INSTITUTO SUPERIOR TÉCNICO

UNIVERSIDADE DE LISBOA

PROJETO INTEGRADOR DE 1º CICLO

Implementing Dommaschk potentials for the movement of a single particle in the NEAT code

Author:
Miguel PEREIRA

Supervisor:
Prof. Rogério JORGE

Research work performed for the Bachelor in Engineering Physics

at

Instituto de Plasmas e Fusão Nuclear
Physics Department

June 16, 2023

Abstract

Energy production is one of Humanity's foremost concerns. Nuclear fusion has been a tantalizing candidate to help in this endeavor, as it would provide abundant energy fueled by common materials, however, it has not shown working concepts yet. A promising technological path to this goal is the stellarator, a magnetic confinement device. In the present work, we focus on a type of analytical stellarator magnetic field studied by W. Dommaschk from 1977 onwards, that can model magnetic islands. These Dommaschk fields are implemented within the particle tracing repository *gyronimo*, while interfacing with it using NEAT, another repository. This method may serve as a faster alternative to other techniques which are either more computationally intensive at modeling islands, such as the SPEC code, or as a more exact alternative to codes that use approximations, such as the VMEC code. Finally, some example results and graphs are shown, namely, particle trajectories, and representations of the field through both tracing and Poincaré plots.

1 Introduction

1.1 Energy and Fusion

We live in a developing world, where demand and consumption of energy are increasing[1], but also where the main ways of producing energy have deleterious effects on our environment, and are at risk of shortage within our lifetimes[2]. As with any real problem, the alternatives come with their own caveats. Nuclear fission, with a low carbon footprint, has a strong social stigma associated due to past catastrophes[3], and renewable energies such as solar and wind would need significant power storage infrastructure to be solely relied upon.[4, 5]

A long theorized and experimented upon future alternative to this modern conundrum has been nuclear fusion, a type of reaction where two nuclei come together to form a heavier nuclide, releasing energy in the process, and in a much more energy-dense way than the methods previously mentioned [6]. The most energetically relevant reaction at the lowest viable temperatures is the Deuterium-Tritium reaction, which occurs several orders of magnitude more often than the other allowable fusion reactions.[7]. It is represented as follows:

$$D + T \rightarrow \mathrm{He}^4(3.52\ \mathrm{Mev}). \tag{1}$$

Although it has the potential to solve many of the world's problems, the issue has always been in engineering, since to reach the energy associated with fusion you must first overcome electromagnetic repulsion between nuclei, to then have the strong force pull them together, which in practice requires high temperatures or high pressures.

The two main approaches in fusion energy that are considered to have the potential to overcome current challenges are: inertial confinement, which holds the fuel in compressed capsules, and then heats it up either directly or indirectly with lasers [8]; and magnetic confinement, which tries to get the fuel hot enough to be in a sustained plasma state, where fusion can happen, without being pressurized. In the latter approach, magnetic coils are used to prevent the plasma from hitting the walls of the device and stopping the reaction. Magnetic confinement can be divided into two different approaches [9]: tokamaks, which have an axisymmetric toroidal shape, along with a central solenoid that twists the magnetic field such that the particles oscillate between being farther or closer to the center, thus preventing drift; and stellarators, which twist the magnetic field into whichever shape is necessary to have a steady state operation, organizing their coils in a somewhat toroidal shape, without a central stabilization coil (see Fig. 1). In this work, we focus on stellarators.

As computing power has progressed, so too have stellarators, because it is a very complex system [10], with many different parameters to be optimized, such as the plasma shape, the magnetic field, the coils which generate the field, the stability of the plasma and its heat exhaust. One widely used code for these purposes is SIMSOPT [11], which uses the VMEC code [12] to calculate the magnetic field equilibrium. However, VMEC assumes the existence of nested magnetic flux surfaces[13]. In other words, it assumes that the magnetic field lines ergodically cover a set of toroidal surfaces. In reality, it is known that this

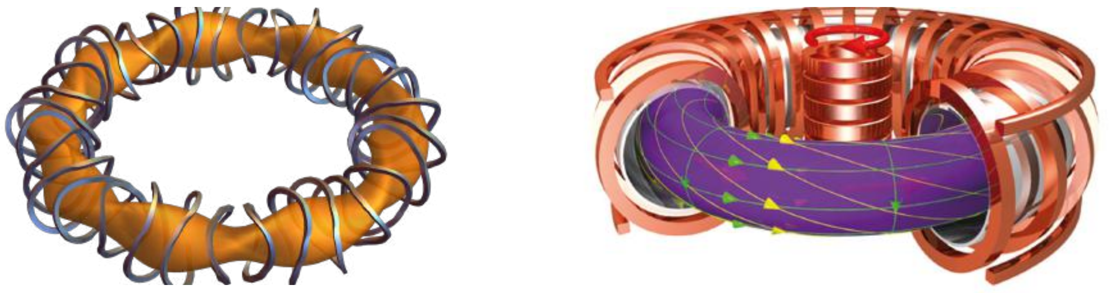

FIGURE 1: The stellarator (left) and the tokamak (right) concepts, both their plasma and coil shapes

