QVORT

A quantised vortex code

Andrew Baggaley
School of Mathematics and Statistics
Newcastle Univeristy
U.K.
a.w.baggaley@gmail.com

March 23, 2018

Contents

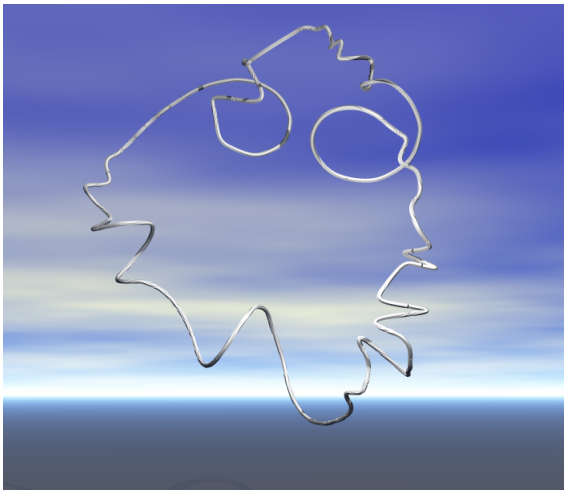

Figure 1: A 3D rendering of a vortex filament with wave perturbations, outputted from the QVORT code (using the bryce_out.m script) and then rendered using the Bryce software package

1 Introduction

Welcome to the documentation for the QVORT quantised vortex code. The code is written in well commented Fortran, arranged in a modular structure. The primary aim of this documentation is to highlight which module(s) are involved in the various sequences of the code, and also the parameters which can be set in the *run.in* file. To run QVORT you need to be on a *UNIX*-type system (*Make* and *bash* are heavily used) with a Fortran compiler. All the Fortran code can be found in the directory *src*. MATLAB is used for post-processing, although it is probably very straightforward to convert the MATLAB scripts (found in *bin*) to work with *Octave*, freely available under the GNU license. Please do not hesitate to contact with any queries, or if you are interested in helping to further develop the code.

2 Boundary conditions

AT present the code can be run with 3 different boundary conditions all of which apply to a cube with a variable size which can be set as a runtime

parameter. These are open, periodic or reflective boundaries. If running with open boundaries the box plays no role and the filaments can move freely in space, the box size is simply used as a guide by the plotting routines. Periodic boundaries mean that if a filament leaves one side of the box it wil re-enter on the opposite side. Whilst this is appealing in order to investigate the properties of a the vortices in the box it does not come without some compromises. Firstly the energy diagnostic is no longer valid, and its results should not be used. Secondly in order to keep the velocity field continuous an extra 27 permutations of the field are required, slowing the code down considerably. Finally one can impose reflective boundaries using the method of images in which mirror vortices are used at each of the boundaries, again significantly increasing the workload.

3 Equations

The QVORT code is a vortex-filament code, a technique pioneerd by Schwarz in the early 1980s. Each line-vortex in the system is discretised into a number of points which are evolved according to an equation of motion. Points along the line are added (or removed) if the vortex is stretched (or compressed). If any two lines become very close (a distance less than the separation along the line) then the filaments reconnect, changing the topology of the system. More precisely we define our vortex filament as a three dimensional curve $\mathbf{s} = \mathbf{s}(\xi, t)$. Here ξ represents arc-lengths and t is time. We can construct the tangent vector $\mathbf{s}'$, then normal vector, $\mathbf{s}''$, and the binormal vector, $\mathbf{s}' \times \mathbf{s}''$ by taking numerical derivatives. Note $\mathbf{s}' = d\mathbf{s}/d\xi$, and so on. We denote the velocity at a point $\mathbf{s}$ as $\mathbf{u}(\mathbf{s})$ and consider the velocity field to be made up of two parts,

$$\mathbf{u} = \mathbf{u}_\mathrm{s} + \mathbf{u}_\mathrm{n}, \tag{1}$$

where $\mathbf{u}_\mathrm{s}$ is the superfluid component of the velocity field (i.e. motion induced by vortices), and $\mathbf{u}_\mathrm{n}$ is the normal fluid component. In the code there are various options for both the normal and superfluid components. For the superfluid component one can choose between the local induction approximation (LIA),

$$\mathbf{u}_\mathrm{s} = \beta \mathbf{s}' \times \mathbf{s}'', \qquad \beta = \frac{\Gamma}{4\pi} \ln\left(\frac{R}{a}\right), \tag{2}$$

where Γ is the quantum of circulation (a parameter which can be set in *run.in*), R is the radius of curvature ($1/|\mathbf{s}''|$), and a is the vortex core size

(fixed 10^{-8}cm). We note that the time per iteration with this method scales like $\mathcal{O}(N)$, where N is the total number of particles in the simulation.

