Introduction to Algorithmic Differentiation

Motivation. Essential Calculus. Finite Differences ($f : \boldsymbol{R} \to \boldsymbol{R}$)

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Why $f : R \to R$?

Rationale

- "low-hanging fruit"
- easy entry into subject
- simple notation
- comprehension of generalization for $F : R^n \to R^m$ facilitated by conceptual understanding
- actually $f(x, \mathbf{p}) : R \times R^{\tilde{n}} \to R$ with passive parameters $\mathbf{p} \in R^{\tilde{n}}$ and interest in $f'(x, \mathbf{p}) \equiv \frac{df}{dx}(x, \mathbf{p})$ (see case study)

Contents

Outline

Motivation

Let $x = x(\mathbf{p}) \in \boldsymbol{R}$ be defined implicitly for a vector of passive parameters $\mathbf{p} \in \boldsymbol{R}^{\tilde{n}}$ as the solution of the nonlinear equation

$$f(x, \mathbf{p}) = 0$$

with continuously differentiable (at points $x^i = x^i(\mathbf{p})$, $i = 0, 1, \ldots$) residual

$$f : \boldsymbol{R} \times \boldsymbol{R}^{\tilde{n}} \rightarrow \boldsymbol{R}$$

and first derivative $f'(x^i, \mathbf{p}) \equiv \frac{df}{dx}(x^i, \mathbf{p})$.

For a given start value $x^0 \in \boldsymbol{R}$, Newton's method aims to approximate the solution iteratively as

$$x^{i+1} = x^i - \frac{f(x^i, \mathbf{p})}{f'(x^i, \mathbf{p})} \quad \text{for } i = 0, 1, \ldots$$

until $|f(x^i, \mathbf{p})| < \epsilon$ for given accuracy $0 < \epsilon \ll 1$.

Let $xi = x(\mathbf{p}) \in \boldsymbol{R}$ be defined implicitly for $\mathbf{p} \in \boldsymbol{R}^{\tilde{n}}$ as the solution of the optimization problem

$$\min_{x=x(\mathbf{p})} f(x,\mathbf{p})$$

with twice continuously differentiable (at $x^i(\mathbf{p})$, $i = 0, 1, \ldots$) objective

$$f : \boldsymbol{R} \times \boldsymbol{R}^{\tilde{n}} \to \boldsymbol{R}$$

and first and second derivative $f'(x^i)$ and $f''(x^i) \equiv \frac{df'}{dx}(x^i, \mathbf{p})$.

For a given start value $x^0 \in \boldsymbol{R}$, Newton's method aims to approximate a stationary point $x^* = x^*(\mathbf{p})$, $f'(x^*) = 0$, iteratively as

$$x^{i+1} = x^i - \frac{f'(x^i,\mathbf{p})}{f''(x^i,\mathbf{p})} \quad \text{for } i = 0, 1, \ldots$$

until $|f'(x^i,\mathbf{p})| < \epsilon$ for given accuracy $0 < \epsilon \ll 1$. A local minimum is found if additionally $f''(x^i,\mathbf{p}) > 0$.

Outline

Contents

- continuity
- differentiability and derivatives
- linearity and beyond
- convexity / concavity
- Taylor series
- linearization
- chain rule

Let $[(a, b) \subseteq]$ $\boldsymbol{R}$ be the (open) domain of the univariate scalar function $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ with image in $\boldsymbol{R}$.

$f(x)$ is right-continuous at $\tilde{x} \in \boldsymbol{R}$ if

$$\lim_{\Delta x \to 0, \Delta x > 0} f(\tilde{x} + \Delta x) = f(\tilde{x}) \quad .$$

f is left-continuous at $\tilde{x}$ if

$$\lim_{\Delta x \to 0, \Delta x > 0} f(\tilde{x} - \Delta x) = f(\tilde{x}) \quad .$$

f is continuous at $\tilde{x}$ if it is both left- and right-continuous at $\tilde{x}$.

Continuity is a necessary condition for differentiability.

Let R be the domain of the univariate scalar function $f : R \rightarrow R$.

