

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

Generated by Doxygen 1.9.1

1 Finite Differences Navier-Stokes Solver	1
1.1 Requirements	1
1.2 Installation	1
2 Namespace Index	3
2.1 Namespace List	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 problem_setting Namespace Reference	9
5.1.1 Function Documentation	10
5.1.1.1 pRef()	10
5.1.1.2 uxRef()	10
5.1.1.3 uyRef()	10
5.1.1.4 uzRef()	11
5.1.2 Variable Documentation	11
5.1.2.1 a	11
5.1.2.2 base_path	11
5.1.2.3 brinkman	12
5.1.2.4 check_convergence	12
5.1.2.5 d	12
5.1.2.6 dt	12
5.1.2.7 iter	12
5.1.2.8 Lx	12
5.1.2.9 Lx_0	12
5.1.2.10 Ly	12
5.1.2.11 Ly_0	12
5.1.2.12 Lz	12
5.1.2.13 Lz_0	12
5.1.2.14 monitor_convergence	13
5.1.2.15 nx	13
5.1.2.16 ny	13
5.1.2.17 nz	13
5.1.2.18 problem_type	13
5.1.2.19 Re	13
5.1.2.20 theta	13
6 Class Documentation	15
6.1 advection Class Reference	15

6.1.1 Detailed Description	15
6.2 advection_diffusion_problem Class Reference	15
6.2.1 Constructor & Destructor Documentation	17
6.2.1.1 advection_diffusion_problem() [1/2]	17
6.2.1.2 advection_diffusion_problem() [2/2]	18
6.2.1.3 ~advection_diffusion_problem()	18
6.2.2 Member Function Documentation	18
6.2.2.1 assemble_magnitude()	18
6.2.2.2 compute_magnitude()	18
6.2.2.3 exodus()	18
6.2.2.4 solve()	18
6.2.2.5 update_velocity()	19
6.2.3 Member Data Documentation	19
6.2.3.1 dmGrid_cent_rich	19
6.2.3.2 dmGrid_centered	19
6.2.3.3 dmGrid_shift_transp	19
6.2.3.4 dmGrid_stag_transp	19
6.2.3.5 dmGrid_staggered_x	19
6.2.3.6 dmGrid_staggered_y	19
6.2.3.7 dmGrid_staggered_z	19
6.2.3.8 Magnitude	20
6.2.3.9 mask_U	20
6.2.3.10 mask_V	20
6.2.3.11 mask_W	20
6.2.3.12 U_up	20
6.2.3.13 V_up	20
6.2.3.14 W_up	20
6.3 euler_problem Class Reference	20
6.3.1 Detailed Description	22
6.3.2 Constructor & Destructor Documentation	23
6.3.2.1 euler_problem() [1/2]	23
6.3.2.2 euler_problem() [2/2]	23
6.3.2.3 ~euler_problem()	23
6.3.3 Member Function Documentation	23
6.3.3.1 assemble_magnitude()	23
6.3.3.2 compute_magnitude()	23
6.3.3.3 exodus()	24
6.3.3.4 solve()	24
6.3.3.5 update_velocity()	24
6.3.4 Member Data Documentation	24
6.3.4.1 dmGrid_cent_rich	24
6.3.4.2 dmGrid_centered	24

6.3.4.3 dmGrid_shift_transp	24
6.3.4.4 dmGrid_stag_transp	24
6.3.4.5 dmGrid_staggered_x	24
6.3.4.6 dmGrid_staggered_y	25
6.3.4.7 dmGrid_staggered_z	25
6.3.4.8 Magnitude	25
6.3.4.9 mask_U	25
6.3.4.10 mask_V	25
6.3.4.11 mask_W	25
6.3.4.12 P	25
6.3.4.13 P_x	25
6.3.4.14 P_y	25
6.3.4.15 P_z	25
6.3.4.16 U_up	26
6.3.4.17 V_up	26
6.3.4.18 W_up	26
6.4 navier_stokes_problem Class Reference	26
6.4.1 Detailed Description	28
6.4.2 Constructor & Destructor Documentation	29
6.4.2.1 navier_stokes_problem() [1/2]	29
6.4.2.2 navier_stokes_problem() [2/2]	29
6.4.2.3 ~navier_stokes_problem()	29
6.4.3 Member Function Documentation	29
6.4.3.1 assemble_magnitude()	29
6.4.3.2 compute_magnitude()	29
6.4.3.3 exodus()	30
6.4.3.4 solve()	30
6.4.3.5 update_velocity()	30
6.4.4 Member Data Documentation	30
6.4.4.1 dmGrid_cent_rich	30
6.4.4.2 dmGrid_centered	30
6.4.4.3 dmGrid_shift_transp	30
6.4.4.4 dmGrid_stag_transp	30
6.4.4.5 dmGrid_staggered_x	30
6.4.4.6 dmGrid_staggered_y	31
6.4.4.7 dmGrid_staggered_z	31
6.4.4.8 Magnitude	31
6.4.4.9 mask_U	31
6.4.4.10 mask_V	31
6.4.4.11 mask_W	31
6.4.4.12 P	31
6.4.4.13 P_x	31

6.4.4.14 P_y	31
6.4.4.15 P_z	31
6.4.4.16 U_up	32
6.4.4.17 V_up	32
6.4.4.18 W_up	32
6.5 parabolic_problem_x Class Reference	32
6.5.1 Detailed Description	33
6.5.2 Constructor & Destructor Documentation	33
6.5.2.1 parabolic_problem_x() [1/2]	34
6.5.2.2 parabolic_problem_x() [2/2]	34
6.5.2.3 ~parabolic_problem_x()	34
6.5.3 Member Function Documentation	34
6.5.3.1 assemble_lhs()	34
6.5.3.2 assemble_rhs()	34
6.5.3.3 exodus()	34
6.5.3.4 get_U()	34
6.5.3.5 set_U()	34
6.5.3.6 solve()	35
6.5.3.7 solve_step()	35
6.5.4 Member Data Documentation	35
6.5.4.1 A	35
6.5.4.2 dmGrid	35
6.5.4.3 mask_U	35
6.5.4.4 rhs	35
6.5.4.5 U_up	35
6.6 parabolic_problem_y Class Reference	35
6.6.1 Detailed Description	37
6.6.2 Constructor & Destructor Documentation	37
6.6.2.1 parabolic_problem_y() [1/2]	37
6.6.2.2 parabolic_problem_y() [2/2]	37
6.6.2.3 ~parabolic_problem_y()	37
6.6.3 Member Function Documentation	38
6.6.3.1 assemble_lhs()	38
6.6.3.2 assemble_rhs()	38
6.6.3.3 exodus()	38
6.6.3.4 get_V()	38
6.6.3.5 set_V()	38
6.6.3.6 solve()	38
6.6.3.7 solve_step()	38
6.6.4 Member Data Documentation	38
6.6.4.1 A	38
6.6.4.2 dmGrid	39

6.6.4.3 mask_V	39
6.6.4.4 rhs	39
6.6.4.5 V_up	39
6.7 parabolic_problem_z Class Reference	39
6.7.1 Detailed Description	41
6.7.2 Constructor & Destructor Documentation	41
6.7.2.1 parabolic_problem_z() [1/2]	41
6.7.2.2 parabolic_problem_z() [2/2]	41
6.7.2.3 ~parabolic_problem_z()	41
6.7.3 Member Function Documentation	42
6.7.3.1 assemble_lhs()	42
6.7.3.2 assemble_rhs()	42
6.7.3.3 exodus()	42
6.7.3.4 get_W()	42
6.7.3.5 set_W()	42
6.7.3.6 solve()	42
6.7.3.7 solve_step()	42
6.7.4 Member Data Documentation	42
6.7.4.1 A	42
6.7.4.2 dmGrid	43
6.7.4.3 mask_W	43
6.7.4.4 rhs	43
6.7.4.5 W_up	43
6.8 poisson_problem Class Reference	43
6.8.1 Detailed Description	46
6.8.2 Constructor & Destructor Documentation	46
6.8.2.1 poisson_problem() [1/2]	46
6.8.2.2 poisson_problem() [2/2]	46
6.8.2.3 ~poisson_problem()	46
6.8.3 Member Function Documentation	46
6.8.3.1 assemble_divergence()	46
6.8.3.2 assemble_lhs()	47
6.8.3.3 compute_divergence()	47
6.8.3.4 derive_x_P()	47
6.8.3.5 derive_y_P()	47
6.8.3.6 derive_z_P()	48
6.8.3.7 exodus()	48
6.8.3.8 manage_pressure()	48
6.8.3.9 manage_pressure_x()	48
6.8.3.10 manage_pressure_y()	49
6.8.3.11 manage_pressure_z()	49
6.8.4 Member Data Documentation	49

6.8.4.1 A	49
6.8.4.2 dmGrid_cent_rich	49
6.8.4.3 dmGrid_centered	49
6.8.4.4 dmGrid_staggered_x	49
6.8.4.5 dmGrid_staggered_y	50
6.8.4.6 dmGrid_staggered_z	50
6.8.4.7 P	50
6.8.4.8 P_x	50
6.8.4.9 P_y	50
6.8.4.10 P_z	50
6.8.4.11 U_up	50
6.8.4.12 V_up	50
6.8.4.13 W_up	50
6.9 stokes_problem Class Reference	51
6.9.1 Detailed Description	52
6.9.2 Constructor & Destructor Documentation	53
6.9.2.1 stokes_problem() [1/2]	53
6.9.2.2 stokes_problem() [2/2]	53
6.9.2.3 ~stokes_problem()	53
6.9.3 Member Function Documentation	53
6.9.3.1 assemble_magnitude()	53
6.9.3.2 compute_magnitude()	53
6.9.3.3 exodus()	54
6.9.3.4 solve()	54
6.9.3.5 update_velocity()	54
6.9.4 Member Data Documentation	54
6.9.4.1 dmGrid_cent_rich	54
6.9.4.2 dmGrid_centered	54
6.9.4.3 dmGrid_staggered_x	54
6.9.4.4 dmGrid_staggered_y	54
6.9.4.5 dmGrid_staggered_z	54
6.9.4.6 Magnitude	54
6.9.4.7 mask_U	55
6.9.4.8 mask_V	55
6.9.4.9 mask_W	55
6.9.4.10 P	55
6.9.4.11 P_x	55
6.9.4.12 P_y	55
6.9.4.13 P_z	55
6.9.4.14 U_up	55
6.9.4.15 V_up	55
6.9.4.16 W_up	55

6.10 transport_problem_x Class Reference	56
6.10.1 Detailed Description	57
6.10.2 Constructor & Destructor Documentation	57
6.10.2.1 transport_problem_x() [1/2]	58
6.10.2.2 transport_problem_x() [2/2]	58
6.10.2.3 ~transport_problem_x()	58
6.10.3 Member Function Documentation	58
6.10.3.1 CenterU()	58
6.10.3.2 Derive_x()	58
6.10.3.3 FirstDerive_y()	58
6.10.3.4 FirstDerive_z()	58
6.10.3.5 FirstShiftU_y()	59
6.10.3.6 FirstShiftU_z()	59
6.10.3.7 FirstShiftV_y()	59
6.10.3.8 FirstShiftW_z()	59
6.10.3.9 solve_step_x()	59
6.10.4 Member Data Documentation	59
6.10.4.1 dmGrid_Centered	59
6.10.4.2 dmGrid_Shifted	60
6.10.4.3 dmGrid_Staggered	60
6.10.4.4 mask_U	60
6.10.4.5 mask_V	60
6.10.4.6 mask_W	60
6.10.4.7 U_n	60
6.10.4.8 V_n	60
6.10.4.9 W_n	60
6.11 transport_problem_y Class Reference	60
6.11.1 Detailed Description	62
6.11.2 Constructor & Destructor Documentation	62
6.11.2.1 transport_problem_y() [1/2]	62
6.11.2.2 transport_problem_y() [2/2]	63
6.11.2.3 ~transport_problem_y()	63
6.11.3 Member Function Documentation	63
6.11.3.1 CenterV()	63
6.11.3.2 Derive_y()	63
6.11.3.3 FirstShiftU_y()	63
6.11.3.4 FirstShiftV_y()	63
6.11.3.5 SecondDerive_x()	63
6.11.3.6 SecondDerive_z()	64
6.11.3.7 SecondShiftV_z()	64
6.11.3.8 SecondShiftW_z()	64
6.11.3.9 solve_step_y()	64

6.11.4 Member Data Documentation	64
6.11.4.1 dmGrid_Centered	64
6.11.4.2 dmGrid_Shifted	64
6.11.4.3 dmGrid_Staggered	64
6.11.4.4 mask_U	65
6.11.4.5 mask_V	65
6.11.4.6 mask_W	65
6.11.4.7 U_n	65
6.11.4.8 V_n	65
6.11.4.9 W_n	65
6.12 transport_problem_z Class Reference	65
6.12.1 Detailed Description	67
6.12.2 Constructor & Destructor Documentation	67
6.12.2.1 transport_problem_z() [1/2]	67
6.12.2.2 transport_problem_z() [2/2]	68
6.12.2.3 ~transport_problem_z()	68
6.12.3 Member Function Documentation	68
6.12.3.1 CenterW()	68
6.12.3.2 Derive_z()	68
6.12.3.3 FirstShiftU_z()	68
6.12.3.4 FirstShiftW_z()	68
6.12.3.5 SecondShiftV_z()	68
6.12.3.6 SecondShiftW_z()	69
6.12.3.7 solve_step_z()	69
6.12.3.8 ThirdDerive_x()	69
6.12.3.9 ThirdDerive_y()	69
6.12.4 Member Data Documentation	69
6.12.4.1 dmGrid_Centered	69
6.12.4.2 dmGrid_Shifted	69
6.12.4.3 dmGrid_Staggered	69
6.12.4.4 mask_U	70
6.12.4.5 mask_V	70
6.12.4.6 mask_W	70
6.12.4.7 U_n	70
6.12.4.8 V_n	70
6.12.4.9 W_n	70
7 File Documentation	71
7.1 config_problem.hpp File Reference	71
7.1.1 Detailed Description	72
7.2 include/advection_diffusion.hpp File Reference	73
7.2.1 Macro Definition Documentation	73

7.2.1.1 ADVECTION_DIFFUSION_PROBLEM_HPP	73
7.3 include/inviscid_euler.hpp File Reference	74
7.3.1 Macro Definition Documentation	74
7.3.1.1 EULER_PROBLEM_HPP	74
7.4 include/macros.hpp File Reference	75
7.4.1 Macro Definition Documentation	76
7.4.1.1 BACK	76
7.4.1.2 BACK_DOWN	76
7.4.1.3 BACK_LEFT	76
7.4.1.4 BACK_RIGHT	76
7.4.1.5 BACK_UP	76
7.4.1.6 DOWN	76
7.4.1.7 DOWN_LEFT	76
7.4.1.8 DOWN_RIGHT	76
7.4.1.9 ELEMENT	76
7.4.1.10 FRONT	76
7.4.1.11 FRONT_DOWN	77
7.4.1.12 FRONT_LEFT	77
7.4.1.13 FRONT_RIGHT	77
7.4.1.14 FRONT_UP	77
7.4.1.15 LEFT	77
7.4.1.16 RIGHT	77
7.4.1.17 UP	77
7.4.1.18 UP_LEFT	77
7.4.1.19 UP_RIGHT	77
7.4.2 Variable Documentation	77
7.4.2.1 eps	77
7.4.2.2 pi	77
7.5 include/navier_stokes.hpp File Reference	78
7.6 include/parabolic.hpp File Reference	78
7.6.1 Macro Definition Documentation	79
7.6.1.1 PARABOLIC_PROBLEM_Y_HPP	79
7.6.1.2 PARABOLIC_PROBLEM_Z_HPP	79
7.7 include/poisson.hpp File Reference	80
7.8 include/stokes.hpp File Reference	80
7.9 include/transport.hpp File Reference	81
7.9.1 Macro Definition Documentation	82
7.9.1.1 TRANSPORT_PROBLEM_Y_HPP	82
7.9.1.2 TRANSPORT_PROBLEM_Z_HPP	82
7.10 include/utils.hpp File Reference	83
7.10.1 Detailed Description	84
7.10.2 Function Documentation	84

7.10.2.1 CheckSolution()	84
7.10.2.2 CreateAnalyticalP()	85
7.10.2.3 CreateAnalyticalU()	85
7.10.2.4 CreateAnalyticalV()	85
7.10.2.5 CreateAnalyticalW()	85
7.10.2.6 CreateGrid()	85
7.10.2.7 createMaskU()	86
7.10.2.8 createMaskV()	86
7.10.2.9 createMaskW()	86
7.10.2.10 isPointInsideMesh()	87
7.10.2.11 PrintSimulationParameters()	87
7.10.2.12 rayIntersectsTriangle()	87
7.10.2.13 reader()	88
7.10.3 Variable Documentation	88
7.10.3.1 D_x	88
7.10.3.2 D_y	88
7.10.3.3 D_z	88
7.10.3.4 faces	88
7.10.3.5 filename	88
7.10.3.6 vertices	88
7.11 README.md File Reference	89
7.12 src/advection_diffusion.cpp File Reference	89
7.13 src/inviscid_euler.cpp File Reference	89
7.14 src/main.cpp File Reference	89
7.14.1 Function Documentation	90
7.14.1.1 main()	90
7.15 src/navier_stokes.cpp File Reference	91
7.16 src/parabolic.cpp File Reference	91
7.17 src/poisson.cpp File Reference	92
7.18 src/stokes.cpp File Reference	92
7.19 src/transport.cpp File Reference	93
7.20 src/utils.cpp File Reference	93
7.20.1 Function Documentation	94
7.20.1.1 CheckSolution()	94
7.20.1.2 CreateAnalyticalP()	94
7.20.1.3 CreateAnalyticalU()	94
7.20.1.4 CreateAnalyticalV()	95
7.20.1.5 CreateAnalyticalW()	95
7.20.1.6 CreateGrid()	95
7.20.1.7 createMaskU()	96
7.20.1.8 createMaskV()	96
7.20.1.9 createMaskW()	96

7.20.1.10 isPointInsideMesh()	96
7.20.1.11 PrintSimulationParameters()	97
7.20.1.12 rayIntersectsTriangle()	97
7.20.1.13 reader()	97

Index	99
--------------	-----------

Chapter 1

Finite Differences Navier-Stokes Solver

This project implements a solver based on finite differences for the Navier-Stokes equations. The solver can be used both on a default cubic domain as well as on an input original mesh which exploits the brinkman penalization method.

