Introduction to Algorithmic Differentiation (AD)

Second-(and Higher-) Order AD by Overloading ($f : \boldsymbol{R} \rightarrow \boldsymbol{R}$)

Uwe Naumann

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

RWTH Aachen

Contents

We aim to compute second and higher derivatives

$$f^{[k]} \equiv \frac{d^k f}{dx^k} \ , \ k = 2, 3, \ldots$$

of scalar primal target functions $f(x, \mathrm{p}) : \boldsymbol{R} \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}$ with active $x \in \boldsymbol{R}$ and with passive parameters $\mathrm{p} \in \boldsymbol{R}^{n^*}$.

The primal function is assumed to be sufficiently often (continuously) differentiable at all points of interest.

Whenever possible and without loss of generality, we omit p during the discussion of theory and algorithmics in order to simplify the notation.

This sets the stage for subsequent generalization to multivariate vector functions of the form

$$F(\mathrm{x}, \mathrm{p}) : \boldsymbol{R}^n \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}^m \ .$$

Outline

Second-order tangents (tangents of 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 tangents of $f^{(1)}$ in direction $x^{(2)}$, i.e,

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

where

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

The superscript $*^{(2)}$ is used to denote first-order tangents of $f^{(1)}$.

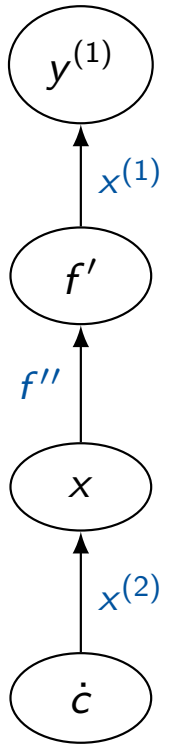
Second-order tangents carry the superscript $*^{(1,2)} \equiv *^{(1)^{(2)}}$.

Differentiation of

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

with respect to x in direction $x^{(2)}$ yields

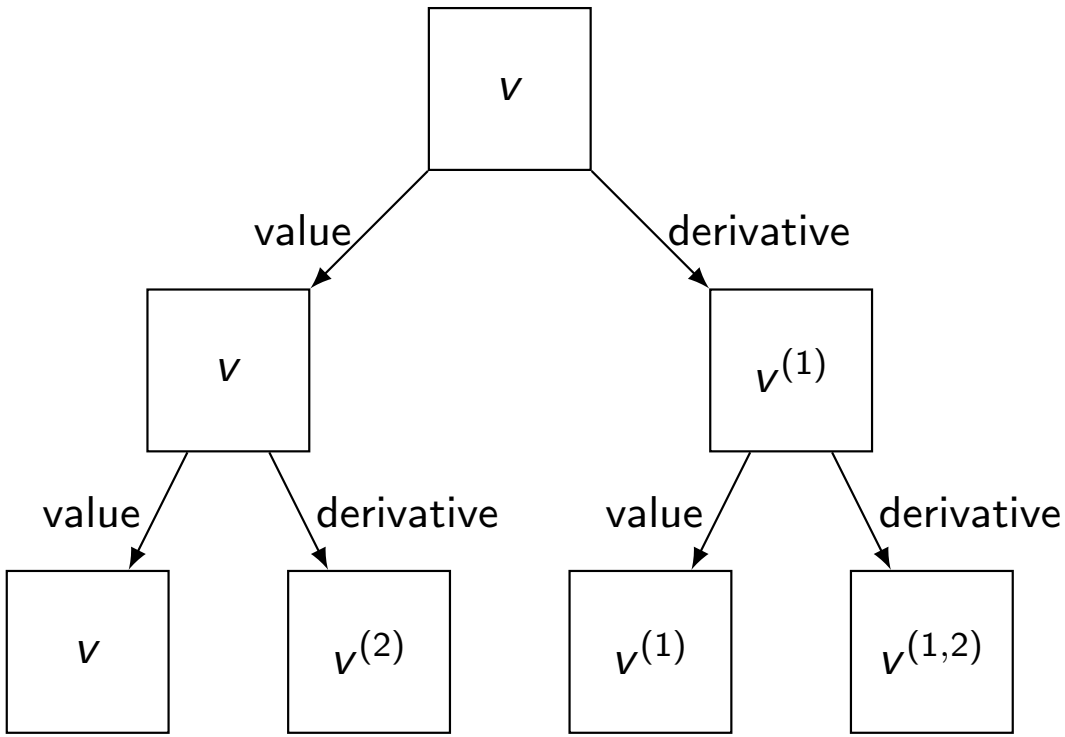
$$\begin{aligned} y^{(1,2)} &= \frac{df^{(1)}}{dx}(x, x^{(1)}) \cdot x^{(2)} = \frac{df'(x) \cdot x^{(1)}}{dx} \cdot x^{(2)} \\ &= x^{(1)} \cdot \frac{df'(x)}{dx} \cdot x^{(2)} = x^{(1)} \cdot f''(x) \cdot x^{(2)} \ . \end{aligned}$$



