.Stokes.dt
```

7.8.4.3 f

```
GenerateData.Stokes.f
```

7.8.4.4 inflow

```
GenerateData.Stokes.inflow
```

7.8.4.5 k

```
GenerateData.Stokes.k
```

7.8.4.6 L0

```
GenerateData.Stokes.L0
```

7.8.4.7 mu

```
GenerateData.Stokes.mu
```

7.8.4.8 p

```
GenerateData.Stokes.p
```

7.8.4.9 pt

```
GenerateData.Stokes.pt
```

7.8.4.10 Q

```
GenerateData.Stokes.Q
```

7.8.4.11 rho

```
GenerateData.Stokes.rho
```

7.8.4.12 T

```
GenerateData.Stokes.T
```

7.8.4.13 ts

```
GenerateData.Stokes.ts
```

7.8.4.14 u

```
GenerateData.Stokes.u
```

7.8.4.15 U0

```
GenerateData.Stokes.U0
```

7.8.4.16 ut

```
GenerateData.Stokes.ut
```

7.8.4.17 V

```
GenerateData.Stokes.V
```

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

- scripts/GenerateData.py

Chapter 8

File Documentation

8.1 README.md File Reference

8.2 scripts/datagen.py File Reference

Namespaces

- namespace datagen

Variables

- int datagen.ngraphs = 2
- string datagen.output_dir = "data/graphs_test/"
- string datagen.mesh_dir = "data/mesh_test/"
- datagen.mesh_info = json.load(open(mesh_dir + 'mesh_info.json'))
- datagen.mesh_name = mesh_info['mesh_name']
- datagen.nmesh = mesh_info['nmesh']
- datagen.nodes = mesh_info['nodes']
- datagen.f = Constant(0.0)
- datagen.g = Expression('a∗exp(-(t-b)∗(t-b)/c/c)',degree=2,a=5,b=2.5,c=1,t=0)
- datagen.u0 = Expression('0.0',degree=0)
- int datagen.T = 5
- int datagen.timesteps = 50
- int datagen.dt = T/timesteps
- int datagen.kmax = 100
- int datagen.kmin = 1
- datagen.imesh = np.random.randint(0,nmesh)
- datagen.mesh_load = mutil.MeshLoader(mesh_dir + mesh_name + f"_{imesh}")
- datagen.mesh = mesh_load.mesh
- datagen.bounds = mesh_load.bounds
- datagen.face = mesh_load.face
- datagen.V = FunctionSpace(mesh_load.mesh,"DG",1)
- datagen.k = round(np.random.uniform(kmin, kmax),2)
- datagen.heat_gaussian = gd.Heat(mesh_load,V,k,f,u0,dt,T,g)
- datagen.data = gd.DataHeat(heat_gaussian,mesh_load)

8.3 scripts/GenerateData.py File Reference

Classes

- class GenerateData.Solver
- class GenerateData.Stokes
- class GenerateData.Heat
- class GenerateData.DataGenerator
- class GenerateData.DataNS
- class GenerateData.DataHeat

Namespaces

- namespace GenerateData

  This file implements a solver class to solve a variational problem and a data generator class to store the data and the solutions of the problem solved in a dgl graph.

8.4 scripts/GenerateGraph.py File Reference

Namespaces

- namespace GenerateGraph

  This file contains functions to generate DGL graphs from data generated by the GenerateData script.

Functions

- def GenerateGraph.generate_graph (point_data, points, edges_data, edges1, edges2)
- def GenerateGraph.add_field (graph, field, field_name, offset=0)
- def GenerateGraph.save (graph, filename, output_dir="../data/graphs/")

8.5 scripts/MeshUtils.py File Reference

Classes

- class MeshUtils.MeshCreator
- class MeshUtils.MeshLoader

Namespaces

- namespace MeshUtils

  This file contains utilities for creating and loading meshes.

Variables

- MeshUtils.parser = argparse.ArgumentParser(description='Mesh creation')
- MeshUtils.help
- MeshUtils.type
- MeshUtils.int
- MeshUtils.default
- MeshUtils.float
- MeshUtils.bool
- MeshUtils.args = parser.parse_args()
- string MeshUtils.filename = "test_mesh"
- string MeshUtils.output_dir = "data/mesh_test/"
- MeshUtils.mesh_creator = MeshCreator(args)

8.6 scripts/test_installation.py File Reference

Namespaces

- namespace test_installation

Variables

- list test_installation.packages = ['numpy', 'matplotlib', 'torch', 'dolfin', 'meshio', 'dgl', 'scipy', 'tqdm','jupyter']
- bool test_installation.flag = False

Index