Lecture 7: Approximating a function and its derivative(s)

TANA21/22: Beräkningsmatematik *Andrew R. Winters* *Autumn 2019*

We have seen that we can use interpolation to approximate a function $f(x)$ using a discrete set of known function values at a unique set of points. However, we saw that a global approximation could lead to large, spurious oscillations in the approximation near the endpoints of the interval, so-called Runge phenomena. We could remedy this issue if we changed ***how*** the polynomial interpolant was created moving from a global approximation to a local one. This was the idea behind splines.

In this lecture we will remedy this issue for a global interpolation by changing ***where*** the interpolation occurs. In particular, we will use very specific sets of interpolation nodes that are nonuniform but concentrate the nodes close to the interval boundary (where the pointwise error was the largest). After this we will discuss some basic strategies to approximate the derivative of a function in a global sense.

7.1 Improving the global approximation

We return to the approximation of a function $f(x)$ with a global interpolating polynomial that, recall, was denoted $p_n(x)$. This interpolating polynomial is *unique* for a given set of data points

$$\{(x_i, f(x_i))\}_{i=0}^n = \{(x_i, f_i)\}_{i=0}^n$$

although it can be written in a variety of different forms, like with the Lagrange basis polynomials. Further, we had the pointwise error between the function and its polynomial interpolant take the form

$$E_n(x) = f(x) - p_n(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(n+1)!}f^{(n+1)}(\xi) = \frac{\Psi_n(x)}{(n+1)!}f^{(n+1)}(\xi),$$

for some value of $\xi \in [a,b]$. Ideally, the pointwise error should become smaller as the number of nodes increases in the polynomial interpolant. However, we have seen that this may not be the case particularly for equidistant interpolation points. Such nodes could result in the Runge phenomena we saw in the previous lecture. So, should we even expect this error to shrink? Can we approximate a function arbitrarily well with a global interpolating polynomial? It turns out ***yes*** we can, due to a Theorem by Weierstrass. But Weierstrass' work is a mathematical result that has to do with the existence of this so-called "best" approximation, but it is difficult to apply the result in practice.

We at least know that what we aim to do is possible. That is, use a global polynomial to represent a function where the pointwise error vanishes as the degree of the polynomial interpolation increases

$$\lim_{n\to\infty} |E_n(x)| = \lim_{n\to\infty} |f(x) - p_n(x)| = 0.$$

If we examine this error piece-by-piece we see that it involves three terms:

1. The monic polynomial $\Psi_n(x)$.
2. The higher order derivative $f^{(n+1)}(\xi)$ with $\xi \in [a,b]$.
3. The factorial $(n+1)!$

Therefore, the magnitude of the pointwise error is a balancing act between the product (or division) of these three components. Over which of these terms do we have some type of *control*? The factorial term is directly related to the degree of the polynomial interpolant we construct. The higher order derivative term makes an implicit assumption that $f^{(n+1)}(\xi)$ does not grow too rapidly with n. But we have no real control over the given function or how its derivatives behave. What we ***can*** do is force the product

$$\Psi_n(x) = (x - x_0)(x - x_1)\cdots(x - x_n) = \prod_{i=0}^{n}(x - x_i),$$

to be as small as possible on the given interval $[a, b]$.

For the simple choice of equally spaced interpolation points, the magnitude of this monic polynomial increases as one moves from the interior of the interval toward the endpoints. Further, this magnitude becomes larger and more pronounced as the number of interpolation nodes n increases. This is why the interpolant becomes less reliable as one approaches the leftmost or rightmost endpoint of the interval $[a, b]$.

Fortunately, we are free to change the location of the nodes on which the interpolation is built. This is true for the global interpolation strategy as well as the piecewise strategy with splines. We have this freedom, but how do we get a "better" set of interpolation points?

To answer this question we have to take a brief detour to discuss orthogonal polynomials. Conceptually, we can think of functions in a similar fashion as we think about vectors in linear algebra. As such, we say that polynomials of degree n form a function space $\mathbb{P}^n$ (analogous to a vector space $\mathbb{R}^n$). We have already seen that members of this space of polynomials $\mathbb{P}^n$ can be represented with many different basis functions. One example is the monic polynomials

$$\{1, x, x^2, x^3, \ldots, x^n\}.$$

But, as we discussed before, some of the basis representations for the unique interpolating polynomial are easier to work with than others. Just like working with vectors, an orthogonal basis will make the representation and approximation much "cleaner". We must define what it means for two polynomial functions to be orthogonal. Continuing the analogue to linear algebra, we need a mechanism to compute the inner product between two polynomials. That is, we need a mathematical tool to multiply two polynomials and obtain a scalar value. To do this we define the following:

Definition 7.1 (Inner product of two polynomials). On a given interval $[a, b]$ the *inner product* between two polynomials $p(x)$ and $q(x)$ is defined by

$$(p, q) = \int_a^b p(x)\, q(x)\, w(x)\, \mathrm{d}x,$$

where $w(x)$ is a nonnegative weight function. ◀

Remark 7.1. Now it is well-defined what we mean if two polynomials are *orthogonal* because then their inner is zero

$$(p, q) = 0.$$

⋈

The subject of orthogonal polynomials is vast and we will only scratch the surface here. Two particularly useful families of orthogonal polynomials that appear in many aspects of mathematics (not just in scientific computing) are the *Legendre polynomials* and the *Chebyshev polynomials*.

