

1

Introduction to Finite Difference Methods

1.1 A Few Historical Notes

Finite difference (FD)-type discrete approximations can be traced back much earlier than when Gottfried Leibniz¹ and Isaac Newton² gave the first descriptions of calculus (in 1684 and 1687, respectively). The introduction of FDs (at first for interpolation) is often attributed to Jost Bürgi,³ around 1592. Finite difference formulas of high orders of accuracy (especially for integration) were used by James Gregory in 1670.⁴ Significant further early FD perspectives were provided by Isaac Newton and later by Brook Taylor⁵ in 1715. Finite difference formulas were quite widely used for solving ordinary differential equations (ODEs) in the nineteenth century (notably for problems in fluid mechanics and for planetary orbit calculations). The pioneering work on the use of FD for partial differential equations (PDEs) dates back to the study by Lewis F. Richardson (1911)⁶ (on potentially dangerous stresses in the first dam over the Nile in Aswan).

1.2 Finite Difference Formulas

The standard definition of the first derivative

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (1.1)$$

¹ 1646–1716, German mathematician, philosopher, scientist, and diplomat.

² 1643–1727, English mathematician, physicist, astronomer, alchemist, and theologian; widely recognized as one of the greatest scientists of all time.

³ 1552–1632, Swiss mathematician, also a maker of clocks and astronomical instruments.

⁴ Described further in Section 7.3.

⁵ 1685–1731, English mathematician, of Taylor series fame.

⁶ 1881–1953, English mathematician and pioneer of numerical weather forecasting.

is a simple example of an FD formula. Taylor expansion of (1.1) shows that

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2!} f''(x) + \frac{h^2}{3!} f'''(x) + \cdots = f'(x) + O(h^1), \quad (1.2)$$

that is, the approximation $f'(x) \approx \frac{f(x+h) - f(x)}{h}$ is accurate to *first order* (at the location x). The FD *weights* at the *nodes* x and $x+h$ can be abbreviated as $[-1 \ 1] / h$. We will soon find it convenient to sketch FD *stencils* graphically. In this case:

$$\begin{array}{ccc} \circ & & \leftarrow \text{entry for } f', \text{ weight } \{1\}, \\ \square & \square & \leftarrow \text{entries for } f, \text{ weights } \{-\frac{1}{h}, \frac{1}{h}\}, \\ \uparrow & \uparrow & \\ x & x+h & \leftarrow \text{spatial locations.} \end{array} \quad (1.3)$$

The circle indicates the location in x for a derivative entry, and the squares indicate x -locations for function values (vertical spacing is here of no significance). Each of these locations has a *weight* associated with it. While the simplicity of this approximation is convenient (it uses only two adjacent function values to produce a derivative approximation), its low order of accuracy (first order; exact only for polynomials up through degree one) makes it almost entirely useless for practical computing.

A significantly better approximation to the first derivative is provided by

$$f'(x) = \frac{-\frac{1}{2}f(x-h) + \frac{1}{2}f(x+h)}{h} + O(h^2). \quad (1.4)$$

Figure 1.1 illustrates the two FD approximations (1.2) and (1.4). Although both give the same approximation for the tangent slope in the limit $h \rightarrow 0$, it is clear also visually that the second-order approximation is the more accurate one for small but nonzero h .

Taylor expansions provide a helpful means of verifying the order of accuracy if an FD formula is proposed. However, there are numerous more convenient ways to generate such formulas, as described in Section 1.2.1. Since the value of x in (1.2) and (1.4) – and in general for FD formulas – has no influence on the weights, we simplify the notation in Section 1.2.1 by approximating derivatives at $x = 0$.

1.2.1 Some Algorithms for Generating FD Weights

In all but the last of the five methods described in Sections 1.2.1.1–1.2.1.5, the independent variable x can just as well be a complex variable z (with node points distributed in the complex plane rather than along the real axis).

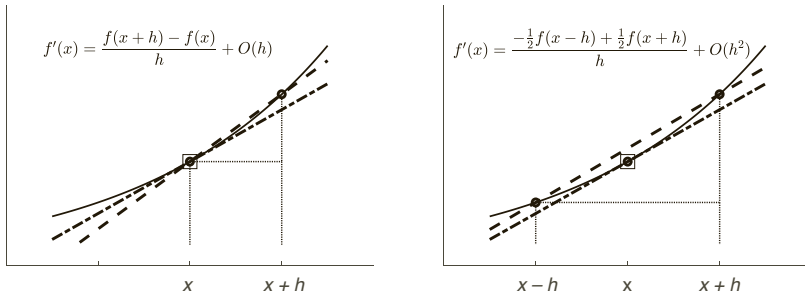


Figure 1.1 Graphical comparison between the first- and second-order approximations for $f'(x)$, as given by (1.2) and (1.4), respectively. The slope of the tangent lines (dash-dotted) corresponds to the true derivative value at the location x , and the slopes of the secant lines (dashed) correspond to the respective approximations.

