Basic Numerical Calculus

Tamas Kis | tamas.a.kis@outlook.com | https://tamaskis.github.io

Contents

Copyright © 2021 Tamas Kis

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/ *or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.*

1 DISCRETE FUNCTIONS AND DATA

1.1 Discrete Functions

A **discrete function** is a function that only has defined values at discrete points in its domain. While a continuous function may be defined as $y = f(x)$, a discrete function is written as $\mathbf{y}$ vs. $\mathbf{x}$. Essentially, the vector $\mathbf{x}$ is a collection of x values, while the vector $\mathbf{y}$ is a collection of the corresponding y values. By convention, we say that there are $N+1$ points in $\mathbf{x}$ (see Section 1.2 for why this is the case).

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N+1} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N+1} \end{bmatrix}$$

1.2 Continuous Functions and Their Discretization

When analyzing a *continuous* function numerically, the first thing we do is discretize the interval we are analyzing. We refer to this discretized interval as the **computational domain**. Essentially, we consider a function $f(x)$ not as a continuous function, but rather as values corresponding to discrete locations, called **nodes**, in space. To discretize the domain, we first need to specify three quantities:

1. a: the left endpoint of the domain (i.e. the minimum value of x)
2. b: the right endpoint of the domain (i.e. the maximum value of x)
3. N: the number of subintervals (if we specify N subintervals, we will have $N+1$ nodes)

The length L of the domain is then

$$L = b - a$$

The discrete values of x (i.e. $x_1, ..., x_{N+1}$) are the nodes. Collectively, the set of nodes is referred to as the **mesh**. There are many different ways to create a mesh. In our case, we use a uniform mesh; this means that the nodes are equally spaced [2]. Thus, for a uniform mesh, the **grid spacing** is given by

$$\Delta x = \frac{L}{N}$$

The vector $\mathbf{x}$ storing the nodes can be defined as

$$\boxed{\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_N \\ x_{N+1} \end{bmatrix} = \begin{bmatrix} a \\ a+\Delta x \\ a+2\Delta x \\ \vdots \\ a+(i-1)\Delta x \\ \vdots \\ a+(N-1)\Delta x \\ a+N\Delta x \end{bmatrix}} \tag{1}$$

We can then also define a vector $\mathbf{y}$ to store all the y_i's, where y_i is the evaluation of $f(x)$ at $x = x_i$.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{N+1} \end{bmatrix} = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{N+1}) \end{bmatrix}$$

Thus, the vectors $\mathbf{y}$ and $\mathbf{x}$ esssentially form a discrete function (essentially a discretized representation of $y = f(x)$) as defining a discrete form of $f(x)$.

$$y = f(x) \quad \xrightarrow{\text{discretization}} \quad \mathbf{y} \text{ vs. } \mathbf{x}$$

$$\mathbf{f} \text{ vs. } \mathbf{x} \quad \equiv \quad \{(x_i, f(x_i))\}_{i=1}^{N+1}$$

An example of the discretization of a univariate function onto a uniform 1D mesh is shown in Fig. 1 below.

