Introduction to Algorithmic Differentiation (AD)

Second-(and Higher-)Order Tangent AD by Overloading ($F : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$)

Uwe Naumann

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

RWTH Aachen

Contents

Outline

Newton's method approximates a solution to the unconstrained nonlinear optimization problem $\min_{x \in \mathbb{R}^n} f(x)$ through linearization of the first-order optimality condition

$$f'(x + \Delta x) = f'(x) + f''(x) \cdot \Delta x = 0$$

yielding systems of linear equations

$$f'' \cdot \Delta x = -f'$$

to be solved for Δx and followed by updating $x := x + \alpha \cdot \Delta x$ iteratively for a suitable start value x and damping parameter $\mathbb{R} \ni \alpha > 0$ determined by line search.

The objective function f is assumed to be twice continuously differentiable (implying $f''^T = f''$) at all points of interest.

Outline

Tangents of Tangents

Definition ($f : \boldsymbol{R}^n \to \boldsymbol{R}$)

A second derivative code generated in tangent of tangent AD mode computes

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

as

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

where $f' = f'(\mathrm{x}) \in \boldsymbol{R}^{1 \times n}$ and $f'' = f''(\mathrm{x}) \in \boldsymbol{R}^{n \times n}$.

Notes: nonincremental version; both $y^{(1)} \in \boldsymbol{R}$ and $y^{(2)} \in \boldsymbol{R}$ required and non-vanishing $\mathrm{x}^{(1,2)} \in \boldsymbol{R}^n$ in context of chain rule; $f''^T = f'' \in \boldsymbol{R}^{n \times n}$ as f twice continuously differentiable

Algorithmic differentiation of the (nonincremental) first-order tangent

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

in tangent mode yields

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

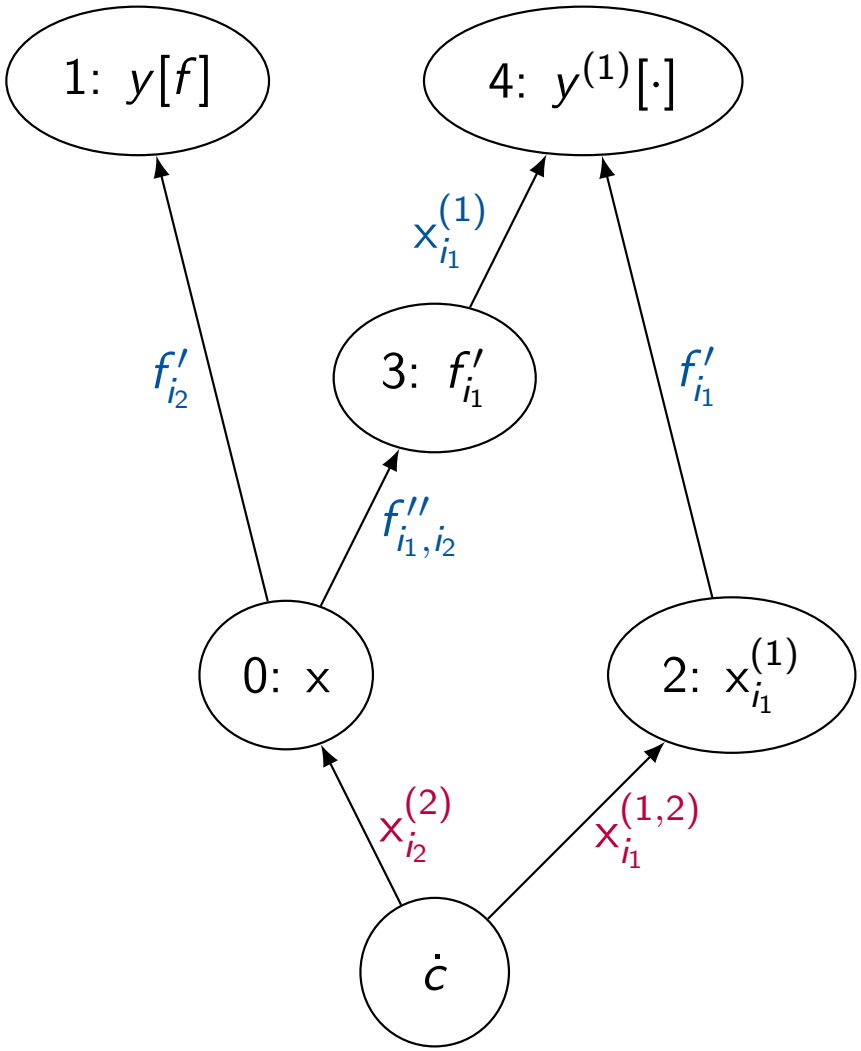
implying with[1] $y^{(1)} = {x^{(1)}}^T \cdot {f'}^T$ and ${f''}^T = f''$

