Introduction to Algorithmic Differentiation (AD)

First-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

Prerequisites

We consider implementations of multivariate vector functions

$$F : \boldsymbol{R}^n \to \boldsymbol{R}^m : \mathrm{y} = F(\mathrm{x})$$

as differentiable computer programs.

Such programs decompose into sequences of $q = p + m$ differentiable elemental functions φ_j evaluated conceptually as a single assignment code (SAC)[1]

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

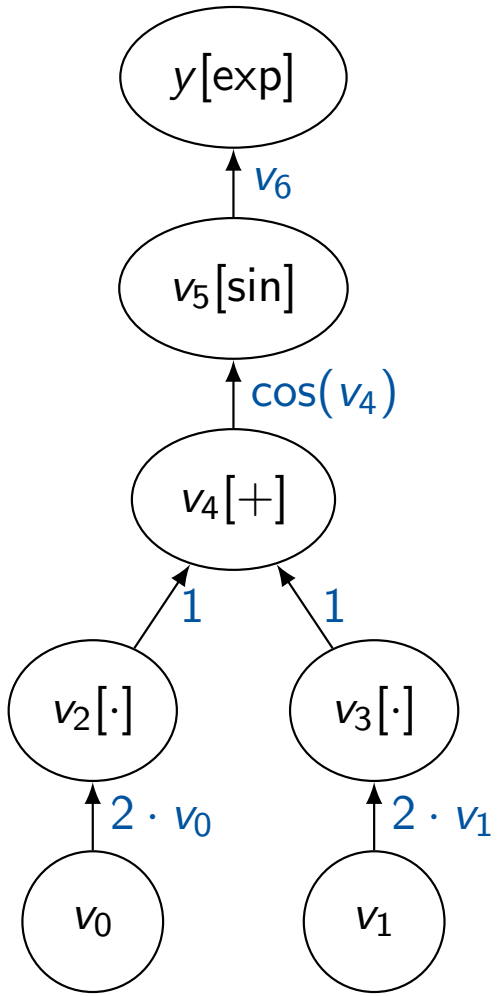
and where $v_i = x_i$ for $i = 0, \ldots, n-1$, $y_k = v_{n+p+k}$ for $k = 0, \ldots, m-1$ and $k \prec j$ if v_k is an argument of φ_j.

A directed acyclic graph (DAG) $G = (V, E)$ is induced. Derivatives of the elemental functions with respect to their arguments are associated with the corresponding edges.

[1] Variables are written once.

$$y = f(\mathrm{x}) = e^{\sin\left(\|\mathrm{x}\|_2^2\right)} = e^{\sin\left(\mathrm{x}^T \cdot \mathrm{x}\right)} = e^{\sin\left(\sum_{i=0}^{n-1} x_i^2\right)}, \quad n = 2$$

$$v_0 = x_0$$
$$v_1 = x_1$$
$$v_2 = v_0^2; \qquad \frac{d\varphi_2}{dv_0} = 2 \cdot v_0$$
$$v_3 = v_1^2; \qquad \frac{d\varphi_3}{dv_1} = 2 \cdot v_1$$
$$v_4 = v_2 + v_3; \qquad \frac{d\varphi_4}{dv_2} = \frac{d\varphi_4}{dv_3} = 1$$
$$v_5 = \sin(v_4); \qquad \frac{d\varphi_5}{dv_4} = \cos(v_4)$$
$$v_6 = e^{v_5}; \qquad \frac{d\varphi_6}{dv_5} = v_6$$
$$y = v_6$$



Outline

The first-order tangent / directional derivative

$$y^{(1)} = F' \cdot x^{(1)} \equiv \frac{dF}{dx}(\tilde{x}) \cdot x^{(1)}$$

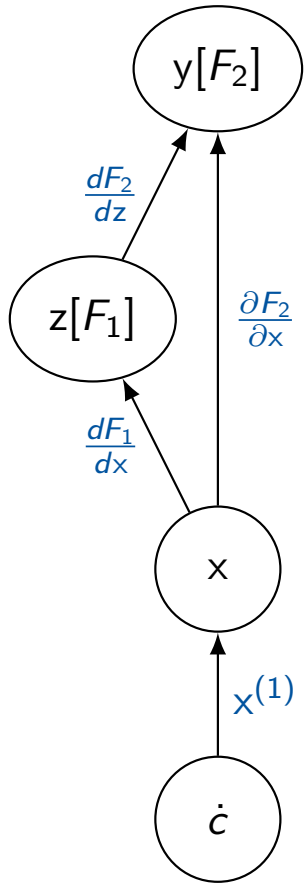
of $y = F(x)$ at $\tilde{x}$ in direction $x^{(1)}$ can be represented as the (total) derivative of $y = y(x(\dot{c}))$ with respect to a (input or tangent) context $\dot{c} \in \boldsymbol{R}$ at $\tilde{x}$ such that

$$\frac{dx}{d\dot{c}} = x^{(1)} \in \boldsymbol{R}^n \ .$$

The chain rule yields

$$y^{(1)} \equiv \frac{dF}{d\dot{c}} = \frac{dF}{dx}(\tilde{x}) \cdot \frac{dx}{d\dot{c}} = \frac{dF}{dx}(\tilde{x}) \cdot x^{(1)} \ .$$

The tangent DAG is induced.



Tangent DAG

Think of $x = x + x^{(1)} \cdot \dot{c}$ at $x = \tilde{x}$ and $\dot{c} = 0$.

The tangent of F in direction $\mathrm{x}^{(1)}$ is equal to the Jacobian $\times$ vector product

$$\mathrm{y}^{(1)} \equiv \left(y_j^{(1)}\right)_{j=0,\ldots,m-1} = \left(\sum_{i=0}^{n-1} F'_{j,i} \cdot x_i^{(1)}\right)_{j=0,\ldots,m-1} = F' \cdot \mathrm{x}^{(1)} \ .$$

The $\sum$ symbol is omitted in index notation yielding

$$y_j^{(1)} = F'_{j,i} \cdot x_i^{(1)}$$

and implying summation over the common (here i) index on the right-hand side. This expression is uniquely determined for known m and n.

Index notation will become particularly useful in the context of second- and higher-order tangents.

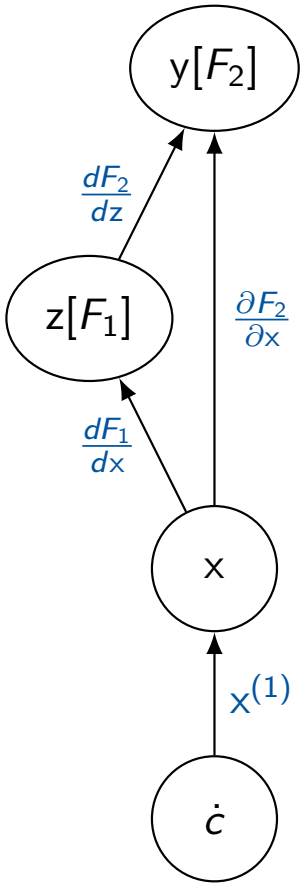
The DAG $G = G(\mathrm{x})$ of F induces a linear mapping (matrix-free Jacobian × vector product)

$$G^{(1)} : \boldsymbol{R}^n \times \boldsymbol{R}^n \to \boldsymbol{R}^m : \quad \mathrm{y}^{(1)} = G \cdot \mathrm{x}^{(1)}$$

defined by the chain rule applied to $F(\mathrm{x}(\dot{c}))$ at $\mathrm{x} = \tilde{\mathrm{x}}$.
This DAG × vector product is evaluated in tangent AD as

$$v_i^{(1)} = \sum_{j \prec i} \frac{d\varphi_i(v_k)_{k \prec i}}{dv_j} \cdot v_j^{(1)} \quad \text{for } i = n, \ldots, n+q-1$$

and where $v_i^{(1)} = x_i^{(1)}$ for $i = 0, \ldots, n-1$ and $y_k^{(1)} = v_{n+p+k}^{(1)}$ for $k = 0, \ldots, m-1$.



