Algorithmic Differentiation I

First Directional Derivatives of Univariate Scalar Functions

Uwe Naumann

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

RWTH Aachen University

Contents

Outline

Objective and Learning Outcomes

Recall

First-Order Tangents
- Finite Difference Approximation
- dco/c++

Second-Order Tangents
- Finite Difference Approximation
- dco/c++

Higher-Order Tangents

Summary and Next Steps

Objective

- Introduction to first- and second-order algorithmic differentiation of univariate scalar functions with dco/c++

Learning Outcomes

- You will understand
  - tangent-mode algorithmic differentiation
  - finite difference approximation of directional derivatives

  of univariate scalar functions.

- You will be able to
  - use dco/c++ for the computation of derivatives of arbitrary order for implementation of univariate scalar functions as C++ programs
  - compare the results with approximations computed by finite differences

Outline

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

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

Outline

First-order tangents

$$f^{(1)} : \boldsymbol{R} \times \boldsymbol{R} \to \boldsymbol{R} : \quad y^{(1)} = f^{(1)}(x, x^{(1)})$$

of

$$f : \boldsymbol{R} \to \boldsymbol{R} : \quad y = f(x)$$

are first directional derivatives, that is, they are scaled first derivatives of f, i.e,

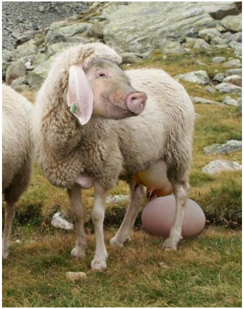
$$y^{(1)} = f^{(1)}(x, x^{(1)}) \equiv f'(x) \cdot x^{(1)} \ ,$$

where

$$f'(x) \equiv \frac{df}{dx}(x) \in \boldsymbol{R} \ .$$

Superscripts $*^{(1)}$ are used to denote first-order tangents.

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



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.

Note that the linearization of a 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.

Finite differences can be used to approximate tangents at a given point $\tilde{x} \in \boldsymbol{R}$.

$$
\begin{aligned}
f'(\tilde{x}) \cdot x^{(1)} &\approx_1 \frac{f(\tilde{x} + \Delta x \cdot x^{(1)}) - f(\tilde{x})}{\Delta x} && \text{(forward)} \\
&\approx_1 \frac{f(\tilde{x}) - f(\tilde{x} - \Delta x \cdot x^{(1)})}{\Delta x} && \text{(backward)} \\
&\approx_2 \frac{f(\tilde{x} + \Delta x \cdot x^{(1)}) - f(\tilde{x} - \Delta x \cdot x^{(1)})}{2 \cdot \Delta x} && \text{(central)}
\end{aligned}
$$

where $\Delta x = \Delta x(\tilde{x}) \in \boldsymbol{R}$ is a suitable perturbation.

Forward and backward finite differences exhibit first-order accuracy ($\approx_1$; error scales with Δx) while central finite differences turn out to be second-order accurate ($\approx_2$; error scales with Δx^2).

dco/c++ (derivative code by overloading in C++)[1]

- is an algorithmic differentiation tools developed by STCE in collaboration with the Numerical Algorithms Group Ltd. (`nag.co.uk`)
- supports the (semi-)automatic generation of derivative code of arbitrary order for numerical simulations written in C++.
- uses overloading of arithmetic operators (e.g, `+,\`) and elementary functions (e.g, `sin`, `log`) for a set of custom data types (e.g, `gt1s`) to implement algorithmic differentiation.
- is actively used by numerical simulation projects world-wide.

[1] Lotz, Leppkes, Naumann: dco/c++: Derivative Code by Overloading in C++. Aachener Informatik-Berichte AIB-2011-06.

First Derivatives with dco/c++

Tangent / Forward Mode

```
template<typename T> void f(const T& x, T& y);

template<typename T>
void f_t(const T& x_v, T& y_v, T& dydx) {
  using DCO_T=typename dco::gt1s<T>::type;
  DCO_T x,y;
  dco::value(x)=x_v; dco::derivative(x)=1;
  f(x,y);
  y_v=dco::value(y); dydx=dco::derivative(y);
}

int main() {
  double x=1, y, dydx;
  f_t(x,y,dydx); ...
  return 0;
}
```

