Theory and applications of helicity

Submitted by Jack Robert Campbell to the University of Exeter
as a thesis for the degree of
Doctor of Philosophy in Mathematics
In February 2014

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

Signature: .....

Acknowledgements

I must thank my supervisor, Mitchell Berger, for his keen insight, ideas and informed advice. It would not have been possible for me to complete this thesis without all of the kind help and support from my friends and family – it is much appreciated! Also, I thank Walter Gekelman and his team at UCLA for granting me access to their data. I thank the Leverhulme Trust for the studentship that made this work possible. Last, but certainly not least, I wish to thank Claire for her support, encouragement, understanding and love x.

Abstract

Linking number, writhe and twist are three important measures of a curve's geometry. They have been well studied and their definitions extended to open curves situated between two horizontal planes [1]. However, many applications of these tools involve geometries that have a curved nature to them [2]. For example, the magnetic coronal-loops in the Sun's atmosphere share a spherical boundary (the photosphere). We reformulate these ideas in a spherical geometry, and then explore the oddities of this curved space to show that our new concept is consistent with its older, flat counterpart.

The second part of this project concerns a series of datasets from plasma experiments at Basic Plasma Science Facility, UCLA, Los Angeles. These experiments involve the creation of flux ropes inside a large (18m) plasma machine. A strong background magnetic field is applied which ensures that field lines travel from one end of the cylindrical device to the other. Due to mutual $\boldsymbol{J} \times \boldsymbol{B}$ forces, the flux ropes twist and tangle about each other.

We study three separate datasets: the first one involving two flux ropes; the second, three flux ropes; the final two flux ropes. The last experiment is perhaps the most exciting as the plasma velocity has been recorded. This extra data allows us to employ two different non-equivalent concepts of magnetic helicity. First, we use the surface flux formulation that makes various *ideal* assumptions, discarding several terms in Ohm's law. This is compared to helicity calculated by use of winding numbers – a construction without these *ideal* assumptions. By examining the difference of these two results, it is shown that we may arrive at a measure of the resistivity present in the system.

The plasma investigations described above rely on being able to seed magnetic field lines across the length of the machine. This is not a simple process. The dataset itself is spatially non-uniform which makes numerical integration to obtain field lines difficult. Even before integration is considered, a method to interpolate on our data grid of magnetic flux density is needed. This requires further careful considerations. Any interpolator must ensure that the data remains divergence-free; this requirement imposes conditions on the continuity of the derivatives. We have written a code to perform tricubic spline interpolation, and demonstrate that by using a particular method for fixing the coefficients, this level of continuity can be achieved.

Contents

List of Figures

1 Introduction

1.1 Units and abbreviations

For the numerical results in this project we choose to use the standard international (SI) system of units. In many cases we could define our own unit so as to incorporate various physical constants, but this approach might obscure matters when sharing our results with our plasma physicist partners at UCLA.

We use the following abbreviations:

Abbreviation	Meaning
MHD	Magnetohydrodynamics
UCLA	University of California, Los Angeles
QSL	Quasi-separatrix
LaPD	The large plasma device
HDF5	Hierarchical Data Format version 5
GSL	GNU Scientific Library

1.2 Introduction

This project is about the entanglement of curves. Perhaps the most basic question one could ask about this subject is: given two closed curves that are somehow entwined, what is the extent of their entanglement? This question has been much studied and is not, in general, simple. A first approach might consider counting the signed number of crossings of the two curves. This approach has its problems: consider for example, the knot in figure 1a. It is not immediately apparent that this knot is equivalent to the un-knot. This problem, the 'un-knotting problem', is so difficult that a polynomial time algorithm has not yet been found [8]. The issue here is that crossing number does not tell us enough about the topology of the curves, but rather their geometry.

In 1833 the German mathematician Carl Friedrich Gauss mentioned the idea of linking number as a brief note in his diary [10]. Gauss was considering the path of asteroids across the sky when he realised that an asteroid's path would be restricted to a range of latitudes if its orbit did not link with Earth's [3]. It is not clear how Gauss formalised this idea into the Gauss linking integral, but Ricca et al. [10] suggests that he might have used his knowledge of terrestrial magnetism for this purpose.

(a) (b)

Figure 1: (a) The un-knot. Determining whether a knot is equivalent to the unknot is a NP-problem. (b) The Borromean rings. Although none of the links are free, the pairwise linking numbers are zero (for details see [9]).

As Gauss noted [11], linking number is–most importantly–invariant to smooth deformation of the curves as long as they are not permitted to pass through each other: it tells you about the topology of a curve rather than its geometry. However, this measure isn't without its problems. Consider the set of links in 1b. The linking number of any individual link is zero, but all three links are bound in place. Note that it is possible to define higher order linking numbers which are non-zero in this situation [12][13]. Putting aside this issue, linking number is an important and widely used tool in many areas of study including molecular biology[14][15], fluid dynamics[16] and super-fluid dynamics[17].

1.3 Linking number

Consider two closed curves entangled in some arbitrary way. If we give both curves a direction, then from one particular viewing angle it is possible to label each crossing with a sign according to figure 2.

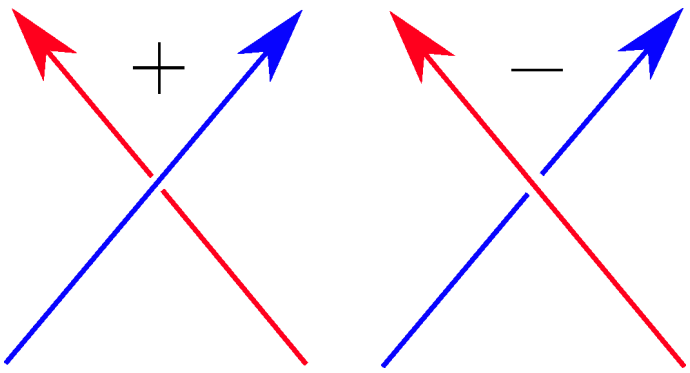

Figure 2: Each crossing is given a sign.

Definition: **Linking number**.

Let $\boldsymbol{x}(s)$ and $\boldsymbol{y}(s')$ be two closed curves parameterised by arc-length. Define the unit tangent vectors as $\hat{\boldsymbol{T}}_x = d\boldsymbol{x}/ds$ and $\hat{\boldsymbol{T}}_y = d\boldsymbol{y}/ds'$. The linking of the two curves is:

$$L_k = \frac{1}{4\pi} \oint_{\boldsymbol{x}} \oint_{\boldsymbol{y}} \hat{\boldsymbol{T}}\boldsymbol{x}(s) \times \hat{\boldsymbol{T}}\boldsymbol{y}(s') \cdot \frac{\boldsymbol{x}(s) - \boldsymbol{y}(s')}{|\boldsymbol{x}(s) - \boldsymbol{y}(s')|^3} \, ds \, ds'. \tag{1}$$

The linking number of two curves is invariant to the motions of the curves as long they are not allowed to pass through each other. All types of crossings will come in pairs which will ensure that the linking number remains an integer.

Theorem: The linking number of two closed curves equals half the signed number of crossings seen in any projection i.e.:

$$L_k = \frac{1}{2}(C_+ - C_-), \tag{2}$$

where C_+ & C_- equal the number of positive and negative crossings respectively. For a proof, or for six alternative (albeit more abstract) equivalent concepts of linking number, see [18].

1.4 Linking number derivation

Let $\gamma_1(s), \gamma_2(t) : \boldsymbol{S} \to \mathbb{R}^3$ be two closed curves. The Gauss map $\Gamma : \boldsymbol{T}^2 \to \boldsymbol{S}^2$ is defined as:

$$\Gamma(s,t) = \frac{\gamma_1(s) - \gamma_2(t)}{\|\gamma_1(s) - \gamma_2(t)\|}. \tag{3}$$

This mapping associates each pair of points on the curves with a point, say $\boldsymbol{p}$, on the unit sphere. Let the domain of Γ be M and the image of Γ be N as in figure 3. This mapping will not in general be a bijection.

Consider the projection of the two curves onto the plane perpendicular to a sphere at $\boldsymbol{p}$. In this projection a crossing occurs when $(s,t) \to \boldsymbol{p}$, which may happen any number of times depending on the curves. The signed average number of crossings will be the number of times the map covers the sphere. This will be the number of times the image of Γ covers the sphere: the number of times M wraps around N. This is the degree of map. We can find this by calculating the area of the image of Γ and dividing it by the area of a unit sphere. It will be necessary to define the following:

Definition: **Derivative map**.

Let $\boldsymbol{E}$ denote Euclidean space and let $F : \boldsymbol{E}^m \to \boldsymbol{E}^n$ be a mapping. Define $\boldsymbol{v}$

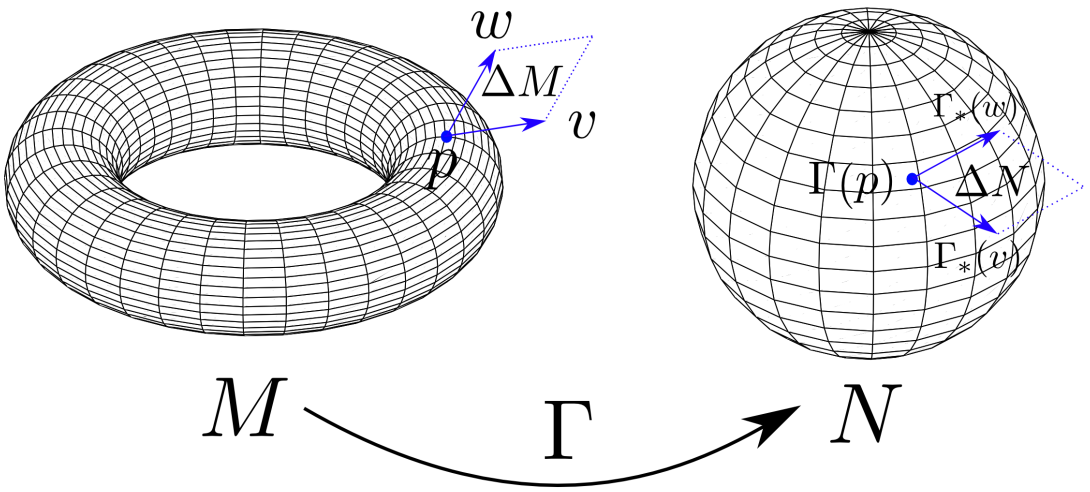

Figure 3: The Gauss map. Here the torus is mapped onto a sphere.

to be a tangent vector to $\boldsymbol{E}^m$ at $\boldsymbol{p}$. The derivative map $F_*(\boldsymbol{v})$ is the initial velocity of the curve $t \to F(\boldsymbol{p}+t\boldsymbol{v})$ in $\boldsymbol{E}^n$. F_* is then the function that maps tangent vectors of $\boldsymbol{E}^m$ to tangent vectors of $\boldsymbol{E}^n$. Note that by definition, a tangent vector $\boldsymbol{v}$ at $\boldsymbol{p}$ will be mapped to a tangent vector $F_*(\boldsymbol{v})$ at $F(\boldsymbol{p})$.

Definition: **Pullback map**.

Let $F : \boldsymbol{E}^m \to \boldsymbol{E}^n$ be a mapping. The pullback map F^* is the function that maps forms from $\boldsymbol{E}^n$ back on to $\boldsymbol{E}^m$.

Definition: **Jacobian**.

As M and N are orientable surfaces we can define the Jacobian function, J on M as:

$$\Gamma^*(dN) = J\, dM. \tag{4}$$

Let $\boldsymbol{v}$,$\boldsymbol{w}$ be any two tangent vectors at a point $\boldsymbol{p}$ on M. If these vectors are small, they can be used to approximate an area ΔM, and the corresponding vectors, $\Gamma_*\boldsymbol{v}$,$\Gamma_*\boldsymbol{w}$, approximate a small area at $\Gamma(\boldsymbol{p})$ on N (see figure 3).

Definition: **Area form**.

Let $\boldsymbol{U}$ be the unit normal vector at $\boldsymbol{p}$ and $\bar{\boldsymbol{U}}$ be the unit normal vector at $\Gamma(\boldsymbol{p})$. The area 2-forms are:

$$dM(\boldsymbol{v},\boldsymbol{w}) = \quad \boldsymbol{U}(\boldsymbol{p}) \cdot \boldsymbol{v} \times \boldsymbol{w} \quad = \pm\|\boldsymbol{v} \times \boldsymbol{w}\|, \tag{5}$$

$$dN(\Gamma_*\boldsymbol{v},\Gamma_*\boldsymbol{w}) = \bar{\boldsymbol{U}}(\Gamma(\boldsymbol{p})) \cdot \Gamma_*\boldsymbol{v} \times \Gamma_*\boldsymbol{w} = \pm\|\Gamma_*\boldsymbol{v} \times \Gamma_*\boldsymbol{w}\|. \tag{6}$$

From equations 4 and 6 we have:

$$\begin{aligned} J(\boldsymbol{p})\, dM(\boldsymbol{v},\boldsymbol{w}) &= (\Gamma^* dN)(\boldsymbol{v},\boldsymbol{w}), && (7)\\ &= dN(\Gamma_*\boldsymbol{v},\Gamma_*\boldsymbol{w}), && (8)\\ &= \bar{\boldsymbol{U}}(\Gamma(\boldsymbol{p}))\cdot\Gamma_*\boldsymbol{v}\times\Gamma_*\boldsymbol{w}, && (9)\\ &= \|\Gamma_*\boldsymbol{v}\times\Gamma_*\boldsymbol{w}\|, && (10)\\ &= \Gamma(\Delta M). && (11) \end{aligned}$$

From equation 11 we can deduce that $|J(\boldsymbol{p})|$ measures how much Γ is expanding area. Recall that we wished to know how many times M covers N. Therefore, what we need to do is find the total area of the image and divide by the area of the unit sphere (4π). The area of $\Gamma(M)$ is:

$$\begin{aligned} \iint_M J\, dM &= \iint_M \Gamma^* dN, && (12)\\ &= \iint_N dN(\Gamma_*\boldsymbol{v},\Gamma_*\boldsymbol{w}), && (13)\\ &= \iint_N \bar{\boldsymbol{U}}(\boldsymbol{p})\cdot\Gamma_*\boldsymbol{v}\times\Gamma_*\boldsymbol{w}, && (14)\\ &= \oint_{\gamma_1}\oint_{\gamma_2} \hat{\boldsymbol{T}}_{\gamma_1}(s)\times\hat{\boldsymbol{T}}_{\gamma_2}(t)\cdot\frac{\gamma_1(s)-\gamma_2(t)}{|\gamma_1(s)-\gamma_2(t)|^3}dsdt. && (15) \end{aligned}$$

Finally, dividing by 4π gives the well known formula for the linking of the two curves

$$L_k = \frac{1}{4\pi}\oint_{\gamma_1}\oint_{\gamma_2} \hat{\boldsymbol{T}}_{\gamma_1}(s)\times\hat{\boldsymbol{T}}_{\gamma_2}(t)\cdot\frac{\gamma_1(s)-\gamma_2(t)}{|\gamma_1(s)-\gamma_2(t)|^3}dsdt. \quad (16)$$

1.5 Twist

Twist measures the rotation rate of a secondary curve about an axis curve. For the axis curve define: $\hat{\boldsymbol{T}}$ as the unit tangent vector and $\hat{\boldsymbol{V}}$ as the unit normal vector oriented towards the secondary curve. Define twist to be:

$$T_w = \frac{1}{2\pi}\oint_{\boldsymbol{x}} \hat{\boldsymbol{T}}(s)\cdot\hat{\boldsymbol{V}}(s)\times\frac{d\hat{\boldsymbol{V}}(s)}{ds}ds. \quad (17)$$

Twist measures the rotation rate of a secondary curve about an axis curve: it measures the winding of the secondary curve about $\hat{\boldsymbol{T}}$. It will be of help later to define the concept of a ribbon. Consider again an axis curve together with a nearby secondary curve: a ribbon is the surface that spans between the two curves; it is the surface in which the series of $\hat{\boldsymbol{V}}(s)$ vectors live.

1.6 Writhe

Writhe equals the number of crossings of a curve with itself averaged over all projection angles. Writhe is a measure of the extent to which the axial curve would lay

flat in a 2-D plane. Writhe is not a topological invariant: although it is unaffected by the Reidermeister moves II and III, move I changes the writhe by one (see figure 4). This means that writhe cannot be a topological measure as it depends on the geometry of the curves. Writhe is equal to:

$$W_r = \frac{1}{4\pi} \oint_{\boldsymbol{x}} \oint_{\boldsymbol{x}} \hat{\boldsymbol{T}}(s) \times \hat{\boldsymbol{T}}(s') \cdot \frac{\boldsymbol{x}(s) - \boldsymbol{x}(s')}{|\boldsymbol{x}(s) - \boldsymbol{x}(s')|^3} ds ds'. \tag{18}$$

In contrast to twist, writhe is not a local quantity and is independent of the second curve. Writhe measures how much a ribbon is helical; how much it is corkscrewed up on a larger scale than just twist.

1.7 Călugăreanu's theorem

In 1959 Călugăreanu formalised these ideas and proved that linking number could be decomposed into twist and writhe. These two quantities vary as the curves are deformed, but their sum is constant provided the curves aren't permitted to pass through each other. Any deformation of the curves can be arrived at by a series of Reidermeister moves (described in figure 4). We say two knots are equivalent if they can be related to each other by a series of Reidermeister moves [19].

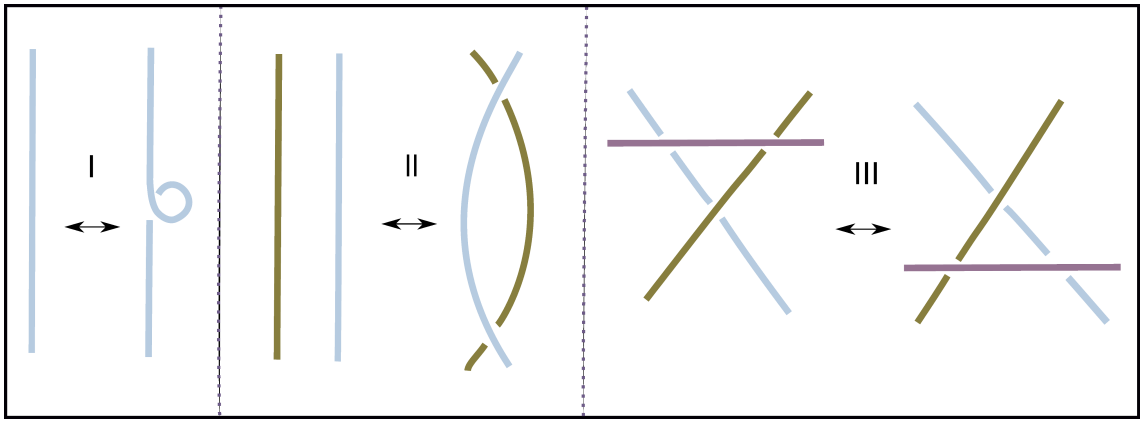

Figure 4: The three Reidermeister moves. Linking number is invariant to any of these deformations. Note that any deformation that doesn't allow the curves to pass through one another can be build up from these three moves.

Călugăreanu's theorem:

$$L_k = T_w + W_r. \tag{19}$$

This theorem says that linking number is the equal to the sum of twist and writhe. For Călugăreanu's proof see [20], or for an alternative modern proof in English see [21]. To get an intuitive sense of this theorem, consider an untwisted strip of elastic band. If one starts to twist the end points, twist will be injected into the band. At

some point, an instability will be reached, and the band will buckle; this buckling process converts twist into writhe.

1.8 The Frenet frame

The Frenet frame is a set of basis vectors that are defined in terms of vectors that originate on a specific curve. Here we will describe how to formulate the Frenet frame in $\mathbb{R}^3$; however, it is possible to extend this to $\mathbb{R}^n$ [22]. Let κ be the curvature of a curve $\boldsymbol{x}$ at a point s, that is

$$\kappa = \left| \frac{d\hat{\boldsymbol{T}}(s)}{ds} \right|. \tag{20}$$

So κ is a measure of how fast the tangent vector changes direction, or equivalently, the fraction of acceleration due to a change of direction for a particle following the curve – tight fast turns will result in large values of κ. Define the normal and the binormal vectors to be

$$\hat{\boldsymbol{N}} = \frac{1}{\kappa}\frac{d\hat{\boldsymbol{T}}}{ds}, \qquad\qquad \hat{\boldsymbol{B}} = \hat{\boldsymbol{T}} \times \hat{\boldsymbol{N}}. \tag{21}$$

These quantities satisfy the Frenet-Serret equations

$$\frac{d\hat{\boldsymbol{T}}(s)}{ds} = \kappa\hat{\boldsymbol{N}}, \tag{22}$$

$$\frac{d\hat{\boldsymbol{N}}(s)}{ds} = \tau\hat{\boldsymbol{B}} - \kappa\hat{\boldsymbol{T}}, \tag{23}$$

$$\frac{d\hat{\boldsymbol{B}}(s)}{ds} = -\tau\hat{\boldsymbol{N}}. \tag{24}$$

Where torsion, $\tau = -\hat{\boldsymbol{N}} \cdot \frac{d\hat{\boldsymbol{B}}}{ds}$, measures the speed of rotation of the binormal vector $\hat{\boldsymbol{B}}$. Note that if the curvature is zero (e.g. a straight line), then $\hat{\boldsymbol{N}}$ and $\hat{\boldsymbol{B}}$ are not well defined (for a study of what happens at such points of inflexion, see [23]. These three vectors define an orthonormal basis $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{N}}, \hat{\boldsymbol{B}}\}$.

1.9 Winding number

Consider two curves spiralling upwards in the z-direction parametrised by z. Let the relative position vector point from one curve to the other, and let θ be the angle of this vector against the positive x-axis. The winding of the two curves is

$$\Delta\theta \ \texttt{mod}\ 2\pi = \theta_{z_{top}} - \theta_{z_{bottom}}. \tag{25}$$

This is the most efficient way to calculate winding numbers. However, this equation is only useful if the curves rotate less than one complete turn – if this occurs a more

sophisticated approach is needed. We consider a small change in θ, say $d\theta$, and then integrate upwards

$$w = \frac{1}{2\pi}\int_C \frac{d\theta}{dz}dz. \tag{26}$$

In terms of the relative position vector itself this may be written

$$w_{ab} = \frac{1}{2\pi}\int \frac{1}{r_{ab}^2}\hat{z}\cdot \boldsymbol{r}_{ab}\times\frac{d\boldsymbol{r}_{ab}}{dz}dz. \tag{27}$$

1.10 Electromagnetism

Electromagnetism is the study of the interaction between charges, and electric and magnetic fields. Maxwell's equations relate magnetic and electric fields. In the usual way we denote the vector magnetic flux density as $\boldsymbol{B}$, the electric field as $\boldsymbol{E}$, and the charge density as ρ. Maxwell's equations are

$$\nabla\cdot\boldsymbol{E} = \frac{\rho}{\epsilon_0}, \qquad \text{(Gauss' law)} \tag{28}$$

$$\nabla\cdot\boldsymbol{B} = 0, \qquad \text{(No magnetic monopoles)} \tag{29}$$

$$\nabla\times\boldsymbol{E} = -\frac{\partial\boldsymbol{B}}{\partial t}, \qquad \text{(Faraday's law of induction)} \tag{30}$$

$$\nabla\times\boldsymbol{B} = \mu_0\left(\boldsymbol{J}+\epsilon_0\frac{\partial\boldsymbol{E}}{\partial t}\right). \qquad \text{(Ampère's circuital law)} \tag{31}$$

Qualitatively, these equations state: electric fields are produced by either charges or time varying magnetic fields; magnetic fields are generated either by currents or changing electric fields. The divergence-free condition is required so that the magnetic field is physical. The constant that appears in Gauss' law is the electric constant ϵ_0. This is a measure of the permittivity of free space, that is, a measure of how much space resists the formation of an electric field – it has the value

$$\epsilon_0 = \frac{1}{\mu_0 c^2} \approx 8.854\times10^{-12}Fm^{-1}. \tag{32}$$

The term on the right hand side contains a constant μ_0. This is the magnetic counterpart to ϵ_0. The vacuum permeability of free space is:

$$\mu_0 = 4\pi\times10^{-7}Hm^{-1}. \tag{33}$$

For a thorough introduction to electrodynamics see the Feynman Lectures [24].

1.11 Magnetohydrodynamics

Magnetohydrodynamics (MHD) is the study of the dynamics of electrically conducting fluids and the magnetic fields that interact with them. The name of the field itself was coined by Hannes Alfvén in 1942 [25], however, the first MHD experiment was performed long before this in 1832 by the British physicist Michael Faraday.

Faraday demonstrated that electricity could be generated by moving a conductor through a stationary magnetic field [26].

The generalised form Ohm's law [27] is

$$\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B} = \eta \boldsymbol{J} + \frac{1}{ne}\boldsymbol{J} \times \boldsymbol{B} - \frac{1}{ne}\nabla \cdot \boldsymbol{P}_e + \frac{m_e}{ne^2}\frac{\partial \boldsymbol{J}}{\partial t}. \tag{34}$$

The first term on the right is the Spitzer resistivity, the second the Hall term, the third the electron pressure term, and the last the electron inertial term. n is the free electron density per volume, e the amount of charge that one electron carries, m_e the electron rest mass and $\boldsymbol{v}$ the velocity of the plasma. In figure 5 Cothran et al. compare the magnitude of the terms in this equation.

	One Experiment			Average Data Set		
[V/m]	t = 37 μs	t = 64 μs		t = 37 μs	t = 64 μs	
$\eta\lvert\mathbf{J}\rvert$	44	15	±7	33	19	±2
$\frac{1}{ne}\lvert\mathbf{J} \times \mathbf{B}\rvert$	2800	700	±100	2000	330	±20
$\frac{m_e}{ne^2}\lvert\partial\mathbf{J}/\partial t\rvert$	0.2	0.04	±0.03	0.03	0.01	±0.01

Figure 5: Figure from a paper by Cothran et al. [27]. The authors recorded data from 3-D plasma experiments and then compared the magnitude of the terms in Generalised Ohm's law (equation 34).

The equations of MHD follow from Maxwell's equations after making a few assumptions. A key assumption of ideal MHD is that equation 34 can be taken as

$$\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B} = 0. \tag{35}$$

Here a number of terms have been omitted. One of the assumptions of ideal MHD is that the plasma has infinite electrical conductance so the diffusion is zero; this is often a good approximation if the magnetic diffusion time scale is longer than the time scale of interest. Gombosi et al. [28] note that even under this assumption, resistivity can still manifest as a numerical phenomenon (also see [29]). As the fluid is modelled as having zero resistivity and so is a perfect conductor the magnetic field will remain frozen in place (in the reference frame of the plasma). For a discussion of the appropriateness of this simplification see [30]. The equations of MHD will not be used in this project and so we will not state them in full here – for a good introduction to MHD see [31]. However, in the final plasma experiment we study we will test the extent to which the magnetic field remains frozen in the plasma.

It can be shown that Maxwell's equations together with the ideal form of Ohm's law lead to the induction equation

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{v} \times \boldsymbol{B}) + \eta \nabla^2 \boldsymbol{B}. \tag{36}$$

Noting that the vector potential $\boldsymbol{A}$ satisfies $\nabla \times \boldsymbol{A} = \boldsymbol{B}$, we can uncurling both sides of the above equation to obtain

$$\frac{\partial \boldsymbol{A}}{\partial t} = \boldsymbol{v} \times \boldsymbol{B} + \nabla \phi, \tag{37}$$

where the ϕ term results from our choice of gauge.

1.12 Magnetic energy

The electromotive force $\mathscr{E}$ around a stationary closed path is defined as

$$\mathscr{E} = \oint \boldsymbol{E} \cdot d\boldsymbol{l} = -\frac{d\Phi}{dt}, \tag{38}$$

where Φ is the magnetic flux. The amount of flux in a circuit is proportional to the current, I, so we may write

$$\Phi = LI, \tag{39}$$

for some constant L. Note that the magnetic flux through a surface $\boldsymbol{S}$ is

$$\Phi = \iint_S \boldsymbol{B} \cdot d\boldsymbol{S}. \tag{40}$$

Writing $\boldsymbol{B}$ in terms of its vector potential $\boldsymbol{A}$ this becomes

$$\Phi = \oint \boldsymbol{A} \cdot d\boldsymbol{l}, \tag{41}$$

where the integral is taken over the closed curve that bounds the surface.

Consider an electrical circuit in which there is no current flowing. If we start to allow current to flow in this circuit – so here $\frac{\partial \boldsymbol{E}}{\partial t} \neq 0$ – it is clear from Ampère's law (equation 31) that a magnetic field will form. The total work done in the circuit is

$$\frac{dW}{dt} = -\mathscr{E} I = I \frac{d\Phi}{dt} = IL \frac{dI}{dt}. \tag{42}$$

The work done in a circuit as the current starts to flow and reaches a value of I is given by the time integral of this equation [32] and an application of equation 41

$$W = \frac{1}{2} LI^2, \tag{43}$$

$$= \frac{1}{2} I \oint \boldsymbol{A} \cdot d\boldsymbol{l}. \tag{44}$$

Replacing the line current by a current distribution and applying Stokes' theorem gives

$$W = \frac{1}{2} \int_V \boldsymbol{A} \cdot \boldsymbol{J} dV. \tag{45}$$

Using Ampère's law this is

$$W = \frac{1}{2\mu_0} \int \boldsymbol{A} \cdot (\nabla \times \boldsymbol{B}) dV. \tag{46}$$

Vector identities give

$$\boldsymbol{A} \cdot (\nabla \times \boldsymbol{B}) = \boldsymbol{B} \cdot \boldsymbol{B} - \nabla \cdot (\boldsymbol{A} \times \boldsymbol{B}). \tag{47}$$

So the energy integral becomes

$$W = \frac{1}{2\mu_0} \left(\int B^2 dV - \int_S \boldsymbol{A} \times \boldsymbol{B} \cdot d\boldsymbol{S} \right). \tag{48}$$

If we integrate over all of space the second term goes to zero (this is due to the fact that the cross product term decreases more quickly than bounding area increases as you move away from a current loop). So finally,

$$W = \frac{1}{2\mu_0} \int B^2 dV. \tag{49}$$

This energy, W, measures the amount of energy needed to create a magnetic field.

1.13 Helicity

Whether the term helicity is used in fluid dynamics [33] or plasma physics [34], it is meant as a quantity which measures the extent of twistedness. Magnetic helicity is a measure of how much a magnetic field wraps around itself. It measures something about a magnetic field's topology rather than its geometry, allowing us to compare two different situations even if properties, such as the field strength, are very different. One of the reasons it is so useful is that it is a conserved quantity as long as the field lines remain frozen in the plasma [35] [36]. It can be shown that, under certain conditions, even if the magnetic field undergoes reconnection, helicity will be approximately conserved [37]. The Magnetic helicity of two flux tubes $\{\boldsymbol{x}, \boldsymbol{y}\}$, is defined as [38]

$$H = -\frac{1}{4\pi} \iint \boldsymbol{B}(\boldsymbol{x}) \cdot \frac{\boldsymbol{r}}{r^3} \times \boldsymbol{B}(\boldsymbol{y}) \, d^3y d^3x, \tag{50}$$

where $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{x}$. By writing $\boldsymbol{B}$ in terms of vector potential

$$\mathbf{B} = \nabla \times \mathbf{A}, \tag{51}$$

and using the Coulomb gauge ($\nabla \cdot \boldsymbol{A} = 0$), this becomes

$$H = \int_V \mathbf{A} \cdot \boldsymbol{B} dV. \tag{52}$$

This vector potential may be found using

$$\boldsymbol{A}(\boldsymbol{x}) = -\frac{1}{4\pi} \int_V \frac{\boldsymbol{r}}{r^3} \times \boldsymbol{B}(\boldsymbol{y}) \, d^3y. \tag{53}$$

This expression for helicity will be gauge invariant as long as the magnetic field has no component in the normal direction to the boundaries of the volume in question ($\mathbf{B}\cdot\boldsymbol{n}|_s = 0$). To see this, consider a gauge transformation of the form $\boldsymbol{A} \rightarrow \boldsymbol{A}+\nabla\psi$, which means that H changes by

$$\begin{aligned}
\delta H &= \int \nabla\psi \cdot \boldsymbol{B} dV, && (54)\\
&= \int \nabla \cdot (\psi \boldsymbol{B}) dV, && (55)\\
&= \int_S \psi \boldsymbol{B} \cdot d\boldsymbol{S}, && (56)\\
&= 0. && (57)
\end{aligned}$$

To see that it is conserved in ideal MHD we take the time derivative of equation 52.

$$\begin{aligned}
\frac{dH}{dt} &= \int \frac{\partial \boldsymbol{A}}{\partial t} \cdot \boldsymbol{B} dV + \int \boldsymbol{A} \cdot \frac{\partial \boldsymbol{B}}{\partial t} dV, && (58)\\
&= \int \frac{\partial \boldsymbol{A}}{\partial t} \cdot \nabla \times \boldsymbol{A} dV + \int \boldsymbol{A} \cdot \left(\nabla \times \frac{\partial \boldsymbol{A}}{\partial t} \right) dV && (59)
\end{aligned}$$

The electric field is given by

$$\boldsymbol{E} = -\nabla V - \frac{\partial \boldsymbol{A}}{\partial t}, \qquad (60)$$

where V is the electric potential. This can be used to eliminate the time derivative of $\boldsymbol{A}$ so that dH/dt becomes

$$\begin{aligned}
\frac{dH}{dt} &= -\int (\nabla V + \boldsymbol{E}) \cdot \boldsymbol{B} dV - \int \boldsymbol{A} \cdot (\nabla \times \boldsymbol{E}) dV, && (61)\\
&= -\int \nabla \cdot (V \boldsymbol{B}) dV - \int \boldsymbol{E} \cdot \boldsymbol{B} dV - \int \boldsymbol{A} \cdot (\nabla \times \boldsymbol{E}) dV. && (62)
\end{aligned}$$

The first term on the right here is arrived at by noting that $\boldsymbol{B}$ is divergence-free. Using the vector identity

$$\boldsymbol{A} \cdot (\nabla \times \boldsymbol{E}) = \boldsymbol{E} \cdot (\nabla \times \boldsymbol{A}) - \nabla \cdot (\boldsymbol{A} \times \boldsymbol{E}), \qquad (63)$$

and Stoke's theorem we have

$$\frac{dH}{dt} = -\int_S V \boldsymbol{B} \cdot d\boldsymbol{S} - \int \boldsymbol{E} \cdot \boldsymbol{B} dV - \int \boldsymbol{E} \cdot (\nabla \times \boldsymbol{A}) dV + \int \nabla \cdot (\boldsymbol{A} \times \boldsymbol{E}) dV. \qquad (64)$$

Our earlier requirement that $\boldsymbol{B} \cdot \hat{\boldsymbol{n}} = 0$ means that the first term is zero. The second and the third term are the same, which gives:

$$\begin{aligned}
\frac{dH}{dt} &= -2 \int \boldsymbol{E} \cdot \boldsymbol{B} dV + \int \nabla \cdot (\boldsymbol{A} \times \boldsymbol{E}) dV, && (65)\\
&= -2 \int \boldsymbol{E} \cdot \boldsymbol{B} dV + \int_S \boldsymbol{A} \times \boldsymbol{E} \cdot d\boldsymbol{S}, && (66)\\
&= -2 \int \boldsymbol{E} \cdot \boldsymbol{B} dV + \int_S (\boldsymbol{A} \cdot \boldsymbol{B}) \boldsymbol{v} - (\boldsymbol{A} \cdot \boldsymbol{v}) \boldsymbol{B} \cdot d\boldsymbol{S}. && (67)
\end{aligned}$$

From Ohm's law the first term cancels in the resistivity-free limit $\eta = 0$. Let us examine the other two terms. If we have outward plasma flow then the normal component of the velocity will be non-zero – this contribution is the first of these two terms. The last term concerns footpoints of magnetic flux tubes (see figure 6) moving on the boundary. For example, one method of building up helicity in a solar coronal loop is boundary point motions due to differential rotational [39].

It is important to note that we began the above calculation by differentiating equation 52: this equation is only valid if we integrate over all space, or otherwise, in a magnetically closed volume. In 1984, Berger & Field [37] defined a gauge invariant helicity for open volumes in terms of a relative minimum energy state background field (see figure 6). Let $\boldsymbol{P}$ be this background field and $\boldsymbol{A}_p$ its corresponding potential. Then we define

$$H = \int_V (\boldsymbol{A} + \boldsymbol{A}_p) \cdot (\boldsymbol{B} - \boldsymbol{P}) d^3x. \tag{68}$$

where $\boldsymbol{A}_p$ is the unique vector potential that satisfies

$$\hat{\boldsymbol{n}} \cdot \nabla \times \boldsymbol{A}_p = B_n, \tag{69}$$

$$\nabla \cdot \boldsymbol{A}_p = 0, \tag{70}$$

$$\boldsymbol{A}_p \cdot \hat{\boldsymbol{n}} = 0. \tag{71}$$

Here $\hat{\boldsymbol{n}}$ is the vector normal to the boundary of the volume. Note that $\boldsymbol{B}$ is the total magnetic field within the volume and $\boldsymbol{A}$ its potential. We will refer to equation 68

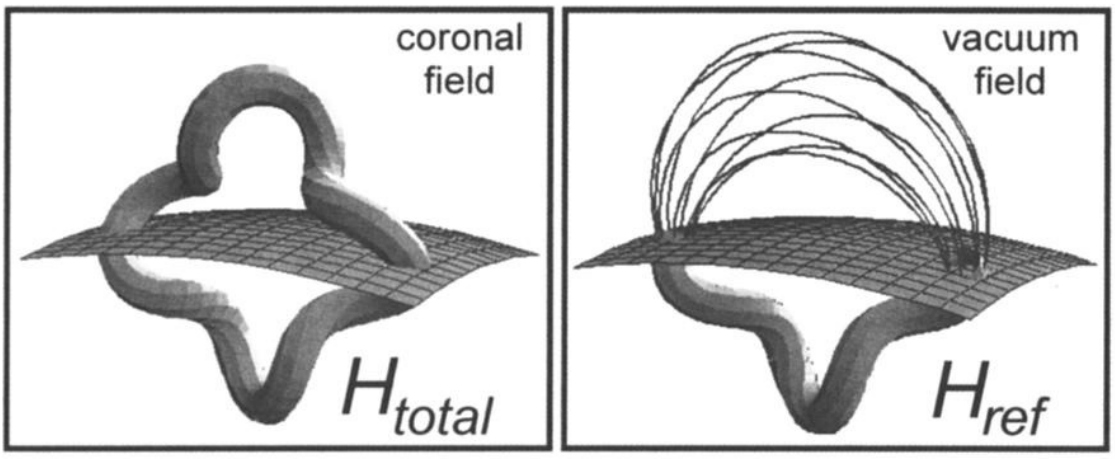

Figure 6: Relative helicity is defined in terms of a background magnetic field and corresponding potential. By using the vacuum field (with no helicity) as the relative field we can calculate the helicity of the open segment in a topological and gauge invariant way. Reproduced from [3].

as the relative helicity equation. As the more involved calculation to take the time

derivative of this equation is given in detail by Berger & Field, we just state the result

$$\frac{dH}{dt} = -2\int \boldsymbol{E}\cdot\boldsymbol{B}dV + 2\int_S (\boldsymbol{A}_p\cdot\boldsymbol{v})\boldsymbol{B} - (\boldsymbol{A}_p\cdot\boldsymbol{B})\boldsymbol{v}\cdot d\boldsymbol{S}. \tag{72}$$

Note that here $\boldsymbol{B}$ is the total magnetic field density including the background field $\boldsymbol{P}$, and also that $\boldsymbol{A}_p$ is only the vector potential of the background field ($\boldsymbol{P} = \nabla\times\boldsymbol{A}_p$). This equation is of particular use when working with open flux tubes such as coronal loops emanating from the solar photosphere. Splitting this up, we define

$$\frac{dH}{dt}_{surfaceFlux} = 2\int_S (\boldsymbol{A}_p\cdot\boldsymbol{v})\boldsymbol{B} - (\boldsymbol{A}_p\cdot\boldsymbol{B})\boldsymbol{v}\cdot d\boldsymbol{S}, \tag{73}$$

$$\frac{dH}{dt}_{dissipation} = -2\int \boldsymbol{E}\cdot\boldsymbol{B}dV. \tag{74}$$

Helicity can be expressed in terms of winding numbers [38]. Consider a collection of i linked flux tubes within a volume $\mathbf{V}$, and note that $\mathbf{B} = 0$ outside of each tube. Each tube carries a flux Φ_i which is aligned along the direction of the field. The total flux is then

$$\Phi_i = \int_{S_i} \mathbf{B}\cdot d\mathbf{S}, \tag{75}$$

where S_i is the cross sectional surface area of the i^{th} flux tube. Noting that $\mathbf{B}$ is zero everywhere except inside the flux tubes, the original equation for helicity (equation 50) can be re-written as a sum of over all the flux tubes in a volume

$$H = -\frac{1}{4\pi}\sum_i\sum_j \int_i d^3x_i \int_j d^3x_j \left[\boldsymbol{B}_i\cdot\frac{\boldsymbol{r}}{r^3}\times\boldsymbol{B}_j\right], \tag{76}$$

$$= -\frac{1}{4\pi}\sum_i\sum_j \oint_i ds_i \int_{\boldsymbol{A}_i} d\boldsymbol{A}_i \oint_j ds_j \int_{\boldsymbol{A}_j} d\boldsymbol{A}_j \left[\boldsymbol{B}_i\cdot\frac{\boldsymbol{r}}{r^3}\times\boldsymbol{B}_j\right], \tag{77}$$

$$= -\frac{1}{4\pi}\sum_i\sum_j \int_{\boldsymbol{A}_i}\int_{\boldsymbol{A}_j}\oint_i\oint_j \frac{d\boldsymbol{x}}{ds}\cdot\frac{\boldsymbol{r}}{r^3}\times\frac{d\boldsymbol{y}}{ds} ds_i ds_j |\boldsymbol{B}_i|\, d\boldsymbol{A}_i\, |\boldsymbol{B}_j|\, d\boldsymbol{A}_j. \tag{78}$$

Here $\boldsymbol{A}_i$ denotes the cross-sectional area of the i^{th} flux tube. Then from the definition of linking number (equation 1), and also equation 75 we have

$$H = \sum_i\sum_j L_{ij}\Phi_i\Phi_j. \tag{79}$$

Berger and Prior [1] show that, for curves that always travel upwards, we cab write linking number in terms of winding numbers:

$$L_k = \sum_i\sum_j w_{ij}. \tag{80}$$

Thus we can write helicity as

$$H = \sum_i\sum_j w_{ij}\Phi_i\Phi_j. \tag{81}$$

Magnetic helicity can be related to the magnetic energy as defined in the previous sections [3].

$$\left|\frac{dH}{dt}\right| \leq \sqrt{2\eta W \left|\frac{dW}{dt}\right|}. \tag{82}$$

Where W is the magnetic energy, η the resistivity.

1.14 Solar applications of helicity

The Sun is the 4.6 billion year old [2] star at the centre of our solar system. It is a spherical ball of hot plasma with intense magnetic fields, composed of mostly hydrogen, with smaller amounts of helium and other elements. It has a solar cycle of about 11 years. During this time the level of solar activities, such as the appearance of sunspots or the eruptions of solar flares, vary.

A sunspot is an area on the surface of the sun (the photosphere) which is experiencing intense magnetic activity. Sunspots are seen as a different colour to neighbouring areas due to being cooler as the magnetic field inhibits convection. They appear in pairs, each having a different magnetic polarity. There is a fairly strong relationship between the rate of appearance of sunspots and the Sun's solar cycle. Figure 7 shows the famous "butterfly" diagram. From this we can make a number of observations: sunspots only appear at a range of latitudes; the current stage of the cycle affects the chance of finding a sunspot at a certain latitude; early in the cycle sunspots appear at high latitudes and move towards the equator.

