Assignment 1

Zeyuan Wang

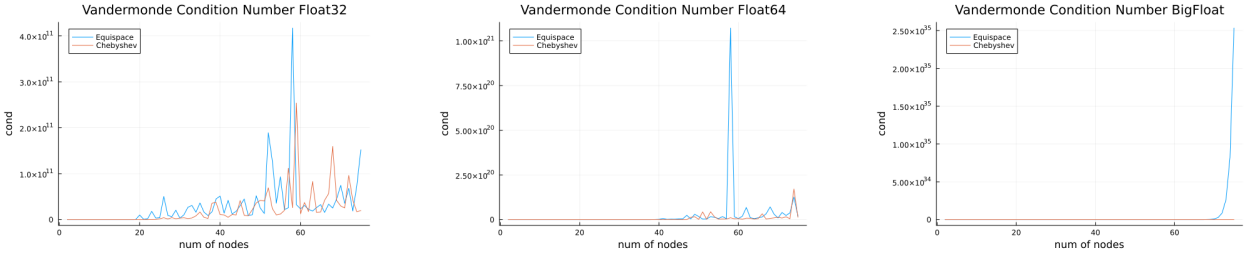
January 29, 2024

1 Problems

1.1

I use Julia language and my platform is MacBook Pro M1.

(a) To study the effect of float precision on result, I test float32, float64 and bigfloat, an arbitrary precision type supported by Julia.

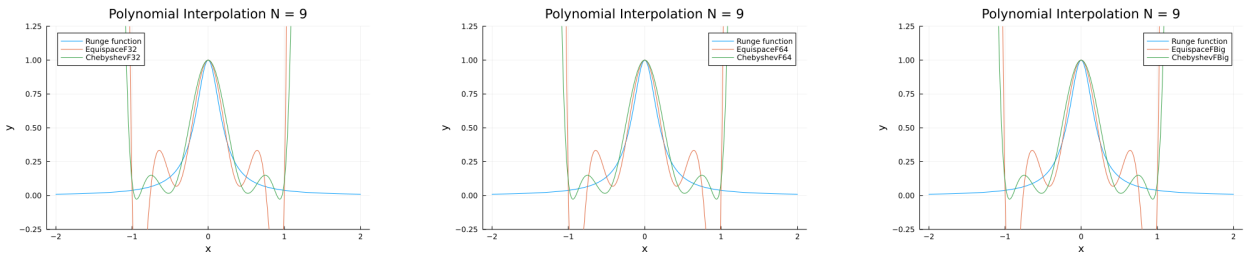


(a) 32-bit float (b) 64-bit float (c) big float

Figure 1: Condition number of Vandermonde

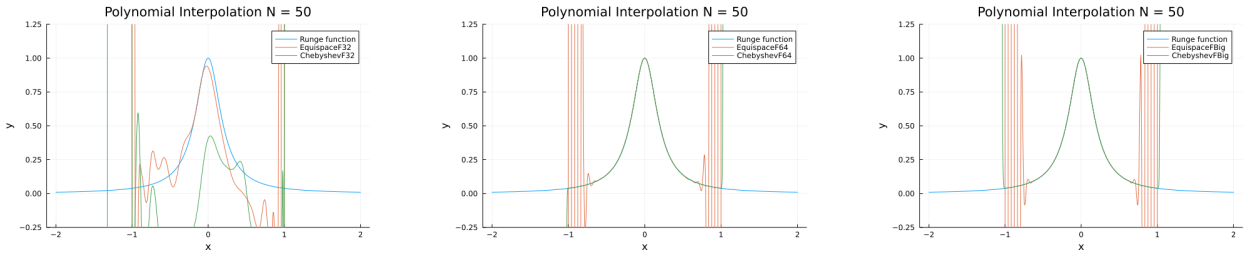
We can see from the graph that when use 32bit float, neither of types behavior really well. When use 64bit float, Chebyshev extrema perform much better, while equispace obviously suffer from ill-condition problem. When use big float, Chebyshev is nearly perfect. Of course, it cost much more time to compute.

(b) Plot of the polynomial interpolants for N = 9 and N = 50 for both types of points.



(a) 32-bit float (b) 64-bit float (c) big float

Figure 2: Polynomial N = 9



(a) 32-bit float (b) 64-bit float (c) big float

Figure 3: Polynomial N = 50

When using only 9 nodes for interpolation, both type behavior similar well, no matter what float precision is used. When it comes to 50 nodes, 32bit float just produce garbage. The result of 64bit float and big float is similar. Both type do well interpolation in the middle of the range, but equispace sample points suffer from severe Runge phenomenon near the end point, while Chebyshev extremas perform much better.