$f(x)$ is right-differentiable at $\tilde{x} \in R$ if the limit

$$\lambda^{+} = \lim_{\Delta x \rightarrow 0} \frac{f(\tilde{x} + \Delta x) - f(\tilde{x})}{\Delta x}$$

exists (is finite). f is left-differentiable at $\tilde{x}$ if

$$\lambda^{-} = \lim_{\Delta x \rightarrow 0} \frac{f(\tilde{x}) - f(\tilde{x} - \Delta x)}{\Delta x}$$

exists (is finite). f is differentiable at $\tilde{x}$ if it is both left- and right-differentiable with first derivative

$$\lambda^{+} = \lambda^{-} \equiv \frac{df}{dx}(\tilde{x}) = f'(\tilde{x}) = f^{[1]}(\tilde{x}) \quad .$$

Wake Up!

Is

$$y = \begin{cases} \sin(x) & x \leq \frac{\pi}{4} \\ \cos(x) & \text{otherwise} \end{cases}$$

continuous / differentiable?

The first derivative of a continuously differentiable univariate scalar function $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ is continuous making f potentially twice differentiable.

The $(i+1)$-th derivative $f^{[i+1]}$ of a $i+1$ times differentiable function $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ is the first derivative of the continuous i-th derivative of f for $i = 1, 2 \ldots,$ i.e,

$$
\begin{aligned}
f^{[2]} &= f'' \equiv \frac{d^2 f}{dx^2} = \frac{d\,\frac{df}{dx}}{dx} = (f')' = (f^{[1]})' \\
f^{[3]} &= f''' \equiv \frac{d^3 f}{dx^3} = (f^{[2]})' \\
f^{[4]} &\equiv \frac{d^4 f}{dx^4} = (f^{[3]})' \\
&\vdots
\end{aligned}
$$

A function $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ is called linear if

- $f(a+b) = f(a) + f(b)$
- $f(\alpha \cdot a) = \alpha \cdot f(a)$

for all $a, b, \alpha \in \boldsymbol{R}$.

Example: $f(x) = p \cdot x$ with constant $p \in \boldsymbol{R}$ is linear.

$$f(a+b) = p \cdot (a+b) = p \cdot a + p \cdot b = f(a) + f(b)$$
$$f(\alpha \cdot a) = p \cdot \alpha \cdot a = \alpha \cdot p \cdot a = \alpha \cdot f(a)$$

Functions of the form $f(x) = p \cdot x + q$ with constant $p, q \in \boldsymbol{R}$ are called affine. Linear functions are affine with $q = 0$.

Roots of affine functions are defined by linear equations $f(x) = p \cdot x + q = 0$.

Let $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ be analytic (infinitely often differentiable), e.g, $x^2, e^x, \sin(x), \ldots$

f is constant [over (a, b)] if its derivatives vanish identically for all $x \in [(a, b) \subseteq]\boldsymbol{R}$, e.g, $f(x) = 42$ is constant over $\boldsymbol{R}$.

f is (at most) affine if its second and higher derivatives vanish identically for all $x \in \boldsymbol{R}$, e.g, $f(x) = 42 \cdot x - 24$ is affine over $\boldsymbol{R}$ while $f(x) = 42 \cdot x$ is linear.

f is (at most) quadratic if its third and higher derivatives vanish identically for all $x \in \boldsymbol{R}$, e.g, $f(x) = 42 \cdot x^2 - 24 \cdot x + 1$ is quadratic over $\boldsymbol{R}$.

f is (at most) cubic if its fourth and higher derivatives vanish identically for all $x \in \boldsymbol{R}$, e.g, $f(x) = 42 \cdot x^3 - 24$ is cubic over $\boldsymbol{R}$.

etc.

A function $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ is

- [strictly] monotonically increasing over $(a, b) \subseteq \boldsymbol{R}$ if

$$\forall x_0, x_1 \in (a, b) : x_0 < x_1 \Rightarrow f(x_0)[<] \leq f(x_1)$$

or, equivalently, if f is differentiable over (a, b), then

$$\forall x \in (a, b) : f'(x)\ [>]\ \geq 0\,.$$

- [strictly] monotonically decreasing over (a, b) if

$$\forall x_0, x_1 \in (a, b) : x_0 < x_1 \Rightarrow f(x_0)[>] \geq f(x_1)$$

or, equivalently, if f is differentiable over (a, b), then

$$\forall x \in (a, b) : f'(x)\ [<]\ \leq 0\,.$$

Let $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ be continuous over $[a, b] \subset \boldsymbol{R}$. Then f is [strictly] convex if

$$\forall x_0, x_1 \in [a, b] : f\left(\frac{x_0 + x_1}{2}\right) \; [<] \; \leq \frac{f(x_0) + f(x_1)}{2}$$

(points of all secants above the graph of f)

Let $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ be twice differentiable over $[a, b] \subset \boldsymbol{R}$. Then f is [strictly] convex if $f''(x)\; [>] \; \geq 0$ for all $x \in [a, b]$.