Definition 7.2 (Legendre polynomials). On the interval $[-1, 1]$ with the weight function $w(x) = 1$ the set of Legendre polynomials of order zero up to order n are defined by the three-term recurrence relation

$$L_0(x) = 1, \qquad L_1(x) = x, \qquad L_{k+1} = \frac{2k+1}{k+1} x L_k(x) - \frac{k}{k+1} L_{k-1}(x),$$

for $k = 1, \ldots, n$. ◀

Remark 7.2. Because the weight function for the Legendre family of polynomials is just the constant function $w(x) = 1$, they are very useful to develop approximations to an integral. We will revisit the Legendre family of polynomials in the next Lecture when we talk about certain integral approximations. $\bowtie$

EXAMPLE 7.1. Write out the next three Legendre polynomials $L_2(x)$, $L_3(x)$, $L_4(x)$. We start with the $k = 1$ case to generate

$$L_2(x) = \frac{2+1}{1+1}xL_1(x) - \frac{1}{1+1}L_0(x) = \frac{3}{2}x(x) - \frac{1}{2}(1) = \frac{1}{2}(3x^2 - 1).$$

Next, for $k = 2$ we find

$$L_3(x) = \frac{2(2)+1}{2+1}xL_2(x) - \frac{2}{2+1}L_1(x) = \frac{5}{3}x\left(\frac{1}{2}(3x^2-1)\right) - \frac{2}{3}(x) = \frac{1}{2}(5x^3 - 3x).$$

Finally, for $k = 3$ the recursion produces

$$L_4(x) = \frac{2(3)+1}{3+1}xL_3(x) - \frac{3}{3+1}L_2(x) = \frac{7}{4}x\left(\frac{1}{2}(5x^3-3x)\right) - \frac{3}{4}\left(\frac{1}{2}(3x^2-1)\right) = \frac{1}{8}(35x^4 - 30x^2 + 3).$$

$\blacklozenge$

Of particular interest to the topic of interpolation are another set of orthogonal polynomials.

DEFINITION 7.3 (CHEBYSHEV POLYNOMIALS). On the interval $[-1, 1]$ with weight function

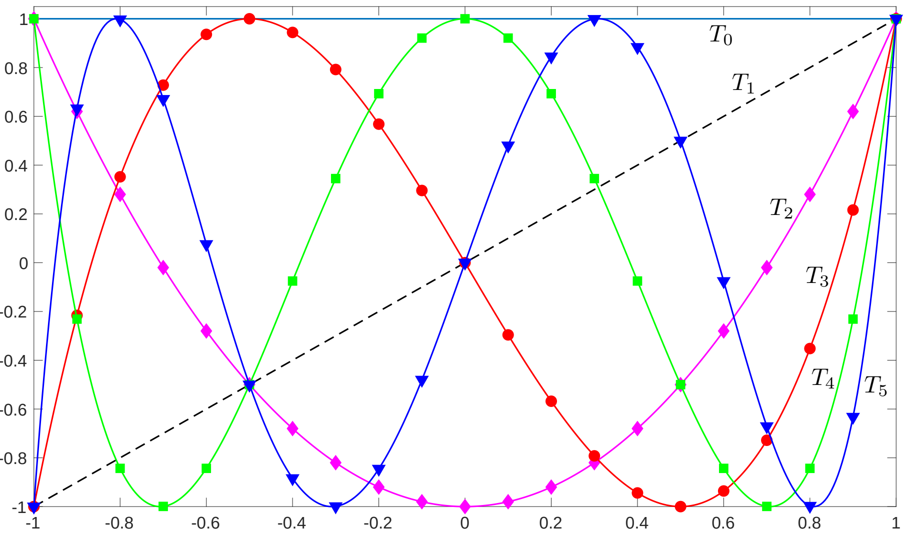
$$w(x) = \frac{1}{\sqrt{1-x^2}}$$

the set of Chebyshev polynomials of order zero up to order n are defined by the three-term recurrence relation

$$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_{k+1} = 2xT_k(x) - T_{k-1}(x),$$

for $k = 1, \ldots, n$. $\blacktriangleleft$

For either the Legendre or the Chebyshev polynomials an important observation is that a polynomial of degree k contains *only even* degree terms when k is even and *only odd* degree terms when k is odd. Perhaps more important is an equioscillation property that can be observed visually. We plot the first six Chebyshev polynomials T_k, $k = 0, \ldots, 5$ below.



We see that the successive extrema of the polynomial functions $T_k(x)$ are equal in magnitude and alternate sign. This is of interest for us because such a property tends to uniformly distruibute the pointwise error within the interval when we use interpolation to approximate an arbitrary continuous function $f(x)$.

This equioscillation has particularly useful implications for polynomial interpolation because the maximum value of the pointwise error over the interval is minimized when the interpolation points are chosen to be the extrema of the Chebyshev polynomials of appropriate degree. So the Chebyshev polynomials give us a set of interpolations nodes such that the monic polynomial term $\Psi_n(x)$ is as small as possible, which was our goal.

Great! We have identified the new set of interpolations points. Now we need to determine an explicit form for the locations of these extrema. In order to do so, we note the following usual property of Chebyshev polynomial family.

Remark 7.3. An equivalent form of the Chebyshev polynomials is

$$T_k(x) = \cos(k\cos^{-1}(x)), \quad x \in [-1,1],$$

where $k = 0, 1, \ldots, n$. ⋈

Though it is not obvious, this representation is still a polynomial function. This can be verified through the application of several trigonometric identities. From this trigonometric form of the Chebyshev polynomials it is straightforward to determine the value of the extrema.

