18.335 Problem Set 2 Solutions

Problem 1: Floating-point

(a) 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 found by looking at the last few decimal digits as we increment the numbers. e.g. the last 3 decimal digits of m in Matlab are returned by `rem(m,1000)`. `rem(2^53-2,1000)=990`, `rem(2^53-1,1000)=991`, `rem(2^53,1000)=992`, `rem(2^53+1,1000)=992`, `rem(2^53+2,1000)=994`, `rem(2^53+3,1000)=996`, and `rem(2^53+4,1000)=996`. That is, incrementing up to $n-1$ increments the last digit as expected, while going from $n-1$ to n the last digit (and indeed, the whole number) doesn't change, and after that the last digit increments in steps of 2. In particular, $n+1$ and $n+3$ are both exactly represented, because they are even numbers: a factor of two can be pulled into the exponent, since $2^{53}+2 = (2^{52}+1)\cdot 2$ and $2^{53}+4 = (2^{52}+2)\cdot 2$, and hence the significand is still exactly represented.

(b) What we want to show, for a function $g(x)$ with a convergent Taylor series at $x = 0$, that $g(O(\varepsilon)) = g(0) + g'(0)O(\varepsilon)$. [We must also assume $g'(0) \neq 0$, otherwise it is obviously false.] The first thing we need to do is to write down precisely what this means. We know what it means for a function $f(\varepsilon)$ to be $O(\varepsilon)$: it means that, for ε sufficiently small ($0 \le \varepsilon < \delta$ for some δ), then $|f(\varepsilon)| < C_1\varepsilon$ for some $C_1 > 0$. Then, by $g(O(\varepsilon))$, we mean $g(f(\varepsilon))$ for any $f(\varepsilon) \in O(\varepsilon)$; we wish to show that $f(\varepsilon)$ being $O(\varepsilon)$ implies that

$$g(f(\varepsilon)) = g(0) + g'(0)h(\varepsilon)$$

for some $h(\varepsilon)$ that is also $O(\varepsilon)$.

Since $g(x)$ has a convergent Taylor series, we can explicitly write

$$h(\varepsilon) = f(\varepsilon) + \frac{1}{g'(0)} \sum_{n=2}^{\infty} \frac{g^{(n)}(0)}{n!} f(\varepsilon)^n.$$

But since $|f(\varepsilon)| < C_1\varepsilon$ for some C_1 (and for sufficiently small ε), it immediately follows that

$$|h(\varepsilon)| < C_1\varepsilon \left[1 + \frac{1}{|g'(0)|} \sum_{n=1}^{\infty} \frac{\left|g^{(n+1)}(0)\right|}{(n+1)!} C_1^n \varepsilon^n \right],$$

which is clearly $< 2C_1\varepsilon$ for sufficiently small ε (and indeed, is $< C_2\varepsilon$ for any $C_2 > C_1$), since the summation of ε^n must go to zero as $\varepsilon \to 0$ [if it doesn't, it is trivial to show that the Taylor series won't converge to a function with the correct derivative $g'(0)$ at $\varepsilon = 0$].

Problem 2: Addition

(a) We can prove this by induction on n. For the base case of $n = 2$, $\tilde{f}(x) = (0 \oplus x_1) \oplus x_2 = x_1 \oplus x_2 = (x_1 + x_2)(1 + \varepsilon_2)$ for $|\varepsilon_2| \le \varepsilon_{\text{machine}}$ 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 $\varepsilon_{\text{machine}}$ of the exact result).