Tangent DAG

Outline

Numerical simulation programs typically feature various arguments and results not all of which are subject to differentiation.

Hence, we consider differentiable computer programs implementing

$$F : \boldsymbol{R}^n \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}^m \times \boldsymbol{R}^{m^*} : \quad (\mathrm{y}, \mathrm{y}^*) = F(\mathrm{x}, \mathrm{x}^*) \, ,$$

where the superscript $*$ marks passive arguments / results of F which are not subject to differentiation.

We are interested in the first [partial] directional derivative of the active result y with respect to the active argument x at a given point $(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*)$. An approximation can be computed by finite differences.

First-order tangents of

$$F : \boldsymbol{R}^n \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}^m \times \boldsymbol{R}^{m^*} : \quad (\mathrm{y}, \mathrm{y}^*) = F(\mathrm{x}, \mathrm{x}^*) \,,$$

can be evaluated in (scalar) tangent [forward] AD mode as

$$\mathrm{y}^{(1)} = \mathrm{y}^{(1)} + F^{(1)}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*, \mathrm{x}^{(1)}) \equiv \mathrm{y}^{(1)} + \frac{\partial F}{\partial \mathrm{x}}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*) \cdot \mathrm{x}^{(1)}$$

for $\mathrm{x}^{(1)} \in \boldsymbol{R}^n$ and $\mathrm{y}^{(1)} \in \boldsymbol{R}^m$.

$\mathrm{y}^{(1)}$ can be nonzero due to prior tangent computation with different activity patterns. Incrementation of $\mathrm{y}^{(1)}$ ensures correctness according to the chain rule.

Typically, $\mathrm{y}^{(1)} = 0$ initially.

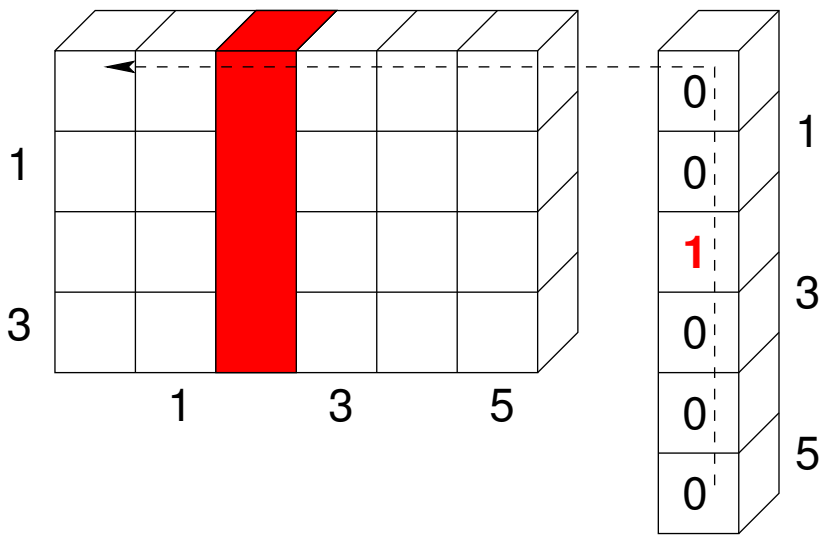
The partial Jacobian $F'_x \equiv \frac{\partial F}{\partial x}$ can be accumulated at $(\tilde{x}, \tilde{x}^*)$ by n evaluations of $F^{(1)}$ with $x^{(1)}$ ranging over the Cartesian basis vectors in R^n.[2] Hence,

$$\text{Cost}(F'_x) = O(n) \cdot \text{Cost}(F^{(1)}) = O(n) \cdot \text{Cost}(F) \ .$$

The second identity follows immediately from the fact that tangents are implemented as DAG $\times$ vector product; see below.

If $n^* = m^* = 0$ then $y = F(x)$ and

$$\begin{aligned} y^{(1)} &= y^{(1)} + F^{(1)}(\tilde{x}, x^{(1)}) \\ &\equiv y^{(1)} + \frac{dF}{dx}(\tilde{x}) \cdot x^{(1)} \ . \end{aligned}$$



[2] This number can be decreased by detecting and exploiting potential sparsity F'_x.

Tangent AD propagates tangents of all elemental functions evaluated by the primal SAC yielding the augmented primal SAC:

$$i = 0, \ldots, n-1: \quad \begin{pmatrix} v_i \\ v_i^{(1)} \end{pmatrix} = \begin{pmatrix} x_i \\ x_i^{(1)} \end{pmatrix}$$

$$j = n, \ldots, n+q-1: \quad \begin{pmatrix} v_j \\ d_{j,i} \\ v_j^{(1)} \end{pmatrix} = \begin{pmatrix} \varphi_j(v_k)_{k \prec j} \\ \frac{d\varphi_j(v_k)_{k \prec j}}{dv_i} \\ \sum_{i \prec j} d_{j,i} \cdot v_i^{(1)} \end{pmatrix}$$

$$k = 0, \ldots, m-1: \quad \begin{pmatrix} y_k \\ y_k^{(1)} \end{pmatrix} = \begin{pmatrix} v_{n+p+k} \\ y_k^{(1)} + v_{n+p+k}^{(1)} \end{pmatrix}$$

Tangents of auxiliary variables v_j, $j = 0, \ldots, n+q-1$ are initially equal to zero.

The above lends itself to implementation by operator and function overloading (e.g, in C++). The entire arithmetic can be overloaded for a custom data type $(v, v^{(1)})$ comprising both value and tangent.

Seeding a Cartesian basis vector, e.g, $v_0^{(1)} = x_0^{(1)} = 1$ and $v_1^{(1)} = x_1^{(1)} = 0$, and initializing $v_i^{(1)} = 0$, $i = 2, \ldots, 6$ followed by propagating tangents alongside the primal function values as

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

yields the corresponding column of the Jacobian, e.g, the first gradient entry, to be harvested from $y^{(1)} = y^{(1)} + v_6^{(1)}$.

Wake Up!

Derive tangent SAC and tangent DAG for the product reduction

$$y = \prod_{i=0}^{n-1} x_i$$

for $n = 3$.

