A flexible Object-Oriented First Order 3-D Finite-Difference algorithm Chorin-Temam based solver for Brinkman Equation

Generated by Doxygen 1.9.1

Chapter 1

Finite Differences Navier-Stokes Solver

This project implements a solver based on finite differences for the Navier-Stokes equations. The solver is designed to handle multiple physics problems, with the final goal of solving Brinkman flow for a generic geometry. It allows computations on a default cubic domain as well as on an input mesh using the Brinkman penalization method.

Our multiphysics solver is capable of solving a variety of problems, including Navier-Stokes and parabolic problems. This flexibility enables a broad range of simulations in fluid dynamics.

1.1 Requirements

To compile and run the project, make sure you have the following tools and libraries installed and configured:

- **Compiler**: GCC with C++20 support
- **Libraries**:
  - PETSc
  - VTK
  - Eigen
- **Environment modules:** `https://github.com/pcafrica/mk`

1.2 Installation

1. **Load the required modules**:
   Before compiling or running the program, load the necessary modules. Ensure that your environment is correctly set up by running:
   ```
   source /u/sw/etc/bash.bashrc
   module load gcc-glibc
   module load eigen
   module load vtk
   module load petsc
   ```
2. **Set config_problem.hpp**: This file contains the configuration for the problem to solve. User can set their parameters and set more flow variables if needed. This file is well documented and should receive careful consideration by the user. Current configuration is set for an analytical solution of of 3D Navier-Stokes Equations with no Brinkman Penalization (cfr. Doxygen)

3. **Compile the project**: After loading the modules and properly set flow and domain parameters, compile the project according to the chosen solver by running: ```bash make clean make FILES="navier_stokes.cpp" #for Navier-Stokes and Brinkman problem make FILES="stokes.cpp" #for Navier-Stokes and Brinkman problem make FILES="inviscid_euler.cpp" #for Navier-Stokes and Brinkman problem make FILES="advection_↩ diffusion.cpp" #for Navier-Stokes and Brinkman problem make FILES="parabolic.cpp" #for parabolic problems ```

4. **Usage** To run the solver, execute the following command: ```bash ./bin/main ``For the Brinkman solver, the name of the.stlfile must be specified. An example mesh is provided. To run the solver on the example mesh, execute the following command: ``bash ./bin/main caroitd.stl ``The program will compute the solution and store the results in theresults` directory.

1.3 Simulation Results

Velocity Magnitude for Navier-Stokes incompressible flow at Re=1	**Velocity Magnitude for Navier-Stokes incompressible flow at Re=2000**
Parabolic Flow with mu=10	**Brinkman Flow Simulation Re=200**

These examples showcase the solver's ability to handle various flow regimes and conditions.

Developed by **Davide Galbiati** and **Alessandra Gotti**

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

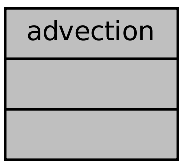
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 advection Class Reference

Solves the Advection Diffusion evolutionary incompressible equations. It consists of a transport problem for each velocity component and a parabolic problem for each velocity component. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.
Collaboration diagram for advection:



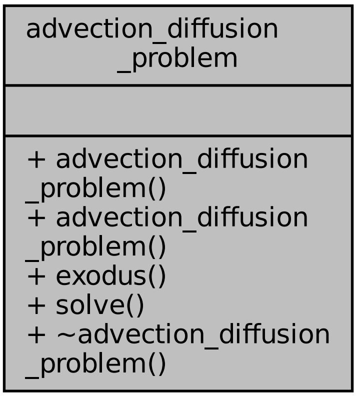
4.1.1 Detailed Description

Solves the Advection Diffusion evolutionary incompressible equations. It consists of a transport problem for each velocity component and a parabolic problem for each velocity component. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.
The documentation for this class was generated from the following file:

- include/advection_diffusion.hpp

4.2 advection_diffusion_problem Class Reference

Collaboration diagram for advection_diffusion_problem:



Public Member Functions

- advection_diffusion_problem (DM const &dmGrid_staggered_x, DM const &dmGrid_staggered_y, DM const &dmGrid_staggered_z, DM const &dmGrid_centered, DM const &dmGrid_cent_rich, DM const &dmGrid_↩ stag_transp, DM const dmGrid_shift_transp, Vec const &U_up, Vec const &V_up, Vec const W_up)

    Constructor that initializes the Advection-Diffusion problem with given grids and velocity fields. This costructor has been defined for consistency and if future implementations will require solving Advection-Diffusion in a broader context. For our applications only a stand-alone constructor is used.
- advection_diffusion_problem ()

    Default constructor that initializes stand-alone Advection-Diffusion problem with automatically created grids.
- PetscErrorCode exodus (size_t i)

    Exports simulation in .vtk format format for visualization of x,y,z-componets, pressure and magnitude.
- PetscErrorCode const solve ()

    Solves the Advection-Diffusion equations leveraging parabolic_problem_x, y, z, transport_problem_x, y, z, and poisson_problem classes.
- ∼advection_diffusion_problem ()

    Destructor to clean up allocated resources. Automatically calls sub-problems destructors. After destruction, a message is printed.

4.2.1 Detailed Description

Definition at line 43 of file advection_diffusion.hpp.
The documentation for this class was generated from the following files:

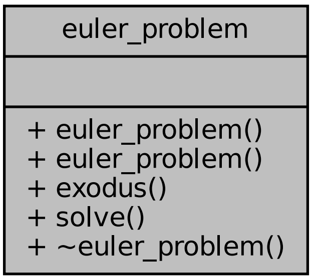
- include/advection_diffusion.hpp
- src/advection_diffusion.cpp

4.3 euler_problem Class Reference

Solves the Euler evolutionary incompressible equations, applying a first order Chorin-Temam algorithm. It consists of a transport problem for each velocity component and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation DOES NOT ALLOW for Brinkamn penalty method.

`#include "inviscid_euler.hpp"`

Collaboration diagram for euler_problem:



Public Member Functions

- euler_problem (DM const &dmGrid_staggered_x, DM const &dmGrid_staggered_y, DM const &dmGrid_staggered_z, DM const &dmGrid_centered, DM const &dmGrid_cent_rich, DM const &dmGrid_stag_transp, DM const dmGrid_shift_transp, Vec const &U_up, Vec const &V_up, Vec const W_up)

  Constructor that initializes the Euler problem with given grids and velocity fields. This costructor has been defined for consistency and if future implementations will require solving Euler in a broader context. For our applications only a stand-alone constructor is used.
- euler_problem ()

  Default constructor that initializes stand-alone Euler problem with automatically created grids.
- PetscErrorCode exodus (size_t i)

  Exports simulation in .vtk format format for visualization of x,y,z-componets, pressure and magnitude.
- PetscErrorCode const solve ()

  Solves the Euler equations leveraging parabolic_problem_x, y, z, transport_problem_x, y, z, and poisson_problem classes.
- ∼euler_problem ()

  Destructor to clean up allocated resources. Automatically calls sub-problems destructors. After destruction, a message is printed.

4.3.1 Detailed Description

Solves the Euler evolutionary incompressible equations, applying a first order Chorin-Temam algorithm. It consists of a transport problem for each velocity component and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation DOES NOT ALLOW for Brinkamn penalty method.

Definition at line 43 of file inviscid_euler.hpp.

