Microbial Counts Trajectories Infinite Mixture Model Engine (MC-TIMME) Software Tutorial

Version 1.00

This is the first release. Please refer to the main manuscript and SI Methods for details on the algorithm.

Installation

You will need Matlab^{TM} and the Matlab^{TM} Statistics, Optimization and Bioinformatics Toolboxes, available commercially from The Mathworks (http://www.mathworks.com). We have tested the source code only with Matlab^{TM} R2011b, although it should work fine with R2010 or above. You will also need to install the free Lightspeed Matlab^{TM} Toolbox by Tom Minka, available at: http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/

Unzip the file `MCTIMME_source.zip`, which contains the MC-TIMME Matlab source code, to a directory of your choice. Make sure this directory is on your Matlab path.

Unzip the file `data.zip`, which contains the Dethlefsen *et al*. data [1] formatted for MC-TIMME, in another directory of your choice, `$dataDir$`.

You can also unzip the file `MCMC_runs.zip`, which contains example MCMC runs (in case you don't want to wait while the sampler runs), to yet another directory of your choice, `$exampleRuns$`.

Data file formats

The data directory contains 10 files for the Dethlefsen *et al*. data, which has 3 subjects and *n* = 2582 refOTUs.

For each subject, there are 3 tab-delimited files:

1. `subjX_data.txt` – matrix with *n* rows and *m* columns. Each row represents a refOTU and each column represents a time-point; each entry (*i, j*) is the number of counts for refOTU *i* at time-point *j*.

2. `subjX_times.txt` – *m* lines, each line specifies a time-point in days.

3. `subjX_timeIntervals.txt` – each line specifies a time-interval with 3 components: 1) flag `treat` or `no_treat` indicating a treatment or non-treatment interval, 2) starting time-point for the interval (in days), 3) ending time-point for the interval (in days).

All subjects share a common file `refOTU_taxonomy.txt` with *n* lines, in which line *i* specifies the set of taxonomic labels for refOTU *i*. Labels start at the kingdom level and go down to the genus level, with each label separated by a semicolon. This file is only used to calculate significant enrichments for taxonomic labels in consensus signature groups (CSGs), and can be omitted for datasets in which taxonomic classifications of this sort are not available.

Loading and preprocessing data

To load the sample Dethlefsen *et al*. data, type the following at the Matlab prompt:

```
[D_raw,times,timeIntervals,refOTU_taxonomy] = loadDethlefsenData('$dataDir$');
```

This will load all the data files described in the last section. Here, `$dataDir$` is a string specifying the directory containing your data file (e.g., for Windows users, something like `'c:\MCTIMME\data\'`).

Next, you will need to filter the data to remove refOTUs with small numbers of counts. To do this, type:

```
[D_filtered,psi,medianCounts,origOTUIdx,dataIdx,filterDataIdx] = filterData(D_raw);
```

This will return the filtered data, as well as some additional parameters and indices needed later.

Estimating hyperparameters

To estimate model hyperparameters from the data, type:

```
hparams = estimateHyperParameters(D_filtered,psi,times,timeIntervals,dataIdx);
```

This will take a few minutes. When it's done, you'll have a hyperParameters object. To see what's in it, type `hparams.`

This should return the following values (maybe something slightly different if you're using a different version of Matlab):

```
hparams =

  hyperParameters

  Properties:
           omega_alpha1: 1.0000e-004
           omega_alpha2: 1.0000e-004
                  gamma: {3x1 cell}
            m_epsilon_1: -1.1035
        sigma_epsilon_1: 1.9354
            m_epsilon_2: -0.4626
        sigma_epsilon_2: 4.0969
               m_beta_0: 0.5373
           sigma_beta_0: 12.0019
                v_rho_0: 1.4404
               nu_rho_0: 5
             v_rho_mu_1: 4.7294
            nu_rho_mu_1: 5
             v_rho_mu_2: 2.1199
            nu_rho_mu_2: 5
        m_beta_lambda_1: -2.2744
    sigma_beta_lambda_1: 1.6825
         v_rho_lambda_1: 2.8306
        nu_rho_lambda_1: 5
         v_rho_lambda_2: 5.4465
        nu_rho_lambda_2: 5
              lambdaMin: 1
              lambdaMax: 45
          omega_pi_c_mu: 1
      omega_pi_c_lambda: 1
```

Running the MCMC sampler

Now we'll actually run the MCMC sampler on the dataset. This takes a long time (about 12 hours on my workstation), so if you're impatient, you can skip to the next section and use the example runs provided.

