SOEN-6011 Software Engineering Processes

Calculate: CosInverse (Arccos(x))

Professor: Dr. Pankaj Kamthan
Submitted by: Jhanvi Arora

1 Problem: Arccosine(x)

1.1 Description [1]

Compute the value of arccos(x) for any given number x with minimum absolute error difference in resemblance to scientific calculators, given x satisfies the domain requirements of the aforementioned function.

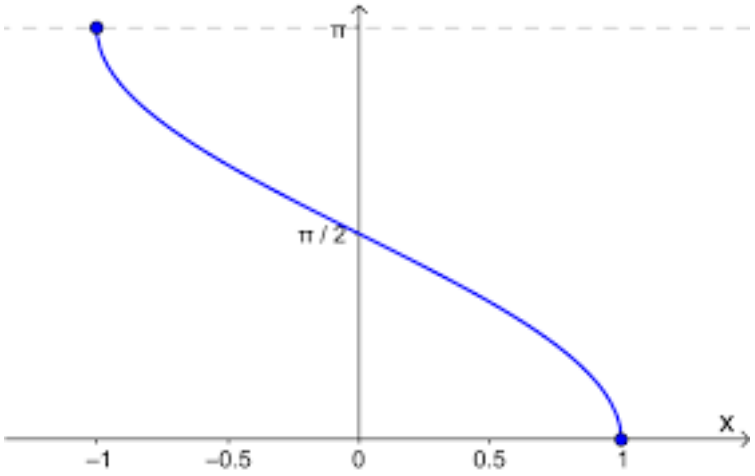

Figure 1: Graph of function $(\arccos(x), x)$

From the above graph 1.1 following properties of function are inferred:

- It is decreasing function.
- It has no mirror image across x axis nor in y-axis.

1.2 Domain

For function,

$$y = arccos(x)$$

:

The value of x should satisfy: $x \in R, \quad -1 \leq x \leq 1$

1.3 Co-domain

For function,

$$y = arccos(x)$$

The value of y should satisfy: $y \in R, \quad 0 \leq y \leq \pi$

2 Context-Use-Model

In the following environment scenarios, the various interactions with the developed system are described as follows:

2.1 Technical Environment

1. Developers:

   Understand reusability and maintainability of modules in the calculators

2. Testers:

   Understand and test functionality of modules.

3. DevOps Personnel:

   Setup and maintain the CI/CD pipeline for the developed system and handle its deployment options.

2.2 Non-Technical Environment

1. Students:

   University and high school students whose coursework include math/math-based functions

2. Architects and Design Planners:

   Planning of various construction sites, determine elevation and tilt requires use of the function.

3. Mathematicians:

   Apply the output results to derive their implications.

3 Functional Requirements

3.1 Assumptions

- ID: A1

  Input is a real number.

- ID: A2

  The results are precise upto 6 decimal places.

- ID: A3

  The scientific calculator derived input constants used in function calculations are correct.

3.2 Requirements

- ID: R1
  - Version: 1.0
  - Type: Functional Requirement
  - Priority: 1
  - Risk: High.
  - Description: Should not accept any out of defined-domain values.
  - Rationale: Function undefined for out-of-domain values.
- ID: R2
  - Version: 1.0
  - Type: Functional Requirement
  - Priority: 1
  - Risk: High.
  - Description: Result should not be out of bounds of co-domain.
  - Rationale: Output of function is fixed within its co-domain.
- ID: R3
  - Version: 1.0
  - Type: Functional Requirement
  - Priority: 2
  - Risk: Low.
  - Description: Error should be in [0, 0.0001]
  - Rationale: It should provide precision upto atleast 4 decimal places.
- ID: R4
  - Version: 1.0
  - Type: Functional Requirement

  - Priority: 3
  - Risk: Low.
  - Description: Display result in degree.
  - Rationale: Some applications require degree output instead of default radians.

- ID: R5
  - Version: 1.0
  - Type: Non Functional Requirement
  - Priority: 3
  - Risk: Low.
  - Description: Yield output in lowest time.
  - Rationale: To function as Calculator it should provide output in milliseconds.

4 Algorithm

4.1 Main Algorithm

Algorithm 1: Fast Arc Cosine Algorithm [3].