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

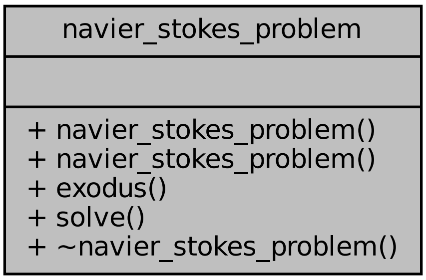
- include/inviscid_euler.hpp
- src/inviscid_euler.cpp

4.4 navier_stokes_problem Class Reference

Solves the Navier-Stokes evolutionary incompressible Navier-Stokes equations, applying a first order Chorin-Temam algorithm. It consists of a transport problem for each velocity component, a parabolic problem for each velocity component, and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.

`#include "navier_stokes.hpp"`

Collaboration diagram for navier_stokes_problem:



Public Member Functions

- navier_stokes_problem (DM const &dmGrid_staggered_x, DM const &dmGrid_staggered_y, DM const &dmGrid_staggered_z, DM const &dmGrid_centered, DM const &dmGrid_cent_rich, DM const &dmGrid_stag_transp, DM const dmGrid_shift_transp, Vec const &U_up, Vec const &V_up, Vec const W_up)

  Constructor that initializes the Navier-Stokes problem with given grids and velocity fields. This costructor has been defined for consistency and if future implementations will require solving Navier-Stokes in a broader context. For our applications only a stand-alone constructor is used.
- navier_stokes_problem ()

  Default constructor that initializes stand-alone Navier-Stokes problem with automatically created grids.
- PetscErrorCode exodus (size_t i)

  Exports simulation in .vtk format format for visualization of x,y,z-componets, pressure and magnitude.
- PetscErrorCode const solve ()

  Solves the Navier-Stokes equations leveraging parabolic_problem_x, y, z, transport_problem_x, y, z, and poisson_problem classes.
- ∼navier_stokes_problem ()

  Destructor to clean up allocated resources. Automatically calls sub-problems destructors. After destruction, a message is printed.

4.4.1 Detailed Description

Solves the Navier-Stokes evolutionary incompressible Navier-Stokes equations, applying a first order Chorin-Temam algorithm. It consists of a transport problem for each velocity component, a parabolic problem for each velocity component, and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.

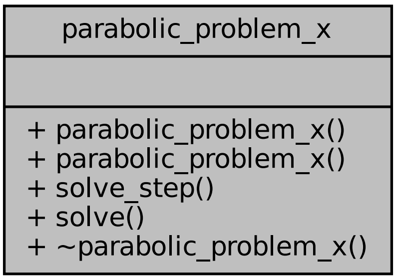
Definition at line 44 of file navier_stokes.hpp.

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

- include/navier_stokes.hpp
- src/navier_stokes.cpp

4.5 parabolic_problem_x Class Reference

Represents a parabolic problem in the x-direction.
`#include "parabolic.hpp"`
Collaboration diagram for parabolic_problem_x:



Public Member Functions

- parabolic_problem_x (DM const &dmGrid)

  Constructor that initializes the problem with a given grid. Use when parabolic problem is just a step of a bigger problem, like in Chorin-Temam method.
- parabolic_problem_x ()

  Default constructor for stand-alone problem.
- PetscErrorCode solve_step (PetscReal const &theta, std::optional< std::reference_wrapper< Vec >> U↩
  _up_opt=std::nullopt)

  Performs a single time step of the numerical solution.
- PetscErrorCode solve ()

  Solves the entire parabolic problem in the x-direction.
- ~parabolic_problem_x ()

  Destructor to clean up allocated resources. Fundamental in PETSc implementation lo avoid leaks and unexpected RAM overhead.

4.5.1 Detailed Description

Represents a parabolic problem in the x-direction.
This class solves a generic evolutionary diffusive problem in the x-direction with fully Dirichlet bc's. Boundary conditions are imposed by means of a reference solution. Discretization is performed on a staggered grid and in x-direction only, meaning that variables are located in position LEFT and RIGHT. Linear problem is solved by means of a PETSc KSP solver, with GMRES and Jacobi preconditioner.
Definition at line 44 of file parabolic.hpp.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 parabolic_problem_x()

```
parabolic_problem_x::parabolic_problem_x (
            DM const & dmGrid )
```

Constructor that initializes the problem with a given grid. Use when parabolic problem is just a step of a bigger problem, like in Chorin-Temam method.

Parameters

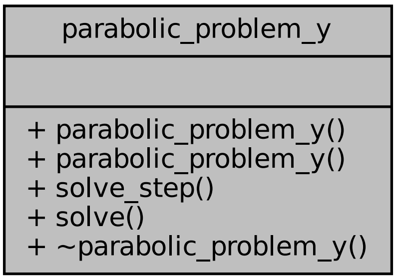
dmGrid	staggered grid petsc-object must be already be defined.

Definition at line 3 of file parabolic.cpp.
The documentation for this class was generated from the following files:

- include/parabolic.hpp
- src/parabolic.cpp

4.6 parabolic_problem_y Class Reference

Represents a parabolic problem in the y-direction.
`#include "parabolic.hpp"`
Collaboration diagram for parabolic_problem_y:



Public Member Functions

- parabolic_problem_y (DM const &dmGrid)

  Constructor that initializes the problem with a given grid. Use when parabolic problem is just a step of a bigger problem, like in Chorin-Temam method.
- parabolic_problem_y ()

  Default constructor for stand-alone problem.
- PetscErrorCode solve_step (PetscReal const &theta, std::optional< std::reference_wrapper< Vec >> V_↩ up_opt=std::nullopt)

  Performs a single time step of the numerical solution.
- PetscErrorCode solve ()

  Solves the entire parabolic problem in the Y-direction.
- ∼parabolic_problem_y ()

  Destructor to clean up allocated resources. Fundamental in PETSc implementation lo avoid leaks and unexpected RAM overhead.

4.6.1 Detailed Description

Represents a parabolic problem in the y-direction.
This class solves a generic evolutionary diffusive problem in the y-direction with fully Dirichlet bc's. Boundary conditions are imposed by means of a reference solution. Discretization is performed on a staggered grid and in x-direction only, meaning that variables are located in position DOWN and UP. Linear problem is solved by means of a PETSc KSP solver, with GMRES and Jacobi preconditioner.
Definition at line 112 of file parabolic.hpp.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 parabolic_problem_y()

```
parabolic_problem_y::parabolic_problem_y (
        DM const & dmGrid )
```

Constructor that initializes the problem with a given grid. Use when parabolic problem is just a step of a bigger problem, like in Chorin-Temam method.

Parameters

dmGrid	staggered grid petsc-object must be already be defined.

Definition at line 683 of file parabolic.cpp.

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

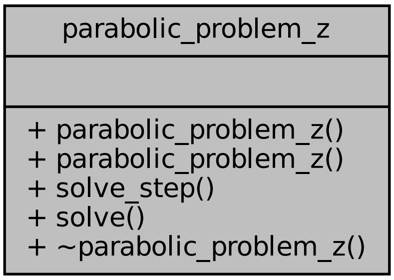
- include/parabolic.hpp
- src/parabolic.cpp

4.7 parabolic_problem_z Class Reference

Represents a parabolic problem in the z-direction.

`#include "parabolic.hpp"`

Collaboration diagram for parabolic_problem_z:



Public Member Functions

- parabolic_problem_z (DM const &dmGrid)

  Constructor that initializes the problem with a given grid. Use when parabolic problem is just a step of a bigger problem, like in Chorin-Temam method.
- parabolic_problem_z ()

  Default constructor for stand-alone problem.
- PetscErrorCode solve_step (PetscReal const &theta, std::optional< std::reference_wrapper< Vec >> W↩_up_opt=std::nullopt)

  Performs a single time step of the numerical solution.
- PetscErrorCode solve ()

  Solves the entire parabolic problem in the Z-direction.
- ∼parabolic_problem_z ()

Destructor to clean up allocated resources. Fundamental in PETSc implementation lo avoid leaks and unexpected RAM overhead.

