Firedrake-Fluids Documentation

Release 0.2-dev

Imperial College London

May 28, 2015

Contents

Contents:

CHAPTER 1

Introduction

1.1 Overview

Firedrake-Fluids is a collection of finite element-based numerical models for the study of fluid dynamical systems. It uses the Firedrake framework to automate the solution of the governing equations written in their weak form using the high-level, compact, near-mathematical Unified Form Language (UFL). The complexity of writing a numerical model is hidden through abstraction, and model developers do not need to concern themselves with hand-writing the low-level (e.g. C or Fortran) code required to solve the equations; this is all derived and optimised automatically from a high-level specification. Furthermore, model developers do not need to be experts in parallel programming to enable their code to be portable across different hardware architectures (e.g. a cluster of multi-core CPUs, or a single GPU); the generated code is targetted, compiled and executed automatically on a desired architecture using the PyOP2 library with which Firedrake is coupled.

Some information briefly outlining Firedrake's automated solution technique and the setup of Firedrake-Fluids can be found in the sections following this one. The remaining chapters provide details on the models available within Firedrake-Fluids, along with any auxiliary parameterisations that the user may wish to include. In addition, information regarding how to set up a model is also given.

1.1.1 Automated solution technique

When a model in Firedrake-Fluids is executed by the Python interpreter, the model's UFL (along with the computational mesh used to discretise the domain) is first passed to the Firedrake framework. Within this framework, the UFL is first converted to an abstract syntax tree (AST) by a modified version of the FEniCS Form Compiler (FFC). Additionally, the topology of the mesh is described by a PETSc DMPlex object to allow the efficient execution of the generated code over the whole mesh. The DMPlex object and the AST are then passed to the PyOP2 library which, after the AST has been optimised by the COFFEE compiler and converted into low-level generated C code, targets and compiles the generated code towards a specific hardware architecture and executes it on that hardware.

As an example, consider the UFL statement in Figure *ufl_expression*.

```
c_initial = Expression('''exp(-( pow(x[0]-x0, 2)/(2*pow(spread_x, 2)) +
                        pow(x[1]-y0, 2)/(2*pow(spread_y, 2)) ))''',
                        x0=0.5, y0=0.5, spread_x=0.075, spread_y=0.075)
```

Fig. 1.1: An example of a UFL expression.

This one single line of UFL is converted to a kernel comprising many lines of generated C code, which perform the evaluation of the expression, as shown in Figure *c_kernel*.

```
void expression_kernel (double A[3] , double** x_ , double * spread_x_ ,
                        double * spread_y_ , double * x0_ , double * y0_ )
{
  const double spread_x = *spread_x_;const double spread_y = *spread_y_;
  const double x0 = *x0_;const double y0 = *y0_;
  const double X[3][3] = {{1.0,2.77555756156e-17,0.0},
  {-2.77555756156e-17,1.0,0.0},
  {0.0,5.55111512313e-17,1.0}};

  double x[2];
  const double pi = 3.141592653589793;

  {
    #pragma pyop2 itspace
    for (int k  = 0; k < 3; ++k)
    {

      for (unsigned int d=0; d < 2; d++) {
        x[d] = 0;
        for (unsigned int i=0; i < 3; i++) {
          x[d] += X[k][i] * x_[i][d];
        };
      };

      A[k] = exp(-( pow(x[0]-x0, 2)/(2*pow(spread_x, 2)) +
                              pow(x[1]-y0, 2)/(2*pow(spread_y, 2)) ));

    }

  }

}
```

Fig. 1.2: An example of C code, generated automatically, for the purpose of evaluating an expression defined by a high-level, near-mathematical UFL statement.

1.2 Directory structure

The directory structure of the Firedrake-Fluids codebase is as follows:

- `/`: The Firedrake-Fluids base directory contains general information in the README file, information about the license in the COPYING file, and a full list of authors in the AUTHORS file.
- `/docs`: Contains the source code and images for this documentation.
- `/firedrake_fluids`: Comprises a collection of Python files containing the implementation of the different models and auxiliary functionality.
- `/schema`: Contains a set of schema files used to define the different options a simulation configuration file can take (see Section [sect:configuring$_{as}$imulation] for more details).
- `/tests`: A set of test cases to help ensure the correctness of the models.

