

Rules. Follow Handout 2 guidelines if you plan to submit more than one files and for other details. For a complete submission, two of the submitted files must be named `montepi_WXYZ.m` and `montetest_WXYZ`. We use an underscore `_` rather than a dash `-` because MATLAB will get confused otherwise. It is assumed in this example that the last four digits of your id are `WXYZ`; replace accordingly otherwise. Other information for submission is available in Handout 2 (eg. how to form an email).

Due Date: Before noon of April 14, 2011 per Handout 2 guidelines. (Note that this is one week later than the Handout 1 quoted due date.)

1 Mini Project 2: Monte Carlo-based computation of π

In Figure 1 we have a circle of radius $1/2$ inscribed into a square of side 1. The area of the square A_s is naturally $A_s = 1 \times 1 = 1$ whereas the area A_r of the circle is $A_r = \pi \times (1/2)^2 = \pi/4$. Therefore the ratio of the area of the circle over the area of the square is $A_r/A_s = (\pi/4)/1 = \pi/4$. From now on we call this fraction A_r/A_s the **magic number M** i.e.

$$M = \frac{A_r}{A_s} = \pi/4.$$

We then generate N points p_k , $1 \leq k \leq N$, randomly so that they lie within the square. 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 $(0, 0), (0, 1), (1, 1), (1, 0)$. For every such random square point p_k that will be generated, we will then check whether it also lies within the circle. 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 $C \leq N$. After all N points are inspected, we would be interested in figuring out what fraction of the N points lie in the circle. The availability of counter C makes it easy: the answer is C/N .

This ratio C/N should be close to (if not identical) to the magic number M since the chances that a square point is inside (or on) the circle is proportional to the area of the circle relative to the area of the square in which it is inscribed.

Checking whether p_k lies within/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 is a square point and thus it lies within the square 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 appropriately.

