Contents

1. Simulations (10 pgs)

This part of thesis serves as a widen documentation for the `PyVort` codebase, a new platform to simulate quantum vortex rings. The code is written in well commented Python 3, arranged in a modular structure. The primary aim of this documentation is to highlight which module(s) are involved and how they work. In appendix, one can find a table of the parameters (user's options) which can be set in the `config.py` file.

To run the code, there has to be installed only Python 3 (with various libraries) on any OS. All the Python code can be found in the directory `src/`. The entire project is open-source and can be found as a public GitHub repository. Pull requests of any further development would be definitely appreciated.

At present, we use infinite boundary conditions. Therefore, only closed-loop vortices can be realized using simulation. However, the codebase is flexible and supports the potential implementation of unclosed loops.

1.1 Vortex filament model

The `PyVort` code is based on vortex filament (VF) model, a technique pioneerd by Schwarz in the early 1980s. Superfluid vortex filament is represented by a series of mesh points (segments) distributed along the centerline of the VF. The motion of the whole VF is summed up by the motion of each mesh point.

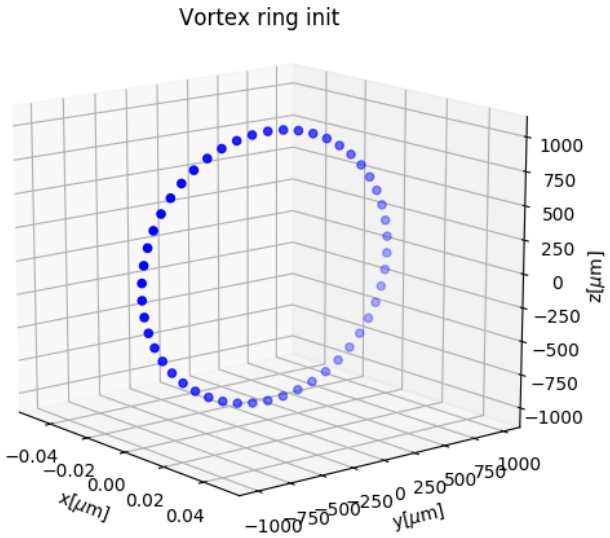

Figure 1.1: Example of vortex ring segments

As introduced in Theoretical Background, we define the VF more precisely as a three dimensional curve $\mathbf{s}(\xi, t)$. Here, ξ represent an arc-lengths and t is time. Each segment is given by its coordinates $\mathbf{s}_i$, direct neighbour indices (previous $(i-1)$ and next $(i+1)$). This resolves in a directed digraph of segments, which is a good starting point for the initial data structure.

Next we define the tangent vector $\mathbf{s}'$, then normal vector $\mathbf{s}''$, and the binormal vector $\mathbf{s}' \times \mathbf{s}''$ by taking numerical derivatives. Note $\mathbf{s}' = \mathrm{d}\mathbf{s}/\mathrm{d}\xi$, and so on. Numerical derivatives are achieved using Finite Differences method, as introduced in next subsection.

Finite differences

In order to properly calculate the numerical derivatives $\mathbf{s}'$ and $\mathbf{s}''$, belonging to the directed curve in 3D, we need to use a sophisticated numerical method. At a particular segment with position $\mathbf{s}_i$, we define the distance to the particle in-front $\mathbf{s}_{i+1}$ as $l_i = |\mathbf{s}_i - \mathbf{s}_{i+1}|$ and the distance to the particle behind $\mathbf{s}_{i-1}$ as $l_{i-1} = |\mathbf{s}_{i-1} - \mathbf{s}_i|$. By in-front/behind we refer to the particles next/previous along the filament. Similarly, we can define the l_{i+1} and l_{i-2} for the farther segment lines.

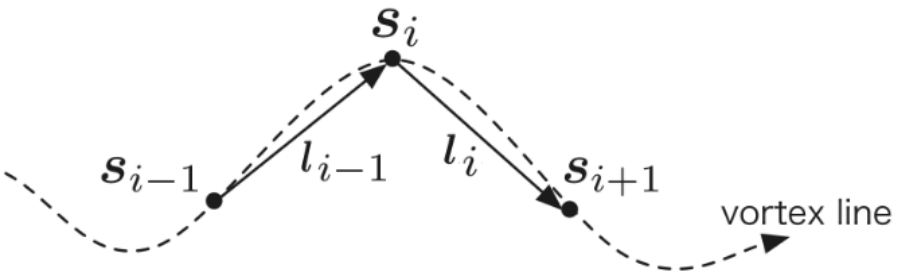

Figure 1.2: Depiction of a few segments and corresponding lengths

For accuracy, we approximate all the spatial derivatives $\mathbf{s}_i'$, $\mathbf{s}_i''$ by a fourth-order finite difference method (FD), which can also account the varying distances along the vortex filament. With this, the first and second derivatives can be obtained based on coordinates of 2 closest neigbours (on each side).