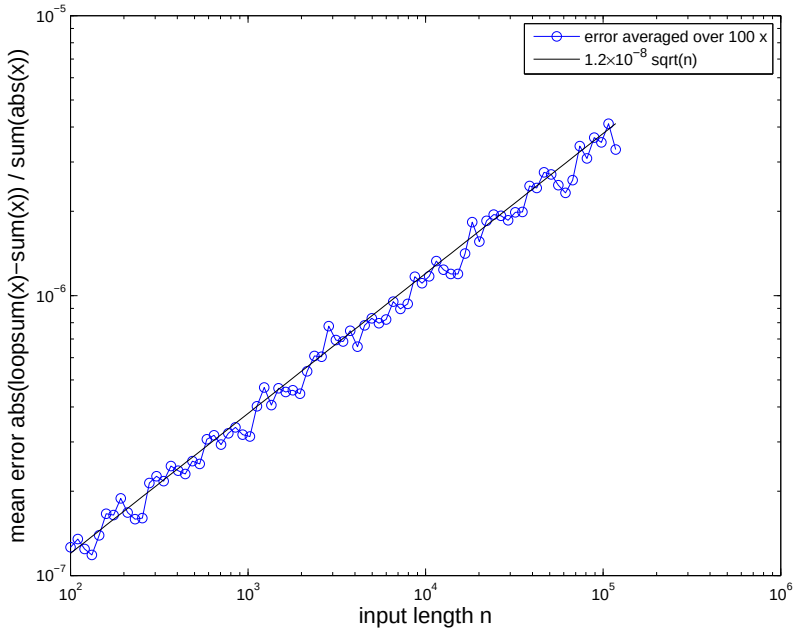

Figure 1: Error $|\tilde{f}(x) - f(x)|/\sum_i |x_i|$ for random $x \in [0,1)^n$, where $\tilde{f}$ is computed by a simple loop in single precision, averaged over 100 random x vectors, as a function of n. Notice that it fits very well to $\approx 1.2 \times 10^{-8}\sqrt{n}$, matching the expected $\sqrt{n}$ growth for random errors.

Now for the inductive step. Suppose $\tilde{s}_{n-1} = (x_1 + x_2)\prod_{k=2}^{n-1}(1+\varepsilon_k) + \sum_{i=3}^{n-1} x_i \prod_{k=i}^{n-1}(1+\varepsilon_k)$. Then $\tilde{s}_n = \tilde{s}_{n-1} \oplus x_n = (\tilde{s}_{n-1} + x_n)(1+\varepsilon_n)$ where $|\varepsilon_n| < \varepsilon_{\text{machine}}$ is guaranteed by floating-point addition. The result follows by inspection: the previous terms are all multiplied by $(1+\varepsilon_n)$, and we add a new term $x_n(1+\varepsilon_n)$.

(b) This is trivial: just multiplying out the terms $(1+\varepsilon_i)\cdots(1+\varepsilon_n) = 1 + \sum_{k=i}^n \varepsilon_k + (\text{products of } \varepsilon) = 1 + \delta_i$, where the products of ε_k terms are $O(\varepsilon_{\text{machine}}^2)$, and hence (by the triangle inequality) $|\delta_i| \le \sum_{k=i}^n |\varepsilon_k| + O(\varepsilon_{\text{machine}}^2) \le (n-i+1)\varepsilon_{\text{machine}} + O(\varepsilon_{\text{machine}}^2)$.

(c) We 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 $|\delta_i| \le n\varepsilon_{\text{machine}} + O(\varepsilon_{\text{machine}}^2)$ for all i, from the previous part, and hence $|\tilde{f}(x) - f(x)| \le n\varepsilon_{\text{machine}} \sum_{i=1}^n |x_i|$.

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

(e) Results of the suggested numerical experiment are plotted in figure 1. For each n, I averaged the error $|\tilde{f}(x) - f(x)|/\sum_i |x_i|$ over 100 runs to reduce the variance.

Problem 3: Addition, another way

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

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

where $|\varepsilon_{i,k}| \leq \varepsilon_{\text{machine}}$. 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

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

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

Then, we use the result from problem 2 that $\prod_{k=1}^{m} (1 + \varepsilon_{i,k}) = 1 + \delta_i$ with $|\delta_i| \leq m\varepsilon_{\text{machine}} + O(\varepsilon_{\text{machine}}^2)$. 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 $\varepsilon_{i,k}$ values and grows as $\sqrt{m}$. That is, we expect that the average error grows as $\sqrt{\log_2 n} O(\varepsilon_{\text{machine}}) \sum_i |x_i|$.

(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.

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 $[N + \log_2(n/N)]\varepsilon_{\text{machine}} \sum_i |x_i|$. Asymptotically, this is not only $\log_2(n)\varepsilon_{\text{machine}} \sum_i |x_i|$ error growth, but with the same asymptotic constant factor!

(d) Instead of "if (n < 2)," just do "if (n < 200)". To keep everything in single precision, one should, strictly speaking, call loopsum instead of the built-in function sum (which uses at least double precision, and probably uses extended precision).

The logarithmic error growth is actually so slow that it is practically impossible to see the errors growing at all. In an attempt to see it more clearly, I wrote a C program to implement the same function (orders of magnitude quicker than doing recursion in Matlab), and went up to $n = 10^9$ or so. As in problem 2, I averaged over 100 random x to reduce the variance. The results are plotted in figure 2 for two cases: $N = 1$ (trivial base case) and $N = 200$ (large base case, much faster). Asymptotically, the error is growing extremely slowly with n, as expected, although it is hard to see even a logarithmic growth; it looks pretty flat. There are also a few surprises.

