MemToolbox tutorial

(This tutorial uses a beta version of the toolbox and was last updated on 6/18/2012.)

The MemToolbox is a collection of MATLAB functions designed for analyzing data from studies of visual working memory. In particular, it contains functions that make it easy 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. The toolbox comes loaded with a number of models—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];
```

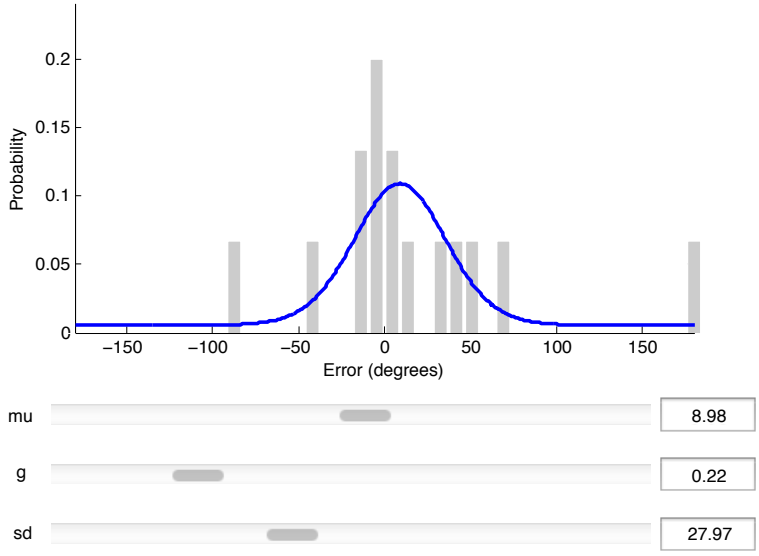
Next, call `MemFit` on the error vector.

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

The toolbox will now 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 on your behalf. Choosing a different model is easy. The toolbox comes loaded 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, provide 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.

The two models we have seen 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 SwapModel

   SwapModel 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
```

Now let's test out the model. Rather than making you type out a long vector of errors and matrix of distractor values, we'll use one of the datasets that comes with the toolbox. These can be accessed through the function `MemDataset`, which returns a structure array with fields containing different aspects of the data. For example, data set #3 includes a vector of errors and distractor values for each trial, just what you need to use the swap model:

```
>> data = MemDataset(3)

data =

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

Now, instead of giving `MemFit` the error vector, you can give it the entire data structure:

```
>> 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 it all.

Demo 4: Digging deeper into MemFit()

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 `MemFit`. 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`. 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 MemToolbox uses multiple chains, whose starting values are specified in the model file (and modifiable), and continues running them until they have converged to a comparable range of parameters. Every 500 steps, the toolbox will update you on its progress:

```
... not yet converged (500)
```

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

```
... chains converged after 1000 samples!
```