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
-

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. **Compile the project**:
   After loading the modules, compile the project by running:
   ``bash
   make
3. **Usage**
   To run the solver, execute the following command:
   ``bash
   ./bin/main
```

For the brinkman solver the name of the .stl has to be indicated. An Example one is given, 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 the results directory

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

problem_setting	9
---	---

Chapter 3

Class Index

3.1 Class List

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

advection	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 Brinkam penalty method	15
advection_diffusion_problem		15
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 Brinkam penalty method	20
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 Brinkam penalty method	26
parabolic_problem_x	Represents a parabolic problem in the x-direction	32
parabolic_problem_y	Represents a parabolic problem in the y-direction	35
parabolic_problem_z	Represents a parabolic problem in the z-direction	39
poisson_problem	Represents a Poisson equation solver for pressure correction in fluid simulations	43
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 Brinkam penalty method	51
transport_problem_x	Represents a transport problem in the x-direction	56
transport_problem_y	Represents a transport problem in the y-direction	60
transport_problem_z	Represents a transport problem in the z-direction	65

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

config_problem.hpp	
Configuration file for setting up problem parameters	71
include/advection_diffusion.hpp	73
include/inviscid_euler.hpp	74
include/macros.hpp	75
include/navier_stokes.hpp	78
include/parabolic.hpp	78
include/poisson.hpp	80
include/stokes.hpp	80
include/transport.hpp	81
include/utils.hpp	
Utility functions for grid creation, analytical solutions, and geometric operations	83
src/advection_diffusion.cpp	89
src/inviscid_euler.cpp	89
src/main.cpp	89
src/navier_stokes.cpp	91
src/parabolic.cpp	91
src/poisson.cpp	92
src/stokes.cpp	92
src/transport.cpp	93
src/utils.cpp	93

Chapter 5

Namespace Documentation

5.1 problem_setting Namespace Reference

Functions

- constexpr PetscReal [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 [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 [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 [pRef](#) (PetscReal const &x, PetscReal const &y, PetscReal const &z, PetscReal const &theta)

Variables

- constexpr const char * [problem_type](#) = "navier_stokes"
Set the problem type. Possible values are "navier_stokes", "stokes", "euler", "advection_diffusion", "parabolic_x", "parabolic_y", "parabolic_z", "transport_x", "transport_y", "transport_z".
- constexpr const char * [base_path](#) = "results/"
Set this as base path to store results. Default is "results/".
- constexpr bool [brinkman](#) {false}
Set flag to (dis)able brinkman penalization. Required for complex geometry handling.
- constexpr bool [monitor_convergence](#) {true}
Set flag to (dis)able convergence error of KSP solver.
- constexpr bool [check_convergence](#) {true}
Set flag to (dis)able convergence error of variable. Only implemented for NAvier-Stokes and parabolic problems.
- constexpr PetscInt [nx](#) {32}
Set the number of elements in x-direction.
- constexpr PetscInt [ny](#) {32}
Set the number of elements in y-direction.
- constexpr PetscInt [nz](#) {32}
Set the number of elements in z-direction.
- constexpr PetscReal [Lx_0](#) {-0.5}
Set the domain limits in x, y and z directions.
- constexpr PetscReal [Ly_0](#) {-0.5}
- constexpr PetscReal [Lz_0](#) {-0.5}
- constexpr PetscReal [Lx](#) {0.5}

- constexpr PetscReal [Ly](#) {0.5}
- constexpr PetscReal [Lz](#) {0.5}
- constexpr PetscReal [dt](#) {0.00625/2}
Set time-step.
- constexpr PetscReal [iter](#) {32}
Set final time.
- PetscReal [theta](#)
Set starting time.
- constexpr PetscReal [Re](#) {1}
Set Reynolds number. For non advective problems, set $\mu = 1/Re$ (adimensional framework)
- constexpr PetscReal [a](#) = [pi](#) / 4
Set flow parameters. You can declare as many constexpr variables as you need.
- constexpr PetscReal [d](#) = 1.5 * [pi](#)

5.1.1 Function Documentation

5.1.1.1 pRef()

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

5.1.1.2 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

<i>x</i>	X-coordinate.
<i>y</i>	Y-coordinate.
<i>z</i>	Z-coordinate.
<i>theta</i>	Time-dependent parameter.

Returns

Computed reference solution in the x-direction.

5.1.1.3 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

<i>x</i>	X-coordinate.
<i>y</i>	Y-coordinate.
<i>z</i>	Z-coordinate.
<i>theta</i>	Time-dependent parameter.

Returns

Computed reference solution in the y-direction.

5.1.1.4 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

<i>x</i>	X-coordinate.
<i>y</i>	Y-coordinate.
<i>z</i>	Z-coordinate.
<i>theta</i>	Time-dependent parameter.

Returns

Computed reference solution in the z-direction.

5.1.2 Variable Documentation

5.1.2.1 a

```
constexpr PetscReal problem_setting::a = pi / 4 [constexpr]
```

Set flow parameters. You can declare as many constexpr variables as you need.

5.1.2.2 base_path

```
constexpr const char* problem_setting::base_path = "results/" [constexpr]
```

Set this as base path to store results. Default is "results/".

5.1.2.3 brinkman

```
constexpr bool problem_setting::brinkman {false} [constexpr]
```

Set flag to (dis)able brinkman penalization. Required for complex geometry handling.

5.1.2.4 check_convergence

```
constexpr bool problem_setting::check_convergence {true} [constexpr]
```

Set flag to (dis)able convergence error of variable. Only implemented for NAvier-Stokes and parabolic problems.

5.1.2.5 d

```
constexpr PetscReal problem_setting::d = 1.5 * pi [constexpr]
```

5.1.2.6 dt

```
constexpr PetscReal problem_setting::dt {0.00625/2} [constexpr]
```

Set time-step.

5.1.2.7 iter

```
constexpr PetscReal problem_setting::iter {32} [constexpr]
```

Set final time.

5.1.2.8 Lx

```
constexpr PetscReal problem_setting::Lx {0.5} [constexpr]
```

5.1.2.9 Lx_0

```
constexpr PetscReal problem_setting::Lx_0 {-0.5} [constexpr]
```

Set the domain limits in x, y and z directions.

5.1.2.10 Ly

```
constexpr PetscReal problem_setting::Ly {0.5} [constexpr]
```

5.1.2.11 Ly_0

```
constexpr PetscReal problem_setting::Ly_0 {-0.5} [constexpr]
```

5.1.2.12 Lz

```
constexpr PetscReal problem_setting::Lz {0.5} [constexpr]
```

5.1.2.13 Lz_0

```
constexpr PetscReal problem_setting::Lz_0 {-0.5} [constexpr]
```

5.1.2.14 monitor_convergence

```
constexpr bool problem_setting::monitor_convergence {true} [constexpr]
```

Set flag to (dis)able convergence error of KSP solver.

5.1.2.15 nx

```
constexpr PetscInt problem_setting::nx {32} [constexpr]
```

Set the number of elements in x-direction.

5.1.2.16 ny

```
constexpr PetscInt problem_setting::ny {32} [constexpr]
```

Set the number of elements in y-direction.

5.1.2.17 nz

```
constexpr PetscInt problem_setting::nz {32} [constexpr]
```

Set the number of elements in z-direction.

5.1.2.18 problem_type

```
constexpr const char* problem_setting::problem_type = "navier_stokes" [constexpr]
```

Set the problem type. Possible values are "navier_stokes", "stokes", "euler", "advection_diffusion", "parabolic_x", "parabolic_y", "parabolic_z", "transport_x", "transport_y", "transport_z".

5.1.2.19 Re

```
constexpr PetscReal problem_setting::Re {1} [constexpr]
```

Set Reynolds number. For non advective problems, set $\mu = 1/\text{Re}$ (adimensional framework)

5.1.2.20 theta

```
PetscReal problem_setting::theta [inline]
```

Set starting time.

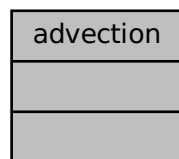
Chapter 6

Class Documentation

6.1 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:



6.1.1 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.

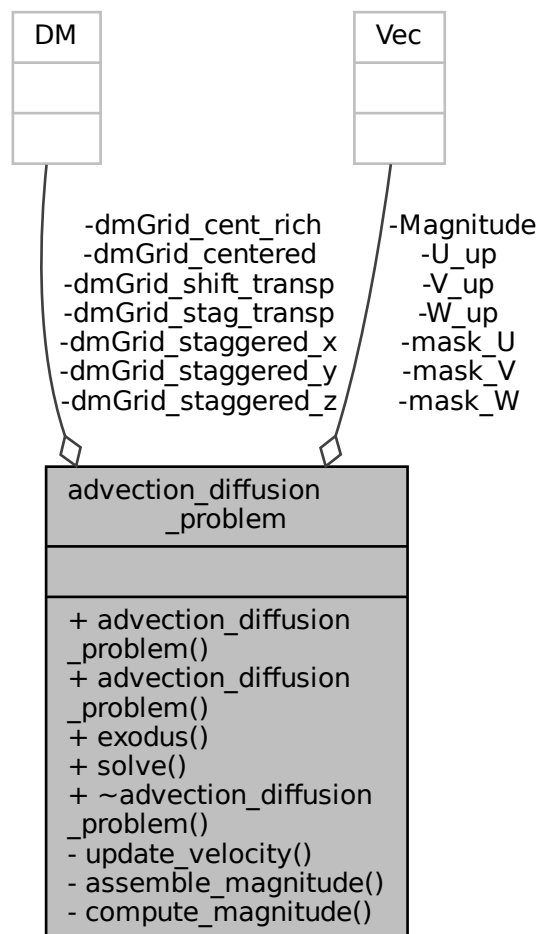
The documentation for this class was generated from the following file:

- [include/advection_diffusion.hpp](#)

6.2 advection_diffusion_problem Class Reference

```
#include <advection_diffusion.hpp>
```

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 constructor 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 for visualization of x,y,z-components, 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.

Private Member Functions

- PetscErrorCode const [update_velocity](#) (PetscReal const &theta)
Updates the velocity field, opearating final pressure correction for CT scheme.
- PetscErrorCode const [assemble_magnitude](#) (Vec &Magnitude_Shifted, Vec const &U, Vec const &V, Vec const &W)
Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.
- PetscErrorCode const [compute_magnitude](#) ()
Final assembly of magnitude vector on cell-centers.

Private Attributes

- DM [dmGrid_staggered_x](#)
Discretized grid for the staggered x-direction.
- DM [dmGrid_staggered_y](#)
Discretized grid for the staggered y-direction.
- DM [dmGrid_staggered_z](#)
Discretized grid for the staggered z-direction.
- DM [dmGrid_centered](#)
Discretized grid for magnitude.
- DM [dmGrid_cent_rich](#)
Discretized grid for shifted quantities.
- DM [dmGrid_stag_transp](#)
Staggered grid for transport equations.
- DM [dmGrid_shift_transp](#)
Shifted grid for transport computations.
- Vec [Magnitude](#)
Magnitude of velocity vectors.
- Vec [U_up](#)
- Vec [V_up](#)
- Vec [W_up](#)
Velocity fields in the x, y, and z directions.
- Vec [mask_U](#)
- Vec [mask_V](#)
- Vec [mask_W](#)
Mask vectors for boundary conditions.

6.2.1 Constructor & Destructor Documentation

6.2.1.1 advection_diffusion_problem() [1/2]

```
advection_diffusion_problem::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 ) [inline]

```

Constructor that initializes the Advection-Diffusion problem with given grids and velocity fields. This constructor 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.

6.2.1.2 advection_diffusion_problem() [2/2]

```
advection_diffusion_problem::advection_diffusion_problem ( ) [inline]
```

Default constructor that initializes stand-alone Advection-Diffusion problem with automatically created grids.

6.2.1.3 ~advection_diffusion_problem()

```
advection_diffusion_problem::~~advection_diffusion_problem ( ) [inline]
```

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

6.2.2 Member Function Documentation

6.2.2.1 assemble_magnitude()

```

PetscErrorCode const advection_diffusion_problem::assemble_magnitude (
    Vec & Magnitude_Shifted,
    Vec const & U,
    Vec const & V,
    Vec const & W ) [private]

```

Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.

Parameters

<i>Magnitude_Shifted</i>	Output shifted magnitude vector on dmGrid_cent_rich (dofs on faces and cell=centers.)
<i>U</i>	Velocity component in the x-direction.
<i>V</i>	Velocity component in the y-direction.
<i>W</i>	Velocity component in the z-direction.

6.2.2.2 compute_magnitude()

```
PetscErrorCode const advection_diffusion_problem::compute_magnitude ( ) [private]
```

Final assembly of magnitude vector on cell-centers.

6.2.2.3 exodus()

```

PetscErrorCode advection_diffusion_problem::exodus (
    size_t i )

```

Exports simulation in .vtk format for visualization of x,y,z-components, pressure and magnitude.

6.2.2.4 solve()

```
PetscErrorCode const advection_diffusion_problem::solve ( )
```


Solves the Advection-Diffusion equations leveraging [parabolic_problem_x](#), y , z , [transport_problem_x](#), y , z , and [poisson_problem](#) classes.

6.2.2.5 update_velocity()

```
PetscErrorCode const advection_diffusion_problem::update_velocity (
    PetscReal const & theta ) [private]
```

Updates the velocity field, operating final pressure correction for CT scheme.

6.2.3 Member Data Documentation

6.2.3.1 dmGrid_cent_rich

```
DM advection_diffusion_problem::dmGrid_cent_rich [private]
```

Discretized grid for shifted quantities.

6.2.3.2 dmGrid_centered

```
DM advection_diffusion_problem::dmGrid_centered [private]
```

Discretized grid for magnitude.

6.2.3.3 dmGrid_shift_transp

```
DM advection_diffusion_problem::dmGrid_shift_transp [private]
```

Shifted grid for transport computations.

6.2.3.4 dmGrid_stag_transp

```
DM advection_diffusion_problem::dmGrid_stag_transp [private]
```

Staggered grid for transport equations.

6.2.3.5 dmGrid_staggered_x

```
DM advection_diffusion_problem::dmGrid_staggered_x [private]
```

Discretized grid for the staggered x-direction.

6.2.3.6 dmGrid_staggered_y

```
DM advection_diffusion_problem::dmGrid_staggered_y [private]
```

Discretized grid for the staggered y-direction.

6.2.3.7 dmGrid_staggered_z

```
DM advection_diffusion_problem::dmGrid_staggered_z [private]
```

Discretized grid for the staggered z-direction.

6.2.3.8 Magnitude

`Vec advection_diffusion_problem::Magnitude [private]`
Magnitude of velocity vectors.

6.2.3.9 mask_U

`Vec advection_diffusion_problem::mask_U [private]`

6.2.3.10 mask_V

`Vec advection_diffusion_problem::mask_V [private]`

6.2.3.11 mask_W

`Vec advection_diffusion_problem::mask_W [private]`
Mask vectors for boundary conditions.

6.2.3.12 U_up

`Vec advection_diffusion_problem::U_up [private]`

6.2.3.13 V_up

`Vec advection_diffusion_problem::V_up [private]`

6.2.3.14 W_up

`Vec advection_diffusion_problem::W_up [private]`
Velocity fields in the x, y, and z directions.
The documentation for this class was generated from the following files:

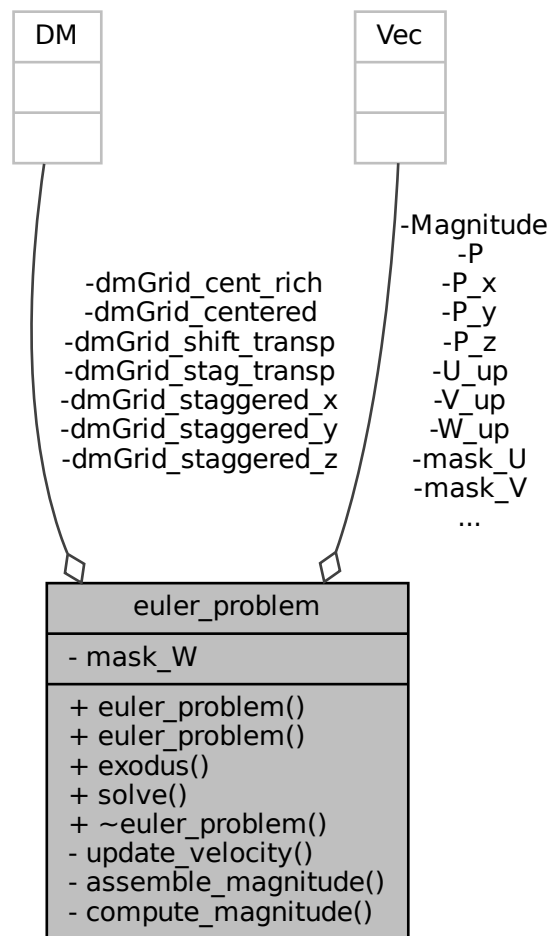
- [include/advection_diffusion.hpp](#)
- [src/advection_diffusion.cpp](#)

6.3 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 Brinkam 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 constructor 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 for visualization of x,y,z-components, 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.

Private Member Functions

- PetscErrorCode const [update_velocity](#) (PetscReal const &theta)
Updates the velocity field, opearating final pressure correction for CT scheme.
- PetscErrorCode const [assemble_magnitude](#) (Vec &Magnitude_Shifted, Vec const &U, Vec const &V, Vec const &W)
Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.
- PetscErrorCode const [compute_magnitude](#) ()
Final assembly of magnitude vector on cell-centers.

Private Attributes

- DM [dmGrid_staggered_x](#)
Discretized grid for the staggered x-direction.
- DM [dmGrid_staggered_y](#)
Discretized grid for the staggered y-direction.
- DM [dmGrid_staggered_z](#)
Discretized grid for the staggered z-direction.
- DM [dmGrid_centered](#)
Discretized grid for the centered formulation.
- DM [dmGrid_cent_rich](#)
Refined centered grid for pressure computation.
- DM [dmGrid_stag_transp](#)
Staggered grid for transport equations.
- DM [dmGrid_shift_transp](#)
Shifted grid for transport computations.
- Vec [P](#)
Pressure field.
- Vec [P_x](#)
Pressure derivative in the x-direction.
- Vec [P_y](#)
Pressure derivative in the y-direction.
- Vec [P_z](#)
Pressure derivative in the z-direction.
- Vec [Magnitude](#)
Magnitude of velocity vectors.
- Vec [U_up](#)
- Vec [V_up](#)
- Vec [W_up](#)
Velocity fields in the x, y, and z directions.
- Vec [mask_U](#)
- Vec [mask_V](#)
- Vec [mask_W](#)
Mask vectors for boundary conditions.

6.3.1 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 Brinkam penalty method.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 euler_problem() [1/2]

```
euler_problem::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 ) [inline]
```

Constructor that initializes the Euler problem with given grids and velocity fields. This constructor 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.

6.3.2.2 euler_problem() [2/2]

```
euler_problem::euler_problem ( ) [inline]
```

Default constructor that initializes stand-alone Euler problem with automatically created grids.

6.3.2.3 ~euler_problem()

```
euler_problem::~~euler_problem ( ) [inline]
```

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

6.3.3 Member Function Documentation

6.3.3.1 assemble_magnitude()

```
PetscErrorCode const euler_problem::assemble_magnitude (
    Vec & Magnitude_Shifted,
    Vec const & U,
    Vec const & V,
    Vec const & W ) [private]
```

Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.

Parameters

<i>Magnitude_Shifted</i>	Output shifted magnitude vector on dmGrid_cent_rich (dofs on faces and cell=centers.)
<i>U</i>	Velocity component in the x-direction.
<i>V</i>	Velocity component in the y-direction.
<i>W</i>	Velocity component in the z-direction.

6.3.3.2 compute_magnitude()

```
PetscErrorCode const euler_problem::compute_magnitude ( ) [private]
```

Final assembly of magnitude vector on cell-centers.

6.3.3.3 exodus()

```
PetscErrorCode euler_problem::exodus (
    size_t i )
```

Exports simulation in .vtk format format for visualization of x,y,z-componets, pressure and magnitude.

6.3.3.4 solve()

```
PetscErrorCode const euler_problem::solve ( )
```

Solves the Euler equations leveraging [parabolic_problem_x](#), y, z, [transport_problem_x](#), y, z, and [poisson_problem](#) classes.

6.3.3.5 update_velocity()

```
PetscErrorCode const euler_problem::update_velocity (
    PetscReal const & theta ) [private]
```

Updates the velocity field, opearating final pressure correction for CT scheme.

6.3.4 Member Data Documentation

6.3.4.1 dmGrid_cent_rich

```
DM euler_problem::dmGrid_cent_rich [private]
```

Refined centered grid for pressure computation.

6.3.4.2 dmGrid_centered

```
DM euler_problem::dmGrid_centered [private]
```

Discretized grid for the centered formulation.

6.3.4.3 dmGrid_shift_transp

```
DM euler_problem::dmGrid_shift_transp [private]
```

Shifted grid for transport computations.

6.3.4.4 dmGrid_stag_transp

