Code Documentation

Fajans Lab and & Friends

January 9, 2014

Preface

This file is intended to act as a manual for all of the code used for the Lorentz Invariance investigations. It is a work in progress and you should feel free to edit it. When working on it, edit a local copy on your machine (not saved in your Dropbox folder) so that if anyone's work gets overwritten, they still have a copy of it. When you add a section, make sure you add it to the most recent version so it includes all the latest additions.

I'm not sure what the best way to organize this document is, but for now I think it will suffice to make a new chapter for each category of functions (parsing elogs, calculating positions of heavenly bodies, etc.), each of which should have its own folder in the Dropbox. Within each chapter, create a section for each function that describes its inputs, outputs, algorithm, example usage, and compilation instructions if necessary. We should also develop some kind of naming scheme (for example camelCase or names_with_underscores) at least for function names, and ideally for code within functions. This should also be recorded here.

Contents

Chapter 1

Matlab Functions in SimulationData/

The SimulationData/ directory contains a mixture of data files separated into different folders. Because a lot of simulation data was generated[1], only small portions of each form of data are included in the Dropbox. If you need larger portions of the data, contact Zak at uphgreat@gmail.com.

1.1 SimulationOutput and get_sim_data.m

This folder contains the end results of the simulation and this data is used to generate all the simulated data sets. The data is stored in two different ways: .mat and .ellip files. The .ellip files are text files and the .mat files are Matlab binary files that are smaller and can be read/written faster than the .ellip files. Therefore, only use the .ellip files for visual inspection with a text editor.

Each .ellip file in this folder has two columns. The first column is the wait time it took (in seconds) for the $\bar{H}$ to hit the trap walls after the quench. Note that this is *not* the wall time of the event. The second column is the z-position (in meters) of the annihilation with respect to the trap center. This set of data includes effects from detector smearing and cuts[2]. The simplest way to read this data is to use get_sim_data() which returns the data as an array with two columns. The .mat files store that data array as a binary file.

The .ellip files can be read by the get_sim_data() function while the .mat files can be read with the load_mat() function. The functions text_to_mat() and mat_to_text() can convert between these file types[3].

Currently two files are included in this directory: SmallSimData.ellip and MediumSimData.ellip. Each contains a portion of the full set of simulation output, which is ∼400MB so it is not included in the Dropbox folder.

[1] ∼400MB of time and position data was output by the simulations, which in turn is saved in many different ways, thereby increasing disk usage significantly.

[2] Cuts are criteria used to distinguish $\bar{H}$ annihilations from other events, such as those due to cosmic rays.

[3] The mat_to_text() function should not be used in general. Some accuracy may be lost when converting back and forth between the file types and the original data from the simulations is stored in the .ellip files. Therefore, that data should not be overwritten. This function is only included so that the output of other code, which is stored in the .mat format, can be converted to a human-readable format if necessary

1.2 Data Set Files

Each data set consists of text files with three columns and/or .mat files with the same structure. The first column gives the date/time of an event in the datenum() format, the second column gives the wait time (time between the quench and the annihilation), and the third gives the z-position of the event.

The wait times and z-positions used for the data sets are selected in order from the SimulationOutput data. The algorithm for choosing the date/times varies between groups of data sets, and the algorithm for each group is described in Chapter 3. Again, a lot of data was generated so only portions of it are included in the dropbox.

1.3 write/read_data_set.m

These functions write to and read from .set files. This is slower than writing to and reading from .mat files, so the save_mat() and load_mat() functions should be used instead if possible. The read_data_set() function takes the file_name as the input and returns an array with three columns. The first records the time of events in the datenum() format, the second records the wait times of the events, and the third records the z-position of the annihilations. The write_data_set() function takes an output file_name and an array with this structure and records the data into a .set file with these three columns of data.

