Horner's Rule Analysis, Program 1

Colton Piper

October 10th, 2017

1 Executive Summary

This report focuses on many aspects of Horner's Rule. Horner's rule is a recursive algorithm for evaluating polynomials at a specific point. We will analyze the algorithm and discover different properties of the rule when using single and double precision floating point arithmetic. For example we will show that Horner's Rule is backwards stable and that we can get two different error bounds. We will also be plotting some example polynomials with their respective bounds to show how Horner's rule does stay well within the bounds we find as it should.

2 Statement of the Problem

Using computers to solve problems numerically has been a great addition to the collection of tools we have today. Many problems that would take months to solve can be solved simply and quickly with the use of a computer. But computers are not magic and it is good to know when we can use them to solve problems and how much error, if any, a computer will make as it only has a finite number of numbers.

Polynomials are used in numerical mathematics all the time. Thus it is a good idea to study different techniques of how to evaluate them and the pros and cons of such techniques. The technique we will be studying here is Horner's rule. More specifically finding error bounds that work well with using this rule to evaluate polynomials.

First we will find a relative condition number for this algorithm and then we will also show that it is backward stable which is a big conclusion numerical analysis. The condition number can tell us if the problem itself is well or ill-conditioned. This is important because an ill-conditioned problem can lead to bad and inaccurate results. Now backwards stability tells us that the computation we get is the actual solution of another problem and also that the algorithm is forward stable as well. This means that we will have a bound for the absolute error.

Next we will code Horner's rule and evaluate a polynomial at many points and show that the errors all lie within the forward error bound. Then we will derive a new bound called a running error bound. We will again code this new running error bound which has error accumulating at every recursive step and plot it with the forward error bounds. What we will find is that it is a much tighter bound around high degree polynomials where the roots are close to 0.

3 Description of the Mathematics

The first problem wants us to show that the relative error of Horner's rule is

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

and therefore the relative condition number is $\widetilde{p}(|x|)/|p(x)|$. Well we can see from using equation (2) and the fact that $|\theta_k| \leq \gamma_k$ we can see that

$$\begin{aligned}
|p_n(x) - \hat{c}_0| &= |\theta_1 \alpha_0 + \theta_3 \alpha_1 x + \theta_5 \alpha_2 x^2 + \cdots + \theta_{2n-1} \alpha_{n-1} x^{n-1} + \theta_{2n} \alpha_n x^n| \\
&\leq |\theta_1 \alpha_0| + |\theta_3 \alpha_1 x| + |\theta_5 \alpha_2 x^2| + \cdots + |\theta_{2n-1} \alpha_{n-1} x^{n-1}| + |\theta_{2n} \alpha_n x^n| \\
&\leq \gamma_{2n}(|\alpha_0| + |\alpha_1||x| + |\alpha_2||x|^2 + \cdots + |\alpha_{n-1}||x|^{n-1} + |\alpha_n||x|^n) = \gamma_{2n} \widetilde{p}_n(|x|) \\
\frac{|p_n(x) - \hat{c}_0|}{|p_n(x)} &\leq \gamma_{2n} \frac{\widetilde{p}_n(|x|)}{|p_n(x)|}.
\end{aligned}$$

Now from class notes we know that the relative condition number is the maximum relative error divided by the relative perturbation. Which means it is a worst case bound on the impact of perturbations in a neighborhood around the original problem, which will occur because of floating point representation and arithmetic. Now we know that γ_{2n} is a bound for the relative perturbations around the problem and $\gamma_k \leq \gamma k + 1$ for $k \leq 2n$. Thus if we divide both sides of the inequality by γ_{2n} we will have a bound for the amount a relative perturbation can affect the original problem. Therefore we get that

$$k_{rel} = \frac{\widetilde{p}(|x|)}{|p(x)|}.$$

Next is showing that the the equation

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

where $|\theta_k| \leq \gamma_k$ is backwards stable. Backwards stable is showing that the computed result is the exact result of solving the problem with nearby input data. The relative error has to be bounded by $k_{rel}p(n)u$ where $p(n)$ is a mildly growing function of the degree of the polynomial and u is the unit roundoff. Now we already have solved for k_{rel}. Notice that from the previous that γ_{2n} is equivalent to the $p(n)u$. This is because $\gamma_{2n} = \frac{2n}{1-2nu}u$. Thus $p(n) = \frac{2n}{1-2nu}$ which is slow growing for most polynomials because $2nu$ will usually be tiny making $p(n) \approx 2n$ for relatively small n.