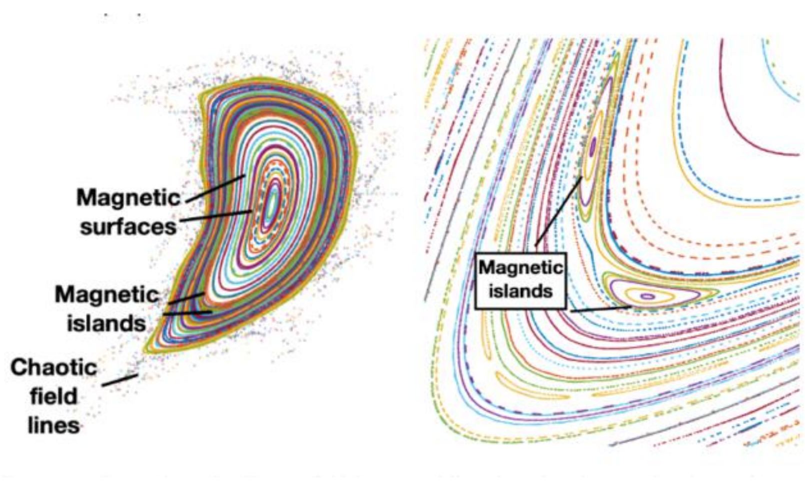

FIGURE 2: Representation of magnetic islands, taken from Ref. [17].

approximation does not hold everywhere. Further away from the center of the equilibrium, in regions of resonant surfaces and at increasing pressure values, phenomena such as magnetic islands and chaos start to become more significant. Magnetic islands are a type of structure within magnetic equilibriums. They are essentially secondary axes in the magnetic field[14], and they are an important focus of research in stellarator design, either to control them and utilize them, or to predict how sensitive they are to imperfections of the magnetic coils[15]. In Fig. 2 a section of the magnetic field formed by the NCSX coils[16] is shown. This type of graph is called a Poincaré plot and will be explored further in Section 3. It is important to be able to simulate such structures, as they are known to be critical for the functioning of stellarators.[18, 19] The SPEC [20, 21] code can handle the existence of such complex possibilities [15], however, optimizing stellarators with SPEC is quite a computationally intensive task [13], as the equilibrium is found through quite a complex method, namely, multi-region relaxed magnetohydrodynamics. There is, however, a type of analytically solvable field that can model magnetic islands.

1.2 Dommaschk Potentials

Starting in 1977 with an internal document[22] and continuing through a series of papers[23, 24] culminating in the seminal work in Ref. [25], Dommaschk potentials are an analytical solution to the Laplace equation $\Delta U = 0$. This comes as a result of Maxwell's equations in a vacuum, for our system. Since the rotational of the magnetic field is null, its potential can be written as $\mathbf{B} = \nabla U$. The other relevant equation states that the divergence of the magnetic field is null, thus, if we combine the two, we get the Laplace equation. It is worth mentioning that it is not the only of its kind, as shown in Ref. [26]. They are, however, particular in that they use cylindrical coordinates, instead of more complex curvilinear coordinates such as Boozer coordinates [13]. They can be written in a closed analytical form using only powers, logarithms, sines, and cosines. This means they could possibly be a quicker alternative to other codes that include islands, such as SPEC, although it does assume that the pressure vanishes, unlike SPEC[15]. Each Dommaschk potential is characterized by two numbers, m and l, representing the number of toroidal and poloidal periods of the field, respectively, and two coefficients. As they are solutions to the Laplace equation, it is then possible to linearly combine several instances of these potentials, to achieve the desired properties of the overall field. The lengths are normalized such that the plasma region of interest is in the neighborhood of $R = 1$ and $Z = 0$. These potentials are defined as follows:

$$V(R,\phi,Z) = \phi + \sum_{m,l} V_{m,l}(R,\phi,Z),\ m \geq 0,\ l = 0,1,2,\ldots\ . \tag{2}$$

To determine the value of these potentials $V_{m,l}$, the following ansatz is used, which separates the variables of the magnetic potential

$$\begin{aligned} U &= I_{m,n}(Z,R)e^{\pm im\phi}, \\ I_{m,n} &= \sum_{2k\leq n}^{k=0} \frac{Z^{n-2k}}{(n-2k)!}C_{m,k}(R),\ k = 0,1,\ldots n/2. \end{aligned} \tag{3}$$

By plugging the ansatz back in the Laplace equation, and depending on whether Dirichlet or Neumann boundary conditions are used, two systems of solutions for the coefficients $C_{m,k}(R)$ are obtained, namely

$$C^D_{m,k} = -[\alpha_j(\alpha^* ln(R) + \gamma^* - \alpha)_{k-m-j} - \gamma_j\alpha^*_{k-m-j} + \alpha_j\beta^*_{k-j}]R^{2j+m} + \beta_j\alpha^*_{k-j}R^{2j-m}, \tag{4}$$
$$C^N_{m,k} = +[\alpha_j(\alpha ln(R) + \gamma)_{k-m-j} - \gamma_j\alpha_{k-m-j} + \alpha_j\beta_{k-j}]R^{2j+m} - \beta_j\alpha_{k-j}R^{2j-m}, \tag{5}$$

with summation over $j = 0,1,2,\ldots,k$ implied, and where

$$\alpha_n = \frac{(-1)^n}{\Gamma(m+n+1)\Gamma(n+1)2^{2n+m}}, \qquad \alpha^*_n = (2n+m)\alpha_n, \tag{6}$$
$$\beta_n = \frac{\Gamma(m-n)}{\Gamma(n+1)2^{2n-m+1}}, \qquad \beta^*_n = (2n-m)\beta_n. \tag{7}$$

The two solutions found in Eq. (4) can then be linearly combined to yield:

$$V_{m,l} = [a_{m,l}\cos(m\phi) + b_{m,l}\sin(m\phi)]D_{m,l} + [c_{m,l}\cos(m\phi) + d_{m,l}\sin(m\phi)]N_{m,l-1}, \tag{8}$$

where $D_{m,l}$ and $N_{m,l}$ contain the dependence on Z and R, i.e. it is the term $I_{m,l}$ defined in Eq. (3), however, applied to the case of Dirichlet and Neumann boundary conditions). If we impose the usual stellarator symmetry that the magnetic field equilibrium should be identical when rotated upside down, $V_{m,l}(R,\phi,Z) = -V_{m,l}(R,-\phi,-Z)$, two of the coefficients will always vanish,

$$\begin{aligned} a_{m,l} = d_{m,l} &= 0 \text{ for even } l, \\ b_{m,l} = c_{m,l} &= 0 \text{ for odd } l. \end{aligned} \tag{9}$$

To obtain the magnetic field, the gradient of these potentials must be calculated, which is done below in Section 2.1.

