Mini Project 2: Computing π (6 pages)

Rule 1. Submit per Handout 2 guidelines, at least `mp2_ABC_WXYZ.m`, where `ABC` is the section number and `WXYZ` the last 4 digits of your NJIT ID. Observe capitalization. Use underscore _ not a dash -.
Use semicolons to SUPPRESS output. We instruct you explicitly otherwise.

Rule 2. Send an email (with subject line that includes the three words `mp2 ABC WXYZ`, separated with one space, as applied to you)
(a) to the instructor `alexg+cs101@njit.edu`,
(b) to the grader/TA `tt72+cs101@njit.edu`, and
(c) to yourself, to make sure that everything went fine with the transmission of your mail.

Rule 3. It is imperative that you fully conform to Rules 1 and 2. If you deviate from these rules points will get deducted. Observe variable names and capitalization.

Due Date: Before **noon time of Tue Dec 2, 2014**.

1 Part 0: Primary function (15 points)

You will create a text-based M-file named (Rule 1) `mp2_ABC_WXYZ.m` where `ABC` and `WXYZ` are as specified in Rule 1. **Always use semicolons to suppress output**.

1. First 4 lines (7 points). The first and fourth line of this M-file are going to be empty MATLAB comment lines. The second line will contain in the form of a MATLAB comment the words `MP2` (capitalized), your section number, and the last four digits of your id. The third line will contain in the form of a MATLAB comment, your last name (capitalized), and then your first or other given names. Any number of 1,2,3 spaces may be used to delimit any two of the items described.

2. (Primary) Function (8 points). The fifth line would contain the definition of the primary function of this file. It returns no values and has only one parameter `NN`. You should be able to figure out what the name of this function should be. From this point one we refer to the lines of the primary function in step 3,4,5,8, 11 below in Parts A,B,C to follow. The 8 points assigned to this item 2 are shared among these other items that describe primary function code. Thus "first line" below in item 3 would refer to the first line of the primary function after its definition: this can be written as the sixth line of the file. Empty lines that you use won't be counted; it's up to you whether you want to use them or not.

3. First line of (primary) function. It uses an fprintf statement to print your last name capitalized in a box of width 20 right justified, followed by a single space, and then one (and only one) of your other given names in a box of width 15 right justified, followed by a single space, and then the last four digits of your NJIT ID in a box of width 7 as a decimal integer, left-justified. The fprintf statement should move the cursor/prompt to the next line after it does what it is supposed to do. If I were to print my info according to this specification it would appear on screen as the last line below (the comment line is not output and is there for your convenience to assist you in counting spaces etc.)

```
%234567890123456789012345678901234567890123456789012345678901234567890
       GERBESSIOTIS            Alex 1234
```

2 Part A: $S_1(n)$ and π (20 points)

4. Second line of (primary) function. A MATLAB subfunction `mp2_part_a` is to be called with argument 100; it accepts only one argument and returns no values. It is described in items 6 and 7 below. This function computes an approximation of π (not MATLAB's `pi`) by computing a finite number of terms n of an infinite sum that converges to $\pi^2/6$. Sum $S_1(n)$ is given below. The limit $\lim_{n\to\infty} S_1(n)$ is $\pi^2/6$.

$$S_1(n) = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{n^2} = \sum_{k=1}^{n} \frac{1}{k^2} \approx \frac{\pi^2}{6}$$

Subfunction `mp2_part_a` would only compute the first n terms of the sum, i.e. $S_1(n)$ and will thus compute an approximation to π of order n. The actions of the subfunction are described below in items 6 and 7 to follow.

5. Third, Fourth lines of (primary) function. MATLAB subfunction `mp2_part_a` is to be called with arguments 1000 and 10000 respectively, that is three times altogether (item 4 included).

6. Subfunction `mp2_part_a`. This subfunction is defined inside the M-file as necessary. It has exactly one parameter that is called `n` (note the case) and would return no values to the calling program. It works by performing a sequence of side effects.

7. Subfunction code. Use a variable `v1` to compute and store the result $S_1(n)$. The computation of $S_1(n)$ will be realized using array operations and neither for loops nor while loops. For this generate in variables named `v1a, v1b, ...` the following

7.1 A vector $1, 2, \ldots, n$ into `v1a`,

7.2 then square the elements of `v1a` to obtain $1^2, 2^2, \ldots, n^2$, and

7.3 then invert those elements to obtain the terms of $S_1(n)$, followed

7.4 by a sum operation to obtain $S_1(n)$ that will then be stored into `v1`.

