Algorithmic Differentiation IV

Second-Order Tangents of Multivariate Scalar Functions

Uwe Naumann

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

RWTH Aachen University

Contents

Outline

Objective

- Introduction to the evaluation of Hessians using second-order finite difference approximation and tangent-of-tangent mode algorithmic differentiation (AD).

Learning Outcomes

- You will understand
  - second-order finite differences
  - second-order tangents.
- You will be able to implement
  - second-order finite differences
  - second-order tangents with dco/c++.

Outline

The Hessian

$$f'' = f''(\mathbf{x}) \equiv \frac{d^2 f}{d\mathbf{x}^2}(\mathbf{x}) = \left(\frac{d^2 f}{dx_i dx_j}(\mathbf{x}) \right) \in \boldsymbol{R}^{n \times n}$$

of a twice continuously differentiable multivariate scalar function $f : \boldsymbol{R}^n \to \boldsymbol{R}$ can be approximated at a given point $\tilde{\mathbf{x}} \in \boldsymbol{R}^n$ as a (central) finite difference approximation of the Jacobian of a (central) finite difference approximation of the gradient

$$f' = f'(\mathbf{x}) \equiv \frac{df}{d\mathbf{x}}(\mathbf{x}) = \left(\frac{df}{dx_i}(\mathbf{x}) \right) \in \boldsymbol{R}^n$$

of f :

$$\frac{d^2 f}{dx_i dx_j}(\tilde{\mathbf{x}}) \approx \frac{\frac{df}{dx_i}(\tilde{\mathbf{x}} + \mathbf{e}_j \cdot \Delta x_j) - \frac{df}{dx_i}(\tilde{\mathbf{x}} - \mathbf{e}_j \cdot \Delta x_j)}{2 \cdot \Delta x_j} .$$

$\mathbf{e}_j$ denotes the j-th Cartesian basis vector in $\boldsymbol{R}^n$.

Alternatively, the Hessian $f'' = f''(\mathbf{x})$ of $f : \boldsymbol{R}^n \to \boldsymbol{R}$ can be approximated at a given point $\tilde{\mathbf{x}} \in \boldsymbol{R}^n$ as a (central) finite difference approximation of the Jacobian of a tangent mode accumulation of the gradient $f' = f'(\mathbf{x})$ of f :

$$\frac{d^2 f}{dx_i dx_j}(\tilde{\mathbf{x}}) \approx \frac{\frac{df}{dx_i}(\tilde{\mathbf{x}} + \mathbf{e}_j \cdot \Delta x_j) - \frac{df}{dx_i}(\tilde{\mathbf{x}} - \mathbf{e}_j \cdot \Delta x_j)}{2 \cdot \Delta x_j} ,$$

where

$$\frac{df}{dx_i}(\tilde{\mathbf{x}}) = \frac{df}{d\mathbf{x}}(\tilde{\mathbf{x}}) \cdot \mathbf{x}_i^{(1)}$$

for $\mathbf{x}_i^{(1)} = \mathbf{e}_i$.

The computational cost of accumulating the Hessian in either finite-difference-of-finite-difference or finite-difference-of-tangent (or tangent-of-finite-difference) modes is $O(n^2) \cdot \mathrm{Cost}(f)$.

Finite Difference Approximation

Implementation


```
1  #include "f.hpp"
2  #include "Eigen/Dense"
3  #include <limits>
4
5  template<typename T, int N>
6  void f_[cfd|t](Eigen::Matrix<T,N,1>& x, T& y, Eigen::Matrix<T,N,1>& dydx);
7
8  template<typename T, int N>
9  void f_[cfd|t]_cfd(Eigen::Matrix<T,N,1>& x, T& y, Eigen::Matrix<T,N,1>& dydx,
10                    Eigen::Matrix<T,N,N>& ddydxx) {
11   auto n=x.size();
12   for (auto i=0;i<n;i++) {
13     T dx=fabs(x(i))<1 ? sqrt(sqrt(std::numeric_limits<T>::epsilon()))
14           : sqrt(sqrt(std::numeric_limits<T>::epsilon())*fabs(x(i)));
15     T yd;
16     Eigen::Matrix<T,N,1> dydxp(n), dydxm(n);
17     x(i)+=dx; f_[cfd|t](x,yd,dydxp); x(i)-=2*dx; f_[cfd|t](x,yd,dydxm); x(i)+=dx;
18     for (auto j=0;j<n;j++) ddydxx(j,i)=(dydxp(j)-dydxm(j))/(2*dx);
19   }
20   f_[cfd|t](x,y,dydx);
21 }
```

