Math 128A, Summer 2019

PSET #3 (due Wednesday, 7/17/2019)

Problem 1. Let $\{\lambda_k\}, \{t_j\}$ each be $n+1$ distinct real numbers.

(a) Show that

$$a(t) = \sum_{k=0}^{n} a_k e^{\lambda_k t}$$

can vanish for all real t only if $a_0 = a_1 = \cdots = a_n = 0$.

Solution. In this problem we construct a linear combination of exponentials functions. Because our λ_k are $(n+1)$ *distinct* reals, this surely gives us a $(n+1)$-dimensional basis of exponential functions. To see this explicitly, consider the homogenous equation:

$$a_0 e^{\lambda_0 t} + a_1 e^{\lambda_1 t} + \cdots + a_n e^{\lambda_n t} = 0.$$

Given λ_k all distinct, if we show the only solution to this is $a_0 = a_1 = \cdots = a_n = 0$, then we have linear independence as desired.

To not invoke (Strain's anger for using) proof by contradiction, let us construct a formalization via induction. Consider the set $U \subset \mathbb{N}$ of n for which the above statement is true. Surely, e^x is positive (nonnegative) for all x, so a single term $a_0 e^{\lambda_0 t} = 0$ requires $a_0 = 0$, so $a_0, \ldots, a_n$ for $n = 0$ holds, and hence $0 \in U$. Assume $k \in U$, so that $a_0 = a_1 = \cdots = a_k$ is the only solution to the homogenous equation, for distinct λ_i.

$$0 = \sum_{i=0}^{k} a_i e^{\lambda_i t} \quad \text{(inductive assumption)}$$

Now suppose

$$0 = \underbrace{a_0 e^{\lambda_0 t} + \cdots + a_k e^{\lambda_k t}} + a_{k+1} e^{\lambda_{k+1} t}. \tag{1}$$

From our induction step, we know the underbraced terms only equal zero for $a_0 = \cdots = a_k = 0$. Because $a_{k+1} e^{\lambda_{k+1} t} \neq 0$ if $a_{k+1} \neq 0$ (per our above argument that e^x nonzero for all x), if a nontrivial solution exists, we must have $a_{k+1} \neq 0$. Then we can divide across to obtain:

$$0 = 1 + \sum_{j=0}^{k} \frac{a_j}{a_{k+1}} e^{(\lambda_j - \lambda_{k+1})t},$$

and the only solution to homogenous equation (1) above is the trivial solution, and hence $k \in U \implies (k+1) \in U$, and we conclude $U = \mathbb{N}$ as desired to complete our induction. Formalizing our argument seems to require higher level familiarity with things like the Vandermonde matrix as discussed in lecture, or some further case-work in our induction. For our purposes, we take the general reasoning to be sufficient. □

(b) Show that for the exponential interpolation problem

$$a(t_j) = \sum_{k=0}^{n} a_k e^{\lambda_k t_j} = f_j, \qquad 0 \leq j \leq n,$$

there exists a unique solution $a(t)$ for any data values f_j.

Solution. This result is given by (a) above for free. Because t_j are all distinct, no more than one can be zero, and likewise no more than one λ_k can be zero. Hence by (a) above, the only solution to $a(t_j) = f_j$ above is the trivial solution $a_0 = \cdots = a_n = 0$. Because we have a $(n+1) \times (n+1)$ system of linear equations for each t_j, by (a) above our matrix is invertible and hence we have a unique solution. □

(c) Interpolate the function

$$f(t) = \frac{1}{1+t^6}$$

via $(n+1)$ exponentials with $\lambda_k = \frac{-k}{n}$, $k = \texttt{0:n}$, at $(n+1)$ equidistant points $t_j = \frac{5j}{n}$ for $j = \texttt{0:n}$ on the interval $[0, 5]$, and tabulate the error for $n = 3, 5, 9, 17, 33$.

Solution. Firstly, we acknowledge in lecture that we should not obtain good results here. We may play around with bases other than the monomial basis to look for possibly better interpolations. With the given definitions, this gives a system of linear equations where for each t_j we have:

$$\begin{aligned} a(t_j) &:= a_0 e^{\lambda_0 t_j} + a_1 e^{\lambda_1 t_j} + \cdots + a_n e^{\lambda_n t_j} \\ &= a_0 [e^{t_j}]^{\frac{-0}{n}} + a_1 [e^{t_j}]^{\frac{-1}{n}} + \cdots + a_n [e^{t_j}]^{\frac{-n}{n}} \\ &= a_0 [e^{-t_j/n}]^0 + a_1 [e^{-t_j/n}]^1 + \cdots + a_n [e^{-t_j/n}]^n \end{aligned}$$

Hence for each t_j, letting the substitution $\theta_j := e^{-t_j/n}$ gives us a familiar polynomial in θ:

$$a(\theta_j) := a_0 \theta_j^0 + a_1 \theta_j^1 + \cdots + a_k \theta_j^k + \cdots + a_n \theta_j^n.$$

For $n = 3, 5, 9, 17, 33$, we have the following interpolating polynomials:

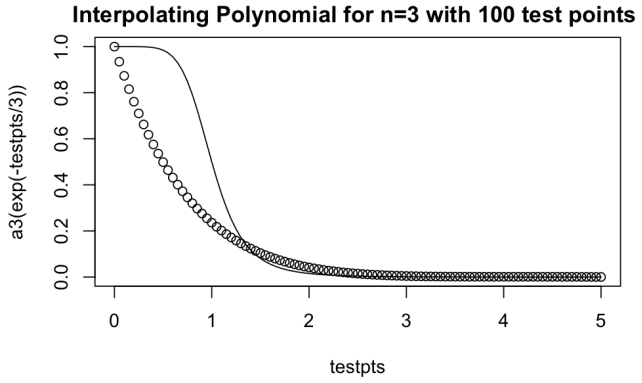


Figure 1: `-0.04237186 + 0.5013957*x - 1.922716*x^ 2 + 2.463692*x^ 3`

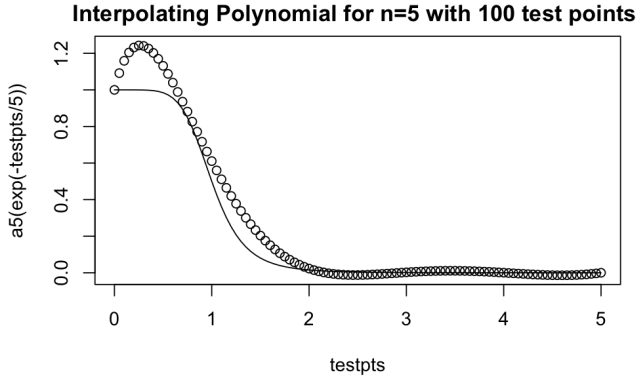


Figure 2: `21.81665 - 201.5867*x + 725.3772*x^ 2 - 1267.443*x^ 3 + 1071.527*x^ 4 - 348.6912*x^ 5`

