Solution of Week 4 Lab (Prepared by Yuan Yin)

December 20, 2019

1 Confidence interval

a.

[1]:
```
%%file Ex1PartA.m
function [probDoubleSix, math_prob, left_terminal, right_terminal] = Ex1PartA(numReps)

    numRolls = 24;
    numDoubleSixes = 0;
    math_prob = 1 - (35 / 36) ^ (numRolls);

    for run = 1: numReps

        roll1 = randi(6,numRolls,1); % the random expt
        roll2 = randi(6,numRolls,1);

        if any((roll1 + roll2) == 12) % add together and check with 12
            numDoubleSixes = numDoubleSixes + 1; % the quantity of interest
        end

    end

    probDoubleSix = numDoubleSixes / numReps; % the frequency of a 6

    % fprintf('\nThe absolute difference between the simulation result and the exact answer is %6.4f\n', abs(math_prob - probDoubleSix));

    % Compute the 95% confidence interval:

    half_width = 1.96 * (probDoubleSix * (1 - probDoubleSix) / numReps) ^ (1 / 2);
    left_terminal = probDoubleSix - half_width;
    right_terminal = probDoubleSix + half_width;

    % fprintf('\nThe 95%% confidence interval is (%6.4f, %6.4f).\n', left_terminal, right_terminal);
```

```
end
```

```
Created file '/Users/hailongguo/lib/Dropbox/teaching/UoM/2020/MAST30028/jupyter/
week4/Ex1PartA.m'.
```

[2]:
```
[probDoubleSix, math_prob, left_terminal, right_terminal] = Ex1PartA(10000)
```

```
probDoubleSix =

    0.4907


math_prob =

    0.4914


left_terminal =

    0.4809


right_terminal =

    0.5005
```


b.

[3]:
```
%%file Ex1PartB.m
function EX1PartB

clc

num_simulation = 100;
numReps = 1000;
numFail = 0;

for i = 1 : num_simulation

    [probDoubleSix, math_prob, left_terminal, right_terminal] = Ex1PartA(numReps);

    if ((math_prob < left_terminal) || (math_prob > right_terminal))
        numFail = numFail + 1;
    end
```

```
end

fprintf('Through running the simulation %d times, there are %d times the␣
↪confidence interval fail to contain the exact answer.\n', num_simulation,␣
↪numFail);

end
```

```
Created file '/Users/hailongguo/lib/Dropbox/teaching/UoM/2020/MAST30028/jupyter/
week4/Ex1PartB.m'.
```

[4]:
```
Ex1PartB
```

```
Through running the simulation 100 times, there are 2 times the confidence
interval fail to contain the exact answer.
```

c.

[5]:
```
%%file Ex1PartC.m
function [probSix, L, R] = Ex1PartC(numReps)

numRolls = 4;
numSixes = 0;

difference_array = ones(1, numReps); % initialise the array

for run = 1 : numReps

    roll = randi(6,numRolls,1); % the random expt

    if any(roll == 6)
        numSixes = numSixes + 1; % the quantity of interest
    end

    largest = max(roll);
    smallest = min(roll);
    difference_array(run) = largest - smallest;

end

 probSix = numSixes/numReps; % the frequency of a 6
% fprintf('Prob of a 6 is %6.4f\n',probSix)


% Calculate the 95% interval:

mean_value = mean(difference_array);
```

```
standard_err = std(difference_array);

half_width = 1.96 * standard_err / sqrt(numReps);
L = mean_value - half_width;
R = mean_value + half_width;

fprintf('The 95%% confidence interval is: (%6.4f, %6.4f).\n', L, R);

end
```

```
Created file '/Users/hailongguo/lib/Dropbox/teaching/UoM/2020/MAST30028/jupyter/
week4/Ex1PartC.m'.
```

[6]:
```
[probSix, L, R] = Ex1PartC(10000)
```

```
The 95% confidence interval is: (3.4739, 3.5201).

probSix =

    0.5183
```



```
L =

    3.4739


R =

    3.5201
```

2 Monte Carlo Integration

a.

[7]:
```
integrand = @(x) exp( - (x.^ 2)./ 2);
upper_terminal = 1;
lower_terminal = 0;

integral_value = integral(integrand, lower_terminal, upper_terminal);

true_value = 0.5 + (1 / sqrt(2 * pi)) * integral_value;
disp(true_value)
```

```
    0.8413
```

b.

[8]:
```
%%file Crude_MC.m
function resultMC = Crude_MC(a, b, num_points, func)

xi = a + (b - a).*rand(num_points,1);

resultMC = ((1 / num_points) * sum(func(xi))) * (b - a);

end
```

```
Created file '/Users/hailongguo/lib/Dropbox/teaching/UoM/2020/MAST30028/jupyter/
week4/Crude_MC.m'.
```

[9]:
```
% Crude Monte Carlo Method:
n = 10;
max_num_points = 10^8;

while n <= max_num_points

    integral_MC = Crude_MC(lower_terminal, upper_terminal, n, integrand);
    calc_value = 0.5 + (1 / sqrt(2 * pi)) * integral_MC;

    if abs(calc_value - true_value) < 0.001
        fprintf('After %d iterations, we get the 2 decimal place accuracy.\n',␣
 ↪n);
        fprintf('The true value is %6.4f\n', true_value);
        fprintf('The calculated value is %6.4f\n', calc_value)
        break
    else
        n = n + 10;
    end
end
```

```
After 10 iterations, we get the 2 decimal place accuracy.
The true value is 0.8413
The calculated value is 0.8410
```

c.