```
DM euler_problem::dmGrid_stag_transp [private]
```

Staggered grid for transport equations.

6.3.4.5 dmGrid_staggered_x

```
DM euler_problem::dmGrid_staggered_x [private]
```

Discretized grid for the staggered x-direction.

6.3.4.6 dmGrid_staggered_y

DM euler_problem::dmGrid_staggered_y [private]
Discretized grid for the staggered y-direction.

6.3.4.7 dmGrid_staggered_z

DM euler_problem::dmGrid_staggered_z [private]
Discretized grid for the staggered z-direction.

6.3.4.8 Magnitude

Vec euler_problem::Magnitude [private]
Magnitude of velocity vectors.

6.3.4.9 mask_U

Vec euler_problem::mask_U [private]

6.3.4.10 mask_V

Vec euler_problem::mask_V [private]

6.3.4.11 mask_W

Vec euler_problem::mask_W [private]
Mask vectors for boundary conditions.

6.3.4.12 P

Vec euler_problem::P [private]
Pressure field.

6.3.4.13 P_x

Vec euler_problem::P_x [private]
Pressure derivative in the x-direction.

6.3.4.14 P_y

Vec euler_problem::P_y [private]
Pressure derivative in the y-direction.

6.3.4.15 P_z

Vec euler_problem::P_z [private]
Pressure derivative in the z-direction.

6.3.4.16 U_up

```
Vec euler_problem::U_up [private]
```

6.3.4.17 V_up

```
Vec euler_problem::V_up [private]
```

6.3.4.18 W_up

```
Vec euler_problem::W_up [private]
```

Velocity fields in the x, y, and z directions.

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

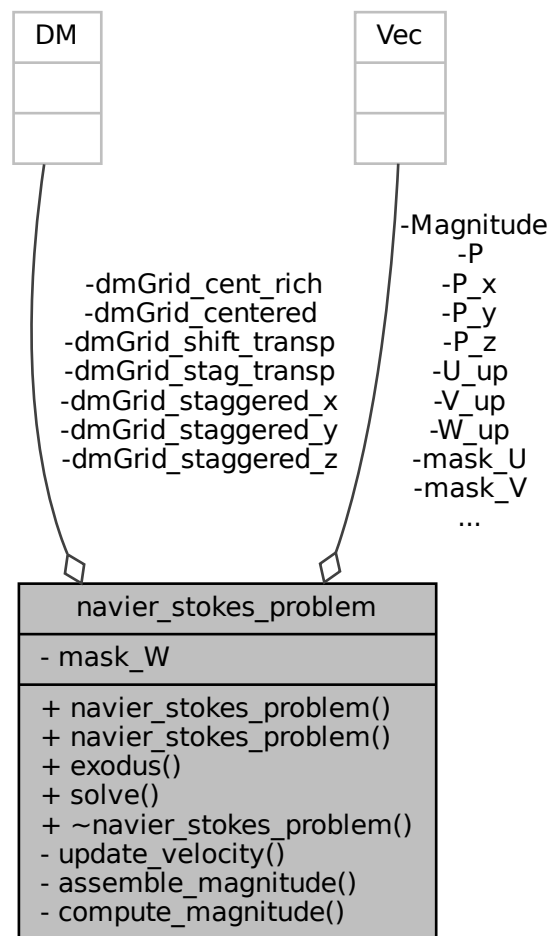
- [include/inviscid_euler.hpp](#)
- [src/inviscid_euler.cpp](#)

6.4 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 Brinkam 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 constructor 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 for visualization of x,y,z-components, 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.

Private Member Functions

- PetscErrorCode const [update_velocity](#) (PetscReal const &theta)
Updates the velocity field, opearating final pressure correction for CT scheme.
- PetscErrorCode const [assemble_magnitude](#) (Vec &Magnitude_Shifted, Vec const &U, Vec const &V, Vec const &W)
Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.
- PetscErrorCode const [compute_magnitude](#) ()
Final assembly of magnitude vector on cell-centers.

Private Attributes

- DM [dmGrid_staggered_x](#)
Discretized grid for the staggered x-direction.
- DM [dmGrid_staggered_y](#)
Discretized grid for the staggered y-direction.
- DM [dmGrid_staggered_z](#)
Discretized grid for the staggered z-direction.
- DM [dmGrid_centered](#)
Discretized grid for the centered formulation.
- DM [dmGrid_cent_rich](#)
Refined centered grid for pressure computation.
- DM [dmGrid_stag_transp](#)
Staggered grid for transport equations.
- DM [dmGrid_shift_transp](#)
Shifted grid for transport computations.
- Vec [P](#)
Pressure field.
- Vec [P_x](#)
Pressure derivative in the x-direction.
- Vec [P_y](#)
Pressure derivative in the y-direction.
- Vec [P_z](#)
Pressure derivative in the z-direction.
- Vec [Magnitude](#)
Magnitude of velocity vectors.
- Vec [U_up](#)
- Vec [V_up](#)
- Vec [W_up](#)
Velocity fields in the x, y, and z directions.
- Vec [mask_U](#)
- Vec [mask_V](#)
- Vec [mask_W](#)
Mask vectors for boundary conditions.

6.4.1 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 Brinkam penalty method.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 navier_stokes_problem() [1/2]

```
navier_stokes_problem::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 ) [inline]
```

Constructor that initializes the Navier-Stokes problem with given grids and velocity fields. This constructor 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.

6.4.2.2 navier_stokes_problem() [2/2]

```
navier_stokes_problem::navier_stokes_problem ( ) [inline]
```

Default constructor that initializes stand-alone Navier-Stokes problem with automatically created grids.

6.4.2.3 ~navier_stokes_problem()

```
navier_stokes_problem::~~navier_stokes_problem ( ) [inline]
```

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

6.4.3 Member Function Documentation

6.4.3.1 assemble_magnitude()

```
PetscErrorCode const navier_stokes_problem::assemble_magnitude (
    Vec & Magnitude_Shifted,
    Vec const & U,
    Vec const & V,
    Vec const & W ) [private]
```

Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.

Parameters

<i>Magnitude_Shifted</i>	Output shifted magnitude vector on dmGrid_cent_rich (dofs on faces and cell=centers.)
<i>U</i>	Velocity component in the x-direction.
<i>V</i>	Velocity component in the y-direction.
<i>W</i>	Velocity component in the z-direction.

6.4.3.2 compute_magnitude()

```
PetscErrorCode const navier_stokes_problem::compute_magnitude ( ) [private]
```

Final assembly of magnitude vector on cell-centers.

6.4.3.3 exodus()

```
PetscErrorCode navier_stokes_problem::exodus (
    size_t i )
```

Exports simulation in .vtk format format for visualization of x,y,z-componets, pressure and magnitude.

6.4.3.4 solve()

```
PetscErrorCode const navier_stokes_problem::solve ( )
```

Solves the Navier-Stokes equations leveraging [parabolic_problem_x](#), [y](#), [z](#), [transport_problem_x](#), [y](#), [z](#), and [poisson_problem](#) classes.

6.4.3.5 update_velocity()

```
PetscErrorCode const navier_stokes_problem::update_velocity (
    PetscReal const & theta ) [private]
```

Updates the velocity field, opearating final pressure correction for CT scheme.

6.4.4 Member Data Documentation

6.4.4.1 dmGrid_cent_rich

```
DM navier_stokes_problem::dmGrid_cent_rich [private]
```

Refined centered grid for pressure computation.

6.4.4.2 dmGrid_centered

```
DM navier_stokes_problem::dmGrid_centered [private]
```

Discretized grid for the centered formulation.

6.4.4.3 dmGrid_shift_transp

```
DM navier_stokes_problem::dmGrid_shift_transp [private]
```

Shifted grid for transport computations.

6.4.4.4 dmGrid_stag_transp

```
DM navier_stokes_problem::dmGrid_stag_transp [private]
```

Staggered grid for transport equations.

6.4.4.5 dmGrid_staggered_x

```
DM navier_stokes_problem::dmGrid_staggered_x [private]
```

Discretized grid for the staggered x-direction.

6.4.4.6 dmGrid_staggered_y

DM navier_stokes_problem::dmGrid_staggered_y [private]
Discretized grid for the staggered y-direction.

6.4.4.7 dmGrid_staggered_z

DM navier_stokes_problem::dmGrid_staggered_z [private]
Discretized grid for the staggered z-direction.

6.4.4.8 Magnitude

Vec navier_stokes_problem::Magnitude [private]
Magnitude of velocity vectors.

6.4.4.9 mask_U

Vec navier_stokes_problem::mask_U [private]

6.4.4.10 mask_V

Vec navier_stokes_problem::mask_V [private]

6.4.4.11 mask_W

Vec navier_stokes_problem::mask_W [private]
Mask vectors for boundary conditions.

6.4.4.12 P

Vec navier_stokes_problem::P [private]
Pressure field.

6.4.4.13 P_x

Vec navier_stokes_problem::P_x [private]
Pressure derivative in the x-direction.

6.4.4.14 P_y

Vec navier_stokes_problem::P_y [private]
Pressure derivative in the y-direction.

6.4.4.15 P_z

Vec navier_stokes_problem::P_z [private]
Pressure derivative in the z-direction.

6.4.4.16 U_up

```
Vec navier_stokes_problem::U_up [private]
```

6.4.4.17 V_up

```
Vec navier_stokes_problem::V_up [private]
```

6.4.4.18 W_up

```
Vec navier_stokes_problem::W_up [private]
```

Velocity fields in the x, y, and z directions.

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

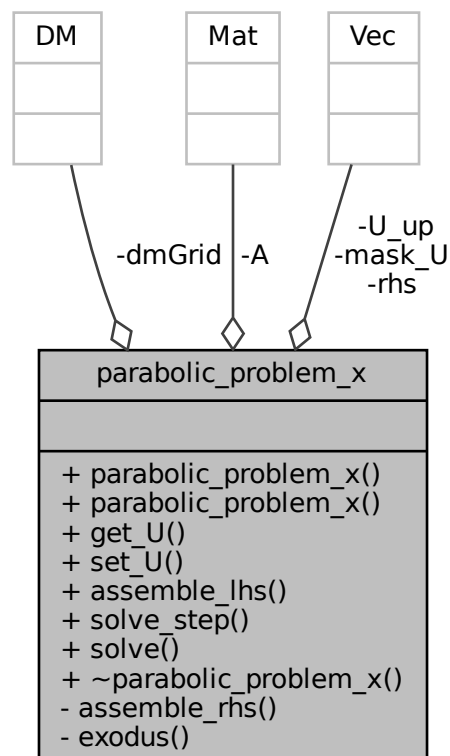
- include/[navier_stokes.hpp](#)
- src/[navier_stokes.cpp](#)

6.5 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.
- Vec [get_U](#) ()
- void [set_U](#) (Vec const &U)
- PetscErrorCode [assemble_lhs](#) ()
Assembles the left-hand side (LHS) matrix.
- 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 to avoid leaks and unexpected RAM overhead.

Private Member Functions

- const PetscErrorCode [assemble_rhs](#) (PetscReal const &theta, Vec const &U_up)
Assembles the right-hand side (RHS) vector.
- PetscErrorCode [exodus](#) (size_t const &i)
Exports results in .vtk format for post-processing.

Private Attributes

- DM [dmGrid](#)
Discretized grid for the problem.
- Mat [A](#)
Matrix representing the left-hand side of the system.
- Vec [rhs](#)
Right-hand side vector.
- Vec [U_up](#)
Solution vector for the x-direction. If not provided, it must be passed as reference in solve_step(...)
- Vec [mask_U](#)
Mask vector used for Brinkman flow.

6.5.1 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.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 parabolic_problem_x() [1/2]

```
parabolic_problem_x::parabolic_problem_x (
    DM const & dmGrid ) [inline]
```

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

<i>dmGrid</i>	staggered grid petsc-object must be already be defined.
---------------	---

6.5.2.2 parabolic_problem_x() [2/2]

```
parabolic_problem_x::parabolic_problem_x ( ) [inline]
```

Default constructor for stand-alone problem.

6.5.2.3 ~parabolic_problem_x()

```
parabolic_problem_x::~~parabolic_problem_x ( ) [inline]
```

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

6.5.3 Member Function Documentation

6.5.3.1 assemble_lhs()

```
PetscErrorCode parabolic_problem_x::assemble_lhs ( )
```

Assembles the left-hand side (LHS) matrix.

6.5.3.2 assemble_rhs()

```
const PetscErrorCode parabolic_problem_x::assemble_rhs (
    PetscReal const & theta,
    Vec const & U_up ) [private]
```

Assembles the right-hand side (RHS) vector.

6.5.3.3 exodus()

```
PetscErrorCode parabolic_problem_x::exodus (
    size_t const & i ) [private]
```

Exports results in .vtk format for post-processing.

6.5.3.4 get_U()

```
Vec parabolic_problem_x::get_U ( )
```

6.5.3.5 set_U()

```
void parabolic_problem_x::set_U (
    Vec const & U )
```


6.5.3.6 solve()

`PetscErrorCode parabolic_problem_x::solve ()`
 Solves the entire parabolic problem in the x-direction.

6.5.3.7 solve_step()

`PetscErrorCode parabolic_problem_x::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.

6.5.4 Member Data Documentation

6.5.4.1 A

`Mat parabolic_problem_x::A [private]`
 Matrix representing the left-hand side of the system.

6.5.4.2 dmGrid

`DM parabolic_problem_x::dmGrid [private]`
 Discretized grid for the problem.

6.5.4.3 mask_U

`Vec parabolic_problem_x::mask_U [private]`
 Mask vector used for Brinkman flow.

6.5.4.4 rhs

`Vec parabolic_problem_x::rhs [private]`
 Right-hand side vector.

6.5.4.5 U_up

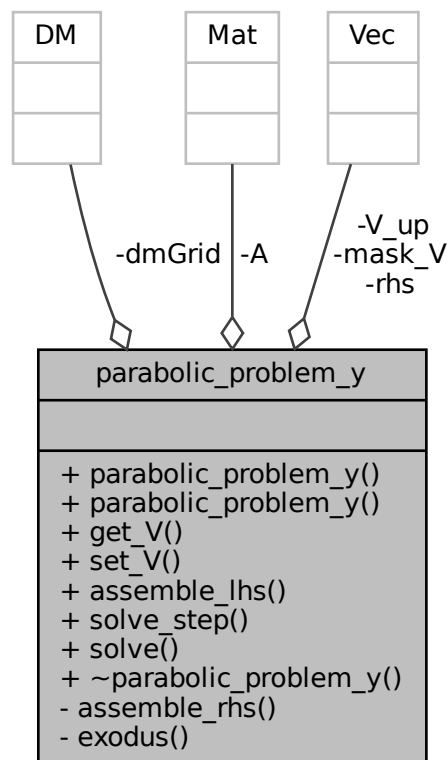
`Vec parabolic_problem_x::U_up [private]`
 Solution vector for the x-direction. If not provided, it must be passed as reference in `solve_step(...)`
 The documentation for this class was generated from the following files:

- [include/parabolic.hpp](#)
- [src/parabolic.cpp](#)

6.6 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.
- Vec [get_V](#) ()
- void [set_V](#) (Vec const &V)
- PetscErrorCode [assemble_lhs](#) ()
Assembles the left-hand side (LHS) matrix.
- 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 to avoid leaks and unexpected RAM overhead.

Private Member Functions

- const PetscErrorCode [assemble_rhs](#) (PetscReal const &theta, Vec const &[V_up](#))
Assembles the right-hand side (RHS) vector.
- PetscErrorCode [exodus](#) (size_t const &i)
Exports results in .vtk format for post-processing.

Private Attributes

- DM [dmGrid](#)
Discretized grid for the problem.
- Mat [A](#)
Matrix representing the left-hand side of the system.
- Vec [rhs](#)
Right-hand side vector.
- Vec [V_up](#)
Solution vector for the y-direction. If not provided, it must be passed as reference in solve_step(...)
- Vec [mask_V](#)
Mask vector used for Brinkman flow.

6.6.1 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.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 parabolic_problem_y() [1/2]

```
parabolic_problem_y::parabolic_problem_y (
    DM const & dmGrid ) [inline]
```

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.
------------------------	---

6.6.2.2 parabolic_problem_y() [2/2]

```
parabolic_problem_y::parabolic_problem_y ( ) [inline]
```

Default constructor for stand-alone problem.

6.6.2.3 ~parabolic_problem_y()

```
parabolic_problem_y::~~parabolic_problem_y ( ) [inline]
```

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

6.6.3 Member Function Documentation

6.6.3.1 assemble_lhs()

```
PetscErrorCode parabolic_problem_y::assemble_lhs ( )
```

Assembles the left-hand side (LHS) matrix.

6.6.3.2 assemble_rhs()

```
const PetscErrorCode parabolic_problem_y::assemble_rhs (
    PetscReal const & theta,
    Vec const & V_up ) [private]
```

Assembles the right-hand side (RHS) vector.

6.6.3.3 exodus()

```
PetscErrorCode parabolic_problem_y::exodus (
    size_t const & i ) [private]
```

Exports results in .vtk format for post-processing.

6.6.3.4 get_V()

```
Vec parabolic_problem_y::get_V ( )
```

6.6.3.5 set_V()

```
void parabolic_problem_y::set_V (
    Vec const & V )
```

6.6.3.6 solve()

```
PetscErrorCode parabolic_problem_y::solve ( )
```

Solves the entire parabolic problem in the Y-direction.

6.6.3.7 solve_step()

```
PetscErrorCode parabolic_problem_y::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.

6.6.4 Member Data Documentation

6.6.4.1 A

```
Mat parabolic_problem_y::A [private]
```

Matrix representing the left-hand side of the system.

6.6.4.2 dmGrid

DM parabolic_problem_y::dmGrid [private]
Discretized grid for the problem.

6.6.4.3 mask_V

Vec parabolic_problem_y::mask_V [private]
Mask vector used for Brinkman flow.

6.6.4.4 rhs

Vec parabolic_problem_y::rhs [private]
Right-hand side vector.

6.6.4.5 V_up

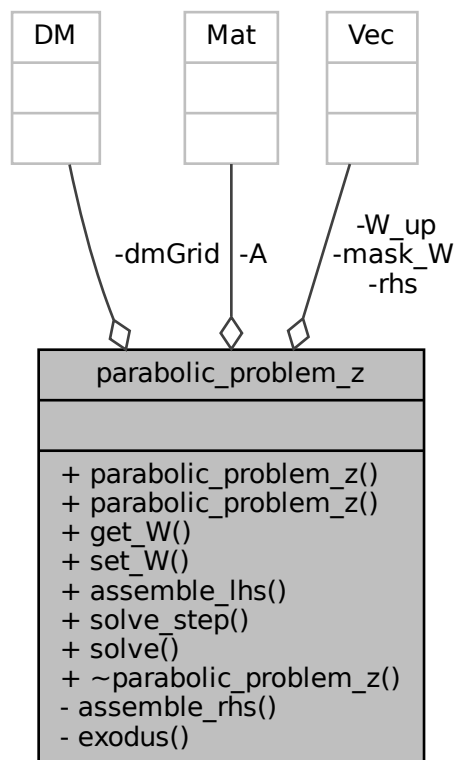
Vec parabolic_problem_y::V_up [private]
Solution vector for the y-direction. If not provided, it must be passed as reference in solve_step(...)
The documentation for this class was generated from the following files:

- include/[parabolic.hpp](#)
- src/[parabolic.cpp](#)

6.7 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.
- `Vec` `get_W` ()
- void `set_W` (`Vec` const &`W`)
- `PetscErrorCode` `assemble_lhs` ()
Assembles the left-hand side (LHS) matrix.
- `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 to avoid leaks and unexpected RAM overhead.

Private Member Functions

- PetscErrorCode const [assemble_rhs](#) (PetscReal const &theta, Vec const &[W_up](#))
Assembles the right-hand side (RHS) vector.
- PetscErrorCode [exodus](#) (size_t const &i)
Exports results in .vtk format for post-processing.

