Program 1

David Miller
MAD5403: Foundations of Computational Math I

December 4, 2017

Note: Solutions to analysis problems can be found in sections 3 and 5 labeled according to their respective question on the writeup.

1 Executive Summary

Horner's method is a fast and efficient way of evaluating the polynomial $p_n(x)$. In this program we propose three polynomials in which we numerically approximate via Horner's method and compare it to the "exact" value. Floating point error with respect to single and double precision is also computed and analyzed against a prior error bounds.

2 Statement of Problem

The evaluation of a polynomial $p_n(x)$ is often used in numerics. Thus it is important to have a robust, fast, and efficient algorithm for doing so. Horner's method provides this given the coefficients of the polynomial are known. Horner's method requires $2n$ floating point operations (*flops*), its forward error requires $2n + 1$ *flops*, and its running error evaluation requires $4n+1$ *flops*. This leads to a very desired $\mathcal{O}(n)$ time complexity as well $\mathcal{O}(n)$ memory complexity (shown later). In this program we evaluate three polynomials along with their associated errors and "exact: solutions

- $p(x) = (x - 2)^9$
- $p(x) = (x + 5)^6$
- $p(x) = (x - 8)^3$

where we use the product form of the polynomial as the "exact" solution. The effect of single and double precision are also taken into consideration.

3 Description of Mathematics

An error analysis of Horner's rule shows that the computed value of the polynomials satisfies

$$\tilde{c}_0 = (1 + \theta_1)\alpha_0 + (1 + \theta_3)\alpha_1 x + \ldots + (1 + \theta_{2n-1}\alpha_{n-1}x^{n-1} + (1 + \theta_{2n}\alpha_n)x^n \tag{1}$$

where $|\theta_k| \leq \gamma_k$ (Higham 2002 Accuracy and Stability of Numerical Algorithms, Second Edition). The pattern on the subscript is odd numbers, i.e., increment of 2, until the last which is even, i.e., last increment is 1.
Let

$$\tilde{(p_n(x))} = |\alpha_0| + |\alpha_1|x + \ldots + |\alpha_n|x^n.$$

From this we can show that

$$\frac{|p_n(x) - \tilde{c}_0|}{|p_n(x)|} \leq \gamma_{2n} \frac{\tilde{p}_n(|x|)}{|p_n(x)|} \tag{2}$$

and therefore

$$\kappa_{rel} = \frac{\tilde{p}(|x|)}{|p(x)|}$$

is a relative condition number for perturbations to the coefficients bounded by γ_{2n}

PROBLEM 1.1 Expanding the numerator of the LHS of (3) we get

$$\begin{aligned}
|p_n(x) - \tilde{c}_0| &= \big|(\alpha_0 + \alpha_1 x \ldots + \alpha_{n-1}x^{n-1} + \alpha_n x^n) \\
&\quad - ((1+\theta_1)\alpha_0 + (1+\theta_3)\alpha_1 x + \ldots + (1 + \theta_{2n-1}\alpha_{n-1}x^{n-1} + (1+\theta_{2n}\alpha_n)x^n))\big| \\
&= |-\alpha_0\theta_1 - \theta_3\alpha_1 x - \ldots - \theta_{2n-1}\alpha_{n-1}x^{n-1} - \theta_{2n}\alpha_n x^n| \\
&\leq |\alpha_0\theta_1| + |\theta_3\alpha_1 x| + \ldots + |\theta_{2n-1}\alpha_{n-1}x^{n-1}| + |\theta_{2n}\alpha_n x^n| \\
&\leq |\max_{1\leq i\leq 2n}(\theta_i)||\sum_{i=0}^{n}\alpha_i x^i| \\
&\leq |\max_{1\leq i\leq 2n}\theta_i| \sum_{i=0}^{n}|\alpha_i||x|^i \\
&= |\theta_{2n}|\tilde{p}(|x|) \\
&\leq \gamma_{2n}\tilde{p(|x|)}
\end{aligned}$$

Since we have that $|p_n(x) - \tilde{c}_0| \leq \gamma_{2n}\tilde{p}(|x|)$ it follows directly that $\frac{|p_n(x)-\tilde{c}_0|}{|p_n(x)|} \leq \gamma_{2n}\frac{\tilde{p}(|x|)}{|p_n(x)|}$. From this result we can see that $\frac{\tilde{p}(|x|)}{|p(x)|}$ is an amplification factor to the upper bound of relative error γ_k. We can then conclude that $\kappa_{rel} = \frac{\tilde{p}(|x|)}{|p(x)|}$ by definition.