Data: $-1 \leq x \leq 1$
Result: $y = arccos(x)$
if $x < 0$ **then**
  $negative \leftarrow 1$
else
  $negative \leftarrow 0$
end
if $(x == 1)$ **then**
  $y = 0$
end
if $(x == -1)$ **then**
  $y = \pi$
end
if $(x < -1)|(x > 1)$ **then**
  $y = -1$
end
if $(x < -0)$ **then**
  $x = x * -1$
end
$result \leftarrow fv1$;
$result \leftarrow result * x$;
$result \leftarrow result + fv2$;
$result \leftarrow result * x$;
$result \leftarrow result + fv3$;
$result = result * x$;
$result \leftarrow result + fv4$;
$result \leftarrow result * squareRoot(1 - x)$;
$result \leftarrow result - 2 * negative * result$;
$y \leftarrow negative * \pi + result$;

4.2 Description

The aforementioned algorithm Fast Arc Cosine is an implementation of the arc cosine derived by the approximation expansion of taylor series of arccos(x) -[3]. It has several constant approximation values that are used in calculations at each step- the sub functions used in approximation process are squareRoot and absolute which are a part of the developed system in code.

Algorithm 2: SquareRoot

Data: $x > 0$
Result: $y = squareRoot(x)$
$y = x/2$;
$temp \leftarrow None$;
while $temp - y == 0$ **do**
    $temp \leftarrow y$;
    $y = (temp + (x/temp))/2$;
end

Algorithm 3: Radian To Degree

Data: $xinrad$
Result: $y = degree(x)$
$y = x * 180/\pi$;

The other algorithm described as Strict Math Algorithm [4] is the second choice- and it derives the arc cos functions' approximation constants based on its arc sine calculations.

4.3 Algorithm Choices

The options to choose from while implementing the function :

- **Fast Arc Cosine Algorithm[3].**
  Advantages
  - Approximation Algorithm - hence with derived memory constants takes lesser time.
  - Uses less number of memory constants in comparison to Strict Math Algorithm .

  Disadvantages
  - Precision upto only 5 decimal values. However in terms of requirements, it satisfies R3 and even satisfies R5 better than Strict Math.
- **Strict Math Approximation Algorithm [4].**
  Advantages
  - Use of memory constants and no dependency on other functions.
  - Provides precision upto 8 decimal places.

  Disadvantages
  - Takes more time than Strict Fast Arc Cosine.
  - Requires a lot of memory for saving up precise constants.

Algorithm 4: Strict Math Algorithm [4]

Data: $-1 \leq x \leq 1$
Result: $y = arccos(x)$
$negate \leftarrow True?x < 0 : False$
$x = (negate)? - x : x$
if $(x == 1)$ **then**
| $y \leftarrow 0$
end
if $(x == -1)$ **then**
| $y \leftarrow \pi$
end
if $(x < -1)|(x > 1)$ **then**
| $y \leftarrow -1$
end
if $x < 1/\pi/2$ **then**
| $y \leftarrow \pi/2$
end
$a \leftarrow (x < 0.5)?x * x : (negate?(1 + x) * 0.5 : (1 - x) * 0.5)$
$b \leftarrow a * ((b0 + a * (b1 + a * (b2 + a * (b3 + a * (b4 + a * b5)))))$
$c \leftarrow 1 + a * (c0 + a * (c1 + a * (c2 + a * c3)))$
if $x < 0.5$ **then**
| $d \leftarrow x - (d1/2 - x * (b/c)$
| y = (negate ? $\pi/2 - x * (b/c)$)
end
$e \leftarrow squareRoot(a)$
$ggets(a - e * e)/(e + e)$
$f \leftarrow negate?b/c * e - d1/2 : b/c * e + g$
$y = 2 * (e + f)$

- **Value of arccos(x) derived from arctan(x) [2].**
  Advantages
  - arctan(x) is easier to obtain in terms of taylor series expansion.

  Disdvantages
  - Time taking Recursive Function.
  - Not exact Approximation in comparison to other two algorithms.

The first two algorithms are both approximation based approaches with fixed constant values and defined set of calculations. The third approach is polynomial based approach which entails the calculation of arctan(x) first and then using the approximated value it derives the value of arccos(x).

Strict Math was initially implemented while developing the calculation system. However owing to time of yielding result and space to save up constants - Fast Arc Cosine was instead preferred.