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

[1] Note differentiation of column vectors rather than row vectors yielding Jacobians rather than transposed Jacobians for seamless application of the chain rule.

Tangents of Tangents

Tangent of Tangent DAG (using index notation)

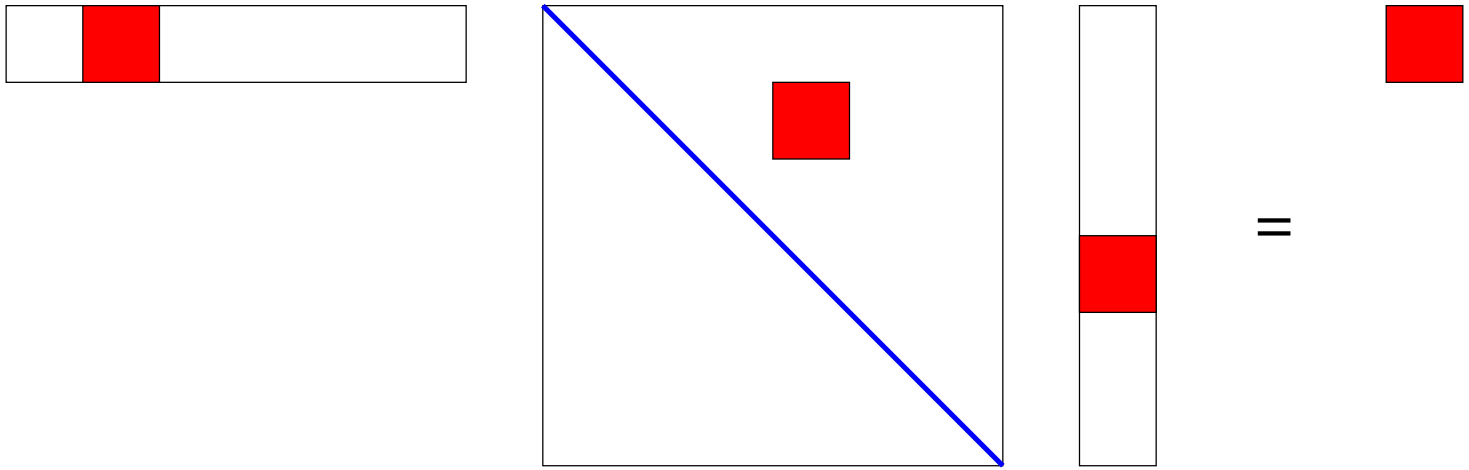


$$y^{(2)} \equiv \frac{dy}{d\dot{c}} = f'_{i_2} \cdot x^{(2)}_{i_2} = f' \cdot x^{(2)}$$

$$\begin{aligned} y^{(1,2)} &\equiv \frac{dy^{(1)}}{d\dot{c}} \\ &= x^{(1)}_{i_1} \cdot f''_{i_1,i_2} \cdot x^{(2)}_{i_2} + f'_{i_1} \cdot x^{(1,2)}_{i_1} \\ &= x^{(1)\,T} \cdot f'' \cdot x^{(2)} + f' \cdot x^{(1,2)} \end{aligned}$$

Wake Up!

