KINEMATIC PARTICLE CODE - USER GUIDE

James Threlfall*[*], *Thomas Neukirch, Paolo Guiliani
School of Mathematics and Statistics, University of St Andrews,
St Andrews, Fife, KY16 9SS, U.K.

November 2013

Abstract

We describe a numerical model designed to study the kinematic behaviour of individual particles in a magnetic field which obeys Ohm's law, using both relativistic and non-relativistic forms of the guiding centre approximation. This document describes how to obtain and run the code, the structure of the code itself, and various IDL scripts designed to aid data analysis. The contents of this document are aimed at code-users rather than developers. Lets do some science!

---

1 Introduction

The code is a modification of that used to study particle motion in collapsing magnetic trap (CMT) models of the solar atmosphere. Initial details (and scientific results from the initial CMT investigation) can be found in **?**; the scheme has since been used in other investigations of CMT models (**??**). The code itself can be broadly split into two parts; a global (large/MHD-scale) field is required within which individual particle dynamics are calculated. While CMT models use a time-dependent transformation to a Lagrangian frame of reference to mimick post-flare loop "shrinkage", we have employed a large scale time-dependent model of magnetic reconnection in the presence of a separator, based on the work of **?**. We have also included modules to allow the global field to be determined by the output of a `LareXd` simulation (see e.g. **?**). The latest version of our code can be downloaded from:
`http://www-solar.mcs.st-andrews.ac.uk/~jamest/files/rnrcode_v2.tar.gz`.

2 Equations and normalisation

We will now describe the normalisation of both the global fields in our separator reconnection model and the equations which describe particle dynamics. By illustrating the normalisation of the separator reconnection field, we of course assume that if other global field configurations (e.g. CMT fields, test fields or fields determined through the `Lare` plugin) are normalised in a similar way.

[*]E-mail address: jamest@mcs.st-and.ac.uk

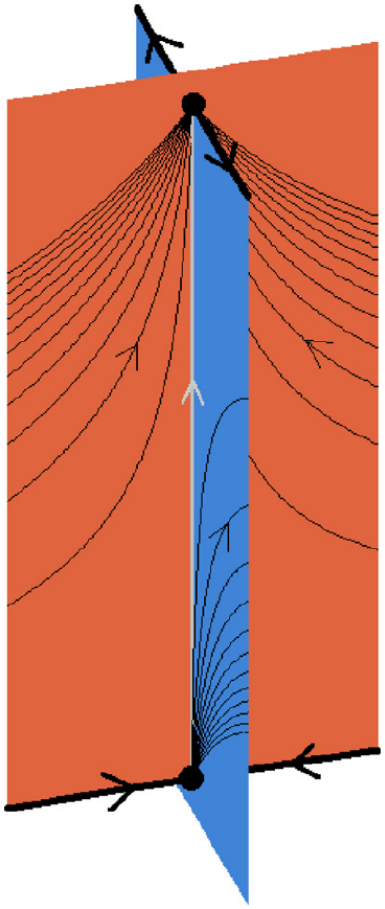


Figure 1: Original magnetic configuration of the model of **?**; the fan plane of the upper (lower) null is seen in orange (blue) and a separator (white) links both nulls at the intersection of the fan planes.

2.1 Global field

We base our global separator field model on the model of **?**; the global separator field is formed by a potential magnetic field of the form

$$\mathbf{B}_0 = \frac{b_0}{L^2}\left[x(z-3z_0)\hat{\mathbf{x}} + y(z+3z_0)\hat{\mathbf{y}} + \frac{1}{2}\left(z_0^2 - z^2 + x^2 + y^2\right)\hat{\mathbf{z}}\right],$$

with magnetic null points at $\pm z_0$, while b_0 and L determine the characteristic field strength and length-scale of the model. For the original (essentially scale-free) model of **?**, $z_0 = 5$, $b_0 = 1$ and $L = 1$. The null at $(0, 0, -z_0)$ is classified as a *positive null*, while that at $(0, 0, +z_0)$ is a *negative null* (due to the orientation of the magnetic field along the spine and fan of each null). The magnetic separator is formed by the intersection of the two fan planes associated with each null. The separator and general field configuration can be seen in Fig. 1.

With Faraday's Law coupling (time-varying) magnetic and electric fields, introducing a ring of magnetic flux of the form

$$\mathbf{B}_r = \nabla \times \left[b_1 a \exp\left(-\frac{(x-x_c)^2}{a^2} - \frac{(y-y_c)^2}{a^2} - \frac{(z-z_c)^2}{l^2}\right)\hat{\mathbf{z}}\right],$$

(centred on (x_c, y_c, z_c), with radius (in the xy plane) controlled by the parameter a, the height (in z) by

l and the field strength by b_1) induces an electric field along the separator with the form

$$\mathbf{E} = -\frac{b_1 a}{\tau} \exp\left(-\frac{(x-x_c)^2}{a^2} - \frac{(y-y_c)^2}{a^2} - \frac{(z-z_c)^2}{l^2}\right)\hat{\mathbf{z}}.$$

(provided that the time evolution satisfies Faraday's Law, i.e. that

$$\mathbf{B} = \mathbf{B}_0 + \frac{t}{\tau}\mathbf{B}_r, \qquad 0 \le t \le \tau,$$

taking place over a timescale τ).

We will input variables into the code in dimensional form, but the core equations use dimensionless variables. By choosing a normalising g magnetic field b_{scl}, lengthscale l_{scl} and timescale t_{scl}, we may therefore relate dimensional and dimensionless quantities through

$$\mathbf{B} = b_{scl}\,\bar{\mathbf{B}}, \qquad x = l_{scl}\,\bar{x}, \qquad t = t_{scl}\,\bar{t}.$$

These three basic normalising quantities also determine how velocity, electric field and energy are scaled within the code:

$$v_{scl} = \frac{l_{scl}}{t_{scl}}, \qquad e_{scl} = \frac{b_{scl}\,l_{scl}}{t_{scl}} = b_{scl}\,v_{scl}, \qquad en_{scl} = \frac{1}{2}m{v_{scl}}^2,$$

(where, for example, assessing the dimensions of Faraday's Law allows us to relate the electric field with our chosen quantities).