Examples: $f(x) = x^2$ and $f(x) = e^x$ are strictly convex over $\boldsymbol{R}$; $f(x) = \sin(x)$ is strictly convex over $(\pi, 2 \cdot \pi)$; $f(x) = 42 \cdot x$ is (not strictly) convex over $\boldsymbol{R}$.

Let $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ be continuous over $[a, b] \subset \boldsymbol{R}$.

f is [strictly] concave if

$$\forall x_0, x_1 \in [a, b] : f\left(\frac{x_0 + x_1}{2}\right) \; [>] \; \geq \frac{f(x_0) + f(x_1)}{2}$$

(points of all secants below the graph of f)

Let $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ be twice differentiable over $[a, b] \subset \boldsymbol{R}$. Then f is [strictly] concave if $f''(x) \; [<] \; \leq 0$ for all $x \in [a, b]$.

Examples: $f(x) = -x^2$ and $f(x) = -e^x$ are strictly concave over $\boldsymbol{R}$; $f(x) = \cos(x)$ is strictly concave over $(-\frac{\pi}{2}, \frac{\pi}{2})$; $f(x) = 42 \cdot x$ is (not strictly) concave over $\boldsymbol{R}$.

Wake Up!

Are quadratic functions always strictly convex or strictly concave?

Let $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ be n-times continuously differentiable.

Given the value of $f(x)$ at some point $\tilde{x} \in \boldsymbol{R}$ the function value $f(\tilde{x} + \Delta x)$ at a neighboring point can be approximated by a Taylor series as

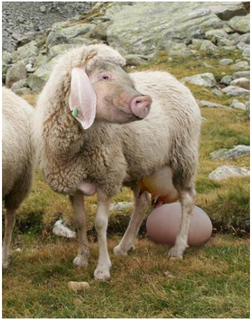
$$f(\tilde{x} + \Delta x) \approx_{O(\Delta x^n)} f(\tilde{x}) + \sum_{k=1}^{n-1} \frac{1}{k!} \cdot \frac{d^k f}{dx^k}(\tilde{x}) \cdot \Delta x^k \; .$$

Throughout this course we assume convergence of the Taylor series for $k \rightarrow \infty$ to the true value of $f(\tilde{x} + \Delta x)$ within all subdomains of interest, which is not the case for arbitrary functions.

For $n = 4$ we get

$$f(\tilde{x} + \Delta x) = f(\tilde{x}) + f'(\tilde{x}) \cdot \Delta x + \frac{1}{2} \cdot f''(\tilde{x}) \cdot \Delta x^2 + \frac{1}{6} \cdot f'''(\tilde{x}) \cdot \Delta x^3 + O(|\Delta x|^4) \; .$$

The solution of linear equations amounts to simple scalar division. The solution of nonlinear equations can be challenging.



"Egg-Laying, Wool-Bearing, Milk-Giving Sow"
©Georg Mittenecker @ Wikipedia

Many numerical methods for nonlinear problems are built on local (at $\tilde{x}$) replacement of the target function with a linear (affine; in Δx) approximation derived from the truncated Taylor series expansion and "hoping" that

$$f(\tilde{x}+\Delta x) \approx f(\tilde{x}) + f'(\tilde{x}) \cdot \Delta x \; ,$$

i.e, hoping for a reasonably small remainder.

The solution of a sequence of linear problems is then expected to yield an iterative approximation of the solution to the nonlinear problem. Newton's method is THE example.

Let $y = f(x) : \boldsymbol{R} \to \boldsymbol{R}$ (or open subdomains in $\boldsymbol{R}$) be such that

$$y = f(x) = f_2(f_1(x)) = f_2(z)$$

with (continuously) differentiable $f_1, f_2 : \boldsymbol{R} \to \boldsymbol{R}$.