1.3 Setup

1.3.1 Dependencies

Before running the models in Firedrake-Fluids, please ensure that all the dependencies specified in the README file are satisfied. Installations for Firedrake (and its dependencies) can be found here. Firedrake-Fluids also relies on the libspud library (and the Python bindings) to retrieve simulation-related options (e.g. the time-step size and initial conditions) from a configuration/setup file. Following the steps below at the command line will download and build libspud, and install the Python bindings:

```
bzr checkout lp:~spud/spud/trunk libspud
cd libspud
./configure
make
cd python
sudo python setup.py install
```

1.3.2 Installation

The Firedrake-Fluids Python module can be installed with:

```
sudo python setup.py install
```

Alternatively, the `firedrake_fluids` directory may be added to the `PYTHONPATH` environment variable in order to use the module. This can be done at the command line, e.g.:

```
export PYTHONPATH=$PYTHONPATH:/home/christian/firedrake-fluids/firedrake_fluids
```

Following this, it is recommended that you run `make test` (from the Firedrake-Fluids base directory) to ensure that the setup and models are working correctly.

CHAPTER 2

Shallow water model

The shallow water model solves the non-linear, non-rotational shallow water equations which describe the dynamics of a free surface and a depth-averaged velocity field. For modelling purposes, the free surface is split up into a mean component H (i.e. the hydrostatic depth to the seabed) and a perturbation component h (see Figure *shallow_water_setup*).

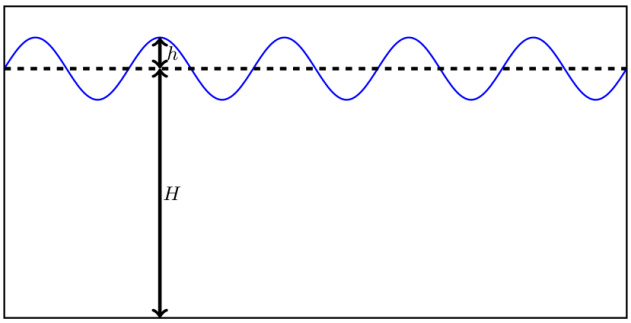

Fig. 2.1: Single-layer shallow water set-up.

2.1 Model equations

The shallow water equation set comprises a momentum equation and a continuity equation, each of which are defined below. These are defined on a domain Ω and for a time $t \in [0, T]$.

2.1.1 Momentum equation

The momentum equation is solved in non-conservative form such that

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -g\nabla h + \nabla \cdot \mathbb{T} - C_D \frac{||\mathbf{u}||\mathbf{u}}{(H+h)},$$

where g is the acceleration due to gravity, $\mathbf{u}$ is the velocity, and C_D is the non-dimensional drag coefficient. The stress tensor $\mathbb{T}$ is given by

$$\mathbb{T} = \nu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathrm{T}}\right) - \frac{2}{3}\nu \left(\nabla \cdot \mathbf{u}\right) \mathbb{I},$$

where ν is the isotropic kinematic viscosity, and $\mathbb{I}$ is the identity tensor.

2.1.2 Continuity equation

The continuity equation is given by

$$\frac{\partial h}{\partial t} + \nabla \cdot ((H + h)\,\mathbf{u}) = 0.$$

2.1.3 Discretisation and solving

The model equations are discretised using a Galerkin finite element method. Essentially, this begins by deriving the weak form of the equations by multiplying through by a test function $\mathbf{w} \in H^1(\Omega)^3$ (where $H^1(\Omega)^3$ is the first Hilbertian Sobolev space) and integrating over Ω. In the case of the momentum equation, this becomes

$$\int_\Omega \mathbf{w} \cdot \frac{\partial \mathbf{u}}{\partial t}\,\mathrm{dV} + \int_\Omega \mathbf{w} \cdot (\mathbf{u} \cdot \nabla \mathbf{u})\,\mathrm{dV} = -\int_\Omega g\mathbf{w} \cdot \nabla h\,\mathrm{dV} - \int_\Omega \nabla \mathbf{w} \cdot \mathbb{T}\,\mathrm{dV} - \int_\Omega C_D \mathbf{w} \cdot \frac{||\mathbf{u}||\mathbf{u}}{(H+h)}\,\mathrm{dV}.$$

A solution $\mathbf{u} \in H^1(\Omega)^3$ is sought such that it is valid $\forall \mathbf{w}$.