It is good to know when the problem you are working with is well or ill-conditioned. Thus let us find out when k_{rel} blows up and is ill conditioned and when well conditioned. First let us expand k_{rel} to

$$k_{rel} = \frac{|\alpha_0| + |\alpha_1||x| + |\alpha_2||x|^2 + \cdots + |\alpha_{n-1}||x|^{n-1} + |\alpha_n||x|^n}{|\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{n-1}x^{n-1} + \alpha_n x^n|}.$$

From this expansion we can determine when the problem is perfectly conditioned and when it will become ill-conditioned. When $x > 0$ it follows that for $k_{rel} = 1$ all the coefficients or alphas have to have the same signs, $sign(\alpha_i) = sign(\alpha_j)$ for all $i, j \leq n$. When $x < 0$ because odd powers of x will be negative we must have the even alphas be opposite signs of the odd alphas to ensure perfect conditioning. This will ensure that no cancellation will occur in the denominator. Thus evaluating polynomials of this form at the correct x will be perfectly conditioned. Ill-conditioned will occur when k_{rel} is large and that will happen when the denominator is small relative to the numerator. The denominator can get small when evaluating near roots of the polynomial and possibly when cancellation occurs in the denominator.

Problem 2 is mainly coding the results from what we discussed above and seeing the difference between these forward bounds when using single precision and double precision. Now the third problem introduces us to a bound for the running error of Horner's rule. The running error has error accumulating at every iteration of Horner's rule. Now we can see that the computed value on step i of of Horner's rule satisfies $(1+\epsilon_i)\hat{c}_i = x\hat{c}_{i+1}(1+\delta_i)+\alpha_i$ where $|\delta_i|, |\epsilon_i| \le u$ where u is the unit round off. First let $\hat{c}_i = c_i + e_i$ with $e_n = 0$ and c_i is the exact. We can see that

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

Then we can bound e_i with the fact that $|\delta_i|, |\epsilon_i| \le u$. We now have

$$\begin{aligned}
e_i &= xe_{i+1} + x\hat{c}_{i+1}\delta_i - \epsilon_i\hat{c}_i \\
|e_i| &\le |x||e_{i+1}| + |x||\hat{c}_{i+1}||\delta_i| + |\epsilon_i||\hat{c}_i| \\
|e_i| &\le |x||e_{i+1}| + u(|x||\hat{c}_{i+1}| + |\hat{c}_i|).
\end{aligned}$$

This can also be expressed as $|e_i| \le u\beta_i$ where $\beta_n = 0$ and $\beta_i = |x|\beta_{i+1} + |x||\hat{c}_{i+1}| + |\hat{c}_i|$. Thus we have that $|p_n(x) - \hat{c}_0| \le u\beta_0$, which is our running error bound. The running error will be tighter than the forward bound in some cases while the forward bound may be tighter in other cases. It is good to have both bounds though because whichever is the better you know that Horner's evaluation will be in the tighter bound.

4 Description of the Algorithms and Implementation

The goal of this study was to analyze Horner's rule using different floating point systems, single and double precision. Thus all of our functions will have a duplicate, one that implements floating point arithmetic in single precision and in double precision. Now most of these functions are relatively simple and there are not many ways you could implement them differently. Thus efficiency of the code is not a huge problem in this analysis because there is really only one way to implement this algorithm.

The first function we need is to implement is the our exact value of our polynomial. We are using the product form of our polynomial and using the evaluation as the exact evaluation of the polynomial at that point. This was just the use of a for loop with the the amount of iterations equal to the degree of the product. Thus the number of computations is equal to n. Then we have Horner's Rule as another function which has $2n$ computations, implemented with a simple for loop as well. Next is a function that calculates the forward error which has $4n + 7$ computations. Which is implemented the same way as Horner's rule except both terms are within the absolute value operation adding the extra $2n$ to the number of computations. Then lastly the running error function computes the running error at a speed of $10n$ computations. This is because to compute the running error you need to also compute Horner's rule inside the function and has many more operations inside the for loop. All in all the algorithms implemented in this program were not difficult to implement and thus do not have any major different ways to implement them.

