Introduction to Algorithmic Differentiation (AD)

First-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 first derivatives

$$f' \equiv \frac{df}{dx}$$

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 (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

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

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

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

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).

Outline

First-order adjoints

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

of

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

are defined as

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

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

Wake Up!

What is the value of the first-order tangent of $f(x) = x^2$ at $x = 2$ in direction $x^{(1)} = 2$?

Which value of $y_{(1)}$ yields the same value ($y^{(1)}$) for the first-order adjoint?

Outline

First-order tangents and adjoints turn out to be equivalent in the univariate scalar case due to commutativity of scalar multiplication, that is,

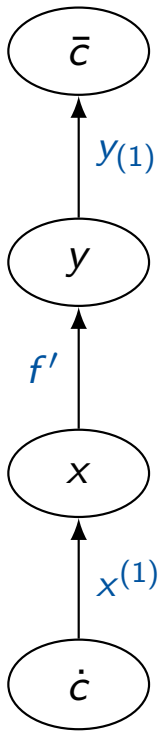
$$f'(x) \cdot x^{(1)} = y_{(1)} \cdot f'(x) \quad \Leftrightarrow \quad x^{(1)} = y_{(1)} \ .$$

This will not be the case for multivariate vector functions $F : \boldsymbol{R}^n \to \boldsymbol{R}^m$.

In general, the following first-order differential invariant holds:

$$x_{(1)} \cdot x^{(1)} = (y_{(1)} \cdot f'(x)) \cdot x^{(1)} = y_{(1)} \cdot (f'(x) \cdot x^{(1)}) = y_{(1)} \cdot y^{(1)} \ .$$

The primal function f is assumed to be embedded into source and target contexts. Both its active argument x and result y become functions of the active source context argument $\dot{c} \in \boldsymbol{R}$.



We define

$$x^{(1)} \equiv \frac{dx}{d\dot{c}} \quad \text{and} \quad y^{(1)} \equiv \frac{dy}{d\dot{c}}.$$

Similarly, the active target context result $\bar{c} \in \boldsymbol{R}$ becomes a function of both x and y with

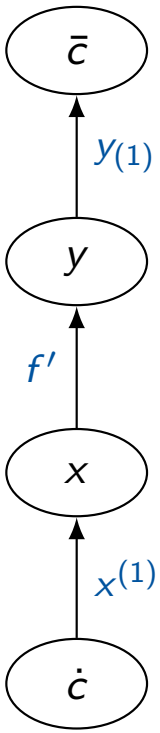
$$x_{(1)} \equiv \frac{d\bar{c}}{dx} \quad \text{and} \quad y_{(1)} \equiv \frac{d\bar{c}}{dy}.$$

For given $x^{(1)}$ and $y_{(1)}$, both $y^{(1)}$ and $x_{(1)}$ follow immediately from the chain rule of differential calculus.

First Derivatives

Perspectives

DAG



ALGEBRA

Tangent

$$y^{(1)} \equiv \frac{dy}{d\dot{c}} = \frac{dy}{dx} \cdot \frac{dx}{d\dot{c}} \equiv f' \cdot x^{(1)}$$

Adjoint

$$x_{(1)} \equiv \frac{d\bar{c}}{dx} = \frac{d\bar{c}}{dy} \cdot \frac{dy}{dx} \equiv y_{(1)} \cdot f'$$

CODE

```
// primal
set(x); y=sin(x);

// tangent
set(xt1); yt1=cos(x)*xt1;

// adjoint
set(ya1); xa1=ya1*cos(x);
```

The chain rule amounts to products of local derivatives; equiv. products of edge labels in the directed acyclic graph (DAG).

Outline

AD treats the given primal function as a sequence of $q \geq 0$ (continuously) differentiable elemental functions φ_j evaluated as a single assignment code (SAC)[1]

$$v_j = \varphi_j(v_k)_{k \prec j} \quad \text{for } j = 1, \ldots, q$$

and where $v_0 = x$, $y = v_q$ and $k \prec j$ if v_k is an argument of φ_j.

A DAG $G = (V, E)$ is induced. Partial derivatives of the elemental functions wrt. their arguments are associated with the corresponding edges.

The DAG turns out to be a trivial chain if all φ_j are univariate scalar functions.

Example: $e^{\sin(x^2)} - p$ with passive p

$$v_0 = x; \; v_1 = v_0^2; \; v_2 = \sin(v_1); \; v_3 = e^{v_2}; \; y = v_4 = v_3 - p$$

[1] The results of elemental functions are stored in unique variables.

Tangents are computed by application of the chain rule in tangent (or forward) AD mode.

If all φ_j are univariate scalar functions ($|k \prec j| = 1$), then

$$v_j^{(1)} = \varphi_j'(v_k)_{k \prec j} \cdot v_k^{(1)} \quad \text{for } j = 1, \ldots, q \ .$$

Example: $e^{\sin(x^2)} - p$ with passive p

$$v_0^{(1)} = x^{(1)};\ v_1^{(1)} = 2 v_0 v_0^{(1)};\ v_2^{(1)} = \cos(v_1) v_1^{(1)};\ v_3^{(1)} = v_3 v_2^{(1)};\ y^{(1)} = v_4^{(1)} = v_3^{(1)}$$

Note that explicit construction of the DAG is not required as the primal function evaluation can simply be augmented with the propagation of the $v_j^{(1)}$. ($\rightarrow$ overloading)

Adjoints are computed by application of the chain rule in adjoint (or reverse) AD mode.