Draw the tangent of tangent DAG for $y = x_0 \cdot x_1$.



$$y^{(1,2)} = \mathrm{x}^{(1)\,T} \cdot f'' \cdot \mathrm{x}^{(2)} + f' \cdot \mathrm{x}^{(1,2)}$$

... accumulation of the whole Hessian element-wise by seeding input directions $\mathrm{x}^{(1)} \in \boldsymbol{R}^n$ and $\mathrm{x}^{(2)} \in \boldsymbol{R}^n$ independently with the Cartesian basis vectors in $\boldsymbol{R}^n$ for $\mathrm{x}^{(1,2)} = 0$; harvesting from $y^{(1,2)}$.

For $j = n, \ldots, n+q-1$:

$$
\begin{aligned}
v_j &= \varphi_j(v_i)_{i \prec j} \\
v_j^{(2)} &= 0; \quad v_j^{(2)} \mathrel{+}= (\varphi_j')_{i_2} \cdot v_{i_2}^{(2)} \quad \forall\, i_2 \prec j \\
v_j^{(1)} &= 0; \quad v_j^{(1)} \mathrel{+}= (\varphi_j')_{i_1} \cdot v_{i_1}^{(1)} \quad \forall\, i_1 \prec j \\
v_j^{(1,2)} &= 0; \quad v_j^{(1,2)} \mathrel{+}= v_{i_1}^{(1)} \cdot (\varphi_j'')_{i_1,i_2} \cdot v_{i_2}^{(2)} + (\varphi_j')_{i_1} \cdot v_{i_1}^{(1,2)} \quad \forall\, i_1, i_2 \prec j\ .
\end{aligned}
$$

The above formula lends itself naturally to implementation by overloading through nesting of first-order tangent types. This approach is taken by dco/c++.

Seeding Cartesian basis vectors, e.g, $v_0^{(1)} = x_0^{(1)} = 1$ and $v_1^{(2)} = x_1^{(2)} = 1$ while initializing all remaining inputs to zero and followed by propagating second-order tangents alongside first-order tangents (and primal function values) as

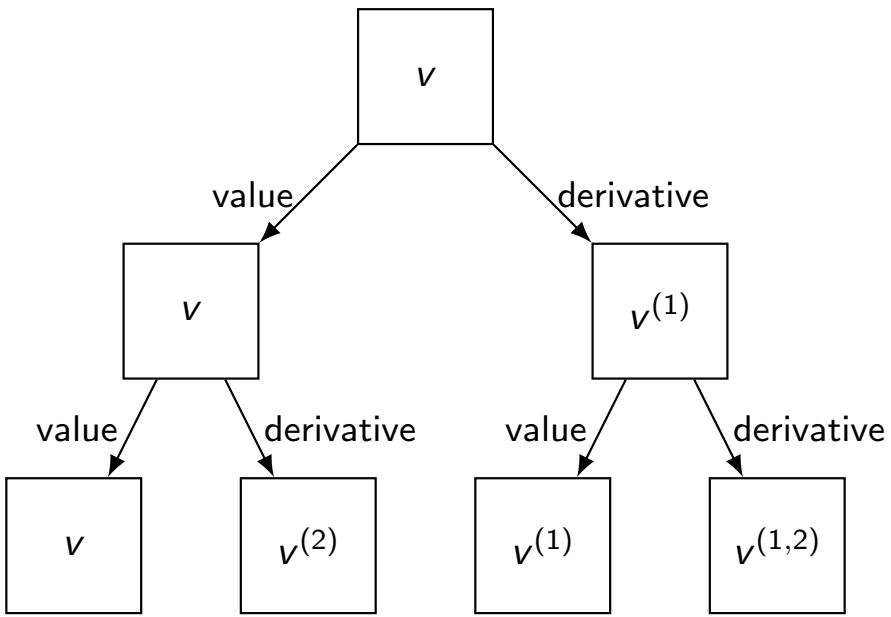
$$
\begin{aligned}
v_2^{(1)} &+= 2 \cdot v_0 \cdot v_0^{(1)} & v_2^{(1,2)} &+= 2 \cdot (v_0^{(2)} \cdot v_0^{(1)} + v_0 \cdot v_0^{(1,2)}) \\
v_3^{(1)} &+= 2 \cdot v_1 \cdot v_1^{(1)} & v_3^{(1,2)} &+= 2 \cdot (v_1^{(2)} \cdot v_1^{(1)} + v_1 \cdot v_1^{(1,2)}) \\
v_4^{(1)} &+= v_2^{(1)} & v_4^{(1,2)} &+= v_2^{(1,2)} \\
v_4^{(1)} &+= v_3^{(1)} & v_4^{(1,2)} &+= v_3^{(1,2)} \\
v_5^{(1)} &+= \cos(v_4) \cdot v_4^{(1)} & v_5^{(1,2)} &+= -\sin(v_4) \cdot v_4^{(1)} \cdot v_4^{(2)} + \cos(v_4) \cdot v_4^{(1,2)} \\
v_6^{(1)} &+= v_6 \cdot v_5^{(1)} & v_6^{(1,2)} &+= v_6^{(2)} \cdot v_5^{(1)} + v_6 \cdot v_5^{(1,2)}
\end{aligned}
$$