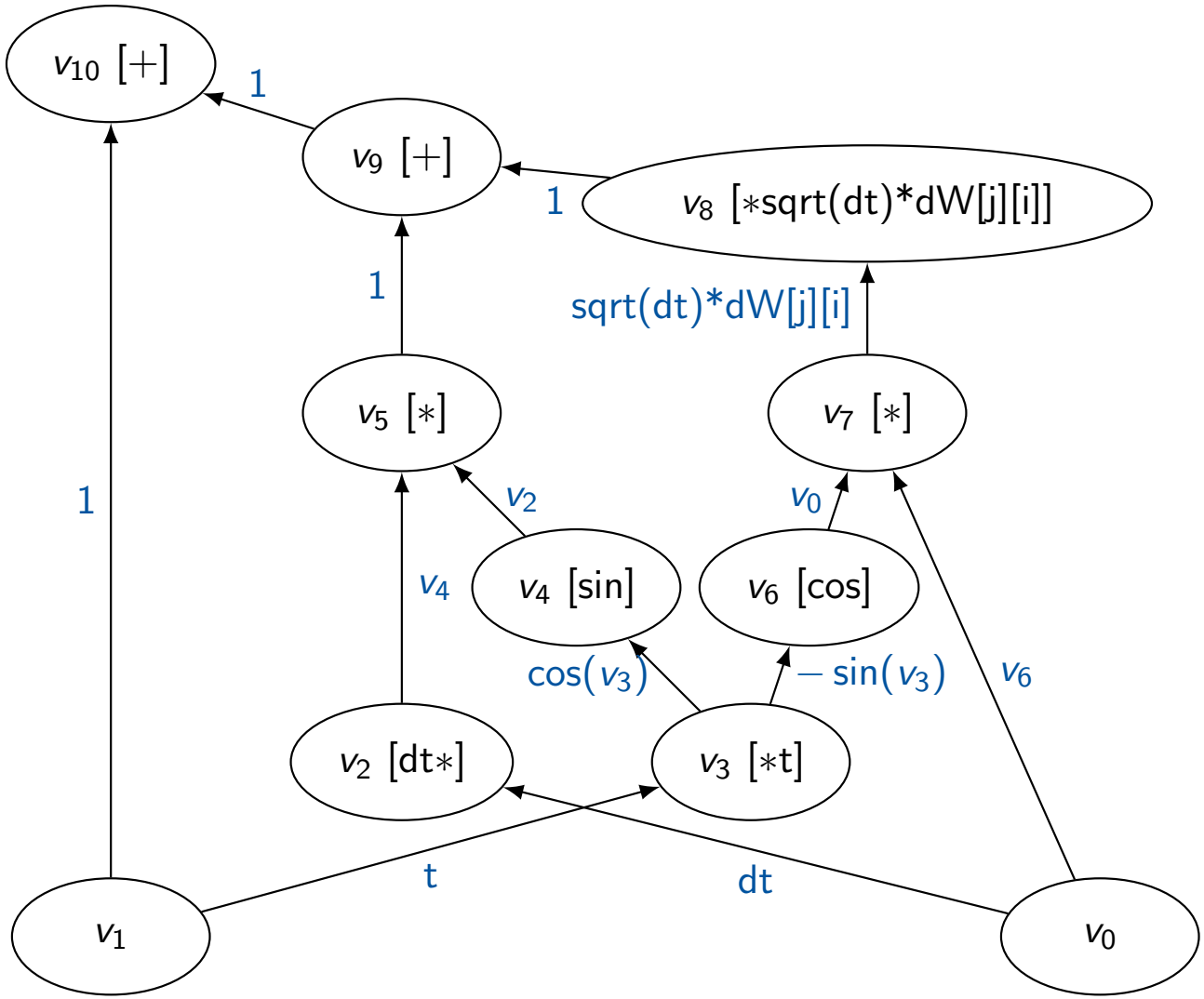
Hello AD World: SAC

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];

$$
\begin{aligned}
v_0 &= \text{p[i]} \\
v_1 &= \text{x} \\
v_2 &= \text{dt} * v_0 & d_{2,0} &= \text{dt} \\
v_3 &= v_1 * \text{t} & d_{3,1} &= \text{t} \\
v_4 &= \sin(v_3) & d_{4,3} &= \cos(v_3) \\
v_5 &= v_2 * v_4 & d_{5,2} &= v_4 & d_{5,4} &= v_2 \\
v_6 &= \cos(v_3) & d_{6,3} &= -\sin(v_3) \\
v_7 &= v_0 * v_6 & d_{7,0} &= v_6 & d_{7,6} &= v_0 \\
v_8 &= v_7 * \text{sqrt(dt)*dW[j][i]} & d_{8,7} &= \text{sqrt(dt)*dW[j][i]} \\
v_9 &= v_5 + v_8 & d_{9,5} &= 1 & d_{9,8} &= 1 \\
v_{10} &= v_1 + v_9 & d_{10,1} &= 1 & d_{10,9} &= 1 \\
\text{x} &= v_{10}
\end{aligned}
$$

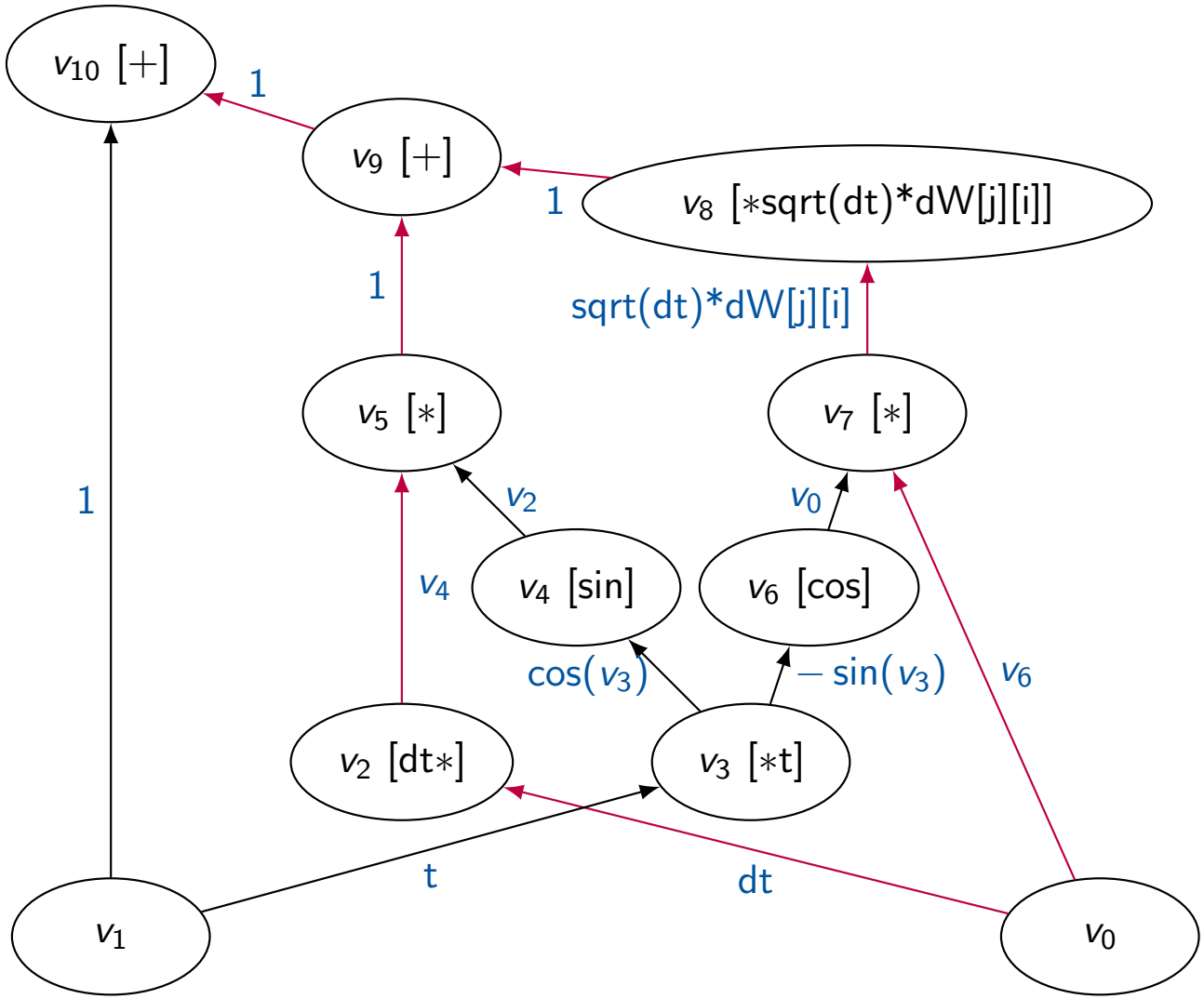
Hello AD World: DAG

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];