Therefore, the system of equations representing the behaviour of the global field structure is obtained as follows (where we have dropped the bar notation for system variables):

$$\mathbf{B}_0 = \bar{b_0}\left[x\,(z - 3\bar{z_0})\,\hat{\mathbf{x}} + y\,(z + 3\bar{z_0})\,\hat{\mathbf{y}} + \frac{1}{2}\left(\bar{z_0}^2 - z^2 + x^2 + y^2\right)\hat{\mathbf{z}}\right], \tag{1a}$$

$$\mathbf{B}_r = \frac{2\bar{b_1}}{\bar{a}}\left[-y\hat{\mathbf{x}} + x\hat{\mathbf{y}}\right]\exp\left(-\frac{(x-\bar{x_c})^2}{\bar{a}^2} - \frac{(y-\bar{y_c})^2}{\bar{a}^2} - \frac{(z-\bar{z_c})^2}{\bar{l}^2}\right), \tag{1b}$$

$$\mathbf{B} = \mathbf{B}_0 + \frac{t}{\bar{\tau}}\mathbf{B}_r, \qquad 0 \le t \le \bar{\tau}, \tag{1c}$$

$$\mathbf{E} = -\frac{\bar{b_1}\bar{a}}{\bar{\tau}}\exp\left(-\frac{(x-\bar{x_c})^2}{\bar{a}^2} - \frac{(y-\bar{y_c})^2}{\bar{a}^2} - \frac{(z-\bar{z_c})^2}{\bar{l}^2}\right)\hat{\mathbf{z}}, \tag{1d}$$

with $\bar{b_0} = \dfrac{b_0}{b_{scl}}$, $\bar{z_0} = \dfrac{z_0}{l_{scl}}$, $\bar{b_1} = \dfrac{b_1}{b_{scl}}$, $\bar{a} = \dfrac{a}{l_{scl}}$, $\bar{l} = \dfrac{l}{l_{scl}}$, $\bar{\tau} = \dfrac{\tau}{t_{scl}}$ $(= \bar{t}_{\max})$, $\dfrac{\bar{b_1}\bar{a}}{\bar{\tau}} = \dfrac{b_1 a}{\tau e_{scl}}$.

2.2 Non-relativistic particle dynamics

Having now established the global environment into which these particles will be inserted, we briefly turn our attention to the details of the particle motion itself. The drift equations solved by the code are based upon **?**:

$$\frac{dv_\parallel}{dt} = \frac{qE_\parallel}{m_0} - \frac{\mu_B}{m_0}\frac{\partial B}{\partial s} + \mathbf{u}_E \cdot \left(\frac{\partial \mathbf{b}}{\partial t} + v_\parallel \frac{\partial \mathbf{b}}{\partial s} + (\mathbf{u}_E \cdot \boldsymbol{\nabla})\,\mathbf{b}\right), \tag{2a}$$

$$\dot{\mathbf{R}}_\perp = \frac{\mathbf{b}}{B} \times \left[-\mathbf{E} + \frac{\mu_B}{q}\boldsymbol{\nabla}B + \frac{m_0}{q}\left(v_\parallel \frac{\partial \mathbf{b}}{\partial t} + v_\parallel^2 \frac{\partial \mathbf{b}}{\partial s} + v_\parallel\,(\mathbf{u}_E \cdot \boldsymbol{\nabla})\,\mathbf{b} + \frac{\partial \mathbf{u}_E}{\partial t} + v_\parallel \frac{\partial \mathbf{u}_E}{\partial s} + (\mathbf{u}_E \cdot \boldsymbol{\nabla})\,\mathbf{u}_E\right)\right], \tag{2b}$$

$$\frac{d}{dt}E_K = q\dot{\mathbf{R}} \cdot \mathbf{E} + \mu_B \frac{\partial B}{\partial t}, \tag{2c}$$

$$E_K = \frac{m_0 {v_\parallel}^2}{2} + \mu_B B + \frac{m_0 {u_E}^2}{2}, \tag{2d}$$

where $\mu_B = 1/2mv_g^2/B$ is the magnetic moment for a particle with gyro-velocity v_g, subject to $E \times B$ drift velocity $\mathbf{u}_E (= \mathbf{E} \times \mathbf{b}/B)$, $\mathbf{b} (= \mathbf{B}/B)$ is the unit vector in the direction of the local magnetic field, $\mathbf{R}$ is the vector location of the guiding centre, $v_\parallel (= \mathbf{b} \cdot \dot{\mathbf{R}})$ and $E_\parallel (= \mathbf{b} \cdot \mathbf{E})$ are the magnitudes of the velocity and electric field parallel to the local magnetic field, $\dot{\mathbf{R}}_\perp (= \dot{\mathbf{R}} - v_\parallel \mathbf{b})$ is the component of velocity perpendicular to $\mathbf{b}$, and s is a line element parallel to $\mathbf{b}$. We will only consider electrons in our model, thereby fixing mass $m = 1.9 \times 10^{-31}\text{kg}$ and charge $q = |e| = 1.6022 \times 10^{-19}\text{C}$.

Non-dimensionalising the first of these equations (and remembering that $\mathbf{b}$ is a unit vector, and as such has no units, i.e. $\mathbf{b} \equiv \bar{\mathbf{b}}$) proceeds as follows

$$\frac{v_{scl}}{t_{scl}} \frac{d\bar{v}_\parallel}{d\bar{t}} = \frac{q}{m} e_{scl} \bar{E}_\parallel - \frac{v_{scl}{}^2}{\cancel{b_{scl}}} \frac{\cancel{b_{scl}}}{l_{scl}} \bar{\mu}_B \frac{\partial \bar{\mathbf{B}}}{\partial \bar{s}} + v_{scl} \bar{\mathbf{u}}_E \cdot \left[\frac{1}{t_{scl}} \frac{\partial \bar{\mathbf{b}}}{\partial \bar{t}} + \frac{v_{scl}}{l_{scl}} \frac{\partial \bar{\mathbf{b}}}{\partial \bar{s}} \bar{v}_\parallel + \frac{v_{scl}}{l_{scl}} \left(\bar{\mathbf{u}}_E \cdot \bar{\boldsymbol{\nabla}} \right) \bar{\mathbf{b}} \right],$$

$$\frac{d\bar{v}_\parallel}{d\bar{t}} = \frac{q}{m} \frac{\cancel{v_{scl}} b_{scl} t_{scl}}{\cancel{v_{scl}}} \bar{E}_\parallel - \frac{v_{scl}{}^2 \cancel{t_{scl}}}{\cancel{v_{scl}} l_{scl}} \bar{\mu}_B \frac{\partial \bar{\mathbf{B}}}{\partial \bar{s}} + \frac{\cancel{t_{scl}} \cancel{v_{scl}}}{\cancel{v_{scl}} \cancel{t_{scl}}} \bar{\mathbf{u}}_E \cdot \left[\frac{\partial \bar{\mathbf{b}}}{\partial \bar{t}} + \frac{\partial \bar{\mathbf{b}}}{\partial \bar{s}} \bar{v}_\parallel + \left(\bar{\mathbf{u}}_E \cdot \bar{\boldsymbol{\nabla}} \right) \bar{\mathbf{b}} \right].$$

