Vortices in Superfluid Helium

Ali Ehlen, under Professor Rena Zieve

REU at UC Davis, Summer 2012

Abstract

This paper summarizes the work I did in Professor Rena Zieve's lab at UC Davis during the summer of 2012. I continued her "vibrating wire" project, which focuses on understanding fluid vortices using circulating superfluid helium in cylindrical brass cells. For the first half of the summer, I built cells to test the stability of these vortices, and for the second half of the summer, I wrote a program to numerically calculate their behavior. This paper begins with the motivation for the project and the basics of the vibrating wire experiment, and then details my contribution in two parts: cell construction and vortex calculations.

1 Introduction

Vortices are important in many fluid dynamics applications. Because their presence affects the motion of any body in a fluid, the influence of vortices is a crucial consideration in varied fields such as astrophysics, meteorology, aerospace engineering, and more. In Professor Rena Zieve's lab at the University of California, Davis, we study the behavior of vortices in simplified systems—superfluids—in order to better understand how they operate in the context of more complex systems.

A vortex is defined by circulation, or nonzero fluid velocity around a vortex core, as shown in Fig. 1. This can be expressed mathematically as

$$\kappa = \oint v \cdot dl, \tag{1}$$

where the integral is taken over a loop enclosing the core, and κ is the circulation and $v \cdot dl$ is the velocity of the fluid tangent to the closed loop. From this equation, it can be seen that for any type of nonvortex flow, such as diverging flow or unidirectional flow, the velocity components around the closed loop will cancel, so the circulation κ will be zero.

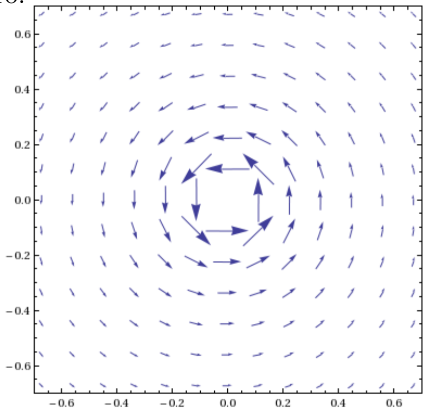


Figure 1: The velocity field around a vortex core at (0,0).

For a given vortex system, κ is constant. For this to be true, the velocity of fluid closer to the core must be higher than the velocity of the fluid further away. This relationship is pictured in Fig. 1, and is consistent with common intuition about vortices.

However, a description of vortex motion becomes complicated quickly when friction within the fluid is taken into account; this is why superfluids are useful. Superfluids flow with no viscosity, and so dissipate no energy as they move. This lack of friction and energy dissipation simplifies their flow patterns. In addition, superfluid circulation κ is limited to integer values for a given single-vortex system. To transition between these integer (quantum) circulation states, a vortex must gain or lose energy in specific, allowed amounts. The combination of zero frictional dissipation and quantized circulation results in very stable vortices. These are some benefits of studying superfluid systems; undisturbed stationary superfluid vortices in the ground state can persist indefinitely, and superfluid vortex behavior is much simper than normal fluid vortex behavior. Once the behavior in a superfluid medium is understood, it can be generalized to more complicated fluids.

1.1 Experimental Setup

Our process for measuring vortex behavior can be broken into three components: the cryostat, the cell, and the measurement system.

The cryostat, shown in schematic form in Fig. 2, cools the helium to create a superfluid. Helium is a gas at room temperature, but liquefies below 2.4 K. As it is cooled further, an increasingly greater portion of the fluid becomes superfluid. For this reason, we use the cryosystem to cool our helium down to below 1K to ensure that most helium involved in the experiment is superfluid.

The most important piece of the fridge is the cell, which sits in a constant magnetic field at the bottom of the apparatus. A close-up of the cell is shown in Fig. 3. Each cell is two inches long, with a hollow cylindrical brass body and caps on either end made of Stycast, a clear epoxy. A 16 μm diameter superconducting niobium-titanium (NbTi) wire runs lengthwise through the center of the cell, as shown in the Fig. 4a.

During a run, we fill the cell with cooled helium through an inlet hole in one of the caps, and rotate the cryostat, inducing circulation in the helium. The flow pattern is initially a complicated matrix of vortices, but in most cases, all will annihilate but a single vortex around the central wire, as shown in Fig.4b. In this setup, the vortex is in its quantum ground state, with the wire acting as its core.