The solution fields $\mathbf{u}$ and h are each represented by a set of interpolating basis functions, such that

$$\mathbf{w} = \sum_{i=1}^{N_{\mathrm{u\_nodes}}} \phi_i \mathbf{w}_i,$$

$$\mathbf{u} = \sum_{i=1}^{N_{\mathrm{u\_nodes}}} \phi_i \mathbf{u}_i,$$

and

$$h = \sum_{i=1}^{N_{\mathrm{h\_nodes}}} \psi_i h_i,$$

where ϕ_i and ψ_i are the basis functions representing the velocity and free surface perturbation fields, respectively; $N_{\mathrm{u\_nodes}}$ and $N_{\mathrm{h\_nodes}}$ are the number of velocity and free surface solution nodes, respectively; and the coefficients $\mathbf{u}_i$ and h_i are to be found by a solution method. If the basis functions ϕ_i are continuous across each cell/element in the mesh, then the method is known as a *continuous* Galerkin (CG) method, whereas if the basis functions are discontinuous, then the method is known as a *discontinuous* Galerkin (DG) method.

The momentum equation, discretised in space, then becomes a matrix system:

$$\mathbf{M}\frac{\partial \mathbf{u}}{\partial t} + \mathbf{A}(\mathbf{u})\mathbf{u} = -\mathbf{C}h - \mathbf{K}\mathbf{u} - \mathbf{D}(\mathbf{u}, h)\mathbf{u},$$

where $\mathbf{M}$, $\mathbf{A}$, $\mathbf{K}$, $\mathbf{C}$ and $\mathbf{D}$ are the mass, advection, stress, gradient and drag matrices, respectively.

The time-derivative is discretised using the implicit backward Euler method, yielding a fully discrete system of equations:

$$\mathbf{M}\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \mathbf{A}(\mathbf{u}^{n+1})\mathbf{u}^{n+1} = -\mathbf{C}h^{n+1} - \mathbf{K}\mathbf{u}^{n+1} - \mathbf{D}(\mathbf{u}^{n+1}, h^{n+1})\mathbf{u}^{n+1},$$

where Δt is the time-step.

The finite element method is also applied to the continuity equation, which must be solved along with the momentum equation, yielding a block-coupled system. In Firedrake-Fluids, this system is preconditioned using a fieldsplit preconditioner and solved with the GMRES linear solver.

2.2 Configuring a simulation

The configuration/setup of a shallow water simulation in Firedrake-Fluids is defined in a Shallow Water Markup Language (.swml) file. This is essentially an XML file that contains tags/elements which are specific to the context of a shallow water simulation. The full range of possible options that are available to the user are defined by a set of schema files in the `schemas` directory; these can be thought of as 'templates' from which an .swml setup file can be constructed.

Creating a shallow water setup/configuration file is best done using the Diamond graphical user interface (GUI) that is supplied with the libspud dependency. At the command line, from the Firedrake-Fluids base directory, creating an .swml file called `example.swml` can be done using:

```
diamond -s schemas/shallow_water.rng example.swml
```

Note that the -s flag is used to specify the location of the schema file `shallow_water.rng`, while the final command line argument is the name of the setup file we want to create. The Diamond GUI will look something like the one shown in Figure *diamond*.