Using FD theorem, we can construct the approximations by taking the Taylor's series expansions. We can then write:

$$\frac{\mathrm{d}^n \mathbf{s}_i}{\mathrm{d}\xi^n} \approx A_i \mathbf{s}_{i-2} + B_i \mathbf{s}_{i-1} + C_i \mathbf{s}_i + D_i \mathbf{s}_{i+1} + E_i \mathbf{s}_{i+2} \qquad \text{for } n \in \{1, 2\} \tag{1.1}$$

Calculation of coefficients A, B, C, D, E can be done in many ways. In code, we use

both the analytical solution (closed form) and the solution by inverting the Vandermonde matrix. The first one is obviously faster, but the second one is scalable. By *scalability*, we mean that if we would decide to calculate derivatives based on nearest $n > 4$ neighbours, the Vandermonde method can be still used, whereas the closed-form solutions won't be more available. However, the inverting process of Vandermonde matrix is computationally expensive and doesn't have to converge sometimes.

In this work, we used the closed-form solution (4 neighbours) and left the Vandermonde method as an `except` case for the higher-level mode.

Biot-Savart discretisation

We denote the static external sources of velocity fields (which can be set in `config.py`) as $\mathbf{v}_{n,ext}$ and $\mathbf{v}_{s,ext}$. The equation of motion for given segment is then given directly by Schwarz's equation(**??**):

$$\frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}t} = \mathbf{v}_{s,ext} + \mathbf{v}_{\mathrm{ind}}^{(i)} + \mathbf{v}_{\mathrm{drive}}^{(i)} \tag{1.2}$$

The first difficulty in the VF model comes from the calculation of term $\mathbf{v}_{\mathrm{ind}}$. As we shown previously (**??**), this advection term can be split into the LIA part and a Biot-Savart integral:

$$\mathbf{v}_{\mathrm{ind}}^{(i)} = \mathbf{v}_{\mathrm{LIA}}^{(i)} + \mathbf{v}_{\mathrm{BIOT}}^{(i)} = \frac{\varkappa}{4\pi}(\mathbf{s}_i' \times \mathbf{s}_i'') \ln\left(\frac{2\sqrt{l_{i-1}l_i}}{a}\right) + \frac{\varkappa}{4\pi}\int_{\mathcal{L}'} \frac{(\mathbf{r}' - \mathbf{s}_i) \times \mathrm{d}\mathbf{r}'}{|\mathbf{r}' - \mathbf{s}_i|^3}\,, \tag{1.3}$$

where l_{i-1} and l_i are the arc lengths of the curve between points $\mathbf{s}_{i-1}$ and $\mathbf{s}_i$ and between $\mathbf{s}_i$ and $\mathbf{s}_{i+1}$ respectively, and $\mathcal{L}'$ is the original vortex line without the two segment lines between $\mathbf{s}_{i-1}$ and $\mathbf{s}_{i+1}$.

Using the segment discretisation, the Biot-Savart integral can be rewritten into the sum of single-line contributions between each j-th and $j+1$-th segment (except for the ones attached to the i-th point):

$$\mathbf{v}_{\mathrm{BIOT}}^{(i)} \approx \frac{\varkappa}{4\pi} \sum_{j \notin \{i-1,i\}} \frac{(R_j + R_{j+1})(\mathbf{R}_j \times \mathbf{R}_{j+1})}{R_j R_{j+1}(R_j R_{j+1} + \mathbf{R}_j \cdot \mathbf{R}_{j+1})}\,, \tag{1.4}$$

where $\mathbf{R}_j = \mathbf{s}_j - \mathbf{s}_i$ and $\mathbf{R}_{j+1} = \mathbf{s}_{j+1} - \mathbf{s}_i$ are the relative vectors from the given point.

