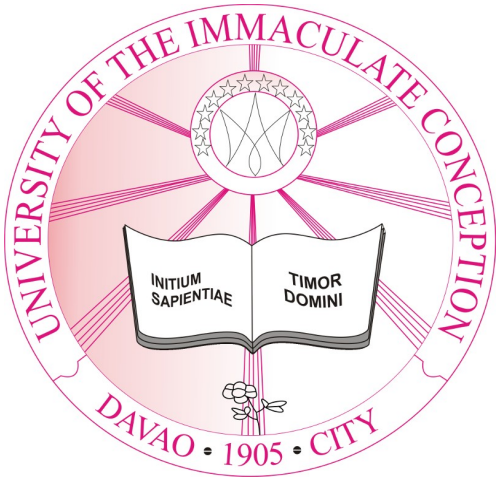
Trigonometric Calculations Using Programming Paradigms



Presented by
Raper, Julieza Jane Bella A.

Presented for
CS321 | Programming Languages
Prof. Christopher Josh Dellosa

University of the Immaculate Conception

INTRODUCTION

In this case study, dealing with diverse *procedural*, *functional*, *object-oriented*, and *logic programming paradigms* in the context of **trigonometric computations** is examined in terms of their practical application. This paper offers important insights into the applicability and worth of various programming paradigms for trigonometric calculations through thorough investigation. Numerous fields, such as mathematics, physics, engineering, computer graphics, and navigation, frequently require trigonometric computations.

Trigonometric functions like sine, cosine, and tangent are computed by these applications. To handle complex problems and imitate real-world events, trigonometric computations must be accurate and effective. Distinct programming paradigms offer different approaches to structuring code and resolving issues. Procedural, functional, object-oriented, and logic programming are four well-known paradigms in programming. Each paradigm offers a distinct set of guidelines and methods for resolving programming-related problems.

The procedural paradigm, emphasizes the sequential execution of commands, the functional paradigm, emphasizes the use of pure functions and immutability, the object-oriented paradigm, which organizes code around objects and encapsulation, and the logic paradigm, which is based on logical rules and constraints, will all be covered in this case study.

Able to determined and choose the best paradigm for their unique needs by being aware of how each programming paradigm performs and differs from one another in the context of trigonometric computations. Additionally, by demonstrating how programming paradigms may be used to solve trigonometric problems, this case study will help us better grasp the implications of programming paradigms in general and could serve as inspiration for future research and development in this area.

METHODOLOGY

Programming languages that fit each programming paradigm have been specifically chosen for this case study.

- **Procedural Paradigm:** *Java programming language* will be used for implementing the procedural paradigm. Defining functions or procedures that directly calculate the trigonometric functions using standard mathematical formulas.
- **Functional Paradigm**: *JavaScript programming language* will be employed to implement the functional paradigm. Focus on defining pure functions that calculate the trigonometric functions based on mathematical formulas.
- **Object-Oriented Paradigm:** *Python programming language* will be chosen for implementing the object-oriented paradigm. Involve creating classes and objects to represent angles and trigonometric functions.
- **Logic Paradigm:** *Prolog programming language* will be utilized to implement the logic paradigm. Revolve around defining logical rules and constraints that capture the relationships between angles and trigonometric functions.

Calculating an angle's **sine**, **cosine**, and **tangent** is one of the trigonometric problems chosen for analysis. These issues contain mathematical computations that can be tackled using different programming paradigms and encompass basic trigonometric functions.

