MATH 211.3 Winter Term 2024 Assignment

Assignment #10

Name: Kanwar Pannu

Student Number: 11336732

NSID: eza147

**Problem 1**

**A close-up of a paper with mathematical equations

Description automatically generated**

**A math equations on a piece of paper

Description automatically generated**

**3**

clear;

clc;

deltaP = @(x) (-x.^2 + x + 1/2) - (x.^2 - x + 1/2);

area\_estimates = zeros(1, 5);

for i = 2:6

n = 10^i;

x\_values = rand(n, 1);

area\_estimates(i-1) = mean(deltaP(x\_values));

end

disp('Area estimates for n = 10^i (i from 2 to 6):');

disp(area\_estimates);

deltaP = @(x) (-x.^2 + x + 1/2) - (x.^2 - x + 1/2);

area\_estimates = zeros(1, 5);

for i = 2:6

n = 10^i;

x\_values = rand(n, 1);

area\_estimates(i-1) = mean(deltaP(x\_values));

end

disp('Area estimates for n = 10^i (i from 2 to 6):');

disp(area\_estimates);

**A screenshot of a computer

Description automatically generated**

**5**

clear;

clc;

ellipseMonteCarlo(1e4);

ellipseMonteCarlo(1e6);

function ellipseMonteCarlo(n)

ellipseA = @(x, y) 13\*x.^2 + 34\*x.\*y + 25\*y.^2 <= 1;

ellipseB = @(x, y) 40\*x.^2 + 25\*y.^2 + y + 9/4 <= 52\*x.\*y + 14\*x;

xA = 2\*rand(n, 1) - 1;

yA = 2\*rand(n, 1) - 1;

insideA = ellipseA(xA, yA);

areaEstimateA = 4\*mean(insideA);

xB = rand(n, 1);

yB = rand(n, 1);

insideB = ellipseB(xB, yB);

areaEstimateB = mean(insideB);

correctAreaA = pi/6;

correctAreaB = pi/18;

errorA = abs(areaEstimateA - correctAreaA);

errorB = abs(areaEstimateB - correctAreaB);

fprintf('Estimate for ellipse (a) with n = %d: %f, Error: %f\n', n, areaEstimateA, errorA);

fprintf('Estimate for ellipse (b) with n = %d: %f, Error: %f\n', n, areaEstimateB, errorB);

end

**A number and numbers on a white background

Description automatically generated**

**7**

clear;

clc;

integralResult = integral2(@(x,y) x.\*y, 0, 1, 0, @(x) x.^2);

fprintf('Calculus-based integral evaluation: %f\n', integralResult);

n = 1e6;

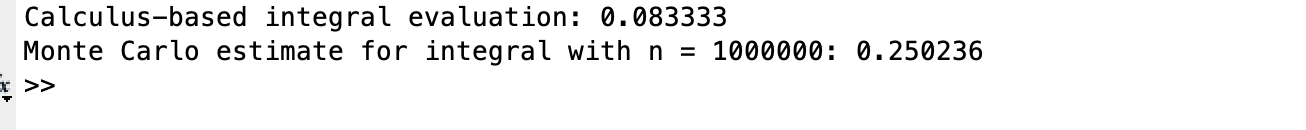
x = rand(n, 1);

y = rand(n, 1);

inDomain = y <= x.^2;

monteCarloEstimate = mean(x(inDomain) .\* y(inDomain));

fprintf('Monte Carlo estimate for integral with n = %d: %f\n', n, monteCarloEstimate);