Thus only a single normalising factor remains, which may be expressed in terms of a cyclotron or gyro-frequency, $\Omega (= qB/m)$. We can ultimately express the drift equations in normalised form using such a frequency (dropping the bar notation for system variables), finding

$$\frac{dv_\parallel}{dt} = \Omega_e^{scl} t_{scl} E_\parallel - \mu_B \frac{\partial \mathbf{B}}{\partial s} + \mathbf{u}_E \cdot \left[\frac{\partial \mathbf{b}}{\partial t} + \frac{\partial \mathbf{b}}{\partial s} v_\parallel + \left(\mathbf{u}_E \cdot \boldsymbol{\nabla} \right) \mathbf{b} \right], \tag{3a}$$

$$\dot{\mathbf{R}}_\perp = \mathbf{u}_E + \frac{1}{\Omega_e^{scl} t_{scl}} \frac{\mathbf{b}}{B} \times \left[\mu_B \boldsymbol{\nabla} B + v_\parallel \frac{\partial \mathbf{b}}{\partial t} + v_\parallel^2 \frac{\partial \mathbf{b}}{\partial s} + v_\parallel \left(\mathbf{u}_E \cdot \boldsymbol{\nabla} \right) \mathbf{b} + \frac{\partial \mathbf{u}_E}{\partial t} + v_\parallel \frac{\partial \mathbf{u}_E}{\partial s} + \left(\mathbf{u}_E \cdot \boldsymbol{\nabla} \right) \mathbf{u}_E \right] \tag{3b}$$

$$\frac{d}{dt} E_K = \Omega_e^{scl} t_{scl} \dot{\mathbf{R}} \cdot \mathbf{E} + \mu_B \frac{\partial B}{\partial t}, \tag{3c}$$

$$E_K = {v_\parallel}^2 + {v_\perp}^2 + {u_E}^2, \tag{3d}$$

where $\Omega_e^{scl} = \dfrac{q\, b_{scl}}{m}$. The factor of $\Omega_e^{scl} t_{scl}$ thus plays a key role in controlling the scales at which certain guiding centre drifts become important.

2.3 Relativistic particle dynamics

In cases where particle velocities become a significant fraction of the speed of light, c, relativistic effects are not accounted for in the particle dynamics outlined in the previous section. We therefore also include the fully relativistic guiding centre equations, also outlined in **?** (based on the treatment of **?**):

$$m_0 \frac{d}{dt} \left(\gamma v_\parallel \right) = \frac{dp_\parallel}{dt} = m_0 \gamma \mathbf{u}_E \cdot \frac{d\mathbf{b}}{dt} + qE_\parallel - \frac{\mu_r}{\gamma} \frac{\partial B}{\partial s}, \tag{4a}$$

$$\dot{\mathbf{R}}_\perp = \frac{\mathbf{b}}{B^{\star\star}} \times \left[-\mathbf{E}^{\star\star} + \frac{\mu_r}{\gamma q} \boldsymbol{\nabla} B^\star + \frac{m_0 \gamma}{q} \left(v_\parallel \frac{d\mathbf{b}}{dt} + \frac{d\mathbf{u}_E}{dt} \right) + \frac{1}{c^2} \left(v_\parallel E_\parallel \mathbf{u}_E + \frac{\mu_r}{\gamma q} \mathbf{u}_E \frac{\partial}{\partial t} B^\star \right) \right], \tag{4b}$$

$$\frac{d}{dt} \left(m_0 c^2 \gamma \right) = q \left(\dot{\mathbf{R}}_\perp + \mathbf{b} v_\parallel \right) \cdot \mathbf{E} + \frac{\mu_r}{\gamma} \frac{\partial B^\star}{\partial t}, \tag{4c}$$

$$\mu_r = \frac{m_0 \gamma^2 v_\perp^2}{2B} = \frac{{p_\perp}^2}{2B}, \tag{4d}$$

where γ is the Lorentz factor ($\gamma = 1/(1 - v^2/c^2) = c^2/(v^2 - c^2)$) and the non-relativistic adiabatic invariant (μ_B) is now modified to depend on perpendicular *momentum* ($p_\perp$); the corresponding relativistic invariant is μ_r. Finally, several quantities now also depend on the ratio of perpendicular electric field ($E_\perp$) to the size of the magnetic field (B); for a given quantity H, $H^\star$ and $H^{\star\star}$ are defined as

$$H^\star = H\left(1 - \frac{1}{c^2}\frac{{E_\perp}^2}{B^2}\right)^{\frac{1}{2}}, \qquad H^{\star\star} = H\left(1 - \frac{1}{c^2}\frac{{E_\perp}^2}{B^2}\right).$$

Such quantities retain their original dimensions; in normalised form, these factors become

$$\bar{H}^\star = \bar{H}\left(1 - \frac{{v_{scl}}^2}{c^2}\frac{\bar{E}_\perp^{\,2}}{\bar{B}^2}\right)^{\frac{1}{2}}, \qquad \bar{H}^{\star\star} = \bar{H}\left(1 - \frac{{v_{scl}}^2}{c^2}\frac{\bar{E}_\perp^{\,2}}{\bar{B}^2}\right),$$

i.e $H^\star$ and $H^{\star\star}$ do not modify the dimensions of H.

With this in mind, we proceed to normalise the relativistic guiding centre equations. In a similar manner to that outlined in Section 2.2, we find

$$\frac{du_\parallel}{dt} = \frac{d}{dt}\left(\gamma v_\parallel\right) = \gamma \mathbf{u}_E \cdot \frac{d\mathbf{b}}{dt} + \Omega_e^{scl} t_{scl} E_\parallel - \frac{\mu_r}{\gamma}\frac{\partial B}{\partial s}, \tag{5a}$$

$$\dot{\mathbf{R}}_\perp = \mathbf{u}_E + \frac{\mathbf{b}}{B^{\star\star}} \times \left\{\frac{1}{\Omega_e^{scl} t_{scl}}\left[\frac{\mu_r}{\gamma}\left(\boldsymbol{\nabla} B^\star + \frac{{v_{scl}}^2}{c^2}\mathbf{u}_E \frac{\partial B^\star}{\partial t}\right) + u_\parallel \frac{d\mathbf{b}}{dt} + \gamma \frac{d\mathbf{u}_E}{dt}\right] + \frac{{v_{scl}}^2}{c^2}\frac{u_\parallel}{\gamma} E_\parallel \mathbf{u}_E\right\}, \tag{5b}$$

$$\frac{d\gamma}{dt} = \frac{{v_{scl}}^2}{c^2}\left[\Omega_e^{scl} t_{scl}\left(\dot{\mathbf{R}}_\perp + \frac{u_\parallel}{\gamma}\mathbf{b}\right)\cdot \mathbf{E} + \frac{\mu_r}{\gamma}\frac{\partial B^\star}{\partial t}\right], \tag{5c}$$

$$\mu_r = \frac{\gamma^2 v_\perp^2}{B}, \tag{5d}$$

where the magnetic moment is normalised by $\mu_{scl} (= m_0 {v_{scl}}^2/2b_{scl})$, and we have formed a relativistic parallel velocity $u_\parallel (= \gamma v_\parallel)$.

