NSB Solver

Generated by Doxygen 1.9.1

Chapter 1

NSB Solver Documentation

Welcome to the NSB Solver project

The **NSB Solver** project is a high-performance C++ library dedicated to fluid simulation, particularly the solution of the **Navier-Stokes-Brinkman Equations**.

It is developed as part of the HPSC for Aerospace Engineering course and aims to provide an efficient and parallelizable solution for three-dimensional grid fluid dynamics problem.

1.0.1 Navigation

Use the menus above to explore the sections:

- Go to `Class List` to see all classes and structs.
- Check the `Files` for the complete list of headers.

Chapter 2

Deprecated List

Member Field::getData ()

- Use [idx] instead.

Member Field::getData () const

- Use [idx] instead.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 ConfigFuncs Namespace Reference

Functions

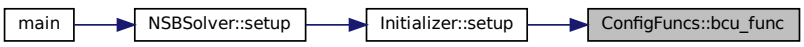
- double bcu_func (double x, double y, double z, double t)
- double bcv_func (double x, double y, double z, double t)
- double bcw_func (double x, double y, double z, double t)
- double fx_func (double x, double y, double z, double t)
- double fy_func (double x, double y, double z, double t)
- double fz_func (double x, double y, double z, double t)
- double u_init_func (double x, double y, double z, double t=0)
- double v_init_func (double x, double y, double z, double t=0)
- double w_init_func (double x, double y, double z, double t=0)
- double p_init_func (double x, double y, double z, double t=0)
- double k_func (double, double, double, double=0)

6.1.1 Function Documentation

6.1.1.1 bcu_func()

```
double ConfigFuncs::bcu_func (
            double x,
            double y,
            double z,
            double t )  [inline]
```

Here is the caller graph for this function:

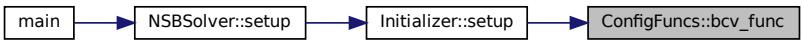


6.1.1.2 bcv_func()

```
double ConfigFuncs::bcv_func (
            double x,
            double y,
```

```
            double z,
            double t )  [inline]
```

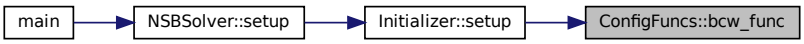
Here is the caller graph for this function:



6.1.1.3 bcw_func()

```
double ConfigFuncs::bcw_func (
            double x,
            double y,
            double z,
            double t )  [inline]
```

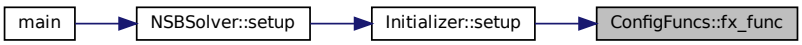
Here is the caller graph for this function:



6.1.1.4 fx_func()

```
double ConfigFuncs::fx_func (
            double x,
            double y,
            double z,
            double t )  [inline]
```

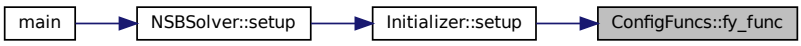
Here is the caller graph for this function:



6.1.1.5 fy_func()

```
double ConfigFuncs::fy_func (
            double x,
            double y,
            double z,
            double t )  [inline]
```

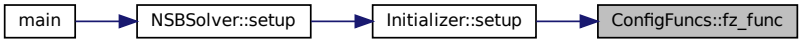
Here is the caller graph for this function:



6.1.1.6 fz_func()

```
double ConfigFuncs::fz_func (
            double x,
            double y,
            double z,
            double t )  [inline]
```

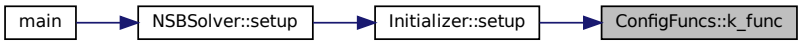
Here is the caller graph for this function:



6.1.1.7 k_func()

```
double ConfigFuncs::k_func (
            double ,
            double ,
            double ,
            double  = 0 )  [inline]
```

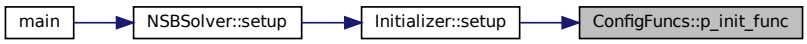
Here is the caller graph for this function:



6.1.1.8 p_init_func()

```
double ConfigFuncs::p_init_func (
            double x,
            double y,
            double z,
            double t = 0 )  [inline]
```

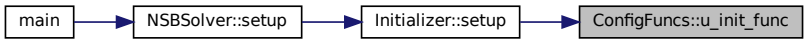
Here is the caller graph for this function:



6.1.1.9 u_init_func()

```
double ConfigFuncs::u_init_func (
            double x,
            double y,
            double z,
            double t = 0 )  [inline]
```

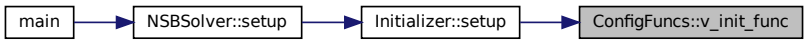
Here is the caller graph for this function:



6.1.1.10 v_init_func()

```
double ConfigFuncs::v_init_func (
            double x,
            double y,
            double z,
            double t = 0 )  [inline]
```

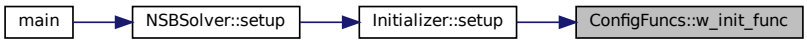
Here is the caller graph for this function:



6.1.1.11 w_init_func()

```
double ConfigFuncs::w_init_func (
            double x,
            double y,
            double z,
            double t = 0 )  [inline]
```

Here is the caller graph for this function:



6.2 displayError Namespace Reference

Functions

- def u_ana (points, t, h)
- def v_ana (points, t, h)
- def w_ana (points, t, h)
- def p_ana (points, t)
- def get_step_from_filename (filename)
- def main ()

Variables

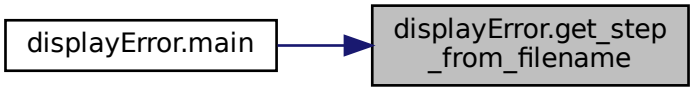
- string EXECUTABLE = "../build/main"
- string CONFIG_FILE = "../data/config.json"
- string OUTPUT_DIR = "../output/"
- ERROR_DIR = os.path.join(OUTPUT_DIR, "error_analysis")
- int NX = 50
- float DT = 0.001
- float T_END = 0.1
- int OUTPUT_FREQ = 1
- float DOMAIN_LEN_X = 6.0

6.2.1 Function Documentation

6.2.1.1 get_step_from_filename()

```
def displayError.get_step_from_filename (
            filename )
```

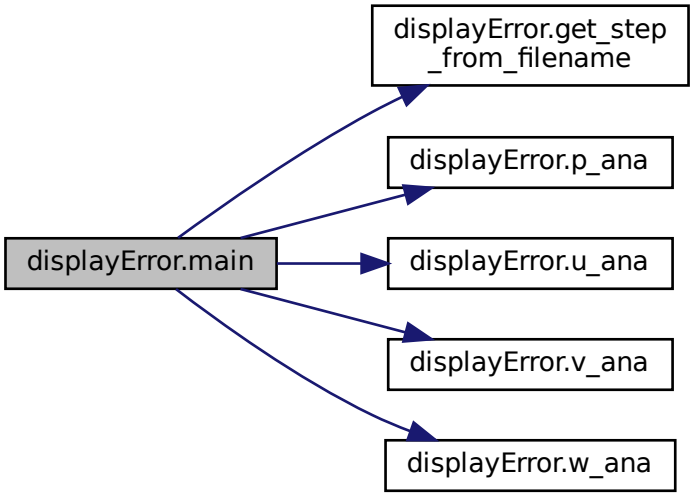
Here is the caller graph for this function:



6.2.1.2 main()

```
def displayError.main ( )
```

Here is the call graph for this function:



6.2.1.3 p_ana()

```
def displayError.p_ana (
              points,
              t )
```

Here is the caller graph for this function:

6.2.1.4 u_ana()

```
def displayError.u_ana (
              points,
              t,
              h )
```

Here is the caller graph for this function:

6.2.1.5 v_ana()

```
def displayError.v_ana (
              points,
              t,
              h )
```

Here is the caller graph for this function:

6.2.1.6 w_ana()

```
def displayError.w_ana (
              points,
              t,
              h )
```

Here is the caller graph for this function:

6.2.2 Variable Documentation

6.2.2.1 CONFIG_FILE

```
string displayError.CONFIG_FILE = "../data/config.json"
```

6.2.2.2 DOMAIN_LEN_X

```
float displayError.DOMAIN_LEN_X = 6.0
```

6.2.2.3 DT

```
float displayError.DT = 0.001
```

6.2.2.4 ERROR_DIR

```
displayError.ERROR_DIR = os.path.join(OUTPUT_DIR, "error_analysis")
```

6.2.2.5 EXECUTABLE

```
string displayError.EXECUTABLE = "../build/main"
```

6.2.2.6 NX

```
int displayError.NX = 50
```

6.2.2.7 OUTPUT_DIR

```
string displayError.OUTPUT_DIR = "../output/"
```

6.2.2.8 OUTPUT_FREQ

```
int displayError.OUTPUT_FREQ = 1
```

6.2.2.9 T_END

```
float displayError.T_END = 0.1
```

6.3 Initializer Namespace Reference

Functions

- SimulationData setup (const InputData &inputData)

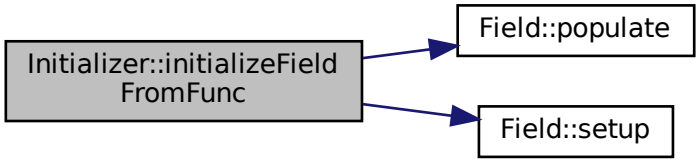
  Setup and initialize the SimulationData from input data.
- Field initializeFieldFromFunc (const double time, const GridPtr &grid, const Func &func)
- VectorField initializeVectorFieldFromFunc (const double time, const GridPtr &grid, const Func &func_u, const Func &func_v, const Func &func_w)

6.3.1 Function Documentation

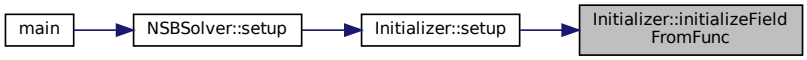
6.3.1.1 initializeFieldFromFunc()

```
Field Initializer::initializeFieldFromFunc (
            const double time,
            const GridPtr & grid,
            const Func & func )
```

Here is the call graph for this function:



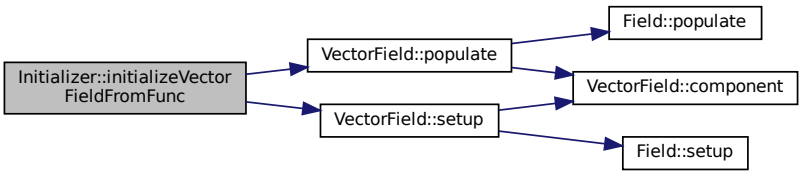
Here is the caller graph for this function:



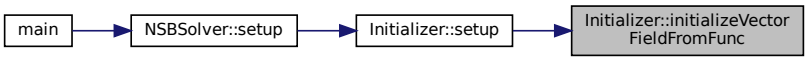
6.3.1.2 initializeVectorFieldFromFunc()

```
VectorField Initializer::initializeVectorFieldFromFunc (
            const double time,
            const GridPtr & grid,
            const Func & func_u,
            const Func & func_v,
            const Func & func_w )
```

Here is the call graph for this function:



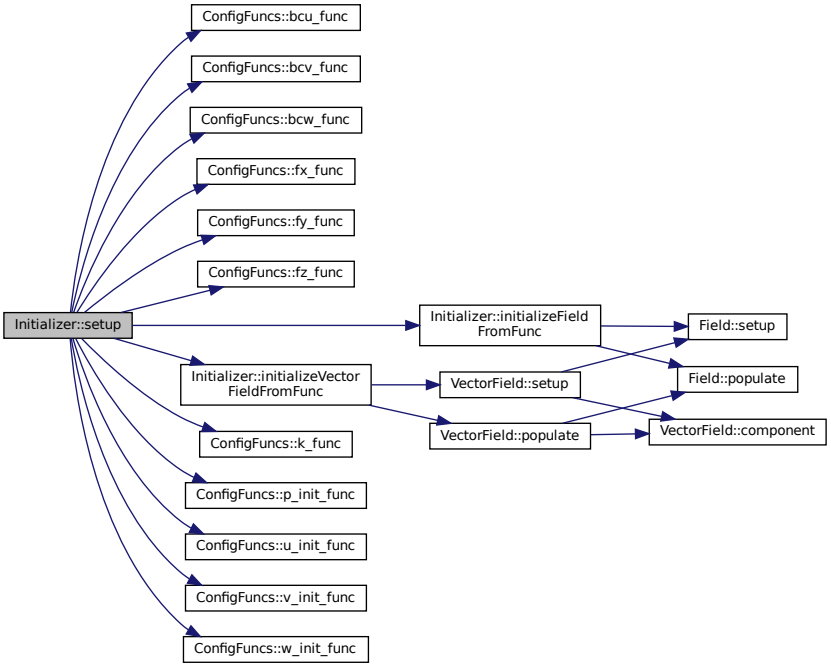
Here is the caller graph for this function:



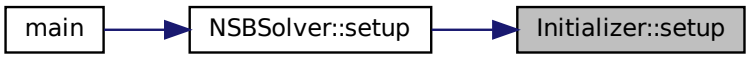
6.3.1.3 setup()

```
SimulationData Initializer::setup (
            const InputData & inputData )
```

Setup and initialize the SimulationData from input data.
Here is the call graph for this function:



Here is the caller graph for this function:



6.4 InputReader Namespace Reference

Functions

- InputData read (const std::string &filename)

  Read and parse input data from a JSON configuration file.

6.4.1 Function Documentation

6.4.1.1 read()

```
InputData InputReader::read (
            const std::string & filename )
```

Read and parse input data from a JSON configuration file.

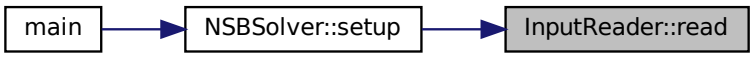
Parameters

filename	path to the configuration file

Returns

InputData structure containing all parsed data

Here is the caller graph for this function:



6.5 plot_convergence Namespace Reference

Functions

- int get_step_from_filename (str filepath)
- np.ndarray compute_analytical_u_at_x_faces (np.ndarray points, float t, float h)
- np.ndarray compute_analytical_v_at_y_faces (np.ndarray points, float t, float h)
- np.ndarray compute_analytical_w_at_z_faces (np.ndarray points, float t, float h)
- np.ndarray compute_analytical_p_at_centers (np.ndarray points, float t)
- float calculate_l2_rms (np.ndarray data)
- float calculate_l2_rms_error (np.ndarray field_numerical, np.ndarray field_analytical)
- def plot_convergence (list h_values, dict error_data, str title, str save_filename)
- def run_analysis (custom_simulations=None)

Variables

- list SIMULATIONS
- int DOMAIN_LENGTH_X = 6
- float DT = 0.001
- string VELOCITY_FIELD_NAME = "velocity"
- string PRESSURE_FIELD_NAME = "pressure"

6.5.1 Detailed Description

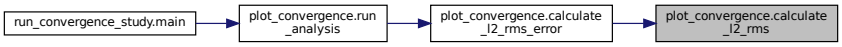
```
This script performs a spatial convergence analysis for a Method of
Manufactured Solutions (MMS) test.

It reads multiple VTK files, each from a simulation with a different
grid resolution. For each file, it computes the L2 RMS error
of velocity components (u, v, w) and pressure (p) against a
known analytical solution.

This version is adapted for staggered grid solvers where the VTK output
is co-located (e.g., at cell centers). It compares the numerical
co-located data against the analytical solution evaluated at the
appropriate staggered locations (e.g., cell faces).

Script modified from a previous version to:
1. Use only L2 RMS (absolute) errors.
2. Add O(h) and O(h^2) reference lines.
3. Translate all comments and outputs to English.
4. Remove all divergence calculations.
5. Save plots to high-definition image files instead of showing them.
```

6.5.2 Function Documentation

6.5.2.1 calculate_l2_rms()

```
float plot_convergence.calculate_l2_rms (
            np.ndarray data )

Calculates the L2 Root Mean Square (RMS) value of a data field.
This works for both scalar (N,) and vector (N, 3) fields.
```

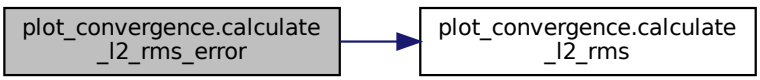
Here is the caller graph for this function:



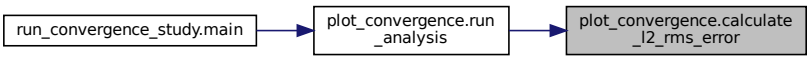
6.5.2.2 calculate_l2_rms_error()

```
float plot_convergence.calculate_l2_rms_error (
            np.ndarray field_numerical,
            np.ndarray field_analytical )

Calculates the L2 RMS of the error (field_numerical - field_analytical).
This is now an absolute error (the RMS norm of the error field).
```

Here is the call graph for this function:



Here is the caller graph for this function:

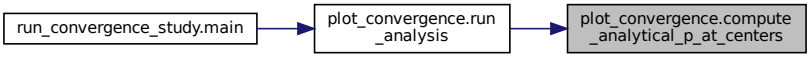


6.5.2.3 compute_analytical_p_at_centers()

```
np.ndarray plot_convergence.compute_analytical_p_at_centers (
            np.ndarray points,
            float t )
```

```
Computes analytical pressure.
Assumes 'points' are cell centers (i, j, k) and evaluates p there.
```

Here is the caller graph for this function:

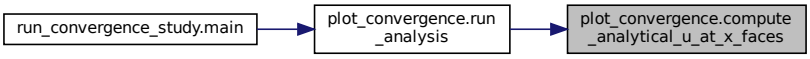


6.5.2.4 compute_analytical_u_at_x_faces()

```
np.ndarray plot_convergence.compute_analytical_u_at_x_faces (
            np.ndarray points,
            float t,
            float h )
```

```
Computes analytical u-velocity.
Assumes 'points' are cell centers (i, j, k).
Evaluates 'u' at the x-face center (i+0.5, j, k)
by shifting x coordinates by +h/2.
```

Here is the caller graph for this function:

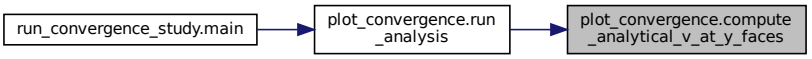


6.5.2.5 compute_analytical_v_at_y_faces()

```
np.ndarray plot_convergence.compute_analytical_v_at_y_faces (
            np.ndarray points,
            float t,
            float h )
```

```
Computes analytical v-velocity.
Assumes 'points' are cell centers (i, j, k).
Evaluates 'v' at the y-face center (i, j+0.5, k)
by shifting y coordinates by +h/2.
```

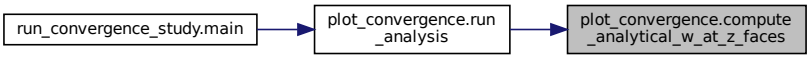
Here is the caller graph for this function:



6.5.2.6 compute_analytical_w_at_z_faces()

```
np.ndarray plot_convergence.compute_analytical_w_at_z_faces (
            np.ndarray points,
            float t,
            float h )

Computes analytical w-velocity.
Assumes 'points' are cell centers (i, j, k).
Evaluates 'w' at the z-face center (i, j, k+0.5)
by shifting z coordinates by +h/2.
```

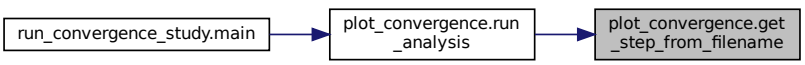
Here is the caller graph for this function:



6.5.2.7 get_step_from_filename()

```
int plot_convergence.get_step_from_filename (
            str filepath )

Parses a filename (e.g., "sim_0319.vtk") and extracts
the step number (319).
```

Here is the caller graph for this function:



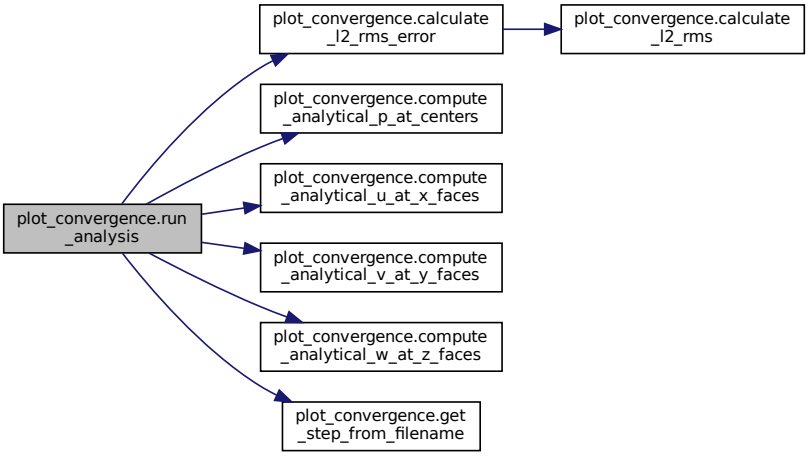
6.5.2.8 plot_convergence()

```
def plot_convergence.plot_convergence (
            list h_values,
            dict error_data,
            str title,
            str save_filename )
```

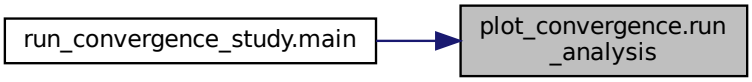
6.5.2.9 run_analysis()

```
def plot_convergence.run_analysis (
             custom_simulations = None )

Main function to loop through simulations, compute errors, and plot.
If custom_simulations is provided, it uses that list instead of the global SIMULATIONS.
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3 Variable Documentation

6.5.3.1 DOMAIN_LENGTH_X

```
int plot_convergence.DOMAIN_LENGTH_X = 6
```

6.5.3.2 DT

```
float plot_convergence.DT = 0.001
```

6.5.3.3 PRESSURE_FIELD_NAME

```
string plot_convergence.PRESSURE_FIELD_NAME = "pressure"
```

6.5.3.4 SIMULATIONS

```
list plot_convergence.SIMULATIONS
```

Initial value:

```
1 =  [
2 {"nx": 30, "file": "../output/n30simulation_output_0030.vtk"},
3 {"nx": 20, "file": "../output/n20simulation_output_0030.vtk"},
4 # {"nx": 40, "file": "../output/n40simulation_output_0030.vtk"},
5 # {"nx": 45, "file": "../output/n45simulation_output_0030.vtk"},
6 # {"nx": 35, "file": "../output/n35simulation_output_0030.vtk"},
7 # {"nx": 60, "file": "../output/n60simulation_output_0030.vtk"},
8 # {"nx": 50, "file": "../output/n50simulation_output_0030.vtk"},
9 # {"nx": 80, "file": "../output/n80simulation_output_0030.vtk"},
10 ]
```

6.5.3.5 VELOCITY_FIELD_NAME

```
string plot_convergence.VELOCITY_FIELD_NAME = "velocity"
```

6.6 plot_convergence_new Namespace Reference

Functions

- int get_step_from_filename (str filepath)
- np.ndarray compute_analytical_u_at_x_faces (np.ndarray points, float t, float h)
- np.ndarray compute_analytical_v_at_y_faces (np.ndarray points, float t, float h)
- np.ndarray compute_analytical_w_at_z_faces (np.ndarray points, float t, float h)
- np.ndarray compute_analytical_p_at_centers (np.ndarray points, float t)
- float calculate_l2_rms (np.ndarray data)
- float calculate_l2_rms_error (np.ndarray field_numerical, np.ndarray field_analytical)
- def plot_convergence (list x_values, dict error_data, str title, str save_filename, str x_label, int expected_↩ order)
- def run_analysis (custom_simulations=None)

Variables

- list SIMULATIONS
- int DOMAIN_LENGTH_X = 6
- float DT = 0.001
- string VELOCITY_FIELD_NAME = "velocity"
- string PRESSURE_FIELD_NAME = "pressure"

6.6.1 Detailed Description

```
This script performs a spatial convergence analysis for a Method of
Manufactured Solutions (MMS) test.

It reads multiple VTK files, each from a simulation with a different
grid resolution. For each file, it computes the L2 RMS error
of velocity components (u, v, w) and pressure (p) against a
known analytical solution.
```

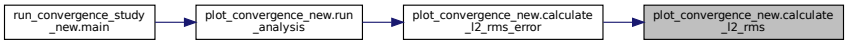
```
This version is adapted for staggered grid solvers where the VTK output
is co-located (e.g., at cell centers). It compares the numerical
co-located data against the analytical solution evaluated at the
appropriate staggered locations (e.g., cell faces).

