Numerical Differentiation

MATLAB Implementation

Tamas Kis | `kis@stanford.edu`

TAMAS KIS
https://github.com/tamaskis

Contents

differentiate

Numerically evaluates the derivative of a univariate function over a domain or at a specified point (or set of points).

Syntax

Discrete Implementation: `f` and `x` are vectors, where `f` stores the evaluation of $f(x)$ at every point in `x`.

```
df = differentiate(f,x)
df = differentiate(f,x,x_star)
```

Continuous Implementation: `f` is a function handle that defines $f(x)$.

```
[df,x] = differentiate(f,[a,b])
[df,x] = differentiate(f,[a,b],dx)
df = differentiate(f,x_star)
df = differentiate(f,x_star,dx)
```

Description

Discrete Implementation

`df = differentiate(f,x)` returns the derivative of a function $f(x)$ over a domain. `x` is a vector of points defining the domain, and `f` is the vector storing the evaluation of $f(x)$ corresponding to every point in `x`. `x` and `f` can also be thought of as the data set `f` vs. `x`.

`df = differentiate(f,x,x_star)` returns the derivative of the function $f(x)$ at a specified point x^* (or set of points $\mathbf{x}^*$). `x` is a vector of points defining the domain, `f` is the vector storing the evaluation of $f(x)$ corresponding to every point in `x`, and `x_star` is either a scalar x^* or vector $\mathbf{x}^*$ storing the point(s) where we wish to evaluate the derivative of $f(x)$.

Continuous Implementation

`[df,x] = differentiate(f,[a,b])` returns the derivative of a function $f(x)$ over the domain $x \in [a,b]$. `f` specifies the function handle for $f(x)$, while `a` and `b` are the lower and upper bounds of the domain. `df` is a vector storing the evaluations of the derivative corresponding to the points in `x`. This syntax defaults to a grid spacing of $dx = (b-a)/1000$.

`[df,x] = differentiate(f,[a,b],dx)` returns the derivative of a function $f(x)$ over the domain $x \in [a,b]$. `f` specifies the function handle for $f(x)$, `a` and `b` are the lower and upper bounds of the domain, and `dx` is the grid spacing dx. `df` is a vector storing the evaluations of the derivative corresponding to the points in `x`.

`df = differentiate(f,x_star)` returns the derivative of a function $f(x)$ evaluated at a specified point x^* (or set of points $\mathbf{x}^*$). `f` specifies the function handle for $f(x)$ and `x_star` is either a scalar x^* or vector $\mathbf{x}^*$ storing the point(s) at which to differentiate. This syntax defaults to a grid spacing of $dx = 10000\varepsilon$, where ε is the machine epsilon (i.e. smallest possible nonzero number).

`df = differentiate(f,x_star,dx)` returns the derivative of a function $f(x)$ evaluated at a specified point x^* (or set of points $\mathbf{x}^*$). `f` specifies the function handle for $f(x)$, `x_star` is either a scalar x^* or vector $\mathbf{x}^*$ storing the point(s) at which to differentiate, and `dx` specifies the grid spacing dx.

Note

The syntaxes involving `x_star` do NOT work when $\mathbf{x}^* \in \mathbb{R}^2$ due to the logic of the code. Therefore, if you wish to evaluate the derivative at two specific points, then you should add a third "dummy" point to $\mathbf{x}^*$. For example, if you wanted to evaluate a derivative at $x = 5$ and $x = 7$, you should define $\mathbf{x}^* = (5, 7, 0)^T$ (where 0 serves as the dummy variable) and *not* $\mathbf{x}^* = (5, 7)^T$. You can then just discard the result you get for $x = 0$. This is demonstrated in Example 7.

Examples

Example 1

Find the derivative of the following data set:

x	$f(x) = x^3$
0	0
1	1
2	8
3	27
4	64
5	125

■ ***SOLUTION***

Our first step is to define vectors to store this data set.

$$\mathbf{x} = (0, 1, 2, 3, 4, 5)^T, \qquad \mathbf{f} = (0, 1, 8, 27, 64, 125)^T$$

Defining these vectors in MATLAB,

```
x = [0,1,2,3,4,5];
f = [0,1,8,27,64,125];
```