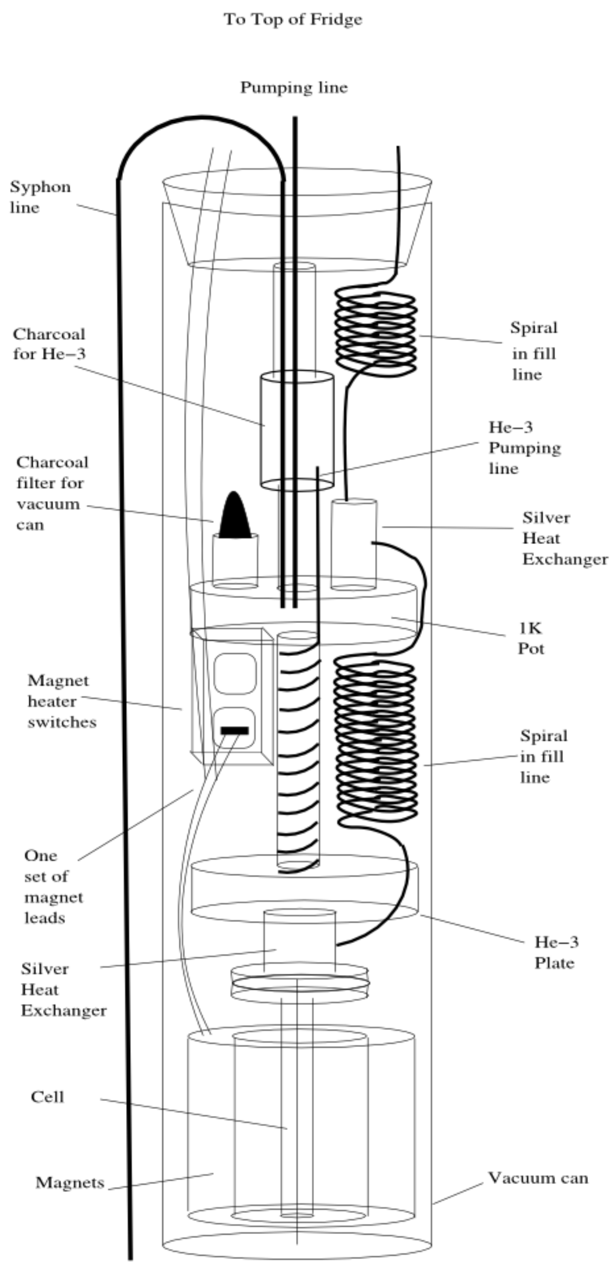


Figure 2: Schematic of the fridge that cools down helium for experiments [1].

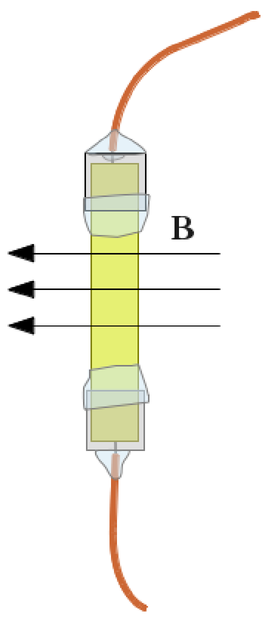
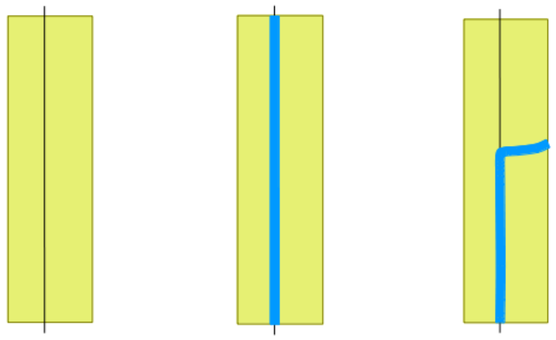


Figure 3: Schematic of a basic cell.

Because a superfluid vortex is so stable, it is necessary to perturb the system to observe interesting dynamics. We do this either by jerking the cryostat (adding kinetic energy), or by heating up part of the cell (adding thermal energy). In the ideal case, the extra energy causes one end of the vortex to move off of the central wire and attach to the wall of the cell. This configuration is pictured in Fig. 4c, and allows us to take measurements of the interior of the cell.



(a) Internal view of cell body with wire, and no vortex.

(b) Internal view of cell body with a straight vortex covering the wire.

(c) Internal view of cell body with a partially free vortex on the wire.

Figure 4: Different configurations of the vortex discussed in this paper.

1.2 Measurement of Vortex Decay

Due to the section of the vortex remaining on the wire, helium continues to circulate around the cell. The length of the vortex that extends from the wire to the cell wall—the free part of the vortex—exists in this circulating helium and is swept around the cell accordingly. This phenomenon is called vortex precession, and allows the vortex to dissipate energy. As the free section of the vortex precesses around the cell, it winds down the wire. Eventually, the length of the trapped vortex will shorten until it vanishes entirely, and the helium in the cell will stop circulating. To gain insight into vortex dynamics, we study the behavior of this free vortex, and how quickly it decays.

An example of data is shown in Fig. 5. This is a plot of circulation in the cell; as the vortex winds down, the length of the trapped section of the vortex shrinks, which decreases the circulation in the cell. Each data point in Fig. 5 represents a circulation value, and there are several steps involved in obtaining each point.

To find the value of circulation in the cell, we run a pulse of current through the NbTi wire, and measure the voltage across it. The initial pulse of current moves charges up the wire through the magnetic field, inducing a Lorenz force perpendicular to both the direction of the magnetic field and the length of the wire. This force "twangs" the wire, and starts it vibrating side-to-side. This oscillating sideways motion of the wire through the magnetic field causes another Lorenz force on the charge carriers, this time up through the wire. The resultant movement of charge carriers is manifested as a readable voltage across the NbTi wire.

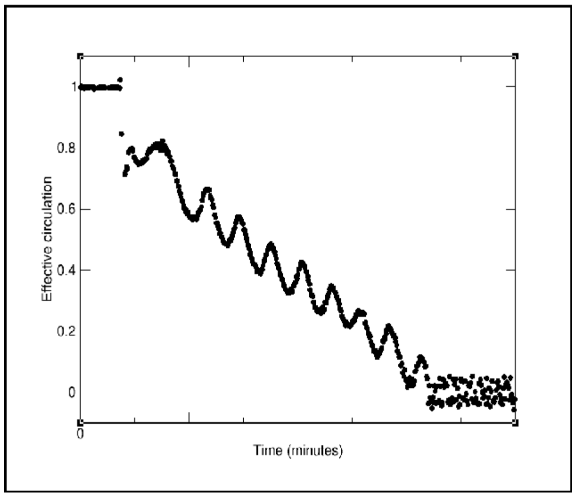


Figure 5: Circulation in the cell over time. As the vortex shrinks, there is less circulation in the cell [1].

