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 three main components:

1. A handle class, `final.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.main`

Please place the class, the main script, and all other needed functions in the new package folder called `+final`.

@Metropolis

Pseudocode for the Metropolis algorithm can be found below (and 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 `run(final.Metropolis.test)`. Currently most tests fail.

One important method, `.DIC()`, should compute DIC $= M(D) + V(D)/2$, where D is the vector of -2 log posterior $(-2\log(p(\theta|X)))$ 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.

Algorithm pseudocode

Given a function $f(\theta) \propto p(\theta|D)$, and a symmetric *candidate generating distribution* $Q(a|b) = Q(b|a)$, a Metropolis sampling algorithm proceeds as follows:

```
1. Set $i \leftarrow 1$ and choose $R \leftarrow 10000$
2. Choose, arbitrarily, $\theta^{(0)}$
3. Draw a randomly selected $\theta^c$ from $Q\left(\theta|\theta^{(i-1)}\right)$
4. Compute the log acceptance ratio $\log(\alpha) = \log(f(\theta^c)) - \log(f(\theta^{(i-1)}))$
5. Draw a randomly selected $u$ from $U(0,1)$. If $\log(\alpha) > \log(u)$, set $\theta^{(i)} \leftarrow \theta^c$, otherwise set $\theta^{(i)} \leftarrow \theta^{(i-1)}$
6. Set $i \leftarrow i+1$. If $i \leq R$, return to Step 3, otherwise halt
```

We then compute the mean, variance, standard deviation, etc. of the sampled θ values. Remember to discard the first B samples, where B is the burn-in, before calculating summary statistics.

Specific log-posterior functions

The application is in the domain of reaction time (RT) analysis. I have written a class, `final.getFinalData`, that simulates data for you. The RT data come from a simulated experiment with a single participant, three conditions, and 512 reaction time data points per condition:

```
>> data = final.getFinalData()
data =
  data set with conditions:
      "easy": [512x1] double
    "medium": [512x1] double
      "hard": [512x1] double
```

We are interested in knowing the relationship between these conditions, mainly to confirm if the "easy" condition is indeed easier and the "hard" condition is indeed harder. (It is possible those labels are completely wrong.)

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

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 $\log(f_s)$:

$$\begin{aligned}\log(f_s(A_e, A_m, A_h, B_e, B_m, B_h \mid X)) &= \log(p(X \mid A_e, A_m, A_h, B_e, B_m, B_h)) \\ &+ \log(p(A_e, A_m, A_h, B_e, B_m, B_h))\end{aligned}$$

Here, the likelihood $p(X \mid A_e, A_m, A_h, B_e, B_m, B_h)$ is the product of the likelihood at each data point, with the likelihood at each data point depending on the condition. Specifically,

$$\begin{aligned}p(X \mid A_e, A_m, A_h, B_e, B_m, B_h) &= \left[\prod_{i=1}^{n_e} W(x_i^{\text{easy}}|A_e, B_e)\right] \\ &\times \left[\prod_{i=1}^{n_m} W(x_i^{\text{medium}}|A_m, B_m)\right] \\ &\times \left[\prod_{i=1}^{n_h} W(x_i^{\text{hard}}|A_h, B_h)\right],\end{aligned}$$

where $n_e = n_m = n_h = 512$.

Moreover,

$$\begin{aligned}p(A_e, A_m, A_h, B_e, B_m, B_h) &= \text{Exp}(A_e|1) \\ &\times \text{Exp}(B_e|1) \\ &\times \text{Exp}(A_m|1) \\ &\times \text{Exp}(B_m|1) \\ &\times \text{Exp}(A_h|1) \\ &\times \text{Exp}(B_h|1).\end{aligned}$$

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	$\beta_1^A \sim \text{Laplace}(0,1)$
Shape intercept	β_0^B	$\beta_0^B \sim \text{Exp}(1)$
Shape slope	β_1^B	$\beta_1^B \sim \text{Laplace}(0,1)$

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:

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

Here, the likelihood $p(X \mid \beta_0^A, \beta_1^A, \beta_0^B, \beta_1^B)$ has a very similar form to before. Again, it is the product of the likelihood at each data point, with the likelihood at each data point depending on the condition. Specifically,

$$\begin{aligned} p(X \mid \beta_0^A, \beta_1^A, \beta_0^B, \beta_1^B) &= \left[\prod_{i=1}^{n_e} W(x_i^{\text{easy}} | A_e, B_e)\right] \\ &\times \left[\prod_{i=1}^{n_m} W(x_i^{\text{medium}} | A_m, B_m)\right] \\ &\times \left[\prod_{i=1}^{n_h} W(x_i^{\text{hard}} | A_h, B_h)\right], \end{aligned}$$

with $n_e = n_m = n_h = 512$ and:

$$\begin{aligned} A_e &= \beta_0^A \\ B_e &= \beta_0^B \\ A_m &= \beta_0^A + \beta_1^A \\ B_m &= \beta_0^B + \beta_1^B \\ A_h &= \beta_0^A + 2\beta_1^A \\ B_h &= \beta_0^B + 2\beta_1^B. \end{aligned}$$

Finally,

$$\begin{aligned} p(\beta_0^A, \beta_1^A, \beta_0^B, \beta_1^B) &= \text{Exp}(\beta_0^A | 1) \times \text{Laplace}(\beta_1^A | 1) \\ &\times \text{Exp}(\beta_0^B | 1) \times \text{Laplace}(\beta_1^B | 1). \end{aligned}$$

Some helpful hints

- Don't forget to take logarithms of the likelihood and prior distributions.
- It is helpful to consider that in the Metropolis algorithm, the target function can be specified up to a multiplicative constant: $f(\theta) \propto p(\theta | D)$.

MATLAB having trouble dealing with super small ab, thus, we should better to log(a)+log(b)+… for each data points then sum up
log(ab) = log(a) + log(b)

- The multiplicative constant becomes an additive constant on the log scale.
- For example, a Laplace prior distribution looks like this: $\text{Laplace}(x|a,b) = \frac{1}{2b}\exp\left(-\frac{|x-a|}{b}\right)$, so $\log[\text{Laplace}(x|0,1)] = -\log(2) - |x|$, and you can ignore the constant $-\log(2)$.
- MATLAB has multiple functions built in to evaluate the Weibull distribution, including `wbllike` – but don't get confused by the signs!
- While allowed, it should not be necessary to edit `Metropolis.m` at all.
- Make sure that your final product passes the entire test suite. Note that the test suite assumes you do not change the names of methods or properties.

final.main

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

```
%% Final assignment [your name]
% This is my solution to the final assignment for COGS 205B: Computational Lab Skills
clear

%% Get data
data = final.getFinalData();

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

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

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

%% Conclude
% The model that fits better is the [saturated|constrained] model.
%
% The Scale parameter [goes up|goes down|remains constant] from the "easy" to the "hard" condition.
%
% The Shape parameter [goes up|goes down|remains constant] from the "easy" to the "hard" condition.
```

Please edit the comments in square brackets `[]` in `+final/main.m` so the statements are true.