User's Guide to Diablo

John R. Taylor
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
(Dated: July 18, 2012)

This version of Diablo solves the incompressible, Boussinesq Navier-Stokes equations in a two-dimensional geometry. The code was written and tested using MATLAB, version R2011b (7.13.0.564). The code is intended for educational purposes, and not all functionality has been fully tested.

I. BACKGROUND

Why MATLAB?

My goal when writing this code was to provide an accurate and flexible Navier-Stokes solver that is easy for the user to modify, and use for quick 'numerical experiments'. While the choice of MATLAB as a scripting language significantly reduces the speed of the solver, it provides a single environment where the user can easily create arbitrary initial/boundary conditions and process and visualize the results. Hopefully this will allow the users to spend more time experimenting with different physical configurations (the fun part!) and less time initializing results, running code. I have tried to make the solver run as fast as possible, given the constraints of a scripting language. Unfortunately, in the latest version, several intrinsic functions (such as *ilu*) are used which are not currently supported in the open-source Octave environment. For those who do not have access to MATLAB, it should be relatively straightforward to alter the code to work in Octave. For those who are looking for increased performance, an open-source Fortran version of Diablo is available at `http://renaissance.ucsd.edu/Diablo.html`.

Numerical method

Diablo is based on an algorithm developed by Tom Bewley at UCSD using second-order centered finite differences, with stencils chosen to ensure discrete conservation of mass, momentum, and energy. See the forthcoming textbook *Numerical Renaissance* (`http://renaissance.ucsd.edu`) for details. The equations are time-stepped using the third-order accurate low-storage Runge-Kutta-Wray algorithm, and all viscous/diffusive terms are treated with the semi-implicit Crank-Nicolson method. This version is implemented using a staggered, stretched cartesian grid. The continuity equation is enforced using the fractional step method.

II. GOVERNING EQUATIONS

Diablo solves the incompressible, Boussinesq equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \mathbf{f} \times \mathbf{u} = -\nabla p + \mathrm{Ri}\theta + \mathrm{Re}\nabla^2 \mathbf{u} + \boldsymbol{\mathcal{F}}(x, y), \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

$$\frac{\partial \theta}{\partial t} + \mathbf{u} \cdot \nabla \theta + \mathbf{u} \cdot (\nabla \theta)^B = \frac{1}{\mathrm{Re\ Pr}} \nabla^2 \theta, \tag{3}$$

where $\mathbf{f}$ is the Coriolis parameter for rotating flows, Re is the Reynolds number, $\mathrm{Pr} = \nu/\kappa$ is the Prandtl (or Schmidt) number, Ri is the Richardson number to allow for buoyancy effects, $\boldsymbol{\mathcal{F}}(x, y)$ is an optional spatially-varying body force, and $(\nabla\theta)^B$ is an optional background scalar gradient.

III. CODE ORGANIZATION

The main directory, diablo_mat is consists of the following scripts and directories:

- **code** - This directory contains the scripts where all of the time-stepping work is done. You shouldn't need to modify any of these, but you are strongly encouraged to have a look through them. The contents are described in detail below.

- **docs** - A place for some documentation files (like this one!)

- A number of scripts, which can be edited by the user:
  - *diablo.m* - This is the main program for Diablo. Essentially it just calls the initialization scripts and runs the time-stepping loop.
  - *create_flow.m* - This script creates initial conditions for the velocity and scalars.
  - *create_grid.m* - Create arrays containing the x,y coordinates of the staggered, stretched grid. Two grid types are included to create grids with points clustered more tightly at the edges or center of the domain. Other grid types can be created, but the same procedure should be followed to ensure that the staggered grid has the right properties. For more details on the grid definition, see the grid layout diagram below.
  - *display_flow.m* - This script is called periodically during a simulation to visualize the results of the ongoing calculation. The display interval in time steps (N_DISP_FLOW) is set in set_params.m. You will want to change the contents of this script to plot something useful for your simulation. You might also want to calculate and display basic statistics of the flow, to keep track of things as they develop.
  - *save_flow.m* - Each time we update the velocity, scalars, and pressure, the old values are over-written with the new ones. We want to save the fields, periodically, and this script is intended to do that.

– *save_stats.m* - In addition to saving the full flow variables, it is often useful to do some data processing while during the simulation, and save the results. Since these variables are often much smaller than the full arrays, we typically call this script more often than save_flow.m. There are a few basic statistics already here, but you will probably want to add some more.

