UFiT - Universal Fieldline Tracer

Summary

This package is intended to trace magnetic field lines and calculate the squashing factor Q. It is intended to read the outputs from a number of astrophysical codes, currently: `ARMS`, `Lare3d` and `DUMFRIC`; it saves the result in a self-contained manner.

Quick start

Example 1: (command line) Analytic B

Obtain all the source code and execute the following command in the terminal:

```
make
```

Ensure that you have `Python` installed, as well as the `numpy` and `matplotlib` libraries. Run the following script:

```
python Prepare_Spherical_Example.py
```

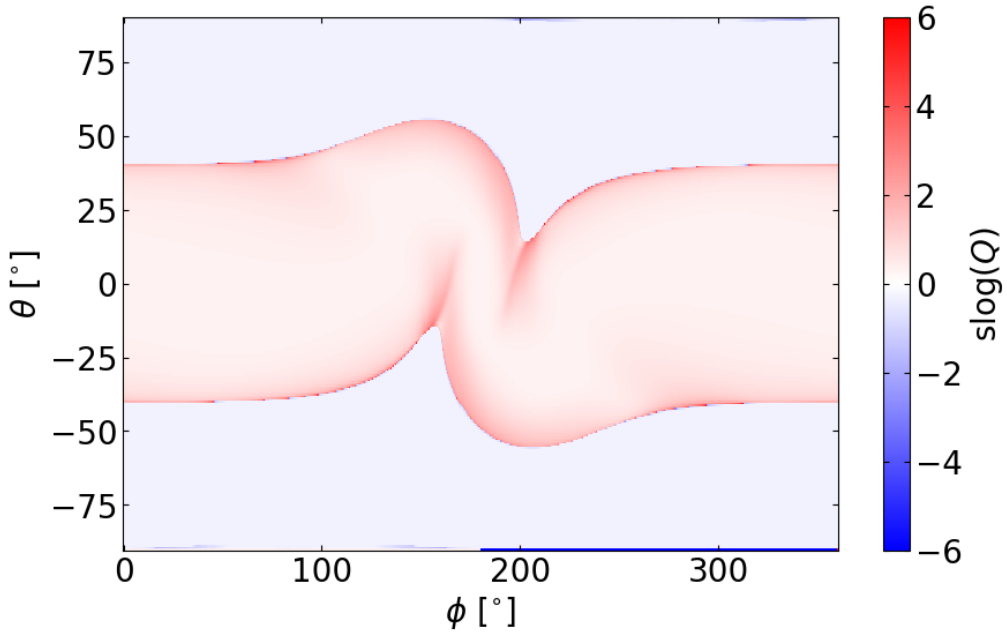
This will create a sample B-field on a grid from an analytical formula. It will also create an input file, containing the start points of all the fieldlines. Now, run `UFiT` itself by executing the following command:

```
./Run_UFiT -g 1 -pp -nb -sq -b Example.bin -i Example.inp -o Example.flf
```

This might take a minute or so. If all goes well, run the following script:

```
python Visualize_Spherical_Example.py
```

You should see the following map of slog(Q), the signed logarithm of the squashing factor (positive/red indicates closed and negative/blue indicates open field lines).



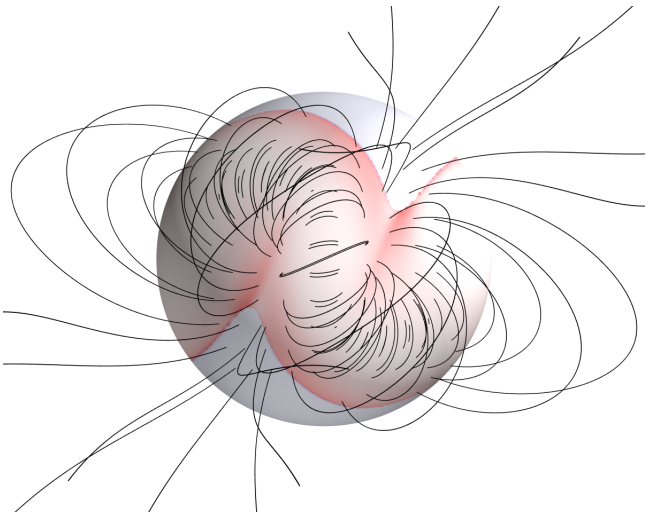
(Optional) If you have the `Python` library `mayavi` installed, to can then execute the following command:

```
./Run_UFiT -g 1 -pp -nb -sf -b Example.bin -i Example2.inp -o Example2.flf
```