The data for these files can be generated by generate_simulated_data_sets() which takes wait times and z-positions from SimulationOutput files and calls the function generate_event_times() to choose times. Different data sets will use different algorithms to pick event times, so the generate_event_times() functions are kept in the folder with their output files.

```
>> date_times=[datenum(2011,8,1),datenum(2011,9,2),datenum(2011,10,15,5,22,10)];
>> wait_times=0.01*rand(1,length(times));
>> z_positions=0.01*rand(1,length(times));
>> data_array=transpose( [date_times;wait_times;z_positions] )

data_array =

   1.0e+05 *

   7.347160000000000   0.000000065050764   0.000000087757393
   7.347480000000000   0.000000072662953   0.000000001436214
   7.347912237268519   0.000000009448856   0.000000029430263

>> write_data_set('test',data_array); %could also write 'test.set'
>> clear data_array
>> data_array=read_data_set('test.set')

data_array =

   1.0e+05 *
```

```
   7.347160000000000   0.000000065050764   0.000000087757393
   7.347480000000000   0.000000072662953   0.000000001436214
   7.347912237268519   0.000000009448856   0.000000029430263
```

1.4 load/save_mat.m

These functions provide the same services as read_data_sat() and write_data_set(), except they work with .mat files for improved performance. This is how the data should be read and stored.

```
>> date_times=[datenum(2011,8,1),datenum(2011,9,2),datenum(2011,10,15,5,22,10)];
>> wait_times=0.01*rand(1,length(times));
>> z_positions=0.01*rand(1,length(times));
>> data_array=transpose( [date_times;wait_times;z_positions] )

data_array =

   1.0e+05 *

   7.347160000000000   0.000000017991488   0.000000058109323
   7.347480000000000   0.000000092629427   0.000000063715122
   7.347912237268519   0.000000006818044   0.000000065126926

>> save_mat('test',data_array);
>> clear data_array
>> data_array=load_mat('test.mat')

data_array =

   1.0e+05 *

   7.347160000000000   0.000000017991488   0.000000058109323
   7.347480000000000   0.000000092629427   0.000000063715122
   7.347912237268519   0.000000006818044   0.000000065126926
```

1.5 text_to_mat.m/mat_to_text.m

These functions allow for conversion between different file types. The text_to_mat() function creates a .mat file from either a .set or .ellip file. The mat_to_text() function creates a .set file from a .mat file. As mentioned in a previous footnote, it should never be necessary to convert .mat data back into .ellip data, so mat_to_text() converts only to .set files. Both functions only require the input file name as an argument.

```
>> text_to_mat('MediumSimData.ellip')
Created MediumSimData.mat
>> text_to_mat('data_set_1.set')
Created data_set_1.mat
>> mat_to_text('data_set_1.mat')
Created data_set_1.set
```

1.6 generate_simulated_data_sets.m

This function creates pseudo data sets by pairing wait times and z-positions from the given file_name (which should refer to .mat data from SimulationOutput) with times from the function generate_event_times(). The length of the data set is chosen to be the length of the array returned by generate_event_times(). The function produces data sets until it runs out of wait time/z-position data, so the number of output files is determined by the length of the SimulationOutput file specified. Each output file is saved as LorentzInvariance/DataSetOutput/data_set_k.set where k is an integer. Some informational output is printed while the function runs. This function can be run for different versions of generate_event_times() to create different groups of data sets with times chosen differently.

Because so much simulation data was generated, early versions of this function took several minutes to process all the data. To speed things up, the code now saves/loads data in the .mat format rather than the .set format. It can return to outputting .set data by changing OUTPUT_FILE_EXTENSION to '.set' in the code. Furthermore, the main iteration of this code is parallelized to provide further performance boosts. Because of this, it may not run on copies of Matlab that do not include the Parallel Processing Toolbox.