However, the wire's plane of vibration is affected by the helium circulating around it. Helium pushing on the wire causes its plane of vibration to rotate, resulting in a periodic change in the measured voltage. The wire vibrates initially in a plane perpendicular to the external magnetic field, resulting in the maximum Lorenz force on the wire's charge carriers, and the maximum possible induced voltage. Once the plane of vibration rotates to become parallel with the magnetic field, the charge carriers feel no force because they are moving in the same direction as the magnetic field. When this occurs, the output voltage drops to zero. Then, as the plane of vibration continues to rotate, measured voltage continues to rise until it reaches another peak when the plane of vibration is again perpendicular to the magnetic field. In addition to this, as the wire moves, the vibration is damped both by materials in the wire and by the helium surrounding it, resulting in the decay of the output voltage signal.

To obtain relevant data from this signal, it is broken down by a computer program into its constituent sinusoidal and exponentially decaying components. The period of sinusoidal voltage oscillation due to rotation of the wire's plane of vibration is related to the circulation in the cell. Therefore, the program finds the period of the sinusoid and converts this to a circulation value, which is proportional to the amount of vortex left on the wire. This process of calculating circulation based on the beat frequency of the vibrating wire produces each point on the plot in Fig. 5.

Other than the eventual decay of the vortex system, an important feature of Fig. 5 is the oscillation in circulation as the vortex winds down. This phenomenon results from the fact that the NbTi wire is never exactly in the center of the cell. Because of this, as the free section of the vortex precesses around the cell, it moves between areas where the wire is closer to the wall of the cell and areas where the wire is farther away. When the free vortex moves to an area where the wire is farther from the cell wall, the free vortex must span a longer distance. To compensate, some of the vortex that was trapped around the wire peels off to become free vortex. Decreasing the length of vortex around the central wire also decreases circulation in the cell, causing one of the local minima in Fig. 5. Similarly, when the free portion of the vortex precesses to an area where the wire is closer to the cell wall, some of the free vortex moves back onto the wire, increasing the length of trapped wire, and therefore increasing the circulation in the cell. This corresponds to a local maximum in Fig. 5.

1.3 Double-Diameter Cells

In order to study the dynamics of the vortex during this precession, it is often useful to know which end of the vortex separated from the wire to attach to the cell wall. We do this by using modified cells with a double-diameter shape, where the diameter of the cell changes about halfway down the cell, as shown in Fig. 6.

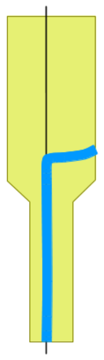


Figure 6: Schematic of a double-diameter cell. The black in the NbTi wire and the blue is the vortex.

This change in diameter is useful because the average velocity of helium within a small radius of the core is larger than the average velocity of helium included in a larger area. Because the free section of the vortex precesses at a rate near the average velocity of the fluid carrying it, a vortex in a thin cell precesses faster than a vortex in a wider cell. Thus, as the vortex precesses through the change in diameter in the double-diameter cells, the period of precession will change. This is apparent in the data shown in Fig. 7, where the vortex starts precessing on the wide end of the cell (longer precession period) and transitions to the thinner end (shorter precession period). Additionally, the precession period is predictable given a cell diameter, so it is possible to determine the position of the vortex by looking at just a small chunk of precession data.

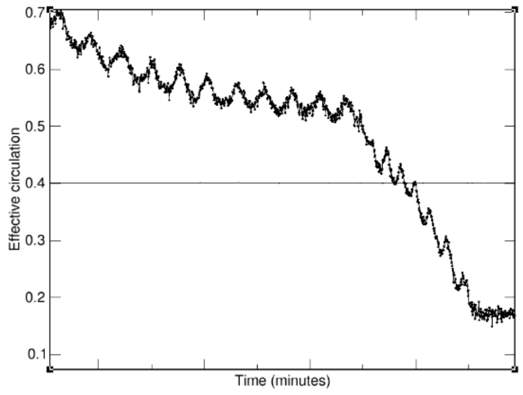


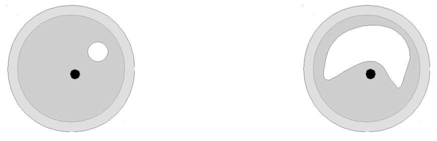
Figure 7: Circulation as a function of time as a vortex precesses around the cell. The vortex starts precessing in the wider end of the cell (as indicated by the longer precession period in the first half of the graph), then transitions to precessing in the thinner end. [1].

This summer, I worked with these double-diameter cells. My work was split roughly into two phases. For the first half of the program, I built new double-diameter cells for testing in the cryostat. When it became clear that helium was unavailable and so testing was not possible, I started a second piece of the project, working on a simulation to better understand the behavior of vortices in these oddly-shaped cells.