4.7.1 Detailed Description

Represents a parabolic problem in the z-direction.
This class solves a generic evolutionary diffusive problem in the z-direction with fully Dirichlet bc's. Boundary conditions are imposed by means of a reference solution. Discretization is performed on a staggered grid and in x-direction only, meaning that variables are located in position BACK and FRONT. Linear problem is solved by means of a PETSc KSP solver, with GMRES and Jacobi preconditioner.
Definition at line 182 of file parabolic.hpp.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 parabolic_problem_z()

```
parabolic_problem_z::parabolic_problem_z (
            DM const & dmGrid )
```

Constructor that initializes the problem with a given grid. Use when parabolic problem is just a step of a bigger problem, like in Chorin-Temam method.

Parameters

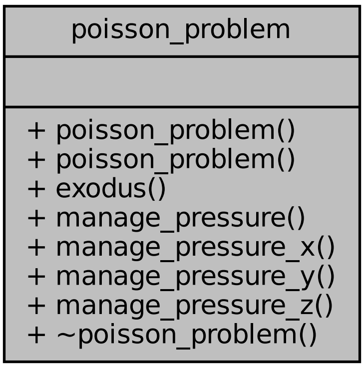
dmGrid	staggered grid petsc-object must be already be defined.

Definition at line 1351 of file parabolic.cpp.
The documentation for this class was generated from the following files:

- include/parabolic.hpp
- src/parabolic.cpp

4.8 poisson_problem Class Reference

Represents a Poisson equation solver for pressure correction in fluid simulations.
`#include "poisson.hpp"`
Collaboration diagram for poisson_problem:



Public Member Functions

- poisson_problem (DM const &dmGrid_staggered_x, DM const &dmGrid_staggered_y, DM const &dmGrid↩_staggered_z, DM const &dmGrid_centered, DM const &dmGrid_cent_rich)

    Constructor to initialize the Poisson problem with pre-allocated grids.
- poisson_problem ()

    Constructor to solve stand-alone problem.
- PetscErrorCode const exodus (size_t i)

    Exports simulation results in .vtk format for visualization.
- PetscErrorCode const manage_pressure (std::optional< std::reference_wrapper< Vec >> U_opt=std↩::nullopt, std::optional< std::reference_wrapper< Vec >> V_up_opt=std::nullopt, std::optional< std↩::reference_wrapper< Vec >> W_up_opt=std::nullopt, std::optional< std::reference_wrapper< Vec >> P_opt=std::nullopt)

    Solves the elliptic Poisson equation with GMRES and multigrid preconditioner.
- PetscErrorCode const manage_pressure_x (std::optional< std::reference_wrapper< Vec >> P_opt=std↩::nullopt, std::optional< std::reference_wrapper< Vec >> P_x_opt=std::nullopt)

    Assembles the pressure derivative in the x-direction on the staggered grid.
- PetscErrorCode const manage_pressure_y (std::optional< std::reference_wrapper< Vec >> P_opt=std↩::nullopt, std::optional< std::reference_wrapper< Vec >> P_y_opt=std::nullopt)

    Assembles the pressure derivative in the y-direction on the staggered grid.
- PetscErrorCode const manage_pressure_z (std::optional< std::reference_wrapper< Vec >> P_opt=std↩::nullopt, std::optional< std::reference_wrapper< Vec >> P_z_opt=std::nullopt)

    Assembles the pressure derivative in the z-direction on the staggered grid.
- ∼poisson_problem ()

    Destructor to clean up allocated resources. required by PETSc management of native objects.

4.8.1 Detailed Description

Represents a Poisson equation solver for pressure correction in fluid simulations.
This class solves the Poisson equation for a variabble like pressure, with homogeneous Neumann boundary conditions. It manages the assembly of matrices, calculation of divergence. In our framework, it is used to solve the pressure correction in the Navier-Stokes equations. For our purposes, enforcing compatibility condition was not required. Beware that for a stand-alone problem, compatible-to-null-bc's source must be provided.
Definition at line 42 of file poisson.hpp.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 poisson_problem()

```
poisson_problem::poisson_problem (
            DM const & dmGrid_staggered_x,
            DM const & dmGrid_staggered_y,
            DM const & dmGrid_staggered_z,
            DM const & dmGrid_centered,
            DM const & dmGrid_cent_rich )
```

Constructor to initialize the Poisson problem with pre-allocated grids.

Parameters

dmGrid_staggered↩_x	Grid for staggered x-direction.
dmGrid_staggered↩_y	Grid for staggered y-direction.
dmGrid_staggered↩_z	Grid for staggered z-direction.

Parameters

dmGrid_centered	Grid for centered formulation.
dmGrid_cent_rich	Grid for refined pressure computation.

Definition at line 4 of file poisson.cpp.

4.8.3 Member Function Documentation

4.8.3.1 manage_pressure()

```
PetscErrorCode const poisson_problem::manage_pressure (
            std::optional< std::reference_wrapper< Vec >> U_opt = std::nullopt,
            std::optional< std::reference_wrapper< Vec >> V_up_opt = std::nullopt,
            std::optional< std::reference_wrapper< Vec >> W_up_opt = std::nullopt,
            std::optional< std::reference_wrapper< Vec >> P_opt = std::nullopt )
```

Solves the elliptic Poisson equation with GMRES and multigrid preconditioner.

Parameters

U_up	Velocity field in the x-direction.
V_up	Velocity field in the y-direction.
W_up	Velocity field in the z-direction.
P	Pressure field.

Definition at line 1380 of file poisson.cpp.

4.8.3.2 manage_pressure_x()

```
PetscErrorCode const poisson_problem::manage_pressure_x (
            std::optional< std::reference_wrapper< Vec >> P_opt = std::nullopt,
            std::optional< std::reference_wrapper< Vec >> P_x_opt = std::nullopt )
```

Assembles the pressure derivative in the x-direction on the staggered grid.

Parameters

P	Pressure field.
P_x	Updated pressure derivative in the x-direction.

Definition at line 1313 of file poisson.cpp.

4.8.3.3 manage_pressure_y()

```
PetscErrorCode const poisson_problem::manage_pressure_y (
            std::optional< std::reference_wrapper< Vec >> P_opt = std::nullopt,
            std::optional< std::reference_wrapper< Vec >> P_y_opt = std::nullopt )
```

Assembles the pressure derivative in the y-direction on the staggered grid.

Parameters

P	Pressure field.
P_y	Updated pressure derivative in the y-direction.

Definition at line 1337 of file poisson.cpp.

4.8.3.4 manage_pressure_z()

```
PetscErrorCode const poisson_problem::manage_pressure_z (
            std::optional< std::reference_wrapper< Vec >> P_opt = std::nullopt,
            std::optional< std::reference_wrapper< Vec >> P_z_opt = std::nullopt )
```

Assembles the pressure derivative in the z-direction on the staggered grid.

Parameters

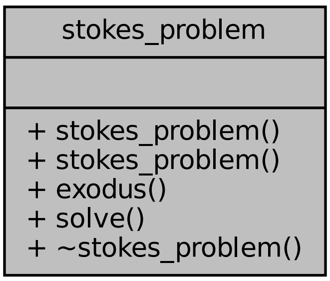
P	Pressure field.
P←↩ _z	Updated pressure derivative in the z-direction.