If all φ_j are univariate scalar functions, then

$$v_{k(1)} = v_{j(1)} \cdot \varphi_j'(v_k)_{k \prec j} \quad \text{for } j = q, \ldots, 1 \, .$$

Example: $e^{\sin(x^2)} - p$ with passive p

$$v_{3(1)} = v_{4(1)} = y_{(1)}; \; v_{2(1)} = v_3 v_{3(1)}; \; v_{1(1)} = \cos(v_1) v_{2(1)}; \; x_{(1)} = v_{0(1)} = 2 v_0 v_{1(1)}$$

Note that the primal function evaluation needs to be extended with the explicit construction of the DAG in order to enable the backward propagation of the $v_{j(1)}$ by interpretation of the DAG (also: tape; $\rightarrow$ data flow reversal)

Wake Up!

Derive first-order tangent and adjoint single assignment code for

$$y = f(x) = \sin(x)^2 - e^0 + \cos(x)^2 \ .$$

Outline

The AD software tool dco/c++ stands for

derivative code by overloading in C++ .

It is developed and distributed by the Numerical Algorithms Group Ltd. in close collaboration with STCE.

Active users of dco/c++ include various academic institutions as well as engineering companies, tier-1 investment banks and formula 1 racing teams.

See

https://www.nag.com/content/algorithmic-differentiation-software

for further information on dco/c++.

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6
7
8
9
10
11 }
12
13 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_T=typename dco::gt1s<T>::type; // tangent type
7
8
9
10
11 }
12
13 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_T=typename dco::gt1s<T>::type; // tangent type
7    DCO_T x; // activate
8
9
10
11 }
12
13 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_T=typename dco::gt1s<T>::type; // tangent type
7    DCO_T x; // activate
8    dco::value(x)=x_v; dco::derivative(x)=1; // seed
9
10
11 }
12
13 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_T=typename dco::gt1s<T>::type; // tangent type
7    DCO_T x; // activate
8    dco::value(x)=x_v; dco::derivative(x)=1; // seed
9    f(x,p,w); // compute
10
11 }
12
13 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_T=typename dco::gt1s<T>::type; // tangent type
7    DCO_T x; // activate
8    dco::value(x)=x_v; dco::derivative(x)=1; // seed
9    f(x,p,w); // compute
10   dx=dco::derivative(x); x_v=dco::value(x); // harvest
11 }
12
13 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // tangent driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_T=typename dco::gt1s<T>::type; // tangent type
7    DCO_T x; // activate
8    dco::value(x)=x_v; dco::derivative(x)=1; // seed
9    f(x,p,w); // compute
10   dx=dco::derivative(x); x_v=dco::value(x); // harvest
11 }
12
13 #include "g_main.hpp" // main
```

```
./g_t.exe 1.1 1 0.5
0.650594 -0.502455
```

AD with dco/c++

Adjoints

```
1  #include "f.hpp" // primal source
2
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6
7
8
9
10
11
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7
8
9
10
11
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8
9
10
11
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9
10
11
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10
11
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints

```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13   f(x,p,w); // compute augmented primal
14
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(x); // get primal result
15
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(x); // get primal result
15   dco::derivative(x)=1; // seed
16
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(x); // get primal result
15   dco::derivative(x)=1; // seed
16   DCO_M::global_tape->interpret_adjoint(); // interpret tape
17
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(x); // get primal result
15   dco::derivative(x)=1; // seed
16   DCO_M::global_tape->interpret_adjoint(); // interpret tape
17   dx=dco::derivative(x_in); // harvest
18
19 }
20
21 #include "g_main.hpp" // main
```

AD with dco/c++

Adjoints


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void first_derivative(T &x_v, const PT &p, const PT &w, T &dx) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    DCO_T x_in; dco::value(x_in)=x_v; // activate
10   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
11   DCO_M::global_tape->register_variable(x_in); // register active input
12   DCO_T x=x_in; // read-only inoutput
13   f(x,p,w); // compute augmented primal
14   x_v=dco::value(x); // get primal result
15   dco::derivative(x)=1; // seed
16   DCO_M::global_tape->interpret_adjoint(); // interpret tape
17   dx=dco::derivative(x_in); // harvest
18   DCO_TT::remove(DCO_M::global_tape); // deallocate tape
19 }
20
21 #include "g_main.hpp" // main
```

Let f be the sigmoid function with $f_1 = \sin$ and $f_2 = \cos$ and passive arguments $p = 1$ and $w = 0.5$.

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

```
./g_a.exe 1.1 1 0.5
0.650594 -0.502455
```

Outline

Summary

Tangents

Adjoints

Differential Invariant

AD
- Tangents
- Adjoints

AD with dco/c++
- Tangents
- Adjoints

Summary