Hello AD World: Chain Rule on DAG

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];



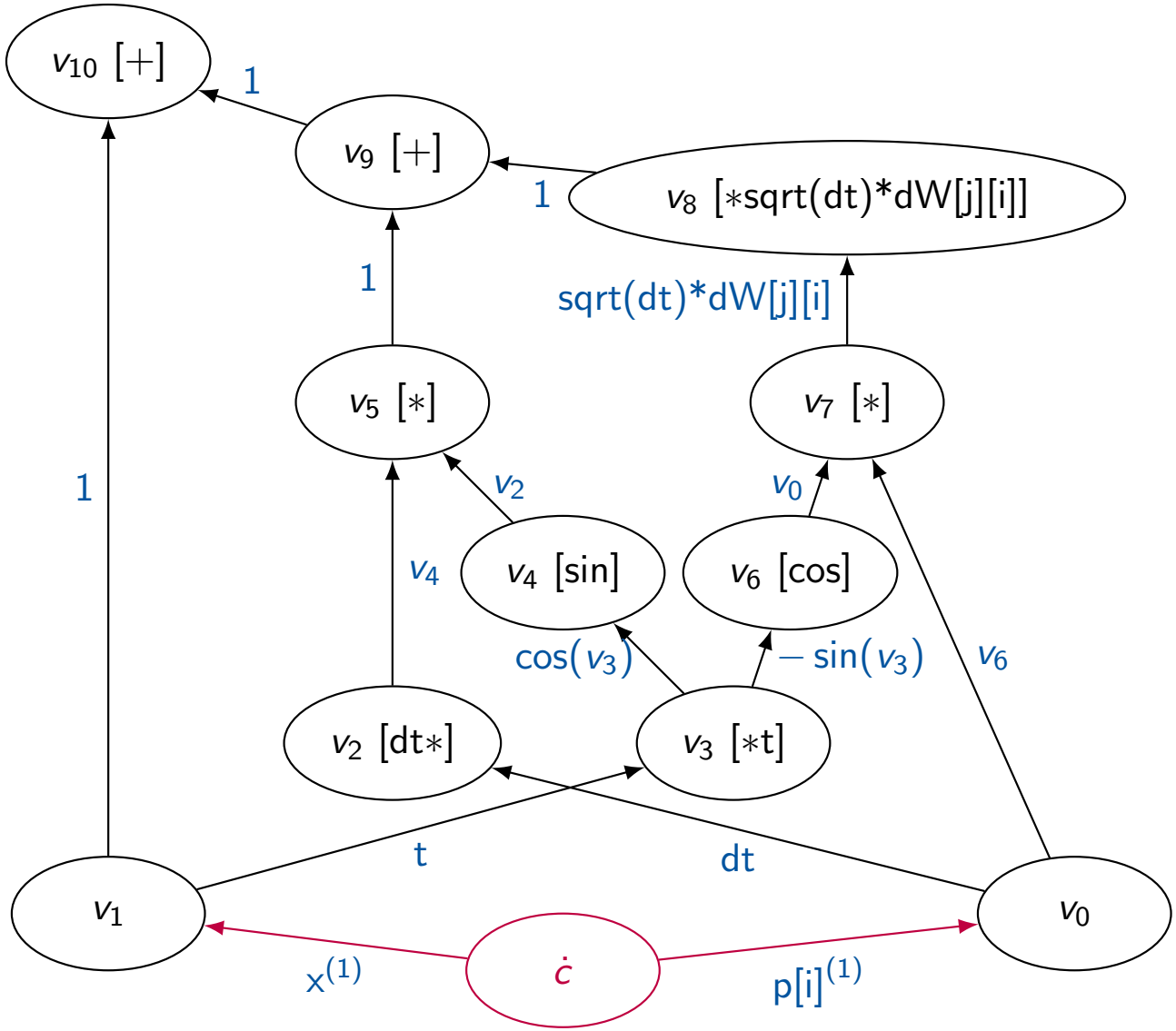
Hello AD World: Tangent SAC

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];

$v_0 = \text{p[i]}$	$v_0^{(1)} = \text{p[i]}^{(1)}$
$v_1 = \text{x}$	$v_1^{(1)} = \text{x}^{(1)}$
$v_2 = \text{dt} * v_0$	$v_2^{(1)} = \text{dt} * v_0^{(1)}$
$v_3 = v_1 * \text{t}$	$v_3^{(1)} = v_1^{(1)} * \text{t}$
$v_4 = \sin(v_3)$	$v_4^{(1)} = \cos(v_3) * v_3^{(1)}$
$v_5 = v_2 * v_4$	$v_5^{(1)} = v_2^{(1)} * v_4 + v_2 * v_4^{(1)}$
$v_6 = \cos(v_3)$	$v_6^{(1)} = -\sin(v_3) * v_3^{(1)}$
$v_7 = v_0 * v_6$	$v_7^{(1)} = v_0^{(1)} * v_6 + v_0 * v_6^{(1)}$
$v_8 = v_7 * \text{sqrt(dt)*dW[j][i]}$	$v_8^{(1)} = v_7^{(1)} * \text{sqrt(dt)*dW[j][i]}$
$v_9 = v_5 + v_8$	$v_9^{(1)} = v_5^{(1)} + v_8^{(1)}$
$v_{10} = v_1 + v_9$	$v_{10}^{(1)} = v_1^{(1)} + v_9^{(1)}$
$\text{x} = v_{10}$	$\text{x}^{(1)} = v_{10}^{(1)}$ // no increment of inoutput tangent