yields individual entries of the Hessian, e.g, $\frac{dy^2}{dx_0 dx_1}$, to be harvested from $y^{(1,2)} = v_6^{(1,2)}$ for $v_i^{(2)} = v_i^{(1,2)} = 0$; omitting computation of v_i and $v_i^{(2)}$

Tangents of Tangents with dco/c++

Cheat Sheet

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




```
dco::value(dco::value(v))
[dco::passive_value(v)]
dco::derivative(dco::value(v))
dco::value(dco::derivative(v)
dco::derivative(dco::derivative(v)
```

```
1  template<typename T, typename PT>
2  void f(size_t ncp, size_t ncs, const PT &eps, T& x, const std::vector<T>& p, const
       std::vector<std::vector<PT>>& dW) {
3    newton(x,p[0],eps);
4    T s=0;
5    size_t m=dW.size();
6    for (size_t j=0;j<m;j+=ncp) paths(ncs,j,j+ncp,x,p,dW,s);
7    x=s/m;
8  }
```

We are looking for the Hessian of the final `x` with respect to `p`.

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)\,T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$


```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
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 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$


```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$


```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8
9
10
11
12
13
14
15
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering | RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9
10
11
12
13
14
15
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering | RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10
11
12
13
14
15     }
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12
13
14
15     }
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering | RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12       f(ncp,ncs,eps,x,p,dW); // compute primal
13
14
15     }
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = p^{(1)^T} \cdot f'' \cdot p^{(2)} + f' \cdot p^{(1,2)}$


```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12       f(ncp,ncs,eps,x,p,dW); // compute primal
13       ddxdpp[i][j]=dco::derivative(dco::derivative(x)); // harvest Hessian
14
15     }
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering | RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12       f(ncp,ncs,eps,x,p,dW); // compute primal
13       ddxdpp[i][j]=dco::derivative(dco::derivative(x)); // harvest Hessian
14       dco::derivative(dco::value(p[j]))=0; // unseed p_j^{(2)}
15     }
16
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = p^{(1)^T} \cdot f'' \cdot p^{(2)} + f' \cdot p^{(1,2)}$


```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12       f(ncp,ncs,eps,x,p,dW); // compute primal
13       ddxdpp[i][j]=dco::derivative(dco::derivative(x)); // harvest Hessian
14       dco::derivative(dco::value(p[j]))=0; // unseed p_j^{(2)}
15     }
16     dxdp[i]=dco::value(dco::derivative(x)); // harvest gradient
17
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = \mathrm{p}^{(1)^T} \cdot f'' \cdot \mathrm{p}^{(2)} + f' \cdot \mathrm{p}^{(1,2)}$

Software and Tools for Computational Engineering | RWTH AACHEN UNIVERSITY

```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12       f(ncp,ncs,eps,x,p,dW); // compute primal
13       ddxdpp[i][j]=dco::derivative(dco::derivative(x)); // harvest Hessian
14       dco::derivative(dco::value(p[j]))=0; // unseed p_j^{(2)}
15     }
16     dxdp[i]=dco::value(dco::derivative(x)); // harvest gradient
17     dco::value(dco::derivative(p[i]))=0; // unseed p_i^{(1)}
18   }
19
20 }
```

Hello AD World and dco/c++

$x^{(1,2)} = {p^{(1)}}^T \cdot f'' \cdot p^{(2)} + f' \cdot p^{(1,2)}$


