Problem 1: Calculate the machine epsilon of your computer using the machine epsilon function you are required to write.

To solve this problem, I utilized the program “*machine\_epsilon.py*” that I wrote for estimating my machine’s machine epsilon. The program divides a number one by two many times until the difference between the results is undetectable by the machine due to precision limitations of the machine’s binary representation of floating-point values. The Figure 1 below shows output of my machine’s program.

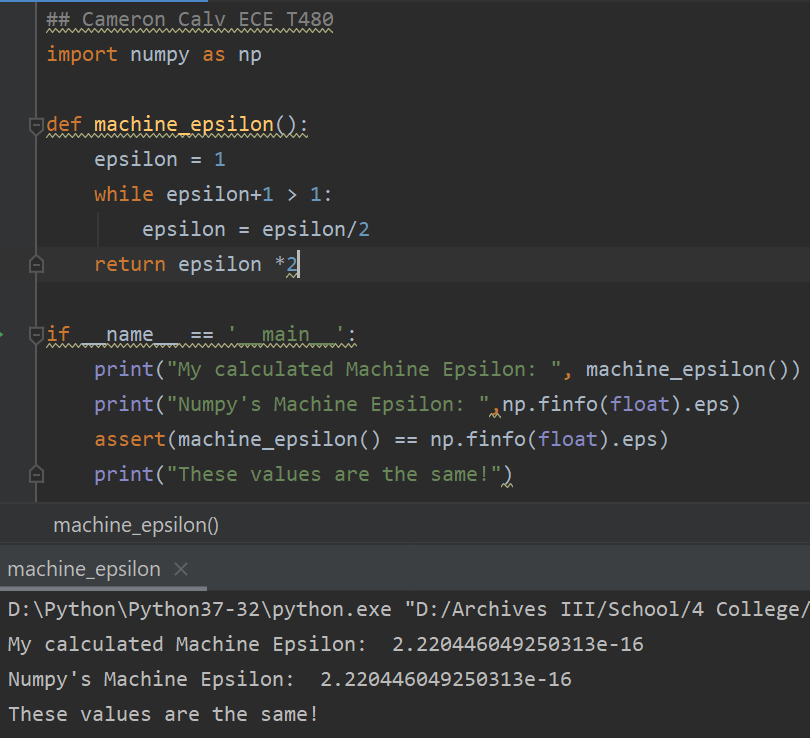


Figure : Calculating machine epsilon of my system.

According to the program (which is assert-checked with *numpy*’s *finfo* property) my machine epsilon is **2.220446049250313e-16*.***

Problem 2: The infinite series f(n) = Xn i=1 1 i 4 1 converges on a value of f(n) = π 4 90 as n approaches infinity. Write a program in single precision to calculate f(n) for n=10,000 by computing the sum from i=1 to 10,000. Then repeat the calculation but in reverse order-that is, from i=10,000 to 1 using increments of -1. In each case, compute the true percent relative error. Explain the results.

Within the file *problems\_hw1.py* problem 2 is worked out. A snippet of the code and the output is shown below in Figure 2.