Script modified from a previous version to:
1. Use only L2 RMS (absolute) errors.
2. Add O(h) and O(h^2) reference lines.
3. Translate all comments and outputs to English.
4. Remove all divergence calculations.
5. Save plots to high-definition image files instead of showing them.
```

6.6.2 Function Documentation

6.6.2.1 calculate_l2_rms()

```
float plot_convergence_new.calculate_l2_rms (
            np.ndarray data )
```

Here is the caller graph for this function:

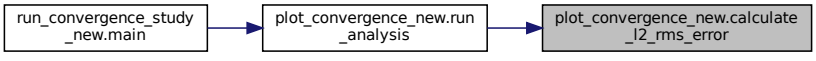


6.6.2.2 calculate_l2_rms_error()

```
float plot_convergence_new.calculate_l2_rms_error (
            np.ndarray field_numerical,
            np.ndarray field_analytical )
```

Here is the call graph for this function:

Here is the caller graph for this function:

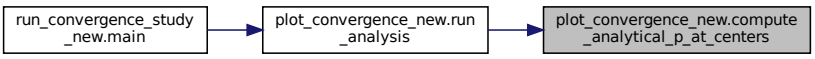


6.6.2.3 compute_analytical_p_at_centers()

```
np.ndarray plot_convergence_new.compute_analytical_p_at_centers (
            np.ndarray points,
```

```
            float t )
```

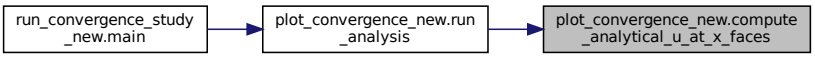
Here is the caller graph for this function:



6.6.2.4 compute_analytical_u_at_x_faces()

```
np.ndarray plot_convergence_new.compute_analytical_u_at_x_faces (
            np.ndarray points,
            float t,
            float h )
```

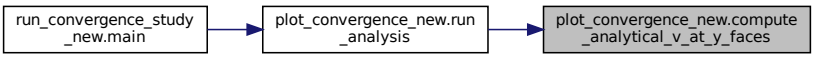
Here is the caller graph for this function:



6.6.2.5 compute_analytical_v_at_y_faces()

```
np.ndarray plot_convergence_new.compute_analytical_v_at_y_faces (
            np.ndarray points,
            float t,
            float h )
```

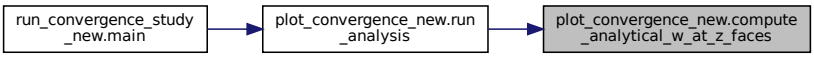
Here is the caller graph for this function:



6.6.2.6 compute_analytical_w_at_z_faces()

```
np.ndarray plot_convergence_new.compute_analytical_w_at_z_faces (
            np.ndarray points,
            float t,
            float h )
```

Here is the caller graph for this function:

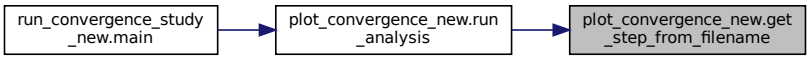


6.6.2.7 get_step_from_filename()

```
int plot_convergence_new.get_step_from_filename (
             str filepath )
```

```
[... Funzione get_step_from_filename non modificata ...]
```

Here is the caller graph for this function:



6.6.2.8 plot_convergence()

```
def plot_convergence_new.plot_convergence (
             list x_values,
             dict error_data,
             str title,
             str save_filename,
             str x_label,
             int expected_order )
```

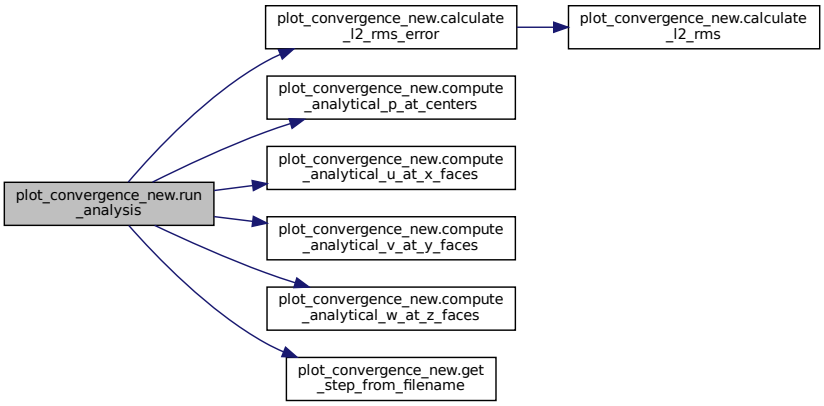
```
Plots the convergence data using generic x-axis values and a dynamic label/order.
```

6.6.2.9 run_analysis()

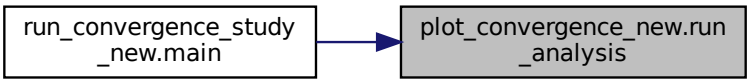
```
def plot_convergence_new.run_analysis (
              custom_simulations = None )
```

```
Main function to loop through simulations, compute errors, and plot.
Now supports different convergence modes.
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3 Variable Documentation

6.6.3.1 DOMAIN_LENGTH_X

```
int plot_convergence_new.DOMAIN_LENGTH_X = 6
```

6.6.3.2 DT

```
float plot_convergence_new.DT = 0.001
```

6.6.3.3 PRESSURE_FIELD_NAME

```
string plot_convergence_new.PRESSURE_FIELD_NAME = "pressure"
```

6.6.3.4 SIMULATIONS

```
list plot_convergence_new.SIMULATIONS
```

Initial value:

```
1 =  [
2 {"nx": 30, "file": "../output/n30simulation_output_0030.vtk"},
3 {"nx": 20, "file": "../output/n20simulation_output_0030.vtk"},
4 # ... (altre simulazioni rimosse)
5 ]
```

6.6.3.5 VELOCITY_FIELD_NAME

```
string plot_convergence_new.VELOCITY_FIELD_NAME = "velocity"
```

6.7 run_convergence_study Namespace Reference

Functions

- def main ()

Variables

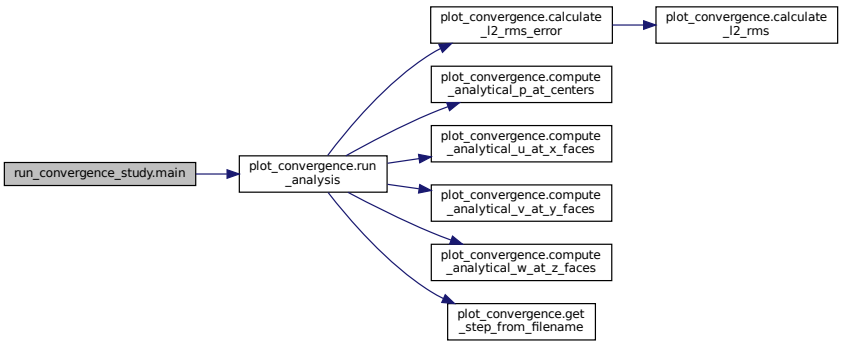
- string EXECUTABLE_PATH = "../build/main"
- string CONFIG_PATH = "../data/config.json"
- string OUTPUT_DIR = "../output/"
- int EXPECTED_LAST_STEP = 30
- list NX_VALUES = [20, 25, 30]

6.7.1 Function Documentation

6.7.1.1 main()

```
def run_convergence_study.main ( )
```

Here is the call graph for this function:



6.7.2 Variable Documentation

6.7.2.1 CONFIG_PATH

```
string run_convergence_study.CONFIG_PATH = "../data/config.json"
```

6.7.2.2 EXECUTABLE_PATH

```
string run_convergence_study.EXECUTABLE_PATH = "../build/main"
```

6.7.2.3 EXPECTED_LAST_STEP

```
int run_convergence_study.EXPECTED_LAST_STEP = 30
```

6.7.2.4 NX_VALUES

```
list run_convergence_study.NX_VALUES = [20, 25, 30]
```

6.7.2.5 OUTPUT_DIR

```
string run_convergence_study.OUTPUT_DIR = "../output/"
```

6.8 run_convergence_study_new Namespace Reference

Functions

- def calculate_params_for_mode (loop_val, base_dt, study_mode)
- def main ()

Variables

- string EXECUTABLE_PATH = "../build/main"
- string CONFIG_PATH = "../data/config.json"
- string OUTPUT_DIR = "../output/"
- string STUDY_MODE = "TEMPORAL_ONLY"
- int BASE_NX = 20
- float BASE_DT = 0.001
- float T_END = 0.03
- list NX_VALUES_FOR_SPATIAL_STUDY = [20, 25, 30, 35, 40, 45, 50, 60, 80]
- list DT_MULTIPLIERS_FOR_TEMPORAL_STUDY = [1.0, 0.5, 0.25, 0.125, 0.0625]
- int FIXED_NX_FOR_TEMPORAL_STUDY = 50

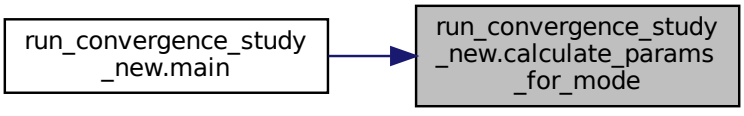
6.8.1 Function Documentation

6.8.1.1 calculate_params_for_mode()

```
def run_convergence_study_new.calculate_params_for_mode (
              loop_val,
              base_dt,
              study_mode )
```

```
Calculates the actual Nx, dt, and number of steps based on the study mode.
```

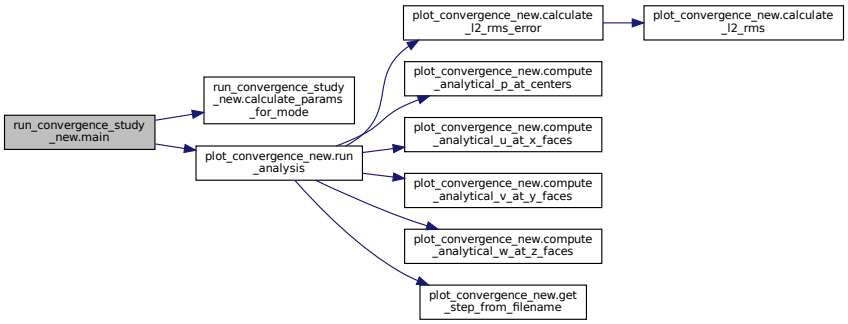
Here is the caller graph for this function:



6.8.1.2 main()

```
def run_convergence_study_new.main ( )
```

Here is the call graph for this function:



6.8.2 Variable Documentation

6.8.2.1 BASE_DT

```
float run_convergence_study_new.BASE_DT = 0.001
```

6.8.2.2 BASE_NX

```
int run_convergence_study_new.BASE_NX = 20
```

6.8.2.3 CONFIG_PATH

```
string run_convergence_study_new.CONFIG_PATH = "../data/config.json"
```

6.8.2.4 DT_MULTIPLIERS_FOR_TEMPORAL_STUDY

```
list run_convergence_study_new.DT_MULTIPLIERS_FOR_TEMPORAL_STUDY = [1.0, 0.5, 0.25, 0.125,
0.0625]
```

6.8.2.5 EXECUTABLE_PATH

```
string run_convergence_study_new.EXECUTABLE_PATH = "../build/main"
```

6.8.2.6 FIXED_NX_FOR_TEMPORAL_STUDY

```
int run_convergence_study_new.FIXED_NX_FOR_TEMPORAL_STUDY = 50
```

6.8.2.7 NX_VALUES_FOR_SPATIAL_STUDY

```
list run_convergence_study_new.NX_VALUES_FOR_SPATIAL_STUDY = [20, 25, 30, 35, 40, 45, 50, 60,
80]
```

6.8.2.8 OUTPUT_DIR

```
string run_convergence_study_new.OUTPUT_DIR = "../output/"
```

6.8.2.9 STUDY_MODE

```
string run_convergence_study_new.STUDY_MODE = "TEMPORAL_ONLY"
```

6.8.2.10 T_END

```
float run_convergence_study_new.T_END = 0.03
```

Chapter 7

Class Documentation

7.1 Derivatives Class Reference

Handler for computing spatial derivatives of scalar fields.
`#include <derivatives.hpp>`

Public Member Functions

- void computeGradient (const Field &field, VectorField &gradient) const

  Compute the gradient of a scalar field (forward difference).
- void computeDx_fwd (const Field &field, Field &dx) const

  Compute the derivative of a scalar field in the x-direction (forward difference).
- void computeDy_fwd (const Field &field, Field &dy) const

  Compute the derivative of a scalar field in the y-direction (forward difference).
- void computeDz_fwd (const Field &field, Field &dz) const

  Compute the derivative of a scalar field in the z-direction (forward difference).
- void computeDx_bwd (const Field &field, Field &dx) const

  Compute the derivative of a scalar field in the x-direction (backward difference).
- void computeDy_bwd (const Field &field, Field &dy) const

  Compute the derivative of a scalar field in the y-direction (backward difference).
- void computeDz_bwd (const Field &field, Field &dz) const

  Compute the derivative of a scalar field in the z-direction (backward difference).
- void computeDivergence (const VectorField &field, Field &divergence) const

  Compute the divergence of a vector field (backward difference).
- void computeDxx (const VectorField &field, VectorField &dxx) const

  Compute the second derivative of a vector field in the x-direction.
- void computeDyy (const VectorField &field, VectorField &dyy) const

  Compute the second derivative of a vector field in the y-direction.
- void computeDzz (const VectorField &field, VectorField &dzz) const

  Compute the second derivative of a vector field in the z-direction.
- void computeDxx (const Field &field, Field &dxx) const

  Compute the second derivative of a scalar field in the x-direction.
- void computeDyy (const Field &field, Field &dyy) const

  Compute the second derivative of a scalar field in the y-direction.
- void computeDzz (const Field &field, Field &dzz) const

  Compute the second derivative of a scalar field in the z-direction.
- double Dxx_local (const Field &f, size_t i, size_t j, size_t k) const

  Compute the local second derivative $^2 f/x^2$ *at a single grid point.*
- double Dyy_local (const Field &f, size_t i, size_t j, size_t k) const

Compute the local second derivative $\partial^2 f/\partial y^2$ at a single grid point.

- double Dzz_local (const Field &f, size_t i, size_t j, size_t k) const

  Compute the local second derivative $\partial^2 f/\partial z^2$ at a single grid point.

7.1.1 Detailed Description

Handler for computing spatial derivatives of scalar fields.

7.1.2 Member Function Documentation

7.1.2.1 computeDivergence()

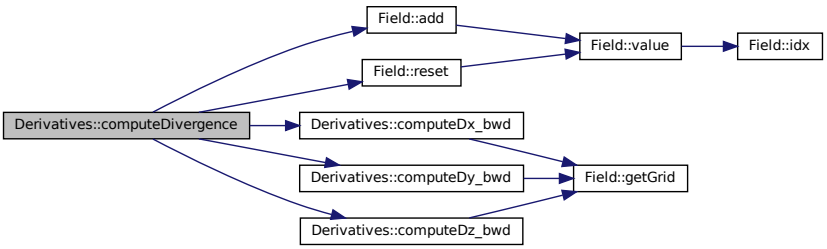
```
void Derivatives::computeDivergence (
            const VectorField & field,
            Field & divergence ) const
```

Compute the divergence of a vector field (backward difference).

Parameters

field	the input vector field
divergence	the output scalar field to store the divergence

Here is the call graph for this function:



Here is the caller graph for this function:

7.1.2.2 computeDx_bwd()

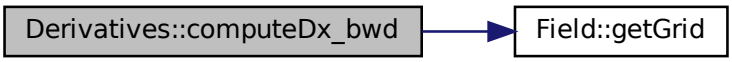
```
void Derivatives::computeDx_bwd (
            const Field & field,
            Field & dx ) const
```

Compute the derivative of a scalar field in the x-direction (backward difference).

Parameters

field	the input vector field
dx	the output field to store the derivative in x-direction

Here is the call graph for this function:



Here is the caller graph for this function:

7.1.2.3 computeDx_fwd()

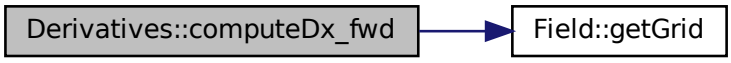
```
void Derivatives::computeDx_fwd (
            const Field & field,
            Field & dx ) const
```

Compute the derivative of a scalar field in the x-direction (forward difference).

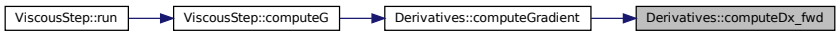
Parameters

field	the input scalar field
dx	the output field to store the derivative in x-direction

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.4 computeDxx() [1/2]

```
void Derivatives::computeDxx (
            const Field & field,
            Field & dxx ) const
```

Compute the second derivative of a scalar field in the x-direction.

Parameters

field	the input scalar field
dxx	the output field to store the second derivative in x-direction

Here is the call graph for this function:

7.1.2.5 computeDxx() [2/2]

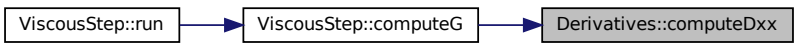
```
void Derivatives::computeDxx (
            const VectorField & field,
            VectorField & dxx ) const
```

Compute the second derivative of a vector field in the x-direction.

Parameters

field	the input vector field
dxx	the output vector field to store the second derivative in x-direction

Here is the caller graph for this function:



7.1.2.6 computeDy_bwd()

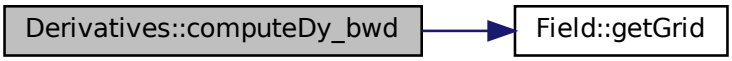
```
void Derivatives::computeDy_bwd (
            const Field & field,
            Field & dy ) const
```

Compute the derivative of a scalar field in the y-direction (backward difference).

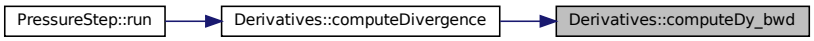
Parameters

field	the input vector field
dy	the output field to store the derivative in y-direction

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.7 computeDy_fwd()

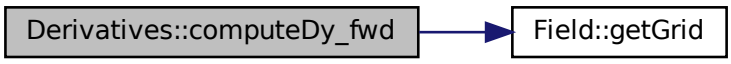
```
void Derivatives::computeDy_fwd (
            const Field & field,
            Field & dy ) const
```

Compute the derivative of a scalar field in the y-direction (forward difference).

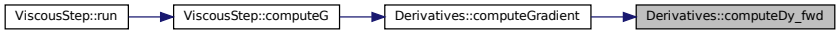
Parameters

field	the input scalar field
dy	the output field to store the derivative in y-direction

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.8 computeDyy() [1/2]

```
void Derivatives::computeDyy (
            const Field & field,
            Field & dyy ) const
```

Compute the second derivative of a scalar field in the y-direction.

Parameters

field	the input scalar field
dyy	the output field to store the second derivative in y-direction

Here is the call graph for this function:

7.1.2.9 computeDyy() [2/2]

```
void Derivatives::computeDyy (
            const VectorField & field,
            VectorField & dyy ) const
```

Compute the second derivative of a vector field in the y-direction.

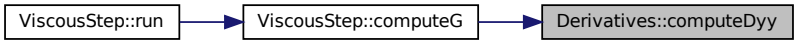
Parameters

field	the input vector field

Parameters

dyy	the output vector field to store the second derivative in y-direction

Here is the caller graph for this function:



7.1.2.10 computeDz_bwd()

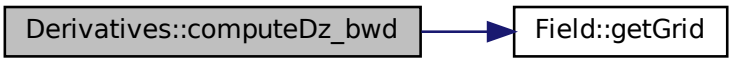
```
void Derivatives::computeDz_bwd (
            const Field & field,
            Field & dz ) const
```

Compute the derivative of a scalar field in the z-direction (backward difference).

Parameters

field	the input vector field
dz	the output field to store the derivative in z-direction

Here is the call graph for this function:



Here is the caller graph for this function:

7.1.2.11 computeDz_fwd()

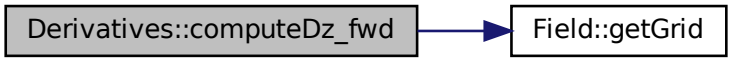
```
void Derivatives::computeDz_fwd (
            const Field & field,
            Field & dz ) const
```

Compute the derivative of a scalar field in the z-direction (forward difference).

Parameters

field	the input scalar field
dz	the output field to store the derivative in z-direction

Here is the call graph for this function:



Here is the caller graph for this function:

7.1.2.12 computeDzz() [1/2]

```
void Derivatives::computeDzz (
            const Field & field,
            Field & dzz ) const
```

Compute the second derivative of a scalar field in the z-direction.

Parameters

field	the input scalar field
dzz	the output field to store the second derivative in z-direction

Here is the call graph for this function:

7.1.2.13 computeDzz() [2/2]

```
void Derivatives::computeDzz (
            const VectorField & field,
```

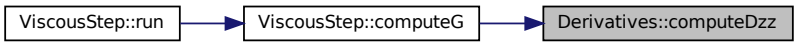
```
            VectorField & dzz ) const
```

Compute the second derivative of a vector field in the z-direction.

Parameters

field	the input vector field
dzz	the output vector field to store the second derivative in z-direction

Here is the caller graph for this function:



7.1.2.14 computeGradient()

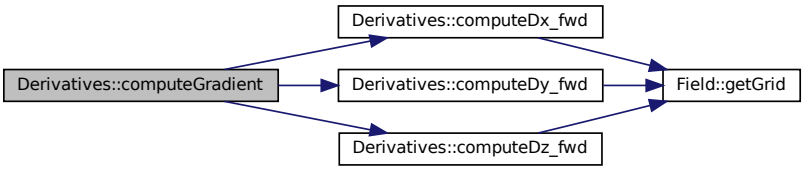
```
void Derivatives::computeGradient (
            const Field & field,
            VectorField & gradient ) const
```

Compute the gradient of a scalar field (forward difference).

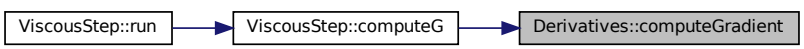
Parameters

field	the input scalar field
gradient	the output vector field to store the gradient

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.15 Dxx_local()

```
double Derivatives::Dxx_local (
            const Field & f,
            size_t i,
            size_t j,
            size_t k ) const
```

Compute the local second derivative $^2f/x^2$ at a single grid point.
Evaluates the second–order central finite difference at coordinates (i,j,k): (f(i+1,j,k) + f(i-1,j,k) 2 f(i,j,k)) / dx^2.
Performs minimal boundary checking: if `i` is on the domain boundary, returns 0.0.

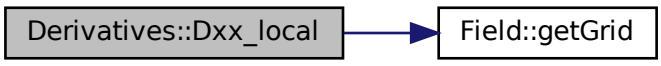
Parameters

f	The input scalar field.
i	Grid index in the x–direction.
j	Grid index in the y–direction.
k	Grid index in the z–direction.

Returns

The second derivative $^2f/x^2$ at (i,j,k).

Here is the call graph for this function:



7.1.2.16 Dyy_local()

```
double Derivatives::Dyy_local (
            const Field & f,
            size_t i,
            size_t j,
            size_t k ) const
```

Compute the local second derivative $^2f/y^2$ at a single grid point.
Evaluates the second–order central finite difference at coordinates (i,j,k): (f(i,j+1,k) + f(i,j-1,k) 2 f(i,j,k)) / dy^2.
Performs minimal boundary checking: if `j` is on the domain boundary, returns 0.0.

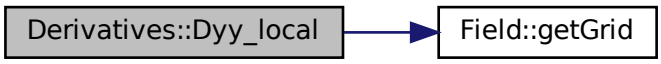
Parameters

f	The input scalar field.
i	Grid index in the x–direction.
j	Grid index in the y–direction.
k	Grid index in the z–direction.

Returns

The second derivative $^2f/y^2$ at (i,j,k).

Here is the call graph for this function:



7.1.2.17 Dzz_local()

```
double Derivatives::Dzz_local (
            const Field & f,
            size_t i,
            size_t j,
            size_t k ) const
```

Compute the local second derivative $^2f/z^2$ at a single grid point.
Evaluates the second–order central finite difference at coordinates (i,j,k): (f(i,j,k+1) + f(i,j,k-1) 2 f(i,j,k)) / dz^2.
Performs minimal boundary checking: if k is on the domain boundary, returns 0.0.