First, set up a directory to store the MCMC samples, `$dirSamples$`. Then, set a string variable to specify the base file name for outputting MCMC samples. In Windows Matlab, type:

`baseDir = '$dirSamples$\output';` [If you're using MacOS/Linux, use a forward slash.]

Now run the sampler:

```
MCTIMME(baseDir,D_filtered,psi,hparams,times,timeIntervals);
```

By default, the sampler will perform 10,000 burn-in iterations and then write samples to disk for another 5,000 iterations. You can change these setting through the variables `num_MCMC_iters` and `num_MCMC_burnin` in `MCTIMME.m`.

The function will output some status information every 100 iterations, which will look something like this:

```
iter=100 nc=128 alpha=204.168912 SD1_mu=0.789063 SD1_lm=0.523438 ep1=0.814986
ep2=2.383284 acX=0.276662 acEp1=0.580000 acEp2=0.440000
```

Here's what the outputs mean:

`iter` = MCMC iteration number
`nc` = number of prototype signatures at this iteration
`alpha` = concentration parameter at this iteration
`SD1_mu/SD1_lm` = estimates of $SD1_{\mu}$ and $SD1_{\lambda}$ (pooled across all subjects) at this iteration
`ep1` = estimate of ε_1, the variable parameterizing the negative binomial noise model on the non-treatment intervals
`ep2` = estimate of ε_2, the variable parameterizing the negative binomial noise model on the treatment intervals
`acX` = acceptance rate for the main Metropolis-Hastings/reversible jump moves
`acEp1` = acceptance rate for the Metropolis-Hastings move for sampling ε_1
`acEp2` = acceptance rate for the Metropolis-Hastings move for sampling ε_2

In general, most of these outputs are just useful for debugging purposes and you needn't worry much about them. However, if you're analyzing a new dataset, you may need to tune the acceptance rates. For `acX`, a "good" acceptance rate is ≈ 0.23. If it's much different than this after a few thousand iterations, you can change the parameters `tune_lambda_1` and `tune_lambda_2` in `MCTIMME.m` to tune the acceptance rate (decreasing the tuning parameters should increase the acceptance rate). For `acEp1` and `acEp2`, "good" acceptance rates are ≈ 0.44. These can similarly be tuned with the parameters `tune_epsilon_1` and `tune_epsilon_2`.

Computing Signature Diversity scores

You'll need to set the `baseDir` variable, which specifies the base output filename for the MCMC samples. If you ran the sampler in the last section, you've already done this. If not, you can use the example MCMC runs provided. In that case, in Windows the setting for `baseDir` would be:

`baseDir = '$exampleRuns$\output';` [If you're using MacOS/Linux, use a forward slash.]

Now call the function to compute SD scores:

```
[SD1_mu,SD1_lambda,SD2,SD2I,SD2D] =
signatureDiversityScores(psi,hparams,timeIntervals,baseDir);
```

This will take a few minutes. You'll get cell arrays containing medians and 95% credible intervals for SD scores for each subject. For instance, the $SD1_\mu$ score for subject#1 is stored in `SD1_mu{1}`.

Estimating and plotting individual signatures

To estimate individual signatures (one for each refOTU), type:

```
signatures = individualOTUSignatures(hparams,baseDir,times,timeIntervals);
```

The function will start outputting which subject and refOTU it's processing. It takes a while to run (about 30 minutes on my workstation), as the function goes through all the MCMC samples and estimates individual signatures.