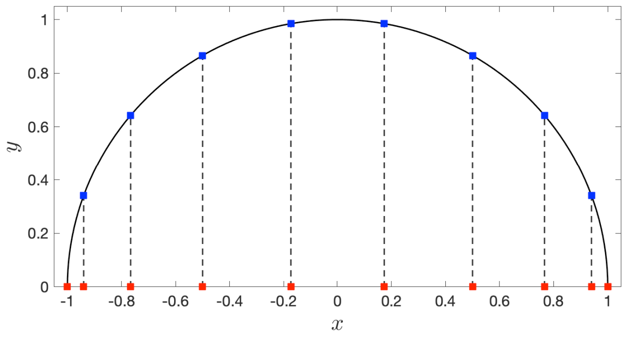
Definition 7.4 (Chebyshev extrema). The Chebyshev polynomial of degree k takes on $k+1$ extrema at the nodes

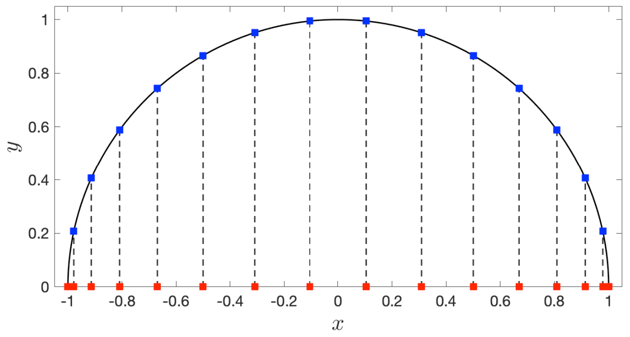
$$x_j = \cos\left(\frac{j\pi}{k}\right),$$

where $j = 0, \ldots, k$. All the extrema occur in the interval $[-1,1]$ and the endpoints are *included*. ◀

From here on we refer to this set of extreme values as the *Chebyshev nodes*. Note, they are not uniformly distributed in the interval $[-1,1]$. But they are symmetrically distributed around the point $x = 0$ and concentrated near the ends of the interval. So, heuristically, the pointwise error in the global interpolation is reduced because we put more nodes near the endpoints of the interval (where the pointwise error was the worst) and fewer points in the interior of the interval (where the pointwise error was better).

Before we examine their performance in creating a global polynomial interpolant we note one last interesting feature about the Chebyshev nodes. We know that they are not uniformly distributed in the interval $[-1,1]$ but they ***are*** spaced uniformly on the unit circle. We present this in the figures below for $n = 9$ on the left and $n = 15$ on the right.





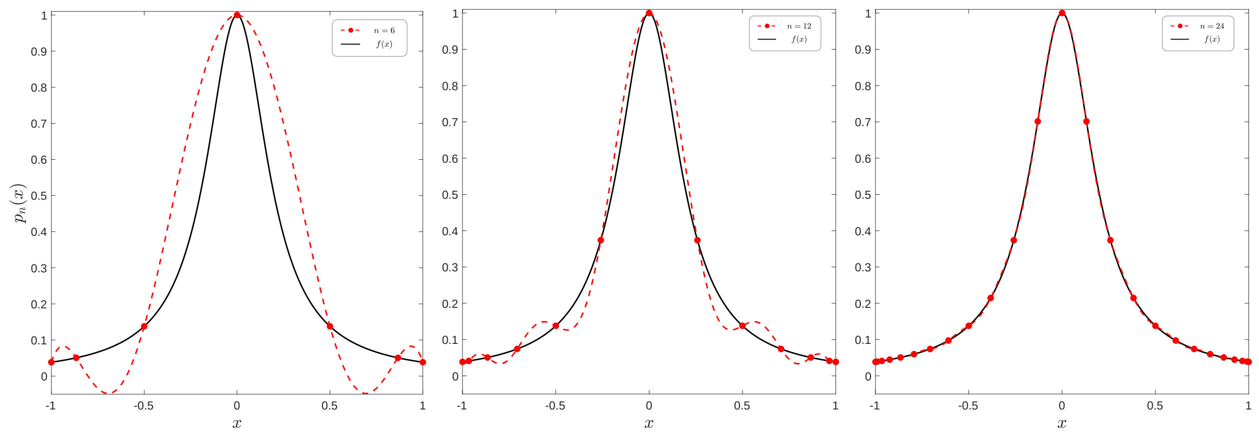
The uniform distribution of point on the unit circle are then projected onto the interval $[-1,1]$ that results in the clustering near the endpoints because we project values from a curve onto a straight line. Also, we see that as the number of Chebyshev nodes increases, the interior has nearly uniform nodes whereas but there is still clustering near the endpoints of the interval.

Let us now see how the Chebyshev nodes perform to create a global interpolating polynomial for a function $f(x)$

Example 7.2. We revisit the function

$$f(x) = \frac{1}{1+25x^2}$$

that gave us trouble with uniformly spaced interpolation nodes and produced Runge phenomena. For this we construct the Lagrange interpolating polynomial basis and select the Chebyshev nodes as the interpolation points. We plot three instances of the global interpolating polynomial below for increasing values of the degree of the polynomial interpolant $n = 6$, $n = 12$, and $n = 24$.



We see that increasing the polynomial degree of the interpolation no longer produces Runge phenomena. The Chebyshev nodes has removed this deleterious effect. Moreover, we see that the use of the Chebyshev nodes more evenly distributes the pointwise error throughout the interval. $\blacklozenge$

Now that we use the Chebyshev nodes the Runge phenomena are removed. Furthermore, we now observe the desired behavior that the global interpolating polynomial becomes more accurate as the degree increases. In fact, it appears to do so quite rapidly! For example, look above at the approximation when we double the number of Chebyshev nodes from $n = 12$ to $n = 24$. In the pointwise error we have minimized the value of the monic polynomial, which is all we had control over. The convergence of the global interpolating polynomial now depends solely on the behavior on the derivatives of the function $f(x)$.