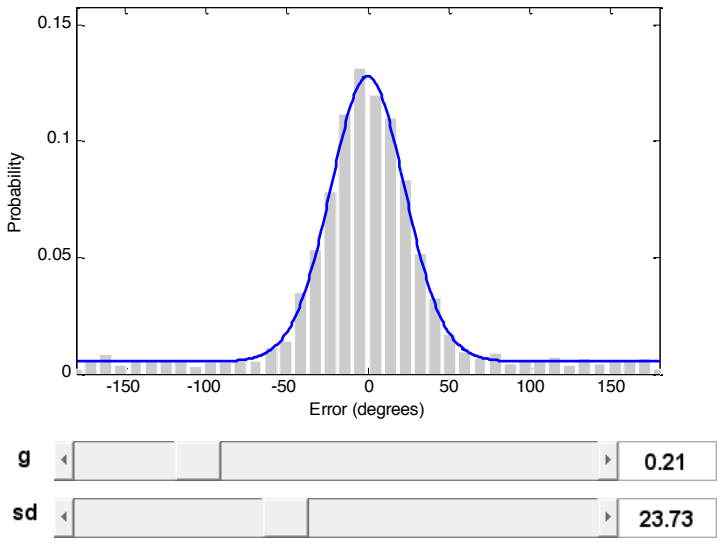
The final estimate is based on samples taken after convergence:

```
... 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:



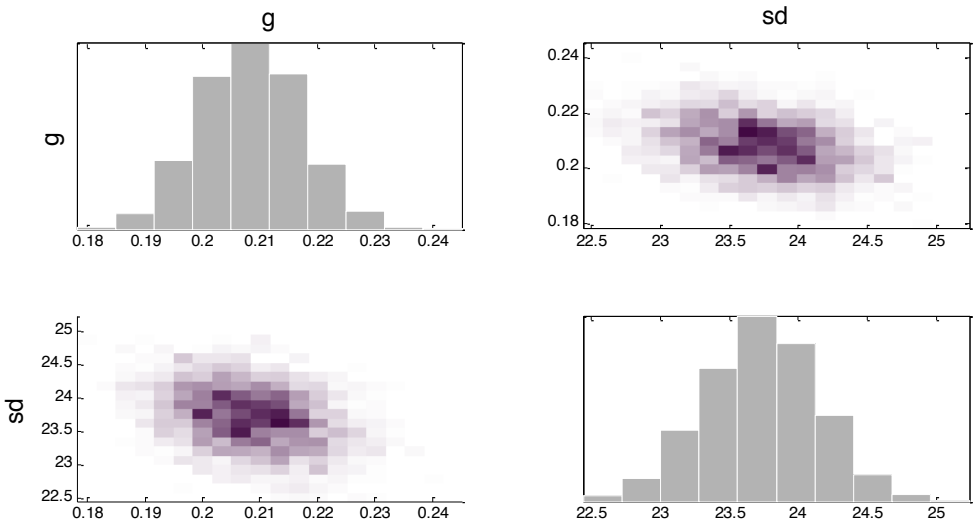
This is mostly useful for familiarizing yourself with the behavior of the model, and for recognizing gross inconsistencies between the model and data. At first pass, this looks okay.

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:

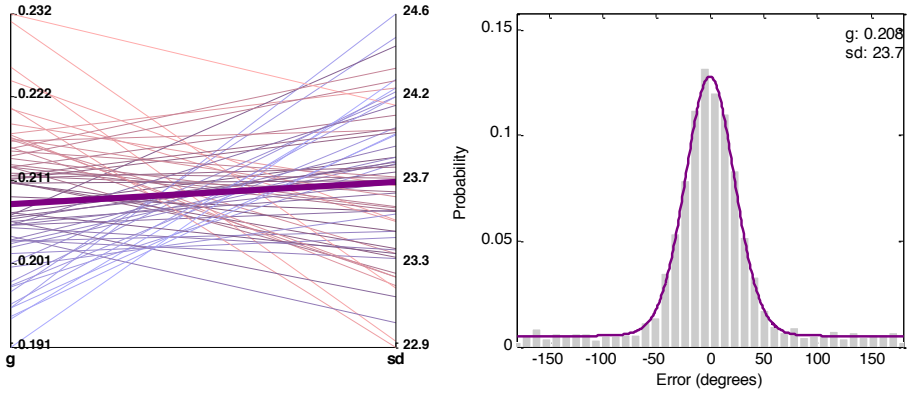
1) The first shows the full posterior distribution for each parameter and a heat map for each pair of them. This can be useful for diagnosing correlations between parameters and for understanding how much the data constrain the parameters:



For example, in the standard Zhang & Luck (2008) mixture model, there is a correlation between the guess rate parameter (*g*) and the standard deviation parameter (*sd*). The data is equally consistent with a slightly higher guess rate and slightly lower standard deviation, or with a slightly lower guess rate and slightly higher standard deviation. This is apparent in the negative slope of the heat map plots in the bottom left and upper right quadrant.

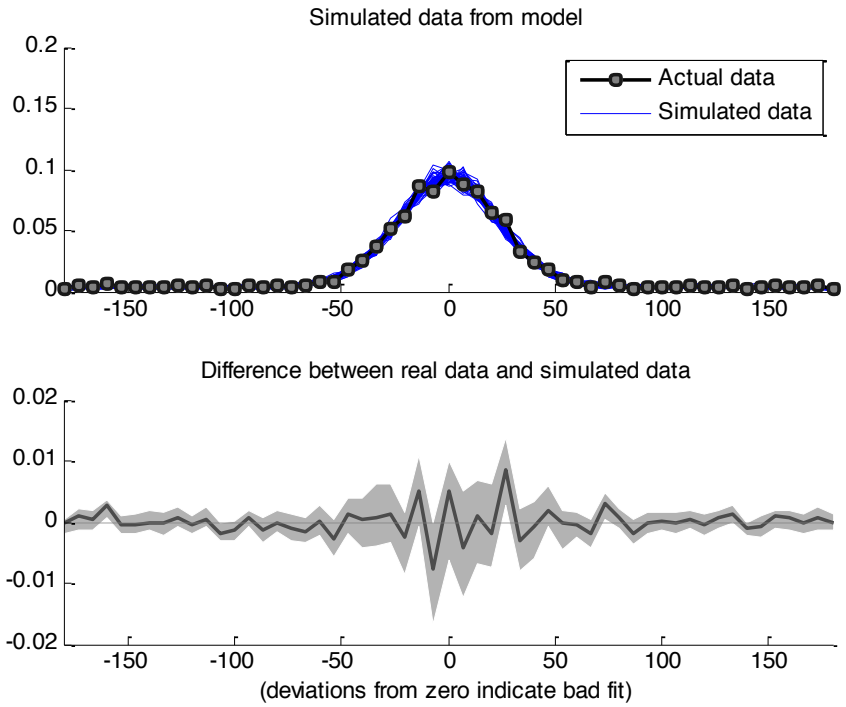
2) The second plot produced by MemFit shows the parameters of the model in a parallel coordinates plot (http://en.wikipedia.org/wiki/Parallel_coordinates), along with a visualization of their fit to the data. Each line on the parallel coordinates plot corresponds to a reasonable set of parameters for fitting the data (e.g., a sample from the posterior on the parameters given the data). The lines are colored such that higher values on the first parameter (*g*) get red lines, and lower values get blue lines. Thus, the negative correlation between the parameters of this particular model is indicated by the fact that on the right side of the plot, the red lines tend to be lower than the blue lines (the opposite of the left side). A positive correlation would be shown by the lines being mostly parallel between the left and right sides, meaning high values of one parameter give high values of the other.

With only two parameters, the parallel coordinates plot does not provide as much information as the scatter plots and heat maps in the first figure. However, when more parameters are present, parallel coordinates plots allow you to simultaneously understand how all of the parameters relate to each other, rather than just how pairs of parameters relate; we will explore this later in the tutorial.