Definition at line 1358 of file poisson.cpp.
The documentation for this class was generated from the following files:

- include/poisson.hpp
- src/poisson.cpp

4.9 stokes_problem Class Reference

Solves the Stokes evolutionary incompressible Stokes equations, applying a first order Chorin-Temam algorithm. It consists of a parabolic problem for each velocity component, and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.
`#include "stokes.hpp"`
Collaboration diagram for stokes_problem:



Public Member Functions

- stokes_problem (DM const &dmGrid_staggered_x, DM const &dmGrid_staggered_y, DM const &dmGrid←↩ _staggered_z, DM const &dmGrid_centered, DM const &dmGrid_cent_rich, Vec const &U_up, Vec const &V_up, Vec const W_up)

  Constructor that initializes the Stokes problem with given grids and velocity fields. This costructor has been defined for consistency and if future implementations will require solving Stokes in a broader context. For our applications only a stand-alone constructor is used.
- stokes_problem ()

Constructor for stand-alone problem.

- PetscErrorCode **exodus** (size_t i)
- PetscErrorCode const solve ()

  Solves the Stokes equations leveraging parabolic_problem_x, y, z, transport_problem_x, y, z, and poisson_problem classes.
- ~stokes_problem ()

  Destructor to clean up allocated resources. Automatically calls sub-problems destructors. After destruction, a message is printed.

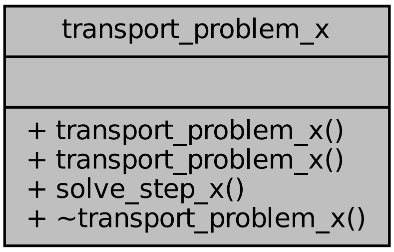
4.9.1 Detailed Description

Solves the Stokes evolutionary incompressible Stokes equations, applying a first order Chorin-Temam algorithm. It consists of a parabolic problem for each velocity component, and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.
Definition at line 43 of file stokes.hpp.
The documentation for this class was generated from the following files:

- include/stokes.hpp
- src/stokes.cpp

4.10 transport_problem_x Class Reference

Represents a transport problem in the x-direction.
`#include "transport.hpp"`
Collaboration diagram for transport_problem_x:



Public Member Functions

- transport_problem_x (DM const &dmGrid_Shifted, DM const &dmGrid_Staggered, DM const &dmGrid_Centered)

  Constructor initializing the transport problem with given grids.
- transport_problem_x ()

  Default constructor for stand-alone transport problem.
- PetscErrorCode const solve_step_x (PetscScalar const &theta, std::optional< std::reference_wrapper< Vec >> U_n_opt=std::nullopt, std::optional< std::reference_wrapper< Vec >> V_n_opt=std::nullopt, std::optional< std::reference_wrapper< Vec >> W_n_opt=std::nullopt)

  Performs a single time step for solving the transport equation.
- ~transport_problem_x ()

  Destructor to clean up allocated resources.

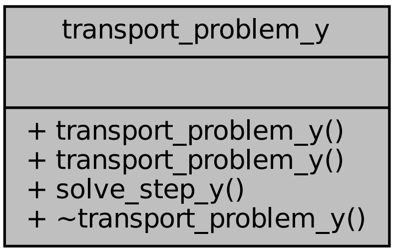
4.10.1 Detailed Description

Represents a transport problem in the x-direction.
This class solves an advection-transport problem in the x-direction using 2nd order centered differences and a fully explicit approach.
Definition at line 41 of file transport.hpp.
The documentation for this class was generated from the following files:

- include/transport.hpp
- src/transport.cpp

4.11 transport_problem_y Class Reference

Represents a transport problem in the y-direction.
`#include "transport.hpp"`
Collaboration diagram for transport_problem_y:



Public Member Functions

- transport_problem_y (DM const &dmGrid_Shifted, DM const &dmGrid_Staggered, DM const &dmGrid_↩ Centered)

    Constructor initializing the transport problem with given grids.
- transport_problem_y ()

    Default constructor for stand-alone transport problem.
- PetscErrorCode const solve_step_y (PetscScalar const &theta, std::optional< std::reference_wrapper< Vec >> U_n_opt=std::nullopt, std::optional< std::reference_wrapper< Vec >> V_n_opt=std::nullopt, std↩ ::optional< std::reference_wrapper< Vec >> W_n_opt=std::nullopt)

    Performs a single time step for solving the transport equation.
- ∼transport_problem_y ()

    Destructor to clean up allocated resources.

4.11.1 Detailed Description

Represents a transport problem in the y-direction.
This class solves an advection-transport problem in the y-direction using 2nd order centered differences and a fully explicit approach.
Definition at line 144 of file transport.hpp.
The documentation for this class was generated from the following files:

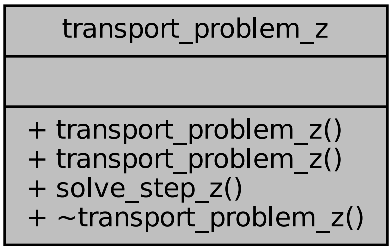
- include/transport.hpp
- src/transport.cpp

4.12 transport_problem_z Class Reference

Represents a transport problem in the z-direction.

`#include "transport.hpp"`

Collaboration diagram for transport_problem_z:



Public Member Functions

- transport_problem_z (DM const &dmGrid_Shifted, DM const &dmGrid_Staggered, DM const &dmGrid_Centered)

  Constructor initializing the transport problem with given grids.
- transport_problem_z ()

  Default constructor for stand-alone transport problem.
- PetscErrorCode const solve_step_z (PetscScalar const &theta, std::optional< std::reference_wrapper< Vec >> U_n_opt=std::nullopt, std::optional< std::reference_wrapper< Vec >> V_n_opt=std::nullopt, std::optional< std::reference_wrapper< Vec >> W_n_opt=std::nullopt)

  Performs a single time step for solving the transport equation.
- ~transport_problem_z ()

  Destructor to clean up allocated resources.

4.12.1 Detailed Description

Represents a transport problem in the z-direction.

This class solves an advection-transport problem in the z-direction using 2nd order centered differences and a fully explicit approach.

Definition at line 233 of file transport.hpp.

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

- include/transport.hpp
- src/transport.cpp

Chapter 5

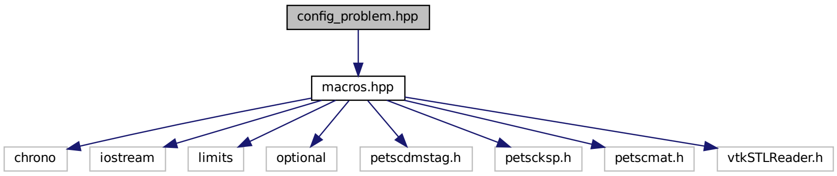
File Documentation

5.1 config_problem.hpp File Reference

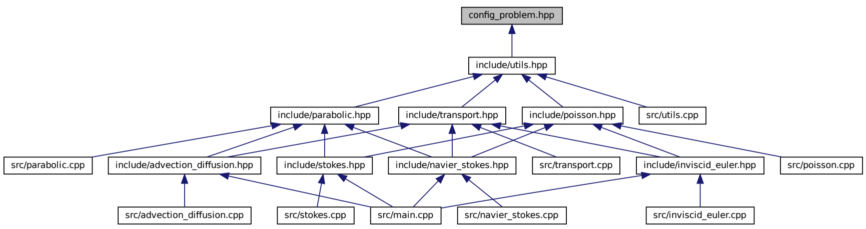
Configuration file for setting up problem parameters.

