1 Explanation

Listed below are the equations which must be callable in your Python module. Please ensure that the methods can handle *both* integers and doubles, as while Python is casual about types, C++ is not. Pay special attention when receiving arrays, as they can contain both integers and doubles.

For trigonometric functions, assume that the input and output are in radians. Additionally, make sure to account for proper domain restrictions, no imaginary numbers!

If a function has both one and multiple arguments, use fun_name(*args). You can then use len(args) to figure out which method is used.

For exceptions, you are required to create a custom exception that extends the Exception class.

2 Equations to Implement

1. sc.sin(x)
2. sc.cos(x)
3. sc.tan(x)
4. sc.sinh(x)
5. sc.cosh(x)
6. sc.tanh(x)
7. sc.asin(x)
8. sc.acos(x)
9. sc.atan(x)
10. sc.exp(x)
11. sc.exp(x, y)
12. sc.root(x)
13. sc.root(x, y)
14. sc.log(x)
15. sc.log(x, y)
16. sc.mean(array)
17. sc.median(array)
18. sc.std_dev(array)
19. sc.min(array)
20. sc.max(array)

3 Select Implementation

This section provides the implementation of select functions. In the full project, all functions would be implemented. You will be implementing parts of the IBM Accurate Mathematical Library as defined in the GNU C Standard Library.

3.1 sc.sin()

The domain of this function is $\mathbb{R}$. You are implementing a slightly smaller version of the code in the IBM Accurate Mathematics Library.

If $|x| \leq \pi$, run $f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$. It's helpful to have predefined values of 1/3!, 1/5!, etc.

If $|x| \geq \pi$, keep subtracting or adding 2π until it is in this range. The full library has more explicit changes due to challenges with floating points and a larger want for precision, but this is enough for our demonstration.

3.2 sc.exp(*args)

3.2.1 One Argument

The domain of this function is $\mathbb{R}$. The equation you are implementing is $f(x) = e^x$. To do this, start with x and calculate r, where $x = k \ln 2 + r$ and k is the largest integer possible with $r \geq 0$. ln(2) can be a constant in your program found using an external calculator. Then, find $z = 1 + r + r^2(\frac{1}{2!} + r(\frac{1}{3!} + r(\frac{1}{4!} + r\frac{1}{5!})))$. z is an approximation of e^r. Finally, $f(x) = 2^k z$, which is easy to calculate as you are multiplying 2 by itself an integer number of times.

This is true as $f(x) = e^x = e^{k \ln 2 + r} = e^{k \ln 2} e^r = 2^k z$.

3.2.2 Two Arguments

The acceptable values of x are $x \geq 0$. However, special consideration must be made for y. If $y < 0$, ensure that $x \neq 0$ to not divide by zero. As log(0) is not defined, $x = 0$ should be a special case.

The equation you are implementing is $f(x, y) = x^y$ is $y \geq 0$ and $\frac{1}{x^{-y}}$ if $y < 0$. To implement this function, use $\exp(y \cdot \log(x))$ where log and exp are already implemented.

3.2.3 Errors

1. TooManyArgs (run if the number of arguments is not 1 or 2)
2. DomainException (run if value is not in the domain of function)

3.3 sc.mean

The equation you are implementing is $\frac{\sum_{i=1}^{N} v_i}{N}$.

3.4 Errors

1. EmptyArray (run if the array has no elements)
2. NonNumeric (run if the array has a non numeric element)
3. NonArray (run if the element is not an array)

3.5 sc.std_dev

The equation you are implementing is $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$ where N is the number of elements, x_i is the ith element, and μ is the mean It might be a good idea to use the other equations you have implemented for this.

3.6 Errors

1. EmptyArray (run if the array has no elements)
2. NonNumeric (run if the array has a non numeric element)
3. NonArray (run if the element is not an array)

4 Notes

In a fully fleshed document, this would be explained in the Implementation section. For sc.root(x), $f(x) = \sqrt{x}$. and for sc.root(x, y), $f(x, y) = (x)^{\frac{1}{y}}$. For sc.log(x) $f(x) = \ln(x)$ and for sc.log(x, y), $f(x, y) = \log_x(y)$.

I would have implemented all of them, but I do not want to cry myself to sleep trying to understand sysdeps/ieee754/dbl-64/s_sin.c and the fun it's having with floating points. All I can muster is good job IBM, y'all and Bell Labs somehow just built the foundation of computing in your spare time.