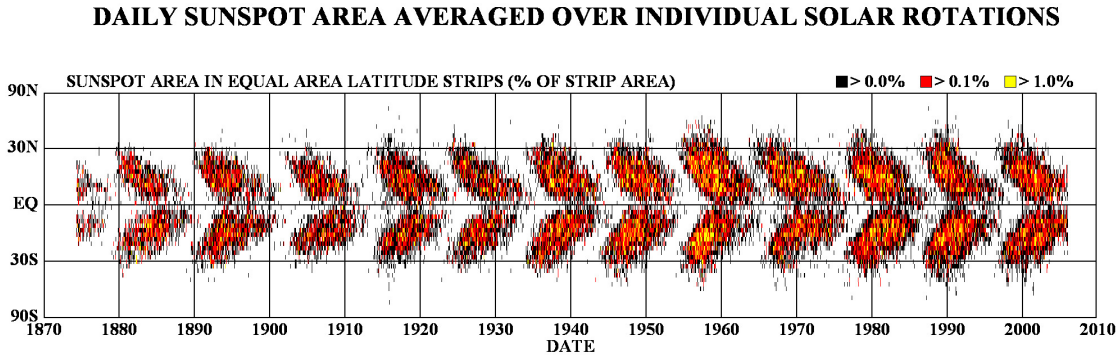

Figure 7: The Sun's 11 year solar cycle. Sunspots appear far away from the equator at a solar minimum and move towards lower latitudes. Reproduced from `http://www.nasa.gov`.

In 1925, the astronomer George Hale published a paper [40] suggesting that, roughly 80% of the time, the sign of helicity in sunspots can be predicted by the sunspot's location. If the sunspot was located in the northern hemisphere, the

sunspot would swirl counter-clockwise, southern hemisphere, clockwise. This relationship is not dependant upon where the Sun is in its solar cycle. These results have later been confirmed, for example [41].

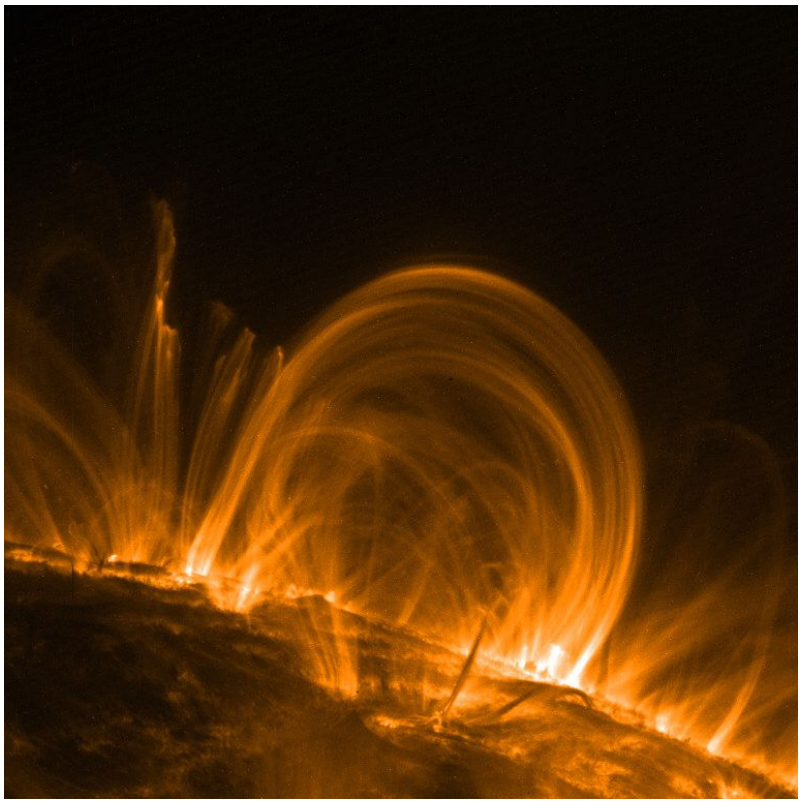

Figure 8: A magnetic coronal loop emanating out of the photosphere. We cannot observe the magnetic field directly – ionised gas aligns itself along lines of magnetic flux. Reproduced from `http://www.nasa.gov`.

Another solar feature of interest is coronal loops. Coronal loops (shown in figure 8) originate at sunspots and emanate upwards, into the solar corona. They are intense magnetic fields that have had helicity injected, through processes such as differential rotation. Using equation 68, helicity in these coronal loops can be estimated (for example see [42]). As the magnetic field below the photosphere cannot be observed, a minimum energy state field is used (see figure 6).

An open question is why the corona ($\sim$1 million K) is so much hotter than the photosphere ($\sim$5000K). One explanation [43] is that the magnetic fields in these coronal loops undergo reconnection, falling to a lower energy state by releasing massive amounts of energy as heat. Such reconnection events have been observed [44][45].

1.15 C++

When writing numerical codes there are several important matters to consider. The first is the choice of language. Fortran has been around for a long time and has many mature software libraries, which is one of the reasons that it is the most frequently used in numerical computation [46]. Although Fortran would be a fine choice for the programs in this project, we choose C++ as a personal preference.

Like Fortran, C++ is a compiled language. C++ is an extension of C and offers many features not found in the original. In fact, if something is possible in a programming language, it is probably possible in C++. C++ is a high-level, multi-paradigm language supporting procedural, functional, object-oriented and generic programming.

The main user difference between Fortran and C++ is that a C++ programmer has greater access to low-level elements, such as memory, which are taken care of in the former case. This can be both a blessing and a curse. For example, consider a situation in which a calculation requires a fixed-sized workspace of memory. If this calculation has to be repeated multiple times, one after another, it may be advantageous not to release the workspace in memory after one calculation and use it for the next – thus avoiding an allocation delay. The trade-off here is that this optimisation, possible in C++, adds an extra avenue for programmer errors.

If speed is of concern, then there is one matter that every programmer should be aware of. Consider a dataset residing in a computer's RAM. Each time an individual value is required the CPU will fetch the value and place it in its cache. However, the processor is designed in such a way that it fetches a fixed amount of memory each time. So, in practice, a request for a single value will mean many neighbouring data values (a typical desktop computer has a cache line size of 64K = 8 double precision floats) are copied at the same time. If possible, programmers should design their programs so that they access data sequentially. For a good overview of numerical methods in C++ see [47].

1.16 HDF5

Hierarchical Data Format version 5 [4] is an award-winning [48] data format that is widely used in scientific applications which require the storage of large quantities of data. Many experiments require the recording of data with at least three dimensions. Two dimensional data can quite easily be stored as a table in a text file. When we consider three or more dimensions more thought has to be taken, and it would be

difficult, without explanation, for another user to guess the structure of a file. HDF5 provides a solution to this and has, in addition, the following advantages:

- Files are self-describing
- Supports very large datasets with no limits on the complexity of data structure.
- Will run on a huge variety of platforms.
- Native support in Matlab and Mathematica.
- High-level APIs in C/C++/Fortan and Java with parallel I/O support.
- Supports compression.

A HDF5 file is composed of two types of objects:

- HDF5 group
- HDF5 dataset

Each HDF5 file contains at least one group. This group can then contain one or more datasets. The process of accessing a dataset involves the use of two hyper-slabs. A hyper-slab is selection of a file or memory area that one wishes to access. Normally users would have a source and destination hyper-slab which need not be of the same size, or even, the same dimension (see figure 9). As the data we have access to is stored in the HDF5 format we have written general C++ container classes for reading/writing to HDF5 files. These are included in the appendix.

1.17 GSL

The GNU Scientific Library (GSL) [49] is a software library designed for numerical computation that has been in use for nearly twenty years. It is written in C but has wrappers for several other programming languages including C++. The library contains routines for many different types of integrators, interpolators and numerical derivatives. We use the library to fit cubic splines so that we may take derivatives of our data.

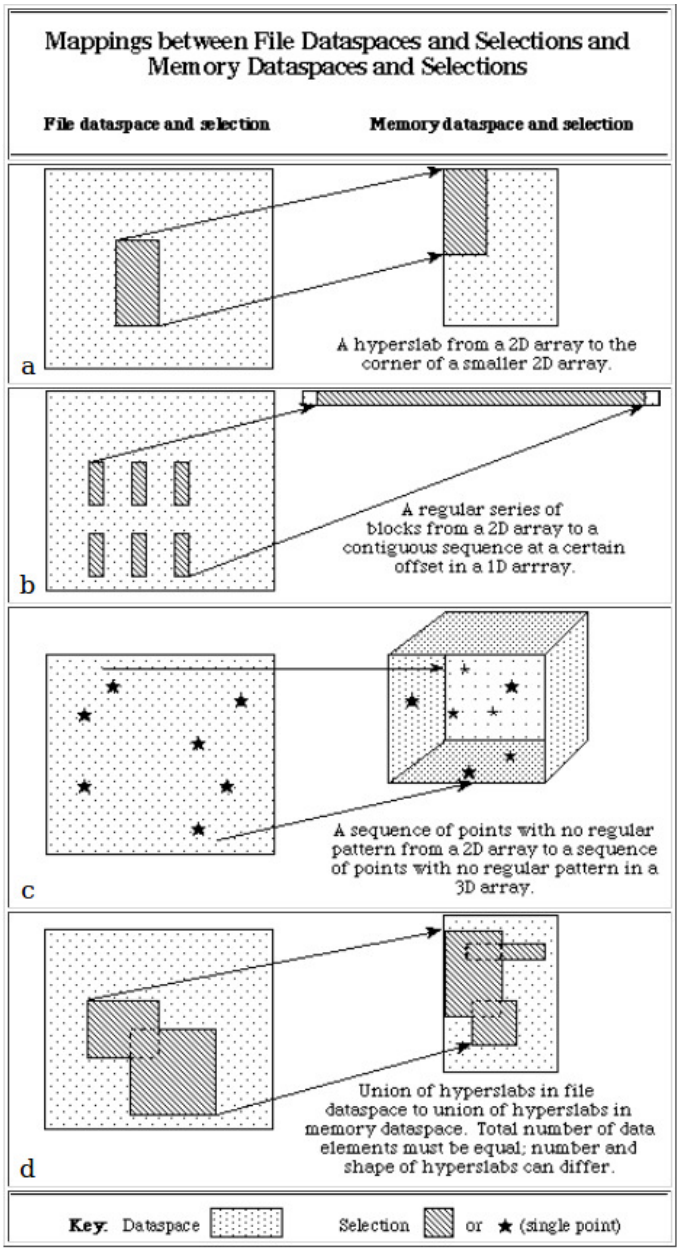

Figure 9: The structure of a HDF5 file. Image from [4].

2 Writhe in a spherical geometry

"'I couldn't afford to learn it." said the Mock Turtle with a sigh. "I only took the regular course."

"What was that?" inquired Alice.

"Reeling and Writhing, of course, to begin with," the Mock Turtle replied; "and then the different branches of Arithmetic Ambition, Distraction, Uglification, and Derision."' Lewis Carroll, Alice's Adventures in Wonderland [50].

2.1 Linking in a spherical geometry

Many applications of the measures detailed here involve geometries that have a curved nature to them [2]. For example, the magnetic coronal-loops in the Sun's atmosphere share a spherical boundary (the photosphere). Also, biologists are interested in calculating the writhe of DNA which has a curved central axis [51]. The purpose of this work is to extend the existing concepts of linking number, twist and writhe, so that they work in a consistent manner in spherical geometry. The work from this chapter has been accepted for publication [52].

Spherical geometry has several intrinsic differences to flat space. Firstly, the usual coordinate system (ρ, θ, ϕ) contains points that cannot be uniquely described; if $\rho = 0$ or $\theta = 0$ or $\theta = \pi$, then one or more of the other components can be changed without altering a point's position. Another difference is that on a sphere, Euclid's fifth postulate breaks down: two parallel lines need not stay a fixed distance apart. A consequence of this is that in general it is not possible to define parallel vectors; one must first define how to move a vector around – the connection – before this is possible.

It is worth noting that not all curved manifolds exhibit the features described in the last paragraph – consider the surface of a cone. Intuitively, it might seem that the surface of a cone is somehow *less curved* than that of a sphere. The difference between a cone and a sphere is that a cone can be unrolled into something flat, whilst a sphere cannot: a sphere has innate curvature.

Another consideration becomes apparent when you attempt to differentiate a vector. The basis vectors in spherical geometry are not constant, and so the derivative of a vector must contain an expression describing how the basis changes – the Christoffel symbols.

2.2 Relative position vector

Working in a spherical coordinate system (ρ, θ, ϕ) (where ϕ is the azimuthal angle), consider two curves that wind around each other. Assume that these curves continually travel upwards radially so that they may be parametrised by ρ. Define the relative position vector $\boldsymbol{r}(\rho)$ to be the vector originating on the axis curve and pointing along the geodesic towards the secondary curve at a particular height in ρ (see figure 10). The magnitude of this vector is given by the great-circle distance between the two curves. Note that $\boldsymbol{r}(\rho)$ is not well defined if the points are antipodal.

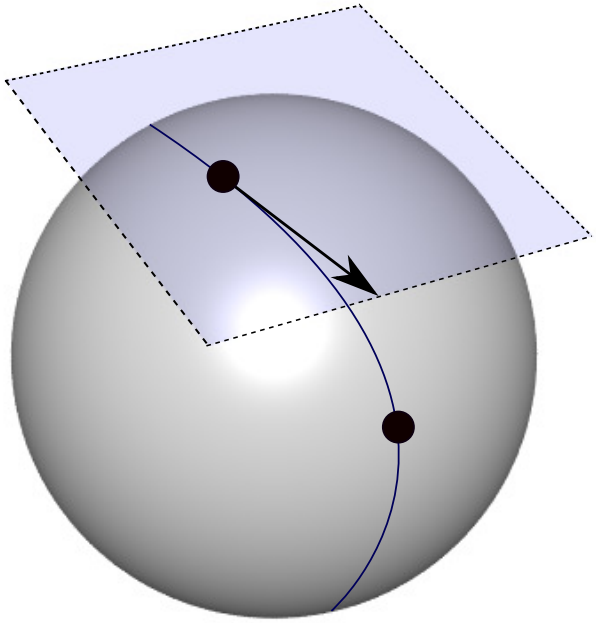

Figure 10: Two points on the curves at a particular height in ρ. The relative position vector $\boldsymbol{r}(\rho)$ points along the geodesic on a constant ρ surface from the first curve to the second.

2.3 Crossings

Suppose two closed curves wind around each other. At a given radial height ρ, a crossing will be seen from some range of observation angles. Specifically, a crossing will occur at an observation angle when the relative position vector, $\boldsymbol{r}(\rho)$, is pointing either directly towards or away from an observer who is looking along geodesics in

the shell of constant ρ. So if $\boldsymbol{r}(\rho)$ makes n complete rotations, all observers will see $2n$ crossings. Later we will use rotations of $\boldsymbol{r}$ to measure the number of crossings.

2.4 Division

We wish to use the Frenet frame as a basis, so we will – following Berger & Prior [1] – split the curve into sections, with end points at extrema. Curves can then be parametrised in terms of ρ. Let $\sigma_i(\rho)$ be defined as:

$$\sigma_i(\rho) = \begin{cases} 1, & \rho \in (\rho_i, \rho_{i+1}) \text{ and } ds/d\rho > 0; \\ -1, & \rho \in (\rho_i, \rho_{i+1}) \text{ and } ds/d\rho < 0; \\ 0, & \rho \notin (\rho_i, \rho_{i+1}). \end{cases} \tag{83}$$

σ can then be used to keep track of whether a section i is a monotonically increasing or monotonically decreasing segment.

2.5 Winding

When measuring how much the curves wind around each other we are measuring how much the relative position vector $\boldsymbol{r}$ rotates. Let α be the amount $\boldsymbol{r}$ rotates, then the rotation rate of α is given by:

$$\frac{d\alpha}{d\rho} = \frac{\hat{\boldsymbol{\rho}} \cdot \boldsymbol{r}(\rho) \times \boldsymbol{r}'(\rho)}{|\boldsymbol{r}(\rho)|^2}. \tag{84}$$

The net winding angle between two spherical heights is

$$\Delta\alpha = \int_{\rho_1}^{\rho_2} \frac{d\alpha}{d\rho} d\rho. \tag{85}$$

The winding number is:

$$w = \frac{1}{2\pi} \int_{\rho_1}^{\rho_2} \frac{1}{r^2} \hat{\boldsymbol{\rho}} \cdot \boldsymbol{r} \times \frac{d\boldsymbol{r}}{d\rho} d\rho. \tag{86}$$

2.6 Linking number

For a closed curve the linking number equals half the signed (according to figure 2) number of crossings seen from any projection. We wish to establish linking number in terms of winding numbers in the ρ direction, so we will be interested in directions perpendicular to ρ: namely, the directions obtained by varying β in:

$$\hat{\boldsymbol{n}}(\beta) = \cos(\beta)\hat{\boldsymbol{\theta}} + \sin(\beta)\hat{\boldsymbol{\phi}}. \tag{87}$$

Theorem 1.

Let $\boldsymbol{x}$ be a closed curve with pieces $i = 1, ..., n$, and curve $\boldsymbol{y}$ be a closed curve

with pieces $j = 1, .., m$. Let α_{ij} be the orientation of the relative position vector $\boldsymbol{r}_{ij}(\rho) = \boldsymbol{y}_j(\rho) - \boldsymbol{x}_i(\rho)$ relative to $\hat{\boldsymbol{\phi}}$.

Then the linking of the two curves is:

$$L_k = \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{1}{2\pi}\int_0^\infty \sigma_i\sigma_j\frac{d\alpha_{ij}}{d\rho}d\rho, \tag{88}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\sigma_i\sigma_j}{2\pi}\Delta\alpha_{ij}. \tag{89}$$

Proof of theorem 1.
Following Berger and Prior (2001) [1], the theorem will be proved by examining the crossings of the two curves. First, recall how the crossing number $C(\hat{\boldsymbol{n}})$ relates to linking:

$$L_k = \frac{1}{2}C(\hat{\boldsymbol{n}}), \tag{90}$$

where $C = C_+ - C_-$ and $\hat{\boldsymbol{n}}$ is a direction obtained by varying β in equation 87.

Consider a region in ρ where both curves exist and are travelling upwards (so $\sigma_i\sigma_j = 1$). Note that if only one or neither of the curves exist then $\sigma_i\sigma_j = 0$. These curves wind around each other through an angle $\Delta\alpha_{ij}$ according to equation 85. Consider observers perpendicular to $\hat{\boldsymbol{\rho}}$; that is, an observer viewing from a direction $\hat{\boldsymbol{n}}$ in equation 87. If $\boldsymbol{r}_{ij}$ makes a complete rotation then all observers will see two crossings. If $\boldsymbol{r}_{ij}$ only makes a partial rotation then $\Delta\alpha_{ij}/\pi$ gives the fraction of observers that see a crossing. If an observer witnesses a series of crossings due to $\boldsymbol{r}_{ij}$ continually oscillating back and forth, then each crossing will have a different sign and cancel out. So $\Delta\alpha_{ij}/\pi$ gives the average number of crossings over all projection angles.

Next, consider what would happen if one of σs is of a different sign. When they are both of the same sign and $\Delta\alpha_{ij} > 0$ the crossing will be positive. If one of the curves has a different sign then the crossing becomes negative. Hence the average signed crossing number is

$$\bar{C} = \frac{1}{\pi}\sum_{i=1}^{n}\sum_{j=1}^{m}\sigma_i\sigma_j\Delta\alpha_{ij}. \tag{91}$$

The linking number equals half this number which gives equation 89.

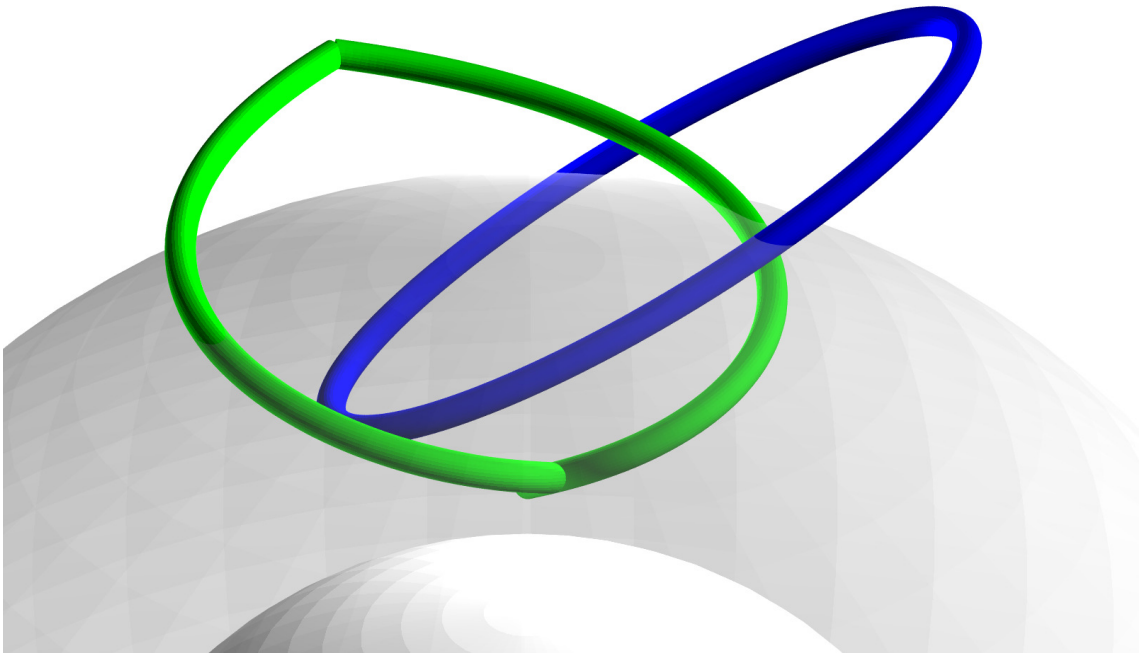

Figure 11: We wish to formulate an expression for the linking number of the portion of the curves caught between the two shells.

2.7 The linking of open curves

From now on we will make a notational distinction between classical closed-curve linking, L_k, and linking, $\mathcal{L}$, concerning the linking of open curves. Later on this will be done similarly for open writhe $\mathcal{W}$ and open twist $\mathcal{T}$.

We wish to define linking number for curves that do not close upon themselves. Consider two entangled closed curves that are both cut by a spherical shell as in figure 11. Let ρ_{min} and ρ_{max} be the maximum and minimum heights both curves reach and let ρ_0 define a shell somewhere in between. Define the winding of the curves below ρ_0 as:

$$\mathcal{L}(\rho_0) = \int_0^{\rho_0} \frac{d\mathcal{L}}{d\rho} d\rho = \int_{\rho_{min}}^{\rho_0} \frac{d\mathcal{L}}{d\rho} d\rho. \tag{92}$$

Also,

$$\frac{d\mathcal{L}}{d\rho} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{d\mathcal{L}_{ij}}{d\rho}, \tag{93}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\sigma_i\sigma_j}{2\pi} \frac{d\alpha_{ij}}{d\rho}. \tag{94}$$

Theorem 2.

$\mathcal{L}(\rho_0)$ is invariant to motions which do not move the intersecting points of the curves

with the $\rho = \rho_0$ shell.

Proof of theorem 2.
First consider motions of the curves below ρ_0 which do not move the intersecting points. In the region where $\rho > \rho_0$ the winding of the curves is unaffected, so the linking

$$\int_{\rho_0}^{\rho_{max}} \frac{d\mathcal{L}}{d\rho} d\rho \tag{95}$$

does not change. The total linking number, L_k, is invariant, so the difference

$$\mathcal{L}(\rho_0) = L_k - \int_{\rho_0}^{\rho_{max}} \frac{d\mathcal{L}}{d\rho} d\rho \tag{96}$$

must be as well. Hence motions below ρ_0 do not alter $\mathcal{L}(\rho_0)$. Next, consider motions above ρ_0. These motions do not change the winding of the curves below ρ_0, so $\mathcal{L}(\rho_0)$ is unaffected.

Corollary to theorem 2. Now consider a situation where the two curves are bound between two spherical shells, and let the shells be positioned so that the curves are open in their extent in-between – these open segments can be thought of as a braid (see figure 12). By theorem 2 the linking of the two curves is invariant to motions which do not move the end points.

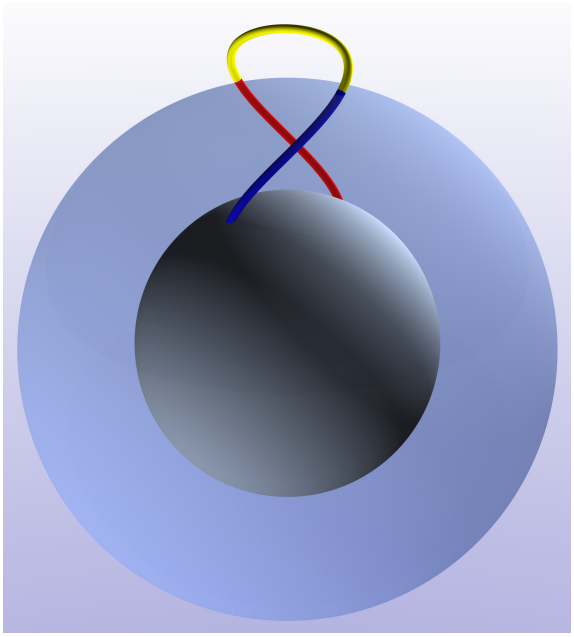

Figure 12: By theorem 2 the linking of the red and blue curves is invariant to motions that disappear at the boundary.

2.8 Deformations

The nature of spherical geometry means that we must examine an additional issue. Consider the two red and black ribbons in figure 12. Can we smoothly deform the red ribbon so as to reverse the sign of the crossing? It turns out you can – see figure 13. Here the end points of the ribbons have remained fixed, but the black ribbon is now on top.

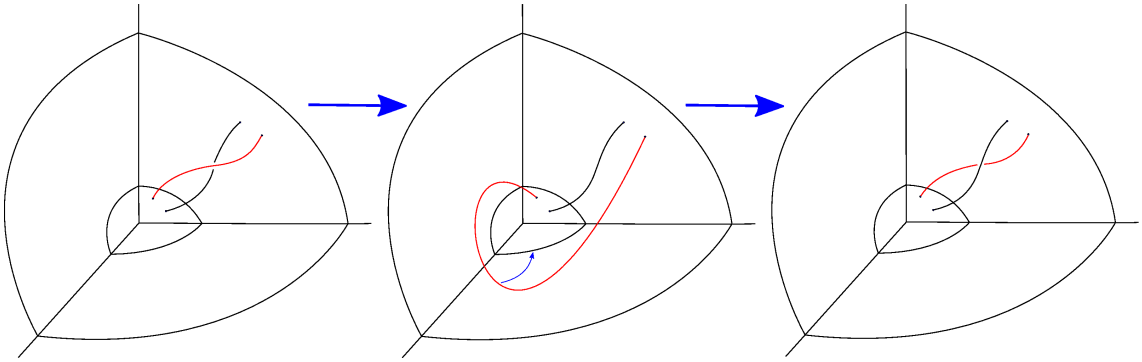

Figure 13: The ribbon in red is smoothly deformed keeping the end points fixed. The winding of the two ribbons has changed by a degree of one.

It might seem that the linking number according to equation 89 has changed, however, this is not the case: the linking number is still equal to one. Let's examine what is happening by looking at figure 14. In the process of moving the middle part of the ribbon, twist has been injected. As the ribbon is deformed writhe is converted into twist. If we expand the sum of winding numbers in equation 89

$$\mathcal{L} = \frac{1}{2\pi}(\Delta\alpha_{12} + \Delta\alpha_{21} + \Delta\alpha_{11} + \Delta\alpha_{22}). \tag{97}$$

The self-winding quantities refer to internal twist in the ribbons, whilst the cross-term represents the writhe contribution. For the diagram on the left of figure 14:

$$\Delta\alpha_{12} = \Delta\alpha_{21} = \pi, \quad \Delta\alpha_{11} = \Delta\alpha_{22} = 0, \quad \mathcal{L} = \frac{1}{2\pi}(\pi + \pi) = 1.$$

Whilst on the right we have:

$$\Delta\alpha_{12} = \Delta\alpha_{21} = -\pi, \quad \Delta\alpha_{11} = 4\pi, \quad \Delta\alpha_{22} = 0, \quad \mathcal{L} = \frac{1}{2\pi}(-\pi - \pi + 4\pi) = 1.$$

Note that this procedure could be performed any number of times. For example, if we deformed the green ribbon in figure 14 so that it returned to its original configuration we would have:

$$\begin{aligned} \Delta\alpha_{12} &= \Delta\alpha_{21} = \pi, \quad \Delta\alpha_{11} = 4\pi, \quad \Delta\alpha_{22} = -4\pi, \\ \mathcal{L} &= \frac{1}{2\pi}(\pi + \pi + 4\pi - 4\pi) = 1. \end{aligned}$$

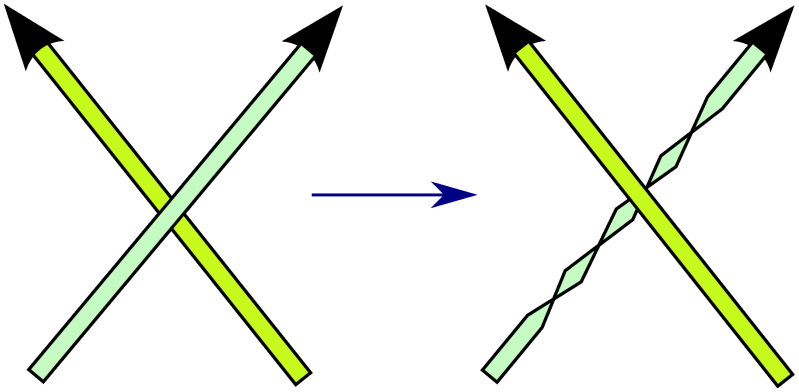

Figure 14: Before deforming: $\mathcal{L} = 1$, $\mathcal{T} = 0$ and $\mathcal{W} = 1$. Post deformation: $\mathcal{L} = 1$, $\mathcal{T} = 2$ and $\mathcal{W} = -1$. Hence in both cases $\mathcal{L} = 1$ as expected.

So we can conclude that a deformation of this form does not affect the linking number.

The above example illustrates an important point: in this spherical system, only for ribbons, and not single curves, is linking number conserved – a single curve has no thickness to become twisted.

2.9 Twist of open curves

Twist is a purely local quantity; it only involves vectors in the immediate neighbourhood of the axis curve. Thus it is meaningful to write

$$\mathcal{T}(\rho_0) = \int_{\rho_{min}}^{\rho_0} \frac{d\mathcal{T}(\rho)}{d\rho} d\rho, \tag{98}$$

to denote the twist of an open curve. We will now use equation 83 to split the curve into pieces and express twist as the sum of these individual contributions. Let the curve in question be parameterised by arc-length s. Recall that twist is defined as

$$T_w = \frac{1}{2\pi} \oint_{\boldsymbol{x}} \hat{\boldsymbol{T}}(s) \cdot \hat{\boldsymbol{V}}(s) \times \frac{d\hat{\boldsymbol{V}}(s)}{ds} ds. \tag{99}$$

And so

$$\begin{aligned} \mathcal{T} &= \sum_i \mathcal{T}_i, & (100)\\ &= \sum_i \int_{s_i^{min}}^{s_i^{max}} \frac{d\mathcal{T}_i}{ds} ds, & (101)\\ &= \sum_i \int_{\rho_i^{min}}^{\rho_i^{max}} \frac{d\mathcal{T}_i}{ds} \left|\frac{ds}{d\rho}\right| d\rho. & (102) \end{aligned}$$

Note that σ_i keeps track of the sign of $\frac{ds}{d\rho}$. From equation 99 and the fundamental theorem of calculus:

$$\frac{d\mathcal{T}}{d\rho} = \sum_i \frac{d\mathcal{T}_i}{ds} \left|\frac{ds}{d\rho}\right| = \sum_i \frac{\sigma_i}{2\pi} \hat{\boldsymbol{T}}_i(\rho) \cdot \hat{\boldsymbol{V}}_i(\rho) \times \hat{\boldsymbol{V}}_i'(\rho). \tag{103}$$

Finally,

$$\mathcal{T}(\rho_0) \;=\; \int_{\rho_{min}}^{\rho_0} \sum_i \frac{\sigma_i}{2\pi} \hat{\boldsymbol{T}}_i(\rho) \cdot \hat{\boldsymbol{V}}_i(\rho) \times \hat{\boldsymbol{V}}'_i(\rho)\, d\rho. \tag{104}$$

2.10 Writhe of open curves

Define the writhe between two radial heights as

$$\mathcal{W}(\rho_0) = \int_{\rho_{min}}^{\rho_0} \frac{d\mathcal{W}(\rho)}{d\rho} d\rho, \tag{105}$$

where the derivative is still to be determined.

Note that we can split up equation 89 into a local part where $i = j$, and a non-local part where $i \neq j$. This can be used to write the change of winding as

$$\mathcal{L}'(\rho) = \sum_i \mathcal{L}'_i(\rho) + \sum_{i \neq j} \mathcal{L}'_{ij}(\rho). \tag{106}$$

Here the first term is the local, and the second, the non-local contribution respectively. Writhe can be decomposed in a similar manner:

$$\mathcal{W}'(\rho) = \sum_i \mathcal{W}'_i(\rho) + \sum_{i \neq j} \mathcal{W}'_{ij}(\rho). \tag{107}$$

Using Călugăreanu's theorem (equation 19), this quantity can be expressed in terms of twist and linking number. The local part is

$$\mathcal{W}'_i(\rho) = \mathcal{L}'_i(\rho) - \mathcal{T}'_i(\rho), \tag{108}$$

and the non-local contribution:

$$\mathcal{W}'_{ij}(\rho) = \mathcal{L}'_{ij}(\rho) - \mathcal{T}'_{ij}(\rho) = \mathcal{L}'_{ij}(\rho) = \frac{\sigma_i \sigma_j}{2\pi} \alpha'_{ij}(\rho) \quad \text{for } (i \neq j). \tag{109}$$

For consistency with the standard concept of writhe, it would be good if these new quantities $\mathcal{W}'_i$ and $\mathcal{W}'_{ij}$ just depended on the axis curve. It turns out – see theorem 3 – that the local part $\mathcal{W}'_i$ does, but unfortunately, $\mathcal{W}'_{ij}$ depends on the secondary curve as well.

To examine the local part of $\mathcal{W}$ we will need the concept of a twist-tube. Let the axis curve $\boldsymbol{x}$ define the central axis of a tube and let the secondary curve $\boldsymbol{y}$ lie in the tube's surface. We will keep track of how the tube is twisted by using something similar to cylindrical coordinates and setting $\boldsymbol{y}$ to be the azimuthal axis; a particular choice of $\boldsymbol{y}$ is called a framing [1]. So the winding of $\boldsymbol{y}$ corresponds to the amount by which the tube is twisted (in an untwisted tube, $\boldsymbol{y}$ would not wind around $\boldsymbol{x}$). A

twisted-tube is a tube constructed in this manner whose surface is filled with curves parallel to $\boldsymbol{y}$ (a tube like this with elastic energy is called an isotropic rod [53]).

It will be useful to define some notation and two sets of basis vectors. First define $\lambda = d\rho/ds$ and $\mu = |\hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{T}}|$. Parameterise $\boldsymbol{x}$ by arc-length s and $\boldsymbol{y}$ by arc-length s_1. Define the unit tangent vector

$$\hat{\boldsymbol{T}}(s) = \frac{d\boldsymbol{x}}{ds} = \lambda\hat{\boldsymbol{\rho}} + \mu\hat{\boldsymbol{T}}_{\perp}, \tag{110}$$

where $\hat{\boldsymbol{T}}_{\perp}$ denotes the part of the tangent vector that is perpendicular to $\hat{\boldsymbol{\rho}}$.

Surround the axis curve $\boldsymbol{x}$ with a tube and let $\hat{\boldsymbol{V}}$ be the unique (as described below) unit vector normal to $\hat{\boldsymbol{T}}$ i.e. $\hat{\boldsymbol{V}} \cdot \hat{\boldsymbol{T}} = \boldsymbol{0}$. Let $\hat{\boldsymbol{W}} = \hat{\boldsymbol{T}} \times \hat{\boldsymbol{V}}$ so that $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{V}}, \hat{\boldsymbol{W}}\}$ forms the first orthonormal basis. There exists a family of secondary curves $\boldsymbol{y}$ on this tube, one for each value of $\beta \in [0, 2\pi)$ in

$$\boldsymbol{y}(\rho, \beta) = \boldsymbol{x}(\rho) + \epsilon\hat{\boldsymbol{U}}(\rho, \beta), \tag{111}$$

where

$$\hat{\boldsymbol{U}}(\rho, \beta) = \cos(\beta)\hat{\boldsymbol{V}}(\rho) + \sin(\beta)\hat{\boldsymbol{W}}(\rho). \tag{112}$$

Note that $\hat{\boldsymbol{V}}$ is unique as it always points to the $\beta = 0$ secondary curve. Define the second basis as (see the left-hand side of figure 16)

$$\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{f}}, \hat{\boldsymbol{g}}\} = \{\hat{\boldsymbol{T}}, \hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{T}}/\mu, \hat{\boldsymbol{T}} \times (\hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{T}}/\mu)\}. \tag{113}$$

(ignoring for time being the possibility of $\hat{\boldsymbol{T}}$ being parallel to $\hat{\boldsymbol{\rho}}$). As $\hat{\boldsymbol{V}}$ and $\hat{\boldsymbol{W}}$ are perpendicular to $\hat{\boldsymbol{T}}$ we may write

$$\begin{pmatrix} \hat{\boldsymbol{V}} \\ \hat{\boldsymbol{W}} \end{pmatrix} = \begin{pmatrix} \cos\psi(\rho) & \sin\psi(\rho) \\ -\sin\psi(\rho) & \cos\psi(\rho) \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{f}} \\ \hat{\boldsymbol{g}} \end{pmatrix}. \tag{114}$$

for some angle of rotation $\psi(\rho)$.

Theorem 3.
Consider the set of secondary curves living in a twisted tube's surface and let

$$\mathcal{W}_i'(\rho) \equiv < \mathcal{L}_i'(\rho) - \mathcal{T}_i'(\rho) >, \tag{115}$$

where $<>$ denotes an average over all secondary curves in the tube surface. Then $\mathcal{W}_i'$ only depends on the axis curve and is given by

$$\mathcal{W}_i'(\rho) = \frac{\kappa_i B_{\rho_i}}{2\pi\lambda_i} \frac{1}{(1 + |\lambda_i|)}, \tag{116}$$

where B_ρ is the radial component of the binormal vector (see equation 21). Also

$$\mathcal{T}_i'(\rho) = \frac{1}{2\pi}(\psi' + \mu^{-2}\kappa B_\rho), \tag{117}$$

and

$$\mathcal{L}_i'(\rho) = \frac{1}{2\pi}\left(\psi' + \lambda^{-1}\mu^{-2}\kappa B_\rho\right). \tag{118}$$

Proof of theorem 3.

Equation 103 can now be written in terms of the contribution from piece i according to equation 83:

$$\mathcal{T}_i = \int_{s_i^{min}}^{s_i^{max}} \frac{d\mathcal{T}_i}{ds} ds = \int_{\rho_i^{min}}^{\rho_i^{max}} \frac{d\mathcal{T}_i}{ds}\left|\frac{ds}{d\rho}\right| d\rho, \tag{119}$$

and so

$$\mathcal{T}_i'(\rho) = \frac{d\mathcal{T}_i}{ds}\left|\frac{ds}{d\rho}\right| = \frac{\sigma_i}{2\pi}\hat{\boldsymbol{T}}_i(\rho)\cdot\hat{\boldsymbol{V}}_i(\rho)\times\hat{\boldsymbol{V}}_i'(\rho). \tag{120}$$

We will consider a neighbourhood of some point on the curve which is not a maxima or minima so that the curve can be parametrised by ρ, i.e. $\lambda = d\rho/ds \neq 0$. As twist and writhe do not change under a reversal of a curve we can assume $\lambda > 0$, so by our definition $\sigma > 0$.

As $\hat{\boldsymbol{V}}$ and $\hat{\boldsymbol{W}}$ are orthogonal, $\hat{\boldsymbol{V}}\cdot\hat{\boldsymbol{W}} = 0$. By the product rule

$$\hat{\boldsymbol{V}}'\cdot\hat{\boldsymbol{W}} = -\hat{\boldsymbol{W}}'\cdot\hat{\boldsymbol{V}} \equiv \omega. \tag{121}$$

This expression – note its lack of dependence on choice of secondary curve (β) – will simplify things later on. From equation (103) with $\sigma > 0$ we have,

$$\begin{aligned}
2\pi\mathcal{T}' &= \hat{\boldsymbol{T}}\cdot\hat{\boldsymbol{U}}\times\hat{\boldsymbol{U}}', && (122)\\
&= \hat{\boldsymbol{T}}\times(\cos(\beta)\hat{\boldsymbol{V}} + \sin(\beta)\hat{\boldsymbol{W}})\cdot(\cos(\beta)\hat{\boldsymbol{V}}' + \sin(\beta)\hat{\boldsymbol{W}}'), && (123)\\
&= (\cos(\beta)\hat{\boldsymbol{W}} - \sin(\beta)\hat{\boldsymbol{V}})\cdot(\cos(\beta)\hat{\boldsymbol{V}}' + \sin(\beta)\hat{\boldsymbol{W}}'), && (124)\\
&= \omega. && (125)
\end{aligned}$$

So $\mathcal{T}'$ only depends on the geometry of the axis curve, which by equation 98 means that $\mathcal{T}$ only depends on the axis curve.

Consider the relative position vector $\boldsymbol{r}$ at some point $\boldsymbol{x}(s)$. The tip of the $\boldsymbol{r}(\rho)$ vector points to a point on a secondary curve $\boldsymbol{y}$ at the different arc-length s_1 at the same radial height. We wish to express $\boldsymbol{x}(s_1)$ and $\hat{\boldsymbol{U}}(s_1,\beta)$ in terms of vectors involving s (see figure 15). Taking a Taylor expansion about s and putting in s_1 gives

$$\begin{aligned}
\boldsymbol{x}(s_1) &= \boldsymbol{x}(s) + (s_1 - s)\frac{d\boldsymbol{x}}{ds} + ..., && (126)\\
\hat{\boldsymbol{U}}(s_1,\beta) &= \hat{\boldsymbol{U}}(s,\beta) + (s_1 - s)\frac{d\hat{\boldsymbol{U}}}{ds} + ... && (127)
\end{aligned}$$

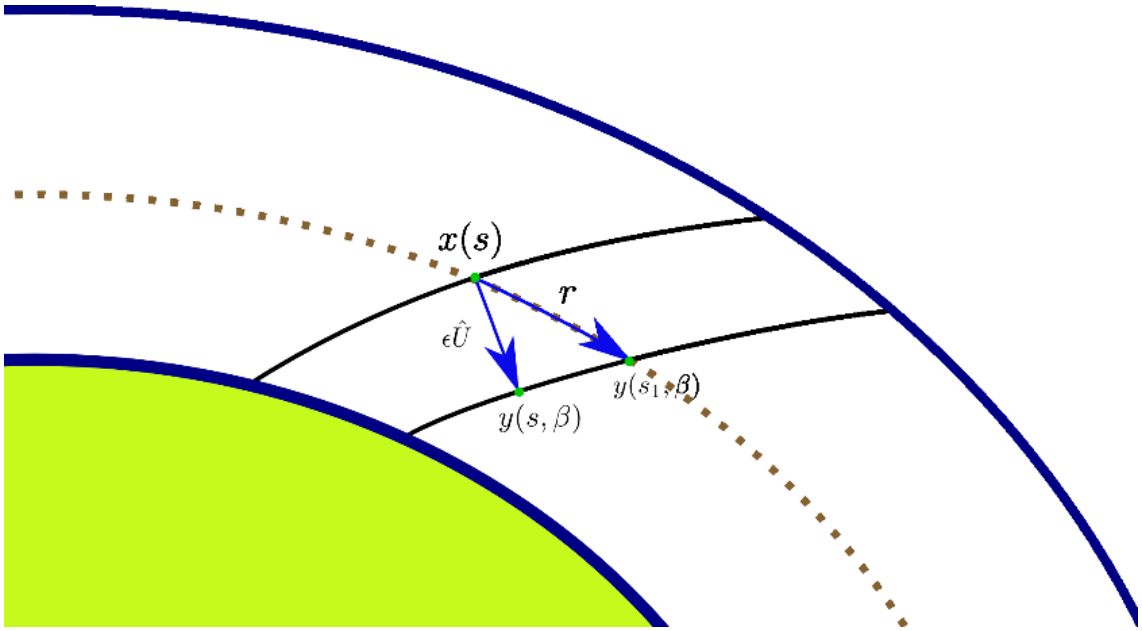

Figure 15: We seek to express $\boldsymbol{x}(s_1)$ and $\hat{\boldsymbol{U}}(s_1,\beta)$ in terms of vectors involving s.