(c) Plot of total time vs the accuracy just for Chebyshev extrema

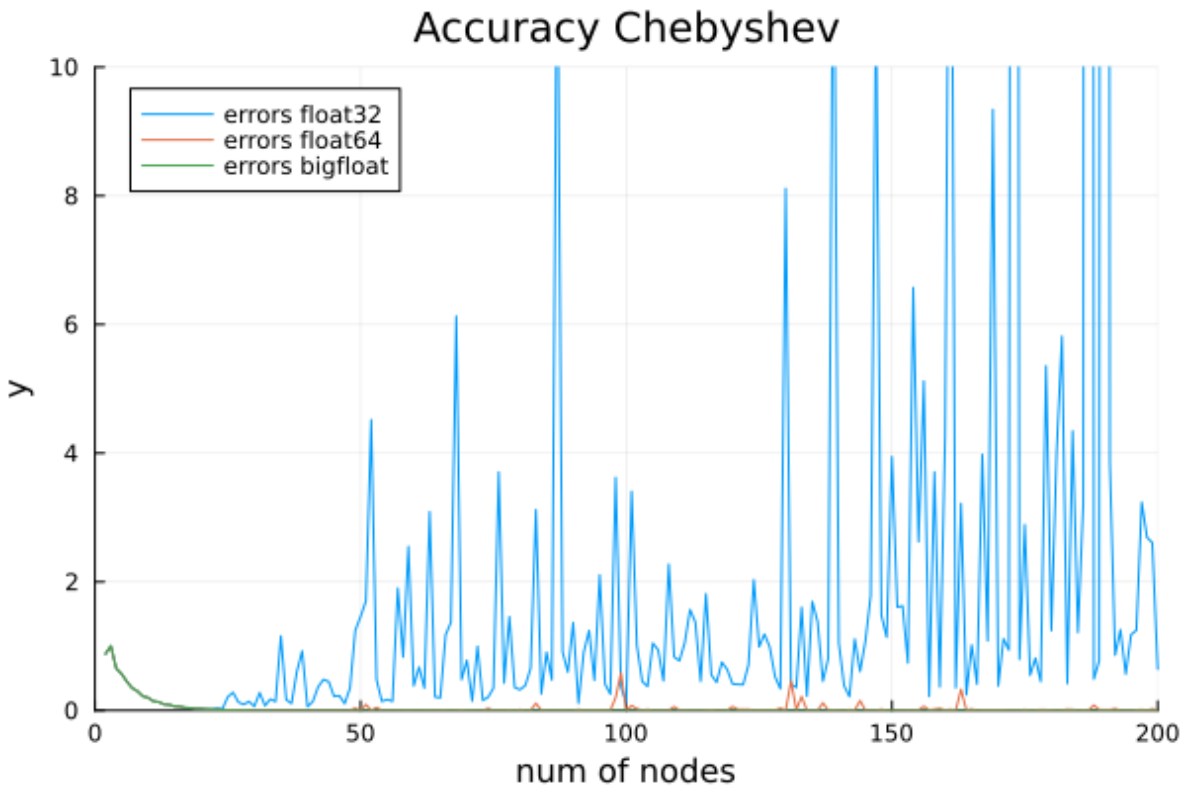


Figure 4: Accuracy measure(L2) for Chebyshev

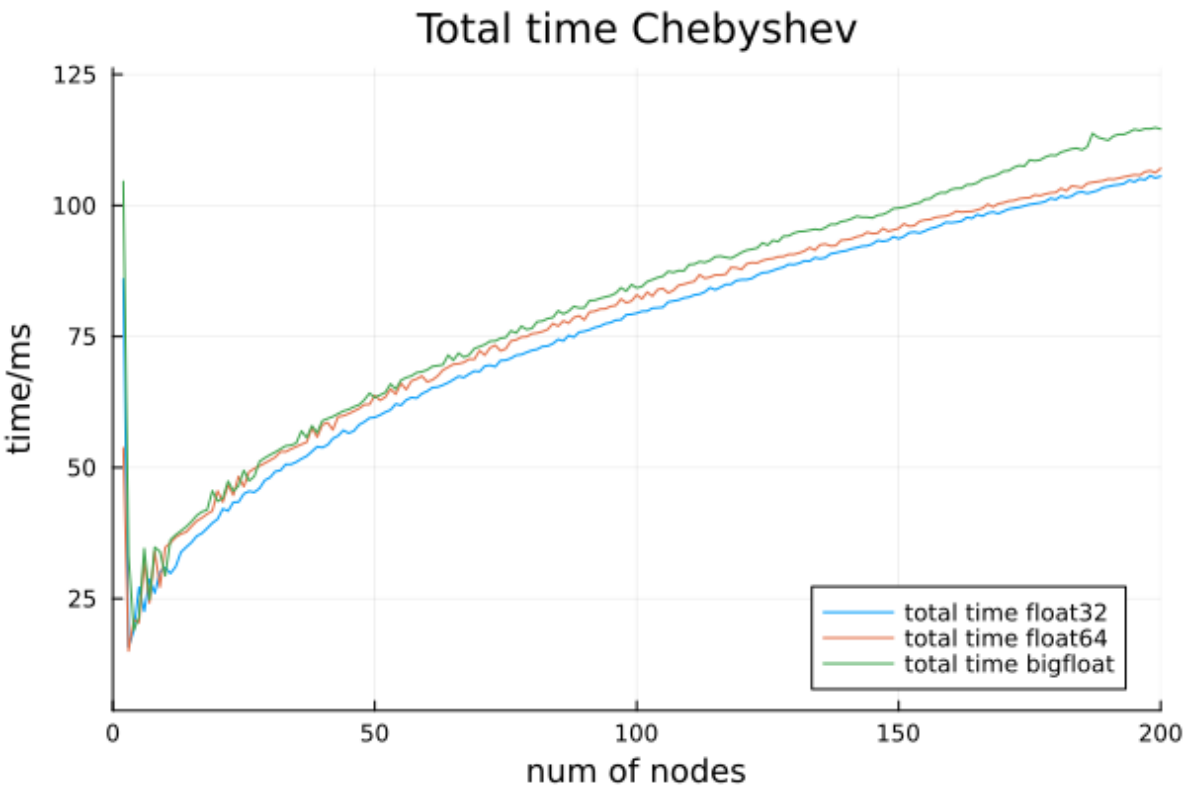


Figure 5: Total time measure(cube root) for Chebyshev

From the graph we can see that although 32-bit float gives us the fastest computation time, the accuracy is total a catastrophe. 64-bit provides overall acceptable accuracy, and the total time is just slightly higher. The arbitrary precision float type provides a perfect accurate result, with most expensive computation price.

However, there are two points I fail to explain. First is the strange "tail" at the beginning. I don't understand what Julia does during solving a small size of linear system. (I can make sure it's not a bug or something to do with plotting). The second thing is the curve look like sub-linear. I take the cbrt of measured time, expecting to see something linear, but it turn out to be sub-linear, which means the original result

should be lower than $O(N^3)$. It is counter-intuitive since I can make sure Julia default linear solver use LU decomposition, which should have $O(N^3)$ complexity. So I don't know it's there other magic optimizations or there is a bug in my code, or I pick the wrong way to do the benchmark.

