Homework 4

Kensuke Suzuki

October 20, 2018

Problem 1: Dart-throwing method using Quasi-Monte Carlo

First define the function `pi_ind.m` which is the indicator function $\mathbb{1}\{x^2 + y^2 \leq 1\}$.

```
function val = pi_ind(x,y)

ind = x.^2 + y.^2;
val = zeros(length(x),1);

for sim = 1:length(x)
if ind(sim,1) <= 1
    val(sim,1) = 1;
else
    val(sim,1) = 0;
end
end
```

Using `rand()` command, I draw 10,000 two-dimensional random numbers between 0 and 1. Specify the seed for random number generator. Using the quasi-Monte Carlo approach, I approximate the integration as:

$$\int_0^1 \int_0^1 \mathbb{1}\{x^2 + y^2 \leq 1\} dy dx \approx \frac{(1-0)(1-0)}{10000} \sum_{j=1}^{10000} \mathbb{1}\{x_j^2 + y_j^2 \leq 1\} \tag{1}$$

where x_j and y_j is the sequence of random draws. Our result is:

$$\pi \approx 3.1676$$

Problem 2: Dart-throwing method using Newton-Cotes

We use the function `pi_ind.m` and employ the Newton-Cotes approach. First define the width of partition h as

$$h = \frac{1-0}{10000}$$

where 10,000 is same as the number of draws we used in problem 1. Using this partition, we creates the sequences of x and y as

$$x_j = 0 + (j - 1/2) \times h$$
$$y_j = 0 + (j - 1/2) \times h$$

for $j = 1, ..., 10000$. I first integrate over y for given x_k as follows:

$$f(x_k) = \int_0^1 \mathbb{1}\{x_k^2 + y^2 \leq 1\} dy \approx h \times \sum_{j=1}^{10000} \mathbb{1}\{x_k^2 + y_j^2 \leq 1\} \equiv \tilde{f}(x_k) \tag{2}$$

Then we integrate over x as:

$$\int_0^1 f(x_k) dx_k \approx h \times \sum_{k=1}^{10000} \tilde{f}(x_k) \tag{3}$$

The result is:

$$\pi \approx 3.1416$$

Problem 3: Pythagorean formula with Quasi-Monte Carlo

Define the function `pi_root.m` which is returns $\sqrt{1-x^2}$. Now we only need to draw random numbers for x. We again use `rand()` command to draw 1. Numerical integration is given by

$$\int_0^1 \sqrt{1-x^2} dx \approx \frac{1-0}{10000} \sum_{j=1}^{10000} \sqrt{1-x_j^2} \tag{4}$$

The result is

$$\pi = 3.1485$$

Problem 4: Pythagorean formula with Newton-Cotes

We use the same width of partition h as in the problem 2 and create the sequence of x analogously. Numerical integration is given by

$$\int_0^1 \sqrt{1-x^2}dx \approx h \times \sum_{j=1}^{1000} \sqrt{1-x_j^2}$$

The result is

$$\pi = 3.1416$$

Problem 5: Comparison

I computed the mean squared errors for the quasi-Monte Carlo method for each number of random draws (1000, 10000, and 100000 times) and the correspondence squared error for the Newton-Cotes method. Errors are computed by subtracting the approximation values (obtained using each method) from the real π value (computed using matlab `pi` command).

	1,000	10,000	100,000
Quasi-Monte Carlo (MSE)	1.0e-03 $\times$ 0.1711	1.0e-03 $\times$ 0.0016	1.0e-03 $\times$ 0.0000
Newton-Cotes (Sq'd error)	1.0e-09 $\times$ 0.1186	1.0e-09 $\times$ 0.0001	1.0e-09 $\times$ 0.0000

Errors are, in general, smaller for Newton-Cotes method. If we make the number of draws (nodes) for 100,000, then the (mean) squared errors are 0 for both methods.

