Algorithmic Diferentiation with dco/c++

Uwe Naumann

NAG Ltd. and RWTH Aachen University

The Chain Rule of Differentiation is Associative

Let

$$y = F(x) = f(g(h(x))) : \mathbb{R}^n \to \mathbb{R}^m \; .$$

The Chain Rule of Differentiation is Associative

Let

$$y = F(x) = f(g(h(x))) : \mathbb{R}^n \to \mathbb{R}^m \, .$$

If f, g, h are differentiable with Jacobians $f' = f'(g), g' = g'(h), h' = h'(x)$, then so is F and

$$F' = F'(x) = f' \cdot g' \cdot h' \, .$$

The Chain Rule of Differentiation is Associative

Let

$$y = F(x) = f(g(h(x))) : \mathbb{R}^n \to \mathbb{R}^m \ .$$

If f, g, h are differentiable with Jacobians $f' = f'(g), g' = g'(h), h' = h'(x)$, then so is F and

$$F' = F'(x) = f' \cdot g' \cdot h' \ .$$

Matrix multiplication is associative, that is, in infinite precision arithmetic

$$F' = f' \cdot (g' \cdot h') = (f' \cdot g') \cdot h' \ .$$

E.g., $y = F(x) = e^{\sin(x^2)} \Rightarrow F'(x) = y \cdot (\cos(x^2) \cdot 2x) = (y \cdot \cos(x^2)) \cdot 2x$

So What?

The order matters (for computational cost)!

Different orders induce different costs, e.g., for $a_i \in \mathbb{R}^n$ (Note $F : \mathbb{R} \to \mathbb{R}^n$)

$$a_5 \cdot a_4^T \cdot a_3 \cdot a_2^T \cdot a_1 = a_5 \cdot (a_4^T \cdot (a_3 \cdot (a_2^T \cdot a_1))) \qquad \Rightarrow \mathcal{O}(n)$$

$$| \; (\; - \; (\; | \; (\; -| \;) \;) \;)$$

So What?

The order matters (for computational cost)!

Different orders induce different costs, e.g., for $a_i \in \mathbb{R}^n$ (Note $F : \mathbb{R} \to \mathbb{R}^n$)

$$a_5 \cdot a_4^T \cdot a_3 \cdot a_2^T \cdot a_1 = a_5 \cdot (a_4^T \cdot (a_3 \cdot (a_2^T \cdot a_1))) \qquad \Rightarrow \mathcal{O}(n)$$

$$|\;(\;-\;(\;|\;(\;-|\;)\;)\;)$$

$$= (((a_5 \cdot a_4^T) \cdot a_3) \cdot a_2^T) \cdot a_1 \qquad \Rightarrow \mathcal{O}(n^2)$$

$$(\;(\;(\;|-\;)\;|\;)\;-|$$

So What?

The order matters (for computational cost)!

Different orders induce different costs, e.g., for $a_i \in \mathbb{R}^n$ (Note $F : \mathbb{R} \to \mathbb{R}^n$)

$$a_5 \cdot a_4^T \cdot a_3 \cdot a_2^T \cdot a_1 = a_5 \cdot (a_4^T \cdot (a_3 \cdot (a_2^T \cdot a_1))) \qquad \Rightarrow \mathcal{O}(n)$$

$$|\;(\;-\;(\;|\;(\;-|\;)\;)\;)$$

$$= (((a_5 \cdot a_4^T) \cdot a_3) \cdot a_2^T) \cdot a_1 \qquad \Rightarrow \mathcal{O}(n^2)$$

$$(\;(\;(\;|-\;)\;|\;)\;-|$$

$$= ((a_5 \cdot a_4^T) \cdot (a_3 \cdot a_2^T)) \cdot a_1 \qquad \Rightarrow \mathcal{O}(n^3)\;.$$

$$(\;(\;|-\;)\;(|-\;)\;)\;|$$

So What?

The order matters (for computational cost)!

Different orders induce different costs, e.g., for $a_i \in \mathbb{R}^n$ (Note $F : \mathbb{R} \to \mathbb{R}^n$)

$$a_5 \cdot a_4^T \cdot a_3 \cdot a_2^T \cdot a_1 = a_5 \cdot (a_4^T \cdot (a_3 \cdot (a_2^T \cdot a_1))) \qquad \Rightarrow \mathcal{O}(n)$$

$$| \; (\; - \; (\; | \; (\; -| \;) \;) \;)$$

$$= (((a_5 \cdot a_4^T) \cdot a_3) \cdot a_2^T) \cdot a_1 \qquad \Rightarrow \mathcal{O}(n^2)$$

$$(\; (\; (\; |- \;) \; | \;) \; -|$$

$$= ((a_5 \cdot a_4^T) \cdot (a_3 \cdot a_2^T)) \cdot a_1 \qquad \Rightarrow \mathcal{O}(n^3) \; .$$

$$(\; (\; |- \;) \; (|- \;) \;) \; |$$

... and what about numerical stability?

A. Griewank, K. Kulshreshtha and A. Walther: On the numerical stability of algorithmic differentiation. Computing 94, pages 125–149, 2012.
U. N.: *Numerical Stability of Tangents and Adjoints of Implicit Functions.* ICCS 2022, pages 181–187.

Introduction

- Member (since 2008) and Principal Scientist (2022), NAG Ltd., Oxford, UK
- Visiting Lecturer, University of Oxford, UK (2017)
- Professor of Computer Science, RWTH Aachen University, Aachen, Germany (since 2004)
- Assistant Computer Scientist, Argonne National Laboratory, Argonne, IL, US (2002–2004)
- Visiting Scientist, MIT, Cambridge, MA, US (2001)
- Senior Lecturer for Computer Science, University of Hertfordshire (UHerts), Hatfield, UK (2000–2001)
- Postdoctoral Researcher, INRIA, Sophia-Antipolis, France (1999–2000)
- MSc/PhD in Applied Mathematics, Technical University Dresden, Germany, University of York, UK, Chuo University, Tokyo. Japan, UHerts, UK (1990–1999, supervisor: Andreas Griewank)
- Military service in Bad Düben, East Germany (1988–1990)
- went to school in Chemnitz, East Germany and Dubna, Russia (1976–1988)

Contents

Outline

- Motivation
  - Computational cost of gradients
  - Training artificial neural networks
  - Beyond backpropagation
- Playground
- Typical workflow
- Prerequisites for Algorithmic Differentiation (AD)
- Tangents / Tangent AD (with dco/c++) / Finite differences
- Adjoints / Adjoint AD (with dco/c++)
- Outlook: AD mission planning
- Differential invariant
- Hands-on: Heston stochastic volatility model

Setting the Stage

We consider sufficiently often differentiable computer programs implementing multivariate vector functions

$$F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^m : y = F(x,p)$$

with Jacobians

$$F' = F'(x,p) = (y_x, y_p) \equiv \left(\frac{dF}{dx}, \frac{dF}{dp}\right) \in \mathbb{R}^{m \times (n_x+n_p)}$$

Hessians

$$F'' = F''(x,p) = \begin{pmatrix} y_{xx} & y_{xp} \\ y_{px} & y_{pp} \end{pmatrix} \in \mathbb{R}^{m \times (n_x+n_p) \times (n_x+n_p)}$$

and corresponding higher derivative tensors, for example

```
std::vector<double> F(const std::vector<double>& x, const std::vector<double>& p);
```

p is omitted whenever possible to simplify the notation $\Rightarrow y = F(x)$.