About a third of the time on each call to generate_simulated_data_sets() is spent setting up and taking down a pool of workers. If multiple calls to generate_simulated_data_sets() are expected, set up a pool of workers before the function call. If a pool already exists, the function will use it and leave it running when it has finished, significantly speeding things up. A pool can be set up by calling matlabpool('open') and closed using matlabpool('close') as shown below.

```
>> generate_simulated_data_sets('../TracerOutput/AllSimData.mat')
N_DATA_POINTS
    12911100

N_EVENTS
   312

Slicing up data for parallelization
Starting matlabpool using the 'SevenLocalWorkers' profile ... connected to 7 workers.
Beginning main iteration
Sending a stop signal to all the workers ... stopped.
Generated this many data sets:
       41381
```

```
Took this long (seconds):
  33.804467000000002

>>
>> matlabpool('open')
Starting matlabpool using the 'SevenLocalWorkers' profile ... connected to 7 workers.
>> generate_simulated_data_sets('../TracerOutput/AllSimData.mat')
N_DATA_POINTS
    12911100

N_EVENTS
   312

Slicing up data for parallelization
Beginning main iteration
Generated this many data sets:
       41381

Took this long (seconds):
  22.328147999999999

>> matlabpool('close')
Sending a stop signal to all the workers ... stopped.
```

1.7 datenum_to_sun_position.m

This function takes times in datenum() format (given in Geneva time with Daylight Savings effects already removed) and returns the corresponding positions of the sun. It returns three arrays: altitude angle (in degrees), azimuth angle (in degrees), and distance (in AU) in that order. These values are calculated by the Mex function sun_position.c.

Because datenum_to_sun_position() uses a Mex Function, it requires some set up before it can be used, which is discussed in Chapter 2. Furthermore, because the Mex function uses Aephem, the library libaephem.so must be loaded. This generates several warnings, but these are inconsequential and so they are suppressed. If you would like to see these warnings, comment out the lines in the function that disable the warnings (these lines are easy to identify). Because the library is loaded only once per Matlab session, these warnings will only appear the first time you call a function that loads it in each Matlab session.

In order to facilitate testing of this function, a script with the name test_sun.m is included. This script will get four different times from generate_event_times(), compile the mex file from sun_position.c, then call datenum_to_sun_position() in order to test it.

```
>> test_sun %Input times are random, so your results will differ
Input times:
```

```
   1.0e+05 *

   7.348347028956812
   7.347951769621109
   7.348136403301930
   7.347713633204648

Altitude Angles (degrees)
 -33.523106831333557
 -48.426156349585526
 -44.265885795249218
 -44.483760867979612

Azimuthal Angles (degrees)
   1.0e+02 *

   2.737977248511227
   3.237035750192755
   2.974887156438333
   3.576023494954170

Distances (AU)
   0.986779021502397
   0.996202575443074
   0.991254353812390
   1.003038367934557

>> times=generate_event_times();
>> times=times(1:4); %shorten the data set for testing
>> disp(times); %datenum() format in Geneva time
   1.0e+05 *

   7.347261849032595
   7.347923670937909
   7.347966352360696
   7.348050297263487

>> [altitude_angles,azimuthal_angles,distances]=datenum_to_sun_position(times);
>> altitude_angles %Degrees

altitude_angles =

 -18.855431900039129
 -48.503379507476261
 -48.327077411463314
```

```
 -46.964272534957651

>> azimuthal_angles %Degrees

azimuthal_angles =

   1.0e+02 *

   0.403413145609679
   3.279057384436758
   3.215299601191740
   3.092656764031824

>> distances %Astronomical Units

distances =

   1.013595152713113
   0.996980530762410
   0.995803278933232
   0.993529922540726
```

1.8 datenum_to_moon_position.m

This function takes times in datenum() format (given in Geneva time with Daylight Savings effects already removed) and returns the corresponding positions of the moon. It returns three arrays: altitude angle (in degrees), azimuth angle (in degrees), and distance (in AU) in that order. These values are calculated by the Mex function moon_position.c.

Because datenum_to_moon_position() uses a Mex Function, it requires some set up before it can be used, which is discussed in Chapter 2. Furthermore, because the Mex function uses Aephem, the library libaephem.so must be loaded. This generates several warnings, but these are inconsequential and so they are suppressed. If you would like to see these warnings, comment out the lines in the function that disable the warnings (these lines are easy to identify). Because the library is loaded only once per Matlab session, these warnings will only appear the first time you call a function that loads it in each Matlab session.