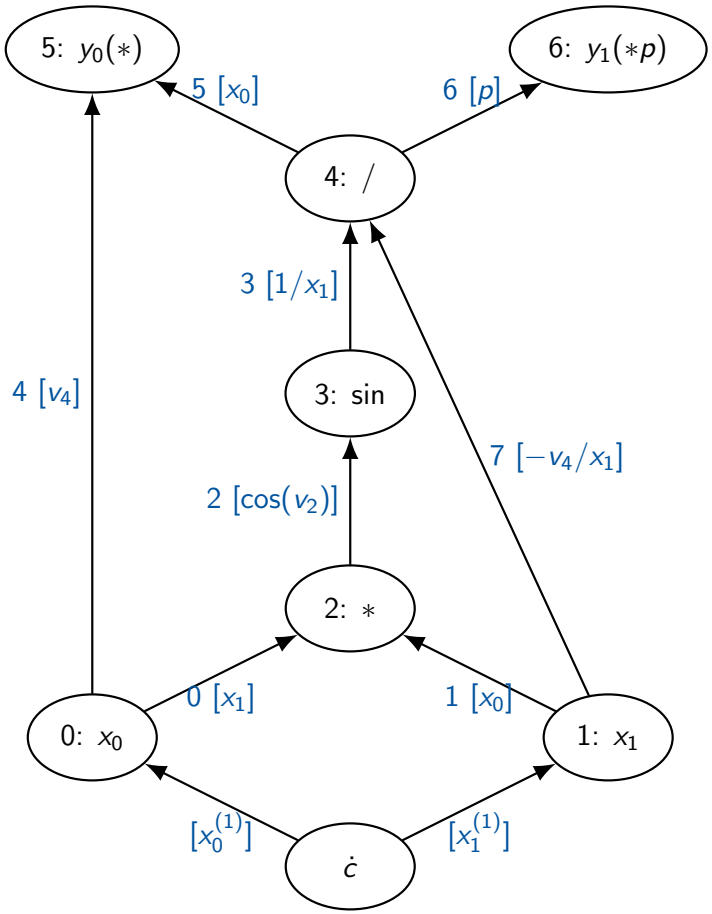
Hello AD World: Tangent DAG

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];



Tangent AD by Overloading (Movie)

Chain Rule on Tangent DAG



Tangent DAG

We consider

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 * \sin(x_0 * x_1)/x_1 \\ \sin(x_0 * x_1)/x_1 * p \end{pmatrix}$$

implemented as

$$t = \sin(x_0 * x_1)/x_1$$
$$y_0 = x_0 * t;\ y_1 = t * p$$

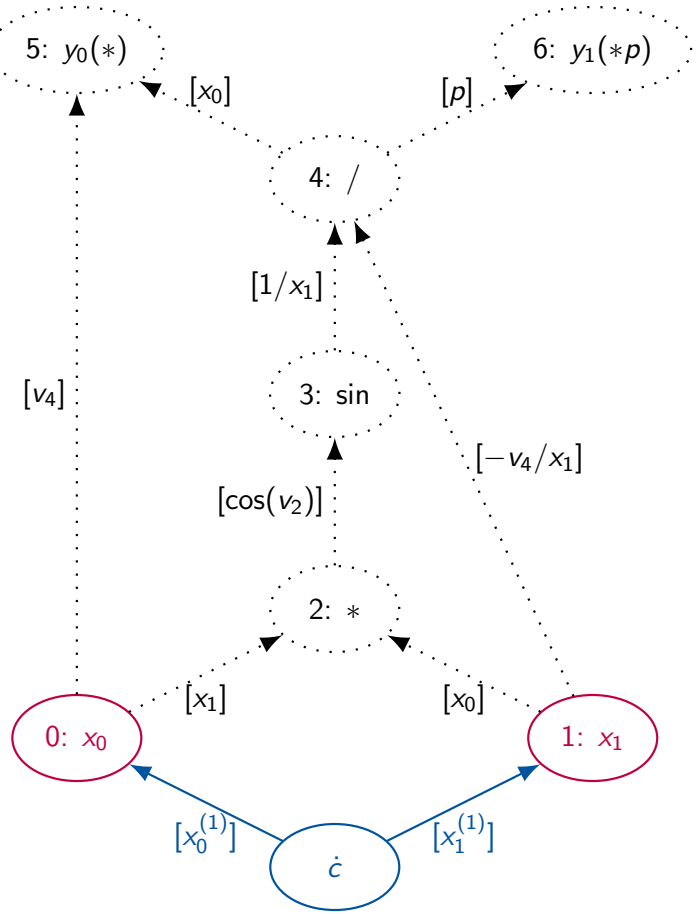
yielding SAC

$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_3/x_1$$
$$y_0 = x_0 * v_4;\ y_1 = v_4 * p$$

for some passive value p, i.e, no derivatives of or with respect to required; x, y, and t are active.

Tangent AD by Overloading (Movie)

Seed

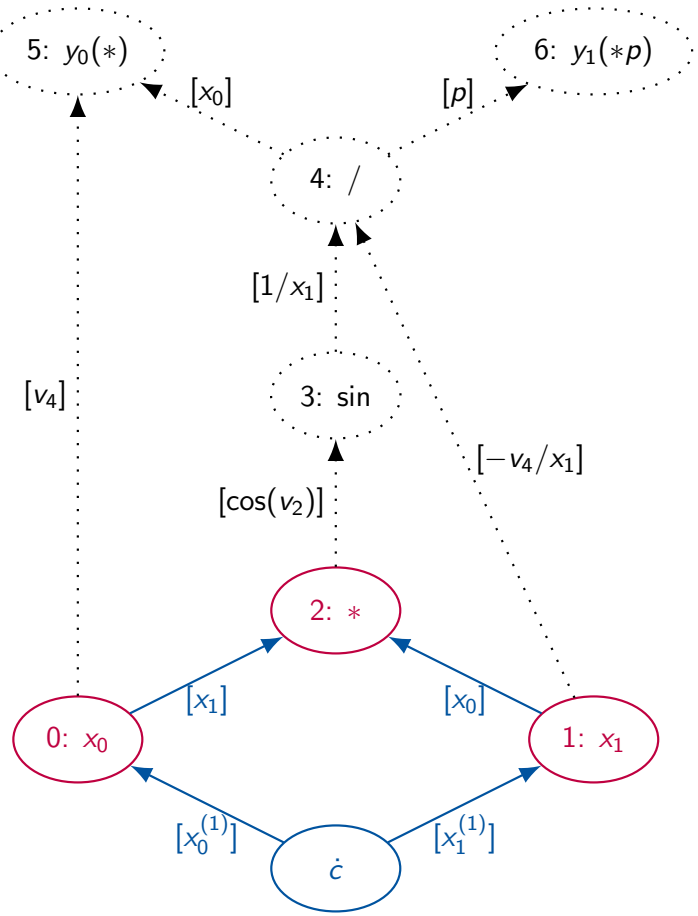


$$x_0 = ?$$
$$x_1 = ?$$

$$x_0^{(1)} = ?$$
$$x_1^{(1)} = ?$$

Tangent AD by Overloading (Movie)

Propagate (Local Directional Derivatives)



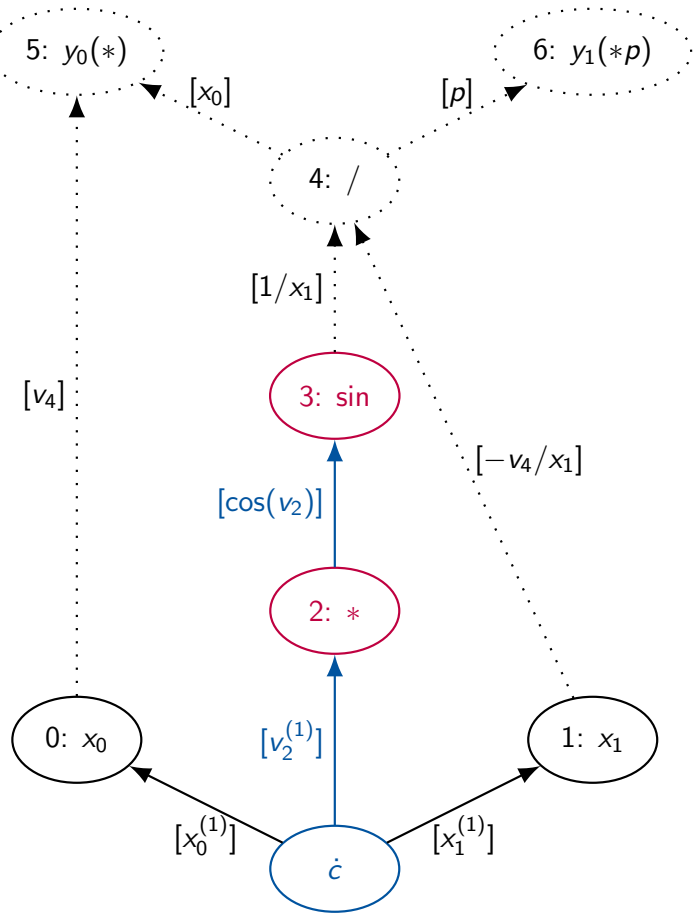
// overloaded $*$

$$v_2 = x_0 * x_1$$

$$v_2^{(1)} = x_1 * x_0^{(1)} + x_0 * x_1^{(1)}$$

Tangent AD by Overloading (Movie)