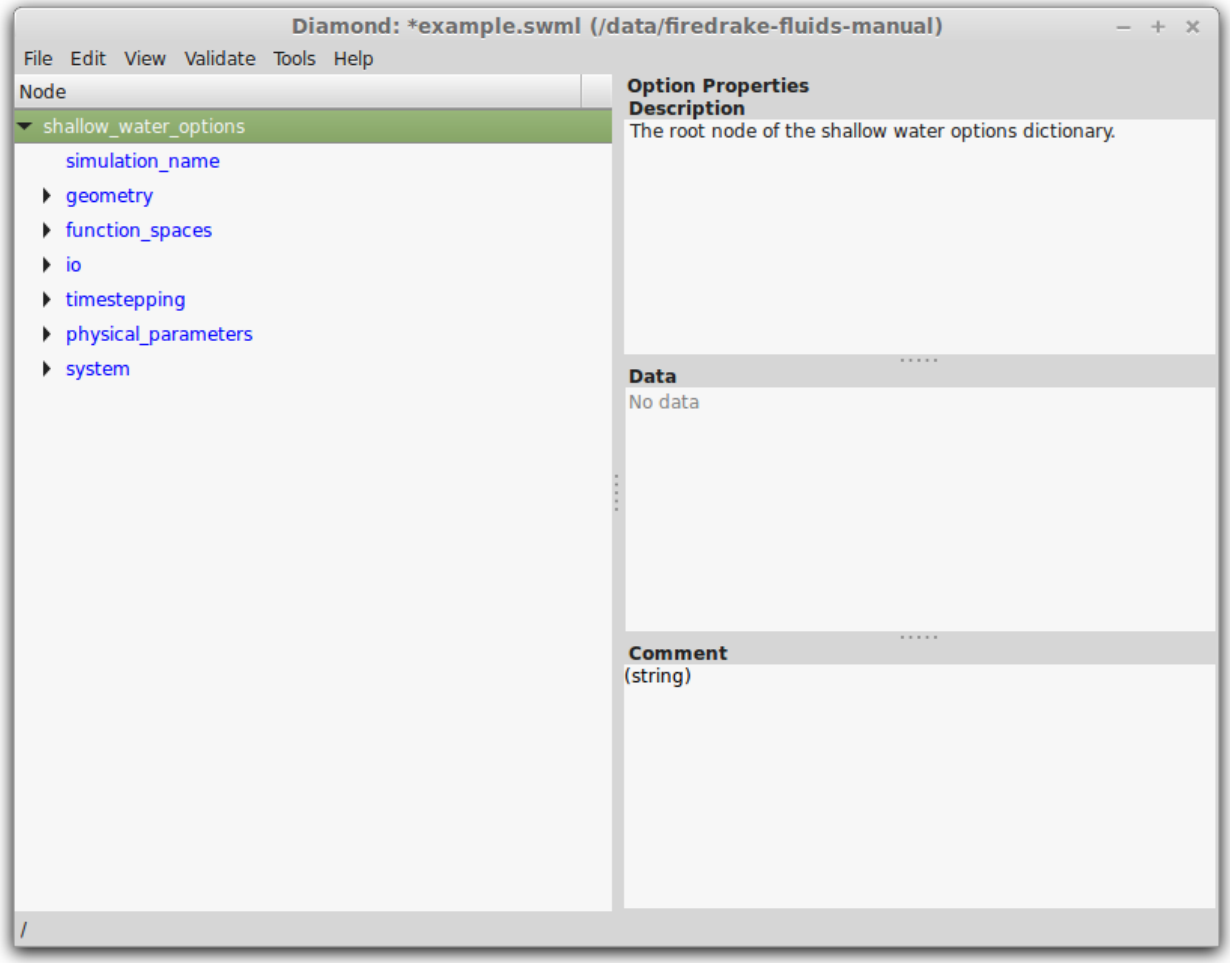

Fig. 2.2: The Diamond graphical user interface. Notice that all the available options are currently in blue; this means that they still need to be specified the user, after which the font colour will turn black.

Details of each of the options (and sub-options underneath, displayed by clicking the black arrows) are given in the

following sub-sections.

2.2.1 Simulation name

All simulations must be given a name under `/simulation_name`. This name is used when outputting solution files created during the simulation. Please use alpha-numeric characters and avoid using non-standard characters such as ampersands, commas, semi-colons, etc here.

2.2.2 Geometry

The `/geometry` section of the setup file concerns the dimension of the problem, and the location of the computational mesh used to discretise the domain.

The dimension should be one of the first options to be set. Be careful here; this option can only be set once because other options further down the list rely on it.

In the case of the mesh file location, note that only Gmsh `.msh` files are supported.

2.2.3 Function spaces

Since two fields, velocity $\mathbf{u}$ and free surface perturbation h, have to be solved for in the shallow water model, two function spaces may be specified. In Firedrake-Fluids, the function spaces are assumed to be composed of Lagrange polynomial basis functions. The order of these polynomials can be specified in the `degree` sub-option of each `function_space`. The `family` refers to whether the basis functions are continuous or discontinuous across the cells/elements of the mesh.

2.2.4 Input/output (I/O)

Solution files may be dumped at specific intervals, specified in time units. Setting the `io/dump_period` option to zero will result in dumps at every time-step. Note that solution files can currently only be written in VTU format (see http://www.vtk.org for more information).

Users can also enable checkpointing which allows them to resume the simulation at a later time. The checkpoint data will be written to a file called `checkpoint.npz`. The time interval between checkpoint dumps can be specified under `io/checkpoint/dump_period`. The simulation can be later resumed by specifying the location of this file with the `-c` flag (see *Running a simulation* for more details).

2.2.5 Timestepping

The time-step Δt and finish time T are specified under `timestepping/timestep` and `timestepping/finish_time`, respectively. The `timestepping/start_time` (i.e. the initial value of t) is usually set to zero.

For simulations which are known to converge to a steady-state, Firedrake-Fluids can stop the simulation when the maximum difference of all solution fields (i.e. $\mathbf{u}$ and h) between time n and $n+1$ becomes less than a user-defined tolerance; this is specified in `timestepping/steady_state/tolerance`.