5 Implementation

The implementation of the described function using the aforementioned Algorithm is carried out using Java. The project work for the same is carried out on JetBrains IntelliJ, however it is developed independent of the IDE environment. Hence, can be run on any compatible systems.

The functions used in the code do not make use of any Java in-built libraries, the math functions used in the function are implemented using code.

5.1 Debugger

The debugger used in the system development is IntelliJ IDE's inbuilt debugger - which automates the process of debugging by providing multiple options of evaluating various breakpoints.

Advantages

- Selects breakpoints on its own when called upon a function.
- Can Step out of the breakpoint - when necessary.
- Update the value of the data attributes while in debug mode.

Disadvantages

- Difficult to analyze big volumes of code.
- Doesn't trace data-flow when it comes from different modules.

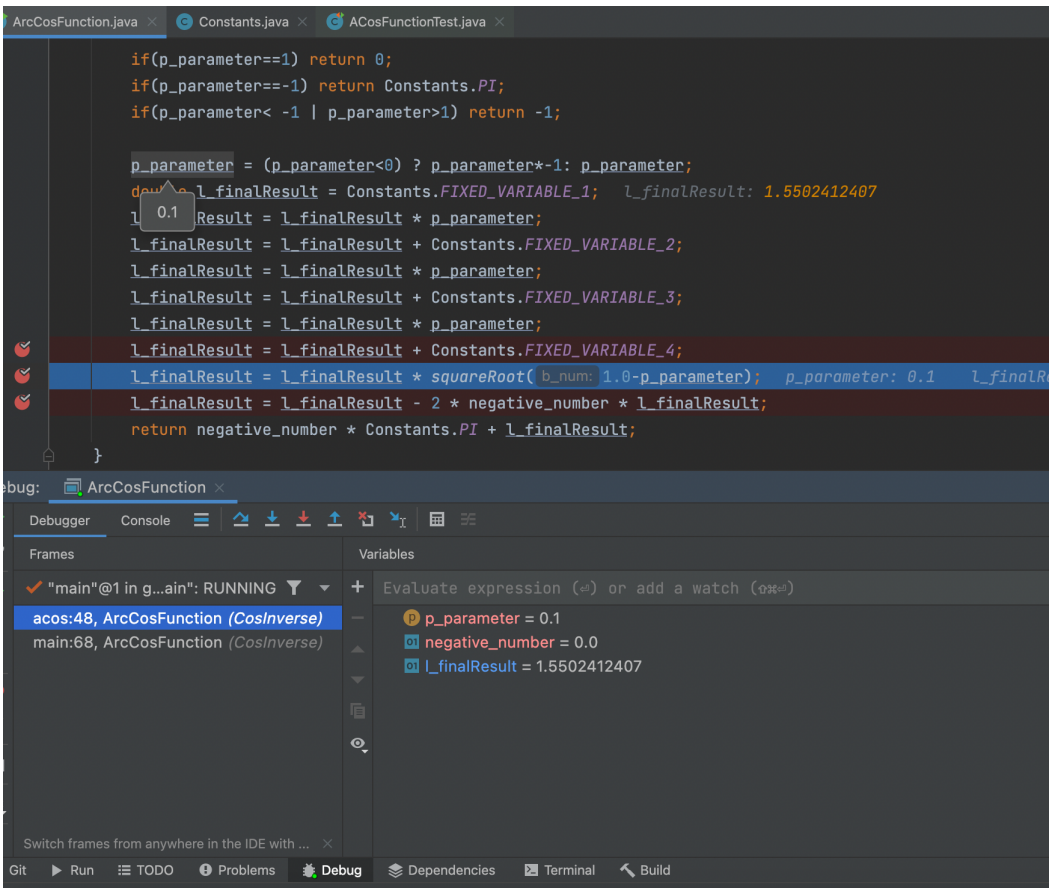

Figure 2: Evaluation of Line Breakpoint

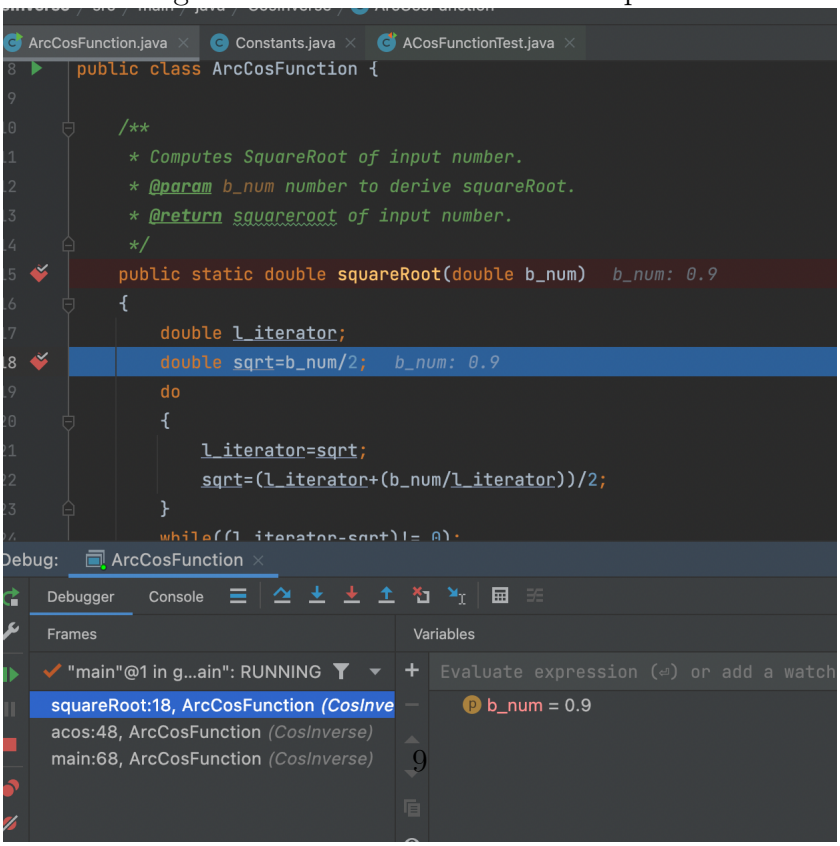

Figure 3: Evaluation of Method Breakpoint

As in Figure- the debugging of line breakpoints causes the debugger to display the current state information of all variables at that time. The method breakpoint implies all the functional lines of the code are set as breakpoints by default and we can monitor at each the step the contents of various data parameters.

5.2 Code Quality

The following enumerates the code quality characteristics ensured while implementation of the calculator function:

1. **Space Efficient**
   Effort: Used the algorithm that uses lesser number of memory constants.

2. **Portable**