To plot an individual signature, type:

```
plotIndividualOTUSignature(1,3,signatures,hparams,times,timeIntervals,D_filtered,psi,medi
anCounts,origOTUIdx);
```

The first parameter to the function is the subject number, and the second parameter is the refOTU index (e.g., 1 to 218 for subject#1).

Plotting Relaxation Time Distributions

To plot a Relaxation Time Distribution (RTD), type:

```
plotRelaxationTimeDistribution(signatures,hparams,1);
```

The last parameter to the function specifies the post-treatment interval to use (i.e., `1` for the first post-antibiotic treatment interval, `2` for the second, etc.)

Estimating, plotting and analyzing Consensus Signature Groups

We'll first need to run `consensusSignatureGroups.m` to find the consensus groups (sets of refOTUs that consistently share prototype signatures) from the MCMC samples. To do this, type:

```
CSGs = consensusSignatureGroups(hparams,origOTUIdx,baseDir);
```

The function will eventually start outputting which refOTUs indices it's merging. It takes a while to run (about 20 minutes on my workstation).

Once we have consensus groups, we'll need to calculate the consensus signatures. To do this, type:

```
consensusSignatures =
generateConsensusSignatures(CSGs,2,hparams,times,timeIntervals,baseDir);
```

The second parameter to the function specifies which subject's time-line to use for conforming the consensus signatures. I used subject #2, because this subject has the longest time-line (344 days). The function will start outputting which consensus signature it's computing. It takes about 25 minutes to run on my workstation.

Now, you can plot a consensus signature with the following command:

```
plotConsensusSignature(consensusSignatures{7},2,times,timeIntervals);
```

Here, we're passing the function the 7th consensus signature. The second parameter to the function specifies which subject's time-line to use for conforming the consensus signatures.

Finally, you can output some text files with information on the consensus signature groups, using the following command:

```
outputCSGInfo(CSGs,refOTU_taxonomy,origOTUIdx,baseDir);
```

This will output several files to the directory specified in `baseDir`:

- `CSGEnrichOrder.txt`, `CSGEnrichFamily.txt`, `CSGEnrichGenus.txt` = tab-delimited files, with line *j* in each file containing taxonomic labels at the order, family, or genus level that are significantly enriched among refOTUs belonging to CSG *j* (you can change the false discovery rate threshold by setting the variable `FDR` in `outputCSGInfo.m`).

- `CSGMembers.txt` = a tab-delimited file, listing the refOTUs in each CSG.

- `CSGSummary.txt` = a tab-delimited file, with line *j* containing the number of refOTUs in CSG *j* belonging to each subject.

Generating optimal experimental designs

We'll first need to generate information matrices for the experimental design algorithm, by typing:

```
[Bt,R_inv] = experimentalDesignMatrices(1,hparams,baseDir,times,timeIntervals);
```

The first parameter to the function specifies the subject for which we want to generate matrices. The function will start outputting which MCMC samples it's using to compute the matrices. It takes <10 minutes on my workstation to run.

Once it's complete, run the experimental design algorithm with:

```
[bestTimes,HM] = experimentalDesign(1,times,Bt,R_inv,56);
```

The first parameter to the function specifies the subject for which we want to generate an optimal experimental design. The last parameter to the function specifies how many time-points we want for the optimal design–from 1, up to the total experimental run time (326 days for subject#1). In this case, I've specified the same number of time-points as in the original design (56).

The experimental design algorithm will take a while to run (2.5 hours on my workstation). It outputs a status line for every sweep it makes through the time-points, and then the time-point it selects in that sweep (in this case, 56 sweeps). When it's done, it'll return a list of selected time-points, `bestTimes`, and corresponding utility function scores in `HM`.

References

1. Dethlefsen L, Relman D (2010) Incomplete recovery and individualized responses of the human distal gut microbiota to repeated antibiotic perturbation. Proceedings of the National Academy of Sciences 108: 4554–4561.