1.2

We use the Lagrange polynomial of function $f(x)$ around x_1 by samples $f(x_0)$,$f(x_1)$ and $f(x_2)$

$$L(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + f(x_2)l_2(x) \tag{1}$$

Since we approximate the second derivative of $f(x)$ at x_0 by instead differentiating $L(x)$

$$\begin{aligned} \frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\Big|_{x=x_1} &\approx \frac{\mathrm{d}^2 L}{\mathrm{d}x^2}\Big|_{x=x_1} \\ &= \sum_{i=0}^{2} f(x_i) \frac{\mathrm{d}^2 l_i(x)}{\mathrm{d}x^2}\Big|_{x=x_1} \end{aligned} \tag{2}$$

And we have

$$l_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)},\ l_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)},\ l_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \tag{3}$$

then

$$\frac{\mathrm{d}^2 l_0(x)}{\mathrm{d}x^2} = \frac{2}{(x_0-x_1)(x_0-x_2)} \tag{4}$$

Symmetrically, we get

$$\frac{\mathrm{d}^2 l_1(x)}{\mathrm{d}x^2} = \frac{2}{(x_1-x_0)(x_1-x_2)},\ \frac{\mathrm{d}^2 l_2(x)}{\mathrm{d}x^2} = \frac{2}{(x_2-x_0)(x_2-x_1)} \tag{5}$$