Therefore

$$\begin{aligned} \boldsymbol{y}(s_1,\beta) &= \boldsymbol{x}(s_1)+\epsilon\hat{\boldsymbol{U}}(s_1,\beta), && (128)\\ &\approx \boldsymbol{x}(s)+\epsilon\hat{\boldsymbol{U}}(s,\beta)+\left(\hat{\boldsymbol{T}}(s)+\epsilon\frac{d\hat{\boldsymbol{U}}(s,\beta)}{ds}\right)(s_1-s). && (129)\end{aligned}$$

Dropping higher order terms

$$\begin{aligned} \boldsymbol{r} &= \boldsymbol{y}(s_1,\beta)-\boldsymbol{x}(s), && (130)\\ &\approx \epsilon\hat{\boldsymbol{U}}(s,\beta)+\hat{\boldsymbol{T}}(s)(s_1-s). && (131)\end{aligned}$$

The radial component of $\boldsymbol{r}$, $\boldsymbol{r}_\rho$, is zero by definition. We have

$$s_1-s\approx -\epsilon U_\rho(s)/T_\rho(s)=-\epsilon U_\rho(s)/\lambda(s). \tag{132}$$

So to the first order in ϵ,

$$\boldsymbol{r}=\epsilon(\hat{\boldsymbol{U}}-\lambda^{-1}U_\rho\hat{\boldsymbol{T}}). \tag{133}$$

Define $\boldsymbol{R}=\boldsymbol{r}/\epsilon$. Using equations 84 and 94, and letting $\epsilon\to 0$ gives

$$\mathcal{L}'=\frac{\hat{\boldsymbol{\rho}}\cdot\boldsymbol{R}\times\boldsymbol{R}'}{2\pi|\boldsymbol{R}|^2}. \tag{134}$$

We will now need to express $\boldsymbol{R}$ in two new coordinate systems. Recall that $\mu=|\hat{\boldsymbol{\rho}}\times\hat{\boldsymbol{T}}|$ and that

$$\{\hat{\boldsymbol{T}},\hat{\boldsymbol{f}},\hat{\boldsymbol{g}}\}=\{\hat{\boldsymbol{T}},\hat{\boldsymbol{\rho}}\times\hat{\boldsymbol{T}}/\mu,\hat{\boldsymbol{T}}\times(\hat{\boldsymbol{\rho}}\times\hat{\boldsymbol{T}}/\mu)\}, \tag{135}$$

and

$$\begin{pmatrix}\hat{\boldsymbol{V}}\\ \hat{\boldsymbol{W}}\end{pmatrix}=\begin{pmatrix}\cos\psi(\rho) & \sin\psi(\rho)\\ -\sin\psi(\rho) & \cos\psi(\rho)\end{pmatrix}\begin{pmatrix}\hat{\boldsymbol{f}}\\ \hat{\boldsymbol{g}}\end{pmatrix}. \tag{136}$$

This can be used to rewrite equation 112 in terms of $\hat{\boldsymbol{f}}$ and $\hat{\boldsymbol{g}}$.

$$\begin{aligned}\hat{\boldsymbol{U}} &= \cos(\beta)\hat{\boldsymbol{V}}(\rho) + \sin(\beta)\hat{\boldsymbol{W}}(\rho), &(137)\\ &= \cos(\beta+\psi)\hat{\boldsymbol{f}} + \sin(\beta+\psi)\hat{\boldsymbol{g}}. &(138)\end{aligned}$$

$\mathcal{L}'$ is a measure of how much winding the relative position vector $\boldsymbol{r}$ does about $\hat{\boldsymbol{\rho}}$. With this in mind, we will seek to decompose $\boldsymbol{r}$ into basis vectors that lie in the plane orthogonal to $\hat{\boldsymbol{\rho}}$. Let

$$\hat{\boldsymbol{h}} = \hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{f}} = -\hat{\boldsymbol{T}}_\perp/\mu. \tag{139}$$

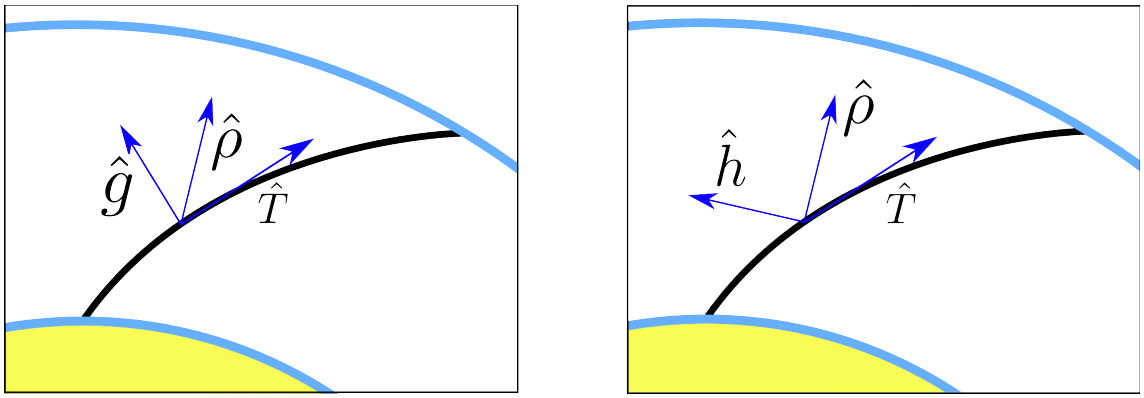

Figure 16: The two frames $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{f}}, \hat{\boldsymbol{g}}\}$ and $\{\hat{\boldsymbol{\rho}}, \hat{\boldsymbol{f}}, \hat{\boldsymbol{h}}\}$. For both figures $\hat{\boldsymbol{f}}$ is directed into the page.

So we now have such a frame (see figure 16). We need to express the old basis $\{\hat{\boldsymbol{T}}, \hat{\boldsymbol{f}}, \hat{\boldsymbol{g}}\}$ in terms of this one. The 'bac-cab' vector identity gives

$$\begin{aligned}\hat{\boldsymbol{g}} &= \hat{\boldsymbol{T}} \times (\hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{T}}/\mu), &(140)\\ &= \hat{\boldsymbol{\rho}}(\hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{T}}/\mu) - \hat{\boldsymbol{T}}/\mu(\hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{\rho}}), &(141)\\ &= \mu^{-1}(\hat{\boldsymbol{\rho}} - T_\rho \hat{\boldsymbol{T}}), &(142)\\ &= \mu^{-1}(\hat{\boldsymbol{\rho}} - T_\rho(\hat{\boldsymbol{T}}_\rho + \hat{\boldsymbol{T}}_\perp)), &(143)\\ &= \mu^{-1}(\hat{\boldsymbol{\rho}} - \lambda(\hat{\boldsymbol{T}}_\perp + \lambda\hat{\boldsymbol{\rho}})), &(144)\\ &= \mu^{-1}((1-\lambda^2)\hat{\boldsymbol{\rho}} - \lambda\hat{\boldsymbol{T}}_\perp). &(145)\end{aligned}$$

Using Pythagoras' theorem

$$\begin{aligned}1-\lambda^2 &= 1 - \hat{\boldsymbol{T}}_\rho^2, &(146)\\ &= \hat{\boldsymbol{T}}_\perp^2, &(147)\\ &= |\hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{T}}_\perp|^2, &(148)\\ &= \mu^2. &(149)\end{aligned}$$

So finally we can write

$$\hat{\boldsymbol{g}} = \mu\hat{\boldsymbol{\rho}} + \lambda\hat{\boldsymbol{h}}. \tag{150}$$

We can do the same for the tangent vector

$$\begin{aligned}\hat{\boldsymbol{T}} &= \hat{\boldsymbol{T}}_\rho + \hat{\boldsymbol{T}}_\perp, &(151)\\ &= \lambda\hat{\boldsymbol{\rho}} - \mu\hat{\boldsymbol{h}}. &(152)\end{aligned}$$

Now expressing $\hat{\boldsymbol{U}}$ in this basis using 138

$$\hat{\boldsymbol{U}} = \mu\sin(\beta+\psi)\hat{\boldsymbol{\rho}} + \cos(\beta+\psi)\hat{\boldsymbol{f}} + \lambda\sin(\beta+\psi)\hat{\boldsymbol{h}}. \tag{153}$$

We can use this to rewrite $\boldsymbol{R}$ in this new frame

$$\begin{aligned}\boldsymbol{R} &= \cos(\beta+\psi)\hat{\boldsymbol{f}} + \lambda^{-1}\sin(\beta+\psi)\hat{\boldsymbol{h}}; &(154)\\ \boldsymbol{R}^2 &= \cos^2(\beta+\psi) + \lambda^{-2}\sin^2(\beta+\psi). &(155)\end{aligned}$$

The last quantity we need to work out for equation 134 is

$$\boldsymbol{R}' = \cos(\beta+\psi)(\lambda^{-1}\psi'\hat{\boldsymbol{h}} + \hat{\boldsymbol{f}}') + \sin(\beta+\psi)(-\psi'\hat{\boldsymbol{f}} + \lambda^{-1}\hat{\boldsymbol{h}}' - \lambda'\lambda^{-2}\hat{\boldsymbol{h}}). \tag{156}$$

Using vector identities gives us

$$\begin{aligned}(\hat{\boldsymbol{f}}\times\hat{\boldsymbol{h}}' + \hat{\boldsymbol{h}}\times\hat{\boldsymbol{f}}')\cdot\hat{\boldsymbol{\rho}} &= \left((\hat{\boldsymbol{f}}\cdot\hat{\boldsymbol{f}}')\hat{\boldsymbol{\rho}} - (\hat{\boldsymbol{f}}\cdot\hat{\boldsymbol{\rho}})\hat{\boldsymbol{f}}' + (\hat{\boldsymbol{\rho}}\cdot\hat{\boldsymbol{f}}')\hat{\boldsymbol{f}} - (\hat{\boldsymbol{\rho}}\cdot\hat{\boldsymbol{f}})\hat{\boldsymbol{f}}'\right)\cdot\hat{\boldsymbol{\rho}},\\ &= 0.\end{aligned}$$

Note that

$$\begin{aligned}\hat{\boldsymbol{f}}\times\hat{\boldsymbol{f}}' &= \hat{\boldsymbol{h}}\times\hat{\boldsymbol{h}}' = \left(\mu^{-2}\hat{\boldsymbol{\rho}}\cdot(\hat{\boldsymbol{T}}\times\hat{\boldsymbol{T}}')\right)\hat{\boldsymbol{\rho}}, &(157)\\ &= \mu^{-2}\lambda^{-1}\kappa\left(\hat{\boldsymbol{\rho}}\cdot(\hat{\boldsymbol{T}}\times\hat{\boldsymbol{N}})\right)\hat{\boldsymbol{\rho}}, &(158)\\ &= \mu^{-2}\lambda^{-1}\kappa B_\rho\hat{\boldsymbol{\rho}}. &(159)\end{aligned}$$

where B_ρ is the radial component of the binormal vector $\hat{\boldsymbol{B}}$. Now we are ready to calculate $\mathcal{L}'$.

$$\begin{aligned}\hat{\boldsymbol{\rho}}\cdot\boldsymbol{R}\times\boldsymbol{R}' &= \lambda^{-1}\psi' + \sin(\beta+\psi)\cos(\beta+\psi)\lambda'\lambda^{-2} &(160)\\ &\quad +(\sin^2(\beta+\psi)\lambda^{-2} + \cos^2(\beta+\psi))(\lambda^{-1}\mu^{-2}\kappa B_\rho).\end{aligned}$$

So finally,

$$\begin{aligned}\mathcal{L}' &= \frac{\hat{\boldsymbol{\rho}}\cdot\boldsymbol{R}\times\boldsymbol{R}'}{2\pi|\boldsymbol{R}|^2},\\ &= \frac{1}{2\pi}\left(\frac{\lambda\psi' - \lambda'\sin(\beta+\psi)\cos(\beta+\psi)}{\sin^2(\beta+\psi) + \lambda^2\cos^2(\beta+\psi)} + \kappa\lambda^{-1}\mu^{-2}B_\rho\right). &(161)\end{aligned}$$

Recall that a choice of secondary curve in the twisted-tube is equivalent to a choice of $\beta \in [0, 2\pi)$. We wish to average this expression over all secondary curves $\boldsymbol{y}$ in the twisted-tube, so we need to integrate 161 over β. The last term does not depend on β and so is constant. We deal with the first two terms separately. The first one gives

$$I = \frac{\lambda\psi'}{2\pi}\int_0^{2\pi} \frac{1}{\lambda^2\cos^2(\beta+\psi)+\sin^2(\beta+\psi)}d\beta. \tag{162}$$

Let $t = \beta + \psi$, so $dt = d\beta$:

$$= \frac{\lambda\psi'}{2\pi}\int \frac{1}{\lambda^2\cos^2(t)+\sin^2(t)}dt, \tag{163}$$

$$= \frac{\lambda\psi'}{2\pi}\int \frac{1}{\lambda^2+(1-\lambda^2)\sin^2(t)}dt. \tag{164}$$

As $\cos^2(t) = 1 - \sin^2(t)$. Now multiplying through by $\sec^2(t)$ gives:

$$= \frac{\lambda\psi'}{2\pi}\int \frac{\sec^2(t)}{\lambda^2\sec^2(t)+\tan^2(t)-\lambda^2\tan^2(t)}dt, \tag{165}$$

$$= \frac{\lambda\psi'}{2\pi}\int \frac{\sec^2(t)}{\lambda^2+\tan^2(t)}dt. \tag{166}$$

(as $\sec^2(t) = 1 + \tan^2(t)$). Now let $u = \tan(t)$, so $du = \sec^2(t)dt$.

$$= \frac{\lambda\psi'}{2\pi}\int \frac{1}{\lambda^2+u^2}du, \tag{167}$$

$$= \frac{\psi'}{2\pi\lambda}\int \frac{1}{1+(\frac{u}{\lambda})^2}du. \tag{168}$$

Let $s = u/\lambda$, so $ds = 1/\lambda$ du.

$$= \frac{\psi'}{2\pi}\int \frac{1}{1+s^2}ds, \tag{169}$$

$$= \frac{\psi'}{2\pi}\tan^{-1}(s), \tag{170}$$

$$= \frac{\psi'}{2\pi}\tan^{-1}\left(\frac{\tan(\psi+\beta)}{\lambda}\right). \tag{171}$$

Let s_0 be the value of s when $\beta = 0$ i.e.

$$s_0 \equiv \frac{\tan(\psi)}{\lambda}. \tag{172}$$

Note that $\tan(\beta+\psi)$ will go to infinity when $\beta+\psi = \pi/2$ and $\beta+\psi = -\pi/2$. $\tan^{-1}$ is a multi-valued function, so we have a choice of which branch to select. We will choose the one at the origin and proceed by breaking up the integral at its discontinuities. Suppose $-\pi/2 < s_0 < \pi/2$. If s_0 is not in this range subtract

multiples of π until it is.

$$\begin{aligned} I &= \frac{\psi'}{2\pi}\left(\int_{s_0}^{\infty}\frac{1}{1+s^2}ds+\int_{-\infty}^{\infty}\frac{1}{1+s^2}ds+\int_{-\infty}^{s_0}\frac{1}{1+s^2}ds\right) \\ &= \frac{\psi'}{2\pi}\left[(\tan^{-1}(\infty)-\tan^{-1}(s_0))+(\tan^{-1}(\infty)-\tan^{-1}(-\infty))\right., \\ &\quad \left.+(\tan^{-1}(s_0)-\tan^{-1}(-\infty))\right], \\ &= \frac{\psi'}{2\pi}(\pi/2-\tan^{-1}(s_0))+(\pi/2-(-\pi/2))+(\tan^{-1}(s_0)-(-\pi/2)), \\ &= \psi'. \end{aligned}$$

And then looking at the second term:

$$I=\int_0^{2\pi}\frac{\cos(\beta+\psi)\sin(\beta+\psi)}{\lambda^2\cos^2(\beta+\psi)+\sin^2(\beta+\psi)}d\beta. \tag{173}$$

Let $u=\lambda^2\cos^2(\beta+\psi)+\sin^2(\beta+\psi)$ so

$$du=2\sin(\beta+\psi)\cos(\beta+\psi)-2\lambda^2\cos(\beta+\psi)\sin(\beta+\psi)d\beta. \tag{174}$$

Which means we can write

$$\begin{aligned} I &= \frac{1}{2-2\lambda}\int\frac{du}{u}, && (175)\\ &= \frac{1}{2-2\lambda}\left[ln(u)\right]_{\lambda^2}^{\lambda^2}, && (176)\\ &= 0. && (177) \end{aligned}$$

Putting these results together

$$\mathcal{L}'=\frac{1}{2\pi}\left(\psi'+\lambda^{-1}\mu^{-2}\kappa B_\rho\right). \tag{178}$$

All that remains now is to express twist in this new frame and subtract it from the above equation. From equation 125

$$\begin{aligned} 2\pi\mathcal{T}' &= \hat{\boldsymbol{V}}'\cdot\hat{\boldsymbol{W}}, \\ &= (\cos(\psi)\hat{\boldsymbol{f}}+\sin(\psi)\hat{\boldsymbol{g}})'\cdot(\cos(\psi)\hat{\boldsymbol{g}}-\sin(\psi)\hat{\boldsymbol{f}}), && (179)\\ &= \psi'+\hat{\boldsymbol{f}}'\cdot\hat{\boldsymbol{g}}, && (180)\\ &= \psi'+\hat{\boldsymbol{\rho}}\times\left(\frac{\hat{\boldsymbol{T}}}{\mu}\right)'\cdot\hat{\boldsymbol{T}}\times(\hat{\boldsymbol{\rho}}\times\hat{\boldsymbol{T}}/\mu), && (181)\\ &= \psi'+\frac{\lambda}{\mu^2}(\hat{\boldsymbol{\rho}}\cdot\hat{\boldsymbol{T}}\times\hat{\boldsymbol{T}}'), && (182)\\ &= \psi'+\mu^{-2}\kappa B_\rho. && (183) \end{aligned}$$

Also,

$$\begin{aligned} 2\pi\mathcal{W}' &= 2\pi<\mathcal{L}'-\mathcal{T}'>, && (184)\\ &= \left(\psi'+\mu^{-2}\lambda^{-1}\kappa B_\rho\right)-\left(\psi'+\mu^{-2}\kappa B_\rho\right), && (185)\\ &= \frac{\kappa B_\rho}{\mu^2}\frac{(1-\lambda)}{\lambda}, && (186)\\ &= \frac{\kappa B_\rho}{\lambda}\frac{1}{(1+\lambda)}. && (187) \end{aligned}$$

So we have proved the theorem for $0 < \lambda < 1$. If $\lambda = 1$ then the tangent vector is parallel to the $\hat{\boldsymbol{\rho}}$ axis, so the rate of winding about $\hat{\boldsymbol{\rho}}$, $\mathcal{L}'$, is the same as the rate winding about the tangent vector $\mathcal{T}'$, thus $\mathcal{W}' = 0$ (to see this note that B_ρ will be zero in this case). For consistency with standard writhe (equation 18), $\mathcal{W}'$ should not change sign if $s \to -s$. If this happens, $B_\rho \to -B_\rho$ so define $\mathcal{W}'$ here as

$$\mathcal{W}' = \frac{\kappa B_\rho}{2\pi\lambda} \frac{1}{(1 + |\lambda|)}. \tag{188}$$

2.11 Surface winding

The last section was concerned with how linking changes as the radial distance is varied. In this section we keep the radial distance constant and look at how linking changes in time due to motions. Let ρ_0 define the layer that is of interest.

In defining the problem we want to avoid the possibility of magnetic monopoles. There are two ways to go about this. Firstly, we could have two flux tubes cutting our layer, each having the same magnitude but oppositely directed radial fluxes. The net radial flux would then be zero, so $\nabla \cdot \boldsymbol{B} = 0$ as required. The problem with this approach is that it is then not possible to study each tube individually. Instead, we will introduce a net return flux as in figure 17.

For each flux tube i:

$$B_{i_\rho} = \begin{cases} \frac{\Phi_i}{A_1} - \frac{\Phi_i}{4\pi\rho_0^2}, & \textit{inside spot} \\ -\frac{\Phi_i}{4\pi\rho_0^2}. & \textit{outside spot} \end{cases} \tag{189}$$

So each tube's contribution is shared out over the surface area of the sphere.

So we have two flux tubes bounded between two spherical shells. Suppose that flux tube one is at the north pole and that the second tube is at co-latitude θ_0. Let them both have flux Φ. We wish to examine the change in helicity as the end point of one of the flux tubes moves around on the bottom shell. Let tube two move though an angle of ϕ_0 on the circle of co-latitude defined by θ_0 (see figure 18).

From equation 79 the change of helicity is equal to

$$\delta\mathcal{H} = \Phi^2 \left(2\delta\mathcal{L}_{12} + \delta\mathcal{T}_1 + \delta\mathcal{T}_2\right). \tag{190}$$

We will examine the linking term first. As tube two moves it will wind with tube one and also with the return flux. The amount of return flux it winds with will be

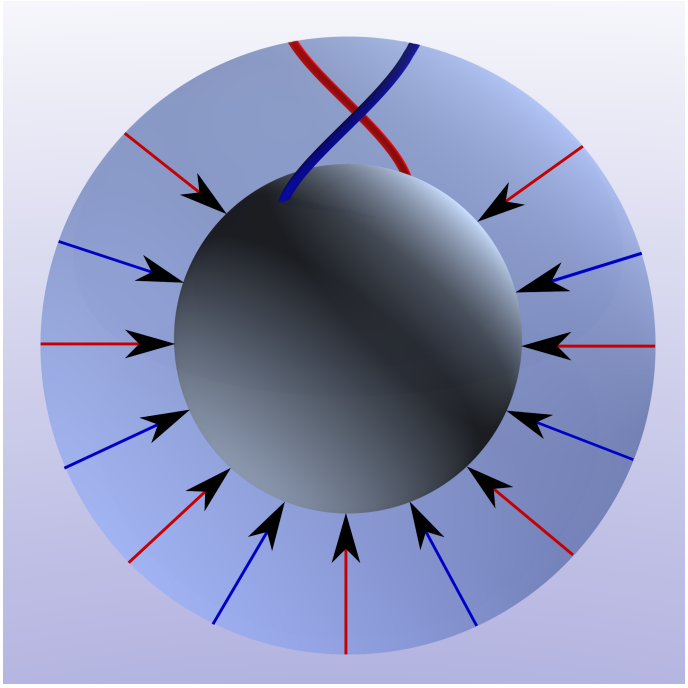

Figure 17: We introduce a net return flux to ensure that there are no magnetic monopoles. Each tube's B-field is shared out evenly and directed back into the surface.

proportional to the surface area of the polar cap that it circumscribes:

$$\delta\mathcal{L}_{12} = \frac{\phi_0}{2\pi}\left(1 - \frac{\text{area of polar cap at } \theta_0}{4\pi\rho_0^2}\right), \tag{191}$$

$$= \frac{\phi_0}{2\pi}\left(\frac{1}{2} + \frac{\cos(\theta_0)}{2}\right). \tag{192}$$

Next, we will need to work out the self-linking terms; these refer to the internal twist inside each flux tube. So we will need to calculate how much tube two twists as it moves through ϕ_0. To investigate this, we will move an arbitrary vector along the same path and measure how much it rotates. We will need to introduce some additional tools beforehand.

2.12 Parallel transport

In this section we will review several aspects of differential geometry that will be required in future calculations; for a general introduction see [54].

Define the 2-sphere as

$$\boldsymbol{S}(\theta,\phi) = \{\rho\sin(\theta)\cos(\phi), \rho\sin(\theta)\sin(\phi), \rho\cos(\theta)\}. \tag{193}$$

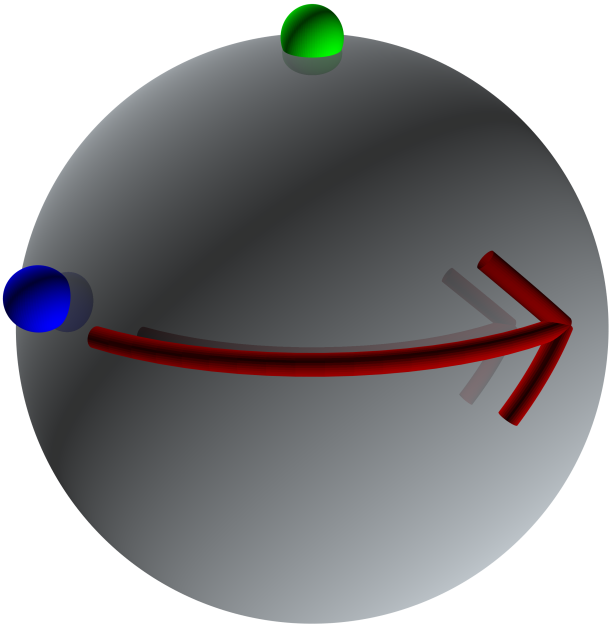

Figure 18: Tube two rotates around the circle defined by its co-latitude.

The metric will be

$$g_{ab} = \begin{pmatrix} \partial_\theta \boldsymbol{S} \cdot \partial_\theta \boldsymbol{S} & \partial_\theta \boldsymbol{S} \cdot \partial_\phi \boldsymbol{S} \\ \partial_\theta \boldsymbol{S} \cdot \partial_\phi \boldsymbol{S} & \partial_\phi \boldsymbol{S} \cdot \partial_\phi \boldsymbol{S} \end{pmatrix}, \tag{194}$$

$$= \begin{pmatrix} \rho^2 & 0 \\ 0 & \rho^2 \sin^2(\theta) \end{pmatrix}. \tag{195}$$

Let the arbitrary vector we will study be $\boldsymbol{v}$. It will be of the form

$$\boldsymbol{v}(\phi) = (\boldsymbol{v}^\rho(\phi), \boldsymbol{v}^\theta(\phi), \boldsymbol{v}^\phi(\phi)). \tag{196}$$

We will need to use differentiation to look at changes in the neighbourhood of $\boldsymbol{v}$. For example, one could look at the $\alpha = \rho$ components:

$$\frac{\partial \boldsymbol{v}}{\partial \rho} = \left(\frac{\partial \boldsymbol{v}^\rho}{\partial \rho} \hat{\boldsymbol{e}}_\rho + \frac{\partial \boldsymbol{v}^\theta}{\partial \rho} \hat{\boldsymbol{e}}_\theta + \frac{\partial \boldsymbol{v}^\phi}{\partial \rho} \hat{\boldsymbol{e}}_\phi \right) + \left(\boldsymbol{v}^\rho \frac{\partial \hat{\boldsymbol{e}}_\rho}{\partial \rho} + \boldsymbol{v}^\theta \frac{\partial \hat{\boldsymbol{e}}_\theta}{\partial \rho} + \boldsymbol{v}^\phi \frac{\partial \hat{\boldsymbol{e}}_\phi}{\partial \rho} \right).$$

The first three terms here describe how the vector itself has change, whilst the last three seek to express how the basis or coordinate grid changes.

In general we have

$$\frac{\partial \boldsymbol{v}}{\partial x^\alpha} = \frac{\partial \boldsymbol{v}^\beta}{\partial x^\alpha} \hat{\boldsymbol{e}}_\beta + \boldsymbol{v}^\beta \frac{\partial \hat{\boldsymbol{e}}_\beta}{\partial x^\alpha}. \tag{197}$$

The components of the last term in this equation describe how the basis vectors change, and are given a special name, the Christoffel symbols $\Gamma^\gamma_{\beta\alpha}$. Using these we can rewrite equation 197 as

$$\frac{\partial \boldsymbol{v}}{\partial x^\alpha} = \left(\frac{\partial \boldsymbol{v}^\beta}{\partial x^\alpha} + \boldsymbol{v}^\gamma \Gamma^\beta_{\gamma\alpha} \right) \hat{\boldsymbol{e}}_\beta. \tag{198}$$

Definition: **Covariant derivative**.

The covariant derivative of the vector $\boldsymbol{v}$ in the vector field $\boldsymbol{u}$ is defined as

$$\nabla_{\boldsymbol{u}}\boldsymbol{v} = u^i \left(\partial_i v^j + v^k \Gamma^j_{ki}\right). \tag{199}$$

This derivative describes how a vector varies as it is transported along a curve. We wish to transport $\boldsymbol{v}$ keeping it as close to parallel as possible. More precisely, we require that for each infinitesimal amount $\boldsymbol{v}$ is transported, it remains locally parallel.

Definition: **Parallel transport**.

Let $\boldsymbol{u}$ be the curve we wish to transport $\boldsymbol{v}$ along. Then we say that $\boldsymbol{v}$ is parallel transported along $\boldsymbol{u}$ if

$$\nabla_{\boldsymbol{u}}\boldsymbol{v} = 0. \tag{200}$$

Definition: **Geodesic**.

A geodesic is a curve whose tangent vector remains parallel when transported along itself. For a sphere, these are curves that follow a great circle. So when transporting $\boldsymbol{v}$, we would expect it to rotate – relative to the curve's tangent vector – for every angle θ_0 that gives a curve which does not lie on the equator.

Definition: **Inner product**.

The inner product of two vectors $\boldsymbol{a}$ & $\boldsymbol{b}$ is

$$< \boldsymbol{a}, \boldsymbol{b} >= g(\boldsymbol{a}, \boldsymbol{b}) = g_{ij} a^i b^j. \tag{201}$$

The angle α between these two vectors is

$$\cos(\alpha) = \frac{< \boldsymbol{a}, \boldsymbol{b} >}{||\boldsymbol{a}||\ ||\boldsymbol{b}||}\ ,\ ||\boldsymbol{a}|| = \sqrt{g(\boldsymbol{a}, \boldsymbol{a})}. \tag{202}$$

2.13 Twist calculation

We can now calculate how much $\boldsymbol{v}$ rotates. Note that the radial part of $\boldsymbol{v}$ will remain constant as the vector moves across the surface; we will ignore this component and treat $\boldsymbol{v}$ as a 2D vector. Expanding equation 200 gives

$$0 = u^i \left(\partial_i v^j + v^k \Gamma^j_{ki}\right). \tag{203}$$

where

$$\Gamma^i_{jk} = g^{im}\Gamma_{mjk}; \tag{204}$$

$$\Gamma_{mjk} = \frac{1}{2}(g_{mj,k} + g_{mk,j} - g_{jk,m}). \tag{205}$$

There are only three non-zero Christoffel symbols

$$\Gamma^{\phi}_{\phi\theta} = \Gamma^{\phi}_{\theta\phi} = g^{\phi\phi}\Gamma_{\phi\phi\theta} = cot(\theta); \tag{206}$$

$$\Gamma^{\theta}_{\phi\phi} = g^{\theta\theta}\Gamma_{\theta\phi\phi} = -\sin(\theta)\cos(\theta). \tag{207}$$

By using equation 203 we get

$$\partial_{\phi} v^{\theta} + v^{\phi}\Gamma^{\theta}_{\phi\phi} = 0; \tag{208}$$

$$\partial_{\phi} v^{\phi} + v^{\theta}\Gamma^{\phi}_{\theta\phi} = 0. \tag{209}$$

We are left with two partial differential equations to solve.

$$\frac{\partial v^{\theta}}{\partial \phi} = v^{\phi}\sin\theta_0\cos\theta_0; \tag{210}$$

$$\frac{\partial v^{\phi}}{\partial \phi} = -v^{\theta}\frac{\cos\theta_0}{\sin\theta_0}. \tag{211}$$

Differentiating the first equation, putting it into the second, and doing the same the other way around we obtain

$$\frac{\partial^2 v^{\theta}}{\partial \phi^2} = -v^{\theta}\cos^2(\theta_0); \tag{212}$$

$$\frac{\partial^2 v^{\phi}}{\partial \phi^2} = -v^{\phi}\cos^2(\theta_0). \tag{213}$$

Letting $\Omega = \cos(\theta_0)$, these equations have solutions of the form

$$v^{\theta}(\phi) = A\sin(\Omega\phi) + B\cos(\Omega\phi); \tag{214}$$

$$v^{\phi}(\phi) = C\sin(\Omega\phi) + D\cos(\Omega\phi). \tag{215}$$

Using the initial condition that $\boldsymbol{v}(\phi = 0) = (v^{\theta}_0, v^{\phi}_0)$ we find that

$$B = v^{\theta}_0, \quad D = v^{\phi}_0. \tag{216}$$

The other two coefficient may be found by differentiating equations (214-215)

$$\frac{\partial v^{\theta}}{\partial \phi} = \Omega A\cos(\Omega\phi) - \Omega v^{\theta}_0\sin(\Omega\phi), \tag{217}$$

$$\frac{\partial v^{\phi}}{\partial \phi} = \Omega C\cos(\Omega\phi) - \Omega v^{\phi}_0\sin(\Omega\phi), \tag{218}$$

and solving these equal to equations (210-211). This gives

$$v^{\theta} = v^{\phi}_0\sin(\theta_0)\sin(\Omega\phi) + v^{\theta}_0\cos(\Omega\phi), \tag{219}$$

$$v^{\phi} = -v^{\theta}_0\frac{\sin(\Omega\phi)}{\sin(\theta_0)} + v^{\phi}_0\cos(\Omega\phi). \tag{220}$$

We wish to compare this vector to its original form before it has been transported using equation 202. To do this both vectors must live in the same tangent space.

With this in mind, let $\phi_0 = 2\pi$ so that $\boldsymbol{v}$ traverses one complete circle of constant co-latitude, and both the transported vector and the original one are in the same tangent space. Now using equation 202 we can write

$$\begin{aligned} \cos(\alpha) &= \frac{g_{ab}v(0)^a v(2\pi)^b}{\sqrt{g_{ab}v(0)^a v(0)^b}\sqrt{g_{ab}v(2\pi)^a v(2\pi)^b}}, & (221)\\ &= \cos(2\pi\Omega) = \cos(2\pi\cos(\theta_0)). & (222)\\ \implies \alpha &= 2\pi\cos(\theta_0). & (223) \end{aligned}$$

Given that in the circle of constant co-latitude that $\boldsymbol{v}$ travelled, curvature will be uniform, this rotation can be evenly divided out over the whole circle. Thus it makes sense to divide by 2π and write

$$\alpha = \phi_0\cos(\theta_0). \tag{224}$$

So if $\boldsymbol{v}$ is moved through an angle ϕ_0, at co-latitude θ_0, then it will rotate by $\phi_0\cos(\theta_0)$. Recall that our aim here was to calculate how much tube two rotates as it moves as pictured in figure 18. If this tube is rotated positively, then the twist induced will be negative and vice-versa. This means that the change in twist after the motion will be

$$\delta\mathcal{T}_1 = 0 \ , \ \delta\mathcal{T}_2 = -\frac{\phi_0}{2\pi}\cos(\theta_0). \tag{225}$$

So now we have all the terms in equation 190 we can work out:

$$\begin{aligned} \delta\mathcal{H} &= \Phi^2\left(2\delta\mathcal{L}_{12} + \delta\mathcal{T}_1 + \delta\mathcal{T}_2\right), & (226)\\ &= \Phi^2\frac{\phi_0}{2\pi}\left[2\left(\frac{1}{2} + \frac{\cos(\theta_0)}{2}\right) + 0 + (-\cos(\theta_0))\right], & (227)\\ &= \Phi^2\frac{\phi_0}{2\pi}. & (228) \end{aligned}$$

This coincides with the change in helicity we would have if we were working in a flat geometry.

2.14 Planetary rotation

Next we will consider what would happen if both tubes moved around 2π. This could be thought of as one complete solid rotation of the sphere keeping the top of the tubes fixed. The tube at the north pole will rotate by 2π causing a negative twist of $\mathcal{T}_1 = -1$. The lower tube will also rotate about by $2\pi\cos(\theta_0)$, and so twist by $\mathcal{T}_2 = -2\pi\cos(\theta_0)$. The tubes will also wind with each other according to equation 192.

The change in helicity is

$$\begin{aligned}
\delta\mathcal{H} &= \Phi^2\left(\delta\mathcal{T}_1 + \delta\mathcal{T}_2 + 2\delta\mathcal{L}_{12}\right), && (229)\\
&= \Phi^2\Big[-1-\cos(\theta_0)+2\left(\frac{1}{2}+\frac{\cos(\theta_0)}{2}\right)\Big], && (230)\\
&= 0. && (231)
\end{aligned}$$

This is what is expected as, relative to each other, there is no winding going on.

2.15 Conclusion

In this section we have extended the measures, linking number, twist and writhe to a spherical geometry. By splitting curves at extrema we were able to formulate these quantities in terms of functions parameterised by radius. Through this process we explored several issues that do not arise in a flat geometry. One such point is the possibility to smoothly deform a ribbon in such a way that its winding number changes. However, we demonstrated that this does not affect the linking number which remains a valid measure.

In a similar way to Berger and Prior's [1] definition of open writhe in flat space, we proved that if the difference between linking number and twist was averaged over all secondary curves in a tube, then writhe is independent of the axis curve. This was important so as to be consistent with closed formulations of writhe.

Finally, we examined the implications of calculating helicity in this curved space. We considered the situation of flux tubes bound between two spherical shells of constant radius. Here we introduced a net return flux so as to avoid the problem of magnetic monopoles. By looking at examples, we showed that our concept of helicity is consistent with its flat counterpart.

In further work we hope to apply the measures detailed here to numerical data. In particular, it would be interesting to compare our open curve equations for writhe to other attempts to calculate writhe for open curves – one such is Fuller (1978) [14]. Fuller details how writhe can be calculated for open curves by extending the ends of a main curve with straight lines, and then connecting them at great distance from the main curve (this join's impact going to zero as its distance to the main curve goes to infinity). Writhe could be calculated using both methods and the results compared. In addition to this it would be interesting to see how these ideas could be extended to other geometries.

3 The Large Plasma Device

"How far is truth susceptible of embodiment – that is the question, that is the experiment." Friedrich Nietzsche, The Gay Science [55].

3.1 Introduction

We study several plasma experiments performed by Professor Walter Gekelman and his team at University of California, Los Angeles (UCLA). The device used to obtain data is the upgraded **La**rge **P**lasma **D**evice – LaPD from here on – shown in figure 19. Construction of the original device began in 1985 and was completed five

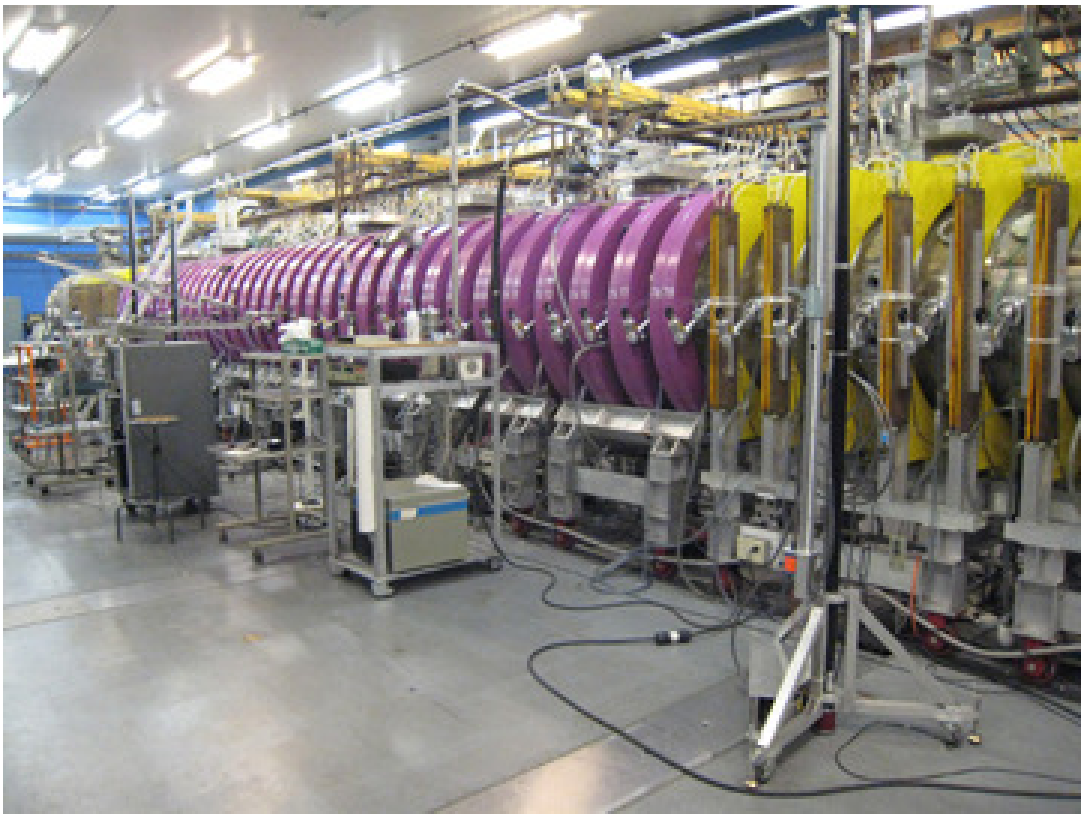

Figure 19: The Large Plasma Device at Basic Plasma Science Facility, UCLA, Los Angeles. Image reproduced from http://plasma.physics.ucla.edu.

years later, for details please see [56]. In 2001, work was completed that significantly improved the device's capabilities. In its current form, the LaPD is an 18m long cylindrical plasma machine capable of maintaining steady-state magnetic fields of up to 3.5kG. It was designed as a general purpose facility, and has advanced many areas of plasma physics including magnetic reconnection [7], waves [57], instabilities [58], turbulence and transport [59]. Around half the experimental time is offered at no cost to outside users.

Plasma is created through means of a cathode-anode pulsed discharge at one end of the cylinder (see figure 20). The barium-oxide coated cathode is heated to approximately 900°C to induce electron emission through thermionic means. A capacitor bank stores charge until it is released to produced the plasma. The experiments we study are stable enough so that they can be repeated and diagnosed at a different spatial location until a complete dataset can be obtained. Data is obtained using

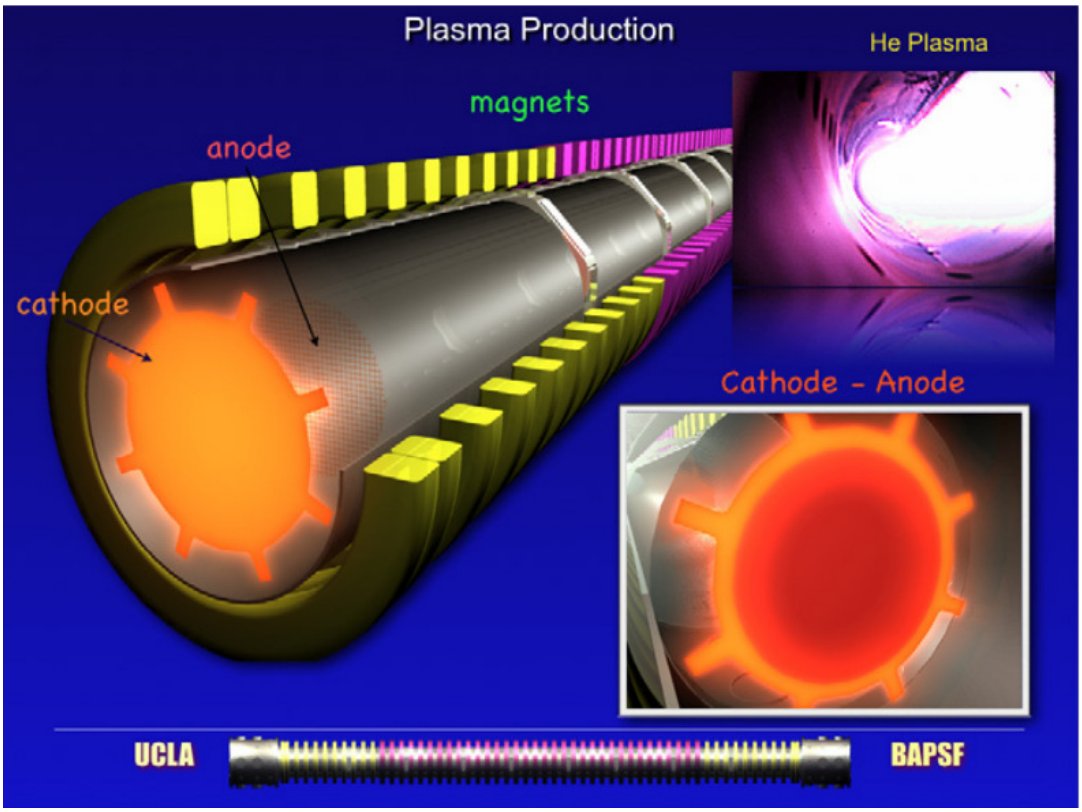

Figure 20: A schematic of the LAPD. The circular rings around the edge are electromagnets that are used to create an axial background field. Plasma is generated by a discharge of current from the cathode. Image obtained from [5].

probes via one of the device's 450 ports. Several different types of probes are used. Magnetic flux density is obtained by first measuring $d\boldsymbol{B}/dt$ using a $\dot{B}$-probe and then integrating. A $\dot{B}$-probe measures magnetic flux by exploiting Faraday's law (equation 30): a changing magnetic field induces an electric field. The scalar electric potential – voltage – induced across the probe can be measured. The component of the electric field is equal to minus the rate of change of the electric potential in that direction i.e.

$$\boldsymbol{E} = -\nabla V. \tag{232}$$

A Mach probe, depicted in figure 21, is an electric probe which is used to measure the velocity field of the plasma. The probe is aligned so as one side is facing upstream and the other downstream, and the current at either side obtained by use of Ampère's law (equation 31). The plasma velocity at the probe's position is proportional to

the ratio of these currents [60] i.e.

$$v \propto ln\left(\frac{I_{upstream}}{I_{downstream}}\right). \tag{233}$$

For more detail please see a recent review paper by Kyu-Sun Chung [61].