Definition 7.5 (Convergence of global interpolation). Suppose we create a global polynomial interpolant for the function $f(x)$ on $n+1$ Chebyshev nodes in the interval $[-1, 1]$. If the function $f(x)$ is continuous and bounded on $[-1, 1]$ ***and*** it is also continuous and bounded on $[-1, 1]$ up to its m^{th} derivative then the pointwise error decays according to

$$|E_n(x)| = |f(x) - p_n(x)| \sim \mathcal{O}(n^{-m}).$$

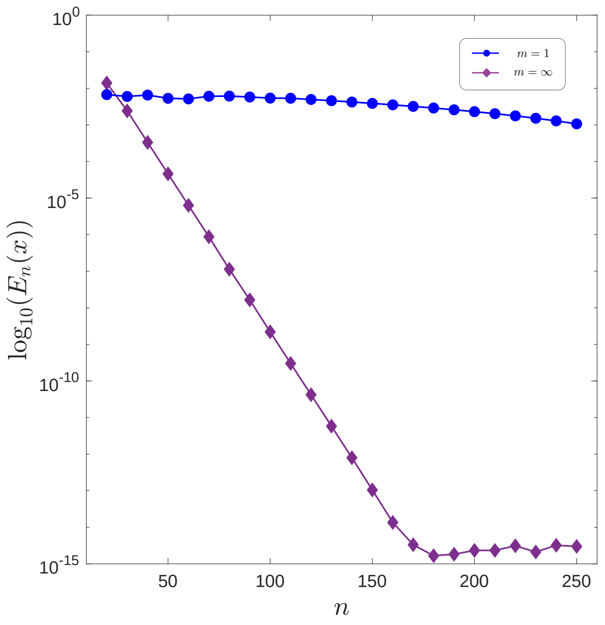
which is a negative power on the degree of the interpolating polynomial. In other words, the error in the global polynomial interpolant decays according to the "smoothness" of the function $f(x)$. $\blacktriangleleft$

Remark 7.4. This accounts for the rapid rate of convergence for the function $f(x) = (1 + 25x^2)^{-1}$. It is infinitely differentiable and bounded in the interval $[-1, 1]$. Therefore, the convergence actually occurs faster than $\mathcal{O}(n^{-m})$ no matter how large we take m. This is commonly called *exponential accuracy*. $\bowtie$

Example 7.3. We examine the behavior of this type of convergence that depends on how many times a function is differentiable. To do so, we consider the interval $[-1, 1]$ and take the very smooth function $f(x) = (1 + 25x^2)^{-1}$ (where $m = \infty$) and the function $f(x) = |x|$ that is continuous but has only one derivative (so $m = 1$). We present a plot of the error as a function of the polynomial degree on the top right of the next page. Note that it is in a semilog scale. This is to demonstrate that as we increase the polynomial order n the pointwise error (in the scale of the common logarithm $\log_{10}$) decays along a straight line for the infinitely differentiable function whereas the convergence stagnates for $f(x) = |x|$ because it lacks sufficient smoothness. Also, notice that eventually, if n is large enough, the pointwise error for the global interpolating polynomial and $f(x) = (1 + 25x^2)^{-1}$ reaches the magnitude of double precision roundoff. $\blacklozenge$

Of course, one may have no choice in placing the interpolation nodes, either because of existing measured data or because a particular distribution (like uniformly space) is required for other reasons. But when one does have the freedom to choose the locations of the interpolation points, which is often the case for problems in scientific computing, the Chebyshev nodes are an excellent choice.

We can also use the Chebyshev nodes to approximate a function that is not in the interval $[-1, 1]$. To do this we use a simple variable transformation:



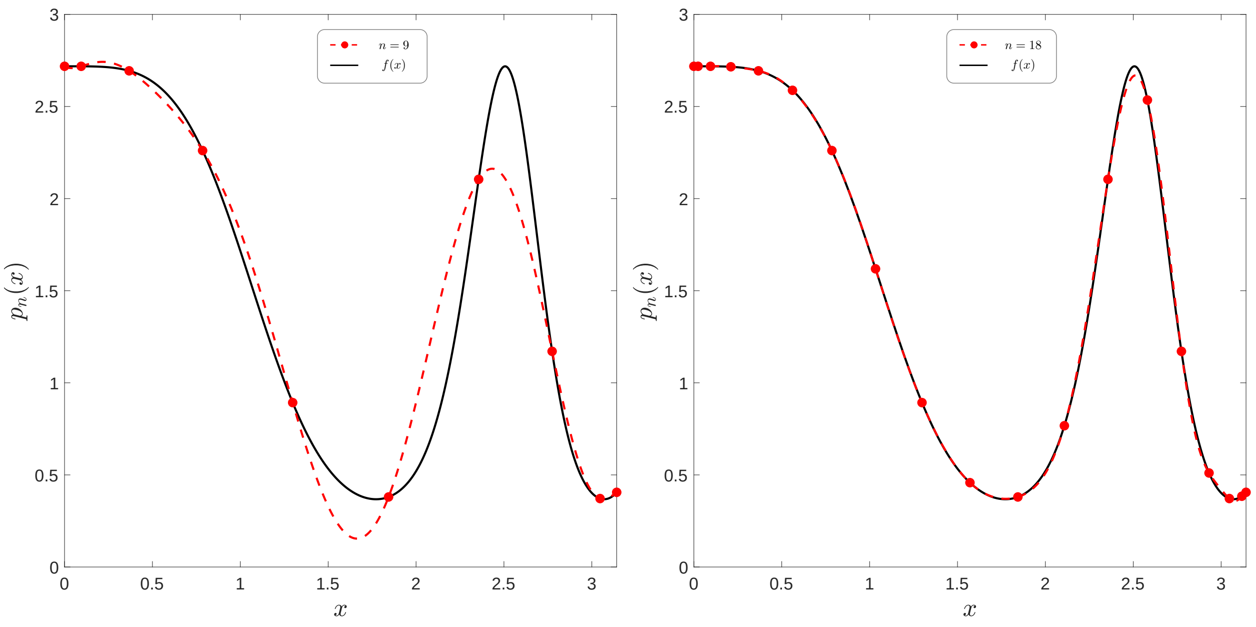
Definition 7.6 (Affine map). For a general interval of interest $[a, b]$ it is possible to rewrite any value $x \in [a, b]$ with the *affine map*