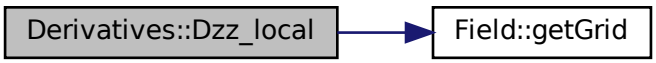
Parameters

f	The input scalar field.
i	Grid index in the x–direction.
j	Grid index in the y–direction.
k	Grid index in the z–direction.

Returns

The second derivative $^2f/z^2$ at (i,j,k).

Here is the call graph for this function:



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

- include/numerics/derivatives.hpp
- src/numerics/derivatives.cpp
- src/numerics/derivativesOpt.cpp

7.2 Field Class Reference

Class representing a scalar field defined on a 3D grid.
`#include <Fields.hpp>`

Public Types

- using Scalar = double

Public Member Functions

- std::vector< Field::Scalar > & getData ()
- const std::vector< Field::Scalar > & getData () const
- void setup (const GridPtr &grid, const Func &populateFunction=ZERO_FUNC, GridStaggering offset=GridStaggering::CELL_CE Axis offsetAxis=Axis::X)

  Setup the field with a function to populate it (e.g., initial condition).
- void populate (double time=0)

  Populates the underlying field based on the stored function in (x,y,z,t).
- const Grid & getGrid ()

  Getter for the grid information.
- const Grid & getGrid () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- size_t idx (size_t i, size_t j, size_t k) const

  Computes the linear index for 3D access in row-major order.
- Scalar & operator[ ] (size_t index)

  Access the value in the field at given linear index.
- const Scalar & operator[ ] (size_t index) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Scalar & operator() (size_t i, size_t j, size_t k)

  Access the value in the field at given position.
- const Scalar & operator() (size_t i, size_t j, size_t k) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Scalar & value (size_t i, size_t j, size_t k)

  Access the value in the field at given position.
- const Scalar & value (size_t i, size_t j, size_t k) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Scalar & valueWithOffset (size_t i, size_t j, size_t k, Axis offsetDirection, int offset)

  Access the value in the field at given position considering direction offset.
- const Scalar & valueWithOffset (size_t i, size_t j, size_t k, Axis offsetDirection, int offset) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void reset (Scalar value=Scalar(0))

  Reset the field values to a specified value (default is zero).
- void add (Scalar value)

  Add a scalar value to all elements in the field.
- void add (const Field &other)

  Add another field to this field element-wise.
- void multiply (Scalar value)

  Multiply all elements in the field by a scalar value.

Private Attributes

- GridPtr gridPtr_

  The pointer to the grid information.
- GridStaggering offset_

  The offset of the field in the staggered grid.
- Axis offsetAxis_

  The axis where to apply the offset in the staggered grid.
- std::vector< Scalar > data_

  Vector storing the field values in a flattened, row-major indexed 1D array.
- Func populateFunction_

  The function used for populating the field.

Friends

- class VectorField

7.2.1 Detailed Description

Class representing a scalar field defined on a 3D grid.

7.2.2 Member Typedef Documentation

7.2.2.1 Scalar

```
using Field::Scalar = double
```

7.2.3 Member Function Documentation

7.2.3.1 add() [1/2]

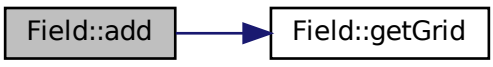
```
void Field::add (
            const Field & other )
```

Add another field to this field element-wise.

Parameters

other	the other field to add

Here is the call graph for this function:



7.2.3.2 add() [2/2]

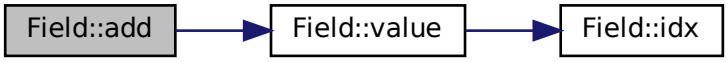
```
void Field::add (
            const Field::Scalar value )
```

Add a scalar value to all elements in the field.

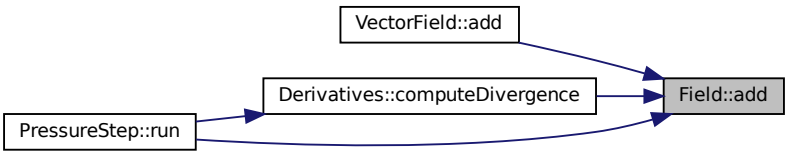
Parameters

value	the scalar value to add

Here is the call graph for this function:



Here is the caller graph for this function:

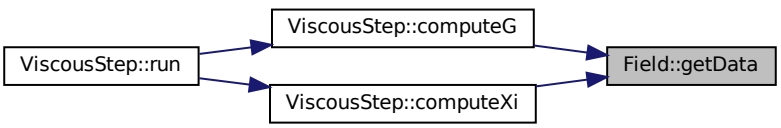


7.2.3.3 getData() [1/2]

```
std::vector<Field::Scalar>& Field::getData ( )  [inline]
```

Deprecated • Use [idx] instead.

Here is the caller graph for this function:



7.2.3.4 getData() [2/2]

```
const std::vector<Field::Scalar>& Field::getData ( ) const  [inline]
```

Deprecated • Use [idx] instead.

7.2.3.5 getGrid() [1/2]

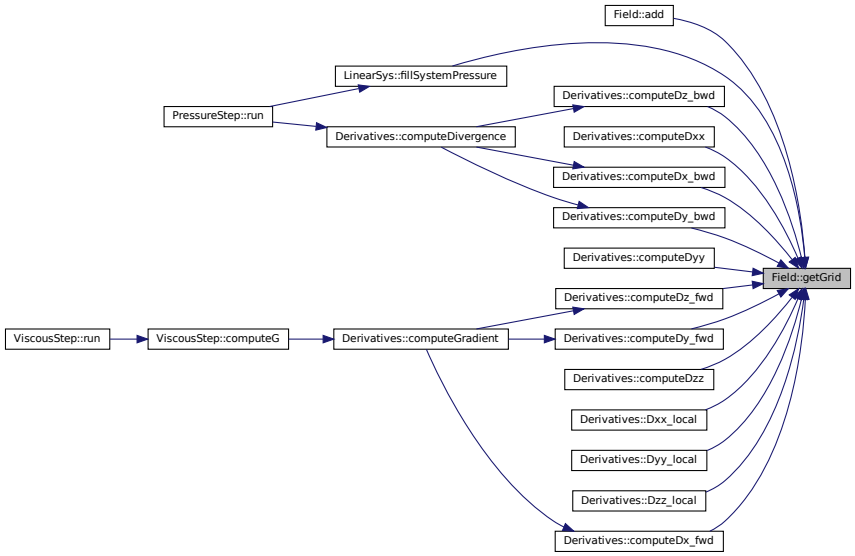
```
const Grid& Field::getGrid ( )  [inline]
```

Getter for the grid information.

Returns

the pointer to the grid information

Here is the caller graph for this function:



7.2.3.6 getGrid() [2/2]

```
const Grid& Field::getGrid ( ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.2.3.7 idx()

```
size_t Field::idx (
            size_t i,
            size_t j,
            size_t k ) const  [inline]
```

Computes the linear index for 3D access in row-major order.

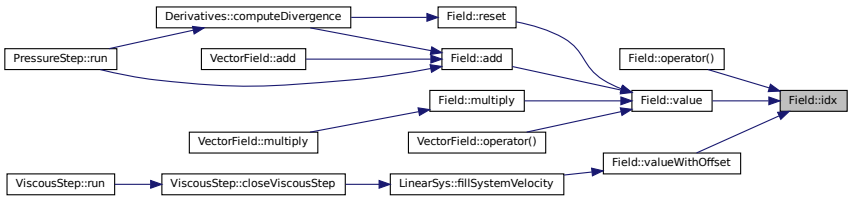
Parameters

i	the x-index
j	the y-index
k	the z-index

Returns

the corresponding 1D index

Here is the caller graph for this function:



7.2.3.8 multiply()

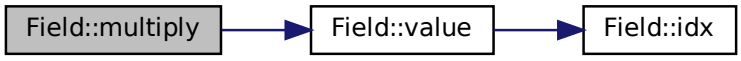
```
void Field::multiply (
            const Field::Scalar value )
```

Multiply all elements in the field by a scalar value.

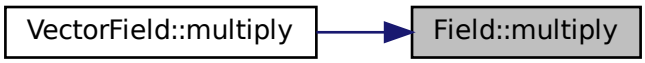
Parameters

value	the scalar value to multiply by

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.9 operator()() [1/2]

```
Scalar& Field::operator() (
            size_t i,
```

```
            size_t j,
            size_t k )  [inline]
```

Access the value in the field at given position.

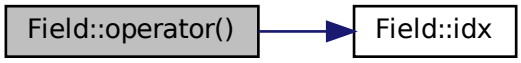
Parameters

i	the x-index
j	the y-index
k	the z-index

Returns

the value in the field at position (i,j,k)

Here is the call graph for this function:

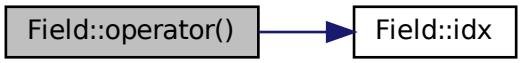


7.2.3.10 operator()() [2/2]

```
const Scalar& Field::operator() (
            size_t i,
            size_t j,
            size_t k ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Here is the call graph for this function:



7.2.3.11 operator[]() [1/2]

```
Scalar& Field::operator[] (
            size_t index )  [inline]
```

Access the value in the field at given linear index.

Parameters

index	the linear index

Returns

the value in the field at the given index

7.2.3.12 operator[]() [2/2]

```
const Scalar& Field::operator[] (
            size_t index ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.2.3.13 populate()

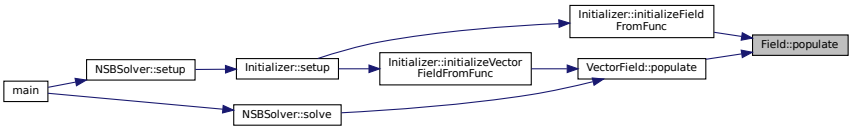
```
void Field::populate (
            double time = 0 )
```

Populates the underlying field based on the stored function in (x,y,z,t).

Parameters

time	the time affecting the stored function in (x,y,z,t)

Here is the caller graph for this function:



7.2.3.14 reset()

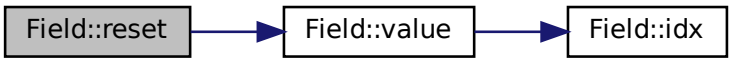
```
void Field::reset (
            const Field::Scalar value = Scalar(0) )
```

Reset the field values to a specified value (default is zero).

Parameters

value	the value to reset the field to (default is zero)

Here is the call graph for this function:



Here is the caller graph for this function:

7.2.3.15 setup()

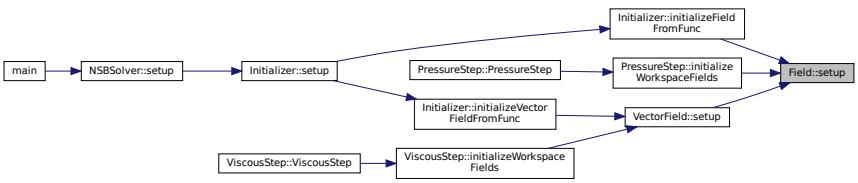
```
void Field::setup (
            const GridPtr & grid,
            const Func & populateFunction = ZERO_FUNC,
            GridStaggering offset = GridStaggering::CELL_CENTERED,
            Axis offsetAxis = Axis::X )  [inline]
```

Setup the field with a function to populate it (e.g., initial condition).

Parameters

grid	the pointer to the grid information
populateFunction	the function to use for populating the field
offset	the offset of the field in the staggered grid
offsetAxis	the axis where to apply the offset in the staggered grid

Here is the caller graph for this function:



7.2.3.16 value() [1/2]

```
Scalar& Field::value (
            size_t i,
```

```
            size_t j,
            size_t k )  [inline]
```

Access the value in the field at given position.

Parameters

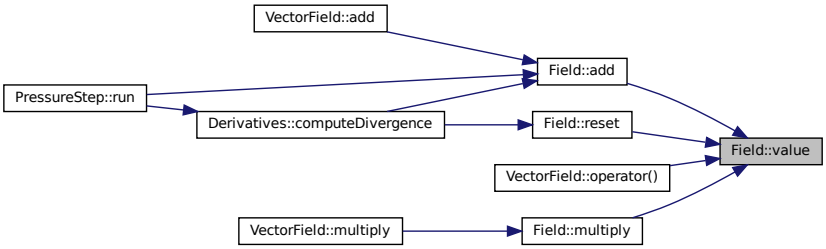
i	the x-index
j	the y-index
k	the z-index

Returns

the value in the field at position (i,j,k)

This method is a named alias for the access operator. Here is the call graph for this function:

Here is the caller graph for this function:



7.2.3.17 value() [2/2]

```
const Scalar& Field::value (
            size_t i,
            size_t j,
            size_t k ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Here is the call graph for this function:

7.2.3.18 valueWithOffset() [1/2]

```
Field::Scalar & Field::valueWithOffset (
            size_t i,
            size_t j,
            size_t k,
            Axis offsetDirection,
            int offset )
```

Access the value in the field at given position considering direction offset.

Parameters

i	the x-index
j	the y-index
k	the z-index
offsetDirection	the direction for the offset (Axis::X, Axis::Y, or Axis::Z)
offset	the offset value (can be positive or negative)

Returns

the value in the field at position (i,j,k)

Here is the call graph for this function:

Here is the caller graph for this function:

7.2.3.19 valueWithOffset() [2/2]

```
const Field::Scalar & Field::valueWithOffset (
            size_t i,
            size_t j,
            size_t k,
            Axis offsetDirection,
            int offset ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
Here is the call graph for this function:

7.2.4 Friends And Related Function Documentation

7.2.4.1 VectorField

```
friend class VectorField [friend]
```

7.2.5 Member Data Documentation

7.2.5.1 data_

```
std::vector<Scalar> Field::data_ [private]
```

Vector storing the field values in a flattened, row-major indexed 1D array.

7.2.5.2 gridPtr_

```
GridPtr Field::gridPtr_ [private]
```

The pointer to the grid information.

7.2.5.3 offset_

```
GridStaggering Field::offset_ [private]
```

The offset of the field in the staggered grid.

7.2.5.4 offsetAxis_

```
Axis Field::offsetAxis_ [private]
```

The axis where to apply the offset in the staggered grid.

7.2.5.5 populateFunction_

`Func Field::populateFunction_ [private]`
The function used for populating the field.
The documentation for this class was generated from the following files:

- include/core/Fields.hpp
- src/core/Fields.cpp

7.3 Grid Struct Reference

Struct to hold grid dimensions and spacing information along each direction.
`#include <Grid.hpp>`

Public Member Functions

- Grid (size_t Nx_, size_t Ny_, size_t Nz_, double dx_, double dy_, double dz_)

  Constructor to initialize the grid with given physical sizes and number of points.
- constexpr size_t size ()

  Getter for the total number of grid points.
- constexpr size_t size () const
- double to_x (double i, GridStaggering offset, Axis offsetAxis) const

  Converts a grid index to a physical coordinate along the X-axis, considering staggering.
- double to_y (double j, GridStaggering offset, Axis offsetAxis) const

  Converts a grid index to a physical coordinate along the Y-axis, considering staggering.
- double to_z (double k, GridStaggering offset, Axis offsetAxis) const

  Converts a grid index to a physical coordinate along the Z-axis, considering staggering.

Public Attributes

- double Lx

  Physical size of the grid in each direction.
- double Ly
- double Lz
- size_t Nx

  Number of grid points in each direction.
- size_t Ny
- size_t Nz
- double dx

  Grid spacing in each direction (derived from physical sizes and number of grid points).
- double dy
- double dz

7.3.1 Detailed Description

Struct to hold grid dimensions and spacing information along each direction.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Grid()

```
Grid::Grid (
            size_t Nx_,
            size_t Ny_,
            size_t Nz_,
            double dx_,
            double dy_,
            double dz_ )  [inline]
```

Constructor to initialize the grid with given physical sizes and number of points.

Parameters

*Nx_*	the number of grid points in the X direction
*Ny_*	the number of grid points in the Y direction
*Nz_*	the number of grid points in the Z direction
*dx_*	the grid spacing in the X direction
*dy_*	the grid spacing in the Y direction
*dz_*	the grid spacing in the Z direction

7.3.3 Member Function Documentation

7.3.3.1 size() [1/2]

```
constexpr size_t Grid::size ( )  [inline], [constexpr]
```

Getter for the total number of grid points.

Returns

the total number of grid points

7.3.3.2 size() [2/2]

```
constexpr size_t Grid::size ( ) const  [inline], [constexpr]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.3.3.3 to_x()

```
double Grid::to_x (
            double i,
            GridStaggering offset,
            Axis offsetAxis ) const  [inline]
```

Converts a grid index to a physical coordinate along the X-axis, considering staggering.

Parameters

i	the x-index
offset	the staggering offset
offsetAxis	the axis where to apply the offset

Returns

the physical x-coordinate

7.3.3.4 to_y()

```
double Grid::to_y (
            double j,
            GridStaggering offset,
            Axis offsetAxis ) const  [inline]
```

Converts a grid index to a physical coordinate along the Y-axis, considering staggering.

Parameters

j	the y-index
offset	the staggering offset
offsetAxis	the axis where to apply the offset

Returns

the physical y-coordinate

7.3.3.5 to_z()

```
double Grid::to_z (
            double k,
            GridStaggering offset,
            Axis offsetAxis ) const  [inline]
```

Converts a grid index to a physical coordinate along the Z-axis, considering staggering.

Parameters

k	the z-index
offset	the staggering offset
offsetAxis	the axis where to apply the offset

Returns

the physical z-coordinate

7.3.4 Member Data Documentation

7.3.4.1 dx

```
double Grid::dx
```

Grid spacing in each direction (derived from physical sizes and number of grid points).

7.3.4.2 dy

```
double Grid::dy
```

7.3.4.3 dz

```
double Grid::dz
```

7.3.4.4 Lx

```
double Grid::Lx
```
Physical size of the grid in each direction.

7.3.4.5 Ly

```
double Grid::Ly
```

7.3.4.6 Lz

```
double Grid::Lz
```

7.3.4.7 Nx

```
size_t Grid::Nx
```
Number of grid points in each direction.

7.3.4.8 Ny

```
size_t Grid::Ny
```

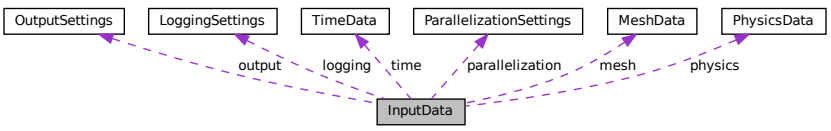
7.3.4.9 Nz

```
size_t Grid::Nz
```
The documentation for this struct was generated from the following file:

- include/core/Grid.hpp

7.4 InputData Struct Reference

Structure to hold all input data from configuration file.
`#include <inputReader.hpp>`
Collaboration diagram for InputData:



Public Attributes

- MeshData mesh
- PhysicsData physics
- TimeData time

- OutputSettings output
- LoggingSettings logging
- ParallelizationSettings parallelization

7.4.1 Detailed Description

Structure to hold all input data from configuration file.

7.4.2 Member Data Documentation

7.4.2.1 logging

```
LoggingSettings InputData::logging
```

7.4.2.2 mesh

```
MeshData InputData::mesh
```

7.4.2.3 output

```
OutputSettings InputData::output
```

7.4.2.4 parallelization

```
ParallelizationSettings InputData::parallelization
```

7.4.2.5 physics

```
PhysicsData InputData::physics
```

7.4.2.6 time

```
TimeData InputData::time
```

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

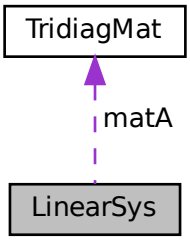
- include/io/inputReader.hpp

7.5 LinearSys Class Reference

Manages and solves a tridiagonal linear system (Ax = b) used for pressure and velocity steps.

```
#include <LinearSys.hpp>
```

Collaboration diagram for LinearSys:



Public Member Functions

- LinearSys (int n, BoundaryType boundaryType)

  Constructor.
- void setRhs (const std::vector< double > &newRhs)

  Sets the right-hand side (rhsC) vector for the solver.
- const std::vector< double > & getSolution () const

  Gets the solution vector (unknownX) computed by the solver.
- void fillSystemPressure (const Field &phi, const Axis direction)

  Fill the linear system for Pressure variables.
- void fillSystemVelocity (const SimulationData &simData, const VectorField &xi, const Axis fieldComponent, const Axis derivativeDirection, const size_t iStart, const size_t jStart, const size_t kStart)

  Fill the linear system for Velocity variables.
- const TridiagMat & getMatrix () const

  Getter to access the internal matrix (needed for Schur).
- const std::vector< double > & getRhs () const

  Getter to access the internal RHS (modified by fillSystem).
- void ThomaSolver ()

  Solve the linear system.

Private Attributes

- TridiagMat matA
- std::vector< double > rhsC
- std::vector< double > unknownX
- BoundaryType boundaryType

Friends

- class LinearSysTestFixture
- class SchurSolverTestFixture
- class SchurSequentialSolver

7.5.1 Detailed Description

Manages and solves a tridiagonal linear system (Ax = b) used for pressure and velocity steps.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 LinearSys()

```
LinearSys::LinearSys (
            int n,
            BoundaryType boundaryType )
```

Constructor.

Parameters

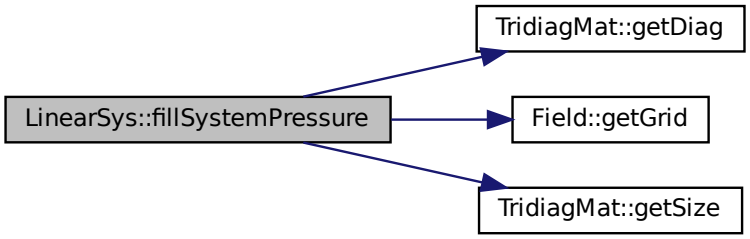
n	Size of the linear system (n x n)
boundaryType	Boundary condition type, Normal or Tangent (set Normal for pressure)

7.5.3 Member Function Documentation

7.5.3.1 fillSystemPressure()

```
void LinearSys::fillSystemPressure (
            const Field & phi,
            const Axis direction )
```

Fill the linear system for Pressure variables.
Here is the call graph for this function:

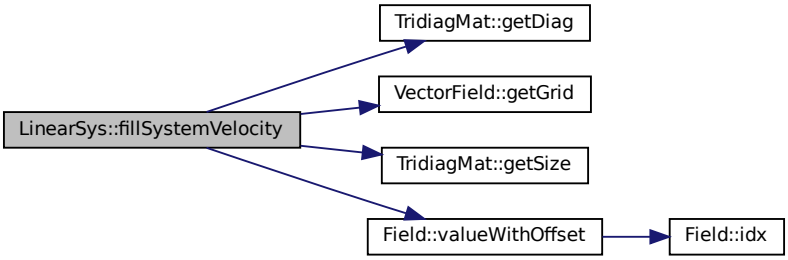


Here is the caller graph for this function:

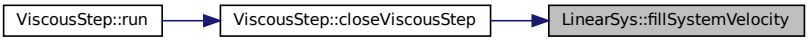
7.5.3.2 fillSystemVelocity()

```
void LinearSys::fillSystemVelocity (
            const SimulationData & simData,
            const VectorField & xi,
            const Axis fieldComponent,
            const Axis derivativeDirection,
            const size_t iStart,
            const size_t jStart,
            const size_t kStart )
```

Fill the linear system for Velocity variables.
Here is the call graph for this function:



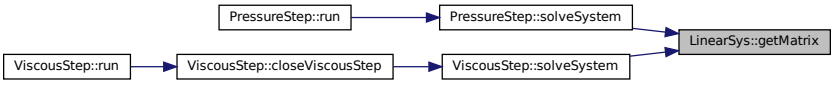
Here is the caller graph for this function:



7.5.3.3 getMatrix()

```
const TridiagMat& LinearSys::getMatrix ( ) const  [inline]
```

Getter to access the internal matrix (needed for Schur).
Here is the caller graph for this function:

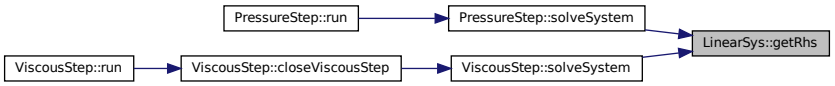


7.5.3.4 getRhs()

```
const std::vector<double>& LinearSys::getRhs ( ) const  [inline]
```

Getter to access the internal RHS (modified by fillSystem).