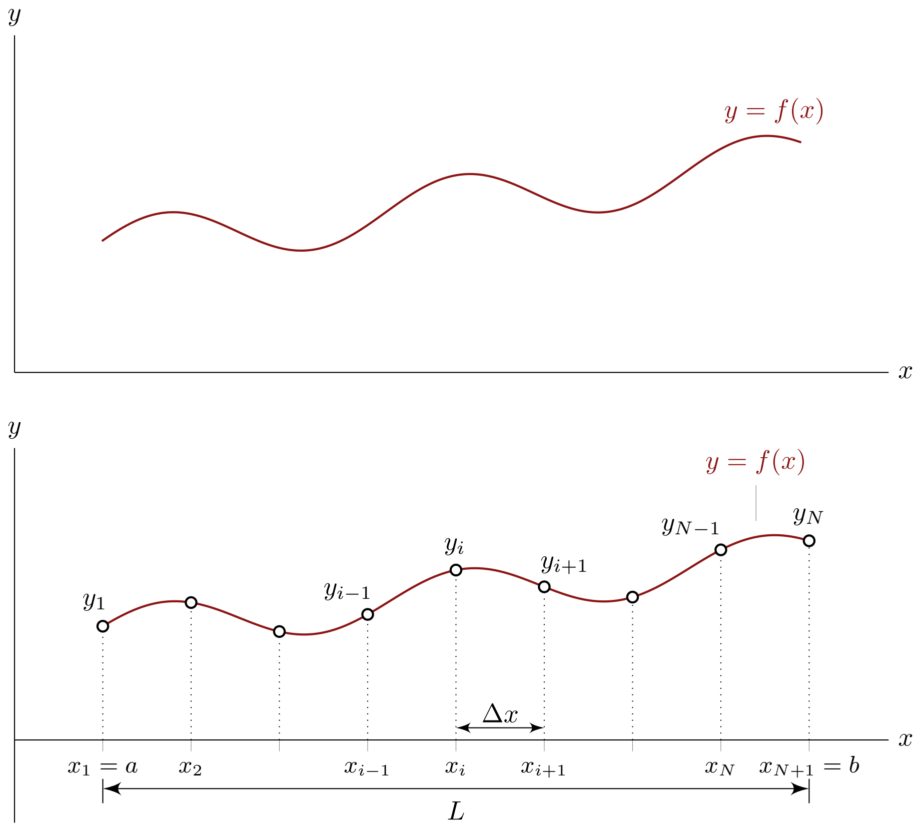

Figure 1: Discretization of a continuous function.

When discretizing a continuous function it is easiest to discretize it onto a uniform mesh. However, unevenly spaced nodes may be used as well. The numerical calculus techniques introduced in Sections 2 and 3 make no assumptions with regards to how the nodes are spaced.

1.3 Interpreting Data as a Discrete Function

When we sample data from the physical world, we built a data set

$$\{(x_i, y_i)\}_{i=1}^{N+1}$$

If we store all the x_i's in a vector $\mathbf{x}$, and all the y_i's in a vector $\mathbf{y}$, then we have the discrete function

$$\mathbf{y} \text{ vs. } \mathbf{x}$$

Thus, a data set is essentially a discrete function.

$$\mathbf{y} \text{ vs. } \mathbf{x} \quad \equiv \quad \{(x_i, y_i)\}_{i=1}^{N+1}$$

In reality, the data may be explained by some underlying, continuous function $y = f(x)$. Let's say for some reason we needed either the derivative,

$$\frac{dy}{dx} = f'(x)$$

the definite integral,

$$\int_a^b f(x)\,dx$$

or a function defined by an integral,

$$g(x) = \int_a^x f(x)\,dx$$

If we knew $f(x)$, we could (in almost all cases) obtain its derivative, $f'(x)$. However, even with knowledge of $f(x)$, it is likely that we would not be able to evaluate either of the integrals shown above. The numerical differentiation and integration methods presented in Sections 2 and 3 will allow us to approximate these derivatives/integrals using the discrete function $\mathbf{y}$ vs. $\mathbf{x}$ (i.e. the data set).