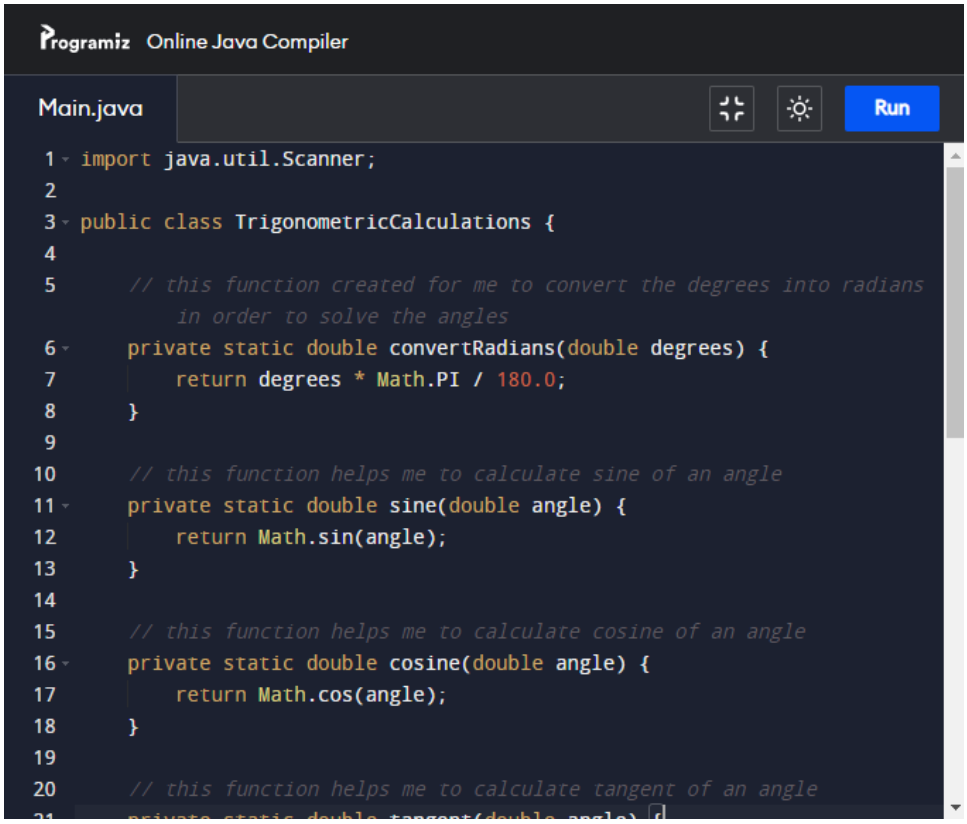
Performing these programs in an online compiler, for Java, JavaScript, and Python in order to perform procedural, functional, OOP paradigms, respectively, I used the Programiz Online Compiler (https://www.programiz.com/). For Prolog, I used the replit to perform the logic paradigm (https://replit.com/).