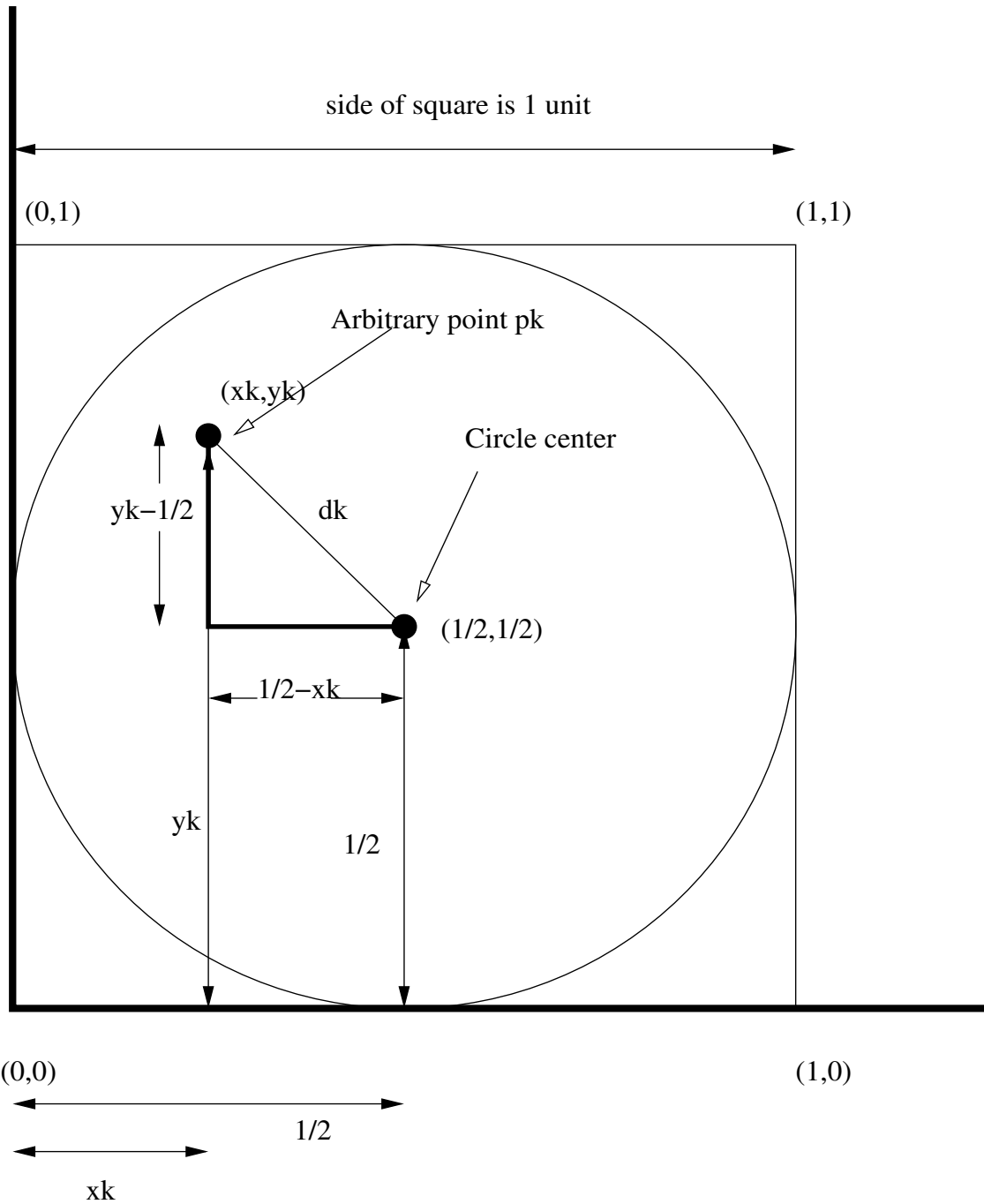


Figure 1: Inscribed circle (of radius $1/2$) of a 1×1 square

Estimating π from C and N .

Because the points p_1, p_2, \dots, p_N are randomly drawn we expect them to be uniformly distributed over the square area. Thus the fraction C/N of points lying within (or on) the circle would approximate the fraction of the area of the circle over the area of the square, i.e. the magic number M . But we know that $M = \pi/4$, thus C/N would approximate $M = \pi/4$ or equivalently by solving for π that $4C/N$ would approximate π .

In our experiment, we know N , we can compute C , and thus determine $4C/N$. This will determine our approximation to π which will be $4C/N$.

Another way to view this problem is by considering throwing darts at a target. In that case though, we count (leniently) as a success the fact that we just managed to hit the round area of the square target, which is not that difficult and less challenging than say hitting the bull's eye of the target!

2 MATLAB Logistics: Generating random numbers

Function `rand` generates random points in MATLAB. Its syntax is similar to `ones` except that it can fill a scalar or a matrix with random values rather than ones. For information and proper use, type `help rand` 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 of Figure 1. Not much else is needed other than perhaps using the `sqrt` function. It is imperative that you are careful with your implementation of the otherwise simple code. One reason for that is that the benchmarking that will be done and described in section 3 might be affected by your implementation choices. If you do not implement things properly you might risk of running out of memory, or output might get delayed (i.e. you wait in vain without getting answers)!

3 MATLAB implementation requirements

Part 1 (Function `montepi_WXYZ`).

You will implement a MATLAB function (in an M-file) appropriately named `montepi_WXYZ`, where `WXYZ` are the last four digits of your NJIT ID. Function `montepi_WXYZ` will have one parameter denoted by `N`. Thus it will look like `montepi_WXYZ(N)`. Parameter `N` is the number of points that will be generated in the call to `montepi_WXYZ(N)`. A counter variable C will be properly initialized. It will then generate N points p_k with coordinates x_k and y_k that are random using MATLAB function `rand`, and compute an approximation to π as previously explained using a loop structure (choice of a `for` or a `while` loop is yours to make); there is a way to implement this part using array operations only. At the end an estimate of π will be computed and printed as explained below. The output of function `montepi_WXYZ` will print three values: N , C , and the estimate of π as computed in this N point Monte-Carlo simulation. The output **should look like** as follows. Spacing is important: two spaces precede `N= C=` and `pi_estimate`, and the `=` follows the previous letter without any spacing.

```
MontePi:  N= 1234567890  C=  123456789  pi_estimate= 3.14159
```

Things to note for the output. The output value of N should accommodate exactly 11 digits. And so will C . The estimate for π should print 5 decimal digits, the decimal point and exactly one (of course) digit left of the decimal. Note that in the example above C was a 9-digit number. Two spaces appear after the equal sign of `C=` to accommodate the 11-digit requirement. (Also observe what happened to `N`, `pi_estimate`.