As soon as variable `v1` holds $S_1(n)$, use an fprintf statement to print n and through `v1` an approximation to `pi`. The argument value of parameter n would be printed as a decimal integer of length 8 left-justified, starting from the left margin of the available output line (make sure that previous fprintf statements moved to a new line), followed by a single space, followed by the approximation of `pi` as computed through variable `v1`. The latter approximation would be printed as a floating point number of total width 9 and with 4 decimal digits, right justified. If i were to print $n = 50$ as specified and 3.123456 with 5 and not 4 decimal digits the result might look like. (Beware of the difference between this output and the stated for part 7.)

```
%234567890123456789012345678901234567890123456789
50         3.12346
```

Subsequently the subfunction terminates its execution. (Again note that `v1` as soon as the execution of 7.4 completes does not hold an approximation to `pi` but of $S_1(n)$. It's your responsibility to compute an approximation of `pi` from that value. Minimize the number of additional variables used! The fewer the better.)

3 Part B: More to come (20 points)

In item 5 above we had stopped after the fourth line of the primary function. We go on adding code to that primary function.

8. Lines 5,6,7 of (primary) function. These three lines would call a second subfunction named `mp2_part_b` with argument values 100, 1000, 10000 respectively, one call per line used. This subfunction computes $S_2(n)$, where

$$S_2 = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \ldots + \frac{1}{n^4} = \sum_{k=1}^{n} \frac{1}{k^4} \approx \frac{\pi^4}{90}$$

is an approximation to $pi^4/90$ i.e. the limit $\lim_{n\to\infty} S_2(n)$ is $\pi^4/90$.

9. Subfunction `mp2_part_b`. This subfunction is defined inside the M-file as necessary and after the complete definition of the previous subfunction. It has exactly one parameter that is called `n` and would return no values to the calling program. It works by performing a sequence of side effects.

10. Subfunction code. Use a variable `v1` to compute and store the result $S_2(n)$. The computation of $S_2(n)$ will be realized using array operations and neither for loops nor while loops. See step 7 before for something similar for $S_1(n)$. Towards this generate

10.1 a vector $1, 2, \ldots, n$ into `v1a`,

10.2 then quadruple the elements of `v1a` to obtain $1^4, 2^4, \ldots, n^4$, and

10.3 then invert those elements to obtain the terms of $S_2(n)$, followed

10.4 by a sum operation to obtain $S_2(n)$ that will then be stored into `v1`.

As soon as variable `v1` holds $S_2(n)$, use an fprintf statement to print n and through `v1` an approximation to `pi`. The argument value of parameter n would be printed as a decimal integer of length 8 left-justified, starting from the left margin of the available output line (make sure that previous fprintf statements moved to a new line), followed by a single space, followed by the approximation of `pi` as computed through variable `v1`. The latter approximation would be printed as a floating point number of total width 9 and with 4 decimal digits, right justified. Subsequently the subfunction terminates its execution.

4 Part C: Monte carlo approximation to π (30 points)

In Figure 1 we have a circle C with center coordinates $(1/2, 1/2)$ of radius $1/2$ inscribed into a square S of side 1. The area of the square A_s is $A_s = 1 \times 1 = 1$ and the area A_c of the circle is $A_c = \pi \times (1/2)^2 = \pi/4$. The ratio of the area of the circle over the area of the square is $A_c/A_s = (\pi/4)/1 = \pi/4$; then $4A_c/A_s = \pi$. From now on we call this fraction $4A_c/A_s$ the `magic number` M i.e.

$$M = \frac{4A_c}{A_s} = \pi.$$

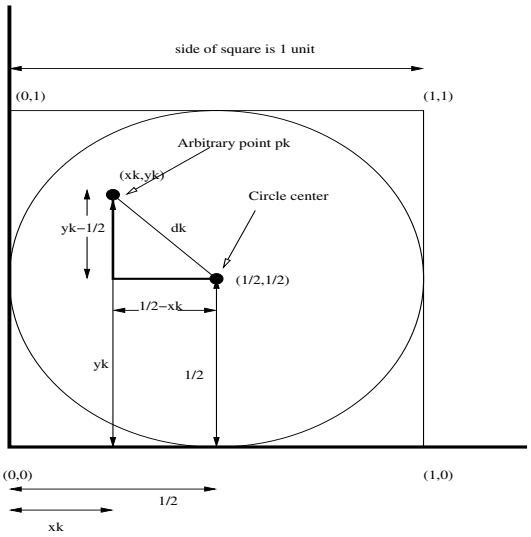

Figure 1: Inscribed circle (of radius 1/2) of a 1×1 square

A **Monte-Carlo** computation of π involves generating $N_s = N$ random points within square S and then counting how many of them fall within the inscribed circle C. The argument is that the number of points N_c within the circle, and the number of points N inside the square will be proportional to the corresponding areas A_c and A_s. Thus $N_c/N_s = N_c/N \approx A_c/A_s = \pi/4$ and thus $4N_c/N \approx \pi$. Thus estimating π is reduced to the problem of (generating) points within square S and then counting points within circle C. We can describe this process in terms of the following steps.