$$x = \frac{b-a}{2}\xi + \frac{b+a}{2},$$

such that x is in terms of a variable $\xi \in [-1, 1]$. ◀

We close this discussion on interpolation with a final example using a high degree global interpolating polynomial defined on the Chebyshev nodes on an interval that is not $[-1, 1]$.

Example 7.4. We create a global interpolating polynomial in Lagrange form for the function $f(x) = e^{\cos(x^2)}$ on the interval $[0, \pi]$. The function is infinitely differentiable and its derivatives are bounded on the interval $[0, \pi]$, so we expect the global interpolant to perform well when the interpolation points are chosen to be the Chebyshev nodes. We plot the interpolant against the function $f(x)$ for a visual comparison with global polynomial degrees of $n = 9$ (at the left) and $n = 18$ (at the right). We see that doubling the points improves the approximation considerably.



♦

7.2 Discrete differentiation

We now have the ability to approximate a function $f(x)$ with a polynomial as long as we are given some discrete set of data points $\{x_i, f(x_i)\}_{i=0}^n$. But often problems we are interested in involve not only function data, but derivatives as well. This is especially true to numerically approximate the solution of ordinary differential equations as well as partial differential equations. In essence, we approximate the derivative(s) to reduce a differential equation into a form that we can more easily solve with tools we learned from the linear algebra portion of the course.

The numerical approximation of derivatives will look very familiar because we will motivate them from Taylor's theorem and other background knowledge we have from Calculus. For example, recall from the definition of the derivative

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

To simplify the discussion of discrete derivatives we make the assumption that the data points $\{x_i\}_{i=0}^n$ are *uniformly spaced* such that the interval $[a,b]$ is divided into equal pieces each with length h:

$$h = \frac{b-a}{n}, \qquad x_i = a + ih, \qquad i = 0, \dots, n.$$

This motivates that we can approximate the derivative of a function with a *finite difference*

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i)}{h} = \frac{f(x_{i+1}) - f(x_i)}{h} = \frac{f_{i+1} - f_i}{h}, \qquad i = 0, \dots, n.$$

This is referred to as a *forward* difference because of how the data is accessed by the approximation. What kind of error did we just make? If we examine the Taylor expansion of the function about the point x_i we have

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(\xi),$$

for some value of $\xi \in (x_i, x_i + h)$. Rearranging this truncated Taylor expansion we find the error between the true derivative and the forward difference approximation to be

$$f'(x_i) - \left(\frac{f(x_i+h) - f(x_i)}{h}\right) = -\frac{h}{2}f''(x_i) - \frac{h^2}{6}f'''(\xi).$$

Remark 7.5. Note that we have implicitly assumed that the function $f(x)$ is smooth such that higher order derivative of the function exist and are bounded. ⋈

Definition 7.7 (Forward finite difference). A first order approximation to the derivative of a function $f(x)$ on the interval $[a,b]$ is given by the *forward difference*

$$f'(x_i) \approx \frac{f_{i+1} - f_i}{h}, \qquad i = 0, \dots, n.$$

where $h = \frac{b-a}{n}$. The error in the approximation is $\mathcal{O}(h)$ such that doubling the number of approximation points decreases the error by a factor of two. ◀

Remark 7.6. Analogously, we can approximate the derivative with a *backward finite difference*

$$f'(x_i) \approx \frac{f_i - f_{i-1}}{h}, \qquad i = 0, \dots, n.$$

where $h = \frac{b-a}{n}$. This is also a first order approximation of the derivative. ⋈

The choice of forward or backward finite differences can be motivated by the physics of the problem you want to solve. But more critically, we need to address what to do at the boundaries of the interval $[a,b]$.

Example 7.5. We want to apply the forward finite difference to approximate the derivative of $f(x)$ on the interval $[a,b]$. This is okay until we reach the right most point in the domain, then the formula becomes

$$f'(x_n) \approx \frac{f_{n+1} - f_n}{h}.$$

This requires the knowledge of $f_{n+1} = f(x_{n+1})$, which is information we do not have. Therefore, at the right most endpoint we switch to a backward difference to select information we ***do*** have

$$f'(x_n) \approx \frac{f_n - f_{n-1}}{h}.$$

Further, we can rewrite the approximation of the derivative into a matrix-vector structure with

$$\begin{pmatrix} f_0' \\ f_1' \\ \vdots \\ f_{n-1}' \\ f_n' \end{pmatrix} = \frac{1}{h} \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ & & & -1 & 1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \qquad \text{or} \qquad \mathbf{f}' = \mathbf{D}_x \mathbf{f}.$$

♦

Here the elements of $\mathbf{D}_x$ that are not shown are zeroes, a convention we use throughout this lecture. We call the matrix $\mathbf{D}_x$ a *differentiation matrix.* Each row of $\mathbf{D}_x$ gives the weights of the finite difference formula being used at one of the nodes. Thus, multiplication of the differentiation matrix from the left with the vector of function values yields the derivative approximation at all the nodes.