Substitute back to equation(2), and replace $x_i - x_j$ with h

$$\begin{aligned}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\Big|_{x=x_1} &\approx \sum_{i=0}^{2} f(x_i)\frac{\mathrm{d}^2 l_i(x)}{\mathrm{d}x^2}\Big|_{x=x_1} \\ &= \frac{2f(x_0)}{h^2} - \frac{2f(x_1)}{h^2} + \frac{2f(x_2)}{h^2} \\ &= \frac{2f(x_0) - 2f(x_1) + 2f(x_2)}{h^2}\end{aligned} \tag{6}$$

The remainder(error estimation) of polynomial interpolation can be expressed as

$$R(x) = f(x) - L(x) = \frac{f^{(N+1)}(\xi)}{(N+1)!}\prod_{i=0}^{k}(x - x_i),\ \xi \in (x_0, x_k) \tag{7}$$

If we take the second derivative

$$\begin{aligned}\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} - \frac{\mathrm{d}^2 L(x)}{\mathrm{d}x^2} &= \frac{\mathrm{d}^2}{\mathrm{d}x^2}[\frac{f^{(N+1)}(\xi)}{(N+1)!}\prod_{i=0}^{k}(x - x_i)] \\ &= \frac{\mathrm{d}^2}{\mathrm{d}x^2}[\frac{f^{(N+1)}(\xi)}{(N+1)!}]\frac{\mathrm{d}}{\mathrm{d}x}[\prod_{i=0}^{k}(x - x_i)] \\ &+ 2\frac{\mathrm{d}}{\mathrm{d}x}[\frac{f^{(N+1)}(\xi)}{(N+1)!}]\frac{\mathrm{d}}{\mathrm{d}x}[\prod_{i=0}^{k}(x - x_i)] \\ &+ \frac{\mathrm{d}}{\mathrm{d}x}[\frac{f^{(N+1)}(\xi)}{(N+1)!}]\frac{\mathrm{d}^2}{\mathrm{d}x^2}[\prod_{i=0}^{k}(x - x_i)]\end{aligned} \tag{8}$$

Since we only get 3 nodes in this problem, it seems not too bad to directly compute the derivative of $\prod_{i=0}^{k}(x - x_i)$. I directly take the second derivative and get

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}[\prod_{i=0}^{k}(x - x_i)] = 6x - 2x_0 - 2(x_0 + x_1) \tag{9}$$

Evaluate at x_1, it happen to be zero.($x_1 = \frac{(x_0+x_2)}{2}$) For the first derivative, we actually have a clever way. Denote $l(x) = \prod_{i=0}^{k}(x - x_i)$, then we have

$$\begin{aligned} \ln l(x) &= \ln(\prod_{i=0}^{k}(x - x_i)) = \sum_{i=0}^{k} \ln(x - x_i) \\ \frac{l^{'}(x)}{l(x)} &= \sum_{i=0}^{k} \frac{1}{(x - x_i)} \\ l^{'}(x) &= \prod_{i=0}^{k}(x - x_i) \sum_{i=0}^{k} \frac{1}{(x - x_i)} = \sum_{i=0}^{k} \prod_{\substack{j=0 \\ j\neq i}}^{k}(x - x_j) \end{aligned} \tag{10}$$

In our case, evaluate it at $x = x_1$ produce $(x_1 - x_0)(x1 - x_2) = h^2$. Thus

$$\begin{aligned} \frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} - \frac{\mathrm{d}^2 L(x)}{\mathrm{d}x^2} = \{\frac{\mathrm{d}^2}{\mathrm{d}x^2}[\frac{f^{(N+1)}(\xi)}{(N+1)!}] \\ +2\frac{\mathrm{d}}{\mathrm{d}x}[\frac{f^{(N+1)}(\xi)}{(N+1)!}]\}h^2 \end{aligned} \tag{11}$$

The derivative of $\frac{f^{(N+1)}(\xi)}{(N+1)!}$ is not something we can really deal with. We can conclude that the error is $O(h^2)$.