Then f is (continuously) differentiable and

$$\frac{df}{dx}(\tilde{x}) = \frac{df_2}{dx}(\tilde{z}) = \frac{df_2}{dz}(\tilde{z}) \cdot \frac{df_1}{dx}(\tilde{x})$$

for all $\tilde{x} \in \boldsymbol{R}$ and $\tilde{z} = f_1(\tilde{x})$.

Wake Up!

What is the first derivative of $e^{\sin(x^2)}$?

Outline

Contents

- derivation: forward, backward, central
- accuracy
- issues with floating-point arithmetic
- second (and higher) order

The linearization of a differentiable nonlinear function f at some point $\tilde{x}$ is a function in Δx:

$$\bar{f}(\Delta x) = f(\tilde{x}) + f'(\tilde{x}) \cdot \Delta x \; .$$

Under the assumption that

$$f(\tilde{x} + \Delta x) \approx f(\tilde{x}) + f'(\tilde{x}) \cdot \Delta x$$

the derivative of $\bar{f}(\Delta x)$ wrt. Δx

$$\bar{f}'(\Delta x) = \bar{f}'(\tilde{x}) = \frac{f(\tilde{x} + \Delta x) - f(\tilde{x})}{\Delta x} \approx f'(\tilde{x})$$

can be used to approximate the derivative $f'(\tilde{x})$ of the nonlinear function f.

This method is known as finite difference approximation.

Variants include

$$\text{forward finite differences}: \quad f'(\tilde{x}) \approx \frac{f(\tilde{x}+\Delta x) - f(\tilde{x})}{\Delta x}$$

$$\text{backward finite differences}: \quad f'(\tilde{x}) \approx \frac{f(\tilde{x}) - f(\tilde{x}-\Delta x)}{\Delta x}$$

$$\text{central finite differences}: \quad f'(\tilde{x}) \approx \frac{f(\tilde{x}+\Delta x) - f(\tilde{x}-\Delta x)}{2 \cdot \Delta x}$$

where $\Delta x = \Delta x(\tilde{x}) \in \boldsymbol{R}$ is a "suitable perturbation" typically picked as a compromise between accuracy and numerical stability, e.g,

$$\Delta x = \begin{cases} \sqrt{\epsilon} & \tilde{x} = 0 \\ \sqrt{\epsilon} \cdot |\tilde{x}| & \tilde{x} \neq 0 \end{cases}$$

with machine epsilon ϵ dependent on the floating-point precision.

For forward finite differences we get

$$f(\tilde{x}+\Delta x)=f(\tilde{x})+\frac{df}{dx}(\tilde{x})\cdot\Delta x+\frac{1}{2!}\cdot\frac{d^2f}{dx^2}(\tilde{x})\cdot\Delta x^2+\frac{1}{3!}\cdot\frac{d^3f}{dx^3}(\tilde{x})\cdot\Delta x^3+\ldots$$

and similarly for backward finite difference

$$f(\tilde{x}-\Delta x)=f(\tilde{x})-\frac{df}{dx}(\tilde{x})\cdot\Delta x+\frac{1}{2!}\cdot\frac{d^2f}{dx^2}(\tilde{x})\cdot\Delta x^2-\frac{1}{3!}\cdot\frac{d^3f}{dx^3}(\tilde{x})\cdot\Delta x^3+\ldots\ .$$

Truncation after the first derivative terms yields, e.g,

$$f(\tilde{x}+\Delta x)=f(\tilde{x})+\Delta x\frac{df}{dx}(\tilde{x})+O(\Delta x^2)\ .$$

For $0<\Delta x<<1$ the truncation error is dominated by the value of the Δx^2 term which implies that only accuracy up to the order of $\Delta x=\frac{\Delta x^2}{\Delta x}$ (and hence first-order accuracy) can be expected for, e.g,

$$\frac{df}{dx}(\tilde{x})=\frac{f(\tilde{x}+\Delta x)-f(\tilde{x})+O(\Delta x^2)}{\Delta x}=\frac{f(\tilde{x}+\Delta x)-f(\tilde{x})}{\Delta x}+O(\Delta x)\ .$$