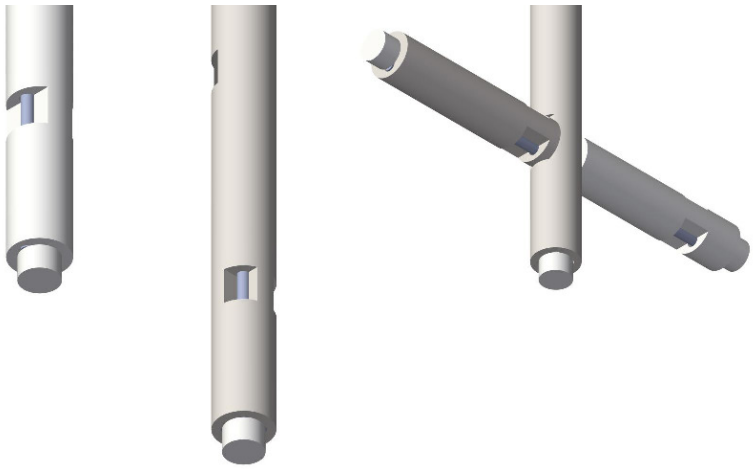

Figure 21: Several different Mach probes. As you move from left to right in the picture the probes gain the ability to pick up currents from more directions. Image from `http://plasma.physics.ucla.edu`.

3.2 The experiments

We have access to three separate plasma experiments which we will discuss in turn. They all involve three-dimensional flux ropes entwined along the length of the cylinder. A strong background guide-field, which is a couple of orders of magnitude stronger than the traverse field, is used to create flux ropes that travel from one end of the cylinder to the other. The team at UCLA is able to visualise the flux ropes and identify the quasi-separatrix – a region where the gradients of the mapping of magnetic field lines are much larger than unity [62] – as shown in figure 22.

Each flux rope is created from its own cathode emitter and experiences mutual Lorentz forces. The force on a point charge is given by

$$\boldsymbol{F} = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}). \tag{234}$$

However, instead of a single discrete charge we have a charge distribution that is in motion. Let $d\boldsymbol{F}$ and dq be the force on, and charge of, a small section of this distribution. We may write

$$d\boldsymbol{F} = dq(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}), \tag{235}$$

and then divide by a small volume element dV to give

$$\boldsymbol{f} = \rho(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}). \tag{236}$$

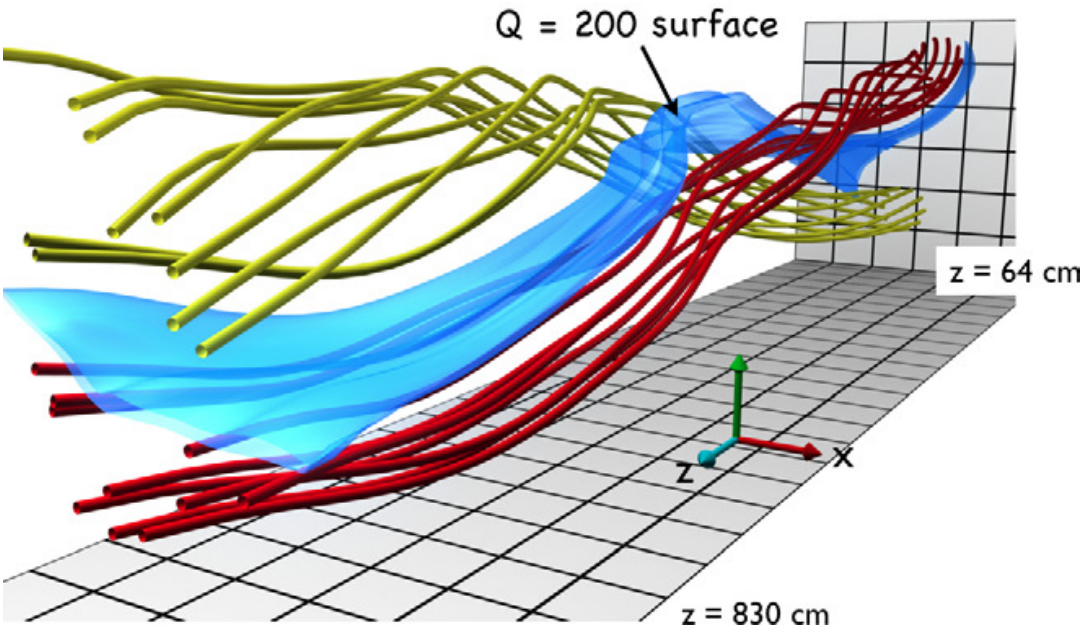

Figure 22: A visualisation of our first experiment. The surface in blue is the quasi-separatrix. Figure reproduced from [5].

Here $\boldsymbol{f}$ is the force density and ρ is the charge density. Noting that $\boldsymbol{J} = \rho \boldsymbol{v}$ this can be written as

$$\boldsymbol{f} = \rho \boldsymbol{E} + \boldsymbol{J} \times \boldsymbol{B}. \tag{237}$$

The flux ropes will exert mutual $\boldsymbol{J} \times \boldsymbol{B}$ forces, causing twisting and entanglement of the field lines as shown in figure 22.

For each experiment we wish to use winding numbers to calculate helicity as per equation 81. This method of calculating helicity has a number of advantages. Firstly, we can isolate any subset of the field lines and calculate helicity in a gauge-invariant way. This would allow one to calculate helicity even if the whole region had not been diagnosed. If we study the winding of field lines from the whole region, then the answer will be the same as if we used the relative helicity equation (equation 68). It is clear that before this is possible, field lines will need to be seeded across the volume. This will be discussed in the next section.

3.3 Interpolation

In 1885 the German mathematician Karl Weierstrass proved an important theorem which we now know as the Weierstrass approximation theorem. It states: every continuous curve on a closed interval can be approximated by a polynomial to within any desired tolerance. By any prescribed accuracy it is meant that for any curve, there is a polynomial of some degree which will approximate it to within any prescribed error. To see Weierstrass' proof (in German) see [63], or for a modern overview see [64]. Note that using higher order polynomials does not automatically improve the

accuracy of splines; in fact, high order splines should be avoided as they tend to oscillate too wildly creating problems known as Runge's phenomenon [65]. A lower order spline, defined over many smaller intervals, is usually advisable over a higher order spline, defined over fewer larger intervals [66].

Our data is typically arranged as a $[1001, 65, 65, 14, 3] = [t, x, y, z, v]$ array where $v = \{0, 1, 2\}$ corresponds to the $\{x, y, z\}$ component of the vector field. Typical dimensions are:

$x \in [-6, 10]$ (cm)	$\Delta x = 0.25$ (cm)
$y \in [-10, 6]$ (cm)	$\Delta y = 0.25$ (cm)
$z \in [127.8, 958.5]$ (cm)	$\Delta z = 63.9$ (cm)
$t \in [4.12, 4.76]$ (ms)	$\Delta t = 0.00064$ (ms)

To calculate helicity using winding numbers we will need to seed field and follow lines in the magnetic field. This means that we need to use an interpolator. We are presented with two problems:

1. How to keep $\nabla \cdot \boldsymbol{B} = 0$,

2. How to ensure that the interpolated values are continuous.

One way we can ensure that the first condition holds is by interpolating on the vector potential $\boldsymbol{A}$, and then taking the curl (as $\boldsymbol{B} = \nabla \times \boldsymbol{A}$) of the result. This is possible as we have been supplied with data for the vector potential. In 2006, Mackay et al. [67] investigated the impact of the loss of the first condition on numerical simulations. They found that breaking these conditions can lead to sizeable errors as seen in figure 23.

It turns out that the two issues mentioned above are actually related: $\nabla \cdot \boldsymbol{B} = 0$ can only be satisfied if

$$\frac{\partial^2 A_i}{\partial x_j \partial x_k} = \frac{\partial^2 A_i}{\partial x_k \partial x_j}, \tag{238}$$

which can only be guaranteed to be true if the interpolating function is continuous in its first and second derivatives: that is, it is $\mathcal{C}^2$.

There are several ways in which we could interpolate. To discuss these techniques it will be helpful to visualise our data as a three dimensional grid made up of lots of cubic cells, so that that each point has associated with it eight neighbouring vertices (see figure 24).

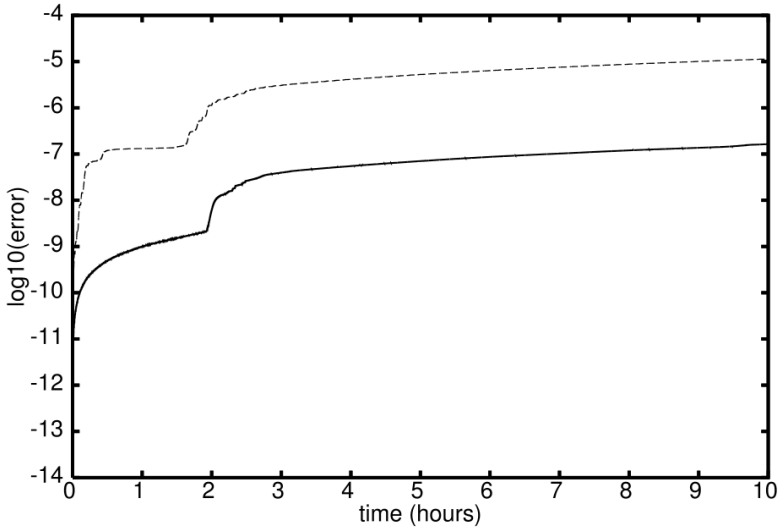

Figure 23: Comparison of errors from a paper by Mackay et al. [67]. The energy of a particle was calculated. A cubic spline interpolator (with $\nabla \cdot \boldsymbol{B} = 0$) is shown in the solid line, and a multi-linear interpolator ($\nabla \cdot \boldsymbol{B} \neq 0$) in the dotted one.

The most simple method would be to use the eight vertices of the cubic cell and linearly interpolate – this method is called trilinear interpolation. However, this method will not produce a result with a continuous derivative (recall that we require $\mathcal{C}^2$ from equation 238) so it is unsuitable.

Another method would be to use splines. A spline is a piece-wise defined curve that can be used to approximate a function to a required degree of smoothness. We will be interested in cubic splines as we require the result to be $\mathcal{C}^2$. For an interval i, define its cubic spline element as

$$f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3. \tag{239}$$

To find the coefficients we impose the continuity condition: that is, we specify that the polynomial's values, first and second derivatives match up at the boundary points between intervals. Consider the points x_i in figure 25. If we have n curve segments then we have $2n$ equations fixing the end points of the segments. We will gain another $2(n-1)$ equation from ensuring the first and second derivatives are continuous at the boundaries (the tangents match). This means in total we have $4n-2$ equation for a system with $4n$ unknowns. We need another two equations so that the system is well defined. One way to go about this is to use periodic boundary conditions. In this case you would require that $f'(x_0) = f'(x_n)$ and $f''(x_0) = f''(x_n)$. We use what is known as natural boundary conditions and set $f''(x_0) = f''(x_n) = 0$ as this offers greater control and makes more sense with respect to the experiment under study, as we are in Euclidean space rather than on a torus. For details of how a system like this is solved mathematically see [68] or [69], or for details of how to

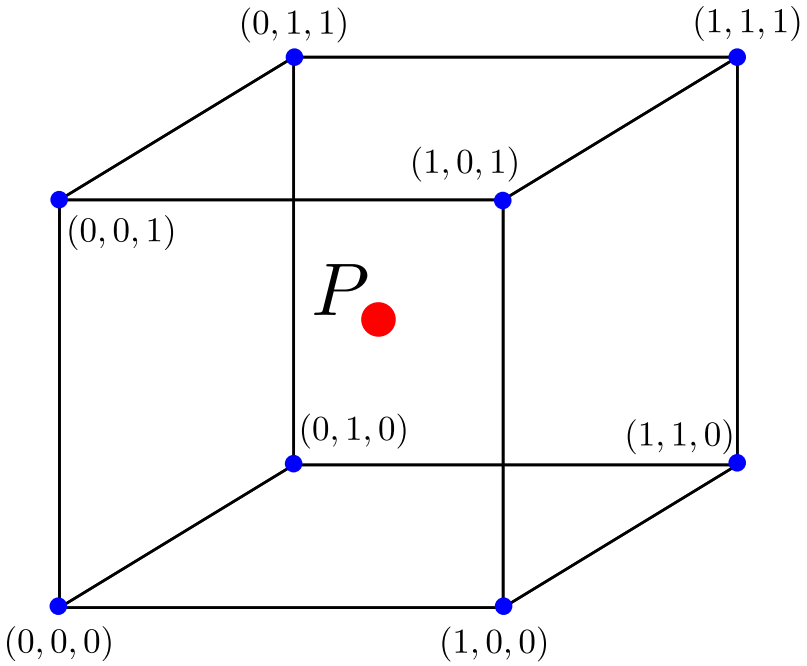

Figure 24: One method of interpolating is trilinear interpolation. Here the value at P is an average of all the neighbouring nodes.

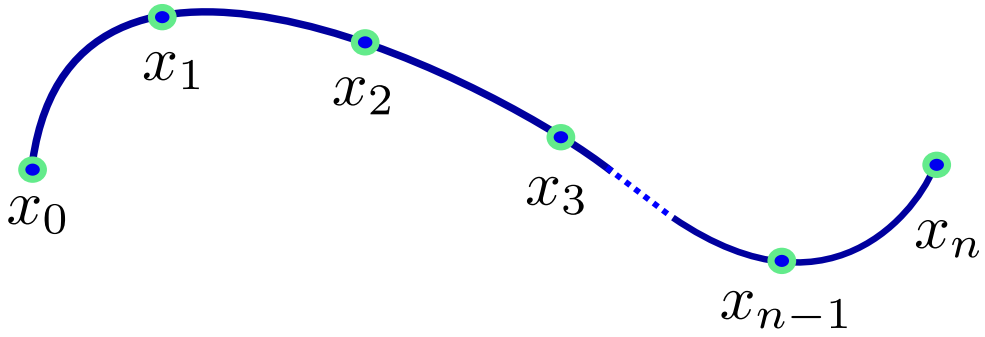

Figure 25: A cubic spline made up of n curve segments will have $4n$ unknown. The continuity condition only fixes $4n - 2$ of the coefficients. We use natural boundary condition to fix the remaining two.

implement this on a computer see [70].

Numerical Recipes [70] suggests that the 1D spline interpolation method can be extended to 2D by first fitting splines across all rows in our grid, and then using these splines to interpolate at the required point. These y-values can then be used as knots to fit a final vertical spline, allowing interpolation at a 2D grid-point. Notice that although the same horizontal splines are used every time and could be saved, each vertical spline will be unique to the look-up point.

In 3D the vertical knot values could be found by applying the 2D technique at each layer in height. Our dataset has dimensions $(x \times y \times z) = (65 \times 65 \times 14)$, so each 2D interpolation will require the fitting of $64 + 1$ splines. This needs to be

done at each z-height, so we have in total $(64+1)*14+1 = 911$ splines per single (x, y, z) look-up. This must be done for A_x, A_y and A_z. We wrote a program that does this, but after testing over a few times shots, it was found that the computing time was prohibitive. A faster solution was needed.

3.4 Cubic Hermite interpolation

The problem with using cubic splines is that each evaluation of the vector-field requires construction of fresh splines. This is because each call for the vector potential $\boldsymbol{A}$ will necessitate the costly process of constructing a vertical spline, as previously mentioned. Hermite interpolation offers a solution to our problem by creating an interpolator that is unique to each grid cell, rather than to each point. An interpolator for each 3D cell can be calculated and stored before it is needed in further calculations. As Hermite interpolation uses polynomials we are guaranteed $\mathcal{C}^\infty$ within the interval. The overall continuity will depend on the continuity of the derivatives at the boundaries.

A cubic Hermite spline is a polynomial defined by the values and derivative of a function at its end points. In one dimension it has the the form

$$f(x) = a_3x^3 + a_2x^2 + a_1x + a_0. \tag{240}$$

Let $[x_1, x_2]$ be an interval. The four coefficients can be found by solving the system of equations

$$\begin{aligned} f(x_0) &= a_3x_0^3 + a_2x_0^2 + a_1x_0 + a_0, & (241)\\ f(x_1) &= a_3x_1^3 + a_2x_1^2 + a_1x_1 + a_0, & (242)\\ f'(x_0) &= 3a_3x_0^2 + 2a_2x_0 + a_1, & (243)\\ f'(x_1) &= 3a_3x_1^2 + 2a_2x_1 + a_1. & (244) \end{aligned}$$

This method can be extended to higher dimensions. Bicubic interpolation uses a polynomial of the form

$$f(x, y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^iy^j. \tag{245}$$

To fix the sixteen coefficients a_{ij}, we need to use higher derivatives than before so that we have sixteen independent equations. Four equations can be obtained from the four corners, and then another eight can be found by taking the partial derivative of these four. The last four will require knowledge of the cross derivative f_{xy}.

3.5 Tricubic interpolation

As we will be working with three dimensional data we will need to use tricubic interpolation. We split the region into cells and define an interpolator piece-meal

for each 3D grid cell. We follow the method described by Lekien and Marsden [71]. For each 3D grid-cell we fit a polynomial of the form

$$f(x,y,z) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} a_{ijk}x^i y^j z^k, \tag{246}$$

where the $4^3 = 64$ coefficients need to be determined. The coefficients can be fixed by requiring the polynomials to be $\mathcal{C}^1$, that is, we require knowledge of the following at each node

$$\left\{ f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial x \partial z}, \frac{\partial^2 f}{\partial y \partial z}, \frac{\partial^3 f}{\partial x \partial y \partial z} \right\}. \tag{247}$$

For example consider the labelling of a grid cell as seen in figure 24. From the first few values of f we have

$$f(0,0,0) = a_{000}; \qquad f(0,0,1) = \sum_{k=0}^{3} a_{00k};$$
$$f(1,0,0) = \sum_{i=0}^{3} a_{i00}; \quad f(1,0,1) = \sum_{i=0}^{3}\sum_{k=0}^{3} a_{i0k}.$$

We end up with a huge system of linear equations. If we define

$$x \equiv [f(0,0,0)...f_{xyz}(1,1,1)]^T, \tag{248}$$
$$\alpha \equiv [a_{000}...a_{111}]^T. \tag{249}$$

Then the problem can be reduced to finding an inverse of A such that

$$A\alpha = x. \tag{250}$$

This inverse, $A^{-1}x$ then contains all of the coefficients a_{ijk}. By scaling, this matrix can then be used to fit a Hermite polynomial on a grid cell even when the cube isn't of unit dimensions. For example, if the cube has dimensions $(x_{dim}, y_{dim}, z_{dim})$, then the interpolation at (x, y, z) is

$$f = f(x/x_{dim}, y/y_{dim}, z/z_{dim}); \tag{251}$$

we are mapping our cube on to the unit cube. Note that the derivatives in equation 247 also require scaling.

As the matrix A^{-1} is so integral to the interpolator, Lekien and Marsden kindly provide a link to download the 64x64 matrix. Unfortunately the link is dead. However, the matrix and and some functions to use the interpolator can be found bundled as part of an image-processing library [72].

We now need to consider two things. Firstly, we mentioned that a requirement for an interpolation method was that it had to be $\mathcal{C}^2$. Secondly, how do we go about

calculating the derivatives in equation 247? Lekien and Marsden [71] suggest that we could use some type of finite-difference approximation.

Figure 26 details the continuity of the derivatives under such an interpolating scheme. Examining this, we note that all we require for $\mathcal{C}^2$ is to have continuity in three extra derivatives

$$\left\{ \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2}, \frac{\partial^2 f}{\partial z^2} \right\}. \tag{252}$$

It turns out that these three derivatives are not linearly independent from the lower derivatives in equation 247. For example, if we let p_1 be the point $(0, 0, 0)$ in figure 24, and p_2 be the point $(1, 0, 0)$, then:

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_{p_1} = 2a_{200} = 6f|_{p_2} - 6f|_{p_1} - 4 \left.\frac{\partial f}{\partial x}\right|_{p_1} - 2 \left.\frac{\partial f}{\partial x}\right|_{p_2}. \tag{253}$$

A sense of why this is can be found by considering what happens along one edge of the cubic cell. Along this edge, equation 246 reduces to a 1D cubic polynomial, which is already uniquely determined by its first derivatives from equations (241-244). This means that the derivatives in equation 252 are already prescribed by the function and its first derivatives.

A standard result from the study of functions (for example see [73]) is that the sum, or linear combination of continuous functions, is itself continuous. Hence, if we fix the derivatives in equation 247 in such a way as they are continuous, then the derivatives in equation 252 will also be continuous and our interpolator $\mathcal{C}^2$! Note that is it possible, by using high-order polynomials, to construct an interpolator that is $\mathcal{C}^2$ without placing a restriction on how the derivatives are obtained; for example see [74]. We will use cubic splines, which are $\mathcal{C}^2$, to find the derivatives. Figure 27 displays, as an example, the results of this method by plotting the derivative $\partial^2_{yz} A_x$ along side $\partial^2_{zy} A_x$.

	$x = 0, 1$	$y = 0, 1$	$z = 0, 1$	Global
f	y	y	y	y
$\frac{\partial f}{\partial x}$	y	y	y	y
$\frac{\partial f}{\partial y}$	y	y	y	y
$\frac{\partial f}{\partial z}$	y	y	y	y
$\frac{\partial^2 f}{\partial x^2}$	n	y	y	n
$\frac{\partial^2 f}{\partial y^2}$	y	n	y	n
$\frac{\partial^2 f}{\partial z^2}$	y	y	n	n
$\frac{\partial^2 f}{\partial x \partial y}$	y	y	y	y
$\frac{\partial^2 f}{\partial x \partial z}$	y	y	y	y
$\frac{\partial^2 f}{\partial y \partial z}$	y	y	y	y
$\frac{\partial^3 f}{\partial x^3}$	n	y	y	n
$\frac{\partial^3 f}{\partial y^3}$	y	n	y	n
$\frac{\partial^3 f}{\partial z^3}$	y	y	n	n
$\frac{\partial^3 f}{\partial x^2 \partial y}$	n	y	y	n
$\frac{\partial^3 f}{\partial x^2 \partial z}$	n	y	y	n
$\frac{\partial^3 f}{\partial x \partial y^2}$	y	n	y	n
$\frac{\partial^3 f}{\partial y^2 \partial z}$	y	n	y	n
$\frac{\partial^3 f}{\partial x \partial z^2}$	y	y	n	n
$\frac{\partial^3 f}{\partial y \partial z^2}$	y	y	n	n
$\frac{\partial^3 f}{\partial x \partial y \partial z}$	y	y	y	y

Figure 26: Continuity of the derivatives in this tricubic spline interpolator. For an explanation and proof see [71].

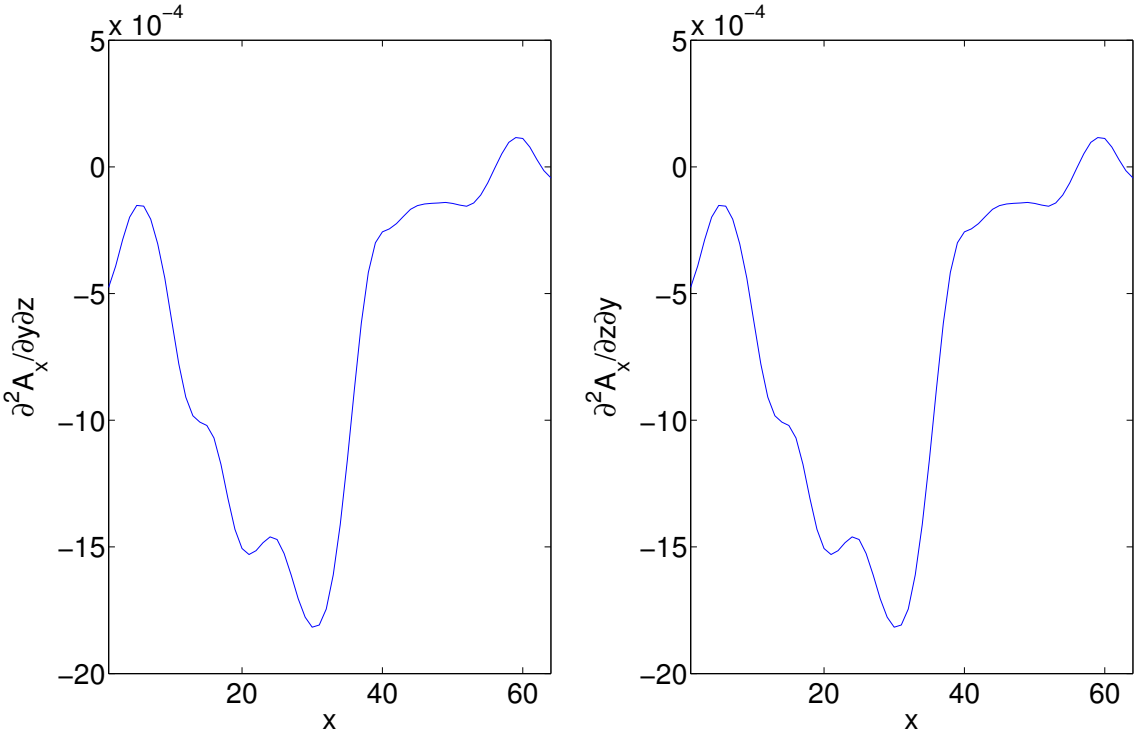

Figure 27: An example of the continuity this method ensures. $\partial^2_{yz}A_x$ is plotted along side $\partial^2_{zy}A_x$ at a particular instance in time. This result is from the third experiment.

3.6 Runge-Kutta

Now that we have found a way to interpolate $\boldsymbol{B}$ we need a method to integrate upwards in height to seed field lines. We can view this as an initial value problem. Let

$$\boldsymbol{x}(z) = \begin{pmatrix} x(z) \\ y(z) \end{pmatrix}, \tag{254}$$

be the path of field line originating from a point $\boldsymbol{y}_0$. Let $B_{i_n} = B_i(\boldsymbol{x}_n)$. Then

$$\dot{\boldsymbol{x}} = \frac{1}{|\boldsymbol{B}|} \begin{pmatrix} B_{x_n} \\ B_{y_n} \end{pmatrix}, \boldsymbol{x}(0) = \boldsymbol{y}_0. \tag{255}$$

There is an issue of scales that we need to consider. Recall the typical dimensions that were given in the preceding section:

$x \in [-6, 10]$ (cm)	$\Delta x = 0.25$ (cm)
$y \in [-10, 6]$ (cm)	$\Delta y = 0.25$ (cm)
$z \in [127.8, 958.5]$ (cm)	$\Delta z = 63.9$ (cm)
$t \in [4.12, 4.76]$ (ms)	$\Delta t = 0.00064$ (ms)

The grid in the X-Y plane is much finer (over sixty times) than the grid along the length of the cylinder in the Z plane. Consider a simple Euler stepping scheme:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{h}{|\boldsymbol{B}|} \begin{pmatrix} B_{x_n} \\ B_{y_n} \end{pmatrix}, \tag{256}$$

$$z_{n+1} = z_n + h\frac{B_{z_n}}{|\boldsymbol{B}|}. \tag{257}$$

Note that in the last term $B_{z_n}/|\boldsymbol{B}| \approx 1$ as the guide field along z is much stronger than the traverse field. We need h to be quite small so that equation 256 does not jump around too much; however, such a small choice of h will mean that equation 257 takes many steps to span the interval. To avoid this issue we define

$$\tilde{h} \equiv h\frac{B_{z_n}}{|\boldsymbol{B}|}. \tag{258}$$

This means that equations (256-257) become

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{\tilde{h}}{B_{z_n}} \begin{pmatrix} B_{x_n} \\ B_{y_n} \end{pmatrix}; \tag{259}$$

$$z_{n+1} = z_n + \tilde{h}. \tag{260}$$

Now a reasonably sized step in $\tilde{h}$ won't cause $\boldsymbol{x}_{n+1}$ to oscillate wildly. Euler's method is a first order approximation and as such is not a very accurate. A better choice

of integrator is a Runge-Kutta method. The classical Runge-Kutta method can be defined as:

$$\begin{aligned}
\boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{1}{B_z(\boldsymbol{x}_n)}\frac{h}{6}(\boldsymbol{k}_1 + \boldsymbol{k}_2 + \boldsymbol{k}_3 + \boldsymbol{k}_4); && (261)\\
z_{n+1} &= z_n + h; && (262)\\
\boldsymbol{k}_1 &= \begin{pmatrix} B_x(z_n, \boldsymbol{x}_n) \\ B_y(z_n, \boldsymbol{x}_n) \end{pmatrix}; && (263)\\
\boldsymbol{k}_2 &= \begin{pmatrix} B_x(z_n + \frac{1}{2}h, \boldsymbol{x}_n + \frac{h}{2}\boldsymbol{k}_1) \\ B_y(z_n + \frac{1}{2}h, \boldsymbol{x}_n + \frac{h}{2}\boldsymbol{k}_1) \end{pmatrix}; && (264)\\
\boldsymbol{k}_3 &= \begin{pmatrix} B_x(z_n + \frac{1}{2}h, \boldsymbol{x}_n + \frac{h}{2}\boldsymbol{k}_2) \\ B_y(z_n + \frac{1}{2}h, \boldsymbol{x}_n + \frac{h}{2}\boldsymbol{k}_2) \end{pmatrix}; && (265)\\
\boldsymbol{k}_4 &= \begin{pmatrix} B_x(z_n + h, \boldsymbol{x}_n + h\boldsymbol{k}_3) \\ B_y(z_n + h, \boldsymbol{x}_n + h\boldsymbol{k}_3) \end{pmatrix}. && (266)
\end{aligned}$$

The main difficulty with using Runge-Kutta methods is selecting an appropriate step-size: one too small and the calculation will be too expensive, one too large and the end result could be extremely inaccurate. Also, it should be noted that the ideal size of the step-size may not be constant throughout the domain of a problem; some areas may require a fine resolution to resolve, whereas, other more stable regions may be sped through more quickly with a larger step-size.

In 1969, E. Fehlberg [75] described one possible solution: the Runge-Kutta-Fehlberg method. Under this scheme, one additional calculation is performed which allows for a higher order error estimate. If this error estimate is too large the step-size is reduced, too small, then it is increased. Let h be the step-size. This method is accurate to $O(h^4)$ with an error estimate of $O(h^5)$, and so is often referred to as RK45. For thorough treatment of this method and others in the Runge-Kutta family see [76].

3.7 Resistivity

We will use our measurements of helicity to estimate a value for the diffusion term. From equation 81 we can write $dH/dt_{non-ideal}$ as

$$\frac{dH}{dt}_{non-ideal} = \frac{-1}{\pi}\sum_i\sum_j\left(\frac{dw_i}{dt}\Phi_i^2 + \frac{dw_j}{dt}\Phi_j^2 + 2\frac{d\theta_{ij}}{dt}\Phi_i\Phi_j\right), \qquad (267)$$

and recall that in equation 73 we had

$$\frac{dH}{dt}_{ideal} = 2\int_S((\boldsymbol{A}_p\cdot\boldsymbol{v})\boldsymbol{B} - (\boldsymbol{A}_p\cdot\boldsymbol{B})\boldsymbol{v})\cdot d\boldsymbol{S}. \qquad (268)$$

Note that here the background field $\boldsymbol{A}_p$ is equal to the guide field, and also that $\boldsymbol{B}$ is the total magnetic flux density including this guide field.

As Berger [34] demonstrates, the non-ideal helicity dissipation due to resistivity may be written as

$$\frac{dH}{dt}_{dissipation} = -2\int \boldsymbol{E}\cdot\boldsymbol{B}d^3x. \tag{269}$$

The last three equations can be put together to give

$$\frac{dH}{dt}_{non-ideal} = \frac{dH}{dt}_{ideal} + \frac{dH}{dt}_{dissipation}. \tag{270}$$

Or rearranging we have

$$\frac{dH}{dt}_{non-ideal} - \frac{dH}{dt}_{ideal} = -2\int \boldsymbol{E}\cdot\boldsymbol{B}d^3x. \tag{271}$$

Now we must re-examine the terms in Ohm's law (equation 34) with the aim of making a substitution for $\boldsymbol{E}$. We do not have data to work out the pressure term so it will be neglected. The electron inertial term will only be significant on collision time-scales so this is also neglected. This gives

$$\frac{dH}{dt}_{ideal} - \frac{dH}{dt}_{non-ideal} = 2\int \eta\boldsymbol{J}\cdot\boldsymbol{B}d^3x. \tag{272}$$

We define an effective resistivity η_{eff} by making the assumption that the currents move parallel to the magnetic field:

$$\eta_{H_{eff}} \equiv \frac{\frac{dH}{dt}_{ideal} - \frac{dH}{dt}_{non-ideal}}{2\int \boldsymbol{J}\cdot\boldsymbol{B}d^3\boldsymbol{x}}. \tag{273}$$

We need to calculate the current density $\boldsymbol{J}$. Ampère's law states

$$\nabla\times\boldsymbol{B} = \left(\mu_0\boldsymbol{J} + \mu_0\epsilon_0\frac{\partial\boldsymbol{E}}{\partial t}\right), \tag{274}$$

where μ_0 is the permeability of free space, that is

$$\mu_0 = 4\pi\times10^{-7}VsA^{-1}m^{-1}, \tag{275}$$

and ϵ_0 is the vacuum permittivity

$$\epsilon_0 = \frac{1}{\mu_0 c^2}. \tag{276}$$

Recall that Faraday's law of induction is

$$\nabla\times\boldsymbol{E} = -\frac{\partial\boldsymbol{B}}{\partial t}. \tag{277}$$

Let l_0 and t_0 be typical values for length and time respectively. Using Faraday's law of induction (equation 277), and letting E_0 and B_0 be typical values of the electric and magnetic fields we may make the approximation

$$\frac{E_0}{l_0} \approx \frac{B_0}{t_0}. \tag{278}$$

Applying this approximation to the last term in equation 274 gives

$$\mu_0\epsilon_0\frac{E_0}{t_0} \approx \frac{B_0 l_0}{c^2 t_0^2} = \frac{v_0^2 B_0}{c^2 l_0} \approx \frac{v_0^2}{c^2}|\nabla \times \boldsymbol{B}|. \tag{279}$$

where v_0 is a typical plasma velocity. A typical value for our data is $2 \times 10^3 ms^{-1}$. This makes

$$\mu_0\epsilon_0\frac{\partial \boldsymbol{E}}{\partial t} \approx \frac{v_0^2}{c^2}|\nabla \times \boldsymbol{B}| \approx 10^{-27}. \tag{280}$$

This term is significantly smaller than the first term of the right-hand side in equation 274 and so will be neglected; it is only important on collision time scales. Thus we can use equation 274 to calculate $\boldsymbol{J}$ ignoring the second term of the right.

Now for the $\partial \boldsymbol{J}/\partial t$ term in equation 34:

$$\frac{m_e}{ne^2}\frac{\partial \boldsymbol{J}}{\partial t} = \frac{m_e}{ne^2\mu_0}\left(\frac{\partial(\nabla \times \boldsymbol{B})}{\partial t}\right) \approx \frac{m_e}{ne^2}\frac{B_0}{t_0\mu_0 l_0} = \frac{m_e\epsilon_0 c^2 B_0}{ne^2 t_0 l_0}. \tag{281}$$

Typical values here are

$$B_0 = 10^2 G, t_0 = 10^{-4} s, l_0 = 10^{-1} m, n = 2 \times 10^{12} cm^{-3}. \tag{282}$$

This gives

$$\frac{m_e}{ne^2}\frac{\partial \boldsymbol{J}}{\partial t} \approx \frac{(9.1 \times 10^{-31})(8.9 \times 10^{-12})(3 \times 10^8)^2(10^{-6})}{(2 \times 10^{18})(1.6 \times 10^{-19})^2(10^{-4})(10^{-1})} \approx 10^{-6}. \tag{283}$$

This term is significantly smaller than the first term of the right-hand side in equation 34 and so will be neglected; it is only important on collision time scales. The only term we need to calculate here is the $\boldsymbol{J} \cdot \boldsymbol{B}$ term on the right hand side of equation 273.

3.8 Quadrature

The word quadrature originates from the Latin word "quadratura" which means the process of squaring [77]. Historically, the term refers to the calculation of area by any method. We use it here to denote some numerical method that seeks to find the *approximate* area under a curve. The most basic of such schemes is known as the trapezium rule. This involves using trapezia – the area of which are easily found – to approximate the region. Figure 28 shows one particular section of the region where a trapezium has been fitted; note that for functions that have a lot of curvature this method could be quite inaccurate.

Our first approach, to a later problem in this project, was to use the trapezium rule. The trapezium rule or any numerical integration scheme will struggle when the integrand changes quickly. Figure 29 shows our efforts. A more accurate method with some error control is needed for our purposes. For a detailed study of the

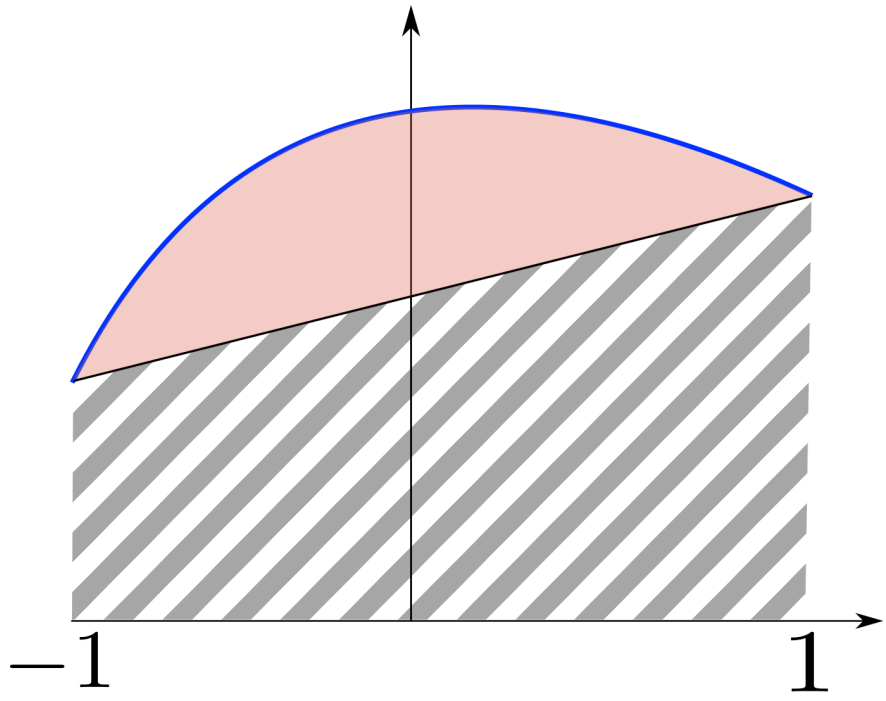

Figure 28: The trapezium rule will provide an under-estimation if the function is concave.

quadrature of oscillating functions see [78].

A more accurate method comes from a careful study of figure 28. What would happen if we increased the height of the trapezium? The top would no longer meet the curve segment at its end-points and, more importantly, we would gain some extra area that might cancel out the pink area we missed in our approximation. Figure 30 suggests this is possible. It turns out that for polynomial curves this method yields an exact answer. Estimate the integral by writing

$$\int_{-1}^{1} f(x)dx \approx \sum_{=1}^{n} w_i f(x_i). \tag{284}$$

We need to solve this to determine the intersection points, x_i, and their corresponding weights w_i. We will use $n = 2$ points. Then we have

$$\int_{-1}^{1} f(x)dx \approx w_1 f(x_1) + w_2 f(x_2). \tag{285}$$

Letting $f = 1, x, x^2, x^3$ successively leads to four equations with four unknowns:

$$w_1 + w_2 = 2; \tag{286}$$

$$w_1 x_1 + w_2 x_2 = 0; \tag{287}$$

$$w_1 x_1^2 + w_2 x_2^2 = 2/3; \tag{288}$$

$$w_1 x_1^3 + w_2 x_2^3 = 0. \tag{289}$$

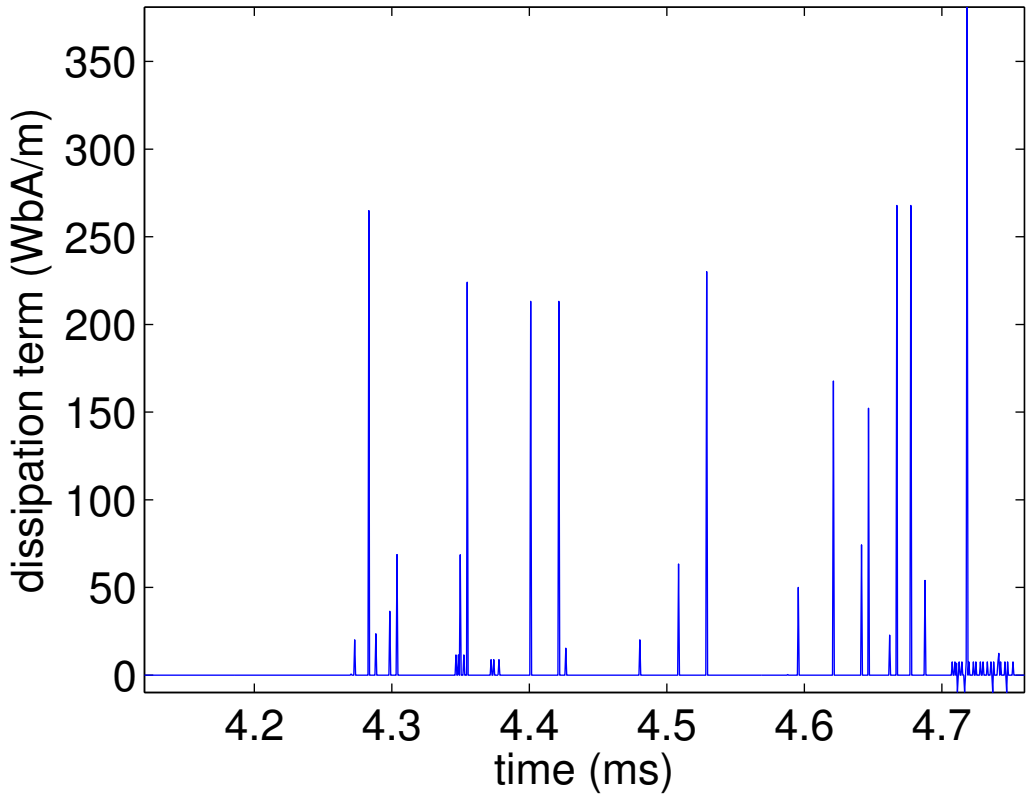

Figure 29: The diffusion term (equation 272) calculated by using a trapezium rule. A more accurate method is needed.

Solving this system gives

$$\int_{-1}^{1} f(x)dx \approx f(-1/\sqrt{3}) + f(1/\sqrt{3}). \tag{290}$$

As we have used a linear combination of polynomials up to degree three, this method will be exact if the curve is a polynomial of degree less than three. In general, the result will be exact if its order is less than or equal to $2n-1$ [64]. So, if our function was a quartic polynomial we would need to use $n = 3$ and solve a system of five equations containing terms up to x^5. There is a better way to do this by considering orthogonal polynomials.

Definition: **Orthogonal polynomials**.