Next, we will explain the theoretical and software framework of particle tracing that these fields will be implemented upon.

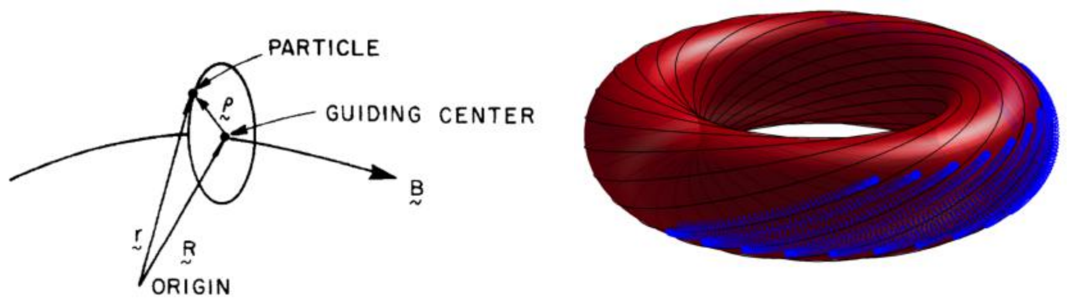

FIGURE 3: Guiding center coordinates (left), taken from Ref. [28] and trajectory described by particles in a tokamak plasma (right), taken from Ref. [29]

1.3 *Gyronimo* and Guiding Center

The ultimate goal is to perform optimization using these fields, using metrics such as the Greene residue[15]. However, in this preliminary phase, we aim to first implement Dommaschk fields into a particle tracing code, in this case, *gyronimo*[27], written with the C++ language. *Gyronimo* uses the guiding center equations for the movement of charged particles in magnetic fields. In the guiding center model, the position $\boldsymbol{r}$ of each particle is divided into two components via the following equation

$$\boldsymbol{r} = \boldsymbol{R} + \boldsymbol{\rho}, \tag{10}$$

where $\boldsymbol{R}$ is the guiding center position, the axis around which the particle does its cyclotronic motion, and $\boldsymbol{\rho}$, which is the offset from that axis to the real particle position. These coordinates can be visualized in Fig. 3, along with an example of a particle trajectory found in tokamak plasmas. If we then use the so-called guiding center approximation, which assumes the ρ coordinate is close enough to zero, a coordinate is effectively eliminated from the description of the system, which means one less equation to solve, due to the Lagrangian description. Additionally, the invariance of the magnetic moment, μ, makes its associated equation trivial, effectively leaving us with four degrees of freedom[30].

We will be interfacing with *gyronimo* via the NEAT[31] code, which is built precisely to be compatible with efficient C++ codes, while using more user-friendly *python* code. Using NEAT also means that in the future, optimization of Dommaschk fields could be done within the same repository, since NEAT is meant to be a harbor for particle tracing, equilibrium fields, and optimization.

2 Materials and Methods

2.1 Derivation of the Derivatives

To implement a steady-state three-dimensional field into NEAT, two methods must be created in a C++ class. The first one calculates the contravariant[32] components of the magnetic field, and the second one calculates the derivatives of such components (9 terms in total), to use in the numerical solver. Due to the fact that Ref. [25] only provides us with the formulas for the potentials, the first and second-order derivatives of the potentials must be derived in order to implement those methods. Because the potentials are in a separate form, $V_{m,l}^{D/N} = \mathcal{R}(R)\Phi(\phi)\mathcal{Z}(Z)$, those derivatives can be worked out for each coordinate individually, and then combined accordingly to find the cross derivatives

$$\begin{aligned}
\frac{\partial}{\partial R}C^D_{m,k} =& -[\alpha_j(\alpha^* ln(R)+\gamma^*-\alpha+\frac{\alpha^*}{2j+m})_{k-m-j}-\gamma_j\alpha^*_{k-m-j}+\alpha_j\beta^*_{k-j}]R^{2j+m-1}(2j+m) \\
&+\beta_j\alpha^*_{k-j}R^{2j-m-1}(2j-m), \qquad (11)\\
\frac{\partial^2}{\partial R^2}C^D_{m,k} =& -[\alpha_j(\alpha^* ln(R)+\gamma^*-\alpha+\frac{\alpha^*(4j+2m+1)}{(2j+m)(2j+m-1)})_{k-m-j}-\gamma_j\alpha^*_{k-m-j} \\
&+\alpha_j\beta^*_{k-j}]R^{2j+m-2}(2j+m)(2j+m-1)+\beta_j\alpha^*_{k-j}R^{2j-m-2}(2j-m)(2j-m-1), \qquad (12)\\
\frac{\partial}{\partial R}C^N_{m,k} =& -[\alpha_j(\alpha ln(R)+\gamma+\frac{\alpha}{2j+m})_{k-m-j}-\gamma_j\alpha^{k-m-j}+\alpha_j\beta^{k-j}]R^{2j+m-1}(2j+m) \\
&-\beta_j\alpha^{k-j}R^{2j-m-1}(2j-m), \qquad (13)\\
\frac{\partial^2}{\partial R^2}C^N_{m,k} =& -[\alpha_j(\alpha ln(R)+\gamma+\frac{\alpha(4j+2m+1)}{(2j+m)(2j+m-1)})_{k-m-j}-\gamma_j\alpha_{k-m-j} \\
&+\alpha_j\beta_{k-j}]R^{2j+m-2}(2j+m)(2j+m-1)-\beta_j\alpha_{k-j}R^{2j-m-2}(2j-m)(2j-m-1), \qquad (14)\\
\frac{\partial}{\partial Z}I_{m,n} =& \sum_{2k\leq n}^{k=0}\frac{Z^{n-2k-1}(n-2k)}{(n-2k)!}C_{m,k}(R), \qquad (15)\\
\frac{\partial^2}{\partial Z}I_{m,n} =& \sum_{2k\leq n}^{k=0}\frac{Z^{n-2k-2}(n-2k)(n-2k-1)}{(n-2k)!}C_{m,k}(R), \qquad (16)
\end{aligned}$$

