MemToolbox tutorial

(This tutorial uses MTB X.X, and was last updated on 6/16/2012.)

The MemToolbox is a collection of MATLAB functions that is designed for analyzing data from visual working memory studies. In particular, it contains a variety of functions that make it easier to fit models to continuous report data (as made popular by Wilken & Ma, 2004 and Zhang & Luck, 2008), as well as to plot these model fits and understand where they capture the data and where they fail to capture the data. The toolbox contains a number of models built in—including those made popular by Zhang & Luck (2008), Bays et al. (2009), among others. Through a series of demonstrations, code for which can be found in the MemDemos folder, the following tutorial covers most of the toolbox's core functionality.

Demo 1: Mixture modeling in just two lines of code.

The simplest way to use the MemToolbox is through `MemFit`, a function that houses much of the toolbox's functionality under one roof. A full analysis of your data using `MemFit` is just two steps away. First, specify your data as a vector of errors, one for each trial, in units of degrees on the color wheel.

```
>> errors = [-89, 29, -2, 6, -16, 65, 43, -12, 10, 0, 178, -42, 52, 1, -2];
```

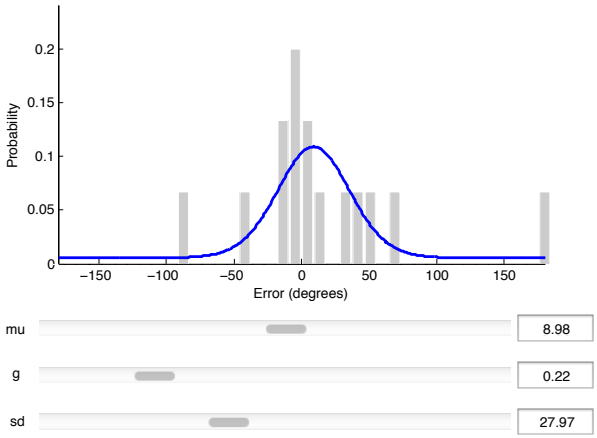
Next, call `MemFit` on the error vector.

```
>> MemFit(errors);
```

The toolbox will then run through its analysis of your data, showing you a histogram of the errors, the name and parameterization of the model it is fitting to the data, and the maximum a posteriori and credible intervals of the model parameter values inferred from the data. Then it will ask whether you'd like to see the fit:

```
Error histogram:   -180 _________.'_.________ +180
          Model:   Standard mixture model with bias
     Parameters:   mu, g, sd

Just a moment while MTB fits a model to your data...

...finished. Now let's view the results:

parameter  MAP estimate  lower CI     upper CI
---------  ------------  --------     --------
mu         7.490         -18.367      29.573
g          0.219         0.037        0.666
sd         27.827        8.963        68.719

Would you like to see the fit? (y/n): y
```

If you respond in the affirmative (by entering the letter `y` and hitting the return key), you'll find an interactive visualization that lets you manipulate the values of the model parameters and see the impact on the predicted distribution of data.



Hooray! In just two lines of code, you've performed mixture modeling using the MemToolbox.

Demo 2: Choosing a different model.

In the previous example, the toolbox picked a model for you. Choosing a different model is easy. The toolbox comes preloaded with a number of them, and you can see a full listing by typing `help MemModels` at the command line. One of the models, `StandardMixtureModel`, is identical to the model used in the previous example, except that the bias parameter μ is constrained to be zero—i.e., the model is unbiased. To use this model, the first step is again to specify your data:

```
>> errors = [-89, 29, -2, 6, -16, 65, 43, -12, 10, 0, 178, -42, 52, 1, -2];
```

Finally, call `MemFit`. This time, give it both the error data and the model that you'd like to use:

```
>> MemFit(errors, StandardMixtureModel);
```

The toolbox will then walk through the same analysis as before, but using the model that you picked.

Demo 3: Specifying your data as a structure with auxiliary variables.