2 Building Cells

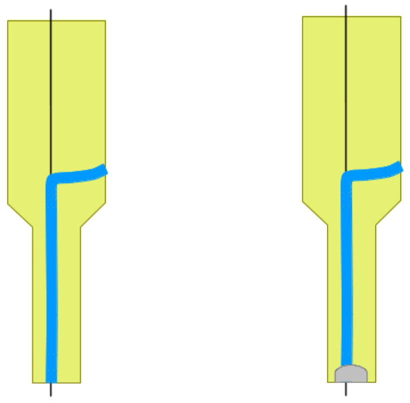
A recent Ph.D. student, Ingrid Neumann, ran tests to explore the effects of the cell cap shape on stability of the vortex. She did this by testing cells with differently-shaped Stycast caps, then—taking advantage of the double-diameter nature of the cell—recorded which end of the cell the vortex came off after the brief addition of energy. She had hoped to find a correlation between the shape of the caps and how much energy was necessary to knock the vortex off of the wire. However, she found that, regardless of cell cap shape, the vortex came off of the wire at the end of the cell with the helium inlet hole. It appears that the helium entering the cell was creating enough turbulence that almost every vortex was less stable near the inlet.

However, the inlet holes in her cells were small, meaning that helium entered the cell at high velocities. This summer, my initial project was to replicate some of Ingrid's tests, but building cells with much larger inlet holes; for a schematic, see Fig. 8. The idea was to be able to finally test the dependence of vortex stability on cap shape, rather than placement of the helium inlet hole. To this end, my goal this summer was to build four cells; two with a flat cap and two with a "bump" cap; see Fig. 9.



(a) The type of inlet holes in Ingrid's cells. (b) The larger inlet holes in the caps of my cells.

Figure 8: Bird's eye view of the different types of helium inlet holes in the Stycast end caps. The black dots are the NbTi wires coming out of the page, and the white sections are the helium inlet holes.



(a) Flat. (b) Bump.

Figure 9: Two different end cap geometries. Note that the inlet hole is in the end cap with unchanging geometery.

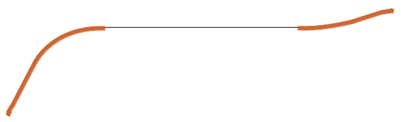
The procedure I followed to build the cells is pictured schematically in five steps Fig. 10. First, a length of 187-strand NbTi wire must be stripped of its insulation, copper matrix, and most of its constituent strands. This leaves a section of a single NbTi strand just longer than the length of the brass body of a cell, as in Fig. 10a.

Then, one end of the wire is glued to a shaped cap (flat or bumped), making sure that the Stycast seals around both the thin and thick parts of the wire. Using the weight of the wire to keep the setup in place facilitates this process, and is shown in Fig. 10b.

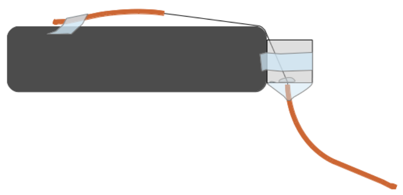
The wire-and-cap combination is then attached to the brass body of the cell. At the same time, the other cap end (the cap with the helium inlet hole) is also epoxied onto the body. While this dries, the unfixed end of the wire rests partially fed through the inlet hole cap, but not taut; this is shown in Fig. 10c.

The most delicate step is pulling the wire taut; this is achieved by taping the cell upright to a shelf, and hanging a weight to the free end of the wire, as in Fig.10d. This end of the wire is then glued in to the cap, with caution exercised to ensure that the inlet hole does not become blocked.

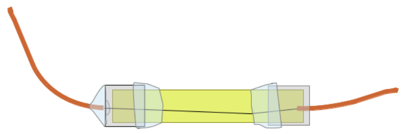
If this step is completed without the wire breaking, then a brass connector and some extra wiring are attached; these can be seen in the photograph of three completed cells in Fig. 11.



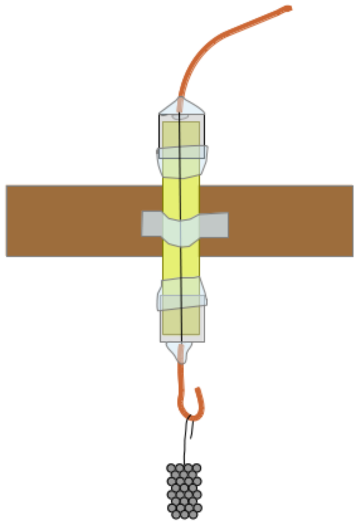
(a) First, one two-inch section of the insulated copper-coated NbTi is stripped to a single NbTi strand.



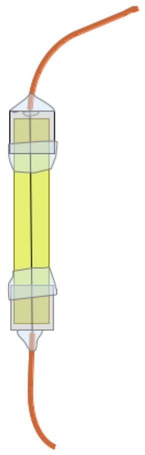
(b) One end of the single strand is glued (with Stycast) to the shaped cap (flat or bumped). The wire and cap are secured to the lab bench with tape, to keep them in place while they dry.



(c) The cap-wire combination is glued to one end of the brass cell, while the cap with the inlet hole is glued to the other. Note: the wire is shown here to demonstrate that it is not taut at this stage of the building process, but in reality, it is inside the brass cylinder.



(d) When the caps are fixed onto the body, the body is taped to a shelf and a weight hung to the end of the wire to keep it taut while it is glued to the inlet-hole end cap.



(e) A finished cell. Again, the wire is shown to demonstrate the fact that it is taut within the cell.

Figure 10: The steps involved in constructing a cell.