****

**9**

clear;

clc;

m = 2^48;

a = 2^24 + 3;

b = 0;

u = zeros(1, 10002);

u(1) = 1;

for i = 1:(length(u) - 1)

u(i+1) = mod(a\*u(i) + b, m);

end

u = u / m;

figure;

subplot(1, 2, 1);

plot3(u(1:end-2), u(2:end-1), u(3:end), '.');

title('Questionable RNG - 3D plot');

xlabel('u\_i');

ylabel('u\_{i+1}');

zlabel('u\_{i+2}');

axis([0 1 0 1 0 1]);

grid on;

check = u(3:end) == mod(6\*u(2:end-1) - 9\*u(1:end-2), 1);

if all(check)

disp('The relationship holds for all generated values.');

else

disp('The relationship does not hold for some generated values.');

end

**A screenshot of a computer

Description automatically generated**

**Problem 2**

**1**

clear;

clc;

P1 = @(x) x.^2 - x + 0.5;

P2 = @(x) -x.^2 + x + 0.5;

between\_parabolas = @(x, y) (y > P1(x)) & (y < P2(x));

for k = 2:5

n = 10^k;

index = 1:n;

x = halton\_seq(index, 2);

y = halton\_seq(index, 3);

area\_estimate\_type1 = mean(P2(x) - P1(x));

area\_estimate\_type2 = mean(between\_parabolas(x, y));

fprintf('For n = 10^%d:\n', k);

fprintf('Type 1 Monte Carlo Estimate: %f\n', area\_estimate\_type1);

fprintf('Type 2 Monte Carlo Estimate: %f\n', area\_estimate\_type2);

end

function hseq = halton\_seq(index, base)

hseq = zeros(size(index));

for i = 1:length(index)

n = index(i);

f = 1;

r = 0;

while n > 0

f = f / base;

n = floor(n);

r = r + f \* mod(n, base);

n = n / base;

end

hseq(i) = r;

end

end

**A screenshot of a computer

Description automatically generated**

**3**

clear;

clc;

estimateEllipseArea(1e4);

estimateEllipseArea(1e5);

function estimateEllipseArea(n)

ellipseA = @(x, y) (13\*x.^2 + 34\*x.\*y + 25\*y.^2) <= 1;

ellipseB = @(x, y) (40\*x.^2 + 25\*y.^2 + y + 9/4) <= (52\*x.\*y + 14\*x);

xA = 2 \* rand(n, 1) - 1;

yA = 2 \* rand(n, 1) - 1;

insideA = ellipseA(xA, yA);

areaEstimateA = 4 \* mean(insideA);

xB = rand(n, 1);

yB = rand(n, 1);

insideB = ellipseB(xB, yB);

areaEstimateB = mean(insideB);

correctAreaA = pi / 6;

correctAreaB = pi / 18;

errorA = abs(areaEstimateA - correctAreaA);

errorB = abs(areaEstimateB - correctAreaB);

fprintf('n = %d\n', n);

fprintf('Estimated area of ellipse (a): %f, Error: %f\n', areaEstimateA, errorA);

fprintf('Estimated area of ellipse (b): %f, Error: %f\n', areaEstimateB, errorB);

end

**A computer error message

Description automatically generated with medium confidence**

**5**

clear;

clc;

exact\_volume = pi^2 / 2;

n = 1e5;

monte\_carlo\_volume = ball\_volume(4, 1, n);

mc\_error = abs(monte\_carlo\_volume - exact\_volume);

fprintf('Monte Carlo approximation: %f\n', monte\_carlo\_volume);

fprintf('Error in Monte Carlo approximation: %f\n', mc\_error);

hseq\_dim1 = halton\_seq(1:n, 2);

hseq\_dim2 = halton\_seq(1:n, 3);

hseq\_dim3 = halton\_seq(1:n, 5);

hseq\_dim4 = halton\_seq(1:n, 7);

qmc\_count = 0;

for i = 1:n

point = [hseq\_dim1(i), hseq\_dim2(i), hseq\_dim3(i), hseq\_dim4(i)] \* 2 - 1;

if norm(point) <= 1

qmc\_count = qmc\_count + 1;

end

end

qmc\_volume = (2\*1)^4 \* (qmc\_count / n);

qmc\_error = abs(qmc\_volume - exact\_volume);