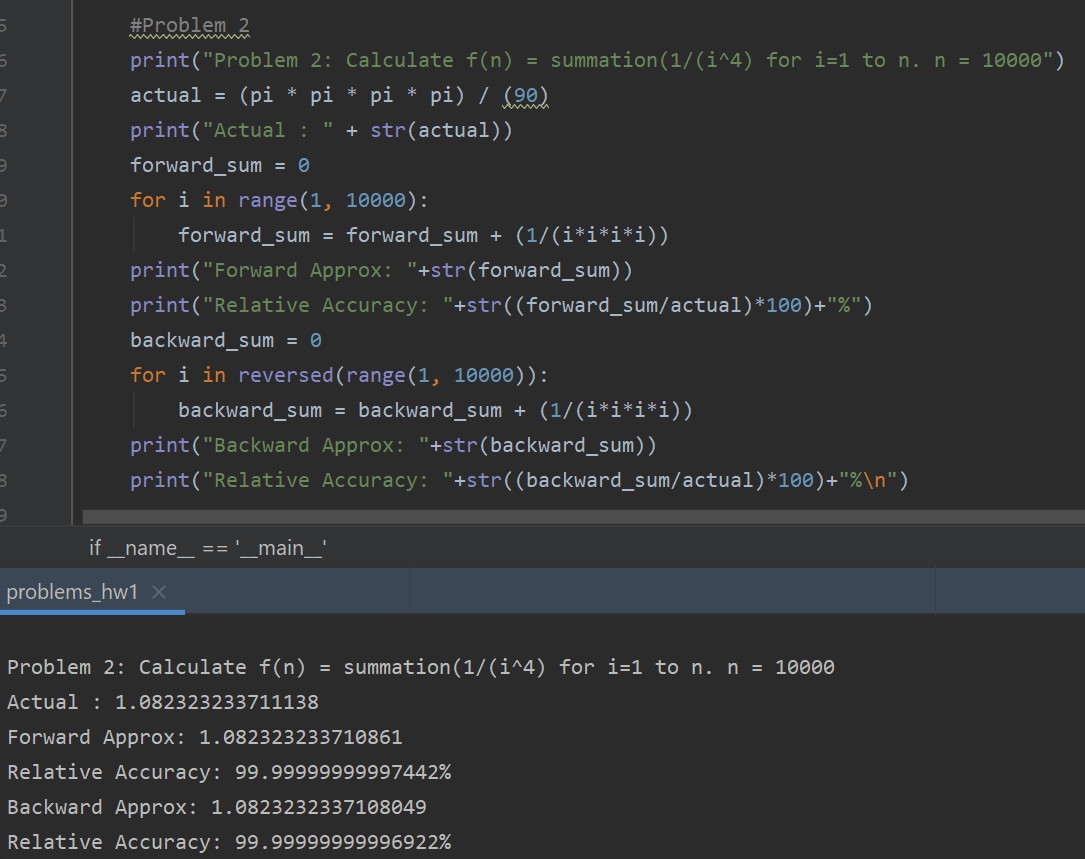


Figure : Code and output showing the solution for Problem 2.

According to the output of the code the relative error for both approximations are **almost 99.999% accurate with the forward approximation being slightly better than the backwards approximation**. This is most likely the case because the there is truncation when dealing with the higher *i* numbers as the summation continues. This truncation makes adding backwards and forwards different when done on a limited accuracy machine.

Problem 3: Evaluate e −5 using two approaches: e −x = 1 − x + x 2 2 − x 3 3! + ... and e −x = 1 1 + x + x 2 2 + x 3 3! + ... and compare with the true value of 6.737947 × 10−3 . Use 20 terms to evaluate each series and compute true and approximate relative errors as terms are added. Use the fact function you are required to write to calculate the factorials needed.

Within the file *problems\_hw1.py* problem 3 is worked out. A snippet of the code and the output is shown below in Figure 3.