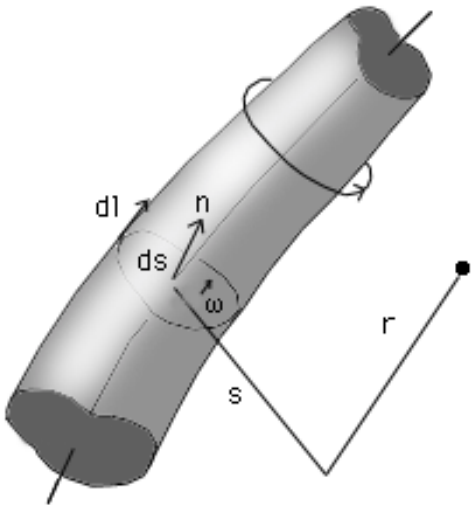

Figure 1.3: The infinitesimal contribution of a segment line between two points $\mathbf{s}_j$ and $\mathbf{s}_{j+1}$ at a given point $\mathbf{r}$.

Note that, if one takes in account the Biot-Savart law for N mesh points, the computational time is proportional to $\mathcal{O}(N^2)$, while, if one uses just the LIA term, it is only $\mathcal{O}(N)$. Numerical simulations based on Biot-Savart are therefore far expensive and not practical, even with the speed of today's computers.

One way to get around this difficulty is to update the LIA term. In this method we neglect completely the non-local Biot-Savart integral and keep just the local term. This is typically done with a minor adjustments within the log term:

$$\mathbf{v}_{\mathrm{LIA}}^{(i)} = \frac{\varkappa}{4\pi}(\mathbf{s}_i' \times \mathbf{s}_i'') \ln\left(\frac{2R_i}{a}\right), \tag{1.5}$$

where R_i is a filament length scale - may be taken as a local curvature of $i-th$ segment: $R_i = 1/|\mathbf{s}_i''|$. Updated LIA is a very convenient approximation and works very well for calculating the motion of a single vortex ring.

State definition

In code, a single vortex ring object is represented using the `class` structure. This structure is partially updated after each time step and works as a source of truth. We will call it as the *state* of the vortex ring and it is defined (and initialised) with following properties:

- shape - a dictionary of three parameters: ring center coordinates $[x_c, y_c, z_c]$, radius

R and the direction of desired motion $\{x, y, z\}$ (three possible axis)

- velocity - the actual velocity magnitude of vortex ring center $|\mathbf{v}_c|$
- number of segments - number of segments N the vortex ring is composed of
- segments - an array of all segments, each one with following attributes:
  - coordinates - an array of segment coordinates $\mathbf{s}_i = [x_i, y_i, z_i]$
  - previous/next neighbour - array localisation indices of the *previous* $(i-1)$ and the *next* $(i+1)$ segment within the context of the directed vortex
  - tangent/curvature - a tangential and normal vectors $\mathbf{s}_i'$ and $\mathbf{s}_i''$
  - LIA velocity - a self-induced velocity $\mathbf{v}_{\mathrm{LIA}}^{(i)}$ driven by the local curvature.
  - BIOT velocity - a self-induced velocity driven by the farther segment lines of the vortex ring $\mathbf{v}_{\mathrm{BIOT}}^{(i)}$
  - Drive velocity - a velocity given by the mutual friction force $\mathbf{v}_{\mathrm{drive}}^{(i)}$
  - Full velocity - the sum of external sources $\mathbf{v}_{s,ext}$, LIA velocity $\mathbf{v}_{\mathrm{LIA}}^{(i)}$, BIOT velocity $\mathbf{v}_{\mathrm{BIOT}}^{(i)}$ and the drive velocity $\mathbf{v}_{\mathrm{drive}}^{(i)}$, resulting in $\mathrm{d}\mathbf{s}_i/\mathrm{d}t$

The proper initialisation of the *state* in the very beginning of simulation includes the following steps:

1. an input from user: *center, radius* and *direction*
2. the gradual initialisation of neigbour indices (to an i-th element in segment array will be assigned $i-1$ index as the *previous* and $i+1$ index as the *next* neighbour index)
3. Calculation of all $\mathbf{s}'$ and $\mathbf{s}''$ using Finite differences method
4. Calculation of all segment velocities using motion equations and their approximative forms
5. Calculation of the center velocity by taking the mean of all segments' full velocites in corresponding direction

1.2 Time evolution

Now when we defined and ran all the necessary calculations leading to the quantum vortex ring's *state* fulfillment, we can start to propagate it in time.