1.2.1.1 FD Weights by Use of Monomial Test Functions

This algorithm is very flexible. Let L be any derivative operator (such as $\frac{d}{dx}$ and $\frac{d^2}{dx^2}$). We require a stencil with n nodes, located at $x_1, x_2, \dots, x_n$ (distinct) to be exact for the first n monomials $1, x, x^2, \dots, x^{n-1}$. This requirement can be written as

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} L 1|_{x=0} \\ L x|_{x=0} \\ \vdots \\ L x^{n-1}|_{x=0} \end{bmatrix}. \quad (1.5)$$

The weights w_k at nodes x_k , $k = 1, 2, \dots, n$, can now be obtained by solving this linear system. The matrix is of Vandermonde type – often ill-conditioned, but never singular assuming the x_k are distinct.⁷

Verification of nonsingularity: The result is true for $n = 1$, in which case the matrix is $[1]$. Applying induction over n , assume the result is true up through size $n - 1$, and consider the size n case. Call temporarily $x_n = x$ and the matrix $A(x)$. Expanding along the last column, $\det(A(x))$ becomes a polynomial in x of degree $n - 1$, not identically zero (as, by the induction hypothesis, the coefficient for x^{n-1} is nonzero). All its $n - 1$ roots are accounted for by $x = x_1, x = x_2, \dots, x = x_{n-1}$. It must therefore be nonzero for $x = x_n$.

Example 1.2.1 Create an FD formula that approximates $f''(0)$ based on function values $f(x_1)$ and $f(x_2)$ and first derivative values $f'(x_3)$ and $f'(x_4)$.

⁷ A different proof to the one immediately following is given in Section A.2.

Solution The monomial test function algorithm generalizes immediately to cases with mixed types of input data. Enforcing

$$f''(0) \approx w_1 f(x_1) + w_2 f(x_2) + w_3 f'(x_3) + w_4 f'(x_4)$$

to be exact for the functions $1, x, x^2, x^3$ gives the following linear system to solve for the weights:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ x_1 & x_2 & 1 & 1 \\ x_1^2 & x_2^2 & 2x_3 & 2x_4 \\ x_1^3 & x_2^3 & 3x_3^2 & 3x_4^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}.$$

The determinant of the coefficient matrix becomes

$$(x_1 - x_2)(x_3 - x_4)(2x_1^2 + 2x_1x_2 + 2x_2^2 - 3x_1x_3 - 3x_2x_3 - 3x_1x_4 - 3x_2x_4 + 6x_3x_4).$$

The first two factors tell that the system is singular in the case of coinciding nodes $x_1 = x_2$ or $x_3 = x_4$. The remaining quadratic form is not positive or negative definite,⁸ implying possibilities of additional singularities for certain node locations. $\square$

1.2.1.2 FD Weights by the Method of Exponential Test Functions

This method leads to the same linear system (1.5). However, it simplifies certain tasks (cf. Sections 7.5.1 and 8.3.1). We now apply L to the test function $e^{\xi x}$ rather than to monomials in x . Equating leading Taylor expansion terms in ξ in $\sum_{k=1}^n w_k e^{\xi x_k} = L e^{\xi x} \Big|_{x=0}$ becomes, when written out more explicitly,

$$\begin{Bmatrix} w_1 \left(1 + \xi x_1 + \frac{\xi^2 x_1^2}{2!} + \cdots \right) + \\ w_2 \left(1 + \xi x_2 + \frac{\xi^2 x_2^2}{2!} + \cdots \right) + \\ \vdots \\ w_n \left(1 + \xi x_n + \frac{\xi^2 x_n^2}{2!} + \cdots \right) \end{Bmatrix} = L \left(1 + \xi x + \frac{\xi^2 x^2}{2!} + \cdots \right) \Big|_{x=0},$$

again giving the linear system (1.5).

⁸ Seen, for example, by noting that it is linear in x_4 (and x_3). For any choice of x_1, x_2, x_3 , it evaluates to zero for $x_4 = \frac{2x_1^2 + 2x_1x_2 + 2x_2^2 - 3x_1x_3 - 3x_2x_3}{3(x_1 + x_2 - 2x_3)}$. More generally, writing it as

$$[x_1, x_2, x_3, x_4] \begin{bmatrix} 2 & 1 & -3/2 & -3/2 \\ 1 & 2 & -3/2 & -3/2 \\ -3/2 & -3/2 & 0 & 3 \\ -3/2 & -3/2 & 3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix},$$

it is seen from the fact that the (symmetric) matrix has both positive and negative eigenvalues ($\lambda_1 = 6, \lambda_2 = 1, \lambda_3 = 0, \lambda_4 = -3$). However, if $x_3 = x_1$ and $x_4 = x_2$ (then a case of Hermite interpolation, cf. Section 1.4.3), the quadratic form evaluates to $-(x_1 - x_2)^2$, ensuring nonsingularity for $x_1 \neq x_2$.