First, we see that the errors are oscillating, at a constant rate on a semilog scale. In fact, the period of the oscillations corresponds to powers of two—the error decreases as a power of two is approached, and then jumps up again when n exceeds a power of 2. Intuitively, what is happening is this: the reason for the slow error growth is that we are recursively dividing x into equal-sized chunks, and are therefore adding quantities with nearly equal magnitudes on average (which minimized roundoff error), but when n is not a power of two some of the chunks are unequal in size and the error increases.

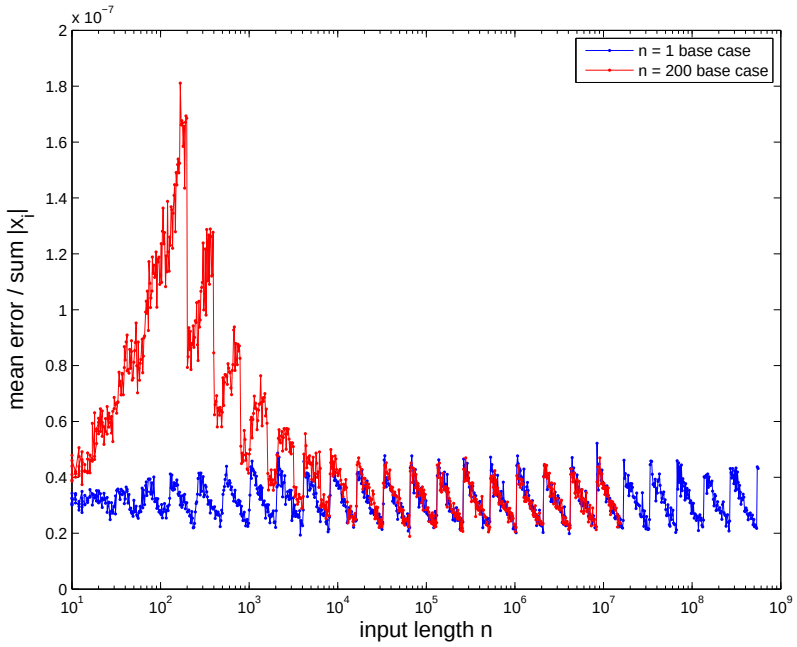

Figure 2: Error $|\tilde{f}(x) - f(x)| / \sum_i |x_i|$ for random $x_i \in [0,1)^n$, averaged over 100 x vectors, for $\tilde{f}$ computed in single precision by recursively dividing the sum in two halves until $n < N$, at which point a simple loop is employed. Results for $N = 1$ and $N = 200$ base cases are shown.

Second, for the $N = 200$ base case, the errors initially increase much faster—as $\sqrt{n}$, in fact, and then come back down for $n \gg N$. Obviously, for $n < N$ the errors must increase as $\sqrt{n}$ as in problem 2, since for this case we do no recursion and just sum via a loop. However, when $n \gg N$, the logarithmic terms in the error dominate over the $O(N)$ term, and the error approaches the error for $N = 1$ with the same constant factor, as predicted above!

However, predicting the exact functional dependence is clearly quite difficult!

(e) An $m \times m$ matrix multiplication is just a bunch of length-m dot products. The only error accumulation in a dot product will occur in the summation, so the error growth with m should be basically the same as in our analysis of the corresponding summation algorithm.

If you use the simple 3-loop row-column algorithm, you are doing the summation(s) via simple loops, and the errors should thus grow as $O(\varepsilon_{\text{machine}}\sqrt{m})$ on average as above. The cache-oblivious algorithm, on the other hand, corresponds to recursively dividing each dot product in two, and hence the errors should grow as $O(\varepsilon_{\text{machine}}\sqrt{\log m})$ as above.

In most cases, however, m isn't large enough for people to care about this difference in accuracy for matrix multiplies.

Problem 4: Stability

(a) Trefethen, exercise 15.1. In the following, I abbreviate $\varepsilon_{\text{machine}} = \varepsilon_m$, and I use the fact (from problem 1) that we can replace any $g(O(\varepsilon))$ with $g(0) + g'(0)O(\varepsilon)$. I also assume that $\text{fl}(x)$ is

deterministic—by a stretch of Trefethen's definitions, it could conceivably be nondeterministic in which case one of the answers changes as noted below, but this seems crazy to me (and doesn't correspond to any real machine).