2 NUMERICAL DIFFERENTIATION

2.1 Finite Difference Approximations [2, 5]

2.1.1 | Forward Difference Approximation

$$\boxed{\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i}} \tag{2}$$

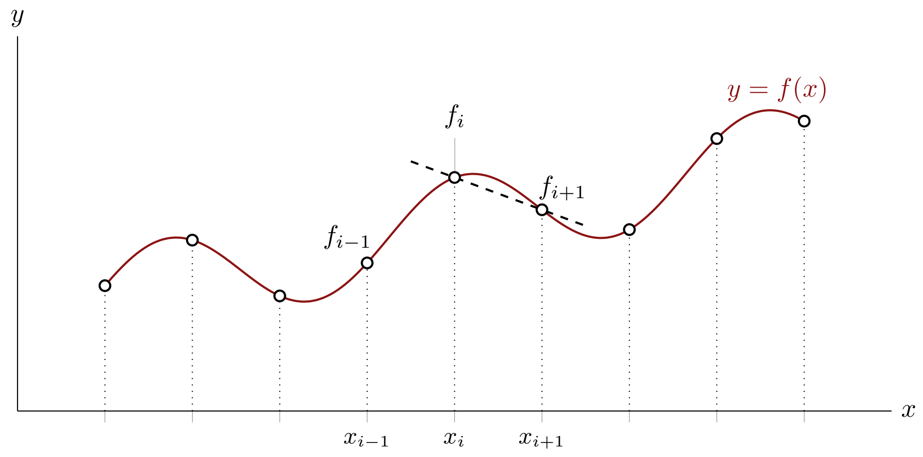

Figure 2: Forward approximation.

2.1.2 | Backward Difference Approximation

$$\boxed{\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_i - y_{i-1}}{x_i - x_{i-1}}} \tag{3}$$

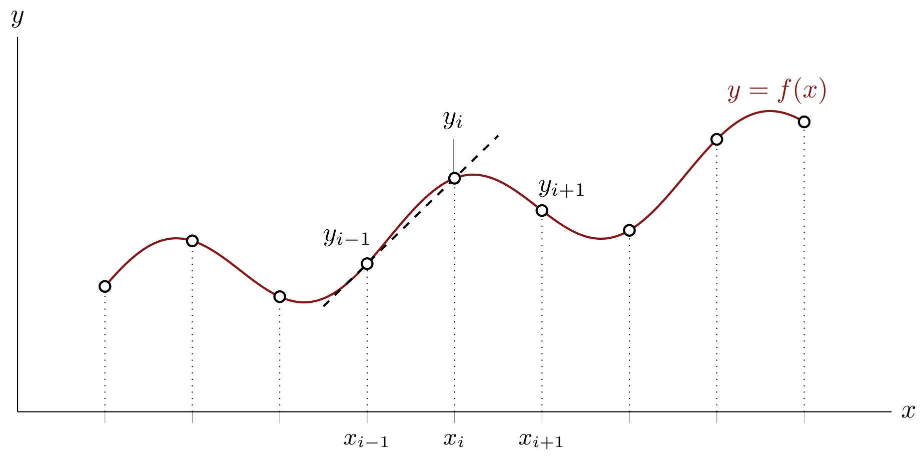

Figure 3: Backward approximation.

2.1.3 | Central Difference Approximation

$$\boxed{\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_{i+1}-y_{i-1}}{x_{i+1}-x_{i-1}}} \tag{4}$$

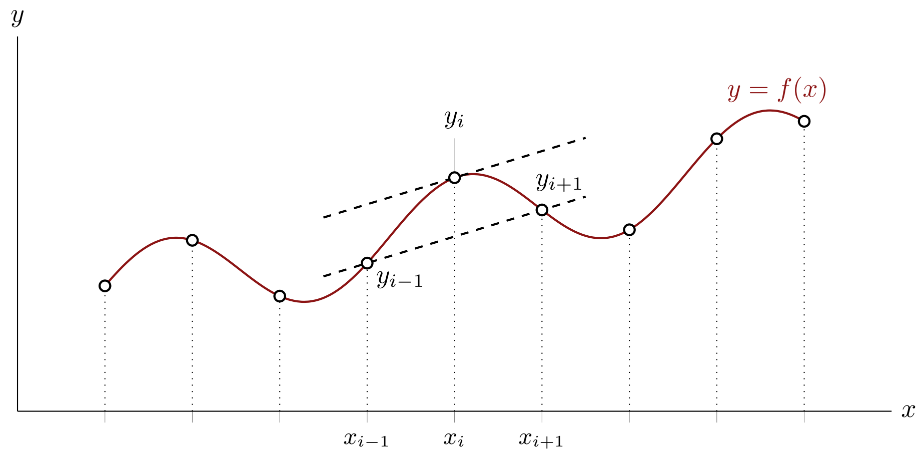

Figure 4: Central approximation.