Motivation: Computational Cost of Gradients

$$f : \mathbb{R}^n \to \mathbb{R} \quad \Rightarrow \quad f' \in \mathbb{R}^{1 \times n}$$

	n	Ert (s)	Rss (MB)
primal	200	0.2	4
central finite differences	200	99.3	4
scalar tangent AD	200	52.9	4
black-box adjoint AD	200	0.8	1,061
vector tangent AD	200	5.8	4

Motivation: Computational Cost of Gradients

$$f : \mathbb{R}^n \to \mathbb{R} \quad \Rightarrow \quad f' \in \mathbb{R}^{1\times n}$$

	n	ERT (s)	RSS (MB)
primal	200	0.2	4
central finite differences	200	99.3	4
scalar tangent AD	200	52.9	4
black-box adjoint AD	200	0.8	1,061
vector tangent AD	200	5.8	4
primal	400	0.5	4
black-box adjoint AD	400	-	$> 16,000$
vector tangent AD	400	23.5	5
beyond black-box adjoint AD	400	1.5	5

Notes: ERT : elapsed run time; RSS : maximum resident set size; Generalized Geometric Brownian Motion example with $n_p = 10^5$, $n_s = 100, 200$

(Dense feed-forward deep) artificial neural networks

$$y = F(F(\ldots F(F(x, p_0), p_1)\ldots), p_{q-1})$$

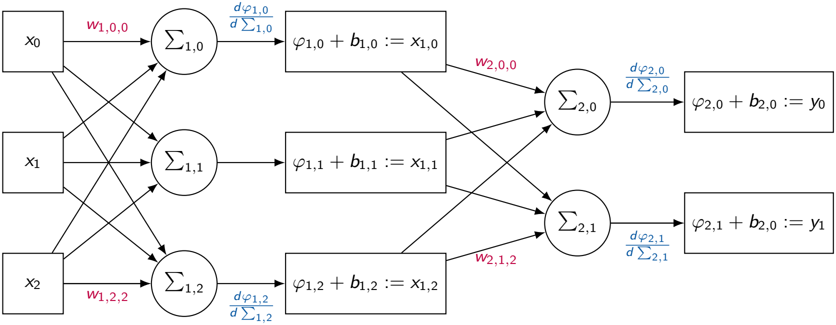
fall into this category with

$$F_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x^2+n_x} \to \mathbb{R}^{n_x} : x_i = F_i(x_{i-1}, p_{i-1})$$

evolving states x_i depending on parameters (weights, biases) p_i for $i = 1, \ldots, q$ and where

$$\begin{aligned}
x_0 &= x \in \mathbb{R}^{n_x} \\
y &= x_q \in \mathbb{R}^m \\
n_p &= (q-1) \cdot (n_x^2 + n_x) + n_x \cdot m + m = ((q-1) \cdot n_x + m) \cdot (n_x + 1) \\
p &= (p_0, \ldots, p_{q-1})^T \in \mathbb{R}^{n_p} \\
m &= \mathcal{O}(n_x) \;\Rightarrow\; n_p = \mathcal{O}(q \cdot n_x^2)
\end{aligned}$$

Motivation: Artifical Neural Network Example



$q = 2,\ n_x = 3,\ m = 2,\ n_p = 20$

Motivation: Training

Training amounts to estimation of p, e.g., to minimize the objective

$$f(X, p, Y) \equiv \sum_{j=0}^{d} \|F(x^j, p) - y^j\|$$

for given observations / data

$$(X, Y) = ((x^j), (y^j)) \in \mathbb{R}^{n_x \times d} \times \mathbb{R}^{m \times d} \ .$$

Most numerical optimization methods require (at least) the typically very large gradient

$$f_p = (f_{p_0} \dots f_{p_{q-1}}) \in \mathbb{R}^{q \cdot n_p} \ .$$

Motivation: Backpropagation

According to the chain rule of differentiation

$$\begin{aligned} \frac{df}{dx_{i-1}} &= \frac{df}{dx_i} \cdot \frac{dF_i}{dx_{i-1}} \\ \frac{df}{dp_{i-1}} &= \frac{df}{dx_i} \cdot \frac{dF_i}{dp_{i-1}} \end{aligned} \quad \text{for } i = q, \ldots, 1. \tag{1}$$

The local Jacobians $\frac{dF_i}{dx_{i-1}}$ and $\frac{dF_i}{dp_{i-1}}$ as well as $\frac{df}{dx_q}$ can typically be derived symbolically.

According to the chain rule of differentiation

$$\begin{aligned} \frac{df}{dx_{i-1}} &= \frac{df}{dx_i} \cdot \frac{dF_i}{dx_{i-1}} \\ \frac{df}{dp_{i-1}} &= \frac{df}{dx_i} \cdot \frac{dF_i}{dp_{i-1}} \end{aligned} \quad \text{for } i = q, \ldots, 1. \tag{1}$$

The local Jacobians $\frac{dF_i}{dx_{i-1}}$ and $\frac{dF_i}{dp_{i-1}}$ as well as $\frac{df}{dx_q}$ can typically be derived symbolically.

Bracketing of the resulting Jacobian chain products

$$\frac{df}{dp_{i-1}} = \frac{df}{dx_q} \cdot \frac{dF_q}{dx_{q-1}} \cdot \ldots \cdot \frac{dF_{i+1}}{dx_i} \cdot \frac{dF_i}{dp_{i-1}}$$

from right to left yields a (prohibitive) computational cost of $\mathcal{O}(q \cdot n_x^4)$
Bracketing from left to right (backpropagation as in Eq. (1) reduces the computational cost to (manageable) $\mathcal{O}(q \cdot n_x^3)$.

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

$$y = F_3(F_2(\underbrace{\underbrace{F_1(x)}_{u \in \mathbb{R}^p})}_{v \in \mathbb{R}^q})) \quad \text{yields}$$

$$\frac{dy}{dx} = \frac{dy}{dv} \cdot \frac{dv}{du} \cdot \frac{du}{dx} = \frac{dy}{dv} \cdot \left(\frac{dv}{du} \cdot \frac{du}{dx} \right) = \left(\frac{dy}{dv} \cdot \frac{dv}{du} \right) \cdot \frac{du}{dx}$$

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 (OPS; e.g. measured in terms of fused multiply-adds) compare as

$$OPS\left(\frac{dy}{dv} \cdot \left(\frac{dv}{du} \cdot \frac{du}{dx} \right) \right) = 1100 > 200 = OPS\left(\left(\frac{dy}{dv} \cdot \frac{dv}{du} \right) \cdot \frac{du}{dx} \right) .$$

Adjoint algorithmic differentiation (AAD) generalizes backpropagation for arbitrary differentiable programs.

The C++ library dco/c++ supports algorithmic differentiation (AD) more generally including

1. first-order tangents and adjoints over generic base data types
2. combinations thereof at arbitrary levels of the target code
3. second- and higher-order tangents and adjoints.

Very good performance can be achieved.

Let

$$\underbrace{y}_{\in\mathbb{R}^2} = F_4(\underbrace{F_3(\underbrace{F_2(\underbrace{F_1(\underbrace{x}_{\in\mathbb{R}^2})}_{u\in\mathbb{R}})}_{v\in\mathbb{R}})}_{w\in\mathbb{R}}) \quad\Rightarrow\quad \frac{dy}{dx} = \underbrace{\frac{dy}{dw}}_{\in\mathbb{R}^2} \cdot \underbrace{\frac{dw}{dv}}_{\in\mathbb{R}} \cdot \underbrace{\frac{dv}{du}}_{\in\mathbb{R}} \cdot \underbrace{\frac{du}{dx}}_{\in\mathbb{R}^{1\times 2}}$$

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

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

See also: Vertex, edge, face elimination

All tests are performed on a Dell Precision 3530 mobile workstation with the following hardware specifications

- Intel Xeon E-2176M CPU @ 2.70GHz
- cpu cores : 6
- cpu threads : 12
- cache size : 12MB
- RAM size : 16GB
- hard disc : 1TB ssd

and running Ubuntu 20.04LTS.