PROBLEM 1.2 The computed result using Horner's rule and its associated a priori bound, $|\theta_k| \leq \gamma_{2k}$, is backward stable. The computed result $\tilde{c}_0$ is the exact solution to the polynomial

$$(1+\theta_1)\alpha_0 + (1+\theta_3)\alpha_1 x + \ldots + (1+\theta_{2n-1})\alpha_{n-1}x^{n-1} + (1+\theta_{2n})\alpha_n x^n$$

where each θ_k is the perturbation to the coefficients (the data). Since θ_k is bounded above by $\gamma_k = \frac{k\epsilon_M}{1-k\epsilon_M}$, $k\epsilon_M \approx k \times 10^{-7}$ in single precision, and $\approx k \times 10^{-16}$ in double precision, we have that γ_k is very small and grows at a very slow rate. Therefore we can say that $\tilde{c}_0$ is the exact solution to

$$(1+\theta_1)\alpha_0 + (1+\theta_3)\alpha_1 x + \ldots + (1+\theta_{2n-1})\alpha_{n-1}x^{n-1} + (1+\theta_{2n})\alpha_n x^n$$

for some $\tilde{d}$ in some small neighborhood around our data d.

PROBLEM 1.3 For $p_n(x)$ to be perfectly conditioned with respect to perturbations to the coefficients of the monomial form we must have that $\frac{\tilde{p}(|x|)}{|p(x)|} = 1$. If we expand this fraction we can see when this happens:

$$\begin{aligned}
\frac{\tilde{p}(|x|)}{|p(x)|} &= \frac{|\alpha_0| + |\alpha_1||x| + \ldots |\alpha_{n-1}||x|^{n-1} + |\alpha_n||x|^n}{|\alpha_0 + \alpha_1 x + \ldots \alpha_{n-1}x^{n-1} + \alpha_n x^n|} \\
&= \begin{cases} \frac{|\alpha_0|+|\alpha_1|x+\ldots+|\alpha_{n-1}x^{n-1}+|\alpha_n|x^n|}{|\alpha_0+\alpha_1 x+\ldots\alpha_{n-1}x^{n-1}+\alpha_n x^n|} & x > 0 \\ \frac{|\alpha_0|+|\alpha_1||x|+\ldots|\alpha_{n-1}||x|^{n-1}+|\alpha_n||x|^n}{|\alpha_0-\alpha_1|x|+\alpha_2 x^2-\alpha_3|x|^3+\ldots|} & x < 0 \\ 1 & x = 0 \end{cases} \\
&= \begin{cases} 1 & x > 0, sign(\alpha_i) = sign(\alpha_j)\,\forall i,j \\ 1 & x < 0, sign(\alpha_i) = sign(\alpha_{i=2}), \\ & sign(\alpha_i) \neq sign(\alpha_{i+1})\,\forall i \\ 1 & x = 0 \end{cases}
\end{aligned}$$

Therefore if we evaluate $p_n(x)$ for some $x \in \mathbb{R}^+$ we must have that all the coefficients have the same sign for perfectly conditioned. However, if we evaluate $p_n(x)$ for some $x \in \mathbb{R}^-$ then we must have coefficients with alternating signs for perfectly conditioned. The case when we evaluate $p_n(x)$ at $x = 0$ is perfectly conditioned.

PROBLEM 1.4 Now let's assume that our polynomial is not perfectly conditioned. We can then model the behavior of our conditioning number as it approaches some root x_0 for the polynomial $p(x)$:

$$\begin{aligned}
\lim_{x\to x_0} \kappa &= \lim_{x\to x_0} \frac{\tilde{p}(|x|)}{|(p(x)|} \\
&= \lim_{x\to x_0} \frac{|\alpha_0| + |\alpha_1||x_0| + \ldots + |\alpha_{n-1}||x_0|^{n-1} + |\alpha_n||x_0|^n}{|\alpha_0 + \alpha_1 x_0 + \ldots + \alpha_{n-1}x_0^{n-1} + \alpha_n x_0^n|}
\end{aligned}$$

The last expression will blow up if x_0 is not a root for $\tilde{p}(|x|)$. Therefore I would expect $p_n(x)$ to be relatively sensitive when we are near some point x_0 such that $p(x_0) = 0$ and $\tilde{p}(|x|) \neq 0$ since the conditioning number κ will be some large number.

