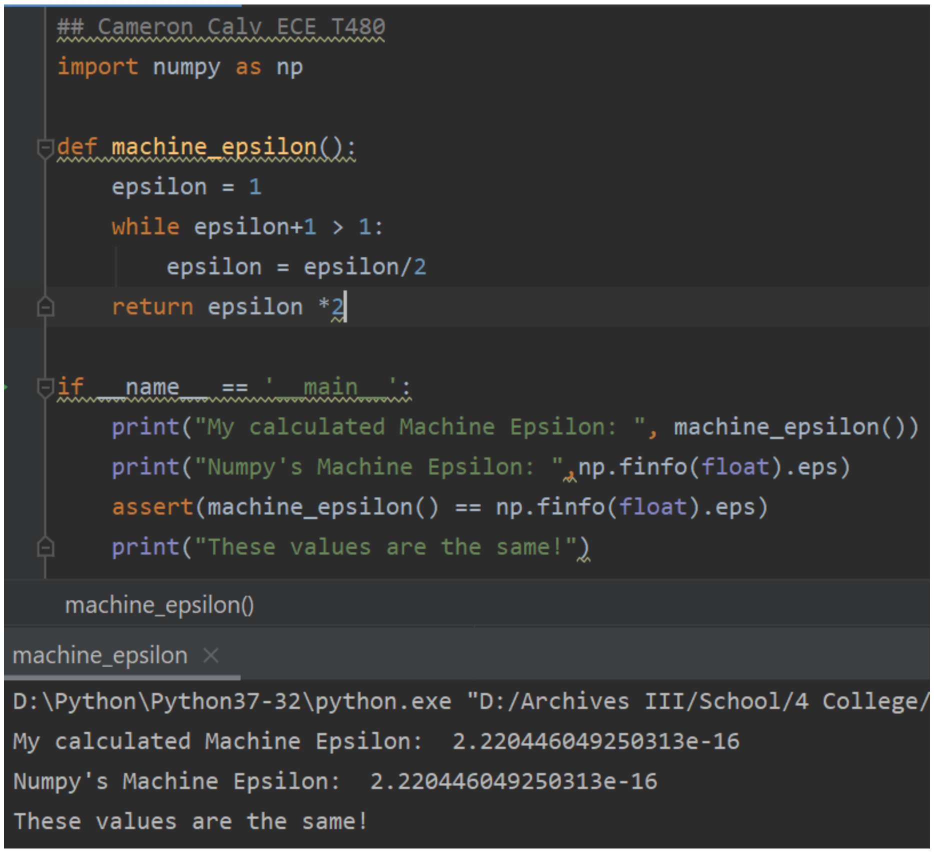
Cameron Calv
ECE T-480 Numerical Methods
Homework Problems 1

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.



```python
## Cameron Calv ECE T480
import numpy as np

def machine_epsilon():
    epsilon = 1
    while epsilon+1 > 1:
        epsilon = epsilon/2
    return epsilon *2

if __name__ == '__main__':
    print("My calculated Machine Epsilon: ", machine_epsilon())
    print("Numpy's Machine Epsilon: ",np.finfo(float).eps)
    assert(machine_epsilon() == np.finfo(float).eps)
    print("These values are the same!")
```

```
D:\Python\Python37-32\python.exe "D:/Archives III/School/4 College/
My calculated Machine Epsilon:  2.220446049250313e-16
Numpy's Machine Epsilon:  2.220446049250313e-16
These values are the same!
```