Examples and live coding sessions are based on an implementation of the L_2^2-norm of the numerical approximation of the expected solution $\mathbb{E}(x) \in \mathbb{R}$ of a parameterized scalar stochastic initial value problem.

We consider the generalized Geometric Brownian Motion (gGBM) described by the stochastic differential equation (SDE)

$$dx = f_1(x(p_1(t), t), p_1(t), t))dt + f_2(x(p_2(t), t), p_2(t), t)dW$$

with drift and volatility parameterized by time-dependent $p_1 = p_1(t)$ and $p_2 = p_2(t)$, respectively, initial condition $x(0) = x^0$, unit target time $t = 1$ and Wiener Process dW.

Application of forward finite differences in time with time step $0 < \Delta t \ll 1$ to the SDE yields the Euler-Maruyama scheme

$$x^{i+1} = x^i + \Delta t \cdot f_1(x^i, p_1^i, i \cdot \Delta t) + \sqrt{\Delta t} \cdot f_2(x^i, p_2^i, i \cdot \Delta t) \cdot dW^i$$

for $i = 0, \ldots, n_s - 1$, $n_s > 0$, target time $n_s \cdot \Delta t = 1$, parameter vectors $p_j = (p_j^i) \in \mathbb{R}^{n_s}$, $j = 1, 2$, and with random numbers dW^i drawn from the standard normal distribution $N(0, 1)$.

The solution $\mathbb{E}(x)$ is approximated using Monte Carlo simulation over $n_p > 0$ Euler-Maruyama paths.

We are interested in first- and second-order sensitivities of

$$y = \|\mathbb{E}(x)\|_2^2 = \mathbb{E}(x)^2$$

wrt. x^0, p_1, and p_2 to simulate use in the context of calibration.

Live: Playground

`SDE/main.cpp`

- inspect
  - `SDE/f.h` using f1 and f2
  - `SDE/f12.h` in modes ({A,B},{C,D})
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)

np	ns	Ert (s)	Rss (MB)
10^5	10	< 0.1	4
10^5	100	0.2	4
10^5	1000	2.5	4
10^6	10	0.2	4
10^6	100	2.5	4
10^6	1000	24.4	4

Scenario BC

Hands-on: Run and time on your computer.

Typical Work Flow

1. Finite difference approximation (given)

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording
5. Optimized adjoint (efficient scalable cheap gradients)

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording
5. Optimized adjoint (efficient scalable cheap gradients)
   5.1 (local) symbolic adjoints / extended set of elementals

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording
5. Optimized adjoint (efficient scalable cheap gradients)
   5.1 (local) symbolic adjoints / extended set of elementals
   5.2 (local) parallelization / vectorization

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording
5. Optimized adjoint (efficient scalable cheap gradients)
   5.1 (local) symbolic adjoints / extended set of elementals
   5.2 (local) parallelization / vectorization
   5.3 (local) adjoint code generation

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording
5. Optimized adjoint (efficient scalable cheap gradients)
   5.1 (local) symbolic adjoints / extended set of elementals
   5.2 (local) parallelization / vectorization
   5.3 (local) adjoint code generation
   5.4 full combinatorics

Typical Work Flow

1. Finite difference approximation (given)
2. Tangent (for accuracy, validation)
3. Black-box adjoint (cheap gradients)
4. Feasible adjoint (scalable cheap gradients)
   4.1 pathwise adjoints; early back-propagation
   4.2 preaccumulation; early/late preaccumulation
   4.3 checkpointing; late recording
5. Optimized adjoint (efficient scalable cheap gradients)
   5.1 (local) symbolic adjoints / extended set of elementals
   5.2 (local) parallelization / vectorization
   5.3 (local) adjoint code generation
   5.4 full combinatorics
6. Higher-order tangents and adjoints

- Single assignment code
- Directed acyclic graph
- Chain rule of differentiation
- Tangents
- Approximate Tangents (Finite Differences)
- Tangent AD with dco/c++
- Adjoints
- Adjoint AD with dco/c++

For given values of its inputs all differentiable programs decompose into sequences of $q = p + m$ differentiable elemental functions (also: elementals) φ_j evaluated conceptually as a single assignment code (SAC)

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

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

```
void F(std::vector<T>& x) {
  while (x[1]<=0) x[1]=exp(x[0]*x[1])+x[1];
  x[0]=x[0]/x[1];
  x[1]=pow(x[1],3);
}
```

called with `x[0]=1; x[1]=0` ⇒ single iteration of **while**-loop.

Directed Acyclic Graph

The data dependences within a differential program $F = F(x)$ induce a directed acyclic graphs (DAG), in the following referred to as the tape $T = T(F, x) = (V, E)$ with integer vertices $i \in V = \{1, \ldots, |V|\}$ representing (instances of vectors of program) variables v_i and edges $(i, j) \in E \subseteq V \times V$.

The flow of control in F is determined by the given x.

The data dependences within a differential program $F = F(x)$ induce a directed acyclic graphs (DAG), in the following referred to as the tape $T = T(F, x) = (V, E)$ with integer vertices $i \in V = \{1, \ldots, |V|\}$ representing (instances of vectors of program) variables v_i and edges $(i, j) \in E \subseteq V \times V$.

The flow of control in F is determined by the given x.

The elementals induce a cover of T by bipartite subdags. Variables can be read but not written by different elementals, that is, edges emanating from a vertex can belong to distinct elementals. All incoming edges of a vertex are part of the same elemental.

In the simplest case, all elementals are scalar functions mapping the variables that correspond to the $|P_j|$ predecessors of vertex $j \in V$ to v_j.

Edges $(i, j) \in E$ are labelled with local partial derivatives $d_{j,i} \equiv \frac{dv_j}{dv_i}$.

Tape Example

$$F : \mathbb{R}^2 \to \mathbb{R}^2$$

```
void F(std::vector<T>& x) {
  while (x[1]<=0)
    x[1]=exp(x[0]*x[1])+x[1];
  x[0]=x[0]/x[1];
  x[1]=pow(x[1],3);
}
```

$$v_1 = x_0$$
$$v_2 = x_1$$

$$v_3 = v_1 \cdot v_2$$
$$v_4 = e^{v_3}$$
$$v_5 = v_4 + v_2$$
$$v_6 = v_1 / v_5$$
$$v_7 = v_5^3$$

$$y_0 = v_6$$
$$y_1 = v_7$$

Tape Example

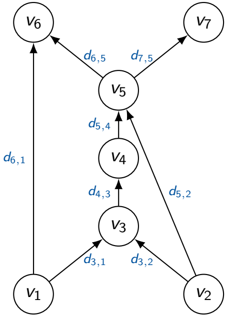
$$F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

```
void F(std::vector<T>& x) {
  while (x[1]<=0)
    x[1]=exp(x[0]*x[1])+x[1];
  x[0]=x[0]/x[1];
  x[1]=pow(x[1],3);
}
```

$$v_1 = x_0$$
$$v_2 = x_1$$

$$v_3 = v_1 \cdot v_2$$
$$v_4 = e^{v_3}$$
$$v_5 = v_4 + v_2$$
$$v_6 = v_1 / v_5$$
$$v_7 = v_5^3$$

