SOEN 6011
SOFTWARE ENGINEERING PROCESSES

DELIVERABLE 1

ETERNITY : FUNCTIONS

SUBMITTED BY:
Prabhpreet Singh

July 20, 2019

Contents

List of Figures

1 Problem 1

1.1 Function arccos(x)

The arccosine(x) function is the inverse of cosine of x. It returns the angle at which the cosine is x. It is called by the abbreviated form acos(x). $\arccos(x) = \cos^{-1}(x)$

1.2 Domain and Co-Domain of arcos(x)

$$y = \arccos(x) \tag{1}$$

Domain: $-1 \leq x \leq 1$
Range: $0 \leq y \leq \pi$

1.3 Graph of arccos(x)

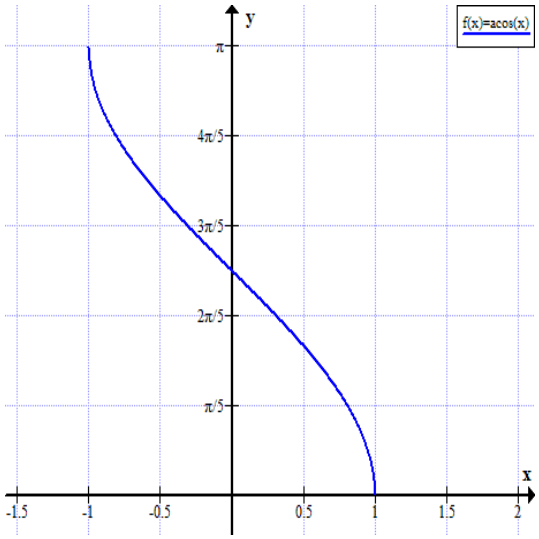

Figure 1: Graph of arccos(x)

1.4 Characteristics of arccos(x)

- The value of angle is the highest at -1, then it decreases and becomes zero at +1.
- The arccosine(x) is a one-to-one function. Its range is limited to 180° because after that values repeat itself which violates one-to-one property.
- The function can be used to calculate the base angle in a triangle by calculating base/hypotenuse to that angle if we know cos^{-1} of that value.

2 Problem 2

Express Requirements of function Arccos(x)

2.1 Functional Requirements

ID: FR
Description: The functional requirements specify the function or tasks to be performed by the system.
FR1: The function shall accept an input value to give an output.
FR2: The value to be entered by the user should be a real number between -1 to 1. The output returned is a real number between 0 to π
FR3: The function shall return an angle whose cosine value is the input number.
FR4: The function shall return the angle in right triangle whose corresponding base hypotenuse ratio is given.

2.2 Performance Requirements

ID: PR
Description: The performance requirements specify that the function is performed under some specified conditions.
PR1: The response time for the function should be less than 2 seconds.
PR2: It should also be able to handle the exception when the entered input is outside the domain of the function.

2.3 Usability Requirements

ID: UR
Description: The requirement for user performance and satisfaction.
UR1: The user should have knowledge of the function and its domain and co-domain.

2.4 Design Constraints Requirements

ID: DCR
Description: The requirements that limit the developer to follow a particular protocol to improve the design.
DCR1: Input Constraints: The input value should be a real number between -1 to 1 and the output should be between 0 to π. It should not be a character data type or a string.
DCR2: Function Language: The function shall be implemented in Java Language.

2.5 Non Functional Requirements

ID: NFR
Description: The Non Functional Requirements specify how a function is supposed to operate. It includes **Quality Requirements** and **Human Factors Requirements**.

- **Quality Requirements**
  ID: QR
  It includes:
  Reusability
  ID: RU
  The function shall be reused any number of times depending on the user.
  Reliability
  ID: RL
  The function shall provide accurate and reliable results.

- **Human Factors Requirements**
  ID: HFR
  It specifies the requirements of the Humans or Users with respect to the result of the function.
  It includes:
  Measure of usability
  ID: MUS
  The result of the function should be useful to the user or any stakeholder. The result should be accurate and must satisfy the user.
  Human Reliability
  ID: HRL
  The function should perform the task correctly. The value of the output should be within the co-domain of the function.

3 Problem 3

3.1 PseudoCode For Implementation of Arccos Function

$cos^{-1}x = 3.1415926535/2 - sin^{-1}x$
$sin^{-1}x = x + 1/2 * x^3/3 + 1/2 * 3/4 * x^5/5 + 1/2 * 3/4 * 5/6 * x^7/7 + ... \; where - 1 \leq x \leq 1$
$cos^{-1}x = 3.1415926535/2 - x + 1/2 * x^3/3 + 1/2 * 3/4 * x^5/5 + 1/2 * 3/4 * 5/6 * x^7/7 + ...$

3.2 Technical Reasons For Choosing Algorithms

The Arccos(x) function is solved by using the Taylor Series Expansions of Inverse Cosine Function. The series is an infinite series with Geometric Progression. The two algorithms used for solving the problems are Iterative Approach and Recursive Approach.

