18.335 Take-Home Midterm Exam Solutions: Spring 2021

Problem 1: (34 points)

As suggested, let's define a recurrence for Horner's rule so that $s_k = c_k + x s_{k+1}$ with $s_{n-1} = c_{n-1}$, so that $f = s_0$, and hence the corresponding floating-point algorithm for inputs in $\mathbb{F}$(as in the summation notes from class, this is just a convenience — inputs in $\mathbb{R}$ would just give some additional $1+\varepsilon$ factors that we could trivially incorporate into the same analysis) is

$$\tilde{s}_{n-1} = c_{n-1}$$

$$\tilde{s}_k = c_k \oplus (x \otimes \tilde{s}_{k+1}) = [c_k + x\tilde{s}_{k+1}(1+\varepsilon_k)]\,(1+\varepsilon'_k)$$

for some $|\varepsilon_k| \leq \varepsilon_{\text{machine}}$ and $|\varepsilon'_k| \leq \varepsilon_{\text{machine}}$, by the fundamental axiom of floating-point arithmetic.

First Solution

Hence, dropping $O(\varepsilon_{\text{machine}}^2)$ terms, we have:

$$\tilde{s}_k = \underbrace{\left(c_k + c_k\varepsilon'_k + x\tilde{s}_{k+1}[\varepsilon_k + \varepsilon'_k] + O(\varepsilon_{\text{machine}}^2)\right)}_{\tilde{c}_k} + x\tilde{s}_{k+1} = \sum_{j=k}^{n-1} \tilde{c}_j x^{j-k},$$

where we have defined $\tilde{c}_k$. By construction

$$\boxed{\tilde{f}(c,x) = \tilde{s}_0 = f(\tilde{c},x),}$$

i.e. the output of $\tilde{f}$ is equal to the exact polynomial output with altered coefficients $\tilde{c}$. To establish backwards stability, we now need to show that $\|\tilde{c}-c\| = \|c\|O(\varepsilon_{\text{machine}})$ in some norm. Similar to the summation analysis in class, we will choose the L^1 norm for convenience. In particular,

$$|\tilde{c}_k - c_k| \leq |c_k|\,|\varepsilon'_k| + |\varepsilon_k + \varepsilon'_k|\,|x|\,|\tilde{s}_{k+1}| + O(\varepsilon_{\text{machine}}^2).$$

Now, use the fact that

$$|\tilde{s}_k| = \left|\sum_{j=k}^{n-1} \tilde{c}_j x^{j-k}\right| \leq \left|\sum_{j=k}^{n-1} \tilde{c}_j\right| \left(\max\{1, |x|^{n-k}\}\right) \leq \|\tilde{c}\|_1 \left(\max\{1, |x|^{n-1}\}\right)$$

to obtain

$$|\tilde{c}_k - c_k| \leq |c_k|\,|\varepsilon'_k| + |\varepsilon_k + \varepsilon'_k|\,|x|\,\|\tilde{c}\|_1 \left(\max\{1,|x|^{n-1}\}\right) + O(\varepsilon_{\text{machine}}^2) \leq |c_k|\varepsilon_{\text{machine}} + 2|x|\varepsilon_{\text{machine}}\|\tilde{c}\|_1 \left(\max\{1,|x|^{n-1}\}\right) + O(\varepsilon_{\text{machine}}^2)$$

and hence

$$\|\tilde{c} - c\|_1 \leq \|c\|_1\varepsilon_{\text{machine}} + 2n|x|\varepsilon_{\text{machine}}\|\tilde{c}\|_1 \left(\max\{1,|x|^{n-1}\}\right) + O(\varepsilon_{\text{machine}}^2) = \|c\|_1 O(\varepsilon_{\text{machine}}) + \|\tilde{c}\|_1 O(\varepsilon_{\text{machine}})$$

where we can put any higher-order terms and c-independent coefficients into the $\varepsilon_{\text{machine}}$. Finally, exactly as in Sec. 5.1 of the summation-stability notes from class, $\|\tilde{c}\|_1 O(\varepsilon_{\text{machine}}) = \|c\|_1 O(\varepsilon_{\text{machine}})$, because the difference is higher-order in $\varepsilon_{\text{machine}}$, so we finally have

$$\boxed{\|\tilde{c} - c\|_1 = \|c\|_1 O(\varepsilon_{\text{machine}})}$$

and our backwards-stability proof is complete.