A second option is to solve the de-singulised Biot-Savart integral,

$$\mathbf{u}_{\rm s} = \beta' \mathbf{s}' \times \mathbf{s}'' + \frac{\Gamma}{4\pi} \int_{\ell'} \frac{(\mathbf{s}-\mathbf{r}) \times d\mathbf{s}}{|\mathbf{s}-\mathbf{r}|^3}, \qquad \beta' = \frac{\Gamma}{4\pi} \ln\left(\frac{\sqrt{\ell_i \ell_{i-1}}}{a}\right). \tag{3}$$

ℓ' represents the full system with the local region around the point of interest removed (to avoid division by 0). This local contribution enters back in as a binormal vector with a magnitude affected by the size of local region ℓ_i/ℓ_{i-1}. Whilst a more realistic scheme the downside of using the full integral comes in the scaling of the time per iteration with N, which is quadratic ($\mathcal{O}(N^2)$).

The current options for the normal fluid are zero, $\mathbf{s}_{\rm n} = 0.$, a constant x-flow, $\mathbf{u}_{\rm n} = (\text{const}, 0, 0)$, the ABC flow,

$$\mathbf{u}_{\rm n} = \tag{4}$$

We can also set a constant flow in the x-direction, $\mathbf{u}_{\rm n}(1) = $ const. or use the KS model to give a synthetic turbulent velocity field, using a summation of random Fourier-modes with an imposed Energy spectrum. These two velocities combined together to give the total equation of motion

$$\frac{d\mathbf{s}}{dt} = \mathbf{u}_{\rm s} + \alpha \mathbf{s}' \times (\mathbf{u}_{\rm n} - \mathbf{u}_{\rm s}) + \alpha' \mathbf{s}' \times [\mathbf{s}' \times (\mathbf{u}_{\rm n} - \mathbf{u}_{\rm s})] \tag{5}$$

4 Numerical schemes

At a particular particle on a filament, with position $\mathbf{s}_i$, define the distance to the particle in-front ($\mathbf{s}_{i+1}$) as ℓ_{i+1} and the distance to the particle behind ($\mathbf{s}_{i-1}$) as ℓ_{i-1}. By in-front/behind we refer to the particles next/previous along the filament. We can construct finite difference approximations to the first and second derivatives by taking Taylor's series expansions. The resulting forms are as follows,

$$\frac{d\mathbf{s}_i}{d\xi} = \frac{\ell_{i-1}\mathbf{s}_{i+1} + (\ell_{i+1} - \ell_{i-1})\mathbf{s}_i + \ell_{i+1}\mathbf{s}_{i-1}}{2\ell_{i+1}\ell_{i-1}} + \mathcal{O}(\ell^2), \tag{6}$$

$$\frac{d^2\mathbf{s}_i}{d\xi}^2 = \frac{2\mathbf{s}_{i+1}}{\ell_{i+1}(\ell_{i+1}+\ell_{i-1})} - \frac{2\mathbf{s}_i}{\ell_{i+1}\ell_{i-1}} + \frac{2\mathbf{s}_{i-1}}{\ell_{i-1}(\ell_{i+1}+\ell_{i-1})} + \mathcal{O}(\ell^2). \tag{7}$$

We must also use some numerical scheme to integrate in time. Due to the cost of calculating the velocity field at each point (scales with N^2 in the Biot-Savart case) it is very intensive to use a Runge-Kutta type scheme. Therefore we use the $3^r d$ order Adams-Bashforth method where the evolution of each particle at a point $\mathbf{x}_i$ is given by:

$$\mathbf{s}_i^{n+1} = \mathbf{s}_i^n + h\mathbf{u_i}^n. \tag{8}$$

We denote h as our time-step, and note that this is controlled by the maximum resolution discussed below.