`#include "macros.hpp"`

Include dependency graph for config_problem.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- constexpr PetscReal problem_setting::uxRef (PetscReal const &x, PetscReal const &y, PetscReal const &z, PetscReal const &theta)

    Computes the reference solution for the problem in the x-direction.
- constexpr PetscReal problem_setting::uyRef (PetscReal const &x, PetscReal const &y, PetscReal const &z, PetscReal const &theta)

    Computes the reference solution for the problem in the y-direction.
- constexpr PetscReal problem_setting::uzRef (PetscReal const &x, PetscReal const &y, PetscReal const &z, PetscReal const &theta)

    Computes the reference solution for the problem in the z-direction.
- constexpr PetscReal **problem_setting::pRef** (PetscReal const &x, PetscReal const &y, PetscReal const &z, PetscReal const &theta)

Variables

- constexpr const char * problem_setting::base_path = "results/"

  Set this as base path to store results. Default is "results/".
- constexpr bool problem_setting::brinkman {false}

  Set flag to (dis)able brinkman penalization. Required for complex geometry handling.
- constexpr bool problem_setting::monitor_convergence {true}

  Set flag to (dis)able convergence error of KSP solver.
- constexpr bool problem_setting::check_convergence {true}

  Set flag to (dis)able convergence error of variable. De-activate for non-analytical solutions, as results would be meaningless.
- constexpr PetscInt problem_setting::nx {32}

  Set the number of elements in x-direction.
- constexpr PetscInt problem_setting::ny {32}

  Set the number of elements in y-direction.
- constexpr PetscInt problem_setting::nz {32}

  Set the number of elements in z-direction.
- constexpr PetscReal problem_setting::Lx_0 {-0.5}

  Set the domain limits in x, y and z directions. For Brinkman FLow we provided an already set geometry which comes the the commented domanin limits. This choice is deliberate and intended to only provide an example. Users should be in charge to provide their own geometry with the correct domain limits.
- constexpr PetscReal **problem_setting::Ly_0** {-0.5}
- constexpr PetscReal **problem_setting::Lz_0** {-0.5}
- constexpr PetscReal **problem_setting::Lx** {0.5}
- constexpr PetscReal **problem_setting::Ly** {0.5}
- constexpr PetscReal **problem_setting::Lz** {0.5}
- constexpr PetscReal problem_setting::dt {0.00625/2}

  Set time-step.
- constexpr PetscReal problem_setting::iter {32}

  Set final time.
- PetscReal problem_setting::theta

  Set starting time.
- constexpr PetscReal problem_setting::Re {1}

  Set Reynolds number. For non advective problems, set mu = 1/Re (adimensional framework)
- constexpr PetscReal problem_setting::a = pi / 4

  Set flow parameters. You can declare as many constexpr variables as you need.
- constexpr PetscReal **problem_setting::d** = 1.5 * pi

5.1.1 Detailed Description

Configuration file for setting up problem parameters.
This file defines the parameters for simulations. Before starting any simulation, careful consideration of the parameters is required. It is recommended to work in a dimensionless framework.

5.1.2 Function Documentation

5.1.2.1 uxRef()

```
constexpr PetscReal problem_setting::uxRef (
            PetscReal const & x,
            PetscReal const & y,
            PetscReal const & z,
            PetscReal const & theta )  [constexpr]
```

Computes the reference solution for the problem in the x-direction.
This function provides an analytical reference solution for the given problem in the x-direction. It is time-dependent and useful for benchmarking numerical methods.

Parameters

x	X-coordinate.
y	Y-coordinate.
z	Z-coordinate.
theta	Time-dependent parameter.

Returns

Computed reference solution in the x-direction.

Definition at line 107 of file config_problem.hpp.

5.1.2.2 uyRef()

```
constexpr PetscReal problem_setting::uyRef (
            PetscReal const & x,
            PetscReal const & y,
            PetscReal const & z,
            PetscReal const & theta )  [constexpr]
```

Computes the reference solution for the problem in the y-direction.
This function provides an analytical reference solution for the given problem in the y-direction. It is used for validation and testing of numerical simulations.

Parameters

x	X-coordinate.
y	Y-coordinate.
z	Z-coordinate.
theta	Time-dependent parameter.

Returns

Computed reference solution in the y-direction.

Definition at line 130 of file config_problem.hpp.

5.1.2.3 uzRef()

```
constexpr PetscReal problem_setting::uzRef (
            PetscReal const & x,
            PetscReal const & y,
            PetscReal const & z,
            PetscReal const & theta )  [constexpr]
```

Computes the reference solution for the problem in the z-direction.

This function provides an analytical reference solution for the given problem in the z-direction. It is useful for testing the consistency of numerical solvers.

Parameters

x	X-coordinate.
y	Y-coordinate.
z	Z-coordinate.
theta	Time-dependent parameter.

Returns

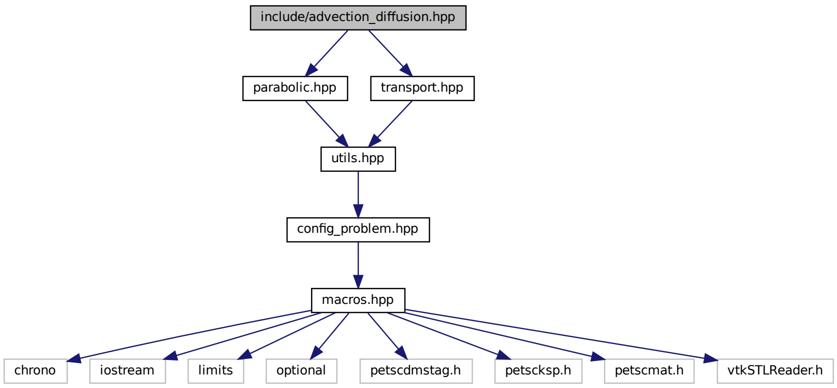
Computed reference solution in the z-direction.

Definition at line 174 of file config_problem.hpp.

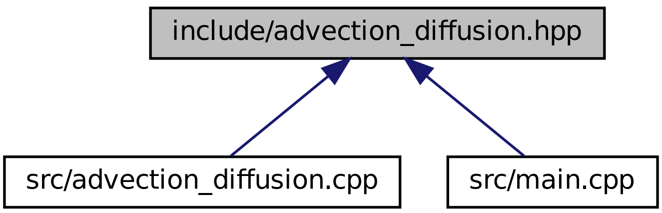
5.2 include/advection_diffusion.hpp File Reference

```
#include "parabolic.hpp"
#include "transport.hpp"
```

Include dependency graph for advection_diffusion.hpp:



This graph shows which files directly or indirectly include this file:



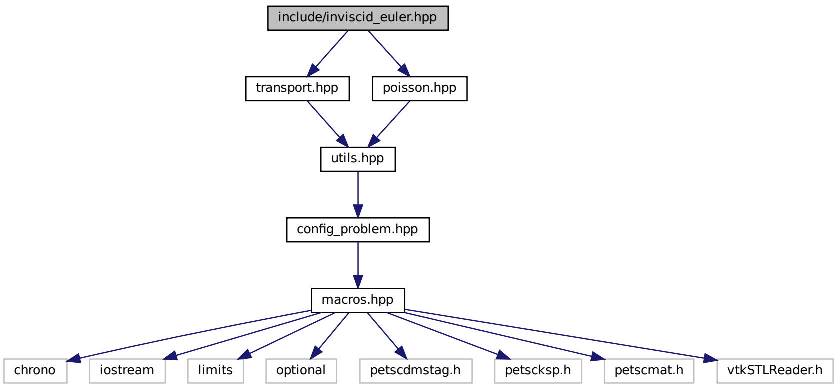
Classes

- class advection_diffusion_problem

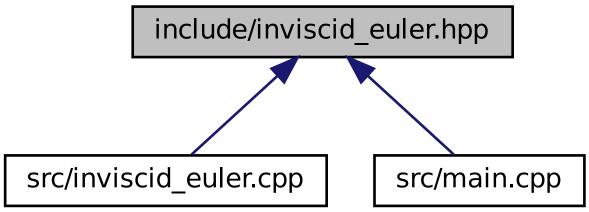
5.3 include/inviscid_euler.hpp File Reference

```
#include "transport.hpp"
#include "poisson.hpp"
```

Include dependency graph for inviscid_euler.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class euler_problem

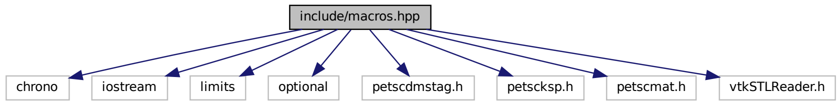
  Solves the Euler evolutionary incompressible equations, applying a first order Chorin-Temam algorithm. It consists of a transport problem for each velocity component and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation DOES NOT ALLOW for Brinkamn penalty method.