Private Attributes

- DM [dmGrid](#)
Discretized grid for the problem.
- Mat [A](#)
Matrix representing the left-hand side of the system.
- Vec [rhs](#)
Right-hand side vector.
- Vec [W_up](#)
Solution vector for the x-direction. If not provided, it must be passed as reference in solve_step(...)
- Vec [mask_W](#)
Mask vector used for Brinkman flow.

6.7.1 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.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 parabolic_problem_z() [1/2]

```
parabolic_problem_z::parabolic_problem_z (
    DM const & dmGrid ) [inline]
```

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.
------------------------	---

6.7.2.2 parabolic_problem_z() [2/2]

```
parabolic_problem_z::parabolic_problem_z ( ) [inline]
```

Default constructor for stand-alone problem.

6.7.2.3 ~parabolic_problem_z()

```
parabolic_problem_z::~~parabolic_problem_z ( ) [inline]
```

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

6.7.3 Member Function Documentation

6.7.3.1 assemble_lhs()

`PetscErrorCode parabolic_problem_z::assemble_lhs ()`
 Assembles the left-hand side (LHS) matrix.

6.7.3.2 assemble_rhs()

`PetscErrorCode const parabolic_problem_z::assemble_rhs (`
 `PetscReal const & theta,`
 `Vec const & W_up) [private]`
 Assembles the right-hand side (RHS) vector.

6.7.3.3 exodus()

`PetscErrorCode parabolic_problem_z::exodus (`
 `size_t const & i) [private]`
 Exports results in .vtk format for post-processing.

6.7.3.4 get_W()

`Vec parabolic_problem_z::get_W ()`

6.7.3.5 set_W()

`void parabolic_problem_z::set_W (`
 `Vec const & W)`

6.7.3.6 solve()

`PetscErrorCode parabolic_problem_z::solve ()`
 Solves the entire parabolic problem in the Z-direction.

6.7.3.7 solve_step()

`PetscErrorCode parabolic_problem_z::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.

6.7.4 Member Data Documentation

6.7.4.1 A

`Mat parabolic_problem_z::A [private]`
 Matrix representing the left-hand side of the system.

6.7.4.2 dmGrid

DM parabolic_problem_z::dmGrid [private]
Discretized grid for the problem.

6.7.4.3 mask_W

Vec parabolic_problem_z::mask_W [private]
Mask vector used for Brinkman flow.

6.7.4.4 rhs

Vec parabolic_problem_z::rhs [private]
Right-hand side vector.

6.7.4.5 W_up

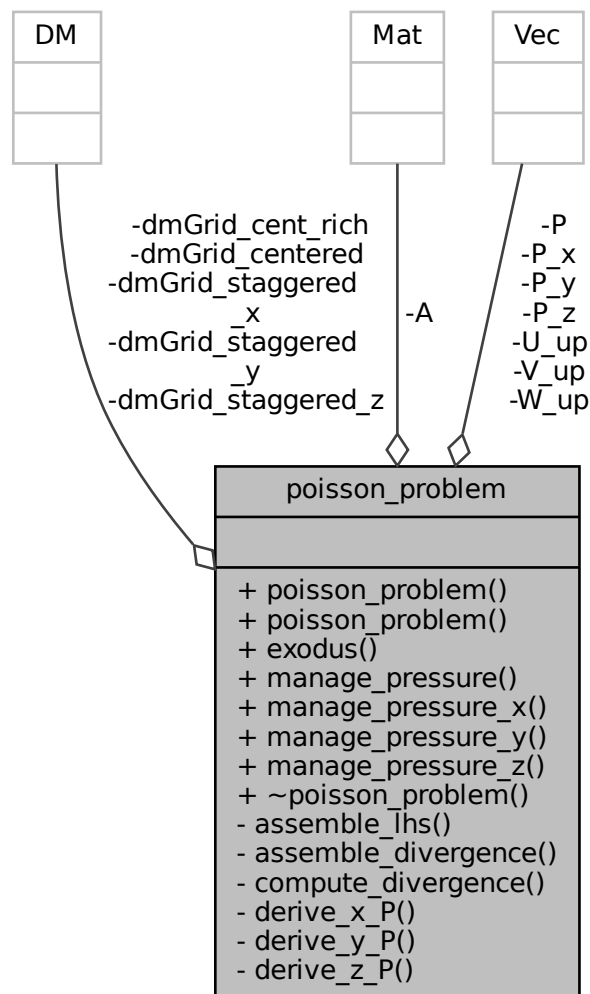
Vec parabolic_problem_z::W_up [private]
Solution vector for the x-direction. If not provided, it must be passed as reference in solve_step(...)
The documentation for this class was generated from the following files:

- [include/parabolic.hpp](#)
- [src/parabolic.cpp](#)

6.8 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` ()
- 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.

Private Member Functions

- PetscErrorCode const [assemble_lhs](#) ()
Assembles the left-hand side (LHS) matrix for the Poisson problem.
- PetscErrorCode const [assemble_divergence](#) (Vec &div, Vec const &U, Vec const &V, Vec const &W)
Computes the divergence term for velocity fields and outputs on dmGrid_centered_rich grid.
- PetscErrorCode const [compute_divergence](#) (Vec &div, Vec const &U_n, Vec const &V_n, Vec const &W_n)
Assemble routines for a previously found divergence of the given velocity field and migrates it on the correct dmGrid_centered grid.
- PetscErrorCode const [derive_x_P](#) (Vec &P_x_shifted, Vec const &vec)
Computes the derivative of the pressure field in the x-direction and outputs on dmGrid_centered_rich grid.
- PetscErrorCode const [derive_y_P](#) (Vec &P_y_shifted, Vec const &vec)
Computes the derivative of the pressure field in the y-direction and outputs on dmGrid_centered_rich grid.
- PetscErrorCode const [derive_z_P](#) (Vec &P_z_shifted, Vec const &vec)
Computes the derivative of the pressure field in the z-direction and outputs on dmGrid_centered_rich grid.

Private Attributes

- DM [dmGrid_staggered_x](#)
Discretized grid for staggered x-direction.
- DM [dmGrid_staggered_y](#)
Discretized grid for staggered y-direction.
- DM [dmGrid_staggered_z](#)
Discretized grid for staggered z-direction.
- DM [dmGrid_centered](#)
Discretized grid for centered formulation.
- DM [dmGrid_cent_rich](#)
Refined centered grid for pressure computation.
- Vec [P](#)
Pressure field.
- Vec [P_x](#)
Pressure derivative in x-direction.
- Vec [P_y](#)
Pressure derivative in y-direction.
- Vec [P_z](#)
Pressure derivative in z-direction.
- Vec [U_up](#)
Velocity component in the x-direction.
- Vec [V_up](#)
Velocity component in the y-direction.
- Vec [W_up](#)
Velocity component in the z-direction.
- Mat [A](#)
Matrix for the discretized Poisson equation.

6.8.1 Detailed Description

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

This class solves the Poisson equation for a variable 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.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 poisson_problem() [1/2]

```
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 ) [inline]
```

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

Parameters

<i>dmGrid_staggered_x</i>	Grid for staggered x-direction.
<i>dmGrid_staggered_y</i>	Grid for staggered y-direction.
<i>dmGrid_staggered_z</i>	Grid for staggered z-direction.
<i>dmGrid_centered</i>	Grid for centered formulation.
<i>dmGrid_cent_rich</i>	Grid for refined pressure computation.

6.8.2.2 poisson_problem() [2/2]

```
poisson_problem::poisson_problem ( ) [inline]
```

6.8.2.3 ~poisson_problem()

```
poisson_problem::~~poisson_problem ( ) [inline]
```

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

6.8.3 Member Function Documentation

6.8.3.1 assemble_divergence()

```
PetscErrorCode const poisson_problem::assemble_divergence (
    Vec & div,
    Vec const & U,
    Vec const & V,
    Vec const & W ) [private]
```

Computes the divergence term for velocity fields and outputs on dmGrid_centered_rich grid.

Parameters

<i>div</i>	Output divergence vector.
<i>U</i>	Velocity component in the x-direction.
<i>V</i>	Velocity component in the y-direction.
<i>W</i>	Velocity component in the z-direction.

6.8.3.2 assemble_lhs()

PetscErrorCode const poisson_problem::assemble_lhs () [private]
 Assembles the left-hand side (LHS) matrix for the Poisson problem.

6.8.3.3 compute_divergence()

```
PetscErrorCode const poisson_problem::compute_divergence (
    Vec & div,
    Vec const & U_n,
    Vec const & V_n,
    Vec const & W_n ) [private]
```

Assemble routines for a previously found divergence of the given velocity field and migrates it on the correct dmGrid_centered grid.

Parameters

<i>div</i>	Output divergence vector.
<i>U</i>	Velocity component in the x-direction.
<i>V</i>	Velocity component in the y-direction.
<i>W</i>	Velocity component in the z-direction.

6.8.3.4 derive_x_P()

```
PetscErrorCode const poisson_problem::derive_x_P (
    Vec & P_x_shifted,
    Vec const & vec ) [private]
```

Computes the derivative of the pressure field in the x-direction and outputs on dmGrid_centered_rich grid.

Parameters

<i>P_x_shifted</i>	Output shifted pressure derivative.
<i>vec</i>	Input pressure field.

6.8.3.5 derive_y_P()

```
PetscErrorCode const poisson_problem::derive_y_P (
    Vec & P_y_shifted,
    Vec const & vec ) [private]
```

Computes the derivative of the pressure field in the y-direction and outputs on dmGrid_centered_rich grid.

Parameters

<i>P_y_shifted</i>	Output shifted pressure derivative.
<i>vec</i>	Input pressure field.

6.8.3.6 derive_z_P()

```
PetscErrorCode const poisson_problem::derive_z_P (
    Vec & P_z_shifted,
    Vec const & vec ) [private]
```

Computes the derivative of the pressure field in the z-direction and outputs on dmGrid_centered_rich grid.

Parameters

<i>P_z_shifted</i>	Output shifted pressure derivative.
<i>vec</i>	Input vector.

6.8.3.7 exodus()

```
PetscErrorCode const poisson_problem::exodus (
    size_t i )
```

Exports simulation results in .vtk format for visualization.

6.8.3.8 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

<i>U_up</i>	Velocity field in the x-direction.
<i>V_up</i>	Velocity field in the y-direction.
<i>W_up</i>	Velocity field in the z-direction.
<i>P</i>	Pressure field.

6.8.3.9 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

<i>P</i>	Pressure field.
<i>P_{↔x}</i>	Updated pressure derivative in the x-direction.

6.8.3.10 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.
$P_{\leftarrow y}$	Updated pressure derivative in the y-direction.

6.8.3.11 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.
$P_{\leftarrow z}$	Updated pressure derivative in the z-direction.

6.8.4 Member Data Documentation

6.8.4.1 A

Mat poisson_problem::A [private]

Matrix for the discretized Poisson equation.

6.8.4.2 dmGrid_cent_rich

DM poisson_problem::dmGrid_cent_rich [private]

Refined centered grid for pressure computation.

6.8.4.3 dmGrid_centered

DM poisson_problem::dmGrid_centered [private]

Discretized grid for centered formulation.

6.8.4.4 dmGrid_staggered_x

DM poisson_problem::dmGrid_staggered_x [private]

Discretized grid for staggered x-direction.

6.8.4.5 dmGrid_staggered_y

DM poisson_problem::dmGrid_staggered_y [private]
Discretized grid for staggered y-direction.

6.8.4.6 dmGrid_staggered_z

DM poisson_problem::dmGrid_staggered_z [private]
Discretized grid for staggered z-direction.

6.8.4.7 P

Vec poisson_problem::P [private]
Pressure field.

6.8.4.8 P_x

Vec poisson_problem::P_x [private]
Pressure derivative in x-direction.

6.8.4.9 P_y

Vec poisson_problem::P_y [private]
Pressure derivative in y-direction.

6.8.4.10 P_z

Vec poisson_problem::P_z [private]
Pressure derivative in z-direction.

6.8.4.11 U_up

Vec poisson_problem::U_up [private]
Velocity component in the x-direction.

6.8.4.12 V_up

Vec poisson_problem::V_up [private]
Velocity component in the y-direction.

6.8.4.13 W_up

Vec poisson_problem::W_up [private]
Velocity component in the z-direction.

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

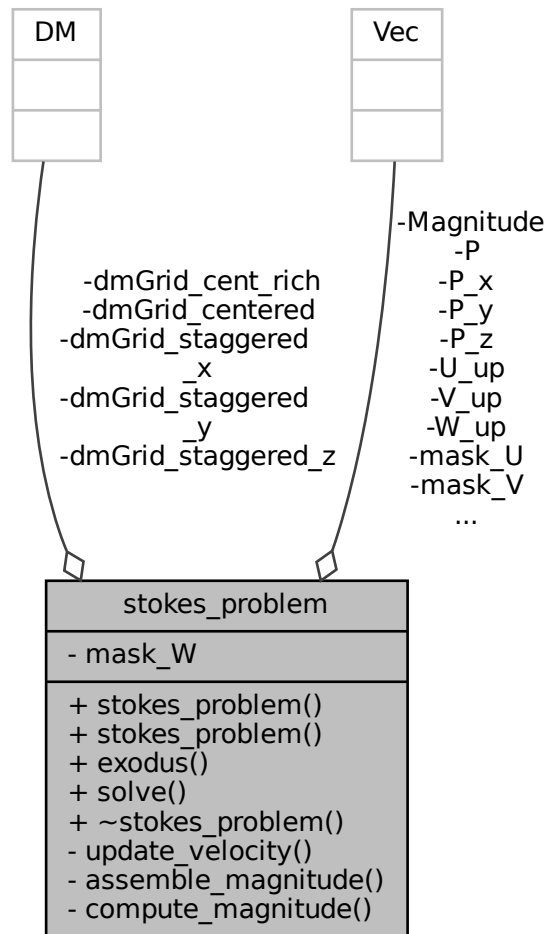
- [include/poisson.hpp](#)
- [src/poisson.cpp](#)

6.9 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 constructor 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` ()

Default constructor that initializes stand-alone Stokes problem with automatically created grids.

- PetscErrorCode `exodus` (size_t i)

Exports simulation in .vtk format for visualization of x,y,z-componets, pressure and magnitude.

- 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.

Private Member Functions

- PetscErrorCode const [update_velocity](#) (PetscReal const &theta)

Updates the velocity field, opearating final pressure correction for CT scheme.

- PetscErrorCode const [assemble_magnitude](#) (Vec &Magnitude_Shifted, Vec const &U, Vec const &V, Vec const &W)

Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.

- PetscErrorCode const [compute_magnitude](#) ()

Final assembly of magnitude vector on cell-centers.

Private Attributes

- DM [dmGrid_staggered_x](#)

Discretized grid for the staggered x-direction.

- DM [dmGrid_staggered_y](#)

Discretized grid for the staggered y-direction.

- DM [dmGrid_staggered_z](#)

Discretized grid for the staggered z-direction.

- DM [dmGrid_centered](#)

Discretized grid for the centered formulation.

- DM [dmGrid_cent_rich](#)

Refined centered grid for pressure computation.

- Vec [P](#)

Pressure field.

- Vec [P_x](#)

Pressure derivative in the x-direction.

- Vec [P_y](#)

Pressure derivative in the y-direction.

- Vec [P_z](#)

Pressure derivative in the z-direction.

- Vec [Magnitude](#)

Magnitude of velocity vectors.

- Vec [U_up](#)

- Vec [V_up](#)

- Vec [W_up](#)

Velocity fields in the x, y, and z directions.

- Vec [mask_U](#)

- Vec [mask_V](#)

- Vec [mask_W](#)

Mask vectors for boundary conditions.

6.9.1 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.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 stokes_problem() [1/2]

```
stokes_problem::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 ) [inline]
```

Constructor that initializes the Stokes problem with given grids and velocity fields. This constructor 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.

6.9.2.2 stokes_problem() [2/2]

```
stokes_problem::stokes_problem ( ) [inline]
```

Default constructor that initializes stand-alone Stokes problem with automatically created grids.

6.9.2.3 ~stokes_problem()

```
stokes_problem::~~stokes_problem ( ) [inline]
```

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

6.9.3 Member Function Documentation

6.9.3.1 assemble_magnitude()

```
PetscErrorCode const stokes_problem::assemble_magnitude (
    Vec & Magnitude_Shifted,
    Vec const & U,
    Vec const & V,
    Vec const & W ) [private]
```

Post-processing function: assembles the magnitude of velocity vectors, locating a new vector in the cell-centers.

Parameters

<i>Magnitude_Shifted</i>	Output shifted magnitude vector on dmGrid_cent_rich (dofs on faces and cell=centers.)
<i>U</i>	Velocity component in the x-direction.
<i>V</i>	Velocity component in the y-direction.
<i>W</i>	Velocity component in the z-direction.

6.9.3.2 compute_magnitude()

```
PetscErrorCode const stokes_problem::compute_magnitude ( ) [private]
```

Final assembly of magnitude vector on cell-centers.

6.9.3.3 exodus()

```
PetscErrorCode stokes_problem::exodus (
    size_t i )
```

Exports simulation in .vtk format format for visualization of x,y,z-componets, pressure and magnitude.

6.9.3.4 solve()

```
PetscErrorCode const stokes_problem::solve ( )
```

Solves the Stokes equations leveraging [parabolic_problem_x](#), y, z, [transport_problem_x](#), y, z, and [poisson_problem](#) classes.

6.9.3.5 update_velocity()

```
PetscErrorCode const stokes_problem::update_velocity (
    PetscReal const & theta ) [private]
```

Updates the velocity field, opearating final pressure correction for CT scheme.

6.9.4 Member Data Documentation

6.9.4.1 dmGrid_cent_rich

```
DM stokes_problem::dmGrid_cent_rich [private]
```

Refined centered grid for pressure computation.

6.9.4.2 dmGrid_centered

```
DM stokes_problem::dmGrid_centered [private]
```

Discretized grid for the centered formulation.

6.9.4.3 dmGrid_staggered_x

```
DM stokes_problem::dmGrid_staggered_x [private]
```

Discretized grid for the staggered x-direction.

6.9.4.4 dmGrid_staggered_y

```
DM stokes_problem::dmGrid_staggered_y [private]
```

Discretized grid for the staggered y-direction.

6.9.4.5 dmGrid_staggered_z

```
DM stokes_problem::dmGrid_staggered_z [private]
```

Discretized grid for the staggered z-direction.

6.9.4.6 Magnitude

```
Vec stokes_problem::Magnitude [private]
```

Magnitude of velocity vectors.

6.9.4.7 mask_U

`Vec stokes_problem::mask_U [private]`

6.9.4.8 mask_V

`Vec stokes_problem::mask_V [private]`

6.9.4.9 mask_W

`Vec stokes_problem::mask_W [private]`

Mask vectors for boundary conditions.

6.9.4.10 P

`Vec stokes_problem::P [private]`

Pressure field.

6.9.4.11 P_x

`Vec stokes_problem::P_x [private]`

Pressure derivative in the x-direction.

6.9.4.12 P_y

`Vec stokes_problem::P_y [private]`

Pressure derivative in the y-direction.

6.9.4.13 P_z

`Vec stokes_problem::P_z [private]`

Pressure derivative in the z-direction.

6.9.4.14 U_up

`Vec stokes_problem::U_up [private]`

6.9.4.15 V_up

`Vec stokes_problem::V_up [private]`

6.9.4.16 W_up

`Vec stokes_problem::W_up [private]`

Velocity fields in the x, y, and z directions.