1.2.1.3 FD Weights by Lagrange's Interpolation Formula

This approach is good for theoretical insight but less convenient to code. Consider, for example, deriving (1.4). The *Lagrange interpolation polynomial*⁹ $p(x)$ that takes the values $f(-h)$, $f(0)$, $f(h)$ at $x = -h, 0, h$ is

$$p(x) = \frac{(x-0)(x-h)}{(-h-0)(-h-h)}f(-h) + \frac{(x+h)(x-h)}{(0+h)(0-h)}f(0) + \frac{(x+h)(x+0)}{(h+h)(h+0)}f(h).$$

A quick calculation shows that $p'(0) = \frac{-\frac{1}{2}f(x-h) + \frac{1}{2}f(x+h)}{h}$, in agreement with (1.4).

1.2.1.4 FD Weights by Recursion

This algorithm is computationally very fast and has excellent numerical stability. It follows from Lagrange's interpolation formula – see Fornberg (1988a) for details. We give it here only in the form of the MATLAB code shown in Algorithm 1.1. For example, the statement **weights(0, -2 : 2, 6)** returns the output

0	0	1.0000	0	0
0.0833	-0.6667	0	0.6667	-0.0833
-0.0833	1.3333	-2.5000	1.3333	-0.0833
-0.5000	1.0000	0	-1.0000	0.5000
1.0000	-4.0000	6.0000	-4.0000	1.0000
0	0	0	0	0
0	0	0	0	0

The top line gives the weights for interpolation, which here is trivial as the interpolation point coincides with one of the grid points. The next four rows give the optimal weights for five-node approximations to derivatives of orders 1 to 4.¹⁰ The last two rows are all zero, as derivatives of orders 5 and higher cannot be approximated based on just five nodes.

1.2.1.5 FD Weights by Padé-Based Algorithm

This algorithm was introduced in Fornberg (1998). Its derivation is a straightforward generalization of the special case described in Example 1.2.5. It requires equispaced nodes but is otherwise very general. It readily applies also to *implicit stencils*,¹¹ and it naturally uses exact rather than floating-point arithmetic. In the display form introduced in (1.3), we consider now stencils of the more general form

⁹ Described in Appendix A, with illustration in Figure 1.3.

¹⁰ Thus, lines 2 and 3 match the second row in Tables 1.1 and 1.2, respectively.

¹¹ Also described as *compact stencils*; see also Sections 3.3.2 and 4.3.1.1.

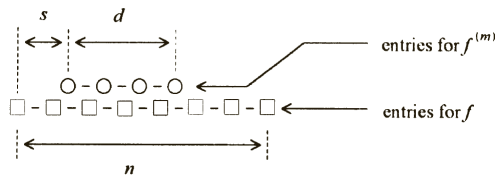
Algorithm 1.1 A MATLAB implementation of the weights algorithm in Section 1.2.1.4.

```

function c = weights(z,x,m)
% Calculates FD weights.
% Input parameters:
% z Location where approximations are to be accurate,
% x Row vector with x-coordinates for grid points (distinct but
%   otherwise arbitrarily located),
% m Highest derivative that we want to find weights for
% Output parameter:
% c Matrix, size (m+1,length(x)), containing in successive rows the
%   weights for derivatives 0,1,...,m.

n = length(x); c = zeros(m+2,n); c(2,1) = 1; x1 = repmat(x,n,1);
A = x1.'-x1;
b = cumprod([ones(n,1),A],2); rm = cumsum(ones(m+2,n-1))-1;
d = diag(b); d(1:n-1) = d(1:n-1)./d(2:n);
for i = 2:n
    mn = min(i,m+1);
    c(2:mn+1,i) = d(i-1)*(rm(1:mn,1).*c(1:mn,i-1)-(x(i-1)-z)* ...
        c(2:mn+1,i-1));
    c(2:mn+1,1:i-1) = ((x(i)-z)*c(2:mn+1,1:i-1)-rm(1:mn,1:i-1).* ...
        c(1:mn,1:i-1))./(x(i)- x1(1:mn,1:i-1));
end
c(1,:) = [];
end

```



relating weights for the m th derivative at $d + 1$ locations spaced h apart with weights for the function at $n + 1$, also h -spaced locations, with the two sets shifted sideways by s units of h , where s need not be an integer. In symbolic languages, such as Mathematica here, just two lines of code suffice¹²:

```

t = PadéApproximant[xs(Log[x]/h)m,{x,1,{n,d}}];
CoefficientList[{Denominator[t],Numerator[t]},x]

```

The following examples illustrate how this algorithm can be used and (Example 1.2.5) also the approach for deriving it.