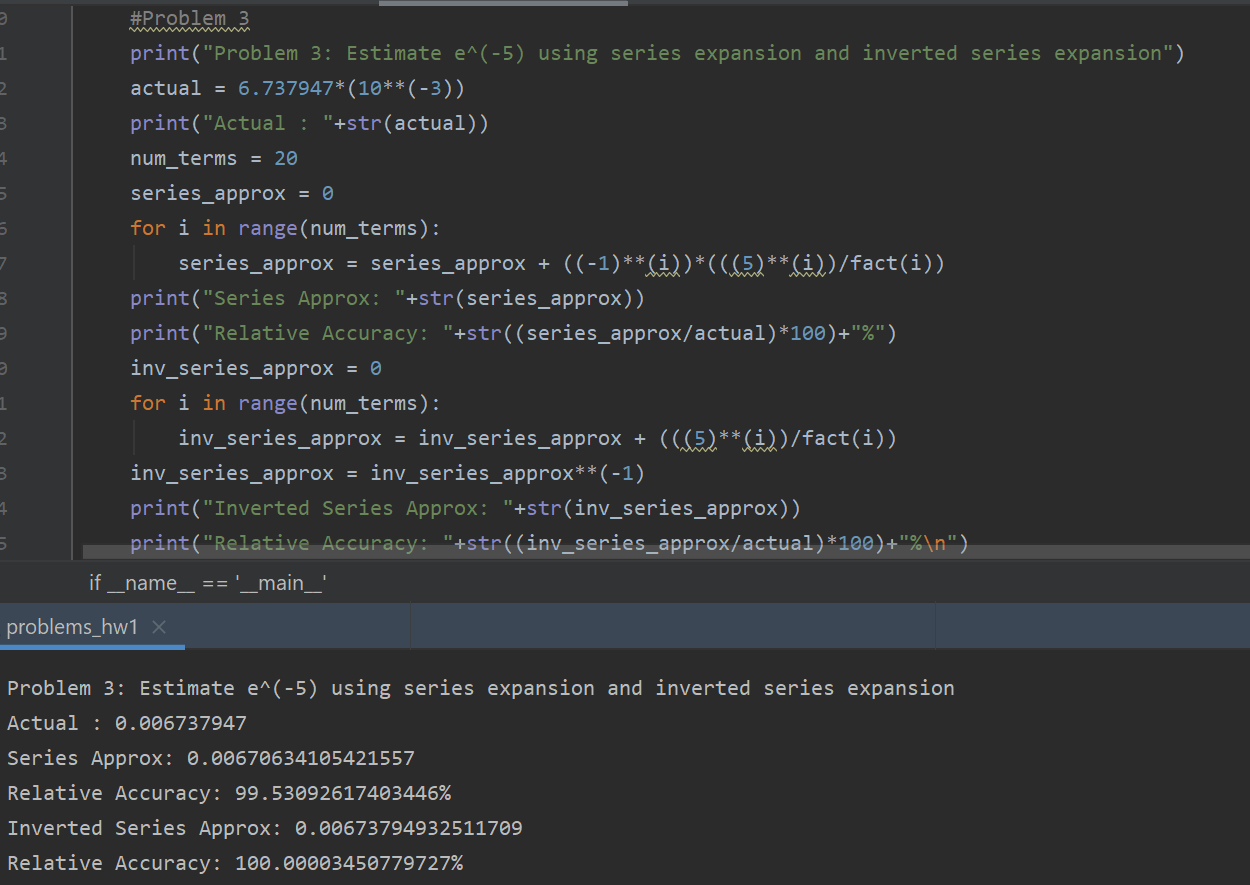


Figure : Code and output for Problem 3.

As shown in the figure, the accuracy of the **first approximation is about 99.53% and the second approximation is about 100.00%.**

Problem 4: The Maclaurin series expansion for cos x is cos x = 1 − x 2 2 + x 4 4! − x 6 6! + x 8 8! − ... Starting with the simplest version, cos x = 1, add terms one at a time to estimate cos(π/3). After each new term is added, compute the true and approximate percent relative errors. Use your pocket calculator to determine the true value. Add terms until the absolute value of the approximate error estimate falls below an error criterion conforming to two significant figures. Use the fact function you are required to write to calculate the factorials needed.

Within the file *problems\_hw1.py* problem 4 is worked out. A snippet of the code and the output is shown below in Figure 4Figure 3.