A polynomial, $p_n(x)$, is an orthogonal polynomial with respect to a weight function $w(x)$ if

$$\int_a^b w(x)x^k p_n(x)dx = 0, \text{ for } k = 0, \ldots, n-1. \tag{291}$$

The Legendre polynomials can be defined recursively as:

$$\begin{aligned} P_0(x) &= 1; & P_1(x) &= x; \\ (n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x). \end{aligned} \tag{292}$$

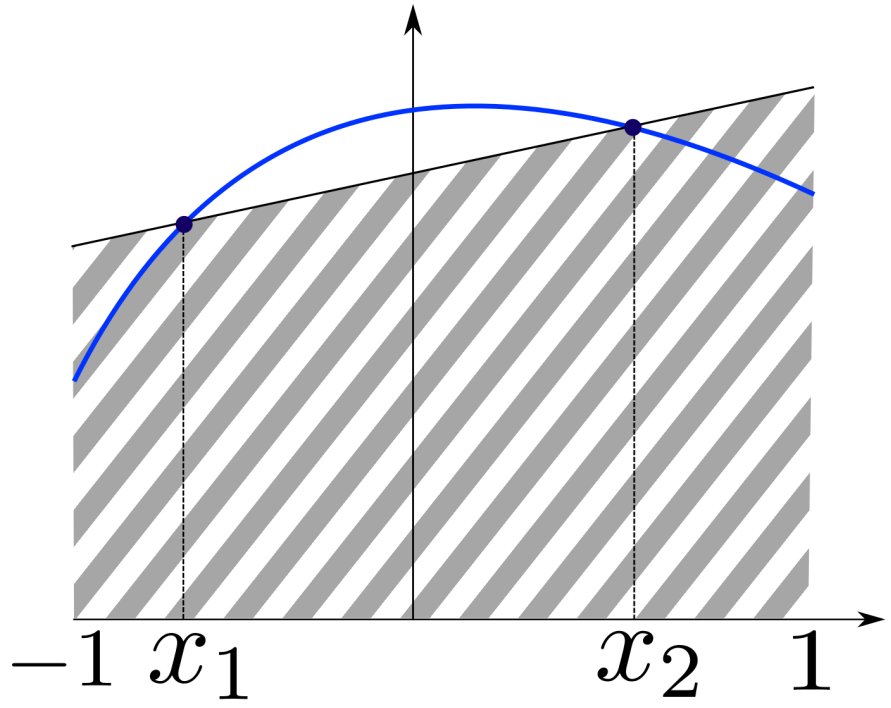

Figure 30: The intersection points (x_1, x_2) have been chosen so that the extra area cancels out the missed area. This is Gaussian quadrature.

In the case of Legendre polynomials $w(x) = 1$. It turns out that it is not a coincidence that the roots of the third polynomial,

$$P_2 = \frac{1}{2}(3x^2 - 1), \tag{293}$$

are $\pm 1/\sqrt{3}$. The roots of the Legendre polynomials determine the x_i points (see [79] for details). The weights are given by the following formula [80]:

$$w_i = \frac{2}{(1 - x_i^2)P_n'(x_i)^2}. \tag{294}$$

Legendre polynomials are not the only polynomials that can be used here; many other orthogonal polynomials can be used instead, for example, Jacobi polynomials, or Hermite polynomials.

It would be nice to be able to obtain an error estimate for our integrator. One common method for doing this is to carry out the calculation twice, the second time at a higher precision, and then take the difference as an error estimate. The trouble with this is that none of the Legendre polynomials share roots, except maybe at the midpoint of the interval. This would mean that it would be impossible to embed the lower order rule inside the higher one, so as to cut down on the number of evaluations (for an example of this see the Runge-Kutta-Fehlberg method used in subsection 3.6).

In 1965, the Russian Aleksandr Semenovish Kronrod proposed an elegant solution to this problem [81]. Let

$$G_n = \sum_{i=1}^{n} w_i f(x_i), \tag{295}$$

be the standard formula for Gaussian quadrature. Konrad suggested a new formula:

$$K_{2n+1} = \sum_{i=1}^{n} a_i f(x_i) + \sum_{j=1}^{n+1} b_j f(y_j). \tag{296}$$

The clever thing about this formula is that it uses all n points evaluated in the Gauss formula. The extra terms require $n+1$ additional points, so this means that K_{2n+1} is a $2n+1$ point rule. G_n can now be worked out at no additional cost, and the difference between it and K_{2n+1} used as an error estimator.

We now need some way of choosing the extra points y_j. In 1905 the Dutch mathematician Thomas Stieltjes defined a series of polynomials that offered a solution. Stieltjes polynomials, E_{n+1}, are polynomials that satisfy

$$\int_a^b P_n(x) E_{n+1}(x) x^k w(x) dx = 0, \quad k = 1, \ldots, n, \tag{297}$$

where $P_n(x)$ is a system of orthogonal polynomials with a corresponding weight function $w(x)$. Stieltjes conjectured that if $P_n(x)$ are the Legendre polynomials, then the zeros of E_{n+1} will be in the interval $[-1, 1]$ and they will alternate with the zero of P_n [6]. In 1934, a Hungarian mathematician, Gábor Szegö, was able to prove these conjectures [82]. For example see figure 31 which shows one of the most popular schemes, (G7,K15), which uses seven Gauss and fifteen Kronrod points.

The method described above forms what is known as Gauss-Konrad quadrature. This is the standard numerical integration scheme used in many numerical integration libraries, including QUADPACK [83], GSL [49], NAG [84] and R [85]. For our purposes, and in general, we need to be able to use this method on any domain, not just $[-1, 1]$; one may use the following rule [86] in such cases:

$$\int_a^b f(x) dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2} z + \frac{a+b}{2}\right) dz. \tag{298}$$

We use an existing software package *Cubature* written by Professor Steven Johnson of Massachusetts Institute of Technology. The Cubature library offers high performance, adaptive, multi-dimensional integration over hypercubes. That is, it computes integrals of the form

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} \ldots \int_{a_n}^{b_n} \boldsymbol{f}(\boldsymbol{x}) d^n \boldsymbol{x}. \tag{299}$$

Seven-Point Gaussian Nodes	Weights
±0.94910 79123 42758	0.12948 49661 68870
±0.74153 11855 99394	0.27970 53914 89277
±0.40584 51513 77397	0.38183 00505 05119
0.00000 00000 00000	0.41795 91836 73469

Fifteen-Point Kronrod Nodes	Weights
±0.99145 53711 20813	0.02293 53220 10529
±0.94910 79123 42759	0.06309 20926 29979
±0.86486 44233 59769	0.10479 00103 22250
±0.74153 11855 99394	0.14065 32597 15525
±0.58608 72354 67691	0.16900 47266 39267
±0.40584 51513 77397	0.19035 05780 64785
±0.20778 49550 07898	0.20443 29400 75298
0.00000 00000 00000	0.20948 21410 84728

Figure 31: One of the most popular numerical integration schemes (G7,K15). Reproduced from [6].

The algorithm works by recursively partitioning the region into smaller subdomains until convergence is achieved. If the dimension of a system is large, then it might be more efficient to consider alternate schemes such as Monte Carlo methods. The algorithm used in this library is described in papers by A. C. Genz and A. A. Malik [87], and also later by J. Berntsen, T. O. Espelid, and A. Genz [88].

The results of this integrator are displayed later in figures (45-46). The dissipation oscillates in a similar manner to the helicity. It is worth noting that the largest change in helicity (figure 43) occurs at $t = 4.2$ and corresponds to the largest jump in the dissipation term (figure 45).

3.9 Energy dissipation

We can also calculate the dissipation term by using an energy conservation equation. The flux of electromagnetic energy is given by the Poynting vector $\boldsymbol{S}$

$$\boldsymbol{S} = \frac{1}{\mu_0}\boldsymbol{E} \times \boldsymbol{B}. \tag{300}$$

Taking the divergence and using a vector identity, we can write

$$\nabla \cdot (\boldsymbol{E} \times \boldsymbol{B}/\mu_0) = ((\nabla \times \boldsymbol{E}) \cdot \boldsymbol{B} - \boldsymbol{E} \cdot (\nabla \times \boldsymbol{B}))/\mu_0. \tag{301}$$

Ampère's and Faraday's laws are

$$\nabla \times \boldsymbol{B} = \mu_0 \left(\boldsymbol{J} + \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \right), \tag{302}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}. \tag{303}$$

This means we can replace the curl terms in equation 301

$$-\nabla \cdot \boldsymbol{E} \times \boldsymbol{B}/\mu_0 = \frac{1}{\mu_0} \boldsymbol{B} \cdot \frac{\partial \boldsymbol{B}}{\partial t} + \boldsymbol{E} \cdot \left(\boldsymbol{J} + \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \right), \tag{304}$$

$$= \frac{1}{\mu_0} \frac{\partial}{\partial t} \left(\frac{1}{2} B^2 \right) + \epsilon_0 \frac{\partial}{\partial t} \left(\frac{1}{2} E^2 \right) + \boldsymbol{E} \cdot \boldsymbol{J}. \tag{305}$$

Rearranging gives

$$\frac{\partial}{\partial t} \left(\frac{1}{2\mu_0} B^2 + \frac{\epsilon_0}{2} E^2 \right) = -\nabla \cdot \boldsymbol{E} \times \boldsymbol{B}/\mu_0 - \boldsymbol{E} \cdot \boldsymbol{J}. \tag{306}$$

The terms on the left are the magnetic energy and the electric energy. On the right we have the electromagnetic energy flux, and a term measuring the amount of power being transferred into the electromagnetic field. This is an energy continuity equation: it states that the rate of change of the amount of energy in a region is equal to the minus of the amount of energy flowing out, minus the the rate at which the field is doing work.

If we integrate both sides of equation 306 over the whole region and apply the divergence theorem to the $\boldsymbol{E} \times \boldsymbol{B}$ term we obtain:

$$\frac{1}{2\mu_0} \int \frac{\partial (B^2)}{\partial t} d^3\boldsymbol{x} + \frac{\epsilon_0}{2} \int \frac{\partial (E^2)}{\partial t} d^3\boldsymbol{x} + \frac{1}{\mu_0} \int \boldsymbol{E} \times \boldsymbol{B} \cdot d\boldsymbol{S} = -\int \boldsymbol{E} \cdot \boldsymbol{J} d^3\boldsymbol{x}. \tag{307}$$

The third term should be taken over the boundaries of the region. Again we will neglect the electric field term. As our integration domain is time-independent, we can write

$$\int \frac{\partial (B^2)}{\partial t} d^3\boldsymbol{x} = \frac{d}{dt} \int B^2 d^3\boldsymbol{x}. \tag{308}$$

This result follows from the continuity of $\boldsymbol{B}$. So we now have

$$\frac{1}{2\mu_0} \frac{d}{dt} \int B^2 d^3\boldsymbol{x} + \frac{1}{\mu_0} \int \boldsymbol{E} \times \boldsymbol{B} \cdot d\boldsymbol{S} = -\int \boldsymbol{E} \cdot \boldsymbol{J} d^3\boldsymbol{x}. \tag{309}$$

Integrating this equation between $t = 0$ and some later time $t = T$ leads to

$$\frac{1}{2\mu_0} \left(\int B^2|_{t=0} \; d^3\boldsymbol{x} - \int B^2|_{t=T} \; d^3\boldsymbol{x} \right) = \frac{1}{\mu_0} \iint \boldsymbol{E} \times \boldsymbol{B} \cdot d\boldsymbol{S} dt + \iint \boldsymbol{E} \cdot \boldsymbol{J} d^3\boldsymbol{x} dt. \tag{310}$$

This equation says that the total energy change is equal to the negative of the sum of the Poynting flux across the boundaries and the power transferred by the magnetic

field. Using Ohm's law (equation 34) and Ampère's law (equation 274), we can write the Poynting term as

$$\begin{aligned}
\frac{1}{\mu_0}\iint \boldsymbol{E}\times\boldsymbol{B}\cdot d\boldsymbol{S}dt &= \frac{1}{\mu_0}\iint(\eta\boldsymbol{J}+\boldsymbol{B}\times\boldsymbol{v}+\frac{1}{ne}\boldsymbol{J}\times\boldsymbol{B})\times\boldsymbol{B}\cdot d\boldsymbol{S}dt, \quad (311)\\
&= \frac{1}{\mu_0^2}\iint\Big(\eta\nabla\times\boldsymbol{B}+\mu_0\boldsymbol{B}\times\boldsymbol{v}+... \quad (312)\\
&\qquad +\frac{1}{ne}(\nabla\times\boldsymbol{B})\times\boldsymbol{B}\Big)\times\boldsymbol{B}\cdot d\boldsymbol{S}dt.
\end{aligned}$$

Recall that n is the free electron density per volume and e is the amount of charge that one electron carries. Now we need to examine the last term – the total power transferred into the region:

$$\begin{aligned}
\iint \boldsymbol{E}\cdot\boldsymbol{J}d^3\boldsymbol{x}dt &= \iint \eta J^2+(\boldsymbol{B}\times\boldsymbol{v})\cdot\boldsymbol{J}\ d^3\boldsymbol{x}dt, \quad (313)\\
&= \frac{1}{\mu_0^2}\iint\eta(\nabla\times\boldsymbol{B})^2+\mu_0(\boldsymbol{B}\times\boldsymbol{v})\cdot(\nabla\times\boldsymbol{B})\ d^3\boldsymbol{x}dt. \quad (314)
\end{aligned}$$

Using this equation and the equation for the Poynting term above, we wish to rearrange equation 310 so that we can define an effective energy dissipation η_{eff}. We have

$$\begin{aligned}
\eta_{E_{eff}}(T)\equiv\ & \frac{1}{\iint(\nabla\times\boldsymbol{B})\times\boldsymbol{B}\cdot d\boldsymbol{S}dt+\iint(\nabla\times\boldsymbol{B})^2\ d^3\boldsymbol{x}dt} \quad (315)\\
*\ & \Big(\frac{\mu_0}{2}\int B^2|_{t=0}-B^2|_{t=T}\ d^3\boldsymbol{x}+\mu_0\iint(\boldsymbol{v}\times\boldsymbol{B})\cdot(\nabla\times\boldsymbol{B})\,d^3\boldsymbol{x}dt\\
&+\iint\Big[\frac{1}{ne}\boldsymbol{B}\times(\nabla\times\boldsymbol{B})+\mu_0(\boldsymbol{v}\times\boldsymbol{B})\Big]\times\boldsymbol{B}\cdot d\boldsymbol{S}dt\Big),
\end{aligned}$$

where the time integrals are from $t=t_{start}$ to $t=T$. As before with the $\eta_{H_{eff}}$ we are assuming that the currents are parallel to the electric field.

3.10 Winding numbers

Berger and Prior [1] show that, for flux tubes that always travel upwards, we have

$$L_k=\sum_i\sum_j w_{ij}, \quad (316)$$

where w_{ij} is the winding number – see equation 27. Combing this with equation 79 gives

$$H=\sum_i\sum_{j\neq i}\Phi_i\Phi_j w_{ij}+\sum_i\Phi_i^2 w_{ii}. \quad (317)$$

The first term represent the mutual helicity between flux tubes, whilst the second represent self helicity – this arises from winding of field lines inside each flux tube. Consider the self helicity contribution of one individual tube. That tube will contain

many smaller flux tubes, each with a flux of $\delta\Phi$. That tube's contribution to the second term in equation 317 is

$$H = \sum_i \sum_{j \neq i} (\delta\Phi)^2 w_{ij} + H_{self}. \tag{318}$$

It is clear that the first term here will be significantly smaller than the first term in equation 317. H_{self} will be smaller again. Thus, if we fill our region with a large enough number of flux tubes then the self helicities will go to zero.

There are two equivalent ways to define winding number mentioned in the introduction. Consider two curves, say a and b, and let $\boldsymbol{r}_{ab}(z) = \boldsymbol{r}_b(z) - \boldsymbol{r}_a(z)$ be the relative position vector pointing from curve a to curve b. Let $\theta_{ab}(z)$ be the orientation of this vector relative to the positive x-axis. The first method is to directly sum up the displacement of θ_{ab} moving upward in height, that is,

$$w_{ab} = \frac{1}{2\pi} \oint \frac{d\theta_{ab}}{dz} dz. \tag{319}$$

This method has the disadvantage of requiring a call to the arctan function to calculate θ_{ab} as

$$\theta_{ab} = \arctan(\frac{r_{ab_y}}{r_{ab_x}}). \tag{320}$$

Note that it is very important here that a four-quadrant arctan function is used. The second way is to work with the derivatives of $\boldsymbol{r}_{ab}$:

$$w_{ab} = \frac{1}{2\pi} \oint \frac{1}{r_{ab}^2} \hat{z} \cdot \boldsymbol{r}_{ab} \times \frac{d\boldsymbol{r}_{ab}}{dz} dz. \tag{321}$$

If we use a fixed interval width in both cases to calculate the integrals numerically, then the second method will be quicker. However, the results this method produces are significantly more noisy (see figure 32). For this reason the first method is used. The errors accumulated here are not due to errors in the estimation of the derivative. They are a result of the integrand varying on a finer resolution than the interval width. This is confirmed by using a cubic spline to estimate the derivatives.

As mentioned previously, the self helicities go to zero as the number of field lines increase, so they will be ignored. In all the experiments in this project we have a strong guide field along the length of the device. This guide field is several orders of magnitude larger than the transverse field, so we make the approximation

$$H = \frac{\Phi^2}{n^2} \sum_{a \neq b} \sum w_{ab}, \tag{322}$$

where $\Phi = \Phi_{guide}$ and is shared out evenly across all of the field lines.

Each field line winds with every other field line. So the field line at (x, y) winds with the field line at (x', y'), and the field line at (x', y') winds against the line at (x, y). So if we have an $n \times n$ grid of field lines, we have n^2 ways of choosing the first field line, and then $n^2 - 1$ ways to choose the second. This means there are $n^4 - n^2$ pairings, half of which are repeats. We halve the size of the calculation by considering only the unique pairings, and then doubling the answer.

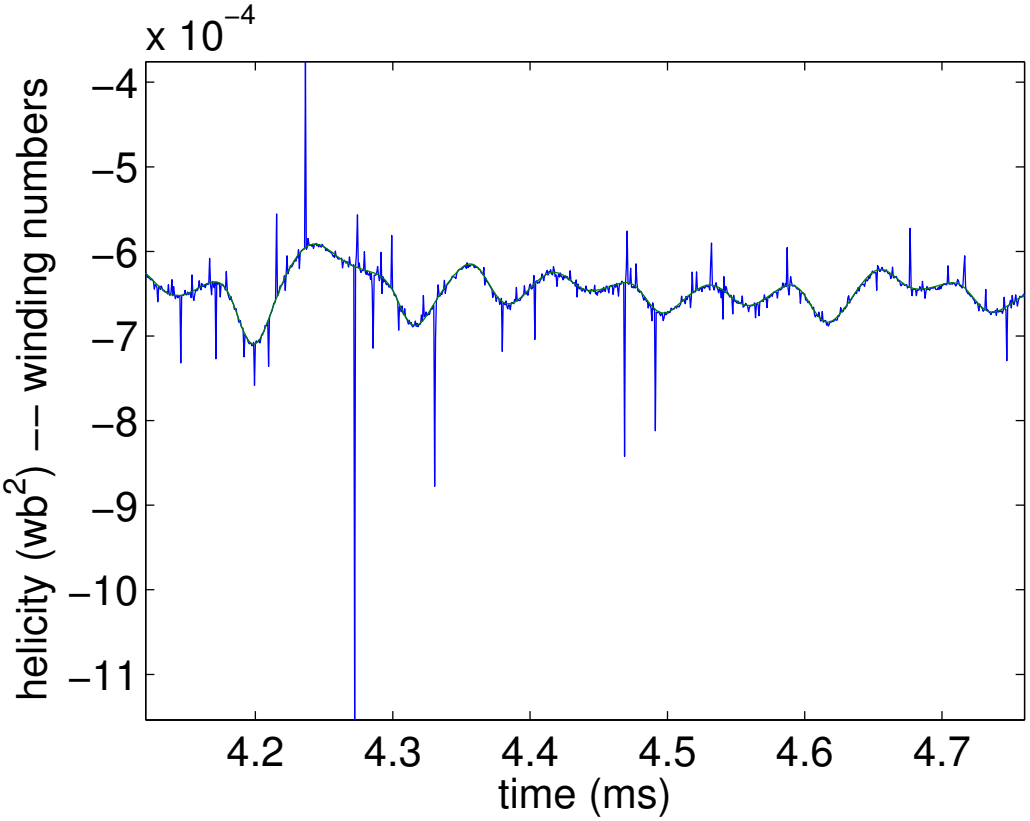

Figure 32: Magnetic helicity via winding numbers. This result is from experiment three. The blue curve shows the result of using equation 321 to find the winding numbers, the green equation 319. Both using the same sized interval width Δz. We use the green result in further calculations.

Note that there are two ways in which we could calculate the derivative of helicity. We have a choice of

$$(1) \qquad \frac{dH}{dt}_{winding} \equiv \frac{d}{dt}H(t). \tag{323}$$

$$(2) \qquad \frac{dH}{dt}_{winding} \equiv \frac{-2}{\pi}\sum_i \sum_j \left(\frac{d\theta_{ij}}{dt}\Phi_i\Phi_j\right). \tag{324}$$

In (1) we calculate winding numbers (using equation 322) and then find the numerical time derivative of the result. Option (2) comes from moving the time derivative inside $H(t)$. The problem with this is that it requires the field lines to be continuous in time. This may not be true if the magnetic field is not frozen into the plasma. We make the first choice.

3.11 Helium experiment

We study a helium plasma experiment involving two flux ropes. The flux ropes are situated in a strong background magnetic field (600G) and run from one end of the cylinder to the other. The transverse component of the magnetic field is several orders of magnitude smaller than the guide field.

Our data is arranged as $[547, 41, 41, 200, 3] = [t, x, y, z, v]$ where $v = \{0, 1, 2\}$ corresponds to the $\{x, y, z\}$ component of the vector field. The dimensions are:

$x \in [-2.4, 1.8]$ (cm)	$\Delta x = 0.105$ (cm)
$y \in [4.2, 9]$ (cm)	$\Delta y = 0.12$ (cm)
$z \in [63.9, 830.7]$ (cm)	$\Delta z = 3.8533$ (cm)
$t \in [-221.44, 2399.4]$ (ms)	$\Delta t = 4.7998$ (ms)

The magnetic flux density $\boldsymbol{B}$ in our dataset is given in units of gauss:

$$1G = \frac{Vs}{10^4 m^2}. \tag{325}$$

We will write helicity in terms of webers, where

$$1Wb = 1Vs. \tag{326}$$

A snap-shot of one position in time is shown in figure 33. The two entangled flux ropes can be identified visually. We calculate helicity using winding numbers (equation 322). Our results are shown in figure 34. Helicity is roughly conserved but oscillates. Figure 35 shows the derivative of helicity over the entire experiment, whilst figure 36 shows an enlarged section of this derivative. A reason for this oscillation is not known, however we suggest that helicity may be leaving the region and then coming back at a later point in time. Gekelman et al. [89] noted that the frequency of helicity oscillation corresponded to the frequency of the observed shear Alfvén waves. Another possibility is that field lines repeatedly undergo a process of entangling and untangling during each oscillation. The team at UCLA also calculated helicity but, instead of winding numbers, they employed the relative helicity equation (equation 68). We would expect our method of formulating helicity in terms of winding numbers to be more reliable as, equation 68 is only valid if integrated over all of space.

This work has been published in the journal *Physica Scripta* [5].

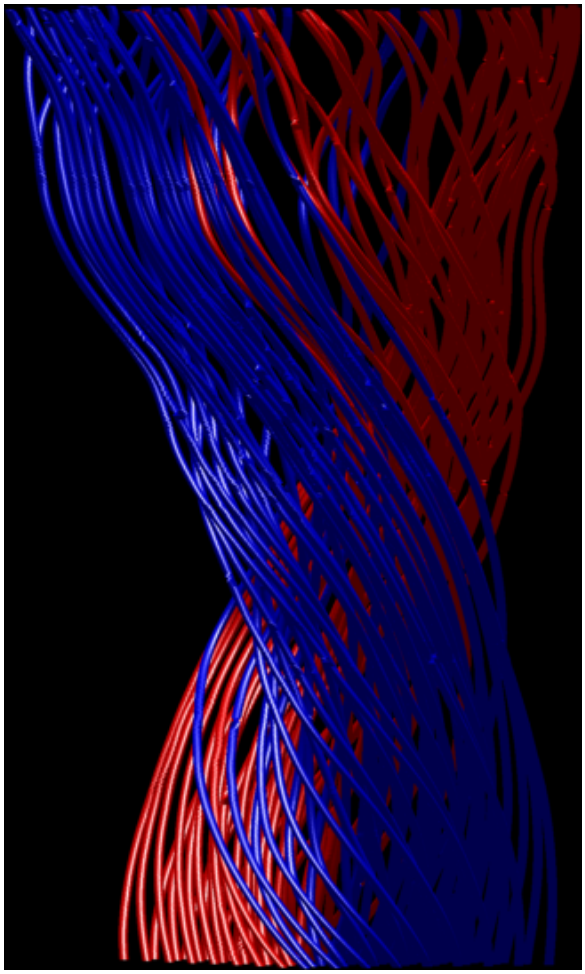

Figure 33: The entire set of fieldlines for one time shot. It appears possible to identify visually the two ropes.

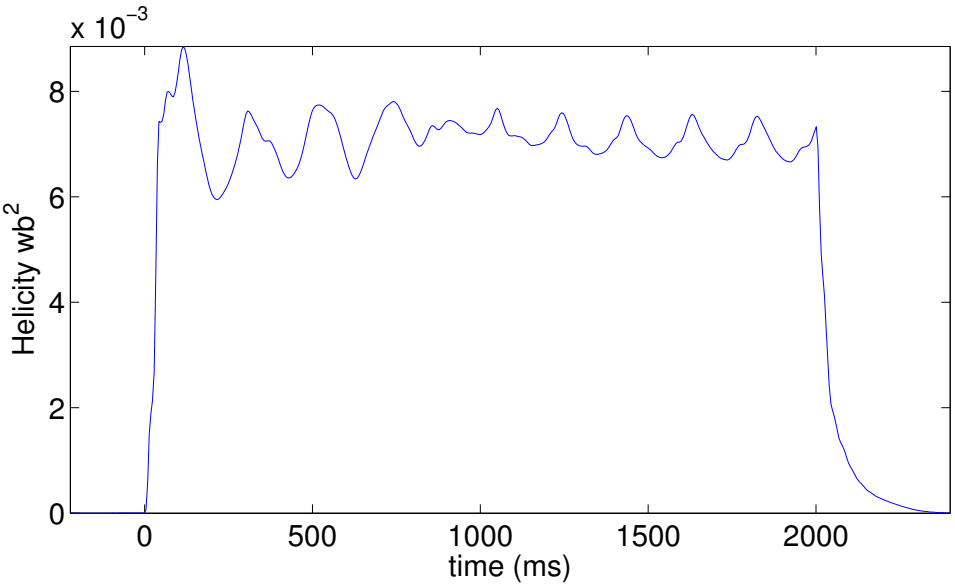

Figure 34: The magnetic helicity calculated using winding numbers (equation 322).

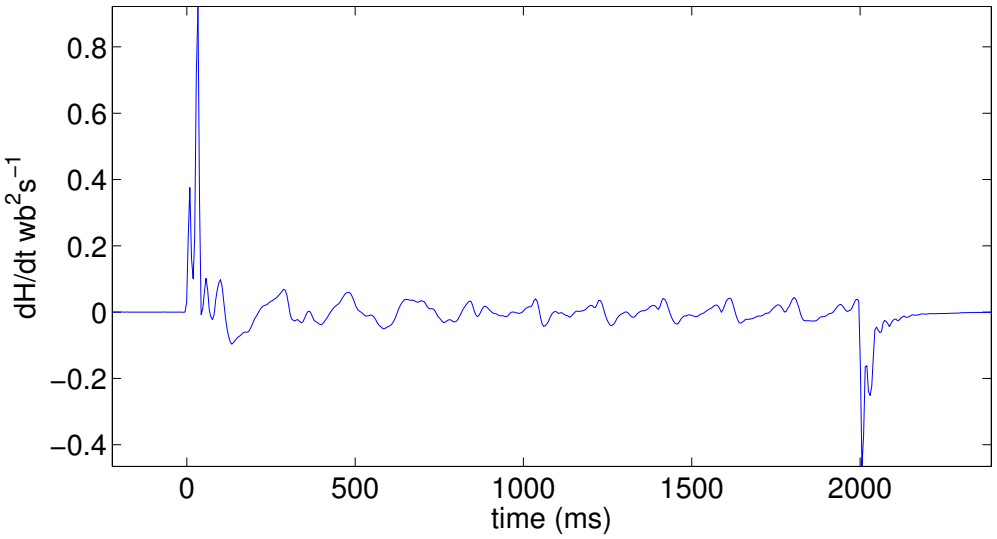

Figure 35: dH/dt from winding numbers.

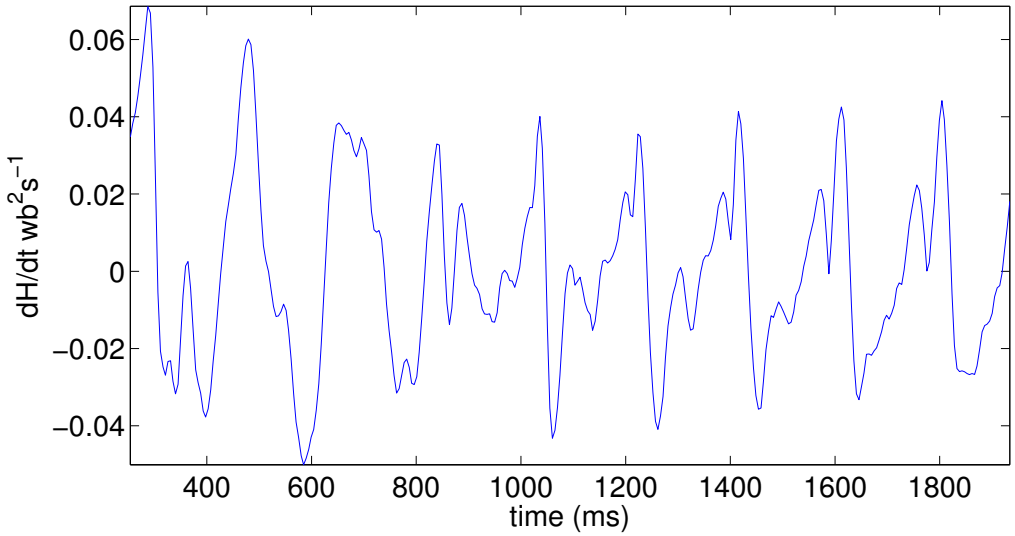

Figure 36: A zoomed in section of dH/dt over the time values in the middle of the experiment where we have less wild oscillation.

3.12 Argon 3-rope experiment

The next experiment we study is with argon plasma, similar in set up to the previous experiment, except that this time three flux ropes are created. The experiment is described in detail in [7]. The main difference is that we have three flux ropes, and multiple QSLs can be observed at the same time – see figure 38. Our results show

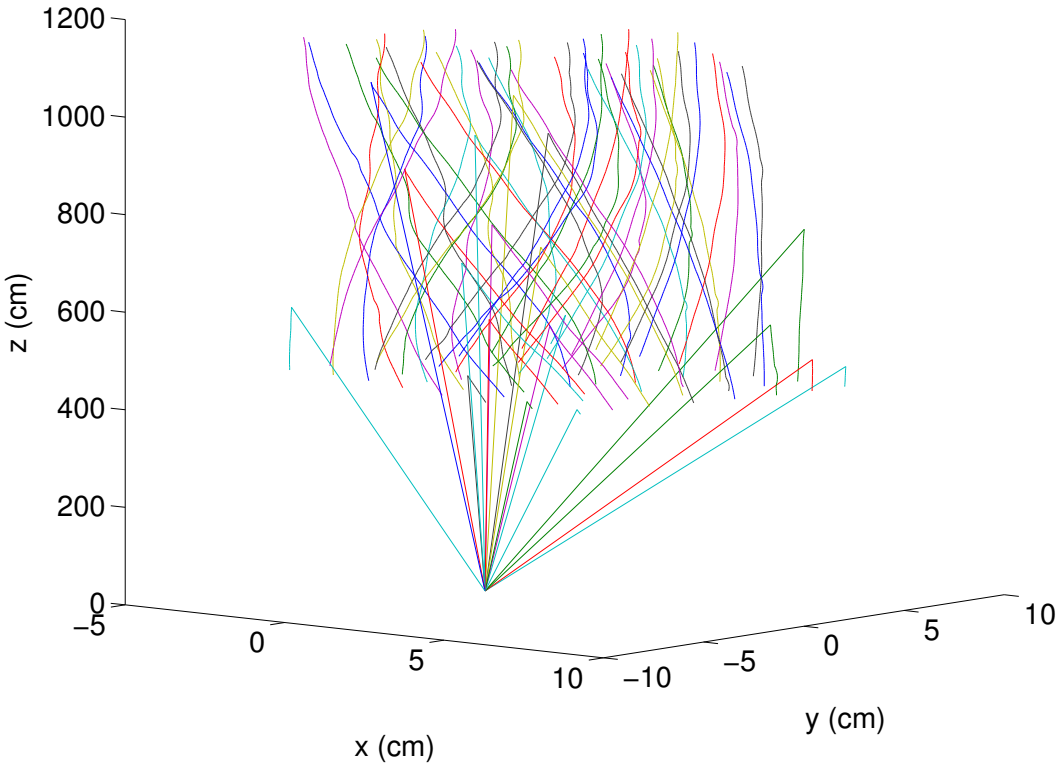

Figure 37: A Runge-kutta attempt to seed fieldlines. As the integrator requests a value outside of the monitored region a value of zero is returned.

that the flux ropes move too far off centre with many of the field lines leaving the monitored region. This is shown in figure 37. The team at UCLA plan to repeat this experiment with larger data planes so as to avoid this problem.

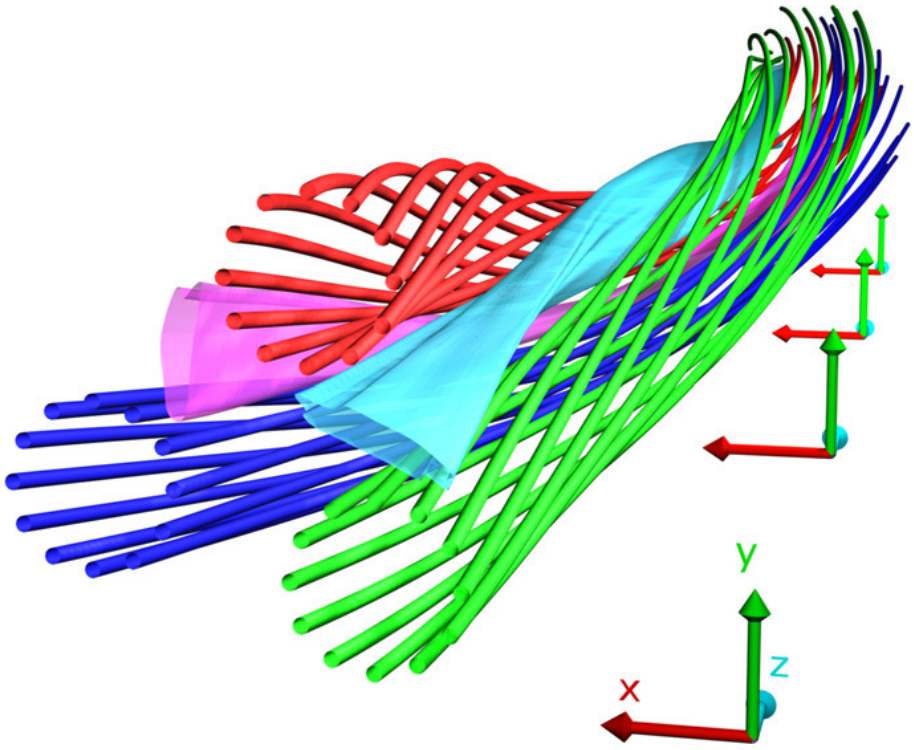

Figure 38: With three flux ropes multiple QSLs can be observed (shown here in light blue and pink).

3.13 Argon 2-rope experiment

We study the results of a 2-rope Argon plasma experiment. The experiment's parameters are listed in figure 39. A plot of the velocity data for a choice of height and time is displayed in figure 40. We will use equation 322 to calculate helicity using winding numbers and then take the time derivative of the result. This will be compared with the surface flux helicity equation (equation 73). The flux Φ in the cross-sectional area is

$$\begin{aligned}
\Phi &= B_{guidefield} \times Area_{grid}, && (327)\\
&= 330G \times 16^2 cm^2, && (328)\\
&= 330G \times 16^2 cm^2, && (329)\\
&= 84480 Gcm^2, && (330)\\
&= 84480 \frac{Wb}{10^8}, && (331)\\
&= 8.448 * 10^{-4} Wb. && (332)
\end{aligned}$$

```
Main Cathode:
   1 Hz, 15 ms Pulses
   60V, 3.7 kAmps

Background Plasma:
   Ar Plasma
   2.7e-5 Torr
   ωi = 12.5 kHz
   ωe = 0.9 GHz
   ne =  2 x 10^12 cm^-3
   ωpe   = 1.3 GHz
   ωpi   = 50 MHz
   ρi = 2.0 cm
   ρe = 0.02 cm
   Bz = 330 Gauss

Flux Ropes:
   2 Flux Ropes
   Length = 11m
   Diameter 3 cm of each rope
   Initial Rope Separation = 2 cm
   Voltage = 155 V, 9ms Pulses
```

Figure 39: The experimental parameters.

To get a feel for the size of helicity that we should find, note that a uniform twist of one turn would give a helicity of

$$H = \frac{\Phi^2}{n^2}(n^4 - n^2), \tag{333}$$

$$= \frac{7.1369 * 10^{-7}}{65^2}(65^4 - 65^2), \tag{334}$$

$$= 3.0146 * 10^{-3} Wb^2. \tag{335}$$

Figure 42 shows our findings. Helicity here is about $-6.4 \times 10^{-4} Wb^2$ which corresponds for about one fifth of a turn clockwise. Figure 41 shows a selection of field lines from a time shot about half way through the data. A twist of about one quarter appears consistent with this figure. Note that a twist of the top plane in the clockwise direction will result in a negative helicity. Helicity oscillates in a similar manner to the previous experiment.

Using figure 42 we can estimate the size of the time derivative of helicity dH/dt. We choose the points $(4.199ms, -7.096{\times}10^{-4} Wb^2)$ and $(4.245ms, -5.914{\times}10^{-4} Wb^2)$ (indicated by the green points on figure 336), and calculate

$$\frac{|\Delta H|}{\Delta t} = \frac{1.182 \times 10^{-4} Wb^2}{4.6 * 10^{-5} s} \approx 2.57 Wb^2 s^{-1}. \tag{336}$$

This appears to be consistent with our results shown in figure 43.

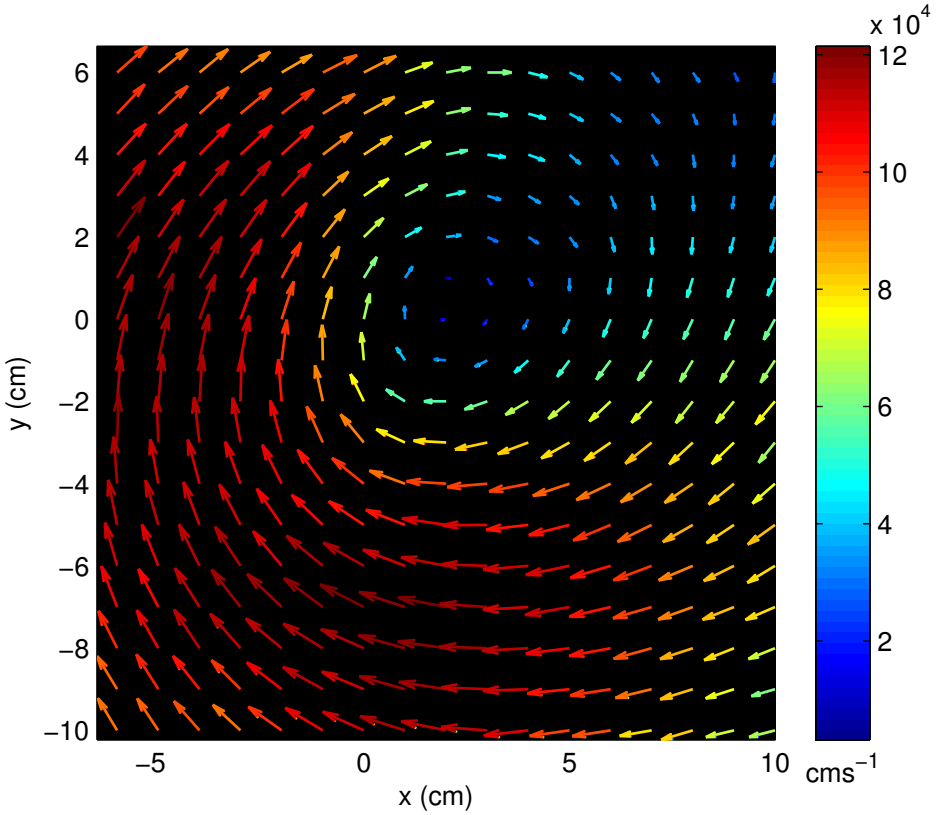

Figure 40: The velocity data for the layer $z = z_{max}$ at a time instance half way through the data.

For the flux surface helicity (equation 73) we need the velocity data. This has been normalised to the ion sound speed, which is given by:

$$c_s = 9.79 \times 10^5 \sqrt{\frac{T_e}{\mu}}. \tag{337}$$

The electron temperature of the background plasma, T_e, is $4eV$. The temperature inside the flux ropes may be higher than this; however, this would be hard to correct for as temperature is spatially non-uniform. The parameter μ is the ratio of the ion mass to the proton mass – for argon $\mu = 40$. This makes $c_s = 3.1 \times 10^5 cms^{-1}$. So the velocity data needs to be multiplied by $3.1 \times 10^5 cms^{-1}$.

Figure 44 shows our results. Helicity is conserved overall, but oscillates. The largest oscillation occurs between 4.2-4.32ms. The derivative oscillates around zero as expected. $dH/dt_{winding}$ is larger than $dH/dt_{surfFlux}$. We would expect this to be the case as we are neglecting the dissipation term (equation 74). Another reason is that with winding numbers we are assuming that none of the flux leaves through the side of the region: this is almost the case, but not quite. The field lines that do leave, which are few in number, leave when they are near the top plane of the region. If this happens, we use the field lines last known position, before it departed

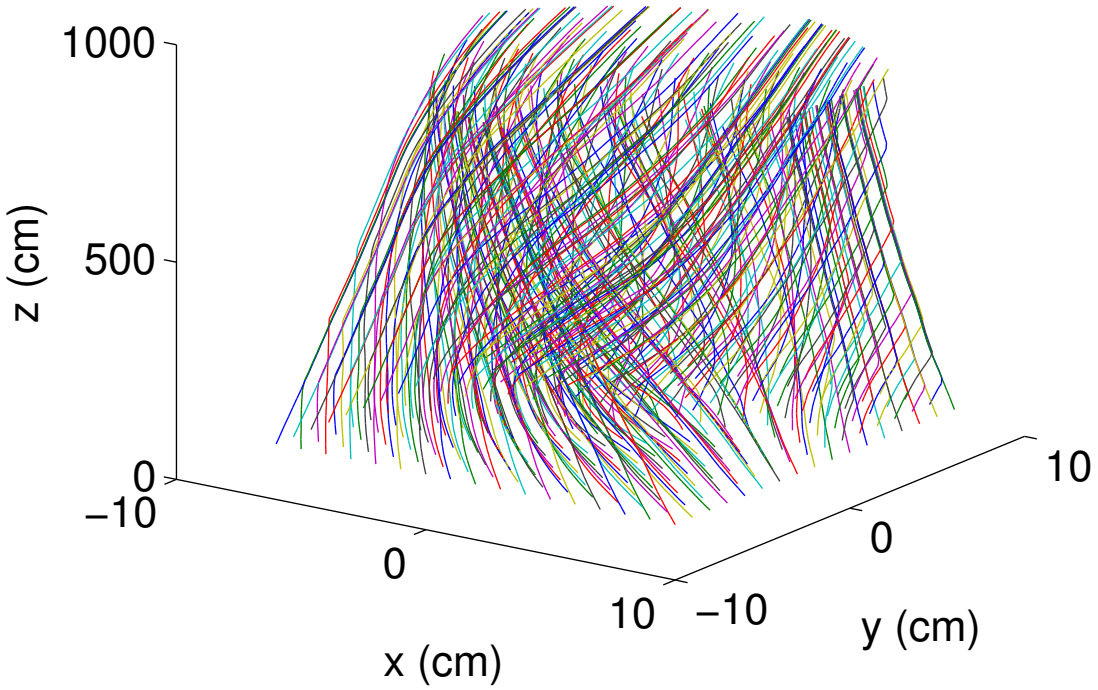

Figure 41: A selection of field lines half way through our data.

the region, to find its winding.