$$y_0 = v_6$$
$$y_1 = v_7$$



Tape Example

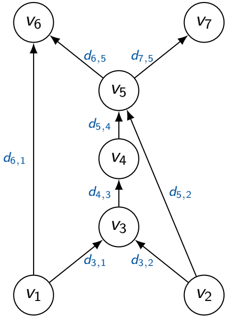
$$F : \mathbb{R}^2 \to \mathbb{R}^2$$

$$v_1 = x_0$$
$$v_2 = x_1$$

$$v_3 = v_1 \cdot v_2$$
$$v_4 = e^{v_3}$$
$$v_5 = v_4 + v_2$$
$$v_6 = v_1 / v_5$$
$$v_7 = v_5^3$$

$$y_0 = v_6$$
$$y_1 = v_7$$

```
void F(std::vector<T>& x)
    {
  while (x[1]<=0)
    x[1]=exp(x[0]*x[1])+x
       [1];
  x[0]=x[0]/x[1];
  x[1]=pow(x[1],3);
}
```



$$F : \mathbb{R}^{200} \to \mathbb{R}^2$$

$$v_1 = (x_0 \ldots x_{99})^T$$
$$v_2 = (x_{100} \ldots x_{199})^T$$

$$v_3 = v_1^T \cdot v_2$$
$$v_4 = e^{v_3}$$
$$v_5 = v_4 \cdot v_2$$
$$\begin{pmatrix} v_6 \\ v_7 \end{pmatrix} = \begin{pmatrix} v_1^T \cdot v_5 \\ v_5^T \cdot v_5 \end{pmatrix}$$

$$y_0 = v_6$$
$$y_1 = v_7$$

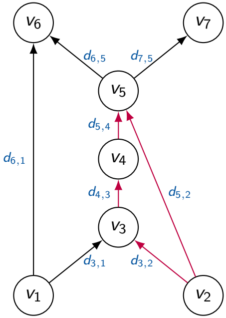
Chain Rule on Tape

The chain rule

$$F'(x) \equiv \frac{dy}{dx} = \sum_{\text{path}(x \to y) \in T(F,x)} \prod_{(i,j) \in \text{path}} \frac{\partial v_j}{\partial v_i}$$

applies to arbitrary pairs of vertices and corresponding subsets of V / subprograms of F, e.g.

$$\frac{dv_5}{dv_2} = d_{3,2} \cdot d_{4,3} \cdot d_{5,4} + d_{5,2}$$

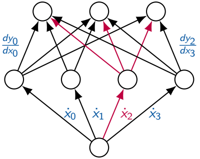


Let $y = F(x)$. Tangents compute directional derivatives

$$\mathbb{R}^m \ni \dot{y} = \dot{F}(x, \dot{x}) \equiv F'(x) \cdot \dot{x} \;,$$

without prior accumulation of the Jacobian $F'(x)$.

$$\begin{pmatrix} \dot{y}_0 \\ \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} \frac{dy_0}{dx_0} & \frac{dy_0}{dx_1} & \frac{dy_0}{dx_2} & \frac{dy_0}{dx_3} \\ \frac{dy_1}{dx_0} & \frac{dy_1}{dx_1} & \frac{dy_1}{dx_2} & \frac{dy_1}{dx_3} \\ \frac{dy_2}{dx_0} & \frac{dy_2}{dx_1} & \frac{dy_2}{dx_2} & \frac{dy_2}{dx_3} \end{pmatrix} \cdot \begin{pmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix}$$

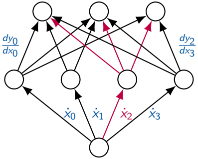


Let $y = F(x)$. Tangents compute directional derivatives

$$\mathbb{R}^m \ni \dot{y} = \dot{F}(x, \dot{x}) \equiv F'(x) \cdot \dot{x} \ ,$$

without prior accumulation of the Jacobian $F'(x)$.

$$\begin{pmatrix} \dot{y}_0 \\ \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} \frac{dy_0}{dx_0} & \frac{dy_0}{dx_1} & \frac{dy_0}{dx_2} & \frac{dy_0}{dx_3} \\ \frac{dy_1}{dx_0} & \frac{dy_1}{dx_1} & \frac{dy_1}{dx_2} & \frac{dy_1}{dx_3} \\ \frac{dy_2}{dx_0} & \frac{dy_2}{dx_1} & \frac{dy_2}{dx_2} & \frac{dy_2}{dx_3} \end{pmatrix} \cdot \begin{pmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix}$$



Implementations of tangent AD typically augment the evaluation of F with $\dot{F}$.

The Jacobian F' is accumulated column-wise by letting $\dot{x}$ range over the Cartesian basis vectors in $\mathbb{R}^n$.

Potential sparsity of the Jacobian can and should be exploited, e.g. 5×5 band structure.

Tangent AD propagates tangents of all elementals evaluated by the primal SAC yielding the augmented primal SAC

$$i = 1, \ldots, n: \quad \begin{pmatrix} v_i \\ \dot{v}_i \end{pmatrix} = \begin{pmatrix} x_{i-1} \\ \dot{x}_{i-1} \end{pmatrix}$$

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

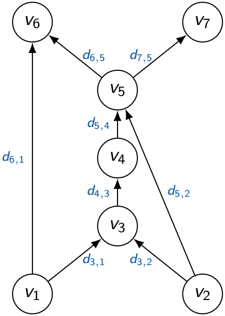
$$k = 1, \ldots, m: \quad \begin{pmatrix} y_{k-1} \\ \dot{y}_{k-1} \end{pmatrix} = \begin{pmatrix} v_{n+p+k} \\ \dot{v}_{n+p+k} \end{pmatrix}$$

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 $(v, \dot{v})$ comprising both value and tangent. Explicit storage of the tape is not necessary.

Tangent AD Example

Elementals are augmented with their tangents, e.g.

$$
\begin{array}{llll}
v_1 = x_0; & & \dot{v}_1 = \dot{x}_0 \\
v_2 = x_1; & & \dot{v}_2 = \dot{x}_1 \\
v_3 = v_1 \cdot v_2; & d_{3,1} = v_2; & \dot{v}_3 = d_{3,1} \cdot \dot{v}_1 + d_{3,2} \cdot \dot{v}_2 \\
 & d_{3,2} = v_1 & \\
v_4 = e^{v_3}; & d_{4,3} = v_4; & \dot{v}_4 = d_{4,3} \cdot \dot{v}_3 \\
v_5 = v_4 + v_2; & d_{5,4} = 1; & \dot{v}_5 = d_{5,4} \cdot \dot{v}_4 + d_{5,2} \cdot \dot{v}_2 \\
 & d_{5,2} = 1 & \\
v_6 = v_1 / v_5; & d_{6,1} = 1/v_5; & \dot{v}_6 = d_{6,1} \cdot \dot{v}_1 + d_{6,5} \cdot \dot{v}_5 \\
 & d_{6,5} = -v_1/v_5^2 & \\
v_7 = v_5^3; & d_{7,5} = 2 \cdot v_5^2; & \dot{v}_7 = d_{7,5} \cdot \dot{v}_5 \\
y_0 = v_6; & & \dot{y}_0 = \dot{v}_6 \\
y_1 = v_7; & & \dot{y}_1 = \dot{v}_7
\end{array}
$$

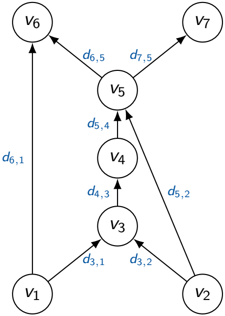


Cost of Tangent AD

$$\textsc{Mem}(F) = \textsc{Mem}(\dot{F}) = 0$$

$$\textsc{Ert}(F) \sim \textsc{Ops}(F) = |V| - n = 7 - 2 = 5$$

$$\textsc{Ert}(\dot{F}) \sim \textsc{Ops}(\dot{F}) = |V| - n + n \cdot |E| = 7 - 2 + 2 \cdot 8 = 21 \ ,$$