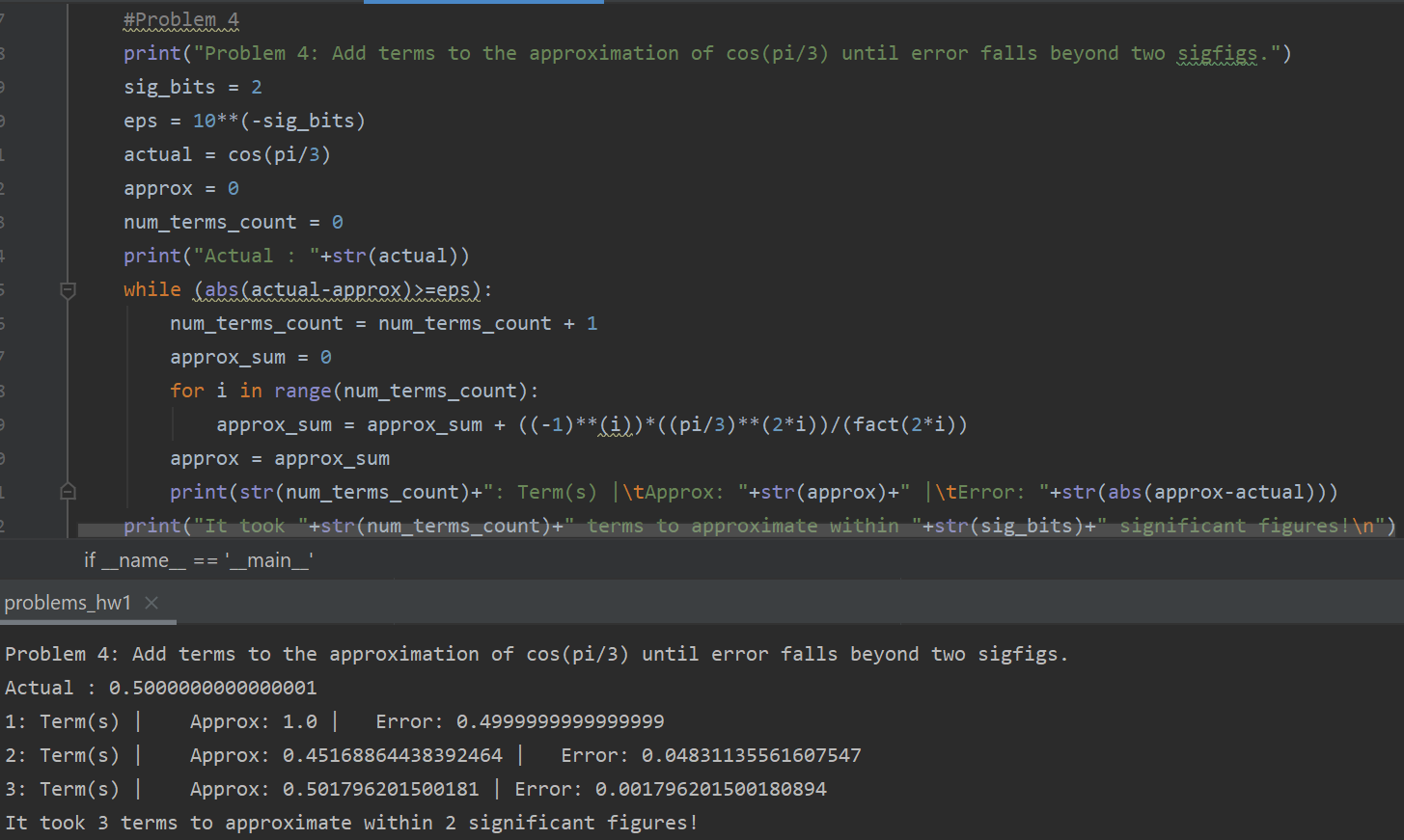


Figure : Code and snippet of output for Problem 4.

It is shown that it takes **three terms** to get the answer precise enough to be within two significant figures of accuracy.

Problem 5: Consider the function f(x) = x 3 − 2x + 4 on the interval [-2,2] with h = 0.25. Use the forward, backward, and centered finite difference approximations for the first and second derivatives so as to graphically illustrate which approximation is most accurate. Graph all three first derivative finite difference approximation along with the theoretical, and do the same for the second derivative as well. Use the deriv1 and deriv2 functions you are required to make to calculate the derivatives.

Within the file *problems\_hw1.py* problem 4 is worked out. The graphs asked for by the problem are shown for in \_\_ and \_\_ respectively.