The central approximation is of higher accuracy than the forward and backward approximations.

2.2 Differentiation Over an Interval (Cumulative Differentiation)

Consider the vectors $\mathbf{y}$ and $\mathbf{x}$ storing sampled points from an underlying function $y = f(x)$. We can consider these vectors as a set of points or a data set:

$$\mathbf{y} \text{ vs. } \mathbf{x} \quad \equiv \quad \{(x_i, y_i)\}_{i=1}^{N+1}$$

$$\therefore \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_{N+1} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{N+1} \end{bmatrix}$$

Note that $y_i = f(x_i)$.

Our goal is to find the derivative $dy/dx = f'(x)$, but without knowledge of $f(x)$, we cannot use simple algebraic differentiation rules. Instead, since we know the discrete $\mathbf{y}$ vs. $\mathbf{x}$ that essentially stores sampled values of $y = f(x)$, we can numerically estimate the value of the derivative at all the points stored in the vector $\mathbf{x}$. The result of this numerical differentiation is a vector $\mathbf{dy}$ that stores the numerical evaluation of $dy/dx = f'(x)$ at all the points in $\mathbf{x}$.

$$\mathbf{dy} = \begin{bmatrix} dy_1 \\ \vdots \\ dy_{N+1} \end{bmatrix} = \begin{bmatrix} \left.\frac{dy}{dx}\right|_{x=x_1} \\ \vdots \\ \left.\frac{dy}{dx}\right|_{x=x_{N+1}} \end{bmatrix}$$

I refer to this numerical differentiation process as **cumulative differentiation**, analogous to cumulative integration in the case of numerical integration (see Section 3.3). The algorithm I use to perform cumulative differentiation is rather simple. At all interior nodes, I use a central approximation to approximate the derivative. At the left endpoint, I use a forward approximation, since there is no x_1 or y_1. At the right endpoint, I use a backward approximation, since there is no x_{N+2} or y_{N+2}.

Algorithm 1:
Cumulative differentiation.

Given:

- $\mathbf{x} \in \mathbb{R}^{N+1}$ - vector of x values
- $\mathbf{y} \in \mathbb{R}^{N+1}$ - vector storing evaluations of $y = f(x)$ at every point in $\mathbf{x}$

Procedure:

1. Determine the number of subintervals, N, given that $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{N+1}$.
2. Preallocate $\mathbf{dy} \in \mathbb{R}^{N+1}$ to store the cumulative derivative.
3. Calculate derivative at left endpoint using forward difference approximation.
$$dy_1 = \frac{y_2 - y_1}{x_2 - x_1}$$
4. Calculate derivative at right endpoint using backward difference approximation.
$$dy_{N+1} = \frac{y_{N+1} - y_N}{x_{N+1} - x_N}$$
5. Calculate derivatives at all other points using central difference approximation.

```
for $i = 2$ to $N$
    $dy_i = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}$
end
```

Return:

- $\mathbf{dy} \in \mathbb{R}^{N+1}$ - vector storing the evaluation of $dy/dx = f'(x)$ at every point in $\mathbf{x}$

2.3 Differentiation at a Point (Point Differentiation)

Previously, we introduced an algorithm (Algorithm 1) for approximating the derivative $dy/dx = f'(x)$ at every node x_i. In this section, we only want to approximate the derivative at a specific point (or at a specific set of points). Note that these points do *not* have to be the nodes we used to discretize $y = f(x)$ (or if using a data set, we can approximate derivatives at points not included in the data set). We refer to this as **point differentiation**. To perform point differentiation, we first find the derivative at every node using cumulative differentiation (i.e. Algorithm 1). Then, we use linear interpolation to linearly interpolate a value for $f'(x_j^*)$ at every x_j^*.

