

User Guide

Curvilinear Navier-Stokes (CNS) C++ Code

Bobby Arthur

Department of Civil and Environmental Engineering

Environmental Fluid Mechanics Laboratory

Stanford University

October 16, 2013

Contents

1	Code Structure	2
1.1	Introduction	2
1.2	File Summary	2
1.3	Getting Started	4
2	Code Output	5
2.1	Binary Output	5
2.2	Loading and Viewing Output Data	5
2.3	Restarts	6
3	Examples	7
3.1	Internal Seiche	7
3.2	Lid-driven Cavity	7
3.3	Lock Exchange	7
3.4	Breaking Internal Wave	7
3.5	Sediment Transport	7

Chapter 1

Code Structure

1.1 Introduction

This curvilinear Navier-Stokes (CNS) C++ code is also known as “cns-koltakov” or, more affectionately, “Sergey’s Code,” after its author, Sergey Koltakov. It is the third version of the original Fortran code written by Yan Zang at Stanford in the early 1990’s (Zang et al., 1994). The original code was serial (one processor). The second version was updated by Cui to run with multiple processors using MPI, and is known as PCUI (Cui and Street, 2001). This latest version of the code is basically a translation of PCUI to C++ using an object-oriented framework, which allowed for the addition of a moving grid (Koltakov and Fringer, 2012). Over the years, these various versions of the code have been used to study a wide range of geophysical flows.

The code is based on the fractional step method of Zang et al. (1994) and includes the LES model of Zang et al. (1993). It solves the incompressible Navier-Stokes equations on a generalized curvilinear grid with a rigid lid, employing a semi-implicit time integration scheme with Adams-Bashforth for the explicit terms and Crank-Nicholson for the implicit terms. Additionally, it uses the QUICK and SHARP schemes for advection of momentum and scalars, respectively, and solves the pressure Poisson equation with a multigrid method.

1.2 File Summary

The code is contained in the directory `/cns-koltakov`, which contains the following files:

`Makefile`

Used to compile and run the code (see Section 1.3).

`parameters.dat`

The main input file for the code, which contains the most commonly adjusted variables. The file is read at runtime, so the user may redefine variables without recompiling the code.

`parameters.h`

Defines all the parameters for the code, including physical variables, boundary conditions, and multigrid parameters. Also allows the user to turn different modules (e.g. scalar advection, moving grid, etc.) on or off, run predefined cases (see Chapter 3), or change the bottom bathymetry. In order to modify `parameters.dat`, the function `Set_Parsable_Values()` must be modified here.

`test.cpp`

The main loop of the code.

`navier_stokes_solver.h/.cpp`

Contains the time-stepping scheme for the momentum equations, including, most importantly, the `Predictor()` and `Corrector()` schemes. Also where all arrays containing physical variables are initialized. Additionally, the scalar solve, background potential energy, post-processing, and other similar functions are called here.

`pressure.h`

Contains the multigrid pressure Poisson equation solver.

`scalar.h`

Updates scalar quantities after the velocity has been updated.

`convection.h`

Contains the advection schemes used in the code, specifically, QUICK for momentum advection and SHARP for scalar advection.

`universal_limiter.h`

Contains the limiter used in the above advection schemes.

`tridiagonal_solver.h`

Contains functions used to solve tridiagonal systems with periodic or non-periodic boundary conditions. Used both in the implicit part of the momentum and scalar solutions, as well as the smoothing operation in the multigrid solver.

`curvilinear_grid.h/.cpp`

Creates the grid and calculates the necessary metric quantities.

`mpi_driver.h/.cpp`

Initializes and controls parallel implementation using MPI. Also defines binary output functions.

`metric_quantities.h`

Defines the metric quantities that are calculated in `curvilinear_grid.cpp`.

`array_Xd.h/.cpp`

Classes that define all array operations in the code, where **X** is a number 1-3.

`potential_energy.h`

Calculates the background potential energy and related quantities. Note that kinetic energy and dissipation are calculated in `navier_stokes_solver.cpp`.

`turbulence.h`

Contains the turbulence model, which is not fully implemented (yet).

`curvilinear_moving_grid.h/moving_grid_engine.h`

Implementation of the moving grid.

`interpolant.h`

Used with the moving grid to interpolate physical variables from the old to new grid.

`data_aggregator.h`

Aggregates time-series of physical variables.

`parameter_file_parser.h`

Reads the input file `parameters.dat`.

1.3 Getting Started

To begin using the code, it must first be compiled with the Intel C++ Compiler (`icc`) and the MPICH compiler (`mpicc`). A free version of the Intel C++ Compiler can be found at <http://software.intel.com/en-us/non-commercial-software-development>, while MPICH can be found at <http://www.mpich.org/downloads/>. Note that MPICH must be compiled on your machine with the Intel compiler `icc` flag, see the MPICH documentation for specifics.

Once the compilers have been downloaded, make sure that the path to `mpicc` is specified correctly in the Makefile next to `CC=`. Then, the code can be compiled with `make test` and run with `make runX`, where `X` is the number of processors. Note that this number must agree with what is specified in `parameters.dat`. That is, `X = num_cpu_x*num_cpu_y*num_cpu_z`.

Chapter 2

Code Output

2.1 Binary Output

The code output is contained in binary files that correspond to a specific processor. Data is output for the variables specified in `parameters.h` at the frequency defined by `save_timestep_period` in `parameters.dat`. These files will be stored in `output_dir` defined in `parameters.dat`. The files are organized by variable and processor. For example, the velocity output from the 0th processor will be stored in `velocity.0`.

2.2 Loading and Viewing Output Data

The `/cns-koltakov/mfiles` directory contains MATLAB scripts to view data from binary output directly, or to load `.mat` files which can be used in your own scripts. The files included in this directory are as follows.

`cns_viz.m`

Plots 2D slices of any physical variable or 3D isosurfaces of density and vorticity.

`cns_load.m`

Loads physical variables into matrices that are stored in a `.mat` file.

`cns_load_energy.m`

Loads a time series of energy quantities and saves a `.mat` file.

`load_binary_X.m`

Functions that load data directly from binary output files and return a 3D (x,y,z) matrix.

`calculate_binary_X.m`

Functions that calculate other quantities (e.g. vorticity) using data from binary output files.

2.3 Restarts

The code is able to save data for restarts if a nonzero value for `restart_timestep_period` is given in `parameters.dat`.

Chapter 3

Examples

3.1 Internal Seiche

3.2 Lid-driven Cavity

3.3 Lock Exchange

3.4 Breaking Internal Wave

3.5 Sediment Transport

Coming soon!

Bibliography

- Cui, A. and Street, R. L. (2001). Large-eddy simulation of turbulent rotating convective flow development. *J. Fluid Mech.*, 447:53–84.
- Koltakov, S. and Fringer, O. B. (2012). Moving grid method for numerical simulation of stratified flows. *Int. J. Numer. Meth. Fl.*
- Zang, Y., Street, R. L., and Koseff, J. R. (1993). A dynamic mixed subgrid-scale model and its application to turbulent recirculating flows. *Physics of Fluids A: Fluid Dynamics*, 5:3186.
- Zang, Y., Street, R. L., and Koseff, J. R. (1994). A non-staggered grid, fractional step method for time-dependent incompressible navier-stokes equations in curvilinear coordinates. *J. Comput. Phys.*, 114(1):18–33.