In order to facilitate testing of this function, a script with the name test_moon.m is included. This script will get four different times from generate_event_times(), compile the mex file from moon_position.c, then call datenum_to_moon_position() in order to test it.

```
>> test_moon %Input times are random, so your results will differ
Input times:
   1.0e+05 *
```

```
   7.348301591664797
   7.347718621327302
   7.347453383644135
   7.348091955492030

Altitude Angles (degrees)
 -51.708581096340168
 -36.979561920095826
 -45.444516960693413
   9.860554431379413

Azimuthal Angles (degrees)
   1.0e+02 *

   3.135492160861178
   0.282011332586069
   0.119312144638621
   2.338023156631048

Distances (AU)
   0.002408605180408
   0.002422322724313
   0.002413445443471
   0.002588245543373

>> times=generate_event_times();
>> times=times(1:4); %shorten the data set for testing
>> disp(times); %datenum() format in Geneva time
   1.0e+05 *

   7.347261849032595
   7.347923670937909
   7.347966352360696
   7.348050297263487

>> [altitude_angles,azimuthal_angles,distances]=datenum_to_moon_position(times);
>> altitude_angles %Degrees

altitude_angles =

  -4.195638160911750
  32.117787548734356
  -9.564156988032691
 -34.946157346554727
```

```
>> azimuthal_angles %Degrees

azimuthal_angles =

   1.0e+02 *

   2.436268288734462
   0.912306838829998
   0.555622766047513
   2.732085666009608

>> distances %Astronomical Units

distances =

   0.002582072141966
   0.002682732729728
   0.002554265090355
   0.002424242457071
```

1.9 datenum_to_cmb_velocity.m

This functions takes times in datenum() format (given in Geneva time with Daylight Savings effects already removed) and returns the velocity of the Earth in J2000 cartesian coordinates. It returns three arrays, one for each component of the velocity, all of which are given in m/s. These values are calculated by the Mex function cmb_velocity.c.

Because datenum_to_cmb_velocity() uses a Mex Function, it requires some set up before it can be used, which is discussed in Chapter 2. Furthermore, because the Mex function uses Aephem, the library libaephem.so must be loaded. This generates several warnings, but these are inconsequential and so they are suppressed. If you would like to see these warnings, comment out the lines in the function that disable the warnings (these lines are easy to identify). Because the library is loaded only once per Matlab session, these warnings will only appear the first time you call a function that loads it in each Matlab session.

In order to facilitate testing of this function, a script with the name test_cmb.m is included. This script will get four different times from generate_event_times(), compile the mex file from cmb_velocity.c, then call datenum_to_cmb_velocity() in order to test it. It also calculates the magnitudes of the resulting velocities.

```
>> test_cmb %Input times are random, so your results will differ
converting times
Elapsed time is 0.000292 seconds.
velocity calculation
Elapsed time is 0.001435 seconds.
Input times:
```

```
   1.0e+05 *

   7.348347028956812
   7.347951769621109
   7.348136403301930
   7.347713633204648

Velocity x-components (m/s)
   1.0e+05 *

  -3.864509329877444
  -3.721512930742001
  -3.800308685434046
  -3.603658673235534

Velocity y-components (m/s)
   1.0e+05 *

   0.887812105085747
   1.019058074180573
   0.969425092132093
   1.045073555101387

Velocity z-components (m/s)
   1.0e+04 *

  -3.944129486583106
  -3.374932025980408
  -3.590177311038611
  -3.262225784947812

Speeds (m/s)
   1.0e+05 *

   3.984746449740051
   3.873246637387350
   3.938403844844726
   3.766292041182909

>> times=generate_event_times();
>> times=times(1:4); %shorten the data set for testing
>> disp(times); %datenum() format in Geneva time
   1.0e+05 *

   7.347261849032595
```

```
   7.347923670937909
   7.347966352360696
   7.348050297263487

>> [v_x,v_y,v_z]=datenum_to_cmb_velocity(times);
converting times
Elapsed time is 0.000996 seconds.
velocity calculation
Elapsed time is 0.000386 seconds.
>> disp(v_x)
   1.0e+05 *

  -3.395412720635101
  -3.708260010759425
  -3.728301051852960
  -3.765806260905132

>> disp(v_y)
   1.0e+05 *

   0.974816115275701
   1.024361870822486
   1.016070340793857
   0.995627080751760

>> disp(v_z)
   1.0e+04 *

  -3.566948235152507
  -3.351925667711104
  -3.387909007399682
  -3.476737793988580
```