5.4 include/macros.hpp File Reference

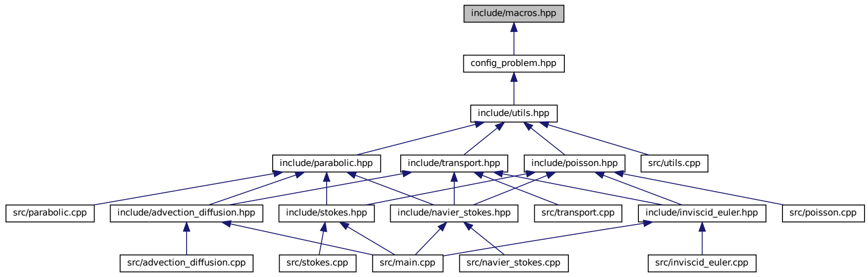
```
#include <chrono>
#include <iostream>
#include <limits>
#include <optional>
#include <petscdmstag.h>
#include <petscksp.h>
#include <petscmat.h>
```

`#include <vtkSTLReader.h>`

Include dependency graph for macros.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define **BACK_DOWN** DMSTAG_BACK_DOWN
- #define **BACK_LEFT** DMSTAG_BACK_LEFT
- #define **BACK** DMSTAG_BACK
- #define **BACK_RIGHT** DMSTAG_BACK_RIGHT
- #define **BACK_UP** DMSTAG_BACK_UP
- #define **DOWN_LEFT** DMSTAG_DOWN_LEFT
- #define **DOWN** DMSTAG_DOWN
- #define **DOWN_RIGHT** DMSTAG_DOWN_RIGHT
- #define **LEFT** DMSTAG_LEFT
- #define **ELEMENT** DMSTAG_ELEMENT
- #define **RIGHT** DMSTAG_RIGHT
- #define **UP_LEFT** DMSTAG_UP_LEFT
- #define **UP** DMSTAG_UP
- #define **UP_RIGHT** DMSTAG_UP_RIGHT
- #define **FRONT_DOWN** DMSTAG_FRONT_DOWN
- #define **FRONT_LEFT** DMSTAG_FRONT_LEFT
- #define **FRONT** DMSTAG_FRONT
- #define **FRONT_RIGHT** DMSTAG_FRONT_RIGHT
- #define **FRONT_UP** DMSTAG_FRONT_UP

Variables

- constexpr PetscReal pi = 3.14159265358979323846

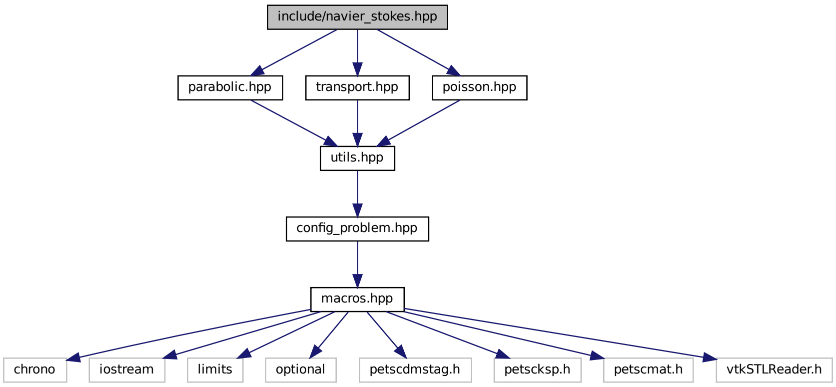
  The mathematical constant pi.
- constexpr PetscReal eps =1e6

  A large numerical value used as an approximation for Brinkman flow penalization parameter.

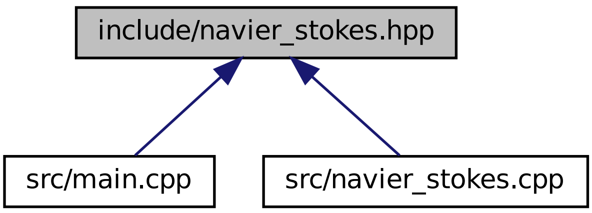
5.5 include/navier_stokes.hpp File Reference

```
#include "parabolic.hpp"
#include "transport.hpp"
#include "poisson.hpp"
```

Include dependency graph for navier_stokes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

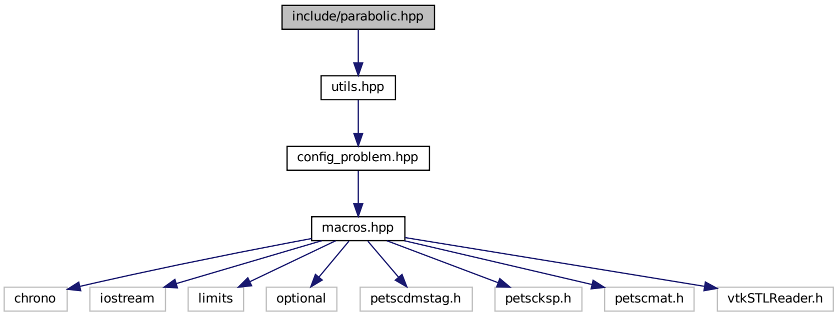
- class navier_stokes_problem

  Solves the Navier-Stokes evolutionary incompressible Navier-Stokes equations, applying a first order Chorin-Temam algorithm. It consists of a transport problem for each velocity component, a parabolic problem for each velocity component, and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.

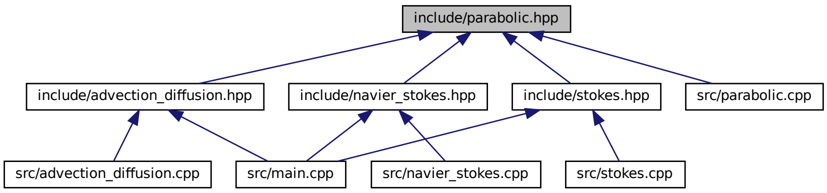
5.6 include/parabolic.hpp File Reference

```
#include "utils.hpp"
```

Include dependency graph for parabolic.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class parabolic_problem_x

  Represents a parabolic problem in the x-direction.
- class parabolic_problem_y

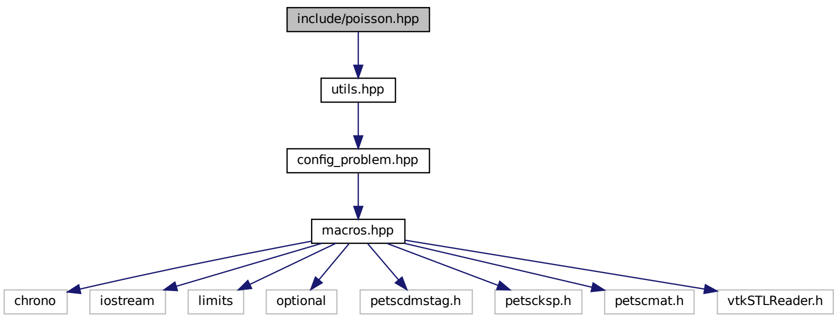
  Represents a parabolic problem in the y-direction.
- class parabolic_problem_z

  Represents a parabolic problem in the z-direction.