```
1  template<typename T, typename PT> // Hessian drives
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, const std::vector<T> &
     p_v, const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp, std::
     vector<std::vector<T>> &ddxdpp) {
3    using DCO_BT=typename dco::gt1s<T>::type; // tangent (base) type
4    using DCO_T=typename dco::gt1s<DCO_BT>::type; // tangent of tangent type
5    size_t n=p_v.size(); // number of rows/columns in Hessian
6    DCO_T x; std::vector<DCO_T> p(n); dco::passive_value(p)=p_v; // activate
7    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
8      dco::value(dco::derivative(p[i]))=1; // seed p_i^{(1)}
9      for (size_t j=0;j<n;j++) { // iterate over Cartesian basis
10       dco::derivative(dco::value(p[j]))=1; // seed p_j^{(2)}
11       x=x_v; // [re]set inoutput
12       f(ncp,ncs,eps,x,p,dW); // compute primal
13       ddxdpp[i][j]=dco::derivative(dco::derivative(x)); // harvest Hessian
14       dco::derivative(dco::value(p[j]))=0; // unseed p_j^{(2)}
15     }
16     dxdp[i]=dco::value(dco::derivative(x)); // harvest gradient
17     dco::value(dco::derivative(p[i]))=0; // unseed p_i^{(1)}
18   }
19   x_v=dco::passive_value(x); // get primal function value
20 }
```

Given a (tangent mode) gradient driver

```
template<typename T, typename PT>
void gradient(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
    const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp);
```

for

```
template<typename T, typename PT>
void f(size_t ncp, size_t ncs, const PT &eps, T& x, const std::vector<T>& p, const
    std::vector<std::vector<PT>>& dW);
```

the Hessian of the final `x` with respect to `p` can be computed as the Jacobian of the corresponding gradient.

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3
4
5
6
7
8
9
10
11
12
13
14
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4
5
6
7
8
9
10
11
12
13
14
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5
6
7
8
9
10
11
12
13
14
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6
7
8
9
10
11
12
13
14
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7
8
9
10
11
12
13
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8
9
10
11
12
13
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8      dco::derivative(p[i])=1; // seed
9
10
11
12
13
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8      dco::derivative(p[i])=1; // seed
9      gradient(ncp,ncs,eps,x,p,dW,dxdp); // compute gradient
10
11
12
13
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8      dco::derivative(p[i])=1; // seed
9      gradient(ncp,ncs,eps,x,p,dW,dxdp); // compute gradient
10     dxdp_v[i]=dco::value(dxdp[i]); // harvest gradient
11
12
13
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8      dco::derivative(p[i])=1; // seed
9      gradient(ncp,ncs,eps,x,p,dW,dxdp); // compute gradient
10     dxdp_v[i]=dco::value(dxdp[i]); // harvest gradient
11     for (size_t j=0;j<n;j++)
12       ddxdpp[i][j]=dco::derivative(dxdp[j]); // harvest Hessian
13
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8      dco::derivative(p[i])=1; // seed
9      gradient(ncp,ncs,eps,x,p,dW,dxdp); // compute gradient
10     dxdp_v[i]=dco::value(dxdp[i]); // harvest gradient
11     for (size_t j=0;j<n;j++)
12       ddxdpp[i][j]=dco::derivative(dxdp[j]); // harvest Hessian
13     dco::derivative(p[i])=0; // unseed read-only input
14   }
15
16 }
```

