Profiling of mathlib.py using cProfile together with SnakeViz by solving a corrected sample standard deviation equation

Test 1:

Processor: Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz

OS: Ubuntu 20.04

Profiler: cProfile (no addition)

Sample: 10 numbers ranging from 1 to 1000

Test execution:

```
ondrej@OndrejovaMasina:~/Documents/VUT/IVS/Projekt2/floor11_calc/floor11_calc/src$ python3 -m cProfile  profiling.py <10_num.txt
214.05388729632224
         934 function calls (901 primitive calls) in 0.001 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:103(release)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:143(__init__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:147(__enter__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:151(__exit__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:157(_get_module_lock)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:176(cb)
     11/1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap>:211(_call_with_frames_removed)
       45    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:222(_verbose_message)
        4    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:232(_requires_builtin_wrapper)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:342(__init__)
        3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:35(_new_module)
        6    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:376(cached)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:389(parent)
```

mathlib.py records:

```
20    0.000    0.000    0.000    0.000 mathlib.py:12(add)
 2    0.000    0.000    0.000    0.000 mathlib.py:17(sub)
 3    0.000    0.000    0.000    0.000 mathlib.py:22(mul)
 2    0.000    0.000    0.000    0.000 mathlib.py:28(div)
11    0.000    0.000    0.000    0.000 mathlib.py:55(exp)
 1    0.000    0.000    0.001    0.001 mathlib.py:7(<module>)
 1    0.000    0.000    0.000    0.000 mathlib.py:70(root)
```

Total runtime: **1ms**

Test conclusion: The time it takes to complete the fucntions is so insignificant, it isn't even registered and shown by cProfile. The program spends most of runtime by importing libraries.

Test 2:

Processor: Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz

OS: Ubuntu 20.04

Profiler: cProfile (no addition)

Sample: 100 numbers ranging from 1 to 1000

Test execution:

```
ondrej@OndrejovaMasina:~/Documents/VUT/IVS/Projekt2/floor11_calc/floor11_calc/src$ python3 -m cProfile  profiling.py <100_num.txt
285.90686394083804
         1298 function calls (1265 primitive calls) in 0.001 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:103(release)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:143(__init__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:147(__enter__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:151(__exit__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:157(_get_module_lock)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:176(cb)
     11/1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap>:211(_call_with_frames_removed)
       45    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:222(_verbose_message)
        4    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:232(_requires_builtin_wrapper)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:342(__init__)
        3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:35(_new_module)
        6    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:376(cached)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:389(parent)
```

mathlib.py records:

```
200    0.000    0.000    0.000    0.000 mathlib.py:12(add)
  2    0.000    0.000    0.000    0.000 mathlib.py:17(sub)
  3    0.000    0.000    0.000    0.000 mathlib.py:22(mul)
  2    0.000    0.000    0.000    0.000 mathlib.py:28(div)
101    0.000    0.000    0.000    0.000 mathlib.py:55(exp)
  1    0.000    0.000    0.001    0.001 mathlib.py:7(<module>)
  1    0.000    0.000    0.000    0.000 mathlib.py:70(root)
```

Total runtime: **1ms**

Test conclusion: Just like in test no. 1, the time it takes to complete the functions is insignificant and the program spends most of runtime by importing libraries.

Test 3:

Processor: Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz

OS: Ubuntu 20.04

Profiler: cProfile (no addition)

Sample: 1000 numbers ranging from 1 to 1000

Test execution:

```
ondrej@OndrejovaMasina:~/Documents/VUT/IVS/Projekt2/floor11_calc/floor11_calc/src$ python3 -m cProfile  profiling.py <1000_num.txt
289.09381735906294
         4937 function calls (4904 primitive calls) in 0.003 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:103(release)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:143(__init__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:147(__enter__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:151(__exit__)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:157(_get_module_lock)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:176(cb)
     11/1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap>:211(_call_with_frames_removed)
       45    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:222(_verbose_message)
        4    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:232(_requires_builtin_wrapper)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:342(__init__)
        3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:35(_new_module)
        6    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:376(cached)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:389(parent)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:397(has_location)
        4    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:406(spec_from_loader)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:477(_init_module_attrs)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:549(module_from_spec)
        7    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:58(__init__)
```