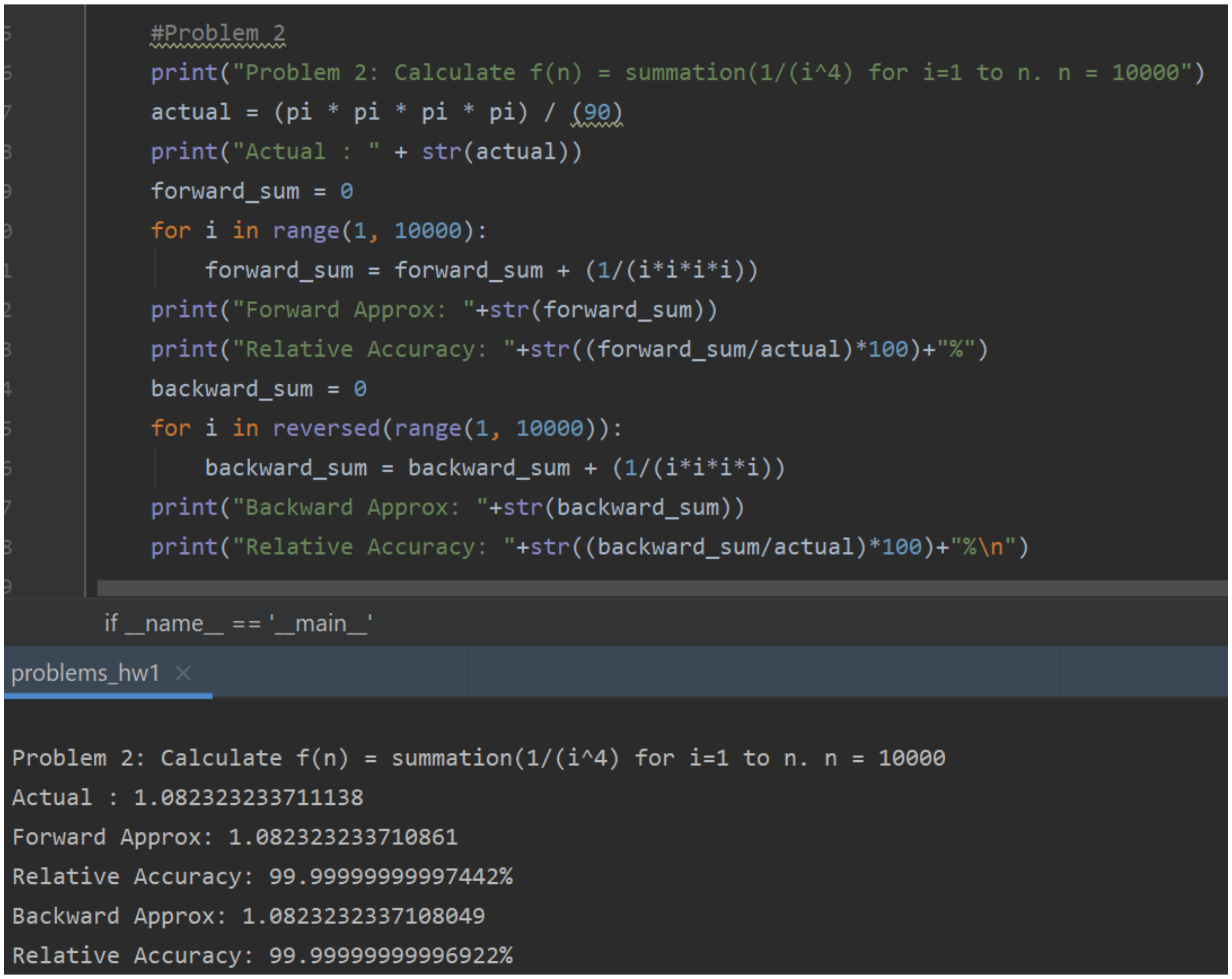
Figure 1: Calculating machine epsilon of my system.

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

Cameron Calv
ECE T-480 Numerical Methods
Homework Problems 1

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.



```
#Problem 2
print("Problem 2: Calculate f(n) = summation(1/(i^4) for i=1 to n. n = 10000")
actual = (pi * pi * pi * pi) / (90)
print("Actual : " + str(actual))
forward_sum = 0
for i in range(1, 10000):
    forward_sum = forward_sum + (1/(i*i*i*i))
print("Forward Approx: "+str(forward_sum))
print("Relative Accuracy: "+str((forward_sum/actual)*100)+"%")
backward_sum = 0
for i in reversed(range(1, 10000)):
    backward_sum = backward_sum + (1/(i*i*i*i))
print("Backward Approx: "+str(backward_sum))
print("Relative Accuracy: "+str((backward_sum/actual)*100)+"%\n")
```

```
Problem 2: Calculate f(n) = summation(1/(i^4) for i=1 to n. n = 10000
Actual : 1.082323233711138
Forward Approx: 1.082323233710861
Relative Accuracy: 99.99999999997442%
Backward Approx: 1.0823232337108049
Relative Accuracy: 99.99999999996922%
```