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

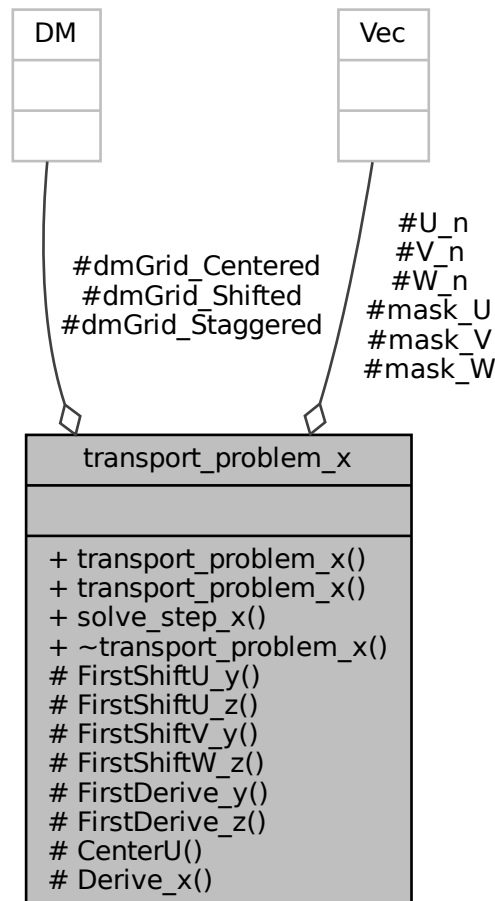
- [include/stokes.hpp](#)
- [src/stokes.cpp](#)

6.10 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.

Protected Member Functions

- PetscErrorCode [FirstShiftU_y](#) (Vec &UShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for x-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.
- PetscErrorCode [FirstShiftU_z](#) (Vec &UShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for x-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [FirstShiftV_y](#) (Vec &VShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for y-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.
- PetscErrorCode [FirstShiftW_z](#) (Vec &WShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [FirstDerive_y](#) (Vec &AB_y, Vec const &AB)
Computes the first derivative in the y-direction, placing the result on faces.
- PetscErrorCode [FirstDerive_z](#) (Vec &AB_z, Vec const &AB)
Computes the first derivative in the z-direction, placing the result on faces.
- PetscErrorCode [CenterU](#) (Vec &UCenter, Vec const &vec, PetscReal const &theta)
Centers the x-velocity component in cell-centers.
- PetscErrorCode [Derive_x](#) (Vec &U2_x, Vec const &vec, PetscReal const &theta)
Computes the first derivative in the x-direction, placing the result on faces.

Protected Attributes

- DM [dmGrid_Shifted](#)
DMGrid with dofs on faces and edges.
- DM [dmGrid_Staggered](#)
DMGrid with dofs of faces.
- DM [dmGrid_Centered](#)
DMGrid with dofs of faces and centers.
- Vec [U_n](#)
- Vec [V_n](#)
- Vec [W_n](#)
Velocity fields at the current time step.
- Vec [mask_U](#)
- Vec [mask_V](#)
- Vec [mask_W](#)
Mask vectors for boundary conditions.

6.10.1 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.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 transport_problem_x() [1/2]

```
transport_problem_x::transport_problem_x (
    DM const & dmGrid_Shifted,
    DM const & dmGrid_Staggered,
    DM const & dmGrid_Centered ) [inline]
```

Constructor initializing the transport problem with given grids.

6.10.2.2 transport_problem_x() [2/2]

```
transport_problem_x::transport_problem_x ( ) [inline]
```

Default constructor for stand-alone transport problem.

6.10.2.3 ~transport_problem_x()

```
transport_problem_x::~~transport_problem_x ( ) [inline]
```

Destructor to clean up allocated resources.

6.10.3 Member Function Documentation**6.10.3.1 CenterU()**

```
PetscErrorCode transport_problem_x::CenterU (
    Vec & UCenter,
    Vec const & vec,
    PetscReal const & theta ) [protected]
```

Centers the x-velocity component in cell-centers.

6.10.3.2 Derive_x()

```
PetscErrorCode transport_problem_x::Derive_x (
    Vec & U2_x,
    Vec const & vec,
    PetscReal const & theta ) [protected]
```

Computes the first derivative in the x-direction, placing the result on faces.

6.10.3.3 FirstDerive_y()

```
PetscErrorCode transport_problem_x::FirstDerive_y (
    Vec & AB_y,
    Vec const & AB ) [protected]
```

Computes the first derivative in the y-direction, placing the result on faces.

6.10.3.4 FirstDerive_z()

```
PetscErrorCode transport_problem_x::FirstDerive_z (
    Vec & AB_z,
    Vec const & AB ) [protected]
```

Computes the first derivative in the z-direction, placing the result on faces.

6.10.3.5 FirstShiftU_y()

```
PetscErrorCode transport_problem_x::FirstShiftU_y (
    Vec & UShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for x-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.

6.10.3.6 FirstShiftU_z()

```
PetscErrorCode transport_problem_x::FirstShiftU_z (
    Vec & UShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for x-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.10.3.7 FirstShiftV_y()

```
PetscErrorCode transport_problem_x::FirstShiftV_y (
    Vec & VShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for y-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.

6.10.3.8 FirstShiftW_z()

```
PetscErrorCode transport_problem_x::FirstShiftW_z (
    Vec & WShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.10.3.9 solve_step_x()

```
PetscErrorCode const transport_problem_x::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.

6.10.4 Member Data Documentation

6.10.4.1 dmGrid_Centered

```
DM transport_problem_x::dmGrid_Centered [protected]
```

DMGrid with dofs of faces and centers.

6.10.4.2 dmGrid_Shifted

DM transport_problem_x::dmGrid_Shifted [protected]
DMGrid with dofs on faces and edges.

6.10.4.3 dmGrid_Staggered

DM transport_problem_x::dmGrid_Staggered [protected]
DMGrid with dofs of faces.

6.10.4.4 mask_U

Vec transport_problem_x::mask_U [protected]

6.10.4.5 mask_V

Vec transport_problem_x::mask_V [protected]

6.10.4.6 mask_W

Vec transport_problem_x::mask_W [protected]
Mask vectors for boundary conditions.

6.10.4.7 U_n

Vec transport_problem_x::U_n [protected]

6.10.4.8 V_n

Vec transport_problem_x::V_n [protected]

6.10.4.9 W_n

Vec transport_problem_x::W_n [protected]

Velocity fields at the current time step.

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

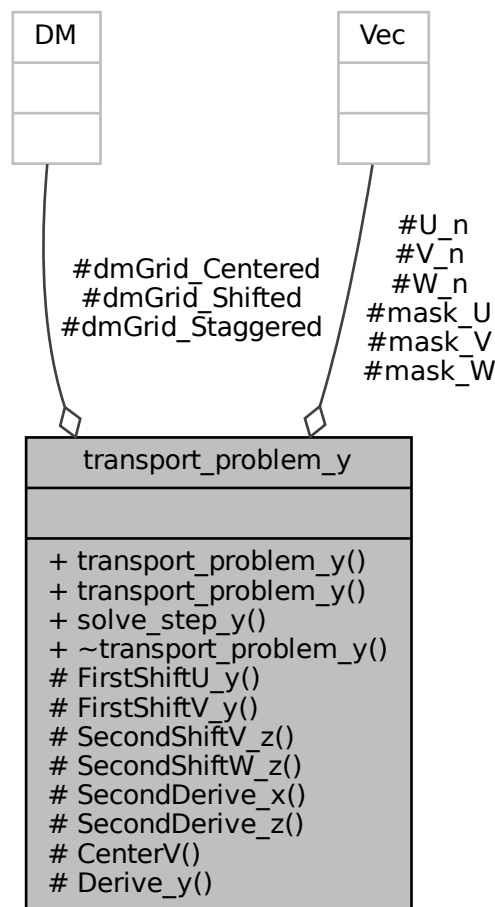
- [include/transport.hpp](#)
- [src/transport.cpp](#)

6.11 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.

Protected Member Functions

- PetscErrorCode [FirstShiftU_y](#) (Vec &UShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for x-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.

- PetscErrorCode [FirstShiftV_y](#) (Vec &VShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for y-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.
- PetscErrorCode [SecondShiftV_z](#) (Vec &VShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for y-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [SecondShiftW_z](#) (Vec &WShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [SecondDerive_x](#) (Vec &AB_x, Vec const &AB)
Computes the first derivative in the x-direction, placing the result on faces.
- PetscErrorCode [SecondDerive_z](#) (Vec &AB_z, Vec const &AB)
Computes the first derivative in the z-direction, placing the result on faces.
- PetscErrorCode [CenterV](#) (Vec &VCenter, Vec const &vec, PetscReal const &theta)
Centers the y-velocity component in cell-centers.
- PetscErrorCode [Derive_y](#) (Vec &V2_y, Vec const &vec, PetscReal const &theta)
Computes the first derivative in the y-direction, placing the result on faces.

Protected Attributes

- DM [dmGrid_Shifted](#)
DMGrid with dofs on faces and edges.
- DM [dmGrid_Staggered](#)
DMGrid with dofs of faces.
- DM [dmGrid_Centered](#)
DMGrid with dofs of faces and centers.
- Vec [U_n](#)
- Vec [V_n](#)
- Vec [W_n](#)
Velocity fields at the current time step.
- Vec [mask_U](#)
- Vec [mask_V](#)
- Vec [mask_W](#)
Mask vectors for boundary conditions.

6.11.1 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.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 `transport_problem_y()` [1/2]

```
transport_problem_y::transport_problem_y (
    DM const & dmGrid_Shifted,
    DM const & dmGrid_Staggered,
    DM const & dmGrid_Centered ) [inline]
```

Constructor initializing the transport problem with given grids.

6.11.2.2 transport_problem_y() [2/2]

`transport_problem_y::transport_problem_y () [inline]`
 Default constructor for stand-alone transport problem.

6.11.2.3 ~transport_problem_y()

`transport_problem_y::~~transport_problem_y () [inline]`
 Destructor to clean up allocated resources.

6.11.3 Member Function Documentation

6.11.3.1 CenterV()

`PetscErrorCode transport_problem_y::CenterV (`
 `Vec & VCenter,`
 `Vec const & vec,`
 `PetscReal const & theta) [protected]`

Centers the y-velocity component in cell-centers.

6.11.3.2 Derive_y()

`PetscErrorCode transport_problem_y::Derive_y (`
 `Vec & V2_y,`
 `Vec const & vec,`
 `PetscReal const & theta) [protected]`

Computes the first derivative in the y-direction, placing the result on faces.

6.11.3.3 FirstShiftU_y()

`PetscErrorCode transport_problem_y::FirstShiftU_y (`
 `Vec & UShifted,`
 `Vec const & vec,`
 `PetscScalar const & theta) [protected]`

Performs the shift for x-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.

6.11.3.4 FirstShiftV_y()

`PetscErrorCode transport_problem_y::FirstShiftV_y (`
 `Vec & VShifted,`
 `Vec const & vec,`
 `PetscScalar const & theta) [protected]`

Performs the shift for y-component in a 3D transport non-linear problem in the y-direction. Interpolates velocity component on edges.

6.11.3.5 SecondDerive_x()

`PetscErrorCode transport_problem_y::SecondDerive_x (`
 `Vec & AB_x,`
 `Vec const & AB) [protected]`

Computes the first derivative in the x-direction, placing the result on faces.

6.11.3.6 SecondDerive_z()

```
PetscErrorCode transport_problem_y::SecondDerive_z (
    Vec & AB_z,
    Vec const & AB ) [protected]
```

Computes the first derivative in the z-direction, placing the result on faces.

6.11.3.7 SecondShiftV_z()

```
PetscErrorCode transport_problem_y::SecondShiftV_z (
    Vec & VShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for y-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.11.3.8 SecondShiftW_z()

```
PetscErrorCode transport_problem_y::SecondShiftW_z (
    Vec & WShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.11.3.9 solve_step_y()

```
PetscErrorCode const transport_problem_y::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.

6.11.4 Member Data Documentation

6.11.4.1 dmGrid_Centered

```
DM transport_problem_y::dmGrid_Centered [protected]
```

DMGrid with dofs of faces and centers.

6.11.4.2 dmGrid_Shifted

```
DM transport_problem_y::dmGrid_Shifted [protected]
```

DMGrid with dofs on faces and edges.

6.11.4.3 dmGrid_Staggered

```
DM transport_problem_y::dmGrid_Staggered [protected]
```

DMGrid with dofs of faces.

6.11.4.4 mask_U

Vec transport_problem_y::mask_U [protected]

6.11.4.5 mask_V

Vec transport_problem_y::mask_V [protected]

6.11.4.6 mask_W

Vec transport_problem_y::mask_W [protected]

Mask vectors for boundary conditions.

6.11.4.7 U_n

Vec transport_problem_y::U_n [protected]

6.11.4.8 V_n

Vec transport_problem_y::V_n [protected]

6.11.4.9 W_n

Vec transport_problem_y::W_n [protected]

Velocity fields at the current time step.

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

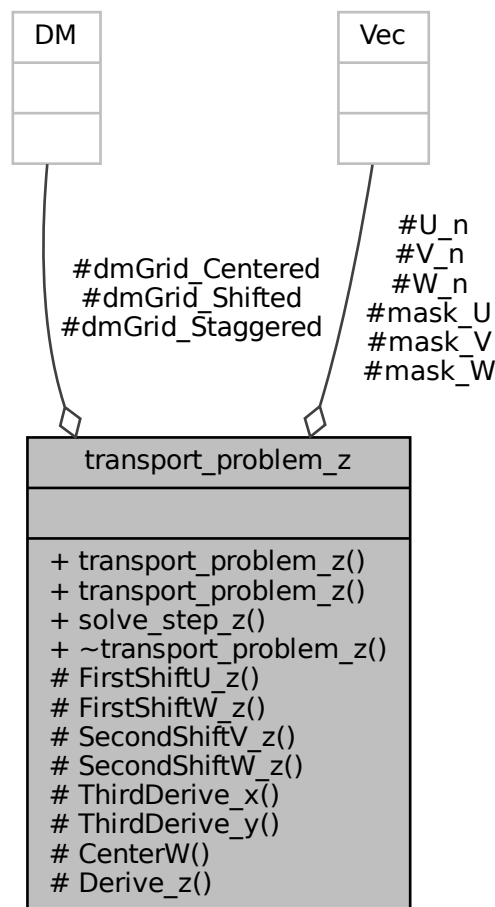
- include/[transport.hpp](#)
- src/[transport.cpp](#)

6.12 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.

Protected Member Functions

- `PetscErrorCode` `FirstShiftU_z` (`Vec` &`UShifted`, `Vec` const &`vec`, `PetscScalar` const &`theta`)
Performs the shift for x-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

- PetscErrorCode [FirstShiftW_z](#) (Vec &WShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [SecondShiftV_z](#) (Vec &VShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for y-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [SecondShiftW_z](#) (Vec &WShifted, Vec const &vec, PetscScalar const &theta)
Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.
- PetscErrorCode [ThirdDerive_x](#) (Vec &AB_x, Vec const &AB)
Computes the first derivative in the x-direction, placing the result on faces.
- PetscErrorCode [ThirdDerive_y](#) (Vec &AB_y, Vec const &AB)
Computes the first derivative in the y-direction, placing the result on faces.
- PetscErrorCode [CenterW](#) (Vec &WCenter, Vec const &vec, PetscReal const &theta)
Centers the z-velocity component in cell-centers.
- PetscErrorCode [Derive_z](#) (Vec &W2_z, Vec const &vec, PetscReal const &theta)
Computes the first derivative in the z-direction, placing the result on faces.

Protected Attributes

- DM [dmGrid_Shifted](#)
DMGrid with dofs on faces and edges.
- DM [dmGrid_Staggered](#)
DMGrid with dofs of faces.
- DM [dmGrid_Centered](#)
DMGrid with dofs of faces and centers.
- Vec [U_n](#)
- Vec [V_n](#)
- Vec [W_n](#)
Velocity fields at the current time step.
- Vec [mask_U](#)
- Vec [mask_V](#)
- Vec [mask_W](#)
Mask vectors for boundary conditions.

6.12.1 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.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 transport_problem_z() [1/2]

```
transport_problem_z::transport_problem_z (
    DM const & dmGrid_Shifted,
    DM const & dmGrid_Staggered,
    DM const & dmGrid_Centered ) [inline]
```

Constructor initializing the transport problem with given grids.

6.12.2.2 transport_problem_z() [2/2]

`transport_problem_z::transport_problem_z () [inline]`
 Default constructor for stand-alone transport problem.

6.12.2.3 ~transport_problem_z()

`transport_problem_z::~~transport_problem_z () [inline]`
 Destructor to clean up allocated resources.

6.12.3 Member Function Documentation

6.12.3.1 CenterW()

`PetscErrorCode transport_problem_z::CenterW (`
 `Vec & WCenter,`
 `Vec const & vec,`
 `PetscReal const & theta) [protected]`

Centers the z-velocity component in cell-centers.

6.12.3.2 Derive_z()

`PetscErrorCode transport_problem_z::Derive_z (`
 `Vec & W2_z,`
 `Vec const & vec,`
 `PetscReal const & theta) [protected]`

Computes the first derivative in the z-direction, placing the result on faces.

6.12.3.3 FirstShiftU_z()

`PetscErrorCode transport_problem_z::FirstShiftU_z (`
 `Vec & UShifted,`
 `Vec const & vec,`
 `PetscScalar const & theta) [protected]`

Performs the shift for x-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.12.3.4 FirstShiftW_z()

`PetscErrorCode transport_problem_z::FirstShiftW_z (`
 `Vec & WShifted,`
 `Vec const & vec,`
 `PetscScalar const & theta) [protected]`

Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.12.3.5 SecondShiftV_z()

`PetscErrorCode transport_problem_z::SecondShiftV_z (`
 `Vec & VShifted,`
 `Vec const & vec,`
 `PetscScalar const & theta) [protected]`

Performs the shift for y-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.12.3.6 SecondShiftW_z()

```
PetscErrorCode transport_problem_z::SecondShiftW_z (
    Vec & WShifted,
    Vec const & vec,
    PetscScalar const & theta ) [protected]
```

Performs the shift for z-component in a 3D transport non-linear problem in the z-direction. Interpolates velocity component on edges.

6.12.3.7 solve_step_z()

```
PetscErrorCode const transport_problem_z::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.

6.12.3.8 ThirdDerive_x()

```
PetscErrorCode transport_problem_z::ThirdDerive_x (
    Vec & AB_x,
    Vec const & AB ) [protected]
```

Computes the first derivative in the x-direction, placing the result on faces.

6.12.3.9 ThirdDerive_y()

```
PetscErrorCode transport_problem_z::ThirdDerive_y (
    Vec & AB_y,
    Vec const & AB ) [protected]
```

Computes the first derivative in the y-direction, placing the result on faces.

6.12.4 Member Data Documentation

6.12.4.1 dmGrid_Centered

```
DM transport_problem_z::dmGrid_Centered [protected]
```

DMGrid with dofs of faces and centers.

6.12.4.2 dmGrid_Shifted

```
DM transport_problem_z::dmGrid_Shifted [protected]
```

DMGrid with dofs on faces and edges.

6.12.4.3 dmGrid_Staggered

```
DM transport_problem_z::dmGrid_Staggered [protected]
```

DMGrid with dofs of faces.

6.12.4.4 mask_U

`Vec transport_problem_z::mask_U [protected]`

6.12.4.5 mask_V

`Vec transport_problem_z::mask_V [protected]`

6.12.4.6 mask_W

`Vec transport_problem_z::mask_W [protected]`

Mask vectors for boundary conditions.

6.12.4.7 U_n

`Vec transport_problem_z::U_n [protected]`

6.12.4.8 V_n

`Vec transport_problem_z::V_n [protected]`

6.12.4.9 W_n

`Vec transport_problem_z::W_n [protected]`

Velocity fields at the current time step.

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

- [include/transport.hpp](#)
- [src/transport.cpp](#)

Chapter 7

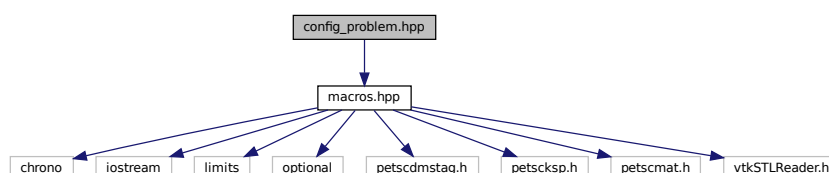
File Documentation

7.1 config_problem.hpp File Reference

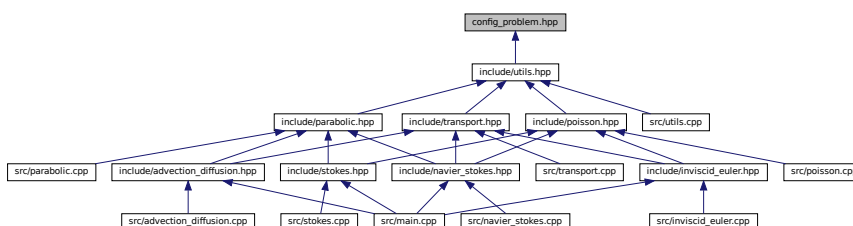
Configuration file for setting up problem parameters.

