0.1 The arccosine function

0.1.1 Description

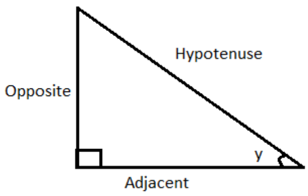
The arccosine function is the inverse trigonometric function.
The arccosine of x is defined as the inverse cosine function of x when -1≤x≤1. When the cosine of y is equal to x:

$$\cos y = x \tag{1}$$

Then the arccosine of x is equal to the inverse cosine function of x, which is equal to y:

$$\arccos x = \cos^{-} 1x = y \tag{2}$$

(Here $\cos^{-}$ 1 x means the inverse cosine and does not mean cosine to the power of -1).[2]
For example,

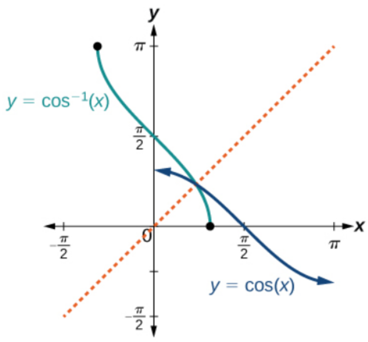


$$\cos y = \frac{Adjacent}{Hypotenuse} \to y = \arccos(\frac{Adjacent}{Hypotenuse}) \tag{3}$$

0.1.2 Domain & Co-Domain of arccosine

- The domain of $\arccos x$ is -1 ≤ x ≤ 1.
- The range of $\arccos x$ is 0 ≤ y ≤ π in radians or 0°≤ y ≤180° in degrees.
- It is most useful when trying to find the angle measure when two sides of a triangle are known.

0.1.3 Properties of arccosine



- For the arccosine function to be a true inverse function of the sine function, the following statement must be true: $\cos(\arccos(x)) = x$ and $\arccos(\cos(x)) = x$
- The arccosine function is a reflection of the cosine function about the line $y = x$.
- The arccosine function is defined when -1 ≤ x ≤1
- The arccosine function is continuous on open interval (-1,1)

0.1.4 Application of arccosine

- Arccosine function are unique function and useful in finding remaining angles of right triangle.
- It is also useful in application of engineering, physics and others.

0.2 Requirements for arccosine

There are some functional and non-function requirement for $F(x) = y = \arccos x$,

- When value for F(x) is defined,the System should print defined value of F(x)
- When value for x is entered outside domain range,the System should print undefined.
- When user enters data of any other types rather than numbers as an input,the System shall not accept that entered data as an input.
- The program should cover all possible test cases

0.3 Requirements

- **ID** : FN1
  TYPE : Functional Requirement
  PRIORITY : 1
  VERSION : 1.0
  DESCRIPTION : If F(x) is defined for the given value of x then function should return the correct value for F(x).
  RATIONALE : The rationale behind this requirement is that the function only outputs the value if x is between -1 and 1(inclusive), undefined otherwise.

- **ID** : FN2
  TYPE : Functional Requirement
  PRIORITY : 1
  VERSION : 1.0
  DESCRIPTION : When user enters data of any other types rather than numbers as an input,the function shall not accept that entered data as an input. **RATIONALE** : The rationale behind this requirement is to prevent users from entering any unsupported inputs.

- **ID** : NFNR1
  TYPE : Non-Functional Requirement
  PRIORITY : 3
  VERSION : 1.0
  DESCRIPTION : The output should be accurate upto 4 decimal places.

- **ID** : NFNR2
  TYPE : Non-Functional Requirement
  PRIORITY : 2
  VERSION : 1.0
  DESCRIPTION : The output should be generated within a stipulated time.