Second-order accuracy follows immediately from the previous Taylor expansions. Their subtraction yields

$$f(\tilde{x}+\Delta x) - f(\tilde{x}-\Delta x) =$$

$$f(\tilde{x}) + \frac{df}{dx}(\tilde{x}) \cdot \Delta x + \frac{1}{2!} \cdot \frac{d^2 f}{dx^2}(\tilde{x}) \cdot \Delta x^2 + \frac{1}{3!} \cdot \frac{d^3 f}{dx^3}(\tilde{x}) \cdot \Delta x^3 + \ldots -$$

$$\left(f(\tilde{x}) - \frac{df}{dx}(\tilde{x}) \cdot \Delta x + \frac{1}{2!} \cdot \frac{d^2 f}{dx^2}(\tilde{x}) \cdot \Delta x^2 - \frac{1}{3!} \cdot \frac{d^3 f}{dx^3}(\tilde{x}) \cdot \Delta x^3 + \ldots\right)$$

$$= 2 \cdot \frac{df}{dx}(\tilde{x}) \cdot \Delta x + \frac{2}{3!} \cdot \frac{d^3 f}{dx^3}(\tilde{x}) \cdot \Delta x^3 + \ldots \quad .$$

Truncation after the first derivative term yields the scalar univariate version of the central finite difference quotient. For small values of Δx the truncation error is dominated by the value of the Δx^3 term which implies that only accuracy up to the order of Δx^2 (second-order accuracy) can be expected, i.e,

$$\frac{df}{dx}(\tilde{x}) = \frac{f(\tilde{x}+\Delta x) - f(\tilde{x}-\Delta x) + O(\Delta x^3)}{2\Delta x} = \frac{f(\tilde{x}+\Delta x) - f(\tilde{x}-\Delta x)}{2\Delta x} + O(\Delta x^2).$$

Finite Differences

Issues with Floating-Point Arithmetic, e.g, $y = f(x) = 3x^2$

Picking a "suitable perturbation" Δx may turn out tricky, e.g, for forward finite differences in single precision floating-point arithmetic (6 significant digits) ...

```
#include<iostream>

float f(float x) { return 3*x*x; }

int main() {
  float x=1;
  for (float dx=1;dx>1e-10;dx/=10) {
    std::cout << x << ", "
              << f(x+dx) << ", "
              << f(x) << ", "
              << f(x+dx)-f(x) << ", "
              << dx << ", "
              << (f(x+dx)-f(x))/dx
              << std::endl;
  }
}
```

```
1, 12, 3, 9, 1, 9
1, 3.63, 3, 0.63, 0.1, 6.3
1, 3.0603, 3, 0.0602999, 0.01, 6.02999
1, 3.006, 3, 0.00600338, 0.001, 6.00338
1, 3.0006, 3, 0.0006001, 0.0001, 6.001
1, 3.00006, 3, 6.00815e-05, 1e-05, 6.00815
1, 3.00001, 3, 5.72205e-06, 1e-06, 5.72205
1, 3, 3, 9.53674e-07, 1e-07, 9.53674
1, 3, 3, 0, 1e-08, 0
1, 3, 3, 0, 1e-09, 0
```

Wake Up!

Is

$$\frac{f(x+42)-f(x-42)}{84}$$

a good approximation of the first derivative of $f(x)=x^2$ at $x=10^{-9}$?

Second derivatives can be approximated as derivatives of [a finite difference approximation of] f', i.e, $f''(\tilde{x}) \approx \ldots$

$$\frac{f'(\tilde{x}+\Delta x, \Delta x) - f'(\tilde{x}-\Delta x, \Delta x)}{2 \cdot \Delta x} = \frac{f(\tilde{x}+2\cdot\Delta x) - 2\cdot f(\tilde{x}) + f(\tilde{x}-2\cdot\Delta x)}{4\cdot\Delta x^2}$$

The first expression yields a natural approach to implementing second-order finite differences by perturbing the gradient driver.

Accuracy suffers from the need to square (the small) Δx. A perturbation of

$$\Delta x = \begin{cases} \sqrt{\sqrt{\epsilon}} & \tilde{x} = 0 \\ \sqrt{\sqrt{\epsilon}} \cdot |\tilde{x}| & \tilde{x} \neq 0 \end{cases}$$

with machine epsilon ϵ dependent on the floating-point precision typically yields a reasonable compromise between accuracy and numerical stability.