Better Solution

The above solution isn't completely satisfying, because the $\|\tilde{c}-c\|_1$ error bound depends on x, which would be a problem if we wanted to prove backwards stability with respect to *both* c and x. Instead, we can apply a different formulation. Start with the formula for $\tilde{s}_k$ at the top, and collect terms as:

$$\tilde{s}_k = c_k(1+\varepsilon_k') + x\tilde{s}_{k+1}(1+\varepsilon_k)(1+\varepsilon_k').$$

By induction on n, we will show that

$$\tilde{s}_k = \sum_{j=k}^{n-1}\left[c_j(1+\varepsilon_j')x^{j-k}\prod_{\ell=k}^{j-1}(1+\varepsilon_\ell)(1+\varepsilon_\ell')\right].$$

It is trivially true for $k=n-1$ (with $\varepsilon_{n-1}'=0$). Proceeding by (downward) induction on k, we have

$$\begin{aligned}\tilde{s}_k &= c_k(1+\varepsilon_k') + x(1+\varepsilon_k)(1+\varepsilon_k')\sum_{j=k+1}^{n-1}\left[c_j(1+\varepsilon_j')x^{j-k-1}\prod_{\ell=k+1}^{j-1}(1+\varepsilon_\ell)(1+\varepsilon_\ell')\right]\\ &= c_k(1+\varepsilon_k') + \sum_{j=k+1}^{n-1}\left[c_j(1+\varepsilon_j')x^{j-k}\prod_{\ell=k}^{n-1}(1+\varepsilon_\ell)(1+\varepsilon_\ell')\right]\end{aligned}$$

and the result follows. Therefore,

$$\tilde{f}(c,x) = \tilde{s}_0 = \sum_{j=0}^{n-1}\left[c_j(1+\varepsilon_j')x^j\prod_{\ell=0}^{j-1}(1+\varepsilon_\ell)(1+\varepsilon_\ell')\right].$$

We can define (differently from the first solution above!):

$$\boxed{\tilde{c}_j = c_j(1+\varepsilon_j')\prod_{\ell=0}^{j-1}(1+\varepsilon_\ell)(1+\varepsilon_\ell')} = c_j\left(1+\varepsilon_j' + \sum_{\ell=0}^{j-1}\varepsilon_\ell + \sum_{\ell=0}^{j-1}\varepsilon_\ell'\right) + O(\varepsilon_{\text{machine}}^2),$$

and it follows that $\tilde{f}(c,x) = \sum_{j=0}^{n-1}\tilde{c}_j x^j = f(\tilde{c},x)$ as desired. Furthermore, we obtain

$$\left|\tilde{c}_j - c_j\right| \le (2j+1)\varepsilon_{\text{machine}}|c_j| + O(\varepsilon_{\text{machine}}^2)$$

and hence

$$\|\tilde{c}-c\|_1 \le (2n-1)\varepsilon_{\text{machine}}\|c\|_1 + O(\varepsilon_{\text{machine}}^2) = \|c\|_1 O(\varepsilon_{\text{machine}})$$

as desired, this time with a coefficient independent of x. (So, if we wanted, we could also say that it is backwards-stable with respect to both c and x simultaneously, setting $\tilde{x}=x$.)

Problem 2: (20+13 points)

Suppose that

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \overline{\beta_1} & \alpha_2 & \beta_2 & & \\ & \overline{\beta_2} & \ddots & \ddots & \\ & & \ddots & \alpha_{m-1} & \beta_{m-1} \\ & & & \overline{\beta_{m-1}} & \alpha_m \end{pmatrix}$$

is an $m\times m$ Hermitian ($T=T^*$) complex tridiagonal matrix.

(a) Let

$$D = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_m \end{pmatrix}.$$

Then

$$\hat{T} = D^{-1}TD = \begin{pmatrix} \alpha_1 & \beta_1 \frac{d_2}{d_1} & & & \\ \overline{\beta_1} \frac{d_1}{d_2} & \alpha_2 & \beta_2 \frac{d_3}{d_2} & & \\ & \overline{\beta_2} \frac{d_2}{d_3} & \ddots & \ddots & \\ & & \ddots & \alpha_{m-1} & \beta_{m-1} \frac{d_m}{d_{m-1}} \\ & & & \overline{\beta_{m-1}} \frac{d_{m-1}}{d_m} & \alpha_m \end{pmatrix}.$$

Note that α_k is already real since $T = T^*$, so we only need to choose D to make the off-diagonal terms real. Write β_k in polar form as $\beta_k = r_k e^{i\phi_k}$. In $D^{-1}TD$, this is rescaled in the upper diagonal to $\beta_k \frac{d_{k+1}}{d_k}$. To make this purely real, we therefore want $\frac{d_{k+1}}{d_k} = e^{-i\phi_k}$ (or $-e^{-i\phi_k}$ would also work). Hence, let us define the diagonals by the recurrence:

$$d_1 = 1 \text{ (arbitrary choice)},$$
$$d_{k+1} = d_k e^{-i\phi_k} = e^{-i\sum_{j\le k}\phi_j}.$$

This also fixes the lower diagonal, since $\frac{d_k}{d_{k+1}} = e^{+i\phi_k}$ and hence:

$$\overline{\beta_k} \frac{d_k}{d_{k+1}} = r_k e^{-i\phi_k} \frac{d_k}{d_{k+1}} = r_k.$$

D is clearly unitary since the diagonal entries all have $|d_k| = 1$ (as long as we chose $|d_1| = 1$).