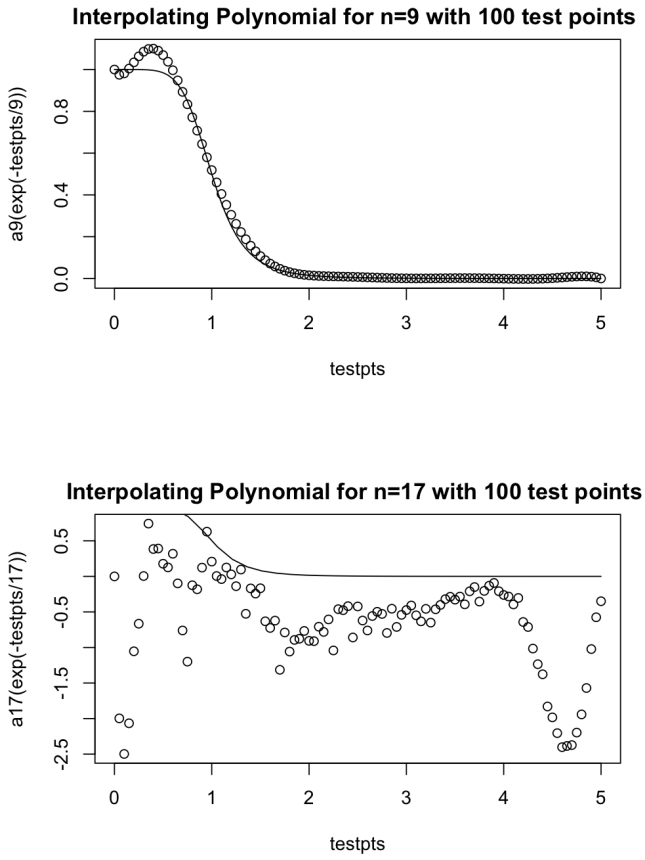
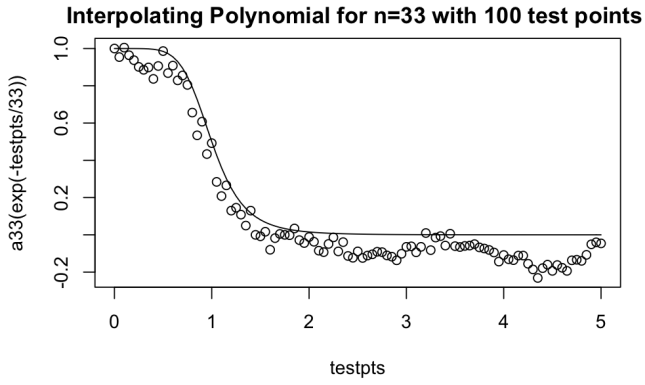
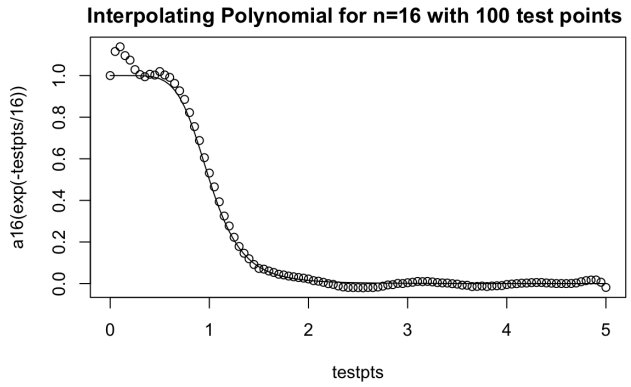
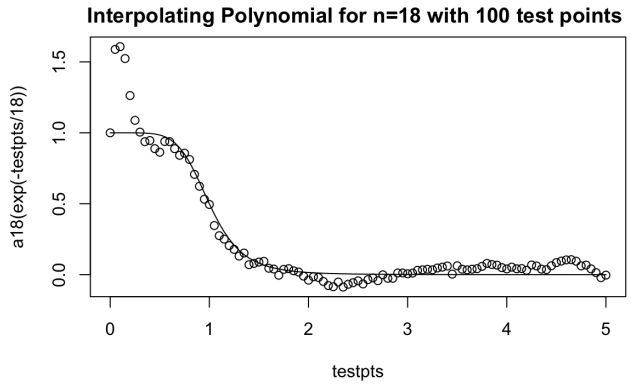


Figure 3: Interestingly poor behavior at n=17, but still relatively acceptable.



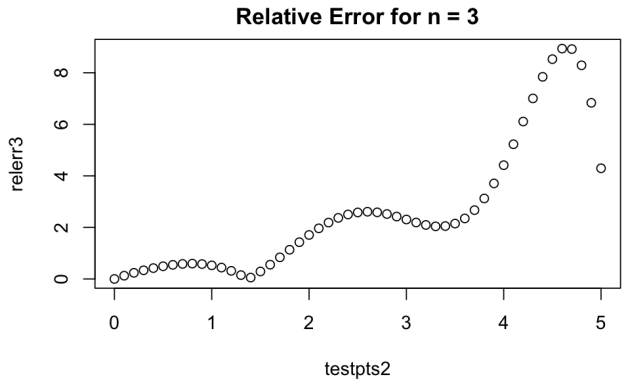
And in attempts to diagnose our issue with $n = 17$, we look at $n = 16$ and $n = 18$:

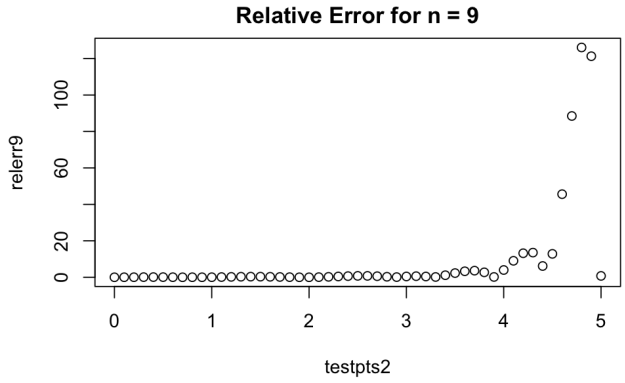


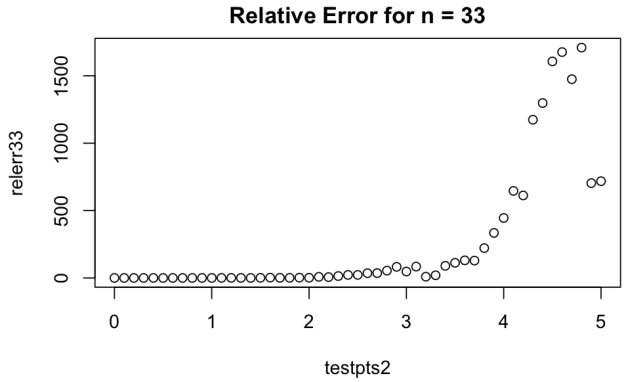


Comparing to $n = 16$ and $n = 18$, there appears to be no reason our resulting interpolation performs so poorly for $n = 17$. There is no bug within my code; however, there must be some intrinsic issue with our matrix inversion for our values and the size $n = 17$, or a bug within R itself. As mentioned earlier, trying interpolation in a basis other than the standard monomial basis may net numerically better results, though theoretically each basis should deliver the the same unique interpolating polynomial for the same sets of interpolation points.

Our relative error looks better for low degrees like $n = 3, n = 5$ but gets wildly inaccurate near sampling $t = 5$ for high degrees like $n = 33$.








```
testpts2 <- seq(0,5, by=0.1); ft <- f1c(testpts2) #true function values
poly3 <- a3(exp(-testpts2/3)); relerr3 = abs( ft - poly3 ) / abs(ft);
poly9 <- a9(exp(-testpts2/9)); relerr9 = abs( ft - poly9 ) / abs(ft);
poly33 <- a33(exp(-testpts2/33)); relerr33 = abs( ft - poly33 ) / abs(ft);
plot(testpts2, relerr3, main='Relative Error for n = 3',)
plot(testpts2, relerr9, main='Relative Error for n = 9',)
plot(testpts2, relerr33, main='Relative Error for n = 33',)
```

Our code to generate these plots (and construct the interpolating polynomials) is included below.

```
## problem 1c:
f1c <- function(t) 1/(1 + t^6)
# for each n and \lambda_k = 0:-1/n:-1, generate vectors t_j = 0:5/n:5
# and corresponding theta_j
t3 <- seq(0,5, by = 5/3); theta3 <- exp(-t3/3); fn3 <-f1c(t3);
t5 <- seq(0,5, by = 5/5); theta5 <- exp(-t5/5); fn5 <- f1c(t5);
t9 <- seq(0,5, by = 5/9); theta9 <- exp(-t9/9);  fn9 <- f1c(t9);
t17 <- seq(0,5, by = 5/17); theta17 <- exp(-t17/17); fn17 <- f1c(t17);
t33 <- seq(0,5, by = 5/33); theta33 <- exp(-t33/33); fn33 <- f1c(t33);

```

```
makematrix <- function(x) {
  n <- length(x)
  owo <- c() # initialize
  for (j in 1:n) {
    for (i in 1:n) {
      owo <- c(owo, x[j]^i);
    }
  }
  t(matrix(owo, nrow = n))
}

```

```
# set up matrix equation Tx = b for solve(a,b)
# A matrix is theta0^0 , theta0
T3 = makematrix(theta3); x3 = solve(T3, fn3)
T5 = makematrix(theta5); x5 = solve(T5, fn5)
T9 = makematrix(theta9); x9 = solve(T9, fn9)
T17 = makematrix(theta17); x17 = solve(T17, fn17, tol = .Machine$double.xmin)
T33 = makematrix(theta33); x33 = solve(T33, fn33, tol = .Machine$double.xmin)