This will use the same B-field as before, but calculate the fieldlines from a smaller set of start points; however, it will save the entire curve for each fieldline. Run the following script:

```
python Visualize_Spherical_Example3D.py
```

You should see the following 3D rendering in `mayavi` of $\mathrm{slog}(Q)$ on a surface and a selection of black fieldlines coming up from it.



Example 2: (`Python` call) Synoptic Magnetogram

This example requires a few additional modules. Besides the usual, you will need to have the `Python` modules `wget`, `astropy`, `pfsspy` and `sunpy` installed. You can do this by setting up a virtual environment *etc*, or executing the following (you may need to have administrator rights or add the suffix `--user`):

```
pip install astropy
```

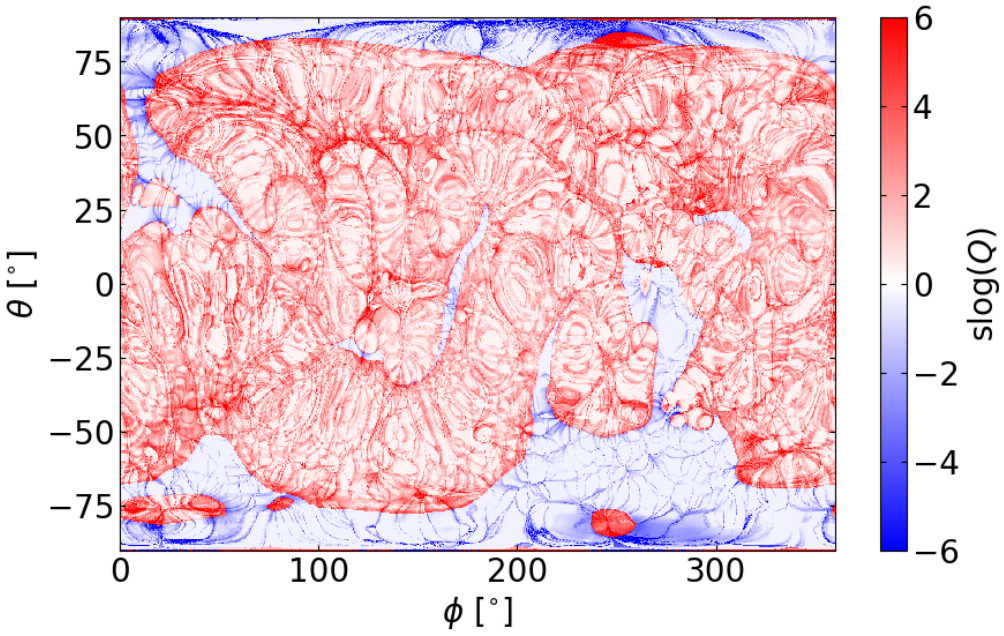
Open the script `HMI_Example.py` for editing. Near the top, you should see the following line:

```
UFiT_home_directory='/change/this/path'
```

This should be set to a path to where `UFiT` is installed. Now run the script by executing:

```
python HMI_Example.py
```

A synoptic magnetogram will be downloaded and a Potential Field Source Surface (PFSS) model be used to calculate B. This time, `Python` will call a compiled library directly, without the go-between of the input files and so on. You should see the following figure:



The squashing factor is unsurprisingly busy, because the realistic data has not been filtered. Note that the original magnetogram has been downscaled for the purposes of the PFSS extrapolation and NaNs in the original data (in the poles where data is not available) replaced with zeros.

Installation

Once the source code has been obtained, execute the following command to build the code:

```
make
```

This will follow the instructions in the `Makefile` to compile `UFiT`. If you wish to read `DUMFRIC` files directly, you will need to build `UFiT` with `netCDF` (instead, you can also use a `Python` script to convert them to a binary format first). Execute the following after substituting for the correct paths to the `netCDF` headers and library:

```
make USE_NC=True NC_INCLUDE=/include/path NC_LIB=/lib/path
```

You can omit the arguments `NC_INCLUDE` and `NC_LIB` to use the defaults, or set them within the makefile directly.