Remark 7.7. We are free to choose whatever finite difference formula we want in each row of the differentiation matrix $\mathbf{D}_x$. However, it makes sense to make the rows as similar as possible because it makes it easier to examine the overall error of the derivative approximation (among other things). ⋈

What about other finite difference approximations? Can we construct an approximation that is more accurate? We can, and it basically all boils down to playing an algebra game with different Taylor expansions. Consider the forward and backward Taylor expansions for a function around the point x_i:

$$\begin{aligned} f(x_i + h) &= f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(x_i) + \frac{h^4}{24}f^{(4)}(\xi), \\ f(x_i - h) &= f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) - \frac{h^3}{6}f'''(x_i) + \frac{h^4}{24}f^{(4)}(\xi). \end{aligned}$$

If we subtract these two expressions and manipulate we obtain a higher order approximation to the first derivative of the function $f(x)$:

$$f'(x_i) - \left(\frac{f(x_i + h) - f(x_i - h)}{2h} \right) = \frac{h^2}{3}f'''(x_i) + \mathcal{O}(h^5).$$

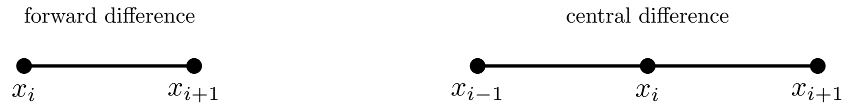
Definition 7.8 (Central difference approximation). A second order approximation to the first derivative of the function $f(x)$ on $[a,b]$ is given by

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i - h)}{2h}, \qquad i = 0, \ldots, n$$

where $h = \frac{b-a}{n}$. The error is on the order of $\mathcal{O}(h^2)$ so doubling the number of points decreases the error by a factor of four. ◀

Remark 7.8. Notice that in order to increase the approximation accuracy we require *more* neighboring information to the function at point x_i. In this instance, the central difference approximation needs information to the left and to the right of the point of interest. $\bowtie$

Including more function information to approximate the derivative at the point x_i is referred to as *widening the stencil* of the finite difference. Below we give two example stencils:



This is okay in the interior of the interval $[a, b]$ where this extra function information is available. However, the issue of approximating the derivative at the boundaries of the interval becomes relevant again. How do we give the finite difference approximation the necessary information to approximate the derivative when the stencil requires information we do not have? There are three options typically employed:

1. *Switch* to a lower order derivative approximation near the boundary that requires less function information.

2. *Bias* the stencil of the finite difference to keep the same order on the error but only take information available from within the interval.

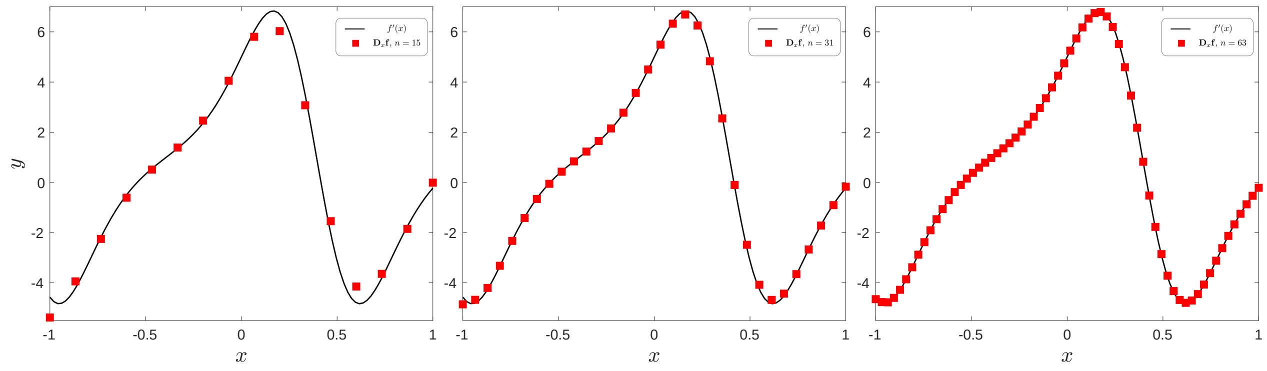
3. Introduce *ghost points* outside the interval that get assigned the unknown information by some means.

Remark 7.9. We mention ghost points for completeness but will not address how this is done in this course. Just know that manufacturing this unknown information outside the interval $[a, b]$ can be done in many ways, e.g., with extrapolation using a polynomial interpolant of the function $f(x)$. $\bowtie$

Example 7.6. Approximate the first derivative of the function $f(x) = x + e^{\sin(4x)}$ on the interval $[-1, 1]$ using a central difference in the interior and forward/backward differences on the boundary. First we assemble the derivative matrix we use for the approximation that takes the form

$$\begin{pmatrix} f_0' \\ f_1' \\ f_2' \\ \vdots \\ f_{n-1}' \\ f_n' \end{pmatrix} = \frac{1}{h}\begin{pmatrix} -1 & 1 & & & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & & & \\ & -\frac{1}{2} & 0 & \frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & & -1 & 1 \end{pmatrix}\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix}.$$

Notice, we have the total derivative approximation with the forward difference on the left boundary, the backward difference on the right boundary, and then a repeating stencil in the interior with the central difference. We plot the values of the derivative approximation below against the known derivative of the function $f'(x) = 1 + 4\cos(4x)e^{\sin(4x)}$ for an increasing number of nodes with $n = 15$, $n = 31$, and $n = 63$.



