Chapter 1

# Brief Summary of Finite Difference Methods

This chapter provides a brief summary of FD methods, with a special emphasis on the aspects that will become important in the subsequent chapters.

## 1.1 ▪ Finite difference formulas

Finite differences (FD) approximate derivatives by combining nearby function values using a set of *weights*. Several different algorithms for determining such weights are mentioned in Sections 1.1.1–1.1.5. In the very simplest case, illustrated in Figure 1.1, we use the mathematical definition of a derivative

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

to arrive at a two-node FD formula.

Taylor expansion of (1.1) shows that

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2!} f''(x) + \frac{h^2}{3!} f'''(x) + \ldots = f'(x) + O(h^1);$$

i.e., the approximation $f'(x) \approx \frac{f(x+h) - f(x)}{h}$ is accurate to *first order*. The FD *weights* at the *nodes* $x$ and $x+h$ are in this case $[-1 \;\; 1] \,/\, h$. The FD *stencil* can graphically be illustrated as

$$\begin{array}{cc} \circ & \leftarrow \text{ entry for } f', \text{ value } \{1\} \\ \blacksquare \quad \blacksquare & \leftarrow \text{ entries for } f, \text{ values } \{-\tfrac{1}{h}, \tfrac{1}{h}\} \\ \uparrow \quad \uparrow & \\ x \quad x+h & \leftarrow \text{ spatial locations.} \end{array} \tag{1.2}$$

The open circle indicates a (typically) unknown derivative value and the filled squares (typically) known function values. While the *compactness* of this approximation is convenient (it uses only two adjacent function values), its low order of accuracy (first order; exact only for polynomials up to degree one) makes it almost entirely useless for practical computing. Before considering the application of FD formulas to tasks such as approximating ODEs and PDEs (ordinary and partial differential equations), we consider next some different procedures for creating higher-order FD approximations.

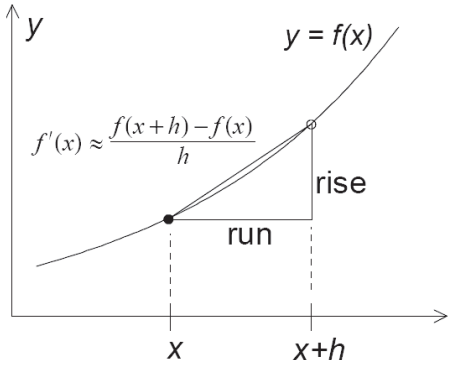



**Figure 1.1.** *Illustration of the approximation* $f'(x) \approx \frac{rise}{run} = \frac{f(x+h)-f(x)}{h}$; *this is increasingly accurate as* $h \to 0$.

## 1.1.1 ▪ Some direct approaches for generating FD stencils

The three approaches described next are flexible and conceptually quite straightforward but also rather inefficient in terms of their operation count. Even so, it should be noted that the linear systems approach (Section 1.1.1.3) will become prominent later in the RBF and RBF-FD contexts. For simplicity of notation, we do not describe the approaches in their most general form but choose the specific example of finding the weight vector $[-\frac{1}{2}\ 0\ \frac{1}{2}]/h$ in the second-order approximation to the first derivative

$$f'(x) \approx \frac{-\frac{1}{2}f(x-h)+\frac{1}{2}f(x+h)}{h}. \tag{1.3}$$

### 1.1.1.1 ▪ Derivative of Lagrange's interpolation polynomial

The value for $x$ does not influence the weights in a formula such as (1.3), so we can assume that the stencil is centered at $x = 0$. The *Lagrange interpolation polynomial* $p(x)$, taking the desired values at the nodes $x = -h, 0, h$, becomes

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

Differentiating this polynomial with respect to $x$ and then setting $x = 0$ gives $p'(0) = (-\frac{1}{2}f(-h) + 0\,f(0) + \frac{1}{2}f(+h))/h$, in agreement with (1.3).

### 1.1.1.2 ▪ Taylor expansions

Expressing $f(-h)$, $f(0)$, $f(h)$ by Taylor expansion around $x = 0$ gives

$$\begin{cases} f(-h) &= f(0) - \frac{h}{1!}f'(0) + \frac{h^2}{2!}f''(0) - + \ldots, \\ f(0) &= f(0), \\ f(h) &= f(0) + \frac{h}{1!}f'(0) + \frac{h^2}{2!}f''(0) \ + \ldots. \end{cases} \tag{1.4}$$

We want to find weights $w_{-1}$, $w_0$, $w_1$ such that

$$w_{-1}f(-h) + w_0 f(0) + w_1 f(h) = 0\,f(0) + 1\,f'(0) + 0\,f''(0) + \ldots.$$

Using the expansions from (1.4) and equating coefficients for $f(0)$, $f'(0)$, $f''(0)$ gives rise to a linear system to solve for the unknown coefficients

$$\begin{bmatrix} 1 & 1 & 1 \\ -\frac{h}{1!} & 0 & \frac{h}{1!} \\ \frac{h^2}{2!} & 0 & \frac{h^2}{2!} \end{bmatrix} \begin{bmatrix} w_{-1} \\ w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \tag{1.5}$$

with the solution $w_{-1} = -\frac{1}{2h}$, $w_0 = 0$, $w_1 = \frac{1}{2h}$.