```
#include "macros.hpp"
```

Include dependency graph for config_problem.hpp:



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



Namespaces

- [problem_setting](#)

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::problem_type](#) = "navier_stokes"
Set the problem type. Possible values are "navier_stokes", "stokes", "euler", "advection_diffusion", "parabolic_x", "parabolic_y", "parabolic_z", "transport_x", "transport_y", "transport_z".
- 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. Only implemented for NAvier-Stokes and parabolic problems.
- 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.
- 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 $\mu = 1/Re$ (dimensional 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](#)

7.1.1 Detailed Description

Configuration file for setting up problem parameters.

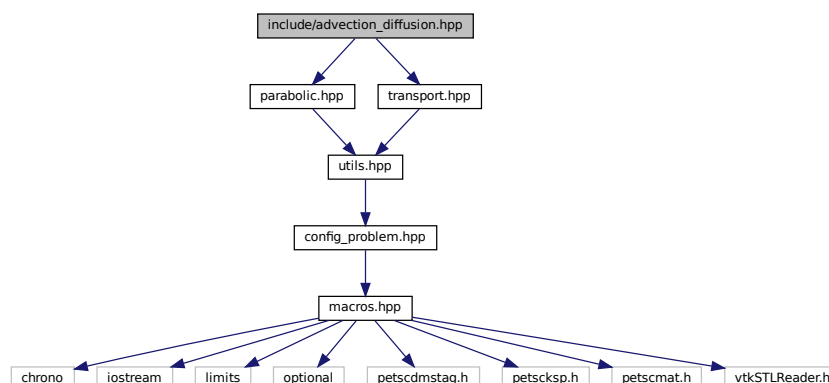
This file defines the parameters for simulations. Before starting any simulation, careful consideration of the parameters is required. It is recommended to work in a dimensionless framework.

7.2 include/advection_diffusion.hpp File Reference

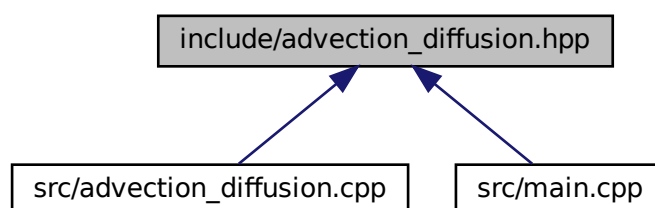
```
#include "parabolic.hpp"
```

```
#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](#)

Macros

- `#define` [ADVECTION_DIFFUSION_PROBLEM_HPP](#)

7.2.1 Macro Definition Documentation

7.2.1.1 ADVECTION_DIFFUSION_PROBLEM_HPP

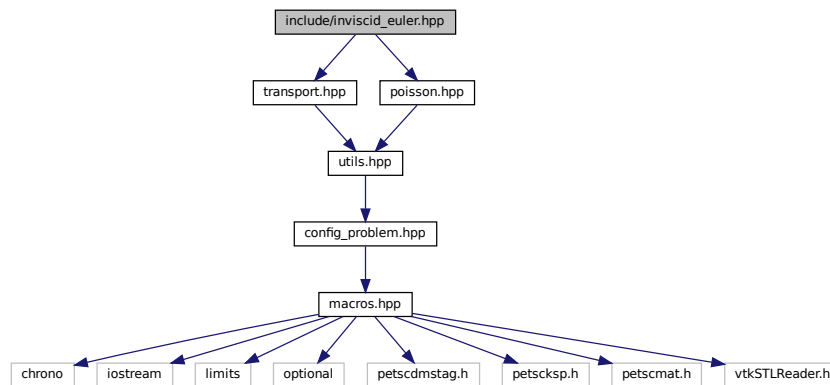
```
#define ADVECTION_DIFFUSION_PROBLEM_HPP
```

7.3 include/inviscid_euler.hpp File Reference

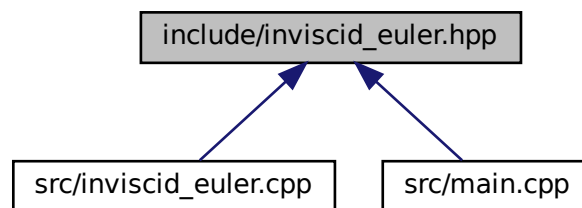
```
#include "transport.hpp"
```

```
#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 Brinkam penalty method.

Macros

- `#define` [EULER_PROBLEM_HPP](#)

7.3.1 Macro Definition Documentation

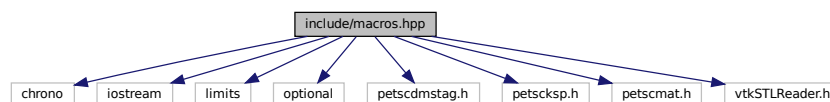
7.3.1.1 EULER_PROBLEM_HPP

```
#define EULER_PROBLEM_HPP
```

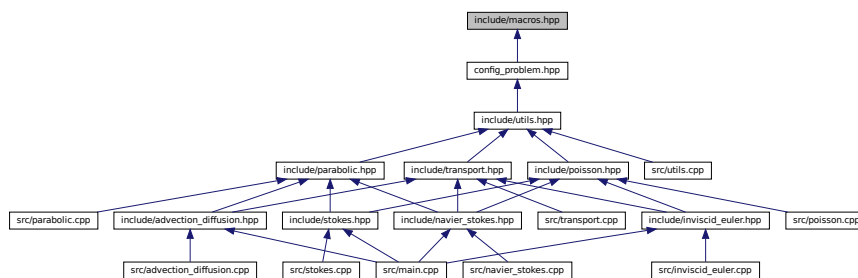

7.4 include/macros.hpp File Reference

```
#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.

7.4.1 Macro Definition Documentation

7.4.1.1 BACK

```
#define BACK DMSTAG_BACK
```

7.4.1.2 BACK_DOWN

```
#define BACK_DOWN DMSTAG_BACK_DOWN
```

7.4.1.3 BACK_LEFT

```
#define BACK_LEFT DMSTAG_BACK_LEFT
```

7.4.1.4 BACK_RIGHT

```
#define BACK_RIGHT DMSTAG_BACK_RIGHT
```

7.4.1.5 BACK_UP

```
#define BACK_UP DMSTAG_BACK_UP
```

7.4.1.6 DOWN

```
#define DOWN DMSTAG_DOWN
```

7.4.1.7 DOWN_LEFT

```
#define DOWN_LEFT DMSTAG_DOWN_LEFT
```

7.4.1.8 DOWN_RIGHT

```
#define DOWN_RIGHT DMSTAG_DOWN_RIGHT
```

7.4.1.9 ELEMENT

```
#define ELEMENT DMSTAG_ELEMENT
```

7.4.1.10 FRONT

```
#define FRONT DMSTAG_FRONT
```

7.4.1.11 FRONT_DOWN

```
#define FRONT_DOWN DMSTAG_FRONT_DOWN
```

7.4.1.12 FRONT_LEFT

```
#define FRONT_LEFT DMSTAG_FRONT_LEFT
```

7.4.1.13 FRONT_RIGHT

```
#define FRONT_RIGHT DMSTAG_FRONT_RIGHT
```

7.4.1.14 FRONT_UP

```
#define FRONT_UP DMSTAG_FRONT_UP
```

7.4.1.15 LEFT

```
#define LEFT DMSTAG_LEFT
```

7.4.1.16 RIGHT

```
#define RIGHT DMSTAG_RIGHT
```

7.4.1.17 UP

```
#define UP DMSTAG_UP
```

7.4.1.18 UP_LEFT

```
#define UP_LEFT DMSTAG_UP_LEFT
```

7.4.1.19 UP_RIGHT

```
#define UP_RIGHT DMSTAG_UP_RIGHT
```

7.4.2 Variable Documentation

7.4.2.1 eps

```
constexpr PetscReal eps = 1e6 [constexpr]
```

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

7.4.2.2 pi

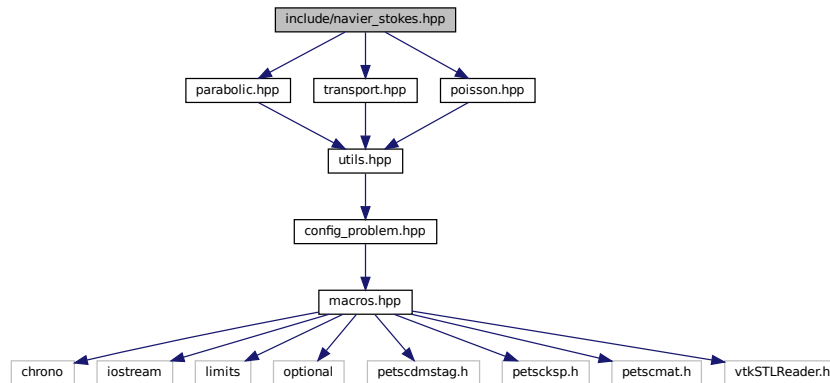
```
constexpr PetscReal pi = 3.14159265358979323846 [constexpr]
```

The mathematical constant pi.

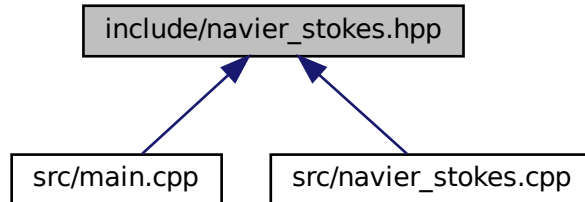
7.5 include/navier_stokes.hpp File Reference

```
#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

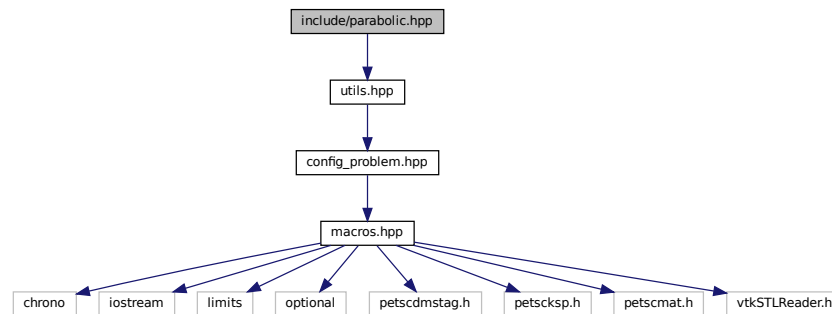
- class [navier_stokes_problem](#)

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 Brinkam penalty method.

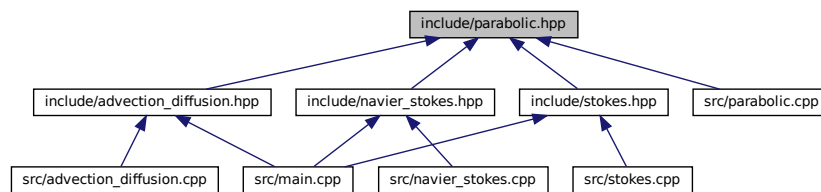
7.6 include/parabolic.hpp File Reference

```
#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.

Macros

- `#define` [PARABOLIC_PROBLEM_Y_HPP](#)
- `#define` [PARABOLIC_PROBLEM_Z_HPP](#)

7.6.1 Macro Definition Documentation

7.6.1.1 PARABOLIC_PROBLEM_Y_HPP

```
#define PARABOLIC_PROBLEM_Y_HPP
```

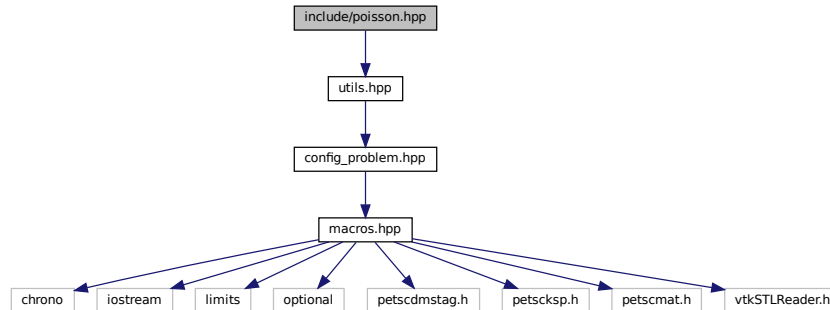
7.6.1.2 PARABOLIC_PROBLEM_Z_HPP

```
#define PARABOLIC_PROBLEM_Z_HPP
```

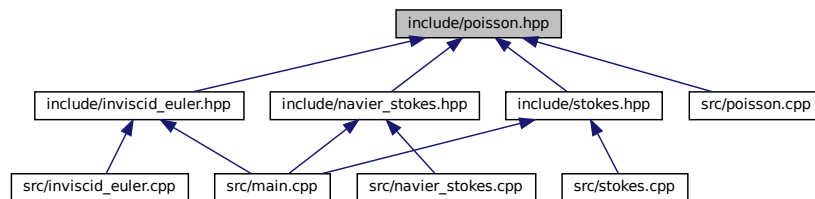
7.7 include/poisson.hpp File Reference

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

Include dependency graph for poisson.hpp:



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



Classes

- class [poisson_problem](#)

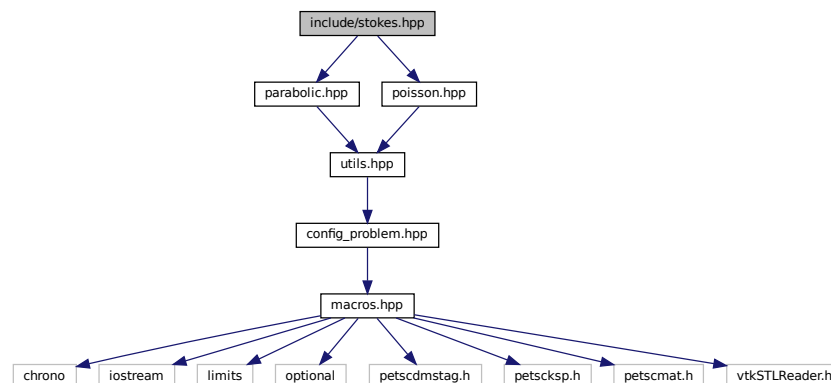
Represents a Poisson equation solver for pressure correction in fluid simulations.

7.8 include/stokes.hpp File Reference

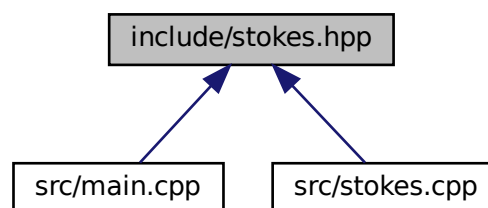
```
#include "parabolic.hpp"
```

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

Include dependency graph for stokes.hpp:



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



Classes

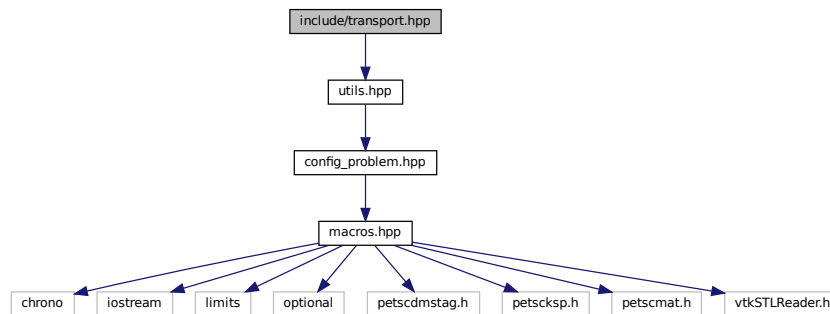
- class [stokes_problem](#)

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 Brinkam penalty method.

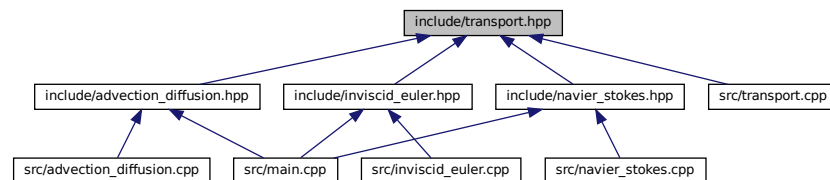
7.9 include/transport.hpp File Reference

```
#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.

Macros

- `#define` [TRANSPORT_PROBLEM_Y_HPP](#)
- `#define` [TRANSPORT_PROBLEM_Z_HPP](#)

7.9.1 Macro Definition Documentation

7.9.1.1 TRANSPORT_PROBLEM_Y_HPP

```
#define TRANSPORT_PROBLEM_Y_HPP
```

7.9.1.2 TRANSPORT_PROBLEM_Z_HPP

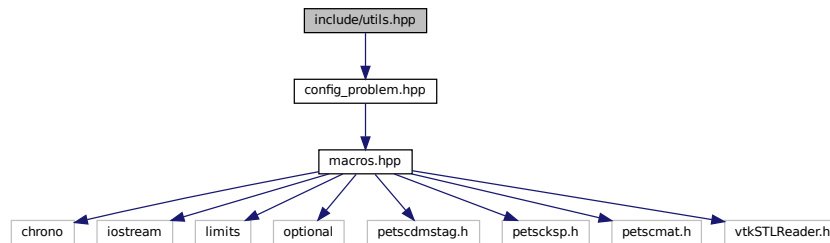
```
#define TRANSPORT_PROBLEM_Z_HPP
```


7.10 include/utils.hpp File Reference

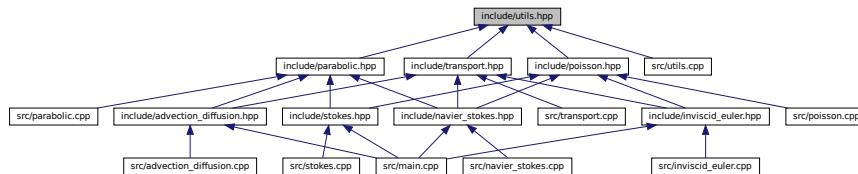
Utility functions for grid creation, analytical solutions, and geometric operations.

```
#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 evaluates 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-centers. 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 > &ray← Vector, 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.

7.10.1 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.

7.10.2 Function Documentation

7.10.2.1 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 evaluates L2-norm.

Parameters

<i>sol</i>	Computed solution vector.
<i>solRef</i>	Reference solution vector.

7.10.2.2 CreateAnalyticalP()

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

7.10.2.3 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

<i>dmGrid</i>	Discretized grid.
<i>vec</i>	Output velocity field.

7.10.2.4 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

<i>dmGrid</i>	Discretized grid.
<i>vec</i>	Output velocity field.