Consider the case where there are p points x_j^* (where $j = 1, ..., p$) at which we wish to evaluate the derivative of $y = f(x)$. We then define a vector $\mathbf{x}^*$ as

$$\mathbf{x}^* = \begin{bmatrix} x_1^* \\ \vdots \\ x_p^* \end{bmatrix}$$

Let $\mathbf{dy}^*$ be the vector in which we store the evaluations of $f'(x_j^*)$. Then

$$\mathbf{dy}^* = \begin{bmatrix} dy_1 \\ \vdots \\ dy_p \end{bmatrix} = \begin{bmatrix} \left.\dfrac{dy}{dx}\right|_{x=x_1^*} \\ \vdots \\ \left.\dfrac{dy}{dx}\right|_{x=x_p^*} \end{bmatrix}$$

Algorithm 2:
Point differentiation.

Given:

- $\mathbf{x} \in \mathbb{R}^{N+1}$ - vector of x values
- $\mathbf{y} \in \mathbb{R}^{N+1}$ - vector storing evaluations of $y = f(x)$ at every point in $\mathbf{x}$
- $\mathbf{x}^* \in \mathbb{R}^p$ - point(s) at which to differentiate

Procedure:

1. Find $\mathbf{dy}$ (i.e. the cumulative derivative of $\mathbf{f}$ vs. $\mathbf{x}$) using Algorithm 1.
2. Find $\mathbf{dy}^*$ (i.e. the point derivatives at $\mathbf{x}^*$) by linearly interpolating/extrapolating $\mathbf{dy}$ at every point in $\mathbf{x}^*$ (can be done using MATLAB's `interp1` function with the ``linear`` and ``extrap`` options specified).

Return:

- $\mathbf{dy}^* \in \mathbb{R}^p$ - vector storing the evaluation of $dy/dx = f'(x)$ at every point in $\mathbf{x}^*$

3 NUMERICAL INTEGRATION

3.1 Types of Integration

In most math courses, we encounter two main types of integration: **definite integration** and **indefinite integration**. At its core, the definite integral of a function $f(x)$ over the interval $[a, b]$ computes the area bounded by f, the x-axis, $x = a$, and $x = b$. The actual calculation is performed as

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

where $F(x)$ is the *anti*derivative of $f(x)$. On the other hand, the indefinite integral of $f(x)$ is just its antiderivative *plus* a constant C.

$$\int f(x)\,dx = F(x) + C$$

Whereas the definite integral gives us a number, an indefinite integral gives us a *family* of functions (since there are infinite possibilities for C). Definite integration is discussed in Section 3.2, while the numerical analog of indefinite integration, called **cumulative integration**, is discussed in Section 3.3.

3.1.1 | Motivation for Numerical Integration

Numerical integration methods are use to approximate integration. There are three cases in which numerical integration is most often used [3]:

1. The integrand $y = f(x)$ may only be known at certain points (i.e. you have the data set/discrete function **y** vs. **x**).
2. It may be impossible to find the antiderivative of $f(x)$ in closed form.
3. It may be possible to find the antiderivative of $f(x)$ in closed form, but it is easier to just integrate numerically.

3.2 Definite Integration

The definite integral of a univariate function $f(x)$ on the interval $[a, b]$ computes the area bounded by f, the x axis, and the vertical lines $x = a$ and $x = b$. To approximate a definite integral, we can split the interval $[a, b]$ into smaller intervals, *approximate* the area bounded by each of these smaller intervals, and then sum the areas of the smaller intervals. If the function $f(x)$ is unknown and we know only the sampled values f_i at some x values x_i, then approximating the integral is in fact all we can do. The **trapezoidal rule** approximates a definite integral by splitting up the area under the curve into multiple trapezoids, as shown in Fig. 5. The area A of a trapezoid with base b and