This formulation assumes no dependence of $x^{(1)}$ on the source context yielding a vanishing first order contribution to $y^{(1,2)}$.

Context-sensitivity will be discussed later for multivariate vector functions.

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



Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6
7
8
9
10
11
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7
8
9
10
11
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8
9
10
11
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9
10
11
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10
11
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10   dco::value(dco::derivative(x))=1.; // seed x^{(1)}
11
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10   dco::value(dco::derivative(x))=1.; // seed x^{(1)}
11   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
12
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10   dco::value(dco::derivative(x))=1.; // seed x^{(1)}
11   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
12   f(x,p,w); // compute augmented primal
13
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10   dco::value(dco::derivative(x))=1.; // seed x^{(1)}
11   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
12   f(x,p,w); // compute augmented primal
13   x_v=dco::value(dco::value(x)); // get primal function value
14
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10   dco::value(dco::derivative(x))=1.; // seed x^{(1)}
11   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
12   f(x,p,w); // compute augmented primal
13   x_v=dco::value(dco::value(x)); // get primal function value
14   dx=dco::value(dco::derivative(x)); // harvest first derivative
15
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
7    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
8    DCO_T x=0; // activate
9    dco::value(dco::value(x))=x_v; // set active argument
10   dco::value(dco::derivative(x))=1.; // seed x^{(1)}
11   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
12   f(x,p,w); // compute augmented primal
13   x_v=dco::value(dco::value(x)); // get primal function value
14   dx=dco::value(dco::derivative(x)); // harvest first derivative
15   ddx=dco::derivative(dco::derivative(x)); // harvest second derivative
16 }
17
18 #include "h_main.hpp" // main
```

Second-Order Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // second derivative driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx, T &ddx) {
6    ...
7  }
8
9  #include "h_main.hpp" // main
```

```
./h_tt.exe 1.1 1 0.5
0.650594 -0.502455 -1.93886
```

Note: second-order finite differences, finite differences of tangents, tangents of finite differences

Wake Up!

State the tangent of tangent AD model for $y = \sin(x)$.

Outline

Second-order adjoints (adjoints of tangents)

$$f_{(2)}^{(1)} : \boldsymbol{R} \times \boldsymbol{R} \times \boldsymbol{R} \to \boldsymbol{R} : \quad x_{(2)} = f_{(2)}^{(1)}(x, x^{(1)}, y_{(2)}^{(1)})$$

of

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

are adjoints of $f^{(1)}$ in direction $y_{(2)}^{(1)}$, i.e,

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

The subscript $*_{(2)}$ is used to denote first-order adjoints of $f^{(1)}$.