7.10.2.5 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

<i>dmGrid</i>	Discretized grid.
<i>vec</i>	Output velocity field.

7.10.2.6 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-centers. Pass 0 to not allow dofs in some position, 1 allocate dofs.

Parameters

<i>dmGrid</i>	Pointer to the grid.
<i>dof1</i>	Degrees of freedom on edges.
<i>dof2</i>	Degrees of freedom of faces.
<i>dof3</i>	Degrees of freedom at cell-centers.

7.10.2.7 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

<i>dmGrid</i>	Discretized grid.
<i>vec_stag</i>	Output mask vector.
<i>vertices</i>	List of mesh vertices.
<i>faces</i>	List of mesh faces.

7.10.2.8 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

<i>dmGrid</i>	Discretized grid.
<i>vec_stag</i>	Output mask vector.
<i>vertices</i>	List of mesh vertices.
<i>faces</i>	List of mesh faces.

7.10.2.9 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

<i>dmGrid</i>	Discretized grid.
<i>vec_stag</i>	Output mask vector.
<i>vertices</i>	List of mesh vertices.
<i>faces</i>	List of mesh faces.

7.10.2.10 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

<i>point</i>	The point to be checked.
<i>vertices</i>	List of vertices of the mesh.
<i>faces</i>	List of faces of the mesh.

Returns

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

7.10.2.11 PrintSimulationParameters()

```
void PrintSimulationParameters ( )
```

Prints the simulation parameters to the console.

7.10.2.12 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

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

Returns

True if the ray intersects the triangle, false otherwise.

7.10.2.13 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

<i>filename</i>	The name of the mesh file.
<i>vertices</i>	Output list of vertices.
<i>faces</i>	Output list of faces.

7.10.3 Variable Documentation**7.10.3.1 D_x**

```
constexpr PetscReal D_x {Lx - Lx_0} [constexpr]
```

Domain length in the x-direction.

7.10.3.2 D_y

```
constexpr PetscReal D_y {Ly - Ly_0} [constexpr]
```

Domain length in the y-direction.

7.10.3.3 D_z

```
constexpr PetscReal D_z {Lz - Lz_0} [constexpr]
```

Domain length in the z-direction.

7.10.3.4 faces

```
std::vector<std::array<int, 3> > faces [inline]
```

List of mesh faces.

7.10.3.5 filename

```
std::string filename [inline]
```

Filename of the geometry file.

7.10.3.6 vertices

```
std::vector<std::array<double, 3> > vertices [inline]
```

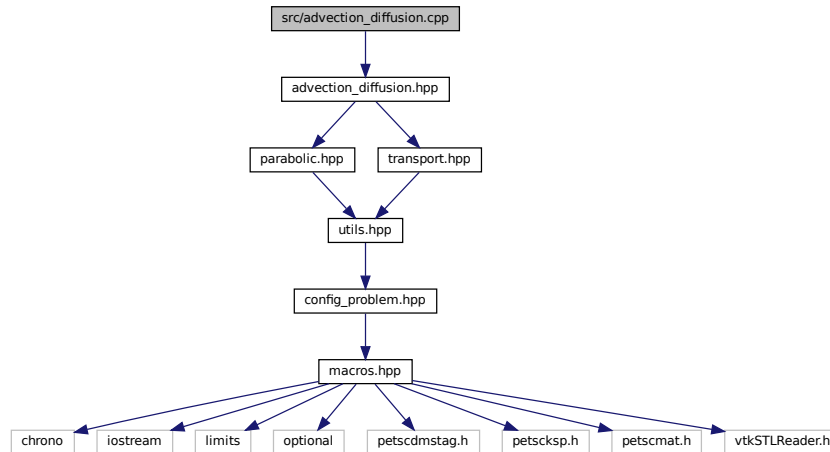
List of mesh vertices.

7.11 README.md File Reference

7.12 src/advection_diffusion.cpp File Reference

```
#include "advection_diffusion.hpp"
```

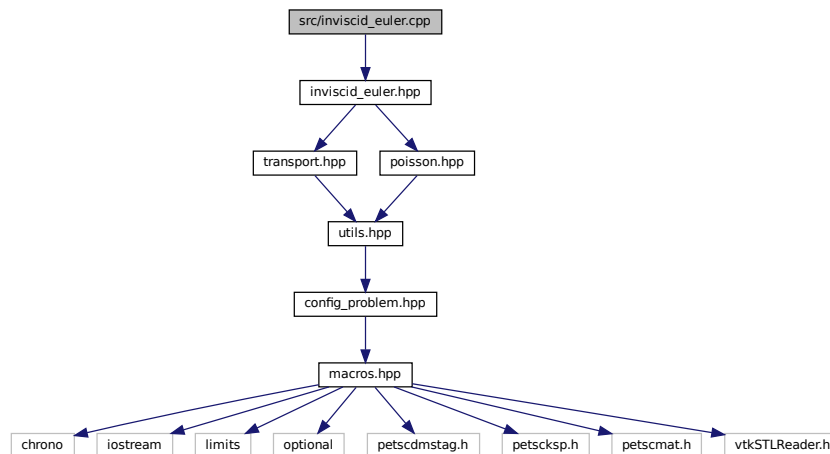
Include dependency graph for advection_diffusion.cpp:



7.13 src/inviscid_euler.cpp File Reference

```
#include "inviscid_euler.hpp"
```

Include dependency graph for inviscid_euler.cpp:



7.14 src/main.cpp File Reference

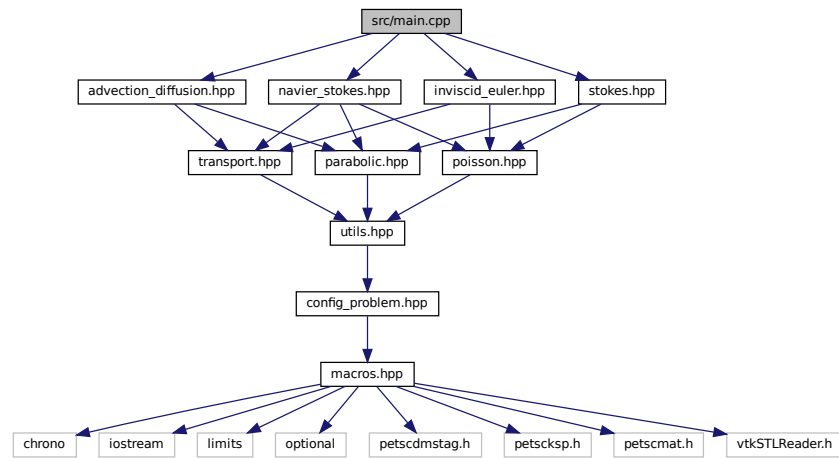
```
#include "navier_stokes.hpp"
```

```
#include "inviscid_euler.hpp"
```

```
#include "stokes.hpp"
```

```
#include "advection_diffusion.hpp"
```

Include dependency graph for main.cpp:



Functions

- int [main](#) (int argc, char **argv)

7.14.1 Function Documentation

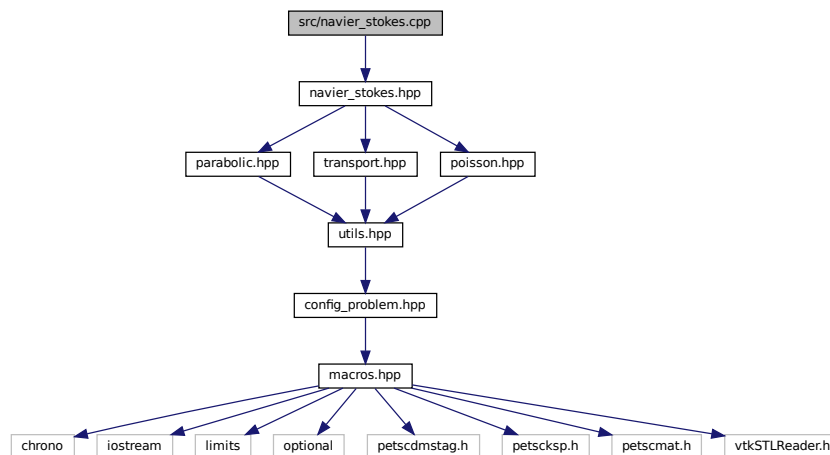
7.14.1.1 main()

```
int main (  
    int argc,  
    char ** argv )
```


7.15 src/navier_stokes.cpp File Reference

```
#include "navier_stokes.hpp"
```

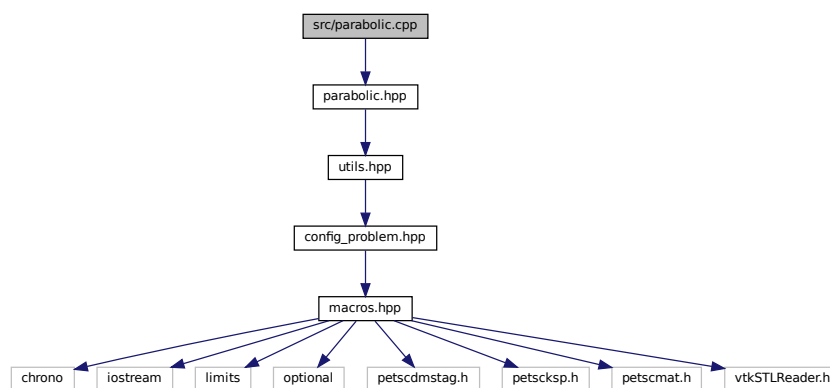
Include dependency graph for navier_stokes.cpp:



7.16 src/parabolic.cpp File Reference

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

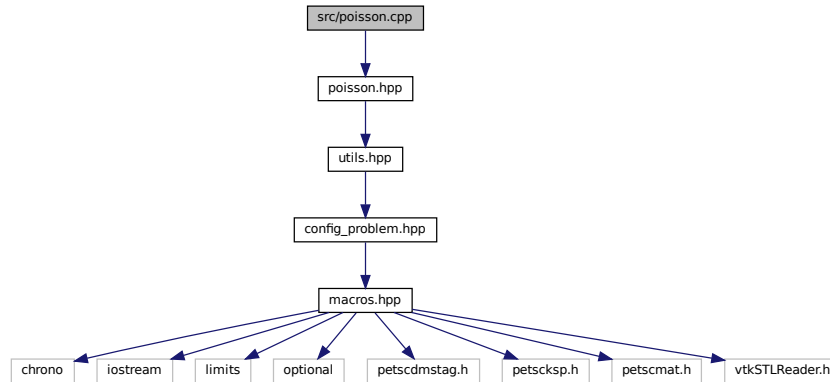
Include dependency graph for parabolic.cpp:



7.17 src/poisson.cpp File Reference

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

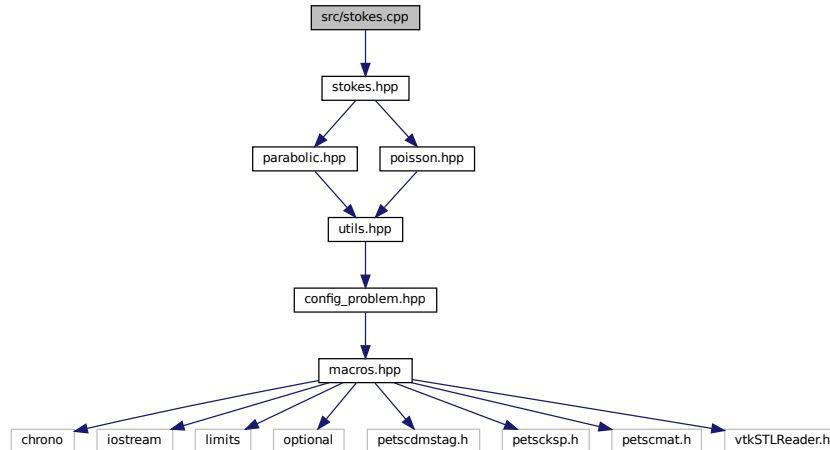
Include dependency graph for poisson.cpp:



7.18 src/stokes.cpp File Reference

```
#include "stokes.hpp"
```

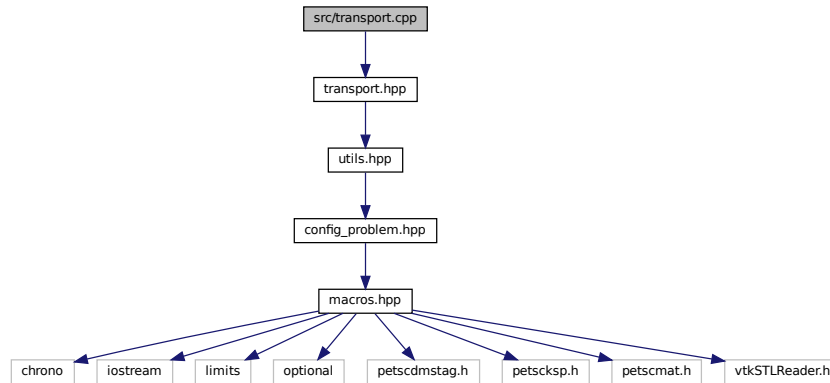
Include dependency graph for stokes.cpp:



7.19 src/transport.cpp File Reference

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

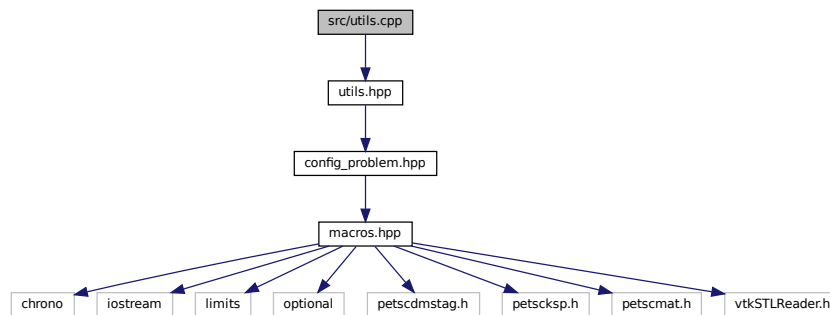
Include dependency graph for transport.cpp:



7.20 src/utils.cpp File Reference

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

Include dependency graph for utils.cpp:



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 evaluates 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 [CreateAnalyticalP](#) (DM const &dmGrid, Vec &vec, PetscReal const &theta)
- 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-centers. Pass 0 to not allow dofs in some position, 1 allocate dofs.

- 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.

7.20.1 Function Documentation

7.20.1.1 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 evaluates L2-norm.

Parameters

<i>sol</i>	Computed solution vector.
<i>solRef</i>	Reference solution vector.

7.20.1.2 CreateAnalyticalP()

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

7.20.1.3 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

<i>dmGrid</i>	Discretized grid.
<i>vec</i>	Output velocity field.

7.20.1.4 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

<i>dmGrid</i>	Discretized grid.
<i>vec</i>	Output velocity field.

7.20.1.5 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

<i>dmGrid</i>	Discretized grid.
<i>vec</i>	Output velocity field.

7.20.1.6 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-centers. Pass 0 to not allow dofs in some position, 1 allocate dofs.

Parameters

<i>dmGrid</i>	Pointer to the grid.
<i>dof1</i>	Degrees of freedom on edges.
<i>dof2</i>	Degrees of freedom of faces.
<i>dof3</i>	Degrees of freedom at cell-centers.

7.20.1.7 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

<i>dmGrid</i>	Discretized grid.
<i>vec_stag</i>	Output mask vector.
<i>vertices</i>	List of mesh vertices.
<i>faces</i>	List of mesh faces.

7.20.1.8 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

<i>dmGrid</i>	Discretized grid.
<i>vec_stag</i>	Output mask vector.
<i>vertices</i>	List of mesh vertices.
<i>faces</i>	List of mesh faces.

7.20.1.9 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

<i>dmGrid</i>	Discretized grid.
<i>vec_stag</i>	Output mask vector.
<i>vertices</i>	List of mesh vertices.
<i>faces</i>	List of mesh faces.

7.20.1.10 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

<i>point</i>	The point to be checked.
<i>vertices</i>	List of vertices of the mesh.
<i>faces</i>	List of faces of the mesh.

Returns

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

7.20.1.11 PrintSimulationParameters()

```
void PrintSimulationParameters ( )
```

Prints the simulation parameters to the console.

7.20.1.12 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

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

Returns

True if the ray intersects the triangle, false otherwise.

7.20.1.13 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

<i>filename</i>	The name of the mesh file.
-----------------	----------------------------

Parameters

<i>vertices</i>	Output list of vertices.
<i>faces</i>	Output list of faces.

Index

- ~advection_diffusion_problem
 - advection_diffusion_problem, 18
- ~euler_problem
 - euler_problem, 23
- ~navier_stokes_problem
 - navier_stokes_problem, 29
- ~parabolic_problem_x
 - parabolic_problem_x, 34
- ~parabolic_problem_y
 - parabolic_problem_y, 37
- ~parabolic_problem_z
 - parabolic_problem_z, 41
- ~poisson_problem
 - poisson_problem, 46
- ~stokes_problem
 - stokes_problem, 53
- ~transport_problem_x
 - transport_problem_x, 58
- ~transport_problem_y
 - transport_problem_y, 63
- ~transport_problem_z
 - transport_problem_z, 68
- A
 - parabolic_problem_x, 35
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
 - poisson_problem, 49
- a
 - problem_setting, 11
- advection, 15
- advection_diffusion.hpp
 - ADVECTION_DIFFUSION_PROBLEM_HPP, 73
- advection_diffusion_problem, 15
 - ~advection_diffusion_problem, 18
 - advection_diffusion_problem, 17, 18
 - assemble_magnitude, 18
 - compute_magnitude, 18
 - dmGrid_cent_rich, 19
 - dmGrid_centered, 19
 - dmGrid_shift_transp, 19
 - dmGrid_stag_transp, 19
 - dmGrid_staggered_x, 19
 - dmGrid_staggered_y, 19
 - dmGrid_staggered_z, 19
 - exodus, 18
 - Magnitude, 19
 - mask_U, 20
 - mask_V, 20
 - mask_W, 20
 - solve, 18
 - U_up, 20
 - update_velocity, 19
 - V_up, 20
 - W_up, 20
- ADVECTION_DIFFUSION_PROBLEM_HPP
 - advection_diffusion.hpp, 73
- assemble_divergence
 - poisson_problem, 46
- assemble_lhs
 - parabolic_problem_x, 34
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
 - poisson_problem, 47
- assemble_magnitude
 - advection_diffusion_problem, 18
 - euler_problem, 23
 - navier_stokes_problem, 29
 - stokes_problem, 53
- assemble_rhs
 - parabolic_problem_x, 34
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
- BACK
 - macros.hpp, 76
- BACK_DOWN
 - macros.hpp, 76
- BACK_LEFT
 - macros.hpp, 76
- BACK_RIGHT
 - macros.hpp, 76
- BACK_UP
 - macros.hpp, 76
- base_path
 - problem_setting, 11
- brinkman
 - problem_setting, 11
- CenterU
 - transport_problem_x, 58
- CenterV
 - transport_problem_y, 63
- CenterW
 - transport_problem_z, 68
- check_convergence
 - problem_setting, 12
- CheckSolution
 - utils.cpp, 94
 - utils.hpp, 84