while the toroidal derivatives trivially follow from equation (8), with one caveat. While we are performing the first-order derivatives, we are technically applying the gradient operator, which in cylindrical coordinates is as follows[33]

$$\nabla V = \left(\frac{\partial V}{\partial R}, \frac{1}{R}\frac{\partial V}{\partial \phi}, \frac{\partial V}{\partial Z}\right). \qquad (17)$$

Only the poloidal derivative term is unique, in that there is a $1/R$ factor included, explaining why the previous derivatives in order of R and Z were direct. A consequence of the above-mentioned factor is that our description no longer has perfect separation of the variables. This means that the case of the second-order derivative that is first poloidal and then radial will be different from the case where does derivatives are done in the reverse order. In respective order, they are as follows:

$$\begin{aligned}
\frac{\partial^2}{\partial R\partial\phi}V_{m,l} =& m[-a_{m,l}\sin(m\phi)+b_{m,l}\cos(m\phi)]\frac{1}{R}\frac{\partial D_{m,l}}{\partial R}+m[-c_{m,l}\sin(m\phi)+d_{m,l}\cos(m\phi)]\frac{1}{R^2}\frac{\partial N_{m,l-1}}{\partial R} \\
&-m[-a_{m,l}\sin(m\phi)+b_{m,l}\cos(m\phi)]\frac{1}{R^2}D_{m,l}+m[-c_{m,l}\sin(m\phi)+d_{m,l}\cos(m\phi)]\frac{1}{R^2}N_{m,l-1}. \qquad (18)\\
\frac{\partial^2}{\partial\phi\partial R}V_{m,l} =& m[-a_{m,l}\sin(m\phi)+b_{m,l}\cos(m\phi)]\frac{\partial D_{m,l}}{\partial R}+m[-c_{m,l}\sin(m\phi)+d_{m,l}\cos(m\phi)]\frac{\partial N_{m,l-1}}{\partial R} \qquad (19)
\end{aligned}$$

This also has one more implication. In Eq. (2), we see that the Dommaschk potentials have a ϕ term that must be added, separate from all the $V_{m,l}$ potentials. The contribution of that term to the magnetic field is a $(0, 1/R, 0)$ that must be added, which was calculated by means of Eq. (17). It also means that a $-1/R^2$ term must be added to the case of the second order derivative term $\partial^2_{\phi,R}$.

2.2 Implementation in *gyronimo*

We can now move on to the implementation. In *gyronimo*, we must implement an `equilibrium` object, in this case, `equilibrium_dommaschk`, which can output the magnetic field and its derivatives. It receives the following inputs: a `metric` object which specifies which coordinates are being used in the system, in this case, `metric_cylindrical`, the value of the coordinates, in this case, R, ϕ, Z, a value for m, l, and the two coefficients, and a value for the normalization of the magnetic field, `B0`. Thus, the C++ constructor is as follows:

```
equilibrium_dommaschk(
  const metric_cylindrical *g, int m, int l, double coeff1, double coeff2, double B0);
```

For the implementation of the derivatives, it was already done first in the SIMSOPT code, one of the most complete open-source stellarator repositories. That implementation calculates the magnetic field, and its derivatives, however, the output is coded to be in Cartesian coordinates. With *gyronimo*, we can use the coordinates we wish, as discussed previously. Additionally, after checking that version, the terms $\alpha/(2j+m)$, $\alpha(4j+2m+1)/[(2j+m)(2j+m-1)]$ in the Neumann coefficients' radial derivatives, along with the similar terms in the Dirichlet radial derivatives, were found to be missing. For this reason, along with the work done on NEAT for this project, a pull request was done for SIMSOPT, where the bug was fixed, and was ultimately incorporated into the open-source repository.

With this, it was possible to implement the methods

```
virtual IR3 contravariant(const IR3& position, double time) const override;
virtual dIR3 del_contravariant(const IR3& position, double time) const override;
```

which output the contravariant terms of the magnetic field, and the derivatives of those terms, respectively. *Gyronimo* already had a built-in class to numerically manipulate a 3D vector, and another for the partial derivatives of a 3D vector, which are the `IR3` and `dIR3` objects used above.

As mentioned before, a `metric_cylindrical` object was also required as input. It needed to be implemented, but the process was straightforward, since most methods were very trivial to do with cylindrical coordinates. Namely, methods needed to be added that would receive as input a three-dimensional position, and output the following differential objects: the metric matrix, its derivative, the cylindrical Jacobian, the derivative of said Jacobian, and then two methods which additionally receive as input a three-dimensional field, such as a magnetic field. One method converts the field to covariant form, from contravariant form, and the other does the opposite. All these operations were documented in Ref. [33].