mathlib.py records:

```
 2000    0.000    0.000    0.000    0.000 mathlib.py:12(add)
    2    0.000    0.000    0.000    0.000 mathlib.py:17(sub)
    3    0.000    0.000    0.000    0.000 mathlib.py:22(mul)
    2    0.000    0.000    0.000    0.000 mathlib.py:28(div)
 1001    0.000    0.000    0.000    0.000 mathlib.py:55(exp)
    1    0.000    0.000    0.001    0.001 mathlib.py:7(<module>)
    1    0.000    0.000    0.000    0.000 mathlib.py:70(root)
```

Total runtime: **2-3ms**

Test conclusion: Even after 2000 calls of the same function, the time it takes to execute it is simply too small. Although in the instance of 1000 numbers we see an increase in the overall execution time by 1-2 ms, this is caused simply by loading the 1000 numbers into an array for further calculations and by the profiler itself measuring the times it takes to execute functions.

Since cProfile by itself didn’t provide sufficient information for optimization, we decided to run the program on a different machine together with **SnakeViz**. This allowed us to get more precise results, visualize the information better and use profile guided optimization to increase the code’s efficiency. This was done by xtalaj00 who researched how to improve the efficiency of python code and applied those changes to both the math library and the profiling.py script. The tests are shown below.

Test 4:

Processor: AMD Ryzen 5 5600X 6-Core Processor 3.70GHz

OS: Windows 10

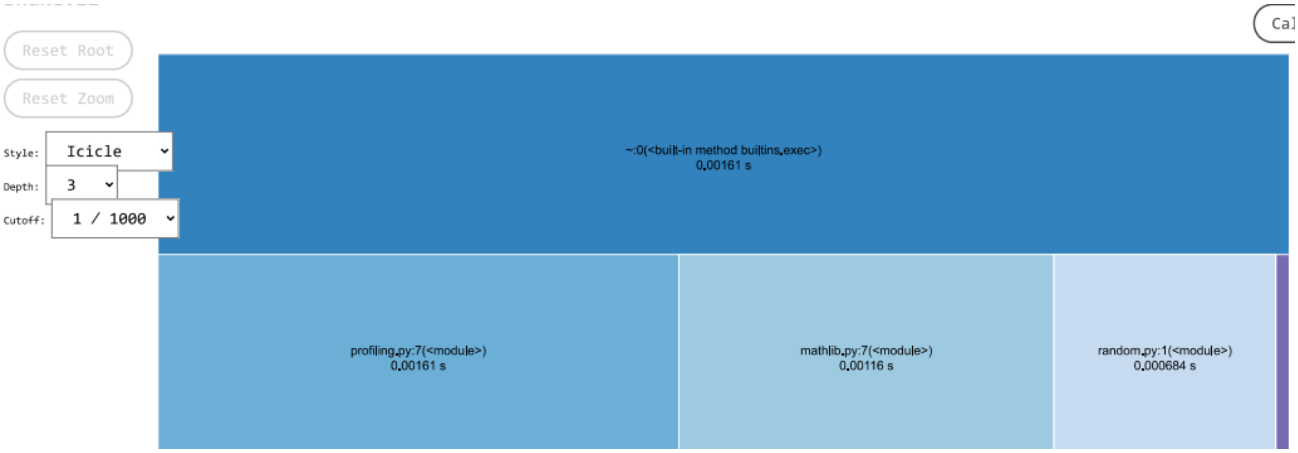
Profiler: cProfile + SnakeViz

Sample: 10 numbers ranging from 1 to 1000

Test execution:

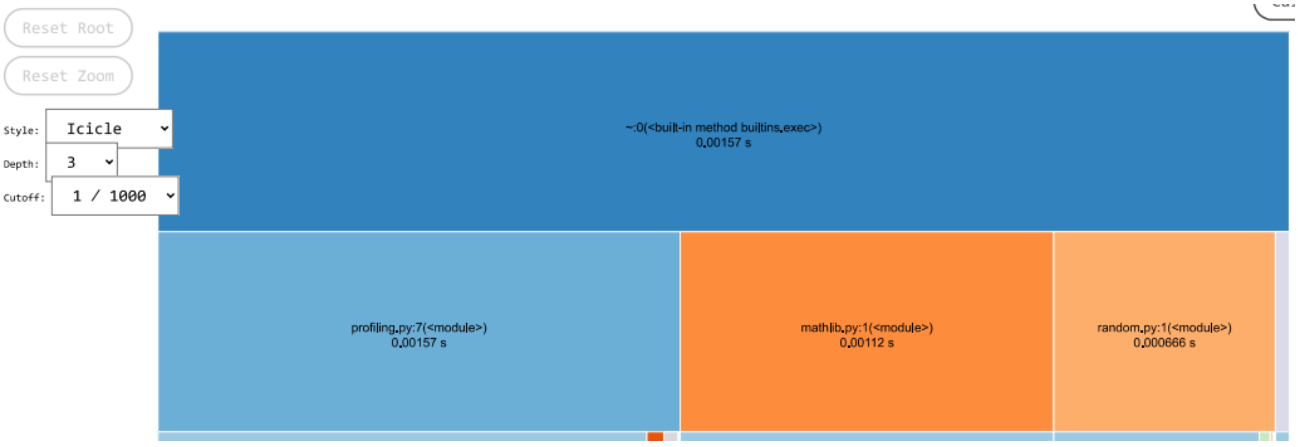
```
C:\Users\Martin T\Desktop\GIT\xtalaj00_xkovac57\repo\floor11_calc-main -OLD\src>python -m cProfile -o p1-1.prof profiling.py < 10_num.txt
214.05388729632224
```

Test output (**before optimization**):



Total runtime: **1.61 ms**

Test output (**after optimization**):



Total runtime: **1.57 ms**

Conclusion: Optimization of the profiling.py script and the mathlib.py library decreased total runtime by **0.04 ms** or **2.5%** in the sample of 10 numbers.

Test 5:

Processor: AMD Ryzen 5 5600X 6-Core Processor 3.70GHz

OS: Windows 10

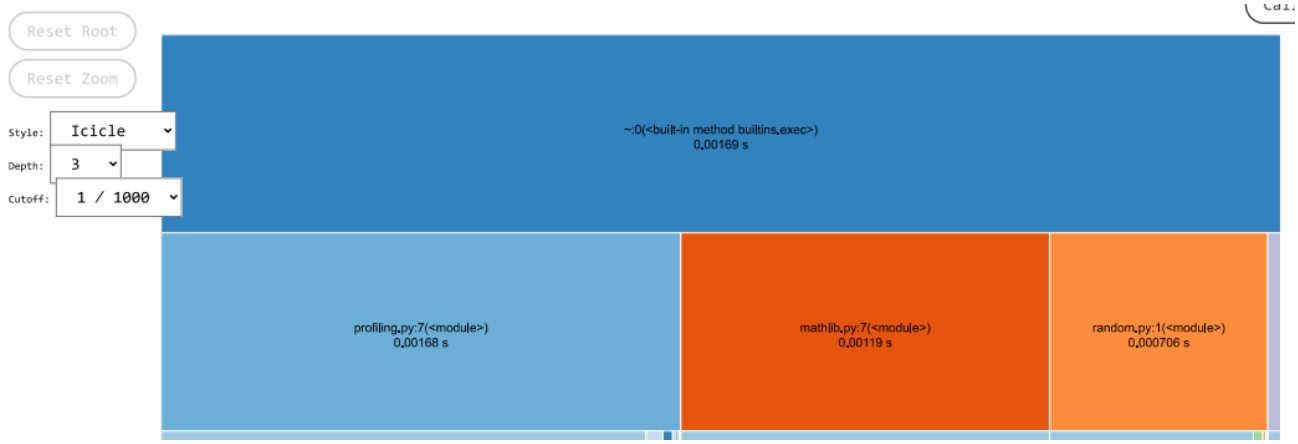
Profiler: cProfile + SnakeViz

Sample: 100 numbers ranging from 1 to 1000

Test execution:

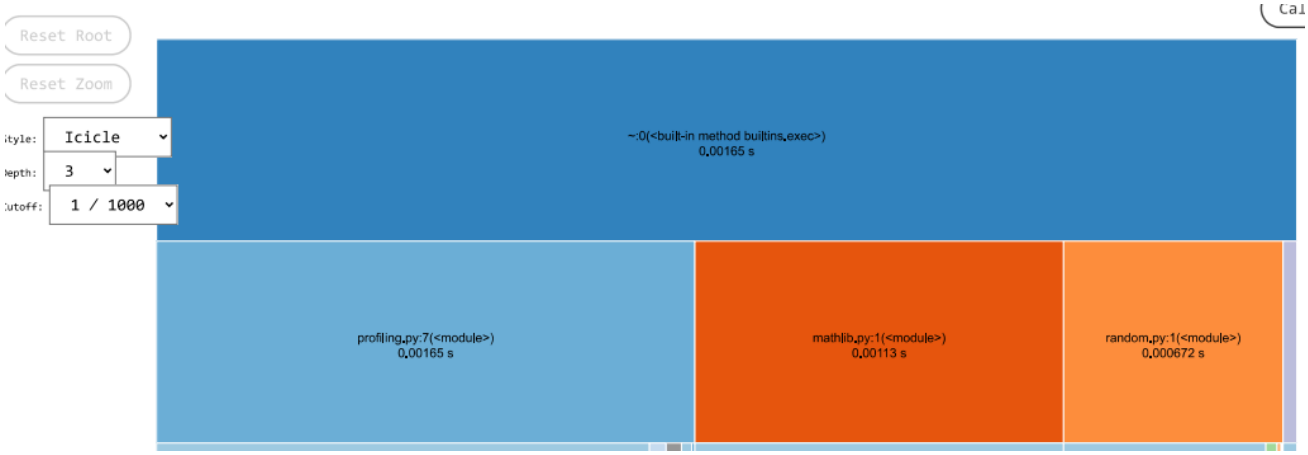
```
C:\Users\Martin T\Desktop\GIT\xtalaj00_xkovac57\repo\floor11_calc-main -OLD\src>python -m cProfile -o p1-2.prof profiling.py < 100_num.txt
285.90686394083804
```

Test output (**before optimization**):



Total runtime: **1.69 ms**

Test output (**after optimization**):



Total runtime: **1.65 ms**

Conclusion: Optimization of the profiling.py script and the mathlib.py library decreased total runtime by **0.04 ms** or **2.4%** in the sample of 100 numbers.

Test 6:

Processor: AMD Ryzen 5 5600X 6-Core Processor 3.70GHz

OS: Windows 10

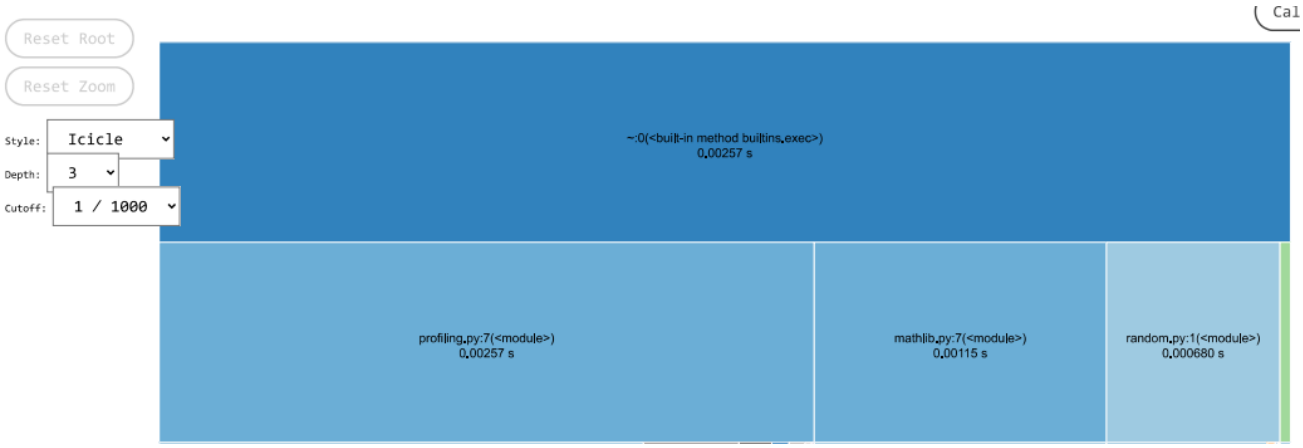
Profiler: cProfile + SnakeViz

Sample: 1000 numbers ranging from 1 to 1000

Test execution:

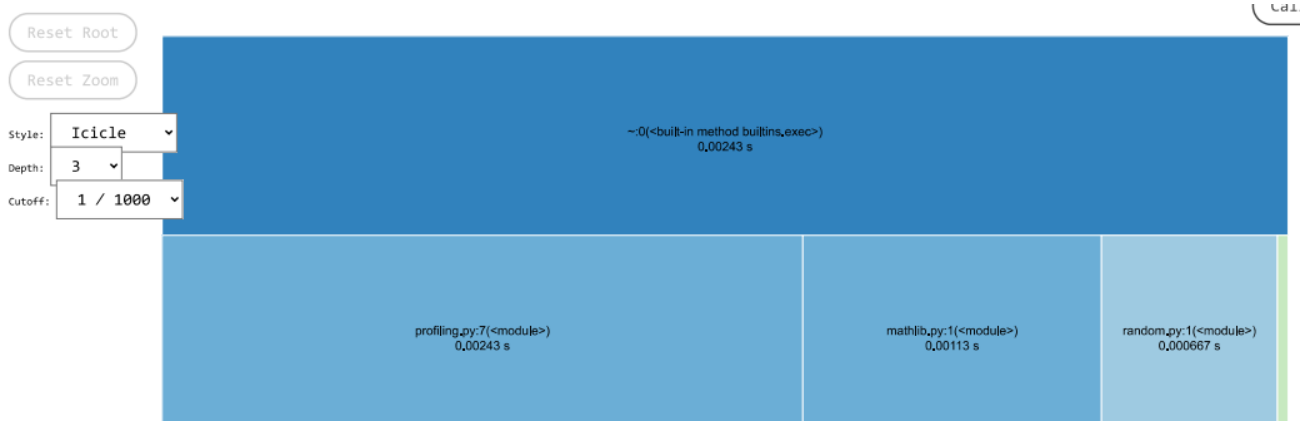
```
C:\Users\Martin T\Desktop\GIT\xtalaj00_xkovac57\repo\floor11_calc-main -OLD\src>python -m cProfile -o p1-3.prof profiling.py < 1000_num.txt
289.09381735906294
```

Test output (**before optimization**):



Total runtime: **2.57 ms**

Test output (**after optimization**):



Total runtime: **2.43 ms**

Conclusion: Optimization of the profiling.py script and the mathlib.py library decreased total runtime by **0.14 ms** or **5.5%** in the sample of 1000 numbers.

Summary:

After these tests we concluded, that the mathlib.py functions are pretty well optimized. The time it takes to execute them is insignificant for a simple calculator, in which we will hardly see over 100 uses of the same function in one run, let alone a few thousands. However even the increase of a few miliseconds is useful, should the math library be used for larger calculations. Still most of the overhead in these cases is caused by the profiler and the need to call the functions themselves.