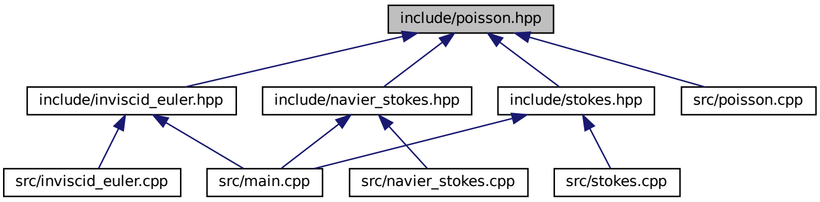
5.7 include/poisson.hpp File Reference

`#include "utils.hpp"`

Include dependency graph for poisson.hpp:



This graph shows which files directly or indirectly include this file:



Classes

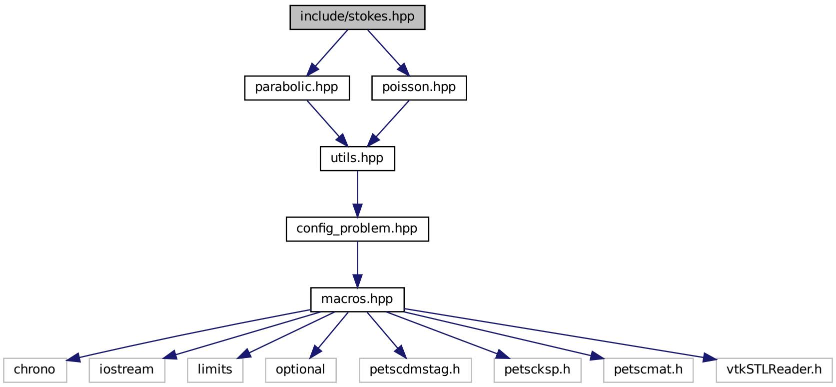
- class poisson_problem

  Represents a Poisson equation solver for pressure correction in fluid simulations.

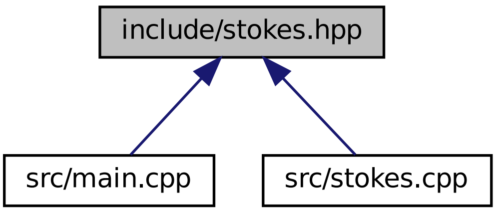
5.8 include/stokes.hpp File Reference

```
#include "parabolic.hpp"
#include "poisson.hpp"
```

Include dependency graph for stokes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

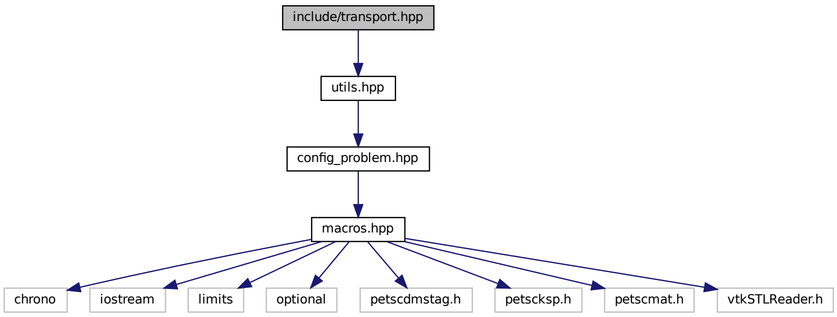
- class stokes_problem

  Solves the Stokes evolutionary incompressible Stokes equations, applying a first order Chorin-Temam algorithm. It consists of a parabolic problem for each velocity component, and a Poisson problem for the pressure. Current implementation allows Dirichlet bc's only. Boundary conditions are accessed from reference solution, which is a function of space and time. Current implementation allows for Brinkamn penalty method.

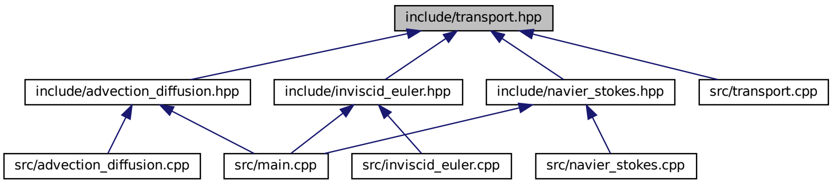
5.9 include/transport.hpp File Reference

```
#include "utils.hpp"
```

Include dependency graph for transport.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class transport_problem_x

  Represents a transport problem in the x-direction.
- class transport_problem_y

  Represents a transport problem in the y-direction.
- class transport_problem_z

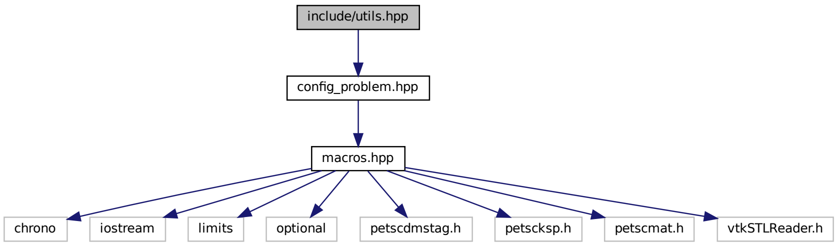
  Represents a transport problem in the z-direction.

5.10 include/utils.hpp File Reference

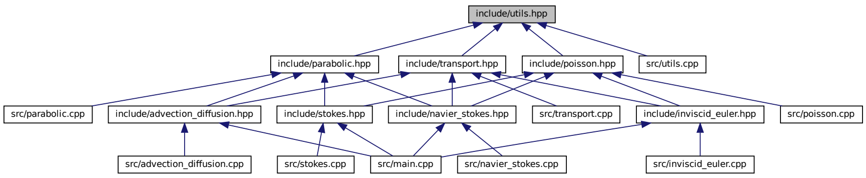
Utility functions for grid creation, analytical solutions, and geometric operations.

```
#include "config_problem.hpp"
```

Include dependency graph for utils.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- PetscErrorCode CheckSolution (Vec const &sol, Vec const &solRef, std::string const &comp)

  Checks the difference between a computed solution and a reference solution and evaulates L2-norm.
- PetscErrorCode CreateAnalyticalU (DM const &dmGrid, Vec &vec, PetscReal const &theta)

  Creates an analytical solution on a staggered grid in the x-direction (dofs in position LEFT and RIGHT)
- PetscErrorCode CreateAnalyticalV (DM const &dmGrid, Vec &vec, PetscReal const &theta)

  Creates an analytical solution on a staggered grid in the y-direction (dofs in position DOWN and UP)
- PetscErrorCode CreateAnalyticalW (DM const &dmGrid, Vec &vec, PetscReal const &theta)

  Creates an analytical solution on a staggered grid in the z-direction (dofs in position BACK and FRONT)
- PetscErrorCode CreateGrid (DM *const dmGrid, PetscInt const &dof1, PetscInt const &dof2, PetscInt const &dof3)

  Creates a computational grid for simulation. Allows to specify dofs ONLY on edges, faces and cell-cenetrs. Pass 0 to not allow dofs in some position, 1 allocate dofs.
- PetscErrorCode **CreateAnalyticalP** (DM const &dmGrid, Vec &vec, PetscReal const &theta)
- void PrintSimulationParameters ()

  Prints the simulation parameters to the console.
- bool rayIntersectsTriangle (const std::array< double, 3 > &rayOrigin, const std::array< double, 3 > &rayVector, const std::array< double, 3 > &v0, const std::array< double, 3 > &v1, const std::array< double, 3 > &v2)

  Checks if a ray intersects with a triangle in 3D space (part of ray-casting algorithm to create Brinkman masks)
- bool isPointInsideMesh (const std::array< double, 3 > &point, const std::vector< std::array< double, 3 >> &vertices, const std::vector< std::array< int, 3 >> &faces)

  Determines if a point is inside a 3D mesh.
- void reader (const std::string &filename, std::vector< std::array< double, 3 >> &vertices, std::vector< std::array< int, 3 >> &faces)

  Reads a mesh file and extracts vertices and faces.
- PetscErrorCode createMaskU (DM const &dmGrid, Vec &vec_stag, std::vector< std::array< double, 3 >> const &vertices, std::vector< std::array< int, 3 >> const &faces)

Creates a mask for the x-component component.