– *set_bcs.m* - This script specifies the boundary conditions to be applied to the velocity, pressure, and scalar fields. The boundary conditions can be of three types: Specify the value of a variable at the boundary (Dirichlet), specify the normal gradient at the boundary (Neumann), or cyclic (periodic) boundary conditions. The type of boundary condition for each variable is specified in a variable like U1_BC_XMIN (U1 is the variable, XMIN specifies that this boundary condition type applies to the lower end of the X-coordinate.) When using Dirichlet and Neumann boundary conditions, we also need to specify the value or gradient to be applied at the boundary. These are specified using arrays like U1_BC_XMIN_C1. The size of these arrays should match the size of the grid tangential to the boundary (i.e. the size of U1_BC_XMIN should match the size of the Y-coordinate).

– *set_params.m* - This important script contains the values of various physical and computational parameters. The parameters are briefly described both within the script, and in the list below.

IV. RUNNING DIABLO

To setup and run a new simulation, follow these steps (the order in which you edit the scripts is not important.)

1. Using a text editor or from within MATLAB itself, edit the following scripts:

   - set_params.m - Specify the physical and computational parameters.
   - create_grid.m - Specify the characteristics of the discrete grid.
   - set_bcs.m - Specify the boundary conditions for each variable.
   - create_flow.m - Specify the initial condition.
   - save_flow.m and save_stats.m - Optionally, edit these scripts to control which variables to save.
   - display_flow.m - Optionally - Specify how to plot the results during the simulation.

2. Run MATLAB from within the main Diablo directory, and type "diablo" to launch a simulation. (You may want to type "clear all" before running Diablo to ensure that the workspace is clean.)

3. Post-process, plot, and save the results before setting up a new simulation.

It is also possible to continue a simulation that was stopped prematurely. Just remove the call to "create_flow" from inside diablo.m, and re-run the script with all variables intact. The simulation should start back where it left off.

V. PARAMETERS

Here is a brief description of the simulation parameters, specified in *set_params.m*.

First, specify some general simulation parameters:

Re - The Reynolds number, or if dimensional units are being used, $1/\nu$, where ν is the kinematic viscosity.

LX - The domain size in the x-direction.

LY - The domain size in the y-direction.

SOLVE_U3 - A logical flag, specifying whether to solve for the component of the velocity in the (z) direction, normal to the coordinate frame. Simulations that do this are sometimes called $2\frac{1}{2}$D.

NX - The number of gridpoints in the x-direction.

NY - The number of gridpoints in the y-direction.

N_TH - The number of passive or active scalars to time step. This number can be any integer from 0 and higher (depending on available memory resources).

For each scalar, specify the following parameters:

PR - The Prandtl (or Schmidt) number, ν/κ, where ν is the kinematic viscosity, and κ is the scalar diffusivity. The time stepping is not currently adjusted to account for this value, so you may need to be careful about using very large or small values.

RI - The Richardson number. This is the constant that multiplies each scalar in the momentum equations, i.e. $\frac{\partial U1}{\partial t} = \ldots + GRAV\_X(1)RI(1)TH(1) + GRAV\_X(2)RI(2)TH(2)$, etc. For example, if the first scalar TH(1) is density, then $RI(1) = g/\rho_0$, normalized properly. For any passive scalar (which does not affect the buoyancy), RI should be set to zero.

GRAV_X, GRAV_Y, GRAV_Z. These components specify the direction of the gravitational unit vector.

DTHDX, DTHDY, DTHDZ - These constants allow us to prescribe a background gradient to each scalar. This is particularly useful when using periodic boundary conditions. For example, we could impose a background vertical density gradient, and solve for periodic perturbations about that gradient. These terms appear in the scalar evolution equation:

$$\frac{\partial \mathrm{TH}}{\partial t} + \mathrm{U1} \cdot \mathrm{DTHDX} \; + \mathrm{U2} \cdot \mathrm{DTHDY} + \mathrm{U3} \cdot \mathrm{DTHDY} = \ldots \tag{4}$$

FORCE_X, FORCE_Y, FORCE_Z - These are the components of a body force to be applied to the right hand side of the momentum equations. These can either be scalars or matrices of size NX,NY to allow for a spatially-varying body force.

Parameters for rotating flows

When are are solving for rotating flow, the Coriolis term appears in the momentum equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{f} \times \mathbf{u} = \ldots \tag{5}$$

Here, the Coriolis vector is represented as $\mathbf{f} = \mathrm{I\_RO} \;\; (\mathrm{CORI\_X} \;\; \hat{x} + \mathrm{CORI\_Y} \;\; \hat{y} + \mathrm{CORI\_Z} \;\; \hat{z})$, where the parameters are:

I_RO - The magnitude of the Coriolis parameter, or the inverse Rossby number in non-dimensional variables.

CORI_X, CORI_Y, CORI_Z - The directional components of the Coriolis unit vector.

Timestepping parameters:

N_TIME_STEPS - The number of time steps to perform in the main time stepping loop in *diablo.m*

VARIABLE_DT - a logical parameter specifying whether to use a variable time step (if = 1) or a fixed time step (if = 0). If a variable time step is used, the time step is dynamically adjusted based on the CFL criteria in the script *courant.m*.