¹² The present usage of Padé expansions differs significantly from their more common applications to convergence acceleration and numerical analytic continuation of both Taylor and asymptotic expansions; see Section E.3 and, for example, Bender and Orszag (1978), Sections 8.3–8.6; Fornberg and Piret (2020), Section 3.2.9; and Trefethen (2013), Chapter 27.

Example 1.2.2 Find the weights in a stencil of the shape $\square \overset{\circ}{\square} \square$ for approximating the second derivative.

Solution For $s = 1$, $d = 0$, $n = 2$, $m = 2$, the algorithm returns

$$\{\{h^2\}, \{1, -2, 1\}\},$$

corresponding to the explicit second-order accurate formula for the second derivative

$$f''(x) \approx \{f(x-h) - 2f(x) + f(x+h)\} \frac{1}{h^2}. \quad (1.6)$$

□

Example 1.2.3 Find the weights in a stencil of the shape $\square \overset{\circ}{\square} \overset{\circ}{\square} \overset{\circ}{\square}$, again for approximating the second derivative.

Solution For $s = 0$, $d = 2$, $n = 2$, $m = 2$, the algorithm returns

$$\left\{ \left\{ \frac{h^2}{12}, \frac{5h^2}{6}, \frac{h^2}{12} \right\}, \{1, -2, 1\} \right\},$$

corresponding to the compact (implicit) fourth-order accurate formula for the second derivative

$$\begin{aligned} \frac{1}{12}f''(x-h) + \frac{5}{6}f''(x) + \frac{1}{12}f''(x+h) \\ \approx \{f(x-h) - 2f(x) + f(x+h)\} \frac{1}{h^2}. \end{aligned} \quad (1.7)$$

□

Example 1.2.4 Find the weights in a stencil of the shape $\overset{\circ}{\square} \overset{\circ}{\square} \overset{\circ}{\square} \square$ for approximating the first derivative.

Solution For $s = -2$, $d = 2$, $n = 1$, $m = 1$, the output

$$\left\{ \left\{ \frac{5h}{12}, -\frac{4h}{3}, \frac{23h}{12} \right\}, \{-1, 1\} \right\}$$

is readily rearranged into

$$f(x+h) = f(x) + \frac{h}{12}(23f'(x) - 16f'(x-h) + 5f'(x-2h)). \quad (1.8)$$

We later (in Section 3.2.3) encounter this formula as the third-order Adams–Bashforth method for solving ODEs. □

Example 1.2.5 For Example 1.2.3, obtain the weights by explicitly carrying out the exponential test function approach.

Algorithm 1.2 A MATLAB implementation of the weights algorithm in Section 1.2.1.5.

```
function [w_f,w_der] = weights_Pade(s,d,n,m)
% Input parameters (type double or symbolic); d,n,m must be integers;
% If s is non-integer, symbolic form for s ensures exact arithmetic.
% s,d,n Define stencil shape
% m Order of derivative approximated
% Output parameters (symbolic variables)
% w_f Weight(s) for function values
% w_der Weight(s) for derivative values
syms x h; [N,D] = numden(pade(x^s*log(x)^m,x,1,'Order',[n,d]));
w_f = fliplr(coeffs(N,'All')); w_der = fliplr(coeffs(D,'All'))*h^m;
end
```

Solution The approximation should be of the form

$$b_{-1}f''(x-h) + b_0f''(x) + b_1f''(x+h) \approx c_{-1}f(x-h) + c_0f(x) + c_1f(x+h).$$

With $f = e^{\xi x}$, this becomes

$$\xi^2 \left(b_{-1}e^{-\xi h} + b_0 + b_1e^{\xi h} \right) e^{\xi x} \approx \left(c_{-1}e^{-\xi h} + c_0 + c_1e^{\xi h} \right) e^{\xi x}.$$

After canceling $e^{\xi x}$ and substituting $e^{\xi h} = s$ (i.e., $\xi = \frac{1}{h} \log s$), this can be written as

$$\left\{ \frac{1}{h} \log s \right\}^2 \approx \frac{c_{-1} + c_0s + c_1s^2}{b_{-1} + b_0s + b_1s^2}. \quad (1.9)$$

This relation should be as accurate as possible for $\xi \rightarrow 0$, that is, for $s \rightarrow 1$. The Padé expansion (cf. Section E.3) of $\left\{ \frac{1}{h} \log s \right\}^2$ around $s = 1$ with numerator and denominator degrees both equal to 2 becomes

$$\left\{ \frac{1}{h} \log s \right\}^2 \approx \frac{1}{h^2} \frac{(s-1)^2}{1 + (s-1) + \frac{1}{12}(s-1)^2} = \frac{12 - 24s + 12s^2}{h^2(1 + 10s + s^2)}.$$

Equating coefficients with (1.9) gives (1.7). □

The MATLAB code in Algorithm 1.2 implements this Padé-based algorithm using the Symbolic Toolbox. For example, to compute the weights in the second row in Table 4.1, the statement

$$[w_f, w_der] = \text{weights_Pade}(\text{sym}(3/2), \mathbf{0}, 3, 1)$$

produces the output $w_f = [1, -27, 27, -1]$, $w_der = 24 * h$.