Here is the caller graph for this function:



7.5.3.5 getSolution()

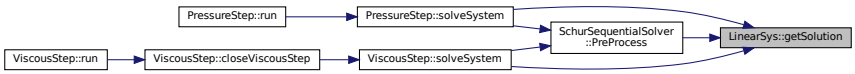
```
const std::vector< double > & LinearSys::getSolution ( ) const
```

Gets the solution vector (unknownX) computed by the solver.

Returns

A const reference to the solution vector.

Here is the caller graph for this function:



7.5.3.6 setRhs()

```
void LinearSys::setRhs (
            const std::vector< double > & newRhs )
```

Sets the right-hand side (rhsC) vector for the solver.

Parameters

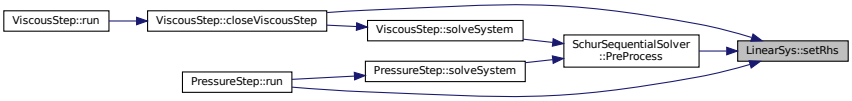
newRhs	The vector to be copied into the internal rhsC.

Exceptions

std::runtime_error	if newRhs size does not match system size.

Here is the call graph for this function:

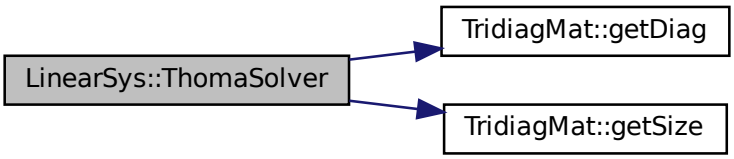
Here is the caller graph for this function:



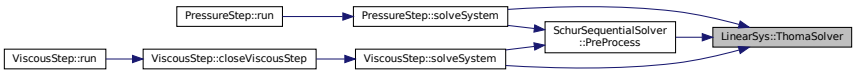
7.5.3.7 ThomaSolver()

```
void LinearSys::ThomaSolver ( )
```

Solve the linear system.
Here is the call graph for this function:



Here is the caller graph for this function:



7.5.4 Friends And Related Function Documentation

7.5.4.1 LinearSysTestFixture

```
friend class LinearSysTestFixture  [friend]
```

7.5.4.2 SchurSequentialSolver

```
friend class SchurSequentialSolver  [friend]
```

7.5.4.3 SchurSolverTestFixture

```
friend class SchurSolverTestFixture  [friend]
```

7.5.5 Member Data Documentation

7.5.5.1 boundaryType

```
BoundaryType LinearSys::boundaryType  [private]
```

7.5.5.2 matA

```
TridiagMat LinearSys::matA  [private]
```

7.5.5.3 rhsC

```
std::vector<double> LinearSys::rhsC  [private]
```

7.5.5.4 unknownX

```
std::vector<double> LinearSys::unknownX  [private]
```

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

- include/numerics/LinearSys.hpp
- src/numerics/LinearSys.cpp

7.6 LoggingSettings Struct Reference

Structure holding configuration parameters for simulation logging (console and file).

`#include <SimulationContext.hpp>`

Public Attributes

- bool logToFile
- bool logToConsole
- std::string dir
- std::string filename
- size_t loggingFrequency

7.6.1 Detailed Description

Structure holding configuration parameters for simulation logging (console and file).

7.6.2 Member Data Documentation

7.6.2.1 dir

```
std::string LoggingSettings::dir
```

7.6.2.2 filename

```
std::string LoggingSettings::filename
```

7.6.2.3 loggingFrequency

```
size_t LoggingSettings::loggingFrequency
```

7.6.2.4 logToConsole

```
bool LoggingSettings::logToConsole
```

7.6.2.5 logToFile

```
bool LoggingSettings::logToFile
```

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

- include/simulation/SimulationContext.hpp

7.7 LogWriter Class Reference

Handles console and file logging.

```
#include <logWriter.hpp>
```

Public Member Functions

- LogWriter (const LoggingSettings &logSettings)
- void write (const std::string &msg)
- void printSimulationHeader (const InputData &input, const SimulationData &simData)
- void printStepHeader ()
- void printStepProgress (int step, double time, double dt, double elapsedSec, bool isOutputStep)
- void printFinalSummary (double totalCpuTimeSec, double meanCpuTimePerCellTimestep, unsigned int totalSteps, const unsigned int totalCells)

Private Member Functions

- std::string separator (int width=60, char c='=') const

Private Attributes

- bool logToFile_
- bool logToConsole_
- std::string logDir_
- std::string filename_
- std::ofstream file_

7.7.1 Detailed Description

Handles console and file logging.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 LogWriter()

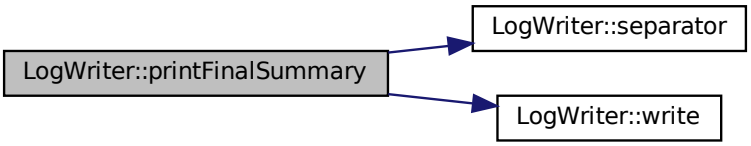
```
LogWriter::LogWriter (
            const LoggingSettings & logSettings )
```

7.7.3 Member Function Documentation

7.7.3.1 printFinalSummary()

```
void LogWriter::printFinalSummary (
            double totalCpuTimeSec,
            double meanCpuTimePerCellTimestep,
            unsigned int totalSteps,
            const unsigned int totalCells )
```

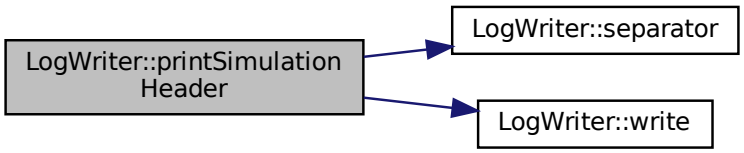
Here is the call graph for this function:



7.7.3.2 printSimulationHeader()

```
void LogWriter::printSimulationHeader (
            const InputData & input,
            const SimulationData & simData )
```

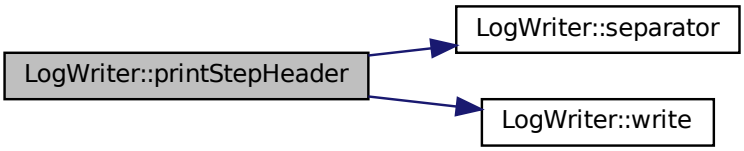
Here is the call graph for this function:



7.7.3.3 printStepHeader()

```
void LogWriter::printStepHeader ( )
```

Here is the call graph for this function:



7.7.3.4 printStepProgress()

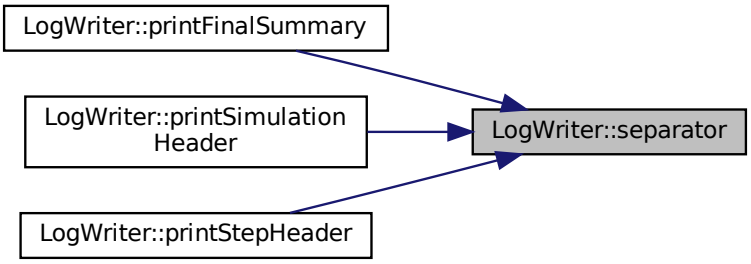
```
void LogWriter::printStepProgress (
            int step,
            double time,
            double dt,
            double elapsedSec,
            bool isOutputStep )
```

Here is the call graph for this function:

7.7.3.5 separator()

```
std::string LogWriter::separator (
            int width = 60,
            char c = '=' ) const  [inline], [private]
```

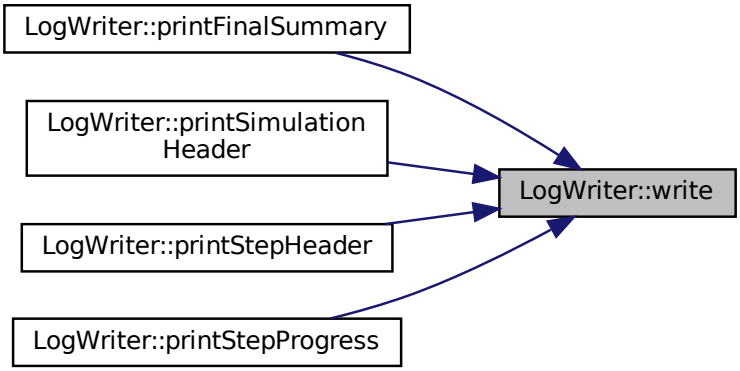
Here is the caller graph for this function:



7.7.3.6 write()

```
void LogWriter::write (
            const std::string & msg )
```

Here is the caller graph for this function:



7.7.4 Member Data Documentation

7.7.4.1 file_

```
std::ofstream LogWriter::file_  [private]
```

7.7.4.2 filename_

```
std::string LogWriter::filename_  [private]
```

7.7.4.3 logDir_

```
std::string LogWriter::logDir_  [private]
```

7.7.4.4 logToConsole_

```
bool LogWriter::logToConsole_  [private]
```

7.7.4.5 logToFile_

```
bool LogWriter::logToFile_  [private]
```

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

- include/io/logWriter.hpp
- src/io/logWriter.cpp

7.8 MeshData Struct Reference

Structure to hold mesh configuration data.

`#include <inputReader.hpp>`

Public Attributes

- int nx
- int ny
- int nz
- double dx
- double dy
- double dz
- bool input_for_manufactured_solution

7.8.1 Detailed Description

Structure to hold mesh configuration data.

7.8.2 Member Data Documentation

7.8.2.1 dx

```
double MeshData::dx
```

7.8.2.2 dy

```
double MeshData::dy
```

7.8.2.3 dz

```
double MeshData::dz
```

7.8.2.4 input_for_manufactured_solution

```
bool MeshData::input_for_manufactured_solution
```

7.8.2.5 nx

```
int MeshData::nx
```

7.8.2.6 ny

```
int MeshData::ny
```

7.8.2.7 nz

```
int MeshData::nz
```

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

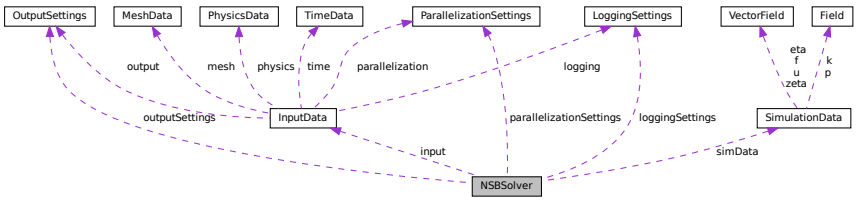
- include/io/inputReader.hpp

7.9 NSBSolver Class Reference

Class responsible for solving the Navier-Stokes-Brinkman equations.

`#include <NSBSolver.hpp>`

Collaboration diagram for NSBSolver:



Public Member Functions

- NSBSolver (const std::string &configFile)
- void setup ()
- void solve ()

Private Attributes

- std::string configFile
- InputData input
- SimulationData simData
- OutputSettings outputSettings
- LoggingSettings loggingSettings
- ParallelizationSettings parallelizationSettings
- std::unique_ptr< ViscousStep > viscousStep
- std::unique_ptr< PressureStep > pressureStep
- std::unique_ptr< VTKWriter > vtkWriter
- std::unique_ptr< LogWriter > logger

7.9.1 Detailed Description

Class responsible for solving the Navier-Stokes-Brinkman equations.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 NSBSolver()

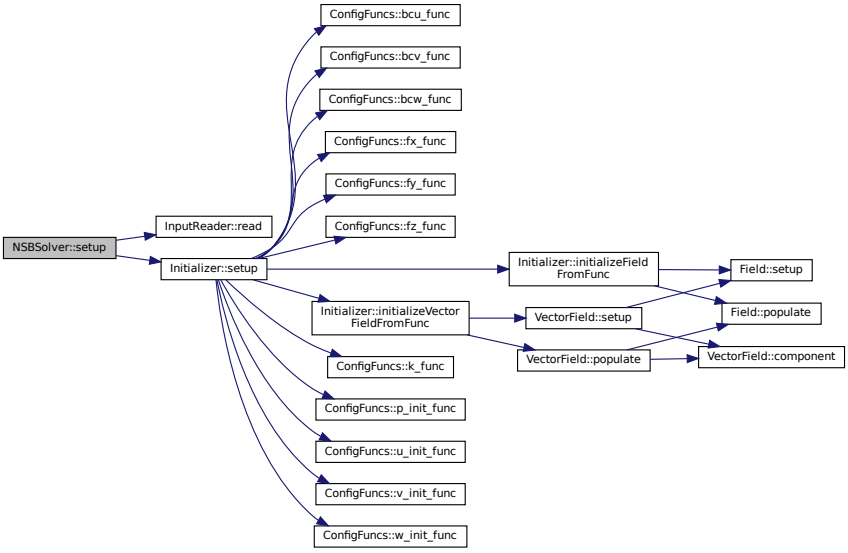
```
NSBSolver::NSBSolver (
            const std::string & configFile )  [explicit]
```

7.9.3 Member Function Documentation

7.9.3.1 setup()

```
void NSBSolver::setup ( )
```

Here is the call graph for this function:



Here is the caller graph for this function:

7.9.3.2 solve()

```
void NSBSolver::solve ( )
```

Here is the call graph for this function:

Here is the caller graph for this function:

7.9.4 Member Data Documentation

7.9.4.1 configFile

```
std::string NSBSolver::configFile  [private]
```

7.9.4.2 input

```
InputData NSBSolver::input  [private]
```

7.9.4.3 logger

```
std::unique_ptr<LogWriter> NSBSolver::logger  [private]
```

7.9.4.4 loggingSettings

```
LoggingSettings NSBSolver::loggingSettings  [private]
```

7.9.4.5 outputSettings

```
OutputSettings NSBSolver::outputSettings  [private]
```

7.9.4.6 parallelizationSettings

```
ParallelizationSettings NSBSolver::parallelizationSettings  [private]
```

7.9.4.7 pressureStep

```
std::unique_ptr<PressureStep> NSBSolver::pressureStep  [private]
```

7.9.4.8 simData

```
SimulationData NSBSolver::simData  [private]
```

7.9.4.9 viscousStep

```
std::unique_ptr<ViscousStep> NSBSolver::viscousStep  [private]
```

7.9.4.10 vtkWriter

```
std::unique_ptr<VTKWriter> NSBSolver::vtkWriter  [private]
```

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

- include/simulation/NSBSolver.hpp
- src/simulation/NSBSolver.cpp

7.10 OutputSettings Struct Reference

Structure holding configuration parameters for outputting simulation results.

`#include <SimulationContext.hpp>`

Public Attributes

- std::string dir
- std::string baseFilename
- bool enabled
- size_t outputFrequency

7.10.1 Detailed Description

Structure holding configuration parameters for outputting simulation results.

7.10.2 Member Data Documentation

7.10.2.1 baseFilename

```
std::string OutputSettings::baseFilename
```

7.10.2.2 dir

```
std::string OutputSettings::dir
```

7.10.2.3 enabled

```
bool OutputSettings::enabled
```

7.10.2.4 outputFrequency

```
size_t OutputSettings::outputFrequency
```

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

- include/simulation/SimulationContext.hpp

7.11 ParallelizationSettings Struct Reference

Structure holding parameters related to parallel execution and domain decomposition.

```
#include <SimulationContext.hpp>
```

Public Attributes

- int schurDomains

7.11.1 Detailed Description

Structure holding parameters related to parallel execution and domain decomposition.

7.11.2 Member Data Documentation

7.11.2.1 schurDomains

```
int ParallelizationSettings::schurDomains
```

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

- include/simulation/SimulationContext.hpp

7.12 PhysicsData Struct Reference

Structure to hold physics parameters.

```
#include <inputReader.hpp>
```

Public Attributes

- double nu

7.12.1 Detailed Description

Structure to hold physics parameters.

7.12.2 Member Data Documentation

7.12.2.1 nu

```
double PhysicsData::nu
```

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

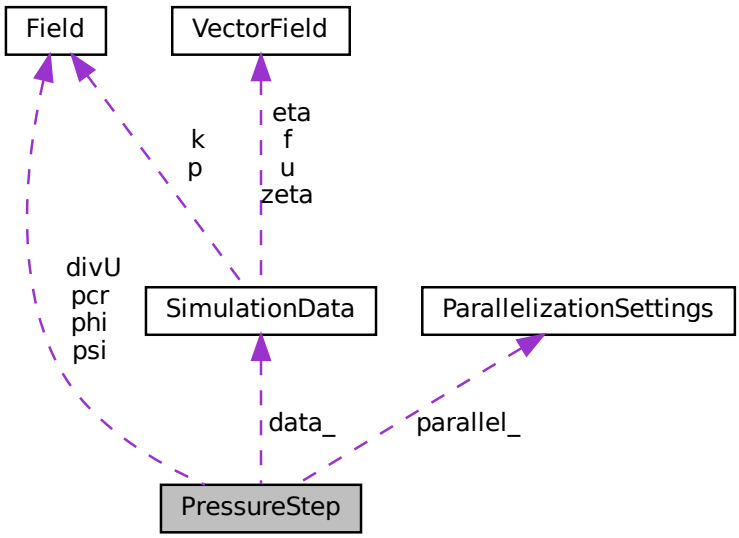
- include/io/inputReader.hpp

7.13 PressureStep Class Reference

Handles all pressure step maniplation. This class does not own data but regulates the workflow.

`#include <pressureStep.hpp>`

Collaboration diagram for PressureStep:



Public Member Functions

- PressureStep (SimulationData &simData, ParallelizationSettings &parallel)

  Constructor.
- void run ()

  Run pressure step.

Private Member Functions

- void initializeWorkspaceFields ()

  Construct temporary fields to proceed in computations.
- std::vector< double > solveSystem (LinearSys &sys, BoundaryType bType)

  Wrapper that chooses whether to use Thomas (P=1) or Schur (P>1).

Private Attributes

- Field psi
- Field phi
- Field pcr
- Field divU
- SimulationData & data_
- ParallelizationSettings parallel_

Friends

- class PressureStepTest
- class PressureStepRobustnessTest

7.13.1 Detailed Description

Handles all pressure step maniplation. This class does not own data but regulates the workflow.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 PressureStep()

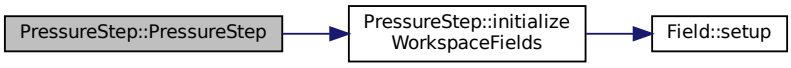
```
PressureStep::PressureStep (
            SimulationData & simData,
            ParallelizationSettings & parallel )
```

Constructor.

Parameters

contex	Contains all simulation data

Here is the call graph for this function:

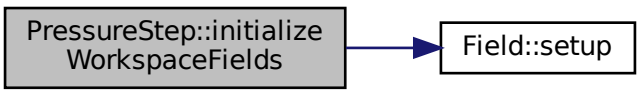


7.13.3 Member Function Documentation

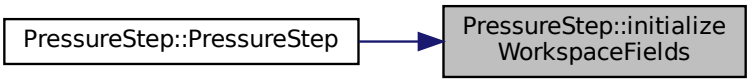
7.13.3.1 initializeWorkspaceFields()

```
void PressureStep::initializeWorkspaceFields ( )  [private]
```

Construct temporary fields to proceed in computations.
Here is the call graph for this function:



Here is the caller graph for this function:

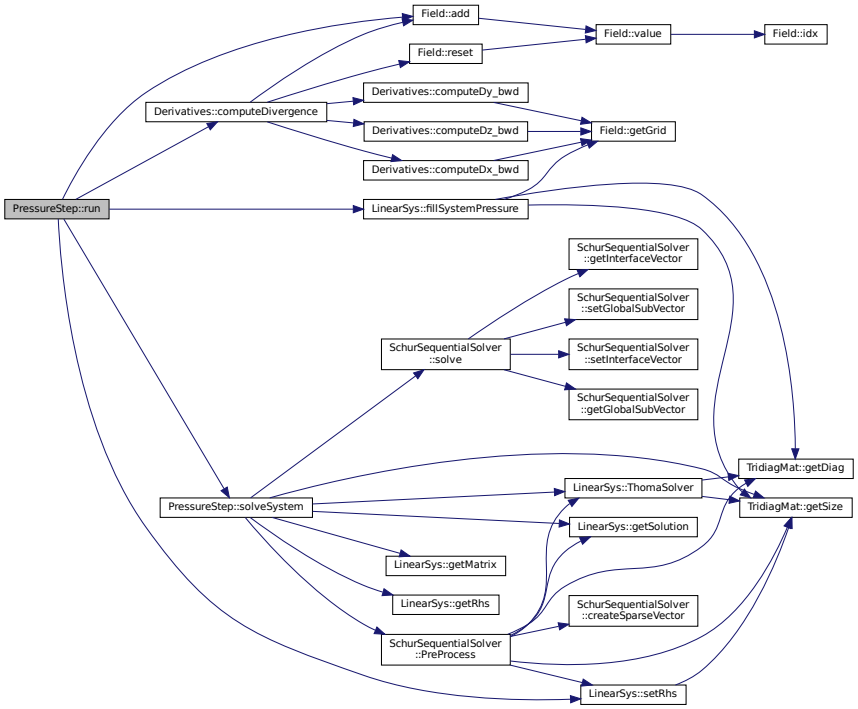


7.13.3.2 run()

```
void PressureStep::run ( )
```

Run pressure step.

Here is the call graph for this function:

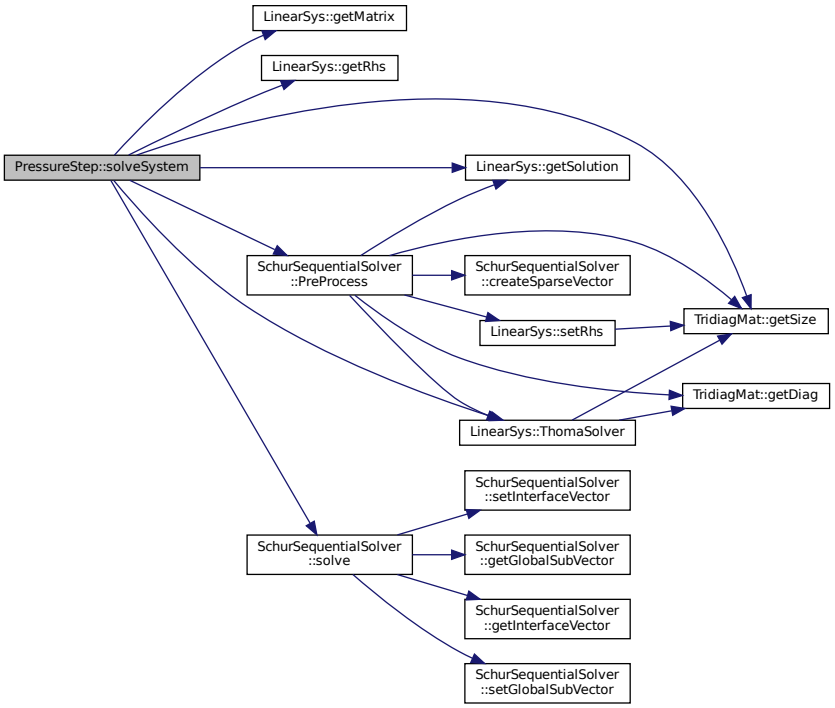


7.13.3.3 solveSystem()

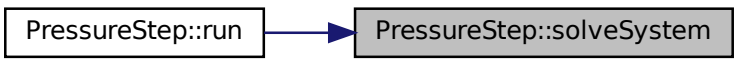
```
std::vector< double > PressureStep::solveSystem (
            LinearSys & sys,
            BoundaryType bType ) [private]
```

Wrapper that chooses whether to use Thomas (P=1) or Schur (P>1).

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.4 Friends And Related Function Documentation

7.13.4.1 PressureStepRobustnessTest

```
friend class PressureStepRobustnessTest  [friend]
```

7.13.4.2 PressureStepTest

```
friend class PressureStepTest  [friend]
```

7.13.5 Member Data Documentation

7.13.5.1 data_

```
SimulationData& PressureStep::data_ [private]
```

7.13.5.2 divU

```
Field PressureStep::divU [private]
```

7.13.5.3 parallel_

```
ParallelizationSettings PressureStep::parallel_ [private]
```

7.13.5.4 pcr

```
Field PressureStep::pcr [private]
```

7.13.5.5 phi

```
Field PressureStep::phi [private]
```

7.13.5.6 psi

```
Field PressureStep::psi [private]
```

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

- include/simulation/pressureStep.hpp
- src/simulation/pressureStep.cpp

7.14 SchurSequentialSolver Class Reference

Implements the Schur Complement method for solving large tridiagonal systems by decomposing them into smaller sub-systems solved sequentially.