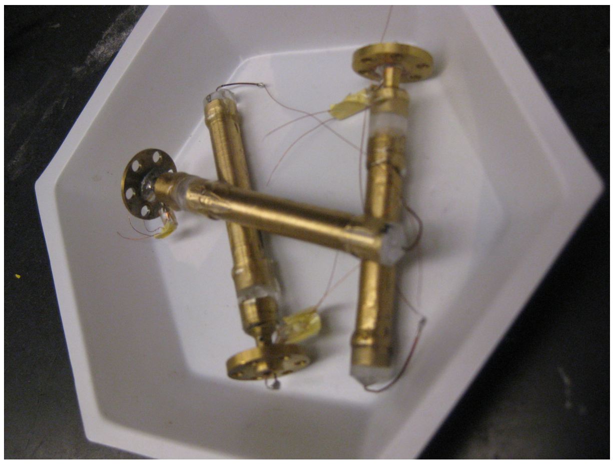


Figure 11: A photograph of three completed cells, including brass attachments that allow them to be screwed on to the cryostat. One of these is from a few years ago; I built the other two.

The Stycast epoxy takes around 12 hours to dry, and each step, particularly the fourth, runs a high risk of breaking the wire. Because of this, despite many attempts, I was able to successfully build one cell and nearly finish another. I checked the completed cell for leaks both and room temperature and at 77K (liquid nitrogen temperature). The other may be functional, but because of attempted repair of a crack in one of the Stycast caps, the helium inlet hole may be plugged.

However, we were not able to obtain the helium we needed to run tests with these cells. So, for the second half of the summer, I worked on a more theoretical piece of this project, to help build up the group's vortex simulation code.

3 Calculations

The lab group has a set of solvers and time-steppers used to solve for the velocity fields around vortex cores in different configurations. This allows us to understand better what is happening inside of a cell; well-written simulations will produce data that matches experiment, and allow for deeper probing into the dynamics of the vortex. For example, we cannot to look inside our cryostat to see the position of the vortex, but we can follow the motion of a simulated vortex directly.

It is possible to analytically solve for the velocity field around a vortex running through or near the center of a straight cylindrical cell. The lab group had also written code to solve for the velocity field around a partially-free vortex, but only in a straight cylindrical cell. In addition, a simulator can step this solver through time, to follow the motion of the vortex around the cell. This code has successfully reproduced experimental results. However, we did not have any code that could solve for the velocity field around a partially-free vortex in a double-diameter cell; this was my project for the rest of the summer.

Finding the velocity field in the cell involves several steps. First, given a particular vortex position, the velocity field around that vortex (ignoring the influence of cell walls) must be found. From now on, this will be called the "initial velocity field." This initial field must then be modified to satisfy the boundary conditions imposed by the walls of the cell.

To achieve this, another field (from now on called the "secondary field") that solves the Laplacian $0 = \nabla^2 v$ within the cell must be found. Because the velocity field around a vortex core has zero curl everywhere but the core, any solution to the full problem will also solve the Laplacian. Adding another solution to the Laplacian will not change the fact that the overall system solves $0 = \nabla^2 v$. Therefore, under these conditions, it is valid to add another field to take into account the effect of the cell walls.

The ideas in the previous two paragraphs can be combined in the following manner to find a full solution. For the vortex to produce a physically possible system, no fluid can flow in or out of the walls of the cell. This means that the components of fluid velocity perpendicular to the cell walls must be zero. This is achieved by calculating the perpendicular components of the initial velocity field at the cell wall, then using their opposites as boundary conditions to find the secondary field. Thus, when the initial and secondary fields are added together, their sum will solve the Laplacian (satisfying conditions for a vortex field) and have no velocity components perpendicular to the cell walls at the cell walls (satisfying conditions required of a physical system).

To complete these steps, I used code from several sources. I wrote a batch of programs in Python to construct a cell of user-determined dimensions (single- or double-diametered) and output this cell in various formats. An example of a virtually constructed cell is shown in Fig. 14. The lab group had written a program called **boundaryValueDouble**[1] in C++ to find the components of a vortex velocity field perpendicular to the cell wall. I modified this to be compatible with the double-diameter cells. I also found **depSolver**, an open source Laplace solver written in C which takes in files of boundary conditions and solves the Laplacian under those constraints. By the end of the summer, the combination of these produced velocity fields that looked reasonable, and could be modified for various cell dimensions and vortex configurations.

3.1 depSolver

depSolver is a robust, open-source program I found online[2]. Its stated purpose is to solve Laplace's equation for

[1]I will notate names of programs in boldface, and names of data-containing text files in italics.
[2]For source code and associated documents, see code.google.com/p/depsolver/wiki/FullInstall
[3]https://code.google.com/p/depsolver/

3-dimensional electrostatic problems, using the boundary element method (BEM)[3]. It takes input files that set the electric potential at boundary points in any three-dimensional shape. It can also take information about the material within that boundary, specifically the dielectric constants and interfaces between different dielectrics. Lastly, it takes a file of internal points of interest. It then calculates the potential, electric field, and electric Lorenz force at each of these points.

We are not attempting to solve an electrostatics problem. However, the calculations that **depSolver** performs are useful to us. If we set a constant dielectric (that of a vacuum), we can take advantage of the program's calculation of electric potential. If there is no free charge in a region, the electric potential in that region solves the Laplacian. Therefore, if we give **depSolver** boundary conditions that it can read in as values of electric potential, we can use its solution for potential at the internal points as the velocity field we are looking for. However, because potential is a scalar, we can't enter velocity in vector form. Therefore, because each component of velocity also solves the Laplacian separately, we can enter each component to **depSolver** separately. The components can be combined into vectors later.

