Homework 1

(a)
Matlab answer:
Example 1. The computed sum is sum1 = 100000 = 40f86a00000165cb (hex)

The true result for Example1 is sum=100000=40f86a0000000000 (hex)
[Matlab command: num2hex(100000)= 40f86a0000000000]

But Matlab output is slightly different. So, sum1 is not equal to the true value.
[hex2num('40f86a00000165cb')=1.000000000013329e+005, change the Matlab format to "long" to see this result by command >>format long]

The error occurs because of the following reasons:

1- 0.1 does not have a finite floating point representation.
   0.1=0.00011001100110011001100….
   Therefore a truncation error occurs due to the finite representation of 0.1 in Matlab.
2- sum1 = sum1+0.1 also introduces a round-off error because of the + operation and when the result is stored back in sum1.

These errors accumulate over the iterations; therefore sum1 is not equal to the true value.

(b)
Matlab answer:
Example 2. The computed sum is sum2 = 5.55551e+013 = 42c94375ad08dc01 (hex)
sum3 = 5.55551e+013 = 42c94375ad08dc00 (hex)

The true value for Example2 is:
sum=h*n*(n+1)/2= 55555055555000=42c94375ad08dc00

As can be seen, sum2 is not equal to the true value while sum3 is equal to that. And also sum2 and sum3 are not equal.

Now we change the value of h and run the program again to find both sum2 and sum3.
h=1.11;
Example 2. The computed sum is sum2 = 5.55001e+011 = 426027138cbf0000 (hex)
sum3 = 5.55001e+011 = 426027138cbeffff (hex)

True answer: sum=h*n*(n+1)/2= 5.5500e+011= 426027138cbf0000

Now, sum2 is equal to the true answer while sum3 has error.
There also may be some cases that both sum2 and sum3 have error. Therefore, what we can say is that, both computation methods for sum2 and sum3 may introduce error. This error is again because of the truncation error in the floating point representation of h and also the round-off error in summation.

(c)
Matlab answer:
Example 3. The computed sum is sum4 = 0 = 0000000000000000 (hex)
sum5 = -0.00130971 = bf55755000000000 (hex)

The true value for Example2 is: sum=0.

So, as can be seen, sum4 is equal to the true value while sum5 is not equal to that. The reason is because in calculation of sum4 the positive (+h*j) and negative (-h*j) values are canceled in each iteration. Therefore, no round-off error will be introduced after several iterations.
For sum5, in the first "for" loop the negative values are added up. Therefore, based on the results of Example 2, round-off error may occur. In the second "for" loop, the positive values are added up and again based on the results of Example 2, round-off error may occur. As the result, the final value of sum5 has error and is not equal to zero.

(d)
The general algorithm that can be used to keep round-off error in adding n numbers as small as possible is as follows:

1- Sort the numbers by their absolute values
2- Sum them form smallest to largest absolute value

As a reference to this question, you can take a look at the following paper:

T. G. Robertazzi and S. C. Schwartz, "Best "ordering" for floating-point addition**",** ACM Trans. Math. Softw, 1988.
Paper link: http://portal.acm.org/citation.cfm?id=42288.42343