Equations 5a, 5b and 5c are, respectively, the relativistic forms of Eqs. 3a, 3b and 3c (as Eqs. 4a, 4b and 4c are the relativistic forms of Eqs. 2a, 2b and 2c). Relativistic effects not only modify existing terms in the non-relativistic equations, but also introduce two new terms in Eq. 5b in the direction of $E_\perp$ (i.e. in the $\mathbf{b} \times \mathbf{u}_E$ direction). Both of these additional terms are scaled by ${v_{scl}}^2/c^2$, and as such are purely relativistic.

3 Code structure

Various aspects of the code are distributed across several subdirectories, but are all compiled through the top-level Makefile (detailed in Section 5). The default code layout can be seen in Fig. 2, which outlines the location of the core modules and program files. The core code is entirely stored in the `src/core` sub-directory. This contains the two main program files and module containing global constants, with four sub-directories `nr_rkmods`, `r_rkmods`, `othermods` and `lare_io` which (unsurprisingly) contain program modules. All program files have a `.f90` extension, with the program modules indicated by a name containing "`_mod`". First the files in the highest source code directory `src/core`; this directory contains two key program files:

- `nr_main.f90`: The main non-relativistic program. Calculates initial position of each particle in x, y, z, α, ke space, before passing off each particle in turn to the Runge-Kutta solver `rkdrive`.

- `r_main.f90`: The main relativistic program. Rather than passing values of $v_\parallel$ and R to the RK solvers, the relativistic version passes γ, $u_\parallel$ and dR/dt to the RK solver.
- `global_mod.f90`: Global constant storage module. Contains/initialises all constants/variables used by multiple subroutines; this includes normalising parameters, a, l, (x_c, y_c, z_c), b_0, b_1, τ and the extent to which the Runge-Kutta solver will continue to operate $(x_{\rm end}, y_{\rm end}, z_{\rm end})$.

Two versions of the Runge-Kutta (RK) subroutines are used, depending on whether the relativistic or non-relativistic program is in use. These are grouped in `src/core/nr_rkmods/` and `src/core/r_rkmods/` respectively. Each contains

- `rkdrive.f90`: The controlling RK program. For a specific particle, evolves the parallel velocity and particle position (and, in the relativistic case, γ) subject to the drifts set up in the relevant version of `othermods/derivs_mod.f90`.
- `rkqs_mod.f90`: Solves a single RK step (repeatedly called by the other routines).
- `rkck_mod.f90`: The "stepper" module, repeatedly stepping through the problem while monitoring truncation error. Outputs new positions, velocity (γ) and time.

The RK modules do not have explicit knowledge of the global field/flows in which the particle is located, nor the drift equations. We can prescribe this information using several modules in `src/core/othermods/`.

- `field_selector_mod.f90`: selects which field setup is to be used; current choices are the fields defined in `separatorfields_mod.f90`, `CMTfields_mod.f90`, `testfields_mod.f90` or even fields from LareXd output files (stored in the `src/core/lare_io/` directory).
- `separatorfields_mod.f90`: field setup for separator reconnection investigations.
- `CMTfields_mod.f90`: field setup for CMT models.
- `testfields_mod.f90`: simple field configurations (for testing).
- `nr_derivs_mod.f90`: calculates the RHS's of Eqs. 3 for a given position and time (requires knowledge of environment from field subroutine).
- `r_derivs_mod.f90`: calculates the RHS's of Eqs. 5 for a given position and time.
- `products_mod.f90`: contains simple dot and cross product functions.

Subroutines required to adapt globals fields from datafiles output by LareXd are found in `src/core/lare_io/`. Many of these subroutines have been lifted verbatim from the LareXd source code, hence we only describe the routines which we have modified:

- `lare_fields_mod.f90`: uses intrinsic MPI routines to read in data from `****.cfd` files.
- `lare_functions_mod.f90`: contains a function to perform trilinear interpolation of quantities at a given point, and a string comparison function.

In order for the LareXd routines to work, the datafiles should be placed in the `src` subdirectory; this can be changed, for example in `src/core/global_mod.f90`.
In its present state, the code can easily be switched between separator, CMT, test or LareXd model fields using the `FMOD` variable in `global_mod.f90`. Switching between models requires the code to be remade. Compilation and Makefile contents are discussed in more detail in Sec. 5.

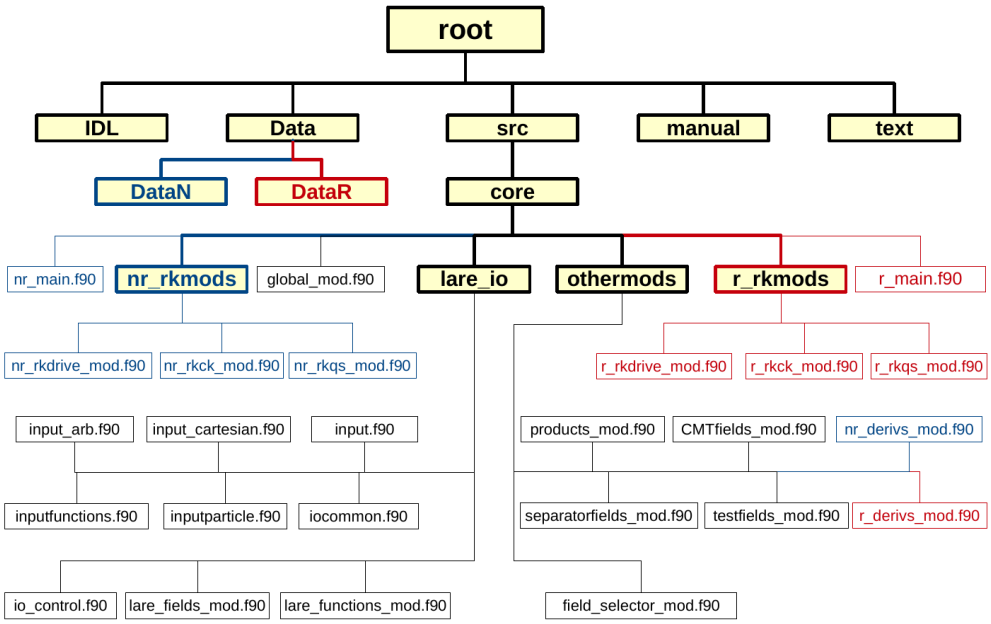


Figure 2: code layout; filled rectangles represent folders, blue items are components of the non-relativistic code, while red items are components of the relativistic code; black represents general (global) files/folders used by both codes.