The core of **depSolver**'s calculations lies in Gaussian quadrature of given boundary elements, where boundary elements are geometric breakdowns of the boundary surface. In this application, these elements are always triangles[4]. This process finds the best points at which to evaluate for a solution within each element[5]. Because Gaussian quadrature does not produce accurate results if one part of an element is singular, **depSolver** checks for singularity at the nodes of each triangle. If it determines that one of the nodes of an element is singular, it changes the geometry of the problem. For weakly singular points, it transforms the triangle into a degenerate square to perform a different kind of integration (Gauss-Jacobi integration). If it detects that one node is a strongly singular point, **depSolver** splits up the element into four smaller triangles by connecting points on each edge.[6] It then performs Gaussian quadrature on the three subtriangles that are not adjacent to the singular node, and splits the remaining subtriangle into four even smaller triangles. It repeats this process a specified number of times. The final smallest triangle adjacent to the singular point will have a value near zero, and is neglected.

Given this basic understanding of how **depSolver** performs calculations, I will describe the structure of the program I wrote to combine **boundaryValueDouble**, **depSolver**, and the double-diameter cells.

3.2 Structure of the program

The basic flow of the program is this: construct a virtual cell and choose a vortex configuration. Use the vortex configuration to calculate the initial velocity field around the vortex core at a specified mesh of points. Then, calculate the velocity at the points that make up the cell boundary, and find the components perpendicular to the cell wall. Separately, send each Cartesian component of these perpendicular components to **depSolver**, labeling them as electric potential. Take **depSolver**'s solution for potential at the mesh of internal points for each component, and add them together to create a three-dimensional secondary velocity field. Finally, add this solution to the initial velocity field. This will cancel all components perpendicular to the cell wall, resulting in the velocity of helium around the superfluid vortex core for a given vortex configuration.

All of this is accomplished by executing one bash script that runs each of the programs in turn, and passes the appropriate information between files. For a more detailed understanding of this process, I will discuss each step in more depth. A visual of the program flow is also provided in Fig. 12.

[4]This is not a detailed description of **depSolver**'s inner workings, nor do these elements always have to be triangles specified by three points. For further detail, see the user's manual at https://code.google.com/p/depsolver/downloads/detail?name=depSolver_UserManual.pdf&can=2&q=

[5]For more information on Gaussian quadrature, see http://en.wikipedia.org/wiki/Gaussian_quadrature and http://mathworld.wolfram.com/GaussianQuadrature.html

[6]Triangles may be split by connecting the midpoints of each edge, but I have not seen the code for this part of the calculation, so I do not know for sure.