Running `UFiT` from the command line

Once this is done, you can run the executable as follows:

```
./Run_UFiT
```

You will need to add some command-line arguments from the tables below:

Argument	Effect
`-g`	**g**eometry: 0 = Cartesian; 1 = Spherical; 2 = Cylindrical
`-r`	print available **r**esources (CPUs for now)
`-np`	**n**umber of **p**rocessors (CPU)
`-cs`	**c**heck **s**tart points
`-px`	**p**eriodic **X**
`-py`	**p**eriodic **Y**
`-pz`	**p**eriodic **Z**
`-pp`	**p**eriodic ϕ
`-se`	**s**ave fieldline **e**ndpoints
`-sf`	**s**ave **f**ieldlines
`-sq`	**s**ave squashing factor **Q**
`-sc`	**s**ave **c**onnectivity type
`-ud`	**u**ser **d**efined calculation
`-nb`	normalized B for Q calculation; **this is highly recommended**
`-dl`	step size
`-ms`	**m**ax **s**teps
`-i`	full path to **i**nput file (defining fieldline starts), *e.g.* `./dir/ufit.inp`
`-c`	full path to **c**ommand file *e.g.* `./dir/ufit.dat`
`-bt`	**B** file **t**ype (see below)
`-b`	full path to **B** file *e.g.* `./dir/ufit.bin`
`-o`	full path to **o**utput file, *e.g.* `./dir/out.flf`

Add any combination of these arguments, as long as you include the correct option afterwards (*e.g.* remembering to put `-o` followed by a space and then the path to the output file). The order of the arguements doesn't matter. For example, the command

```
./Run_UFiT -g 1 -pp -nb -sq -b B01.bin -o Q01.flf
```

would calculate Q in the spherical geometry (periodic longitude ϕ), reading the magnetic field from `B01.bin` and writing the result to `Q01.flf`.

(Optional) Running from a command file

When running from command line, the option `-c` can be used to specify a command file, from which the arguments can be read instead of through the command line. If no command line options are supplied, `UFiT` will automatically look for the default command file `ufit.dat` in the current working directory. The command file overwrites any command line arguments, if

there is a conflict.

Each line in the command file specifies one argument, beginning with one of the following command codes, a semicolon and a space.

Code	Effect
`G:`	**g**eometry: 0 = Cartesian; 1 = Spherical; 2 = Cylindrical
`R:`	print available **r**esources (CPUs for now)
`NP:`	**n**umber of **p**rocessors (CPU)
`CS:`	**c**heck **s**tart points
`PX:`	**p**eriodic **X**
`PY:`	**p**eriodic **Y**
`PZ:`	**p**eriodic **Z**
`PP:`	**p**eriodic ϕ
`SE:`	**s**ave fieldline **e**ndpoints
`SF:`	**s**ave **f**ieldlines
`SQ:`	**s**ave squashing factor **Q**
`SC:`	**s**ave **c**onnectivity type
`UD:`	**u**ser **d**efined calculation
`NB:`	normalized B for Q calculation; **this is highly recommended**
`DL:`	step size
`MS:`	**m**ax **s**teps
`I:`	full path to **i**nput file (defining fieldline starts), *e.g.* `./dir/ufit.inp`
`BT:`	**B** file **t**ype (see below)
`B:`	full path to **B** file *e.g.* `./dir/ufit.bin`
`O:`	full path to **o**utput file, *e.g.* `./dir/out.flf`

Setting the fieldline start points

You will need to create an input file specifying the fieldline starts (sometimes called seeds). By default, it is called `ufit.inp` and contained in the current working directory; for any other name or location, specfity a valid path to it using the input file argument, *e.g.* `-i ./somewhere/starts.something`

There are several input types which allow the fieldline starts to be specified in a convenient way.

Input type	Meaning
`0`	The coordinate of each start point is explicitly specified
`1`	Regular grid of start points where coordinates in each dimension are explicitly defined
`2`	Regular and equally spaced grid of start points in all dimensions; only a start and end point is specified for each dimension

The first line of the input file must read

```
Input type:  X
```

where `X` must be one of the integers above. Thereafter, the structure depends on this type.