0.4 Algorithm for arccosine

After exploring various possible approaches to solve arccosine function using Taylor's series for its evaluation seemed most appropriate.[3]

$$\arccos x = \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2} \frac{x^{2n+1}}{(2n+1)}, |x|<1 \tag{4}$$

In order to arrive at an optimal solution, from a performance perspective, we tried implementing and further comparing an iterative version of the algorithm against a recursive one.
Although the theoretical time complexity for both the approaches was same but by actually measuring the execution-time for a running sample we arrived at a conclusion that the iterative version performed better than the recursive one.
We attributed this to the additional overhead of maintaining a call-stack in the recursive implementation. In-fact this in turn resulted in more memory consumption compared to the iterative approach.

Algorithm 1 Algorithm to implement(using iteration) $y = \arccos(x)$

```
function CalculatePiValue
    PiValue ← 0.0
    for k ← 0 to 9999 do
        First ← Power(−1, k)
        Second ← (2 ∗ k) + 1
        Value ← First/Second
        PiValue ← PiValue + Value
    end for
    PiValue ← 4 ∗ PiValue
    return PiValue
end function

function Arcos(value)
    if num == −1 then
        return 3.14159265
    end if
    if num == 1 then
        return 0.0
    end if
    intermediateValue ← 0
    for n ← 0 to 86 do
        a ← Factorial(2 ∗ steps)
        if Double.isInfinite(a) then
            break
        end if
        b ← Power(2, (2 ∗ steps))
        c ← Factorial(steps)
        d ← Power(c, 2)
        A ← (a/(b ∗ d))
        exp ← (2 ∗ steps) + 1
        e ← power(value, exp)
        B ← e/exp
        AB ← (A ∗ B)
        intermediateValue ← intermediateValue + AB
    end for
    pivalue ← CalculatePiValue()
    finalans ← ((pivalue/2) − ans)
    return finalans
end function

function Factorial(i)
    if i == 0 then
        ans ← 1
        return ans
    end if
    for j ← 1 to i do
        ans ← ans ∗ j
    end for
    return ans
end function
```



```
function Power(c, j)
    ans ← 1.0
    if j == 0 then
        ans ← 1
        return ans
    end if
    for i ← 1toj do
        ans ← c ∗ ans
    end for
    return ans
```

Algorithm 2 Algorithm to implement $y = \arccos(x)$

```
function CalculatePiValue
    PiValue ← 0.0
    for k ← 0 to 9999 do
        First ← Power(−1, k)
        Second ← (2 ∗ k) + 1
        Value ← First/Second
        PiValue ← PiValue + Value
    end for
    PiValue ← 4 ∗ PiValue
    return PiValue
end function

function Arcos(value)
    if num == −1 then
        return 3.14159265
    end if
    if num == 1 then
        return 0.0
    end if
    intermediateValue ← ProcessSeries(value, 0, 0)
    ans ← (PI/2) − intermediateValue
    return ans
end function

function ProcessSeries(value, steps, intermediateValue)
    a ← Factorial(2 ∗ steps)
    if Double.isInfinite(a) then
        return intermediateValue
    end if
    b ← Power(2, (2 ∗ steps))
    c ← Factorial(steps)
    d ← Power(c, 2)
    A ← (a/(b ∗ d))
    exp ← (2 ∗ steps) + 1
    e ← power(value, exp)
    B ← e/exp
    AB ← (A ∗ B)
    intermediateValue ← intermediateValue + AB
    steps ← steps + 1
    return ProcessSeries(value, steps, intermediateValue);
end function

function Factorial(i)
    if i == 0 then
        return 1
    end if
    return i ∗ Factorial(i − 1)
end function

function Power(c, j)
    ans ← 1.0
    if j == 0 then
        ans ← 1
        return ans
    end if
    for i ←1 to j do
        ans ← c ∗ ans
    end for
    return ans
end function
```

Bibliography

[1] https://courses.lumenlearning.com/boundless-algebra/chapter/trigonometric-functions-and-the-unit-circle/

[2] https://www.rapidtables.com/math/trigonometry/arccos.html

[3] https://proofwiki.org/wiki/Power_Series_Expansion_for_Real_Arccosine_Function