Problem Set 1 - ECON 880

Spring 2022 - University of Kansas

Minh Cao, Gunawan

February 4, 2022

Problem 1

We want to compute

$$83521y^8 + 578x^2y^4 - 2x^4 + 2x^6 - x^8 \tag{1}$$

for $x = 9478657$ and $y = 2298912$.

(1a)

We evaluate the expression (1) as it is written, and then by grouping terms using Horner's method[†]. The multivariate Honer's algorithm can be described as follows:

- Step 1: Let our original multivariable polynomial be $f(x, y) = M_1 + M_2 + M_3 + M_4 + M_5$, where M_i is a monomial, for all $i = 1, \ldots, 5$. In this case,

$$M_1 = 83521y^8, M_2 = 578x^2y^4, M_3 = -2x^4, M_4 = 2x^6, M_5 = -x^8.$$

- Step 2: For each monomial M_i, we count, for each variable x and y, the number of monomials M_i that contain this variable. We then select the variable with the largest value of M_i: we choose x, as it appears 4 times, while y only twice. Rewrite the original polynomial as $f(x, y) = A_0 + A_1x$. Where A_0 is the monomial that does not contain x^2, and A_1 is the sum of all the terms $M_k{}' = \frac{M_k}{x}$ corresponding to all the monomials M_k that do contain x. In this case, $A_0 = 83521y^8$ and $A_1 = 578xy^4 - 2x^3 + 2x^5 - x^7$, or

$$f(x, y) = 83521y^8 + x(578xy^4 - 2x^3 + 2x^5 - x^7)$$

- Step 3: Repeat step 1 and 2 until we get:

$$\begin{aligned} f(x, y) &= 83521y^8 + xx(578y^4 - 2x^2 + 2x^4 - x^6) \\ &= \ldots \\ &= 83521y^8 + xx(578y^4 + xx(-2 + xx(2 - xx))) \end{aligned}$$

Result: Direct calculation gives $f(x, y) = -1.088903574147003 \cdot 10^{40}$, while multivariate Horner approach gives $f(x, y) = 6.515900793473058 \cdot 10^{55}$. The tremendous discrepancy is due to [please explain]

[†]Horner's algorithm is coded in Matlab according to Kenneth L. Judd, 1998. "Numerical Methods in Economics," MIT Press Books, The MIT Press, p.35

(1b)

We estimate the order of magnitude (number of digits) of

$$83521y^8, \tag{2}$$

where $y = 2298912$. We can rewrite the expression as follows:

$$\begin{aligned} 83521y^8 &= 83521 \cdot 2298912^8 \\ &= 8.3521 \cdot 10^4 \cdot (2.298912 \cdot 10^6)^8 \end{aligned}$$

Then, the digits can be estimated by using the following formula

$$\begin{aligned} 1 + \lfloor \log_{10}(83521 \cdot 2298912^8) \rfloor &= 1 + \lfloor \log_{10}(83521 \cdot 2298912^8) \rfloor \\ &= 1 + \lfloor \log_{10}(8.3521 \cdot 10^4 \cdot (2.298912 \cdot 10^6)^8) \rfloor \\ &= 1 + \lfloor \log_{10} 8.3521 + \log_{10} 10^4 + \log_{10} 2.298912^8 + \log_{10}(10^6)^8 \rfloor \\ &= 56 \end{aligned}$$

Now, we compute the last digits of that term, first by plugging in only the last digit of y into expression (2): our calculation gives 21,312,256 as a result. However, printing out the complete term using the command fprintf($'\%.0f\backslash n', 83251 * y^8$), we obtain the following 56-digit number:

$$64, 948, 367, 112, 155, 225, 808, 364, 994, 454, 077, 172, 753, 899, 719, 772, 317, 155, 328.$$

Comment: The value calculated using direct method should be true, since our computer uses 64-bit CPU that is able to show all the digits of the calculated number. There is no precision loss in this case.

(1c)

Yes, the 64-bit can show the correct value since the number of digit is less than the maximum number of digit the system can represent. The orders of magnitude of other terms are at most 56.