These plots highlight that the finite difference approximation only "knows" an approximation to the derivative at these discrete data points. The finite difference gives no information about the derivative of the function between the points. ♦

However, we can expect that switching to a lower order finite difference approximation at the boundaries will affect the overall error (or quality) of the approximate derivative. Instead, we seek to bias the finite difference approximation near the endpoints of the interval to maintain the same order of accuracy. However, we know that to do this we will need more function information. A rule of thumb is that if we want an m^{th} order finite difference approximation to the first derivative we need at least $m+1$ nodes of information. For example, a first order approximation required two nodes and we expect that a second order approximation needs three nodes.

To create a biased finite difference for the first derivative we will use the *method of undetermined coefficients* which is best demonstrated through example.

Example 7.7. Create a second order forward biased finite difference for the first derivative of $f(x)$. We expect this finite difference approximation to require three points of information to obtain the desired error of second order. Therefore, this biased finite difference approximation will take the general form

$$f'(x_i) \approx \frac{\gamma f_i + \eta f_{i+1} + \rho f_{i+2}}{h},$$

for the unknown real number coefficients γ, η, and ρ. We use Taylor expansions of the second two terms to find

$$f_{i+1} = f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(x_i) + \frac{h^4}{24}f^{(4)}(\xi_1),$$
$$f_{i+2} = f(x_i + 2h) = f(x_i) + 2hf'(x_i) + \frac{(2h)^2}{2}f''(x_i) + \frac{(2h)^3}{6}f'''(x_i) + \frac{(2h)^4}{24}f^{(4)}(\xi_2),$$

for some values of $\xi_1 \in (x_i, x_i + h)$ and $\xi_2 \in (x_i, x_i + 2h)$. Now we gather like terms to find

$$\frac{\gamma f_i + \eta f_{i+1} + \rho f_{i+2}}{h} = \frac{\gamma + \eta + \rho}{h} f_i + (\eta + 2\rho) f'(x_i) + \frac{h}{2}(\eta + 4\rho) f''(x_i) + \mathcal{O}(h^2).$$

For consistency to the first derivative we require the first coefficient on the right hand side to be zero and the second coefficient to be one. To maintain second order accuracy we require the third coefficient on the right hand side to vanish. This produces three equations for the three unknown coefficients:

$$\gamma + \eta + \rho = 0, \qquad \eta + 2\rho = 1, \qquad \eta + 4\rho = 0.$$

It is straightforward to determine the coefficients to be

$$\gamma = -\frac{3}{2}, \qquad \eta = 2, \qquad \rho = -\frac{1}{2},$$

which gives the second order forward biased difference approximation

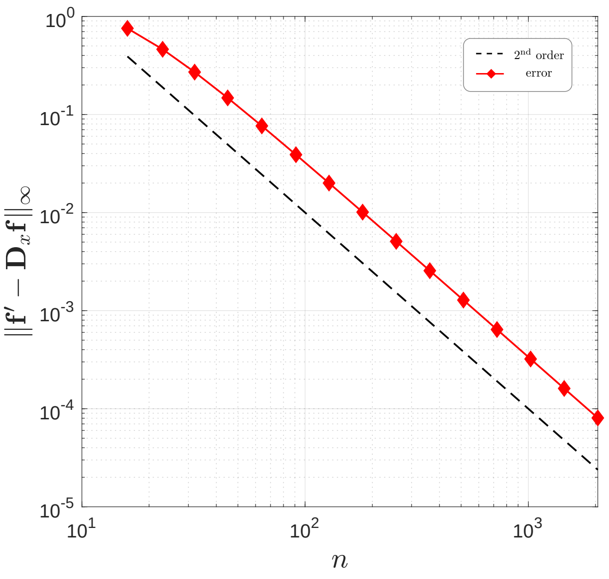
$$f'(x_i) = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2h} + \mathcal{O}(h^2).$$

♦

Remark 7.10. A biased backward difference formula that is second order accurate can be found in a similar way using undetermined coefficients to be

$$f'(x_i) = \frac{f_{i-2} - 4f_{i-1} + 3f_i}{2h} + \mathcal{O}(h^2).$$

We next examine the error behavior of the finite difference approximation.



EXAMPLE 7.8. We revisit the approximation of the derivative $f(x) = x + e^{\sin(4x)}$ this time using the second order biased finite difference at the boundary and central difference in the interior. The derivative matrix now has the structure

$$\begin{pmatrix} f'_0 \\ f'_1 \\ f'_2 \\ \vdots \\ f'_{n-1} \\ f'_n \end{pmatrix} = \frac{1}{h}\begin{pmatrix} -\frac{3}{2} & 2 & -\frac{1}{2} & & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & & & \\ & -\frac{1}{2} & 0 & \frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & \frac{1}{2} & -2 & \frac{3}{2} \end{pmatrix}\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix}$$