Table 1.1 *Weights for centered FD approximations of the first derivative on an equispaced grid (omitting the factor $1/h$).*

Order	Weights				First derivative								
2					$-\frac{1}{2}$	0	$\frac{1}{2}$						
4				$\frac{1}{12}$	$-\frac{2}{3}$	0	$\frac{2}{3}$	$-\frac{1}{12}$					
6			$-\frac{1}{60}$	$\frac{3}{20}$	$-\frac{3}{4}$	0	$\frac{3}{4}$	$-\frac{3}{20}$	$\frac{1}{60}$				
8		$\frac{1}{280}$	$-\frac{4}{105}$	$\frac{1}{5}$	$-\frac{4}{5}$	0	$\frac{4}{5}$	$-\frac{1}{5}$	$\frac{4}{105}$	$-\frac{1}{280}$			
10	$-\frac{1}{1260}$	$\frac{5}{504}$	$-\frac{5}{84}$	$\frac{5}{21}$	$-\frac{5}{6}$	0	$\frac{5}{6}$	$-\frac{5}{21}$	$\frac{5}{84}$	$-\frac{5}{504}$	$\frac{1}{1260}$		
$\vdots$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\vdots$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$		
Limit	$\cdots$	$-\frac{1}{5}$	$\frac{1}{4}$	$-\frac{1}{3}$	$\frac{1}{2}$	-1	0	1	$-\frac{1}{2}$	$\frac{1}{3}$	$-\frac{1}{4}$	$\frac{1}{5}$	$\cdots$

1.2.2 Some Tables of FD Formulas

Especially with the recursion and Padé algorithms (in Sections 1.2.1.4 and 1.2.1.5, respectively), it is very easy to generate tables of FD weights. Four examples are given in Tables 1.1–1.4.¹³

We can make a number of observations from these relating to increasing orders of accuracy:

- i. For centered FD approximations, the weights converge to well-defined limits – derived later in Section 2.1 for an arbitrary-order derivative.
- ii. As seen for the first and second derivatives in Tables 1.1 and 1.2, and for a general-order derivative in (2.1), the weights for centered FD approximations decay in magnitude with distance k from the stencil center at the rate of $O(1/k)$ for odd-order derivatives, and $O(1/k^2)$ for those of even order. These slow decay rates are in some contexts problematic, since analytically, a derivative is a local property of a function and should not depend heavily on distant data.¹⁴
- iii. For one-sided approximations, the weights diverge rapidly with increasing orders of accuracy (as seen in Tables 1.3 and 1.4). Ways to understand and control this will be discussed in several contexts later in this book (including Sections 1.4, 5.3.7, and F.3).

¹³ A simple relation between the entries in Tables 1.1 and 1.2 is given in Section 6.3.3.

¹⁴ This comment does not apply to FD in the complex plane (cf. Chapter 6).

Table 1.4 *Weights for one-sided FD approximations of the second derivative on an equispaced grid (omitting the factor $1/h^2$).*

Order	Weights				Second derivative					
1	1	−2	1							
2	2	−5	4	−1						
3	$\frac{35}{12}$	$-\frac{26}{3}$	$\frac{19}{2}$	$-\frac{14}{3}$	$\frac{11}{12}$					
4	$\frac{15}{4}$	$-\frac{77}{6}$	$\frac{107}{6}$	−13	$\frac{61}{12}$	$-\frac{5}{6}$				
5	$\frac{203}{45}$	$-\frac{87}{5}$	$\frac{117}{4}$	$-\frac{254}{9}$	$\frac{33}{2}$	$-\frac{27}{5}$	$\frac{137}{180}$			
6	$\frac{469}{90}$	$-\frac{223}{10}$	$\frac{879}{20}$	$-\frac{949}{18}$	41	$-\frac{201}{10}$	$\frac{1009}{180}$	$-\frac{7}{10}$		
7	$\frac{29531}{5040}$	$-\frac{962}{35}$	$\frac{621}{10}$	$-\frac{4006}{45}$	$\frac{691}{8}$	$-\frac{282}{5}$	$\frac{2143}{90}$	$-\frac{206}{35}$	$\frac{363}{560}$	
8	$\frac{6515}{1008}$	$-\frac{4609}{140}$	$\frac{5869}{70}$	$-\frac{6289}{45}$	$\frac{6499}{40}$	$-\frac{265}{2}$	$\frac{6709}{90}$	$-\frac{967}{35}$	$\frac{3407}{560}$	$-\frac{761}{1260}$

1.3.1 Truncation Errors

These can be found theoretically by Taylor expanding the FD formula, as in (1.2). Numerically, one can evaluate the approximation and record the error. Then a log–log type plot showing $\log |\text{Error}|$ versus $\log h$ will feature (near-) straight lines with the slope p . These $O(h^p)$ -size errors become smaller when h decreases.