1.10 analyze_data_sets.m

This function is still a work in progress. Right now it takes a list of .mat files and iterates over them, analyzing each. It runs CharmanII and CharmanIV, each with period of one sidereal day then a period of one year. It does this on both the wait time and z-position data. Furthmore, it does all of this for a list of parameters of interest. Right now it just looks as CMB speed, but in the future it could look at moon/sun position parameters or even the boost γ.

```
>> for j=1:40915
    file_name_list{j}=['../DataSets/FourMonthDataSets/DataSets/data_set_', ...
    num2str(j),'.mat'];
```

```
end
>> tic;analyze_data_sets(file_name_list);toc;
Beginning iteration
Elapsed time is 52.718309 seconds.
```

Chapter 2

Mex Functions in SimulationData/

This chapter covers the Mex functions in Simulation data that are used to interface with the Aephem library. It documents both how to compile the functions and how they are used by their corresponding Matlab wrappers.

2.1 Set Up

After some code edits, the set up process has been simplified. If your machine is able to compile Mex files, all that needs to be done is to compile Aephem and put it in the correct directory. The code should be stored with the same directory structure as that of the Dropbox folder. After downloading the tarball for Aephem (I use version 2.0.0-Canopus, which is the latest version as of this writing), the tarball should be extracted into the aephem-2.0.0 directory. The directory name does not have any colons or spaces in it because this seems to cause trouble for the Mex compiler. Run 'make' to build the code. If you've already built the code, you can just move the directory containing the code to the appropriate place (in the same directory as SimulationData/) and rename it to "aephem-2.0.0". Or if you prefer not to have two copies of the software floating around, you might be able to get away with creating symbolic links.

If you have not used Mex on your machine before, it may need to be set up. I ran 'mex -setup' before I tried compiling anything, but this may not have been necessary. If you cannot build, refer to Matlab's documention for further instructions on getting Mex working.

2.2 moon/sun_position.c

These functions are intended to be compiled with Mex. After being successfully compiled, they take an array of Unix times and return three arrays giving the positions of the moon or sun: altitude angle (in degrees), azimuth angle (in degrees), and distance (in AU) in that order.

Because the input times are in Unix time, and because the libaephem library must be loaded before this function can run, wrapper functions datenum_to_moon_position() and datenum_to_sun_position are included in the same directory. The wrapper functions load the libaephem library (if not already loaded), convert times from the datenum() format in

Geneva time to Unix time, then call moon_position() or sun_position() and return the results. It is recommended that you use datenum_to_moon_position() and datenum_to_sun_position() instead of using moon_position() and sun_position directly.

Before moon/sun_position() or datenum_to_moon/sun_position() can be called, moon_position.c and sun_position.c must be compiled. The commands to compile them are included in the comments at the top of each function's source code and the commands are included below for reference. The commands can be run in Matlab and may also work in Bash. The scripts test_moon and test_sun will run these commands, so if you run those scripts you will not need run the Mex commands.

```
>> mex -O CFLAGS="\$CFLAGS -std=c99" ...
-I../../aephem-2.0.0/src/ moon_position.c ...
../../aephem-2.0.0/src/.libs/libaephem.so

>> mex -O CFLAGS="\$CFLAGS -std=c99" ...
-I../../aephem-2.0.0/src/ sun_position.c ...
../../aephem-2.0.0/src/.libs/libaephem.so
```

2.3 cmb_velocity.c

This function is intended to be compiled with Mex. After being successfully built, it takes an array of Unix times and returns three arrays, one for each velocity component. The velocities are given in m/s in J2000 cartesian coordinates.