`#include <SchurSequentialSolver.hpp>`

Public Member Functions

- SchurSequentialSolver (int globalSize, int numDomains, BoundaryType type)
- void PreProcess (const TridiagMat &A_global)

  Phase 0: Pre-processing. Builds the submatrices A_ii, extracts interface coefficients, and assembles the Schur matrix 'S'.
- std::vector< double > solve (const std::vector< double > &f_global)

  Phase 1: Solve. Solves the system A∗x = f using the Schur algorithm. 'PreProcess' must have been called before.

Private Member Functions

- std::vector< double > getGlobalSubVector (const std::vector< double > &globalVec, int start, int size) const
- void setGlobalSubVector (std::vector< double > &globalVec, const std::vector< double > &subVec, int start) const
- std::vector< double > getInterfaceVector (const std::vector< double > &globalVec) const
- void setInterfaceVector (std::vector< double > &globalVec, const std::vector< double > &interfaceVec) const
- std::vector< double > createSparseVector (int size, int index, double value) const

Private Attributes

- int nGlobal
- int num_domains
- int num_interfaces
- BoundaryType bType
- std::vector< int > domain_sizes
- std::vector< int > domain_starts
- std::vector< int > interface_indices
- std::vector< LinearSys > localSolvers
- std::unique_ptr< LinearSys > schurSolver
- double schurScalarS = 0.0
- std::vector< double > A_ie_left
- std::vector< double > A_ie_right
- std::vector< double > A_ei_left
- std::vector< double > A_ei_right

7.14.1 Detailed Description

Implements the Schur Complement method for solving large tridiagonal systems by decomposing them into smaller sub-systems solved sequentially.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 SchurSequentialSolver()

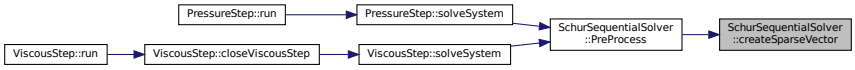
```
SchurSequentialSolver::SchurSequentialSolver (
            int globalSize,
            int numDomains,
            BoundaryType type )
```

7.14.3 Member Function Documentation

7.14.3.1 createSparseVector()

```
std::vector< double > SchurSequentialSolver::createSparseVector (
            int size,
            int index,
            double value ) const  [private]
```

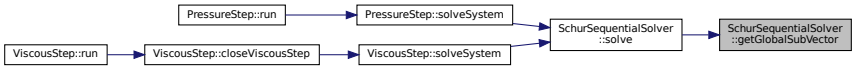
Here is the caller graph for this function:



7.14.3.2 getGlobalSubVector()

```
std::vector< double > SchurSequentialSolver::getGlobalSubVector (
            const std::vector< double > & globalVec,
            int start,
            int size ) const  [private]
```

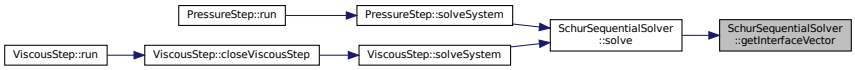
Here is the caller graph for this function:



7.14.3.3 getInterfaceVector()

```
std::vector< double > SchurSequentialSolver::getInterfaceVector (
            const std::vector< double > & globalVec ) const  [private]
```

Here is the caller graph for this function:



7.14.3.4 PreProcess()

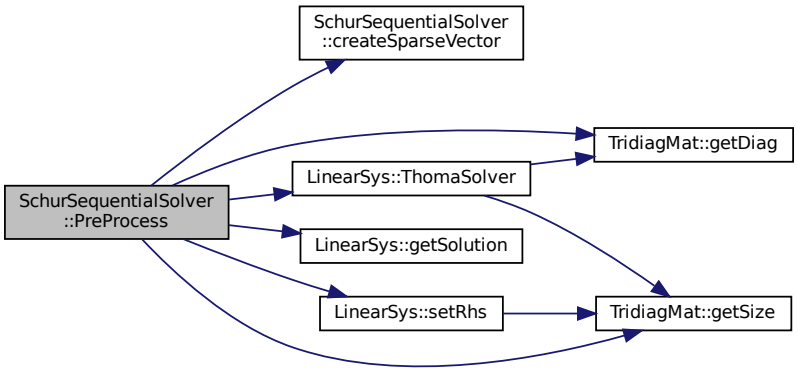
```
void SchurSequentialSolver::PreProcess (
            const TridiagMat & A_global )
```

Phase 0: Pre-processing. Builds the submatrices A_ii, extracts interface coefficients, and assembles the Schur matrix 'S'.

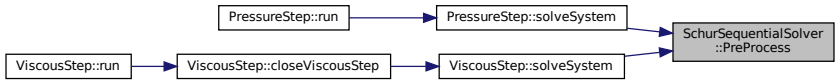
Parameters

A_global	The GLOBAL tridiagonal matrix (N x N).

Here is the call graph for this function:



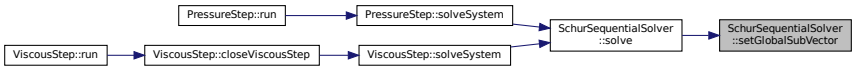
Here is the caller graph for this function:



7.14.3.5 setGlobalSubVector()

```
void SchurSequentialSolver::setGlobalSubVector (
            std::vector< double > & globalVec,
            const std::vector< double > & subVec,
            int start ) const  [private]
```

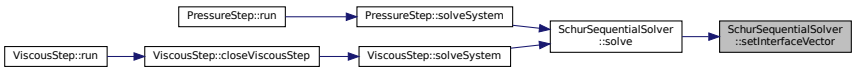
Here is the caller graph for this function:



7.14.3.6 setInterfaceVector()

```
void SchurSequentialSolver::setInterfaceVector (
            std::vector< double > & globalVec,
            const std::vector< double > & interfaceVec ) const  [private]
```

Here is the caller graph for this function:



7.14.3.7 solve()

```
std::vector< double > SchurSequentialSolver::solve (
            const std::vector< double > & f_global )
```

Phase 1: Solve. Solves the system A∗x = f using the Schur algorithm. 'PreProcess' must have been called before.

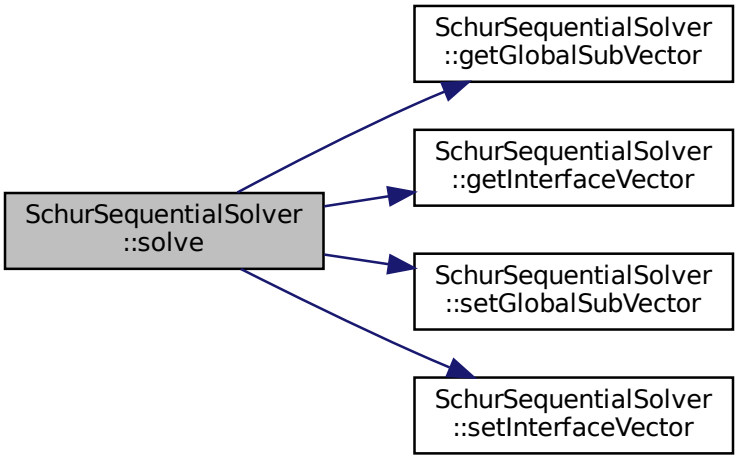
Parameters

f_global	The global RHS vector (length N).

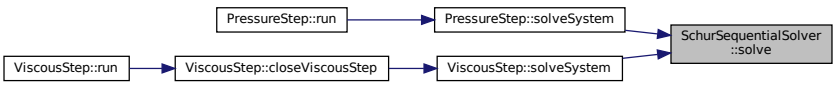
Returns

The global solution vector x (length N).

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.4 Member Data Documentation

7.14.4.1 A_ei_left

```
std::vector<double> SchurSequentialSolver::A_ei_left  [private]
```

7.14.4.2 A_ei_right

```
std::vector<double> SchurSequentialSolver::A_ei_right  [private]
```

7.14.4.3 A_ie_left

```
std::vector<double> SchurSequentialSolver::A_ie_left  [private]
```

7.14.4.4 A_ie_right

```
std::vector<double> SchurSequentialSolver::A_ie_right  [private]
```

7.14.4.5 bType

```
BoundaryType SchurSequentialSolver::bType  [private]
```

7.14.4.6 domain_sizes

```
std::vector<int> SchurSequentialSolver::domain_sizes  [private]
```

7.14.4.7 domain_starts

```
std::vector<int> SchurSequentialSolver::domain_starts  [private]
```

7.14.4.8 interface_indices

```
std::vector<int> SchurSequentialSolver::interface_indices  [private]
```

7.14.4.9 localSolvers

```
std::vector<LinearSys> SchurSequentialSolver::localSolvers  [private]
```

7.14.4.10 nGlobal

```
int SchurSequentialSolver::nGlobal  [private]
```

7.14.4.11 num_domains

```
int SchurSequentialSolver::num_domains  [private]
```

7.14.4.12 num_interfaces

```
int SchurSequentialSolver::num_interfaces  [private]
```

7.14.4.13 schurScalarS

```
double SchurSequentialSolver::schurScalarS = 0.0  [private]
```

7.14.4.14 schurSolver

```
std::unique_ptr<LinearSys> SchurSequentialSolver::schurSolver  [private]
```

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

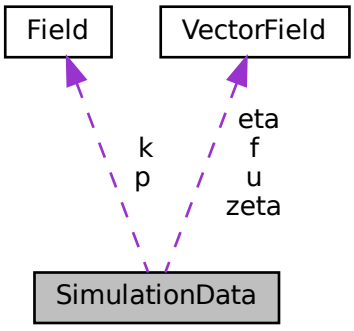
- include/numerics/SchurSequentialSolver.hpp
- src/numerics/SchurSequentialSolver.cpp

7.15 SimulationData Struct Reference

Central structure containing all transient data, fields, and physical properties of the running simulation.

`#include <SimulationContext.hpp>`

Collaboration diagram for SimulationData:



Public Attributes

- GridPtr grid
- double dt
- double currTime
- double totalSimTime
- size_t currStep
- size_t totalSteps
- VectorField eta
- VectorField zeta
- VectorField u
- Field p
- Func bcu
- Func bcv
- Func bcw

- double nu
- Field k
- VectorField f
- Func fx
- Func fy
- Func fz

7.15.1 Detailed Description

Central structure containing all transient data, fields, and physical properties of the running simulation.

7.15.2 Member Data Documentation

7.15.2.1 bcu

```
Func SimulationData::bcu
```

7.15.2.2 bcv

```
Func SimulationData::bcv
```

7.15.2.3 bcw

```
Func SimulationData::bcw
```

7.15.2.4 currStep

```
size_t SimulationData::currStep
```

7.15.2.5 currTime

```
double SimulationData::currTime
```

7.15.2.6 dt

```
double SimulationData::dt
```

7.15.2.7 eta

```
VectorField SimulationData::eta
```

7.15.2.8 f

```
VectorField SimulationData::f
```

7.15.2.9 fx

```
Func SimulationData::fx
```

7.15.2.10 fy

```
Func SimulationData::fy
```

7.15.2.11 fz

```
Func SimulationData::fz
```

7.15.2.12 grid

```
GridPtr SimulationData::grid
```

7.15.2.13 k

```
Field SimulationData::k
```

7.15.2.14 nu

```
double SimulationData::nu
```

7.15.2.15 p

```
Field SimulationData::p
```

7.15.2.16 totalSimTime

```
double SimulationData::totalSimTime
```

7.15.2.17 totalSteps

```
size_t SimulationData::totalSteps
```

7.15.2.18 u

```
VectorField SimulationData::u
```

7.15.2.19 zeta

```
VectorField SimulationData::zeta
```

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

- include/simulation/SimulationContext.hpp

7.16 TimeData Struct Reference

Structure to hold time integration parameters.

`#include <inputReader.hpp>`

Public Attributes

- double dt
- double t_end

7.16.1 Detailed Description

Structure to hold time integration parameters.

7.16.2 Member Data Documentation

7.16.2.1 dt

```
double TimeData::dt
```

7.16.2.2 t_end

```
double TimeData::t_end
```

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

- include/io/inputReader.hpp

7.17 TridiagMat Class Reference

Class representing a tridiagonal matrix and providing access to its elements.

```
#include <TridiagMat.hpp>
```

Public Member Functions

- TridiagMat (int n)

  Constructor.
- void fillMat (std::vector< double > diag, std::vector< double > subdiag, std::vector< double > supdiag)

  Fill the matrix.
- const unsigned int getSize () const

  Get the matrix size.
- double getElement (int i, int j) const

  Get the (i,j) element of the matrix.
- std::vector< double > & getDiag (int w)

  Get the whole diagonal, subdiagonal or supdiagonal.
- std::vector< double > getDiag (int w) const
- double getFirstElementFromDiag (int w) const

  Get the first element from the diagonal w.
- double getLastElementFromDiag (int w) const

  Get the last element from the diagonal w.

Private Attributes

- std::vector< double > diag
- std::vector< double > subdiag
- std::vector< double > supdiag
- const unsigned int size

Friends

- class SchurSequentialSolver

7.17.1 Detailed Description

Class representing a tridiagonal matrix and providing access to its elements.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 TridiagMat()

```
TridiagMat::TridiagMat (
            int n )
```

Constructor.

Parameters

n	Size of the matrix (n x n)

7.17.3 Member Function Documentation

7.17.3.1 fillMat()

```
void TridiagMat::fillMat (
            std::vector< double > diag,
            std::vector< double > subdiag,
            std::vector< double > supdiag )
```

Fill the matrix.

Parameters

diag	matrix diagonal.
subdiag	matrix subdiagonal.
supdiag	matrix upper diagonal.

7.17.3.2 getDiag() [1/2]

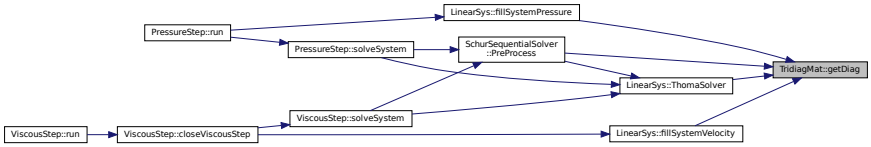
```
std::vector< double > & TridiagMat::getDiag (
            int w )
```

Get the whole diagonal, subdiagonal or supdiagonal.

Parameters

w	number indicating sub (-1), diag (0) or sup (1)

Here is the caller graph for this function:



7.17.3.3 getDiag() [2/2]

```
std::vector< double > TridiagMat::getDiag (
            int w ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.17.3.4 getElement()

```
double TridiagMat::getElement (
            int i,
            int j ) const
```

Get the (i,j) element of the matrix.

7.17.3.5 getFirstElementFromDiag()

```
double TridiagMat::getFirstElementFromDiag (
            int w ) const
```

Get the first element from the diagonal w.

Parameters

w	number indicating sub (-1), main (0) or sup (1) diagonal

7.17.3.6 getLastElementFromDiag()

```
double TridiagMat::getLastElementFromDiag (
            int w ) const
```

Get the last element from the diagonal w.

Parameters

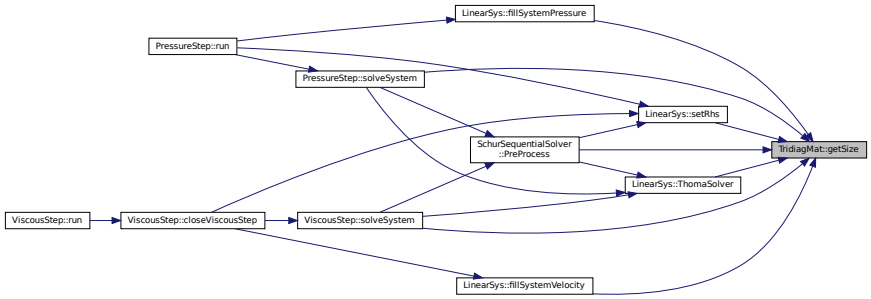
w	number indicating sub (-1), main (0) or sup (1) diagonal

7.17.3.7 getSize()

```
const unsigned int TridiagMat::getSize ( ) const  [inline]
```

Get the matrix size.

Here is the caller graph for this function:



7.17.4 Friends And Related Function Documentation

7.17.4.1 SchurSequentialSolver

```
friend class SchurSequentialSolver  [friend]
```

7.17.5 Member Data Documentation

7.17.5.1 diag

```
std::vector<double> TridiagMat::diag  [private]
```

7.17.5.2 size

```
const unsigned int TridiagMat::size  [private]
```

7.17.5.3 subdiag

```
std::vector<double> TridiagMat::subdiag  [private]
```

7.17.5.4 supdiag

```
std::vector<double> TridiagMat::supdiag  [private]
```

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

- include/core/TridiagMat.hpp
- src/core/TridiagMat.cpp

7.18 VectorField Class Reference

Class representing a 3D vector field defined on a 3D grid.

```
#include <Fields.hpp>
```

Public Types

- using Scalar = Field::Scalar

Public Member Functions

- void setup (const GridPtr &grid, const Func &populateXFunction=ZERO_FUNC, const Func &populate↩ YFunction=ZERO_FUNC, const Func &populateZFunction=ZERO_FUNC)

  Setup the vector field with functions to populate each component (e.g., initial conditions).
- void populate (double time=0)

  Populates each component of the underlying vector field based on the stored functions in (x,y,z,t).
- const Grid & getGrid ()

  Getter for the grid information.
- const Grid & getGrid () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Field & operator() (Axis componentDirection)

  Access the component of the vector field in the specified direction.
- const Field & operator() (Axis componentDirection) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Field & component (Axis componentDirection)

  Access the component of the vector field in the specified direction.
- const Field & component (Axis componentDirection) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Scalar & operator() (Axis componentDirection, size_t i, size_t j, size_t k)

  Access the component value of the vector field in the specified direction at the given position.
- const Scalar & operator() (Axis componentDirection, size_t i, size_t j, size_t k) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Scalar & value (Axis componentDirection, size_t i, size_t j, size_t k)

  Access the component value of the vector field in the specified direction at the given position.
- const Scalar & value (Axis componentDirection, size_t i, size_t j, size_t k) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void add (Scalar value)

  Add a scalar value to all components of the vector field.
- void add (const VectorField &other)

  Add another vector field to this vector field element-wise.
- void multiply (Scalar value)

  Multiply all components of the vector field by a scalar value.

Private Attributes

- GridPtr gridPtr_

  The pointer to the grid information (dimensions and spacing).
- std::array< Field, AXIS_COUNT > components_

  The components of the vector field (x, y, z).

7.18.1 Detailed Description

Class representing a 3D vector field defined on a 3D grid.

7.18.2 Member Typedef Documentation

7.18.2.1 Scalar

```
using VectorField::Scalar = Field::Scalar
```

7.18.3 Member Function Documentation

7.18.3.1 add() [1/2]

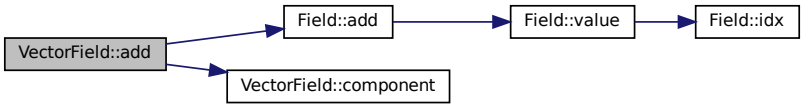
```
void VectorField::add (
            const VectorField & other )
```

Add another vector field to this vector field element-wise.

Parameters

other	the other vector field to add

Here is the call graph for this function:



7.18.3.2 add() [2/2]

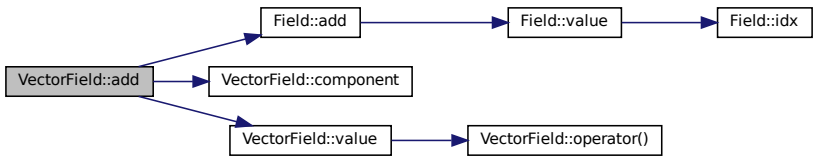
```
void VectorField::add (
            const Field::Scalar value )
```

Add a scalar value to all components of the vector field.

Parameters

value	the scalar value to add

Here is the call graph for this function:



7.18.3.3 component() [1/2]

```
Field& VectorField::component (
```

```
            Axis componentDirection )  [inline]
```

Access the component of the vector field in the specified direction.

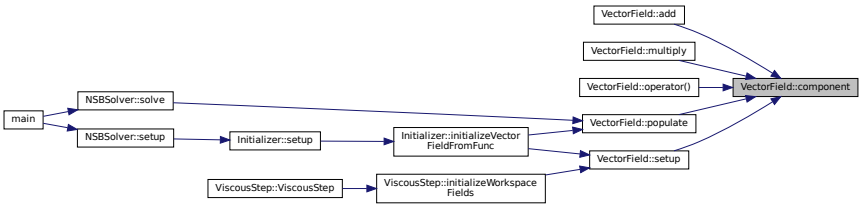
Parameters

componentDirection	the direction (Axis::X, Axis::Y, or Axis::Z)

Returns

the component of the vector field in the specified direction

This method is a named alias for the access operator. Here is the caller graph for this function:



7.18.3.4 component() [2/2]

```
const Field& VectorField::component (
            Axis componentDirection ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.18.3.5 getGrid() [1/2]

```
const Grid& VectorField::getGrid ( )  [inline]
```

Getter for the grid information.

Returns

the pointer to the grid information

Here is the caller graph for this function:

7.18.3.6 getGrid() [2/2]

```
const Grid& VectorField::getGrid ( ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.18.3.7 multiply()

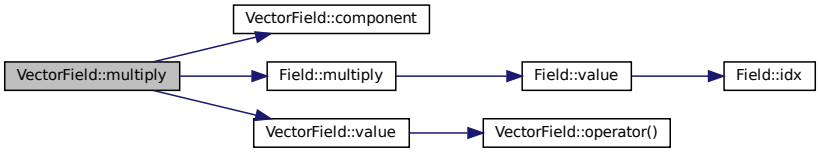
```
void VectorField::multiply (
            const Field::Scalar value )
```

Multiply all components of the vector field by a scalar value.

Parameters

value	the scalar value to multiply by

Here is the call graph for this function:



7.18.3.8 operator()() [1/4]

```
Field& VectorField::operator() (
            Axis componentDirection )  [inline]
```

Access the component of the vector field in the specified direction.

Parameters

componentDirection	the direction (Axis::X, Axis::Y, or Axis::Z)

Returns

the component of the vector field in the specified direction

Here is the caller graph for this function:

7.18.3.9 operator()() [2/4]

```
const Field& VectorField::operator() (
            Axis componentDirection ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

7.18.3.10 operator()() [3/4]

```
Field::Scalar & VectorField::operator() (
            Axis componentDirection,
            size_t i,
            size_t j,
            size_t k )
```

Access the component value of the vector field in the specified direction at the given position.