In a similar experiment, Gekelman el al. [89] calculated helicity using equation 68) and found that it is not conserved. For this experiment our results show that this is not the case, but it does similarly oscillate. A possible explanation for this is that helicity is leaving and then returning to the monitored region. Also, Gekelman el al. note that the frequency with which helicity oscillates seems to correspond to the oscillations in the Alfvén waves, however, this has not been investigated.

In 1974 Taylor conjectured [90] that if the dissipation occurs on small scales due to reconnection, then helicity dissipation should be much smaller than energy dissipation. We calculate

$$\text{mean}(\eta_{H_{eff}}) = 4.6828 \times 10^{-4}\Omega m, \qquad \text{mean}(\eta_{E_{eff}}) = 5.05 \times 10^{-2}\Omega m. \tag{338}$$

In agreement with the conjecture, our energy resistivity is one hundred times larger on average than the helicity resistivity.

Recall that we define an effective resistivity as

$$\eta_{H_{eff}} \equiv \frac{\frac{dH}{dt}_{ideal} - \frac{dH}{dt}_{non-ideal}}{2\int \boldsymbol{J}\cdot\boldsymbol{B}d^3\boldsymbol{x}}. \tag{339}$$

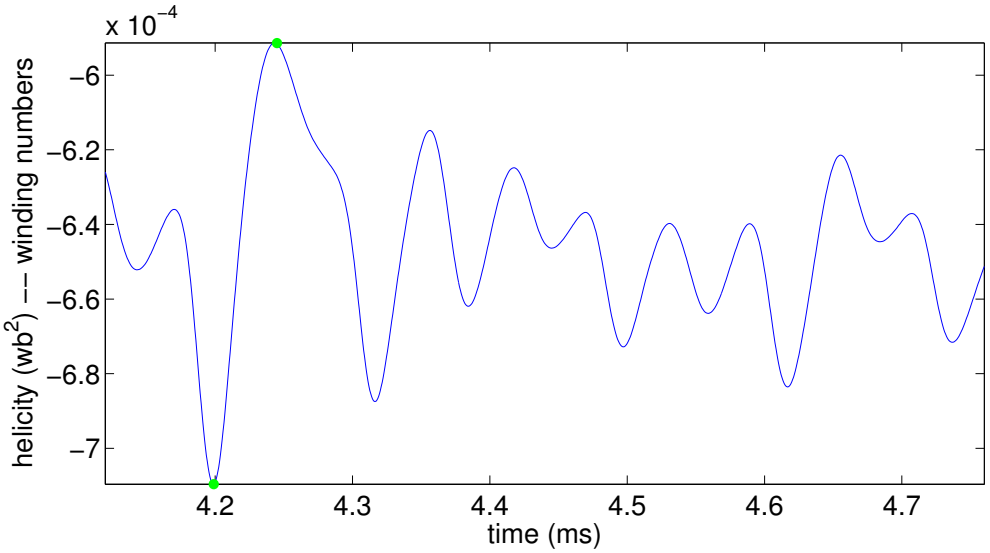

Figure 42: Magnetic helicity via winding numbers. The green points are used to estimate the derivative of helicity in equation 336.

To work this out we need to calculate the denominator. Using equation 274 we substitute for $\boldsymbol{J}$:

$$2\int \boldsymbol{J}\cdot\boldsymbol{B}d^3\boldsymbol{x} = \frac{2}{\mu_0}\int(\nabla\times\boldsymbol{B})\cdot\boldsymbol{B}d^3\boldsymbol{x}. \tag{340}$$

A plot of this term is provided in figure 45. $\eta_{H_{eff}}$ is shown in figure 46. The largest oscillation in resistivity occurs at the beginning of the experiment.

We can also calculate the magnetic energy according to equation 49; this is shown in figure 47. The energy oscillates with the largest jump being again at $t = 4.2$. Figure 48 shows the effective dissipation term. The power term (equation 314 – figure 49) and the Poynting flux term (equation 312 – figures 50-51) show similar patterns.

Recall that earlier – equation 315 – we defined an effective dissipation in terms of energy conservation, η_{eff}, as:

$$\begin{aligned}\eta_{E_{eff}}(T) \equiv \quad & \frac{1}{\iint(\nabla\times\boldsymbol{B})\times\boldsymbol{B}\cdot d\boldsymbol{S}dt + \iint(\nabla\times\boldsymbol{B})^2\ d^3\boldsymbol{x}dt}\cdot \\ * \quad & \left(\frac{\mu_0}{2}\int B^2|_{t=0} - B^2|_{t=T}\ d^3\boldsymbol{x} + \mu_0\int\!\!\int(\boldsymbol{v}\times\boldsymbol{B})\cdot(\nabla\times\boldsymbol{B})\,d^3\boldsymbol{x}dt\right. \\ & \left. + \int\!\!\int\left[\frac{1}{ne}\boldsymbol{B}\times(\nabla\times\boldsymbol{B}) + \mu_0(\boldsymbol{v}\times\boldsymbol{B})\right]\times\boldsymbol{B}\cdot d\boldsymbol{S}dt\right)\end{aligned} \tag{341}$$

We will now calculate this quantity. Figure 48 shows our results. There is some difference in our resistivity measurements. $\eta_{E_{eff}}$ remains positive for the entire experiment, whilst $\eta_{H_{eff}}$ oscillates changing sign and seems to capture more detail about the resistivity after the initial largest oscillation. Differences could be due to

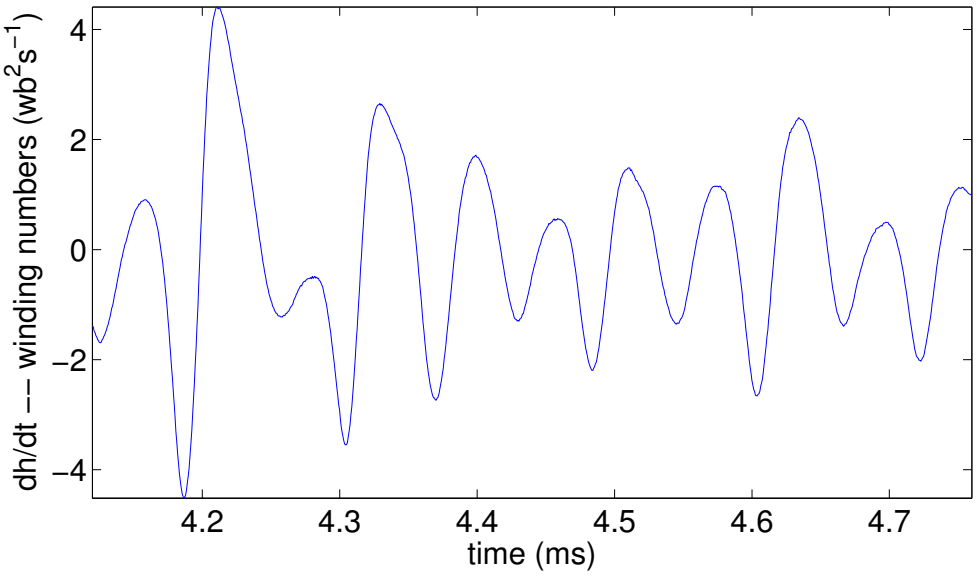

Figure 43: The time derivative of helicity obtained from the winding number formula.

many oppositely signed windings on a small scale cancelling out.

Is there a way we can estimate the resistivity η? With this aim we consider the time for a magnetic field to diffuse into a stationary unmagnetised plasma. Recall the generalised form of Ohm's law:

$$\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B} = \eta \boldsymbol{J} + \frac{1}{ne} \boldsymbol{J} \times \boldsymbol{B} - \frac{1}{ne} \nabla \cdot \boldsymbol{P}_e + \frac{m_e}{ne^2} \frac{\partial \boldsymbol{J}}{\partial t}. \tag{342}$$

For simplicity – following Chen [91] – we neglect the second term (the Hall term) and the last term (electron inertia term). Taking the curl of the remaining terms gives

$$\nabla \times \boldsymbol{E} = \eta (\nabla \times \boldsymbol{J}), \tag{343}$$

$$= \frac{\eta}{\mu_0} [\nabla \times (\nabla \times \boldsymbol{B})]. \tag{344}$$

Using Faraday's law (equation 30) to eliminate the electric field leads to

$$\frac{\partial \boldsymbol{B}}{\partial t} = \frac{\eta}{\mu_0} \nabla^2 \boldsymbol{B}. \tag{345}$$

Let t_{diff} be the time taken for the magnetic field to diffuse into the plasma with length l_{diff}. From above we have

$$t_{diff} = \frac{\mu_0 l_{diff}^2}{\eta}. \tag{346}$$

This is the total time for the magnetic field to diffuse into the plasma. If we consider a partial diffusion, ΔB, we can write

$$\eta_{Spitzer} \equiv \frac{B_0}{\Delta B} \frac{\mu_0 l_{diff}^2}{\Delta t}. \tag{347}$$

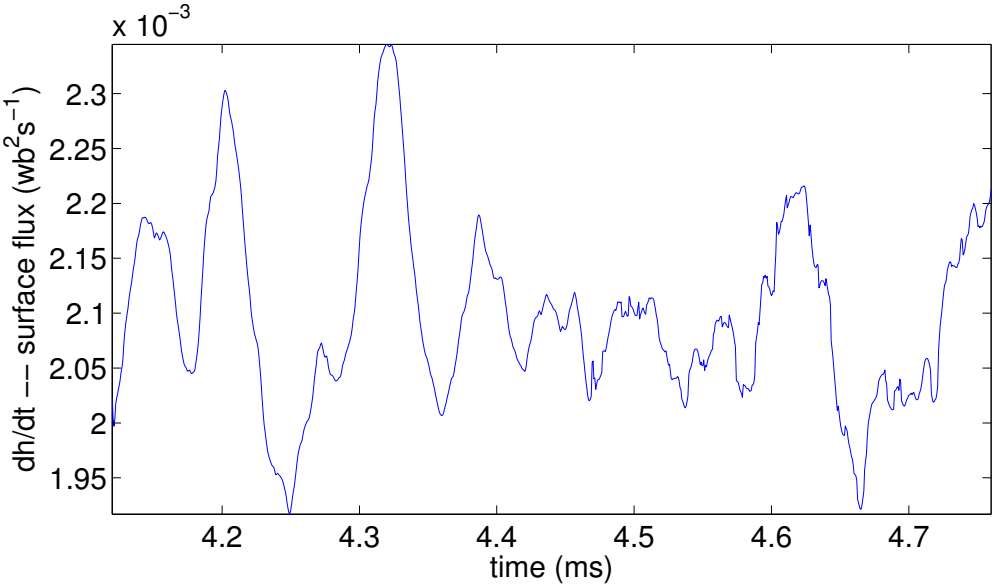

Figure 44: The total surface flux.

To estimate ΔB we need a sense of how much the magnetic field varies: we plot the size of the magnetic field in the $X-Y$ plane and use this. Figure 54 shows our results. For completeness we also include a plot showing how $|B_z|$ varies – figure 55.

$$\eta_{Spitzer} = \frac{B_0}{\Delta B}\frac{\mu_0 l_{diff}^2}{\Delta t}, \tag{348}$$

$$= \frac{10^{-3}}{1.122\times 10^{-4}}\frac{(4\pi\times 10^{-7})(10^{-1})^2}{7.3\times 10^{-5}}, \tag{349}$$

$$= 1.72\times 10^{-3}\Omega m. \tag{350}$$

This is about a factor of ten smaller than $\eta_{H_{eff}}$ and $\eta_{E_{eff}}$. Although the team at UCLA are unable to provide an estimate for the resistivity in this experiment, we do have access to results from other, similar experiments. We can compare our results with the results of Gekelman et al. [5] for the first experiment detailed in this project. Gekelman et al. calculate helicity and then use the helicity Schwarz inequality (equation 82) to estimate η. The ratio of this to $\eta_{Spitzer}$ is shown in figure 53 – $\eta/\eta_{Spitzer}$ is about about half an order of ten.

It is interesting that $\eta_{H_{eff}}$ oscillates about zero while $\eta_{E_{eff}}$ does not. This might be expected as, unlike energy, helicity can be both positive and negative. The energy resistivity cannot be negative as this would violate the second law of thermodynamics: an increase in entropy must be accompanied by a corresponding input of energy.

To investigate the reliability of our results, we re-write equation 309 as

$$-\left(\frac{dW}{dt}+\frac{1}{\mu_0}\int \boldsymbol{E}\times\boldsymbol{B}\cdot d\boldsymbol{S}\right)=\int \boldsymbol{E}\cdot\boldsymbol{J}d^3\boldsymbol{x}, \tag{351}$$

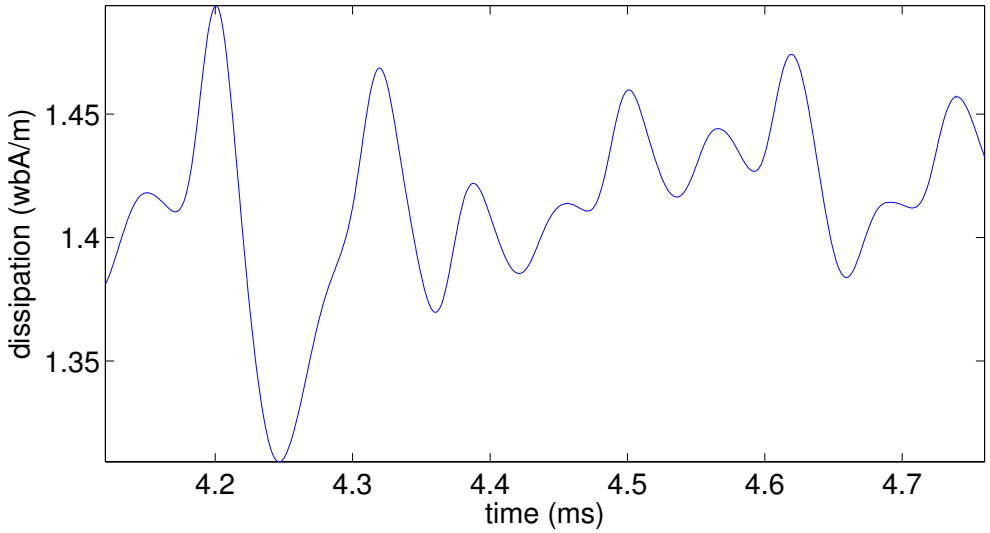

Figure 45: The dissipation term – equation 340.

where W is the magnetic energy. The first term on the left is shown in figure 52 and the complete left-hand side in figure 56. This should be equal to the power term on the right (figure 49).

This is not quite the case but they do appear to be correlated. A possible explanation for their difference is that we have not measured all of the terms in Ohm's law. The team at UCLA do not have a way of measuring the pressure term. It is probably not a scalar quantity.

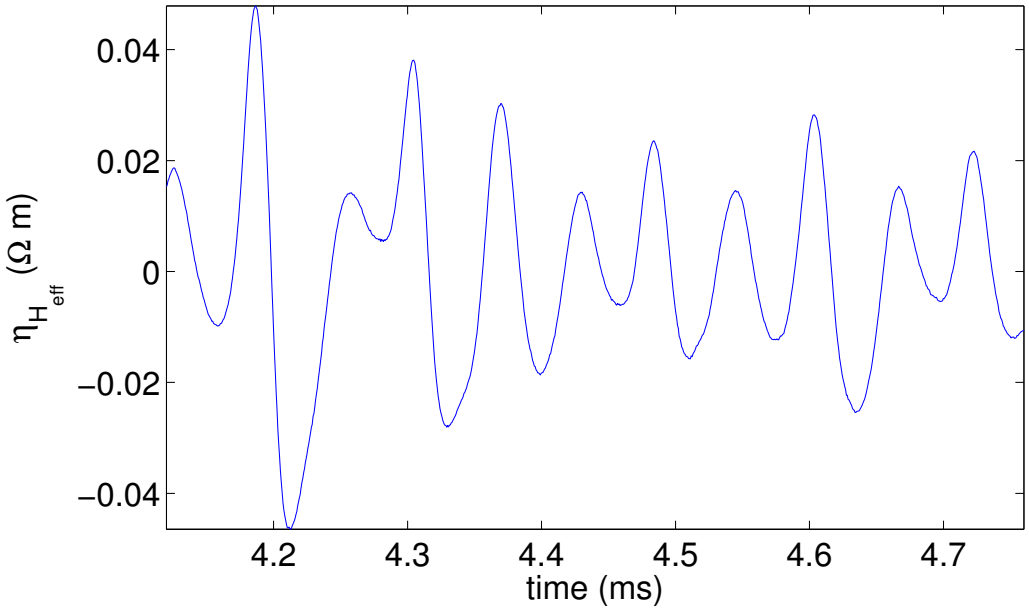

Figure 46: The effective resistivity calculated via helicity.

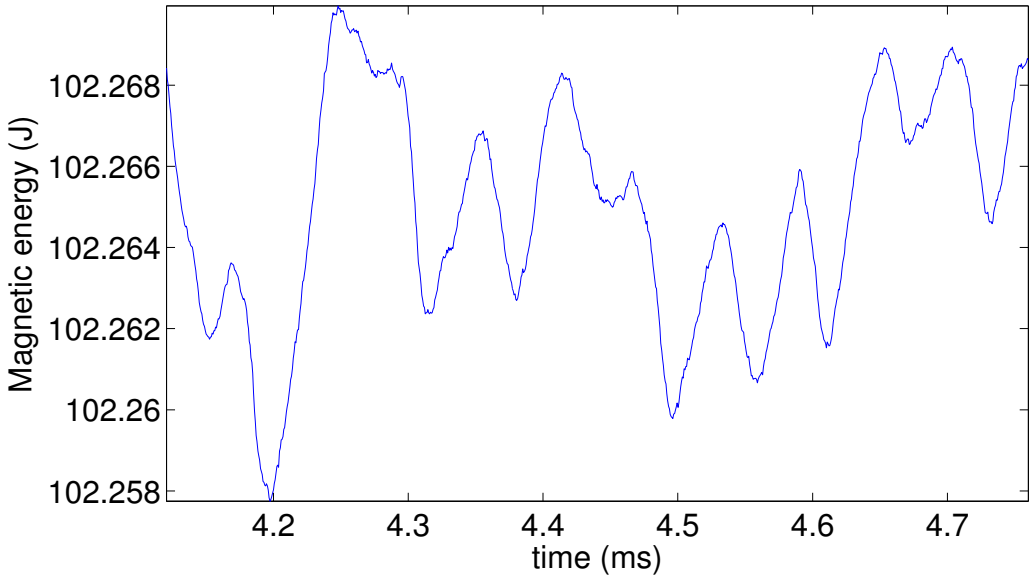

Figure 47: The magnetic energy.

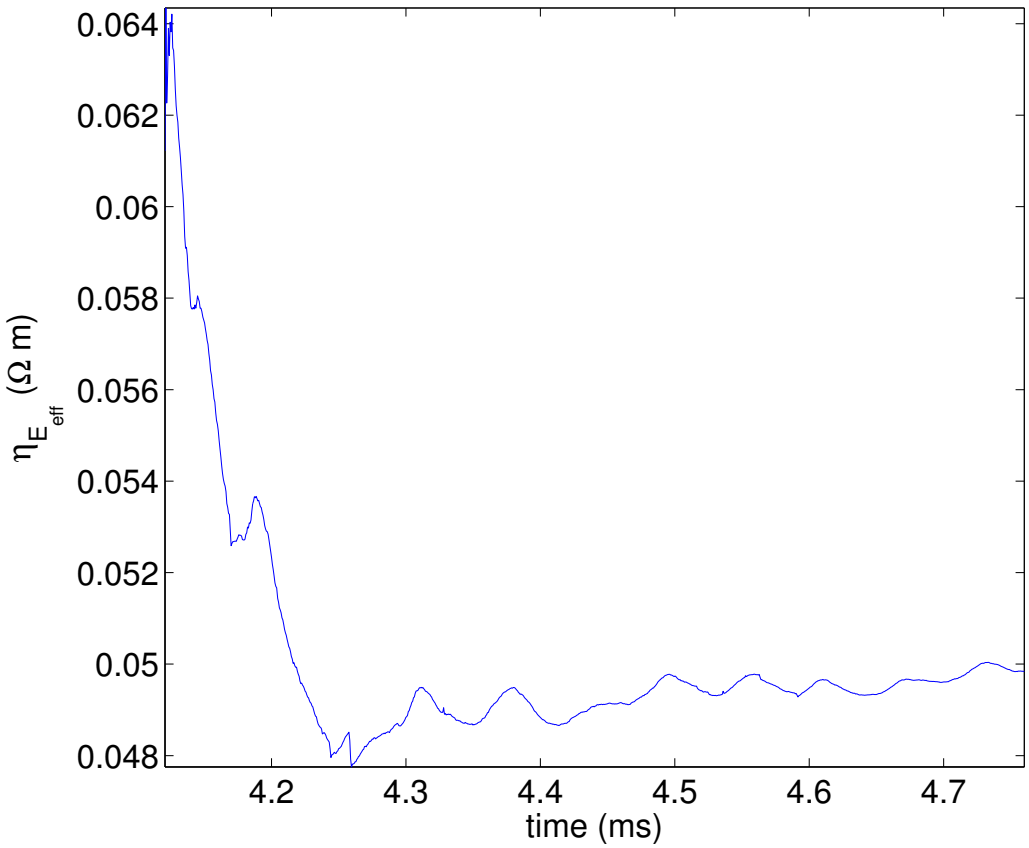

Figure 48: We define an effective resistivity by assuming that the currents move parallel to the magnetic field.

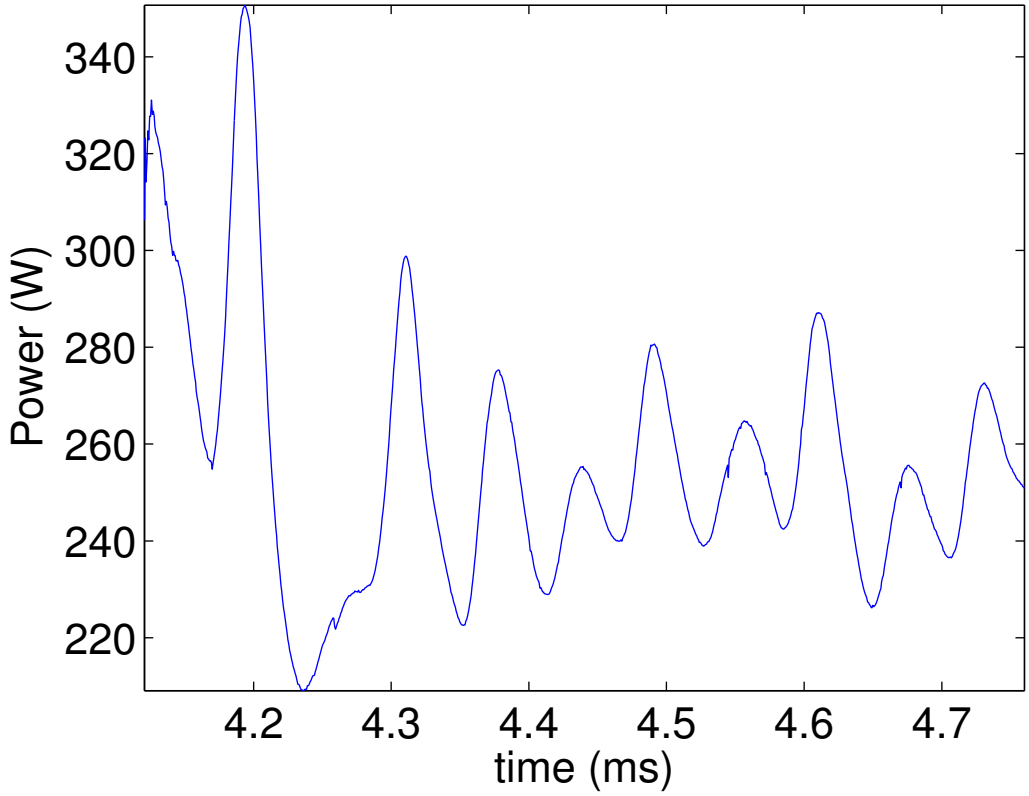

Figure 49: The power being transferred from the magnetic field.

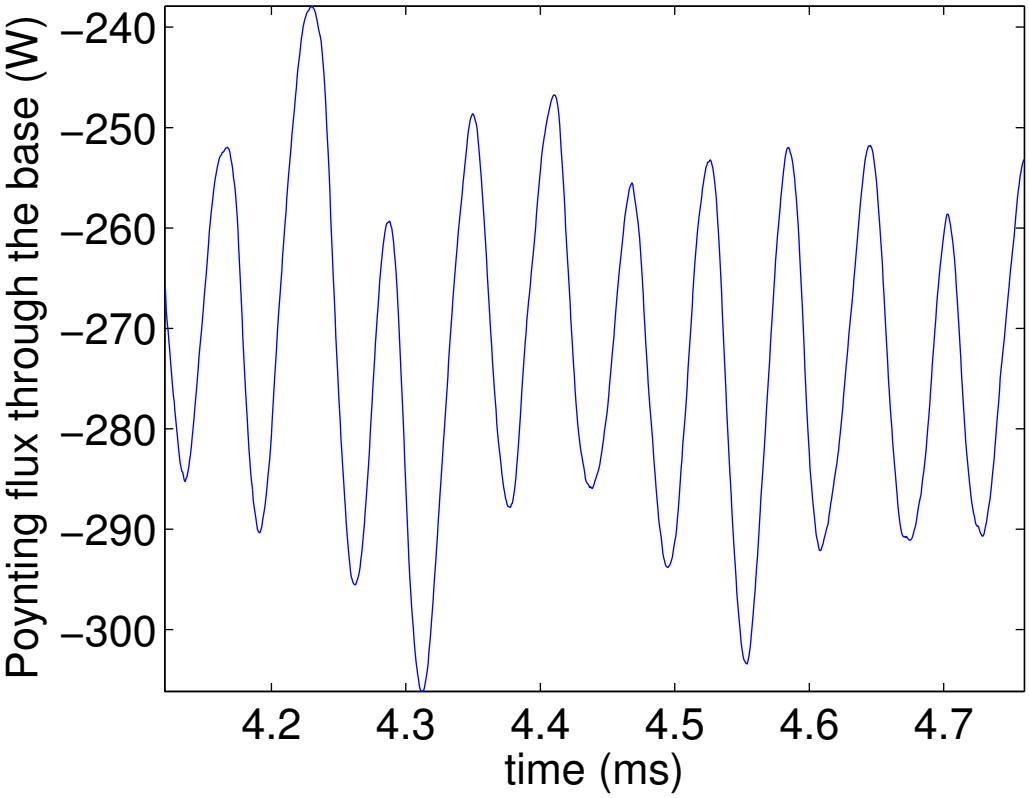

Figure 50: The Poynting flux through the base of the region.

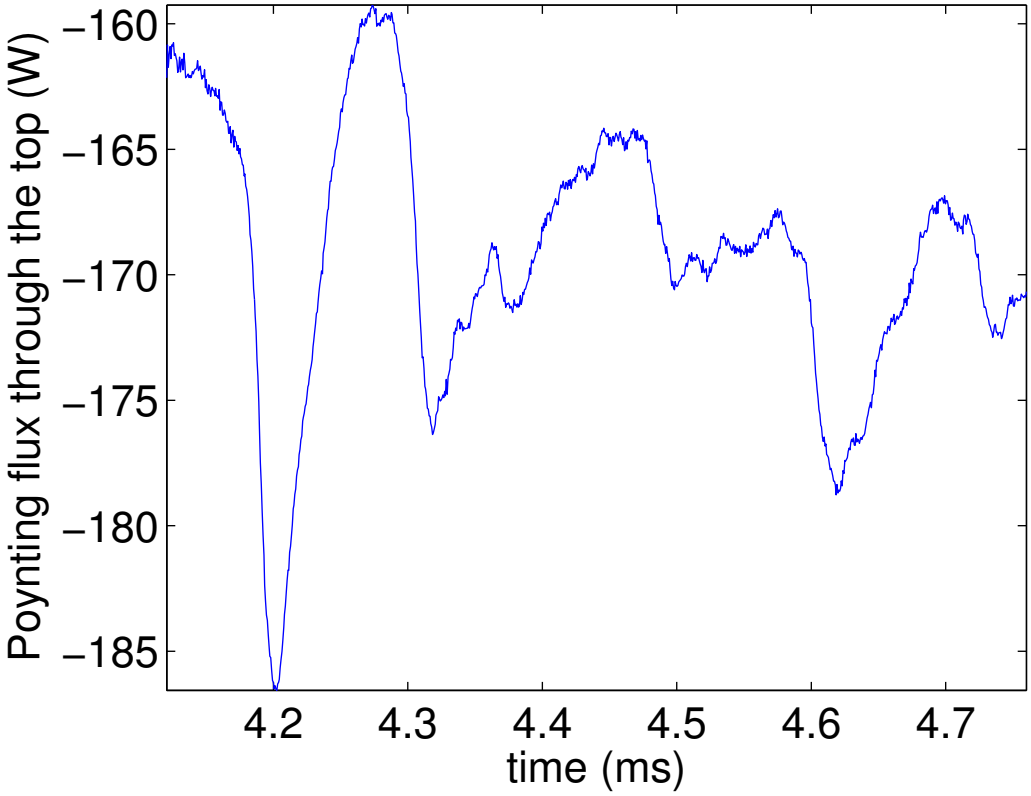

Figure 51: The Poynting flux through the top of the region.

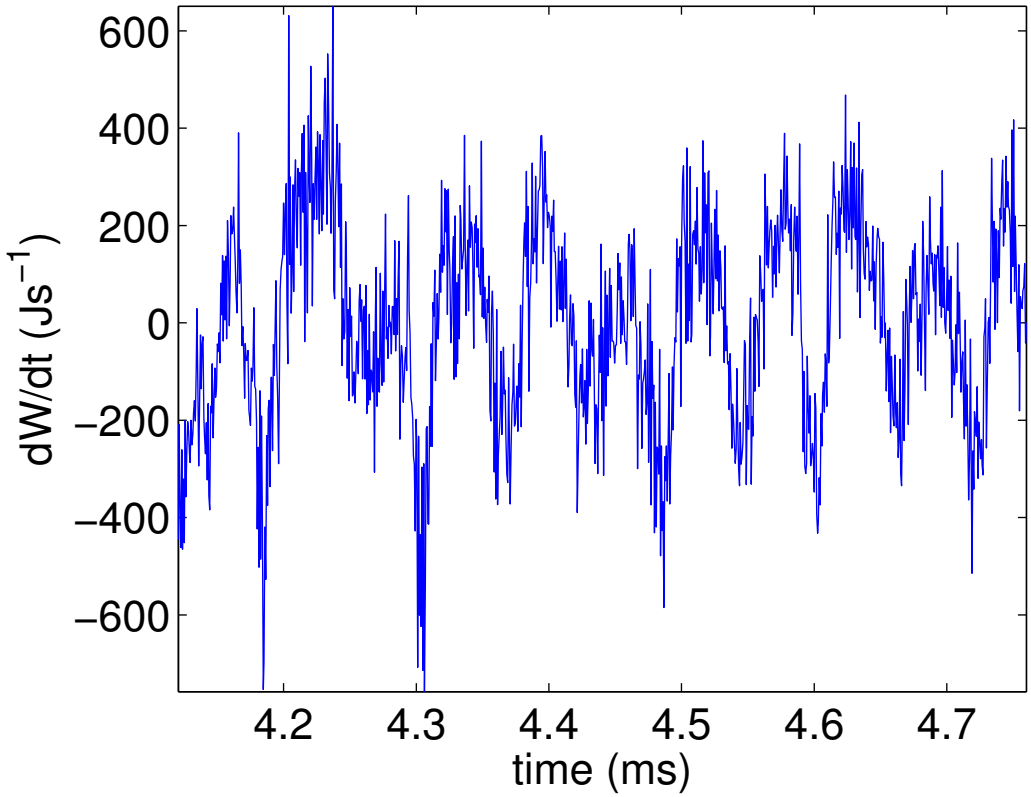

Figure 52: The time derivative of magnetic energy.

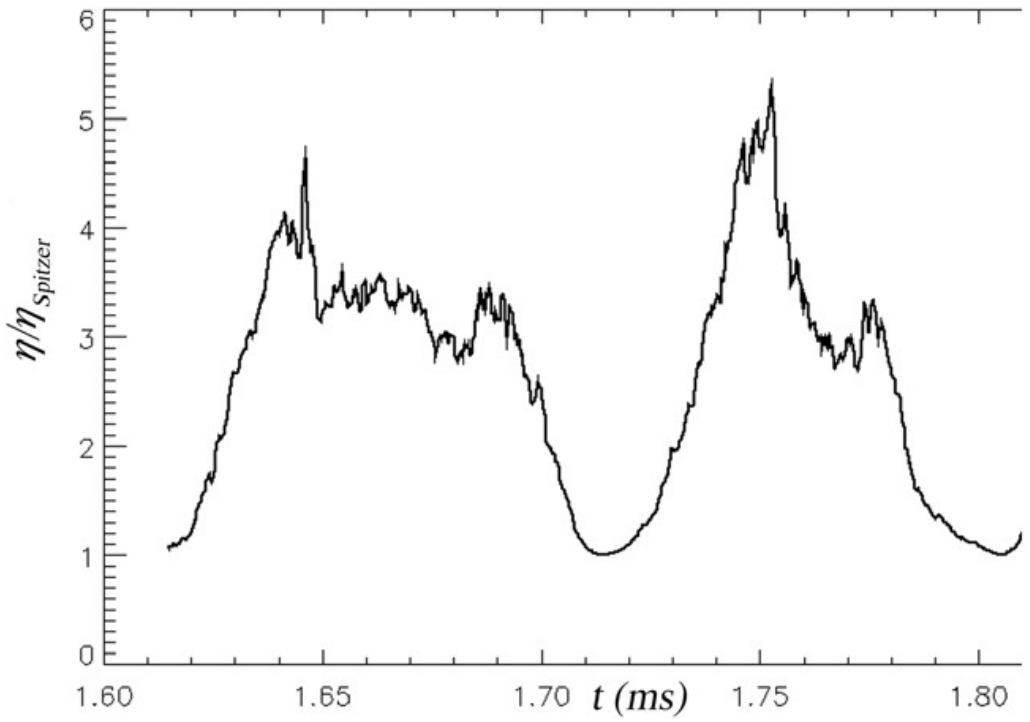

Figure 53: The ratio of η_H to $\eta_{Spitzer}$ reproduced from [5]. This data is from the first experiment in this project. We include it as a point of comparison for our results.

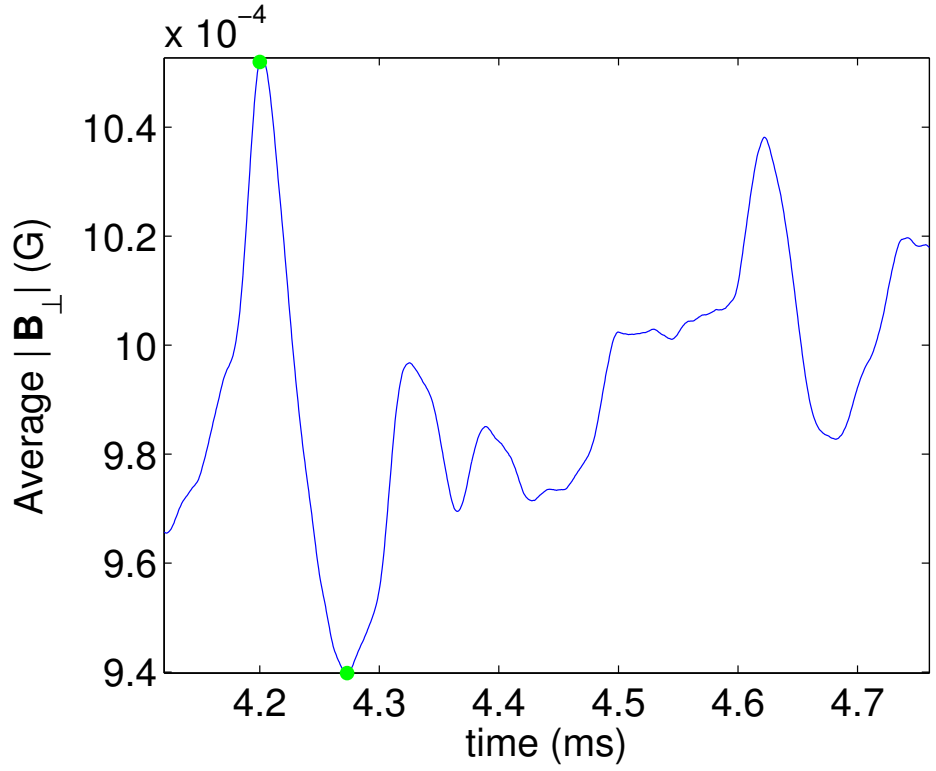

Figure 54: The average value of $|B_{x-y}|$ through different snapshots in time. The points in green are the values used in our calculation to estimate the resistivity (equation 349).

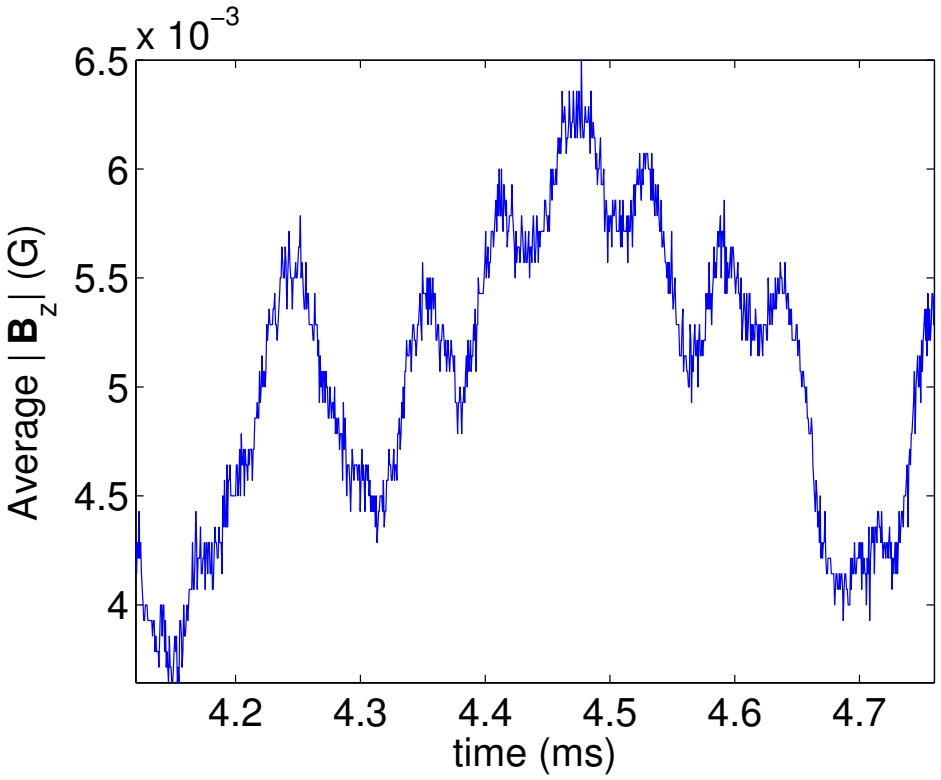

Figure 55: Average value of $|B_z|$. Note that the guide field (330G) has been subtracted.

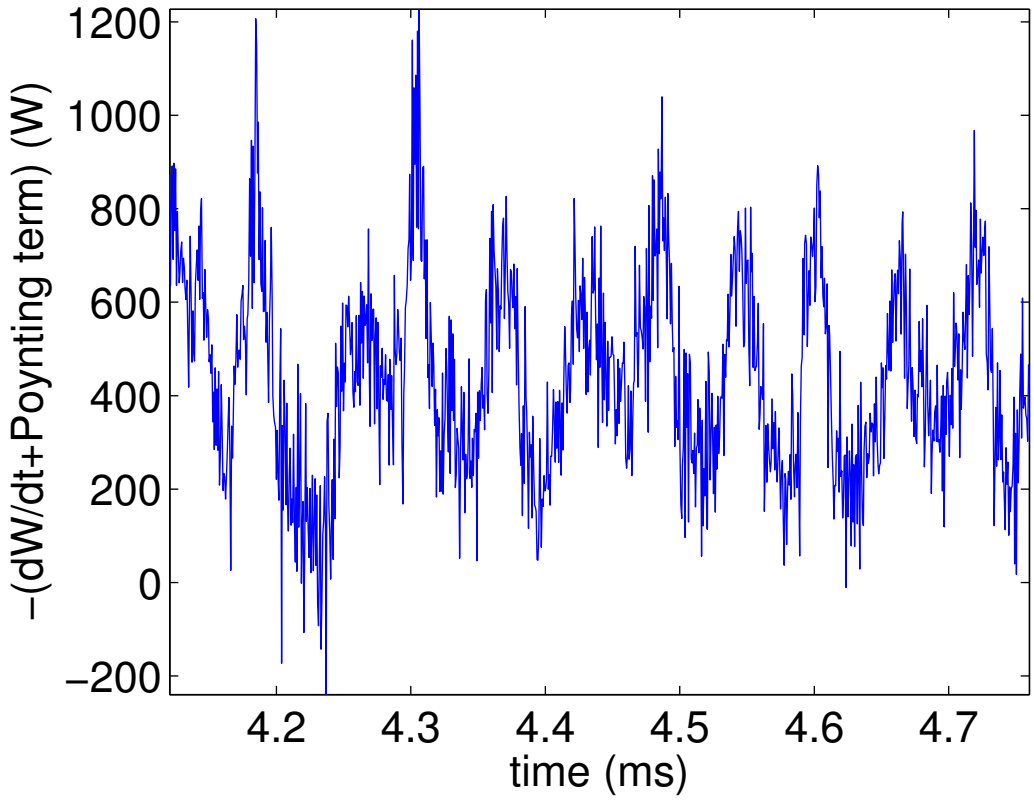

Figure 56: The left hand side of equation 351. This should be equal to the power term on the right (figure 49)

3.14 Drift velocity

As mentioned in the introduction, in ideal MHD the magnetic field remains frozen in the plasma. This is something that we can test. Consider figure 57. The red line shows a field line at $t = 0$, and the blue shows the same field line at $t = 1$. Let $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ be the horizontal placement of the top of field line at $t = 0$ and $t = 1$ respectively. If the field were frozen in to the plasma we would expect

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \Delta t \boldsymbol{v}|_{t=0}. \tag{352}$$

As a measure of the plasma deviation away from an ideal form we measure the difference

$$\boldsymbol{x}_{drift}(t_{n+1}) \equiv |\boldsymbol{x}(t_{n+1}) - (\boldsymbol{x}(t_n) + \Delta t \cdot \boldsymbol{v}(t_n))|. \tag{353}$$

The result is shown in figure 58. Note the similarities between this graph and that of dH/dt (figure 43) and η_{Heff} (figure 46). In a similar manner we can calculate

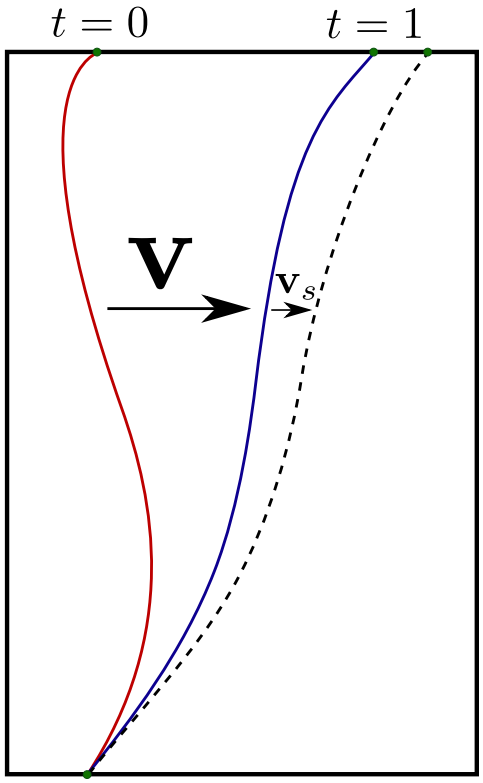

Figure 57: The position of one field line at $t = 0$ and at $t = 1$. If the magnetic field was "frozen in" then the field line would simple move with the plasma velocity $\boldsymbol{v}$, and end up as the line in blue. The slip-velocity $\boldsymbol{v}_s$ measures how much the field line's velocity deviates from this, due to non-ideal behaviour.

drift velocities as the extra velocity vector needed at each spatial point to keep the plasma moving with the magnetic field. As a matter of curiosity, we define

$$\frac{dH}{dt}_{frozenfield} = \int_S (\boldsymbol{A}_p \cdot \boldsymbol{B})(\boldsymbol{v} + \boldsymbol{v}_{drift}) - (\boldsymbol{A}_p \cdot (\boldsymbol{v} + \boldsymbol{v}_{drift}))\boldsymbol{B} \cdot d\boldsymbol{S}. \tag{354}$$

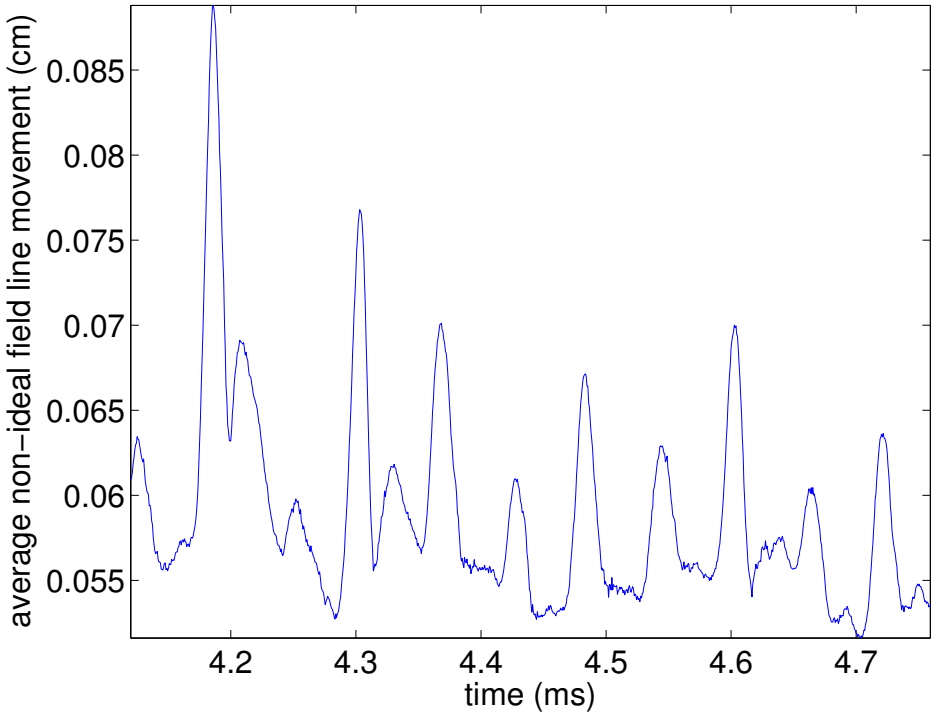

Figure 58: Spatial deviation of the field lines away from the frozen in condition.