Hello AD World and dco/c++

Jacobian of Gradient + Gradient + Function Value


```
1  template<typename T, typename PT> // Hessian driver
2  void hessian(size_t ncp, size_t ncs, const PT &eps, T &x_v, std::vector<T> &p_v,
     const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp_v, std::vector<
     std::vector<T>> &ddxdpp) {
3    using DCO_T=typename dco::gt1s<T>::type; // tangent type
4    size_t n=p_v.size(); // number of rows/columns in Hessian
5    DCO_T x; std::vector<DCO_T> p(n), dxdp(n); dco::value(p)=p_v; // activate
6    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
7      x=x_v; // [re]set inoutput
8      dco::derivative(p[i])=1; // seed
9      gradient(ncp,ncs,eps,x,p,dW,dxdp); // compute gradient
10     dxdp_v[i]=dco::value(dxdp[i]); // harvest gradient
11     for (size_t j=0;j<n;j++)
12       ddxdpp[i][j]=dco::derivative(dxdp[j]); // harvest Hessian
13     dco::derivative(p[i])=0; // unseed read-only input
14   }
15   x_v=dco::value(x); // get primal function value
16 }
```

Outline

Definition ($F : \boldsymbol{R}^n \to \boldsymbol{R}^m$)

A second derivative code generated in tangent of tangent AD mode computes

$$\begin{pmatrix} \mathrm{y} \\ \mathrm{y}^{(2)} \\ \mathrm{y}^{(1)} \\ \mathrm{y}^{(1,2)} \end{pmatrix} = F^{(1,2)}\left(\mathrm{x}, \mathrm{x}^{(2)}, \mathrm{x}^{(1)}, \mathrm{x}^{(1,2)}\right)$$

as

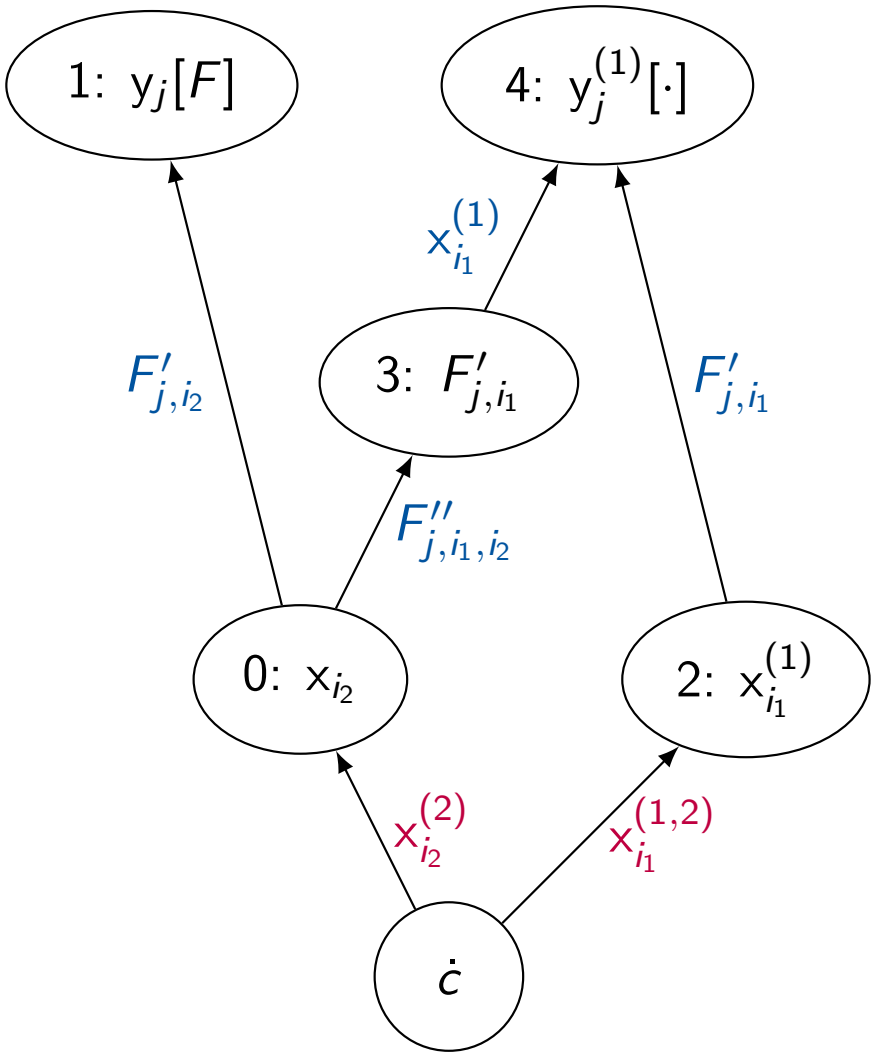
$$\begin{pmatrix} \mathrm{y} \\ \mathrm{y}_j^{(2)} \\ \mathrm{y}_j^{(1)} \\ \mathrm{y}_j^{(1,2)} \end{pmatrix} = \begin{pmatrix} F(\mathrm{x}) \\ F'_{j,i_2} \cdot \mathrm{x}_{i_2}^{(2)} \\ F'_{j,i_1} \cdot \mathrm{x}_{i_1}^{(1)} \\ F''_{j,i_1,i_2} \cdot \mathrm{x}_{i_1}^{(1)} \cdot \mathrm{x}_{i_2}^{(2)} + F'_{j,i_1} \cdot \mathrm{x}_{i_1}^{(1,2)} \end{pmatrix},$$