Matlab Code

```
% Empirical method HW4
% Kensuke Suzuki
% Penn State
% October 20

clear all
delete HW4log.txt
diary('HW4log.txt')
diary on

disp('ECON512 HOMEWORK4: Ken Suzuki')

%% Questtion 1: Quasi-Monte Carlo method

% Number of random draw
numsim = 10000;
```

```

seed = 1534561;
rng(seed);

% Define function: pi_ind
% returns 1 if x^2 + y^2 <= 1 and 0 otherwise

% Generate random sequence for x and y using rand
seq = rand(numsim,2);
x = seq(:,1);
y = seq(:,2);

% We now compute the sequence of values of indicator function using the
% random sequence generated above
pi_QMC_Q1 = pi_ind(x,y);

% Compute numerical integration
display('Problem 1: Quasi-Monte Carlo method')
pi_Q1 = ((1-0)*(1-0))/numsim * 4 * sum(pi_QMC_Q1)

clear x y
%% Questtion 2: Newton-Cotes approach
% Here I use the midpoint rule to compute the integration

% define the width of partition h
h = (1-0)/numsim;

% Define vector of x and y (later filled)
x = zeros(numsim,1);
y = zeros(numsim,1);

% x_j = 0 + (j-1/2)h for j=1,...,numsim
for ind = 1:numsim
    x(ind,1) = 0 + (ind- 1/2)*h;
    y(ind,1) = 0 + (ind- 1/2)*h;
end

% Compute the sequence of values of indicator function for given x_j
% and sum over with weight h, which yields the approximation of integration
% over y (for iven x_j)
pi_NC_Q2 = ones(numsim,1);
for ind = 1:numsim
    x_1 = x(ind,1)*ones(numsim,1);
    pi_NC_x = pi_ind(x_1,y);
    pi_NC_Q2(ind,1) =  h * sum(pi_NC_x);
end

```

```
% Next we integrate over x by summing over with weight h
display('Problem 2: Newton-Cotes approach')
pi_Q2 = 4 * h * sum(pi_NC_Q2)


%% Questtion 3: Newton-Cotes approach: another functional form
% I use Halton sequence to generate random draws

% Define function: pi_root
% returns (1-x^2)^(1/2)

seed = 1534561;
rng(seed);
% Generate random sequence for x
x = rand(numsim,1);

% We now compute the sequence of values of indicator function using the
% random sequence generated above
pi_QMC_Q3 = pi_root(x);

% Compute numerical integration
display('Problem 3: Pythagorean fomula with Quasi-Monte Carlo method')
pi_Q3 = ((1-0))/numsim * 4 * sum(pi_QMC_Q3)

%% Questtion 4: Newton-Cotes approach: another functional form
% Again I use the midpoint rule to compute the integration

% define the width of partition h
h = (1-0)/numsim;

% Define vector of x and y (later filled)
x = zeros(numsim,1);

% x_j = 0 + (j-1/2)h for j=1,...,numsim
for ind = 1:numsim
    x(ind,1) = 0 + (ind- 1/2)*h;
end

% Compute the sequence of values of the function pi_root and
    approximate
% the integration
display('Problem 3: Pythagorean fomula with Newton-Cotes method')
pi_NC_Q4 =  4 * h * sum(pi_root(x))


%% question 5:

numsim_list= [1000,10000,100000];
```

```
realpi = pi;



% Implement numerical integration using QMC with different number of
    draws
% We simulate 200 times and compute the squared error
ErrQMC_200 = ones(200,3);
for i = 1:length(numsim_list)
    numsim = numsim_list(1,i);
    seed = 1534561;
    for sim = 1:200
        seed = seed + sim ;
        rng(seed);
        % comute squared residual for QMC
        x = rand(numsim,1);
        pi_QMC = pi_root(x);
        ErrQMC_200(sim,i) = (realpi - ((1-0))/numsim * 4 * sum(pi_QMC))
            ^2;
    end
    clear x
end
% Mean squaed error is obtained as
MErrQMC = sum(ErrQMC_200)./numsim_list;


% We then use Newton-Coates
ErrNC = ones(1,3);
% Implement numerical integration using NC and compute the squared
    error
for i = 1:length(numsim_list)
    numsim = numsim_list(1,i);
    h = (1-0)/numsim;
    x = zeros(numsim,1);
    for ind = 1:numsim
        x(ind,1) = 0 + (ind- 1/2)*h;
    end
    NC = 4* h * sum(pi_root(x));
    ErrNC(1,i) = (realpi - (4* h * sum(pi_root(x))))^2;
    clear x

end
% Mean squaed error is obtained as
ErrNC;

display('Problem 5: Comparison')
display('Monte-Carlo mean squared error (1000, 10000, and 100000 draws)
    ')
```

```
MErrQMC
display('Newton–Cotes squared error (1000, 10000, and 100000 nodes)')
ErrNC

diary off
```