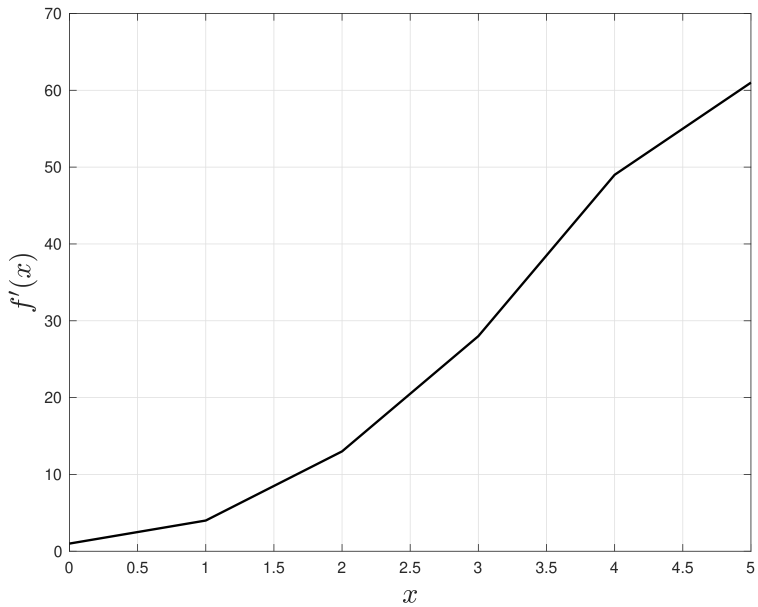
Differentiating `f` over the domain specified by the vector `x`,

```
df = differentiate(f,x);
```

To plot the result,

```
figure;
plot(x,df,'k','linewidth',1.5);
grid on;
xlabel('$x$','interpreter','latex','fontsize',18);
ylabel("$f'(x)$",'interpreter','latex','fontsize',18);
```

The resulting plot is shown below. Note how coarse this plot appears, and how the convexity of $f'(x)$ changes for the last interval (between $x = 4$ and $x = 5$). This is due to numerical errors and the coarse grid (i.e. `x`) used. If we used a much finer distribution of x values, the plot would look much more like the true solution of $f'(x) = 3x^2$. In the next example, we consider the continuous implementation (i.e. using a function handle), which by default uses a discretization with 1001 nodes.



Example 2

Consider the following data set:

x	$f(x) = x^3$
0	0
1	1
2	8
3	27
4	64
5	125

Find $f'(x^*)$ for $x^* = 3.5$ (i.e. evaluate $f'(x^*)$) using the data alone.

■ *SOLUTION*

Our first step is to define vectors to store this data set.

$$\mathbf{x} = (0, 1, 2, 3, 4, 5)^T, \qquad \mathbf{f} = (0, 1, 8, 27, 64, 125)^T$$

Defining these vectors in MATLAB,

```
x = [0,1,2,3,4,5];
f = [0,1,8,27,64,125];
```

Defining the point at which we wish to differentiate $f(x)$,

```
x_star = 3.5;
```

Evaluating the derivative of $\mathbf{f}$ vs. $\mathbf{x}$ at x^*,

```
df = differentiate(f,x_star)
```

This yields the result

```
df =

   38.5000
```

Example 3

Plot the derivative of $f(x) = x^3$ over the domain $x \in [0, 5]$.

■ *SOLUTION*

Defining $f(x)$ using a function handle (note that this function handle does *not* need to be defined using elementwise operations (i.e. `f = @(x) x.^3`)),

