Table 1: (Mean) Squared Error in the dart-throwing setting

# of draws (or quadrature points)	Psuedo-MC (Mean Squared Error)	Quasi-MC	Newton-Cortes
100	0.0234	0.0200	1.0e-05 *0.1458
1,000	0.0027	0.0011	1.0e-05 *0.0007
10,000	0.0003	0.0000	1.0e-05 *0.0000

1: The First Problem

I drew 100^2 points from the 2-dimensional Halton sequence, and counted the ratio of the points whose squared Euclidean norm is weakly less than 1. The estimated π is 3.1448.

2: The Second Problem

I have 100^2 quadrature points in $[0,1] \times [0,1]$, and use a Newton-Cortes method to get an approximation of π, which is 3.1016. The weights are $1/N$, where N denotes the number of quadrature points.

3: The Third Problem

Now I use the implicit function $y = \sqrt{1-x^2}$ for the upper-right part of the unit circle. I have 100^2 points in $[0,1]$ from a Halton sequence, and approximate π. The estimate is 3.1422.

4: The Fourth Problem

I have 10,000 quadrature points in $[0,1]$, and use a Newton-Cortes method. The weights are again $1/N$, where N denotes the number of quadrature points. This time I use the implicit function to get an approximation of π. The estimate is 3.1414.

5: The Fifth Question

(1) First we compare the results from a psuedo-MC with those from a Newton-Cortes and a quasi-MC in the two-dimensional, dart-throwing setting. Table 1 compares them. A Newton-Cortes gets the most accurate estimates. But it's not a fair comparison because a Newton-Cortes has 100, 1,000 or 10,000 quadrature points, which means actually $100^2, 1,000^2, 10,000^2$ draws in $[0,1] \times [0,1]$. But the others have 100, 1,000 or 10,000 draws there. So I should say a quasi-MC does a fairly good job for its unfairly small number of draws. The second column shows the mean squared errors from a psuedo-MC, and the third and fourth column show the squared columns from a quasi-MC and a Newton-Cortes. For a psuedo-MC, I get 200 simulations and calculate the mean squared error.

(2) Next we compare the results in the one-dimensional setting using the implicit function. Table 2 shows the result. Again, in this setting that favors a Newton-Cortes and a psuedo-MC, A quasi-MC makes a comparable performance, although the other two get more accurate estimates.

