18.335 Problem Set 2

Due Monday, 16 February 2015.

Problem 1: (10 points)

The smallest integer that cannot be exactly represented is $n = \beta^t + 1$ (for base-β with a t-digit mantissa). You might be tempted to think that β^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, β^t can be represented by $\beta^{t-1} \cdot \beta^1$, where the β^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: (5+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+10+5 points)

(a) We can prove this by induction on n. For $n = 1$, it is trivial with $\epsilon_1 = 0$; alterenatively, for the case of $n = 2$, $\tilde{f}(x) = (0 \oplus x_1) \oplus x_2 = x_1 \oplus x_2 = (x_1 + x_2)(1 + \epsilon_2)$ for $|\epsilon_2| \le \epsilon_{\text{machine}}$ is a consequence of the correct rounding of $\oplus$ ($0 \oplus x_1$ must equal x_1, and $x_1 \oplus x_2$ must be within $\epsilon_{\text{machine}}$ of the exact result). (If we don't assume correct rounding, then the result is only slightly modified by an additional $1 + \epsilon_1$ factor multiplying x_1.)

Now for the inductive step. Suppose $\tilde{s}_{n-1} = \sum_{i=1}^{n-1} x_i \prod_{k=i}^{n-1}(1 + \epsilon_k)$. Then $\tilde{s}_n = \tilde{s}_{n-1} \oplus x_n = (\tilde{s}_{n-1} + x_n)(1 + \epsilon_n)$ where $|\epsilon_n| < \epsilon_{\text{machine}}$ is guaranteed by floating-point addition. The result follows by inspection: the previous terms are all multiplied by $(1 + \epsilon_n)$, and we add a new term $x_n(1 + \epsilon_n)$.

(b) First, let us multiply out the terms: $(1 + \epsilon_i) \cdots (1 + \epsilon_n) = 1 + \sum_{k=i}^n \epsilon_k +$ (products of ϵ) $= 1 + \delta_i$, where the products of ϵ_k terms are $O(\epsilon_{\text{machine}}^2)$, and hence $|\delta_i| \le \sum_{k=i}^n |\epsilon_k| + O(\epsilon_{\text{machine}}^2) \le (n - i + 1)\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$. Now e have: $\tilde{f}(x) = f(x) + (x_1 + x_2)\delta_2 + \sum_{i=3}^n x_i \delta_i$, and hence (by the triangle inequality):

$$|\tilde{f}(x) - f(x)| \le |x_1|\,|\delta_2| + \sum_{i=2}^n |x_i|\,|\delta_i|.$$

But $|\delta_i| \le n\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$ for all i, and hence $|\tilde{f}(x) - f(x)| \le n\epsilon_{\text{machine}} \sum_{i=1}^n |x_i|$.

Note: This does *not* correspond to forwards stability, since we have only shown that $|\tilde{f}(x) - f(x)| = \|x\| O(\epsilon_{\text{machine}})$, which is different from $|\tilde{f}(x) - f(x)| = |f(x)| O(\epsilon_{\text{machine}})$! Our $O(\epsilon_{\text{machine}})$ is indeed uniformly convergent, however (i.e. the constant factors are independent of x, although they depend on n).

(c) For uniform random ϵ_k, since δ_i is the sum of $(n - i + 1)$ random variables with variance $\sim \epsilon_{\text{machine}}$, it follows from the usual properties of random walks (i.e. the *central limit theorem*) that the mean $|\delta_i|$ has magnitude $\sim \sqrt{n - i + 1} O(\epsilon_{\text{machine}}) \le \sqrt{n} O(\epsilon_{\text{machine}})$. Hence $|\tilde{f}(x) - f(x)| = O\left(\sqrt{n}\epsilon_{\text{machine}} \sum_{i=1}^n |x_i|\right)$.

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

Here you will analyze $f(x) = \sum_{i=1}^n x_i$ as in problem 2, 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, 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 $\le y$ (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 loop-based summation in problem 2.

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

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

where $|\epsilon_{i,k}| \le \epsilon_{\text{machine}}$. 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 problem 2 that $\prod_{k=1}^{m} (1 + \epsilon_{i,k}) = 1 + \delta_i$ with $|\delta_i| \le m\epsilon_{\text{machine}} + O(\epsilon_{\text{machine}}^2)$. Since $m = \log_2(n)$, the desired result follows immediately.

(b) As in problem 2, our δ_i factor is now a sum of random $\epsilon_{i,k}$ values and grows as $\sqrt{m}$. That is, we expect that the average error grows as $\sqrt{\log_2 n}\, O(\epsilon_{\text{machine}}) \sum_i |x_i|$.

(c) 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!

(d) Instead of "if (n < 2)," just do (for example) "if (n < 200)". See the notebook for code and results.