MESMER

TRAP + Anti-TRAP Tutorial

Introduction

TRAP and Anti-TRAP are two homooligomeric components of the tryptophan regulatory system in several species of *Bacillus*. Under some conditions, they reversibly aggregate to form long, heterogeneous chains consisting of multiple copies of each oligomer. In this tutorial, you will use several of the MESMER command-line utilities to generate targets, build component files, fit targets, and make several plots from synthetic data obtained from a model of this system.

During the tutorial, you will be typing commands into your command-line terminal. This tutorial assumes a Linux, Unix, or MacOS X based terminal environment, although in some instances the Windows™ command-line will be similar. Examples of commands the user should type will be preceded by a generic prompt character (`$`), and be set in a `monospace typeface`.

You may also need a text editor capable of outputting raw ASCII text. On linux, EMACS or vi will work nicely. Nano/pico is also available on most systems. On MacOS™, the free program TextWrangler is highly recommended. If using Windows™, you may be able to use Notepad, although due to the differences in line endings, lines in the provided tutorial files may appear "scrunched" together.

Step I – Compile component files

We first need to generate "component" files that contain the calculated attributes (SAXS profiles, FRET distances, etc.) of the possible configurations our system can adopt. For the purposes of this tutorial, SAXS profiles, and FRET lifetime curves for 18 different theoretical TRAP+AT structures have already been computed. Files containing these curves are present in the directories "`saxs_data`" and "`fret_data`" respectively.

When guided by the provided "`component_template.txt`" template , the MESMER utility `make_components` will combine these data into properly-formatted component files. Let's take a look:

```
$ cat component_template.txt
      NAME  $0

      SAXS
      %1

      CURV
      %2
```

All components must possess a unique `NAME`. When you run `make_components`, the name for each component will automatically replace the wildcard "`$0`" symbol in the template. The lines immediately following the `SAXS` and `CURV` lines are also a type of wildcard, which will be explained shortly. They essentially tell `make_components` to "insert data here".

Before we continue, we will need to build a file containing the component's names. We could do this by hand, but if we have more than several dozen components that could become extremely tedious. It is much easier to let the computer do the work:

```
$ ls -1 pdbs | sed -e 's/.pdb//' > names.txt
```

This command lists all of the components in the "`pdb`" directory, removes the ".pdb" from their filenames, and writes them to the "`names.txt`" file. Now we are ready to run `make_components`:

```
$ make_components -template component_template.txt -values names.txt -data
      saxs_data fret_data -out components
```

Note: You can issue any MESMER utility with just the `-h` (help) option to see a brief description and the list of possible arguments.

When you issue this command, `make_components` goes through the "`names.txt`" file line by line, since each line represents a different component. It then replaces each example of the `$0` wildcard with the first element in the current line (in this case, the component's name).

`make_components` then searches through the `saxs_data` and `fret_data` directories for files matching that name, and replaces the `%1` and `%2` in the template with the data it reads from matching files.

Finally, each filled in template is written as a separate file to the newly-created "components" directory as "name.component". You can (and should!) open several of these component files in a text editor to make sure they are correct.

Step II – Generate synthetic target data

We will now use the components to generate synthetic data consisting of a known composition. This is very useful for testing and validation purposes.

To make a synthetic target, you will need two files: an existing target, which will serve as a template to determine how the resulting target data should be formatted, and a "specification" file, which tells MESMER how much to weight each component in the result averaged target data.

These files have already been created for you, they are "`target_template.txt`" and "`target_specification.txt`". As with all MESMER input files, they are simply text, so let's take a look:

```
$ cat target_template.txt
      NAME  template

      SAXS  1.0   -file template_saxs.dat

      CURV  1.0   -file template_fret.dat
```

Just like component files, all target files must possess a unique name, which is defined by the "`NAME`" symbol, followed by a space or a tab, and the unique title. The following line tells MESMER to expect a restraint of type "`SAXS`", and assign that restraint a weight of `1.0`.

Note: The restraint weight is different than the component weighting used in the target specification. Restraint weights are used to differentially emphasize one restraint type over another. In this example, both the SAXS data and the CURV data are equally weighted.

The last option, "`-file`", tells the SAXS plugin to look in a separate file for the actual data. When the synthetic data is created, it will be made according to this format (i.e. the X values will stay the same, but the Y values will be calculated from the average of the specified components). Now let's look at the target specification:

```
$ head target_specification.txt
      00000 0.00
      00001 0.50
      00002 0.00
      00003 0.00
      00004 0.00
      00005 0.00
      00006 0.00
      00007 0.00
      00008 0.50
      00009 0.00
      ...
```

This file has the name of each component in the first column, and the final weight each component's data should have in the second column. As you can see, the components 00001 and 00008 are weighted to 50% in the final target. No other components will have their data included in the synthetic target, and in fact could be omitted entirely from the specification file.

```
$ make_synthetic_target -target target_template.txt -spec
      target_specification.txt -components ./components
```

This will cause two files to be generated: `restraints_tutorial_template_SAXS_00000.out`, and `restraints_tutorial_template_CURV_00000.out`. These files contain three columns: the X dimension, the Y dimension of the template target, and the Y dimension of the synthetic data. We will extract the predicted data to make a new target, or alternatively, you can skip this step and use the pre-generated "`synthetic_saxs.dat`" and "`synthetic_fret.dat`" files already present in the tutorial folder.