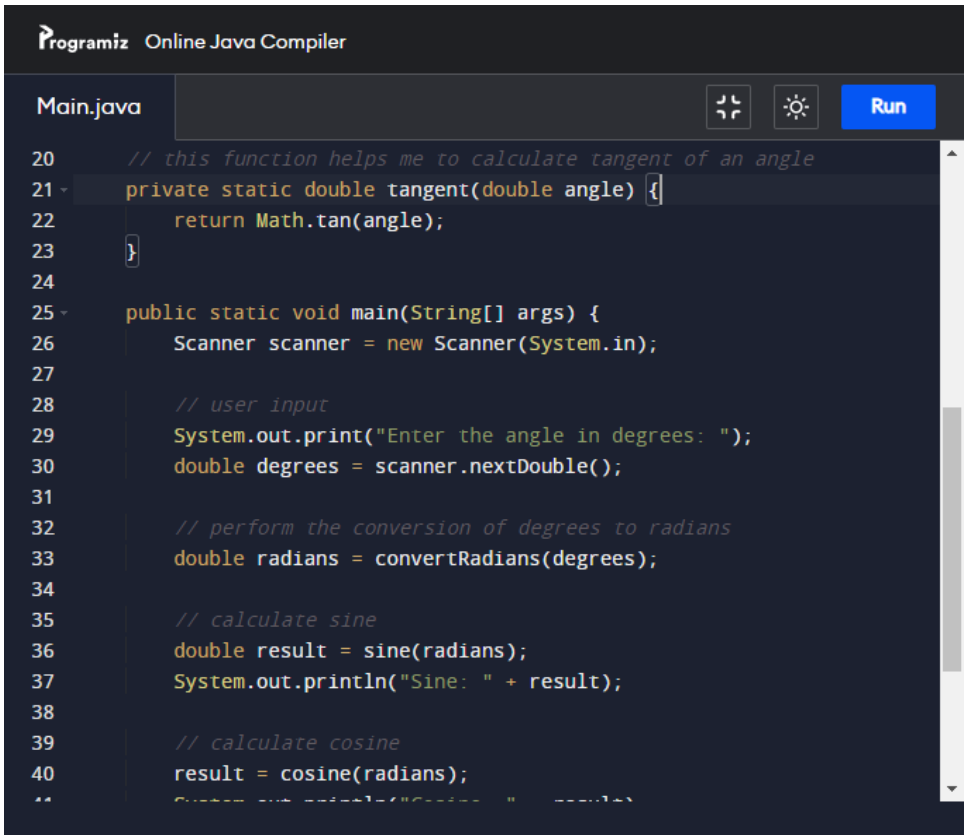
RESULTS AND DISCUSSION

The results of this case study show how trigonometric calculations can be carried out using a variety of programming paradigms, including procedural, functional, object-oriented, and logic paradigms. Each paradigm offers a distinct method for tackling trigonometric issues, with variable degrees of performance, expressiveness, and usefulness.

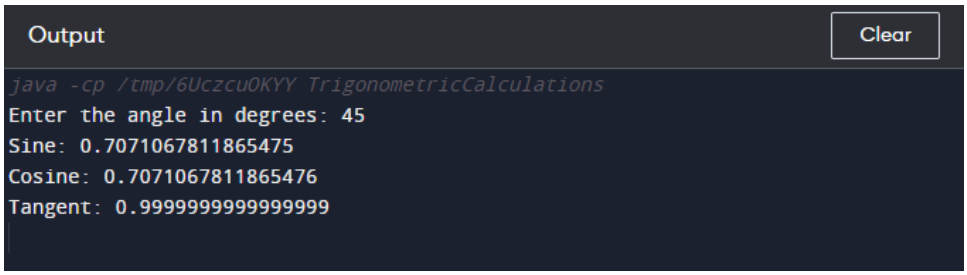
Procedural Paradigm

The sine, cosine, and tangent were calculated separately in the procedural paradigm in order to perform trigonometric computations. The structure of the procedural code allowed for direct variable manipulation and step-by-step execution. But other paradigms' modularity and encapsulation were absent from the code.




Programiz Online Java Compiler
Main.java
Run
// this function helps me to calculate tangent of an angle
private static double tangent(double angle) {
return Math.tan(angle);
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
// user input
System.out.print("Enter the angle in degrees: ");
double degrees = scanner.nextDouble();
// perform the conversion of degrees to radians
double radians = convertRadians(degrees);
// calculate sine
double result = sine(radians);
System.out.println("Sine: " + result);
// calculate cosine
result = cosine(radians);