5 Description of the Experimental Design and Results

The experiments used in this study were designed to show that the bounds derived above do actually function as expected. By that we mean Honer's rule should be well within the forward error bounds and running error bounds. Problem 2 and 3 is implementing the code on a polynomial and seeing if empirical evidence follows the theory above. The polynomial from problem two is $p_9(x) = (x - 2)^2$. Using the code we evaluated 500 uniform points in the interval $[1.91, 2.1]$ and then plotted the exact, Horner's, forward bound, and running bound for those 500 points. Figure 1 is the plot and we can see that it does agree with the theory above.

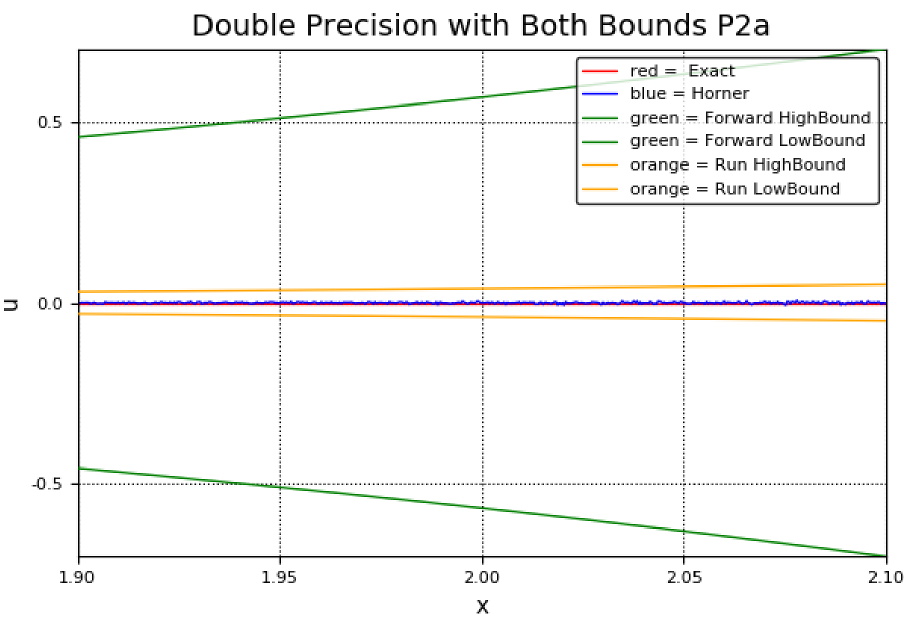


Figure 1: This is plot from 1.91 *to* 2.1 *for the the exact, Horner's, and both error bounds. Where everything is computed using single precision except exact which uses double.*

