Project on Polynomial Interpolation

Robson Edwards

November 23, 2018

1

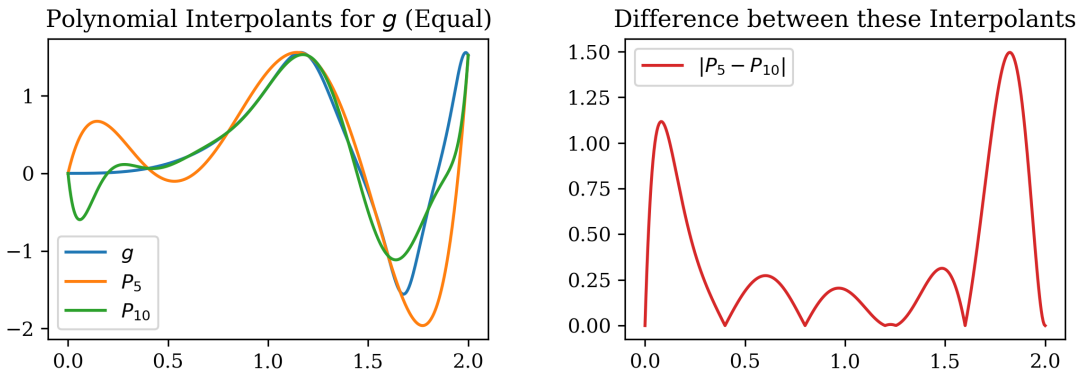
We consider the function $g : [0, 2] \to \mathbb{R}$ defined by $g(x) = \tan(\sin(x^3))$.

1.1

We compute the interpolating polynomials of degree $n = 5$ and $n = 10$ for equally spaced interpolation points. We use the method laid out in the module notes §3.4 (see also attached Python script). We find the polynomial interpolants P_5 and P_{10} to be, to two decimal places,

$$\begin{aligned} P_5 &= 13.82x^5 - 62.10x^4 + 94.02x^3 - 55.28x^2 + 10.85x \\ P_{10} &= 92.73x^{10} - 908.43x^9 + 3781.24x^8 - 8734.57x^7 + 12281.56x^6 \\ &\quad -10851.53x^5 + 5993.46x^4 - 1979.81x^3 + 351.53x^2 - 25.08x. \end{aligned}$$

We plot the interpolants P_5 and P_{10}, along with g, on the left-hand graph below. Note that in a "real" situation where we would be using a polynomial approximation to this function, we would likely not have access to the exact function g. It is included here only as a guide for the eye.
On the right-hand plot below, we graph the absolute difference $|P_5 - P_{10}|$. We can use this difference as a (very) rough estimate of the level of error in the interpolants, assuming g is not available. Because P_5 shares its six interpolation points with P_{10}, this is a better estimate than it would be for e.g. two polynomial interpolants of degree 6 and 10. One must note that P_5 and P_{10} appear to be better approximations within roughly [0.4, 1.6], and worse approximations outwith that interval. We will see that using a better method for selecting interpolation points will improve this issue somewhat.

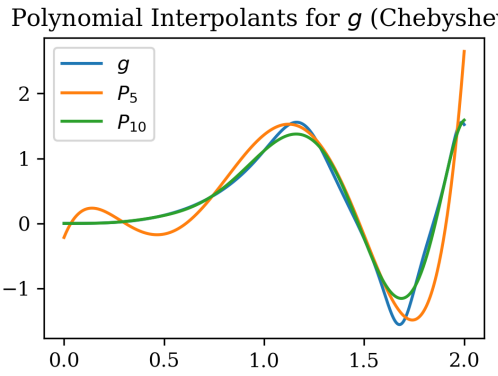


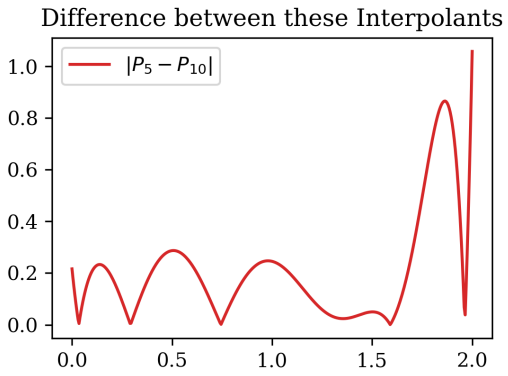
1.2

We compute and graph interpolating polynomials as before, but this time we select interpolation points by the roots of the appropriate Chebyshev polynomial, according to the method in the notes §3.4.1. As noted earlier, we have managed to improve the difference significantly across the entire interval [0, 2], although the effect is more pronounced on [0, 0.4] than it is on [1.6, 2]. I suspect this

is due to the fact that g'' is greater in the latter interval than the former. Regardless, we infer that we have improved the accuracy.