The result is shown in figure 59. We were hoping that this addition would make up some of the difference between $dH/dt_{winding}$ (figure 43) and $dH/dt_{surfFlux}$ (figure 44). Unfortunately, it does not. However, the profiles of the graphs are now extremely similar – see figure 60.

Note that in many of our previous calculations the greatest jump in the quantity occurs at around $t = 4.18$ms. With this in mind, we plot the drift velocities before this point, at this point, and then much later. Figures (61-63) show our results. The drift velocities seem to want to spiral around in two separate swirls (centred at about (6,8) and (8,13) in figures 61 and 63). At the point where we have the largest non-ideal field line movement ($t = 4.18$ – figure 58) this behaviour changes into a circulation about the centre. We do not have an explanation for this.

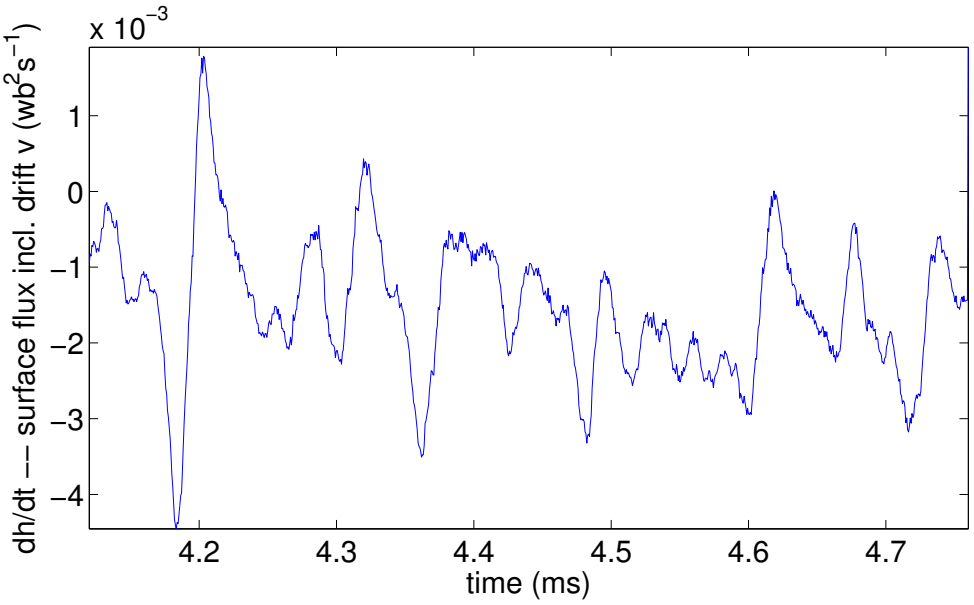

Figure 59: dH/dt after the addition of the drift velocity.

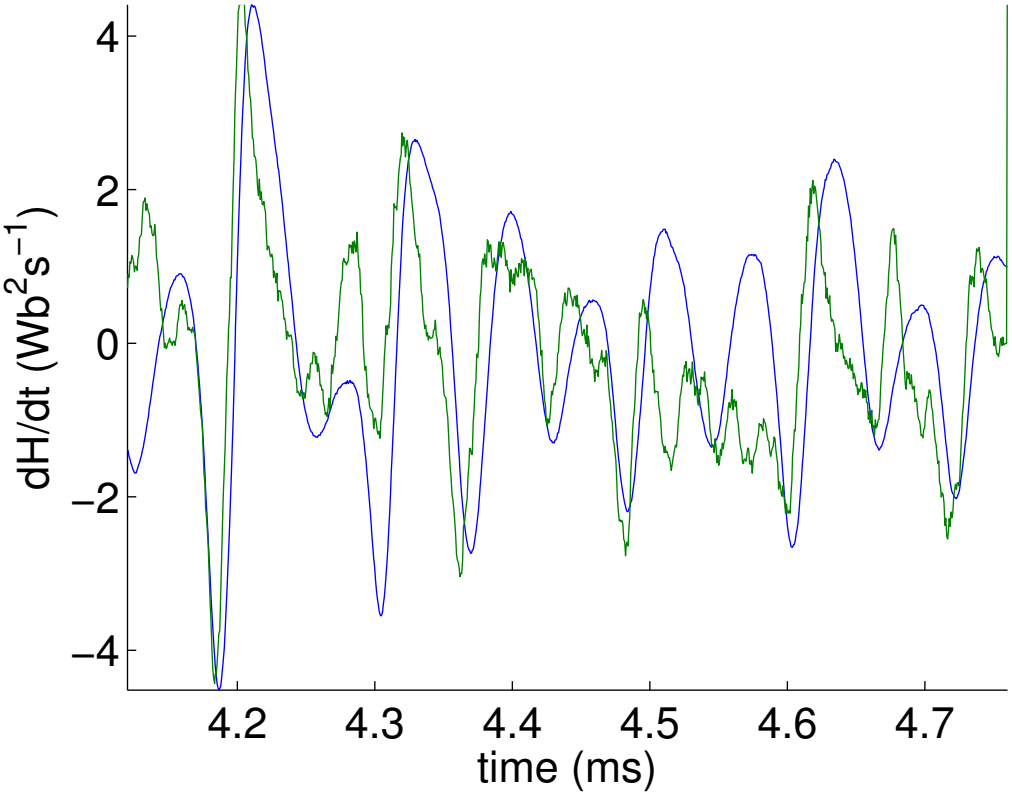

Figure 60: A comparison of the profiles of $dH/dt_{winding}$ (blue) and $dh/dt_{surfaceflux}$ (green). Here $dh/dt_{surfaceflux}$ has been rescaled to match the amplitude of $dH/dt_{winding}$. Notice how similar the profiles are after the addition of the drift velocity to the calculation.

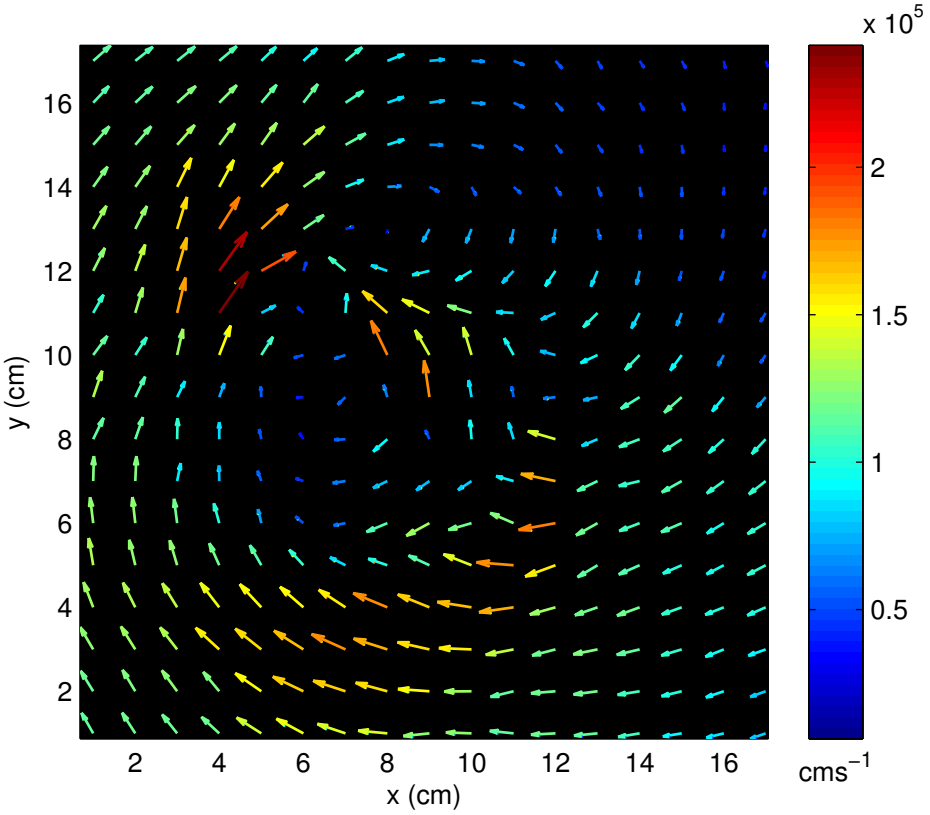

Figure 61: The drift velocity at $t = 4.15$ms .

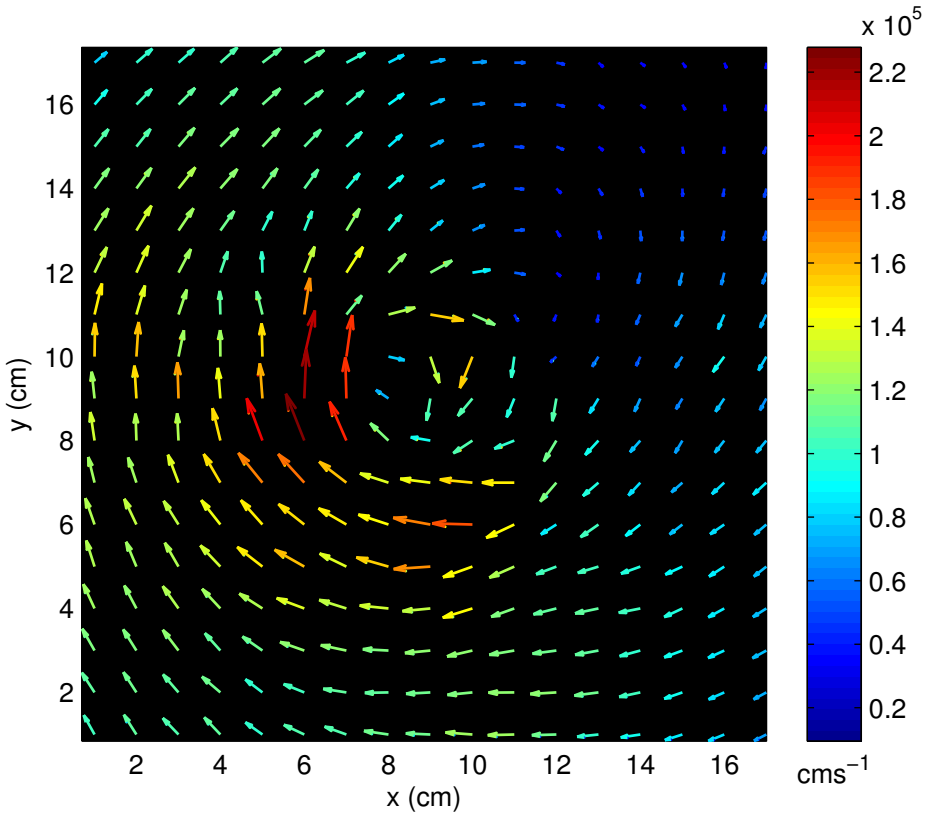

Figure 62: The drift velocity at $t = 4.18$ms.

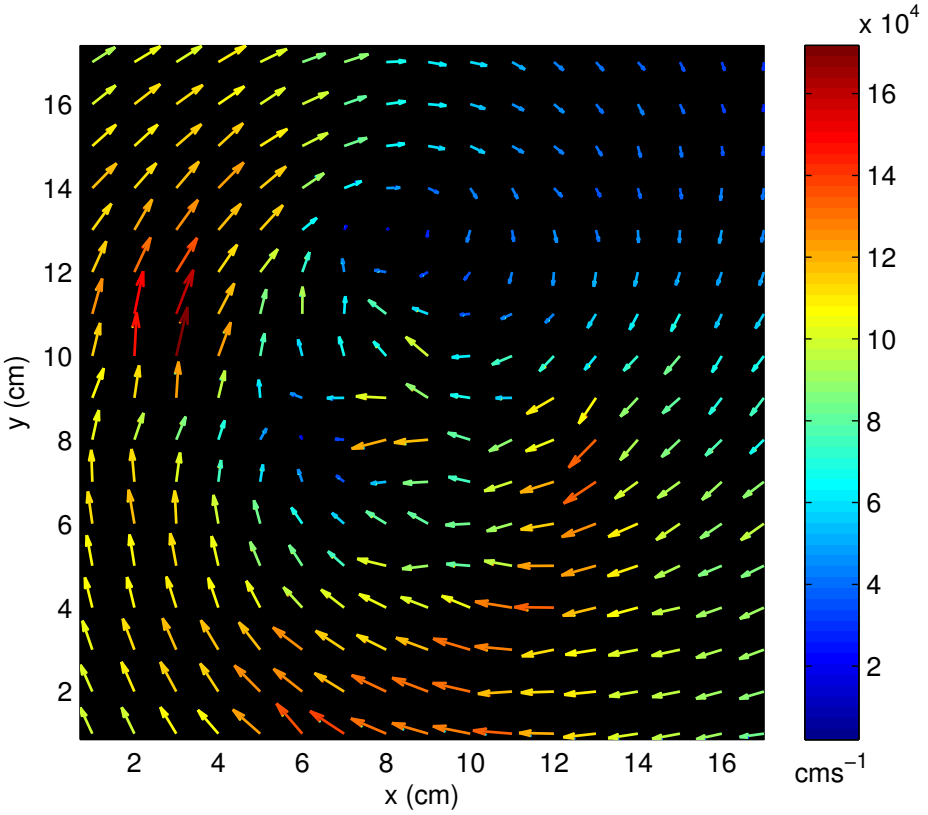

Figure 63: The drift velocity at $t = 4.79$ms.

4 Conclusions

4.1 Spherical writhe

This project has had two separate parts; we will discuss them in order. Our work on linking number, writhe and twist was to demonstrate how these measures can be extended to a spherical geometry. We used methods similar to those developed for open braid-like curves extended between two planes.

The nature of spherical geometry gave the work added complexity. In particular, we examined how it was possible to smoothly deform a ribbon so as to change its winding number with another curve. However, through an example we showed that linking number was unaffected and so still well defined.

In a similar way as for open curves between two flat planes, spherical open writhe was defined. It was proved that by taking an average over all secondary curves in a tube's surface, writhe only depends on the axis curve.

We then considered how surface motions moving the end points of flux tubes would affect the helicity defined in terms of linking number. This led us to derive an expression – using parallel transport – for how much a flux tube is twisted when its end point moves on a path that is not a geodesic. Changing the end point of the two curves does change the linking of the two tubes, but they twist by a corresponding amount, so that the effect cancels out, and is consistent with helicity in flat space. Finally, we gave an example of what would happen if the whole lower surface was to rotate (as might happen in planetary rotation) – helicity doesn't change as, relative to each other, the tubes are not winding.

4.2 UCLA experiments

To analyse the results from our plasma experiments, we needed to develop a way to seed field lines through the length of the cylindrical device. The divergence-free condition on $\boldsymbol{B}$, so that we have a physical field, led to a requirement on the continuity of the interpolator. We showed that if the coefficients of tricubic splines were fixed in a particular manner, then this continuity condition could be realised.

Armed with the interpolator, we set about writing a code to find these field lines. The fact that our grid is highly non-uniform in dimension, meant that we had to be careful how we went about implementing the RK45 stepper method.

Some of the integrals we went on to calculate required sophisticated methods to numerically integrate accurately, within a reasonable time frame. We used an existing adaptive Gauss-Konrad type quadrature package to find these results.

In all of our work we find that helicity oscillates. The team at UCLA have noted in their work that this is also the case. A possible explanation for this is that the flux tubes are continually winding and unwinding on small scales within each oscillation. Overall, helicity in these experiments is roughly conserved.

We use two separate methods to calculate resistivity: first through helicity, and then through energy conservation. In both cases we find resistivity to be of the same order. Our results match a theoretical estimate of resistivity. Our results are in agreement with Taylor's 1974 conjecture [90] that helicity dissipation should be much smaller than energy dissipation.

The helicity resistivity sees more wild oscillation and, interestingly, is both negative and positive. An explanation for this is that, unlike energy which is aware of the second law of thermodynamics, helicity can be both positive and negative. The energy resistivity cannot become negative because this implies a decrease in entropy in the system.

As we have both the magnetic field, and the velocity field, we tested the extent to which the magnetic field moves with the plasma (one of the assumptions of MHD). We use the amount that the magnetic field moves away from the plasma to define a drift velocity. Helicity, using the surface flux equation, is then re-calculated to include the effect of the drift. This has the effect of changing the derivative of helicity so that it oscillates both positively and negatively, in the same way that $dh/dt_{winding}$ does. Although the scales are different, the frequency and shape of the oscillations seem to be a much closer match.

For this project we have developed a set of sophisticated computer programs to analyse many aspects of magnetic field data. Further use of these tools is planned for the repeat of the second experiment – the one involving three flux ropes. In addition to experimental data, we would also like to have the chance to apply these to numerical simulations. Such simulations might allow cleaner conclusions to be drawn.

Time has been a constraint on this project. Our hopes were to do the resistivity

estimate for the second, three rope, experiment. Unfortunately, the data acquisition planes were not large enough and this was not possible. A number of results have had to be re-done due to problems mostly centring around measuring of the velocity data. For the final experiment, access to accurate data was only possible in the first week of February this year – about one year later than originally anticipated.

We calculated all of the terms in the generalised Ohm's law equation with the exception of the electron pressure term. The team at UCLA do not have an idea at present how to measure this. It would be interesting to see, how much the resistivity would change, if we ignored the anomalous terms in Ohm's law, so that $\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B} = \eta \boldsymbol{J}$ (which is a frequently used approximation).

