

## 18.335 Problem Set 1 Solutions

### Problem 1: (10 points)

The smallest integer that cannot be exactly represented is  $n = \beta^t + 1$  (for base- $\beta$  with a  $t$ -digit mantissa). You might be tempted to think that  $\beta^t$  cannot be represented, since a  $t$ -digit number, at first glance, only goes up to  $\beta^t - 1$  (e.g. three base-10 digits can only represent up to 999, not 1000). However,  $\beta^t$  can be represented by  $\beta^{t-1} \cdot \beta^1$ , where the  $\beta^1$  is absorbed in the exponent.

In IEEE single and double precision,  $\beta = 2$  and  $t = 24$  and 53, respectively, giving  $2^{24} + 1 = 16,777,217$  and  $2^{53} + 1 = 9,007,199,254,740,993$ .

Evidence that  $n = 2^{53} + 1$  is not exactly represented but that numbers less than that are can be presented in a variety of ways. In the pset1-solutions notebook, we check exactness by comparing to Julia's `Int64` (built-in integer) type, which exactly represents values up to  $2^{63} - 1$ .

### Problem 2: (10+10 points)

See the pset1 solutions notebook for Julia code, results, and explanations.

### Problem 3: (10+10+10 points)

See the pset1 solutions notebook for Julia code, results, and explanations.

### Problem 4: (10+5+10 points)

Here you will analyze  $f(x) = \sum_{i=1}^n x_i$  as in class, but this time you will compute  $\tilde{f}(x)$  in a different way. In particular, compute  $\tilde{f}(x)$  by a recursive divide-and-conquer approach known in the literature as **pairwise summation**, recursively dividing the set of values to be summed in two halves and then summing the halves:

$$\tilde{f}(x) = \begin{cases} 0 & \text{if } n = 0 \\ x_1 & \text{if } n = 1, \\ \tilde{f}(x_{1:\lfloor n/2 \rfloor}) \oplus \tilde{f}(x_{\lfloor n/2 \rfloor + 1:n}) & \text{if } n > 1 \end{cases}$$

where  $\lfloor y \rfloor$  denotes the greatest integer  $\leq y$  (i.e.  $y$  rounded down). In exact arithmetic, this computes  $f(x)$  exactly, but in floating-point arithmetic this will have very different error characteristics than the simple sequential summation in class.

(a) Suppose  $n = 2^m$  with  $m \geq 1$ . We will first show that

$$\tilde{f}(x) = \sum_{i=1}^n x_i \prod_{k=1}^m (1 + \epsilon_{i,k})$$

where  $|\epsilon_{i,k}| \leq \epsilon_{\text{machine}}$ . We prove the above relationship by induction. For  $n = 2$  it follows from the definition of floating-point arithmetic. Now, suppose it is true for  $n$  and we wish to prove it for  $2n$ . The sum of  $2n$  number is first summing the two halves recursively (which has the above bound for each half since they are of length  $n$ ) and then adding the two sums, for a total result of

$$\tilde{f}(x \in \mathbb{R}^{2n}) = \left[ \sum_{i=1}^n x_i \prod_{k=1}^m (1 + \epsilon_{i,k}) + \sum_{i=n+1}^{2n} x_i \prod_{k=1}^m (1 + \epsilon_{i,k}) \right] (1 + \epsilon)$$

for  $|\epsilon| < \epsilon_{\text{machine}}$ . The result follows by inspection, with  $\epsilon_{i,m+1} = \epsilon$ .

Then, we use the result from class that  $\prod_{k=1}^m (1 + \epsilon_{i,k}) = 1 + \delta_i$  with  $|\delta_i| \leq m\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$ . Since  $m = \log_2(n)$ , the desired result follows immediately.

- (b) Just enlarge the base case. Instead of recursively dividing the problem in two until  $n < 2$ , divide the problem in two until  $n < N$  for some  $N$ , at which point we sum the  $< N$  numbers with a simple loop as in problem 2. A little arithmetic reveals that this produces  $\sim 2n/N$  function calls—this is negligible compared to the  $n - 1$  additions required as long as  $N$  is sufficiently large (say,  $N = 200$ ), and the efficiency should be roughly that of a simple loop. (See the pset1 Julia notebook for benchmarks and explanations.)

Using a simple loop has error bounds that grow as  $N$  as you showed above, but  $N$  is just a constant, so this doesn't change the overall logarithmic nature of the error growth with  $n$ . A more careful analysis analogous to above reveals that the worst-case error grows as  $[N + \log_2(n/N)]\epsilon_{\text{machine}} \sum_i |x_i|$ . Asymptotically, this is not only  $\log_2(n)\epsilon_{\text{machine}} \sum_i |x_i|$  error growth, but with the same asymptotic constant factor (same coefficient of the  $\log_2 n$  term)!

- (c) Instead of “if ( $n < 2$ ),” just do (for example) “if ( $n < 200$ )”. See the notebook for code and results.