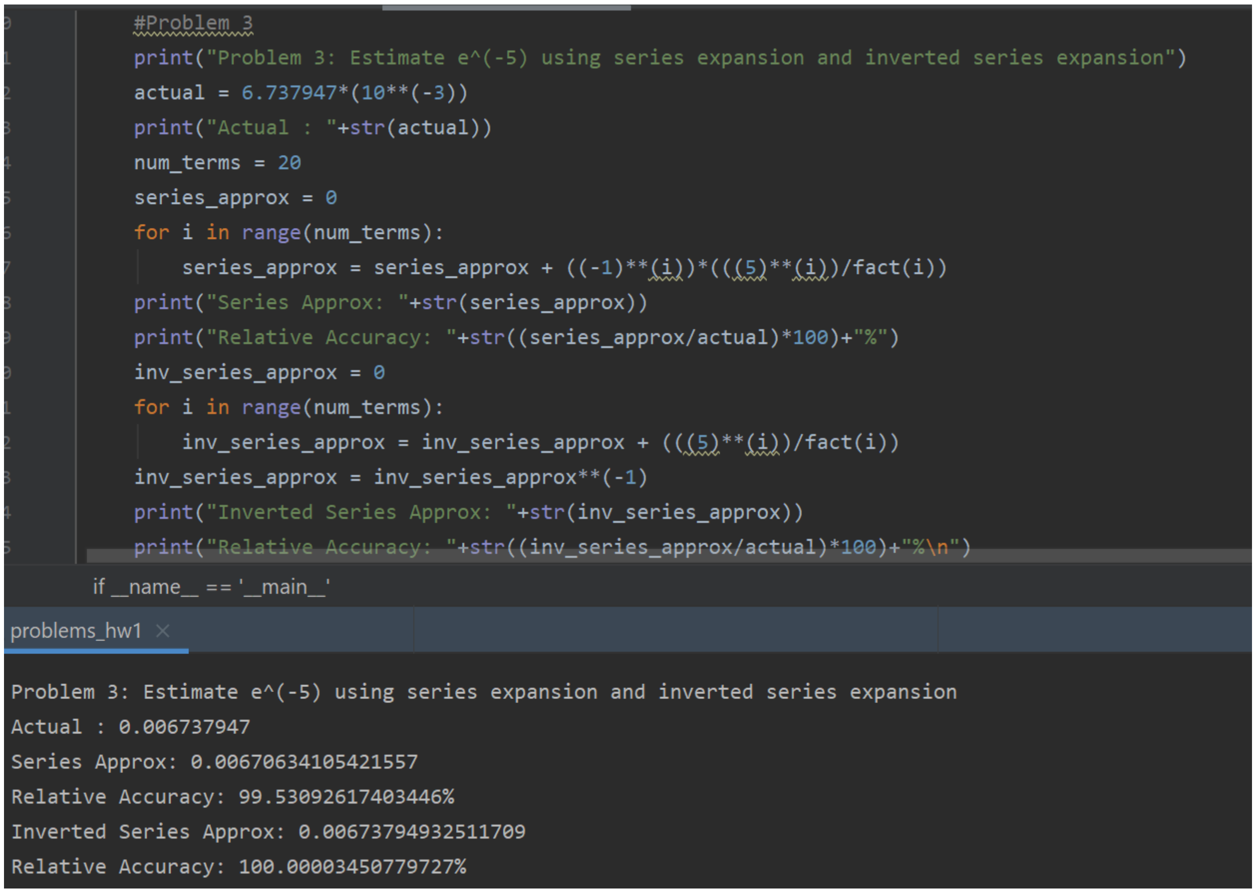
Figure 2: Code and output showing the solution for Problem 2.

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.