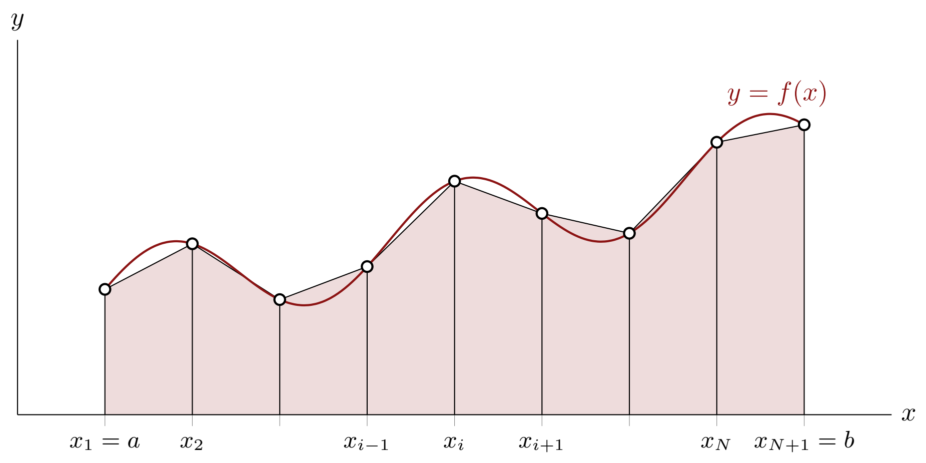

Figure 5: Trapezoidal rule.

heights h_1 (at left boundary) and h_2 (at right boundary) is

$$A = \frac{b\left(h_1 + h_2\right)}{2}$$

The area of the trapezoid defined by (x_i, f_i) and (x_{i+1}, f_{i+1}) is then

$$A = \frac{\overbrace{\left(x_{i+1} - x_i\right)}^{b}\overbrace{\left(f_i + f_{i+1}\right)}^{h_1+h_2}}{2}$$

Thus, to approximate the integral using the trapezoidal rule, we simply add up the areas of all the individual trapezoids [4].

$$\boxed{\int_a^b f(x)\,dx \approx \sum_{i=1}^{N}\left[\frac{\left(f_{i+1} + f_i\right)\left(x_{i+1} - x_i\right)}{2}\right]} \tag{5}$$

Algorithm 3: `definite_integral`
Definite integration.

Given:

- $\mathbf{x} \in \mathbb{R}^{N+1}$ - vector of x values
- $\mathbf{f} \in \mathbb{R}^{N+1}$ - vector storing evaluations of $f(x)$ at every point in $\mathbf{x}$

Procedure:

1. Determine the number of subintervals, N, given that $\mathbf{x}, \mathbf{f} \in \mathbb{R}^{N+1}$.
2. Initialize a variable, I, to store the definite integral.

   $I = 0$

3. Evaluate the definite integral using the trapezoidal rule.

```
for $i = 1$ to $N$
    $I = I + \frac{(f_i + f_{i-1})(x_i - x_{i-1})}{2}$
end
```

Return:

- $I \in \mathbb{R}$ - definite integral of $f(x)$ over the interval defined by $\mathbf{x}$ (and where $f(x)$ is given in discrete form as $\mathbf{f}$ vs. $\mathbf{x}$)

3.3 Cumulative Integration

Definite integration calculates a single area over an interval. What if we want to know the definite integral from the lower bound to *every single* point x in the interval $[a, b]$? Instead of specifying the upper bound of integration, we can leave it parameterized as x.

$$\int_a^x f(x)\,dx = F(x) - F(a)$$

We can note that $F(x)$ is the antiderivative of $f(x)$, while $F(a)$ is this antiderivative evaluated at $x = a$ (and therefore a constant). Thus, a cumulative integral produces a *function*.

We know that using a numerical approach, we can never return a continuous function; we can only return *values* of a function at specified, discrete points (recall that we refer to this set of points as a discrete function). Therefore, from a discrete standpoint, we can consider a data set $\mathbf{f}$ vs. $\mathbf{x}$, or a function f sampled at discrete values of x. Either way, we have a set of $N+1$ points (x_i, y_i) (there are $N+1$ nodes since we define a computational domain as having N subintervals – see Section 1.2). Performing cumulative integration numerically should thus return an array of $N+1$ values, where the value at the i^{th} index represents the definite integral of elements 0 through i of the original data set [1].