Iterative Approach: This approach follows the top down method. In this we use loops for calculating the sum of infinite series. We can keep a note when the loop results into infinity and accordingly terminate iterations till that point. This approach gives close results up to fifteen iterations after which it gives no results.

Advantages:

- The iterative approach is easy to understand as it involves loops which carry out the summation process incrementally.
- It uses less memory.

Disadvantages:

- There are certain complex problems in which number of iterations are very large and so iteration is difficult to perform.
- There can be many loops or nested loops involved in complex problems and the lines of code increases with it.
- The result is less optimized and less accurate than the recursion result.

Recursive Approach: This approach follows the bottom up method. In this we can use the function for performing the summation of the series. The function can call its own functions as many times. The result returned is added to the previous sum value every time a function is called. After some point we can check the values for infinity. We can call the recursive functions less than the infinity point.

Advantages:

- Although Recursion is little difficult to understand, it can solve complex problems in an easy and elegant way.
- In arccos(x) function implementation, we are performing summation for n=1 and later on calling the same function in itself for summation of remaining part.
- It gives more accurate results than the iterative approach.
- It reduces the lines of code for particular problem.

Disadvantages:

- It uses more memory.
- It becomes very complicated to solve problems. For example; when we are performing recursion, the control goes deeper into function and it becomes difficult to understand where the control is and how to get out of recursion to get results.
- Extra counter value is used to keep track of recursive operation and return the result just before the result shows infinite value.

3.3 Brief Description of the Algorithms

Iterative Approach
The solution to the function arccos(x) is based on the Taylor Series Expansion of Inverse Cosine Function.
$cos^{-1}x = 3.1415926535/2 - x + 1/2 * x^3/3 + 1/2 * 3/4 * x^5/5 + 1/2 * 3/4 * 5/6 * x^7/7 + ...$
where $-1 \leq x \leq 1$
This algorithm first does the summation of $1/2*x^3/3+1/2*3/4*x^5/5+1/2*3/4*5/6*x^7/7+....$ It is in gerometric progression. In the numerator, it is going 1, $1 * 3$, $1 * 3 * 5$. It is like the odd factorial, we calculate the odd factorial with while loop where the upper limit increases by 2 everytime.The same goes with the denominator. The denominator goes 2, $2*4$, $2*4*6$ in its terms starting from n=1. So we calculate even factorial separately with while loop. The upper limit of the loop always increases by 2. The power of x is increasing from 3 with increment of 2 every time. We calculate power of x by using for loop and multiplying x with itself till the loop ends. The denominator is the number which is the number on the power to x. So the result of the one term is (Odd factorial*Power of X)/(Even Factorial * Number on the power to X). This gets added to next term with incremented N and so on.

- Main loop of for starts with $N \leftarrow 1$ which increments and ends at 15 because after that result is becoming infinite.
- The power to X loop starts from 0 till 2N+1, and x is multiplied with itself to get X to power 2N+1.
- The Odd Factorial starts with 1 in the product and factorial is calculated to less than 2N+1.So value is 1 in first term which increments to next odd number in next case.
- The Even Factorial starts with in the product and factorial is calculated to less than 2N+1. So value in product is 2 in first case which increments on next cases to next even number.
- The denominator is the same number 2N+1 which is incremented every time with N.
- The value for each term starting from N=1 to 15 is added to the variable $asin$. The summation of all terms is stored in variable $asin$. The summation takes place for fifteen terms.
- This $asin$ value is subtracted from $3.1415926535/2 - x$ according to the formula to get the result which is stored in $acos$.
- The result for this algorithm: $arccos(-1) = 3.01518267$,$arccos(0) = 1.570796$,$arccos(1) = 0.1264099$
- The result is less accurate than recusrive approach. The result has 0.04 of relative error in the value at extreme input values such as 1 and -1.