(b) We should apply QR iterations to $\hat{T}$, which gives the same result because T and $\hat{T}$ are **similar** matrices (differeing only by a unitary change of basis: they have the same eigenvalues, and the eigenvectors differ by a factor of D), and operating on $\hat{T}$ is faster because then the QR factors will also be purely real, and operations on real numbers take fewer floating-point operations than corresponding operations on complex numbers (adding complex numbers takes 2 flops and multiplying them takes 6 flops, versus 1 addition and 1 multiplication, respectively, for real numbers). (It also requires half as much memory and hence will be more cache-efficient.)

Problem 3: (19+14 points)

Suppose A is an $m \times n$ matrix with $n > m$ and rank m (linearly independent rows): a "wide" matrix. In this case, $Ax = b$ has infinitely many solutions x for any right-hand side $b \in \mathbb{C}^m$ (it is an *underdetermined* system of equations). We compute the QR factorization of the conjugate-transpose $A^* = QR = \hat{Q}\hat{R}$ by some backwards-stable algorithm (e.g. Householder QR), where Q is $n \times n$ and R is $n \times m$, and $\hat{Q}$ is the "thin" QR (the first m columns of Q) and $\hat{R}$ is correspondingly the first m rows of R.

(a) As suggested by the hint, let's write the solutions in the Q basis as

$$x = Qy = \begin{pmatrix} \hat{Q} & Q_\perp \end{pmatrix} \begin{pmatrix} \hat{y} \\ y_\perp \end{pmatrix} = \hat{Q}\hat{y} + Q_\perp y_\perp,$$

where we have broken up y into the m coefficients of $\hat{Q}$ and the $n-m$ coefficients of $Q_\perp$. Then

$$b = Ax = \hat{R}^*\hat{Q}^*x = \hat{R}^*\hat{Q}^*\left(\hat{Q}\hat{y} + Q_\perp y_\perp\right) = \hat{R}^*\hat{y},$$

since $\hat{Q}^*\hat{Q} = I$ (being an orthonormal basis) and $\hat{Q}^*Q_\perp = 0$ since $Q_\perp$ spans the orthogonal complement of $\hat{Q}$, which spans the row space $C(A^*)$, and hence $Q_\perp$ spans $C(A^*)^\perp = N(A)$. This tells us several things.

(i) $\hat{y}$ is uniquely determined by solving the $m \times m$ lower-triangular system $\hat{R}^*\hat{y} = b$.

(ii) any value of $y_\perp$ is allowed. (Equivalently, the term $Q_\perp y_\perp$ represents anything in the null space of A.)

(iii) $\|x\|_2^2 = (\hat{Q}\hat{y} + Q_\perp y_\perp)^*(\hat{Q}\hat{y} + Q_\perp y_\perp) = \|\hat{y}\|_2^2 + \|y_\perp\|_2^2 \geq \|\hat{y}\|_2^2$, which is clearly minimized for $y_\perp = 0$.

(iv) Hence the minimum-norm solution is $\boxed{x = \hat{Q}\hat{y} = \hat{Q}\left(\hat{R}^*\right)^{-1} b}$.

(b) As suggested, suppose we are *given* the $A^* = QR$ factorization computed by Householder QR, and now we want to work out the additional cost of computing our minimum-norm solution x above. Then the additional. cost is to (i) solve $\hat{R}^*\hat{y} = b$ and (ii) multiply $\hat{Q}\hat{y}$. Computation (i) can use forward-substitution since $\hat{R}^*$ is lower-triangular, so it is $\Theta(m^2)$.

For computation (ii), remember that Householder QR does not actually compute Q explicitly — rather, it computes a set of reflectors and provides an efficient method (algorithm 10.3 in the book) to multiply Q by vectors as a sequence of reflections. Here, we need to apply that algorithm 10.3, but we are multiplying by $y = \begin{pmatrix} \hat{y} \\ 0 \end{pmatrix}$ so we can maybe save a few operations by skipping multiplications with the zero components. Algorithm 10.3 initializes $x = y$ and then computes $x_{k:n} = x_{k:n} - 2v_k(v_k^* x_{k:n})$ for $k = m$ down to 1 (where I've swapped m and n compared to the book). Here, we hope that this simplifies because $y_{m+1:n} = 0$. However, after the *very first* ($k = m$) step of he algorithm, in which $v_m^* x_{m:n} = \overline{v_m[m]}\hat{y}[m] \neq 0$ in general, we subtract a multiple of $2v_k$ from $x_{m:n}$ and *all* the components become nonzero for subsequent steps. So, in the end the savings from the zero entries of y are asymptotically negligible and the cost of step (ii) is $\Theta(nm)$ exactly as in the book.

(Note that even if you assume we have $\hat{Q}$ explicitly, e.g. if you used modified Gram–Schmidt, multiplying it by a vector still has $\Theta(nm)$ cost.)

Hence, the overall complexity is $\boxed{\Theta(m^2) + \Theta(nm) = \Theta(mn)}$: since $n > m$ the step (ii) dominates.