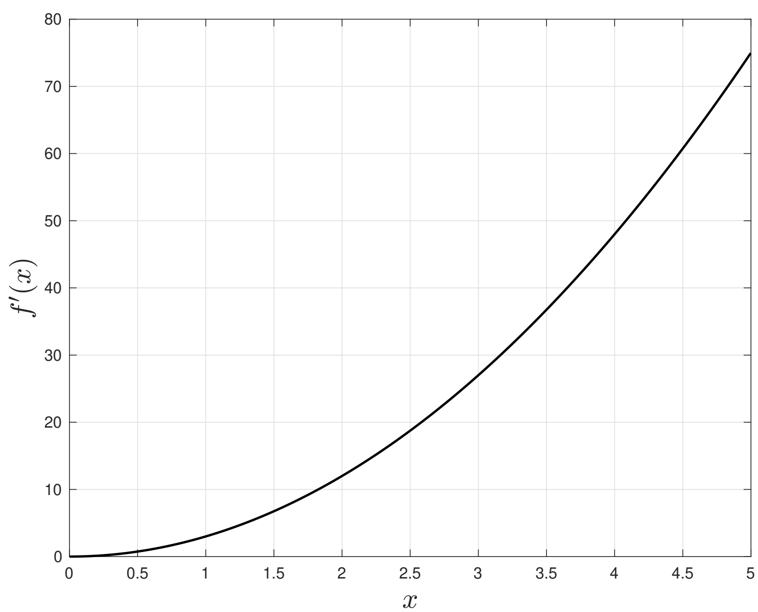
```
f = @(x) x^3;
```

Differentiating $f(x)$ over $x \in [0, 5]$,

```
[df,x] = differentiate(f,[0,5]);
```

Plotting the result,

```
figure;
plot(x,df,'k','linewidth',1.5);
grid on;
xlabel('$x$','interpreter','latex','fontsize',18);
ylabel("$f'(x)$",'interpreter','latex','fontsize',18);
```



Example 4

Plot the derivative of $f(x) = x^3$ over the domain $x \in [0, 5]$ using a grid spacing of $dx = 1$.

■ ***SOLUTION***

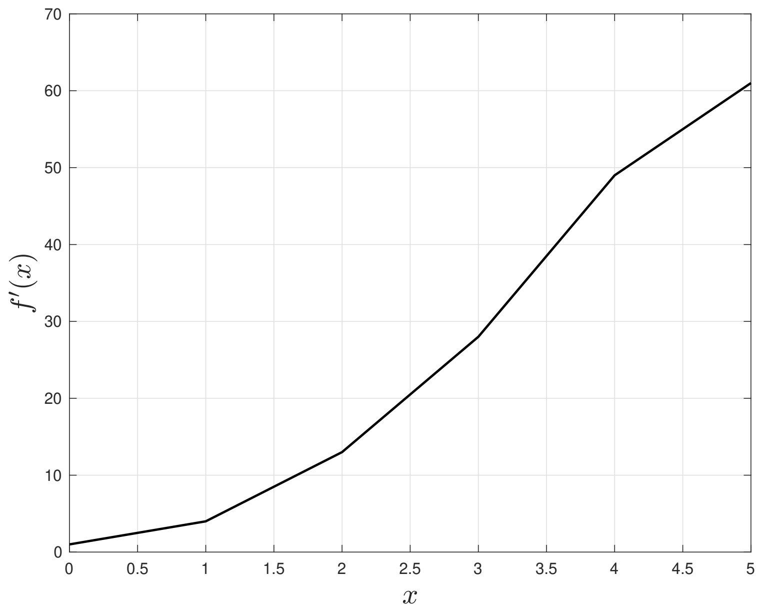
Defining $f(x)$ using a function handle,

```
f = @(x) x^3;
```

Differentiating $f(x)$ over $x \in [0, 5]$ using a grid spacing of $dx = 1$,

```
[df,x] = differentiate(f,[0,5],1);
```

This yields the following plot:



Note that this matches the plot from Example 1. This is because in the differentiation process, the `differentiate` function will essentially create that same data set and then differentiate that.

Example 5

Numerically evaluate the derivative of the function $f(x) = x^3$ at the point $x^* = 3.5$ (i.e. find $f'(3.5)$ for $f(x) = x^3$).

■ ***SOLUTION***

Defining $f(x)$ using a function handle,

```
f = @(x) x^3;
```

Differentiating $f(x)$ at $x^* = 3.5$,

```
[df,x] = differentiate(f,3.5);
```

This yields the result

```
df =

   36.7488
```

which is nearly identical to the true solution:

$$f'(x) = 3x^2 \quad \rightarrow \quad f'(3.5) = 36.75$$

Example 6

Numerically evaluate the derivative of the function $f(x) = x^3$ at the point $x^* = 3.5$ (i.e. find $f'(3.5)$ for $f(x) = x^3$) using a grid spacing of $dx = 1$.

■ ***SOLUTION***

Defining $f(x)$ using a function handle,

```
f = @(x) x^3;
```

Differentiating $f(x)$ at $x^* = 3.5$ using a grid spacing of $dx = 1$,

```
[df,x] = differentiate(f,3.5,1);
```

This yields the result

```
df =

   37.7500
```

Example 7

Numerically evaluate the derivative of the function $f(x) = x^3$ at the points $x^* = 2.5$ and $x^* = 3.5$.