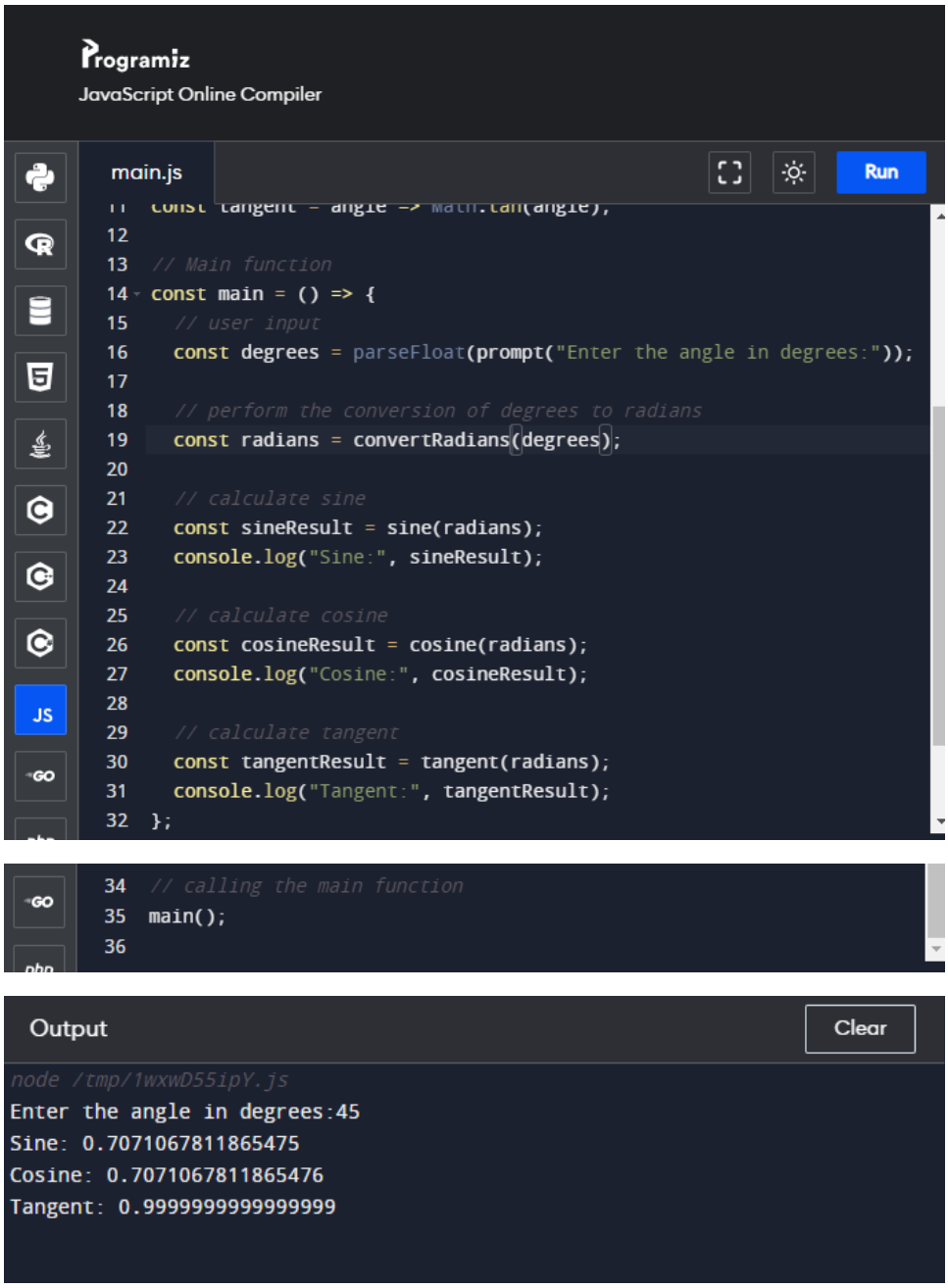

Programiz Online Java Compiler
Main.java
Run
System.out.print("Enter the angle in degrees: ");
double degrees = scanner.nextDouble();
// perform the conversion of degrees to radians
double radians = convertRadians(degrees);
// calculate sine
double result = sine(radians);
System.out.println("Sine: " + result);
// calculate cosine
result = cosine(radians);
System.out.println("Cosine: " + result);
// calculate tangent
result = tangent(radians);
System.out.println("Tangent: " + result);
scanner.close();
}
}



This line of code shows how to convert a degree into a radian, as well as how to compute an angle's sine, cosine, and tangent. The primary method receives a user-provided angle in degrees and converts it to a radian value before computing and displaying the pertinent trigonometric numbers using the Java Math class. Use the Scanner class to read user input from the terminal.