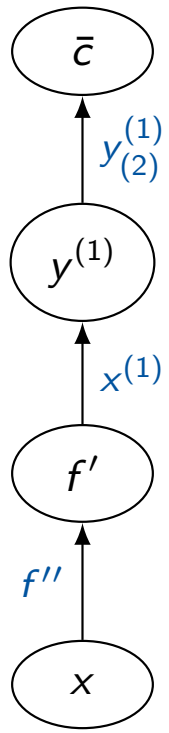
Second-order adjoints carry the mixed super-/subscript $*_{(2)}^{(1)}$.

Adjoint differentiation of

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

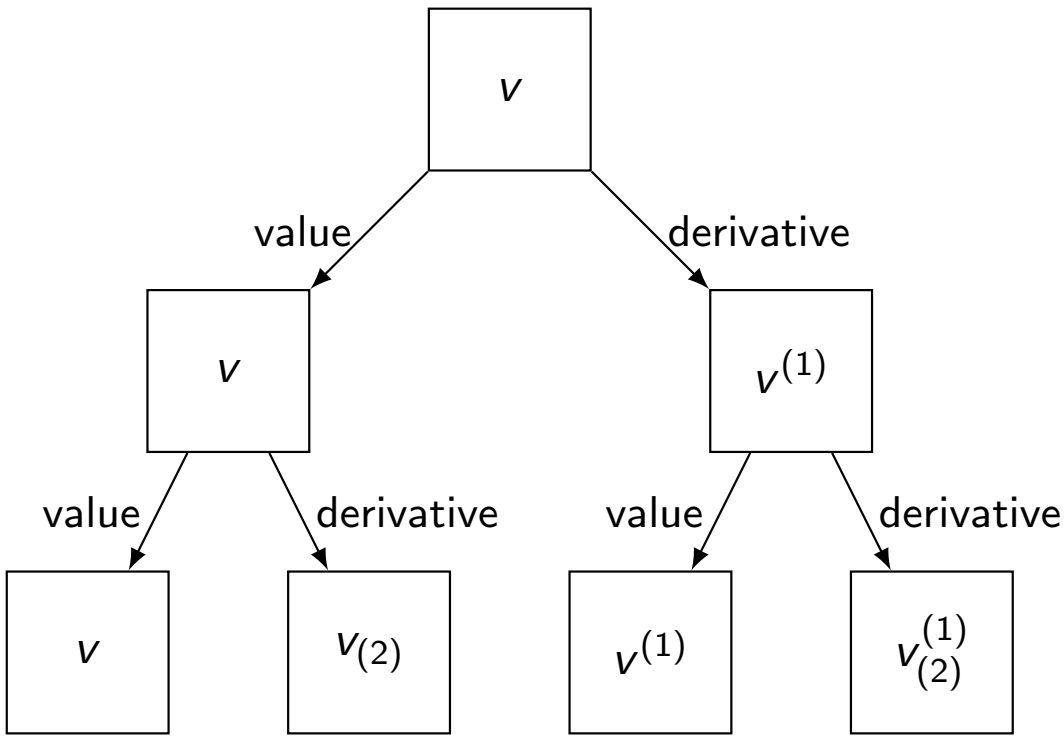
with respect to x in direction $y^{(1)}_{(2)}$ yields

$$\begin{aligned} x_{(2)} &= y^{(1)}_{(2)} \cdot \frac{df^{(1)}}{dx}(x, x^{(1)}) = y^{(1)}_{(2)} \cdot \frac{df'(x) \cdot x^{(1)}}{dx} \\ &= y^{(1)}_{(2)} \cdot x^{(1)} \cdot \frac{df'(x)}{dx} = y^{(1)}_{(2)} \cdot x^{(1)} \cdot f''(x) \, . \end{aligned}$$



This formulation assumes no dependence of $x^{(1)}$ on the source context yielding a vanishing first order contribution to $x_{(2)}$.

$$\begin{pmatrix} y \\ y^{(1)} \\ x_{(2)} \end{pmatrix} = \begin{pmatrix} f(x) \\ f'(x) \cdot x^{(1)} \\ y^{(1)}_{(2)} \cdot x^{(1)} \cdot f''(x) \end{pmatrix}$$



Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver

```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15   x_v=dco::value(dco::value(x)); // get primal function value
16
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15   x_v=dco::value(dco::value(x)); // get primal function value
16   dx=dco::value(dco::derivative(x)); // harvest first derivative
17
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15   x_v=dco::value(dco::value(x)); // get primal function value
16   dx=dco::value(dco::derivative(x)); // harvest first derivative
17   dco::derivative(dco::derivative(x))=1.; // seed x^{(1)}_{(2)}
18
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15   x_v=dco::value(dco::value(x)); // get primal function value
16   dx=dco::value(dco::derivative(x)); // harvest first derivative
17   dco::derivative(dco::derivative(x))=1.; // seed x^{(1)}_{(2)}
18   DCO_BM::global_tape->interpret_adjoint(); // interpret tape
19
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15   x_v=dco::value(dco::value(x)); // get primal function value
16   dx=dco::value(dco::derivative(x)); // harvest first derivative
17   dco::derivative(dco::derivative(x))=1.; // seed x^{(1)}_{(2)}
18   DCO_BM::global_tape->interpret_adjoint(); // interpret tape
19   ddx=dco::derivative(dco::value(x_in)); // harvest second derivative
20
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of tangent driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // tape type
6    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of base type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    dco::value(dco::derivative(x_in))=1.; // seed x^{(1)}
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" tape
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // record x and
12   DCO_BM::global_tape->register_variable(dco::derivative(x_in)); // x^{(1)}
13   DCO_T x=x_in; // read-only active inoutput
14   f(x,p,w); // compute augmented primal
15   x_v=dco::value(dco::value(x)); // get primal function value
16   dx=dco::value(dco::derivative(x)); // harvest first derivative
17   dco::derivative(dco::derivative(x))=1.; // seed x^{(1)}_{(2)}
18   DCO_BM::global_tape->interpret_adjoint(); // interpret tape
19   ddx=dco::derivative(dco::value(x_in)); // harvest second derivative
20   DCO_BTT::remove(DCO_BM::global_tape); // deallocate tape
21 }
```

Adjoints of Tangents with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint of tangent driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
6    ...
7  }
8
9  #include "h_main.hpp" // main
```