where

$$F' = F'(\mathrm{x}) \in \boldsymbol{R}^{m \times n} \quad \text{and} \quad F'' = F''(\mathrm{x}) \in \boldsymbol{R}^{m \times n \times n}.$$

Note: nonincremental version assumes initially vanishing $\mathrm{y}^{(2)}$, $\mathrm{y}^{(1)}$ and $\mathrm{y}^{(1,2)}$.

Tangent of Tangent DAG

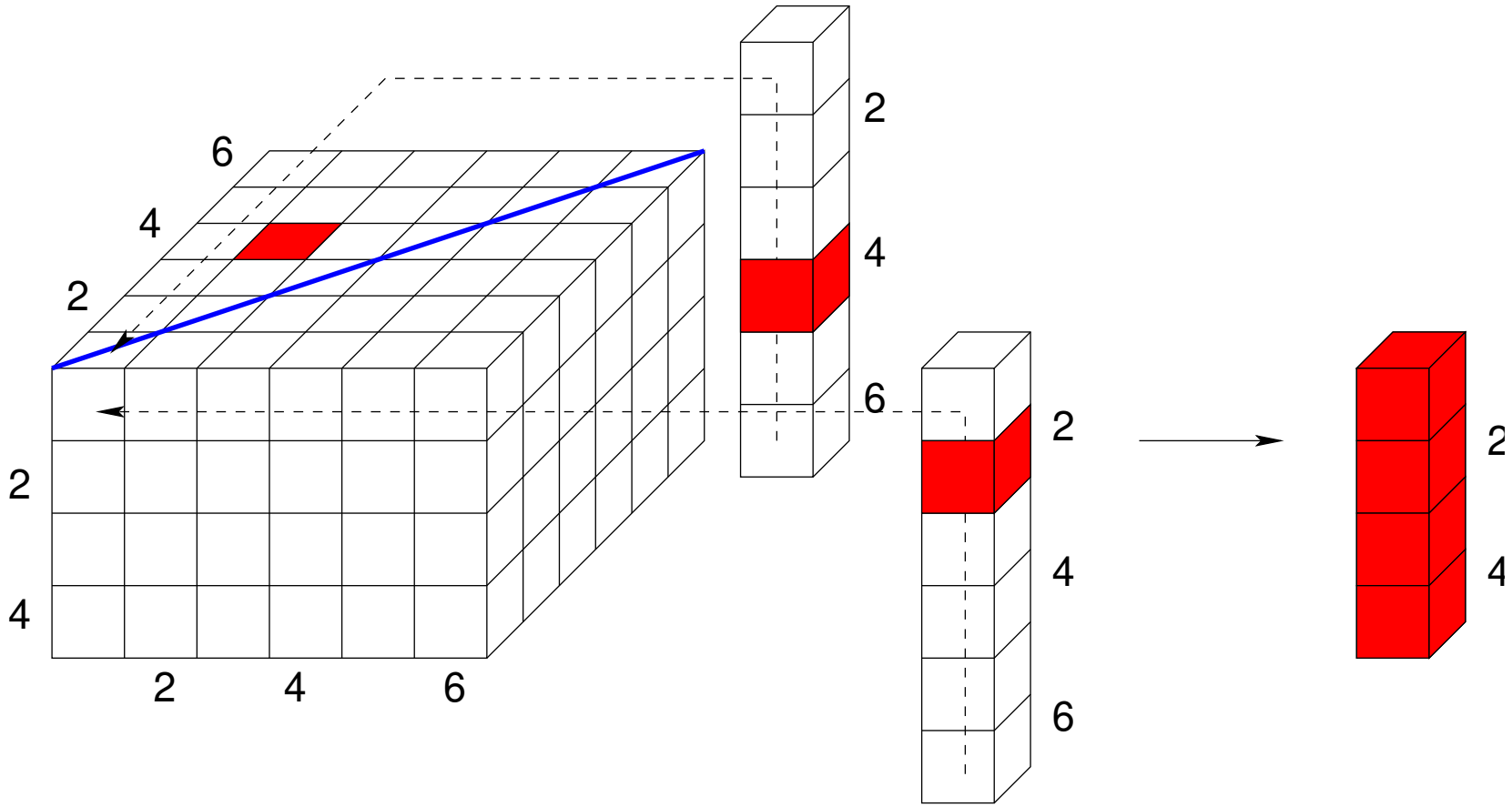


$$y^{(2)} \equiv \frac{dy}{d\dot{c}}$$
$$y_j^{(2)} = F'_{j,i_2} \cdot x_{i_2}^{(2)}$$

$$y^{(1,2)} \equiv \frac{dy^{(1)}}{d\dot{c}}$$
$$y_j^{(1,2)} = F''_{j,i_1,i_2} \cdot x_{i_1}^{(1)} \cdot x_{i_2}^{(2)} + F'_{j,i_1} \cdot x_{i_1}^{(1,2)}$$

Tangents of Tangents

Hessian Tensor Compression



Wake Up!

Draw the tangent of tangent DAG for

$$y_0 = \sin(x_0 \cdot x_1)$$
$$y_1 = \cos(x_0 \cdot x_1) \ .$$

Outline

$$
\begin{aligned}
y &= F(x) \\
y_j^{(3)} &= F'_{j,i_3} \cdot x_{i_3}^{(3)} \\
y_j^{(2)} &= F'_{j,i_2} \cdot x_{i_2}^{(2)} \\