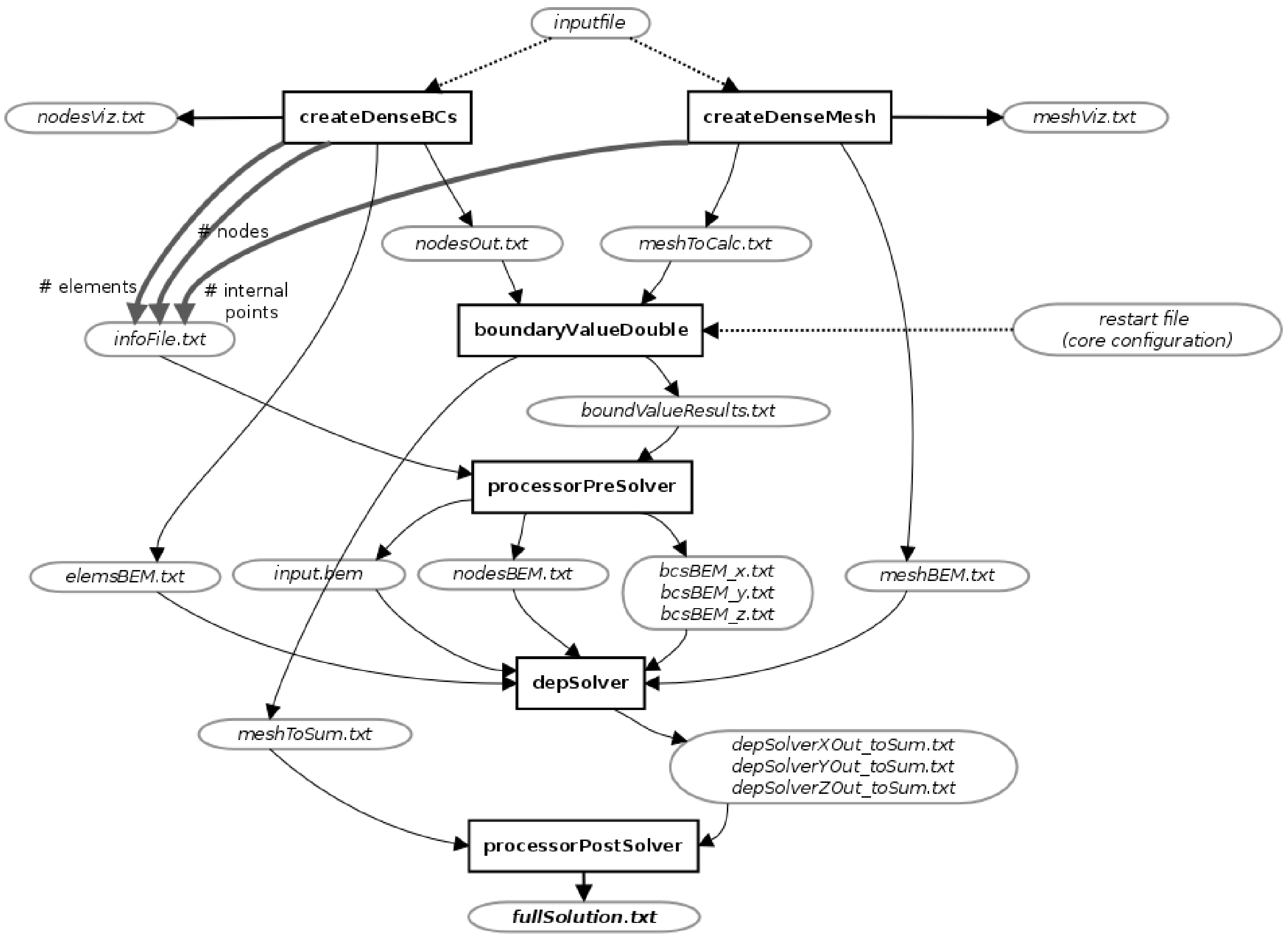


Figure 12: The flow of the program, run by a single bash script. As usual, program files are indicated in boldface and text files in italic. Dotted arrows represent outside or input files used by the program. Solid black lines indicate the flow of text files between programs (from the file that creates them to the file that uses them), and thick gray arrows represent information being added to a file.

3.2.1 Setup: Cell, Vortex Configuration, and Internal Points

There are two main Python programs that create the initial setup of the problem by constructing a virtual cell and a mesh of points internal to the cell at which the velocity will be calculated. These are **createDenseBCs** and **createDenseMesh**, and they each output several files that are used in conjunction with a few other, external files to start calculations.

createDenseBCs creates the files that will eventually act as boundary conditions in **depSovler**. It reads in parameters from a user-manipulated text file called *inputfile*. These parameters specify the dimensions of the cell to be created, including the height of the cell and the diameter of the lower and upper parts of the cell (which can be set to the same number for a straight cylindrical cell). The user can also specify the density of the nodes that will make up the cell's boundary.

createDenseBCs takes these parameters and creates a cell in cylindrical coordinate system (see Fig.13). Using the height of the cell and the number of z-values at which nodes will be created (numZ, specified in *inputfile*), **createDenseBCs** determines the spacing of these z-values. It then iterates through 2π radians at each z-value and creates nodes at user-specified number of θ-values (numR) by calculating and recording their Cartesian coordinates.

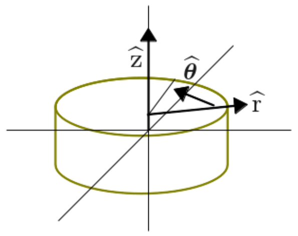


Figure 13: The cylindrical coordinate system used in **createDenseBCs**.

Because the purpose of this program is to determine the behavior of a precessing vortex around the transition between the two diameters of the cell, it is useful to include more boundary nodes near the diameter transition. To this end, the user can specify an attentionRange in *inputfile*, in which the density of nodes is higher (the factor by which the density is multiplied is also specified in *inputfile*). As **createDenseBCs** iterates through z-values, it will check if it is in the attentionRange. If it is, it will add nodes at the specified higher density.

The program also creates nodes at the top and bottom faces of the cell. It does this by iterating through r-values to create nodes at a single z-value (the top or bottom of the cell). The number of circles of nodes it creates is determined by the value of numdR in *inputfile*.

All of this creates nodes in the cell shape specified by the user. An example of this is shown in Fig. 14.

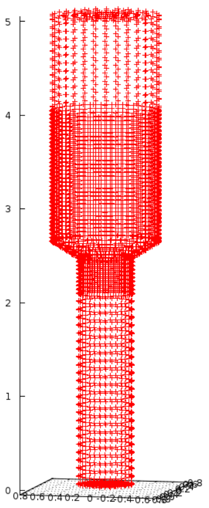
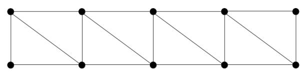


Figure 14: Visualization of double-diameter cell walls. The midsection contains more points, because this is the section where we are interested in finding accurate data.

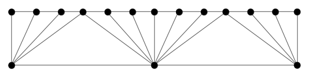
The final step in creating the virtual cell is creating boundary elements. For the most part, the creation of these triangles is a simple process. Each row (one z-value) of nodes is usually neighbored by a row with the same number of nodes, and so can be connected as shown in Fig. 15a. Complications arise, however, at the rows on the edge of the attentionRange. These must be connected differently, which is shown in Fig. 15b. In addition, the connection of elements on the center of the faces of the cell is not straightforward, and is also shown in Fig. 15c.

The second program involved in setting up the calculations is **createDenseMesh**. This is the program that generates the internal points at which the velocity field will be calculated, always including points on the boundary. It operates in much the same manner as **createDenseBCs**, but at every z-value, it creates several circles, again specified by numdR.

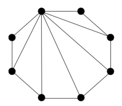
Both of these programs, **createDenseBCs** and **createDenseMesh**, output several files. Some are formatted to be read into **boundaryValueDouble** (these are the boundary nodes *nodesOut.txt* and the internal mesh *meshToCalc*); some are formatted to be read directly into **depSolver** (these are *elemsBEM.txt* and *meshBEM.txt*, which specify the boundary elements and internal mesh nodes); some are formatted to be plotted by gnuplot for a visual check to determine if the program is working (these are *nodesViz.txt* and *meshViz.txt*). In addition, parameters from these programs like the number of nodes in the boundary, the number of elements in the boundary, and the number of nodes in the internal mesh are inserted into a file called *infoFile.txt*, which is used later by **depSolver**.