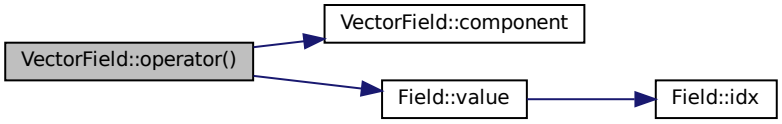
Parameters

direction	the direction (Axis::X, Axis::Y, or Axis::Z)
i	the x-index
j	the y-index
k	the z-index

Returns

the component value of the vector field in the specified direction at position (i,j,k)

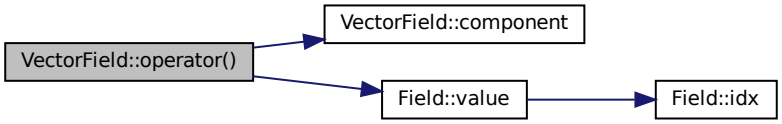
Here is the call graph for this function:



7.18.3.11 operator()() [4/4]

```
const Field::Scalar & VectorField::operator() (
            Axis componentDirection,
            size_t i,
            size_t j,
            size_t k ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
Here is the call graph for this function:



7.18.3.12 populate()

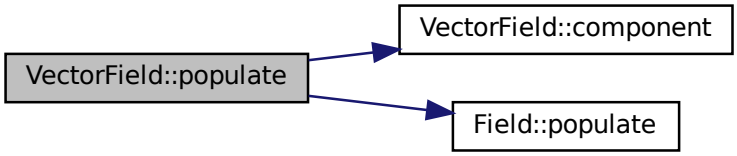
```
void VectorField::populate (
            double time = 0 )  [inline]
```

Populates each component of the underlying vector field based on the stored functions in (x,y,z,t).

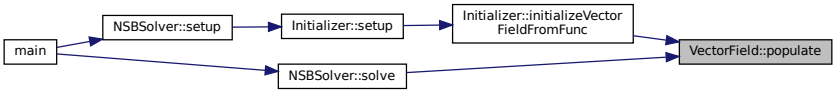
Parameters

time	the time affecting the stored functions in (x,y,z,t)

Here is the call graph for this function:



Here is the caller graph for this function:



7.18.3.13 setup()

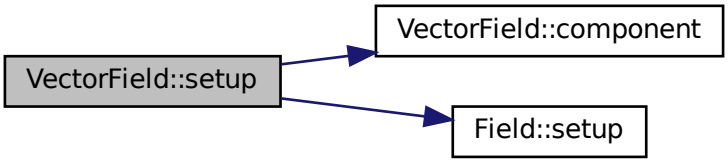
```
void VectorField::setup (
            const GridPtr & grid,
            const Func & populateXFunction = ZERO_FUNC,
            const Func & populateYFunction = ZERO_FUNC,
            const Func & populateZFunction = ZERO_FUNC )  [inline]
```

Setup the vector field with functions to populate each component (e.g., initial conditions).

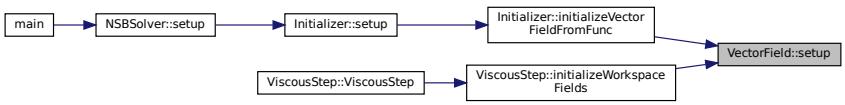
Parameters

grid	the pointer to the grid information
populateXFunction	the function to use for populating the x-component of the vector field
populateYFunction	the function to use for populating the y-component of the vector field
populateZFunction	the function to use for populating the z-component of the vector field

Here is the call graph for this function:



Here is the caller graph for this function:



7.18.3.14 value() [1/2]

```
Scalar& VectorField::value (
            Axis componentDirection,
            size_t i,
            size_t j,
            size_t k )  [inline]
```

Access the component value of the vector field in the specified direction at the given position.

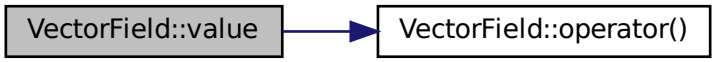
Parameters

direction	the direction (Axis::X, Axis::Y, or Axis::Z)
i	the x-index
j	the y-index
k	the z-index

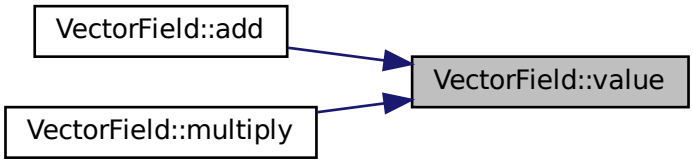
Returns

the component value of the vector field in the specified direction at position (i,j,k)

This method is a named alias for the access operator. Here is the call graph for this function:



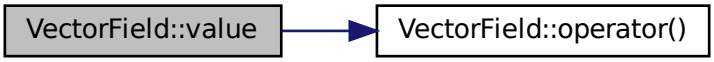
Here is the caller graph for this function:



7.18.3.15 value() [2/2]

```
const Scalar& VectorField::value (
            Axis componentDirection,
            size_t i,
            size_t j,
            size_t k ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
Here is the call graph for this function:



7.18.4 Member Data Documentation

7.18.4.1 components_

```
std::array<Field, AXIS_COUNT> VectorField::components_  [private]
```

The components of the vector field (x, y, z).

7.18.4.2 gridPtr_

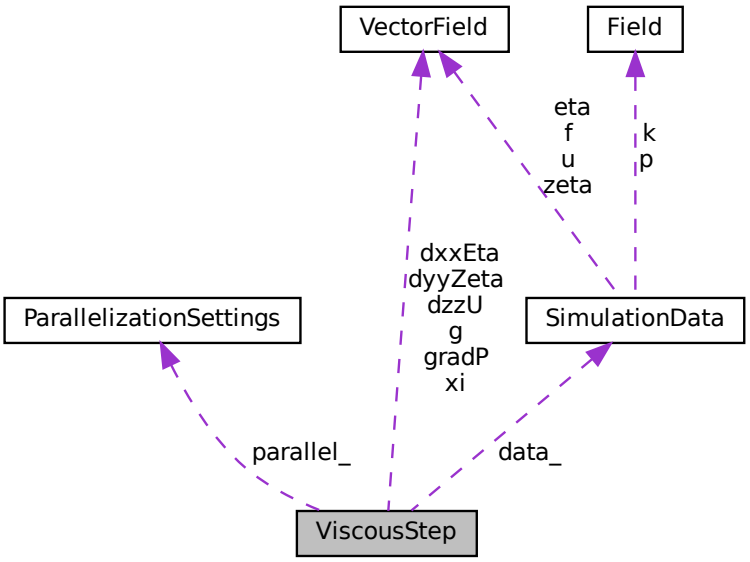
```
GridPtr VectorField::gridPtr_  [private]
```

The pointer to the grid information (dimensions and spacing).
The documentation for this class was generated from the following files:

- include/core/Fields.hpp
- src/core/Fields.cpp

7.19 ViscousStep Class Reference

Handles all viscous step maniplation. This class does not own data but regulates the workflow.
`#include <viscousStep.hpp>`
Collaboration diagram for ViscousStep:



Public Member Functions

- ViscousStep (SimulationData &simData, ParallelizationSettings &parallel)

  Constructor.
- void run ()

  Run viscous step.

Private Member Functions

- void initializeWorkspaceFields ()

  Construct temporary fields to proceed in computations.
- void computeG ()

  Compute G term.
- void computeXi ()

  Compute xi term.
- void closeViscousStep ()

  Closes viscous step filling and solving three linear systems.
- std::vector< double > solveSystem (LinearSys &sys, BoundaryType bType)

  Wrapper that chooses whether to use Thomas (P=1) or Schur (P>1).

Private Attributes

- VectorField g
- VectorField gradP
- VectorField dxxEta

- VectorField dyyZeta
- VectorField dzzU
- VectorField xi
- SimulationData & data_
- ParallelizationSettings parallel_

Friends

- class ViscousStepTest
- class ViscousStepSolverTest
- class ViscousStepRobustnessTest

7.19.1 Detailed Description

Handles all viscous step maniplation. This class does not own data but regulates the workflow.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 ViscousStep()

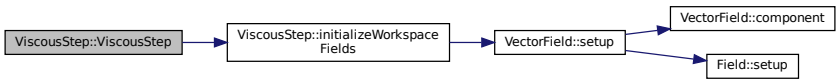
```
ViscousStep::ViscousStep (
            SimulationData & simData,
            ParallelizationSettings & parallel )
```

Constructor.

Parameters

contex	Contains all simulation data

Here is the call graph for this function:



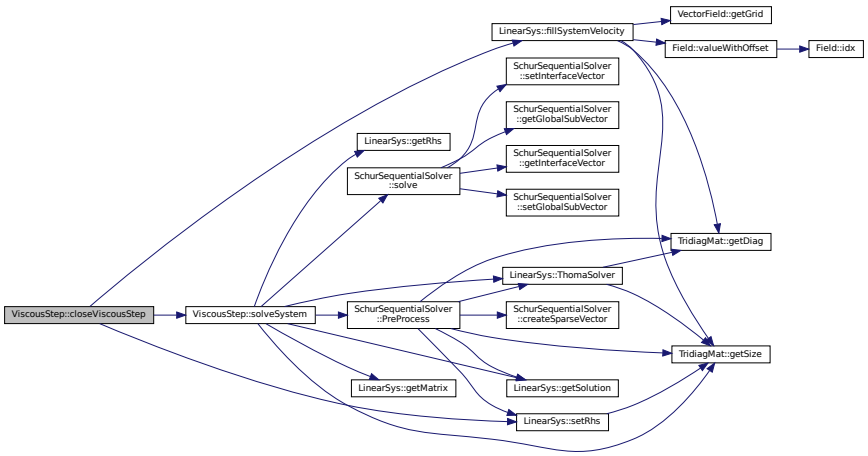
7.19.3 Member Function Documentation

7.19.3.1 closeViscousStep()

```
void ViscousStep::closeViscousStep ( )  [private]
```

Closes viscous step filling and solving three linear systems.

Here is the call graph for this function:



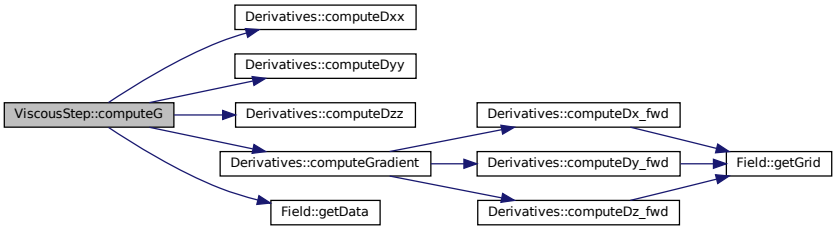
Here is the caller graph for this function:

7.19.3.2 computeG()

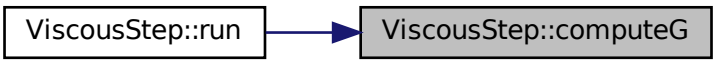
`void ViscousStep::computeG ( ) [private]`

Compute G term.

Here is the call graph for this function:



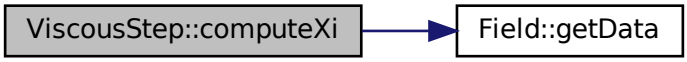
Here is the caller graph for this function:



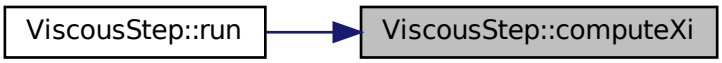
7.19.3.3 computeXi()

```
void ViscousStep::computeXi ( )  [private]
```

Compute xi term.
Here is the call graph for this function:



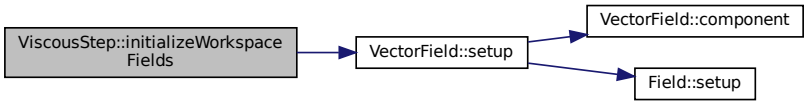
Here is the caller graph for this function:



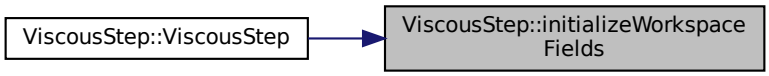
7.19.3.4 initializeWorkspaceFields()

```
void ViscousStep::initializeWorkspaceFields ( )  [private]
```

Construct temporary fields to proceed in computations.
Here is the call graph for this function:



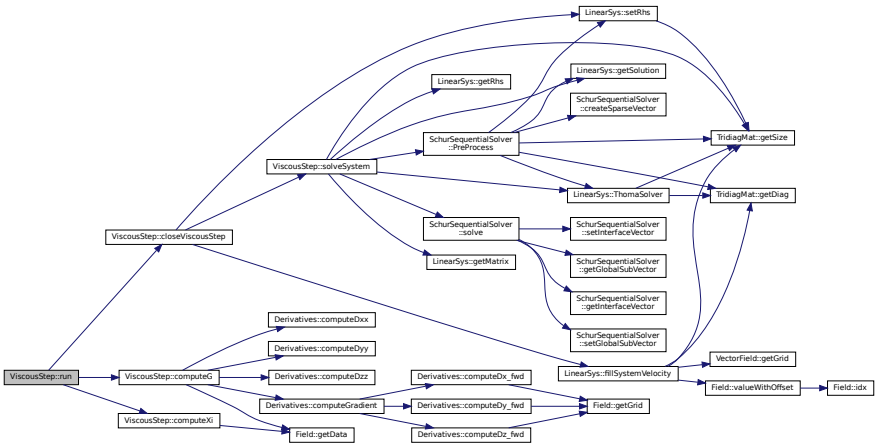
Here is the caller graph for this function:



7.19.3.5 run()

```
void ViscousStep::run ( )
```

Run viscous step.
Here is the call graph for this function:

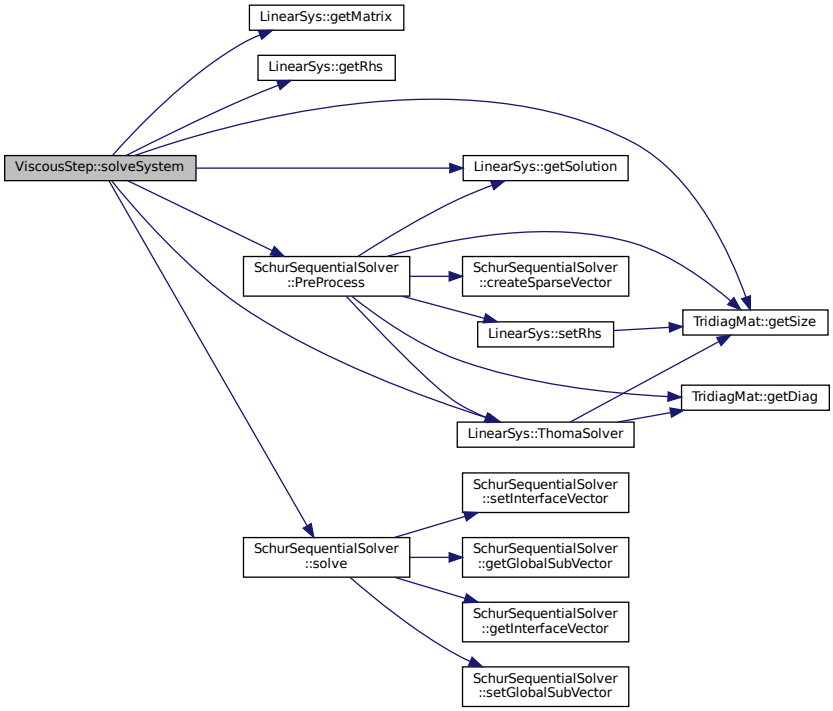


7.19.3.6 solveSystem()

```
std::vector< double > ViscousStep::solveSystem (
            LinearSys & sys,
            BoundaryType bType ) [private]
```

Wrapper that chooses whether to use Thomas (P=1) or Schur (P>1).

Here is the call graph for this function:



Here is the caller graph for this function:

7.19.4 Friends And Related Function Documentation

7.19.4.1 ViscousStepRobustnessTest

```
friend class ViscousStepRobustnessTest  [friend]
```

7.19.4.2 ViscousStepSolverTest

```
friend class ViscousStepSolverTest  [friend]
```

7.19.4.3 ViscousStepTest

```
friend class ViscousStepTest  [friend]
```

7.19.5 Member Data Documentation

7.19.5.1 data_

```
SimulationData& ViscousStep::data_  [private]
```

7.19.5.2 dxxEta

```
VectorField ViscousStep::dxxEta  [private]
```

7.19.5.3 dyyZeta

```
VectorField ViscousStep::dyyZeta  [private]
```

7.19.5.4 dzzU

```
VectorField ViscousStep::dzzU  [private]
```

7.19.5.5 g

```
VectorField ViscousStep::g  [private]
```

7.19.5.6 gradP

```
VectorField ViscousStep::gradP  [private]
```

7.19.5.7 parallel_

```
ParallelizationSettings ViscousStep::parallel_  [private]
```

7.19.5.8 xi

```
VectorField ViscousStep::xi  [private]
```

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

- include/simulation/viscousStep.hpp
- src/simulation/viscousStep.cpp

7.20 VTKWriter Class Reference

Class to write 3D fields to legacy VTK (STRUCTURED_POINTS) for a uniform Cartesian grid, managing output frequency internally.

`#include <VTKWriter.hpp>`

Public Member Functions

- VTKWriter (const OutputSettings &outputSettings, const SimulationData &simData)

  Construct a VTKWriter with output settings and grid dimensions.
- bool write_timestep_if_needed (size_t currStep, const Field &pressure, const VectorField &velocity)

  Writes a timestep file if the current step matches the output frequency. It takes copies of the fields internally to prevent modification during writing.

Private Member Functions

- void write_legacy (const std::string &filename, const std::shared_ptr< Field > &pressure, const std::shared_ptr< VectorField > &velocity) const

  Internal method to write a scalar field and a vector field to a legacy VTK file.

Private Attributes

- int Nx_
- int Ny_
- int Nz_
- double dx_
- double dy_
- double dz_
- bool enabled_
- int outputFrequency_
- std::string basePrefix_

7.20.1 Detailed Description

Class to write 3D fields to legacy VTK (STRUCTURED_POINTS) for a uniform Cartesian grid, managing output frequency internally.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 VTKWriter()

```
VTKWriter::VTKWriter (
            const OutputSettings & outputSettings,
            const SimulationData & simData )
```

Construct a VTKWriter with output settings and grid dimensions.

Parameters

outputSettings	Output settings, including frequency.
simData	Simulation data containing grid dimensions (Nx, Ny, Nz, dx, dy, dz).

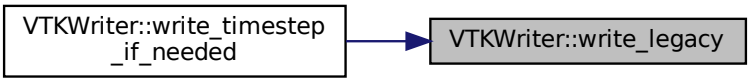
7.20.3 Member Function Documentation

7.20.3.1 write_legacy()

```
void VTKWriter::write_legacy (
            const std::string & filename,
            const std::shared_ptr< Field > & pressure,
            const std::shared_ptr< VectorField > & velocity ) const  [private]
```

Internal method to write a scalar field and a vector field to a legacy VTK file.

Here is the caller graph for this function:



7.20.3.2 write_timestep_if_needed()

```
bool VTKWriter::write_timestep_if_needed (
            size_t currStep,
            const Field & pressure,
            const VectorField & velocity )
```

Writes a timestep file if the current step matches the output frequency. It takes copies of the fields internally to prevent modification during writing.

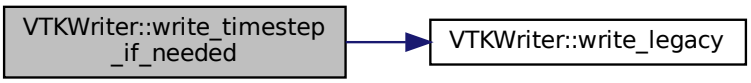
Parameters

currStep	The current simulation step.
pressure	The scalar pressure field.
velocity	The vector velocity field.

Returns

true if the file was written, false otherwise.

Here is the call graph for this function:



7.20.4 Member Data Documentation

7.20.4.1 basePrefix_

```
std::string VTKWriter::basePrefix_  [private]
```

7.20.4.2 dx_

```
double VTKWriter::dx_  [private]
```

7.20.4.3 dy_

```
double VTKWriter::dy_  [private]
```

7.20.4.4 dz_

```
double VTKWriter::dz_  [private]
```

7.20.4.5 enabled_

```
bool VTKWriter::enabled_  [private]
```

7.20.4.6 Nx_

```
int VTKWriter::Nx_  [private]
```

7.20.4.7 Ny_

```
int VTKWriter::Ny_  [private]
```

7.20.4.8 Nz_

```
int VTKWriter::Nz_  [private]
```

7.20.4.9 outputFrequency_

```
int VTKWriter::outputFrequency_  [private]
```

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

- include/io/VTKWriter.hpp
- src/io/VTKWriter.cpp

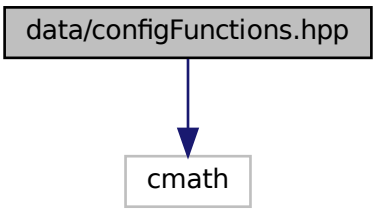
Chapter 8

File Documentation

8.1 data/configFunctions.hpp File Reference

```
#include <cmath>
```

Include dependency graph for configFunctions.hpp:



Namespaces

- ConfigFuncs

Functions

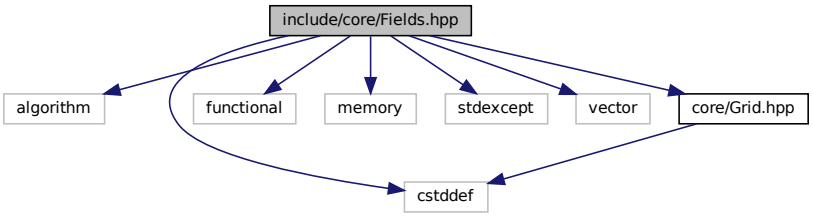
- double ConfigFuncs::bcu_func (double x, double y, double z, double t)
- double ConfigFuncs::bcv_func (double x, double y, double z, double t)
- double ConfigFuncs::bcw_func (double x, double y, double z, double t)
- double ConfigFuncs::fx_func (double x, double y, double z, double t)
- double ConfigFuncs::fy_func (double x, double y, double z, double t)
- double ConfigFuncs::fz_func (double x, double y, double z, double t)
- double ConfigFuncs::u_init_func (double x, double y, double z, double t=0)
- double ConfigFuncs::v_init_func (double x, double y, double z, double t=0)
- double ConfigFuncs::w_init_func (double x, double y, double z, double t=0)
- double ConfigFuncs::p_init_func (double x, double y, double z, double t=0)
- double ConfigFuncs::k_func (double, double, double, double=0)

8.2 include/core/Fields.hpp File Reference

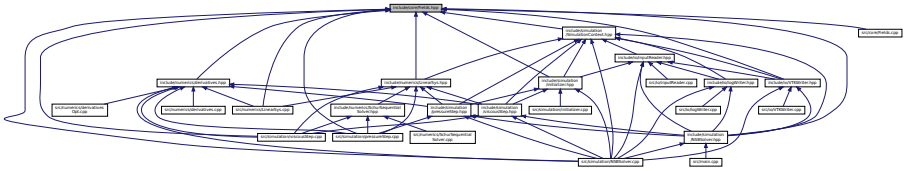
```
#include <algorithm>
#include <cstddef>
```

```
#include <functional>
#include <memory>
#include <stdexcept>
#include <vector>
#include "core/Grid.hpp"
```

Include dependency graph for Fields.hpp:



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



Classes

- class Field

  Class representing a scalar field defined on a 3D grid.
- class VectorField

  Class representing a 3D vector field defined on a 3D grid.

Typedefs

- using Func = std::function< double(double x, double y, double z, double t)>

Variables

- const Func ZERO_FUNC

8.2.1 Typedef Documentation

8.2.1.1 Func

```
using Func = std::function<double(double x, double y, double z, double t)>
```

8.2.2 Variable Documentation

8.2.2.1 ZERO_FUNC

```
const Func ZERO_FUNC
```

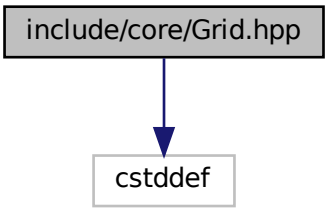
Initial value:

```
= [](double , double , double , double  = 0) {
    return 0.0;
}
```

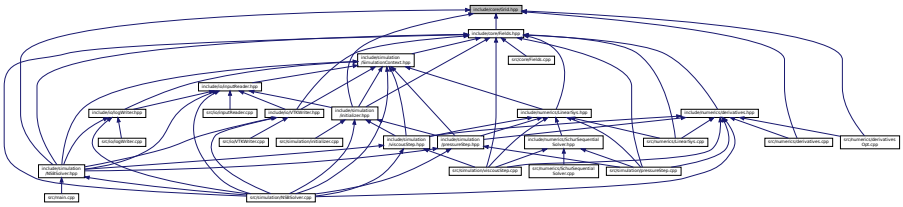
8.3 include/core/Grid.hpp File Reference

```
#include <cstddef>
```

Include dependency graph for Grid.hpp:



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



Classes

- struct Grid

  Struct to hold grid dimensions and spacing information along each direction.

Typedefs

- using GridPtr = std::shared_ptr< const struct Grid >

  Shared pointer type for Grid struct.

Enumerations

- enum class Axis { X = 0 , Y = 1 , Z = 2 }

  Enum class representing the three coordinate axes.
- enum class GridStaggering { CELL_CENTERED , FACE_CENTERED }

  Enum describing the (possibly staggered) position of some points in a grid.

Variables

- const size_t AXIS_COUNT = 3

8.3.1 Typedef Documentation

8.3.1.1 GridPtr

```
using GridPtr = std::shared_ptr<const struct Grid>
```

Shared pointer type for Grid struct.

8.3.2 Enumeration Type Documentation

8.3.2.1 Axis

```
enum Axis  [strong]
```

Enum class representing the three coordinate axes.

Enumerator

X	
Y	
Z	

8.3.2.2 GridStaggering

```
enum GridStaggering  [strong]
```

Enum describing the (possibly staggered) position of some points in a grid.