```

```
# evaluate an input on a polynomial given by a vector of its coefficients
a3 <- as.function(as.polynomial(x3)) ; # same as predict(polynomial(x3), theta3)
a5 <- as.function(as.polynomial(x5)) ;
a9 <- as.function(as.polynomial(x9)) ;
a17 <- as.function(as.polynomial(x17)) ; # same as predict(as.polynomial(x17, theta17))
a33 <- as.function(as.polynomial(x33)) ;

testpts <- seq(0,5, by=0.05); ftrue <- f1c(testpts) ;

plot(testpts, a3(exp(-testpts/3)),
      main='Interpolating Polynomial for n=3 with 100 test points ')
lines(testpts, f1c(testpts))
plot(testpts, a5(exp(-testpts/5)),
      main='Interpolating Polynomial for n=5 with 100 test points ',)
lines(testpts, f1c(testpts))
plot(testpts, a9(exp(-testpts/9)),
      main='Interpolating Polynomial for n=9 with 100 test points ')
lines(testpts, f1c(testpts))
plot(testpts, a17(exp(-testpts/17)),
      main='Interpolating Polynomial for n=17 with 100 test points ')
lines(t33, f1c(t33))
plot(testpts, a33(exp(-testpts/33)),
      main='Interpolating Polynomial for n=33 with 100 test points ')
lines(testpts, f1c(testpts))

```

□

Problem 2. For equidistant points $x_j = j, \quad 0 \leq j \leq n$ (even n), define

$$\omega(x) := (x - x_0)(x - x_1) \cdots (x - x_n).$$

Use Stirling's formula to estimate the ratio

$$\frac{\omega\left[1/2\right]}{\omega\left[n/2 + 1/2\right]}$$

for (sufficiently) large n. Define and explain the Runge phenomenon.

Solution. To estimate the ratio of interest, we can expand our numerator and denominator according to our definition of $\omega(t) := (x - x_0)(x - x_1) \cdots (x - x_n)$ as given above.

$$\begin{aligned} |\omega(1/2)| &= \left(\frac{1}{2}\right)\left(\frac{3}{2}\right)\left(\frac{5}{2}\right)\cdots\left(\frac{2n-1}{2}\right) \\ &= \frac{1}{2^{(2n+1)}}\frac{(2n)!}{n!} \\ |\omega((n+1)/2)| &= [(1/2)(3/2)\ldots((n-1)/2)]^2 = \frac{1}{2^{2n+1}}\frac{(n+1)!n!}{(n/2)!^2} \end{aligned}$$

So our desired ratio can be written as

$$\frac{\omega(1/2)}{\omega(n/2+1/2)} = \frac{(2n)!(n/2)!^2}{(n+1)(n!)^3}$$

Using Stirling's formula $n! \approx \sqrt{2\pi n}(n/e)^n$, we can say that our ratio comes down to about

$$\frac{\omega(1/2)}{\omega(n/2+1/2)} \approx \frac{1}{n}\left(2^{(n-0.5)}\right).$$

Runge phenomenon: When interpolating on equispaced points, we (or rather Runge's grad student, according to Strain) notice that interpolating with high-degree polynomials can cause higher estimation error than might be offered by lower degree polynomials. One work-around for this problem is using locally defined piece-wise polynomials, breaking a large interval down to shorter intervals and using different, lower-degree polynomials.

Another issue is that when using equispaced points, we tend to have high error when sampling or estimating near the ends of our interpolation interval. One work-around for this is to use a larger interpolation interval with the intention of only sampling from the center of the larger interval.

And of course, a better work-around is to not use equispaced points but rather a more clever distribution of interpolation points (i.e. Chebyshev). The Runge Mantra, as given by Strain: "High-degree polynomial interpolation at equispaced points is wildly inaccurate near the ends of the interpolating polynomial."

□

Problem 3. Interpolate the function

$$f(x) := \frac{1}{1+x^6}$$

on the interval $[0,5]$ (with $n := 3, 5, 9, 17, 33$) at:

(a) $n+1$ equidistant points $x_k = \frac{5k}{n}$, and

(b) $n+1$ Chebyshev points $x_k = \frac{1}{2}\left[5 + 5\cos\left(\frac{(2k+1)\pi}{2n+2}\right)\right]$.

Solution. We use R's built-in `polnom` library with `poly.calc` to perform internal Lagrange interpolation and generate a family of polynomials `pa3, pa5, pa9, pa17, pa33` interpolating interspaced points. This seems to work fine; however, for Chebyshev points, doing this gives us very unpredictible (unfavorable) results for Chebyshev. Instead, we manually compute the matrix inverse for Chebyshev points to correspondingly get `pb3, pb5, pb9, pb17, pb33` interpolating at Chebyshev points. It seems to help to lower tolerance from `.Machine$double.eps` to `.Machine$double.xmin`, the analog to realmin in matlab.

```
makeeq <- function(n) {
  seq(0,5, by=(5/n))
}
makecheby <- function(n) {
  chebyuwu <- c() # initialize
  kvalues <- makeeq(n)
  for (k in kvalues ) {
    chebyuwu <- c(0.5*(5 + 5* cos((2*k+1)*pi/(2*n+2)) ), chebyuwu )
  }
  chebyuwu
}

fx <- function(x) 1/(1 + x^6);

# PART A : n+1 equidistant points: (xa3, fna3)
xa3 <- makeeq(3); xa5 <- makeeq(5); xa9 <- makeeq(9); xa17 <- makeeq(17); xa33 <- makeeq(33);
fna3 <- fx(xa3); fna5 <- fx(xa5); fna9 <- fx(xa9); fna17 <- fx(xa17); fna33 <- fx(xa33);

pa3 <- poly.calc(xa3, fna3, tol = .Machine$double.xmin)
pa5 <- poly.calc(xa5, fna5, tol = .Machine$double.xmin)
pa9 <- poly.calc(xa9, fna9, tol = .Machine$double.xmin)
pa17 <- poly.calc(xa17, fna17, tol = .Machine$double.xmin)
pa33 <- poly.calc(xa33, fna33, tol = .Machine$double.xmin)

# PART B : n+1 Chebyshev points: (xb3, fnb3)
xb3 <-makecheby(3); xb5 <- makecheby(5); xb9 <- makecheby(9); xb17 <- makecheby(17); xb33 <-
  makecheby(33);
fnb3 <- fx(xa3); fnb5 <- fx(xa5); fnb9 <- fx(xa9); fnb17 <- fx(xa17); fnb33 <- fx(xa33)

T3 <- makematrix(xb3);
pb3 <- solve(T3, fnb3, tol= .Machine$double.xmin)
T5 <- makematrix(xb5);
pb5 <- solve(T5, fnb5, tol= .Machine$double.xmin)
T9 <- makematrix(xb9);
pb9 <- solve(T9, fnb9, tol= .Machine$double.xmin)
T17 <- makematrix(xb17);
pb17 <- solve(T17, fnb17, tol= .Machine$double.xmin)
T33 <- makematrix(xb33);
pb33 <- solve(T33, fnb33, tol= .Machine$double.xmin)

```

□

(c) Now tabulate the maximum error over 1000 random points $y_k \in [0, 5]$, and

Solution. Interestingly, R seems to give a polynomial interpolation in the least-squares sense, where the error at interpolation values is nonzero. This is verified by running vector-wise or element-wise logical checks == returning false at interpolation points, where our function value should equal the interpolating polynomial value. We see from the plot of absolute error that the error is wildly 'inaccurate' at either end of our interval. Perhaps this is the behavior of the underlying interpolation, or due to intricacies in R.

Our tabulated results for maximum error on the [0,5] interval are as follows, where a is for equispaced point interpolation, and b is for Chebyshev points:

```
> eqabserr
          a3        a5       a9       a17       a33
1 0.05063444 0.1295212 0.153389 0.9201341 77463913
> chebyabserr
        b3       b5       b9       b17          b33
1 1.096548 13.51724 1444.082 0.3580534 869527235959
> eqrelerr
        a3       a5       a9       a17          a33
1 404.0784 627.2195 536.8654 1020.781 7.503752e+12
> chebyrelerr
        b3      b5       b9          b17          b33
1 4706.881 3184.26 1444.082 20647393772 1.038757e+13