Functional Paradigm

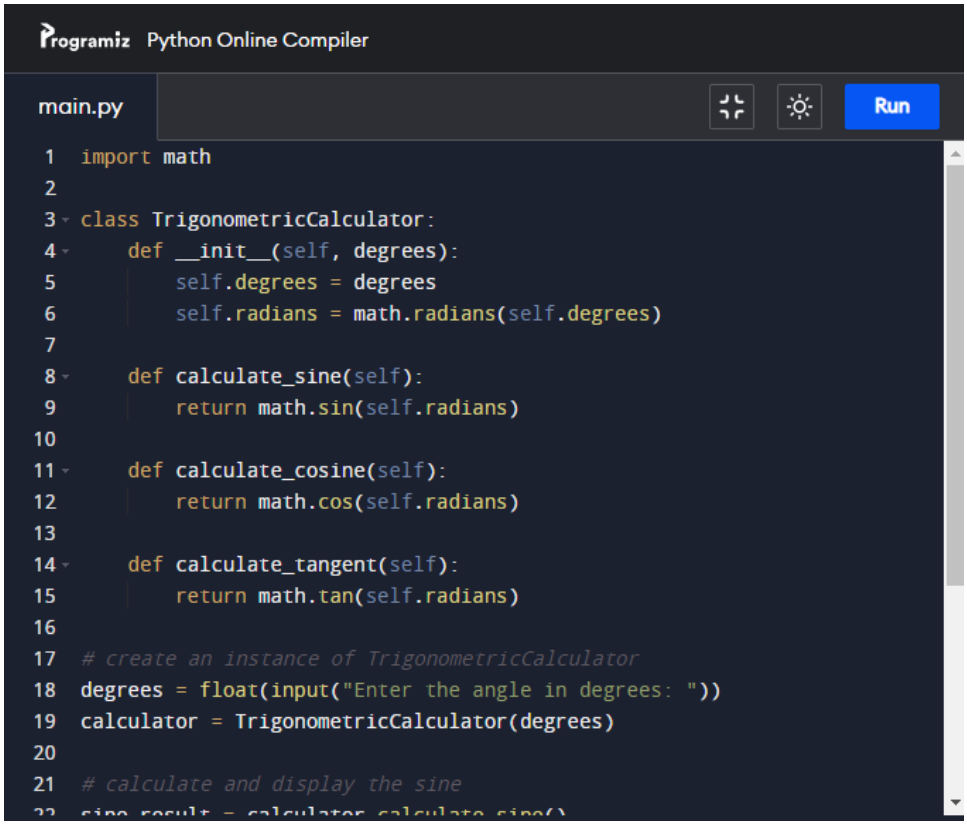
Trigonometric computations were accomplished using pure functions in the functional paradigm. The emphasis on immutability and referential transparency in functional code made it simpler to reason about and test. The functional method, however, introduced a level of complexity in managing function composition and recursion and demanded a paradigm shift in thinking.

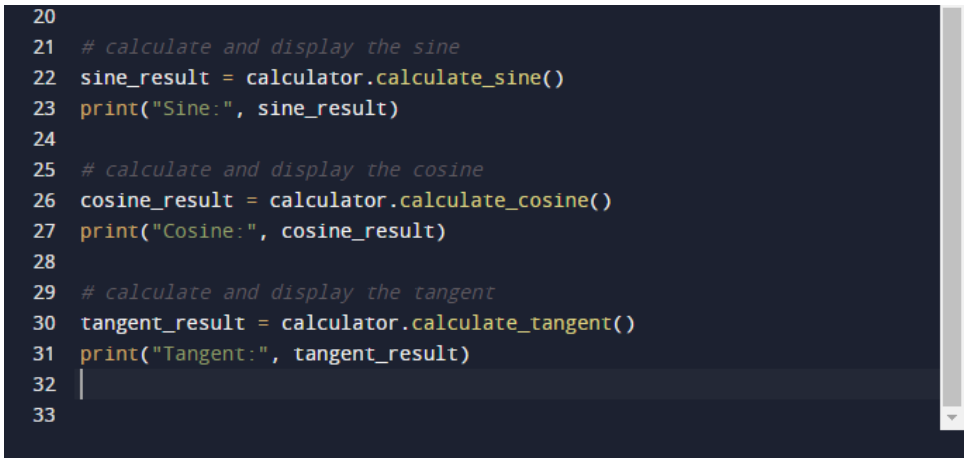


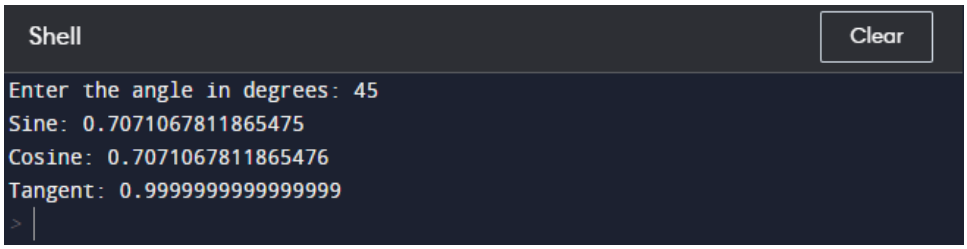
This piece of code defines functions using the syntax of arrow functions to convert degrees to radians and compute the sine, cosine, and tangent of an angle. The main function requests a user-supplied angle in degrees, transforms it to radians, calculates the relevant trigonometric values, and outputs them using the Math object built into JavaScript. To read user input from the console, use the prompt and parseFloat functions. The console.log method is used to display the outcomes.