Outline

A second derivative code $f^{(1,2)} : \boldsymbol{R}^n \times \boldsymbol{R}^n \times \boldsymbol{R}^n \times \boldsymbol{R}^n \to \boldsymbol{R} \times \boldsymbol{R} \times \boldsymbol{R} \times \boldsymbol{R}$, generated in tangent-of-tangent mode of AD computes

$$\begin{pmatrix} y \\ y^{(2)} \\ y^{(1)} \\ y^{(1,2)} \end{pmatrix} = f^{(1,2)}(\mathbf{x}, \mathbf{x}^{(2)}, \mathbf{x}^{(1)}, \mathbf{x}^{(1,2)})$$

as

$$\begin{pmatrix} y \\ y^{(2)} \\ y^{(1)} \\ y^{(1,2)} \end{pmatrix} := \begin{pmatrix} f(\mathbf{x}) \\ f'(\mathbf{x}) \cdot \mathbf{x}^{(2)} \\ f'(\mathbf{x}) \cdot \mathbf{x}^{(1)} \\ {\mathbf{x}^{(1)}}^T \cdot f''(\mathbf{x}) \cdot \mathbf{x}^{(2)} + f'(\mathbf{x}) \cdot \mathbf{x}^{(1,2)} \end{pmatrix} .$$

Note: In context of chain rule both $y^{(1)}$ and $y^{(2)}$ required and non-vanishing $\mathbf{x}^{(1,2)}$; $f''(\mathbf{x})^T = f''(\mathbf{x})$ as f twice continuously differentiable

The computational cost of accumulating the Hessian in tangent-of-tangent mode is $O(n^2) \cdot \mathrm{Cost}(f)$.

Algorithmic differentiation of the first-order tangent

$$\begin{pmatrix} y \\ y^{(1)} \end{pmatrix} = \begin{pmatrix} f(\mathbf{x}) \\ f'(\mathbf{x}) \cdot \mathbf{x}^{(1)} \end{pmatrix}$$

in tangent mode yields

$$\begin{pmatrix} y^{(2)} \\ y^{(1,2)} \end{pmatrix} \equiv \frac{d \begin{pmatrix} y \\ y^{(1)} \end{pmatrix}}{d \begin{pmatrix} \mathbf{x} \\ \mathbf{x}^{(1)} \end{pmatrix}} \cdot \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{x}^{(1,2)} \end{pmatrix} = \begin{pmatrix} \frac{dy}{d\mathbf{x}} \cdot \mathbf{x}^{(2)} \left[+ \frac{dy}{d\mathbf{x}^{(1)}} \cdot \mathbf{x}^{(1,2)} = 0 \right] \\ \frac{dy^{(1)}}{d\mathbf{x}} \cdot \mathbf{x}^{(2)} + \frac{dy^{(1)}}{d\mathbf{x}^{(1)}} \cdot \mathbf{x}^{(1,2)} \end{pmatrix}$$

implying with[1] $y^{(1)} = \mathbf{x}^{(1)^T} \cdot f'(\mathbf{x})^T$ and $f''(\mathbf{x})^T = f''(\mathbf{x})$

$$\begin{pmatrix} y^{(2)} \\ y^{(1,2)} \end{pmatrix} = \begin{pmatrix} f'(\mathbf{x}) \cdot \mathbf{x}^{(2)} \\ \mathbf{x}^{(1)^T} \cdot f''(\mathbf{x}) \cdot \mathbf{x}^{(2)} + f'(\mathbf{x}) \cdot \mathbf{x}^{(1,2)} \end{pmatrix} .$$

[1] Note differentiation of $f'(\mathbf{x})^T : \boldsymbol{R}^n \to \boldsymbol{R}^n$ instead of $f'(\mathbf{x}) : \boldsymbol{R}^n \to \boldsymbol{R}^{1 \times n}$.

Tangents of [Approximate] Tangents

Implementation with dco/c++


```
1  #include "dco.hpp"
2  #include "Eigen/Dense"
3
4  template<typename T, int N>
5  void f_[t|cfd](const Eigen::Matrix<T,N,1>& x_v, T& y_v, Eigen::Matrix<T,N,1>& dydx);
6
7  template<typename T, int N>
8  void f_[t|cfd]_t(const Eigen::Matrix<T,N,1>& x_v, T& y_v, Eigen::Matrix<T,N,1>& dydx_v,
9                   Eigen::Matrix<T,N,N>& ddydxx) {
10   using DCO_T=typename dco::gt1s<T>::type;
11   auto n=x_v.size();
12   Eigen::Matrix<DCO_T,N,1> x(n), dydx(n); DCO_T y=0;
13   for (auto i=0;i<n;i++) x(i)=x_v(i);
14   for (auto i=0;i<n;i++) {
15     dco::derivative(x(i))=1;
16     f_[t|cfd](x,y,dydx);
17     for (auto j=0;j<n;j++) ddydxx(j,i)=dco::derivative(dydx(j));
18     dco::derivative(x(i))=0;
19   }
20   for (auto j=0;j<n;j++) dydx_v(j)=dco::value(dydx(j));
21   y_v=dco::value(y);
22 }
```

Outline

Summary

- Introduction to the evaluation of Hessians using second-order finite difference approximation and tangent-of-tangent mode algorithmic differentiation with dco/c++.

Next Steps

- Download, inspect and play with the sample code.
- Continue the course to find out more ...