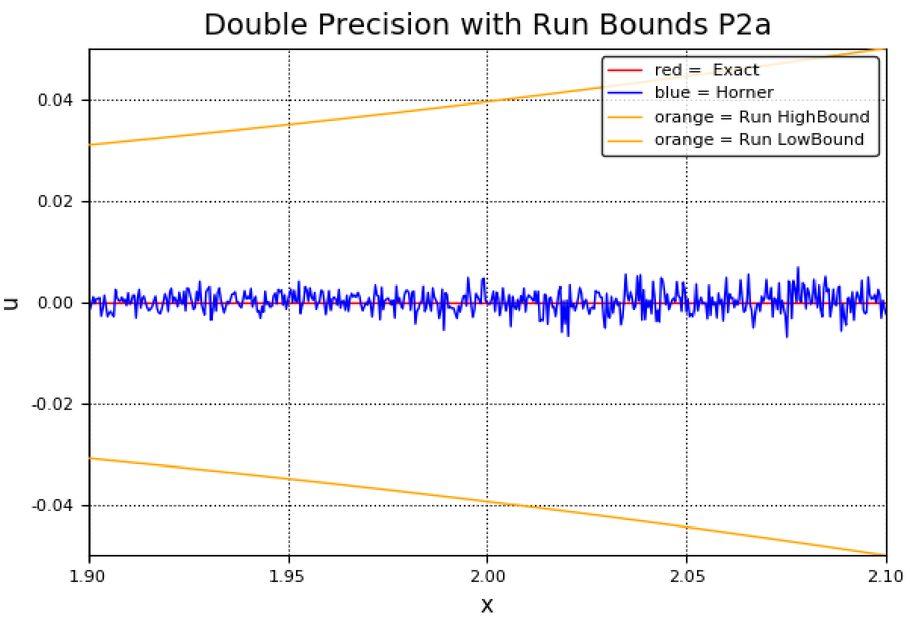
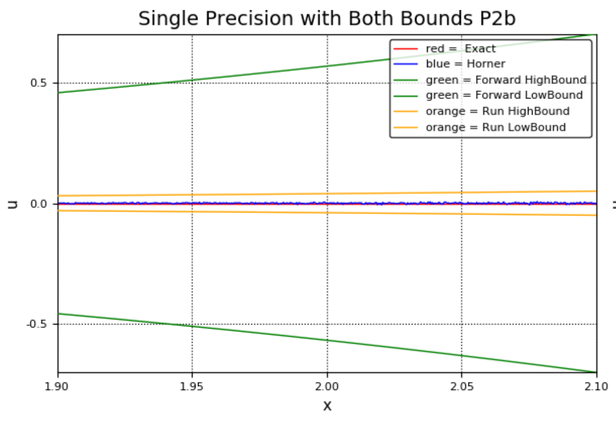


Figure 2: This is plot from 1.91 *to* 2.1 *for the the exact, Horner's, and run error bounds. Where everything is computed using single precision except exact which uses double.*

From the plot we see that the running bound is the better bound for Horner's rule in this case. It makes sense that polynomial would be about zero as well because 0.1^9 is very close to zero and it only gets closer to zero as you approach $x = 2$. Figure 2 zooms with respect to the y-axis to get a better look at the running bounds. From figure 2 we can see that Horner's rule evaluation does not come close to touching the running error bound. Now this was with everything evaluated with single precision arithmetic except the exact which was evaluated using double precision. Next we will look at what would happen if the exact was evaluated using single precision.



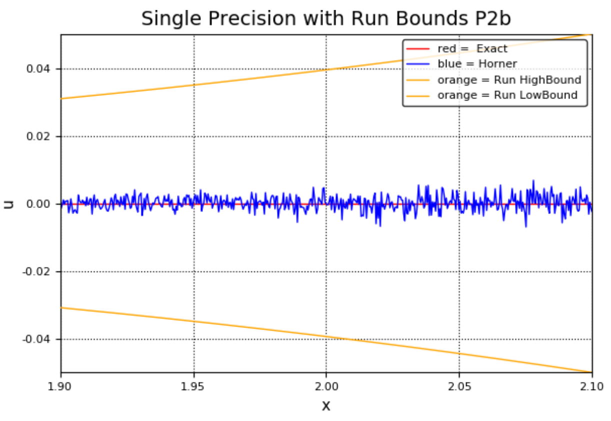
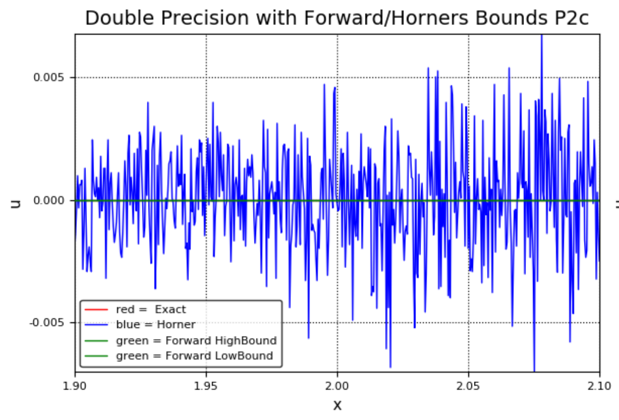


Figure 3: This is plot from 1.91 *to* 2.1 *for the the exact, Horner's, and both error bounds. Where everything is computed using single precision.*
Figure 4: This is plot from 1.91 *to* 2.1 *for the the exact, Horner's, and run error bounds. Where everything is computed using single precision.*

We can see from the two figures above that the exact and graphs in general look nearly identical. In fact if you look at the data we can see that the exact only changes around the 10^{-13} place. Thus changing the exact evaluation to single precision still yields a very good result for the exact evaluation.