In reality we typically see

$$\textsc{Ert}(\dot{F}) \leq n \cdot \textsc{Ert}(F) \ .$$

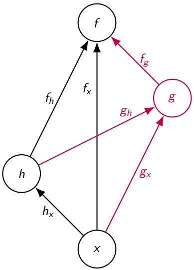
Tangent AD By Hand

```
template<typename T>
T h(const T& x) { return pow(x,2); }

template<typename T>
T g(const T& x, const T& h) { T v=x*h; return sin(v); }

template<typename T, typename Tg>
T f(const T& x, const T& h, const Tg& g) { return x+h+g; }

template<typename T>
T F(const T& x) { T h_=h(x); return f(x,h_,g(x,h_)); }
```



→ live

Exercise: Toward AD Mission Planning

Let

$$F = F_2(F_1(\mathbf{x})) : \mathbb{R}^5 \to \mathbb{R} \quad \text{s.t.}$$

$$F_1 : \mathbb{R}^5 \to \mathbb{R}^2$$
$$MEM(F_1) \sim OPS(F_1) = 100 \quad \text{(respective units)}$$

$$F_2 : \mathbb{R}^2 \to \mathbb{R}$$
$$MEM(F_2) \sim OPS(F_2) = 200\ .$$

Given $\dot{F}_1$ and $\dot{F}_2$, how to compute $F' \in \mathbb{R}^5$ with minimal OPS?

Solution

$$F' = \dot{F}(x, I_5) \text{ at } OPS(F') = 5 \cdot (100 + 200) = 1500$$

vs.

Solution

$$F' = \dot{F}(x, I_5) \text{ at } OPS(F') = 5 \cdot (100 + 200) = 1500$$

vs.

$$F_1' = \dot{F}_1(x, I_5) \text{ at } OPS(F_1') = 5 \cdot 100 = 500$$

$$F_2' = \dot{F}_2(F_1(x), I_2) \text{ at } OPS(F_2') = 2 \cdot 200 = 400$$

$$F' = F_2' \cdot F_1' \text{ at } OPS(F') = 500 + 400 + 1 \cdot 2 \cdot 5 = 910$$

Outlook: $OPS(F') = 300$ in adjoint mode.

Individual columns of the Jacobian can be approximated by (forward, backward, central) finite difference quotients as follows:

$$\nabla F(x) \approx_{\mathcal{O}(h)} \left(\frac{F(x + h \cdot e_i) - F(x)}{h} \right)_{i=0}^{n-1} \approx_{\mathcal{O}(h)} \left(\frac{F(x) - F(x - h \cdot e_i)}{h} \right)_{i=0}^{n-1}$$

$$\approx_{\mathcal{O}(h^2)} \left(\frac{F(x + h \cdot e_i) - F(x - h \cdot e_i)}{2 \cdot h} \right)_{i=0}^{n-1}$$

where $h = h(x_i) \in \mathbb{R}$ is a "suitable perturbation" typically picked as a compromise between accuracy and numerical stability, e.g,

$$h = \begin{cases} \sqrt{\epsilon} & x_i = 0 \\ \sqrt{\epsilon} \cdot |x_i| & x_i \neq 0 \end{cases}$$

with machine epsilon ϵ dependent on the floating-point precision.

Approximate Gradient by Central Finite Differences

`SDE/fd/main.cpp`

- inspect
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)

np	ns	*TIME* (*s*)	*MEM* (*MB*)
10^4	10	0.1	4
10^4	100	10.1	4
10^4	200	39.6	4
10^4	300	99.6	4
10^5	10	1.0	4
10^5	100	99.3	4
10^6	10	10.6	4
10^6	100	≈ 1000	4

Note: Scenario BC

Hands-on: Run and time on your computer.

Live: Gradient by Scalar Tangent AD with dco/c++

`SDE/gt1s/main.cpp`

- implement starting from primal `SDE/main.cpp`
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)
- compare with finite differences

np	ns	*TIME* (s)	*MEM* (MB)
10^4	10	< 0.1	4
10^4	100	5.4	4
10^4	200	21.0	4
10^4	300	47.3	4
10^5	10	0.6	4
10^5	100	52.9	4
10^6	10	5.6	4
10^6	100	≈ 560	4

Note: Scenario BC

Hands-on: Run and time on your computer.

Live: Gradient by Vector Tangent AD with dco/c++

`SDE/gt1v/main.cpp`

- implement starting from primal `SDE/main.cpp`
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)
- compare with scalar tangent AD

np	ns	*TIME* (*s*)	*MEM* (*MB*)
10^4	10	< 0.1	4
10^4	100	0.5	4
10^4	200	2.3	5
10^4	300	5.6	7
10^5	10	0.1	4
10^5	100	5.8	4
10^6	10	1.2	4
10^6	100	54.0	4

Note: Scenario BC

Hands-on: Run and time on your computer.

We consider a third-party implementation of the Heston Stochastic Volatility Model with Euler Discretization[1].

- inspect primal code
- build (`Makefile`)
- run

Use dco/c++ to compute the gradient

- in scalar tangent mode
- in vector tangent mode

Run experiments and compare performances.

[1] `www.quantstart.com/articles/Heston-Stochastic-Volatility-Model-with-Euler-Discretisation-in-C/`

Adjoints compute

$$\mathbb{R}^{1\times n} \ni \bar{x} = \bar{F}(x, \bar{y}) \equiv \bar{y} \cdot F'(x)$$

without prior accumulation of the Jacobian $F'(x)$.

Example:

$$\begin{pmatrix} \bar{x}_0 & \bar{x}_1 & \bar{x}_2 & \bar{x}_3 \end{pmatrix} = \begin{pmatrix} \bar{y}_0 & \bar{y}_1 & \bar{y}_2 \end{pmatrix} \cdot \begin{pmatrix} \frac{dy_0}{dx_0} & \frac{dy_0}{dx_1} & \frac{dy_0}{dx_2} & \frac{dy_0}{dx_3} \\ \frac{dy_1}{dx_0} & \frac{dy_1}{dx_1} & \frac{dy_1}{dx_2} & \frac{dy_1}{dx_3} \\ \frac{dy_2}{dx_0} & \frac{dy_2}{dx_1} & \frac{dy_2}{dx_2} & \frac{dy_2}{dx_3} \end{pmatrix}$$

Adjoint AD (Gradient Taping)

augmented primal (forward section): $(y, T) = \overrightarrow{F}(x)$

$$i = 1, \dots, n: \quad v_i = x_{i-1}$$

$$j = n+1, \dots, n+q: \quad v_j = \varphi_j(v_k)_{k \prec j}; \quad d_{j,i} = \frac{d\varphi_j(v_k)_{k \prec j}}{dv_i} \quad \forall i \prec j$$

$$k = 1, \dots, m: \quad y_{k-1} = v_{n+p+k}$$

Adjoint AD (Gradient Taping)

augmented primal (forward section): $(y, T) = \overrightarrow{F}(x)$

$$
\begin{aligned}
i &= 1, \ldots, n: & v_i &= x_{i-1} \\
j &= n+1, \ldots, n+q: & v_j &= \varphi_j(v_k)_{k \prec j}; \quad d_{j,i} = \frac{d\varphi_j(v_k)_{k \prec j}}{dv_i} \quad \forall i \prec j \\
k &= 1, \ldots, m: & y_{k-1} &= v_{n+p+k}
\end{aligned}
$$

adjoint (reverse section): $\bar{x} = \overleftarrow{F}(T, \bar{y})$

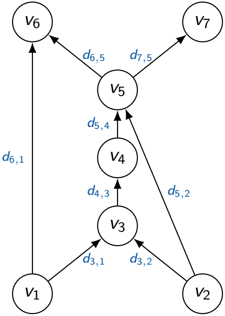
$$
\begin{aligned}
j &= 1, \ldots, n+q: & \bar{v}_j &= 0 \\
k &= 1, \ldots, m: & \bar{v}_{n+p+k} &= \bar{y}_{k-1} \\
j &= n+q, \ldots, n+1: & \bar{v}_i &= \bar{v}_i + \bar{v}_j \cdot d_{j,i} \quad \forall i \prec j \\
i &= 1, \ldots, n: & \bar{x}_{i-1} &= \bar{x}_{i-1} + \bar{v}_i
\end{aligned}
$$

The augmented primal lends itself for implementation by overloading. It records the (gradient) tape (see also: value tape). Adjoints are computed by interpretation of the tape.