6: Code

```
% Motoaki Takahashi
% HW4 for Econ 512 Empirical Method
```

Table 2: (Mean) Squared Error in the implicit function setting

# of draws (quadrature points)	Psuedo-MC	Quasi-MC	Newton-Cortes
100	0.0090	1.0e-04 *0.1989	1.0e-06 *0.1185
1,000	0.0007	1.0e-04 *0.0015	1.0e-06 *0.0001
10,000	0.0001	1.0e-04 *0.0000	1.0e-06 *0.0000

```
clear
diary hw4.out
%% Question 1

% I draw 100^2 points in the unit square from Halton sequence.
n = 100^2;
h = haltonseq(n, 2);
hsq = h.^2;
hsq = sum(hsq, 2);
hsq = hsq(hsq<=1);
pi1=4*length(hsq)/n
clear h hsq

%% Question 2
% weights are 1/100^2 where 100^2 is the # of draws (points)
x = transpose(0.01:0.01:1); % 100 by 1 vector running from 0.01 to 1
y = transpose(0.01:0.01:1); % 100 by 1 vector running from 0.01 to 1
grid =[kron(x, ones(100,1)), kron(ones(100,1), y)];
grid = grid.^2;
grid = sum(grid, 2);
grid = grid(grid<=1);
pi2 = 4*length(grid)/(100^2)
clear grid

%% Question 3

h = haltonseq(n, 1);
h = sqrt(1-h.^2);
pi3 = 4*sum(h)/n

%% Question 4

grid = 0.0001:0.0001:1; % 10000 vector
grid = sqrt(1-grid.^2);
pi4 = 4*sum(grid)/length(grid)
clear grid

%% Question 5

% We have two methods: (1) two-dimensional random draws (2) one-dimensional
% implicit function

% (1) two dimendional

% quasi-MC
% simulated pi's from 100, 1000, 10000 draws from a quasi-MC method
% Here I use the built-in function haltonset to get a Halton sequence.
% https://www.mathworks.com/help/stats/generating-quasi-random-numbers.html
% Following the above web page, get a sequnce that skips the first 1000 values of the Halton sequence and then retains every 101st point
p = haltonset(2,'Skip',1e3,'Leap',1e2);
% Apply reverse-radix scrambling
p = scramble(p,'RR2');
quasi_100 = sim_pi(p(1:100,:))
quasi_1000 = sim_pi(p(1:1000,:))
quasi_10000 = sim_pi(p(1:10000,:))
se_quasi=([quasi_100; quasi_1000; quasi_10000]-pi*ones(3,1)).^2

% Newton-Coates
x = (0.005:0.01:0.995).'; % 100-vector
y = (0.005:0.01:0.995).'; % 100-vector
grid =[kron(x, ones(100,1)), kron(ones(100,1), y)];
nc_100 = sim_pi(grid)

x = (0.0005:0.001:0.9995).'; % 1000-vector
y = (0.0005:0.001:0.9995).'; % 1000-vector
grid =[kron(x, ones(1000,1)), kron(ones(1000,1), y)];
nc_1000 = sim_pi(grid)

x = (0.00005:0.0001:0.99995).'; % 10000-vector
y = (0.00005:0.0001:0.99995).'; % 10000-vector
grid =[kron(x, ones(10000,1)), kron(ones(10000,1), y)];
nc_10000 = sim_pi(grid)
se_nc=([nc_100; nc_1000; nc_10000]-pi*ones(3,1)).^2

% psuedo-MC
k = 100;
sim_pis = ones(200,1);

for i=1:200
    h = rand(k,2);
    sim_pis(i,1) = sim_pi(h);
end

mse100 = mean((sim_pis-pi*ones(200,1)).^2);

k = 1000;
sim_pis = ones(200,1);

for i=1:200
    h = rand(k,2);
    sim_pis(i,1) = sim_pi(h);
end
```

```
mse1000 = mean((sim_pis-pi*ones(200,1)).^2);

k = 10000;
sim_pis = ones(200,1);

for i=1:200
    h = rand(k,2);
    sim_pis(i,1) = sim_pi(h);
end

mse10000 = mean((sim_pis-pi*ones(200,1)).^2);

mse_psuedo = [mse100; mse1000; mse10000]
clear k i

% (2) one-dimensional, implicit function
% quasi-MC
clear p
p = haltonset(1,'Skip',1e3,'Leap',1e2);
p = scramble(p,'RR2');

quasi_oned_pi = [sim_pi2(p(1:100)); sim_pi2(p(101:1100)); sim_pi2(p(1101:11100))]
se_quasi_oned = (quasi_oned_pi-pi*ones(3,1)).^2

% Newton-Cortes
nc_oned_pi = [sim_pi2(0.005:0.01:0.995); sim_pi2(0.0005:0.001:0.9995); sim_pi2(0.00005:0.0001:0.99995)]
se_nc_oned = (nc_oned_pi-pi*ones(3,1)).^2

% psuedo-MC
k = 100;
pi_oned = 6*ones(200,1);
for i=1:200
    h = rand(k,1);
    pi_oned(i) = sim_pi2(h);
end
mse100_2 = mean((pi_oned-pi*ones(200,1)).^2)

k = 1000;
pi_oned = 6*ones(200,1);
for i=1:200
    h = rand(k,1);
    pi_oned(i) = sim_pi2(h);
end
mse1000_2 = mean((pi_oned-pi*ones(200,1)).^2)

k = 10000;
pi_oned = 6*ones(200,1);
for i=1:200
    h = rand(k,1);
    pi_oned(i) = sim_pi2(h);
end
mse10000_2 = mean((pi_oned-pi*ones(200,1)).^2)

mse_psuedo_2 = [mse100_2; mse1000_2; mse10000_2]

diary off
```

7: Output

```
pi1 =

    3.1448


pi2 =

    3.1016


pi3 =

    3.1422


pi4 =

    3.1414


quasi_100 =
```

```
     3


quasi_1000 =

    3.1080


quasi_10000 =

    3.1408


se_quasi =

    0.0200
    0.0011
    0.0000


nc_100 =

    3.1428


nc_1000 =

    3.1417


nc_10000 =

    3.1416


se_nc =

   1.0e-05 *

    0.1458
    0.0007
    0.0000


mse_psuedo =

    0.0234
    0.0027
    0.0003


quasi_oned_pi =

    3.1371
    3.1412
    3.1416
```

```
se_quasi_oned =

   1.0e-04 *

    0.1989
    0.0015
    0.0000


nc_oned_pi =

    3.1419
    3.1416
    3.1416


se_nc_oned =

   1.0e-06 *

    0.1185
    0.0001
    0.0000


mse100_2 =

    0.0090


mse1000_2 =

   7.2973e-04


mse10000_2 =

   7.7822e-05


mse_psuedo_2 =

    0.0090
    0.0007
    0.0001
```