We plot the ∞-norm of the errors at the nodal values of the derivative approximation for increasing values of n. Further, the error plot for the finite difference approximation is presented in a log-log scale at the right. The slope of the line in the error plot is two compared to the dashed reference line, as we expect for a second order approximation. We reiterate that examining errors in the logarithmic scale is often more meaningful to reinforce the mathematical theory of an approximation. ♦

What if the equation we want to solve contains second derivatives? We need a finite difference approximation for those as well. Just as before we examine Taylor expansions and create the finite difference approximation to a given derivative. Recall the expansions

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(x_i) + \frac{h^4}{24}f^{(4)}(x_i) + \mathcal{O}(h^5),$$
$$f(x_i - h) = f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) - \frac{h^3}{6}f'''(x_i) + \frac{h^4}{24}f^{(4)}(x_i) + \mathcal{O}(h^5).$$

If we add these two terms we see that the odd derivatives in the expansion cancel and the even derivatives remain to have

$$f(x_i + h) + f(x_i - h) = 2f(x_i) + h^2 f''(x_i) + \frac{h^4}{12}f^{(4)}(x_i) + \mathcal{O}(h^6).$$

Manipulating this expression we find

$$f''(x_i) = \frac{f(x_i + h) - 2f(x_i) + f(x_i - h)}{h^2} - \frac{h^2}{12}f^{(4)}(x_i) + \mathcal{O}(h^6).$$

DEFINITION 7.9 (CENTRAL DIFFERENCE FOR SECOND DERIVATIVE). A second order approximation to the second derivative of the function $f(x)$ on the interval $[a,b]$ is given by a *central difference*

$$f''(x_i) \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2},$$

where $h = \frac{b-a}{n}$. The approximation is second order accurate so the error is on the order of $\mathcal{O}(h^2)$. ◀

Remark 7.11. We see that the stencil widens as well for the finite difference to approximate a derivative of higher degree. ⋈

Remark 7.12. It is also possible to bias the second order finite difference approximation as either forward or backward to avoid reaching unavailable data. For example, the forward biased finite difference of the second derivative that remains second order accurate in the error, i.e. $\mathcal{O}(h^2)$, is

$$f''(x_i) = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{h^2} + \mathcal{O}(h^2)$$

Note that the stencil widens further in the biased approximation in order to have enough degrees of freedom to bias the approximation of the second degree derivative at second order error. $\bowtie$

In general, a second order accurate finite difference approximation to the second derivative is given by

$$\begin{pmatrix} f_0'' \\ f_1'' \\ f_2'' \\ \vdots \\ f_{n-1}'' \\ f_n'' \end{pmatrix} = \frac{1}{h^2}\begin{pmatrix} 2 & -5 & 4 & -1 & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 \\ & & -1 & 4 & -5 & 2 \end{pmatrix}\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \qquad \text{or} \qquad \mathbf{f}'' = \mathbf{D}_{xx}\mathbf{f}.$$

Note, again, that the finite difference derivative matrix has the biased approximation on the left and right endpoints and a repeating central stencil in the interior of the interval.

Remark 7.13. Note, that we could construct a second derivative approximation matrix by multiplying two first order derivative matrices such that

$$\mathbf{D}_{xx} = \mathbf{D}_x\mathbf{D}_x.$$

This is completely valid, but would further widen the stencils in the matrix $\mathbf{D}_{xx}$. In practice we want to keep the stencils as narrow as possible while maintaining accuracy. $\bowtie$

To close this discussion we examine how to *assemble* such a derivative matrix in practice. Here, there will be a slight change such that we do not have to bias the derivative approximation because we *know* a certain amount of boundary information.

Example 7.9. Suppose we want to numerically approximate the solution to the two-point boundary value problem

$$u''(x) + 4u(x) = 0, \qquad u(0) = -2, \quad u\left(\frac{\pi}{4}\right) = 1.$$

We will convert this linear second order ordinary differential equation (ODE) into a linear system. That is, something we know how to solve numerically. To make the indexing a bit easier (and match the indexing from the linear algebra portion of the course) say we want to solve the problem on $n+1$ subintervals which means that the uniformly spaced points are given by

$$h = \frac{b-a}{n+1}, \qquad x_i = a + ih, \qquad i = 0, \dots, n+1,$$

where the interval $[a,b] = \left[0, \frac{\pi}{4}\right]$. The second order derivative can be approximated on the interior points x_i where $i = 1, \dots, n$ in a matrix-form which gives a discrete version of the second order ODE:

$$\frac{1}{h^2}\begin{pmatrix} 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & 1 & -2 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & 1 \end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_n \\ u_{n+1} \end{pmatrix} + 4\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

where $\mathbf{u} = (u_0, u_1, \ldots, u_n, u_{n+1})^T$ are the unknown quantities which we want to determine. This matrix form has n rows and $n+2$ columns. However, we already know the values

$$u_0 = -2, \qquad \text{and} \qquad u_{n+1} = 1,$$

because of the boundary conditions given by the problem formulation. Applying the boundary conditions shrinks the system by two unknown values to become just in terms of the unknown interior values. We manipulate the equations and multiply through by $-h^2$ to find

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} - 4h^2 \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} 2-4h^2 & -1 & & & \\ -1 & 2-4h^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2-4h^2 & -1 \\ & & & -1 & 2-4h^2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \\ \vdots \\ 0 \\ u_{n+1} \end{pmatrix}.$$

This is now a linear system of the form $\mathbf{Ax} = \mathbf{b}$. Further, we have seen a tridiagonal matrix like this before and know that it is symmetric positive definite (provided h is small enough). So we have a multitude of options to solve this system with either the Cholesky factorization, Gauss-Seidel, or conjugate gradient. ♦