Now with the implementation of our Dommaschk equilibrium field, it was possible to utilize the examples found in the NEAT repository to go about implementing particle tracing, but to do it in a Dommaschk field instead. The process is done through the Runge-Kutta method of integration, applied to the guiding center equations of motion. In essence, a few components are required to make this work. A `dommaschktrace` function is created, which will be the C++ interface for the tracing code. Its header is as follows:

```
vector< vector<double>>  dommaschktrace(const vector<int> m, const vector<int> l,
    const vector<double> coeff1, const vector<double> coeff2, const vector<double> B0,
    double charge, double mass, double Lambda, double vpp_sign, double energy,
    double R0, double phi0, double Z0,
    size_t nsamples, double Tfinal)
```

The first five inputs are the same as the `equilibrium_dommaschk` class, however, they are now in vector form, such that we can achieve our goal of adding multiple Dommaschk fields together, as discussed in Section 1.2. We then have inputs for the charge and mass of the particle being simulated, then three inputs for the initial guiding center parameters, explained further in Ref. [30], another three inputs for the initial position in cylindrical coordinates, and then finally, two inputs for the integration, the total time of the simulation, `Tfinal`, and how many time steps will be taken in that time, `nsamples`. This function will output vectors of parameters that are solutions to the numeric integration, at each time step, meaning the vectors will have a size equal to `nsamples`. We will later focus on four of them, the time and the three-dimensional position, as they are what is required for tracing the trajectory of the particles. The function works as follows: it first creates the `equilibrium_dommaschk` field that will be used, or, in the case of having received vectors in the first 5 inputs, it will instantiate multiple `equilibrium_dommaschk` fields using those parameters, and then combine them through the built-in *gyronimo* class created for this purpose, called `linear_combo_c1`. It is essentially another `equilibrium` field, but which receives as input a variable number of other `equilibrium` fields, and makes the overall object work as though the underlying objects were being added together, which is what it does, for each method.

The goal is then to perform numeric integration, and thankfully there are optimized libraries that can be used for this effect, in this case, the Boost library. The object that defines Runge-Kutta integration is `boost::numeric::odeint::runge_kutta4` which can then be used to specify which integration algorithm the integration object will use. For clarity, here is the initialization of that object in full:

```
boost::numeric::odeint::integrate_const(integration_algorithm,
 odeint_adapter(&gc),initial_state, 0.0, Tfinal, Tfinal/nsamples, observer);
```

This object integrates with a constant step size. We must also specify what equations are being used, and pass that input as an object that is properly formatted to the specifications of this function. Thankfully, this is another part where using *gyronimo* benefits us greatly, as there is already an object that defines the guiding center equations in a magnetic field, called `guiding_centre`, and another that adapts that object to be ready to use in the boost numeric integrator, called `odeint_adapter`, thus explaining the second input. The other inputs are the `initial_state` object, which contains the initial positions and the initial $v_\perp$. We only need four initial conditions because, as explained in Section 1.3, there are only four non-trivial equations out of the initial six-dimensional description. We then have inputs for the starting time, zero for simplicity, the final time, `Tfinal`, and the time step, given by the total integration time divided by the number of samples, as is shown. The final input is the so-called `observer`, which is the object that will be saving the solution arrays, that contain the position, time, and other parameters, for each time step of the integration, as discussed earlier. To define this, a class needed to be implemented, called `push_back_state_and_time_dommaschk`, formatted to save the solution in a set of vectors, each of them being pushed back at each step of the integration. This defines our C++ interface.

2.3 Transition to *python*

As mentioned in Section 1.3, NEAT is built to have *python* code interface with C++, which is the next and final step. The library used is called pybind11, which will allow us to call functions from *python*, while then using a C++ function. In this case, we want to call the newly defined `dommaschktrace` from *python*, such that we can then do physics without being concerned about the underlying code which makes it work. By adding a line of code in the `neatpp.cpp` file, the desired effect can be achieved. The relevant part is shown here in full:

```
PYBIND11_MODULE(neatpp, m) {
[...] Code for the rest of NEAT's functions that need wrapping [...]
 m.def("dommaschktrace",&dommaschktrace,
      "Trace a single particle in a Dommaschk equilibrium magnetic field");
```

Given the fact that the third argument of the `m.def()` line of code is simply for the purposes of documentation, it is a feat of code engineering that this works with so little user input. The `dommaschktrace` function can now be imported elsewhere in the *python* section of the code, so it is possible to define the *python* class that will be the interface for a user of the code. The relevant NEAT file is called `fields.py`. The class that was defined there, called simply `Dommaschk`, has three implemented methods, for the time being. One initializes the parameters for the field, in `List[]` form, to be compatible with the `vector<>` form in C++. Another method returns those parameters, called `gyronimo_parameters`, and the last method runs `dommaschktrace`, and is called `neatpp_solver`. One may notice, however, that `dommaschktrace` also needed particle information to be run. This is explained by the fact that this `neatpp_solver` method will only be called from within the particle tracing section of NEAT, `tracing.py`, from within a class called `ParticleOrbit`. As input, `ParticleOrbit` requires information on the particle, through a `ChargedParticle` object, along with the field being used, in this case, a `Dommaschk` object that was just defined. Finally, it also receives the `nsamples` and `tfinal` parameters. Upon being initialized in this way, it runs the particle tracing, which can then be accessed through the object's methods. A practical example of the completed code is as follows:

```
g_field = Dommaschk(m=[5,5], l=[2,4], coeff1=[1.4,19.25], coeff2=[1.4,0], B0=[1,1])
g_particle = ChargedParticle([...]Starting position and particle parameters [...])
g_orbit = ParticleOrbit(g_particle, g_field, nsamples=nsamples, tfinal=tfinal)
[...]Using tracing information by accessing the g_orbit object [...]
```

The `g_field` configuration shown is the one used throughout Section 3

The bulk of the work was the above implementation, along with unit testing of the results using a *Mathematica* notebook for verification. Some preliminary tests and results were performed and obtained. These are presented in the following section.