2.2.6 Physical parameters

The only physical parameter applicable to the equation set solved in the Firedrake-Fluids shallow water model is the acceleration due to gravity. This is approximately 9.8 ms^{-2} on Earth.

2.2.7 System: Core fields

The model requires three fields to be set up under the `/system/core_fields` section of the setup file. These are the key fields used in shallow water simulations, and are named

- *Velocity* (a prognostic field, corresponding to $\mathbf{u}$).
- *FreeSurfacePerturbation* (a prognostic field, corresponding to h)
- *FreeSurfaceMean* (a prescribed field, corresponding to H)

It is here that the initial and boundary conditions for the fields can be specified. These can either be constant values, or values defined by a C++ expression.

C++ expressions

Non-constant values for initial and boundary conditions can be specified under the `cpp` sub-option; here, a Python function needs to be written which returns a string containing a C++ expression. An example is given in Figure *cpp_expression*.

```
def val(t):
  return "-4*sin(pi*(((4*t)/86400.0) + 0.5))"
```

Fig. 2.3: An example of a Python function returning a string containing a C++ expression. This C++ expression is used to define the non-constant values of a boundary condition. The function must be called `val` and have the argument `t`, which is the current simulation time that may be included in the C++ expression. The variable `x` contains the coordinates of the domain (i.e. `x[0]`, `x[1]` and `x[2]` are the x, y, and z coordinates, respectively).

Boundary conditions

A new boundary condition can be created for a given field by clicking the + button next to the gray `boundary_condition` option. Each boundary condition must be given a unique name.

The surfaces on which the boundary conditions need to be applied should be specified under `boundary_condition/surface_ids`; multiple surface IDs should be separated by a single space. The type of boundary condition should then be specified along with its value; the available types are (for velocity):

- *Dirichlet*: Strong Dirichlet boundary conditions can be enforced for both the FreeSurfacePerturbation and Velocity fields by selecting the `dirichlet` type.
- *No-normal flow*: Imposing a no-normal flow condition for velocity (i.e. $\mathbf{u} \cdot \mathbf{n} = 0$) can currently only be done weakly by integrating the continuity equation by parts (by enabling the `/system/equations/continuity_equation/integrate_by_parts` option) and selecting the `no_normal_flow` boundary condition type.
- *Flather*: A open boundary condition can be imposed weakly by integrating the continuity equation by parts and selecting the `flather` boundary condition type in the configuration options. This boundary condition enforces:

$$\mathbf{u} = \mathbf{u}_{\text{exterior}} + \sqrt{\frac{g}{H}} \left(h - h_{\text{exterior}}\right),$$

  where $\mathbf{u}_{\text{exterior}}$ and h_{exterior} are the known/expected values for velocity and the free surface perturbation. Any difference between the exterior values and the simulated values along the boundary is allowed out of the domain in such a way that minimises spurious reflections. Note that the implementation currently assumes that $\mathbf{u} \cdot \mathbf{n} \geq 0$ along the outflow (e.g. the outflow is through the north or east boundaries, for unit square domains).

For the free surface perturbation field h, only Dirichlet boundary conditions are available.

2.2.8 System: Equations

As already described in *Model equations*, there are two equations which make up the shallow water model: the momentum equation and the continuity equation. Options for both of these fields, concerning their discretisation and parameters (e.g. for C_D and ν), can be found under `/system/equations/momentum_equation` and `/system/equations/continuity_equation`.

Spatial discretisation

The spatial discretisation (continuous or discontinuous Galerkin) currently depends on the continuity of the function spaces in use, rather than on the choices made in this option. However, if `continuous_galerkin` is selected, there are stabilisation-related sub-options available to stabilise the advection term when using CG. See Stabilisation methods for more information on the stabilisation schemes available.

Mass term

An option is available to exclude the mass term in the momentum (or continuity) equation, under `../mass_term/exclude_mass_term`.

Advection term

An option is available to exclude the advection term in the momentum (or continuity) equation, under `../advection_term/exclude_advection_term`. The advection term may also be integrated by parts by enabling the `../advection_term/integrate_by_parts` option; this is required for the imposition of weak velocity boundary conditions.

Drag term

To include the quadratic drag term in the momentum equation, the `drag_term` option must be enabled under `/system/equations/momentum_equation/` and the non-dimensional drag coefficient C_D should be specified.