[10]:
```
sum(randn(20, 1) < 1)
```

```
ans =

    15
```

3 Floating point numbers

(a). The point of this exercise is to show you that machine number is not uniformly distributed, and if you switch to 'log' scale, you can see that now, the spaces between each pair of floating point numbers are the same.

(b).

[11]:
```
format short e;
x = 1; k = 0;
while x ~= 0
    x=x/2; k=k+1;
end
x
k
```

```
x =

     0


k =

        1075
```


Explanation: The smallest non-zero number representable is 2^{-1074}, which is a denormalised number. Any number smaller than this gives underflow and will be stored as 0. Therefore, when $k = 1075, x = 2^{-1075} (< 2^{-1074})$, we break the while loop and the computer stores x as 0.

[12]:
```
format short e;
x=1;k=0;
while isfinite(x)
    x=x*2;k=k+1;
end
x
k
```

```
x =

   Inf


k =

        1024
```

Explanation: Using the largest exponent $e = 1023$ and the largest $1+f = 1.11...1_2$, we get the largest number representable, realmax, in MATLAB. Any number larger than this causes overflow and is stored as infinity in MATLAB. Therefore, when $k = 1024(> 1023)$, $x = 2^k = 2^{1024}$ (>realmax), we break the while loop and the computer stores x as Inf.

[13]:
```
format short e;
x=1;k=0;
while 1+x ~= 1
   x=x/2;k=k+1;
end
x
k
```

```
x =

   1.1102e-16


k =

    53
```


Explanation: From the codes, we see that when the computer stores $x+1$ as 1, we exit the while loop. Now let's investigate the mechanism under $x+1$ more deeply. When adding two numbers in MATLAB, the computer will first raise them to the same power, and then perform the addition on their fractional parts. If we end up getting a non-floating point number during the addition, the computer will round it to the nearest floating point number. So, $x+1 = 2^0(x+1) = 2^0(1) = 1$ iff $x \leq \frac{\varepsilon_M}{2}(= 2^{-53})$. Thus, when $k = 53$ and $x = 2^{-53}$, we exit the while loop.

c.

[14]:
```
x=realmax; x=x+1
x=realmax; x=2*x
x=x/2
```

```
x =

  1.7977e+308


x =

   Inf


x =
```

```
Inf
```

Explanation:: * $x+1 = realmax+1 = 2^{1023}(1.11....1_2+2_{10}^{-1023}) = 2^{1023}(1.11....1_2)$ (since$2_{10}^{-1023} << \varepsilon_M$) $= realmax = 1.7977e+308$;

- $2 \times x = 2 \times realmax > realmax$. Hence, $2 \times x$ will cause overflow and the computer will store it as Inf.
- Since Inf propagates in the subsequent calculations, $x \div 2 = \text{Inf} \div 2 = \text{Inf}$.

[15]:
```
x=realmin;x=x/2
```

```
x =

  1.1125e-308
```

Explanation:: Since realmin is only the smallest normalized machine number and the smallest non-zero number representable is 2^{-1074}, realmin $\div 2 = 2^{-1022} \div 2 = 2^{-1023} = 1.1125e-308 (> 2^{-1074})$.

[16]:
```
format short e
x = 1+eps
x = x-1
x = 1+eps/2
x = x-1
x=8+eps
x=8+4*eps
x=8+5*eps
```

```
x =

   1.0000e+00


x =

   2.2204e-16


x =

     1


x =
```

```
     0


x =

     8


x =

     8


x =

   8.0000e+00
```

Explanation::

- $1+\varepsilon_M = 2^0(1+\varepsilon_M) = 1.00000000000000022204......$, which is already a floating point number;
- $x - 1 = 1 + \varepsilon_M - 1 = \varepsilon_M = 2.2204e - 16$
- $1 + \frac{\varepsilon_M}{2} = 2^0(1 + \frac{\varepsilon_M}{2}) = 2^0(1) = 1$ (since $\frac{\varepsilon_M}{2}$ is smaller than ε_M, so it will be stored as its nearest floating point number, 0);
- $x - 1 = 1 - 1 = 0$;
- $8 + \varepsilon_M = 2^3(1 + \varepsilon_M \times 2^{-3}) = 2^3(1) = 8$ (since $\varepsilon_M \times 2^{-3}$ is smaller than ε_M, so it will be stored as its nearest floating point number, 0);
- By using the same argument as above, $8 + 4 \times \varepsilon_M = 2^3(1 + \frac{\varepsilon_M}{2}) = 2^3(1) = 8$.
- $8 + 5 \times \varepsilon_M = 2^3 \times (1 + \frac{5}{8} \times \varepsilon_M)$. However, $(1 + \frac{5}{8} \times \varepsilon_M)$ is not a floating point number. We need to round this to the nearest machine number, $(1 + \varepsilon_M)$. Therefore, the result is $2^3 \times (1 + \varepsilon_M) = 8.000000000000001776......$