1.3

Very similar to problem 2, while this time we integrate $L(x)$ over $[a, b]$

$$\begin{aligned} \int_a^b f(x)\mathrm{d}x &\approx \int_a^b L(x)\mathrm{d}x \\ &= f(a)\int_a^b l_0(x)\mathrm{d}x + f(\frac{a+b}{2})\int_a^b l_1(x)\mathrm{d}x + f(b)\int_a^b l_2(x)\mathrm{d}x \end{aligned} \tag{12}$$

Before we get our hand dirty, let's think about what we are actually doing. It's easy to find that l_0 and l_2 is two quadratic which has roots $a, \frac{a+b}{2}$ and $\frac{a+b}{2}, b$, respectively. Given that $(x_0 - x_1)(x_0 - x_2) = (x_2 - x1)(x_2 - x0) = 2h^2$, we can conclude that l_0 and l_1 are symmetric

respected to $x = x_1$. In other word, their integral over $[a, b]$ should be exactly the same, let's denote it as c. The real trick happens when try to add up l_0 and l_2

$$\begin{aligned}
c &= \frac{1}{2}\frac{1}{2h^2}\int_a^b (x-x_1)(x-x_2) + (x-x_0)(x-x_1)\mathrm{d}x \\
&= \frac{1}{4h^2}\int_a^b 2x^2 - (2x_1 + x_0 + x_2)x + x_1(x_2 + x_0)\mathrm{d}x \\
&= \frac{1}{2h^2}\int_a^b x^2 - 2x_1 x + x_1^2\mathrm{d}x \\
&= \frac{1}{2h^2}\int_a^b (x-x_1)^2\mathrm{d}x \\
&= \frac{1}{2h^2}\frac{1}{3}(x-x_1)^3\Big|_a^b \\
&= \frac{1}{2h^2}\frac{1}{3}[(b-x_1)^3 - (a-x_1)^3] \\
&= \frac{1}{2h^2}\frac{1}{3}[(b-a)[(b-x_1)^2 + (a-x_1)^2 + (b-x_1)(a-x_1)]] \\
&= \frac{1}{2h^2}\frac{1}{3}(2h\frac{1}{2}h^2) \\
&= \frac{1}{3}h
\end{aligned} \tag{13}$$

$\int_a^b l_1(x)\mathrm{d}x$ is relatively easier to solve because l_1 itself is symmetric respect to $x = x_1$

$$\begin{aligned}
\int_a^b l_1(x)\mathrm{d}x &= \frac{1}{-h^2}\int_a^b (x-x_0)(x-x_2)\mathrm{d}x \\
&= \frac{1}{-h^2}\int_0^h (x-h)(x+h)\mathrm{d}x \\
&= \frac{1}{-h^2}\int_0^h x^2 - h^2\mathrm{d}x \\
&= \frac{1}{-h^2}[\frac{1}{3}x^3 - h^2 x]\Big|_0^h \\
&= \frac{4}{3}h
\end{aligned} \tag{14}$$

So $\int_a^b f(x)\mathrm{d}x \approx \frac{1}{3}f(a) + \frac{4}{3}f(\frac{a+b}{2}) + \frac{1}{3}f(b)$. This quadrature rule is called "Simpson's 1/3 rule"(Because of the $\frac{1}{3}$ factor). To Derive the error we need to integrate over the remainder

$$\begin{aligned}\int_a^b f(x) - L(x)\mathrm{d}x &= \int_a^b \frac{f^{(N+1)}(\xi)}{(N+1)!}\prod_{i=0}^{k}(x-x_i)\mathrm{d}x,\ \xi \in (x_0, x_k) \\ &= [\frac{f^{(N+1)}(\xi)}{(N+1)!}\int_a^b \prod_{i=0}^{k}(x-x_i)\mathrm{d}x]\Big|_a^b \\ &- \int_a^b \int_a^b \prod_{i=0}^{k}(x-x_i)\mathrm{d}x\mathrm{d}\frac{f^{(N+1)}(\xi)}{(N+1)!} \text{ (integral by parts)}\end{aligned} \tag{15}$$

It's really really hard and verbose to get the close form solution of the integral. Instead, if we only care about the high bound of error, we can do it by observe the highest order among all the terms.$\frac{f^{(N+1)}(\xi)}{(N+1)!}$, in theory ξ is a function of x. It's impossible to know what exactly is ξ, but hopefully we can assume it's just a one order function or x. Since $k = 3$, we can expect the order of $\int_a^b \prod_{i=0}^{k}(x-x_i)\mathrm{d}x$ is effectively 4. Now take a look at the last horrible term. We know that $\frac{f^{(N+1)}(\xi)}{(N+1)!}$ is not a constant respect to x, so we cannot just get rid off the term. And we assume that it's one order, so the order of the last term can be determine by the$\int_a^b \int_a^b \prod_{i=0}^{k}(x-x_i)\mathrm{d}x\mathrm{d}x$. This double integral should give as some thing in 5 order, since$\prod_{i=0}^{k}(x-x_i)$ is 3 order.So we can conclude that, although not very rigorously, $\int_a^b f(x) - L(x)\mathrm{d}x = O(h^5)$.