4 Control deck: newinput.dat

Many of the input parameters for the code are specified in a control deck, located at `newinput.dat`. The contents of this file are read as an array, for which character spacing is important; in its current format, one cannot place comments in the control deck. The simplest way to explain how to set up a problem using this file is to go through the control deck line-by-line. A line from the code will be seen in `courier` font.

The file begins with the line
`&inputdata`
which begins the array of input data.
`T1=105.`
`T2=200.`
The first variable to be specified is time, which runs from `T1` to `T2` and is in seconds (provided t_{scl} is also defined in seconds). To avoid complications for the CMT model, the initial time `T1` is usually set to a value greater than 100s. For the separator reconnection field setup, we retain this view (for simplicity) and subtract `T1` from every time in the problem. For the current values of `T1` and `T2`, the code would run from 0s and last approximately 95s (`T2`$-$`T1`).
`H1=0.00001`
`EPS=1.0E-15`
Of the next two values, `H1` represents the initial step-size for use with the RK algorithm, while `EPS` is a tolerance for the RK solvers. Increasing either value will increase the speed at which the code yields a solution, but with a trade-off in accuracy.
`AlphaSteps=2`
`AlphaMin=45.0`
`AlphaMax=45.0`

We now begin specifying particle parameters. The first is α, the initial pitch angle distribution for our set of particles. `Alphasteps` specifies the (integer) number of times the range of α is divided, between an upper and lower bound (`Alphamin` and `Alphamax`, both in degrees). For unspecified reasons, setting `Alphasteps`$= 2$ yields only one value of α for all the particles in the experiment. Increasing `Alphasteps` above two will cause the range to divide, between the upper and lower bounds (effectively, there are `Alphasteps`-1 particles in α space). α represents how the initial energy is divided between perpendicular and parallel components; $\alpha \sim 0^o$ will cause all the energy to go to velocity along the magnetic field lines, while $\alpha \sim 90^o$ will cause all the energy to go into perpendicular motion. In addition, $\alpha \sim 180^o$ will also cause entirely parallel motion, but in the opposite direction to the direction of the magnetic field.

```
R1(1)=-0.25E2
R2(1)=0.25E2
RSTEPS(1)=16
```

We then create three 3-element arrays, containing the start, end and number of positions in x, y, z space. The first three lines above are to specify where the grid of x starts, ends, and how many positions in between these two values. The range `R2(1)`$-$`R1(1)` is divided by `RSTEPS(1)` divisions. Thus if `RSTEPS(1)`$= 1$, then every particle in the experiment will begin at the same position in x. If `RSTEPS(1)`$= 2$, then the two possible values of x are `R1(1)` and `R2(1)`, respectively. In the above example, 16 positions from $-25\mathrm{m} \leq \mathrm{x} \leq 25\mathrm{m}$ are chosen (provided the lengthscale, l_{scl}, is also specified in metres). `R1` and `R2` are real values, while `RSTEPS` **must** be an integer.

```
R1(2)=-0.25E2
R2(2)=0.25E2
RSTEPS(2)=16
```

Similarly the initial y positions have been chosen, for `RSTEPS(2)` particles from minimum `R1(2)` to maximum `R2(2)`.

```
R1(3)=-2.0E7
R2(3)=0.0E7
RSTEPS(3)=5
```

Finally, the initial z positions are chosen, distributed from `R1(3)` to `R2(3)`, `RSTEPS(3)` times.

```
EkinLow=2e2
EkinHigh=2e2
EkinSteps=1/
```

The final parameter to be chosen is the initial kinetic energy. Once again, it is possible to set this to a single value (as shown above) or for a range of initial kinetic energies. Unless otherwise specified, the values of `EkinLow` and `EkinHigh` are in units of eV.

The `newinput.dat` control deck only controls the initial particle parameters. These values must be compatible with the other constants chosen in `global_mod.f90`; the code will terminate if (for example) the initial particle positions are beyond the boundary specified by $x_{\mathrm{end}}, y_{\mathrm{end}}, z_{\mathrm{end}}$. It is worth noting that altering values held in `newinput.dat` does **NOT** require a code recompilation before re-running the code. However, if you intend to reduce the number of particles for the next simulation, remember that you may be left with additional output datafiles from previous runs, which might cause problems when you come to analyse your (now unexpectedly large) dataset.

5 Makefile

The code is written using Fortran 90, and should work on all compilers which support this standard. However, the LareXd interface routines are written in MPI; LareXd datafiles are structured using blocks, not records. Therefore, for simplicity, the code is compiled against MPI compilers, such as mpif90.

To compile the code, simply type
`make`
which will compile both the non-relativistic and relativistic codes (using the default supplied compiler flags), and generate two binaries called
`bin/nrtest`
and
`bin/rtest`
The non-relativistic code can then be run on a workstation by typing
`./bin/nrtest`
at the command line, while
`./bin/rtest`
will run the relativistic code. It is possible to compile and run one version of the code without the other. By typing
`make ./bin/nrtest`
the Makefile will only compile the non-relativistic code. Similarly
`make ./bin/rtest`
yields only the binary containing the relativistic code.
The default data directory specified in the Makefile is `/Data`, with the non-relativistic data being output to `/Data/DataN` and relativistic data being output to `/Data/DataR`; if these settings are changed in the Makefile, be sure to update the output data character(s) in the `/src/core/global_mod.f90` file before you try to run the code. By default, the binary files are automatically placed in the `./bin` subdirectory, while the compiler intermediate files go into the `obj` subdirectory.
To remove the compiled binaries, and the compiler intermediate files, type
`make clean`
To remove the intermediate files only, type
`make tidy`

5.1 Makefile details

The included Makefile is relatively complex and the main working parts should not be changed without a specific reason. However, it is possible for the user to customise various parts of the Makefile for several different reasons. The first customisable option is the compiler flags:

```
# Set the compiler flags
#FFLAGS = -C -g -traceback -check all -warn all # mpif90 error flags
FFLAGS = -O3
```

The `FFLAGS` environment variable is used to control the general command line parameters passed to the MPIFortran compiler.