Let $\text{CI}(x)$ represent the cumulative integral of $f(x)$ at x. In a continuous form, we can write

$$\text{CI}(x) = \int_a^x f(x)\,dx \tag{6}$$

At an arbitrary node x_j, we can approximate $\text{CI}(x_j) = \text{CI}_j$ using the trapezoidal rule defined by Eq. (5).

$$\text{CI}_j = \int_a^{x_j} f(x)\,dx \approx \sum_{i=1}^{j-1} \left[\frac{(f_{i+1}+f_i)(x_{i+1}-x_i)}{2}\right] \tag{7}$$

Similarly, to find CI_{j+1}, we *could* calculate

$$\text{CI}_{j+1} = \int_a^{x_j} f(x)\,dx \approx \sum_{i=1}^{j} \left[\frac{(f_{i+1}+f_i)(x_{i+1}-x_i)}{2}\right]$$

However, we can notice that when calculating CI_{j+1} in the form above, we are essentially repeating almost every single calculation performed when calculating CI_j, which would make this a very inefficient way to program the cumulative integral. Instead, we can split up the summation.

$$\text{CI}_{j+1} \approx \sum_{i=1}^{j-1} \left[\frac{(f_{i+1}+f_i)(x_{i+1}-x_i)}{2}\right] + \frac{(f_{j+1}+f_j)(x_{j+1}-x_j)}{2} \tag{8}$$

Substituting Eq. (7) into Eq. (8),

$$\boxed{\text{CI}_{j+1} = \text{CI}_j + \frac{(f_{j+1}+f_j)(x_{j+1}-x_j)}{2}} \tag{9}$$

With Eq. (9), it becomes simple to write an algorithm to compute the cumulative integral of a function.

Algorithm 4: `cumulative_integral`

Cumulative integration.

Given:

- $\mathbf{x} \in \mathbb{R}^{N+1}$ - vector of x values
- $\mathbf{f} \in \mathbb{R}^{N+1}$ - vector storing evaluations of $f(x)$ at every point in $\mathbf{x}$

Procedure:

1. Determine the number of subintervals, N, given that $\mathbf{x}, \mathbf{f} \in \mathbb{R}^{N+1}$.
2. Preallocate $\mathbf{CI} \in \mathbb{R}^{N+1}$ to store the cumulative integral.
3. Set the first element of $\mathbf{CI}$ equal to 0 (since the integral from $x = a$ to $x = a$ is 0). *Note: This step is only necessary if* $\mathbf{CI}$ *was NOT initialized as a vector of zeros.*

   $$\mathbf{CI}_1 = 0$$

4. Evaluate the cumulative integral.

```
for $i = 1$ to $N$
    $\text{CI}_{i+1} = \text{CI}_i + \dfrac{(f_i + f_{i-1})(x_i - x_{i-1})}{2}$
end
```

Return:

- **CI** $\in \mathbb{R}^{N+1}$ - cumulative integral of $f(x)$ over the interval defined by $\mathbf{x}$ (and where $f(x)$ is given in discrete form as $\mathbf{f}$ vs. $\mathbf{x}$)

REFERENCES

[1] *Cumulative Integral*. National Institute of Standards and Technology. https://www.itl.nist.gov/div898/software/dataplot/refman2/ch3/cum_inte.pdf (accessed: June 3, 2020).

[2] *Finite difference method*. Wikipedia. https://en.wikipedia.org/wiki/Finite_difference_method (accessed: November 24, 2019).

[3] *Numerical integration*. Wikipedia. https://en.wikipedia.org/wiki/Numerical_integration (accessed: June 3, 2020).

[4] *Trapezoidal rule*. Wikipedia. https://en.wikipedia.org/wiki/Trapezoidal_rule (accessed: June 10, 2020).

[5] Todd Young and Martin J. Mohlenkamp. *Lecture 27: Numerical Differentiation*. Introduction to Numerical Methods and Matlab Programming for Engineers. http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture27.pdf (accessed: June 10, 2020).