- compute_divergence
 - poisson_problem, 47
- compute_magnitude
 - advection_diffusion_problem, 18
 - euler_problem, 23
 - navier_stokes_problem, 29
 - stokes_problem, 53
- config_problem.hpp, 71
- CreateAnalyticalP
 - utils.cpp, 94
 - utils.hpp, 84
- CreateAnalyticalU
 - utils.cpp, 94
 - utils.hpp, 85
- CreateAnalyticalV
 - utils.cpp, 95
 - utils.hpp, 85
- CreateAnalyticalW
 - utils.cpp, 95
 - utils.hpp, 85
- CreateGrid
 - utils.cpp, 95
 - utils.hpp, 85
- createMaskU
 - utils.cpp, 95
 - utils.hpp, 86
- createMaskV
 - utils.cpp, 96
 - utils.hpp, 86
- createMaskW
 - utils.cpp, 96
 - utils.hpp, 86
- d
 - problem_setting, 12
- D_x
 - utils.hpp, 88
- D_y
 - utils.hpp, 88
- D_z
 - utils.hpp, 88
- Derive_x
 - transport_problem_x, 58
- derive_x_P
 - poisson_problem, 47
- Derive_y
 - transport_problem_y, 63
- derive_y_P
 - poisson_problem, 47
- Derive_z
 - transport_problem_z, 68
- derive_z_P
 - poisson_problem, 48
- dmGrid
 - parabolic_problem_x, 35
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
- dmGrid_centrich
 - advection_diffusion_problem, 19
- euler_problem, 24
- navier_stokes_problem, 30
- poisson_problem, 49
- stokes_problem, 54
- dmGrid_Centered
 - transport_problem_x, 59
 - transport_problem_y, 64
 - transport_problem_z, 69
- dmGrid_centered
 - advection_diffusion_problem, 19
 - euler_problem, 24
 - navier_stokes_problem, 30
 - poisson_problem, 49
 - stokes_problem, 54
- dmGrid_shift_transp
 - advection_diffusion_problem, 19
 - euler_problem, 24
 - navier_stokes_problem, 30
- dmGrid_Shifted
 - transport_problem_x, 59
 - transport_problem_y, 64
 - transport_problem_z, 69
- dmGrid_stag_transp
 - advection_diffusion_problem, 19
 - euler_problem, 24
 - navier_stokes_problem, 30
- dmGrid_Staggered
 - transport_problem_x, 60
 - transport_problem_y, 64
 - transport_problem_z, 69
- dmGrid_staggered_x
 - advection_diffusion_problem, 19
 - euler_problem, 24
 - navier_stokes_problem, 30
 - poisson_problem, 49
 - stokes_problem, 54
- dmGrid_staggered_y
 - advection_diffusion_problem, 19
 - euler_problem, 24
 - navier_stokes_problem, 30
 - poisson_problem, 49
 - stokes_problem, 54
- dmGrid_staggered_z
 - advection_diffusion_problem, 19
 - euler_problem, 25
 - navier_stokes_problem, 31
 - poisson_problem, 50
 - stokes_problem, 54
- DOWN
 - macros.hpp, 76
- DOWN_LEFT
 - macros.hpp, 76
- DOWN_RIGHT
 - macros.hpp, 76
- dt
 - problem_setting, 12
- ELEMENT
 - macros.hpp, 76

- eps
 - macros.hpp, 77
- euler_problem, 20
 - ~euler_problem, 23
 - assemble_magnitude, 23
 - compute_magnitude, 23
 - dmGrid_cent_rich, 24
 - dmGrid_centered, 24
 - dmGrid_shift_transp, 24
 - dmGrid_stag_transp, 24
 - dmGrid_staggered_x, 24
 - dmGrid_staggered_y, 24
 - dmGrid_staggered_z, 25
 - euler_problem, 23
 - exodus, 24
 - Magnitude, 25
 - mask_U, 25
 - mask_V, 25
 - mask_W, 25
 - P, 25
 - P_x, 25
 - P_y, 25
 - P_z, 25
 - solve, 24
 - U_up, 25
 - update_velocity, 24
 - V_up, 26
 - W_up, 26
- EULER_PROBLEM_HPP
 - inviscid_euler.hpp, 74
- exodus
 - advection_diffusion_problem, 18
 - euler_problem, 24
 - navier_stokes_problem, 30
 - parabolic_problem_x, 34
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
 - poisson_problem, 48
 - stokes_problem, 53
- faces
 - utils.hpp, 88
- filename
 - utils.hpp, 88
- FirstDerive_y
 - transport_problem_x, 58
- FirstDerive_z
 - transport_problem_x, 58
- FirstShiftU_y
 - transport_problem_x, 58
 - transport_problem_y, 63
- FirstShiftU_z
 - transport_problem_x, 59
 - transport_problem_z, 68
- FirstShiftV_y
 - transport_problem_x, 59
 - transport_problem_y, 63
- FirstShiftW_z
 - transport_problem_x, 59
- transport_problem_z, 68
- FRONT
 - macros.hpp, 76
- FRONT_DOWN
 - macros.hpp, 76
- FRONT_LEFT
 - macros.hpp, 77
- FRONT_RIGHT
 - macros.hpp, 77
- FRONT_UP
 - macros.hpp, 77
- get_U
 - parabolic_problem_x, 34
- get_V
 - parabolic_problem_y, 38
- get_W
 - parabolic_problem_z, 42
- include/advection_diffusion.hpp, 73
- include/inviscid_euler.hpp, 74
- include/macros.hpp, 75
- include/navier_stokes.hpp, 78
- include/parabolic.hpp, 78
- include/poisson.hpp, 80
- include/stokes.hpp, 80
- include/transport.hpp, 81
- include/utils.hpp, 83
- inviscid_euler.hpp
 - EULER_PROBLEM_HPP, 74
- isPointInsideMesh
 - utils.cpp, 96
 - utils.hpp, 87
- iter
 - problem_setting, 12
- LEFT
 - macros.hpp, 77
- Lx
 - problem_setting, 12
- Lx_0
 - problem_setting, 12
- Ly
 - problem_setting, 12
- Ly_0
 - problem_setting, 12
- Lz
 - problem_setting, 12
- Lz_0
 - problem_setting, 12
- macros.hpp
 - BACK, 76
 - BACK_DOWN, 76
 - BACK_LEFT, 76
 - BACK_RIGHT, 76
 - BACK_UP, 76
 - DOWN, 76
 - DOWN_LEFT, 76

- DOWN_RIGHT, 76
- ELEMENT, 76
- eps, 77
- FRONT, 76
- FRONT_DOWN, 76
- FRONT_LEFT, 77
- FRONT_RIGHT, 77
- FRONT_UP, 77
- LEFT, 77
- pi, 77
- RIGHT, 77
- UP, 77
- UP_LEFT, 77
- UP_RIGHT, 77
- Magnitude
 - advection_diffusion_problem, 19
 - euler_problem, 25
 - navier_stokes_problem, 31
 - stokes_problem, 54
- main
 - main.cpp, 90
- main.cpp
 - main, 90
- manage_pressure
 - poisson_problem, 48
- manage_pressure_x
 - poisson_problem, 48
- manage_pressure_y
 - poisson_problem, 49
- manage_pressure_z
 - poisson_problem, 49
- mask_U
 - advection_diffusion_problem, 20
 - euler_problem, 25
 - navier_stokes_problem, 31
 - parabolic_problem_x, 35
 - stokes_problem, 54
 - transport_problem_x, 60
 - transport_problem_y, 64
 - transport_problem_z, 69
- mask_V
 - advection_diffusion_problem, 20
 - euler_problem, 25
 - navier_stokes_problem, 31
 - parabolic_problem_y, 39
 - stokes_problem, 55
 - transport_problem_x, 60
 - transport_problem_y, 65
 - transport_problem_z, 70
- mask_W
 - advection_diffusion_problem, 20
 - euler_problem, 25
 - navier_stokes_problem, 31
 - parabolic_problem_z, 43
 - stokes_problem, 55
 - transport_problem_x, 60
 - transport_problem_y, 65
 - transport_problem_z, 70
- monitor_convergence
 - problem_setting, 12
- navier_stokes_problem, 26
 - ~navier_stokes_problem, 29
 - assemble_magnitude, 29
 - compute_magnitude, 29
 - dmGrid_cent_rich, 30
 - dmGrid_centered, 30
 - dmGrid_shift_transp, 30
 - dmGrid_stag_transp, 30
 - dmGrid_staggered_x, 30
 - dmGrid_staggered_y, 30
 - dmGrid_staggered_z, 31
 - exodus, 30
 - Magnitude, 31
 - mask_U, 31
 - mask_V, 31
 - mask_W, 31
 - navier_stokes_problem, 29
 - P, 31
 - P_x, 31
 - P_y, 31
 - P_z, 31
 - solve, 30
 - U_up, 31
 - update_velocity, 30
 - V_up, 32
 - W_up, 32
- nx
 - problem_setting, 13
- ny
 - problem_setting, 13
- nz
 - problem_setting, 13
- P
 - euler_problem, 25
 - navier_stokes_problem, 31
 - poisson_problem, 50
 - stokes_problem, 55
- P_x
 - euler_problem, 25
 - navier_stokes_problem, 31
 - poisson_problem, 50
 - stokes_problem, 55
- P_y
 - euler_problem, 25
 - navier_stokes_problem, 31
 - poisson_problem, 50
 - stokes_problem, 55
- P_z
 - euler_problem, 25
 - navier_stokes_problem, 31
 - poisson_problem, 50
 - stokes_problem, 55
- parabolic.hpp
 - PARABOLIC_PROBLEM_Y_HPP, 79
 - PARABOLIC_PROBLEM_Z_HPP, 79

- parabolic_problem_x, [32](#)
 - ~parabolic_problem_x, [34](#)
 - A, [35](#)
 - assemble_lhs, [34](#)
 - assemble_rhs, [34](#)
 - dmGrid, [35](#)
 - exodus, [34](#)
 - get_U, [34](#)
 - mask_U, [35](#)
 - parabolic_problem_x, [33](#), [34](#)
 - rhs, [35](#)
 - set_U, [34](#)
 - solve, [34](#)
 - solve_step, [35](#)
 - U_up, [35](#)
- parabolic_problem_y, [35](#)
 - ~parabolic_problem_y, [37](#)
 - A, [38](#)
 - assemble_lhs, [38](#)
 - assemble_rhs, [38](#)
 - dmGrid, [38](#)
 - exodus, [38](#)
 - get_V, [38](#)
 - mask_V, [39](#)
 - parabolic_problem_y, [37](#)
 - rhs, [39](#)
 - set_V, [38](#)
 - solve, [38](#)
 - solve_step, [38](#)
 - V_up, [39](#)
- PARABOLIC_PROBLEM_Y_HPP
 - parabolic.hpp, [79](#)
- parabolic_problem_z, [39](#)
 - ~parabolic_problem_z, [41](#)
 - A, [42](#)
 - assemble_lhs, [42](#)
 - assemble_rhs, [42](#)
 - dmGrid, [42](#)
 - exodus, [42](#)
 - get_W, [42](#)
 - mask_W, [43](#)
 - parabolic_problem_z, [41](#)
 - rhs, [43](#)
 - set_W, [42](#)
 - solve, [42](#)
 - solve_step, [42](#)
 - W_up, [43](#)
- PARABOLIC_PROBLEM_Z_HPP
 - parabolic.hpp, [79](#)
- pi
 - macros.hpp, [77](#)
- poisson_problem, [43](#)
 - ~poisson_problem, [46](#)
 - A, [49](#)
 - assemble_divergence, [46](#)
 - assemble_lhs, [47](#)
 - compute_divergence, [47](#)
 - derive_x_P, [47](#)
 - derive_y_P, [47](#)
 - derive_z_P, [48](#)
 - dmGrid_cent_rich, [49](#)
 - dmGrid_centered, [49](#)
 - dmGrid_staggered_x, [49](#)
 - dmGrid_staggered_y, [49](#)
 - dmGrid_staggered_z, [50](#)
 - exodus, [48](#)
 - manage_pressure, [48](#)
 - manage_pressure_x, [48](#)
 - manage_pressure_y, [49](#)
 - manage_pressure_z, [49](#)
 - P, [50](#)
 - P_x, [50](#)
 - P_y, [50](#)
 - P_z, [50](#)
 - poisson_problem, [46](#)
 - U_up, [50](#)
 - V_up, [50](#)
 - W_up, [50](#)
- pRef
 - problem_setting, [10](#)
- PrintSimulationParameters
 - utils.cpp, [97](#)
 - utils.hpp, [87](#)
- problem_setting, [9](#)
 - a, [11](#)
 - base_path, [11](#)
 - brinkman, [11](#)
 - check_convergence, [12](#)
 - d, [12](#)
 - dt, [12](#)
 - iter, [12](#)
 - Lx, [12](#)
 - Lx_0, [12](#)
 - Ly, [12](#)
 - Ly_0, [12](#)
 - Lz, [12](#)
 - Lz_0, [12](#)
 - monitor_convergence, [12](#)
 - nx, [13](#)
 - ny, [13](#)
 - nz, [13](#)
 - pRef, [10](#)
 - problem_type, [13](#)
 - Re, [13](#)
 - theta, [13](#)
 - uxRef, [10](#)
 - uyRef, [10](#)
 - uzRef, [11](#)
- problem_type
 - problem_setting, [13](#)
- rayIntersectsTriangle
 - utils.cpp, [97](#)
 - utils.hpp, [87](#)
- Re
 - problem_setting, [13](#)
- reader

- utils.cpp, 97
- utils.hpp, 88
- README.md, 89
- rhs
 - parabolic_problem_x, 35
 - parabolic_problem_y, 39
 - parabolic_problem_z, 43
- RIGHT
 - macros.hpp, 77
- SecondDerive_x
 - transport_problem_y, 63
- SecondDerive_z
 - transport_problem_y, 63
- SecondShiftV_z
 - transport_problem_y, 64
 - transport_problem_z, 68
- SecondShiftW_z
 - transport_problem_y, 64
 - transport_problem_z, 69
- set_U
 - parabolic_problem_x, 34
- set_V
 - parabolic_problem_y, 38
- set_W
 - parabolic_problem_z, 42
- solve
 - advection_diffusion_problem, 18
 - euler_problem, 24
 - navier_stokes_problem, 30
 - parabolic_problem_x, 34
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
 - stokes_problem, 54
- solve_step
 - parabolic_problem_x, 35
 - parabolic_problem_y, 38
 - parabolic_problem_z, 42
- solve_step_x
 - transport_problem_x, 59
- solve_step_y
 - transport_problem_y, 64
- solve_step_z
 - transport_problem_z, 69
- src/advection_diffusion.cpp, 89
- src/inviscid_euler.cpp, 89
- src/main.cpp, 89
- src/navier_stokes.cpp, 91
- src/parabolic.cpp, 91
- src/poisson.cpp, 92
- src/stokes.cpp, 92
- src/transport.cpp, 93
- src/utils.cpp, 93
- stokes_problem, 51
 - ~stokes_problem, 53
 - assemble_magnitude, 53
 - compute_magnitude, 53
 - dmGrid_cent_rich, 54
 - dmGrid_centered, 54
 - dmGrid_staggered_x, 54
 - dmGrid_staggered_y, 54
 - dmGrid_staggered_z, 54
 - exodus, 53
 - Magnitude, 54
 - mask_U, 54
 - mask_V, 55
 - mask_W, 55
 - P, 55
 - P_x, 55
 - P_y, 55
 - P_z, 55
 - solve, 54
 - stokes_problem, 53
 - U_up, 55
 - update_velocity, 54
 - V_up, 55
 - W_up, 55
- theta
 - problem_setting, 13
- ThirdDerive_x
 - transport_problem_z, 69
- ThirdDerive_y
 - transport_problem_z, 69
- transport.hpp
 - TRANSPORT_PROBLEM_Y_HPP, 82
 - TRANSPORT_PROBLEM_Z_HPP, 82
- transport_problem_x, 56
 - ~transport_problem_x, 58
 - CenterU, 58
 - Derive_x, 58
 - dmGrid_Centered, 59
 - dmGrid_Shifted, 59
 - dmGrid_Staggered, 60
 - FirstDerive_y, 58
 - FirstDerive_z, 58
 - FirstShiftU_y, 58
 - FirstShiftU_z, 59
 - FirstShiftV_y, 59
 - FirstShiftW_z, 59
 - mask_U, 60
 - mask_V, 60
 - mask_W, 60
 - solve_step_x, 59
 - transport_problem_x, 57, 58
 - U_n, 60
 - V_n, 60
 - W_n, 60
- transport_problem_y, 60
 - ~transport_problem_y, 63
 - CenterV, 63
 - Derive_y, 63
 - dmGrid_Centered, 64
 - dmGrid_Shifted, 64
 - dmGrid_Staggered, 64
 - FirstShiftU_y, 63
 - FirstShiftV_y, 63
 - mask_U, 64

- mask_V, 65
- mask_W, 65
- SecondDerive_x, 63
- SecondDerive_z, 63
- SecondShiftV_z, 64
- SecondShiftW_z, 64
- solve_step_y, 64
- transport_problem_y, 62
- U_n, 65
- V_n, 65
- W_n, 65
- TRANSPORT_PROBLEM_Y_HPP
 - transport.hpp, 82
- transport_problem_z, 65
 - ~transport_problem_z, 68
 - CenterW, 68
 - Derive_z, 68
 - dmGrid_Centered, 69
 - dmGrid_Shifted, 69
 - dmGrid_Staggered, 69
 - FirstShiftU_z, 68
 - FirstShiftW_z, 68
 - mask_U, 69
 - mask_V, 70
 - mask_W, 70
 - SecondShiftV_z, 68
 - SecondShiftW_z, 69
 - solve_step_z, 69
 - ThirdDerive_x, 69
 - ThirdDerive_y, 69
 - transport_problem_z, 67
 - U_n, 70
 - V_n, 70
 - W_n, 70
- TRANSPORT_PROBLEM_Z_HPP
 - transport.hpp, 82
- U_n
 - transport_problem_x, 60
 - transport_problem_y, 65
 - transport_problem_z, 70
- U_up
 - advection_diffusion_problem, 20
 - euler_problem, 25
 - navier_stokes_problem, 31
 - parabolic_problem_x, 35
 - poisson_problem, 50
 - stokes_problem, 55
- UP
 - macros.hpp, 77
- UP_LEFT
 - macros.hpp, 77
- UP_RIGHT
 - macros.hpp, 77
- update_velocity
 - advection_diffusion_problem, 19
 - euler_problem, 24
 - navier_stokes_problem, 30
 - stokes_problem, 54
- utils.cpp
 - CheckSolution, 94
 - CreateAnalyticalP, 94
 - CreateAnalyticalU, 94
 - CreateAnalyticalV, 95
 - CreateAnalyticalW, 95
 - CreateGrid, 95
 - createMaskU, 95
 - createMaskV, 96
 - createMaskW, 96
 - isPointInsideMesh, 96
 - PrintSimulationParameters, 97
 - rayIntersectsTriangle, 97
 - reader, 97
- utils.hpp
 - CheckSolution, 84
 - CreateAnalyticalP, 84
 - CreateAnalyticalU, 85
 - CreateAnalyticalV, 85
 - CreateAnalyticalW, 85
 - CreateGrid, 85
 - createMaskU, 86
 - createMaskV, 86
 - createMaskW, 86
 - D_x, 88
 - D_y, 88
 - D_z, 88
 - faces, 88
 - filename, 88
 - isPointInsideMesh, 87
 - PrintSimulationParameters, 87
 - rayIntersectsTriangle, 87
 - reader, 88
 - vertices, 88
- uxRef
 - problem_setting, 10
- uyRef
 - problem_setting, 10
- uzRef
 - problem_setting, 11
- V_n
 - transport_problem_x, 60
 - transport_problem_y, 65
 - transport_problem_z, 70
- V_up
 - advection_diffusion_problem, 20
 - euler_problem, 26
 - navier_stokes_problem, 32
 - parabolic_problem_y, 39
 - poisson_problem, 50
 - stokes_problem, 55
- vertices
 - utils.hpp, 88
- W_n
 - transport_problem_x, 60
 - transport_problem_y, 65
 - transport_problem_z, 70

W_up

- advection_diffusion_problem, [20](#)
- euler_problem, [26](#)
- navier_stokes_problem, [32](#)
- parabolic_problem_z, [43](#)
- poisson_problem, [50](#)
- stokes_problem, [55](#)