Use Excel or another spreadsheet program to extract the X column, and the second Y column. Alternately, you can use the program "cut" on many POSIX-compliant operating systems:

```
$ cut -f 1,3 restraints_template_SAXS_00000.out | tail -256 >
      synthetic_saxs.dat

$ cut -f 1,3 restraints_template_CURV_00000.out | tail -256 >
      synthetic_fret.dat
```

Note: Selecting the desired cells from most spreadsheet programs and pasting into a text-editor will automatically convert them to a tab-delimited format.

Note: MESMER does not apply any noise to predicted/synthetic data. If you wish to simulate the effect of instrument noise or other sources of error, you will have to do this manually.

Step III – Set the target options

Now we will build a target containing the data we just generated. Copy the template target we used previously and open it in a text editor:

```
$ cp target_template.txt synthetic_target.txt
```

You will want to edit the copied file (`synthetic_target.txt`) to look like this:

```
NAME  tutorial_target

SAXS  1.0   -offset -scale -fitness Pearson -file synthetic_saxs.dat

CURV  1.0   -offset -scale -fitness Pearson -file synthetic_fret.dat
```

You should be a little familiar with this format by now. What's new are the command line-style options present between the restraint weighting term and the `-file` option. From left to right for each restraint type:

`-offset`	Allows MESMER to vary the baseline in order to make a better fit. Good for if your baseline subtraction may be slightly off (or unknown).
`-scale`	Allows MESMER to scale each ensemble's average curve by a consistent factor in order to generate a better fit to the target data. This is frequently useful for when your data has no set or intrinsic scale.
`-fitness`	This tells the plugin what algorithm should be used to describe the agreement between the target and ensemble's fit data. If left blank or omitted, MESMER will assume that you want a chi-square term calculated. (But for that you'd need an X, Y, and dY column in your experimental data. For now we'll just use a Pearson goodness-of-fit value.

Note: You can run `mesmer` with the `-plugin` option (e.g. `mesmer -plugin SAXS`) in order to get more information about the plugin that handles that type of experimental data.

Step IV – Testing components

Now that we have our components and synthetic target data, let's try a test run to check for errors:

```
$ mesmer -target ./synthetic_target.txt -components ./components -size 1
     -ensembles 10 -Gmax 0 -name tutorial_1
```

This tells MESMER to build 10 ensembles consisting of a single component each, and to exit after only a single generation.

After issuing this command, MESMER will:

1. Print the parameters for the run
2. Load the input component files
3. Create the initial parent ensembles,

4. Optimize the parent ensemble's relative component weights,
5. Perform one round of mutation and crossing to create offspring ensembles,
6. Exit

Note: By default, MESMER will save all a log of its progress and other output files in the directory specified using the "`-name <dirname>`" option. If no name is specified, the results will be saved in a directory "`MESMER_Results`". MESMER will not overwrite the directory from a previous run unless the -force (force overwrite) command is provided.

Note: To see a list of all available MESMER options along with a brief description of each, invoke MESMER with only the "`-h`" (help) option.

Step V – Generating ensemble solutions

For an actually meaningful MESMER run, typically the iterative process of offspring generation and optimization should be continued until all ensembles fit the experimental data equally well or nearly so. This can be done by specifying by the "`-Smin 0.01`" (Standard relative deviation minimum) option. This will cause the genetic algorithm to exit only when the relative standard error (RSD) of the ensemble population is at or below the specified value (in this case, 1%)

To run MESMER until all ensembles have a total fitness score (in this example the chi-square fit to the target SAXS profile) standard deviation of 1% of the average fitness score (or less), invoke MESMER as follows:

```
$ mesmer -target ./synthetic_target.txt -components ./components -size 3
    -ensembles 1000 -Smin 0.01 -name tutorial_2 -Pextra
```

This tells MESMER to use a pool of 1000 ensembles, each with three components each. The "`-Pextra`" option enables extended output, mostly of best fit results at each generation. You may wish to go get a coffee, depending on the processing power of the machine MESMER is running on.

If you have a computer with multiple cores or processors, you will likely see a significant decrease in computation time by using the "`-threads <#>`" command to specify the number of parallel processing threads you wish to use.

You can also change the ensemble population size by using the "`-ensembles <#>`" command (the default is 1000 ensembles, which is likely excessive when using only 18 different components.)

