COGS 205B: Final assignment

The final assignment will require you to write code that is very similar to code you might end up using in practice. It has two main components:

1. A handle class, `@Metropolis`, that enables Metropolis sampling
2. A small number of functions, possibly only one, that implement specific log-posterior functions
3. A main script called `final.m`

@Metropolis

Pseudocode for the Metropolis algorithm can be found in the slides in chapter 500. I have also written a skeleton version of the classdef file for you, as well as a test suite. I have placed both in your assignment directory. You can call the tests with `Metropolis.test`. Currently all the tests fail.

You should add at least one more method, `.DIC()`, which computes DIC $= M(D) + V(D)/2$, where D is the vector of deviance ($-2\log(\text{likelihood})$) values at all sampled proposals, $M(\cdot)$ is the mean and $V(\cdot)$ is the variance. DIC is a rudimentary model comparison metric, with lower values meaning better model performance.

Specific log-posterior functions

The application is in the domain of reaction time (RT) analysis. I have written a function, `getFinalData`, that simulates data for you. The RT data come from a simulated experiment with three conditions and 128 data points per condition. We are interested in knowing the relationship between these conditions.

Specifically, we want to analyze these data using a two-parameter Weibull distribution $W(A, B)$ with parameters Scale A and Shape B. We want to estimate these two parameters in each condition. To that end, you should create one or more functions to evaluate at least two log-posterior functions. You may implement these as function files, as a class, as a package, or as anonymous functions, as you prefer.

The saturated model

The log-posterior for the saturated model has six free parameters:

Condition	Scale	Shape	Likelihood	Prior
Easy	A_e	B_e	$x_i \mid \text{Easy} \sim W(A_e, B_e)$	$A_e, B_e \sim Exp(1)$
Medium	A_m	B_m	$x_i \mid \text{Medium} \sim W(A_m, B_m)$	$A_m, B_m \sim Exp(1)$
Hard	A_h	B_h	$x_i \mid \text{Hard} \sim W(A_h, B_h)$	$A_h, B_h \sim Exp(1)$

The log-posterior for the saturated model is the sum of all the relevant log-likelihoods and log-priors. Your function should take as input the six free parameters (A_e, A_m, A_h, B_e, B_m, B_h) and return as output the unscaled log-posterior f_s:

$$f_s(A_e, A_m, A_h, B_e, B_m, B_h \mid x) = \log(p(x \mid A_e, A_m, A_h, B_e, B_m, B_h)) + \log(p(A_e, A_m, A_h, B_e, B_m, B_h))$$

The constrained model

Critically, we will compare this saturated model to a constrained model, in which the parameters of the Weibull are not free to vary between conditions, but are rather linear functions:

Condition	Scale	Shape	Likelihood
Easy	$A_e = \beta_0^A$	$B_e = \beta_0^B$	$x_i \mid \text{Easy} \sim W(A_e, B_e)$
Medium	$A_m = \beta_0^A + \beta_1^A$	$B_m = \beta_0^B + \beta_1^B$	$x_i \mid \text{Medium} \sim W(A_m, B_m)$
Hard	$A_h = \beta_0^A + 2\beta_1^A$	$B_h = \beta_0^B + 2\beta_1^B$	$x_i \mid \text{Hard} \sim W(A_h, B_h)$

Where the new parameters are:

Parameter	Symbol	Prior
Scale intercept	β_0^A	$\beta_0^A \sim \text{Exp}(1)$
Scale slope	β_1^A	$\beta_1^A \sim N(0,1)$
Shape intercept	β_0^B	$\beta_0^B \sim \text{Exp}(1)$
Shape slope	β_1^B	$\beta_1^B \sim N(0,1)$

The log-posterior for the saturated model is the sum of all the relevant log-likelihoods and log-priors. Your function should take as input the four free parameters (β_0^A, β_1^A, β_0^B, T) and return as output the unscaled log-posterior f_c:

$$f_c(\beta_0^A, \beta_1^A, \beta_0^B, \beta_1^B \mid x) = \log(p(x \mid \beta_0^A, \beta_1^A, \beta_0^B, \beta_1^B)) + \log(p(\beta_0^A, \beta_1^A, \beta_0^B, \beta_1^B))$$

final.m

Your entry point script should be called `final.m`. It should look a lot like this:

```
%% Final assignment <your name>
clear

%% Get data
data = getFinalData();

%% Saturated model first
saturatedTarget = @(parameter) SaturatedLogPosterior(parameter, data);
saturated = Metropolis(saturatedTarget, [2 2 2 2 2 2]');
saturated.DrawSamples(10000)
saturated.disp

%% Constrained model next
constrainedTarget = @(parameter) ConstrainedLogPosterior(parameter, data);
constrained = Metropolis(constrainedTarget, [2 0 2 0]');
constrained.DrawSamples(10000)
constrained.disp

## Compare the two models
saturated.DIC - constrained.DIC

## Conclude
```

In the concluding comment, please answer:

- Which model fits better?
- In the constrained model, does the Scale parameter go up or down with the condition difficulty?
- What about the Shape parameter?