Adjoint AD Example

Adjoint elementals are evaluated in reverse order, e.g.

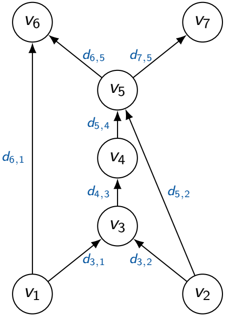
$\downarrow v_1 = x_0;$		$\uparrow \bar{x}_0 = \bar{v}_1$
$\downarrow v_2 = x_1;$		$\uparrow \bar{x}_1 = \bar{v}_2$
$\downarrow v_3 = v_1 \cdot v_2;$	$d_{3,1} = v_2;$	$\uparrow \bar{v}_1 = \bar{v}_1 + \bar{v}_3 \cdot d_{3,1}$
	$d_{3,2} = v_1$	$\uparrow \bar{v}_2 = \bar{v}_2 + \bar{v}_3 \cdot d_{3,2}$
$\downarrow v_4 = e^{v_3};$	$d_{4,3} = v_4;$	$\uparrow \bar{v}_3 = \bar{v}_4 \cdot d_{4,3}$
$\downarrow v_5 = v_4 + v_2;$	$d_{5,4} = 1;$	$\uparrow \bar{v}_4 = \bar{v}_5 \cdot d_{5,4}$
	$d_{5,2} = 1$	$\uparrow \bar{v}_2 = \bar{v}_5 \cdot d_{5,2}$
$\downarrow v_6 = v_1 / v_5;$	$d_{6,1} = 1/v_5;$	$\uparrow \bar{v}_1 = \bar{v}_6 \cdot d_{6,1}$
	$d_{6,5} = -v_1/v_5^2$	$\uparrow \bar{v}_5 = \bar{v}_5 + \bar{v}_6 \cdot d_{6,5}$
$\downarrow v_7 = v_5^3;$	$d_{7,5} = 2 \cdot v_5^2;$	$\uparrow \bar{v}_5 = \bar{v}_7 \cdot d_{7,5}$
$\downarrow y_0 = v_6;$		$\uparrow \bar{v}_6 = \bar{y}_0$
$\downarrow y_1 = v_7;$		$\uparrow \bar{v}_7 = \bar{y}_1$



Cost of Adjoint AD

$$\textsc{Ert}(\bar{F}) = \mathcal{R} \cdot (|V| - n + |E|) + \bar{m} \cdot |E|$$

$$\textsc{Mem}(\bar{F}) = |V| \cdot 8 + |E| \cdot 16$$



$$\textsc{Ert}(\bar{F}) = \frac{\textsc{Ert}(\bar{F})}{\textsc{Ert}(F)} \cdot \textsc{Ert}(F)$$

The minimization of the relative run time

$$0 < \frac{\textsc{Ert}(\bar{F})}{\textsc{Ert}(F)} \leq \infty$$

of an adjoint is a major challenge.

With tangent AD as the competitor, $\frac{\textsc{Ert}(\bar{F})}{\textsc{Ert}(\dot{F})}$ can be considered analogously.

$\mathcal{R}$ quantifies the overhead induced by taping.

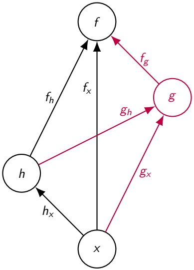
Adjoint AD By Hand

```
template<typename T>
T h(const T& x) { return pow(x,2); }

template<typename T>
T g(const T& x, const T& h) { T v=x*h; return sin(v); }

template<typename T, typename Tg>
T f(const T& x, const T& h, const Tg& g) { return x+h+g; }

template<typename T>
T F(const T& x) { T h_=h(x); return f(x,h_,g(x,h_)); }
```



→ live

Live: Gradient by Scalar Adjoint AD with dco/c++

`SDE/ga1s/main.cpp`

- implement starting from primal `SDE/gt1s/main.cpp`
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)
- compare with tangent AD

Experiments

np	ns	*TIME* (*s*)	*MEM* (*MB*)
10^4	10	< 0.1	14
10^4	100	0.1	109
10^4	200	0.2	214
10^4	300	0.3	320
10^5	10	0.1	111
10^5	100	0.8	1,061
10^6	10	0.8	1,089
10^6	100	-	$< 16,210$

Note: Scenario BC

Hands-on: Run and time on your computer.

- mutable independent `SDE/ga1s/variants/indep_inout/main.cpp`
- tape types and custom tape sizes `SDE/ga1s/variants/custom_tapesize/main.cpp`
- mixed base types `SDE/ga1s/variants/mixed_base_types/main.cpp`
- file tape `SDE/ga1s/variants/filetape/main.cpp`

- inspect
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)
- compare with default adjoint AD

Hands-On

For the given implementation of the Heston Stochastic Volatility Model with Euler Discretization use dco/c++ to compute the gradient in scalar adjoint mode.

Run experiments with variants of adjoint mode including

- different tape types and sizes
- mixed base types
- file tape

Compare performances and relate to tangent modes.

Let

$$F = F_2(F_1(\mathbf{x})) : \mathbb{R}^4 \to \mathbb{R}^{32} \quad \text{s.t.}$$

$$F_1 : \mathbb{R}^4 \to \mathbb{R}^2$$
$$MEM(F_1) = OPS(F_1) = 100 \quad \text{(respective units)}$$

$$F_2 : \mathbb{R}^2 \to \mathbb{R}^{32}$$
$$MEM(F_2) = OPS(F_2) = 100\ .$$

Given $\dot{F}_1$, $\bar{F}_1$, $\dot{F}_2$ and $\bar{F}_2$ how to compute $F' \in \mathbb{R}^{32\times 4}$ with minimal OPS?