```

And to generate the above,

```
    set.seed(100)
    yk <- runif(1000,0,5) # random sample on a uniform distribution

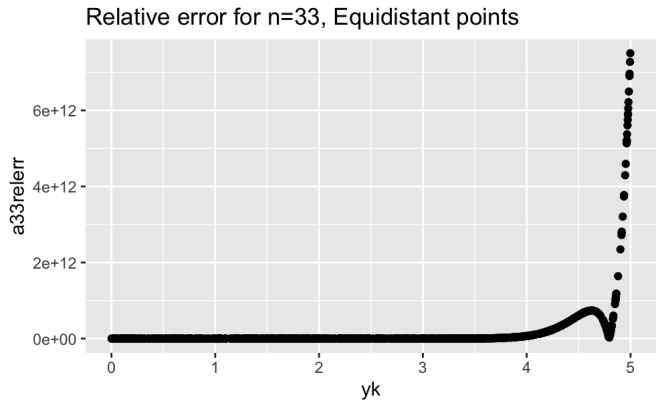
    a3abserr <- as.function(poly.calc(xa3,fna3))(yk) - fx(yk)
    a3relerr <- abs(a3abserr) / fx(yk)
    a5abserr <- as.function(poly.calc(xa5,fna5))(yk) - fx(yk)
    a5relerr <- abs(a5abserr) / fx(yk)
    a9abserr <- as.function(poly.calc(xa9,fna9))(yk) - fx(yk)
    a9relerr <- abs(a9abserr) / fx(yk)
    a17abserr <- as.function(poly.calc(xa17,fna17))(yk) - fx(yk)
    a17relerr <- abs(a17abserr) / fx(yk)
    a33abserr <- as.function(poly.calc(xa33,fna33))(yk) - fx(yk)
    a33relerr <- abs(a33abserr) / fx(yk)

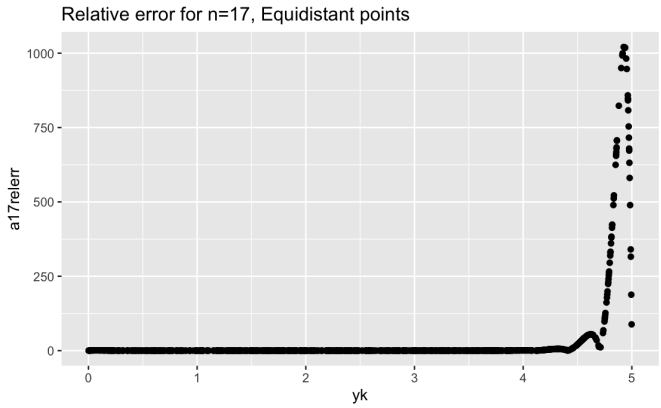
    b3abserr <- as.function(poly.calc(xb3,fnb3))(yk) - fx(yk)
    b3relerr <- abs(b3abserr) / fx(yk)
    b5abserr <- as.function(poly.calc(xb5,fnb5))(yk) - fx(yk)
    b5relerr <- abs(b5abserr) / fx(yk)
    b9abserr <- as.function(poly.calc(xb9,fnb9))(yk) - fx(yk)
    b9relerr <- abs(b9abserr) / fx(yk)
    b17abserr <- as.function(poly.calc(xb17,fnb17))(yk) - fx(yk)
    b17relerr <- abs(b17abserr) / fx(yk)
    b33abserr <- as.function(poly.calc(xb33,fnb33))(yk) - fx(yk)
    b33relerr <- abs(b33abserr) / fx(yk)

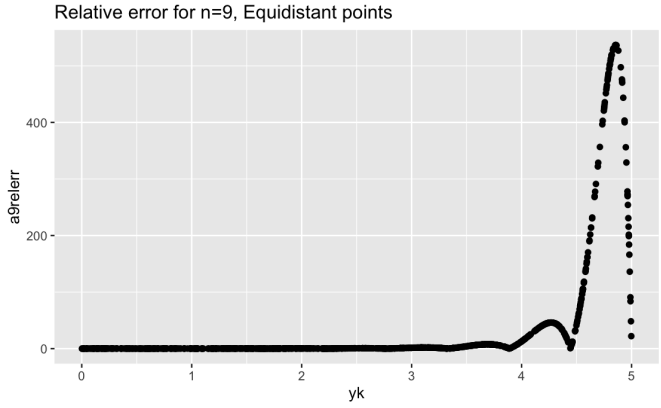
```

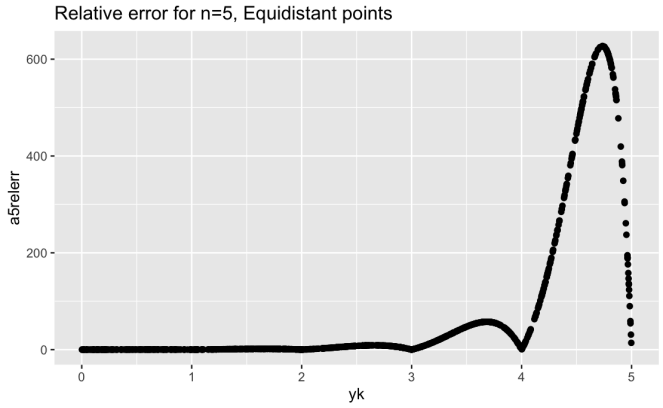
Our maximum errors in the $[0, 5]$ interval look horrendous; however, possibly as a redeeming factor, plots of the relative errors are incuded as follows:

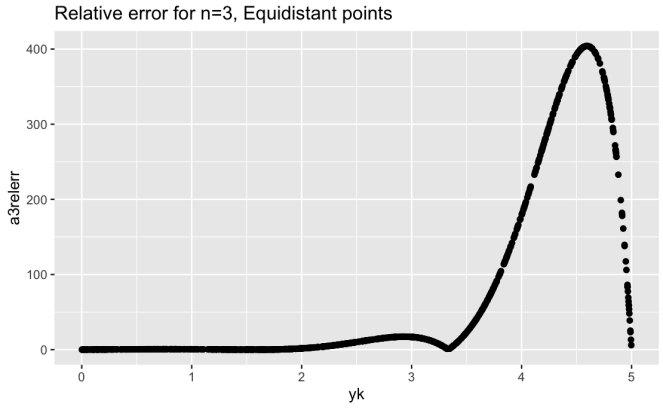
1 Equidistant Points (a3, a5, a9, a17, a33)



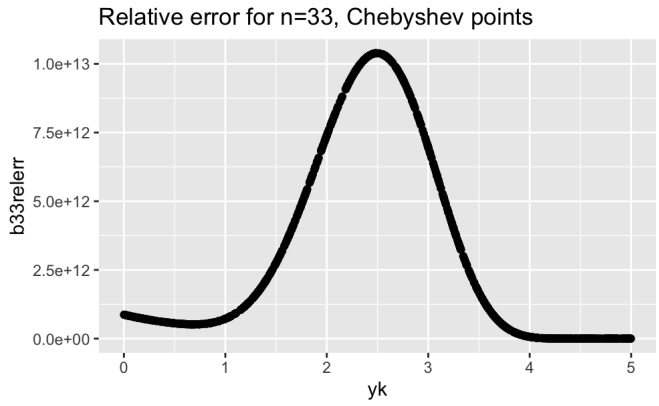


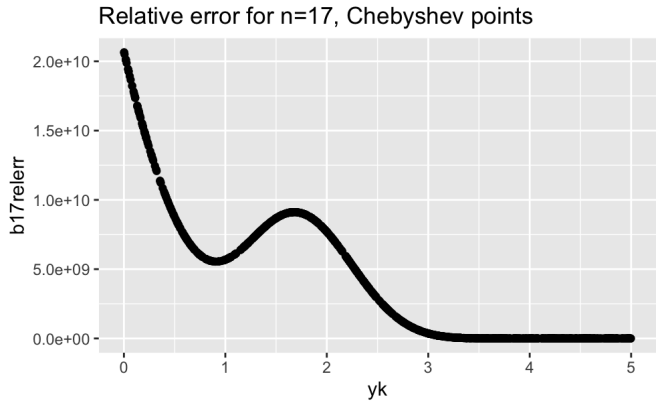


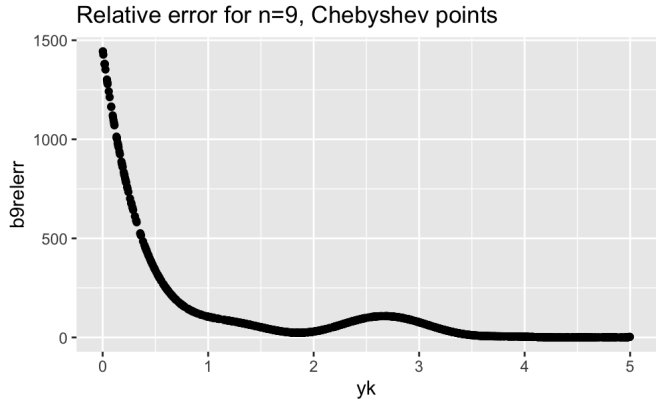

Relative error for n=5, Equidistant points
a5relerr
600
400
200
0
0
1
2
3
4
5
yk

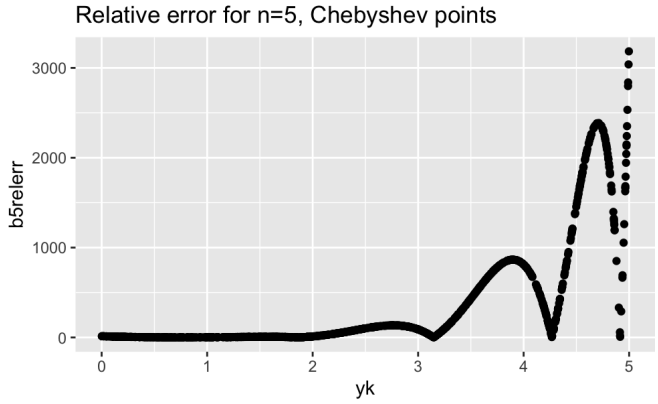

Relative error for n=3, Equidistant points
a3relerr
400
300
200
100
0
0
1
2
3
4
5
yk