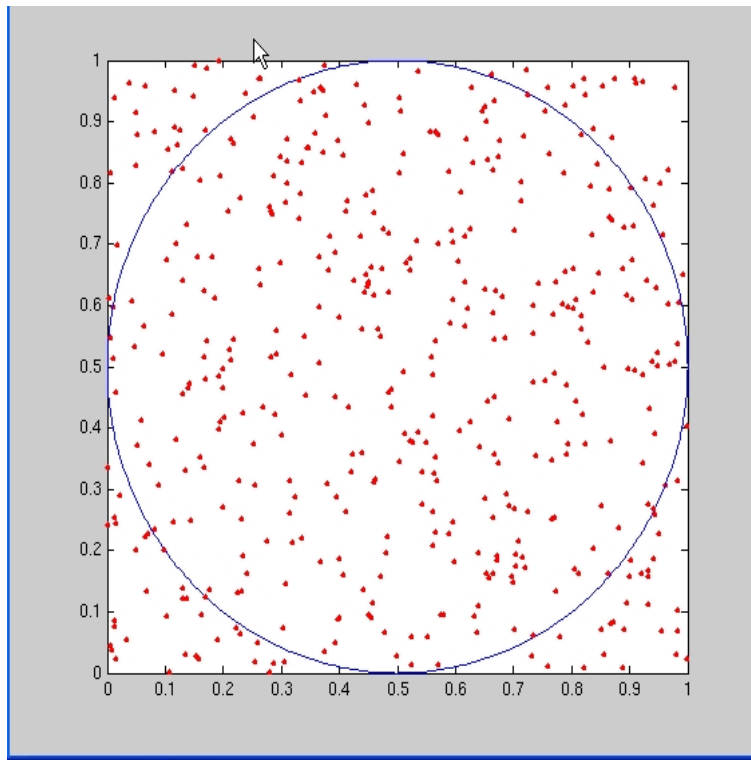


Figure 2: Plotting the problem's points

In addition, function `montepi_WXYZ` would also return two values which will be named `C` and `pi_es` in this specific order i.e. `C` is the first value returned and `pi_es` the second. The former is the C value and the latter is the π estimate computed and also printed in the code of `montepi_WXYZ`.

Part 2 (Plotting of points in `montepi_WXYZ`).

Within the implementation outlined in Part 1, embed code that allows for plotting of the points generated under certain conditions imposed on N . If the number of points N is determined to be LESS than 300, then plot a square of size 1, draw a circle inscribed in that square and plot the random points as tiny bullet points in red ink. If N is greater than or equal to 300, no plotting is required. The code that plots the circle and the square has already been given to you; it is in file `circle.m` of the supplied code in the protected area of the course web-page (usually residing in subdirectory Sub4). You may use it freely. The end result is that a plot similar to the one shown in Figure 2 should result. It's up to you however that you make the Part 2 code compatible with the Part 1 code; no adverse interaction should result.

Part 3 (Benchmarking with `montetest_WXYZ`).

You can test run your code as needed until you are satisfied with its performance and formatting output. However, we also ask you to run your code on specific problem sizes $N = 100$, $N = 10,000$, $N = 1,000,000$, $N = 100,000,000$.

Create for that a testing function `montetest_WXYZ` (in an M-file) where this test will take place. This new function will minimally use the return values of 4 function calls to generate a similar and somewhat duplicated output to the one generated by `montepi_WXYZ`.

The output that `montetest_WXYZ` will generate will be a tabulated summary of the four test runs. The first line of the output will have labels for N , C and π i.e. will look like the first line in the sample output below. The 4 lines that tabulate the results of the four calls, will print the value of N right-justified as shown in the remaining lines in the first column. The second column will print C and the last one the

estimate of π as obtained from C and N . The spacing you need to maintain might not be obvious from the first 5 lines. For that we added for your convenience one extra **phony line** (that you should not try to replicate or print) for a phony test run with $N = 12345678901$.

The output of `montetest_WXYZ` SHOULD not interfere with the output of `montepi_WXYZ`. This means that output of `montetest_WXYZ` will follow the output of `montepi_WXYZ` and in tabulated format will (partially) look like the first 5 lines below.

N	C	pi
100		
10000		
1000000		
100000000		
12345678901	9696265345	3.14159

Remark. It is possible, if you are not very careful with the proper use of `rand` or if you precompute the random coordinates, that you will run out of space in MATLAB on an AFS machine. This will be taken into consideration in grading this problem as discussed next. Therefore be very very careful in dealing with MATLAB arrays/vectors and possible memory restrictions. The last run $N = 100,000,000$ might take several minutes on an AFS machine, so be patient. To figure out how much time you spend executing a program, you can use the MATLAB function `cputime` or the `tic`, `toc` pair. Type in MATLAB `help cputime` for further assistance.

4 Grading

Of the 120 points,

- the correct implementation of `montepi_WXYZ` will earn you upto 15 points,
- the correct implementation of `montetest_WXYZ` will earn you upto 15 points,
- the proper formatting of the output generated by `montepi_WXYZ` will earn you upto 15 points,
- the proper formatting of the output generated by `montetest_WXYZ` will earn you upto 15 points,
- the correct plotting, as explained in Section 3(part 2), will earn you upto 15 points,
- an efficient implementation that generates no memory exceptions will earn you upto 15 points,
- and the overall correctness and soundness of your submission will earn you the remaining 30 points.