(i) Backward stable. $x \oplus x = \text{fl}(x) \oplus \text{fl}(x) = [x(1+\varepsilon_1) + x(1+\varepsilon_1)](1+\varepsilon_2) = 2\tilde{x}$ for $|\varepsilon_i| \le \varepsilon_m$ and $\tilde{x} = x(1+\varepsilon_1+\varepsilon_2+2\varepsilon_1\varepsilon_2) = x[1+O(\varepsilon_m)]$.

(ii) Backward stable. $x \otimes x = \text{fl}(x) \otimes \text{fl}(x) = [x(1+\varepsilon_1) \times x(1+\varepsilon_1)](1+\varepsilon_2) = \tilde{x}^2$ for $|\varepsilon_i| \le \varepsilon_m$ and $\tilde{x} = x(1+\varepsilon_1)\sqrt{1+\varepsilon_2} = x[1+O(\varepsilon_m)]$.

(iii) Stable but not backwards stable. $x \oslash x = [\text{fl}(x)/\text{fl}(x)](1+\varepsilon) = 1+\varepsilon$ (not including $x = 0$ or ∞, which give NaN). This is actually forwards stable, but there is no $\tilde{x}$ such that $\tilde{x}/\tilde{x} \neq 1$ so it is not backwards stable. (Under the stronger assumption of correctly rounded arithmetic, this will give exactly 1, however.)

(iv) Backwards stable. $x \ominus x = [\text{fl}(x) - \text{fl}(x)](1+\varepsilon) = 0$. This is the correct answer for $\tilde{x} = x$. (In the crazy case where fl is not deterministic, then it might give a nonzero answer, in which case it is unstable.)

(v) Unstable. It is definitely not backwards stable, because there is no data (and hence no way to choose $\tilde{x}$ to match the output). To be stable, it would have to be forwards stable, but it isn't because the errors decrease more slowly than $O(\varepsilon_m)$. More explicitly, $1 \oplus \frac{1}{2} \oplus \frac{1}{6} \oplus \cdots$ summed from left to right will give $((1+\frac{1}{2})(1+\varepsilon_1)+\frac{1}{6})(1+\varepsilon_2)\cdots = e + \frac{3}{2}\varepsilon_1 + \frac{10}{6}\varepsilon_2 + \cdots$ dropping terms of $O(\varepsilon^2)$, where the coefficients of the ε_k factors converge to e. The number of terms is n where n satisfies $n! \approx 1/\varepsilon_m$, which is a function that grows very slowly with $1/\varepsilon_m$, and hence the error from the additions alone is bounded above by $\approx n\varepsilon_m$. The key point is that the errors grow at least as fast as $n\varepsilon_m$ (not even counting errors from truncation of the series, approximation of $1/k!$, etcetera), which is *not* $O(\varepsilon_m)$ because n grows slowly with decreasing ε_m.

(vi) Stable. As in (e), it is not backwards stable, so the only thing is to check forwards stability. Again, there will be n terms in the series, where n is a slowly growing function of $1/\varepsilon_m$ ($n! \approx 1/\varepsilon_m$). However, the summation errors no longer grow as n. From right to left, we are summing $\frac{1}{n!} \oplus \frac{1}{(n-1)!} \oplus \cdots \oplus 1$. But this gives $((\frac{1}{n!} + \frac{1}{(n-1)!})(1+\varepsilon_{n-1}) + \frac{1}{(n-2)!})(1+\varepsilon_{n-2})\cdots$,and the linear terms in the ε_k are then bounded by

$$\left| \sum_{k=1}^{n-1} \varepsilon_k \sum_{j=k}^{n} \frac{1}{j!} \right| \le \varepsilon_m \sum_{k=1}^{n-1} \sum_{j=k}^{n} \frac{1}{j!} = \varepsilon_m \left[\frac{n-1}{n!} + \sum_{j=1}^{n-1} \frac{j}{j!} \right] \approx \varepsilon_m e = O(\varepsilon_m).$$

The key point is that the coefficients of the ε_k coefficients grow smaller and smaller with k, rather than approaching e as for left-to-right summation, and the sum of the coefficients converges. The truncation error is of $O(\varepsilon_m)$, and we assume $1/k!$ can also be calculated to within $O(\varepsilon_m)$, e.g. via Stirling's approximation for large k, so the overall error is $O(\varepsilon_m)$ and the algorithm is forwards stable.