Search Space

Let $\mathbf{y} = F(\mathbf{x}) = F_2(F_1(\mathbf{x}))$ with $F_1 : \mathbb{R}^n \to \mathbb{R}^k$ and $F_2 : \mathbb{R}^k \to \mathbb{R}^m$ with tapes T_1 and T_2, respectively. There are eight alternatives for computing F':

- $F' = T_2 \cdot F_1' = T_2 \cdot (T_1 \cdot I_n)$ (homogeneous tangent)
- $F' = T_2 \cdot F_1' = T_2 \cdot (I_k \cdot T_1)$
- $F' = F_2' \cdot T_1 = (I_m \cdot T_2) \cdot T_1$ (homogeneous adjoint)
- $F' = F_2' \cdot T_1 = (T_2 \cdot I_k) \cdot T_1$
- $F' = F_2' \cdot F_1' = (T_2 \cdot I_k) \cdot (I_k \cdot T_1)$
- $F' = F_2' \cdot F_1' = (I_m \cdot T_2) \cdot (I_k \cdot T_1)$
- $F' = F_2' \cdot F_1' = (I_m \cdot T_2) \cdot (T_1 \cdot I_n)$
- $F' = F_2' \cdot F_1' = (T_2 \cdot I_k) \cdot (T_1 \cdot I_n)$

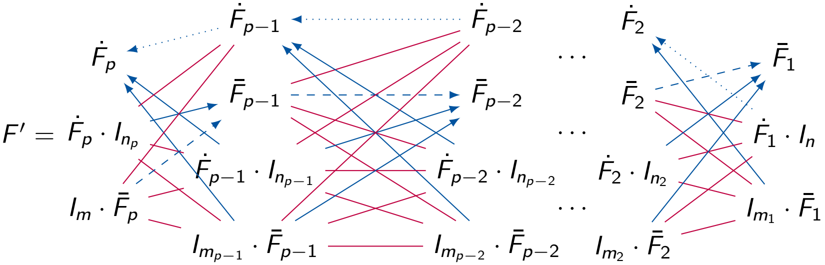
where $I_j \in \mathbb{R}^{j \times j}$ is the identity in $\mathbb{R}^j$ and $A \cdot T$ $[T \cdot B]$ denotes the propagation of A in adjoint mode [of B in (tapeless) tangent mode].

Solution

The corresponding operations counts can vary significantly, e.g, for $n = 4$, $k = 2$, $m = 32$, $|E_1| = |E_2| = 100$

$$
\begin{aligned}
OPS\,(T_2 \cdot (T_1 \cdot I_n)) &= 800 \\
OPS\,(T_2 \cdot (I_k \cdot T_1)) &= \mathbf{600} \\
OPS\,((I_m \cdot T_2) \cdot T_1) &= 6400 \\
OPS\,((T_2 \cdot I_k) \cdot T_1) &= 3400 \\
OPS\,((T_2 \cdot I_k) \cdot (I_k \cdot T_1)) &= 656 \\
OPS\,((I_m \cdot T_2) \cdot (I_k \cdot T_1)) &= 3656 \\
OPS\,((I_m \cdot T_2) \cdot (T_1 \cdot I_n)) &= 3856 \\
OPS\,((T_2 \cdot I_k) \cdot (T_1 \cdot I_n)) &= 856.
\end{aligned}
$$

We have tangents and adjoints – not Jacobians ...



... distinguishing standard and matrix-free matrix products. Note:

- feasible subproblems along directed paths (bidirectional purple edges)
- each layer visited once; $\dot{F}_1 \equiv \dot{F}_1 \cdot I_n$ and $\bar{F}_p \equiv I_m \cdot \bar{F}_p$
- homogeneous tangent dotted; homogeneous adjoit dashed;

Differential Invariant

The identity

$$\bar{x} \cdot \dot{x} = \bar{y} \cdot \dot{y} \quad (\bar{X} \cdot \dot{X} = \bar{Y} \cdot \dot{Y})$$

holds for any compact sub-program of F. ($\Rightarrow$ Use it for validation / *debugging*.)

Proof:

$$\bar{x} \cdot \dot{x} = (\bar{y} \cdot F') \cdot \dot{x} = \bar{y} \cdot (F' \cdot \dot{x}) = \bar{y} \cdot \dot{y}$$

Similarly, differential invariants can be derived for second- and higher-order AD.

U. N.: *Differential Invariants.* arXiv:2101.03334 [math.NA]. Submitted.

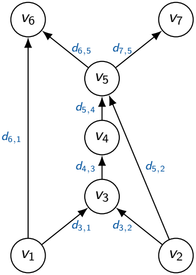
Outline

Contents:

- Early Recording [and Late Back-Propagation] (Default)
- Early Preaccumuation
- Early Back-Propagation
- Early Preaccumulation and Back-Propagation

Motivation: $y = F(x) = f(g(x), x)$ s.t. $g : x \mapsto v_5,\ f : (x_0, v_5) \mapsto y$

$$(y, \bar{x}) = \bar{F}(x, I_2)$$

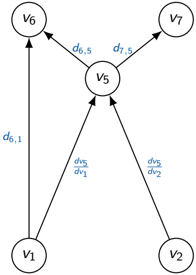


$$\text{MEM}(\bar{F}) = 8 + 7^2 = 15$$

$$OPS(\overrightarrow{F}) = 5 + 8 = 13$$

$$OPS(\overleftarrow{F}) = 2 \cdot 8 = 16$$

$$(y, T^{g',f}) = \bar{g}(x, 1) \circ \overrightarrow{f}(x_0, v_5)$$

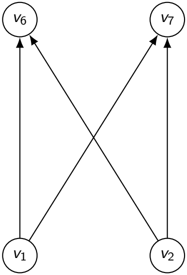


$$\text{MEM}(\bar{F}) = 5 + 5 + \ldots$$

$$OPS(\overrightarrow{F}) = 5 + 8 + \ldots$$

$$OPS(\overleftarrow{F}) = 1 \cdot 5 + \ldots$$

$$\bar{x} = \overleftarrow{f}(I_2, T^{g',f})$$



$$\ldots + 0 = 10$$

$$\ldots + 0 = 13$$

$$\ldots + 2 \cdot 5 = 15$$

[2] ... plus (invariant) memory requirement of primal

Recall: Adjoints by Gradient Taping

Recording (augmented primal section)

$$(y, T) = \overrightarrow{F}(x)$$

Backpropagation (adjoint section)

$$\bar{X} = \overleftarrow{F}(\bar{Y}, T) \equiv \bar{Y} \cdot T$$

Recall: Adjoints by Gradient Taping

Recording (augmented primal section)

$$(y, T) = \overrightarrow{F}(x)$$

$$
\begin{array}{l}
T := (\emptyset, \emptyset) \\
\text{for } i = 1, \ldots, n : \\
\qquad v_i = x_i \\
\qquad T \mathrel{+}= (\{(i, \&\bar{v}_i)\}, \emptyset) \\
\text{for } j = n+1, \ldots, n+q : \\
\qquad v_j := \varphi_j(v_k)_{k \prec j} \\
\qquad T \mathrel{+}= (\{(j, \&\bar{v}_j)\}, \emptyset) \\
\qquad \forall i \prec j : \\
\qquad\qquad d_{j,i} := \dfrac{d\varphi_j}{dv_i}(v_k)_{k \prec j} \\
\qquad\qquad T \mathrel{+}= (\emptyset, \{(i, j, d_{j,i})\}) \\
\{y_1, \ldots, y_m\} \subseteq \{v_1, \ldots v_{n+q}\}
\end{array}
$$

Backpropagation (adjoint section)

$$\bar{X} = \overleftarrow{F}(\bar{Y}, T) \equiv \bar{Y} \cdot T$$

Recall: Adjoints by Gradient Taping

Recording (augmented primal section)

$$(y, T) = \overrightarrow{F}(x)$$

$$
\begin{aligned}
& T := (\emptyset, \emptyset) \\
& \text{for } i = 1, \ldots, n : \\
& \qquad v_i = x_i \\
& \qquad T \mathrel{+}= (\{(i, \&\bar{v}_i)\}, \emptyset) \\
& \text{for } j = n+1, \ldots, n+q : \\
& \qquad v_j := \varphi_j(v_k)_{k \prec j} \\
& \qquad T \mathrel{+}= (\{(j, \&\bar{v}_j)\}, \emptyset) \\
& \qquad \forall i \prec j : \\
& \qquad\qquad d_{j,i} := \frac{d\varphi_j}{dv_i}(v_k)_{k \prec j} \\
& \qquad\qquad T \mathrel{+}= (\emptyset, \{(i, j, d_{j,i})\}) \\
& \{y_1, \ldots, y_m\} \subseteq \{v_1, \ldots v_{n+q}\}
\end{aligned}
$$