■ ***SOLUTION***

Defining $f(x)$ using a function handle,

```
f = @(x) x^3;
```

We want to evaluate $f'(x)$ at two points, so we would have $\mathbf{x}^* \in \mathbb{R}^2$. However, as noted above in the documentation, `differentiate` will not work properly for $\mathbf{x}^* = 2$. Therefore, instead of using $\mathbf{x}^* = (2.5, 3.5)^T$, we will use

$$\mathbf{x}^* = (2.5, 3.5, 0)^T$$

Defining $\mathbf{x}^*$ in MATLAB,

```
x_star = [2.5;3.5;0];
```

Differentiating $f(x)$ at all the points in $\mathbf{x}^*$,

```
[df,x] = differentiate(f,x_star);
```

This yields the result

```
df =

   18.7504
   36.7488
    0.0000
```

The last element of `df` can be just ignored, as it corresponds to $x^* = 0$, which we just included as a dummy to get `differentiate` to work properly.

Links

MATLAB® Central's File Exchange:
https://www.mathworks.com/matlabcentral/fileexchange/89719-numerical-differentiation-differentiate

GitHub®:
https://github.com/tamaskis/differentiate-MATLAB

Numerical Differentiation

Domain Discretization

When analyzing a function numerically, the first thing we do is discretize the 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 $N+1$ points)

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 [1]. 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 denote the evaluation of $f(x)$ at a node x_i as

$$f_i = f(x_i)$$

We can then also define a vector $\mathbf{f}$ to store all the f_i's:

$$\mathbf{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_{N+1} \end{bmatrix}$$

We can consider the vector $\mathbf{f}$ and $\mathbf{x}$ as defining a discrete form of $f(x)$, or as vectors storing a data set:

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

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

In some cases, we can be given a data set $\mathbf{f}$ vs. $\mathbf{x}$, and using the methods we introduce, we can find the derivative of this data set without having any knowledge of the function $f(x)$ from which the data set is sampled.

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

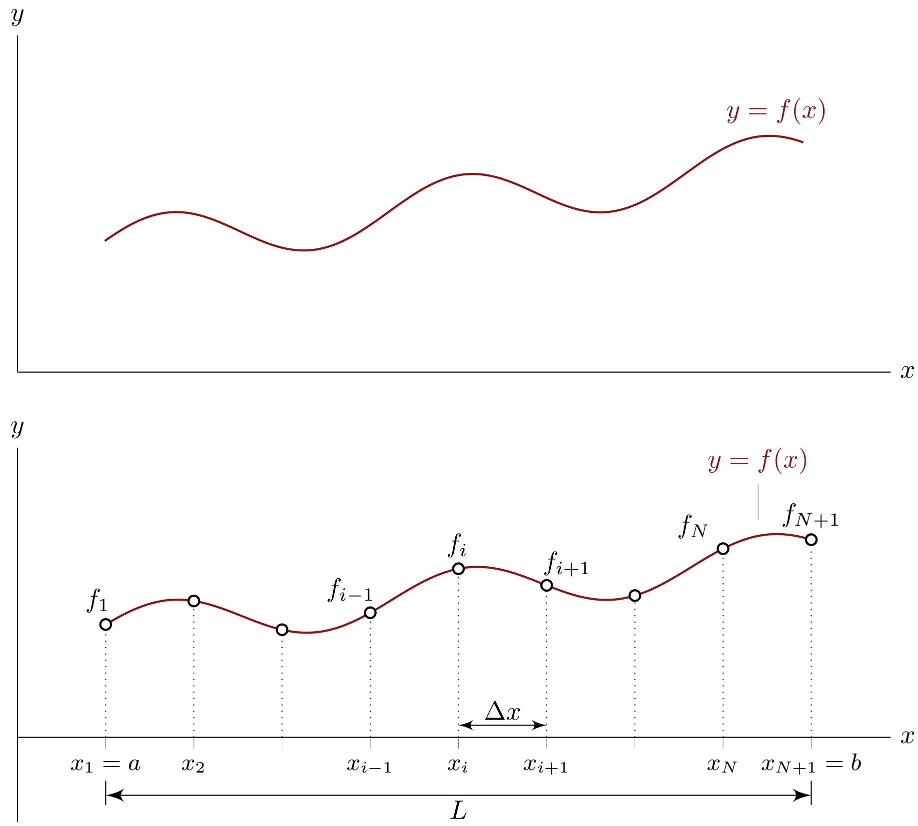


Figure 1: Domain discretization.