Because the input times are in Unix time, and because the libaephem library must be loaded before this function can run, a wrapper function datenum_to_cmb_velocity() is included in the same directory. The wrapper function loads the libaephem library (if not already loaded), converts time from the datenum() format in Geneva time (it assumes Daylight Savings time effects have already been subtracted out) to Unix time, then calls cmb_velocity(), and finally return the results. It is recommended that you use datenum_to_cmb_velocity() instead of using cmb_velcoty() directly.

Before cmb_velocity() or datenum_to_cmb_velocity() can be called, cmb_velocity.c must be compiled. The command to compile it is included in the comment at the top of its source code and just below here for reference. The commands can be run in Matlab and may also work in Bash. The script test_cmb will run that commands, so if you run the test script you will not need run the Mex commands.

```
>> mex -O CFLAGS="\$CFLAGS -std=c99" ...
-I../../aephem-2.0.0/src/ cmb_velocity.c ...
../../aephem-2.0.0/src/.libs/libaephem.so
```

Chapter 3

Data Directories in SimulationData/

As of the moment that this is being written, we have not agreed upon a way to choose the times of the events for the simulated data sets. Therefore I plan on creating multiple groups of data sets, each with a different version of generate_event_times(). This chapter includes descriptions of these different groups of data sets.

3.1 FourMonthDataSets

This data was generated with random times. The distribution function was chosen to be uniform over the period from 8/1/2011 to 12/1/11. This was chosen to be very roughly the time of the year that data was collected in 2010 and 2011.

Chapter 4

CharmanUltra/

This folder contains the functions used to implement Andy Charman's algorithms. His descriptions/derivations of these algorithms are described in Charman_sinusoid_estimator.pdf, which is included in the folder with this document.

There is an important distinction between a Matlab day and a sidereal day. A Matlab day is defined as $60 * 60 * 24 = 86,400$ seconds while a sidereal day is how long the Earth takes to revolve once with respect to the distant stars, which is 86,164.09 seconds. To make the code more straight forward, all the data is stored in terms of Matlab days (because this is what datenum() uses). However, we are more interested in sidereal days when it comes to periodic signals, so some arguments are given in sidereal days and converted inside the function to Matlab days for calculations.

4.1 CharmanII.m

This function returns the complex Fourier Coefficient A_1, given a data set and a period. The argument data_set should be the name of a .mat file containing a simulated data set. These files are created by generate_simulated_data_sets(). The period should be 'day', 'month', 'year', or a time in units of sidereal days.

This function uses the method described in Section II of Charman_sinusoid_estimator.pdf. It is best suited for period='year' because it is not expected to be as powerful as CharmanIV.m for daily or monthly signals, but does not need the data to be well-distributed across the time period.

```
>> A_1=CharmanII('DataSets/FourMonthDataSets/data_set_1.mat','day')

A_1 =

 -0.000738042316639 + 0.003414656452797i
```

4.2 CharmanIV.m

This function returns the complex Fourier Coefficient A_1, given a data set and a period. The argument data_set should be the name of a .mat file containing a simulated data set. These files are created by generate_simulated_data_sets(). The period should be 'day', 'month', 'year', or a time in units of sidereal days.

This function uses the method described in Section IV of Charman_sinusoid_estimator.pdf. It is best suited for period='day' because it requires the data to be well spread throughout the period and assumes a fixed frequency.

```
>> A_1=CharmanIV('DataSets/FourMonthDataSets/data_set_1.mat','day')

A_1 =

 -0.002061251657489 + 0.003264772424234i
```