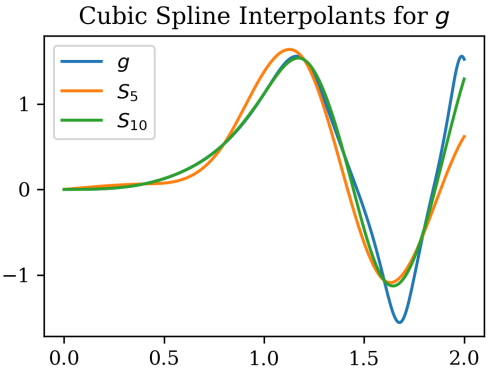
$$\begin{aligned} P_5 &= 12.04x^5 - 52.22x^4 + 75.31x^3 - 41.21x^2 + 7.66x - 0.22 \\ P_{10} &= 2.76x^{10} - 31.07x^9 + 132.97x^8 - 283.79x^7 + 328.83x^6 \\ &\quad -211.39x^5 + 74.17x^4 - 12.46x^3 + 1.15x^2 - 0.04x. \end{aligned}$$

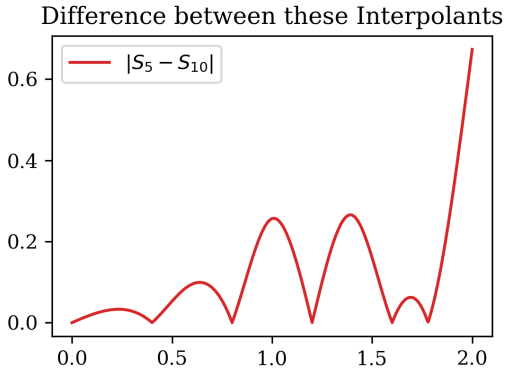




1.3

We compute and graph cubic spline interpolants using the same equally spaced interpolation points as in **§1.1**. We follow the method from notes §3.5.3. The expressions for these interpolants are far too long to write out here. It can be seen from both graphs that we have further improved the accuracy of the approximations.





2

We now consider the function $f : [-1, 2] \to \mathbb{R}$ defined by $f(x) = (1 - x^3)\sin(2\pi x)$.

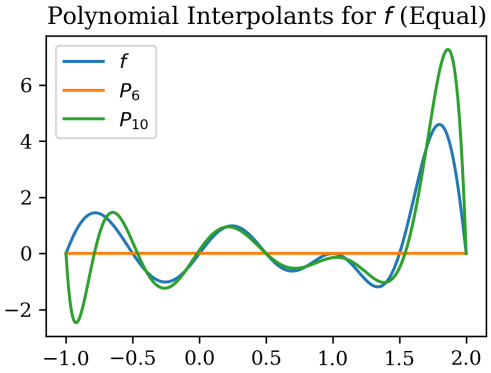
2.1

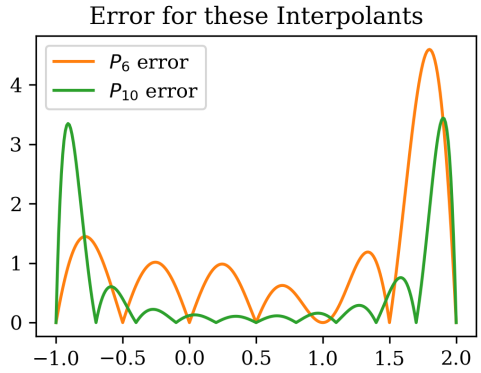
We calculate polynomial interpolants as before, except for f and degree $n = 6$ and $n = 10$. We use equally spaced knots. We plot f, P_6, and P_{10} all on the same graph. We also plot the absolute error between f and P_6, and between f and $P_1 0$. Here, the maximum absolute errors are 4.60 and 3.44 for $n = 6$ and $n = 10$, respectively and to two decimal places. These errors are bounded by (Notes 3.5)

$$|f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \max_{x_0 \leq x \leq x_n} \left|\prod_{k=0}^{n}(x - x_k)\right| \max_{x_0 \leq x \leq x_n} \left|f^{(n+1)}(x)\right|.$$

These bounds can be easily estimated algorithmically (see attached Python script).