Finite Differences

Eq. (2) computes the **forward approximation** of the derivative [1, 2].

$$\boxed{\left.\frac{df}{dx}\right|_{x=x_i} \approx \frac{f_{i+1} - f_i}{x_{i+1} - x_i}} \tag{2}$$

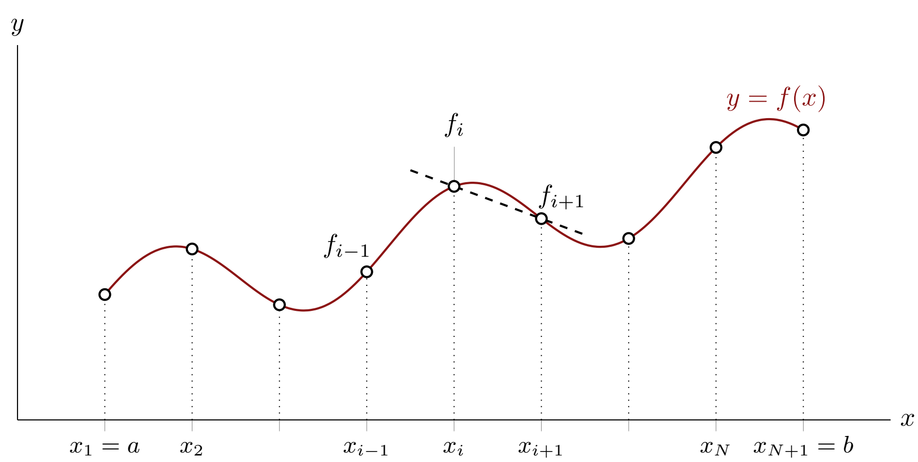


Figure 2: Forward approximation.

Eq. (3) computes the **backward approximation** of the derivative [1, 2].

$$\boxed{\left.\frac{df}{dx}\right|_{x=x_i} \approx \frac{f_i - f_{i-1}}{x_i - x_{i-1}}} \tag{3}$$

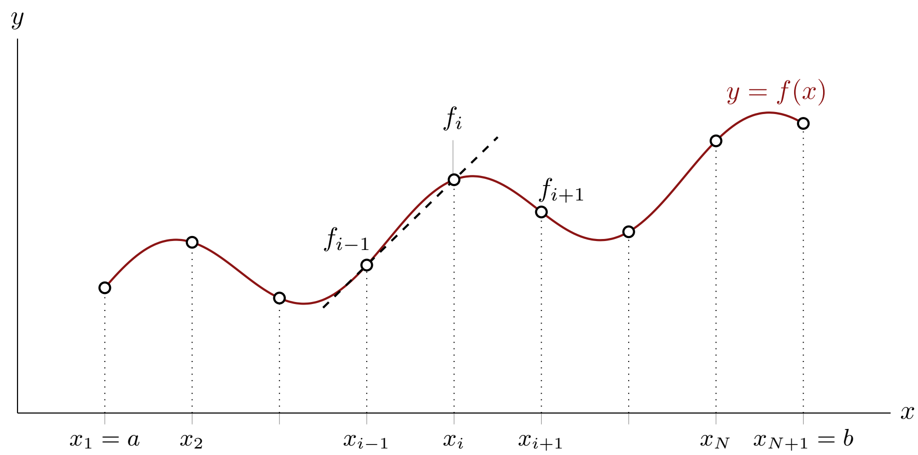


Figure 3: Backward approximation.

Eq. (4) computes the **central approximation** of the derivative. The central approximation is of higher accuracy than the forward and backward approximations [1, 2].