Enumerator

CELL_CENTERED	
FACE_CENTERED	

8.3.3 Variable Documentation

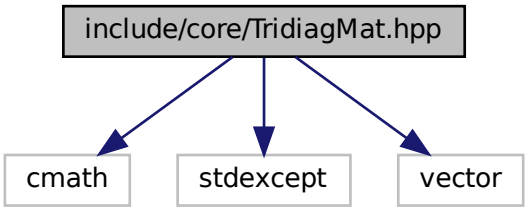
8.3.3.1 AXIS_COUNT

```
const size_t AXIS_COUNT = 3
```

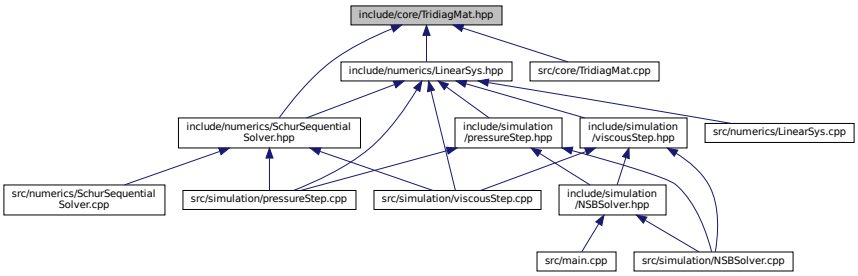
8.4 include/core/TridiagMat.hpp File Reference

```
#include <cmath>
#include <stdexcept>
#include <vector>
```

Include dependency graph for TridiagMat.hpp:



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



Classes

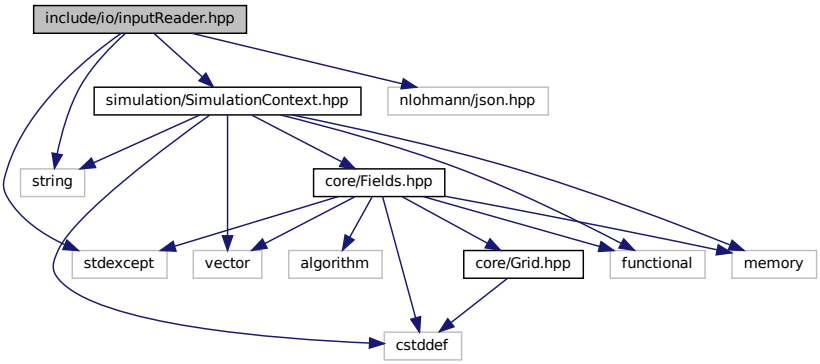
- class TridiagMat

  Class representing a tridiagonal matrix and providing access to its elements.

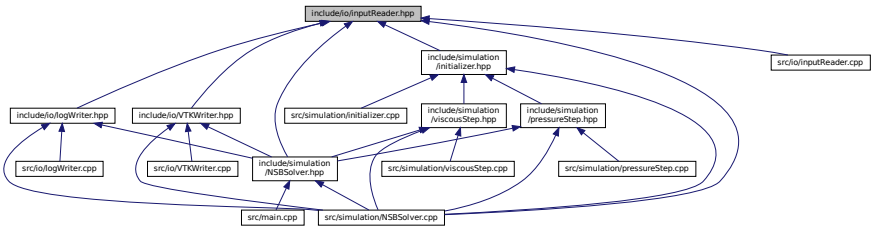
8.5 include/io/inputReader.hpp File Reference

```
#include <string>
#include <stdexcept>
#include <simulation/SimulationContext.hpp>
#include <nlohmann/json.hpp>
```

Include dependency graph for inputReader.hpp:



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



Classes

- struct MeshData

  Structure to hold mesh configuration data.
- struct PhysicsData

  Structure to hold physics parameters.
- struct TimeData

  Structure to hold time integration parameters.
- struct InputData

  Structure to hold all input data from configuration file.

Namespaces

- InputReader

Functions

- InputData InputReader::read (const std::string &filename)

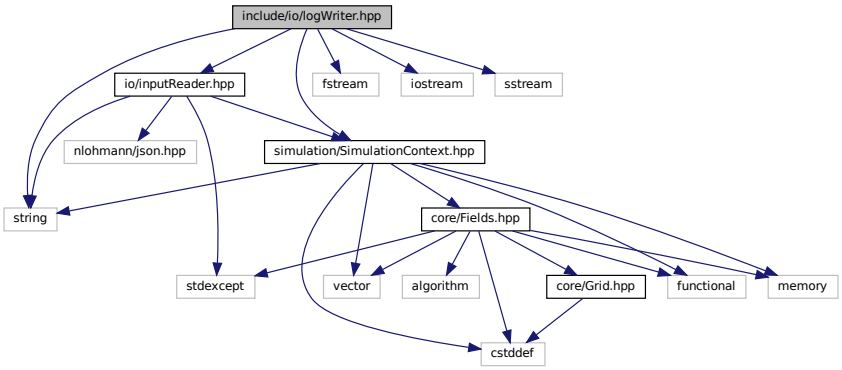
  Read and parse input data from a JSON configuration file.

8.6 include/io/logWriter.hpp File Reference

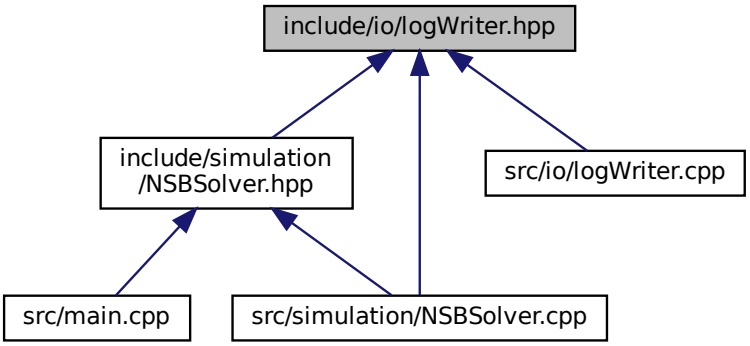
```
#include <string>
#include <fstream>
```

```
#include <iostream>
#include <sstream>
#include "io/inputReader.hpp"
#include "simulation/SimulationContext.hpp"
```

Include dependency graph for logWriter.hpp:



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



Classes

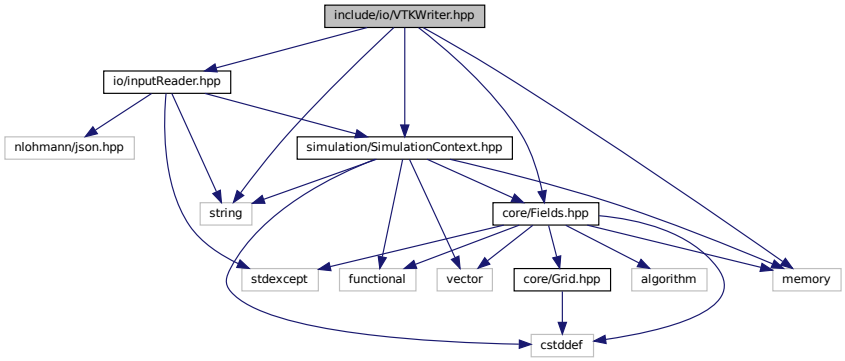
- class LogWriter

  Handles console and file logging.

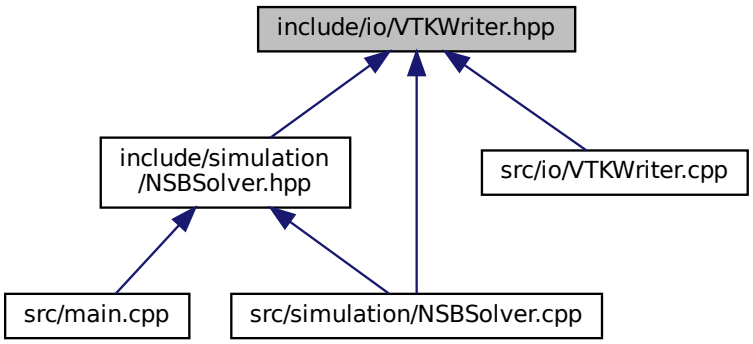
8.7 include/io/VTKWriter.hpp File Reference

```
#include <string>
#include <memory>
#include "core/Fields.hpp"
#include "io/inputReader.hpp"
#include "simulation/SimulationContext.hpp"
```

Include dependency graph for VTKWriter.hpp:



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



Classes

- class VTKWriter

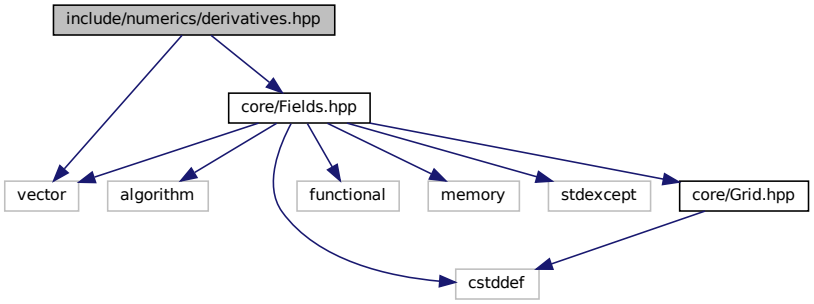
  Class to write 3D fields to legacy VTK (STRUCTURED_POINTS) for a uniform Cartesian grid, managing output frequency internally.

8.8 include/main.dox File Reference

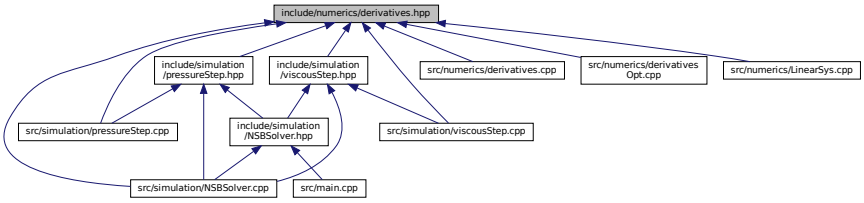
8.9 include/numerics/derivatives.hpp File Reference

```
#include <vector>
#include "core/Fields.hpp"
```

Include dependency graph for derivatives.hpp:



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



Classes

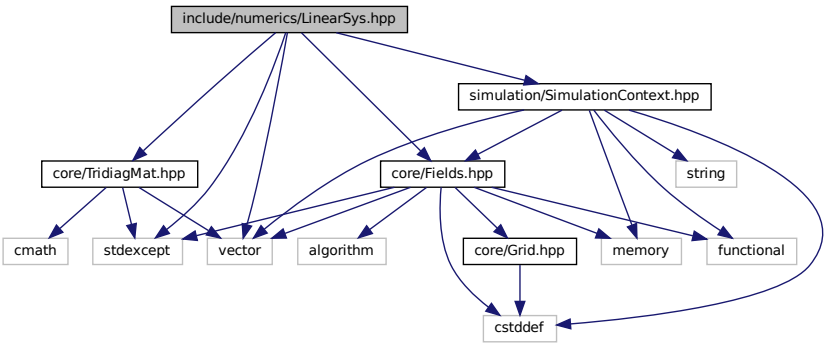
- class Derivatives

  Handler for computing spatial derivatives of scalar fields.

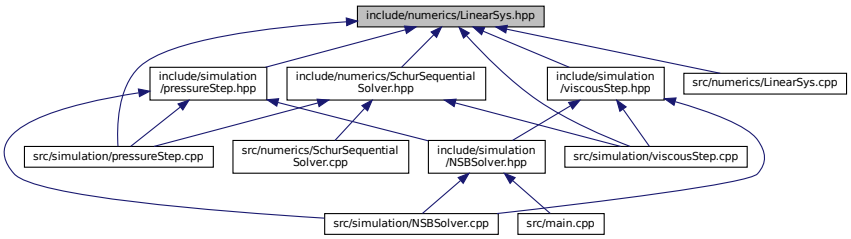
8.10 include/numerics/LinearSys.hpp File Reference

```
#include <vector>
#include <core/TridiagMat.hpp>
#include <core/Fields.hpp>
#include <stdexcept>
#include "simulation/SimulationContext.hpp"
```

Include dependency graph for LinearSys.hpp:



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



Classes

- class LinearSys

  Manages and solves a tridiagonal linear system (Ax = b) used for pressure and velocity steps.

Enumerations

- enum class BoundaryType { Normal , Tangent }

8.10.1 Enumeration Type Documentation

8.10.1.1 BoundaryType

```
enum BoundaryType  [strong]
```

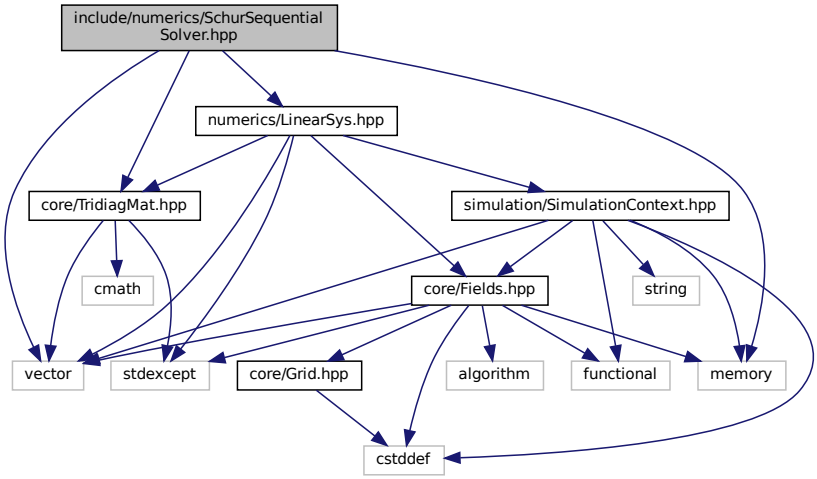
Enumerator

Normal	
Tangent	

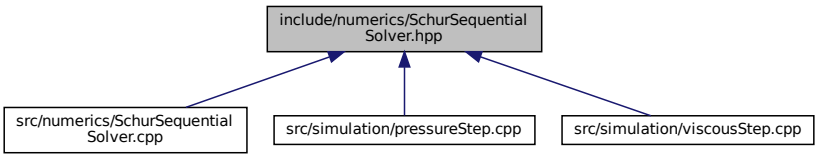
8.11 include/numerics/SchurSequentialSolver.hpp File Reference

```
#include <vector>
#include <memory>
#include "core/TridiagMat.hpp"
#include "numerics/LinearSys.hpp"
```

Include dependency graph for SchurSequentialSolver.hpp:



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



Classes

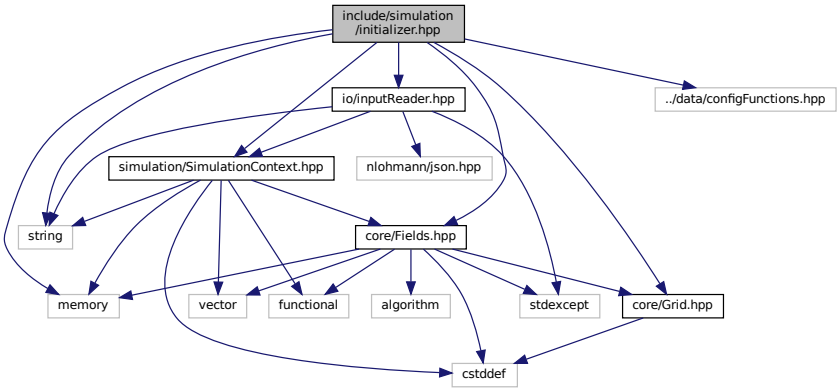
- class SchurSequentialSolver

  Implements the Schur Complement method for solving large tridiagonal systems by decomposing them into smaller sub-systems solved sequentially.

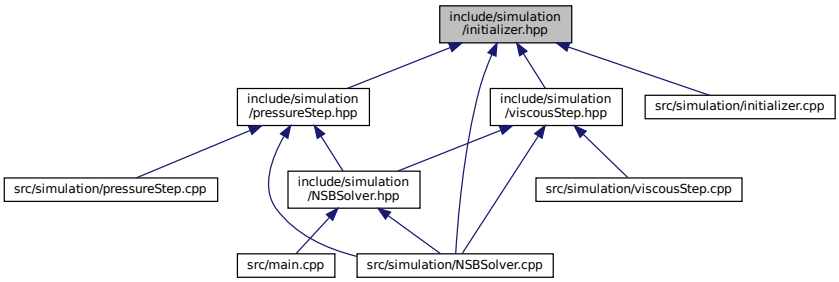
8.12 include/simulation/initializer.hpp File Reference

```
#include <memory>
#include <string>
#include "../data/configFunctions.hpp"
#include "io/inputReader.hpp"
#include "core/Grid.hpp"
#include "core/Fields.hpp"
#include "simulation/SimulationContext.hpp"
```

Include dependency graph for initializer.hpp:



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



Namespaces

- Initializer

Functions

- SimulationData Initializer::setup (const InputData &inputData)

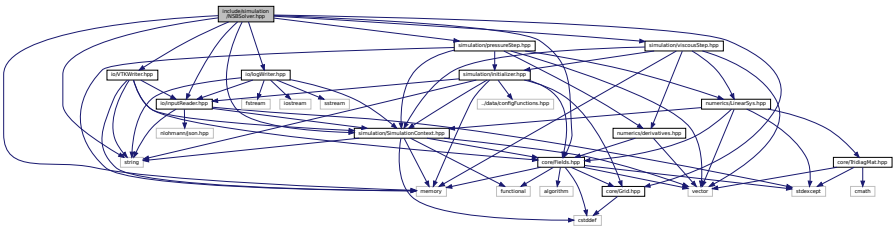
  Setup and initialize the SimulationData from input data.
- Field Initializer::initializeFieldFromFunc (const double time, const GridPtr &grid, const Func &func)
- VectorField Initializer::initializeVectorFieldFromFunc (const double time, const GridPtr &grid, const Func &func_u, const Func &func_v, const Func &func_w)

8.13 include/simulation/NSBSolver.hpp File Reference

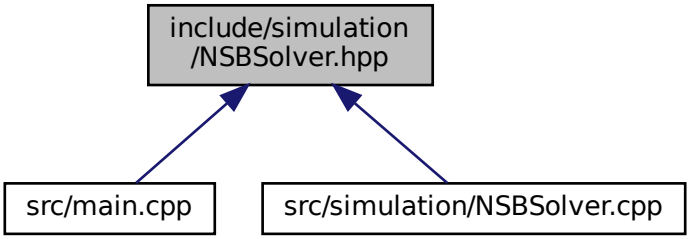
```
#include <memory>
#include <string>
#include "io/inputReader.hpp"
#include "io/VTKWriter.hpp"
#include "io/logWriter.hpp"
#include "simulation/pressureStep.hpp"
#include "simulation/viscousStep.hpp"
#include "core/Grid.hpp"
```

```
#include "core/Fields.hpp"
#include "simulation/SimulationContext.hpp"
```

Include dependency graph for NSBSolver.hpp:



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



Classes

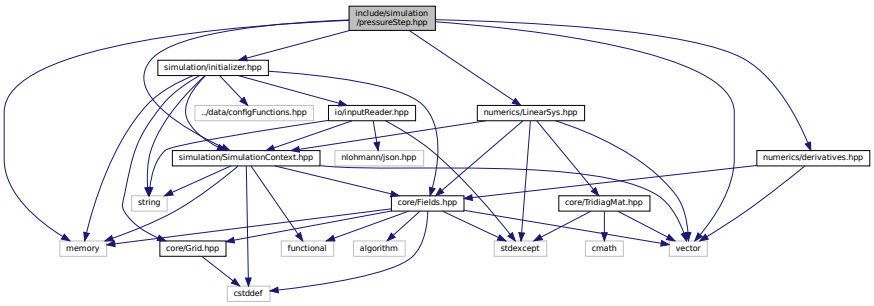
- class NSBSolver

  Class responsible for solving the Navier-Stokes-Brinkman equations.

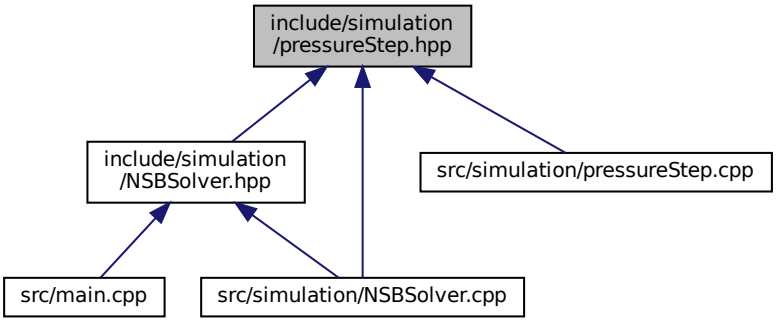
8.14 include/simulation/pressureStep.hpp File Reference

```
#include <vector>
#include <memory>
#include <simulation/initializer.hpp>
#include <numerics/LinearSys.hpp>
#include <numerics/derivatives.hpp>
#include <simulation/SimulationContext.hpp>
```

Include dependency graph for pressureStep.hpp:



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



Classes

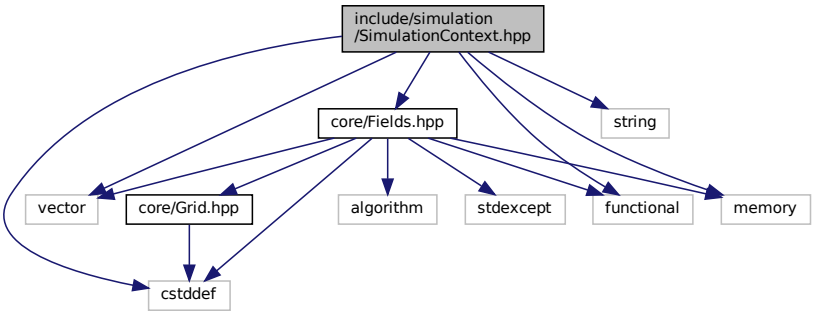
- class PressureStep

  Handles all pressure step maniplation. This class does not own data but regulates the workflow.

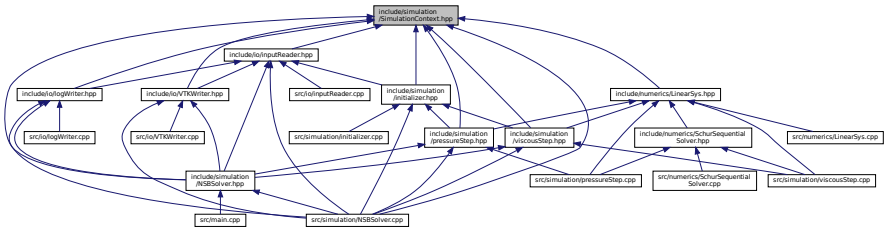
8.15 include/simulation/SimulationContext.hpp File Reference

```
#include <cstddef>
#include <string>
#include <vector>
#include <functional>
#include <memory>
#include "core/Fields.hpp"
```

Include dependency graph for SimulationContext.hpp:



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



Classes

- struct OutputSettings

  Structure holding configuration parameters for outputting simulation results.
- struct LoggingSettings

  Structure holding configuration parameters for simulation logging (console and file).
- struct ParallelizationSettings

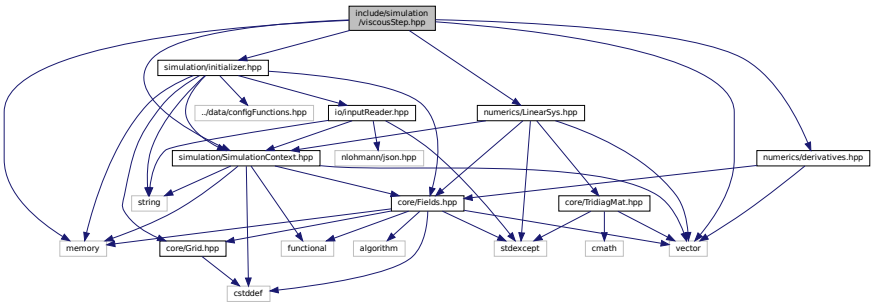
  Structure holding parameters related to parallel execution and domain decomposition.
- struct SimulationData

  Central structure containing all transient data, fields, and physical properties of the running simulation.