Next we should discuss if solving the forward error in single precision is a good decision. The answer to this question is no. The reason is because both errors depend greatly on the unit round-off. Then because of this the error fails as a bound. Horner's Rule using single precision goes way outside the it by a few orders of magnitude. If you look at the two figures below you can see that result.



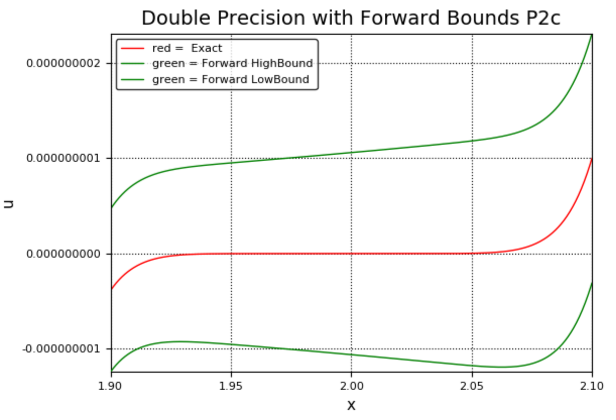


Figure 5: This is plot from 1.91 *to* 2.1 *for the the Horner's in single precision and then exact and forward bounds in double precision.*
Figure 6: This is plot from 1.91 *to* 2.1 *for exact and forward bounds in double precision.*

Next in our discussion is to see what happens to our errors when we have polynomials of different degrees and with different constant c in $(x - c)^n$. The three polynomials we will be working with are $(x+4)^8$, $(x-15)^4$, and $(x-50)^5$. I chose these because I wanted to see what would happen with a large c and small n and also with a positive small c and large exponant. Then The three plots below include the exact evaluation, Horner's evaluation, forwards bounds, and running error bounds.