- `-O3` is a portable general purpose flag for turning on most optimisations on most machines.
- `-C` tests for attempts to write outside the bounds of an array and will print the location of the error and the name of the array which causes the problem (again good for debugging, but turns off **ALL** compiler optimisation).
- `-g` places symbolic debugging information in the executable. This is sometimes useful, but often just annoying.
- `-traceback` enables a stack trace if the program crashes.

- `-check all` performs all runtime checks (includes bounds checking) can be useful when debugging but can slow down compilation and runtime, which can be annoying when you are sure the code works satisfactorily.
- `-warn all` tells gfortran to generate warnings about all common sources of bugs. Also falls into useful but annoying category.

The current settings reflect a successfully working code that requires no error checking. Switching between the two values of `FFLAGS` seen above may be useful when setting up a new problem, but can severely limit code performance if the debugging options are left on.

```
# Set some of the build parameters
TARGETN = nrtest
TARGETR = rtest
```

`TARGETN` and `TARGETR` control the name of the output binary files in the `bin` subdirectory. Normally, there is no need to change this.

```
DATDIR = Data
```

One may also change the output directory (for example, re-running the same experiment using a different field configuration); this is altered using the `DATDIR` variable (but making sure that the

6 Output

We will now discuss the output from the code, and how one extracts scientific information from it.

6.1 IDL

The IDL routines used for loading/visualising data are all held within the IDL directory. The most useful one (upon which everything else is build) is `getdata.pro`, which takes a given particle in the grid, and loads the data from a specific particle into a given data structure. The remaining procedures in `IDL/` are used to set up various aspects of 3D visualisation, and are included as a cheeky bonus!

6.1.1 Starting IDL

In order to load in the IDL routines, the IDL script `Start.pro` needs to be run. This can be run from the command line on starting IDL:

```
$ idl Start.pro
```

6.1.2 `getndata` and `getrdata`

These functions load the data for a given particle number, for both relativistic and non-relativistic schemes. We will focus only on the non-relativistic code here; the syntax and usage for the relativistic version is identical. The syntax is:

```
IDL>  ds = getndata(a[,/flag1,...,/empty,wkdir=directory])
```

After running this `ds` will contain the snapshot data structure for the (non-relativistic) code. Here, `a` is the number associated with the datafile you wish to load. Keeping in mind that there is no zeroth particle, providing a particle number `a` will initially only provide the position and time information from the `RV``a''.dat` file. If you wish to load additional quantities into the data structure `ds`, you must specify additional flags to the `getdata` call. To load all the variables, you can use the `/all` switch, i.e.

```
IDL>  ds = getndata(1,/all)
```

will load all the variables associated with the first particle in the array of parameter space defined in `newinput.dat`. The `wkdir=` option allows data to be loaded from a directory other than `/Data`, i.e. in the first example `directory` should be replaced with a string containing the name of the directory. Some further examples:

```
IDL>  ds = getndata(10)
```

This will only load the position and time variables for the 10th particle in the default data directory `Data`.

```
IDL>  ds = getndata(7,/fields,wkdir="OtherDir")
```

In addition to x, y, z, t, this will also load B_x, B_y, B_z and E_x, E_y, E_z from particle 7 in the data directory `OtherDir` into the structure. The various optional flags (and their related quantities) are defined in Table 1.

6.2 IDL Structure Tips

This section gives a brief overview of how to use IDL structures. The easiest way of doing this is with an example, to create the data structure:

```
IDL>  ds = getndata(1, /all)
```

To view the elements contained in a structure the `help` command can be used with the `/struct` flag:

```
IDL> help, ds, /str
** Structure <11d16b68>, 16 tags, length=52000, data length=52000, refs=1:
   NAME            STRING    '../Data/RV00000001.dat'
   T               DOUBLE    Array[342]
   X               DOUBLE    Array[342]
   Y               DOUBLE    Array[342]
   Z               DOUBLE    Array[342]
   VPAR            DOUBLE    Array[342]
   MUBSQUARED      DOUBLE    Array[342]
   RDOTPERP        DOUBLE    Array[342]
   B               DOUBLE    Array[3, 342]
   E               DOUBLE    Array[3, 342]
   E2              DOUBLE    Array[342]
   E3              DOUBLE    Array[342]
   EK              DOUBLE    Array[342]
   GYROF           DOUBLE    Array[342]
   GYROP           DOUBLE    Array[342]
   GYROR           DOUBLE    Array[342]
```

Ignore the first line. The rest of the lines are the elements, the variable type and (if the variable is an array) the array dimensions or the data value. To access the elements use `datastructure.element`, for example to print the filename:

```
IDL> print, ds.name
../Data/DataN/RV00000001.dat
```

6.3 Variable tables

The code will, by default, only output certain variables for a single particle over the course of the simulation. It is also possible to use the `/all` switch to output all the variables into the required structure. A catalog of all the system variables (including the switches required to output them via `getdata`) can be found in Table 1.

Default variables:
These are all included in a standard `getdata` call.

name	String containing the filename of the file the data was loaded from.
t	Array containing the time of each output dump.
x	Array containing the x coordinate of the particle at each output dump.
y	Array containing the y coordinate of the particle at each output dump.
z	Array containing the z coordinate of the particle at each output dump.

Simulation variables:
Additional variables are included using `/all` or individual switches.

Name	Description	Switch.
B	Three components of the magnetic field (B_x, B_y, B_z).	`/fields`.
E	Three components of the electric field (E_x, E_y, E_z).	`/fields`.
vpar	Array containing the parallel component of velocity, $v_\parallel$.	`/vpar`.
epar	Array containing the parallel Electric field, $E_\parallel$.	`/epar`.
muB	Array containing $\mu B = 1/2mv_\perp^2$.	`/muB`.
rdotperp	Perpendicular displacement velocity, $\|\dot{\mathbf{R}}_\perp\|$.	`/rdotperp`.
gyrof	The gyrofrequency, Ω as a function of time.	`/gyro`.
gyrop	The gyroperiod, $1/\Omega$, as a function of time.	`/gyro`.
gyror	The gyroradius, r_g, as a function of time.	`/gyro`.
ue	Three components of $\mathbf{E} \times \mathbf{B}$ drift velocity.	`/drift`.
e2	Energy associated with the magnetic moment, $\mu\|\mathbf{B}\|$.	`/ke`.
e3	Energy of the $E \times B$ drift velocity.	`/ke`.
ek	The total kinetic energy of the particle $(e2 + e3 + 1/2mv_\parallel^2)$.	`/ke`.

only active using `getrdata.pro`

Name	Description	Switch.
gamma	Lorentz factor γ.	`/rel`.
upar	Parallel component of relativistic velocity $u_\parallel (= \gamma v_\parallel)$	`/rel`.
ek	The total kinetic energy of the particle $\gamma m_0 c^2$.	`/ke`.