- PetscErrorCode createMaskV (DM const &dmGrid, Vec &vec_stag, std::vector< std::array< double, 3 >> const &vertices, std::vector< std::array< int, 3 >> const &faces)

  Creates a mask for the y-component component.
- PetscErrorCode createMaskW (DM const &dmGrid, Vec &vec_stag, std::vector< std::array< double, 3 >> const &vertices, std::vector< std::array< int, 3 >> const &faces)

  Creates a mask for the z-component component.

Variables

- constexpr PetscReal D_x {Lx - Lx_0}

  Domain length in the x-direction.
- constexpr PetscReal D_y {Ly - Ly_0}

  Domain length in the y-direction.
- constexpr PetscReal D_z {Lz - Lz_0}

  Domain length in the z-direction.
- std::vector< std::array< double, 3 > > vertices

  List of mesh vertices.
- std::vector< std::array< int, 3 > > faces

  List of mesh faces.
- std::string filename

  Filename of the geometry file.

5.10.1 Detailed Description

Utility functions for grid creation, analytical solutions, and geometric operations.
This file contains various utility functions used in numerical simulations, including:

- Functions for generating analytical velocity solutions.
- Grid creation and setup.
- Mesh operations such as point-in-mesh checks and ray-triangle intersections.
- Functions for creating Brinkman masks.

5.10.2 Function Documentation

5.10.2.1 CheckSolution()

```
PetscErrorCode CheckSolution (
            Vec const & sol,
            Vec const & solRef,
            std::string const & comp )
```

Checks the difference between a computed solution and a reference solution and evaulates L2-norm.

Parameters

sol	Computed solution vector.
solRef	Reference solution vector.

Definition at line 4 of file utils.cpp.

5.10.2.2 CreateAnalyticalU()

```
PetscErrorCode CreateAnalyticalU (
            DM const & dmGrid,
            Vec & vec,
            PetscReal const & theta )
```

Creates an analytical solution on a staggered grid in the x-direction (dofs in position LEFT and RIGHT)

Parameters

dmGrid	Discretized grid.
vec	Output velocity field.

Definition at line 25 of file utils.cpp.

5.10.2.3 CreateAnalyticalV()

```
PetscErrorCode CreateAnalyticalV (
            DM const & dmGrid,
            Vec & vec,
            PetscReal const & theta )
```

Creates an analytical solution on a staggered grid in the y-direction (dofs in position DOWN and UP)

Parameters

dmGrid	Discretized grid.
vec	Output velocity field.

Definition at line 66 of file utils.cpp.

5.10.2.4 CreateAnalyticalW()

```
PetscErrorCode CreateAnalyticalW (
            DM const & dmGrid,
            Vec & vec,
            PetscReal const & theta )
```

Creates an analytical solution on a staggered grid in the z-direction (dofs in position BACK and FRONT)

Parameters

dmGrid	Discretized grid.
vec	Output velocity field.

Definition at line 107 of file utils.cpp.

5.10.2.5 CreateGrid()

```
PetscErrorCode CreateGrid (
            DM *const dmGrid,
            PetscInt const & dof1,
            PetscInt const & dof2,
            PetscInt const & dof3 )
```

Creates a computational grid for simulation. Allows to specify dofs ONLY on edges, faces and cell-cenetrs. Pass 0 to not allow dofs in some position, 1 allocate dofs.

Parameters

dmGrid	Pointer to the grid.
dof1	Degrees of freedom on edges.
dof2	Degrees of freedom of faces.
dof3	Degrees of freedom at cell-ceneters.

Definition at line 202 of file utils.cpp.

5.10.2.6 createMaskU()

```
PetscErrorCode createMaskU (
            DM const & dmGrid,
            Vec & vec_stag,
            std::vector< std::array< double, 3 >> const & vertices,
            std::vector< std::array< int, 3 >> const & faces )
```

Creates a mask for the x-component component.

Parameters

dmGrid	Discretized grid.
vec_stag	Output mask vector.
vertices	List of mesh vertices.
faces	List of mesh faces.

Definition at line 417 of file utils.cpp.

5.10.2.7 createMaskV()

```
PetscErrorCode createMaskV (
            DM const & dmGrid,
            Vec & vec_stag,
            std::vector< std::array< double, 3 >> const & vertices,
            std::vector< std::array< int, 3 >> const & faces )
```

Creates a mask for the y-component component.

Parameters

dmGrid	Discretized grid.
vec_stag	Output mask vector.
vertices	List of mesh vertices.
faces	List of mesh faces.

Definition at line 484 of file utils.cpp.

5.10.2.8 createMaskW()

```
PetscErrorCode createMaskW (
            DM const & dmGrid,
            Vec & vec_stag,
            std::vector< std::array< double, 3 >> const & vertices,
            std::vector< std::array< int, 3 >> const & faces )
```

Creates a mask for the z-component component.

Parameters

dmGrid	Discretized grid.
vec_stag	Output mask vector.
vertices	List of mesh vertices.
faces	List of mesh faces.

Definition at line 551 of file utils.cpp.

5.10.2.9 isPointInsideMesh()

```
bool isPointInsideMesh (
            const std::array< double, 3 > & point,
            const std::vector< std::array< double, 3 >> & vertices,
            const std::vector< std::array< int, 3 >> & faces )
```

Determines if a point is inside a 3D mesh.

Parameters

point	The point to be checked.
vertices	List of vertices of the mesh.
faces	List of faces of the mesh.

Returns

True if the point is inside the mesh, false otherwise.

Definition at line 303 of file utils.cpp.

5.10.2.10 rayIntersectsTriangle()

```
bool rayIntersectsTriangle (
            const std::array< double, 3 > & rayOrigin,
            const std::array< double, 3 > & rayVector,
            const std::array< double, 3 > & v0,
            const std::array< double, 3 > & v1,
            const std::array< double, 3 > & v2 )
```

Checks if a ray intersects with a triangle in 3D space (part of ray-casting algorithm to create Brinkman masks)

Parameters

rayOrigin	Starting point of the ray.
rayVector	Direction of the ray.
v0	First vertex of the triangle.
v1	Second vertex of the triangle.
v2	Third vertex of the triangle.

Returns

True if the ray intersects the triangle, false otherwise.

Definition at line 254 of file utils.cpp.

5.10.2.11 reader()

```
void reader (
            const std::string & filename,
            std::vector< std::array< double, 3 >> & vertices,
            std::vector< std::array< int, 3 >> & faces )
```

Reads a mesh file and extracts vertices and faces.

Parameters

filename	The name of the mesh file.
vertices	Output list of vertices.
faces	Output list of faces.

Definition at line 324 of file utils.cpp.

Index