Outline

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

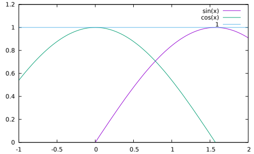
- sample code $y = f(x, p, w)$ (sigmoidal smoothing)
- first derivative $f'(\tilde{x}) = \frac{df}{dx}(\tilde{x}, p, w)$ by finite differences
- second derivative $f''(\tilde{x}) = \frac{d^2 f}{dx^2}(\tilde{x}, p, w)$ by finite differences
- Newton's method for $f(x(p, w), p, w) = 0$
- Newton's method for $\min_{x=x(p,w)} f(x, p, w)$

Sample Code

Potentially Discontinuous Function

Let $\tilde{f} : \boldsymbol{R} \times \boldsymbol{R} \rightarrow \boldsymbol{R}$ be defined as

$$\tilde{f}(x,p) = \begin{cases} f_1(x) & x < p \\ f_2(x) & x \geq p\,. \end{cases}$$



with differentiable univariate scalar f_1 and f_2.

Depending on the choice of f_1 and f_2 the function $\tilde{f}$ can be nondifferentiable or even discontinuous at $x = p$.

Examples:

- $f_1 = \cos,\ f_2 = \sin \Rightarrow$ discontinuous at $x = p = 1$
- $f_1 = \cos,\ f_2 = \sin \Rightarrow$ nondifferentiable at $x = p = \frac{\pi}{4}$
- $f_1 = 1,\ f_2 = \cos \Rightarrow$ differentiable at $x = p = 0$

Sample Code

Sigmoidal Smoothing

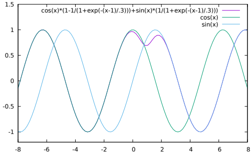
Sigmoidal smoothing replaces $\tilde{f}$ with $f : \boldsymbol{R} \times \boldsymbol{R}^2 \rightarrow \boldsymbol{R}$ defined as

$$f(x,p,w) = (1-\sigma(x,p,w)) \cdot f_1(x) + \sigma(x,p,w) \cdot f_2(x) \; ,$$

where

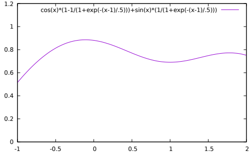
$$\sigma(x,p,w) = \frac{1}{1+e^{-\frac{x-p}{w}}}$$

and both p and w are passive.

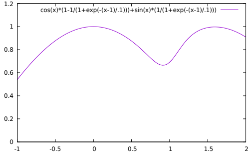


$w = 0.3$

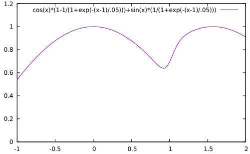
Example: $f_1 = \cos$, $f_2 = \sin$ at $p = 1$



$w = 0.5$



$w = 0.1$



$w = 0.05$

```
#pragma once

#include <cmath>

template<typename T, typename PT>
void f(T &x, const PT &p)
{
  if (x<p)
    x=sin(x);
  else
    x=cos(x);
}
```

```
#pragma once

#include <cmath>

template<typename T, typename PT>
void f(T &x, const PT &p, const PT &w)
{
  T a=sin(x);
  T b=cos(x);
  T c=1./(1.+exp(-(x-p)/w));
  x=a*(1-c)+b*c;
}
```

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {
}
```

```
./g_cfd.exe 1 1 0.1
0.690887 -0.903506
```

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {


  T xph=x+h; f(xph,p,w); // perturb to "right"



}
```

```
./g_cfd.exe 1 1 0.1
0.690887 -0.903506
```

Case Study

$f'(\tilde{x}) = \frac{df}{dx}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {


  T xph=x+h; f(xph,p,w); // perturb to "right"
  T xmh=x-h; f(xmh,p,w); // perturb to "left"


}
```

```
./g_cfd.exe 1 1 0.1
0.690887 -0.903506
```

Case Study

$f'(\tilde{x}) = \frac{df}{dx}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {


  T xph=x+h; f(xph,p,w); // perturb to "right"
  T xmh=x-h; f(xmh,p,w); // perturb to "left"
  dx=(xph-xmh)/(2*h); // finite difference quotient

}
```

```
./g_cfd.exe 1 1 0.1
0.690887 -0.903506
```

