Part3

Zhicheng Fu

Bug description

My library focus on equation interpreting. A given equation, like “w + sin(x1+3)”, can be interpreted as an equation. Later we can pass parameters to the equation. Then there will be the solutions. That means the equations can be evaluated. Until now I only test the interpreting equation part. Here is an example for the input rules.

```
>>> from Equation import Expression

>>> fn = Expression("sin(x+y^2)",["y", "x"])

>>> fn

sin ((x + (y ^ (2+0j))))

>>> print fn

\sin\left(\left(x + y^{(2+0j)}\right)\right)
```

A very important hint is that the Expression function should not return a NONE or “” unless the input is “” (empty string). If it is correct, it should return the equations. If here is something that cannot be understood, it should return TypeError.

Until now I found at least one kind of bugs. The bug is if the function meet something unexpected, it will return NULL, which will be print as “” by the print function. For example if I call the function as fn = Expression (“sin (1+x1), x1”,[“x1”]), then print fn. I can get a output of NONE. That means fn get an empty string. Neither is fn the part before unexpected string Nor it return a TypeError. In the other word, as long as the function meet an unexpected string or character, it will return a NONE without any TypeError prompt.

Progress to date

At the very beginning, I only test some fixed very basic case. For example, I test something like Expression (“1+1”) then print the output. Later I improve that. I generate some arbitrary integer number to print out. Then I begin to generate real number, after that I create imaginary number by adding “j” after a real number. Combine real number and imaginary number I got complex number.

The nest step of my testing TSTL file is adding operators in the expression functions. Firstly, I add some basic operators like “+”, “*”. Then I print the expression to see if the library can handle these kind of expressions. Also I add some basic operator functions, like sin, cos. And the parameters of these functions are also expressions which are generated by the testing system.

What’s more, I evaluate the expressions then. So that I can check if the library can evaluate the expressions correctly.

Estimate my progress

I will improve my test TSTL file to test more cases, some of these cases are general cases, some are edge testing. I will test more to see if there are more hugs that can be found. Also I will try to do coverage analysis for the sut.py.

The quality of the SUT

The library I test on has some bugs. But I think it still a good library.

The good part of the Equation library is that the idea of the library is good. Interpreting an expression is fascinating. It can be very useful in some applications. What’s more, the library can hold complex number to evaluate the expression. That is useful for some specific projects like image filter.

There are some problems of the Equation library. As I discussed above, the library cannot figure out some illegal input, like “,”, “#”. Also when you input an expression, it wouldn’t check if the number of the parameters I input is the same as the expression need. This problems should be find out at the step of interpreting the expression but not at the step of evaluating the expression.

All in all, the Equation library that I test is fascinating but has some bugs.

Coverage

The file “cover.tstl” is for general testing and getting the coverage. The file “bug.tstl” is for testing the bug.

In order to use coverage analysis in TSTL, I type the following commands:

```
tstl cover.tstl
```

```
python randomtester.py --timeout=5 --depth=30
```

Then it will use coverage analysis to analysis the testing. The coverage percent is 41.558.