PROBLEM 3 Forward error provides a decent bound, but a much tighter bound for Horner's method can be computed if we calculate some error at each step, we will define this as the running error. The computed value on step i of Horner's rule satisfies

$$(1 + \epsilon_i)\hat{c}_i = x\hat{c}_{i+1}(1 + \delta_i) + \alpha_i, \quad |\delta_i| \leq u, |\epsilon_i| \leq u$$

If we let $\hat{c}_i = c_i + e_i, e_n = 0$ and c_i the exact value of the parameter in Horner's rule evaluated

in exact arithmetic. Working off of this we can derive the forward error at each step:

$$\begin{aligned}
\hat{c}_i + \epsilon_i \hat{c}_i &= x\hat{c}_{i+1} + \delta_i x\hat{c}_{i+1} + \alpha_i \\
\Rightarrow\ \hat{c}_i + e_i + \epsilon_i \hat{c}_i &= x(\hat{c}_{i+1} + e_{i+1}) + \delta_i x(c_{i+1} + e_{i+1}) + \alpha_i \\
&= x\hat{c}_{i+1} + xe_{i+1} + \delta_i x c_{i+1} + \delta_i x e_{i+1} + \alpha_i \\
\Rightarrow\ e_i + \epsilon_i \hat{c}_i &= xe_{i+1} + \delta_i x c_{i+1} + \delta_i x e_{i+1} \\
e_i &= xe_{i+1} - \epsilon_i \hat{c}_i + \delta_i x(c_{i+1} + e_{i+1}) \\
\Rightarrow\ e_i &= xe_{i+1} - \epsilon_i \hat{c}_i + \delta_i x\hat{c}_{i+1}
\end{aligned}$$

Now that we have what the running error is for each step of Horner's rule we would like to find an upper bound on this. Given

$$\beta_i = |x|\beta_{i+1} + |x||\hat{c}_{i+1}| + |\hat{c}_i|$$

and $\beta_n = e_n = 0$ we can compute B_{n-1} along with e_{n-1}

$$\begin{aligned}
\beta_{n-1} &= |x|\beta_n + |x||\hat{c}_n| + |\hat{c}_{n-1}| = |x||\hat{c}_n| + |\hat{c}_{n-1}| \\
e_{n-1} &= xe_n - \epsilon_{n-1}\hat{c}_{n-1} + \delta_{n-1}x\hat{c}_n = \delta_{n-1}x\hat{c}_n - \epsilon_{n-1}\hat{c}_{n-1}
\end{aligned}$$

Its easy to see from the above that $|e_n| \leq u\beta_n$ where u is machine epsilon. It also follows from continuing this iteration that this inequality will hold for all $i = 0, 1, \ldots, n$.

4 Description of the Algorithm and Implementation

Horner's method evaluates $p_n(x)$ by building the polynomial inside out. Inside the `for loop` is has 2 *flops* for n iterations, therefore the time complexity is $\mathcal{O}(n)$. The coefficients must be stored in some data structure as well as some variable to store the value computed at each step, therefore the memory complexity is $\mathcal{O}(n)$.

Input: Point x, Coefficients $\{\alpha_0, \ldots, \alpha_n\}$
Output: Evaluation of polynomial $p(x)$

Algorithm 1 Horner's Method

1: $q_n(x) \leftarrow \alpha_n$
2: **for** $i = n - 1 : -1 : 0$ **do**
3: $\quad q_i(x) \leftarrow xq_{i+1}(x) + \alpha_i$
4: **end for**
5: $p(x) \leftarrow q_0(x)$
6: **return** $p(x)$

The forward error computation is similar to Horner's method. The only difference is taking the absolute value of x and α_i at each iteration. There is one extra flop and that is at the end when multiplying by γ_{2n}. Therefore we have that the time and memory complexity is $\mathcal{O}(n)$.

Input: Point x, Coefficients $\{\alpha_0, \ldots, \alpha_n\}$
Output: Forward Error ϵ_F

Algorithm 2 Forward Error

$\tilde{p}_n(x) \leftarrow \alpha_n$
for $i = n - 1 : -1 : 0$ **do**
    $\tilde{p}_i(x) \leftarrow |x|\tilde{p}_{i+1}(x) + |\alpha_i|$
end for
$\epsilon \leftarrow \gamma_{2n} \star \tilde{p}_0(x)$
return ϵ_F

Where we define $\gamma_k = k\epsilon_M/(1 - k\epsilon_M)$ for some machine epsilon ϵ_M.