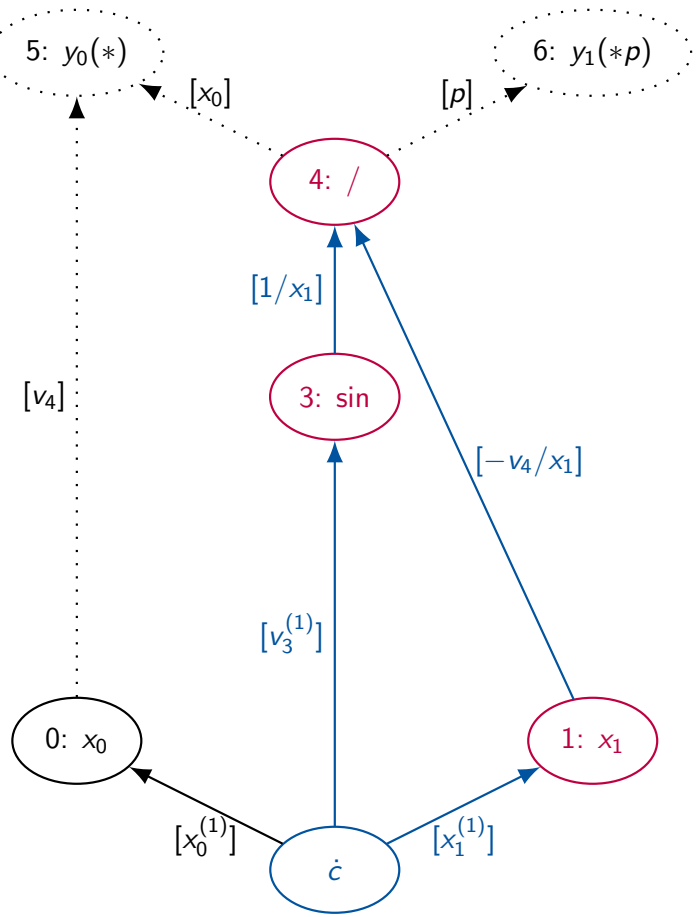
Propagate



$$v_2 = x_0 * x_1$$
$$v_2^{(1)} = x_1 * x_0^{(1)} + x_0 * x_1^{(1)}$$
// overloaded sin
$$v_3 = \sin(v_2)$$
$$v_3^{(1)} = \cos(v_2) * v_2^{(1)}$$

Tangent AD by Overloading (Movie)

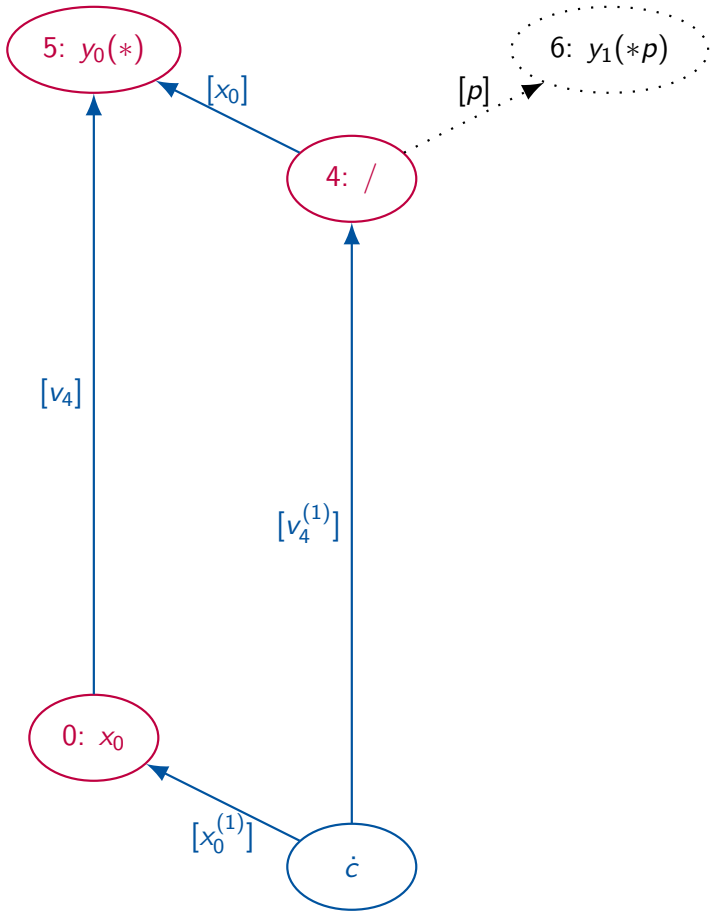
Propagate



$$v_2 = x_0 * x_1$$
$$v_2^{(1)} = x_1 * x_0^{(1)} + x_0 * x_1^{(1)}$$
$$v_3 = \sin(v_2)$$
$$v_3^{(1)} = \cos(v_2) * v_2^{(1)}$$
// overloaded /
$$v_4 = v_3/x_1$$
$$v_4^{(1)} = \left(v_3^{(1)} - v_4 * x_1^{(1)}\right)/x_1$$

Tangent AD by Overloading (Movie)