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

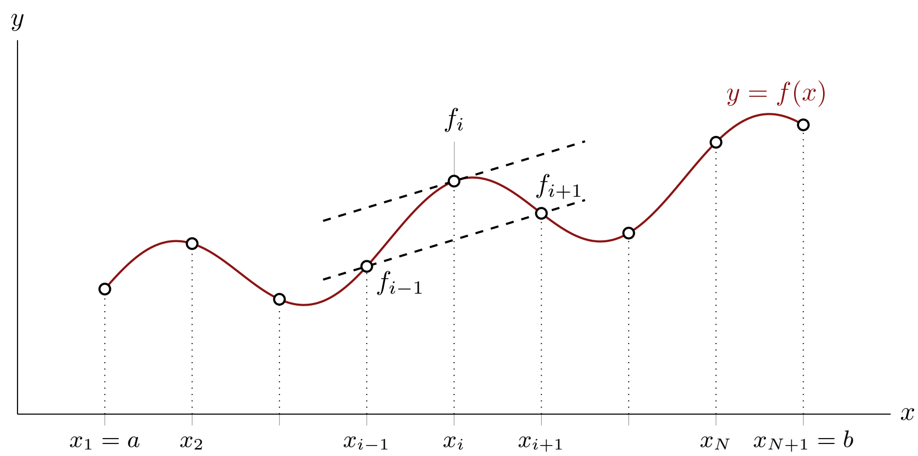


Figure 4: Central approximation.

Cumulative Differentiation

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

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

$$\therefore \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_{N+1} \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_{N+1} \end{bmatrix}$$

Note that we use the notation $f_i = f(x_i)$.

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

$$\mathbf{df} = \begin{bmatrix} df_1 \\ \vdots \\ df_{N+1} \end{bmatrix} = \begin{bmatrix} \left.\dfrac{df}{dx}\right|_{x=x_1} \\ \vdots \\ \left.\dfrac{df}{dx}\right|_{x=x_{N+1}} \end{bmatrix}$$

I refer to this numerical differentiation process as **cumulative differentiation**, analogous to cumulative integration in the case of numerical integration. 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 f_1. At the right endpoint, we use a backward approximation, since there is no x_{N+2} or f_{N+2}.

Algorithm 1: Cumulative differentiation.

Given: $\mathbf{x}$, $\mathbf{f}$

// determines number of subintervals

$N = \text{length}(\mathbf{x}) - 1$

Preallocate the vector $\mathbf{df} \in \mathbb{R}^{N+1}$ to store the cumulative derivative.

// calculates derivative at left endpoint using forward difference approximation

$df_1 = \dfrac{f_2 - f_1}{x_2 - x_1}$

// calculates derivative at right endpoint using backward difference approximation

$df_{N+1} = \dfrac{f_{N+1} - f_N}{x_{N+1} - x_N}$

// calculates derivatives at all other points using central difference approximation

for $i = 2$ **to** N **do**

$\quad df_i = \dfrac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}}$

end

return $\mathbf{df}$

Point Differentiation

Previously, we introduced an algorithm (Algorithm 1) for calculating the derivative ${f_i}'$ at every node x_i. In this section, we only want to find f' 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 $f(x)$. 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_j' at every x_j^*.

Consider the case where there are n points x_j^* (where $j = 1, ..., n$) at which we wish to evaluate the derivative of $f(x)$. The vector $\mathbf{x}^*$ is then

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

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

$$\mathbf{df}^* = \begin{bmatrix} df_1 \\ \vdots \\ df_n \end{bmatrix} = \begin{bmatrix} \left.\frac{df}{dx}\right|_{x=x_1^*} \\ \vdots \\ \left.\frac{df}{dx}\right|_{x=x_n^*} \end{bmatrix}$$

Algorithm 2: Point differentiation.

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

Numerically Differentiating a Continuous Function

In Algorithms 1 and 2, we assumed $f(x)$ was given in the discrete form $\mathbf{f}$ vs. $\mathbf{x}$; essentially, either the discretization of a known function $f(x)$ was already performed, or we simply have *samples* of the function $f(x)$, but don't know what $f(x)$ actually is. Thus, we could modify these algorithms to accept a continuous function $f(x)$ (in the case of MATLAB, passed in as a function handle) and perform the discretization inside the function. This is essentially what is done in the `differentiate` function. This function is set up such that by default, 1000 subintervals (corresponding to 1001 nodes) are used to create a discretized domain, or if differentiating at a point, a default grid spacing of $dx = 10000\varepsilon$ (where ε is the machine epsilon) is used. `differentiate` also accepts the grid spacing as an optional input, if the user wishes to specify it.

References

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

[2] *Lecture 27: Numerical Differentiation*. http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture27.pdf. (accessed: June 10, 2020).