3 Results

With the NEAT repository, there are plotting macros already created, although not all of them are currently compatible with the implementation of Dommaschk fields shown here, such as, for example, three-dimensional trajectory plots, static or animated. This was because there is currently no output file for the above-implemented fields, and because plasma boundaries are not explicitly present, both things that are required for those built-in methods in NEAT. However, it was possible to use the examples in NEAT to re-implement the desired plots through *python*'s `matplotlib` library. In Fig. 4, phase space graphs and also the three-dimensional trajectory line are shown. Each row of the figure shows graphs corresponding to the following positions and particle settings:

```
# Initialize an alpha particle at a radius = r_initial
r_initial = 1.03#1.09#1.15  # meters, one at a time
theta_initial = 0.0  # initial poloidal angle
phi_initial = 0.00  # initial Z
B0 = 1  # Tesla, magnetic field on-axis
energy = 100000 # electron-volt
charge = 2 # times charge of proton
mass = 4  # times mass of proton
Lambda = 0.8  # = mu * B0 / energy
vpp_sign = -1  # initial sign of the parallel velocity, +1 or -1
nsamples = 2000  # resolution in time
tfinal = 3e-5  # seconds
```

How large `nsamples` needs to be was decided based on a conservation of energy plot not shown here, but output by NEAT upon using the `.plot` method of a `ParticleOrbit`. If energy had varied more than 10^{-6} in the graph, `nsamples` was increased further, and the simulation was ran once more. The field configuration is the one mentioned at the end of Section 2.3

One thing of note was the confinement of this field configuration. For energies corresponding to that of alpha particles, 3.52×10^6 eV, the particle tracing showed those specimens escaping. This makes sense, as this configuration was optimized before 1986, far before modern optimization tools, and computing

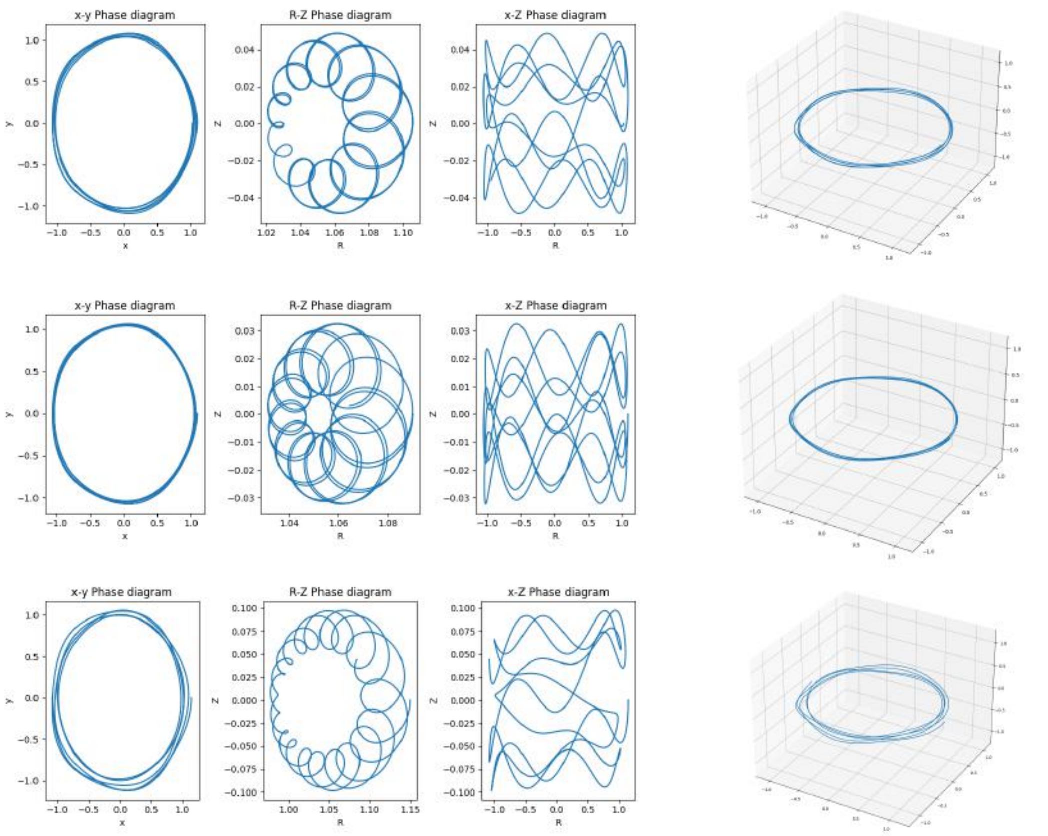

FIGURE 4: Phase-space and three-dimensional representations of trajectories of particles starting from $R = 1.03, 1.09, 1.15$, respectively

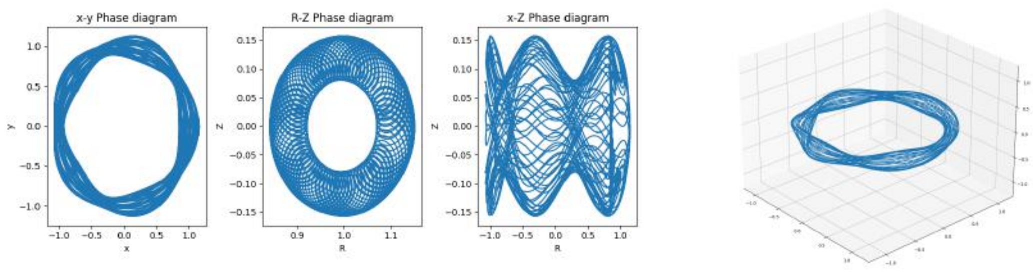

FIGURE 5: Phase-space and three-dimensional representations of field lines, originating at $R = 1.15$

power. The solution to have confined particles in the above graphs was simply to decrease the energy by an order of magnitude, to 10^5.