```
./h_ta.exe 1.1 1 0.5
0.650594 -0.502455 -1.93886
```

Wake Up!

State the adjoint of tangent AD model for $y = \cos(x)$.

Outline

Second-order adjoints (tangents of adjoints)

$$f_{(1)}^{(2)} : \boldsymbol{R} \times \boldsymbol{R} \times \boldsymbol{R} \to \boldsymbol{R} : \quad x_{(1)}^{(2)} = f_{(1)}^{(2)}(x, y_{(1)}, x^{(2)})$$

of

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

are tangents of $f_{(1)}$ in direction $x^{(2)}$, i.e,

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

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

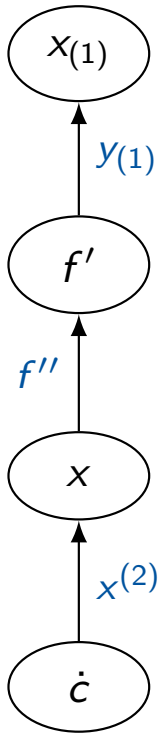
Second-order adjoints carry mixed sub-/superscripts as $*_{(1)}^{(2)}$.

Differentiation of

$$x_{(1)} = f_{(1)}(x, y_{(1)}) \equiv y_{(1)} \cdot f'(x)$$

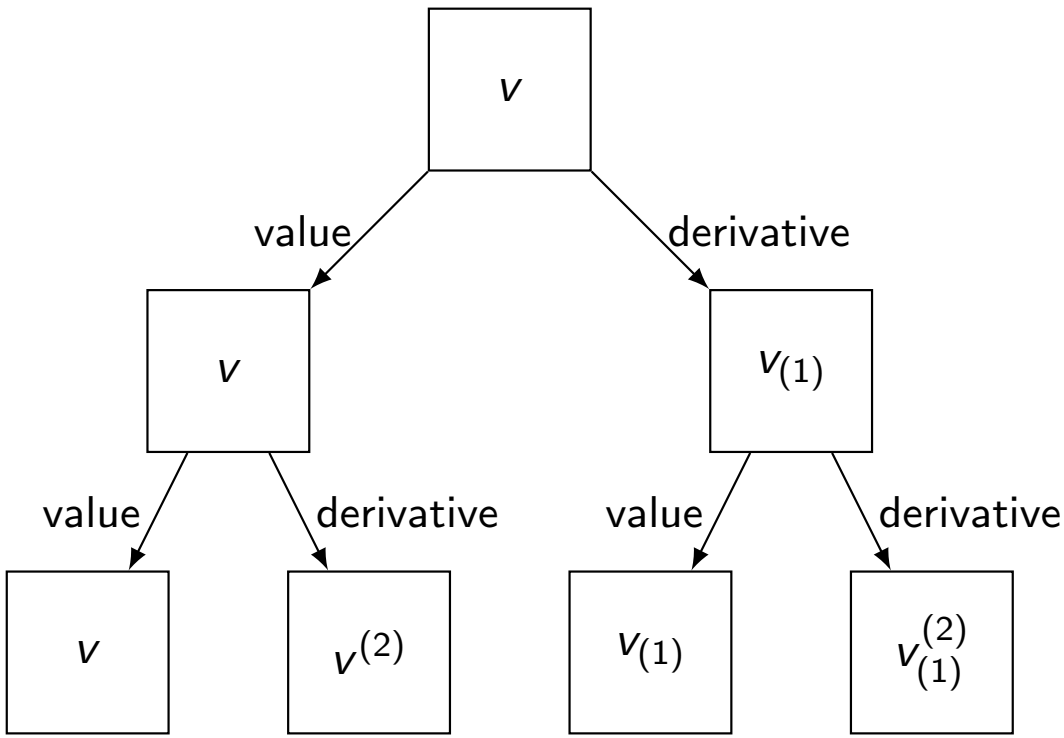
with respect to x in direction $x^{(2)}$ yields

$$\begin{aligned} x_{(1)}^{(2)} &= \frac{df_{(1)}}{dx}(x, y_{(1)}) \cdot x^{(2)} = \frac{dy_{(1)} \cdot f'(x)}{dx} \cdot x^{(2)} \\ &= y_{(1)} \cdot \frac{df'(x)}{dx} \cdot x^{(2)} = y_{(1)} \cdot f''(x) \cdot x^{(2)} \ . \end{aligned}$$



This formulation assumes no dependence of $y_{(1)}$ on source context yielding a vanishing first order contribution to $x_{(1)}^{(2)}$.

$$\begin{pmatrix} y \\ x_{(1)} \\ x_{(1)}^{(2)} \end{pmatrix} = \begin{pmatrix} f(x) \\ y_{(1)} \cdot f'(x) \\ y_{(1)} \cdot f''(x) \cdot x^{(2)} \end{pmatrix}$$



Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver

```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(dco::value(x)); // get primal function value
15
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(dco::value(x)); // get primal function value
15   dx=dco::derivative(dco::value(x)); // harvest first derivative
16
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(dco::value(x)); // get primal function value
15   dx=dco::derivative(dco::value(x)); // harvest first derivative
16   dco::value(dco::derivative(x))=1.; // seed y_{(1)}
17
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver

```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(dco::value(x)); // get primal function value
15   dx=dco::derivative(dco::value(x)); // harvest first derivative
16   dco::value(dco::derivative(x))=1.; // seed y_{(1)}
17   DCO_M::global_tape->interpret_adjoint(); // interpret tape
18
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(dco::value(x)); // get primal function value
15   dx=dco::derivative(dco::value(x)); // harvest first derivative
16   dco::value(dco::derivative(x))=1.; // seed y_{(1)}
17   DCO_M::global_tape->interpret_adjoint(); // interpret tape
18   ddx=dco::derivative(dco::derivative(x_in)); // harvest second derivative
19
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // tangent of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
5    using DCO_T=typename DCO_M::type; // adjoint of base type
6    using DCO_TT=typename DCO_M::tape_t; // tape type
7    DCO_T x_in; // activate
8    dco::value(dco::value(x_in))=x_v; // set active argument
9    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
10   DCO_M::global_tape->register_variable(x_in); // record active argument
11   DCO_T x=x_in; // read-only active input
12   dco::derivative(dco::value(x))=1.; // seed x^{(2)}
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(dco::value(x)); // get primal function value
15   dx=dco::derivative(dco::value(x)); // harvest first derivative