The running error calculation (not including Horner's method in the calculation) performs 4 *flops* per iteration. It performs 3 before before an evaluation of Horner's method is done then another one when we add the result to the running error. Therefore the time complexity of running error is $\mathcal{O}(n)$. Since we have to store the coefficients and a variable we have that the memory complexity is $\mathcal{O}(n)$.

Input: Point x, Coefficients $\{\alpha_0, \ldots, \alpha_n\}$
Output: Running Error ϵ_R

Algorithm 3 Running Error

$q_n(x) \leftarrow \alpha_n$
$\epsilon_n \leftarrow 0$
for $i = n - 1 : -1 : 0$ **do**
    $\epsilon_i \leftarrow |x||e_{i+1}| + |x||q_{i+1}(x)|$
    $q_i(x) \leftarrow xq_{i+1}(x) + \alpha_i$
    $\epsilon_i \leftarrow \epsilon_{i+1} + q_i(x)$
end for
$\epsilon_R \leftarrow \epsilon_M\epsilon_0$
return ϵ_R

Where we define ϵ_M as machine epsilon for the specified precision.

5 Description of the Experiment Design and Results

The execution of the code can be summarized as follows

- The user specifies what polynomial and precision to use; these computations are done in C++.

  Single precision: `g++ -std=c++11 -DSINGLE=1 main.cpp`

  Double precision: `g++ -std=c++11 -DDOUBLE=1 main.cpp`

- The computations are written to some file

- The data is read and graphed; this is done with Python.

The `main function` simply iterates over some domain using a Δx to move forward. Within these iterations our three main algorithms are called and their results are written to a specified file. The logic of the program is simple and easy to understand.
The three polynomials that were chosen are

$$p(x) = (x-8)^3, \quad p(x) = (x+5)^6, \quad p(x) = (x-2)^9$$

. We choose these polynomials for several reasons:

1. Investigate how the degree of the polynomial affects the accuracy and error of Horner's method,
2. Investigate how the magnitude and sign of the root affect the accuracy and errors of Horner's method
3. Investigate the sensitivity (conditioning) of Horner's method near the roots.

From figures (1) - (3) we can see that the magnitude of our errors are relative to both the degree of the polynomial and the magnitude of our root. Figure (3) has much tighter bounds than that of figure (1). Another way to verify this is to notice that our forward error bound magnitude increases as x increases in figure (1) and increases as x decreases in figure (2) (this is because we are in a negative domain).

PROBLEM 2.3 The fact that we used single precision when calculating the a priori bound is of importance. This is because $\epsilon_M \approx 10^{-7}$ in single precision while $\epsilon_M \approx 10^{-16}$ in double precision. Thus double precision bounds will be many magnitudes smaller than that of single precision resulting in forward and error running bounds using double precision being too strict for Horner's method in single precision. This is clearly shown in figure (4).

6 Conclusions

From our results and analysis we can see that Horner's method evaluates $p_n(x)$ in $\mathcal{O}(n)$ time complexity and $\mathcal{O}(n)$ memory complexity with errors proportional to n and the precision used. Our forward error bounds provide a lenient bound while our running error bound provides a bound approximately 20 times tighter than forward error. Horner's method, when evaluated in the same precision as the error, falls within the error bounds illustrating that it is a very accurate algorithm. We also verified that computing the "exact" evaluation using the product form in single precision yields almost identical results to that of double precision. Lastly we illustrated that Horner's method has varying sensitivity with respect to x. We confirmed that higher conditioning numbers incur high sensitivity to perturbations in our data. If we want a polynomial with a low conditioning number than we should try to minimize the degree and magnitude of the root.

7 Figures

$$p(x) = (x-2)^9$$

(a) Global figure showing the "exact" polynomial evaluation, Horner's method, forward error bounds, and running error bounds on the entire domain [1.91,2.1].

(b) Local figure showing the "exact" polynomial evaluation, Horner's method, and running error bounds on the left boundary, right boundary, and middle.

$$p(x) = (x+5)^6$$

(a) Global figure showing the "exact" polynomial evaluation, Horner's method, forward error bounds, and running error bounds on the entire domain [-5.09,-4.9].

(b) Local figure showing the "exact" polynomial evaluation, Horner's method, and running error bounds on the left boundary, right boundary, and middle.

$$p(x) = (x - 8)^3$$

(a) Global figure showing the "exact" polynomial evaluation, Horner's method, forward error bounds, and running error bounds on the entire domain [7.91,8.1].