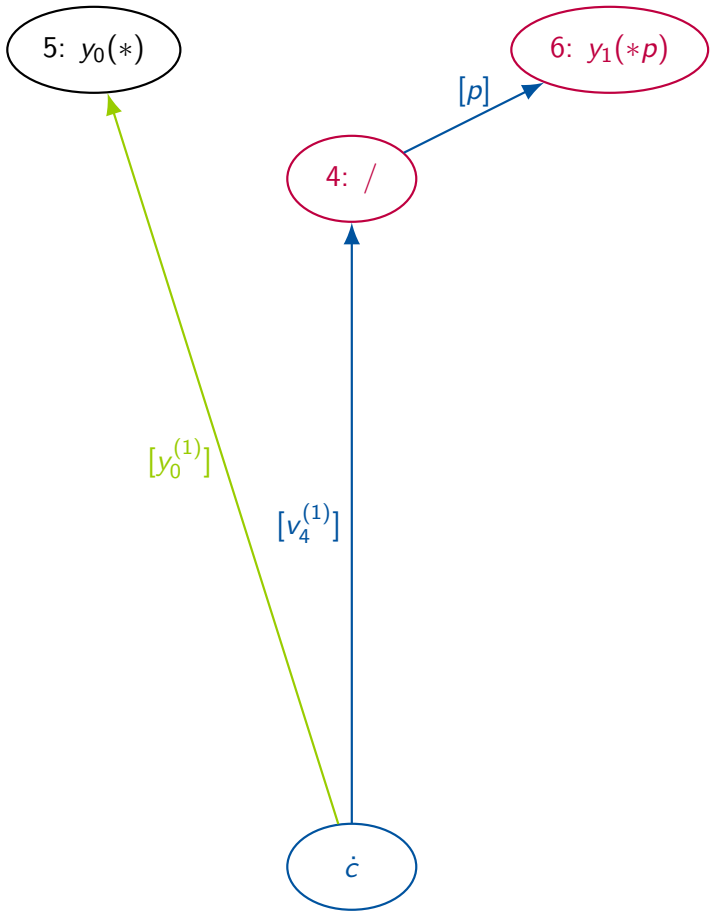
Propagate



$$v_2 = x_0 * x_1$$
$$v_2^{(1)} = x_1 * x_0^{(1)} + x_0 * x_1^{(1)}$$
$$v_3 = \sin(v_2)$$
$$v_3^{(1)} = \cos(v_2) * v_2^{(1)}$$
$$v_4 = v_3 / x_1$$
$$v_4^{(1)} = (v_3^{(1)} - v_4 * x_1^{(1)})/x_1$$
// overloaded $*$
$$y_0 = x_0 * v_4$$
$$y_0^{(1)} = y_0^{(1)} + v_4 * x_0^{(1)} + x_0 * v_4^{(1)}$$

Tangent AD by Overloading (Movie)

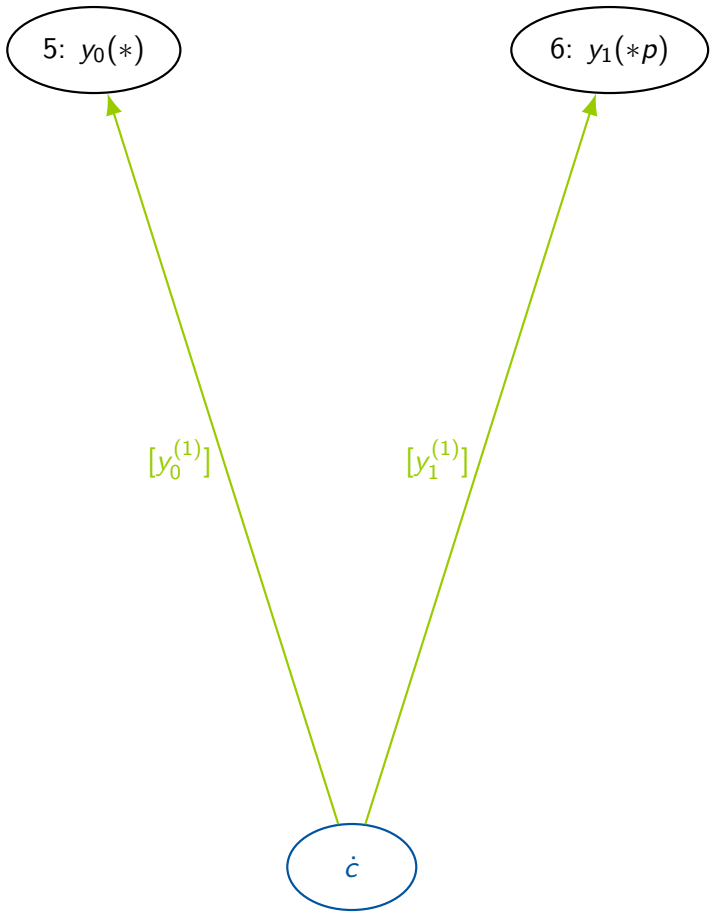
Propagate



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_2^{(1)} &= x_1 * x_0^{(1)} + x_0 * x_1^{(1)} \\
v_3 &= \sin(v_2) \\
v_3^{(1)} &= \cos(v_2) * v_2^{(1)} \\
v_4 &= v_3 / x_1 \\
v_4^{(1)} &= (v_3^{(1)} - v_4 * x_1^{(1)}) / x_1 \\
y_0 &= x_0 * v_4 \\
y_0^{(1)} &= y_0^{(1)} + v_4 * x_0^{(1)} + x_0 * v_4^{(1)} \\
&// \text{ overloaded } * \\
y_1 &= v_4 * p \\
y_1^{(1)} &= y_1^{(1)} + p * v_4^{(1)}
\end{aligned}
$$

Tangent AD by Overloading (Movie)

Harvest



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_2^{(1)} &= x_1 * x_0^{(1)} + x_0 * x_1^{(1)} \\
v_3 &= \sin(v_2) \\
v_3^{(1)} &= \cos(v_2) * v_2^{(1)} \\
v_4 &= v_3 / x_1 \\
v_4^{(1)} &= \left(v_3^{(1)} - v_4 * x_1^{(1)}\right) / x_1 \\
y_0 &= x_0 * v_4 \\
y_0^{(1)} &= y_0^{(1)} + v_4 * x_0^{(1)} + x_0 * v_4^{(1)} \\
y_1 &= v_4 * p \\
y_1^{(1)} &= y_1^{(1)} + p * v_4^{(1)}
\end{aligned}
$$

```
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 gradient of the final `x` with respect to the read-only parameters `p`.

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6
7
8
9
10
11
12
13
14
15
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7
8
9
10
11
12
13
14
15
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8
9
10
11
12
13
14
15
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9
10
11
12
13
14
15
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10
11
12
13
14
15   }
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11
12
13
14
15   }
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     dco::derivative(p[i])=1; // seed
12
13
14
15   }
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     dco::derivative(p[i])=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13
14
15   }
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     dco::derivative(p[i])=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     dxdp[i]=dco::derivative(x); // harvest
14
15   }
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     dco::derivative(p[i])=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     dxdp[i]=dco::derivative(x); // harvest
14     dco::derivative(p[i])=0; // unseed read-only input
15   }
16
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     dco::derivative(p[i])=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     dxdp[i]=dco::derivative(x); // harvest
14     dco::derivative(p[i])=0; // unseed read-only input
15   }
16   x_v=dco::value(x); // get primal function value
17 }
```

Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1s<T>::type; // scalar tangent type
7    size_t n=p_v.size(); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t i=0;i<n;i++) { // iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     dco::derivative(p[i])=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     dxdp[i]=dco::derivative(x); // harvest
14     dco::derivative(p[i])=0; // unseed read-only input
15   }
16   x_v=dco::value(x); // get primal function value
17 }
```

Potential incrementation of output tangents need to be implemented by the driver.

Vectors (also: ensembles) of k first-order tangents of

$$F : \boldsymbol{R}^n \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}^m \times \boldsymbol{R}^{m^*} : \quad (\mathrm{y}, \mathrm{y}^*) = F(\mathrm{x}, \mathrm{x}^*) \,,$$

can be evaluated in vector tangent AD mode as

$$Y^{(1)} = Y^{(1)} + F^{(1)}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*, X^{(1)}) \equiv Y^{(1)} + \frac{\partial F}{\partial \mathrm{x}}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*) \cdot X^{(1)}$$

for $X^{(1)} \in \boldsymbol{R}^{n \times k}$ and $Y^{(1)} \in \boldsymbol{R}^{m \times k}$.

Performance of vector tangent mode typically exceeds that of scalar tangent mode even without the highly desirable explicit parallelization / vectorization. The memory requirement is increased.

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2
3
4  template<typename T, typename PT> // gradient driver
5  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) {
6
7
8
9
10
11
12
13
14
15
16
17
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7
8
9
10
11
12
13
14
15
16
17
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8
9
10
11
12
13
14
15
16
17
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9
10
11
12
13
14
15
16
17
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10
11
12
13
14
15
16
17   }
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11
12
13
14
15
16
17   }
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     for (size_t i=0;i<N;i++) dco::derivative(p[j+i])[i]=1; // seed
12
13
14
15
16
17   }
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     for (size_t i=0;i<N;i++) dco::derivative(p[j+i])[i]=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13
14
15
16
17   }
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     for (size_t i=0;i<N;i++) dco::derivative(p[j+i])[i]=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     for (size_t i=0;i<N;i++) {
14       dxdp[j+i]=dco::derivative(x)[i]; // harvest
15
16     }
17   }
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     for (size_t i=0;i<N;i++) dco::derivative(p[j+i])[i]=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     for (size_t i=0;i<N;i++) {
14       dxdp[j+i]=dco::derivative(x)[i]; // harvest
15       dco::derivative(p[j+i])[i]=0; // unseed read-only inputs
16     }
17   }
18
19 }
```

