Assignment 8 — 2.086 Fall 2015

Due: *Tuesday, 24 November, at 5 PM.*

Upload your solution to Stellar as a zip file "`YOURNAME_ASSIGNMENT_8.zip`" which includes the script for each question in the proper format as well as any MATLAB functions (of your own creation) your scripts may call. Remember to verify your upload before the assignment due time by downloading your solution folder yourself and re-running `grade_o_matic`. Full instructions for assignment preparation are given in the document "Preparing assignments for `grade_o_matic`" available on the 2.086 website.

Questions

1. (20 points) Graduate students admitted to a particular university apply for two national fellowships, Fellowship 1 and Fellowship 2. We consider the application by any particular student for the two fellowships a random experiment. The outcome of the experiment is described by two variables $\mathbb{F}_1$ and $\mathbb{F}_2$. The variable $\mathbb{F}_1$, related to Fellowship 1, has two possible outcomes, W_1 and L_1: W_1 corresponds to "win Fellowship 1" and L_1 corresponds to "lose (do not win) Fellowship 1". The variable $\mathbb{F}_2$, related to Fellowship 2, has two possible outcomes, W_2 and L_2: W_2 corresponds to "win Fellowship 2" and L_2 corresponds to "lose (do not win) Fellowship 2". From this description you should be able to deduce the sample space of outcomes in terms of which you can then define any events of interest.

   The historical data is summarized in a $2 \times n$ logical array `fellowship_record`: the two rows correspond to the two fellowships, and the n columns to the n students who have applied for the two fellowships in the past. The entries of `fellowship_record` are either a logical 0 or a logical 1: a 0 indicates "lose" and a 1 indicates "win". For example, a 0 in `fellowship_record(1,9)` would indicate that student 9 did not win Fellowship 1; a 1 in `fellowship_record(2,11)` would indicate that student 11 did win Fellowship 2.

   We would like you to write a script which, on the basis of the historical data, calculates $\varphi_n(W_1, W_2)$, $\varphi_n(L_1, L_2)$, $\varphi_n(W_2)$, and $\varphi_n(W_2 \,|\, W_1)$, where the notation is adopted from the Cumulative Class Notes. The single script input is the $2 \times n$ array `fellowship_record` for $n > 1$ (note n is not a script input, and must be deduced from `fellowship_record`). The script outputs are four scalar variables: `phiW1W2`, which corresponds to $\varphi_n(W_1, W_2)$; `phiL1L2`, which corresponds to $\varphi_n(L_1, L_2)$; `phiW2`, which corresponds to $\varphi_n(W_2)$; and `phiW2barW1`, which corresponds to $\varphi_n(W_2 \,|\, W_1)$. Note that in the case in which no students win the first fellowship, your script should return `-1` for `phiW2barW1`.

   The script template for this question is provided in `A8Q1_Template`.

   We encourage you to test your script with inputs `fellowship_record` (synthetic historical data) of your own creation, say for $n = 10$ students, such that you can confirm the correct result by inspection.

2. (20 points) A multiple-choice quiz comprises three questions. A 3×2 array `quest_attributes` describes the questions: the three rows correspond to the three questions; the first column corresponds to the number of multiple-choice options (amongst which there is one, and only one, correct response); the second column corresponds to the number of points. For example,

a 3 in `quest_attributes(2,1)` indicates that in the second question there are three multiple choice options; a 20 in `question_attributes(3,2)` indicates that the third question is worth 20 points. Note that the entries in the second column of `quest_attributes` sum to 100.

We would like to analyze the anticipated performance of a guesser. You may assume that a guesser will choose any of the multiple-choice options for a particular question with equal likelihood; you may further assume that the guesser treats each question as independent (in the probabilistic sense). From this description you should be able to create an appropriate sample space — in terms of three Bernoulli variables which represent respectively "correct" and "incorrect" for the three questions — and associated joint (and marginal) probabilities.

We would like you to write a function which, given an array `quest_attributes`, calculates p_{allright}, the probability that a guesser answers all of the questions correctly, $p_{\text{atleastoneright}}$, the probability that a guesser answers at least one of the questions correctly, and finally $N_{\text{pointsearned}}$, the *expectation* of the total number of points (over all three questions) which will be earned by a guesser.

Your function should have "signature"

```
function [p_all,p_atleastone,N_points] = guesser(quest_attributes)
```

with a single function input and three function outputs. We emphasize that the function must be named `guesser` and furthermore must be stored in a file named `guesser.m`. The function takes a single function input: the 3×2 array `quest_attributes` described above; allowable instances are, for entries in the first column, non-negative integers, and for entries in the second column, non-negative entries which sum to 100. The function yields three function outputs, all scalar variables: `p_all` corresponds to p_{allright}; `p_atleastone` corresponds to $p_{\text{atleastoneright}}$; `N_points` corresponds to $N_{\text{pointsearned}}$.

The script template for this question is provided in `A8Q2_Template`; no modifications are required (except to remove the `_Template` from the name). The function template is provided in `guesser_Template.m`; note for function templates (as well as script templates) you should first "save as" with `_Template` removed from the name.

We encourage you to test your function with inputs `quest_attributes` of your own creation such that you can confirm the correct result "by hand." You might initially choose intuitive cases — say three questions each worth 100/3 points and each with two multiple-choice options — and then proceed to more complicated situations.

3. (20 points) A baseball player at the plate may be modeled as a random variable V which represents the base the player will attain as a result of a particular "up." The sample space for V is $\{0, 1, 2, 3, 4\}$, where a 0 corresponds to an "out" (by any means), a 1 to a single or a walk or a hit-by-pitch, a 2 to a double, a 3 to a triple, and a 4 to a home run. (We do not consider subtleties, for example related to stolen bases or sacrifices or errors or balks.) If you do not know anything about baseball, you may disregard this introduction and proceed to the abstraction below.

The probability mass function for the random variable V is given by

$$f_V(v) = \begin{cases} p_1 & v = 0 \\ p_2 & v = 1 \\ p_3 & v = 2 \\ p_4 & v = 3 \\ p_5 & v = 4 \end{cases} , \tag{1}$$

where as always the $0 \leq p_i \leq 1, 1 \leq i \leq 5$, and $\sum_{i=1}^{5} p_i = 1$.

We would like you to develop a function which generates a pseudo-random sample realization of V of size n based on the "continuous uniform to discrete" transformation described in the Cumulative Class Notes. (Note this sample realization can in turn be integrated into a line-up to simulate, very crudely, a baseball game.)

Your function should have "signature"

```
function [v_variates_vec] = base_attained(p_vec,n)
```

with two function inputs and a single function output. We emphasize that the function must be named `base_attained` and furthermore must be stored in a file named `base_attained.m`. The function takes two function inputs. The first function input is a 1×5 array `p_vec` for which `p_vec(i)` $= p_{\mathtt{i}}, \mathtt{i} = 1, \ldots, 5$, as defined above; allowable instances are any $p_i, 1 \leq i \leq 5$, which satisfy the usual probability requirements. The second function input is `n`, which corresponds to n, the size of the pseudo-random sample realization; the allowable instances are `n` $\in \{10, 11, \ldots, 100000\}$. The single function output is the $1 \times n$ array `v_variates_vec` which corresponds to the pseudo-random sample realization of V. Note that `v_variates_vec` must be random not just in terms of the outcome frequencies but also in terms of the order — such that (say) the first `n/2` elements of `v_variates_vec`, or the last `n/2` elements of `v_variates_vec`, or the "middle" `n/2` elements of `v_variates_vec`, should also yield (approximately) the correct outcome frequencies.

The script template for this question is provided in `A8Q3_Template`; no modifications are required (except to remove the `_Template` from the name). The function template is provided in `base_attained_Template.m`; note for function templates (as well as script templates) you should first "save as" with `_Template` removed from the name.

We encourage you to test your function for inputs `p_vec` of your own creation. You may find the MATLAB histogram feature

```
histogram(v_variates_vec,'Normalization','probability','BinMethod','integers')
```

useful for debugging purposes — do the frequencies look consistent with the probabilities, at least for large n? — but please do not include this display in the script or function you submit to Stellar.

4. (20 points) The (univariate) normal density is often a good and also convenient description of the outcome of a random phenomenon. However, in the case in which the outcome must be positive (on physical grounds, for example a spring constant), a normal random variable — which can in principle take on all values negative and positive — can create difficulties. In the case in which negative values are very rare, we can justifiably consider a "rectified" normal

random variable: a zero or negative value is rejected and we draw again from the desired normal population until we obtain a positive value. (Note this procedure creates a density which (is zero for negative values and) has the same *shape* as the normal density for positive values but is uniformly amplified to integrate to unity over the positive real numbers.)

We would like you to write a function which provides, based on the procedure described above, a pseudo-random sample of size n of rectified normal random variables associated with a normal population with mean `mu` and standard deviation `sigma` (note the mean and standard deviation are defined for the normal random variables *before* rectification).

Your function should have signature

```
function [xpts_pos] = positive_normal(mu,sigma,n)
```

with three function inputs and a single function output. We emphasize that the function must be named `positive_normal` and furthermore must be stored in a file named `positive_normal.m`. The function takes three function inputs. The first two inputs, respectively `mu` and `sigma`, are scalars; the allowable instances, or parameter domain, is $0.2 \leq$ `mu` ≤ 2.0 and $0.05 \leq$ `sigma` $\leq$ `2*mu`. The input `n` is a scalar integer; the allowable instances, or parameter domain, is `n` $\in \{10, 11, \ldots, 100000\}$. The function yields a single function output: the output is the $1 \times$ `n` row vector `xpts_pos`, which is our pseudo-random sample realization. Note that `xpts_pos` must be random not just in terms of the outcome frequencies but also in terms of the order — such that (say) the first `n/2` elements of `xpts_pos`, or the last `n/2` elements of `xpts_pos`, or the "middle" `n/2` elements of `xpts_pos`, should also yield (approximately) the correct outcome frequencies.

The script for this question is provided in `A8Q4_Template`; no modifications are required (except to remove the `_Template` from the name). The function template is provided in `positive_normal_Template.m`; note for function templates (as well as script templates) you should first "save as" with `_Template` removed from the name.

We encourage you to test your function for various μ, σ, and n. You may find the MATLAB built-in

```
histogram(xpts_pos,'Normalization','pdf');
```

useful for debugging purposes — are any negative variates present? do the frequencies look correct? — but please do not include this display in the script or function you submit to Stellar.

5. (20 points) A spring for a micro-robot suspension is required to have a spring constant k between $k_{\text{lower}} = 2000$ N/m and $k_{\text{upper}} = 2500$ N/m in order to provide the right balance between isolation (of the cargo) and control for navigation. The robot manufacturer receives a batch of springs from the spring supplier and proceeds to measure the spring stiffness of $n = 10000$ randomly chosen springs from the batch. It is found that, of these 10000 springs, 9851 of the springs do indeed have a spring constant within the desired range — between 2000 N/m and 2500 N/m; the remaining 149 springs have a spring constant outside the desired range.

   To model this situation we introduce a Bernoulli random variable B: a spring constant outside the desirable range — $K > k_{\text{upper}}$ or $K < k_{\text{lower}}$ — is encoded as $B = 0$ and occurs with probability $1 - \theta$; a spring constant inside the desirable range — $k_{\text{lower}} \leq K \leq k_{\text{upper}}$ —

is encoded as $B = 1$ and occurs with probability θ. Here K is the random variable which represents the spring constant and which in turn defines the Bernoulli random variable. We wish to determine the parameter θ from our sample of 10000 randomly chosen springs.

(i) (5 points) Based on the experimental data from the sample of $n = 10000$ springs, the sample-mean estimate for θ, $\hat{\theta}_n$, is

(a) 0.0149

(b) 9851

(c) 0.9870

(d) 0.9851

(ii) (10 points) Based on the experimental data from the sample of $n = 10000$ springs, the two-sided normal-approximation confidence interval for θ at confidence level $\gamma = 0.95$ (as provided in the Cumulative Class Notes[1]) is

(a) $[0.9800, 0.9940]$

(b) $[0.9825, 0.9873]$

(c) $[0.9701, 0.9850]$

(d) can not be evaluated as the normal-approximation criterion (see Cumulative Class Notes) is not satisfied

(iii) (5 points) A value for θ less than 0.97 requires the spring supplier to pay the robot manufacturer a penalty, whereas a value for θ greater than 0.99 requires the robot manufacturer to pay the spring supplier a premium. From your result of part (ii) can you conclude with confidence level 0.95 that $0.97 < \theta < 0.99$?

(a) Yes

(b) No

Note you may assume here that our random model for the spring constant K and hence Bernoulli variable B is valid (as only in this case can you make rigorous statistical inferences).

The template `A8Q5_Template.m` contains the multiple-choice format required by `grade_o_matic`.

[1] The formula in the Cumulative Class Notes is slightly more accurate than the formula in the textbook.