Table 1: List of all the elements of the data structure and a brief description.

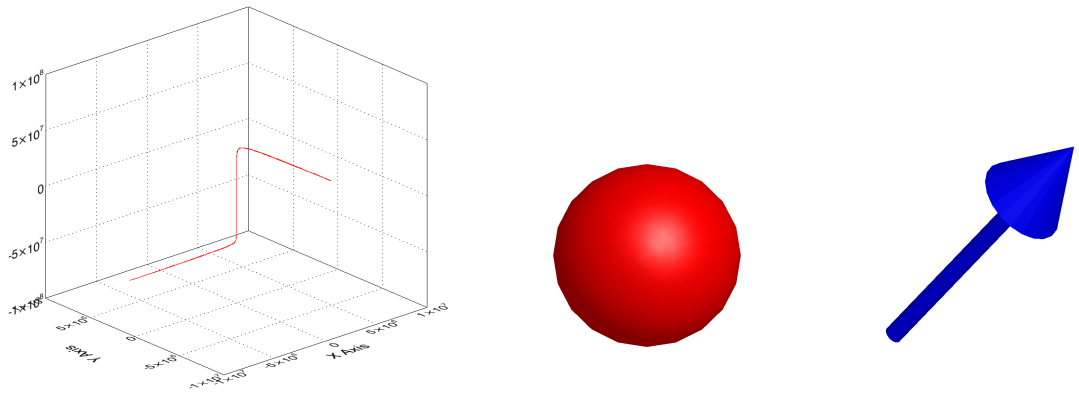


Figure 3: Example of basic plot tools used in more complicated routines; a single particle path as shown in `xplot3d`, together with two 3D graphics objects we will use in our routines; orbs and vectors.

6.4 3D data visualisation with IDL

In addition to some custom routines to read in the data from the output files, we have also included routines to allow the user to visualise magnetic fields, particle paths and quantities in 3D space. The `Start.pro` file will load in several different subroutines to assist with this process. This is very much still work-in-progress; we will outline some basic features of some routines. It is likely we will also include several example programs with these routines, in order to show how one might use them to study specific quantities or problems. The objective of this section is to show what is possible, **not** to document **every** routine in the `IDL` directory, so be warned!

6.4.1 Basic plot tools

Before we describe more complex programs, we will first briefly outline some basic 3D plotting tools. Many of our codes make use of the intrinsic IDL routine `xplot3d`. This creates an interactive 3D graphics object. For example, having read in a series of particle positions using `getndata` or `getrdata`, we may plot the trajectory of the particle using

```
IDL> xplot3d, ds.x, ds.y, ds.z
```

which yields a 3D interactive particle path as shown in Fig. 3. `xplot3d` can also overlay certain 3D symbols at certain positions. In order to display scalar quantities or specific positions, we will use IDL to create orbs, in the following manner

```
IDL> orb = OBJ_NEW('orb', COLOR=[255, 0 ,0])
IDL> osym = OBJ_NEW('IDLgrSymbol', oOrb)
```

This stores a red orb as a 3D symbol, ready to be overlaid into `xplot3d` (shown in the central panel of Fig. 3). Using a custom routine, one can also generate arrow symbols, in order to specify vector quantities at a given position. This is done using the `mk_vector` routine, and is used as follows

```
IDL> oModel=obj_new('IDLgrModel')
IDL> oModel->add,mk_vector([1,1,1],color=[0,0,255])
IDL> asym = OBJ_NEW('IDLgrSymbol', oModel)
```

which generates a blue arrow, extending one unit in x, y, and z (see right panel of Fig. 3). Placing either symbol at a specific position requires the use of the `symbol` keyword in `xplot3d`. If we wanted to overlay each symbol at a certain position on the previous diagram, for example, we might use

```
IDL> xplot3d, [x1,x1], [y1,y1],[z1,z1], SYMBOL=osym, /OVERPLOT
IDL> xplot3d, [x2,x2], [y2,y2],[z2,z2], SYMBOL=asym, /OVERPLOT
```

where $(x1, y1, z1)$ and $(x2, y2, z2)$ are the positions of the centre of the orb and the base of the arrow. Many of the codes which follow build on these simple initial principles. [NB. one should ensure the symbols are rescaled to match the plot range specified in `xplot3d`].

6.4.2 ODEINT

It is often useful to display the magnetic field in the simulation at a given time. We already know the equations which describe the field (Eqs. 1a-1c); like good mathematicians, we can easily solve for the fieldlines in 3d space by solving the coupled set of ODEs:

$$\frac{ds}{B} = \frac{dx}{B_x} = \frac{dy}{B_y} = \frac{dz}{B_z}, \qquad B = |\mathbf{B}| = \sqrt{B_x^2 + B_y^2 + B_z^2}.$$

The `ODEINT.pro` program solves this coupled set of ODEs for an initial position in x, y, z space using another RK algorithm (with the field specified in the `derivs` subroutine). `ODEINT` outputs a set of positions in x, y, z which traces out the location of a single field line (a line of constant magnetic flux).

6.4.3 FLINE

To show the global structure of the magnetic field, we can make repeated use of `ODEINT` for a grid of initial positions, in a program called `FLINE.pro`. This essentially creates a grid of initial positions, and repeatedly calls `ODEINT` for each specified position. `FLINE` also makes use of the `mk_vector` subroutine (to draw arrows in the direction of the magnetic field) and IDL orbs (to indicate the locations of the magnetic null points). All the options (the initial starting positions for the `ODEINT` routine, symbols and colours, 3D plotting area etc) are all specified within the routine itself. To run, simply type

```
IDL> FLINE
```

which should yield a 3D interactive object like those seen in Fig. 4. It is worth noting that the examples shown in this image are for the initial (potential) field only ($t = 0$). The `FLINE` program is also capable of plotting the magnetic field lines at **any** time ($0 \leq t \leq \tau$).