(a) Connection of nodes into triangle elements, between most rows of nodes on the cell boundary.



(b) General connection of nodes in rows on the border of the attentionRange into triangle elements.



(c) General connection of nodes on the faces of the cell into triangle elements.

Figure 15: Various methods of connecting different rows of elements.

3.2.2 Processing Setup to Send to Solver

The first substantial calculation is performed by **boundaryValueDouble**. This program was written by my predecessors in lab, but I modified it to fit the needs of this project. It takes in boundary and internal mesh information (*nodesOut.txt* and *meshToCalc.txt* from **createDenseBCs** and **createDenseMesh**), as well as a vortex configuration. This configuration is either straight (around the wire) or off the wire (precessing), and comes from existing files. The purpose of **boundaryValueDouble** is to calculate the velocity of helium at different points around a particular vortex core configuration. It uses the fact that the velocity field around a vortex core is analogous to the magnetic field around a current-carrying wire. Because of this, it can find superfluid velocity values using the law of Biot-Savart.

These calculations are called in **boundaryValueDouble** twice. First, it takes the internal mesh of points and finds the initial velocity field around the vortex. This is output in *meshToSum.txt*. Second, it takes the boundary nodes and calculates the velocity normal to the cell wall at each. This information it output in *boundValueResults.txt* and will be used to find the secondary velocity field.

Once these calculations are done, there are only a few remaining processing steps required to prepare the information to be sent to **depSolver**. The Python program **processorPreSolver** takes in *infofile* (the file with the number of boundary nodes, boundary elements, and internal mesh nodes) and *boundValueResults.txt* and outputs a *nodesBEM.txt* file that lists the boundary nodes formatted properly for **depSolver**, and *bcsBEM_x.txt*, *bcsBEM_y.txt*, and *bcsBEM_z.txt*, which are the x, y, and z components of the velocity field normal to the cell wall formatted in a way such that **depSolver** can read them in as electric potential. It also creates a file called *input.bem*, which has information from *infofile*, formatted in a way **depSovler** can read.

3.2.3 Calling depSolver and Calculating the Final Velocity Field

Finally, **depSolver** is called three times, once for each Cartesian component of the velocity field. As stated above, **depSolver** requires an input file that gives it information about the number of nodes, elements, dielectrics, etc. involved in the problem (*input.bem*), a file of boundary nodes (*nodesBEM.txt*), a file of boundary elements (*elemsBEM.txt*), a file of the potential at each of the boundary nodes (*bcsBEM_x*, *_y*, or *_z*), and a file of internal nodes (*meshBEM.txt*). It uses this information to solve Laplace's equation at the internal mesh nodes under the constraint of the given boundary conditions.

Once results are returned from all three calls of **depSolver**, in the files *depSolverXOut_toSum.txt*, *depSolverYOut_toSum.txt*, and *depSolverZOut_toSum.txt*, the solution is almost complete. These three components of the Laplace solution (secondary velocity field) must be combined and added to *meshToSum.txt* (initial velocity field). These are summed in the Python program **processorPostSolver** and output as *fullSolution.txt*. This is the velocity field within the cell.

3.2.4 A Note

My goal this summer was to create a program that produced a reasonable solution, which I checked visually by plotting the final field using gnuplot. Once the program started producing results that looked reasonable, I intended to compare them with the known solutions for a straight or partially-free vortex in a straight cell. Unfortunately, I ran out of time. I was able to reach a point where the solutions looked reasonable (see Fig. 16 for an example), but could not verify this numerically. I did, however, check **depSolver** by manipulating provided example problems and solutions for the potential and electric field within parallel-place capacitors; in this context, **depSolver** seemed to be functioning correctly.

4 Conclusion and Future Work

At the conclusion of this project, there are still several pieces left to be finished. In terms of cell construction, the rest of the four initially planned cells should be built, tested, and compared with Ingrid's results. This will allow the geometry-dependence of vortex stability to be determined. To complete the coding project, the accuracy of the solver needs to be verified, and then it should be integrated into existing simulation code to step it through time. This will allow for interesting analysis of vortex dynamics in the double-diametered cells, particularly at the point when the vortex jumps between cell diameters—the experimental data contains certain signatures at this transition that we do not understand.

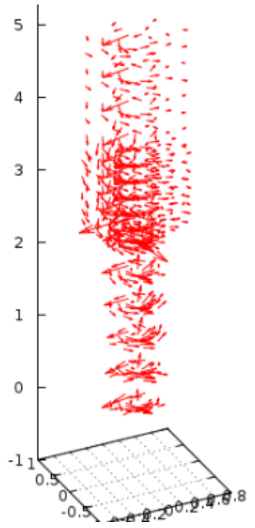


Figure 16: An example solution for a straight vortex in a double-diameter cell, plotted using gnuplot.

5 Acknowledgments

Thank you, Rena, for all your patience. In addition, thank you to Luke Donev [1] and Ingrid Neumann [2]; most of the information in the introduction came from their undergraduate and graduate theses.

References

[1] Luke Donev. Experimental methods and results on the study of superfluid helium.

[2] Ingrid Neumann. *Interactions between a superfluid vortex and its bounding surface.* PhD thesis, UC Davis, 2012.