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

```
template<typename T> void f(const T& x, T& y);

template<typename T>
void f_cfd(const T& x, T& y, T& dydx) {
  f(x,y);
  T dx=fabs(x)<1 ? sqrt(std::numeric_limits<T>::epsilon())
       : sqrt(std::numeric_limits<T>::epsilon())*fabs(x);
  T yp,ym;
  f(x+dx,yp); f(x-dx,ym);
  dydx=(yp-ym)/(2*dx);
}

int main() {
  double x=1, y, dydx;
  f_cfd(x,y,dydx); ...
  return 0;
}
```

Outline

Second-order tangents

$$f^{(1,2)} : \boldsymbol{R} \times \boldsymbol{R} \times \boldsymbol{R} \to \boldsymbol{R} : \quad y^{(1,2)} = f^{(1,2)}(x, x^{(1)}, x^{(2)})$$

of

$$f : \boldsymbol{R} \to \boldsymbol{R} : \quad y = f(x)$$

are first directional derivatives of $f^{(1)}$ in direction $x^{(2)}$, that is, they are scaled first derivatives of $f^{(1)}$ wrt. x, i.e,

$$y^{(1,2)} = f^{(1,2)}(x, x^{(1)}, x^{(2)}) \equiv f''(x) \cdot x^{(1)} \cdot x^{(2)} \ ,$$

where

$$f''(x) \equiv \frac{df'}{dx}(x) = \frac{d^2 f}{dx^2}(x) \in \boldsymbol{R} \ .$$

Superscripts $*^{(2)}$ are used to denote first-order tangents of $f^{(1)}$.

Second-order tangents carry superscripts $*^{(1,2)} \equiv *^{(1)^{(2)}}$.

Second derivatives can be approximated as derivatives of [a finite difference approximation of] f'.

$$f''(x) \approx \frac{f'(x+\Delta x, \Delta x) - f'(x-\Delta x, \Delta x)}{2 \cdot \Delta x}$$

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

The first expression yields a natural approach to implementing second-order finite differences.

```
template<typename T> void f(const T& x, T& y);

template<typename T> void f_cfd(const T& x, T& y, T& dydx);

template<typename T>
void f_cfd_cfd(const T& x, T& y, T& dydx, T& ddydxx) {
  f_cfd(x,y,dydx);
  T dx=fabs(x)<1 ? sqrt(sqrt(std::numeric_limits<T>::epsilon()))
       : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(x);
  T dummy,dydxp,dydxm;
  f_cfd(x+dx,dummy,dydxp); f_cfd(x-dx,dummy,dydxm);
  ddydxx=(dydxp-dydxm)/(2*dx);
}

int main() {
  double x=1, y, dydx, ddydxx;
  f_cfd_cfd(x,y,dydx,ddydxx);
  // ...
  return 0;
}
```

Investigate numerical effects due to finite precision floating-point arithmetic!

Second-Order Tangent / Forward Mode

```
template<typename T> void f(const T& x, T& y);

template<typename T> void f_t(const T& x_v, T& y_v, T& dydx);

template<typename T>
void f_t_t(const T& x_v, T& y_v, T& dydx_v, T& ddydxx) {
  using DCO_T=typename dco::gt1s<T>::type;
  DCO_T x,y,dydx;
  dco::value(x)=x_v; dco::derivative(x)=1;
  f_t(x,y,dydx);
  y_v=dco::value(y);
  dydx_v=dco::value(dydx);
  ddydxx=dco::derivative(dydx);
}

int main() {
  double x=1, y, dydx, ddydxx;
  f_t_t(x,y,dydx,ddydxx);
  // ...
  return 0;
}
```

Outline

The above generalizes to third- and higher-order tangents:

$$
\begin{aligned}
y^{(1,2,3)} &= f^{(1,2,3)}(x, x^{(1)}, x^{(2)}, x^{(3)}) \\
&\equiv f'''(x) \cdot x^{(1)} \cdot x^{(2)} \cdot x^{(3)} \\
\ldots
\end{aligned}
$$

where

$$
f'''(x) \equiv \frac{df''}{dx}(x) = \frac{d^3 f}{dx^3}(x) \in \boldsymbol{R} \quad \text{etc.}
$$

dco/c++ supports tangents of arbitrary order through nesting of its base-type-generic first-order tangent type `gt1s<BASE_TYPE>`.

Finite differences for tangents of third and higher order can rarely be trusted because of numerical issues due to finite precision floating-point arithmetic.

Outline

Summary

- Finite differences for tangents of arbitrary order suffers from numerical issues due to finite precision floating-point arithmetic.
- dco/c++ delivers tangents of arbitrary order with machine accuracy.

Next Steps

- Run sample code and compare results.
- Check out dco/c++ for higher derivatives.
- Continue the course to find out more ...