6.4.4 particletrack and iniorb

To streamline the process of reading in data from individual particles and displaying this information on screen, we have created a routine called `particletrack`. This routine automatically reads in and displays/overlays the particle position as a function of time, together with optional extras (initial and final particle positions, locations of bounce points, colour of field line depends on some quantity, lengthscales of global plot, etc) using a single command. For example, to draw the trajectory of the 55th particle in the simulation, one need only type

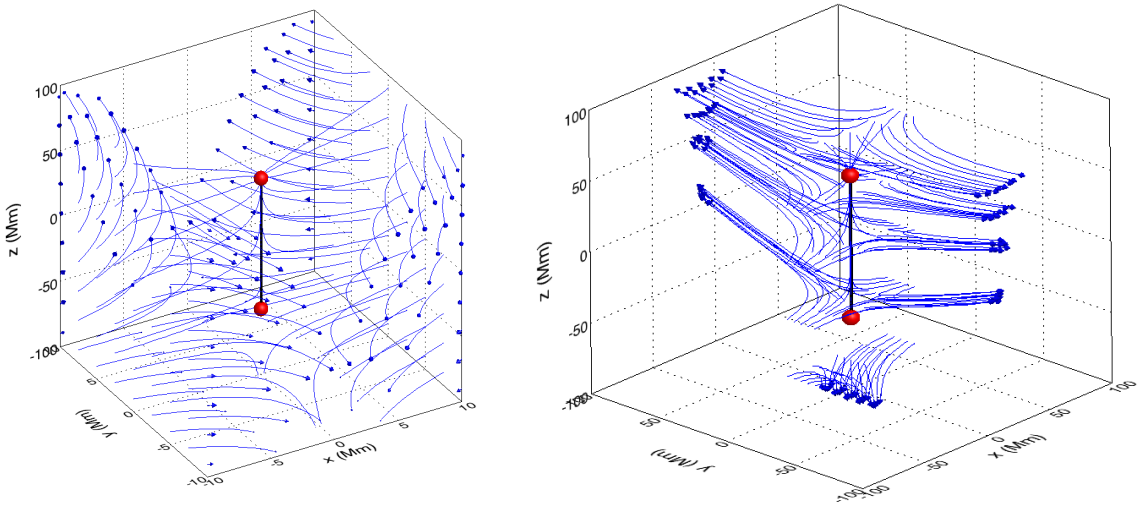
```
IDL> particletrack, 55, xyzt
```

where a 1024 element particle path is output to a variable called `xyzt` automatically. One can then build up several of these paths in the same window, using the `op` switch. For example

```
IDL> particletrack, 65, xyzt, /op
```

would overplot the 65th particle path on to the already created window.

```
IDL> particletrack, 75, xyzt, /op, /bsymb, /symb, lcol=[153,0,153]
```



(a) FLINE example 1.

(b) FLINE example 2.

Figure 4: Illustration of selected field lines given by Eq. 1a, comparing local magnetic field and orientation (blue) relative to the location of the two nulls (red) connected by a separator (black), over two different ranges in x, y, using FLINE.

would overlay the 75th particle track (in purple), a blue orb at locations where the parallel velocity of the particle ($v_{\parallel}$) changes sign, a green orb at the initial location of the particle, and a red orb at the position of the particle when the simulation ends.

We have also created a similar routine to just display the initial positions of every particle, using orbs, called `iniorb`. It is called using exactly the same syntax as `particletrack`.

By repeatedly looping calls to the `particletrack` and `iniorb` routines, one can build up a picture of global trends in particle behaviour. This is demonstrated in Fig. 5, where `particletrack` is used to display the paths and `iniorb` the initial positions of 1280 particles, in order to identify trends in the data.

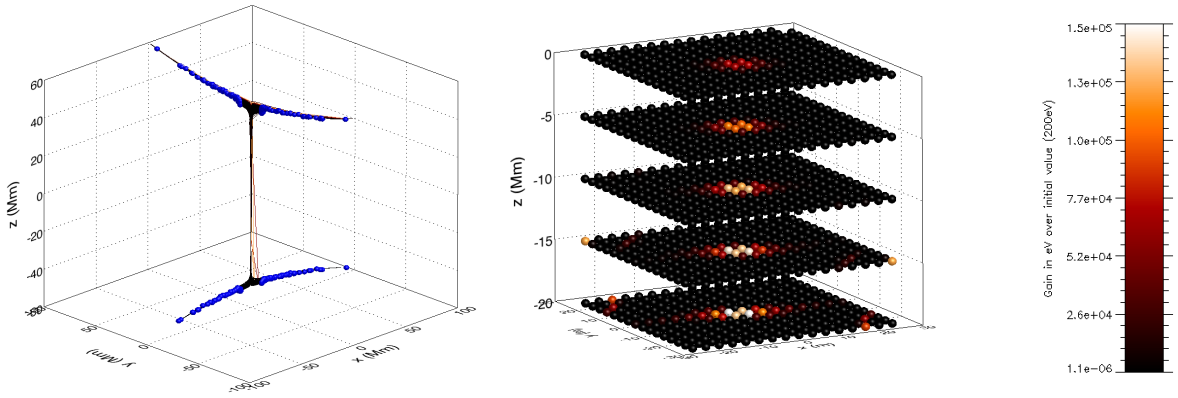
7 Common Questions

How can I create a second copy of my executed code, without also copying the datafiles?

I've found that entering the following on the command line can help with this:
`find . -depth ! -name ''RV*.dat'' | cpio -pdmv ../newfolder`
The command finds all files within a given directory (and subdirectories), and copies them to a new folder whilst omitting those whose name matches `RV*.dat` from the copy. Isn't that nifty?

How do I delete old data that I no longer want?

The Makefile option
`make clean`
will not remove any output data, but will simply reset/delete the executable files. However, typing
`make datatidy`
should remove the datafiles present in the `/Data` subdirectory without doing anything to the code. This is effectively equivalent to



(a) particle tracks

(b) initial positions

Figure 5: Global behaviour example; (a) shows particle trajectories (and mirror points), while (b) shows initial positions, all coloured to identify differences in particle behaviour. This example highlights the peak energy gained by individual particles during the simulation (see colour bar for specific values).

`rm -rf Data/*`
but I tend to panic when the `-f` flag is active on the remove command, so be careful if you do this manually!

What are all the extra files in the folder `IDL/JTfiles`?

This code is a work in progress. I am currently using this code as part of my postdoctoral research. Many of these codes have been written to solve/address specific issues, show specific quantities and how they change during the simulation. Many are simply wrappers for codes like `particletrack`, so that you can see how to repeatedly call those routines in a given setup. I have also toyed with adding various different 3D interactive symbols in one or two codes. How you use these additional codes is up to you, but I have included them as a reference for the time being to show what is possible.