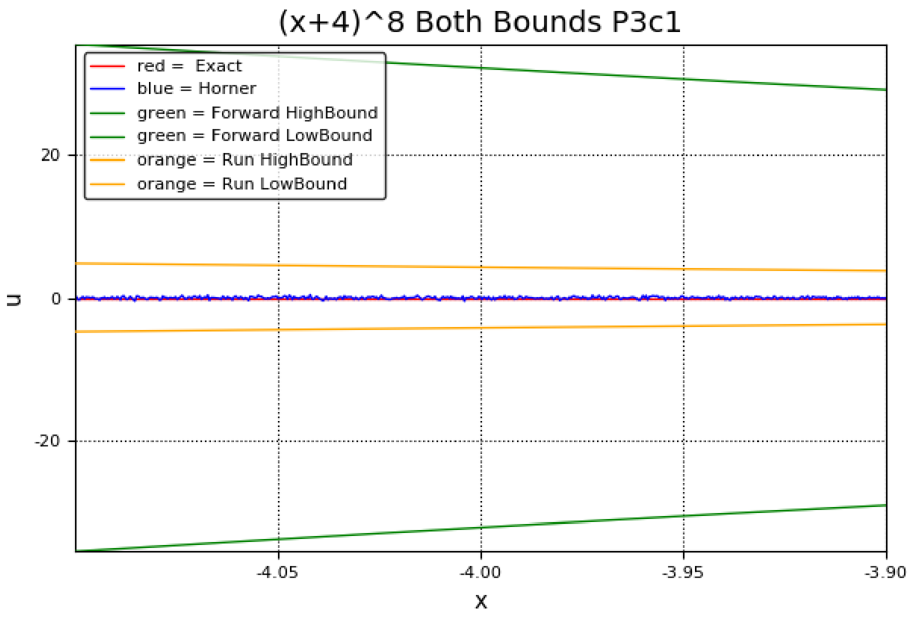


Figure 7: This is the plot for $(x + 4)^8$

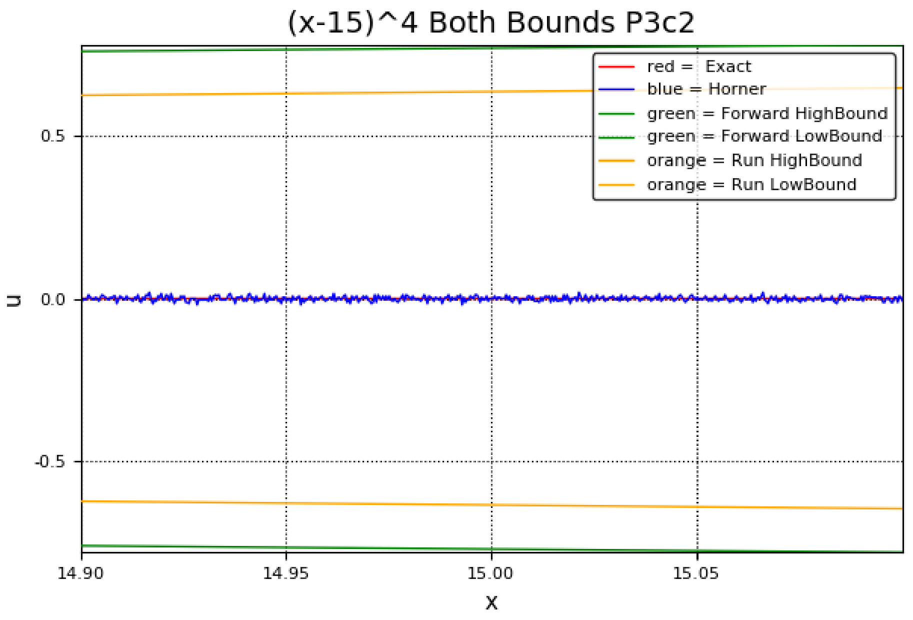


Figure 8: This is the plot for $(x - 15)^4$

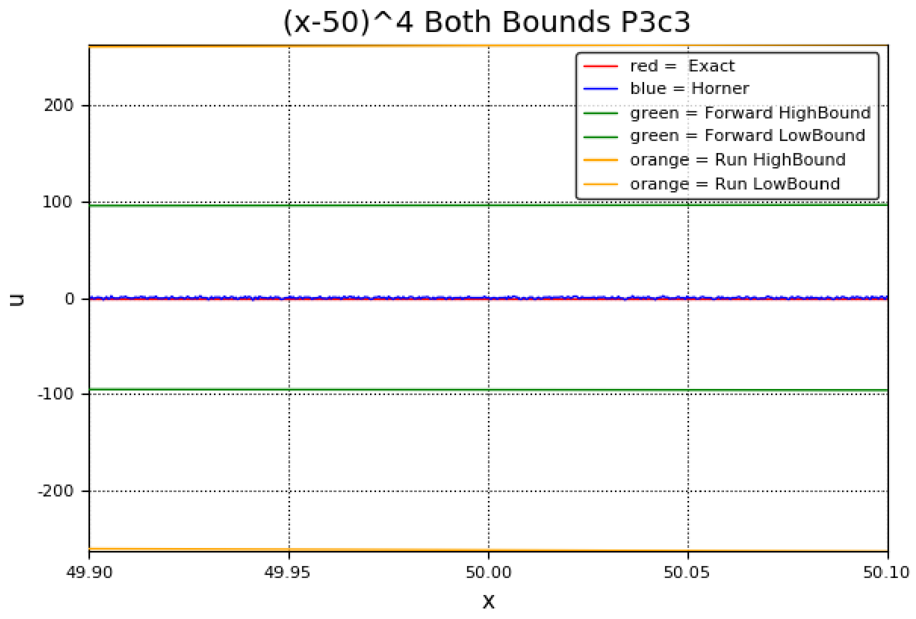


Figure 9: This is the plot for $(x-50)^4$

From these plots we can learn a great deal about the forward error and running error bounds. We know that both bounds do depend on both c and n. But this shows us empirically that the running error bound is much more sensitive to c than the forward error bound. You can see this from going from $c = -15$ to $c = -50$. In fact the run bound is worse than the forward in $(x-50)^4$. They both are affected, but the running error is more greatly increased. What we see next is that the degree impacts the forward error much more than the running error. We see this in $(x+4)^8$ and in the polynomial from problem 2. Finally though from all these graphs even if the bounds are bad at points it seems that Horner's rule still does well for evaluating polynomials in single precision.

6 Conclusions

The point of this report was to study the algorithm to evaluate polynomials at a point called Horner's rule. The rule is much more efficient than evaluating the polynomial in the general form, evaluating it in $2n$ computations compared to $\frac{n(n+1)}{2}$ respectively, where n is the degree of the polynomial. We analyzed the algorithm in a floating point system, trying to get an error bound for different polynomials evaluated at different points.

First we found that the problem can be ill conditioned and perfectly conditioned depending on the polynomial coefficients and where you are trying to evaluate the polynomial at. If you are evaluating where $x > 0$ then to be perfectly conditioned it is necessary that the coefficients all have the same parity. Then if you are evaluating where $x < 0$ the even and odd degree coefficients have to have opposite parity to be perfectly conditioned. Be wary of cancellation that can occur in the denominator of the condition number. It can make the problem ill conditioned if you are not careful. Then we found that the algorithm is also backwards stable where the relative error is bounded below $\gamma_{2n}\frac{\widetilde{p}_n(|x|)}{|p_n(x)|}$. The analysis produced two different error bounds which work better than the other in different scenarios.

The two bounds the study produced is as follows

$$|e_i| \le \gamma_{2n}\widetilde{p}_n(|x|) \tag{1}$$
$$|e_i| \le u\beta_i \tag{2}$$

