Summary of carpolTI

Will Huie

1 Introduction

carpolTI, short for Cartesian-to-Polar Transformer and Interpolator (tentatively), is a set of methods written in C++ with the original intention of translating data from a uniform Cartesian grid to a polar one. While this document will provide a quick run-down of what carpolTI was originally written to do, it should be noted that these methods were written with flexibility in mind, and should be easily adaptable to arbitrary cases.

2 Program Overview and Initial Setup

This section goes over the general steps of the process and the particular arrangement of files that carpolTI requires.

2.1 The General Method

Broadly speaking, the overall procedure of carpolTI can be given by the following.

Algorithm 1: carpolTI

Trim headers and footers off the input data;
Generate set Φ of (r, ϕ, t) polar grid points;
foreach $(r, \phi, t) \in \Phi$ **do**
    Generate maps $E : (x, y, t) \to (E_x, E_y),\ B : (x, y, t) \to B_z$ of electric and magnetic field values, respectively;
    Using E and B, interpolate values at (r, ϕ, t);
    Transform $\langle E_x, E_y \rangle \to \langle E_r, E_\phi \rangle$;
    Using B, calculate $\partial_r B_z,\ \partial_\phi B_z$;
    Write results to output file;

2.2 Preparation

Before beginning, however, carpolTI requires a specific file structure. To start, run in a Bash terminal to generate the necessary directories:

```
cd /path/to/carpolTI
make install
```

Then obey the following guidelines:

- The data files should be placed in `data_raw`
- r- and ϕ-values should be listed per-line in separate files in `gridlists`

2.3 Setting Constants and Compiling

After files are in their places, `main()` should be edited to set the following constants:

- `head`, `foot`: starting and ending line numbers (with 1 at the top) of the desired data in each file.
- `rmin`, `rmax`, `phimin`, `phimax`: the desired bounds (inclusive) in the polar grid.
- `xmin`, `xmax`, `ymin`, `ymax`: the desired bounds (inclusive) in the Cartesian grid.
- `hidx`, `midx`, `sidx`, `msidx`: the string indices (0 on the far left) of the hour, minute, second, and millisecond numbers in the filenames' timestamps.
- `xcolnum`, `ycolnum`, `Excolnum`, `Eycolnum`, `Bzcolnum`: the column indices (0 on the far left) of x-, y-, E_x-, E_y-, and B_z-values.
- `outfile`: path to the output file for the run.
- `rlist`, `philist`: paths to the files containing the r- and ϕ-values of the polar grid points.
- `rkey`, `phikey`, `tkey`: paths to the files which list the r-, ϕ-, and t-values used in the run. Note that these values will also be printed in the main output file.

Once these are in order, compile and run with

```
cd /path/to/carpolTI
make
./main
```

3 The Complete Procedure

This section will go into greater detail on the general method and the individual tools it uses. The complete signatures for each method mentioned below can be found below in Section 4.

3.1 Trimming the Data Files

This action is performed by the single method `trim_data()`. For each file in `data_raw`, `trim_data()` discards all lines outside of the specified range and copies the remaining part to `data_trimmed` under the same filename. As most text editors consider the first line of a file to be line 1, so will `trim_data()`.

3.2 Reading Values

After the data has been trimmed, each file is to be looped over and needs to be scanned for a few things. That is, for each of the files in `data_trimmed`, do everything that follows in the rest of Section 3.

1. Get sorted arrays of unique coordinate values in the Cartesian grid; accomplished via `unique_column_vals()`. This returns a sorted array of doubles pointing to all the unique values in the n-th column (with 0 being the leftmost column) of a file in a given range, along with its size for convenience. This should be called twice to read both x- and y-values.

2. Get maps assigning dependent variable values to (x, y) grid points; accomplished via `make_map()`. This returns a once-nested map object assigning values in two columns to those in a third. `make_map()` should be called for as many dependent variables as are desired – in this case, once for each of $E_x,\ E_y,\ B_z$.