Another experiment that can be done in this particle tracer is a visualization of the field that has been programmed. By increasing the charge of the particle being simulated to several orders of magnitude above that of the elementary charge, while keeping the mass low, a particle that follows the field lines closely is simulated, effectively showing us said field lines. We can apply the same methods as before, while increasing the simulation time, to visualize the three-dimensional surfaces of this stellarator configuration. This can be seen in Fig. 5. Only four settings were changed, `r_initial=1.15`, the charge was set to be `1e12`, `tfinal` was set to be in the order of `1e-4`, and nsamples was increased accordingly, to keep the simulation accurate, as described before.

There is also another way of visualizing field lines, which is that of a Poincaré plot, like Fig. 2. It can be achieved by doing a scatter plot, and adding a point every time the trajectory passes by a plane of constant ϕ, in this case, the chosen plane was $\phi = 0$. As we are doing numerical simulations, a tolerance of 0.1 was added to this angle detection. Two low-resolution examples are represented in Fig. 6. Code for the right plot was as follows:

```
r_initial = 1.02  # meters
theta_initial = 0.0  # initial poloidal angle
phi_initial = 0.00  # initial Z
B0 = 1  # Tesla, magnetic field on-axis
energy = 100000 # electron-volt
charge = 100000000000000 # times charge of proton
mass = 1  # times mass of proton
Lambda = 0.8  # = mu * B0 / energy
vpp_sign = -1  # initial sign of the parallel velocity, +1 or -1
nsamples = 80000  # resolution in time
tfinal = 3e-3  # seconds
g_field = Dommaschk(m=[5,5], l=[2,4], coeff1=[1.4,19.25], coeff2=[1.4,0], B0=[B0,B0])

print("Creating Poincaré  plot")
start_time = time.time()
# Set the tolerance for checking if phi is close to a multiple of 2*pi
tolerance = 0.01
for i in range(8):
    g_particle = ChargedParticle(
    r_initial=r_initial+i*0.0257,
    theta_initial=theta_initial,
    phi_initial=phi_initial,
    energy=energy,
    Lambda=Lambda,
    charge=charge,
    mass=mass,
    vpp_sign=vpp_sign,
    )
    g_orbit = ParticleOrbit(
    g_particle, g_field, nsamples=nsamples, tfinal=tfinal
    )
    # Find the indices where phi is close to a multiple of 2*pi
    indices = np.where(np.abs(g_orbit.varphi_pos % (2*np.pi)) < tolerance)
    # Plot the values where phi passes through 2*pi
    plt.scatter(g_orbit.r_pos[indices], g_orbit.theta_pos[indices],c=np.random.rand(1,3),s=5)
total_time = time.time() - start_time
print(f"Finished in {total_time}s")
# Add labels and title to the plot
plt.xlabel('R')
plt.ylabel('Z')
plt.title('Values of (R, Z) where phi passes through 2*pi')
# Show the plot
plt.show()
```

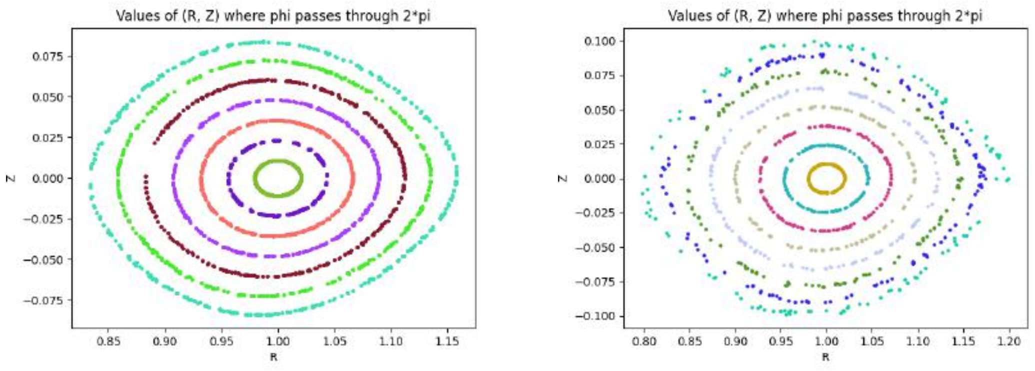

FIGURE 6: Two examples of Poincaré plots obtained, for $\phi = 0$, one with more resolution (left) and the other covering a wider interval (right)

4 Conclusion and Next Steps

As a work focused on implementation, the foremost conclusion is that anyone wanting to explore the physics of these fields should go and use them, as all of the work performed here is open-source, and publicly available in Ref. [31]. In this, there was success in building a tool that traces particles in a magnetic island scenario. By no means is this work complete, however. There are direct things missing that are relatively simple to add using already implemented examples in NEAT, such as doing animated plots of the trajectory, or ensemble particle tracing where many particles are simulated simultaneously. One step beyond that would be performing code optimizations, such as adding parallel computing. Another implementation would be using the `field_line.hh` file in *gyronimo* to simulate field lines more faithfully, instead of the method used of increasing the charge many orders of magnitude.

As mentioned in the text, NEAT is also used for the optimization of fields. It includes interfaces for doing so, with SIMSOPT and scipy as the libraries in charge of that. For this reason, the immediate next step would be to perform field optimization of Dommaschk fields, using the particle tracer to test particle confinement, providing a possible objective test of optimization, though as mentioned, there are many.