   Effort: Complete implementation of the function is encased in a class that is just dependent on the Constants - hence can be plugged in to create a object and calculate values as and when needed.

3. **Maintainable**
   Effort: The functions of the code are reusable and are independent of each other and hence any changes in the one won't affect the other.

4. **Correctness**
   Effort: Each of the functions are tested for all possible input cases to verify the working.

5. **Robustness**
   Effort: The code is robust and handles all the input yielding correct outputs in reference to the given requirements.

6. **Time-Efficient**
   Effort: Over the use of recursive and calculation extensive method algorithms as in previous section- the most efficient and less time taking algorithm is implemented.

7. **Usable**
   Effort: The functions can be immediately accessed by calling upon the object of the class - hence the implemeneted system is fully usable.

5.3 Checkstyle [5]

Checkstyle is a tool that imposes programming code style check on the source code.

- It is to ensure that the written code conforms to the quality measures as defined by the standards.
- The programming style conventions checked for by the checkstyle are Google Java Programming Standards.
- The aforementioned is imposed using the Maven framework using which the project is initialised.

Advantages

- It prompts on any inefficient coding- convention followed.
- It is automated and checks the whole project build on one command.
- Custom - Quality Conventions can also be instilled in Checkstyle to ensure them.

Disadvantages

- It doesn't prompts immediately - but when run on the whole project
- It requires the code to be compiled by the compiler.
- Refactored or ill-structured code aren't checked by checkstyle.