#### 1.1.1.3 ▪ Use of monomial test functions

Continuing with the same example, we want the formula $f'(0) \approx w_{-1}f(-h) + w_0 f(0) + w_1 f(h)$ to be exact for as high degree polynomials as possible. Enforcing it in turn for the monomials $f = 1$, $f = x$ and $f = x^2$ gives

$$\begin{array}{lcllll} f = 1 & \Rightarrow & w_{-1} & +w_0 & +w_1 & = 0, \\ f = x & \Rightarrow & w_{-1}(-h) & & +w_1(h) & = 1, \\ f = x^2 & \Rightarrow & w_{-1}(-h)^2 & & +w_1(h)^2 & = 0, \end{array}$$

equivalent to (1.5).

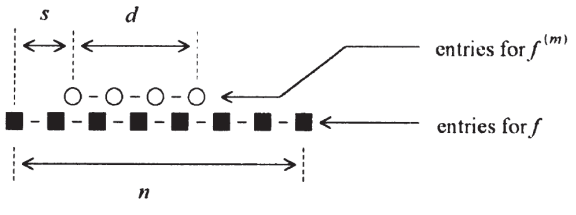
In the more general case of finding the weights $w_1, w_2, \ldots, w_n$ to use at locations $x_1, x_2, \ldots, x_n$ for approximating a linear operator $L$ at some location $x = x_c$, we similarly solve the system

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} L\,1|_{x=x_c} \\ L\,x|_{x=x_c} \\ \vdots \\ L\,x^{n-1}|_{x=x_c} \end{bmatrix}. \tag{1.6}$$

The successive lines of this system enforce that the set of weights lead to the correct result for the functions $1, x, x^2, \ldots, x^{n-1}$ and thus, by linearity, for all polynomials up through degree $n-1$. This direct linear systems approach is very flexible and easy to implement. However, it is not computationally fast ($O(n^3)$ operations), and the coefficient matrix can become ill-conditioned. It will, however, become the primary approach in the context of RBF methods.

### 1.1.2 ▪ Padé-based algorithm for equispaced grids

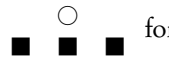
When the nodes have a uniform spacing $h$ (as has been the case in the examples above), a particularly short symbolic algebra algorithm was presented in 1998 [91]. We generalize the stencil (1.2) to



Here, the numbers $s$, $d$, and $n$ describe the stencil shape. In the illustration above, these take the values 3/2, 3, and 7, respectively. The weights, one at each node point, relate nodal values of the $m$th derivative of $f$ with the nodal values of the function $f$. In Mathematica (version 7 and higher), the complete code is

```
t = PadeApproximant[x^s(Log[x]/h)^m,{x,1,{n,d}}];
CoefficientList[{Denominator[t],Numerator[t]},x]
```

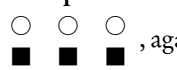
with similar codes in other symbolic languages. The following are three typical applications of this algorithm:

**Example 1.** The choices $s = 1$, $d = 0$, $n = 2$, $m = 2$ describe a stencil of the shape  for approximating the second derivative (since $m = 2$). The algorithm produces the output

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

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

$$f''(x) \approx \{f(x-h)-2f(x)+f(x+h)\}\,\frac{1}{h^2}\,. \tag{1.7}$$

**Example 2.** The choices $s = 0$, $d = 2$, $n = 2$, $m = 2$ describe a stencil of the shape  , again for approximating the second derivative (since $m = 2$). The algorithm produces the output

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

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

$$\frac{1}{12}f''(x-h)+\frac{5}{6}f''(x)+\frac{1}{12}f''(x+h) \approx \{f(x-h)-2f(x)+f(x+h)\}\,\frac{1}{h^2}\,. \tag{1.8}$$

**Example 3.** The choices $s = -2$, $d = 2$, $n = 1$, $m = 1$ describe a stencil of the shape  for approximating the first derivative. The output

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

is readily rearranged into

$$f(x+h) = f(x)+\frac{h}{12}(23f'(x)-16f'(x-h)+5f'(x-2h))\,, \tag{1.9}$$

which we later (in Section 1.2.1.2) will encounter as the third-order Adams-Bashforth method for solving ODEs.