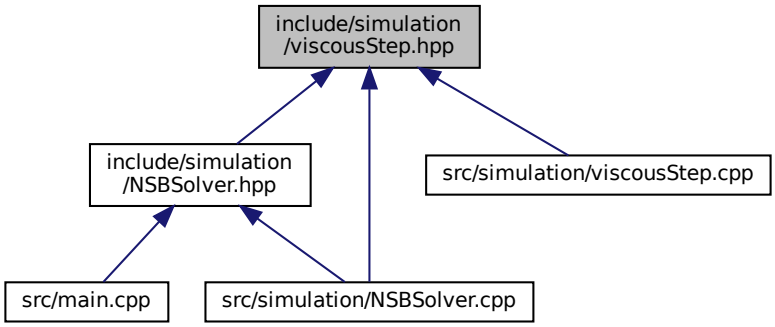
8.16 include/simulation/viscousStep.hpp File Reference

```
#include <vector>
#include <memory>
#include <simulation/initializer.hpp>
#include <numerics/LinearSys.hpp>
#include <numerics/derivatives.hpp>
#include <simulation/SimulationContext.hpp>
```

Include dependency graph for viscousStep.hpp:



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



Classes

- class ViscousStep

  Handles all viscous step maniplation. This class does not own data but regulates the workflow.

8.17 scripts/displayError.py File Reference

Namespaces

- displayError

Functions

- def displayError.u_ana (points, t, h)
- def displayError.v_ana (points, t, h)
- def displayError.w_ana (points, t, h)
- def displayError.p_ana (points, t)
- def displayError.get_step_from_filename (filename)
- def displayError.main ()

Variables

- string displayError.EXECUTABLE = "../build/main"
- string displayError.CONFIG_FILE = "../data/config.json"
- string displayError.OUTPUT_DIR = "../output/"
- displayError.ERROR_DIR = os.path.join(OUTPUT_DIR, "error_analysis")
- int displayError.NX = 50
- float displayError.DT = 0.001
- float displayError.T_END = 0.1
- int displayError.OUTPUT_FREQ = 1
- float displayError.DOMAIN_LEN_X = 6.0

8.18 scripts/plot_convergence.py File Reference

Namespaces

- plot_convergence

Functions

- int plot_convergence.get_step_from_filename (str filepath)
- np.ndarray plot_convergence.compute_analytical_u_at_x_faces (np.ndarray points, float t, float h)
- np.ndarray plot_convergence.compute_analytical_v_at_y_faces (np.ndarray points, float t, float h)
- np.ndarray plot_convergence.compute_analytical_w_at_z_faces (np.ndarray points, float t, float h)
- np.ndarray plot_convergence.compute_analytical_p_at_centers (np.ndarray points, float t)
- float plot_convergence.calculate_l2_rms (np.ndarray data)
- float plot_convergence.calculate_l2_rms_error (np.ndarray field_numerical, np.ndarray field_analytical)
- def plot_convergence.plot_convergence (list h_values, dict error_data, str title, str save_filename)
- def plot_convergence.run_analysis (custom_simulations=None)

Variables

- list plot_convergence.SIMULATIONS
- int plot_convergence.DOMAIN_LENGTH_X = 6
- float plot_convergence.DT = 0.001
- string plot_convergence.VELOCITY_FIELD_NAME = "velocity"
- string plot_convergence.PRESSURE_FIELD_NAME = "pressure"

8.19 scripts/plot_convergence_new.py File Reference

Namespaces

- plot_convergence_new

Functions

- int plot_convergence_new.get_step_from_filename (str filepath)
- np.ndarray plot_convergence_new.compute_analytical_u_at_x_faces (np.ndarray points, float t, float h)
- np.ndarray plot_convergence_new.compute_analytical_v_at_y_faces (np.ndarray points, float t, float h)
- np.ndarray plot_convergence_new.compute_analytical_w_at_z_faces (np.ndarray points, float t, float h)
- np.ndarray plot_convergence_new.compute_analytical_p_at_centers (np.ndarray points, float t)
- float plot_convergence_new.calculate_l2_rms (np.ndarray data)
- float plot_convergence_new.calculate_l2_rms_error (np.ndarray field_numerical, np.ndarray field_analytical)
- def plot_convergence_new.plot_convergence (list x_values, dict error_data, str title, str save_filename, str x_label, int expected_order)
- def plot_convergence_new.run_analysis (custom_simulations=None)

Variables

- list plot_convergence_new.SIMULATIONS
- int plot_convergence_new.DOMAIN_LENGTH_X = 6
- float plot_convergence_new.DT = 0.001
- string plot_convergence_new.VELOCITY_FIELD_NAME = "velocity"
- string plot_convergence_new.PRESSURE_FIELD_NAME = "pressure"

8.20 scripts/run_convergence_study.py File Reference

Namespaces

- run_convergence_study

Functions

- def run_convergence_study.main ()

Variables

- string run_convergence_study.EXECUTABLE_PATH = "../build/main"
- string run_convergence_study.CONFIG_PATH = "../data/config.json"
- string run_convergence_study.OUTPUT_DIR = "../output/"
- int run_convergence_study.EXPECTED_LAST_STEP = 30
- list run_convergence_study.NX_VALUES = [20, 25, 30]

8.21 scripts/run_convergence_study_new.py File Reference

Namespaces

- run_convergence_study_new

Functions

- def run_convergence_study_new.calculate_params_for_mode (loop_val, base_dt, study_mode)
- def run_convergence_study_new.main ()

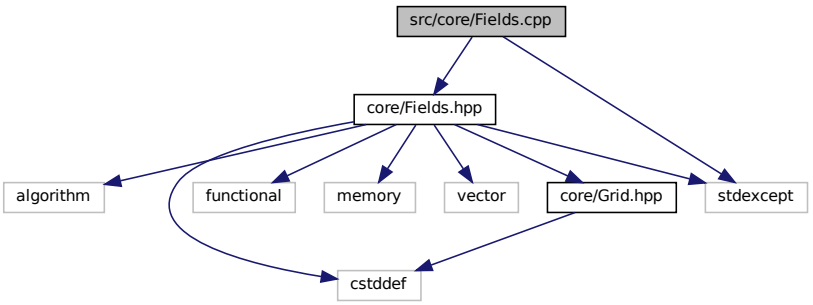
Variables

- string run_convergence_study_new.EXECUTABLE_PATH = "../build/main"
- string run_convergence_study_new.CONFIG_PATH = "../data/config.json"
- string run_convergence_study_new.OUTPUT_DIR = "../output/"
- string run_convergence_study_new.STUDY_MODE = "TEMPORAL_ONLY"
- int run_convergence_study_new.BASE_NX = 20
- float run_convergence_study_new.BASE_DT = 0.001
- float run_convergence_study_new.T_END = 0.03
- list run_convergence_study_new.NX_VALUES_FOR_SPATIAL_STUDY = [20, 25, 30, 35, 40, 45, 50, 60, 80]
- list run_convergence_study_new.DT_MULTIPLIERS_FOR_TEMPORAL_STUDY = [1.0, 0.5, 0.25, 0.125, 0.0625]
- int run_convergence_study_new.FIXED_NX_FOR_TEMPORAL_STUDY = 50

8.22 src/core/Fields.cpp File Reference

```
#include "core/Fields.hpp"
#include <stdexcept>
```

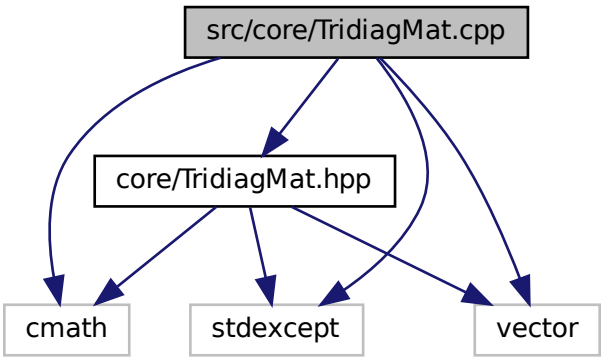
Include dependency graph for Fields.cpp:



8.23 src/core/TridiagMat.cpp File Reference

```
#include "core/TridiagMat.hpp"
#include <cmath>
#include <stdexcept>
#include <vector>
```

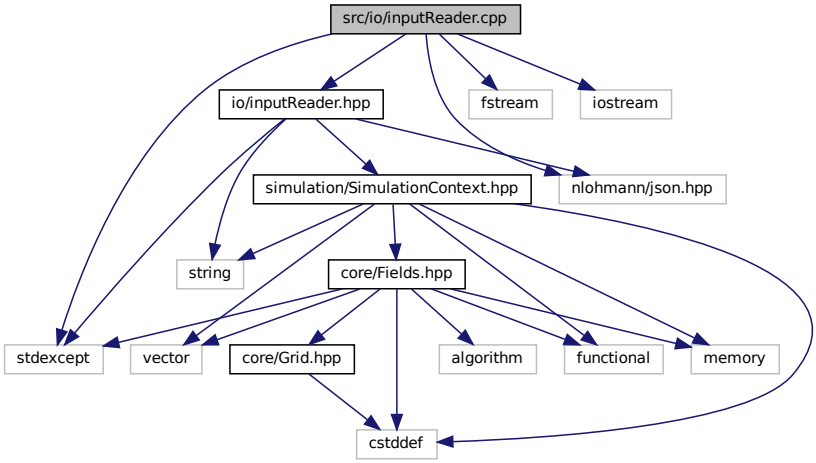
Include dependency graph for TridiagMat.cpp:



8.24 src/io/inputReader.cpp File Reference

```
#include "io/inputReader.hpp"
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <nlohmann/json.hpp>
```

Include dependency graph for inputReader.cpp:



Typedefs

- using json = nlohmann::json

8.24.1 Typedef Documentation

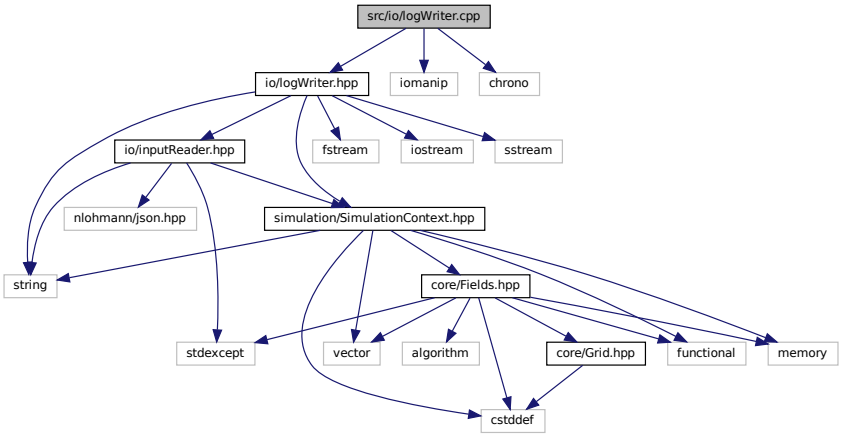
8.24.1.1 json

```
using json = nlohmann::json
```

8.25 src/io/logWriter.cpp File Reference

```
#include "io/logWriter.hpp"
#include <iomanip>
#include <chrono>
```

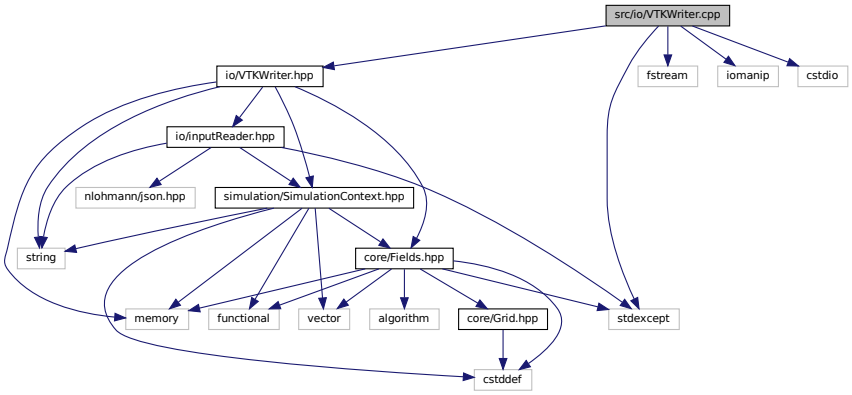
Include dependency graph for logWriter.cpp:



8.26 src/io/VTKWriter.cpp File Reference

```
#include "io/VTKWriter.hpp"
#include <fstream>
#include <iomanip>
#include <stdexcept>
#include <cstdio>
```

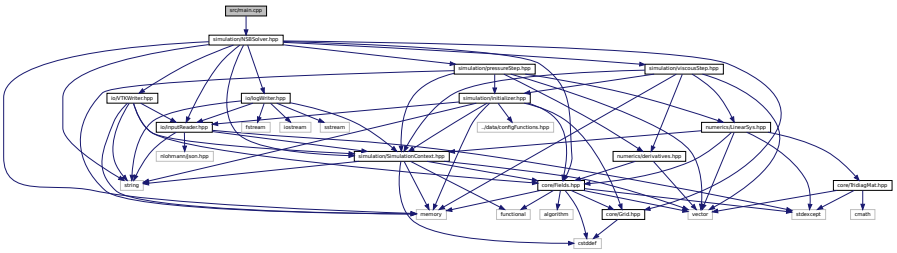
Include dependency graph for VTKWriter.cpp:



8.27 src/main.cpp File Reference

```
#include "simulation/NSBSolver.hpp"
```

Include dependency graph for main.cpp:



Functions

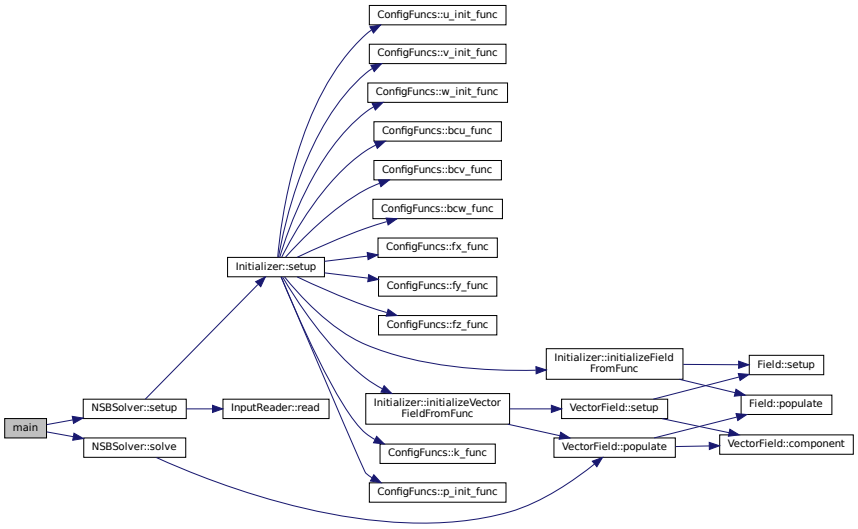
- int main ()

8.27.1 Function Documentation

8.27.1.1 main()

```
int main ( )
```

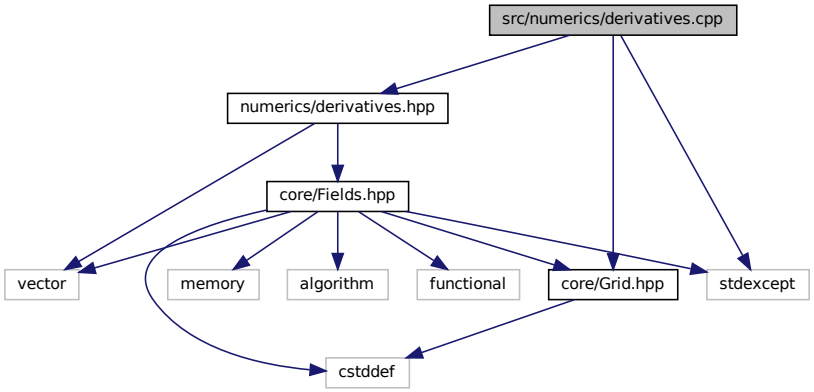
Here is the call graph for this function:



8.28 src/numerics/derivatives.cpp File Reference

```
#include "numerics/derivatives.hpp"
#include "core/Grid.hpp"
#include <stdexcept>
```

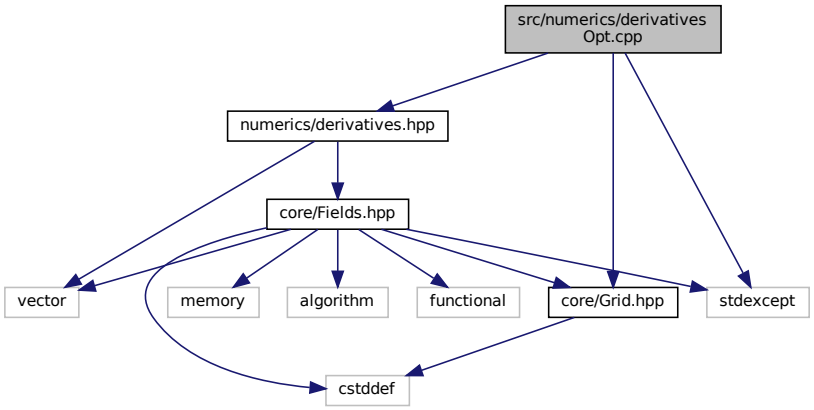
Include dependency graph for derivatives.cpp:



8.29 src/numerics/derivativesOpt.cpp File Reference

```
#include "numerics/derivatives.hpp"
#include "core/Grid.hpp"
#include <stdexcept>
```

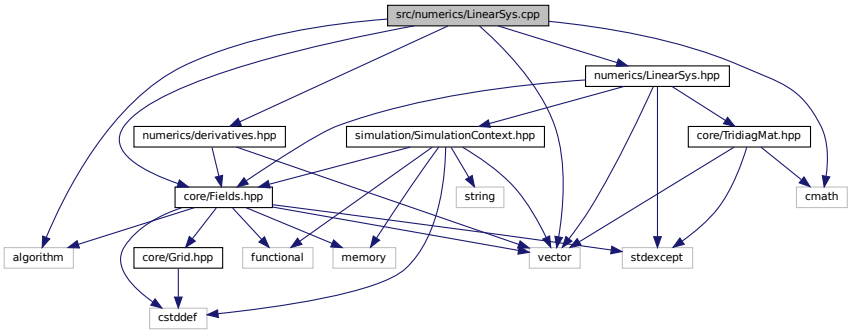
Include dependency graph for derivativesOpt.cpp:



8.30 src/numerics/LinearSys.cpp File Reference

```
#include <algorithm>
#include <cmath>
#include <core/Fields.hpp>
#include <numerics/LinearSys.hpp>
#include <numerics/derivatives.hpp>
#include <vector>
```

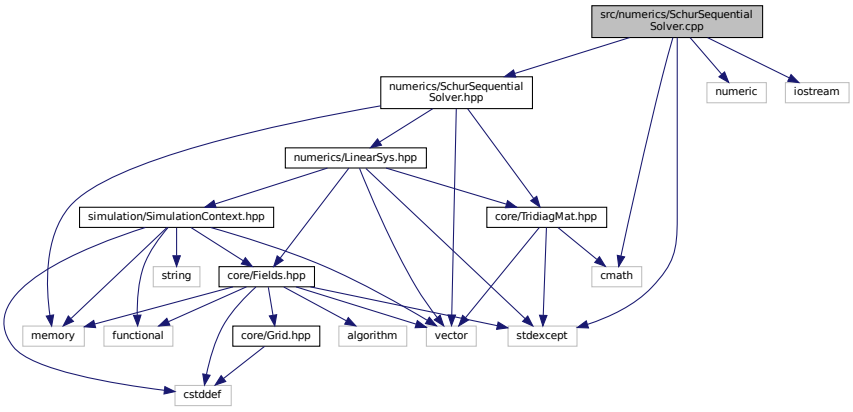
Include dependency graph for LinearSys.cpp:



8.31 src/numerics/SchurSequentialSolver.cpp File Reference

```
#include "numerics/SchurSequentialSolver.hpp"
#include <stdexcept>
#include <numeric>
#include <cmath>
#include <iostream>
```

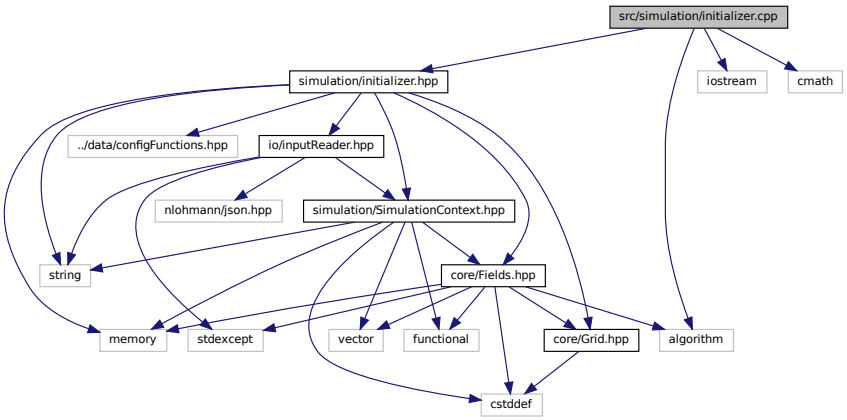
Include dependency graph for SchurSequentialSolver.cpp:



8.32 src/simulation/initializer.cpp File Reference

```
#include "simulation/initializer.hpp"
#include <iostream>
#include <cmath>
#include <algorithm>
```

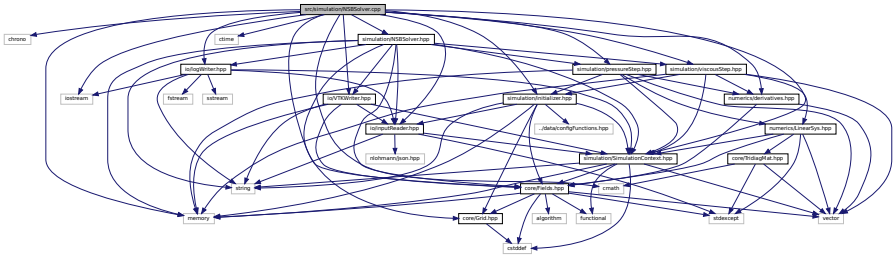
Include dependency graph for initializer.cpp:



8.33 src/simulation/NSBSolver.cpp File Reference

```
#include <chrono>
#include <cmath>
#include <iostream>
#include <memory>
#include <ctime>
#include "simulation/NSBSolver.hpp"
#include "core/Fields.hpp"
#include "io/inputReader.hpp"
#include "io/VTKWriter.hpp"
#include "io/logWriter.hpp"
#include "numerics/derivatives.hpp"
#include "simulation/pressureStep.hpp"
#include "simulation/SimulationContext.hpp"
#include "simulation/viscousStep.hpp"
#include "simulation/initializer.hpp"
```

Include dependency graph for NSBSolver.cpp:

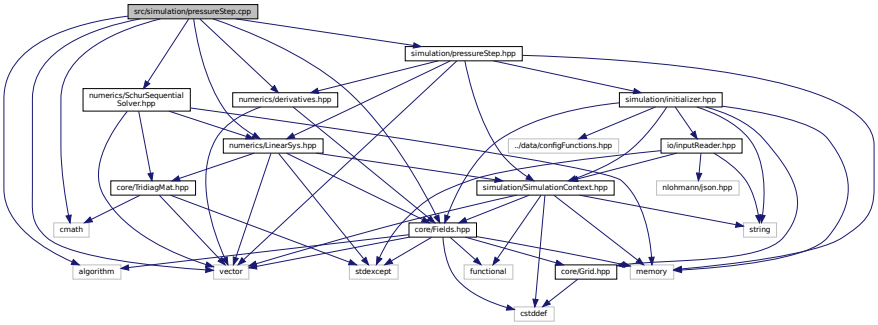


8.34 src/simulation/pressureStep.cpp File Reference

```
#include <algorithm>
#include <cmath>
#include <vector>
#include <core/Fields.hpp>
#include <numerics/derivatives.hpp>
```

```
#include <numerics/LinearSys.hpp>
#include "numerics/SchurSequentialSolver.hpp"
#include <simulation/pressureStep.hpp>
```

Include dependency graph for pressureStep.cpp:



8.35 src/simulation/viscousStep.cpp File Reference

```
#include <algorithm>
#include <cmath>
#include <core/Fields.hpp>
#include <vector>
#include <numerics/derivatives.hpp>
#include <numerics/LinearSys.hpp>
#include "numerics/SchurSequentialSolver.hpp"
#include <simulation/viscousStep.hpp>
```

Include dependency graph for viscousStep.cpp: