Assignment 2 — 2.086 Fall 2014

Due: *Tuesday, 14 October at 5 PM.*

Upload your solution to course website as a zip file "`YOURNAME_ASSIGNMENT_2`" which includes the script for each question *as well as* all MATLAB® functions (of your own creation) called by your scripts; both scripts and functions must conform to the formats described in **Instructions** and **Questions** below.

Instructions

Download (from the course website Assignment 2 page) the `Assignment_2_Templates` folder. This folder contains a template for the script associated with each question (`A2Qy_Template` for Question `y`), as well as a template for each function which we ask you to create (`func_Template` for a function `func`). The `Assignment_2_Template` folder also contains the `grade_o_matic` files (please see Assignment 1 for a description of `grade_o_matic`[1]) as well as all `.mat` files which you will need for Assignment 2.

We indicate here several general format and performance requirements:

(a.) Your script for Question `y` of Assignment `x` *must* be a proper MATLAB ".m" script file and *must* be named `AxQy.m`. In some cases the script will be trivial and you may submit the template "as is" — just remove the `_Template` — in your "`YOURNAME_ASSIGNMENT_2` folder. But note that you still must submit a proper `AxQy.m` script or `grade_o_matic` will not perform correctly.

(b.) In this assignment, for each question `y`, we will specify inputs and outputs both for the script `A2Qy` and (as is more traditional) any requested MATLAB functions; we shall denote the former as script inputs and script outputs and the latter as function inputs and function outputs. For each question and hence each script, and also each function, we will identify *allowable instances* for the inputs — the parameter values or "parameter domains" for which the codes must work.

(c.) Recall that for scripts, input variables must be assigned *outside* your script (of course before the script is executed) — *not* inside your script — in the workspace; all other variables required by the script must be defined *inside* the script. Hence you should test your scripts in the following fashion: `clear` the workspace; assign the input variables in the workspace; run your script. Note for MATLAB functions you need not take such precautions: all inputs and outputs are passed through the input and output argument lists; a function enjoys a private workspace.

(d.) We ask that in the submitted version of your scripts and functions you suppress all display by placing a "`;`" at the end of each line of code. (Of course during debugging you will often choose to display many intermediate and final results.) We also require that **before** you upload your solution to course website you run `grade_o_matic` (from your YOURNAME_ASSIGNMENT_2 folder) for final confirmation that all is in order.

[1]Note that, for display in verbose mode, `grade_o_matic` will "unroll" arrays and present as a row vector.

Note in Assignment 2, for each question, we encourage you to design your own code without reference to the template. You can start with the mathematical statement of the problem and, as appropriate, the numerical method for approximation or solution. You can then design the logic (or "flow") for your code: the reduction of your method to a sequence of steps — an algorithm. Finally, you should consider the particular MATLAB implementation: the capabilities of MATLAB which you will exploit, and the associated syntax. However, if you do need some guidance, the template — one particular approach amongst many — can get you started.

———————— · ————————

Questions

1. (10 points) Write a script which, given a 20×40 array `M`, performs the following operations (in sequence):

   (*i*) creates a new 20×40 array, `D`, which is all zeros except `D(i,i) = 1` for $1 \leq$ `i` ≤ 20 and `D(i,i+20) = 2` for $1 \leq$ `i` ≤ 20;

   (*ii*) creates a new 20×40 matrix `A` as the sum of the corresponding entries of arrays `M` and `D` — for example, `A(1,2) = M(1,2) + D(1,2)`;

   (*iii*) creates a new 20×40 array, `B`, which is the same as array `A` except row 11 for which `B(11,j) = 1/j`, for $1 \leq$ `j` ≤ 40;

   (*iv*) creates a new 20×41 array, `C`, which is the same as array `B` for columns 1 through 40 but also includes a column 41 in which all elements are assigned the value 3;

   (*v*) creates a new 20×41 array, `P`, which is the same as array `C` except the first ten entries on the main diagonal for which `P(i,i) = .1 * i * C(i,i)`, for $1 \leq$ `i` ≤ 10;

   (*vi*) creates a new 20×41 array, `Q`, which is the same as array `P` except the `(1,2)` entry for which `Q(1,2)` is assigned the value 7;

   (*vii*) creates a new 20×41 array, `R`, in which each element is the square of the corresponding element in array `Q` — for example, since `Q(1,2)` $= 7$ then `R(1,2)` $= 49$;

   (*viii*) creates a scalar `bigsum` which is the sum of all the elements (820 in total) of the array `R`.

   No functions are required for this question.

   The script takes a single script input: the input is a 20×40 array `M` and must correspond in your script to MATLAB variable `M`; the allowable instances, or input parameter domain, is given by `abs(M(i,j))` $\leq 10, 1 \leq$ `i,j` ≤ 10. The script yields a single script output: the output is the scalar `bigsum` and must correspond in your script to MATLAB (scalar) variable `bigsum`.

   The script template for this question is provided in `A2Q1_Template`.

2. (15 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.

You are provided with historical data in the form of a 2×171 logical array `fellowship_record` (in the file `fellowship_record.mat`): the two rows correspond to the two fellowships, and the 171 columns to the 171 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, the 0 in `fellowship_record(1,9)` indicates that student 7 did not win Fellowship 1; a 1 in `fellowship_record(2,11)` indicates that student 11 did win Fellowship 2.

We would like you to write a script which, on the basis of the `fellowship_record` data ($n = 171$), 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 nutshell *Introduction to Probability and Statistics.* There are no script inputs. 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 $\varphi(W_2 \,|\, W_1)$.

The script template for this question is provided in `A2Q2_Template`.

3. (10 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; your may further assume that the guesser treats each question as independent (in the probabilistic sense). You may thus describe the guesser procedure by an appropriate trio of independent Bernoulli random variables.

   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_Q3(quest_attributes)
```

   with a single function input and three function outputs. We emphasize that the function must be named `guesser_Q3` and furthermore must be stored in a file named `guesser_Q3.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 A2Q3_Template; no modifications are required (except to remove the _Template from the name). The function template is provided in guesser_Q3_Template.m; note for function templates (as well as script templates) you should first "save as" with _Template removed from the name.

4. (15 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 $0 \le p_i \le 1, 1 \le i \le 5$, 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 nutshell *Random Variables.* (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_Q4(p_vec,n)
```

   with a single function input and a single function output. We emphasize that the function must be named base_attained_Q4 and furthermore must be stored in a file named base_attained_Q4.m. The function takes two function inputs. The first function input is a 1×5 array p_vec for which p_vec(i) $= p_{\mathtt{i}}$, i $= 1, \ldots, 5$, as defined above; allowable instances are any $p_i, 1 \le i \le 5$, 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 A2Q4_Template; no modifications are required (except to remove the _Template from the name). The function template is provided in base_attained_Q4_Template.m; note for function templates (as well as script templates) you should first "save as" with _Template removed from the name.

5. (15 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 $0.2 \le$ `mu` ≤ 2.0 and $0.05 \le$ `sigma` $\le$ `2*mu`. The input `n` is a scalar integer; the allowable instances, or parameter domain, is `n` $\in \{10, 11, \dots, 100000\}$. 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 `A2Q5_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.

You may find the MATLAB built-in `hist` useful for debugging purposes, but please do not include this display in your script or function (rather, apply `hist` to `xpts_pos` directly from the command window).

6. (10 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 — $k_{\text{lower}} \le K \le k_{\text{upper}}$ — 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) (4 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) (4 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 $\gamma = 0.95$ 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 nutshell *Random Variables*) is not satisfied

(iii) (2 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 `A2Q6_Template.m` contains the multiple-choice format required by `grade_o_matic`.

7. (15 points) We would like you to write a function which computes, based on the method described in nutshell *Monte Carlo Integration*, a Monte Carlo estimate $(\hat{A}_D)_n$, and associated 95% confidence-level (normal-approximation two-sided) confidence interval $[\mathrm{ci}_{A_D}]_n$, for the area of the region

$$D = \{x_1^2 + x_2^2 \leq 0.75\} \cup \{(x_1 - \alpha)^2 + x_2^2 \leq 0.75\} \ . \tag{2}$$

Note that D is the union of two circles in which the center of the second circle is shifted by α (in x_1) from the center of the first circle. We choose for our background rectangle R the square $[-c, c]^2$ for c sufficiently large to enclose D.

Your function should have signature

```
function [area_est,area_conf_int] = MC_area(alpha,c,n,x1pts,x2pts)
```

with five function inputs and two function outputs. We emphasize that your function must be named `MC_area` and furthermore must be stored in a file named `MC_area.m`. The function takes five function inputs. The first input, scalar `alpha`, is the center shift α; the allowable instances, or parameter domain, is $0 \leq$ `alpha` ≤ 0.1. The second input, the scalar `c` ($= c$ above), defines the bounding square R: the lower left corner of R is $(x_1, x_2) =$ `(-c,-c)` and the upper right corner of R is $(x_1, x_2) =$ `(c,c)`; the allowable instances are $1 \leq$ `c` ≤ 10. (Note for these instances of `c` the domain D is indeed included in R for all allowable instances of `alpha`.) The third input is scalar integer `n`, the number of random darts; the allowable instances, or parameter domain, is `n` $\in \{10, 11, \ldots, 100000\}$. The fourth and fifth inputs are the coordinates of the `n` random darts, `(x1pts(i),x2pts(i))`, `i` $\in \{1, 2, \ldots$ `n`$\}$, thrown at the square R — on the basis of which you will calculate your area estimate and associated confidence interval: `x1pts` and `x2pts` are each $1\times$ `n` arrays of i.i.d. pseudo-random variates from the (univariate) continuous uniform distribution over the interval $-c \leq x_1 \leq c$; there are no restrictions on allowable instances (as the number of darts is already controlled by `n`). The function yields two function outputs: the first output is a scalar `area_est` which is the Monte-Carlo estimate for the area, $(\hat{A}_D)_n$; the second output is the 1×2 row vector `area_conf_int` such that `area_conf_int(1)` and `area_conf_int(2)` are respectively the lower and upper limits of the confidence interval $[\text{ci}_{A_D}]_n$.

Three further points: First, in the event that the normal-approximation criterion for the construction of the confidence interval is not satisfied *your code should return* `[-1, 1]` for `area_conf_int`. Second, you should set $z_\gamma = 1.96$ which corresponds to confidence level $\gamma = 0.95$. Third, in order to test your function `MC_area` for any desired (`alpha` and) `c` and `n` you must first generate the random darts `x1pts,x2pts` *outside* your function. Note, however, that for purposes of grading, `grade_o_matic` will automatically generate appropriate inputs `x1pts,x2pts` for your function — no action needed on your part.

The script for this question is provided in `A2Q7_Template`; no modifications are required (except to remove the `_Template` from the name). The function template is provided in `MC_area_Template.m`; note for function templates (as well as script templates) you should first "save as" with `_Template` removed from the name.

8 (10 points) We would like you to write a function which computes, based on the sample-mean method described in nutshell *Monte Carlo Integration*, a Monte Carlo estimate $\hat{I}_n$ for the integral

$$I = \int_0^\pi \sin^2(100x)\, dx \ . \qquad (3)$$

Note that the interval of integration is $(a \equiv 0, b \equiv \pi)$. It is no secret that $I = \pi/2$.

Your function should have signature

```
function [int_est] = sample_mean_integral_Q8(n,upts)
```

which takes two function inputs and yields one function output. We emphasize that your function must be named `sample_mean_integral_Q8` and furthermore must be stored in a file named `sample_mean_integral_Q8.m`. The function takes two function inputs. This first input is scalar integer `n`, the size of the random sample `upts`; the allowable instances, or parameter domain, is `n` $\in \{10, 11, \ldots, 100000\}$. The second input is the $1\times$ `n` array `upts` of i.i.d. pseudo-random variates from the (univariate) continuous uniform distribution over the interval $(0, 1)$ — the sample realization on the basis of which, through transformations, you will compute $\hat{I}_n$; there are no restrictions on allowable instances (apart from the restriction on `n`). The function yields a single function output: the scalar `int_est`, which is the sample-mean estimate for I, $\hat{I}_n$.

One further point: In order to test your function `int_est` you must first generate the random sample `upts` *outside* your function. Note, however, that for purposes of grading, `grade_o_matic` will automatically generate appropriate inputs `upts` for your function — no action needed on your part. Attention: `upts` is drawn from the continuous uniform distribution over the interval $(0, 1)$; inside your function you must transform these pseudo-random variates to the correct interval as required in the integral (3).

The script for this question is provided in `A2Q8_Template`; no modifications are required (except to remove the `_Template` from the name). The function template is provided in `sample_mean_integral_Q8_Template.m`; note for function templates (as well as script templates) you should first "save as" with `_Template` removed from the name.