Step VI – Interpreting output

If you open the last "`components_statistics_NNNNN.tbl`" file present in the output folder "`tutorial_2`" created by the above MESMER run, you should see some statistics about the components present in the ensemble population at MESMER's final generation. In my case, it took 9 generations to converge. Let's take a look:

```
$ cat component_statistics_00009.tbl
            Prevalence  Average     Stdev
      00001 100.000 %   5.001e-01   1.506e-03
      00008 100.000 %   4.975e-01   3.411e-03
      00000 16.400 %    3.702e-05   3.570e-05
      00012 13.600 %    5.170e-03   7.476e-03
      00007 12.100 %    5.880e-03   5.952e-03
      00004 9.600 %     2.185e-03   1.780e-03
      ...
```

Notice something interesting? The "Prevalence" value is the percentage that the component appears in the total ensemble population (100% = is present in all ensembles, 0% = present in no ensembles). This is important because it is often the case that multiple components can provide equal or similar fitness to their respective ensembles (they may be indistinguishable from each other, for example).

In this case, MESMER found that to generate reasonable fits, both the components 00001 and 00008 had to be present in the ensemble. Furthermore, there is little consensus as to what the third component need be. This shouldn't be a surprise, because 00001 and 00008 are the only two components we averaged together to make the target data in the first place!

Furthermore, the "Average" value is the average weight of that component in the ensembles that contain it. (A larger average indicates that it is more heavily weighted in the ensemble averaged data used to fit the experimental data).

An inspection of the average weighting reveals that both the 00001 and 00008 components are weighted approximately 50% in their ensembles in order to make reasonable fits. Again, this isn't surprising, given the fact that was the weighting we used to generate the synthetic data to being with. All of the other components are barely weighted at all.

Step VII – Making plots

If you remembered the "`-pExtra`" option when you ran MESMER, you can now change into the output directory and check the ensemble fits to the synthetic target data (where n=the last generation):

```
$ make_saxs_plot restraints_template_SAXS_n.out

$ make_curv_plot restraints_template_CURV_n.out
```

These fits are for the best-fitting ensemble current in the generation. You may save the fits from the window that appears, or use the "`-figure <file.png/pdf/gif>`" option to write directly to an image file. For more complex situations, it may take many dozens or even hundreds of generations before reasonable fits are achieved. Here, a good fit is often generated on the very first generation.

Fit plots are only the tip of the iceberg for MESMER's abilities. Say you had some parameter or "attribute" you were interested in for each component structure. It may be the distance between two domains, or the complex's radius of gyration.

make_attribute_plot allows users to make plots using this information that may reveal attributes shared by the components present in ensembles selected by MESMER. Take a look at the `component_attributes.tbl` file:

```
$ head component_attributes.tbl
      00000 1     57.642086   33.464308
      00001 2     57.800968   35.440955
      00002 3     57.968989   37.579125
      00003 4     58.010933   39.775111
      00004 5     57.957091   41.405546
      00005 4     57.967573   40.131645
      00006 4     57.912172   40.123374
      00007 3     57.919176   38.311396
      00008 4     57.930213   40.321452
      00009 4     57.874812   40.343293
      ...
```

The first column, typically, contains the component name. In this table, the second column contains the number of Anti-TRAPs (AT) in the complex. The third and fourth columns contain the average inter-AT-TRAP distance in the and the complex's radius of gyration, respectively.

`make_attribute_plot` can plot various values against one another. For example:

```
$ make_attribute_plot ../component_attributes.tbl -xCol 2 -yCol 3 -spec
      ../target_specification.txt
```

Because `make_attribute_plot` starts counting columns starting at zero (column zero is the component name), this will plot the average AT-TRAP distance (`-xCol 2`) against the radius of gyration (`-yCol 3`). The "`-spec`" option wsets the coloration of each component's point proportional to the weighting we used to make the synthetic target. Let's now overlay our results from MESMER (again, where n equals the final generation number):

```
$ make_attribute_plot ../component_attributes.tbl -xCol 2 -yCol 3 -spec
      ../target_specification.txt -stats  component_statistics_template_n.tbl
```

This will generate blue circles with radii proportional to their average weighting by reading the statistics from our previous MESMER run. There are many visualization options available in make_attribute_plot. To see these options, use the `-h` (help) option.

Step VIII – Extracting models

In this simplistic example, we already know which components are significantly weighted, and can easily find their pdbs. What if we wanted to get all component pdbs that were weighted at least 10% in all ensembles?

The make_models utility can parse MESMER output files and will copy the pdbs matching a desired set of statistics. For example:

```
$ make_models ../pdbs -stats component_statistics_template_n.tbl -Wmin 0.1
      -out models.pdb
```

This will identify components weighted at least 10% (`-Wmin 0.1`) in the provided statistics file (`-stats <file>`), and collect them from the provided directory containing component coordinate files (assuming they are named correctly!). The resulting structures are then saved as multiple models in a single file.

`make_models` can also generate coloration scripts for pyMol and attribute tables for UCSF Chimera that will allow users to color-code the structures according to their weights or prevalence in the ensemble pool. These are specified using the `-wPyMol` or `-wAttr` options, respectively.

Conclusions

There are several additional scripts and many additional options for the MESMER utilities that are not described in this tutorial. For more information, refer to either the MESMER user manual, additional tutorials, or use the `-h` (help) option present in all MESMER programs.

If you find errors in this tutorial or have helpful suggestions on how to improve it, please report them to the author at elihuihms@elihuihms.com.