where $\beta_i = |x|\beta_{i+1} + |x||\hat{c}_{i+1}| + |\hat{c}_i|$ and $\beta_n = 0$. Bound number 1 is the forward error bound and Bound number 2 is the running error bound. The experimental data showed that the bounds are good and bad for different polynomials of the form $(x + c)^n$. We saw that the forward error can be much better than the run error for large c and we saw that the run error is tighter for large n. Thus forward error is much more sensitive to the degree of the polynomial, while the run error is sensitive to the constant.

7 Program Files

There are two sources of code in the source file. One is a c++ file and the other is a Julia file. The c++ file actually evaluates Horner's rule, the errors, and the exact and puts all into a text file called data.txt or what you choose. If you want a different title for txt file then you must go into code on line 124 and change it. As of right now the program asks the user to input if they want to be using single or double precision and answer with S or D respectively. Then it asks for the degree of the polynomial. Then asks for the coefficients starting with the constant term and going up in degrees. After the code will ask you what the upper bound for the interval you would like to evaluate in is. Then the lower bound of the interval. Then it asks how many points, including the upper and lower bounds, in the interval you would like to evaluate all 4 items at. Now the code prints 1 number per line, but it is grouped by 4. The first group is evaluated at the lower bound and first is the exact evaluation, then Horner's evaluation, then Forward Error, and lastly Running Error. The 5th line is the second group evaluated at lower bound plus dx and so on until the upper bound.

If you do not want to keep typing in everything you can go into the code and hardcode all your answers in for the questions asked. First comment out all the lines for getting data from the user. Lines 131 to 136 for single/double precision and degree of polynomial, lines 144 to 157 for single precision polynomial coefficients, bounds, and number of points, and lines 192 to 205 for double precision polynomial coefficients, bounds and number of points. The hardcode lines are located right under the lines you just commented, so uncomment them and put in the information you want.

The Julia program plots all six lines (Exact Value, Horner's Rule Value, Forward Error Bounds, Running Error Bounds). You will need to hardcode the name of the txt file, bounds, and number of points which is located lines 25 to 28. Then on line 69 you can change the name of the title of the graph the code will produce.