Once this is done, `unique_column_vals()` should be called a few more times on the r- and ϕ-value lists in `gridlists`.

3.3 Doing the Math

To aid in the process of eventually writing calculated values to a file along with associated indices, the arrays of r- and ϕ-values are to be looped over for each file in `data_trimmed`. The values of the dependent variables read above are to be interpolated linearly at each (r, ϕ) grid point. In the case of E_x and E_y, the interpolated values are subsequently transformed

to their corresponding values in (r, ϕ) coordinates. Since the partial derivatives of B_z in r and ϕ are additionally required, we also need four other (r, ϕ) points at which B_z must be calculated. This process is as follows for each (r_i, ϕ_i):

1. Use `find_neighbors()` to get the four (x, y) grid points closest to (r_i, ϕ_i).
2. Use `interpolate_space()` to calculate $E_x(r_i, \phi_i)$, $E_y(r_i, \phi_i)$, and $B_z(r_i, \phi_i)$.
3. From the arrays of r- and ϕ-values, get r_{i-1}, r_{i+1}, ϕ_{i-1}, and ϕ_{i+1}.
4. Use `find_neighbors()` and `interpolate_space()` to calculate $B_z(r_{i-1}, \phi_i)$, $B_z(r_{i+1}, \phi_i)$, $B_z(r_i, \phi_{i-1})$, and $B_z(r_i, \phi_{i+1})$.
5. Use these values to calculate $\frac{\partial B_z}{\partial r}$, $\frac{\partial B_z}{\partial \phi}$ using a first-order central finite difference:

$$\frac{\partial B_z}{\partial r} = \frac{B_z(r_{i+1}, \phi_i) - B_z(r_{i-1}, \phi_i)}{r_{i+1} - r_{i-1}} \tag{1}$$

$$\frac{\partial B_z}{\partial \phi} = \frac{B_z(r_i, \phi_{i+1}) - B_z(r_i, \phi_{i-1})}{\phi_{i+1} - \phi_{i-1}} \tag{2}$$

6. Write the r, ϕ, t indices as well as E_ϕ, E_r, B_z, $\frac{\partial B_z}{\partial r}$, and $\frac{\partial B_z}{\partial \phi}$ to the output file.

4 Source Files

This section will list the complete signatures of all written methods and their locations in the source header files, should there ever be a need to adapt the main method. For extended documentation, see their respective implementation files.

4.1 `readfile.hpp`

`readfile` methods are used to parse input files. Here, line numbers start at 1 from the top while column numbers start at 0 from the left.

```
void trim_data(int head, int foot);

std::set<double> unique_column_vals(std::string filename, double minval,
    double maxval, int colnum);

std::map<double, std::map<double, double>> make_map(std::string filename, int
    xcolnum, int ycolnum, int depcolnum);
```

```
std::map<double, std::set<double>> make_grid_map(std::string filename, int
    xcolnum, double xminval, double xmaxval, int ycolnum, double yminval,
    double ymaxval);

double timeof(std::string filename, int hidx, int midx, int sidx, int msidx);

std::set<double> read_original_tvals(int hidx, int midx, int sidx, int msidx);
```

4.2 interpolate.hpp

`interpolate` methods deal with everything that has to do with the interpolation process. While a method for interpolation in time is included, it is not currently used in the main method.

```
std::tuple<double, double, double, double> find_neighbors(std::set<double>
    xvals, std::set<double> yvals, double r, double phi);

double interpolate_space(std::tuple<double, double, double, double> box,
    std::map<double, std::map<double, double>> datamap, double r, double phi);

double interpolate_time(double target_time, double time1, double val1, double
    time2, double val2);
```

4.3 transform.hpp

The only `transform` method is the one used to transform from Cartesian coordinates to polar coordinates.

```
std::tuple<double, double> transform_vector(double xcomp, double ycomp,
    double r, double phi);
```