y_j^{(2,3)} &= F''_{j,i_2,i_3} \cdot x_{i_2}^{(2)} \cdot x_{i_3}^{(3)} + F'_{j,i_2} \cdot x_{i_2}^{(2,3)} \\
y_j^{(1)} &= F'_{j,i_1} \cdot x_{i_1}^{(1)} \\
y_j^{(1,3)} &= F''_{j,i_1,i_3} \cdot x_{i_1}^{(1)} \cdot x_{i_3}^{(3)} + F'_{j,i_1} \cdot x_{i_1}^{(1,3)} \\
y_j^{(1,2)} &= F''_{j,i_1,i_2} \cdot x_{i_1}^{(1)} \cdot x_{i_2}^{(2)} + F'_{j,i_1} \cdot x_{i_1}^{(1,2)} \\
y_j^{(1,2,3)} &= F'''_{j,i_1,i_2,i_3} \cdot x_{i_1}^{(1)} \cdot x_{i_2}^{(2)} \cdot x_{i_3}^{(3)} + F''_{j,i_1,i_2} \cdot x_{i_1}^{(1,3)} \cdot x_{i_2}^{(2)} + F''_{j,i_1,i_2} \cdot x_{i_1}^{(1)} \cdot x_{i_2}^{(2,3)} \\
&\quad + F''_{j,i_1,i_2} \cdot x_{i_1}^{(1,2)} \cdot x_{i_3}^{(3)} + F'_{j,i_1} \cdot x_{i_1}^{(1,2,3)}
\end{aligned}
$$

Note commutativity of multiplication in index notation.

Wake Up!

Draw a tangent of tangent of tangent DAG.

$y_j^{(1,3,4)} = ?$

Outline

Summary