Stress term

To include the stress term in the momentum equation, the `stress_term` option must be enabled and the isotropic, kinematic physical viscosity of the fluid ν must be specified.

Turbulence parameterisation

By default, the momentum equation does not account for turbulent Reynolds stresses. However, if the `turbulence_parameterisation` option is enabled, then the Reynolds stresses can be parameterised through the calculation of an eddy viscosity, which models the effects of small-scale eddies on the large-scale flow turbulence. This eddy viscosity is added to the background viscosity ν in the stress term. More information can be found in Turbulence parameterisation.

Source term

An additional user-defined source term can be added to the right-hand side of the equation under consideration via the `source_term` sub-option.

2.3 Running a simulation

A shallow water simulation can be run by executing the `shallow_water.py` file with the Python interpreter, and providing the path to the .swml simulation configuration file. An example would be:

```
python firedrake_fluids/shallow_water.py example.swml
```

from the Firedrake-Fluids base directory. Available flags for the shallow water model are:

- `-c`: Initialise a simulation from a specified checkpoint file.
- `-h`: Display a help message.

2.4 Current limitations

- When using a discontinuous Galerkin method, the form of the stress tensor is currently restricted to:

$$\mathbb{T} = \nu \nabla \mathbf{u}.$$

- When using a discontinuous Galerkin discretisation, the interior penalty method is the only method available for determining the value of $\nabla \mathbf{u}$ at the discontinuous interior element boundaries. Similarly, only a simple upwinding method can be used to determine $\mathbf{u}$ along interior element boundaries.

CHAPTER 3

Stabilisation methods

When using a continuous Galerkin discretisation in advection-dominated problems, it may be necessary to stabilise the advection term in the momentum equation.

The implementation of the stabilisation methods can be found in the file `stabilisation.py`.

3.1 Streamline upwind

This method adds some upwind diffusion in the direction of the streamlines. The term is given by

$$\int_\Omega \frac{\bar{k}}{||\mathbf{u}||^2}(\mathbf{u}\cdot\nabla\mathbf{w})(\mathbf{u}\cdot\nabla\mathbf{u})$$

which is added to the LHS of the momentum equation. The term $\bar{k}$ takes the form

$$\bar{k} = \frac{1}{2}\left(\frac{1}{\tanh(\mathrm{Pe})} - \frac{1}{\mathrm{Pe}}\right)||\mathbf{u}||\Delta x$$

where

$$\mathrm{Pe} = \frac{||\mathbf{u}||\Delta x}{2\nu}$$

is the Peclet number, and Δx is the size of each element.

CHAPTER 4

Turbulence parameterisation

This chapter describes the turbulence models that are available in Firedrake-Fluids.

4.1 Large Eddy Simulation (LES)

The UFL implementation of all LES models can be found in the file `les.py`.

4.1.1 Smagorinsky model

The model calculates an eddy viscosity ν'

$$\nu' = (C_s \Delta_e)^2 \, |\mathbb{S}|,$$

where C_s is the Smagorinsky coefficient which is typically between 0.1 and 0.2 , and Δ_e is some measure of the element size. Here it is given by the square root of the element's area in 2D, or cube root of the element's volume in 3D. The term $|\mathbb{S}|$ is the modulus of the strain rate tensor $\mathbb{S}$:

$$\mathbb{S} = \frac{1}{2} \left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathrm{T}} \right).$$

The eddy viscosity ν' is added to the background/physical viscosity of the fluid, thereby contributing to the stress term in the momentum equation.

CHAPTER 5

Diagnostic fields

Some stand-alone functions are available in the `diagnostics.py` file for computing flow diagnostics.

5.1 Courant number

The Courant number diagnostic computes the field defined by

$$\frac{||\mathbf{u}||\Delta t}{\Delta x},$$

where Δt is the time-step size and Δx is the element size (more specifically, it is twice the element's circumradius).

5.2 Grid Reynolds number

The Reynolds number (whose length scale is relative to the element size Δx) is defined by

$$\mathrm{Re} = \frac{\rho||\mathbf{u}||\Delta x}{\mu},$$

where ρ is the fluid density, μ is the dynamic viscosity, and $||\mathbf{u}||$ is the magnitude to the velocity field. Alternatively,

$$\mathrm{Re} = \frac{||\mathbf{u}||\Delta x}{\nu},$$

where ν is the kinematic viscosity.

5.3 Divergence

This diagnostic field computes the divergence

$$\nabla \cdot \mathbf{u},$$

of a vector field $\mathbf{u}$.