Algorithm 1: Pseudocode for Arccos function: Iterative Approach

Data: Integer x where $-1 \leq x \leq 1$
Result: Value of arccos(x)
function getArccos(x);
$asin \leftarrow 0$;
$acos \leftarrow 0$;
for $N \leftarrow 1$ *to* 15 **do**
    $EvenFactorial \leftarrow 1$;
    $OddFactorial \leftarrow 1$;
    $PowertoX \leftarrow 1$;
    $Odd \leftarrow 1$;
    $Even \leftarrow 2$;
    for $Power \leftarrow 0$ *to* $2N + 1$ **do**
        $PowertoX \leftarrow PowertoX * x$;
    end
    while $Odd < 2N + 1$ **do**
        $OddFactorial \leftarrow OddFactorial * Odd$;
    end
    while $Even < 2N + 1$ **do**
        $EvenFactorial \leftarrow EvenFactorial * Even$;
    end
    $asin \leftarrow asin + (EvenFactorial * PowertoX)/(OddFactorial * 2N + 1)$;
end
$acos \leftarrow 3.1415926535/2 - x - asin$;
return $acos$;
end getArccos

Recursive Approach
The same formula for Arccos(x) is used as before. $cos^{-1}x = 3.1415926535/2 - x + 1/2 * x^3/3 + 1/2 * 3/4 * x^5/5 + 1/2 * 3/4 * 5/6 * x^7/7 + ...$ where $-1 \leq x \leq 1$
The summation $1/2*3/4*x^5/5+1/2*3/4*5/6*x^7/7+...$ is taken separately. We find that in the numerator it is odd factorial such as 1, $1 * 3$, $1 * 3 * 5$. The denominator is having Even Factorial such as $2, 2 * 4, 2 * 4 * 6$. The numerator is multiplied by X to power of odd number starting from X^3 , X^5, X^7. The denominator is multiplied by odd number that is same as that on power of X, such as 3, 5 ,7.So we perform operations separately. As odd number here starts from 3, we take a counter starting from 3 which is incremented by 2 after the operations. We take the sum to be 0 at first.

- In the function RecursiveArccos(x), The power to X is calculated by using for loop. In the for loop X is multiplied by itself in the loop to give X^power. Loop goes to less than the counter value. In first term, it goes from 0 to less than 3.
- The Odd Factorial is calculated by using for loop. The product has value 1 initially which stores product of odd factorial. Loop goes from 1 to less than the counter value.
- The Even Factorial is calculated by using for loop. The product has value 2 initially which stores product of even factorial. Loop goes from 2 to less than counter the value.
- The sum is calculated by $(OddFactorial * PowertoX)/(EvenFactorial * Counter)$.
- For first sum is calculated for $x = -1$ as $(1*(-1)^3)/(2*3)$. The Counter is then incremented by 2.The flow goes to $sum = sum + RecursiveArccos(x)$
- Same operations are carried out again.Counter value is 5. The Power to X is X^5. Odd factorial is $1 * 3$ and even factorial is $2 * 4$. The value of sum is calculated again and is equal to $((1 * 3)(-1)^5)/((2 * 4) * 5)$. This value is added to previous sum value. The flow recursively goes to its own function again
- The counter value 301 is taken because after this value goes to infinity.After this, the data flow is returned from all the functions to the main function.
- The returned value is subtracted from $3.1415926535/2 - x$ as mentioned in the formula and stored in the $result$ variable
- The result for this algorithm $arccos(-1) = 3.095514$, $arccos(1) = 0.0460786$,$arccos(0) = 1.57079632$
- The result is more accurate than iterative approach. The result has 0.014 relative error at the extreme values such as 1 and -1.

Algorithm 2: Pseudocode for Arccos function: Recursive Approach

Data: Integer x where $-1 \leq x \leq 1$
Result: Value of arccos(x)
$sum \leftarrow 0$;
$counter \leftarrow 3$;
function RecursiveArccos(x);
$EvenFactorial \leftarrow 1$;
$OddFactorial \leftarrow 1$;
$PowertoX \leftarrow 1$;
for $Power \leftarrow 0$ *to* $Counter$ **do**
    $PowertoX \leftarrow PowertoX * x$;
end

for $Odd \leftarrow 1$ *to* $Counter$ **do**
    $OddFactorial \leftarrow OddFactorial * Odd$;
    $Odd \leftarrow Odd + 2$;
end
for $Even \leftarrow 1$ *to* $Counter$ **do**
    $EvenFactorial \leftarrow EvenFactorial * Even$;
    $Even \leftarrow Even + 2$;
end
$sum \leftarrow (EvenFactorial * PowertoX)/(OddFactorial * Counter)$;
$Counter \leftarrow Counter + 2$;
if $Counter \leq 301$ **then**
    $sum \leftarrow sum + RecursiveArccos(x)$;
    return sum;
else
    return sum;
end
end RecursiveArccos(x)
$result \leftarrow 3.1415926535/2 - x - RecursiveArccos(x)$

4 References

- http://mathworld.wolfram.com/InverseCosine.html
- https://www.efunda.com/math/taylor_series/inverse_trig.cfm
- https://www.mathopenref.com/arccos.html
- https://tex.stackexchange.com/
- https://stackoverflow.com/
- https://www.overleaf.com/learn/latex/Creating_a_document_in_LaTeX