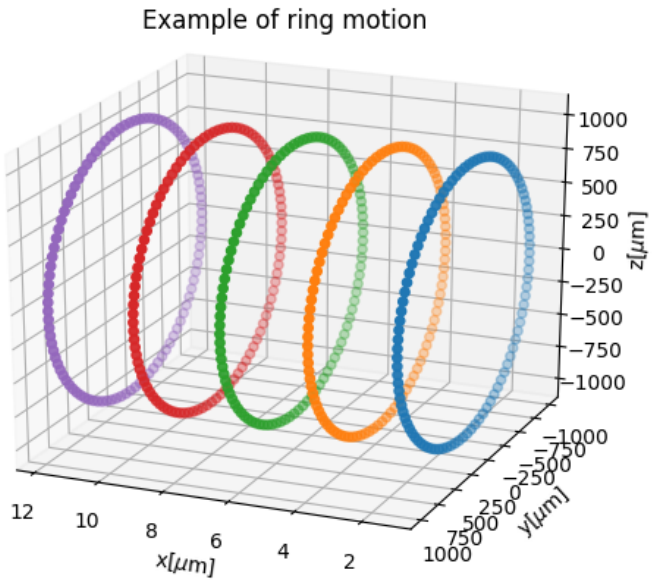

Figure 1.4: Example of a moving vortex ring

Time stepping

Time evolution is based on an explicit iterative method: the fourth-order Runge-Kutta (RK4) scheme. When we consider the Schwarz's equation $\mathrm{d}\mathbf{s}_i/\mathrm{d}t \equiv \mathbf{v}_{\mathrm{full}}^{(i)}$, the stepping algorithm is given as:

$$\mathbf{s}_i(t+\mathrm{d}t) = \mathbf{s}_i(t) + \frac{\mathrm{d}t}{6}(\mathbf{v}_1^{(i)} + 2\mathbf{v}_2^{(i)} + 2\mathbf{v}_3^{(i)} + \mathbf{v}_4^{(i)})\,, \tag{1.6}$$

where $\mathrm{d}t$ is the time step and the velocities $\mathbf{v}_1^{(i)}, \mathbf{v}_2^{(i)}, \mathbf{v}_3^{(i)}, \mathbf{v}_4^{(i)}$ are the induced velocites of partial steps:

$$\mathbf{v}_1^{(i)} = \mathbf{v}_{\text{full}}^{(i)}(\mathbf{s}_i, t)\,, \tag{1.7}$$
$$\mathbf{v}_2^{(i)} = \mathbf{v}_{\text{full}}^{(i)}(\mathbf{s}_i + \mathbf{v}_1^{(i)}\mathrm{d}t/2, t + \mathrm{d}t/2)\,, \tag{1.8}$$
$$\mathbf{v}_3^{(i)} = \mathbf{v}_{\text{full}}^{(i)}(\mathbf{s}_i + \mathbf{v}_2^{(i)}\mathrm{d}t/2, t + \mathrm{d}t/2)\,, \tag{1.9}$$
$$\mathbf{v}_4^{(i)} = \mathbf{v}_{\text{full}}^{(i)}(\mathbf{s}_i + \mathbf{v}_3^{(i)}\mathrm{d}t, t + \mathrm{d}t) \tag{1.10}$$

Lower-order schemes such as basic Euler method is also implemented in code, however, not recommended to use. More on this is discussed in Results part of thesis.

The time step $\mathrm{d}t$ is chosen so that the vortex ring cannot move faster than a 1% of its size in a single step. As we will see later in Results, in case of vortex rings with changing radius, the time step $\mathrm{d}t$ has to be iteratively changing after each break of the above rule:

$$\mathrm{d}t \leftarrow \frac{0.01R}{|\mathbf{v}_c|} \tag{1.11}$$

1.3 Re-segmentation of vortex

Since the CPU time cost of a vortex simulation rises rapidly as the number of segments N increases, it is importnant to manage this number so that the simulation can run as fast as possible. As the distance between neighbouring segments is compressed/enlarged with time, there is need to remove/add segments to conserve the mesh resolution. The closeness (in terms of arclength) of neighbouring segments has to be measured at each step and the criteria for *re-segmentation* heavily depends on the simulation itself and a scale at which we desire to observe any phenomena.