fprintf('Quasi-Monte Carlo approximation: %f\n', qmc\_volume);

fprintf('Error in Quasi-Monte Carlo approximation: %f\n', qmc\_error);

function v = ball\_volume(dim, r, n)

count = 0;

for i = 1:n

point = rand(1, dim) \* (2\*r) - r;

if norm(point) <= r

count = count + 1;

end

end

v = (2\*r)^dim \* (count / n);

end

function hseq = halton\_seq(index, base)

hseq = zeros(size(index));

for i = 1:length(index)

n = index(i);

f = 1;

r = 0;

while n > 0

f = f / base;

n = floor(n);

r = r + f \* mod(n, base);

n = n / base;

end

hseq(i) = r;

end

end

**A close-up of a number

Description automatically generated**

**Problem 3**

**1**

clear;

clc;

n = 10000;

intervals = {[-2, 5], [-5, 3], [-8, 3]};

for i = 1:length(intervals)

interval = intervals{i};

b = abs(interval(1));

a = interval(2);

[probability\_estimate, exact\_probability, error] = random\_walk\_probability(a, b, n);

fprintf('Interval [-%d, %d]:\n', b, a);

fprintf('Monte Carlo Estimate: %f\n', probability\_estimate);

fprintf('Exact Probability: %f\n', exact\_probability);

fprintf('Error: %f\n\n', error);

end

function [probability\_estimate, exact\_probability, error] = random\_walk\_probability(a, b, n)

hits = 0;

for i = 1:n

position = 0;

while position > -b && position < a

step = randi([0 1])\*2 - 1;

position = position + step;

end

if position == a

hits = hits + 1;

end

end

probability\_estimate = hits / n;

exact\_probability = b / (a + b);

error = abs(probability\_estimate - exact\_probability);

end

**A screenshot of a computer

Description automatically generated**

**3**

clear;

clc;

p = 0.7;

n = 10000;

intervals = {[-2, 5], [-5, 3], [-8, 3]};

for i = 1:length(intervals)

interval = intervals{i};

b = abs(interval(1));

a = interval(2);

[mc\_probability, exact\_probability, error] = biased\_random\_walk\_probability(a, b, p, n);

fprintf('Interval [-%d, %d] with p = %.2f:\n', b, a, p);

fprintf('Monte Carlo Probability: %f\n', mc\_probability);

fprintf('Exact Probability: %f\n', exact\_probability);

fprintf('Error: %f\n\n', error);

end

function [mc\_probability, exact\_probability, error] = biased\_random\_walk\_probability(a, b, p, n)

q = 1 - p;

hits = 0;

for i = 1:n

position = 0;

while position > -b && position < a

if rand() < p

position = position + 1;

else

position = position - 1;

end

end

if position == a

hits = hits + 1;

end

end

mc\_probability = hits / n;

if p ~= q

exact\_probability = ((q/p)^b - 1) / ((q/p)^(a+b) - 1);

else

exact\_probability = a / (a + b);

end

error = abs(mc\_probability - exact\_probability);

end

**A screenshot of a computer

Description automatically generated**

**5b**

clear;

clc;

a = pi;

b = 2;

dt = 0.01;

n = 1000;

[mc\_probability, exact\_probability, error] = brownian\_motion\_escape(a, b, dt, n);

fprintf('Monte Carlo Probability of escape through the top: %f\n', mc\_probability);

fprintf('Exact Probability: %f\n', exact\_probability);

fprintf('Error: %f\n', error);

function [mc\_probability, exact\_probability, error] = brownian\_motion\_escape(a, b, dt, n)

escapes = 0;

for i = 1:n

position = 0;

while position > -b && position < a

step = sqrt(dt) \* randn();

position = position + step;

end

if position >= a

escapes = escapes + 1;

end

end

mc\_probability = escapes / n;

exact\_probability = b / (a + b);

error = abs(mc\_probability - exact\_probability);

end

**A white background with black text

Description automatically generated**