Cameron Calv
ECE T-480 Numerical Methods
Homework Problems 1

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.



```
    #Problem 3
    print("Problem 3: Estimate e^(-5) using series expansion and inverted series expansion")
    actual = 6.737947*(10**(-3))
    print("Actual : "+str(actual))
    num_terms = 20
    series_approx = 0
    for i in range(num_terms):
        series_approx = series_approx + ((-1)**(i))*(((5)**(i))/fact(i))
    print("Series Approx: "+str(series_approx))
    print("Relative Accuracy: "+str((series_approx/actual)*100)+"%")
    inv_series_approx = 0
    for i in range(num_terms):
        inv_series_approx = inv_series_approx + (((5)**(i))/fact(i))
    inv_series_approx = inv_series_approx**(-1)
    print("Inverted Series Approx: "+str(inv_series_approx))
    print("Relative Accuracy: "+str((inv_series_approx/actual)*100)+"%\n")
```

```
Problem 3: Estimate e^(-5) using series expansion and inverted series expansion
Actual : 0.006737947
Series Approx: 0.00670634105421557
Relative Accuracy: 99.53092617403446%
Inverted Series Approx: 0.006737949325117O9
Relative Accuracy: 100.00003450779727%
```

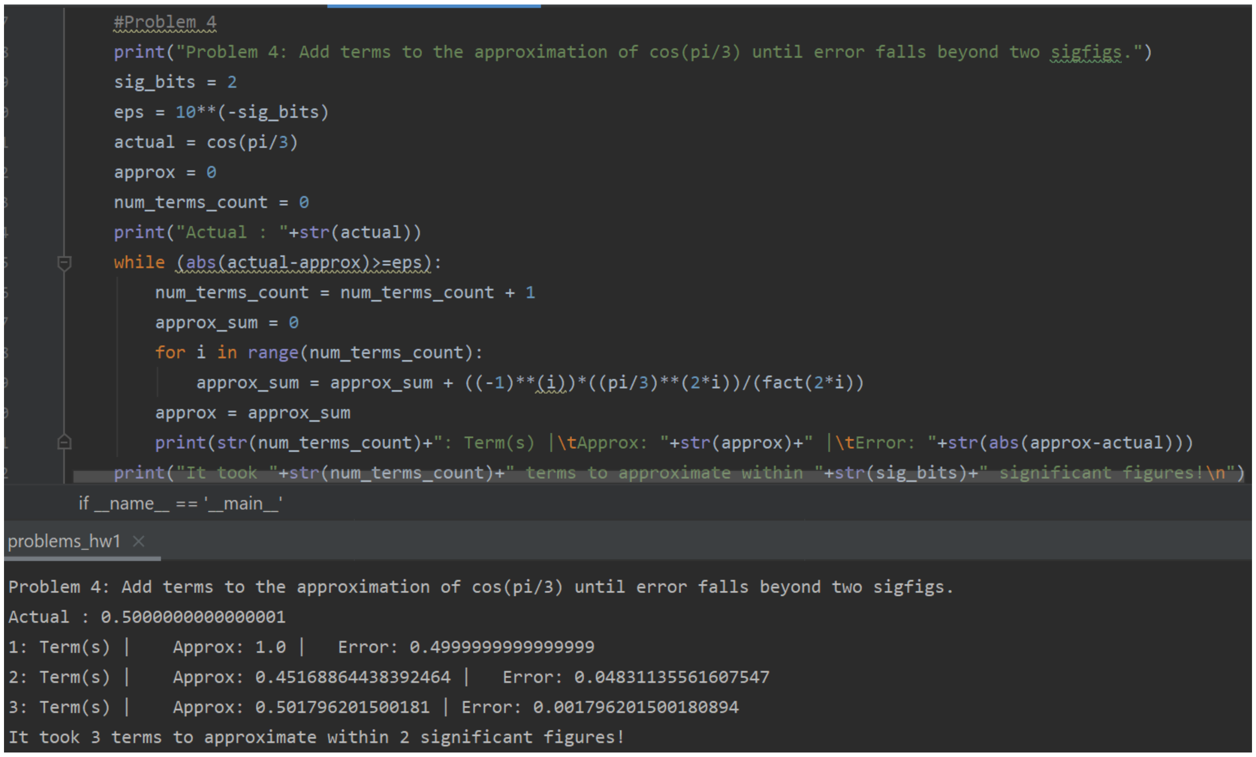
Figure 3: Code and output for Problem 3.