For `Input type: 0` you must then put a single integer for the number of start points on the next line. On each subsequent line, you must put three comma-separated coordinates (X, Y, Z or r, θ, ϕ respectively). For example, in the following file three very different starts are specified:

```
Input type:  0
3
0.3,0.0,0.1
0.7,0.2,0.3
0.5,-0.1,-5.0
```

For `Input type: 1` you must put three integers (each one on a new line) for the number of points in each dimension (X, Y, Z or r, θ, ϕ respectively). On each of the following lines, you must put comma-separated coordinates (as many as you have previously specified). For example, in the following file there are three points in X (if Cartesian), four in Y and one in Z; a total of twelve field lines will be traced.

```
Input type:  1
3
4
1
-1.0,0.0,1.0
-0.5,0.5,1.5,2.5
10.0
```

For `Input type: 2` you must put three integers (each one on a new line) for the number of points in each dimension (X, Y, Z or r, θ, ϕ respectively). On each of the following lines, you must put comma-separated

For example, in the following file there are 15 points in X (if Cartesian), 10 in Y and one in Z for a total of 150 field lines. The fieldline starts will be spaced equally in X from -1 to 1 and so on.

```
Input type:  2
15
10
1
-1.0,1.0
0.0,1.0
0.1,0.1
```

In all cases, any more lines than are strictly necessary will be ignored. After making a valid input file as above, you may make several linebreaks and write some comments, store a previous configuration and so on. Only the first valid block will be used.

Supported B files

`UFiT` is intended to read the magnetic field from common MHD codes directly. By default, the type of file will be determined inherently from its extension (or start in the case of a `flicks` file from `ARMS`). Otherwise, you can use the command line argument `-bt` to specify which code created it, if you have changed the extension for example. The following table shows the currently supported file types and the code which goes after the argument `-bt` to manually specify them:

`-bt`	**Default filename**	**Associated code**
`-1`		Detect automatically
`0`	`.bin`	`UFiT` specific
`10`	`.nc`	`DUMFRIC`
`20`	`.sdf`	`Lare3d`
`30`	`flicks.`	`ARMS`

If you have the magnetic field from some other source, you can always save it in the default `UFiT` format using the `Python` function `write_B_file()` (see below), or create a function in

another language with the `Python` function as a reference.

Using `Python`

`Python` for plotting and analysis

`UFiT` is distributed along with a set of `Python` scripts. All the relevant functions and required libraries are contained in `UFiT_Functions_Python.py`. You will find example scripts for plotting and reading the `UFiT` outputs. Near the top of each of these is the following line

```
UFiT_home_directory='/change/this/path'
```

which must be changed to instead contain a path to where the file `UFiT_Functions_Python.py` is kept. This way you can copy and modify the specific scripts for each task while keeping the main function library in one place (where you can also occasionally modify it).

(Optional) Calling routines directly from `Python`

You may create a `Python` script which calls compiled routines in `UFiT_Python_Callable.so` directly, without the go-between of calling from command line and writing in/outputs to disk. Example 2 above features just such a script.

User defined calculations

You are, of course, free to change the source code as you see fit. The easiest way to run your own bespoke calculation might be to use the `user_defined` or `-ud` option. You will then need to modify the file

```
UFiT_User_Functions.so
```

which by default contains three subroutines:

1. `prepare_user_defined` - called once initially, this gives you the opportunity to allocate memory, make preliminary calculations or read in additional files

2. `trace_cartesian_user` - main tracing routine which integrates forwards and back from a fieldline start if Cartesian coordinates are selected

3. `trace_spherical_user` - as above, for sphericals; feel free to leave this alone if you don't intend to use sphericals and vice versa

The default version calculates $|\mathbf{B}|$ at each end of the field line and then saves the expansion factor $|\mathbf{B}_+|^2/|\mathbf{B}_-|^2$ as the user quantity. Any end quantity can be easily calculated by changing this directly. A quantity, such as a sum or a maximum, along field lines must be calculated by adding lines inside the two `do while` loops. Define and allocate additional variables as required.