5 Programs

This section contains a selection of programs that were used to obtain the results in this project. Two container classes were created to deal with the data in HDF5 files. The first (page 101) is a container class that can read data in, whilst the second (page 103), is a container class than can write its contents to disk. On page 105 is the program used to seed field lines using tricubic splines. Our program to calculate winding numbers is included on page 121.

```
////////////////////////////////////////
// C++ header file for                //
// HDF5 reader/container              //
//                                    //
// Any program using this should be   //
// compiled using h5c++ script.       //
//                                    //
// For large datasets JHDF5 object    //
// should be placed on the freestore. //
//                                    //
// Change dims to dataset dims.       //
////////////////////////////////////////

// dataset dimensions

const int ND=3;
const int NZ=14;
const int NX=65;
const int NY=65;
const int NT=1001;

#include <iostream>
#include <string.h>

#ifndef H5_NO_NAMESPACE
#ifndef H5_NO_STD
    using std::cout;
    using std::endl;
#endif  // H5_NO_STD
#endif

// C++ HDF5 header
#include "H5Cpp.h"

#ifndef H5_NO_NAMESPACE
    using namespace H5;
#endif

class JHDF5{
private:
    H5std_string FILE_NAME;
    H5std_string DATASET_NAME;
public:
    JHDF5(const JHDF5&);
    JHDF5& operator= (const JHDF5&);
    JHDF5(H5std_string fname,H5std_string dname){
        FILE_NAME=fname; DATASET_NAME=dname; };
    // Buffer to store data.
    double buffer[NZ][NY][NX] __attribute__ ((aligned (64)));
    // Function to read data.
    void initialise(int,int);
};

// copy constructor
JHDF5::JHDF5(const JHDF5& other){
    FILE_NAME=other.FILE_NAME; DATASET_NAME=other.DATASET_NAME;
    memcpy(buffer,other.buffer,sizeof(buffer));
}

// copy assignment constructor
JHDF5& JHDF5::operator= (const JHDF5& other){
    this->FILE_NAME=other.FILE_NAME;
    this->DATASET_NAME=other.DATASET_NAME;
    memcpy(this->buffer,other.buffer,sizeof(other.buffer));
    return *this;
}
```

```
67
68  // function to read data
69  void JHDF5::initialise(int v,int tpos){
70      H5File file(FILE_NAME, H5F_ACC_RDONLY);
71      DataSet dataset = file.openDataSet(DATASET_NAME);
72      DataSpace dataspace = dataset.getSpace();
73
74      hsize_t offset[]={0,0,0,0,0};
75      hsize_t count[]={1,NZ,NY,NX,1};
76      hsize_t memoffset[]={0,0,0};
77      hsize_t memcount[]={NZ,NY,NX};
78
79      dataspace.selectHyperslab(H5S_SELECT_SET, count, offset);
80      DataSpace memspace(3, memcount);
81      memspace.selectHyperslab(H5S_SELECT_SET, memcount, memoffset);
82      // Initialise data
83      memset(buffer,0,NX*NY*NZ);
84      // Read data
85      dataset.read(buffer, PredType::NATIVE_DOUBLE, memspace, dataspace);
86  }
```

```cpp
////////////////////////////////////////
// C++ header file for                //
// HDF5 writer/container              //
//                                    //
// Any program using this should be   //
// compiled using h5c++ script.       //
//                                    //
// For large datasets myArray object  //
// should be placed on the freestore. //
//                                    //
//                                    //
////////////////////////////////////////

#include <iostream>
#include <string.h>

#ifndef H5_NO_NAMESPACE
#ifndef H5_NO_STD
    using std::cout;
    using std::endl;
#endif  // H5_NO_STD
#endif

// C++ HDF5 header.
#include "H5Cpp.h"

#ifndef H5_NO_NAMESPACE
    using namespace H5;
#endif

class myArray{
public:
    static const int dimt = 1001;
    myArray(){memset(buffer,0,dimt);}
    myArray(const myArray&);
    ~myArray(){}
    myArray& operator= (const myArray&);
    // Buffer to store data.
    double buffer[dimt];
    // Function to write data.
    void write(const H5std_string,const H5std_string);
    // Function to create data file
    void createFile(const H5std_string,const H5std_string);
};

// copy constructor
myArray::myArray(const myArray& other){
    memcpy(buffer,other.buffer,sizeof(buffer));
}

// copy assignment constructor
myArray& myArray::operator= (const myArray& other){
    memcpy(this->buffer,other.buffer,sizeof(other.buffer));
    return *this;
}

void myArray::createFile(const H5std_string FILE_NAME,const H5std_string DATASET_NAME){
    H5File* file= new H5File( FILE_NAME, H5F_ACC_TRUNC );
    /* Fill value for the dataset */
    int fillvalue = 0;
    DSetCreatPropList plist;
    plist.setFillValue(PredType::NATIVE_DOUBLE, &fillvalue);
    hsize_t dims[]={dimt};
    const int rank=1;
    DataSpace dataspace(rank, dims);
    DataSet* dataset = new DataSet(file->createDataSet(
```

```
67          DATASET_NAME, PredType::NATIVE_DOUBLE, dataspace, plist));
68      delete dataset;
69      delete file;
70  }
71
72  // function to write data.
73
74  void myArray::write(const H5std_string FILE_NAME,const H5std_string DATASET_NAME){
75      H5File* file = new H5File(FILE_NAME,H5F_ACC_RDWR);
76      DataSet* dataset = new DataSet(file->openDataSet(DATASET_NAME));
77
78      int rankmem = 1;
79      hsize_t dimsmem[] = {dimt};
80      DataSpace memspace(rankmem, dimsmem);
81      hsize_t offsetmem[]={0};
82
83      DataSpace dataspace = dataset->getSpace();
84      int rank = dataspace.getSimpleExtentNdims();
85      hsize_t count[] = {dimt};
86      hsize_t offsetdata[] = {0};
87
88      dataspace.selectHyperslab(H5S_SELECT_SET, count, offsetdata);
89      memspace.selectHyperslab(H5S_SELECT_SET, dimsmem, offsetmem);
90      dataset->write(buffer, PredType::NATIVE_DOUBLE, memspace, dataspace);
91      delete dataset;
92      delete file;
93
```

```
1  // command to compile
2  // ./h5c++ 3dsplinesv3.cpp -std=c++0x -pthread -O3 -msse4 -o 3dsplines /usr/local/lib/libgslcblas.a
/usr/local/lib/libgsl.a /usr/local/lib/libtricubic.a
3
4  ////////////////////////////////////////
5  // Program to seed fieldlines using RK45 //
6  // and tricubic splines.                 //
7  ////////////////////////////////////////
8
9  #include <future> // Thread handlers:
10 #include <chrono>
11 #include <thread>
12 #include <mutex>
13 #include <gsl/gsl_errno.h> // Error handler
14 #include <gsl/gsl_spline.h> // Splines
15 #include <gsl/gsl_odeiv2.h> // Adam-Bashforth methods
16 #include "JHDF5interp3v2.h" // contains class to read hdf5 files.
17 #include "myArrayv3.h" // contains class to write hdf5 files.
18 #include "tricubic.h"
19
20 const double XPOS[65]={-6,-5.75,-5.5,-5.25,-5,-4.75,-4.5,-4.25,-4,-3.75,-3.5,-3.25,-3,-2.75,-2.5,-2.25,-2,-
1.75,-1.5,-1.25,-1,-0.75,-0.5,-0.25,0,0.25,0.5,0.75,1,1.25,1.5,1.75,2,2.25,2.5,2.75,3,3.25,3.5,3.75,4,4.25,4.5,
4.75,5,5.25,5.5,5.75,6,6.25,6.5,6.75,7,7.25,7.5,7.75,8,8.25,8.5,8.75,9,9.25,9.5,9.75,10};
21 const double YPOS[65]={-10,-9.75,-9.5,-9.25,-9,-8.75,-8.5,-8.25,-8,-7.75,-7.5,-7.25,-7,-6.75,-6.5,-6.25,-6
,-5.75,-5.5,-5.25,-5,-4.75,-4.5,-4.25,-4,-3.75,-3.5,-3.25,-3,-2.75,-2.5,-2.25,-2,-1.75,-1.5,-1.25,-1,-0.75,-0.5
,-0.25,0,0.25,0.5,0.75,1,1.25,1.5,1.75,2,2.25,2.5,2.75,3,3.25,3.5,3.75,4,4.25,4.5,4.75,5,5.25,5.5,5.75,6};
22 const double ZPOS[14]={127.8,191.7,255.6,319.5,383.4,447.3,511.2,575.1,639,702.9,766.8,830.7,894.6,958.5};
23
24 std::mutex readLock;
25 std::mutex writeLock;
26
27 // Class to seed streamline at a particular time
28 class worker{
29 public:
30     worker();
31     ~worker()
32     {
33         //delete results;
34         delete Ax; delete Ay; delete Az;
35         gsl_interp_accel_free(lookupACCx);
36         gsl_interp_accel_free(lookupACCy);
37         gsl_interp_accel_free(lookupACCz);
38     }
39     void seed(int); // Actually do the seeding
40 private:
41     // array to hold Hermite coefficients
42     double ax[13][64][64][64];
43     double ay[13][64][64][64];
44     double az[13][64][64][64];
45
46     // Return magnetic field
47     double getBx(double,double,double);
48     double getBy(double,double,double);
49     double getBz(double,double,double);
50
51     // Grid spacing
52     const double delX = 0.25;
53     const double delY = 0.25;
54     const double delZ = 63.9;
55     const double minX = -6;
56     const double maxX = 10;
57     const double minY = -10;
58     const double maxY = 6;
59     // Data size
60     const int NX = 65;
61     const int NY = 65;
```

```
62      const int NZ = 14;
63
64      // Pointers for holding the data
65      JHDF5* __restrict__ Ax; // Not strictly std c++0x but supported by g++
66      JHDF5* __restrict__ Ay;
67      JHDF5* __restrict__ Az;
68
69      // GSL requires all parameters to be passed through a void*
70      // create structure to hold parameters and cast to void*
71      struct dparams{
72          worker* inst;
73          double p1, p2;
74      };
75      struct dparams params;
76
77      // Wrapper function needed due to pointer differences function/member function
78      static int wrapfunc(double , const double [], double [],void*);
79      myArray* __restrict__ results;
80      void getCoeffsX(int);
81      void getCoeffsY(int);
82      void getCoeffsZ(int);
83      double getValX(double, double, double);
84      gsl_interp_accel * lookupACCx;
85      gsl_interp_accel * lookupACCy;
86      gsl_interp_accel * lookupACCz;
87
88  };
89
90  // Constructor:  allocate space to read dataset when seed() is called
91  // Initialise variables containing the grid spacing
92  worker::worker(): Ax(new JHDF5("/media/jack/D87ED1CB7ED1A294/Ar_A.hdf5","FRF_A")), Ay(new JHDF5(
"/media/jack/D87ED1CB7ED1A294/Ar_A.hdf5","FRF_A")), Az(new JHDF5("/media/jack/D87ED1CB7ED1A294/Ar_A.hdf5",
"FRF_A")), lookupACCx(gsl_interp_accel_alloc()),
93      lookupACCy(gsl_interp_accel_alloc()),lookupACCz(gsl_interp_accel_alloc()){}
94
95  // Wrapper functions for numerical differentiation.
96  // GSL protype is f(double, void*) so all args much go through
97  // a void*
98
99  // Perform the seeding
100 void worker::seed(int time){
101     results = new myArray();
102     // Read the data corresponding to t=time
103     getCoeffsX(time);
104     getCoeffsY(time);
105     getCoeffsZ(time);
106     // Set up Adams-bashforth system:
107     gsl_odeiv2_system sys = {&wrapfunc,nullptr,2,this};
108     /* GSL driver which will step system.
109     Function prototype: (const gsl_odeiv2_system * sys,
110     const gsl_odeiv2_step_type * T, const double hstart,
111     const double epsabs, const double epsrel */
112     gsl_odeiv2_driver * d =
113     gsl_odeiv2_driver_alloc_y_new (&sys, gsl_odeiv2_step_rkf45,
114                   1e-4, 1e-4, 0.0);// 1e-4 ok
115     double t,ti; // Step from t to ti during each run.
116
117     double yi[2]; // Store result in yi
118     int i,j,count;
119     for (i = 0; i < 65; ++i){
120         for (j = 0; j < 65; ++j){
121             gsl_odeiv2_driver_reset(d);
122             yi[0] = XPOS[i];
123             yi[1] = YPOS[j];
124             t = ZPOS[0];
125
```

```
126             // Store result in output array
127             results->buffer[0][i][j][0]=yi[0];
128             results->buffer[1][i][j][0]=yi[1];
129             results->buffer[2][i][j][0]=t;
130             //cout<<x[i]<<","<<y[j]<<endl; //11
131             for (count = 1; count < 50; ++count){
132                 // interate through height
133                 ti = t + 16.95306;
134                 gsl_odeiv2_driver_apply (d, &t, ti, yi); // Step
135                 if (yi[0] < minX)
136                     yi[0] = minX;
137                 else if (yi[0] > maxX)
138                     yi[0] = maxX;
139                 if (yi[1] < minY)
140                     yi[1] = minY;
141                 else if (yi[1] > maxY)
142                     yi[1] = maxY;
143 
144                 results->buffer[0][i][j][count]=yi[0];
145                 results->buffer[1][i][j][count]=yi[1];
146                 results->buffer[2][i][j][count]=ti;
147             }
148         }
149     }
150 
151     writeLock.lock();
152     results->write("seeds.hdf5","tzero",time);
153     writeLock.unlock();
154     gsl_odeiv2_driver_free (d); // Free memory
155 
156     delete results;
157 }
158 
159 // Perform numerical differentiation.
160 double worker::getBx(double xp, double yp, double zp){
161     // Range check:
162     if (xp < minX)
163         xp = minX;
164     else if (xp > maxX)
165         xp = maxX;
166     if (yp < minY)
167         yp = minY;
168     else if (yp > maxY)
169         yp = maxY;
170 
171     double result1, result2;
172 
173     int i = gsl_interp_accel_find (lookupACCz,ZPOS,14,zp);
174     int j = gsl_interp_accel_find (lookupACCy,YPOS,64,yp);
175     int k = gsl_interp_accel_find (lookupACCx,XPOS,64,xp);
176     result1 = tricubic_eval(az[i][j][k],(xp-XPOS[k])/delX,(yp-YPOS[j])/delY,(zp-ZPOS[i])/delZ,0,1,0)/delY;
177     result2 = tricubic_eval(ay[i][j][k],(xp-XPOS[k])/delX,(yp-YPOS[j])/delY,(zp-ZPOS[i])/delZ,0,0,1)/delZ;
178 
179     return result1-result2;
180 }
181 
182 double worker::getBy(double xp, double yp, double zp){
183     // Range check:
184     if (xp < minX)
185         xp = minX;
186     else if (xp > maxX)
187         xp = maxX;
188     if (yp < minY)
189         yp = minY;
190     else if (yp > maxY)
191         yp = maxY;
```

```
192
193     double result1, result2;
194     int i = gsl_interp_accel_find (lookupACCz,ZPOS,14,zp);
195     int j = gsl_interp_accel_find (lookupACCy,YPOS,64,yp);
196     int k = gsl_interp_accel_find (lookupACCx,XPOS,64,xp);
197     result1 = tricubic_eval(ax[i][j][k],(xp-XPOS[k])/delX,(yp-YPOS[j])/delY,(zp-ZPOS[i])/delZ,0,0,1)/delZ;
198     result2 = tricubic_eval(az[i][j][k],(xp-XPOS[k])/delX,(yp-YPOS[j])/delY,(zp-ZPOS[i])/delZ,1,0,0)/delX;
199
200     return result1-result2;
201 }
202
203 double worker::getBz(double xp, double yp, double zp){
204     // Range check:
205     if (xp < minX)
206         xp = minX;
207     else if (xp > maxX)
208         xp = maxX;
209     if (yp < minY)
210         yp = minY;
211     else if (yp > maxY)
212         yp = maxY;
213
214     double result1, result2;
215     int i = gsl_interp_accel_find (lookupACCz,ZPOS,14,zp);
216     int j = gsl_interp_accel_find (lookupACCy,YPOS,64,yp);
217     int k = gsl_interp_accel_find (lookupACCx,XPOS,64,xp);
218     result1 = tricubic_eval(ay[i][j][k],(xp-XPOS[k])/delX,(yp-YPOS[j])/delY,(zp-ZPOS[i])/delZ,1,0,0)/delX;
219     result2 = tricubic_eval(ax[i][j][k],(xp-XPOS[k])/delX,(yp-YPOS[j])/delY,(zp-ZPOS[i])/delZ,0,1,0)/delY;
220
221     return result1-result2;
222 }
223
224 // Wrapper function for integrator
225 inline int worker::wrapfunc(double t, const double yi[], double f[],void *params){
226     worker* inst = (worker* )params;
227     f[0] = inst->getBx(yi[0],yi[1],t)/inst->getBz(yi[0],yi[1],t);
228     f[1] = inst->getBy(yi[0],yi[1],t)/inst->getBz(yi[0],yi[1],t);
229     return GSL_SUCCESS;
230 }
231
232 void worker::getCoeffsX(int time){
233     double fval[NZ][NY][NX];
234     double dfdxval[NZ][NY][NX];
235     double dfdyval[NZ][NY][NX];
236     double dfdzval[NZ][NY][NX];
237     double d2fdxdyval[NZ][NY][NX];
238     double d2fdxdzval[NZ][NY][NX];
239     double d2fdydzval[NZ][NY][NX];
240     double d3fdxdydzval[NZ][NY][NX];
241
242     // Need variable to hold columns of the arrays
243     double yvals[NZ][NX][NY];
244     double zvals[NX][NY][NZ];
245     // Loop indices
246     int i,j,k,l;
247     readLock.lock();
248     Ax->initialise(0,time);
249     readLock.unlock();
250     // use a gsl cubic spline with natural bcs.
251     const gsl_interp_type *t = gsl_interp_cspline;
252
253     // Single accelerator, change later!
254     gsl_interp_accel * myACC = gsl_interp_accel_alloc();
255     // Hold the splines:
256     gsl_interp* xInterp[NZ][NY];
257     gsl_interp* yInterp[NZ][NX];
```

```
258     gsl_interp* zInterp[NY][NX];
259     gsl_interp* dxdyInterp[NZ][NX];
260     gsl_interp* dxdzInterp[NY][NX];
261     gsl_interp* dydzInterp[NY][NX];
262     gsl_interp* dxdydzInterp[NY][NX];
263
264     // extract columns:
265     for (i = 0; i < NZ; ++i){
266         for (j = 0; j < NY; ++j){
267             for (k = 0; k < NX; ++k){
268                 yvals[i][j][k] = Ax->buffer[i][k][j];
269                 zvals[j][k][i] = Ax->buffer[i][j][k];
270             }
271         }
272     }
273
274     for (i = 0; i < NZ; ++i){
275         for (j = 0; j < NX; ++j){
276             xInterp[i][j] = gsl_interp_alloc(t,NX);
277             yInterp[i][j] = gsl_interp_alloc(t,NY);
278             gsl_interp_init(xInterp[i][j],XPOS,Ax->buffer[i][j],NX);
279             gsl_interp_init(yInterp[i][j],YPOS,yvals[i][j],NY);
280         }
281     }
282
283     for (i = 0; i < NY; ++i){
284         for (j = 0; j < NX; ++j){
285             zInterp[i][j] = gsl_interp_alloc(t,NZ);
286             gsl_interp_init(zInterp[i][j],ZPOS,zvals[i][j],NZ);
287         }
288     }
289
290     for (i = 0; i < NZ; ++i){
291         for (j = 0; j < NY; ++j){
292             for (k = 0; k < NX; ++k){
293                 dfdxval[i][j][k] = gsl_interp_eval_deriv(xInterp[i][j],XPOS,Ax->buffer[i][j],XPOS[k],myACC
);
294                 gsl_interp_accel_reset(myACC);
295                 dfdyval[i][j][k] = gsl_interp_eval_deriv(yInterp[i][k],YPOS,yvals[i][k],YPOS[j],myACC);
296                 gsl_interp_accel_reset(myACC);
297                 dfdzval[i][j][k] = gsl_interp_eval_deriv(zInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC);
298                 gsl_interp_accel_reset(myACC);
299             }
300         }
301     }
302     // Now for second derivatives:
303     for (i = 0; i < NZ; ++i){
304         for (j = 0; j < NY; ++j){
305             for (k = 0; k < NX; ++k){
306                 yvals[i][j][k] = dfdxval[i][k][j];
307                 zvals[j][k][i] = dfdxval[i][j][k];
308             }
309         }
310     }
311     for (i = 0; i < NZ; ++i){
312         for (j = 0; j < NY; ++j){
313             dxdyInterp[i][j] = gsl_interp_alloc(t,NY);
314             gsl_interp_init(dxdyInterp[i][j],YPOS,yvals[i][j],NY);
315         }
316     }
317
318     for (i = 0; i < NY; ++i){
319         for (j = 0; j < NX; ++j){
320             dxdzInterp[i][j] = gsl_interp_alloc(t,NZ);
321             gsl_interp_init(dxdzInterp[i][j],ZPOS,zvals[i][j],NZ);
322         }
```

```
323     }
324
325     for (i = 0; i < NZ; ++i){
326         for (j = 0; j < NY; ++j){
327             for (k = 0; k < NX; ++k){
328                 d2fdxdyval[i][j][k] = gsl_interp_eval_deriv(dxdyInterp[i][j],YPOS,yvals[i][j],YPOS[k],myACC
);
329                 gsl_interp_accel_reset(myACC);
330                 d2fdxdzval[i][j][k] = gsl_interp_eval_deriv(dxdzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC
);
331                 gsl_interp_accel_reset(myACC);
332             }
333         }
334     }
335
336     for (i = 0; i < NZ; ++i){
337         for (j = 0; j < NY; ++j){
338             for (k = 0; k < NX; ++k){
339                 zvals[j][k][i] = dfdyval[i][j][k];
340             }
341         }
342     }
343
344     for (i = 0; i < NY; ++i){
345         for (j = 0; j < NX; ++j){
346             dydzInterp[i][j] = gsl_interp_alloc(t,NZ);
347             gsl_interp_init(dydzInterp[i][j],ZPOS,zvals[i][j],NZ);
348         }
349     }
350
351     for (i = 0; i < NZ; ++i){
352         for (j = 0; j < NY; ++j){
353             for (k = 0; k < NX; ++k){
354                 d2fdydzval[i][j][k] = gsl_interp_eval_deriv(dydzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC
);
355                 gsl_interp_accel_reset(myACC);
356             }
357         }
358     }
359
360     // Now the one third derivative:
361     for (i = 0; i < NZ; ++i){
362         for (j = 0; j < NY; ++j){
363             for (k = 0; k < NX; ++k){
364                 zvals[j][k][i] = d2fdxdyval[i][j][k];
365             }
366         }
367     }
368
369     for (i = 0; i < NY; ++i){
370         for (j = 0; j < NX; ++j){
371             dxdydzInterp[i][j] = gsl_interp_alloc(t,NZ);
372             gsl_interp_init(dxdydzInterp[i][j],ZPOS,zvals[i][j],NZ);
373         }
374     }
375
376     for (i = 0; i < NZ; ++i){
377         for (j = 0; j < NY; ++j){
378             for (k = 0; k < NX; ++k){
379                 d3fdxdydzval[i][j][k] = gsl_interp_eval_deriv(dxdydzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],
myACC);
380                 gsl_interp_accel_reset(myACC);
381                 fval[i][j][k] = Ax->buffer[i][j][k];
382             }
383         }
384     }
```

```
385
386     // Free memory:
387     for (i = 0; i < NZ; ++i){
388         for (j = 0; j < NY; ++j){
389             gsl_interp_free(xInterp[i][j]);
390             gsl_interp_free(yInterp[i][j]);
391             gsl_interp_free(dxdyInterp[i][j]);
392         }
393     }
394
395     for (i = 0; i < NY; ++i){
396         for (j = 0; j < NX; ++j){
397             gsl_interp_free(zInterp[i][j]);
398             gsl_interp_free(dxdzInterp[i][j]);
399             gsl_interp_free(dydzInterp[i][j]);
400             gsl_interp_free(dxdydzInterp[i][j]);
401         }
402     }
403
404     gsl_interp_accel_free(myACC);
405
406     // Calculate coefficients:
407     ///The order of the points is as follow:
408     ///  0: x=0; y=0; z=0;
409     ///  1: x=1; y=0; z=0;
410     ///  2: x=0; y=1; z=0;
411     ///  3: x=1; y=1; z=0;
412     ///  4: x=0; y=0; z=1;
413     ///  5: x=1; y=0; z=1;
414     ///  6: x=0; y=1; z=1;
415     ///  7: x=1; y=1; z=1;
416     // Array to hold coefficients:
417
418     double Pfval[8];
419     double Pdfdxval[8];
420     double Pdfdyval[8];
421     double Pdfdzval[8];
422     double Pd2fdxdyval[8];
423     double Pd2fdxdzval[8];
424     double Pd2fdydzval[8];
425     double Pd3fdxdydzval[8];
426
427     for (i = 0; i < NZ - 1; ++i){
428         for (j = 0; j < NY - 1; ++j){
429             for (k = 0; k < NX -1; ++k){
430
431                 double Pfval[8] = {fval[i][j][k],fval[i][j][k+1],fval[i][j+1][k],fval[i][j+1][k+1],fval[i+1
][j][k],fval[i+1][j][k+1],fval[i+1][j+1][k],fval[i+1][j+1][k+1]};
432                 double Pdfdxval[8] = {dfdxval[i][j][k],dfdxval[i][j][k+1],dfdxval[i][j+1][k],dfdxval[i][j+1
][k+1],dfdxval[i+1][j][k],dfdxval[i+1][j][k+1],dfdxval[i+1][j+1][k],dfdxval[i+1][j+1][k+1]};
433                 double Pdfdyval[8] = {dfdyval[i][j][k],dfdyval[i][j][k+1],dfdyval[i][j+1][k],dfdyval[i][j+1
][k+1],dfdyval[i+1][j][k],dfdyval[i+1][j][k+1],dfdyval[i+1][j+1][k],dfdyval[i+1][j+1][k+1]};
434                 double Pdfdzval[8] = {dfdzval[i][j][k],dfdzval[i][j][k+1],dfdzval[i][j+1][k],dfdzval[i][j+1
][k+1],dfdzval[i+1][j][k],dfdzval[i+1][j][k+1],dfdzval[i+1][j+1][k],dfdzval[i+1][j+1][k+1]};
435                 double Pd2fdxdyval[8] = {d2fdxdyval[i][j][k],d2fdxdyval[i][j][k+1],d2fdxdyval[i][j+1][k],
d2fdxdyval[i][j+1][k+1],d2fdxdyval[i+1][j][k],d2fdxdyval[i+1][j][k+1],d2fdxdyval[i+1][j+1][k],d2fdxdyval[i+1][j+
1][k+1]};
436                 double Pd2fdxdzval[8] = {d2fdxdzval[i][j][k],d2fdxdzval[i][j][k+1],d2fdxdzval[i][j+1][k],
d2fdxdzval[i][j+1][k+1],d2fdxdzval[i+1][j][k],d2fdxdzval[i+1][j][k+1],d2fdxdzval[i+1][j+1][k],d2fdxdzval[i+1][j+
1][k+1]};
437                 double Pd2fdydzval[8] = {d2fdydzval[i][j][k],d2fdydzval[i][j][k+1],d2fdydzval[i][j+1][k],
d2fdydzval[i][j+1][k+1],d2fdydzval[i+1][j][k],d2fdydzval[i+1][j][k+1],d2fdydzval[i+1][j+1][k],d2fdydzval[i+1][j+
1][k+1]};
438                 double Pd3fdxdydzval[8] = {d3fdxdydzval[i][j][k],d3fdxdydzval[i][j][k+1],d3fdxdydzval[i][j+
1][k],d3fdxdydzval[i][j+1][k+1],d3fdxdydzval[i+1][j][k],d3fdxdydzval[i+1][j][k+1],d3fdxdydzval[i+1][j+1][k],
d3fdxdydzval[i+1][j+1][k+1]};
```

```
439
440                 // Adjusting values as box size is not unity.
441                 for (l = 0; l < 8; ++l) {
442                     Pfval[l]*=1.0;
443                     Pdfdxval[l]*=delX;
444                     Pdfdyval[l]*=delY;
445                     Pdfdzval[l]*=delZ;
446                     Pd2fdxdyval[l]*=delX*delY;
447                     Pd2fdxdzval[l]*=delX*delZ;
448                     Pd2fdydzval[l]*=delY*delZ;
449                     Pd3fdxdydzval[l]*=delX*delY*delZ;
450                 }
451                 tricubic_get_coeff(ax[i][j][k],Pfval,Pdfdxval,Pdfdyval,Pdfdzval,Pd2fdxdyval,Pd2fdxdzval,
Pd2fdydzval,Pd3fdxdydzval);
452             }
453         }
454     }
455 }
456
457 void worker::getCoeffsY(int time){
458     double fval[NZ][NY][NX];
459     double dfdxval[NZ][NY][NX];
460     double dfdyval[NZ][NY][NX];
461     double dfdzval[NZ][NY][NX];
462     double d2fdxdyval[NZ][NY][NX];
463     double d2fdxdzval[NZ][NY][NX];
464     double d2fdydzval[NZ][NY][NX];
465     double d3fdxdydzval[NZ][NY][NX];
466
467     // Need variable to hold columns of the arrays
468     double yvals[NZ][NX][NY];
469     double zvals[NX][NY][NZ];
470     // Loop indices
471     int i,j,k,l;
472     readLock.lock();
473     Ay->initialise(1,time);
474     readLock.unlock();
475     // use a gsl cubic spline with natural bcs.
476     const gsl_interp_type *t = gsl_interp_cspline;
477
478     // Single accelerator, change later!
479     gsl_interp_accel * myACC = gsl_interp_accel_alloc();
480     // Hold the splines:
481     gsl_interp* xInterp[NZ][NY];
482     gsl_interp* yInterp[NZ][NX];
483     gsl_interp* zInterp[NY][NX];
484     gsl_interp* dxdyInterp[NZ][NX];
485     gsl_interp* dxdzInterp[NY][NX];
486     gsl_interp* dydzInterp[NY][NX];
487     gsl_interp* dxdydzInterp[NY][NX];
488
489     // extract columns:
490     for (i = 0; i < NZ; ++i){
491         for (j = 0; j < NY; ++j){
492             for (k = 0; k < NX; ++k){
493                 yvals[i][j][k] = Ay->buffer[i][k][j];
494                 zvals[j][k][i] = Ay->buffer[i][j][k];
495             }
496         }
497     }
498
499     for (i = 0; i < NZ; ++i){
500         for (j = 0; j < NX; ++j){
501             xInterp[i][j] = gsl_interp_alloc(t,NX);
502             yInterp[i][j] = gsl_interp_alloc(t,NY);
503             gsl_interp_init(xInterp[i][j],XPOS,Ay->buffer[i][j],NX);
```

```
504             gsl_interp_init(yInterp[i][j],YPOS,yvals[i][j],NY);
505         }
506     }
507 
508     for (i = 0; i < NY; ++i){
509         for (j = 0; j < NX; ++j){
510             zInterp[i][j] = gsl_interp_alloc(t,NZ);
511             gsl_interp_init(zInterp[i][j],ZPOS,zvals[i][j],NZ);
512         }
513     }
514 
515     for (i = 0; i < NZ; ++i){
516         for (j = 0; j < NY; ++j){
517             for (k = 0; k < NX; ++k){
518                 dfdxval[i][j][k] = gsl_interp_eval_deriv(xInterp[i][j],XPOS,Ay->buffer[i][j],XPOS[k],myACC
);
519                 gsl_interp_accel_reset(myACC);
520                 dfdyval[i][j][k] = gsl_interp_eval_deriv(yInterp[i][k],YPOS,yvals[i][k],YPOS[j],myACC);
521                 gsl_interp_accel_reset(myACC);
522                 dfdzval[i][j][k] = gsl_interp_eval_deriv(zInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC);
523                 gsl_interp_accel_reset(myACC);
524             }
525         }
526     }
527     // Now for second derivatives:
528     for (i = 0; i < NZ; ++i){
529         for (j = 0; j < NY; ++j){
530             for (k = 0; k < NX; ++k){
531                 yvals[i][j][k] = dfdxval[i][k][j];
532                 zvals[j][k][i] = dfdxval[i][j][k];
533             }
534         }
535     }
536     for (i = 0; i < NZ; ++i){
537         for (j = 0; j < NY; ++j){
538             dxdyInterp[i][j] = gsl_interp_alloc(t,NY);
539             gsl_interp_init(dxdyInterp[i][j],YPOS,yvals[i][j],NY);
540         }
541     }
542 
543     for (i = 0; i < NY; ++i){
544         for (j = 0; j < NX; ++j){
545             dxdzInterp[i][j] = gsl_interp_alloc(t,NZ);
546             gsl_interp_init(dxdzInterp[i][j],ZPOS,zvals[i][j],NZ);
547         }
548     }
549 
550     for (i = 0; i < NZ; ++i){
551         for (j = 0; j < NY; ++j){
552             for (k = 0; k < NX; ++k){
553                 d2fdxdyval[i][j][k] = gsl_interp_eval_deriv(dxdyInterp[i][j],YPOS,yvals[i][j],YPOS[k],myACC
);
554                 gsl_interp_accel_reset(myACC);
555                 d2fdxdzval[i][j][k] = gsl_interp_eval_deriv(dxdzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC
);
556                 gsl_interp_accel_reset(myACC);
557             }
558         }
559     }
560 
561     for (i = 0; i < NZ; ++i){
562         for (j = 0; j < NY; ++j){
563             for (k = 0; k < NX; ++k){
564                 zvals[j][k][i] = dfdyval[i][j][k];
565             }
566         }
```

```
567     }
568
569     for (i = 0; i < NY; ++i){
570         for (j = 0; j < NX; ++j){
571             dydzInterp[i][j] = gsl_interp_alloc(t,NZ);
572             gsl_interp_init(dydzInterp[i][j],ZPOS,zvals[i][j],NZ);
573         }
574     }
575
576     for (i = 0; i < NZ; ++i){
577         for (j = 0; j < NY; ++j){
578             for (k = 0; k < NX; ++k){
579                 d2fdydzval[i][j][k] = gsl_interp_eval_deriv(dydzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC
);
580                 gsl_interp_accel_reset(myACC);
581             }
582         }
583     }
584
585     // Now the one third derivative:
586     for (i = 0; i < NZ; ++i){
587         for (j = 0; j < NY; ++j){
588             for (k = 0; k < NX; ++k){
589                 zvals[j][k][i] = d2fdxdyval[i][j][k];
590             }
591         }
592     }
593
594     for (i = 0; i < NY; ++i){
595         for (j = 0; j < NX; ++j){
596             dxdydzInterp[i][j] = gsl_interp_alloc(t,NZ);
597             gsl_interp_init(dxdydzInterp[i][j],ZPOS,zvals[i][j],NZ);
598         }
599     }
600
601     for (i = 0; i < NZ; ++i){
602         for (j = 0; j < NY; ++j){
603             for (k = 0; k < NX; ++k){
604                 d3fdxdydzval[i][j][k] = gsl_interp_eval_deriv(dxdydzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],
myACC);
605                 gsl_interp_accel_reset(myACC);
606                 fval[i][j][k] = Ay->buffer[i][j][k];
607             }
608         }
609     }
610
611     // Free memory:
612     for (i = 0; i < NZ; ++i){
613         for (j = 0; j < NY; ++j){
614             gsl_interp_free(xInterp[i][j]);
615             gsl_interp_free(yInterp[i][j]);
616             gsl_interp_free(dxdyInterp[i][j]);
617         }
618     }
619
620     for (i = 0; i < NY; ++i){
621         for (j = 0; j < NX; ++j){
622             gsl_interp_free(zInterp[i][j]);
623             gsl_interp_free(dxdzInterp[i][j]);
624             gsl_interp_free(dydzInterp[i][j]);
625             gsl_interp_free(dxdydzInterp[i][j]);
626         }
627     }
628
629     gsl_interp_accel_free(myACC);
630
```

```
631       // Calculate coefficients:
632       ///The order of the points is as follow:
633       ///  0: x=0; y=0; z=0;
634       ///  1: x=1; y=0; z=0;
635       ///  2: x=0; y=1; z=0;
636       ///  3: x=1; y=1; z=0;
637       ///  4: x=0; y=0; z=1;
638       ///  5: x=1; y=0; z=1;
639       ///  6: x=0; y=1; z=1;
640       ///  7: x=1; y=1; z=1;
641       // Array to hold coefficients:
642
643       double Pfval[8];
644       double Pdfdxval[8];
645       double Pdfdyval[8];
646       double Pdfdzval[8];
647       double Pd2fdxdyval[8];
648       double Pd2fdxdzval[8];
649       double Pd2fdydzval[8];
650       double Pd3fdxdydzval[8];
651
652       for (i = 0; i < NZ - 1; ++i){
653           for (j = 0; j < NY - 1; ++j){
654               for (k = 0; k < NX -1; ++k){
655
656                   double Pfval[8] = {fval[i][j][k],fval[i][j][k+1],fval[i][j+1][k],fval[i][j+1][k+1],fval[i+1
][j][k],fval[i+1][j][k+1],fval[i+1][j+1][k],fval[i+1][j+1][k+1]};
657                   double Pdfdxval[8] = {dfdxval[i][j][k],dfdxval[i][j][k+1],dfdxval[i][j+1][k],dfdxval[i][j+1
][k+1],dfdxval[i+1][j][k],dfdxval[i+1][j][k+1],dfdxval[i+1][j+1][k],dfdxval[i+1][j+1][k+1]};
658                   double Pdfdyval[8] = {dfdyval[i][j][k],dfdyval[i][j][k+1],dfdyval[i][j+1][k],dfdyval[i][j+1
][k+1],dfdyval[i+1][j][k],dfdyval[i+1][j][k+1],dfdyval[i+1][j+1][k],dfdyval[i+1][j+1][k+1]};
659                   double Pdfdzval[8] = {dfdzval[i][j][k],dfdzval[i][j][k+1],dfdzval[i][j+1][k],dfdzval[i][j+1
][k+1],dfdzval[i+1][j][k],dfdzval[i+1][j][k+1],dfdzval[i+1][j+1][k],dfdzval[i+1][j+1][k+1]};
660                   double Pd2fdxdyval[8] = {d2fdxdyval[i][j][k],d2fdxdyval[i][j][k+1],d2fdxdyval[i][j+1][k],
d2fdxdyval[i][j+1][k+1],d2fdxdyval[i+1][j][k],d2fdxdyval[i+1][j][k+1],d2fdxdyval[i+1][j+1][k],d2fdxdyval[i+1][j+
1][k+1]};
661                   double Pd2fdxdzval[8] = {d2fdxdzval[i][j][k],d2fdxdzval[i][j][k+1],d2fdxdzval[i][j+1][k],
d2fdxdzval[i][j+1][k+1],d2fdxdzval[i+1][j][k],d2fdxdzval[i+1][j][k+1],d2fdxdzval[i+1][j+1][k],d2fdxdzval[i+1][j+
1][k+1]};
662                   double Pd2fdydzval[8] = {d2fdydzval[i][j][k],d2fdydzval[i][j][k+1],d2fdydzval[i][j+1][k],
d2fdydzval[i][j+1][k+1],d2fdydzval[i+1][j][k],d2fdydzval[i+1][j][k+1],d2fdydzval[i+1][j+1][k],d2fdydzval[i+1][j+
1][k+1]};
663                   double Pd3fdxdydzval[8] = {d3fdxdydzval[i][j][k],d3fdxdydzval[i][j][k+1],d3fdxdydzval[i][j+
1][k],d3fdxdydzval[i][j+1][k+1],d3fdxdydzval[i+1][j][k],d3fdxdydzval[i+1][j][k+1],d3fdxdydzval[i+1][j+1][k],
d3fdxdydzval[i+1][j+1][k+1]};
664
665                   // Adjusting values as box size is not unity.
666                   for (l = 0; l < 8; l++) {
667                       Pfval[l]*=1.0;
668                       Pdfdxval[l]*=delX;
669                       Pdfdyval[l]*=delY;
670                       Pdfdzval[l]*=delZ;
671                       Pd2fdxdyval[l]*=delX*delY;
672                       Pd2fdxdzval[l]*=delX*delZ;
673                       Pd2fdydzval[l]*=delY*delZ;
674                       Pd3fdxdydzval[l]*=delX*delY*delZ;
675                   }
676                   tricubic_get_coeff(ay[i][j][k],Pfval,Pdfdxval,Pdfdyval,Pdfdzval,Pd2fdxdyval,Pd2fdxdzval,
Pd2fdydzval,Pd3fdxdydzval);
677               }
678           }
679       }
680   }
681
682   void worker::getCoeffsZ(int time){
683       double fval[NZ][NY][NX];
```

```
684     double dfdxval[NZ][NY][NX];
685     double dfdyval[NZ][NY][NX];
686     double dfdzval[NZ][NY][NX];
687     double d2fdxdyval[NZ][NY][NX];
688     double d2fdxdzval[NZ][NY][NX];
689     double d2fdydzval[NZ][NY][NX];
690     double d3fdxdydzval[NZ][NY][NX];
691 
692     // Need variable to hold columns of the arrays
693     double yvals[NZ][NX][NY];
694     double zvals[NX][NY][NZ];
695     // Loop indices
696     int i,j,k,l;
697     readLock.lock();
698     Az->initialise(2,time);
699     readLock.unlock();
700     // use a gsl cubic spline with natural bcs.
701     const gsl_interp_type *t = gsl_interp_cspline;
702 
703     // Single accelerator, change later!
704     gsl_interp_accel * myACC = gsl_interp_accel_alloc();
705     // Hold the splines:
706     gsl_interp* xInterp[NZ][NY];
707     gsl_interp* yInterp[NZ][NX];
708     gsl_interp* zInterp[NY][NX];
709     gsl_interp* dxdyInterp[NZ][NX];
710     gsl_interp* dxdzInterp[NY][NX];
711     gsl_interp* dydzInterp[NY][NX];
712     gsl_interp* dxdydzInterp[NY][NX];
713 
714     // extract columns:
715     for (i = 0; i < NZ; ++i){
716         for (j = 0; j < NY; ++j){
717             for (k = 0; k < NX; ++k){
718                 yvals[i][j][k] = Az->buffer[i][k][j];
719                 zvals[j][k][i] = Az->buffer[i][j][k];
720             }
721         }
722     }
723 
724     for (i = 0; i < NZ; ++i){
725         for (j = 0; j < NX; ++j){
726             xInterp[i][j] = gsl_interp_alloc(t,NX);
727             yInterp[i][j] = gsl_interp_alloc(t,NY);
728             gsl_interp_init(xInterp[i][j],XPOS,Az->buffer[i][j],NX);
729             gsl_interp_init(yInterp[i][j],YPOS,yvals[i][j],NY);
730         }
731     }
732 
733     for (i = 0; i < NY; ++i){
734         for (j = 0; j < NX; ++j){
735             zInterp[i][j] = gsl_interp_alloc(t,NZ);
736             gsl_interp_init(zInterp[i][j],ZPOS,zvals[i][j],NZ);
737         }
738     }
739 
740     for (i = 0; i < NZ; ++i){
741         for (j = 0; j < NY; ++j){
742             for (k = 0; k < NX; ++k){
743                 dfdxval[i][j][k] = gsl_interp_eval_deriv(xInterp[i][j],XPOS,Az->buffer[i][j],XPOS[k],myACC
);
744                 gsl_interp_accel_reset(myACC);
745                 dfdyval[i][j][k] = gsl_interp_eval_deriv(yInterp[i][k],YPOS,yvals[i][k],YPOS[j],myACC);
746                 gsl_interp_accel_reset(myACC);
747                 dfdzval[i][j][k] = gsl_interp_eval_deriv(zInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC);
748                 gsl_interp_accel_reset(myACC);
```

```
749             }
750         }
751     }
752     // Now for second derivatives:
753     for (i = 0; i < NZ; ++i){
754         for (j = 0; j < NY; ++j){
755             for (k = 0; k < NX; ++k){
756                 yvals[i][j][k] = dfdxval[i][k][j];
757                 zvals[j][k][i] = dfdxval[i][j][k];
758             }
759         }
760     }
761     for (i = 0; i < NZ; ++i){
762         for (j = 0; j < NY; ++j){
763             dxdyInterp[i][j] = gsl_interp_alloc(t,NY);
764             gsl_interp_init(dxdyInterp[i][j],YPOS,yvals[i][j],NY);
765         }
766     }
767 
768     for (i = 0; i < NY; ++i){
769         for (j = 0; j < NX; ++j){
770             dxdzInterp[i][j] = gsl_interp_alloc(t,NZ);
771             gsl_interp_init(dxdzInterp[i][j],ZPOS,zvals[i][j],NZ);
772         }
773     }
774 
775     for (i = 0; i < NZ; ++i){
776         for (j = 0; j < NY; ++j){
777             for (k = 0; k < NX; ++k){
778                 d2fdxdyval[i][j][k] = gsl_interp_eval_deriv(dxdyInterp[i][j],YPOS,yvals[i][j],YPOS[k],myACC
);
779                 gsl_interp_accel_reset(myACC);
780                 d2fdxdzval[i][j][k] = gsl_interp_eval_deriv(dxdzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC
);
781                 gsl_interp_accel_reset(myACC);
782             }
783         }
784     }
785 
786     for (i = 0; i < NZ; ++i){
787         for (j = 0; j < NY; ++j){
788             for (k = 0; k < NX; ++k){
789                 zvals[j][k][i] = dfdyval[i][j][k];
790             }
791         }
792     }
793 
794     for (i = 0; i < NY; ++i){
795         for (j = 0; j < NX; ++j){
796             dydzInterp[i][j] = gsl_interp_alloc(t,NZ);
797             gsl_interp_init(dydzInterp[i][j],ZPOS,zvals[i][j],NZ);
798         }
799     }
800 
801     for (i = 0; i < NZ; ++i){
802         for (j = 0; j < NY; ++j){
803             for (k = 0; k < NX; ++k){
804                 d2fdydzval[i][j][k] = gsl_interp_eval_deriv(dydzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],myACC
);
805                 gsl_interp_accel_reset(myACC);
806             }
807         }
808     }
809 
810     // Now the one third derivative:
811     for (i = 0; i < NZ; ++i){
```

```
812             for (j = 0; j < NY; ++j){
813                 for (k = 0; k < NX; ++k){
814                     zvals[j][k][i] = d2fdxdyval[i][j][k];
815                 }
816             }
817         }
818 
819         for (i = 0; i < NY; ++i){
820             for (j = 0; j < NX; ++j){
821                 dxdydzInterp[i][j] = gsl_interp_alloc(t,NZ);
822                 gsl_interp_init(dxdydzInterp[i][j],ZPOS,zvals[i][j],NZ);
823             }
824         }
825 
826         for (i = 0; i < NZ; ++i){
827             for (j = 0; j < NY; ++j){
828                 for (k = 0; k < NX; ++k){
829                     d3fdxdydzval[i][j][k] = gsl_interp_eval_deriv(dxdydzInterp[j][k],ZPOS,zvals[j][k],ZPOS[i],
myACC);
830                     gsl_interp_accel_reset(myACC);
831                     fval[i][j][k] = Az->buffer[i][j][k];
832                 }
833             }
834         }
835 
836         // Free memory:
837         for (i = 0; i < NZ; ++i){
838             for (j = 0; j < NY; ++j){
839                 gsl_interp_free(xInterp[i][j]);
840                 gsl_interp_free(yInterp[i][j]);
841                 gsl_interp_free(dxdyInterp[i][j]);
842             }
843         }
844 
845         for (i = 0; i < NY; ++i){
846             for (j = 0; j < NX; ++j){
847                 gsl_interp_free(zInterp[i][j]);
848                 gsl_interp_free(dxdzInterp[i][j]);
849                 gsl_interp_free(dydzInterp[i][j]);
850                 gsl_interp_free(dxdydzInterp[i][j]);
851             }
852         }
853 
854         gsl_interp_accel_free(myACC);
855 
856         // Calculate coefficients:
857         ///The order of the points is as follow:
858         ///  0: x=0; y=0; z=0;
859         ///  1: x=1; y=0; z=0;
860         ///  2: x=0; y=1; z=0;
861         ///  3: x=1; y=1; z=0;
862         ///  4: x=0; y=0; z=1;
863         ///  5: x=1; y=0; z=1;
864         ///  6: x=0; y=1; z=1;
865         ///  7: x=1; y=1; z=1;
866         // Array to hold coefficients:
867 
868         double Pfval[8];
869         double Pdfdxval[8];
870         double Pdfdyval[8];
871         double Pdfdzval[8];
872         double Pd2fdxdyval[8];
873         double Pd2fdxdzval[8];
874         double Pd2fdydzval[8];
875         double Pd3fdxdydzval[8];
876 
```

```
877      for (i = 0; i < NZ - 1; ++i){
878          for (j = 0; j < NY - 1; ++j){
879              for (k = 0; k < NX -1; ++k){
880
881                  double Pfval[8] = {fval[i][j][k],fval[i][j][k+1],fval[i][j+1][k],fval[i][j+1][k+1],fval[i+1
][j][k],fval[i+1][j][k+1],fval[i+1][j+1][k],fval[i+1][j+1][k+1]};
882                  double Pdfdxval[8] = {dfdxval[i][j][k],dfdxval[i][j][k+1],dfdxval[i][j+1][k],dfdxval[i][j+1
][k+1],dfdxval[i+1][j][k],dfdxval[i+1][j][k+1],dfdxval[i+1][j+1][k],dfdxval[i+1][j+1][k+1]};
883                  double Pdfdyval[8] = {dfdyval[i][j][k],dfdyval[i][j][k+1],dfdyval[i][j+1][k],dfdyval[i][j+1
][k+1],dfdyval[i+1][j][k],dfdyval[i+1][j][k+1],dfdyval[i+1][j+1][k],dfdyval[i+1][j+1][k+1]};
884                  double Pdfdzval[8] = {dfdzval[i][j][k],dfdzval[i][j][k+1],dfdzval[i][j+1][k],dfdzval[i][j+1
][k+1],dfdzval[i+1][j][k],dfdzval[i+1][j][k+1],dfdzval[i+1][j+1][k],dfdzval[i+1][j+1][k+1]};
885                  double Pd2fdxdyval[8] = {d2fdxdyval[i][j][k],d2fdxdyval[i][j][k+1],d2fdxdyval[i][j+1][k],
d2fdxdyval[i][j+1][k+1],d2fdxdyval[i+1][j][k],d2fdxdyval[i+1][j][k+1],d2fdxdyval[i+1][j+1][k],d2fdxdyval[i+1][j+
1][k+1]};
886                  double Pd2fdxdzval[8] = {d2fdxdzval[i][j][k],d2fdxdzval[i][j][k+1],d2fdxdzval[i][j+1][k],
d2fdxdzval[i][j+1][k+1],d2fdxdzval[i+1][j][k],d2fdxdzval[i+1][j][k+1],d2fdxdzval[i+1][j+1][k],d2fdxdzval[i+1][j+
1][k+1]};
887                  double Pd2fdydzval[8] = {d2fdydzval[i][j][k],d2fdydzval[i][j][k+1],d2fdydzval[i][j+1][k],
d2fdydzval[i][j+1][k+1],d2fdydzval[i+1][j][k],d2fdydzval[i+1][j][k+1],d2fdydzval[i+1][j+1][k],d2fdydzval[i+1][j+
1][k+1]};
888                  double Pd3fdxdydzval[8] = {d3fdxdydzval[i][j][k],d3fdxdydzval[i][j][k+1],d3fdxdydzval[i][j+
1][k],d3fdxdydzval[i][j+1][k+1],d3fdxdydzval[i+1][j][k],d3fdxdydzval[i+1][j][k+1],d3fdxdydzval[i+1][j+1][k],
d3fdxdydzval[i+1][j+1][k+1]};
889
890                  // Adjusting values as box size is not unity.
891                  for (l = 0; l < 8; l++) {
892                      Pfval[l]*=1.0;
893                      Pdfdxval[l]*=delX;
894                      Pdfdyval[l]*=delY;
895                      Pdfdzval[l]*=delZ;
896                      Pd2fdxdyval[l]*=delX*delY;
897                      Pd2fdxdzval[l]*=delX*delZ;
898                      Pd2fdydzval[l]*=delY*delZ;
899                      Pd3fdxdydzval[l]*=delX*delY*delZ;
900                  }
901                  tricubic_get_coeff(az[i][j][k],Pfval,Pdfdxval,Pdfdyval,Pdfdzval,Pd2fdxdyval,Pd2fdxdzval,
Pd2fdydzval,Pd3fdxdydzval);
902              }
903          }
904      }
905  }
906
907  int main(){
908
909      myArray* __restrict__ filer = new myArray();
910      filer->createFile("seeds.hdf5","tzero");
911      delete filer;
912      gsl_set_error_handler_off();
913      const int noThreads = 4;
914      worker* workers = new worker[noThreads];
915      std::future<void> future[noThreads];
916      std::future_status status[noThreads];
917
918      for (int i = 0; i < noThreads; ++i){
919          future[i] = std::async(std::launch::async, &worker::seed, workers+i, i);
920          std::cout<<"Initialised thread for t = "<<i<<std::endl;
921      }
922
923      int t = noThreads;
924
925      while (t<1001){
926          for (int i = 0; i < noThreads; ++i){
927              status[i] = future[i].wait_for(std::chrono::milliseconds(0));
928              if (status[i] == std::future_status::ready) {
929                  if (t < 1001){// Needed to address issues at the end of the time interval.
```

```
930                     future[i] = std::async(std::launch::async, &worker::seed, workers+i, t);
931                     std::cout<<"Initialised thread for t = "<<t<<std::endl;
932                     std::cout<<"Total progress: "<<t/10<<"% complete."<<std::endl;
933                     ++t;
934                 }
935             }
936         }
937         std::this_thread::sleep_for(std::chrono::seconds(10));
938     }
939     for (int i = 0; i < noThreads; ++i){
940         future[i].wait();
941     }
942 
943     delete [] workers;
944 
945     return 0;
946 }
```

```cpp
/* Program to calculate helicity via winding numbers. */

/* To compile:
/usr/local/hdf5/bin/h5c++ windingatan.cpp -o windingatan -std=c++0x -pthread -O3
*/

#include <iostream>
#include <math.h>
#include "myArrayv3.h"
#include "JHDF5interp3v2.h"
#include <future>
#include <chrono>
#include <thread>
#include <mutex>

myArray results;

std::mutex readLock;

const double phi = 0.00000000213444;

void WINDING(int time){
    JHDF5* data = new JHDF5("seedsNew.hdf5","tzero");
    readLock.lock();
    data->initialise(time);
    readLock.unlock();
    const double pi = 3.141592654;
    int i,j,l,seed1[2],seed2[2],a;
    const int xdim = 65;
    const int ydim = 65;
    const int zdim = 50;
    double r12x[zdim],r12y[zdim],th12[zdim];
    double diff,totalw=0;
    //Calculate the winding number
    int list[xdim*ydim][2],u,v;
    for (u = 0; u < xdim; u++){
        for (v = 0; v < ydim; ++v){
            list[u+xdim*v][0]=v;
            list[u+xdim*v][1]=u;
        }
    }
    for (i = 0; i < xdim*ydim; ++i){
        for (j = i+1; j < xdim*ydim; ++j){

            seed1[0]=list[i][0];
            seed1[1]=list[i][1];
            seed2[0]=list[j][0];
            seed2[1]=list[j][1];

            r12x[0] = data->buffer[0][seed2[0]][seed2[1]][0]-data->buffer[0][seed1[0]][seed1[1]][0];
            r12y[0] = data->buffer[1][seed2[0]][seed2[1]][0]-data->buffer[1][seed1[0]][seed1[1]][0];
            th12[0] = atan2(r12y[0],r12x[0]);

            //Convert to polars
            for (a = 1; a < zdim; ++a){
                r12x[a] = data->buffer[0][seed2[0]][seed2[1]][a]-data->buffer[0][seed1[0]][seed1[1]][a];
                r12y[a] = data->buffer[1][seed2[0]][seed2[1]][a]-data->buffer[1][seed1[0]][seed1[1]][a];
                //Use four quadrant atan
                th12[a] = atan2(r12y[a],r12x[a]);

                diff = (th12[a]-th12[a-1]);
                if (diff > pi){
                    diff=diff-2*pi;
                }
                else if (diff<-pi){
                    diff=diff+2*pi;
```

```
                }
                totalw+=diff;
            }
         }
    }
    delete data;
    results.buffer[time] = totalw*phi/(65*65*2*pi);
}

int main(){

    const int noThreads = 4;
    std::future<void> future[noThreads];
    std::future_status status[noThreads];

    for (int i = 0; i < noThreads; ++i){
        future[i] = std::async(std::launch::async, &WINDING, i);
        std::cout<<"Initialised thread for t = "<<i<<std::endl;
    }

    int t = noThreads;

    while (t < 1001){
        for (int i = 0; i < noThreads; ++i){
            status[i] = future[i].wait_for(std::chrono::milliseconds(0));
            if (status[i] == std::future_status::ready) {
                if (t < 1001){
                    future[i] = std::async(std::launch::async, &WINDING, t);
                    std::cout<<"Initialised thread for t = "<<t<<std::endl;
                    std::cout<<"Total progress: "<<t/10<<"% complete."<<std::endl;
                    ++t;
                }
            }
        }
        std::this_thread::sleep_for(std::chrono::seconds(2));
    }
    for (int i = 0; i < noThreads; ++i){
        future[i].wait();
    }

    results.createFile("windingatan.hdf5","w");
    results.write("windingatan.hdf5","w");

    return 0;


```

6 References

[1] M. A. Berger and C. Prior, J. Phys. A **39**, 8321 (2006).

[2] E. Priest, *Solar Magneto-Hydrodynamics* (D.Reidel Publishing Company, Dordrecht, Holland, 1943).

[3] M. Berger, Geophys. Monogr. Ser. **111**, 1 (1999).

[4] T. H. Group, Hierarchical data format version 5, `http://www.hdfgroup.org/HDF5/`, 2013.

[5] W. Gekelman *et al.*, Physica Scripta **T142** (2010).

[6] G. Monegato, SIAM Review **24**, 137 (1982).

[7] W. Gekelman, E. Lawrence, and B. Van Compernolle, The Astrophysical Journal **753**, 131 (2012).

[8] J. Hass, J. Lagarias, and N. Pippenger, Journal of the ACM **46**, 185 (1999).

[9] P. Cromwell, E. Beltrami, and M. Rampichini, Math. Intelligencer **20**, 53 (1998).

[10] R. Ricca and B. Nipoti, Journal of Knot Theory and Its Ramifications **20**, 1325 (2011).

[11] M. Epple, Math. Intelligencer **20**, 45 (1998).

[12] W. Massey, J. Knot Theory Ramifications **07**, 393 (1998).

[13] M. A. Berger, Lett Math Phys **55**, 181 (2001).

[14] F. B. Fuller, Proc. Nat. Acad. Sci. U.S.A. **75**, 3557 (1978).

[15] J. White and W. R. Bauer, J. Mol Biol **189**, 329 (1986).

[16] R. L. Ricca, Applications of knot theory in fluid mechanics, in *Knot theory (Warsaw, 1995)*, , Banach Center Publ. Vol. 42, pp. 321–346, Polish Acad. Sci., Warsaw, 1998.

[17] C. Barenghi, R. Donnelly, and W. Vinen, *Quantized Vortex Dynamics and Superfluid Turbulence* (Springer Publishing, New York, USA, 2001).

[18] D. Rolfsen, *Knots and Links*, Second ed. (AMS Chelsea Publishing, Providence RI., 2003).

[19] L. H. Kauffman, *Knots and Physics*, Second ed. (World Scientific, Singapore, 1993).

[20] G. Călugăreanu, Rev. Math. Pures Appl. **4**, 5 (1959).

[21] M. R. Dennis and J. H. Hannay, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. **461**, 3245 (2005).

[22] M. Abate and F. Tovena, *Curves and Surfaces* (Springer Publishing, New York, USA, 2012).

[23] H. K. Moffatt and R. L. Ricca, Proc. R. Soc. A **439**, 411 (1992).

[24] R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics: Mainly Electromagnetism and Matter* (Addison-Wesley, Boston, USA, 2009).

[25] H. Alfvén, Nature **150**, 405 (1942).

[26] F. James, editor, *The Correspondence of Michael Faraday* (Short Run Press Ltd., Exeter, 1993).

[27] C. D. Cothran *et al.*, Geophysical Research Letters **32** (2005).

[28] T. I. Gombosi, K. G. Powell, and B. van Leer, Journal of Geophysical Research: Space Physics **105**, 13141 (2000).

[29] S. Fromang and J. Papaloizou, A&A **476**, 1113 (2007).

[30] R. Lundin *et al.*, Annales Geophysicae **23**, 2565 (2005).

[31] P. Davidson, *An Introduction to Magnetohydrodynamics*Cambridge Texts in Applied Mathematics (Cambridge University Press, Cambridge, UK., 2001).

[32] D. Griffiths, *Introduction to Electrodynamics* (Pearson Education, Upper Saddle Rvr, NJ, USA, 2012).

[33] G. K. Batchelor, *An Introduction to Fluid Dynamics* (Cambridge University Press, Cambridge, UK, 2000).

[34] M. Berger, Plasma Physics and Controlled Fusion **41**, B167 (1999).

[35] L. Woltjer, Proceedings of the National Academy of Science **44**, 489 (1958).

[36] D. Biskamp, *Nonlinear Magnetohydrodynamics* (Cambridge University Press, Cambridge, UK, 1997).

[37] M. A. Berger and G. B. Field, J. Fluid Mech. **147**, 133 (1984).

[38] H. K. Moffatt, J. Fluid Mech. **35**, 117 (1969).

[39] L. van Driel-Gesztelyi, P. Démoulin, and C. H. Mandrini, Advances in Space Research **32**, 1855 (2003).

[40] G. E. Hale, PASP **37**, 268 (1925).

[41] R. S. Richardson, The Astrophysical Journal **93**, 24 (1941).

[42] T. T. Yamamoto, K. Kusano, T. Maeshiro, T. Yokoyama, and T. Sakurai, Astrophysical Journal **624**, 1072 (2005).

[43] E. Priest, Heating the solar corona by magnetic reconnection, in *Plasma Astrophysics And Space Physics*, edited by J. Bchner, I. Axford, E. Marsch, and V. Vasylinas, pp. 77–100, Springer Netherlands, 1999.

[44] H. Isobe and K. Shibata, Journal of Astrophysics and Astronomy **30**, 79 (2009).

[45] P. Démoulin and E. Pariat, Advances in Space Research **43**, 1013 (2009).

[46] L. Fosdick *et al.*, *An Introduction to High-performance Scientific Computing* (MIT Press, Cambridge, USA, 1996).

[47] B. Flowers, *An Introduction to Numerical Methods in C++* (Oxford University Press, New York, USA., 2000).

[48] R. Magazine, R&d 100 awards, `http://www.rdmag.com/award-winners/2002/08/flexible-data-management`, 2012.

[49] M. Galassi *et al.*, GNU scientific library, `http://www.gnu.org/software/gsl/`, 2013.

[50] L. Carroll, *Alice's Adventures in Wonderland* (Mac Millan, London, UK., 1948).

[51] A. Bates and A. Maxwell, *DNA Topology*, Second ed. (Oxford University Press, Oxford, UK, 2005).

[52] J. Campbell and M. Berger, Helicity, linking, and writhe in a spherical geometry, in *Knotted, Linked and Tangled Flux in Quantum and Classical Systems*, , Journal of Physics conference series Vol. -, pp. –, Institute of Physics, London, to appear.

[53] G. H. M. van der Heijden and J. M. T. Thompson, Nonlinear Dyn. **21**, 71 (2000).

[54] O. Barrett, *Elementary Differential Geometry* (Academic Press, London, UK, 1966).

[55] F. Nietzsche and W. Kaufmann, *The Gay Science: With a Prelude in Rhymes and an Appendix of Songs*Vintage (Random House, New York, USA., 2010).

[56] W. Gekelman *et al.*, Review of Scientific Instruments **62**, 2875 (1991).

[57] S. Rosenberg and W. Gekelman, Geophysical Research Letters **25**, 865 (1998).

[58] B. Friedman, T. A. Carter, M. V. Umansky, D. Schaffner, and I. Joseph, Physics of Plasmas **20**, 055704 (2013).

[59] T. A. Carter and J. E. Maggs, Phys. Plasmas **16**, 012304 (2009).

[60] I. H. Hutchinson, Phys. Rev. A **37**, 4358 (1988).

[61] K.-S. Chung, Plasma Sources Science and Technology **21**, 063001 (2012).

[62] E. R. Priest and P. Démoulin, J. Geophys. Res. **100**, 23,443 (1995).

[63] K. Weierstrass, Proceedings of the Prussian Academy of Sciences **2** (1885).

[64] J. D. Faires and R. L. Burden, *Numerical Methods*, Third ed. (Brooks-Cole Publishing, Pacific Grove, USA, 2002).

[65] G. Dahlquist and Å. Björck, *Numerical Methods*Dover Books on Mathematics (Dover Publications, Englewood Cliffs, NJ, USA, 2003).

[66] V. Dmitriev and Z. Ingtem, Computational Mathematics and Modeling **23**, 312 (2012).

[67] F. Mackay, R. Marchand, and K. Kabin, J. Geophys. Res. **111** (2006).

[68] C. de Boor, *A Practical Guide to Splines* (Springer, New York, USA, 2001).

[69] L. Schumaker, *Spline Functions: Basic Theory* (Cambridge University Press, Cambridge, UK, 2007).

[70] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C: the art of scientific computing*, Second ed. (Cambridge University Press, New York, USA, 1992).

[71] F. Lekien and J. Marsden, Int. J. Numer. Meth. Engng **63**, 455 (2005).

[72] J. Kullberg *et al.*, platinum-image, `http://code.google.com/p/platinum-image/`, 2013.

[73] Rahi, *Concepts Of Functions & Calculus Iit* (Tata McGraw-Hill Education, New Delhi, India, 2009).

[74] G. Casciola and L. Romani, *Mathematical methods for curves and surfaces: Tromsø* (Nashboro Press, Brentwood, TN, USA, 2004), chap. A Piecewise Rational Quintic Hermite Interpolant for Use in CAGD.

[75] E. Fehlberg, NASA Technical Report **315** (1969).

[76] K. Atkinson, W. Han, and D. Stewart, *Numerical Solution of Ordinary Differential Equations* (Wiley-Interscience, Hoboken, USA, 2009).

[77] Merriam-Webster, "quadrature.", `http://www.Merriam-Webster.com`, 2013.

[78] A. Iserles, S. Nrsett, and S. Olver, Highly oscillatory quadrature: The story so far, in *Numerical Mathematics and Advanced Applications*, edited by A. de Castro, D. Gmez, P. Quintela, and P. Salgado, pp. 97–118, Springer Berlin Heidelberg, 2006.

[79] L. Trefethen and D. Bau, *Numerical Linear Algebra* (Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997).

[80] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (Dover Publications, Dover, USA, 1964).

[81] D. Kahaner, C. Moler, S. Nash, and G. Forsythe, *Numerical methods and software*Prentice-Hall series in computational mathematics (Prentice-Hall, Englewood Cliffs, NJ, USA, 1989).

[82] G. Szegö, Mathematische Annalen **110**, 501 (1935).

[83] R. Piessens *et al.*, *Quadpack: a subroutine package for automatic integration* (Springer-Verlag, Berlin, USA, 1983).

[84] The Numerical Algorithms Group (NAG), The NAG library, `http://www.nag.com/`, 2013.

[85] R Core Team, R: A language and environment for statistical computing, `http://www.R-project.org`, 2013.

[86] N. Kovvali, *Theory and Applications of Gaussian Quadrature Methods* (Morgan & Claypool Publishers, San Rafael, CA, USA, 2012).

[87] A. Genz and A. Malik, Journal of Computational and Applied Mathematics **6**, 295 (1980).

[88] J. Berntsen, T. O. Espelid, and A. Genz, ACM Trans. Math. Softw. **17**, 437 (1991).

[89] W. Gekelman, A. Collette, and S. Vincena, Physics of Plasmas **14**, 062109 (2007).

[90] J. B. Taylor, Phys. Rev. Lett. **33**, 1139 (1974).

[91] F. Chen, *Introduction to Plasma Physics and Controlled Fusion* (Springer-Verlag, Berlin, USA, 2010).