16   dco::value(dco::derivative(x))=1.; // seed y_{(1)}
17   DCO_M::global_tape->interpret_adjoint(); // interpret tape
18   ddx=dco::derivative(dco::derivative(x_in)); // harvest second derivative
19   DCO_TT::remove(DCO_M::global_tape); // deallocate tape
20 }
```

Tangents of Adjoints with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent of adjoint driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
6    ...
7  }
8
9  #include "h_main.hpp" // main
```

```
./h_at.exe 1.1 1 0.5
0.650594 -0.502455 -1.93886
```

Wake Up!

State the tangent of adjoint AD model for $y = 2 \cdot \sqrt{x}$.

Outline

Second-order adjoints (adjoints of adjoints)

$$f_{(1,2)} : \boldsymbol{R} \times \boldsymbol{R} \times \boldsymbol{R} \to \boldsymbol{R} : \quad x_{(2)} = f_{(1,2)}(x, y_{(1)}, x_{(1,2)})$$

of

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

are adjoints of $f_{(1)}$ in direction $x_{(1,2)}$, i.e,

$$x_{(2)} = f_{(1,2)}(x, y_{(1)}, x_{(1,2)}) \equiv x_{(1,2)} \cdot y_{(1)} \cdot f''(x) \ ,$$

The subscript $*_{(2)}$ is used to denote first-order adjoints of $f_{(1)}$.

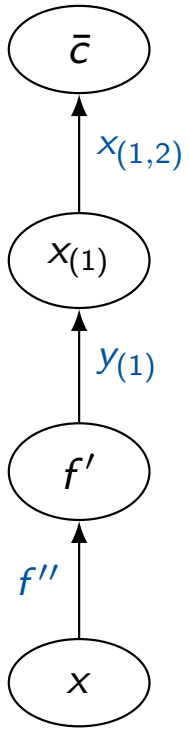
Second-order adjoints carry the subscript $*_{(1,2)} \equiv *_{(1)_{(2)}}$.

Adjoint differentiation of

$$x_{(1)} = f_{(1)}(x, y_{(1)}) \equiv y_{(1)} \cdot f'(x)$$

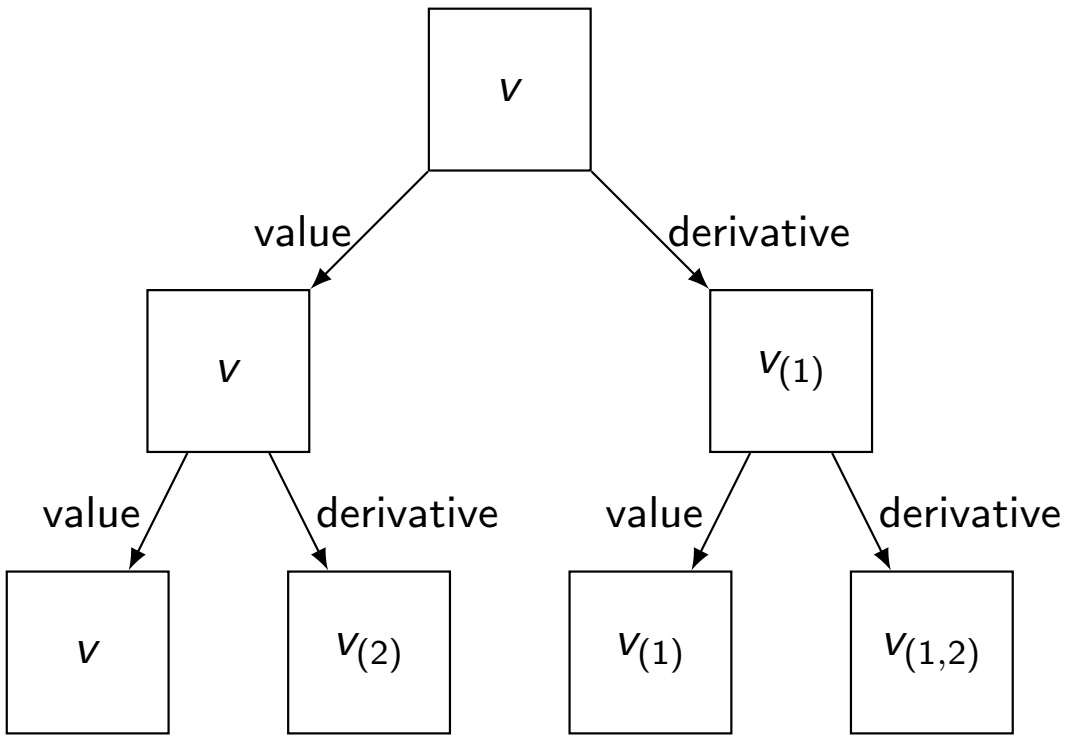
with respect to x in direction $x_{(1,2)}$ yields

$$\begin{aligned} x_{(2)} &= x_{(1,2)} \cdot \frac{df_{(1)}}{dx}(x, y_{(1)}) = x_{(1,2)} \cdot \frac{dy_{(1)} \cdot f'(x)}{dx} \\ &= x_{(1,2)} \cdot y_{(1)} \cdot \frac{df'(x)}{dx} = x_{(1,2)} \cdot y_{(1)} \cdot f''(x) \, . \end{aligned}$$



This formulation assumes no dependence of $y_{(1)}$ on source context yielding a vanishing first order contribution to $x_{(2)}$.

$$\begin{pmatrix} y \\ x_{(1)} \\ x_{(2)} \end{pmatrix} = \begin{pmatrix} f(x) \\ y_{(1)} \cdot f'(x) \\ x_{(1,2)} \cdot y_{(1)} \cdot f''(x) \end{pmatrix}$$



Adjoints of Adjoints with dco/c++

Driver