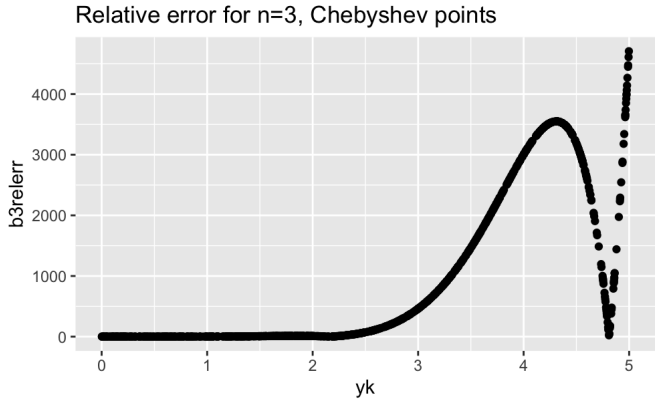
2 Chebyshev Points (b3, b5, b9, b17, b33)







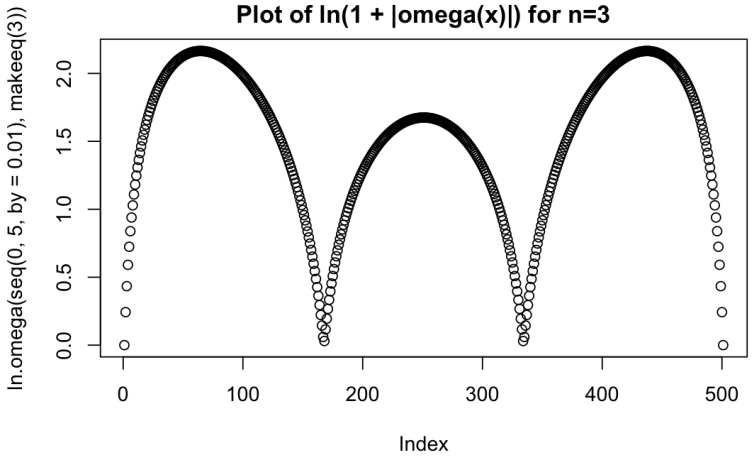

Relative error for n=5, Chebyshev points
b5relerr
yk
0
1000
2000
3000
0
1
2
3
4
5

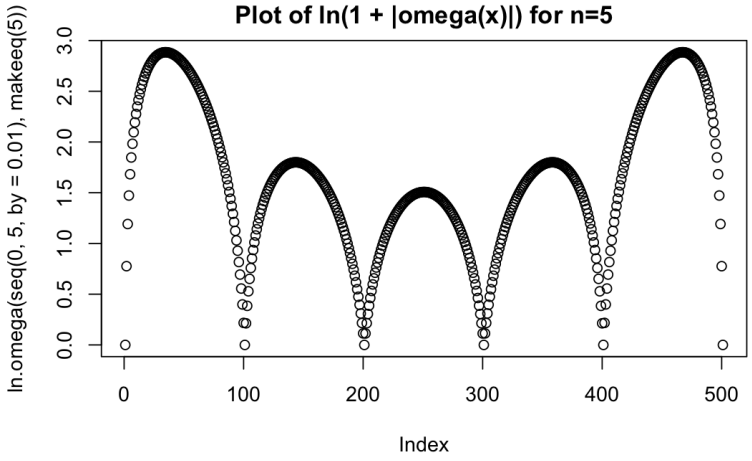

Relative error for n=3, Chebyshev points
b3relerr
yk
0
1000
2000
3000
4000
0
1
2
3
4
5

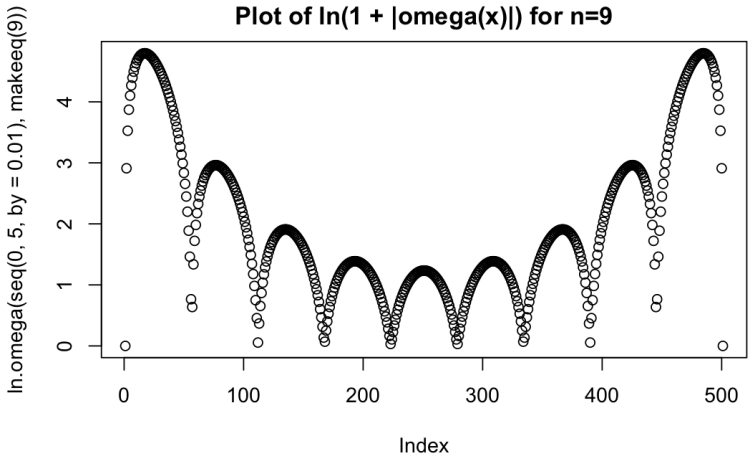
□

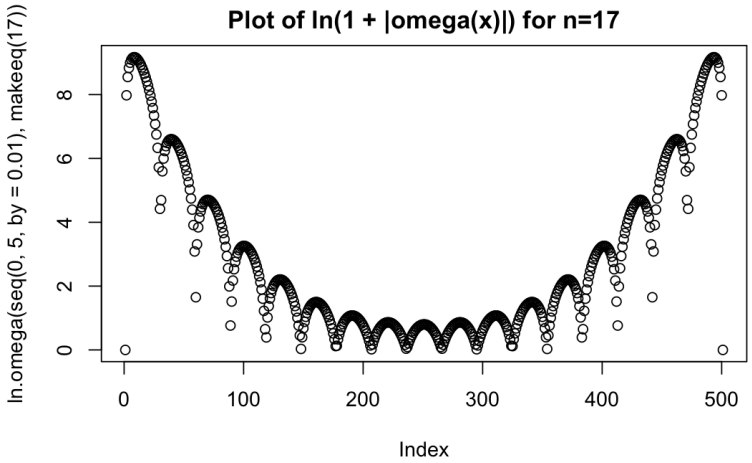
(d) Plot the function $\ln(1 + |\omega(x)|) = \ln(1 + |(x - x_0)(x - x_1) \cdots (x - x_n)|)$.

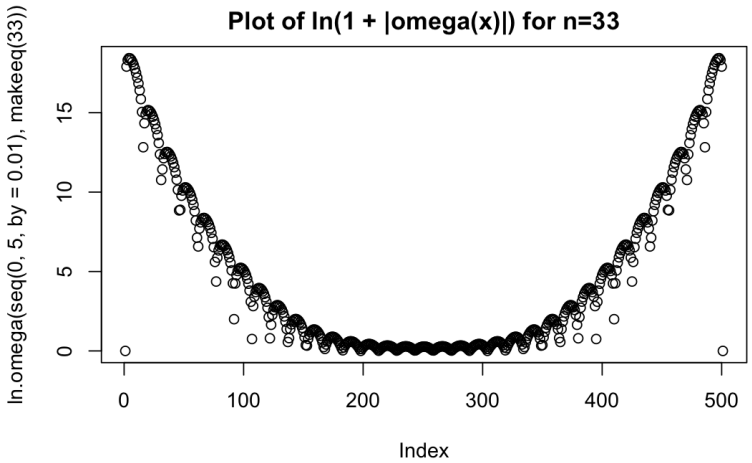
Solution. Recall that for each $n = 3, 5, 9, 17, 33$, we built functions `makeeq(n)`, `makecheby(n)` that build our desired interpolation test points within the interval $[0, 5]$.
Plotting $\ln(1 + |\omega(x)|) = \ln(1 + |(x - x_0)(x - x_1) \dots (x - x_n)|)$ for equidistant x_i and Chebyshev x_i, we get the following figures:






Plot of ln(1 + |omega(x)|) for n=9
ln.omega(seq(0, 5, by = 0.01), makeeq(9))
0
1
2
3
4
0
100
200
300
400
500
Index


Plot of ln(1 + |omega(x)|) for n=17
ln.omega(seq(0, 5, by = 0.01), makeeq(17))
0
2
4
6
8
0
100
200
300
400
500
Index


Plot of ln(1 + |omega(x)|) for n=33
ln.omega(seq(0, 5, by = 0.01), makeeq(33))
0
5
10
15
0
100
200
300
400
500
Index

This is the code used to generate the above plots.

```
bigpi <- function(x,xvec) { # handles the product within the absolute value
  # x : evaluation point, handle
  # xvec : vector of values x0, x1, ..., xn;
  # from makeeq(n) or makecheby(n)
  uwu = 1
  for (i in 1:length(xvec)) {
    uwu <- uwu * (x - xvec[i])
  }
  uwu
}
ln.omega <- function(x,xvec) {log(1 + abs(bigpi(x,xvec))) }
plot(ln.omega(seq(0,5, by = 0.01),makeeq(3)),
     main='Plot of ln(1 + |omega(x)|) for n=3' )
plot(ln.omega(seq(0,5, by = 0.01),makeeq(5)),
     main='Plot of ln(1 + |omega(x)|) for n=5' )
plot(ln.omega(seq(0,5, by = 0.01),makeeq(9)),
     main='Plot of ln(1 + |omega(x)|) for n=9' )
plot(ln.omega(seq(0,5, by = 0.01),makeeq(17)),
     main='Plot of ln(1 + |omega(x)|) for n=17' )
plot(ln.omega(seq(0,5, by = 0.01),makeeq(33)),
     main='Plot of ln(1 + |omega(x)|) for n=33' )

```

□

Problem 4. (See BBF 3.4.11)
Hint: Find a square system of linear equations that determine the coefficients of p in some basis for degree-$(2n+1)$ polynomials. Show that a (possibly non-unique) solution always exists. Use linear algebra.

(a) Show that $H_{2n+1}(x)$ is the unique polynomial p agreeing with f and f' at $x_0, \ldots, x_n$.

Solution. We wish to show that $H_{2n+1}(x)$ is the unique polynomial p agreeing with f and f' at $x = x_0, \ldots, x_n$. To do so, we follow the hint of using a square-matrix representation of a system of linear equations.
We can get equations of the form:

$$\begin{aligned}
P(x_0) = f(x_0) &= a_{2n+1}x_0^{2n+1} + a_{2n}x_0^{2n} + \cdots + a_2x_0^2 + a_1x_0 + a_0 \\
P'(x_0) = f'(x_0) &= (2n+1)a_{2n+1}x_0^{2n} + (2n)a_{2n}x_0^{2n-1} + \cdots + 2a_2x_0 + a_1 \\
P(x_1) = f(x_1) &= a_{2n+1}x_1^{2n+1} + a_{2n}x_0^{2n} + \cdots + a_2x_0^2 + a_1x_0 \\
P'(x_1) = f'(x_1) &= (2n+1)a_{2n+1}x_1^{2n} + (2n)a_{2n}x_1^{2n-1} + \cdots + 2a_2x_1 + a_1 \\
&\vdots \\
P(x_n) = f(x_n) &= a_{2n+1}x_n^{2n+1} + a_{2n}x_n^{2n} + \cdots + a_2x_n^2 + a_1x_n + a_n \\
P'(x_n) = f'(x_n) &= (2n+1)a_{2n+1}x_n^{2n} + (2n)a_{2n}x_n^{2n-1} + \cdots + 2a_2x_n + a_1
\end{aligned}$$

This gives a matrix $A_{(2n+2)\times(2n+2)}$ times a column vector of $[a_{2n+1}; a_{2n}; \ldots; a_1; a_0]$ equals a column vector of $[f(x_0); f'(x_0); f(x_1); f'(x_1); \ldots; f(x_n); f'(x_n)]$. Our matrix A has nonzero determinant, so it is invertible and thus we have that a unique solution exists. The question asks to show that a possibly non-unique solution exists, but we have found that $H_{2n+1}(x)$ is the solution to our system of linear equations, and our solution is unique, so this polynomial is the unique polynomial agreeing with f and f' at those points $x_0, \ldots, x_n$.

□

(b) Derive the error term in Theorem 3.9. (Error expression for Hermite interpolation.)

Solution. To derive the error term in Thm 3.9, we have

$$f(x) - H_{2n+1}(x) - [f(x) - H_{2n+1}(x)] = 0,$$

and as done in lecture, we inherit the Lagrange error and let points merge (we call this "confluence"). This gives $2(n+1)+1 = 2n+3$ zeroes. Because this satisfies the hypotheses for iterated (generalized) Rolles Thm to the $(2n+2)$-th derivative, we have for some $\xi \in [a, b]$

$$|f(x) - H_{2n+1}(x)| = \frac{[(x-x_0)(x-x_1)\ldots(x-x_n)]^2}{(2n+2)!} f^{(2n+2)}(\xi).$$

Notice that this is exactly equivalent to the error expression we are asked to show, which is, from Theorem 3.9 in Burden Faires:

$$f(x) = H_{2n+1}(x) + \frac{(x-x_0)^2 \cdots (x-x_n)^2}{(2n+2)!} f^{(2n+2)}(\xi(x)),$$

and we are done. □

Perhaps the cleanest derivation of this error bound is done by defining the polynomial

$$g(t) := f(t) - H_{2n+1}(t) - [f(x) - H_{2n+1}(x)] \prod_{j=0}^{n} \frac{(t-x_j)^2}{(x-x_j)^2},$$

as is done in Burden Faires' proof for this theorem.

(c) Separate the error into three factors and explain why each factor is inevitable.

Solution. Like we have done in lecture (or as a consequence to theorem 3.9 and its proof), we have that this is the error between the true function and our interpolating Hermite polynomial. For a fixed function f, we know that given sufficient points, its Hermite interpolation polynomial is **unique**. Hence we conclude that the polynomial interpolation error is unique and must be exactly equivalent regardless how we derive the error expression.

We have already shown that the Lagrange error is unique and its three factors are inevitable. This is simply a special case (and thus a direct consequence) from our confluence construction in (b) above.

Or more precisely, our separation of factors is as such:

$$\left[\frac{1}{(2n+2)!}\right]\left[f^{(2n+2)}\left(\xi(x)\right)\right]\left[(x-x_0)^2\cdots(x-x_n)^2\right],$$

where the first factor normalizes when $f(t)=\omega(t)$, the second is makes the zero exactly zero everywhere if f is a degree $(2n+1)$ polynomial, and the third is $\omega(t)^2 := (x-x_0)^2\cdots(x-x_n)^2$ and is zero precisely at the interpolating points. Hence each factor is inevitable and this the correct error expression. □

Problem 5. Let $p \in \mathbb{N}^+$, and for $x \in [0,2]$, define:

$$f(x) := 2^x.$$

(a) Find a formula for the p-th derivative $f^{(p)}(x)$.

Solution. The first derivative is known to be $f'(x) = \ln(2)2^x$. We can show inductively that

$$f^{(p)}(x) = [\ln(2)]^p \, 2^x,$$

but that would be silly because we might as well be defining the infinite-differentiability of $f(x) = 2^x$. □

(b) For $p = 0, 1, 2$, find a formula for the polynomial H_p of degree $(2p+1)$ with

$$H_p^{(k)}(x_j) = f^{(k)}(x_j),$$

for $\quad 0 \le k \le p, \quad 0 \le j \le 1, \quad x_0 := 0, \quad x_1 := 2$.