(vii) Unstable. Not backwards stable since no data, but what about forwards stability? The problem is the squaring of the sine function. Suppose $x = \pi - \delta$ and $x' = x(1+\varepsilon_m)$ for some small $\delta > 0$. Then $\sin(x)\sin(x') \approx \delta(\delta - \varepsilon_m\pi) + O(\delta^2)$. In exact arithmetic, this goes to zero for $\delta = 0$, i.e. $x = \pi$. However, it goes to zero too rapidly: if $\delta = O(\sqrt{\varepsilon_m})$,then $\sin(x)\sin(x') = O(\varepsilon_m)$, and an $O(\varepsilon_m)$ floating-point error in computing sin will cause the product to pass through zero. Therefore, this procedure only finds π to $O(\sqrt{\varepsilon_m})$, which is too slow to be considered stable.

(b) Trefethen, exercise 16.1. Since stability under all norms is equivalent, we are free to choose $\|\cdot\|$ to be the L_2 norm (and the corresponding induced norm for matrices), for convenience, since that norm is preserved by unitary matrices.

(i) First, we need to show that multiplication of A by a *single* unitary matrix Q is backwards stable. That is, we need to find a δA with $\|\delta A\| = \|A\| O(\varepsilon_{\text{machine}})$ such that $\widetilde{QA} = Q(A + \delta A)$. Since $\|Q\delta A\| = \|\delta A\|$, however, this is equivalent to showing $\|\widetilde{QA} - QA\| = \|A\| O(\varepsilon_{\text{machine}})$. It is sufficient to look at the error in the ij-th element of QA, i.e. the error in computing $\sum_k q_{ik} a_{kj}$. Assuming we do this sum by a straightforward loop, the analysis is exactly the same as in problem 2, except that there is an additional $(1+\varepsilon)$ factor in each term for the error in the product $q_{ik}a_{kj}$ [or $(1+2\varepsilon)$ if we include the rounding of q_{ik} to $\tilde{q}_{ik} = \text{fl}(q_{ik})$]. Hence, the error in the ij-th element is bounded by $mO(\varepsilon_{\text{machine}}) \sum_k |q_{ik}a_{kj}|$, and (using the unitarity of Q, which implies that $|q_{ik}| \le 1$, and the equivalence of norms) this in turn is bounded by $mO(\varepsilon_{\text{machine}}) \sum_k |a_{kj}| \le mO(\varepsilon_{\text{machine}}) \sum_{kj} |a_{kj}| \le mO(\varepsilon_{\text{machine}}) \|A\|$. Summing m^2 of these errors in the individual elements of QA, again using norm equivalence, we obtain $\|\widetilde{QA} - QA\| \le O(1) \sum_{ij} |(\widetilde{QA} - QA)_{ij}| \le m^3 O(\varepsilon_{\text{machine}}) \|A\|$. Thus, we have proved backwards stability for multiplying by one unitary matrix (with a very pessimistic m^3 coefficient, but that doesn't matter here).

Now, we will show by induction that multiplying by k unitary matrices is backwards stable. Suppose we have proved it for k, and want to prove for $k+1$. That is, consider $QQ_1 \cdots Q_k A$. By assumption, $Q_1 \cdots Q_k A$ is backwards stable, and hence $B = \widetilde{Q_1 \cdots Q_k A} = Q_1 \cdots Q_k (A + \delta A_k)$ for some $\|\delta A_k\| = O(\varepsilon_{\text{machine}}) \|A\|$. Also, from above, $\widetilde{QB} = Q(B + \delta B)$ for some $\|\delta B\| = O(\varepsilon_{\text{machine}}) \|B\|$. Furthermore, $\|B\| = \widetilde{\|Q_1 \cdots Q_k (A + \delta A_k)\|} = \|A + \delta A_k\| \le \|A\| + \|\delta A_k\| = \|A\| [1 + O(\varepsilon_{\text{machine}})]$. Hence, $QQ_1 \cdots Q_k A = \widetilde{QB} = Q[Q_1 \cdots Q_k (A + \delta A_k) + \delta B] = QQ_1 \cdots Q_k (A + \delta A)$ where $\delta A = \delta A_k + [Q_1 \cdots Q_k]^{-1} \delta B$ and $\|\delta A\| \le \|\delta A_k\| + \|\delta B\| = O(\varepsilon_{\text{machine}}) \|A\|$. Q.E.D.

(ii) Consider XA, where X is some rank-1 matrix xy^* and A has rank > 1. The product XA has rank 1 in exact arithmetic, but after floating-point errors it is unlikely that $\widetilde{XA}$ will be exactly rank 1. Hence it is not backwards stable, because $X\tilde{A}$ will be rank 1 regardless of $\tilde{A}$, and thus is $\neq \widetilde{XA}$. (See also example 15.2 in the text.)

18.335J / 6.337J Introduction to Numerical Methods
Fall 2010