This plot is also interactive. Try clicking one of the lines with a high value of guess rate (*g*). The plot of the data on the right will redraw with the new parameter values you chose. Now try clicking on one of the lines with a low value of guess rate (*g*). Can you see why both provide equally good fits?

3) The third plot shows the *posterior predictive distribution*—the model's residual. The first two plots provide ways of examining which parameter values of the model provide the best fits. This figure instead shows whether or not the model—with its best parameter values—provides a good fit to the data. In particular, we simulate 'fake' data from the posterior of the model and ask whether it looks like the real data. If the model provides a good fit, then the data that is simulated using the best fit parameters (the posterior) will closely resemble the actual data. If the model is systematically wrong in some way, this should be (visually) evident as a mismatch between the simulated and actual data. The top of the plot shows the data in black, and the simulated data in blue. They should be on top of each other if the model provides a good fit. The bottom of the plot shows the difference between the two, both on average (line) and with 95% confidence intervals (gray shading). Any regions where the gray shading does not include zero are regions where the model is systematically wrong.



Demo 5: Using the parallel toolbox to speed up the analysis.

The MemToolbox is compatible with the Parallel Computing 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 if it isn't already open:

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

If your machine has multiple processors or multiple cores, then all of the fitting functions — MCMC, MLE, etc. — will work many times faster after you run `matlabpool`.

Demo 6: Prettier plots of the posterior

Using MCMC means that we are sampling from the posterior, rather than evaluating it directly. Thus, even though the true posterior is almost always a smooth function of the parameters, the plots obtained from the standard `MemFit` call will not always be smooth. We can fix this by using `GridSearch` rather than MCMC to refine our graph. In particular, given the results of a standard call to `MemFit`:

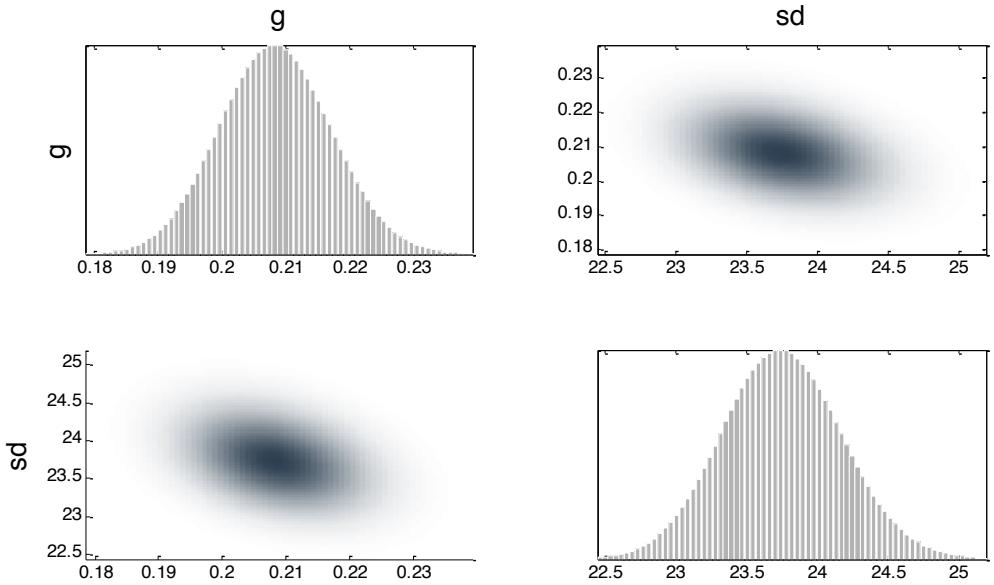
```
>> data = MemDataset(3);
>> model = StandardMixtureModel();
>> fit = MemFit(data, model);
```

We can make a more refined version of the first figure from Demo 4:

```
>> fullPosterior = GridSearch(data, model, ...
    'PosteriorSamples', fit.posteriorSamples)
```

```
>> PlotPosterior(fullPosterior, model.paramNames)
```

Then, rather than the noisy sampling-based-plot, we can get a full posterior :



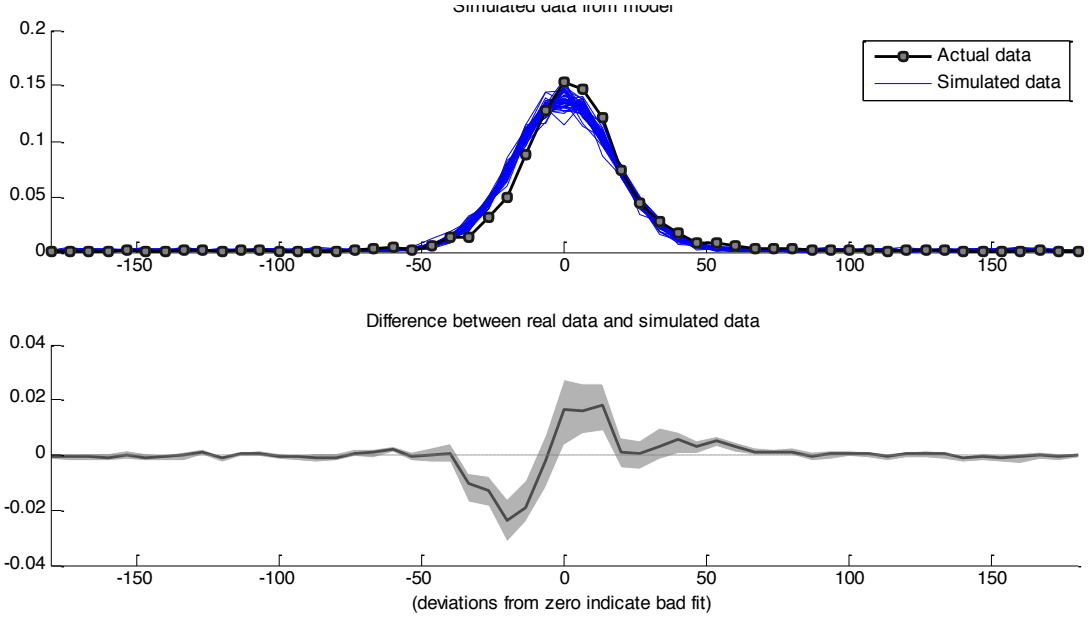
This example also gives you a little bit of a peak under the hood of `MemFit` – in particular, the separate fitting functions of MemToolbox (try `help MemFitting`) and plotting functions (`help MemPlots`). We first fit the data using `GridSearch` and obtain the posterior (which we call `fullPosterior`, since it came from `GridSearch`; the results from MCMC we call `posteriorSamples`). Then we call the plot function `PlotPosterior`, which takes either a `fulPosterior` or `posteriorSamples` as the first parameter, and the names of the model's parameters as the second parameter. More advanced uses of the MemToolbox will generally require separate calls to the MemFitting and MemPlots functions.

Demo 7: Model comparison

In this demo we'll cover model comparison. Let's start by using the standard mixture model on the first built-in dataset:

```
>> data = MemDataset(1);
>> model1 = StandardMixtureModel();
>> MemFit(data,model1)
```

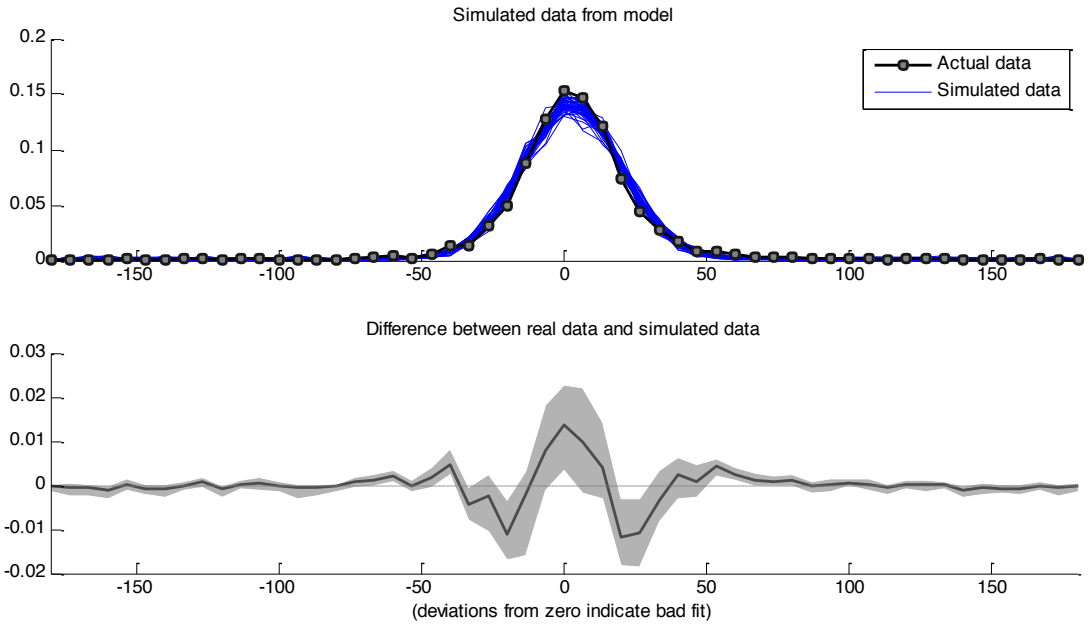
We can see that the residuals for this model show that it does not fit well:



In particular, it looks like the model overestimates counterclockwise error and underestimates clockwise errors. This particular subject had a clockwise-bias in their responses, and the `StandardMixtureModel` is always centered around zero. To fix this, we can use a different model. In particular, we can use the same model, but allow a bias parameter to shift the model's center point:

```
>> model2 = StandardMixtureModel('Bias', true);
>> MemFit(data,model2)
```

This model fits much better than the one with no bias.



However, we can see that this model also seems to be systematically incorrect. In particular, the data appear to be 'peakier' than this model suggests they should be. Even so, this model is still an improvement on the model that assumes no bias. How can we quantify this? MemToolbox has built-in model comparison metrics. In particular, simply calling `MemFit` with more than one model automatically compares them:

```
>> MemFit(data, {model1, model2})
```

This gives the following output:

```
You've chosen to compare the following models:
  1. Standard mixture model
  2. Standard mixture model with bias

Just a moment while MTB fits these models to your data...

model      log L    prop. preferred   log Bayes factor
-----      -----    ---------------   ----------------
1          -14430       0.0069        -4.96
2          -14397       0.9931        4.96
```

It compares the models using a Bayes Factor. The simplest interpretation of the Bayes factor is found in the "proportion preferred" column, which says how often one model should be preferred over the other. In this case, we have greater than 99% certainty ($p < 0.01$) that the second model is to be preferred to the first.

Fin.

Demo N-1: Arranging your plots (…, 'NewFigure', false); subplot(); subfigure(); PalettableX()

Demo N: Creating your own model.