Manual: Field line tracing

Peter Wyper, Sept. 2025

Description:
The field line tracer takes a grid of starting positions and traces field lines out from each position until it either reaches a maximum step number or hits a boundary. The starting positions are defined on a plane aligned with one of the 3 spatial dimensions ((x,y,z) in cartesian, (r,theta,phi) in spherical).

Code facts:
Tracing method: 4^{th} order Runge-Kutta algorithm.
Interpolation: linear within each block.
Uses 64bit block adapted bfield data directly out from the ARMS code.
The step size adjusts adaptively (see below).
Handles Cartesian and Spherical grids.
Can handle periodic or line tied boundaries.
Serial Fortran code.

Outputs:
Always produced:
bmapping – an array with the starting and end positions of each field line traced backwards.
fmapping – an array with the starting and end positions of each field line traced forwards.
bbends – array with the magnetic field data at the start and end positions for the back mapped field lines.
fbends – array with the magnetic field data at the start and end positions for the forward mapped field lines.
qmap – the squashing factor.

User specified:
flen – field line length.
twist – Tw (average twist of neighbouring field lines around the given field line).
fflines – an array containing all the forward traced field lines and their positions in space.
bflines – an array containing all the backward traced field lines and their positions in space.

Running the code:
ARMS:

It is assumed you are familiar with running ARMS to produce the magnetic field data, or at least have been given the files. The files take the form of bfield.******* where the extension is the number of code steps the output was obtained on. The code is written to work on this format as it retains the block adapted grid so avoids resampling to a regular grid which either lowers resolution, or massively increases the file size. In this format the values of the 3 magnetic field components are stored at the cell corners within the adaptive mesh tree structure.

Tracer:

Assuming you have gfortran installed, make the tracer executable by executing:
make –f make_fieldline_gfortran
in the ARMS source code directory. This will produce an executable called: fieldline. Copy this into your working directory where the bfield data files are.

Fieldline uses a control file named: fieldline.cnt with all the instructions for the code (see below). The output will be placed in a new directory called PFLS. Make this directory in your working directory so the code has somewhere to save things.

Here's and example of the contents of fieldline.cnt:

```
.true.                    !spherical?
.false.,.true.,.true.  !save field line: positions? Lengths? Tw?
0, 400, 400               !n1,n2,n3
1.0, -78.75, -180.0       !rlo
1.0, 78.75, 180.0         !rhi
.false.                   !adaptive step during tracing?
3000, 0.01, 0.02, 1e-1    !nsteps,lstart,lmax, error bound
1                         !no. snapshots
0000001
```

Each line in turn:

`.true.  !spherical?`

set to .true. for spherical data and .false. for Cartesian.

`.false.,.true.,.true.  !save field line: positions? Lengths? Tw?`

Sets the additional outputs. Saving the field line positions is very expensive in terms of memory. I advise to only use when you have a small set of starting positions. Typically I create the mapping first (with many starting positions) and then use this to identify a small sub-set of field lines to look at more closely.

`0, 400, 400  !n1,n2,n3`

The resolution of the starting grid of field lines. n1 = 0 sets the grid to a plane of constant r (or x in carts). Actually they set the number of divisions of the grid domain in their direction, so n2=10 and n3=12 will give output arrays of 11 by 13 (see below). Setting n2=0 puts the starting grid in a plane of constant theta (y), and n3=0 sets the starting grid on a plane of constant phi (z).

```
1.0, -78.75, -180.0      !rlo
1.0, 78.75, 180.0        !rhi
```
The bounding box for the starting grid. For spherical r is measured in solar radii, and theta in latitude. Theta and phi are also in degrees. This was done to match the coordinate system used in heliospace (the visualization software for ARMS). The output from the code is also in this format. For Cartesian x,y and z are measured in the simulation space (not scaled as they are in heliospace).

In this case n1=0 so we have a plane of constant r. We want this on the solar surface so set rlo(1) = 1.0 and rhi(1) =1.0. This plane can be any other r plane, but should be consistent with the size of the simulation and having n1=0. This simulation goes out to r=7 so we could set r=7 and trace backwards to get the S-web map. -78.75 and +78.75 give the theta extend of the grid. -180 and 180 give the phi extent. Similar logic applies when tracing from planes of constant theta or phi. In the case of constant theta say (n2=0), set theta = const. in rlo and rhi and put in the ranges for the other two etc.

```
.false.                  !adaptive step during tracing?
```
Set this to .false. when outputting Q as the calculation is sensitive to the step size. This will fix the tracing step to be lstart. Note, this will slow down the calculation. If you don't need Q, but want just the mapping or field line length then set to .true.

```
3000, 0.01, 0.02, 1e-1   !nsteps,lstart,lmax, error bound
```

The tracing parameters.
3000 – the number of steps taken along a field line. Can be high if you're not saving the field lines themselves.
0.01 – the initial step size (lstart). Set this to something small so that polarity inversion lines don't have poor tracing. The step size will increase as it enters the volume and doesn't need that level of accuracy.
0.02 – maximum value for the step size.
1e-1 – the error limit allowed on the tracing (ep). The step size of the tracer adapts by checking against 3 criteria:

(1) the size of the block the current position on the field line is in. lstep < half the block width (to avoid jumping blocks and to help adapt in current layers etc).
(2) the maximum step size: lstep < lmax
(3) the difference between the fourth order method and a less accurate 3rd order method from the same starting point. Ep is a measure of the distance between the two new positions that each method finds. I generally don't use this much so tend to set the value relatively high (so it's ignored).

```
1                              !no. snapshots
```
The number of snapshots to trace through.

```
0000001
```
The snapshot number (i.e. bfield.`0000001`)

Outputs

In PFLS you will get a series of data files (indices in Fortran numbering, i.e. counting from 1):

ffieldline. ******* - field lines if requested of the form fflines(n2+1,n3+1,nstep,3).
bfieldline. ******* - field lines if requested of the form bflines(n2+1,n3+1,nstep,3).

bmapping. ******* - the back mapping of the form bmapping(n2+1,n3+1,2,3).
bmapping(n1+1,n2+1,1,:) – start positions
bmapping(n1+1,n2+1,2,:) – end positions

fmapping. ******* - the forward mapping of the form fmapping(n2+1,n3+1,2,3).
fmapping(n1+1,n2+1,1,:) – start positions
fmapping(n1+1,n2+1,2,:) – end positions

Note that the start positions are the same so:
fmapping(n1+1,n2+1,1,:) = bmapping(n1+1,n2+1,1,:)

bbends. ******* - the end point bfield data of the form bbends(n2+1,n3+1,2,3).
bbends(n1+1,n2+1,1,:) – start positions
bbends(n1+1,n2+1,2,:) – end positions

fbends. ******* - the end point bfield data of the form fbends(n2+1,n3+1,2,3).
fbends(n1+1,n2+1,1,:) – start positions
fbends(n1+1,n2+1,2,:) – end positions

qmap. ******* - the Q map of the form qmap(n2+1,n3+1). Calculated using the method of e.g. Tassev et al. 2017 with the updated formula from Aslanyan et al. 2024.

All arrays are double precision and saved as unformatted Fortran files. These can be read with your own plotting routines, or you can use and adapt the examples in plotting folder.