Problem 2

In this exercise, we write an algorithm to determine the relative speeds of addition, multiplication, division, exponentiation, and the logarithmic function of our computer. The computer uses Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz, 1.50 GHz with 64-bit operating system and 8.00 GB installed RAM. We proceed by generating two matrices A and B of size $10^4 \times 10^4$ using the rand() function in Matlab. Then, we use them in element-wise operations of addition ($A + B$), multiplication ($A. * B$), division ($A./B$), exponentiation ($A.^B$), as well as the logarithmic function ($\log(A)$). The statements `tic` and `toc` are used around them to measure the computation time. This procedure is iterated 100 times, and the average computation time for each operation is computed. The results are as follows:

- Average computation time for variable initiation is 1.54030 seconds
- Average computation time for addition is 0.12073 seconds
- Average computation time for multiplication is 0.12083 seconds

- Average computation time for division is 0.11823 seconds
- Average computation time for exponentiation is 3.70586 seconds
- Average computation time for log function is 0.91528 seconds

Problem 3

In order to find our machine ε, we follow Ken Judd's definition[†] by writing a while loop to subtract (resp. add) progressively smaller numbers ϵ from (resp. to) 1 until the condition $1+\epsilon > 1 > 1-\epsilon$ is no longer satisfied. Repeat the exercise using 0.001 and 1000 instead of 1. We verify our results by comparing them with the epsilons delivered by the built-in Matlab function `eps(x)`, and the exponents with `log2(eps(x))`. The results are shown on Table 1.

x	$\varepsilon(x)$ - decimal	$\varepsilon(x)$ - exponential
0.001	2.16840434497101e-19	2^{-62}
1	2.22044604925031e-16	2^{-52}
1000	1.13686837721616e-13	2^{-43}

Table 1: Machine ε for diverse values of x

Comment: As x increases, the machine $\varepsilon(x)$ also does increase. This is to be expected, since $\varepsilon(x) \approx x\varepsilon(1)$. Thus, $\varepsilon(1000) > \varepsilon(1) > \varepsilon(0.001)$.

Problem 4

We wrote a loop to evaluate the convergence of the following sequences:

(4a) $x_k = \sum_{k=1}^{n} \frac{1}{2^n}$, where $\lim_{k\to\infty} x_k = 1$

(4b) $y_k = \sum_{k=1}^{n} \frac{1}{n}$, where $\lim_{k\to\infty} y_k = \infty$

We use absolute ($||(x_{k+1} - x_k)||$) and relative ($||(x_{k+1} - x_k)||/(1 + ||x_k||)$) convergence criteria, and tolerance distance $\delta \in \{10^{-2}, 10^{-4}, 10^{-6}\}$ for the stopping rule. We limit the maximum number of iterations to 100,000. The number of iterations before convergence, as well the final guess are reported on Table 2 and 3 for x_k and y_k, respectively.

δ	10^{-2}	10^{-4}	10^{-6}
no. of iteration - absolute criteria	6	13	19
final guess - absolute criteria	0.9921875	0.99993896484375	0.999999046325684
no. of iteration - relative criteria	5	12	18
final guess - relative criteria	0.984375	0.9998779296875	0.999998092651367

Table 2: Convergence for x_k

Comment: Table 2 shows that both absolute and relative convergence criteria lead to a number of iterations differing by one unit for the sequence x_k. Already at the tolerance level of $\delta = 10^{-2}$, we see that the final guesses for x_k using both convergence criteria are very close to the true limit

[†]Kenneth L. Judd, 1998. "Numerical Methods in Economics," MIT Press Books, The MIT Press, p.30

δ	10^{-2}	10^{-4}	10^{-6}
no. of iteration - absolute criteria	99	9,999	100,000
final guess - absolute criteria	5.18737751763962	9.78760603604435	12.0901561297633
no. of iteration - relative criteria	21	1,158	77,880
final guess - relative criteria	3.69081325021728	7.63295985258894	11.8401593846097

Table 3: Convergence for y_k

1. As tolerance distance δ lowers, these numbers get even closer to 1. Table 3 shows that both absolute and relative criteria lead to the significantly different number of iterations for the sequence y_k. Since y_k is a divergent sequence, the final guess will be higher (with no upper bound), the more we iterate. By lowering tolerance distance δ, the algorithm needs to iterate longer in order to satisfy the stopping rule, which in turn results in higher final guess. Since there is no upper bound for y_k, we can always make the tolerance distance δ even lower, and obtain even higher final guess for y_k. Based on these numbers, we see that y_k goes to infinity very slowly.