1.3.2 Rounding Errors

In standard double precision arithmetic, both weights and function values are typically uncertain by $O(10^{-16})$. For FD formulas for the k th derivative, having a h^k in their denominator, rounding errors can be roughly estimated as $O(10^{-16})/h^k$, that is growing rapidly when h decreases.

1.3.3 Total Errors

The total error becomes smallest when the two error types match in size, that is, when $O(h^p) \approx O(10^{-16})/h^k$, occurring for $h \approx 10^{-16/(p+k)}$, in which case $|\text{Error}| \approx 10^{-16(\frac{p}{p+k})}$. Figure 1.2 illustrates the schematic observations, by showing the error when $\frac{d}{dx} e^{-x/2}|_{x=0} = \frac{1}{2}$ is approximated using the first four FD stencils in Table 1.1 (denoted according to their accuracy orders as FD2, . . . , FD8). Apart from illustrating the two error sources, the figure also gives a hint about why we in this book focus on higher-order FD methods. When solving differential equations, approximations are needed at all grid

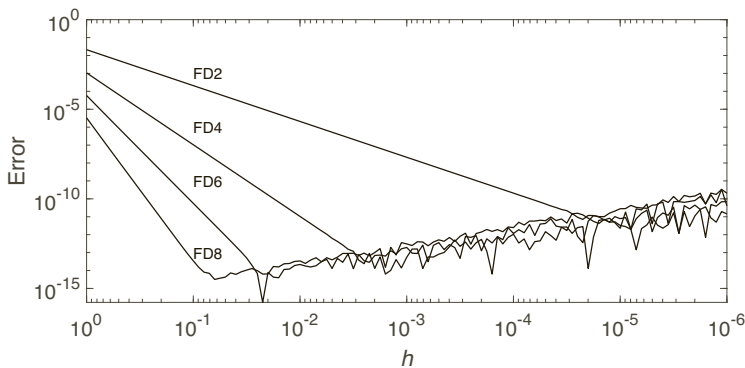


Figure 1.2 Errors when approximating $\left. \frac{d}{dx} e^{-x/2} \right|_{x=0} = \frac{1}{2}$ with different grid spacings h and using the first four FD stencils in Table 1.1. The respective orders of accuracy are confirmed by the slopes of the initial straight-line trends. The jagged trend for smaller h reflect $O(10^{-16})/h$ -sized rounding errors.

points across a domain. If we, for example, want to reach a 10^{-10} error level, the FD8 approximation (with four times as many weights as FD2) is seen to need only about 1/10,000 as many node points, typically making it vastly more cost-effective (already in 1-D; in 2-D, the factor would be 10^{-8} , etc.).

1.4 The Runge Phenomenon

High-degree polynomial interpolation with equispaced nodes is notorious for often being violently oscillatory near the end points of an interval. This plays a big role in the way that increasing order FD methods are designed.

1.4.1 Illustration of Lagrange Polynomials

Figure 1.3 illustrates the Lagrange polynomials $L_k(x)$ (see equation (A.1) in Appendix A) in the case of nodes located at $x_k = k$, $k = 0, 1, \dots, 8$. None of these polynomials feature strikingly large spurious oscillations near the interval center, but several do that near the interval ends. Combining these kernel functions together according to the function values $f(x_k)$ will produce a polynomial interpolant sharing the same property, that is, being relatively smooth near the interval center, but with a high likelihood of oscillations near its ends.

Figure 1.3 also gives insight into the FD weight tables. The weights given in the next to last row of Tables 1.3 and 1.4 are exactly the first and second

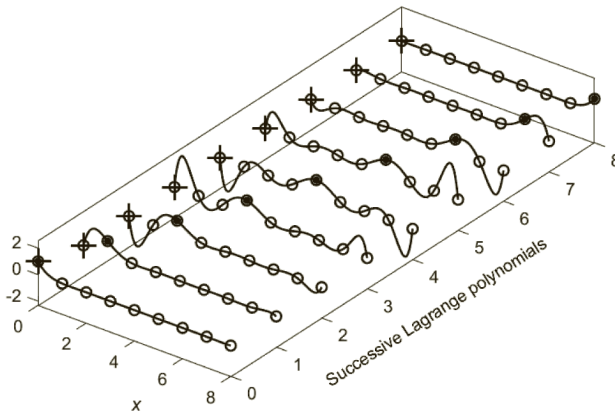


Figure 1.3 The Lagrange polynomials $L_k(x)$ in the case of $x_k = k$, $k = 0, 1, \dots, 8$. The nodes at which they are zero are marked with open circles and the node at which the value is one is shown with a filled circle.

derivative of these shown curves at their left end point $x = 0$ (marked here by crosses). For the centered FD approximations (the next to last rows in Tables 1.1 and 1.2; again of stencil width of 9 nodes), the weights are similarly the matching derivatives of these curves at their center location.