In a more mathematical vein, another future goal would be finding a way to project these Dommaschk fields onto the space of other field configurations, such that any field can be represented by a linear combination of Dommaschk fields, essentially the inverse of what we have now. A possible way to implement this would be to find a change of basis transformation between specific vacuum fields and Dommaschk potentials.

References

[1] Ritchie H. et al. "Energy". In: *Our World in Data* (2022). https://ourworldindata.org/.

[2] Abas N. et al. *Review of fossil fuels and future energy technologies.* 2015, pp. 31–49.

[3] Ramana M.V. "Nuclear power and the public". In: *Bulletin of the Atomic Scientists* 67.4 (2011), pp. 43–51.

[4] Rahman M. et al. "Assessment of energy storage technologies: A review". In: *Energy Conversion and Management* 223 (2020), p. 113295.

[5] Luo X. et al. "Overview of current development in electrical energy storage technologies and the application potential in power system operation". In: *Applied Energy* 137 (2015), pp. 511–536.

[6] Morse E. *Nuclear Fusion.* Springer, 2018.

[7] Lazerson S. et al. "Simulating fusion alpha heating in a stellarator reactor". In: *Plasma Physics and Controlled Fusion* 63 (12 2021), p. 125033.

[8] National Research Council et al. *An assessment of the prospects for inertial fusion energy.* National Academies Press, 2013.

[9] Xu Y. "A general comparison between tokamak and stellarator plasmas". In: *Matter and Radiation at Extremes* 1 (4 2016), pp. 192–200.

[10] Gates D. et al. "Recent advances in stellarator optimization". In: *Nuclear Fusion* 57 (12 2017), p. 126064.

[11] Landreman M. et al. "SIMSOPT: A flexible framework for stellarator optimization". In: *Journal of Open Source Software* 6.65 (2021), p. 3525.

[12] S. P. Hirshman et al. "Steepest-descent moment method for three-dimensional magnetohydrodynamic equilibria". In: *The Physics of Fluids* 26 (12 1983), pp. 3553–3568.

[13] Landreman M. et al. "Stellarator optimization for good magnetic surfaces at the same time as quasisymmetry". In: *Physics of Plasmas* 28 (9 2021). 092505.

[14] F L Waelbroeck. "Theory and observations of magnetic islands". In: *Nucl. Fusion* 49 (2009), p. 15.

[15] Geraldini A. et al. "An adjoint method for determining the sensitivity of island size to magnetic field variations". In: *J. Plasma Phys* 87 (2021), p. 905870302.

[16] Zarnstorff M. et al. "Physics of the compact advanced stellarator NCSX". In: *Plasma Physics and Controlled Fusion* 43 (12A 2001), A237.

[17] Lise-Marie Imbert-Gerard et al. *An Introduction to Stellarators: From magnetic fields to symmetries and optimization.* 2020. arXiv: `1908.05360 [physics.plasm-ph]`.

[18] Pedersen T. et al. "Experimental confirmation of efficient island divertor operation and successful neoclassical transport optimization in Wendelstein 7-X". In: *Nuclear Fusion* 62 (4 2022), p. 042022.

[19] Feng Y. et al. "Understanding detachment of the W7-X island divertor". In: *Nuclear Fusion* 61 (8 2021), p. 086012.

[20] S. R. Hudson et al. “Computation of multi-region relaxed magnetohydrodynamic equilibria”. In: *Physics of Plasmas* 19 (11 2012).

[21] Qu Z. et al. “Plasma Physics and Controlled Fusion Coordinate parameterisation and spectral method optimisation for Beltrami field solver in stellarator geometry”. In: *Plasma Phys. Control. Fusion* 62 (2020), p. 14.

[22] Dommaschk W. “Simple Series of Harmonic Functions for Representing Arbitrary, Nonsingular Vacuum Fields in Toroidal Regimes in Reduce and Fortran Representation”. 1978.

[23] Dommaschk W. “Solution to Stellarator Boundary Value Problems with a new Set of Simple Toroidal Harmonic Functions”. In: *Zeitschrift fur Naturforschung - Section A Journal of Physical Sciences* 36 (3 1981), pp. 251–260.

[24] Dommaschk W. “Finite Field Harmonics for Stellarators with Improved Aspect Ratio”. In: *Zeitschrift fur Naturforschung - Section A Journal of Physical Sciences* 37 (8 1982), pp. 866–875.

[25] Dommaschk W. “Representations for vacuum potentials in stellarators”. In: *Computer Physics Communications* 40 (2-3 1986), pp. 203–218.

[26] Reiman A. et al. “Calculation of three-dimensional MHD equilibria with islands and stochastic regions”. In: *Computer Physics Communications* 43 (1 1986), pp. 157–167.

[27] *prodrigs/gyronimo: gyromotion for the people, by the people.* `https://github.com/prodrigs/gyronimo`.

[28] Theodore G. Northrop. “The guiding center approximation to charged particle motion”. In: *Annals of Physics* 15 (1 1961), pp. 79–101.

[29] Landreman M. *Quasisymmetry: a hidden symmetry of magnetic fields.* 2019.

[30] Cary J. et al. “Hamiltonian theory of guiding-center motion”. In: *Rev. Mod. Phys.* 81 (2 2009), pp. 693–738.

[31] *rogeriojorge/NEAT: NEar-Axis opTimisation.* `https://github.com/rogeriojorge/NEAT`.

[32] D’haeseleer W. et al. *Flux coordinates and magnetic field structure: a guide to a fundamental tool of plasma theory.* Springer Science & Business Media, 2012.

[33] Eric W. Weisstein. “Cylindrical Coordinates”. In: *MathWorld* (2023). https://mathworld.wolfram.com/.