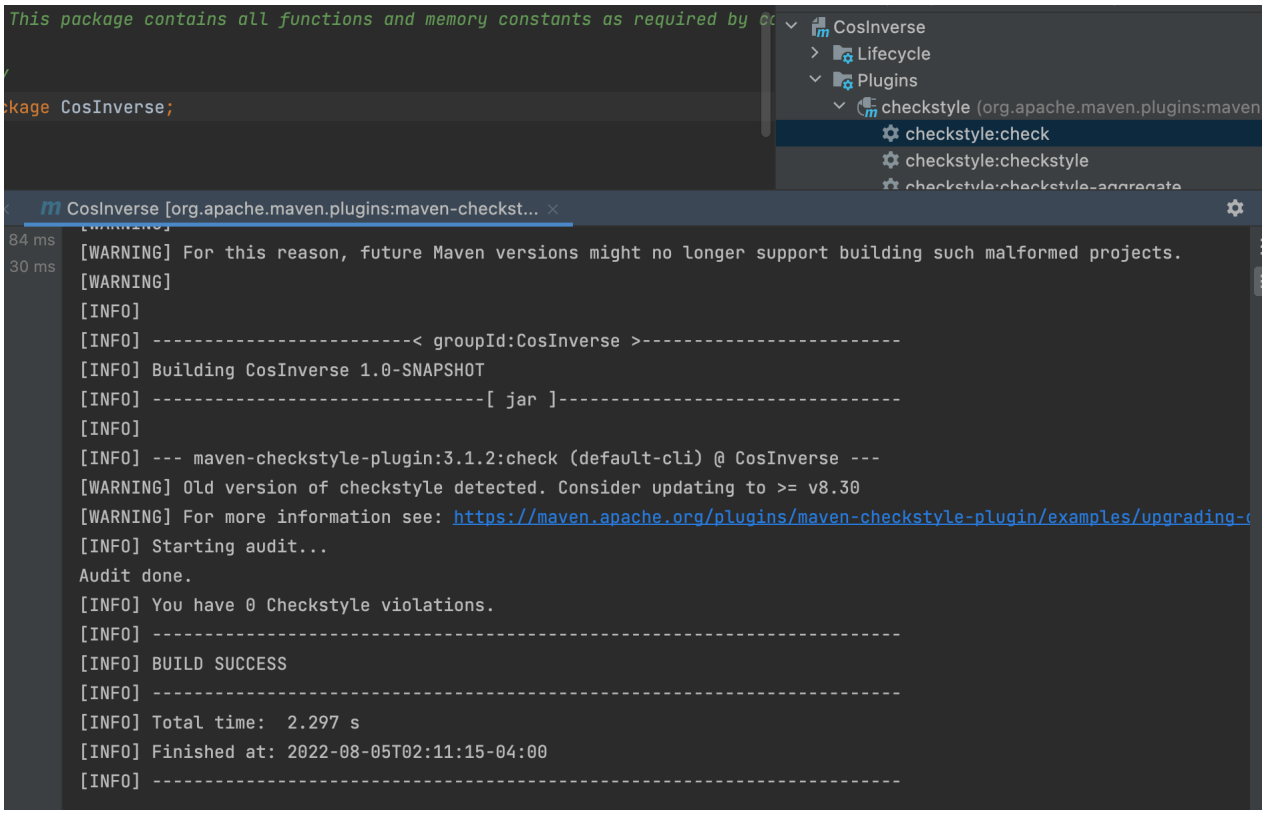

Figure 4: Snapshot of Checkstyle after completion

6 Testing

- The testing of the implemented system has been performed using the JUnit framework 4.13.2 and was installed using the maven dependency.
- The test cases for all functions and each of their possible scenarios is included, verified and checked.

Requirement	Test Name	Description
R1	testOutOfRange, testWrongInput	O/p not defined for out of bound values.
R2	testpos1, testneg1	Checks value of function at boundaries
R3	testpositive	Checks if the error delta is within 0.00001
R4	testDegreeResult	Checks if result in degree is correct

Table 1: Test Mapping to Requirements

The implemented system passes all the aforementioned test cases henceforth satisfying all the functional requirements of the system.

References

[1] Arc cosine,
`https://www.cuemath.com/trigonometry/arccosine/`

[2] Taylor Series Expansion,
`http://scipp.ucsc.edu/ haber/ph116A/taylor11.pdf`

[3] Abramowitz and Stegun,
`https://personal.math.ubc.ca`

[4] Strict Math Algorithm,
`https://developer.classpath.org/doc/java/lang/StrictMath-source.html`

[5] Checkstyle,
`https://checkstyle.sourceforge.io/`