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

Cameron Calv
ECE T-480 Numerical Methods
Homework Problems 1

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.



```
#Problem 4
print("Problem 4: Add terms to the approximation of cos(pi/3) until error falls beyond two sigfigs.")
sig_bits = 2
eps = 10**(-sig_bits)
actual = cos(pi/3)
approx = 0
num_terms_count = 0
print("Actual : "+str(actual))
while (abs(actual-approx)>=eps):
    num_terms_count = num_terms_count + 1
    approx_sum = 0
    for i in range(num_terms_count):
        approx_sum = approx_sum + ((-1)**(i))*((pi/3)**(2*i))/(fact(2*i))
    approx = approx_sum
    print(str(num_terms_count)+": Term(s) |\tApprox: "+str(approx)+" |\tError: "+str(abs(approx-actual)))
print("It took "+str(num_terms_count)+" terms to approximate within "+str(sig_bits)+" significant figures!\n")
```

if __name__ == '__main__'

problems_hw1

```
Problem 4: Add terms to the approximation of cos(pi/3) until error falls beyond two sigfigs.
Actual : 0.5000000000000001
1: Term(s) |    Approx: 1.0 |   Error: 0.4999999999999999
2: Term(s) |    Approx: 0.45168864438392464 |   Error: 0.04831135561607547
3: Term(s) |    Approx: 0.501796201500181 | Error: 0.001796201500180894
It took 3 terms to approximate within 2 significant figures!
```

Figure 4: 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.

Cameron Calv
ECE T-480 Numerical Methods
Homework Problems 1

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.

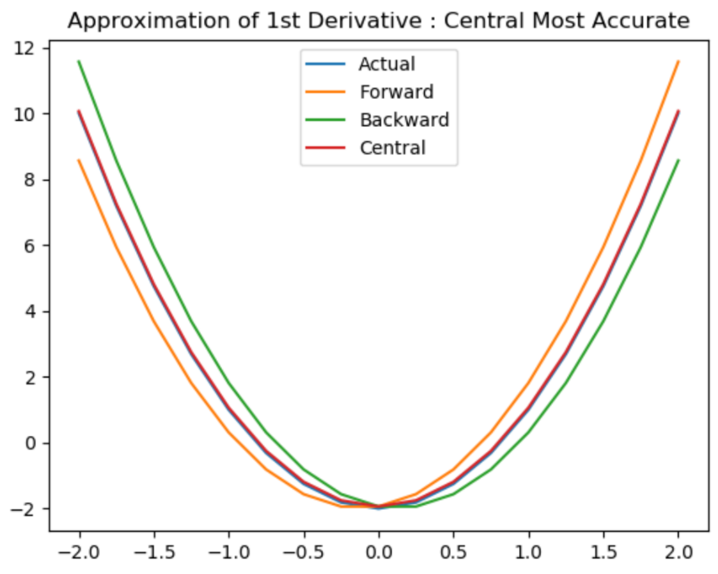


Figure 5: Plot showing comparison of first derivative approximations.

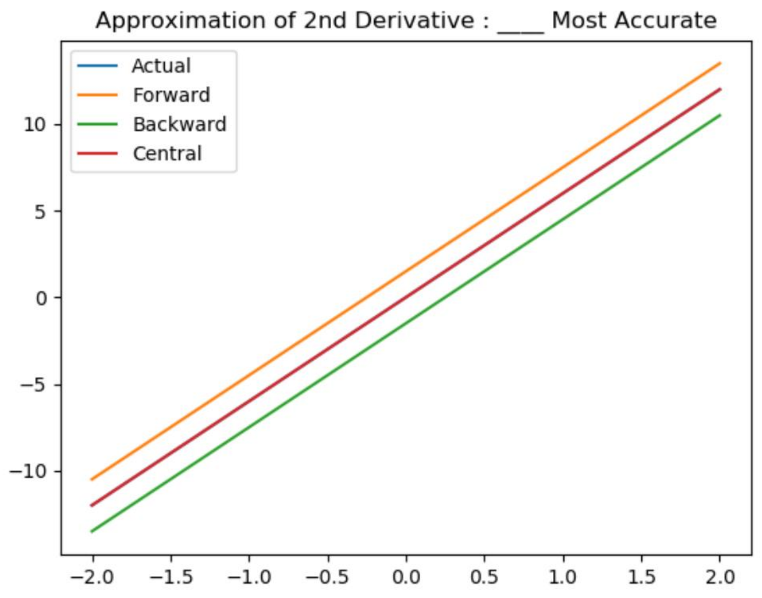


Figure 6: Plot showing comparison of second derivative approximations.