1.4.2 Illustration of the Runge Phenomenon

Figure 1.4 shows a commonly illustrated case of the Runge phenomenon – interpolating the function

$$f(x) = \frac{1}{1 + 16x^2} \quad (1.10)$$

at equispaced nodes over $x \in [-1, 1]$. Without here going into the extensive literature on Runge phenomenon analysis,¹⁵ let us just note some pertinent highlights that are relevant to the present goal of understanding its influence on FD (and PS) methods:

- i. Figure 1.4 again illustrates why centered FD approximations are much more accurate than one-sided ones, as in both cases these return the exact derivative of interpolating polynomials.
- ii. With N nodes, the *envelope* of the oscillations in the error (to leading order;

¹⁵ The pioneering work by C. Runge (1901) has been recalled in, for example, Fornberg (1996) and Trefethen (2000, 2013). One important starting point is the formula (A.7).

$x \in [-1, 1]$) is given by $\alpha(x)^N$, where $\alpha(x) = e^{\phi(x)-c}$, and

$$\phi(x) = \frac{1}{2} ((1-x) \log(1-x) + (1+x) \log(1+x)). \quad (1.11)$$

The function $f(x)$ that is interpolated enters only through the constant c , which in the present case of (1.10) becomes $c = \frac{1}{2} \log \frac{17}{16} + \frac{1}{4} \arctan 4 \approx 0.3618$.¹⁶ Figure 1.5 shows the function $\alpha(x)$ for the present $f(x)$. We can read off that the interpolant converges exponentially fast in a central region (at $x = 0$ as 0.6964^N) and diverges exponentially toward the edges, with the transition points at $x \approx \pm 0.7942$ (independent of N).

- iii. While interpolation generally is much more stable numerically than extrapolation (or approximations near to interval ends), the extreme severity of the Runge phenomenon is to a large extent due to the present use of polynomials as opposed to other types of interpolating functions.¹⁷

Although there is no simple universal remedy against the Runge phenomenon, various ways to reduce the damage it causes have been devised (such as the RBF-FD PHS+poly approach, described in Section 5.4.1 and, focusing on its properties at boundaries, in Section F.3).¹⁸

1.4.3 Hermite Interpolation

In certain applications, both $f(x_k)$ - and $f'(x_k)$ -values are available at all node points. Hermite interpolation can then be an alternative to Lagrange interpolation (again finding the unique lowest degree polynomial that fits the data). Finite difference weights to be simultaneously applied to both these sets can readily be obtained, for example, by generalizing the algorithms in Sections 1.2.1.1 and 1.2.1.4.¹⁹ However, with N nodes, these polynomials will be of degree $2N - 1$ (rather than $N - 1$), off-setting possible advantages with respect to the Runge phenomenon. In the case illustrated in Figure 1.4, the break point between convergence and divergence is unchanged, again at $x \approx \pm 0.7942$; see Figure 1.6.

¹⁶ Its value can be found by generalizing x to z complex and noting that convergence/divergence breakeven should occur at the most limiting singularity of $f(z)$; here at $z = \pm \frac{i}{4}$.

¹⁷ For example, in contrast to rational functions, all polynomials $p(x)$ of high orders must diverge rapidly as $|x|$ increases.

¹⁸ It is shown in Platte et al. (2011) that no stable algorithm can converge exponentially fast for all analytic functions if using only equispaced data on an interval within their region of analyticity. However, “root-exponential” convergence is possible (faster than any algebraic order).

¹⁹ For details and MATLAB code, see Fornberg (2021a).

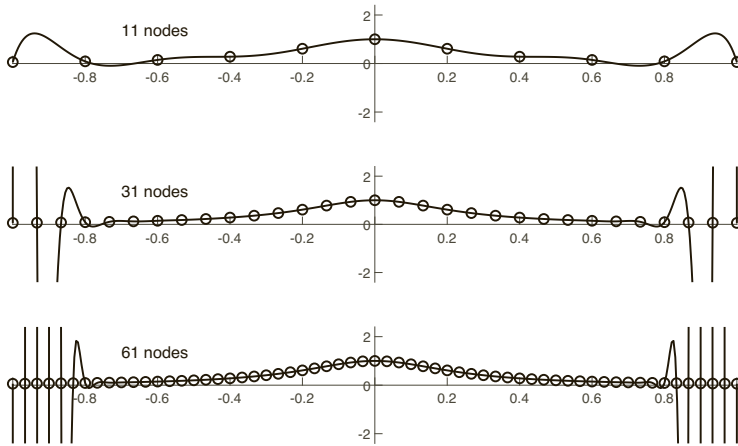


Figure 1.4 Equispaced polynomial interpolation of $f(x) = \frac{1}{1+16x^2}$ over $[-1, 1]$ using 11, 31, and 61 nodes.