Object-oriented Paradigm

Trigonometric computations were built using the object-oriented paradigm by encapsulating data and behavior in classes and objects. For trigonometric operations, this approach offered encapsulation, modularity, and reuse.



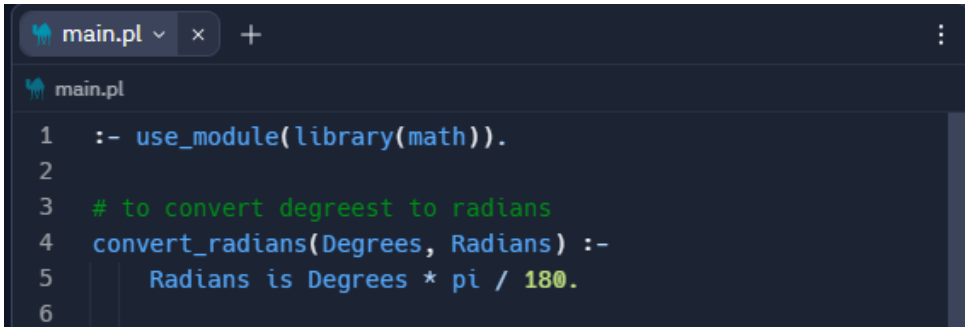




In this piece of code, a class called TrigonometricCalculator contains the functionality for carrying out trigonometric calculations. The degrees and radians attributes are initialized by the constructor __init__ using an angle in degrees. The class also has three methods, calculate_sine, calculate_cosine, and calculate_tangent, which use the math module from the Python standard library to do the necessary trigonometric computations. We may utilize the class by creating a TrigonometricCalculator object and providing an angle expressed in degrees as input. The input function is used to collect user input, while the print function is used to display the results.

Logic Paradigm

Trigonometric computations were accomplished in the logic paradigm by using logical rules and limitations. Logical inference was possible because prolog predicates reflected the connections between angles and trigonometric functions. The logic paradigm offered a declarative method for problem solving, but in contrast to the other paradigms, it demanded a distinct style of thinking about and expressing issues.



```
6
7   # calculate sine
8   sine(Degrees, Result) :-
9       convert_radians(Degrees, Radians),
10      Result is sin(Radians).
11
12  # calculate cosine
13  cosine(Degrees, Result) :-
14      convert_radians(Degrees, Radians),
15      Result is cos(Radians).
16
17  # calculate tangent
18  tangent(Degrees, Result) :-
19      convert_radians(Degrees, Radians),
20      Result is tan(Radians).
21
```

```
>_ Console
swipl main.pl
ERROR: /home/runner/logic/main.pl:1:
ERROR:    source_sink `library(math)' does not exist
Warning: /home/runner/logic/main.pl:1:
Warning:    Goal (directive) failed: user:use_module(library(math))
Welcome to SWI-Prolog (threaded, 64 bits, version 8.3.29)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software
.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- sine(45,Result).
Result = 0.7071067811865475.

?- cosine(45,Result).
Result = 0.7071067811865476.

?- tangent(45,Result).
Result = 0.9999999999999999.

?-
```

This code snippet defines sine/2, cosine/2, and tangent/2 as three predicates in our Prolog program. The first parameter of each predicate is an angle in degrees. Each predicate then computes the matching trigonometric value and stores it in the second argument. Radians = Degrees * pi / 180 is the formula used by the to_radians/2 predicate to convert degrees to radians.

CONCLUSION

I investigated four programming paradigms (**procedural**, **functional**, **object-oriented**, and **logic**) in the context of trigonometric computations in this case study. Each paradigm had advantages and disadvantages.

The procedural paradigm provided a simple and sequential method, but it lacked modularity. The functional paradigm emphasized immutability and expressiveness while adding complexity. The object-oriented paradigm provides modularity and code organization but at a cost. The logic paradigm was concerned with logical inference but demanded a different approach to problem-solving.

The appropriate paradigm is determined by project requirements and developer preferences. Code maintainability, reusability, performance, and learning curve are all factors to consider. Developers can improve their productivity and the efficiency of trigonometric calculations by knowing the consequences of each paradigm.