4.1 Line algorithms

If the separation between two points on the filament becomes two large we must introduce a new point in order to maintain the resolution along the filament. As the evolution equation for a vortex filament is sensitive to the local curvature linear interpolation is not a good idea. A way to keep the curvature constant is to use the following algorithm,

$$\mathbf{s}_{i'} = \frac{1}{2}(\mathbf{s}_i + \mathbf{s}_{i+1}) + \left(\sqrt{R_{i'}^2 - \frac{1}{4}\ell_i^2} - R_{i'}\right)\frac{\mathbf{s}_{i'}^{''}}{|\mathbf{s}_{i'}^{''}|}, \tag{9}$$

where $\mathbf{s}_{i'}$ is the position of the point introduced if the separation between the points $\mathbf{s}_i$ and $\mathbf{s}_{i+1}$ becomes greater than some threshold δ, which we note is the minimum resolution. Note,

$$\mathbf{s}_{i'}^{''} = \frac{\mathbf{s}_i + \mathbf{s}_{i+1}}{2}, \tag{10}$$

and $R_{i'} = |\mathbf{s}_{i'}^{''}|^{-1}$. If the separation between two points becomes less than $\delta/2$ then we remove a point to ensure that we never have a maximum resolution greater than $\delta/2$.

As discussed above our time-step h is controlled by the size of δ in the following way.

At present the time-step is set manually in the code, however you will not be able to run the code and will receive a fatal error.

5 Running the code

To run the code you must first compile it using the following commands under a linux/unix system

```
make
```

ensure that there is a data directory, if not create with the following command,

```
mkdir ./data
```

copy an example run file from the directory examples, for example

```
cp ./examples/single_loop.in ./run.in
```

the code then be run with the following command

```
./run.sh
```

the run script can take a variety of options, if interested run the command

```
./run.sh -h
```

for example the following command will restart the code if possible

```
./run.sh -r
```

5.1 Editing the run.in file

In this next section we provide documentation of all the possible arguments that can be set in the run.in file. This is split into two sections, the essential parameters which must be set for the code to run, and optional parameters.

5.1.1 Essential parameters

parameter	description	arguments
nsteps	the number of time-steps to use	positive (large) integer
shots	how often to print to file	positive integer
shots	how often to print to file	positive integer
pcount	# points are used in the initial conditions	positive integer
dt	the time-step - if too large code will not run	positive real
delta	the resolution (spacing of points on filaments)	positive real
box_size	the length of the side of the cube we run in	positive real
velocity	how we calculate the velocity field	LIA, BS, Tree
initf	our initial configuration	single_loop
		single_line
		random_loops
		crow
		leap-frog
		lined_filaments
		colliding_loops
		kivotedes
		cardoid
		wave_loop
		wave_line
		line_motion
		tangle
boundary	the boundary conditions	open, periodic, mirror

5.1.2 Optional parameters

These all have default values (shown in brackets) which, in general, turn the specific feature off.

parameter	description	arguments
line_count	how many lines or loops for certain initf	positive integer(1)
quant_circ	quantum of circulation	positive real(0.00097)
mesh_size	print normal/superfluid velocity on a mesh	integer (0)
tree_theta	opening angle if using tree algorithms	positive integer(0)
tree_print	print the tree mesh to file	T/F (F)
binary_print	print the files as binary data	T/F (T)
curv_hist	binned curvature information for histogram	T/F (F)
mirror_print	print the image vortices to file	T/F (F)
normal_velocity	normal fluid velocity field	zero (zero)
		xflow
		ABC
		KS
normal_fluid_cutoff	turn off the normal fluid at certain t	positive real (100.)
alpha	mutual friction coefficients	two reals (0. 0.)
quasi_pcount	number of particles in the code	integer (0)
particle_type	the type of particles	fluid, inertial
initg	the initial configuration of the particles	random, one-side, pairs
force	do we want forcing in the code	off (off)
		top_boundary
		box_shake
		delta_corr
force_amp	the amplitude of the forcing (in terms of δ)	real (0.)
force_freq	the forcing frequency	real (0.)
special_dump	dump the filament information at a specific t	real (0.)
wave_count	for wave_line/loop initial condition	integer (1)
wave_slope	for wave_line/loop initial condition	real (-1.5)
wave_amp	for wave_line/loop initial condition	real (10)
wave_type	for wave_line/loop initial condition	planar, helical (planar)

Final section on stiff ode solver