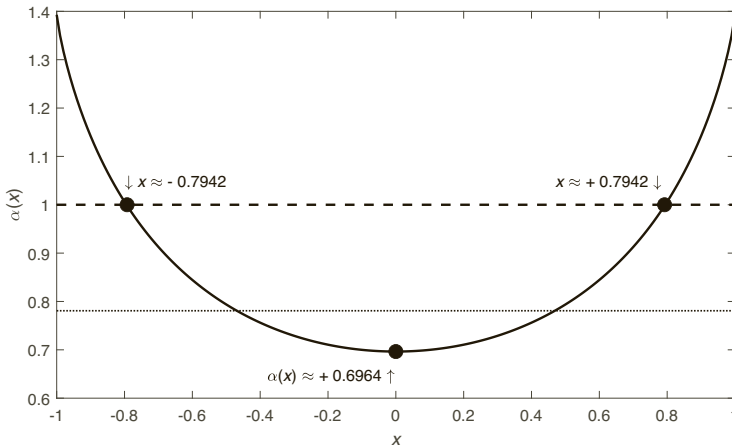


Figure 1.5 Solid curve: The function $\alpha(x)$ describing the Runge phenomenon oscillation envelope for equispaced interpolation over $[-1, 1]$ in the case of $f(x)$ given in (1.10). In particular, it always holds that $\alpha(\pm 1) - \alpha(0) = \log 2 \approx 0.6931$. Dotted straight line: The counterpart curve if the nodes are not equispaced but Chebyshev-distributed. For different functions $f(x)$, this curve and line get shifted up or down (with different amounts), but do not otherwise change shape.

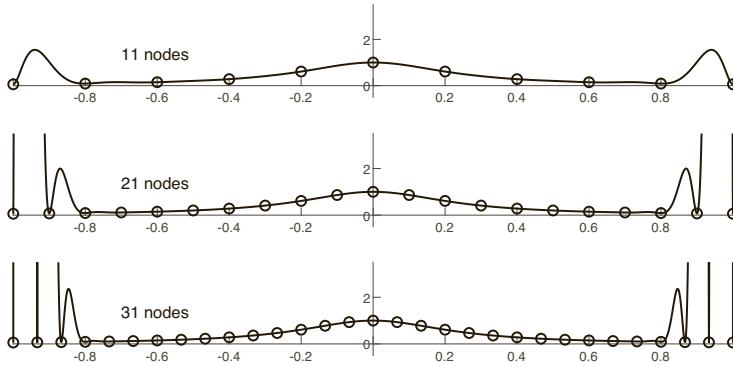


Figure 1.6 Equispaced Hermite interpolation of $f(x) = \frac{1}{1+16x^2}$ over $[-1, 1]$ using 11, 21, and 31 nodes.

1.4.4 Brief Comments on Chebyshev-Distributed Nodes

As seen in Tables 1.3 and 1.4, the Runge phenomenon is quite mild for small FD stencils, but then grows exponentially with the degree of the polynomials that the stencils are based on. Pseudospectral methods (Chapter 2) can be seen as pushing node numbers and polynomial degrees to extremely high values. As an alternative to an equispaced grid with nodes on $[-1, 1]$ located at

$$x_k = -1 + 2k/N, \quad k = 0, 1, \dots, N,$$

a commonly used Runge phenomenon remedy is then to cluster the nodes strongly toward the interval ends, as with the Chebyshev node distribution

$$x_k = -\cos(k\pi/N), \quad k = 0, 1, \dots, N.$$

If using approximations based on a single interpolating polynomial in the $N \rightarrow \infty$ limit, the dotted line in Figure 1.5 shows the counterpart $\alpha(x)$ curve, which in the case of (1.10) works out to become the constant $\alpha(x) \equiv \frac{1}{4}(\sqrt{17} - 1) \approx 0.7808$. The global polynomial interpolant now converges exponentially across $[-1, 1]$. Compared to equispaced nodes, improved accuracy near interval ends has been obtained in exchange for a loss of accuracy near the interval center.²⁰ With the focus in this book on FD approximations well short of the infinite order limit, the Runge phenomenon is often not a major issue.²¹

²⁰ The high accuracy seen near the interval center in Figure 1.5 explains the excellent accuracy of centered high-order FD approximations.

²¹ Node clustering at boundaries can still be beneficial for other purposes, such as for achieving locally increased resolution (e.g., for resolving boundary layers) and for representing irregular boundary shapes. The generalization from grid-based FD to mesh-free RBF-FD discretizations is described in Chapter 5.