NeuroCognitive Memory Search Toolbox Demonstration

This demonstration shows how the NCMS modeling framework was used to allow model parameters to vary according to normalized fMRI BOLD reponses recorded from participants during a free-recall task, as reported by Kragel et al. (2015).

Kragel, J. E., Morton, N. W, and Polyn, S. M. (2015) Neural activity in the medial temporal lobe reveals the fidelity of mental time travel. Journal of Neuroscience, 35 (7), 2914-2926.

Note: Part of this demonstration requires the use of the Behavioral Toolbox (release v1.01), available from the University of Pennsylvania (http://memory.psych.upenn.edu/Behavioral_toolbox).

```
%%
% Baseline
```

The first part of this tutorial reproduces the likelihoods of parameter fits to models examined by Kragel et al. (2015). We start with the neurally-naïve baseline model, in which parameter values were evaluated by the model's ability to produce observed sequences of freely-recalled items. In this baseline model, parameters were fixed across all individual recall events, with no access to neural signals from any brain region.

```
% Evaluate likelihood of baseline model
[param_info_base, fixed] = search_param_KragEtal15('base');
```

The *search_param_KragEtal15* function provides the organizational information of the searched (**param_info_base**) and fixed (**fixed**) parameters for the baseline model, as specified by the input string 'base'. Since most parameter search algorithms simply operate on a vector of values, the **param_info_base** structarray allows both the model and the user to interpret which value corresponds to which parameter in the model. The parameters in **param_info_base** were optimized during the search process (particle swarm optimization in the case of Kragel et al., 2015), and the parameters in the struct **fixed** were held constant during the optimization. In general, the *search_param_KragEtal15* function can return a number of different variants of the search parameters based on the given input string (i.e., 'base', 'tr', 'rs', 'joint'; see Kragel et al., 2015, for details).

```
load('data_KragEtal15.mat','data');
```

Here we load the observed behavioral data from Kragel et al., 2015 (in the variable **data**). Of particular use to the model is the matrix of recall sequences for each trial for each participant (**data.recalls**). Each row of **data.recalls** corresponds to recalls made on a single trial for a single participant, with each value indicating the original list position of the recalled word (and 0 indicating no more recalls were made).

```
fstruct = fixed;
```

```
fstruct.param_info = param_info_base;
fstruct.f_logl = @cmr_general;
fstruct.data = data;
fstruct.neural_mat = [];
```

Next we gather all the information the model will need to operate in a struct variable named **fstruct**, initialized to include the fixed parameters in **fixed**, then extended with the searched parameter information (names and ranges) of the optimized parameters from **param_info_base**. We include the behavioral **data**, to give the model access to the observed recall sequences generated by participants in the lab during the free-recall task. This allows us to evaluate the log-likelihood (or log-transformed probability) of observing a given set of recall sequences given a specified modeling framework. Here, we use a variant of the Context Maintenance and Retrieval modeling framework (Polyn et al., 2009) to observe the likelihood that this specification of the model would generate the observed recalled sequences. The particular modeling framework used is specified in **fstruct.f_logl**. Specifically, this provides a function handle to the function that will be generating the log-likelihood scores (in this case, *cmr_general*). Note that since parameters are not influenced by any neural signal during the model's evaluation, the **neural_mat** field of **fstruct** is left empty.

```
load('param_KragEtal15.mat','param_vec_base');
```

We now have the necessary information gathered to evaluate the likelihood of any set of parameters. Here, we load the optimized parameter values for the baseline model as determined by the particle swarm optimization, as saved in the *param_KragEtal15.mat* file. The specific best-fitting parameters for the baseline model are in the variable **param_vec_base**, as reported in Kragel et al., 2015. Note that this variable is just a vector of values, which must be interpreted using the information contained in **param_info_base**. Also note that this tutorial does not show the actual parameter search process, but rather provides the result of the performed optimization.

```
LL = eval_param_KragEtal15_base(param_vec_base, fstruct)
```

Using the parameter values specified in **param_vec_base**, we can determine the likelihood that the *cmr_general* model—evaluated using these specific parameter values—would generate this observed set of recall sequences. (These values are log transformed for ease of interpretation, since such a probability out of any number of potentially possible recall sequences will be generally quite low). This *eval_param_KragEtal15_base* function unpacks the parameters from the values in the vector **param_vec_base** (using the information in **param_info_base**) then evaluates the specified model in accordance with the provided data. Executing this portion of the tutorial should reproduce the likelihood for the neurally-naïve baseline model reported in Table 1 of Kragel et al., 2015.

```
%%
% Joint
```

Having produced the likelihood score for the neurally-naïve baseline model, here we produce the likelihood score for a model that uses the neural signal to inform the value of a specified parameter for each recall event. Specifically, we show the variant of the CMR model that allows both the contextual reinstatement parameter (β_{rec}) and the stopping parameter (ξ_d) to be linearly scaled by a neural signal as measured at each recall event (in this case, normalized BOLD activity from a cluster of voxels as specified in Kragel et al., 2015).

```
% Evaluate likelihood of neurally-informed joint model
[param_info_joint, fixed] = search_param_KragEtal15('joint');
```

As this model uses two more parameters, each used to scale the neural signal's influence on one of the parameters of interest, we use a different parameter specification, by providing the 'joint' string to *search_param_KragEtal15*. This gives us the new parameter specifications in **param_info_joint** (as well as the **fixed** parameters).

```
load('data_KragEtal15.mat','data');
load('neural_mat_KragEtal15.mat','neural_mat_joint');
```

We load the observed behavioral data as before, but here we also load the neural data. The *neural_mat_KragEtal15.mat* file contains matrices of the neural signal for each recall event in the behavioral **data.recalls** (and 0 elsewhere). Specifically, here we are interested in the normalized BOLD activity in the cluster of voxels found to be informative to both model processes examined by Kragel et al. (2015), so we use the **neural_mat_joint** variable.

```
fstruct = fixed;
fstruct.param_info = param_info_joint;
fstruct.f_logl = @cmr_general;
fstruct.data = data;
fstruct.neural_mat = neural_mat_joint;
```

We collect the relevant information needed by the modeling framework, as before, but here we also provide the measured neural signal for each recall event in **fstruct.neural_mat.**

```
load('param_KragEtal15.mat','param_vec_joint');
```

We then load the vector of optimized parameters, as determined by a particle swarm optimization. These values are reported for the joint model in Kragel et al., (2015), and can be interpreted using the information in **param_info_joint.**

```
LL = eval_param_KragEtal15_joint(param_vec_joint, fstruct)
```

We are then able to evaluate this new set of parameters, while also using the neural signal. This *eval_param_KragEtal_joint* function, in addition to unpacking and preparing the parameter information as in the baseline model, creates a matrix of individual parameter values for the contextual reinstatement parameter (β_{rec}) and the stopping parameter (ξ_d)

for each recall event. This value is determined by base values for each parameter, and linearly scaled by an additional ν parameter according to the value of the measured neural signal at that particular recall event. This **var_param** variable informs the model that this "changing" variable value should be used when recall events are evaluated, thus affecting the likelihood of recalling individual events, which affects the likelihood of recall sequences, and thus the likelihood of the entire set of observed recall sequences. This portion of the tutorial should reproduce the likelihood score for the joint neural model reported in Table 1 of Kragel et al., 2015.

The remainder of this tutorial focuses on the generation of synthetic data according to the specifications of the model. The same core functions are used to run the internal processes of the model (coordinated by the *cmr_general* function), but rather than evaluating probabilities of recall events, synthetic recall events are sampled based on those same probabilities (including the probability of stopping).

```
%%
% Behavioral Data
list_length = fstruct.data.listLength;
figure(1)
summary_plots(data.recalls, data.subject, list_length);
suptitle('Behavioral Data')
```

First we show how observed behavioral data exhibits the standard free-recall phenomena, by examining **(a)** the serial position curve (recall by list position), **(b)** probability of first recall, as a function of list position, **(c)** lag-conditional response probability, **(d)** stop probability as a function of output position. These measures are reported in Kragel et al. (2015). Note that *summary_plots* requires the use of the Behavioral Toolbox (release 1.01) as described in the main README file for this toolbox. Additionally, earlier versions of MATLAB may not include the *suptitle* function. This line is mainly used to identify multiple figures, and this it can be commented out for ease of execution if necessary.

```
%%
% Generate synthetic data

% Generative simulation 1: Neurally-naive baseline model.
fstruct.param_info = param_info_base;
fstruct.neural_mat = [];
fstruct.n_rep = 10;
```

Here we prepare to generate data using the neurally-naïve baseline model. Once again we prepare the information the model needs to operate, specifically information about which value in the **param_vec_base** vector corresponds to which parameter (as indicated by **param_info_base**). We also set the model to generate 10 times the

number of observed trials (as specified by **n_rep**), in order to get a more stable estimate of average model performance.

```
seq_base = gen_KragEtal15_base(param_vec_base, fstruct);

figure(2)
model_subj = ones(size(seq_base,1),1);
summary_plots(seq_base, model_subj, list_length)
suptitle('Neurally-Naive Baseline')
```

We use the generative version of the model (*gen_KragEtal15_base*) to generate synthetic sequences of recall events, in the variable **seq_base**. Note that **seq_base** will have the same layout as the observed **data.recalls** sequences. We imagine that these recall sequences all came from one "model" subject, and create a dummy **model_subj** vector to indicate that for the analysis functions. To observe how this generated data exhibit the standard free-recall phenomena, we again use *summary_plots* to examine the SPC, PFR, lag-CRP, and Stop Probability, just as we did with the observed behavioral data. Tweaking various model parameters and rerunning this analysis will allow you to see how various parameters affect the shape of these curves.

Just as we can examine how individual parameter values influence these summary measures, we can somewhat examine how synthetic "neural" that affects the value of a parameter influence expected behavior as well.

```
% Generative simulation 2: Set neural signal to be constant and
% high (+1) for every recall event.  Generate 10x observed trials.
fstruct.param_info = param_info_joint;
fstruct.neural_mat = 1 * ones(size(neural_mat_joint));
fstruct.n_rep = 10;
```

Once again, we set the necessary **param_info_joint** to interpret the values in **param_vec_joint**, and we set the model for 10 repetitions of trials. Additionally, we create a matrix of synthetic neural data where the neural measure is "high" (equal to 1) for each possible recall event. We can then observe how we expect how allowing this high neural signal to affect temporal reinstatement and the stopping mechanism will be reflected in the expected behavior.

```
seq_high = gen_KragEtal15_joint(param_vec_joint, fstruct);

figure(3)
model_subj = ones(size(seq_high,1),1);
summary_plots(seq_high, model_subj, list_length)
suptitle('High Neural Signal')
```

We generate the synthetic recall sequences according to the model (here, called by *gen_KragEtal15_joint*) and examine the dependent measures under the model. Specifically, we will visually be able to observe the differences when compared with a model in which neural signal is "low" (-1) for all recall events.

```
% Generative simulation 3: Set neural signal to be constant and
% low (-1) for every recall event.  Generate 10x observed trials.
fstruct.param_info = param_info_joint;
fstruct.neural_mat = -1 * ones(size(neural_mat_joint));
fstruct.n_rep = 10;
```

We set the model up as before, but this time we set the neural signal to “low” (-1) for all recall events.

```
seq_low = gen_KragEtal15(param_vec_joint, fstruct);

figure(4)
model_subj = ones(size(seq_low,1),1);
summary_plots(seq_low, model_subj, list_length)
suptitle('Low Neural Signal')
```

After generating the synthetic recall sequences for this “low-signal” model, we can observe how the expected model behavior differs from the “high-signal” model. Notice that when neural signal is used to adjust model parameters within the model’s execution, high neural signal results in a sharper lag-CRP, indicating a higher degree of temporal organization. This is to be expected with a higher degree of temporal reinstatement, as more effectively reinstated contextual states will provide good cues for neighboring study items during recall.

Similarly, we can see the effect of allowing the neural signal to influence the value of the stopping parameter. Here, a higher neural signal corresponds to a lower rate at which stop probability rises over output position, as compared with a model with low neural signal for each recall event.

While these “all-high” or “all-low” neural models are seemingly heavy-handed, these brief demonstrations reflect similar results of similar more-extensive analyses performed on neural signals at individual recall events (Kragel et al., 2015, see Fig. 5). By extending these analysis techniques to examining neural signals at individual events, the modeling framework can be utilized to better understand which neural signals help specific model mechanisms better explain patterns of observed behavior.