DELTA_T - If we are using a fixed time step (VARIABLE_DT=0), then this sets the timestep size.

CFL - Otherwise, if we are using a variable time step, it will be calculated in *courant.m* using this CFL number, where $\Delta t = \mathrm{CFL} \;\; min(\Delta x/\mathrm{U1}, \Delta y/\mathrm{U2})$.

Simulation monitoring - Every so often, we want to monitor the output of the simulation. These parameters control how often:

N_DISP_FLOW - Specify how often (in time steps) to call *display_flow.m* to plot something.

N_SAVE_FLOW - Specify how often (in time steps) to call *save_flow.m* to save the flow field.

N_SAVE_STATS - Specify how often (in time steps) to call *save_stats.m* to calculate and save flow statistics.

Diablo - Discrete Grid

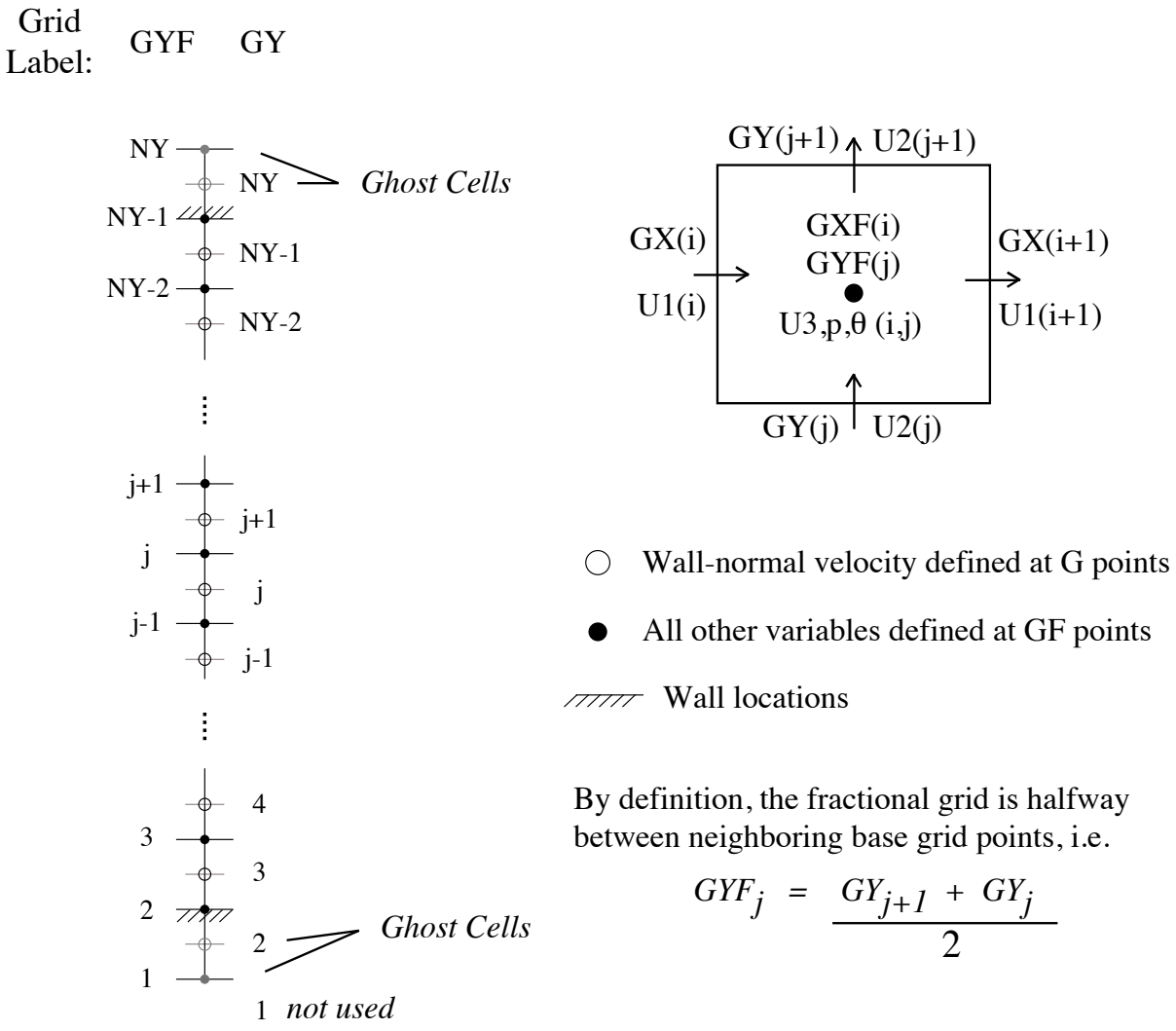

FIG. 1: Grid layout of Diablo