The simplest re-segmenting criteria is to attempt to keep an approximately *uniform distance* between the segments. To ensure this, two important conditions were implemented:

1. The segment $\mathbf{s}_{j+1}$ would be removed if:

$$|\mathbf{s}_{j+1} - \mathbf{s}_j| < \delta_{\min}\,, \tag{1.12}$$

   where $\delta_{\min}$ is the minimal distance between two segments. Also, the segment $\mathbf{s}_j$ would take place somewhere between $\mathbf{s}_{j-1}$ and $\mathbf{s}_{j+2}$ so that it will conserve the curvature of vortex. Such result can be obtained using any spline interpolation along nearest neighbours. We worked with 3D local spline using 4 points (in our

context they would be $\mathbf{s}_{j-2}$, $\mathbf{s}_{j-1}$, $\mathbf{s}_{j+1}$, $\mathbf{s}_{j+2}$) and create another 11 interpolated knots. Consequently, the new position of $\mathbf{s}_j$ will sit on the 6-th (the middle one) knot and also.

2. In a similar manner, we add a new segment $\mathbf{s}_{\text{new}}$ between $\mathbf{s}_j$ and $\mathbf{s}_{j+1}$ if:

$$|\mathbf{s}_{j+1} - \mathbf{s}_j| > \delta_{\max}\,, \tag{1.13}$$

where $\delta_{\max}$ is the maximal allowed distance between two segments.

This will keep all the distances along the vortex rougly in the range $\delta_{\min}$ to $\delta_{\max}$ and also will keep the geometrical properties. However, when adding/removing points from the segment array, one has to also shift the indices of all segments, which adds unnecessary complexity into the simulation.

1.4 Future implementations

To make `Pyvort` a full-fledged quantum vortex simulation, there should be implemented following improvements:

Complexity speedup

Recent numerical research presented [**?**] a new numerical method to compute the evolution of vortex filament. The method is based on a *tree algorithm* which considerably speeds up the calculation of Biot-Savart integrals - computational cost scales as $\mathcal{O}(N\log(N))$ rather than N^2. Properties of the tree method was tested for a variety of vortex configurations, ranging from simple vortex rings to a counterflow vortex tangle and compared with the LIA approac and the exact Biot-Savart's law.

Implementation of such algorithm is not easy, but definitely worth to obtain higher precision.

Re-connection process

If any two lines become very close (less than the minimal separation $\delta_{\min}$ along the line) the filaments can reconnect, changing the topology of the system. Many researchers claim this can happen, from analogies with vortex dynamics in the Navier–Stokes equation.

Also, numerical studies of Gross–Pitaevskii equation showed that quantized vortices can reconnect.

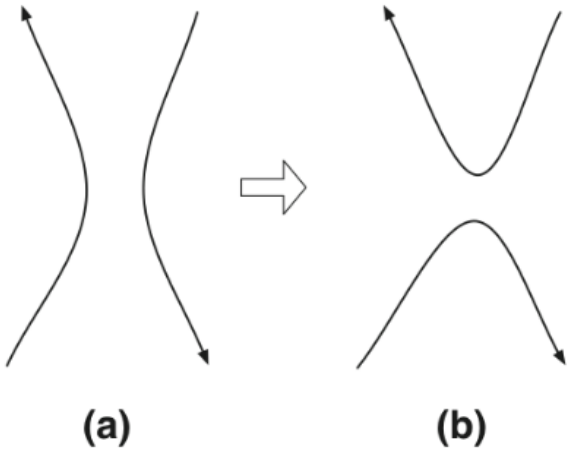

Figure 1.5: Reconnection of quantized vortices. **(a)** Two vortices before reconnection, about to contact each other. **(b)** The new vortices after reconnection.

The VF model itself cannot describe the reconnection process because the vortex core structure is neglected. Hence, some artificial procedures must be introduced to simulate such process. For instance, when two vortices approach within a critical distance $\delta_{\min}$, we will artificially reconnect the vortices.

The main criteria for reconnection is that the total length (this is corresponding with energy) will decrease. Self-reconnections (e.g. caused by a twist of vortex) would be treated in the same way. Since recconection ivolves only antiparallel vortices, one has to check using the inner product whether two vortices could physically reconnect or not.

High-order tests

Once the code of interacting vortex filaments is developed, one has to face with the analysis of all that mass of data. The simplest measurable quantity is the total lenght of all vortex lines. In a finite volume this would be $L = (1/V)\int \mathrm{d}\xi$ for the vortex line per unit volume.

Of course, there can be defined more complicated metrics, measuring the isotrophy of the vortex tangle. One of them is the length of line projected along a given vector $\hat{\mathbf{r}}$, give as:

$$J(\hat{\mathbf{r}}) = \frac{1}{VL}\int_{\mathcal{L}} \sqrt{1-(\mathbf{s}'(\xi)\cdot\hat{\mathbf{r}})^2}\mathrm{d}\xi \tag{1.14}$$