(b) Local figure showing the "exact" polynomial evaluation, Horner's method, and running error bounds on the left boundary, right boundary, and middle.

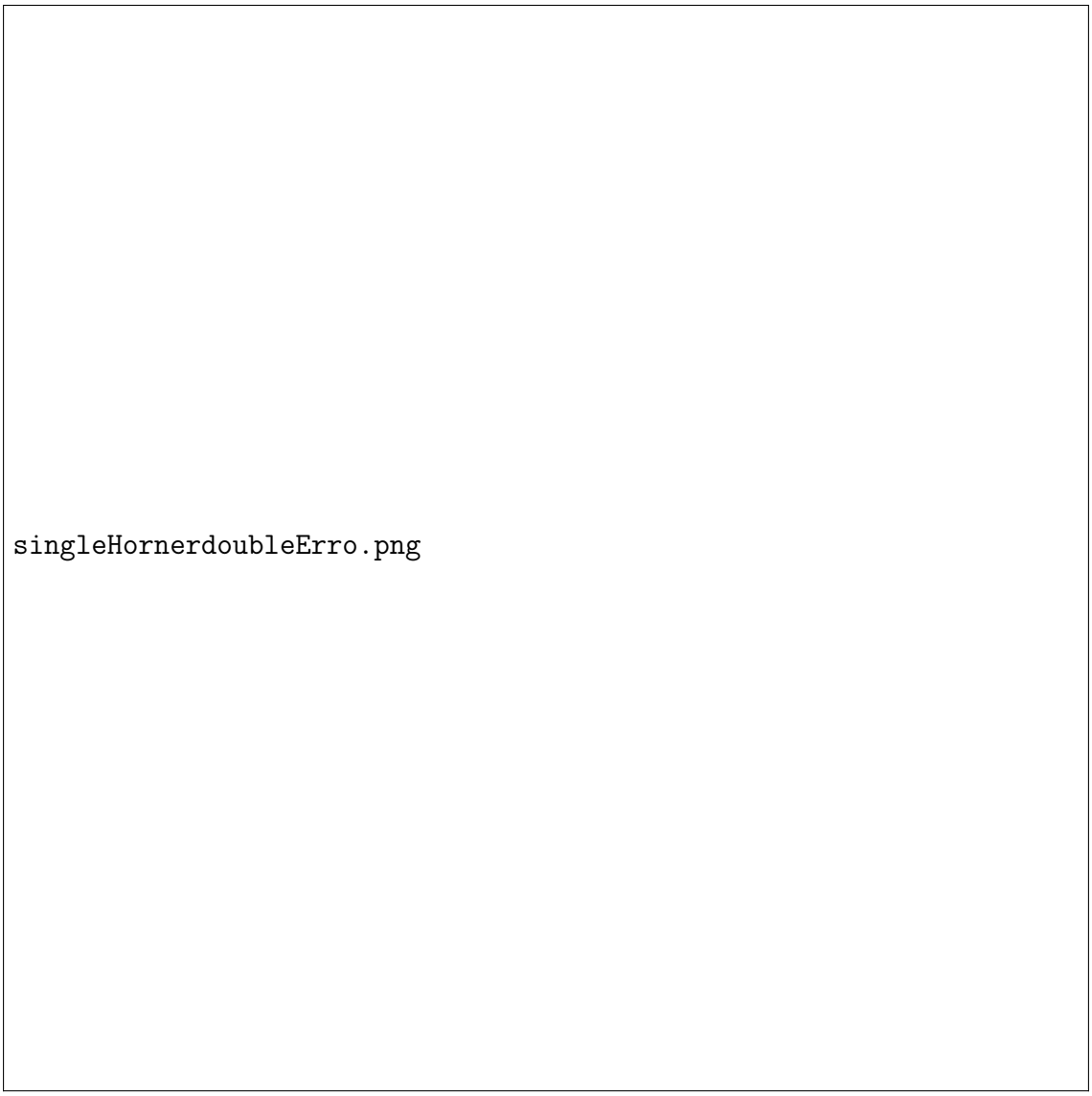

Figure 4: Horner's method calculated using single precision and error bounds using double precision. This does not yield useful results since Horner's method returns values outside of the bounds.

8 References

I used a suggested format for my code provided at

https://stackoverflow.com/questions/14511910/switching-between-float-and-double-precision-at-compile-time