```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3
4
5
6
7
8
9
10
11
12
13
14     ...
```

Adjoints of Adjoints with dco/c++

Driver

```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4
5
6
7
8
9
10
11
12
13
14     ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5
6
7
8
9
10
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6
7
8
9
10
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7
8
9
10
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8
9
10
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9
10
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(dco::value(x_in))=x_v; // activate
10
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(dco::value(x_in))=x_v; // activate
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" base tape and record ...
11
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver


```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(dco::value(x_in))=x_v; // activate
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" base tape and record ...
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // active argument
12
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver

```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(dco::value(x_in))=x_v; // activate
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" base tape and record ...
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // active argument
12   DCO_M::global_tape=DCO_TT::create(); // "touch" tape and record ...
13
14   ...
```

Adjoints of Adjoints with dco/c++

Driver

```
1  template<typename T, typename PT> // adjoint of adjoint driver
2  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
3    using DCO_BM=typename dco::ga1s<T>; // adjoint (base) mode
4    using DCO_BT=typename DCO_BM::type; // adjoint (base) type
5    using DCO_BTT=typename DCO_BM::tape_t; // base tape type
6    using DCO_M=typename dco::ga1s<DCO_BT>; // adjoint of base mode
7    using DCO_T=typename DCO_M::type; // adjoint of base type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(dco::value(x_in))=x_v; // activate
10   DCO_BM::global_tape=DCO_BTT::create(); // "touch" base tape and record ...
11   DCO_BM::global_tape->register_variable(dco::value(x_in)); // active argument
12   DCO_M::global_tape=DCO_TT::create(); // "touch" tape and record ...
13   DCO_M::global_tape->register_variable(x_in); // active argument
14   ...
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2
3
4
5
6
7
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3
4
5
6
7
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4
5
6
7
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5
6
7
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6
7
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7     dco::derivative(dco::derivative(x_in))=1.; // seed x_{(1,2)}
8
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7     dco::derivative(dco::derivative(x_in))=1.; // seed x_{(1,2)}
8     DCO_BM::global_tape->interpret_adjoint(); // interpret base tape
9
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7     dco::derivative(dco::derivative(x_in))=1.; // seed x_{(1,2)}
8     DCO_BM::global_tape->interpret_adjoint(); // interpret base tape
9     dx=dco::value(dco::derivative(x_in)); // harvest first derivative
10
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7     dco::derivative(dco::derivative(x_in))=1.; // seed x_{(1,2)}
8     DCO_BM::global_tape->interpret_adjoint(); // interpret base tape
9     dx=dco::value(dco::derivative(x_in)); // harvest first derivative
10    ddx=dco::derivative(dco::value(x_in)); // harvest second derivative
11
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7     dco::derivative(dco::derivative(x_in))=1.; // seed x_{(1,2)}
8     DCO_BM::global_tape->interpret_adjoint(); // interpret base tape
9     dx=dco::value(dco::derivative(x_in)); // harvest first derivative
10    ddx=dco::derivative(dco::value(x_in)); // harvest second derivative
11    DCO_TT::remove(DCO_M::global_tape); // deallocate tape
12
13  }
```

Adjoints of Adjoints

AD with dco/c++


```
1     ...
2     DCO_T x=x_in; // read-only active argument
3     f(x,p,w); // run augmented primal
4     x_v=dco::value(dco::value(x)); // get primal function value
5     dco::value(dco::derivative(x))=1.; // seed x_{(1)}
6     DCO_M::global_tape->interpret_adjoint(); // interpret tape
7     dco::derivative(dco::derivative(x_in))=1.; // seed x_{(1,2)}
8     DCO_BM::global_tape->interpret_adjoint(); // interpret base tape
9     dx=dco::value(dco::derivative(x_in)); // harvest first derivative
10    ddx=dco::derivative(dco::value(x_in)); // harvest second derivative
11    DCO_TT::remove(DCO_M::global_tape); // deallocate tape
12    DCO_BTT::remove(DCO_BM::global_tape); // deallocate base tape
13  }
```

Adjoints of Adjoints with dco/c++

Driver


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint of adjoint driver
5  void second_derivative(T &x_v, const PT &p, const PT &w, T &dx,T &ddx) {
6    ...
7  }
8
9  #include "h_main.hpp // main
```

```
./h_aa.exe 1.1 1 0.5
0.650594 -0.502455 -1.93886
```

Wake Up!

State the adjoint of adjoint AD model for $y = \frac{1}{x}$.

Outline

Arbitrary nestings of tangent and adjoint modes are possible, for example,

- tangent of adjoint of tangent; Differentiation of the adjoint of tangent with respect to x in direction $x^{(3)}$ yields

$$x^{(3)}_{(2)} = f^{[3]}(x) \cdot x^{(1)} \cdot y^{(1)}_{(2)} \cdot x^{(3)} \ .$$

- adjoint of tangent of adjoint of tangent; Differentiation of the above with respect to x in direction $x^{(3)}_{(2,4)}$ yields

$$x_{(4)} = f^{[4]}(x) \cdot x^{(1)} \cdot y^{(1)}_{(2)} \cdot x^{(3)} \cdot x^{(3)}_{(2,4)} \ .$$

Commutativity of scalar multiplication makes the order of the factors irrelevant (in infinite precision arithmetic).

Outline

Summary