Solution. We construct the Hermite polynomials H_0, H_1, H_2 using a divided-difference table as done in our Midterm. As given in the problem, we define $x_0 := 0, x_1 := 2$.
For $p = 0$, we have a linear Hermite interpolating polynomial H_0, given by two interpolation points:

$$\begin{bmatrix} x_0 := 0 & f(x_0) = 2^0 = 1 & \frac{4-1}{2} = \frac{3}{2} \\ x_1 := 2 & f(x_1) = 4 & \end{bmatrix}$$

Hence we write

$$H_0 := 1 + \frac{3}{2}(x - 0).$$

For $p = 1$, we have a cubic Hermite interpolating polynomial H_1, given by four interpolation points:

$$\begin{bmatrix} x_0 := 0 & 1 & f'(0) = \ln 2 & \frac{3/2 - [\ln 2]}{2} & \left(\frac{5[\ln 2]}{4} - \frac{3}{4}\right) \\ x_0 := 0 & 1 & \frac{4-1}{2-0} = \frac{3}{2} & \frac{4[\ln 2] - 3/2}{2} & \\ x_1 := 1 & 4 & f'(2) = 4\,[\ln 2] & & \\ x_1 := 1 & 4 & & & \end{bmatrix}$$

Hence we write:

$$H_1(x) := 1 + [\ln 2]\,(x-0) + \left(\frac{3}{4} - \frac{[\ln 2]}{2}\right)(x-0)^2 + \left(\frac{5\,[\ln 2]}{4} - \frac{3}{4}\right)\left[(x-0)^2(x-2)^1\right].$$

Now for $p = 2$, we have a degree-5 Hermite interpolating polynomial H_2. The divided difference table looks disgusting even when done neatly on paper, so let's define our favorite squiggle $\xi := \ln 2$ to make this less of a nightmare(?).

$$\begin{bmatrix} 0 & 1 & \xi & \xi^2 & \left(\frac{-\xi^2}{2} - \frac{\xi}{4} + \frac{3}{8}\right) & \left(\frac{\xi^2}{4} + \frac{3\xi}{4} - \frac{9}{16}\right) & \left(\frac{3\xi^2}{8} - \frac{15\xi}{16} + \frac{9}{16}\right) \\ 0 & 1 & \xi & \left(\frac{3}{4} - \frac{\xi}{2}\right) & \left(\frac{5\xi}{3} - \frac{3}{4}\right) & \left(\xi^2 - \frac{9\xi}{8} + \frac{9}{16}\right) & \\ 0 & 1 & \frac{3}{2} & 2\xi - \frac{3}{4} & \left(2\xi^2 - \xi + \frac{3}{8}\right) & & \\ 1 & 4 & 4\xi & 4\xi^2 & & & \\ 1 & 4 & 4\xi & & & & \\ 1 & 4 & & & & & \end{bmatrix}$$

And so we are incredibly excited to announce that we have:

$$H_2(x) = 1 + \xi(x-0) + \xi^2(x-0)^2 + \left(\frac{-\xi^2}{2} - \frac{\xi}{4} + \frac{3}{8}\right)(x-0)^3 + \left(\frac{\xi^2}{4} + \frac{3\xi}{4} - \frac{9}{16}\right)(x-0)^3(x-2) + \left(\frac{3\xi^2}{8} - \frac{15\xi}{16} + \frac{9}{16}\right)(x-0)^3(x-2)^2.$$

□

(c) For general $p \in \mathbb{N}^+$, prove that:

$$|f(x) - H_p(x)| \leq \left(\frac{1}{p+1}\right)^{2p+2}$$

for $0 \leq x \leq 2$.

Solution. We consider any fixed $p = 1, 2, \ldots$. Recall that in lecture and in Burden Faires, we derived the expression:

$$|f(x) - H_p(x)| = \frac{1}{(2p+2)!}\left|f^{(2p+2)}(\xi)\right| \cdot |\omega(x)|,$$

for some $\xi \in [0,2]$. For p fixed, we have $(2p+2)$ interpolation points and $\omega(x)$ has degree $2p+2$; however, this does not precisely matter, as $\omega(x) := (x-0)^{p+1}(x-2)^{p+1}$ takes a maximum value of $\omega(1) = 1$ over all $x \in [0,2]$. Hence we have that $0 \leq x \leq 2$ implies:

$$|\omega(x)| = \left|(x-0)^{p+1}(x-2)^{p+1}\right| \leq 1.$$

Additionally, recall $f^{(2p+2)}(x) = [\ln(2)]^{2p+2}\, 2^x$, from part (a) above. Taking one further derivative shows $f^{(2p+2)}(x)$ is strictly increasing over the interval $[0,2]$, so we conclude that the maximum value is achieved over all $\xi \in [0,2]$ at the endpoint $\xi = 2$. This gives a bound:

$$\left|f^{(2p+2)}(\xi)\right| \leq \left|f^{(2p+2)}(2)\right| = 2^2 \cdot [\ln(2)]^{2p+2}.$$

As for dealing with the factorial, we use a bound from Stirling's approximation. Stirling's gives us:

$$(2p+2)! \geq \sqrt{2\pi\,(2p+2)}\left[\frac{2p+2}{e}\right]^{2p+2},$$

where taking the reciprocal of both sides gives us a useful bound.

Putting all these together, we have:

$$\begin{aligned}
|f(x) - H_p(x)| = \frac{1}{(2p+2)!}\left|f^{(2p+2)}(\xi)\right| \cdot |\omega(x)| &\leq \frac{4\,[\ln(2)]^{2p+2} \cdot 1}{\sqrt{2\pi\,(2p+2)}\left[\frac{2p+2}{e}\right]^{2p+2}} \\
&= \left(\frac{1}{2p+2}\right)^{2p+2}\left[\frac{4}{\sqrt{2\pi\,(2p+2)}}\,(e\ln(2))^{2p+2}\right] \\
&= \left(\frac{1}{p+1}\right)^{2p+2.5}\left(\frac{4}{\sqrt{4\pi}}\left[\underbrace{\frac{e\ln(2)}{2}}_{<1}\right]^{2p+2}\right) \\
p \in \mathbb{N}^+ \implies \qquad &\leq \left(\frac{1}{p+1}\right)^{2p+2},
\end{aligned}$$

as desired. □

(d) Show that one step of Newton's method for solving

$$g(y) = x \ln 2 - \ln y = 0$$

starting from $y = H_4(x)$ gives $y_1 = f(x) = 2^x$ to almost double precision accuracy for $0 \le x \le 2$.

Solution. Recall that Newton's method takes on the form

$$x_{n+1} := f(x_n) - \frac{f(x_n)}{f'(x_n)}.$$

Then for our definition of $g(x) := x \ln 2 - \ln y = \ln\left(\frac{2^x}{y}\right) = 0$, we are finding $y = 2^x = f(x)$. Consider the homogenous differential equation:

$$\frac{d\theta}{dy} := x \ln 2 - \ln y = 0; \qquad \frac{d^2\theta}{dy^2} = \frac{-1}{y}; \qquad \theta = y + yx \ln 2y - y \ln y.$$

Then a Taylor expansion about $\hat{y}$ gives our typical error bound via MVT:

$$\begin{aligned} y_{n+1} = \theta(y_n) &= \theta(\hat{y})(y_n - \hat{y})^0 + \underbrace{\theta'(\hat{y})}_{0 \text{ everywhere}} (y_n - \hat{y})^1 + \frac{\theta''(\xi)}{2!}(y_n - \hat{y})^2 \\ &= \theta(\hat{y}) + \frac{1}{2\xi}(y_n - \hat{y})^2 \end{aligned}$$

for $\xi \in [\min(y_n, y_{n+1}), \max(y_n, y_{n+1})]$. Subtracting $\hat{y} = \theta(\hat{y})$ from both sides and taking absolute values, we get:

$$|y_{n+1} - \hat{y}| = \left|\frac{1}{2\xi}\right| |y_n - \hat{y}|^2,$$

and this holds particularly for $n = 0$:

$$|y_1 - \hat{y}| \le \frac{1}{2\xi} |y_0 - \hat{y}|^2 \le \frac{1}{2} |y_0 - \hat{y}|.$$

From part (c) above, for our starting $y_0 := H_4(x); \quad p = 4$, we found a bound for error to be:

$$\begin{aligned} |f(x) - H_p(x)| = |2^x - y_0| &\le \frac{4\left[\ln(2)\right]^{2p+2} \cdot 1}{\sqrt{2\pi\,(2p+2)}\left[\frac{2p+2}{e}\right]^{2p+2}} \\ &= \frac{4\left[\ln(2)\right]^{2(4)+2}}{\sqrt{2\pi(2\cdot 4+2)}\left[\frac{10}{e}\right]^{10}} \approx 2^{-25.06671} \end{aligned}$$

Plugging this expression into R, this gives:

```
> (4*(log(2))^(10))/((sqrt(2*pi*10)*(10/exp(1))^(10)))
[1] 2.845571e-08
> a = 4 * (log(2))^10; b = sqrt(2* pi * 10); c = (10/exp(1))^10; d = a/b/c;
> d
[1] 2.845571e-08
> log(2.845571e-08, base = 2)
[1] -25.06671

```

Notice that our bounds for $|\omega(x)|$ assume $x = 1$ in the interval $[0, 2]$, whereas the maximum bound for $f^{(2p+2)}(\xi)$ took $\xi = 2$. However, proceeding with this bound,

$$|2^x - y_1| \le \frac{1}{2}\left(2^{-25.06671}\right)^2 \le 2^{-51},$$

and we can conclude that at the worst-case scenario, we get slightly better than 2^{-51}, almost full bits double precision accuracy as required.