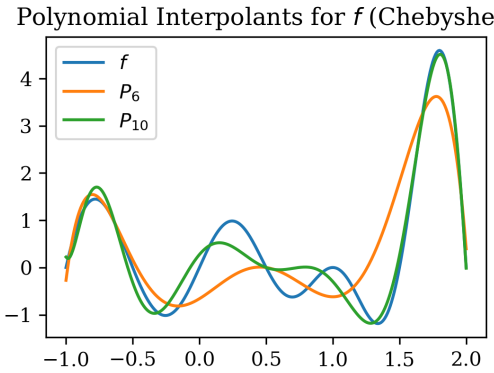
Notably, the degree-6 interpolant is particularly bad here. This is because the equally spaced interpolation points we've selected happen to be roots of the function, so the data points the polynomial selection algorithm receives are all of the form $(x_k, 0)$, and hence it naively selects the polynomial 0. Any lower degree than $n = 6$ would also give a bad interpolant. Even if we had non-zero data points, f oscillates too many times on the interval $[-1, 2]$ for a lower-degree function to give a good approximation.

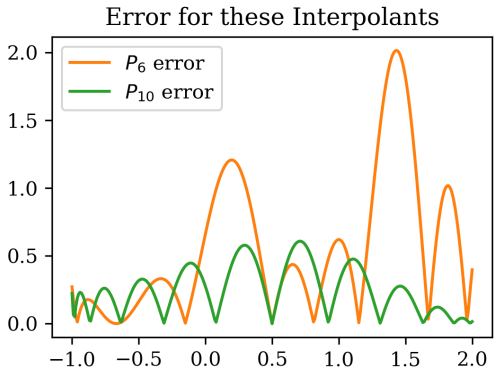




2.2

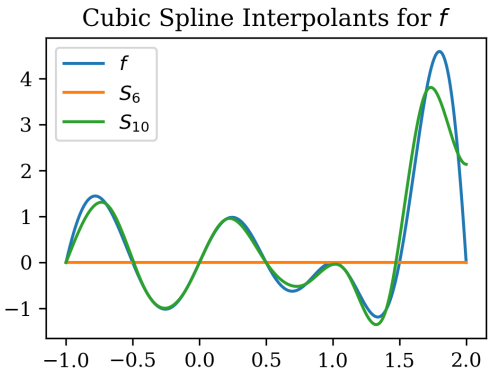
We calculate polynomial interpolants as in **§2.1** but using Chebyshev knots as in **§1.2**. We are rewarded with improved maximum errors of 2.01 and 0.61 for degree $n = 6$ and $n = 10$, respectively. However, it is still clear that P_6 does not have the flexibility required to fit all the oscillations of f.

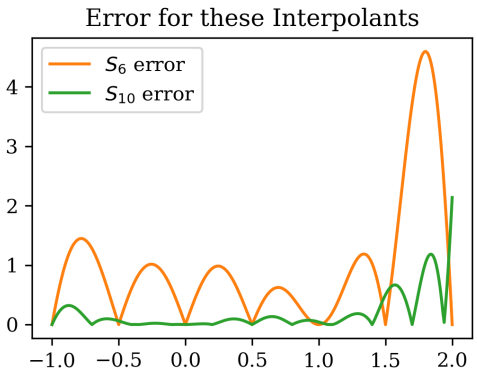




2.3

Finally, we calculate cubic spline interpolants as in **§1.3** with the interpolation points from **§2.1**. Of course, this means our $n = 6$ spline doesn't work at all. The $n = 10$ one works fairly well for almost all of $[-1, 2]$ but gives up towards the end. We have maximum errors of 4.60 and 2.14 for $n = 6$ and $n = 10$, respectively.


Cubic Spline Interpolants for f
f
S_6
S_{10}
4
3
2
1
0
−1
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Error for these Interpolants
S_6 error
S_{10} error
4
3
2
1
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0