So far, the two models we have used care only about the errors made by the participant on the task. But some models require auxiliary data to make their predictions. For example, the swap model advocated by Bays & Husain (2009), available in the toolbox under the name `MixtureModelWithSwaps`, requires not only the errors made on each trial but also the values of the non-target items. These data should be bundled together into a structure array, like this:

```
>> data.errors = [-89, 29, -2, 6, -16 ...

>> data.distractors = [-10, 2, -100, 163, 42 ...
```

The help file for each model lists its required fields:

```
>> help MixtureModelWithSwaps

   MixtureModelWithSwaps returns a structure for a three-component model
   with guesses and swaps. Based on Bays et al. 2009 model.

   Data struct should include:
     data.errors: errors (radians), e.g., distance of response from target
     data.distractors, Row 1: distance of distractor 1 from target
```

```
...
data.distractors, Row N: distance of distractor N from target
```

Let's test out the model. Rather than type out a big vector, we'll use one of the data sets that comes preloaded with the toolbox. These can be accessed through the function `MemData`, which returns a structure array with fields containing different aspects of the data. For example, the third data set includes a vector of errors and the distractor values for each trial:

```
>> data = MemDataset(3)

data =

         errors: [1x4000 double]
    distractors: [2x4000 double]
```

Now, instead of feeding `MemFit` the error vector, we can give it the entire data structure: and the distractor values for each trial:

```
>> MemFit(data, SwapModel);
```

The toolbox will now run through the usual analysis. You may notice some new things in the output of `MemFit`. In the next demo, we'll explain what it all means.

Demo 4: Digging deeper.

Let's peek under the hood at the functions that underlie `MemFit`. By default, the MemToolbox uses Markov Chain Monte Carlo (MCMC) to do model fitting. MCMC samples parameter values in proportion to how well they describe the data. We can then use the distribution of these parameter values to estimate the true underlying model and to express our confidence in that estimate. In this demo, we'll show you how to read the various plots produced by the MemToolbox. Once you've mastered the basics from demos 1–3, we recommend the following as a standard workflow when using the toolbox.

Start by loading in a dataset, either yours or one from the toolbox:

```
>> data = MemDataset(3);
```

Now run `MemFit` using the `StandardMixtureModel`, but this time assign the output of `MemFit` to a variable and don't suppress printing with a semicolon.

```
>> fit = MemFit(data, StandardMixtureModel)
```

First, we see the usual histogram, model name, and parameterization:

```
Error histogram:   -180 _________.'_.________ +180
          Model:   Standard mixture model
     Parameters:   g, sd
```

There should be no surprises here. In the off chance that you have specified your data in an incorrect format (e.g., by using radians instead of degrees or by coding errors in the range (0,360)), the toolbox will try its best to massage your data into the correct format, always throwing a warning letting you know what it has done. Then the toolbox announces:

```
Just a moment while MTB fits a model to your data...

   Running 3 chains...
```

This refers to the “chains” of Markov Chain Monte Carlo, which you can read about on Wikipedia (http://bit.ly/mcmontecarlo). The default MCMC algorithm used by toolbox starts multiple chains, whose values are specified in the model file (and modifiable), and continues running until they have converged to a comparable range of parameters. Every 2000 steps, the toolbox will update you on its progress:

```
... not yet converged (2000)
```

and then at some point, depending on the model, the chains will converge:

```
... chains converged after 4000 samples!
```

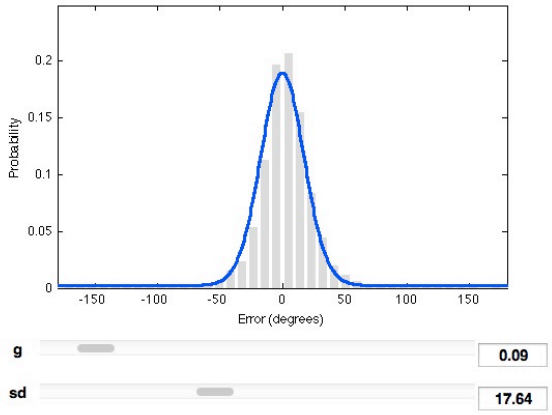
The final estimate is based on samples taken after convergence:

```
... collecting 5000 samples from converged distribution
```

The toolbox then analyzes these samples, returning the MAP (maximum a posteriori) estimate and credible intervals:

```
...finished. Now let's view the results:

parameter     MAP estimate  lower CI      upper CI
---------     ------------  --------      --------
g             0.093         0.080         0.106
sd            17.641        17.091        18.244

Would you like to see the fit? (y/n): y
```

Depending on your interests, this might constitute an answer to your question. In any case, the next step is to visualize the model fit:

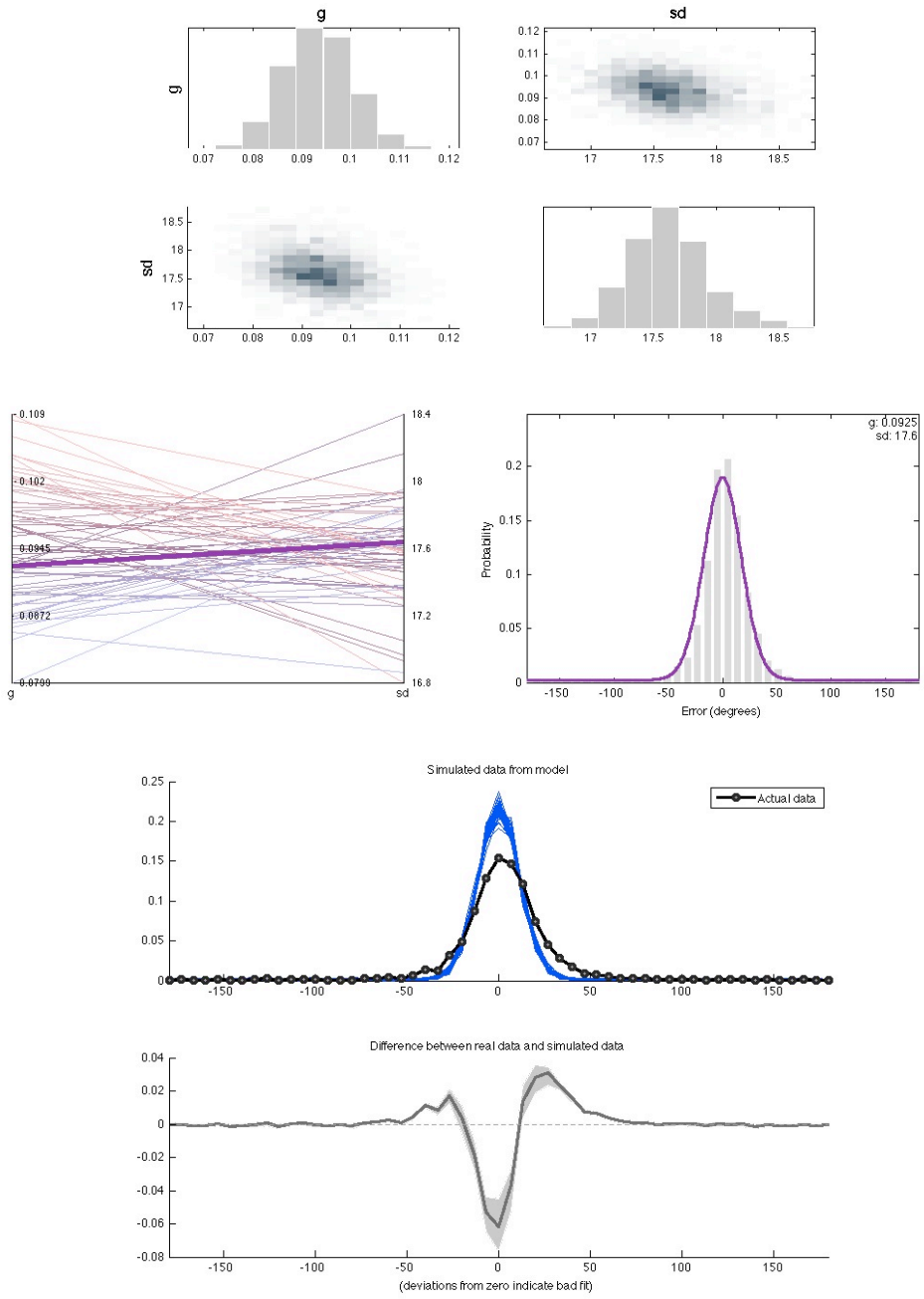


This is mostly useful for familiarizing yourself with the behavior of the model, and for recognizing gross inconsistencies between the model and data. At first pass, this looks okay.

Next, the toolbox will ask:

```
Would you like to see the tradeoffs
between parameters, samples from the posterior
distribution and a posterior predictive check? (y/n): y
```

Answering in the affirmative will bring up three plots. The first shows the full posterior distribution for each parameter and a heatmap for each pair of parameters that can be useful for diagnosing correlations.



Demo 6: Using the parallel toolbox to speed up the analysis.

The MemToolbox is compatible with the Parallel Processing toolbox, and using it can achieve a big speedup. To use it, run the command `matlabpool open;`. You can automate this within your program by opening the pool only if it isn't already open:

```
if(~(matlabpool('size') > 0))
 matlabpool open;
end
```

Demo 7: Model comparison.

A good example for this would be to start with StandardMixtureModel vs. StandardMixtureModelWithBias, and point out how the posterior predictive check shows you exactly where the unbiased one fails. Then compare StandardMixtureModelWithBias to Student's *t* model, and again point out how the posterior predictive check shows you where the non-peaked one fails.

Demo 8: Creating your own model.