□

Problem 6. Let $0 \leq m \leq n$, $a \in \mathbb{R}$, and take $(n+1)$ distinct interpolation points: $\{x_0, x_1, \ldots, x_n\}$. Let $\delta_{nk}^m(a)$ be the differentiation coefficients:

$$\delta_{nk}^m(a) := \left(\frac{d}{dx}\right)^m L_k^n(x)\big|_{x=a},$$

such that the n-degree polynomial $p(x)$ that interpolates $(n+1)$ values $f_j = f(x_j)$, for $j = \texttt{0:n}$ satisfies:

$$p^{(m)}(a) = \sum_{k=0}^{n} \delta_{nk}^m(a) f_k.$$

(a) Derive the recurrence relation for $0 \leq k \leq (n-1)$:

$$\lambda_{nk}^m(a) = \frac{m}{x_k - x_n}\delta_{n-1,k}^{m-1}(a) + \frac{a - x_n}{x_k - x_n}\delta_{n-1,k}^m(a)$$

Solution. As given in office hours, we take the definition of $\delta_{n,k}^m := \left(\frac{d}{dx}\right)^m L_k^n(x)|_{x=a}$ to suggest we have derivatives to supply a Taylor series expansion on the Lagrange basis. From class, we have the recurrence relation:

$$L_j^n(x) = \frac{x - x_n}{x_j - x_n} L_j^{n-1}(x), \tag{2}$$

so we can define

$$L_j^n(x) := \sum_{m=0}^{n} \frac{\delta_{n,k}^m}{m!} [x-a]^m,$$

to be able to substitute in to our equation (2) above. This nets us

$$\sum_{m=0}^{n} \frac{\delta_{nj}^m}{m!} [x-a]^m = \frac{(x-a) - (x_n - a)}{x_j - x_n} \cdot \sum_{m=0}^{n} \frac{\delta_{(n-1),j}^m}{m!} [x-a]^m.$$

Multiplying out, we get:

$$\sum_{m=0}^{n} \frac{\delta_{n,k}^m}{m!}[x-a]^m = \left(\frac{1}{x_k - x_n} \sum_{m=1}^{n} \frac{\delta_{n-1,k}^{m-1}}{(m-1)!}[x-a]^m\right) - \left(\frac{x_n - a}{x_k - x_n} \sum_{m=0}^{n-1} \frac{\delta_{n-1,k}^m}{m!}[x-a]^m\right),$$

which precisely satisfies the recurrence relation we are asked to derive:

$$\lambda_{nk}^m(a) = \frac{m}{x_k - x_n}\delta_{n-1,k}^{m-1}(a) + \frac{a - x_n}{x_k - x_n}\delta_{n-1,k}^m(a)$$

□

(b) Write a Matlab code that evaluates $\delta_{nk}^m(a)$ for $0 \leq m \leq M$, given some n and the points $a, x_j \in \mathbb{R}$.

Solution. Per usual, we do this in R.

```
  ```{r}
getdeltas <- function(n,M,a,x) {
  # n index for delta (diff coefficient)
  #   delta_{n,k}
  # any points a, x_j \in \R
  delta.mx <- matrix(data = 0, nrow = (n+1), ncol = (M+1))
  for (k in (1:(n+1))) {
    d <- matrix(data = 0, nrow = (n+1), ncol = (M+1));
    d[1,1] <- 1
    for (m in 2:(M+1)) {
      for (i in 2:(n+1)){
        d(ni,m) <- ( m*d(ni-1,m-1) + (a-x(n))*d(ni-1,m) )/ (x(1) - x(ni));
      }
    }
    delta.mx[k, ] <- d[n+1, ]
  }
  delta.mx
}

```
```

□

(c) Validate your coefficients $\delta_{nk}^m(a)$ by verifying $O(h^{n-m})$ accuracy for the m-th derivative of $f(x) = e^x$, evaluated at $(n+1)$ equidistant points $x_j = jh$.

Solution. We tabulate the errors $O\left(h^{n-m}\right)$ for $f(x) = e^x$, setting:

```
```{r}
getx <- function(f,h,n) {
  # f: function handle
  # h: used for spacing of points
  # n+1 : number of points
  f( seq(0,h, by = (h/(n+1))) )
}

fapprox <- function(f,h,n,M,a,x) {
  # matrix multiplication to get linear approx
  getx(f,h,n) * getdeltas(n,M,a,x)
}

f <- function(x) exp(x)
```
```

Let $M := 6$ for our test. Setting $h = 0.01$ for `a1` and $h = 0.1$ for `a2` and comparing, we get:

```
> a1
[1] 2.295916e-06 9.693143e-05 3.813814e-04 7.612386e-03
[5] 4.237194e-02 6.862510e-01
> a2
[1] 2.327783e-12 1.437477e-09 2.813867e-08 1.306234e-05
[5] 4.007352e-04 6.881931e-02
> a2/a1
[1] 1.013880e-06 1.482983e-05 7.378093e-05 1.715932e-03
[5] 9.457563e-03 1.002830e-01
> log(a2/a1, base=10)
[1] -5.9940136 -4.8288637 -4.1320559 -2.7654999
[5] -2.0242208 -0.9987727
```

By taking the ratio a_2/a_1, our accuracy looks somewhat close to $O(h^{n-m})$, which is a factor of 10 for our chosen $h = 0.01, 0.1$. □

(d) Fix interpolation points x_j and form an $(n+1) \times (n+1)$ matrix A_m of differentiation coefficients with:

$$(A_m)_{ij} = \delta_{nj}^m(x_i).$$

Is $A_m = A_1^m$? Why or why not?

Solution. This is similar to how we constructed our program to generate the differentiation coefficients, and it would make sense that taking the mth derivative works the same way as taking 1 derivative, m times. By linear algebra, we express differentiation as a linear map, and thus iterations the first derivative A_1^m definitely should be equivalent to A_m, plus or minus computational numerical errors.
Theoretically, they should be equal. Let us formalize by induction. Consider the set $U \subset \mathbb{N}$ of m for which $A_1^m = A_m$. Trivially, $1 \in U$, because $A_1^1 = A_1$. We do not need to reason that they both perform the first derivative; this simply follows from the definition of the first iteration of a linear self-map A_1.
Now assume $m \in U$, so that $[A_1]^m = A_m$. We are interested in iterated powers of A_1, so we want an expression precisely for the first derivative. Given in the question, our n-degree polynomial $p(x)$ interpolates at $(n+1)$ values $f_j = f(x_j)$ and satisfies:

$$p'(a) = \sum_{k=0}^{n} \left[\delta_{n,k}^1(a) f_k\right].$$

Notice that if $f(x) := \frac{d^m}{dx^m} L_k^n(x)|_{x=x_i}$, our polynomial interpolates a function with lesser degree, and hence our interpolation matches the function value at all points. That is,

$$p(x_j) = f_j = \frac{d^m}{dx^m} L_k^n(x)|_{x=x_j} = \delta_{n,k}^m(x_j).$$

Hence taking the $m+1$th derivative of $L_k^n(x)|_{x=x_i}$ gives:

$$\frac{d^{m+1}}{dx^{m+1}} L_k^n(x)|_{x=x_j} = p'(x_j = a) = \sum_{j=0}^{n} \delta_{n,j}^1(a) f_j.$$

But we have $f_j = p(x_j) = \delta_{n,k}^m(x_j)$, so this gives us:

$$\frac{d^{m+1}}{dx^{m+1}} L_k^n(x)|_{x=x_j} = p'(x_j = a) = \sum_{j=0}^{n} \left[\delta_{n,j}^1(a) \delta_{n,k}^m(x_j)\right].$$

Then left-multiplying A_m by A_1, we have:

$$[A_1 \cdot A_m]_{u,v} = \sum_{j=0}^{n} \left[\delta_{n,j}^1(x_u) \cdot \delta_{n,v}^m(x_j)\right] = [A_{m+1}]_{u,v},$$

but this is precisely our expression for $m \in U$. Because all our steps are reversible, we conclude $m \in U \implies m+1 \in U$, and thus by induction we have for all $m \in \mathbb{N}$:

$$A_1^m = A_m,$$

as desired.

□