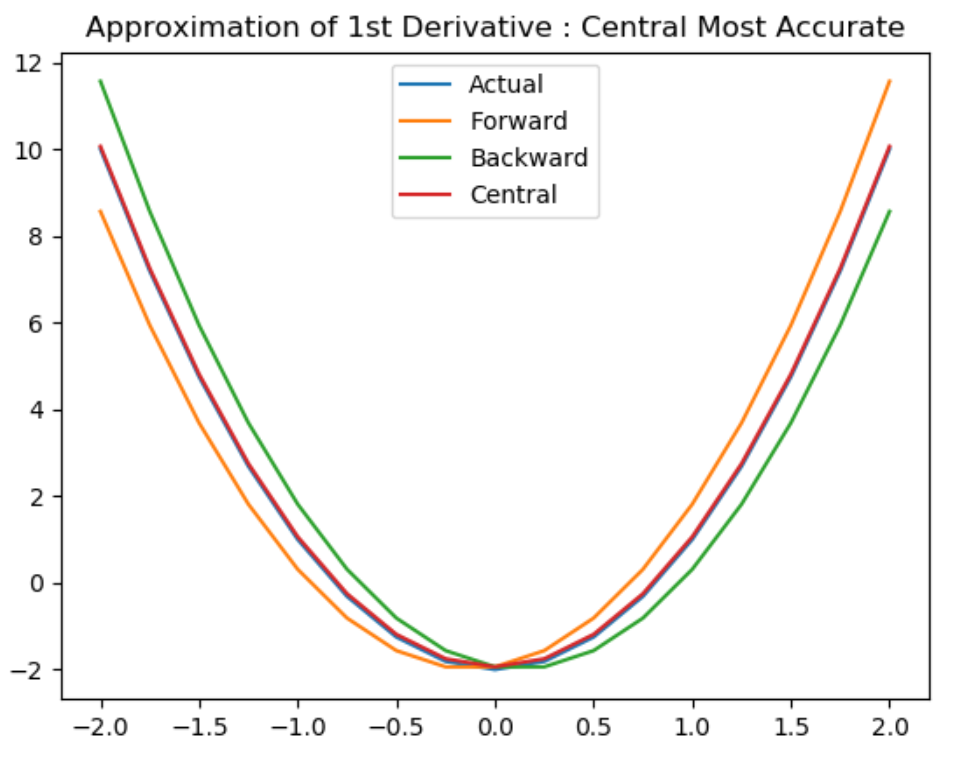


Figure : Plot showing comparison of first derivative approximations.

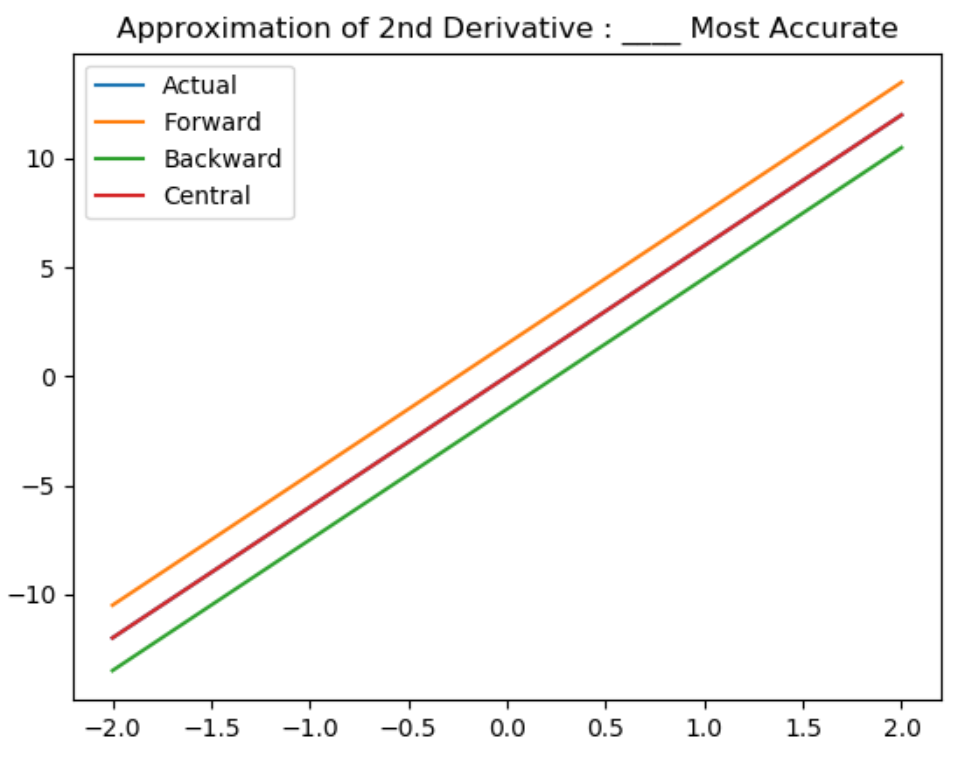


Figure : Plot showing comparison of second derivative approximations.