$f'(\tilde{x}) = \frac{df}{dx}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {
  T h=(x==0) ? sqrt(std::numeric_limits<T>::epsilon()) // perturbation
               : sqrt(std::numeric_limits<T>::epsilon())*fabs(x);
  T xph=x+h; f(xph,p,w); // perturb to "right"
  T xmh=x-h; f(xmh,p,w); // perturb to "left"
  dx=(xph-xmh)/(2*h); // finite difference quotient

}
```

```
./g_cfd.exe 1 1 0.1
0.690887 -0.903506
```

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {
  T h=(x==0) ? sqrt(std::numeric_limits<T>::epsilon()) // perturbation
             : sqrt(std::numeric_limits<T>::epsilon())*fabs(x);
  T xph=x+h; f(xph,p,w); // perturb to "right"
  T xmh=x-h; f(xmh,p,w); // perturb to "left"
  dx=(xph-xmh)/(2*h); // finite difference quotient
  f(x,p,w); // unperturbed function value
}
```

```
./g_cfd.exe 1 1 0.1
0.690887 -0.903506
```

Case Study

$f''(\tilde{x})) = \frac{d^2 f}{dx^2}(\tilde{x}, p, w)$ by Central Finite Differences

Software and Tools for Computational Engineering
RWTH AACHEN UNIVERSITY

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {







}

template<typename T, typename PT> // driver
void second_derivative(T &x, const PT &p, const PT &w, T &dx, T &ddx) {







}
```

```
./h_cfd.exe 1 1 0.1
0.690887 -0.903506 -7.59975
```

Case Study

$f''(\tilde{x})) = \frac{d^2 f}{dx^2}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {


  T xph=x+h; f(xph,p,w); // perturb function to "right"



}

template<typename T, typename PT> // driver
void second_derivative(T &x, const PT &p, const PT &w, T &dx, T &ddx) {


  T xph=x+h, dxph; first_derivative(xph,p,w,dxph); // perturb derivative to "right"



}
```

```
./h_cfd.exe 1 1 0.1
0.690887 -0.903506 -7.59975
```

$f''(\tilde{x})) = \frac{d^2 f}{dx^2}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {


  T xph=x+h; f(xph,p,w); // perturb function to "right"
  T xmh=x-h; f(xmh,p,w); // perturb function to "left"


}

template<typename T, typename PT> // driver
void second_derivative(T &x, const PT &p, const PT &w, T &dx, T &ddx) {


  T xph=x+h, dxph; first_derivative(xph,p,w,dxph); // perturb derivative to "right"
  T xmh=x-h, dxmh; first_derivative(xmh,p,w,dxmh); // perturb derivative to "left"


}
```

```
./h_cfd.exe 1 1 0.1
0.690887 -0.903506 -7.59975
```

Case Study

$f''(\tilde{x})) = \frac{d^2 f}{dx^2}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {


  T xph=x+h; f(xph,p,w); // perturb function to "right"
  T xmh=x-h; f(xmh,p,w); // perturb function to "left"
  dx=(xph-xmh)/(2*h); // finite difference quotient

}

template<typename T, typename PT> // driver
void second_derivative(T &x, const PT &p, const PT &w, T &dx, T &ddx) {


  T xph=x+h, dxph; first_derivative(xph,p,w,dxph); // perturb derivative to "right"
  T xmh=x-h, dxmh; first_derivative(xmh,p,w,dxmh); // perturb derivative to "left"
  ddx=(dxph-dxmh)/(2*h); // finite difference quotient

}
```

```
./h_cfd.exe 1 1 0.1
0.690887 -0.903506 -7.59975
```

Case Study

$f''(\tilde{x})) = \frac{d^2 f}{dx^2}(\tilde{x}, p, w)$ by Central Finite Differences

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {
  T h=(x==0) ? sqrt(sqrt(std::numeric_limits<T>::epsilon())) // perturbation
        : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(x);
  T xph=x+h; f(xph,p,w); // perturb function to "right"
  T xmh=x-h; f(xmh,p,w); // perturb function to "left"
  dx=(xph-xmh)/(2*h); // finite difference quotient