Step 1. We generate N points p_k, $1 \leq k \leq N$, randomly so that they lie within S. Each point $p_k = (x_k, y_k)$ will have coordinates x_k, y_k such that $0 \leq x_k, y_k \leq 1$ since the square is of side 1 and its end points in the coordinate system are $(0, 0), (0, 1), (1, 1), (1, 0)$.

Step 2. For every such random square point p_k that will be generated, we then check whether it also lies within circle C. If the point lies on or is inside the circle we account for it separately using a counter C that counts how many points lie on or inside the circle, among the points of the square. Thus the value of C is the value of N_c and $C = N_c \leq N$.

Step 3. After all N points are generated and inspected we obtain the ratio $4 * C/N$. This becomes an estimate of the magic number M i.e. of π.

I. How to check whether p_k lies in/on circle $(x = 1/2, y = 1/2, r = 1/2)$.

Say we have generated a point within the square (with the help of MATLAB) and let that point be $p_k = (x_k, y_k)$. We then check whether that point lies within the circle. We can check whether this is so by

considering the distance between p_k and the center of the circle that has coordinates $(x = 1/2, y = 1/2)$. If this distance is $1/2$, then the point lies on the circle. If the distance is less than $1/2$ it lies inside, and if it is greater it lies outside the circle. Note that p_k by default (we started the paragraph with a **Say**) is a square point and thus it lies within S in all three cases. The distance check is easy to perform. Let d_k be the distance of p_k from the center of the circle. Then d_k is given by

$$d_k = \sqrt{(x_k - 1/2)^2 + (y_k - 1/2)^2}$$

Thus depending on whether d_k is less than or equal to $1/2$, we increment C as needed.

II. Estimating π from C and N is easy: $4 * C/N$ is an estimate of M and π!

III. MATLAB logistics: random numbers and other things.

Function `rand` generates random numbers in MATLAB. In fact a single call to `rand` or `rand()` returns a single "random" real number in the interval $(0, 1)$. Thus calling rand twice, once for x_k and once for y_k, one could potentially generate a random point of the unit-square S.

In item 8 of Part B we had stopped after the seventh line of the primary function described in this mini-project. We go on adding more lines to this primary function.

11. Lines 8,9, 10 of (primary) function. These three lines would call a third subfunction named `mp2_part_c` with argument values 10000, 1000000, `NN` respectively, one call per line used. This subfunction computes an N approximation to M and to π itself. (Note that now we use N and not n for this part. If the appearance of an `NN` confuses you, it has to do with primary function; read items 1-3 more carefully.)

12. Subfunction `mp2_part_c`. This subfunction is defined inside the M-file as necessary and after the previous subfunctions. It has exactly one parameter that is called `N` and would return no values to the calling program. It works by performing a sequence of side effects.

13. Subfunction code. You are free to use array operations, for or while loops and any additional variables you may wish to use. But make sure that your code works and is not slow!! The input parameter should be named N, and two auxiliary variables inside it C, M would work as specified earlier. Minimally, your code will

13.1 generate N points with coordinates (x_k, y_k) that fall inside the unit square, then

13.2 it would count into C the number of these points that fall into or onto the unit circle, and

13.3 then compute the value of M to provides an N-approximation to π.

As soon as variable `M` holds its value, use an fprintf statement to print the relevant information as before. The argument value of parameter N would be printed as a decimal integer of length 8 left-justified, starting from the left margin of the available output line (make sure that previous fprintf statements moved to a new line), followed by a single space, followed by the approximation of `pi` as computed through variable `M`. The latter approximation would be printed as a floating point number of total width 9 and with 4 decimal digits, right justified. Subsequently the subfunction terminates its execution.

14. Line 11 of (primary) function. It becomes the last line to terminate the primary function. This is the end of it and of this mini project.

5 Part D: Grading

Of the 85 points of the assigment

- 9,9,15 points are assigned to the correct formulation of the fprintf statements, in Parts A, B, and C respectively,
- of the extra 10 points of Part C (30 points instead of 20 points for Part A and Part B) all of them are assigned to an implementation that is efficient and is able to handle gracefully problem sizes of `NN= 10000000` or more (line 10 of primary function in item 11 above).

Note 1. If you do not suppress the output, or your code print something else that is not specified in the 4 fprintf statements of this assignment you will be losing 10pts for every piece of unspecified output. Likewise for variable, parameter names as explicitly specified.

Note 2. In order to call the primary function you need to bind its parameter to an argument value. You might choose one of the 10000 and 1000000 used in lines 8,9. If you feel confident with your implementation use the value for `NN` that the grader will be using to determine whether you get the 10 extra points of Part C: 10 milliion i.e. `NN=10000000`.