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

| Order | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | $\frac{4}{5}$ | $-\frac{1}{5}$ | $\frac{4}{105}$ | $-\frac{1}{280}$ | |
| $\vdots$ | | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\vdots$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | |
| Limit | $\cdots$ | $\frac{1}{4}$ | $-\frac{1}{3}$ | $\frac{1}{2}$ | $-1$ | 0 | 1 | $-\frac{1}{2}$ | $\frac{1}{3}$ | $-\frac{1}{4}$ | $\cdots$ |

**Table 1.2.** *Weights for centered FD approximations of the second derivative on an equispaced grid (omitting the factor $1/h^2$).*

| Order | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | | | | 1 | $-2$ | 1 | | | | |
| 4 | | | | $-\frac{1}{12}$ | $\frac{4}{3}$ | $-\frac{5}{2}$ | $\frac{4}{3}$ | $-\frac{1}{12}$ | | | |
| 6 | | | $\frac{1}{90}$ | $-\frac{3}{20}$ | $\frac{3}{2}$ | $-\frac{49}{18}$ | $\frac{3}{2}$ | $-\frac{3}{20}$ | $\frac{1}{90}$ | | |
| 8 | | $-\frac{1}{560}$ | $\frac{8}{315}$ | $-\frac{1}{5}$ | $\frac{8}{5}$ | $-\frac{205}{72}$ | $\frac{8}{5}$ | $-\frac{1}{5}$ | $\frac{8}{315}$ | $-\frac{1}{560}$ | |
| $\vdots$ | | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | $\downarrow$ | |
| Limit | $\cdots$ | $-\frac{2}{4^2}$ | $\frac{2}{3^2}$ | $-\frac{2}{2^2}$ | $\frac{2}{1^2}$ | $-\frac{\pi^2}{3}$ | $\frac{2}{1^2}$ | $-\frac{2}{2^2}$ | $\frac{2}{3^2}$ | $-\frac{2}{4^2}$ | $\cdots$ |

In every case, the weights will be the optimal ones with regard to formal order of accuracy. The algorithm is particularly convenient to use when only a small number of stencils are considered and when one wants to obtain the weights in exact rational form rather than as floating point numbers.

Table 1.1 shows the lowest-order centered FD formulas for the first derivative and Table 1.2 for the second derivative. The existence of infinite order limits (indicated by the bottom line in each of the two tables) will play a key role in Section 2.1.1 when we introduce PS methods.

The special case illustrated in Example 3 can be generalized to include all the main classes of linear multistep methods. With $m = 1$ and accuracy order $p \geq 1$, the appropriate settings for $s$, $d$, and $n$ become

| | | | |
|---|---|---|---|
| Adams-Bashforth (AB) | $s = 1 - p,$ | $d = p - 1,$ | $n = 1.$ |
| Adams-Moulton (AM) | $s = 2 - p,$ | $d = p - 1,$ | $n = 1.$ |
| Backward Differentiation (BD) | $s = p,$ | $d = 0,$ | $n = p.$ |

### 1.1.3 ▪ Algorithms for arbitrarily spaced grids

FD approximations based on equispaced grids are very accurate when they are centered (extending equally far to both sides) but tend to lose accuracy when boundaries are approached and they have to become increasingly one-sided. The common remedy is to gradually cluster nodes more densely as the boundary is approached, as will be discussed further in Section 2.2.1. A number of effective algorithms for calculating FD weights are available for such (nonequispaced) cases.

#### 1.1.3.1 ▪ FD approximations at select points

We consider here first the case when one merely wants a few stencils. In the case of nodes located at $x_i$, $i = 1,2,\ldots,n$, one can obtain the weights for $d^p/dx^p|_{x=z}$, $p = 0,1,\ldots,m$ (where the location $z$ may or may not coincide with any of the node locations) by means of

```
function c = weights(z,x,m)

% Calculates FD weights.  The parameters are:
%  z  location where approximations are to be accurate,
%  x  vector with x-coordinates for grid points,
%  m  highest derivative that we want to find weights for
%  c  array size (m+1,length(x)), containing (as output) in
%     successive rows the weights for derivatives 0,1,...,m.

n = length(x); c = zeros(m+2,n); c(2,1) = 1; x1 = x(ones(1,n),:);
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,:)  = [];
```

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
```

This output shows the optimal weights to be applied to function values at $x = -2$, $-1$, 0, 1, 2, for approximating the zeroth up through the sixth derivative at $x = 0$. Since the approximation point $z$ coincides with one of the data points, the top line tells the obvious fact that the most accurate "interpolation" is to just use that data value. We recognize lines 2 and 3 from the fourth-order approximations in Tables 1.1 and 1.2, respectively. The last two lines are all zero, reflecting the fact that there exist no formulas for the fifth and sixth derivatives that extend over only five node points.

The derivation of the algorithm, given in [86, 91], is based on recursions that follow from Lagrange's interpolation formula. If one also wants weights for shorter stencil widths (based on $x_i$, $i = 1,2,\ldots,\nu$; $\nu = 1,2,\ldots,n$), these can be picked up "for free," as otherwise discarded intermediate results after each step in the loop `for i = 2:n`. In that case, the algorithm costs only four arithmetic operations per calculated weight. If these shorter stencils are not wanted, the weights algorithm presented in [230] has a somewhat lower operation count and can be coded particularly efficiently in C++. However, that code is much longer, and it runs in MATLAB several times slower than the present algorithm.