}

template<typename T, typename PT> // driver
void second_derivative(T &x, const PT &p, const PT &w, T &dx, T &ddx) {
  T h=(x==0) ? sqrt(sqrt(std::numeric_limits<T>::epsilon())) // perturbation
        : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(x);
  T xph=x+h, dxph; first_derivative(xph,p,w,dxph); // perturb derivative to "right"
  T xmh=x-h, dxmh; first_derivative(xmh,p,w,dxmh); // perturb derivative to "left"
  ddx=(dxph-dxmh)/(2*h); // finite difference quotient

}
```

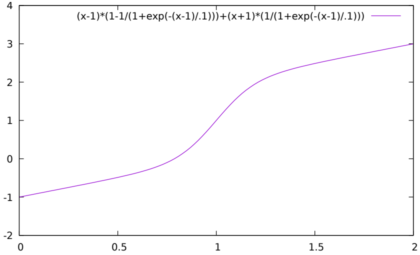
```
./h_cfd.exe 1 1 0.1
0.690887 -0.903506 -7.59975
```

```
template<typename T, typename PT> // driver
void first_derivative(T &x, const PT &p, const PT &w, T &dx) {
  T h=(x==0) ? sqrt(sqrt(std::numeric_limits<T>::epsilon())) // perturbation
        : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(x);
  T xph=x+h; f(xph,p,w); // perturb function to "right"
  T xmh=x-h; f(xmh,p,w); // perturb function to "left"
  dx=(xph-xmh)/(2*h); // finite difference quotient
  f(x,p,w); // unperturbed function value
}

template<typename T, typename PT> // driver
void second_derivative(T &x, const PT &p, const PT &w, T &dx, T &ddx) {
  T h=(x==0) ? sqrt(sqrt(std::numeric_limits<T>::epsilon())) // perturbation
        : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(x);
  T xph=x+h, dxph; first_derivative(xph,p,w,dxph); // perturb derivative to "right"
  T xmh=x-h, dxmh; first_derivative(xmh,p,w,dxmh); // perturb derivative to "left"
  ddx=(dxph-dxmh)/(2*h); // finite difference quotient
  first_derivative(x,p,w,dx); // unperturbed first derivative
}
```

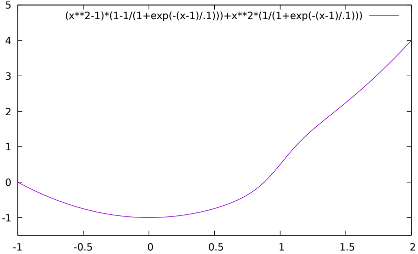
```
./h_cfd.exe 1 1 0.1
0.690887 -0.903506 -7.59975
```

Let $p=1,\ w=0.1,\ f_1(x)=x-1,\ f_2(x)=x+1,\ x^0=2.$




```
./newton_root.exe 2 1 0.1 1e-8
x=-0.997188; f(x)=-1.99719; f'(x)=1
x=1; f(x)=1; f'(x)=6
x=0.833333; f(x)=0.151072; f'(x)=3.67259
x=0.792199; f(x)=0.0147027; f'(x)=2.9775
x=0.787261; f(x)=0.000185824; f'(x)=2.90257
x=0.787197; f(x)=3.06842e-08; f'(x)=2.90161
x=0.787197; f(x)=7.75762e-16; f'(x)=2.90161
```

Let $p = 1$, $w = 0.1$, $f_1(x) = x^2 - 1$, $f_2(x) = x^2$, $x^0 = 2$.




```
./newton_min.exe 2 1 0.1 1e-8
x=-0.004777; f(x)=-0.999934; f'(x)=-0.00912121; f''(x)=2.00441;
x=-0.000226418; f(x)=-0.999955; f'(x)=9.39928e-08; f''(x)=1.9626;
x=-0.000226466; f(x)=-0.999955; f'(x)=-2.42032e-09; f''(x)=1.99767;
```

Outline

Summary

Motivation

Essential Calculus

- Terminology
- Taylor Series
- Linearization
- Chain Rule

Finite Differences

- Derivation
- Accuracy
- Issues with Floating-Point Arithmetic
- Second (and Higher) Order

Case Study

- Sample Code; Sigmoidal Smoothing
- Finite Differences
- Newton's Method

Summary