12.010 Homework 1

The solution to the homework should be submitted as a Jupyter Notebook with embedded output. Make sure that your Notebook runs after re-starting the Kernel. Text parts of answers can be included as Markdown cells.

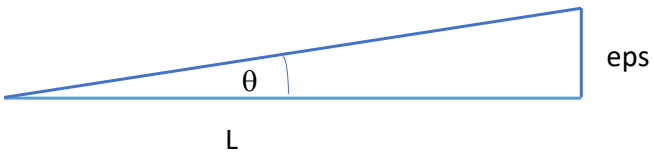
Web sources can be used, but cite the source of any code that is used from other sources. Each question is worth 20-points.

(1) Develop a cell that converts an input in binary, octal, hexadecimal or decimal into all the other bases. Test your codes with 0b10111011101010, 0o27352, 0x2eea.

(2) Test the accuracy of floating point calculations using standard Python float, numpy single, double, and longdouble. Determine how many significant digits are available by direct calculation (try adding and subtracting to see if the value is retained.) What can you determine about outputting longdouble values? For PC users, the behavior of longdoubles may be different from Mac and Unix users.

(3) Compare the speed of computing the sine of elements in a numpy array using array arithmetic versus looping with a for loop. Create a simple plot with time ratios for different-size arrays. Create the array with different types, single, double and longdouble, and see how the speed is affected. Also, look at the output of longdouble values. PC users may see different results.

(4) Explore the accuracy of the arcos function by creating a right triangle with one side 1 unit long and the other eps long where eps will be a small value. For given values of eps compute θ by arcos (compute the other side of the triangle) and by arcsin and arctan. How do the differences in the angle estimates change as eps approaches zero? Graph your results. Consider using non-linear axis scales.



(5) Create a list ([]) and equate it to another variable name. Determine the memory addresses of the two lists and make sure they are the same. Now, start extending the list to see if it moves the address. Look at the address of the list element and compare it to the addresses of elements in other lists. Do integers versus floats make any difference? Do you have an explanation for the behavior?

Analyses of results from different questions.

Analysis of Question 2

The finfo method tell us the characteristic of each of the floating point representations. The eps values of 1.19e-07, 2.22e-16, and 1.08e-19 tell the smallest increment possible at each precision. Looking a the Panda Data Frame output, we see the point where difference calculation (deps = ((1.0+eps)-1.0)-eps) returns a value of -eps (ie., 1.0+eps no longer changes the value) corresponds closely to the finfo values.

The individual test values (lines starting with C) show an interesting pattern where the double long does not show 1.2 well. Also moving 1 at the end of 1.125 to more significant digits, the double long reverts to the short form at the number of significant digits as the double. There seems to some issues with formatting the double long outputs. The epsilon test does show it does have the 19 significant digits expected.

```
C 1.2  ->  1.2 1.2 1.1999999999999999556 \Delta 4.7683715864721420985e-08 0.0
C 1.125000000001     ->  1.125 1.125000000001 1.1250000000010000889
C 1.12500000000001   ->  1.125 1.12500000000001 1.1250000000000009992
C 1.125000000000001  ->  1.125 1.125000000000001 1.1250000000000011102
C 1.125              ->  1.125 1.125 1.125 \Delta 0.0 0.0
```

** Analysis of Question 3**

>> SinSpeed

16	17	18	19	20	21	22
0.0007	0.0006	0.0011	0.0017	0.0035	0.0064	0.0117
0.0009	0.0008	0.0014	0.0028	0.0054	0.0090	0.0215
0.0038	0.0053	0.0065	0.0111	0.0224	0.0436	0.0858
0.0049	0.0045	0.0074	NaN	NaN	NaN	NaN

Notice that single precision speed between MATLAB and Python are similar for array processing, double is considerably slower in Python. MATLAB itself does not higher precision than double but there is toolboxes available that more than double precision accuracies. For loops, MATLAB is considerably faster but there is a caution here: MATLAB may have internally have converted the loop into a vector operation (MATLAB documentation says it can do this.)

** Analysis Question 4 **

** Analysis **

The x-axis has been reversed here. The linear-like log-log plot shows that the relationship between the size of epsilon and the error is a power law. For arcsin, the difference to arctan is often zero. We could experiment with using double long to compute a more accurate epsilon to test the accuracy of double arctan function.

Notice that math.arccos and numpy.arccos generate the same result. This need not to have been the case.

** Analysis Question 5 **

First thing to note here is that and int takes up 24 bytes. Only four of those bites contain information about the value. The remaining bytes I'm not clearly defined but appear probably to be the addresses of where the values are stored and probably the values before and after the current one pointed to by the int. In terms of extending arrays lists we see that the list address simply points to where the values are stored and so when we equate lists this initial list address probably does not change as the list is extended. For the integer values specific integers I'll save that certain addresses and the same address is used in different lists. For floating point values the addresses are different even when the value is the same and interestingly the addresses in the lists do not seem to be contiguous which probably has major effects on the efficiency of doing calculations with lists.

Based on more careful reading of id() function, it is probably not a memory location in most implementations of Python. This becomes very clear with the numpy array at the end where different elements in the array appear to have the same ID value. What id() is returning is not clear.

12.010 Computational Methods of Scientific Programming

Fall 2024