Vector Tangent AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<int N=2, typename T, typename PT> // gradient driver
5  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) {
6    using DCO_T=typename dco::gt1v<T,N>::type; // vector tangent type
7    size_t n=p_v.size(); assert((n-2)%N==0); // size of gradient
8    DCO_T x; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
9    for (size_t j=0;j<n;j+=N) { // block-iterate over Cartesian basis
10     x=x_v; // [re]set inoutput
11     for (size_t i=0;i<N;i++) dco::derivative(p[j+i])[i]=1; // seed
12     f(ncp,ncs,eps,x,p,dW); // compute primal
13     for (size_t i=0;i<N;i++) {
14       dxdp[j+i]=dco::derivative(x)[i]; // harvest
15       dco::derivative(p[j+i])[i]=0; // unseed read-only inputs
16     }
17   }
18   x_v=dco::value(x); // get primal function value
19 }
```

Race

dco/c++ Scalar vs. Vector Tangent Mode


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // scalar tangent driver
5  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) {
6    ... size_t n=p_v.size(); ...
7  }
```

vs.

```
1  template<int N=2, typename T, typename PT> // vector tangent driver
2  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) {
3    ... size_t n=p_v.size(); assert((n-2)%N==0); ...
4  }
```

for varying n and N.

Outline

Application of the chain rule to $F : \boldsymbol{R}^n \to \boldsymbol{R}^m$ defined as

$$\mathrm{y} = F(\underbrace{G(\underbrace{H(\mathrm{x})}_{\mathrm{u}\in\boldsymbol{R}^p})}_{\mathrm{v}\in\boldsymbol{R}^q}) \quad \text{yields}$$

$$\frac{d\mathrm{y}}{d\mathrm{x}} = \frac{d\mathrm{y}}{d\mathrm{v}} \cdot \frac{d\mathrm{v}}{d\mathrm{u}} \cdot \frac{d\mathrm{u}}{d\mathrm{x}} = \frac{d\mathrm{y}}{d\mathrm{v}} \cdot \left(\frac{d\mathrm{v}}{d\mathrm{u}} \cdot \frac{d\mathrm{u}}{d\mathrm{x}}\right) = \left(\frac{d\mathrm{y}}{d\mathrm{v}} \cdot \frac{d\mathrm{v}}{d\mathrm{u}}\right) \cdot \frac{d\mathrm{u}}{d\mathrm{x}}$$

due to associativity of matrix multiplication.

However, assuming dense local Jacobians for $n = 10$, $p = 10$, $q = 10$, and $m = 1$, the respective operations counts (OC) compare as

$$\mathrm{OC}\left(\frac{d\mathrm{y}}{d\mathrm{v}} \cdot \left(\frac{d\mathrm{v}}{d\mathrm{u}} \cdot \frac{d\mathrm{u}}{d\mathrm{x}}\right)\right) = 1100 > 200 = \mathrm{OC}\left(\left(\frac{d\mathrm{y}}{d\mathrm{v}} \cdot \frac{d\mathrm{v}}{d\mathrm{u}}\right) \cdot \frac{d\mathrm{u}}{d\mathrm{x}}\right) .$$

Let

$$\underbrace{y}_{\in R^2} = F(G(H(I(\underbrace{x}_{\in R^2})))) \quad \Rightarrow \quad \frac{dy}{dx} = \underbrace{\frac{dy}{dw}}_{\in R^2} \cdot \underbrace{\frac{dw}{dv}}_{\in R} \cdot \underbrace{\frac{dv}{du}}_{\in R} \cdot \underbrace{\frac{du}{dx}}_{\in R^{1\times 2}}$$

$$\underbrace{\underbrace{\underbrace{I(x)}_{u\in R}}_{v\in R}}_{w\in R}$$

Note

$$\mathrm{OC}\left(\begin{pmatrix}\times\\ \times\end{pmatrix} \cdot [\times \cdot [\times \cdot (\times \quad \times)]]\right) = \mathrm{OC}\left(\left[\left[\begin{pmatrix}\times\\ \times\end{pmatrix} \cdot \times\right] \cdot \times\right] \cdot (\times \quad \times)\right) = 8$$

$$\mathrm{OC}\left(\begin{pmatrix}\times\\ \times\end{pmatrix} \cdot [[\times \cdot \times] \cdot (\times \quad \times)]\right) = \mathrm{OC}\left(\left[\begin{pmatrix}\times\\ \times\end{pmatrix} \cdot [\times \cdot \times]\right] \cdot (\times \quad \times)\right) = 7$$

Let $y = F(x) = F_2(F_1(x))$ with $F_1 : \boldsymbol{R}^n \to \boldsymbol{R}^k$ and $F_2 : \boldsymbol{R}^k \to \boldsymbol{R}^m$ with DAGs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, respectively. There are two alternatives for computing F':

$$F' = G_2 \cdot F_1' = G_2 \cdot (G_1 \cdot I_n) \quad \text{and} \quad F' = F_2' \cdot F_1' = (G_2 \cdot I_k) \cdot (G_1 \cdot I_n)$$

where $I_j \in \boldsymbol{R}^{j \times j}$ denotes the identity in $\boldsymbol{R}^j$.

Let $n = 100$, $k = 10$, $m = 1$ and $|E_2| = |E_1| = 1000$. Then

$$\begin{aligned} \mathrm{OC}(G_2 \cdot F_1') &= n \cdot |E_1| + n \cdot |E_2| \\ &= 100.000 + 100.000 = 200.000 \\ \mathrm{OC}(F_2' \cdot F_1') &= n \cdot |E_1| + k \cdot |E_2| + n \cdot k \cdot m \\ &= 100.000 + 10.000 + 1.000 = 111.000 \end{aligned}$$

assuming dense F_1' and F_2'.

Wake Up!

Modify n, k, and m such that $\mathrm{OC}(G_2 \cdot F_1') < \mathrm{OC}(F_2' \cdot F_1')$.

Outline

Summary

Prerequisites

Tangents

- Chain Rule
- Index Notation
- Jacobian-Free Tangents

Tangent AD

- Forward Mode of AD
- Hello AD World
- dco/c++
  - Scalar Tangent Mode
  - Vector Tangent Mode
  - Race

Toward AD Mission Planning

Summary