Backpropagation (adjoint section)

$$\bar{X} = \overleftarrow{F}(\bar{Y}, T) \equiv \bar{Y} \cdot T$$

$$
\begin{aligned}
& \text{for } j = 1, \ldots, n+q : \quad \bar{v}_j := 0 \\
& \{\bar{y}_1, \ldots, \bar{y}_m\} \subseteq \{\bar{v}_1, \ldots \bar{v}_{n+q}\} \\
& \text{for } j = n+q, \ldots, n+1 \\
& \qquad \forall i \prec j : \\
& \qquad\qquad T \mathrel{-}= (\emptyset, \{(i, j, d_{j,i})\}) \\
& \qquad\qquad \bar{v}_i \mathrel{+}= \bar{v}_j \cdot d_{j,i} \\
& \qquad T \mathrel{-}= (\{(j, \&\bar{v}_j)\}, \emptyset) \\
& \qquad \bar{v}_j := 0 \\
& \text{for } i = 1, \ldots, n : \quad \bar{x}_i = \bar{v}_i
\end{aligned}
$$

Recall: Adjoints by Value Taping

Recording (augmented primal section)

$$(y, T) = \overrightarrow{F}(x)$$

Backpropagation (adjoint section)

$$\bar{X} = \overleftarrow{F}(\bar{Y}, T) \equiv \bar{Y} \cdot T$$

Recall: Adjoints by Value Taping

Recording (augmented primal section)

$$(y, T) = \overrightarrow{F}(x)$$

Backpropagation (adjoint section)

$$\bar{X} = \overleftarrow{F}(\bar{Y}, T) \equiv \bar{Y} \cdot T$$

$$
\begin{array}{l}
T := (\emptyset, \emptyset) \\
\text{for } i = 1, \ldots, n : \\
\qquad v_i = x_i \\
\qquad T \mathrel{+}= (\{(i, v_i, \&\bar{v}_i)\}, \emptyset) \\
\text{for } j = n+1, \ldots, n+q : \\
\qquad v_j := \varphi_j(v_k)_{k \prec j} \\
\qquad T \mathrel{+}= (\{(j, v_j, \&\bar{v}_j)\}, \emptyset) \\
\qquad \forall i \prec j : \quad T \mathrel{+}= (\emptyset, \{(i, j)\}) \\
\{y_1, \ldots, y_m\} \subseteq \{v_1, \ldots v_{n+q}\}
\end{array}
$$

Recall: Adjoints by Value Taping

Recording (augmented primal section)

$$(y, T) = \overrightarrow{F}(x)$$

$$
\begin{array}{l}
T := (\emptyset, \emptyset) \\
\text{for } i = 1, \ldots, n : \\
\qquad v_i = x_i \\
\qquad T \mathrel{+}= (\{(i, v_i, \&\bar{v}_i)\}, \emptyset) \\
\text{for } j = n+1, \ldots, n+q : \\
\qquad v_j := \varphi_j(v_k)_{k \prec j} \\
\qquad T \mathrel{+}= (\{(j, v_j, \&\bar{v}_j)\}, \emptyset) \\
\qquad \forall i \prec j : \quad T \mathrel{+}= (\emptyset, \{(i, j)\}) \\
\{y_1, \ldots, y_m\} \subseteq \{v_1, \ldots v_{n+q}\}
\end{array}
$$

Backpropagation (adjoint section)

$$\bar{X} = \overleftarrow{F}(\bar{Y}, T) \equiv \bar{Y} \cdot T$$

$$
\begin{array}{l}
\text{for } j = 1, \ldots, n+q : \quad \bar{v}_j := 0 \\
\{\bar{y}_1, \ldots, \bar{y}_m\} \subseteq \{\bar{v}_1, \ldots \bar{v}_{n+q}\} \\
\text{for } j = n+q, \ldots, n+1 \\
\qquad \forall i \prec j : \\
\qquad\qquad T \mathrel{-}= (\emptyset, \{(i, j)\}) \\
\qquad\qquad d_{j,i} := \dfrac{d\varphi_j}{dv_i}(v_k)_{k \prec j} \\
\qquad\qquad \bar{v}_i \mathrel{+}= \bar{v}_j \cdot d_{j,i} \\
\qquad T \mathrel{-}= (\{(j, v_j, \&\bar{v}_j)\}, \emptyset) \\
\qquad \bar{v}_j := 0 \\
\text{for } i = 1, \ldots, n : \quad \bar{x}_i = \bar{v}_i
\end{array}
$$

Terminology

AD can be applied naively to differentiable programs. Adjoint AD may require a more fine-grain treatment to ensure feasible memory requirement and optimal run time.

- The augmented primal section records the tape (default: early recording).
- The adjoint section (back-)propagates adjoints / interprets the tape (default: late back-propagation).
- Early intervention interupts recording.
- Late intervention interupts (recording and) back-propagation.
- (Partial) Preaccumulation yields (incomplete) local Jacobians of parts of the program.
- External adjoints can implement
  - joint augmented primal and adjoint (local recording followed immediately by back-propagation);
  - split augmented primal and adjoint (local recording and back-propagation are separated).

Notation

- primal differentiable (sub-)program

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

- derivative (sub-)program

$$F' : \mathbb{R}^n \to \mathbb{R}^{m\times n} : F_x = F'(x) \quad (\text{or } (y, F_x) = F'(x))$$

- (vector) tangent (sub-)program

$$\dot{F} : \mathbb{R}^n \times \mathbb{R}^{n\times \dot{n}} \to \mathbb{R}^m \times \mathbb{R}^{m\times \dot{n}} : (y, \dot{Y}) = \dot{F}(x, \dot{X})$$

- (vector) adjoint (sub-)program

$$\bar{F} : \mathbb{R}^n \times \mathbb{R}^{\bar{m}\times m} \to \mathbb{R}^m \times \mathbb{R}^{\bar{m}\times n} : (y, \bar{X}) = \bar{F}(x, \bar{Y})$$

- augmented section of adjoint (sub-)program

$$\overrightarrow{F} : \mathbb{R}^n \to \mathbb{R}^m \times T : (y, T) = \overrightarrow{F}(x)$$

- adjoint section of adjoint (sub-)program

$$\overleftarrow{F} : \mathbb{R}^{\bar{m}\times m} \times T \to \mathbb{R}^{\bar{m}\times n} : \bar{X} = \overleftarrow{F}(\bar{Y}, T)$$

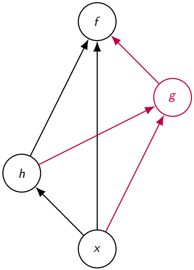
Further Notation

- $\textsc{Ert}(F)$ elapsed run time of primal (sub-)program
- $\textsc{Ert}(\dot{F})$ elapsed run time of tangent (sub-)program
- $\textsc{Ert}(\bar{F})$ elapsed run time of adjoint (sub-)program
- $\textsc{Ert}(\overrightarrow{F})$ elapsed run time of augmented primal section of adjoint (sub-)program
- $\textsc{Ert}(\overleftarrow{F})$ elapsed run time of adjoint section of adjoint (sub-)program
- $\textsc{Mem}(F) = 0$ persistent memory requirement of primal (sub-)program
- $\textsc{Mem}(\dot{F}) = 0$ persistent memory requirement of tangent (sub-)program
- $\textsc{Mem}(\bar{F})$ persistent memory of adjoint (sub-)program
- $\textsc{Mem}(\overrightarrow{F})$ persistent memory of augmented primal section of adjoint (sub-)program
- $\textsc{Mem}(\overleftarrow{F})$ persistent memory of adjoint section of adjoint (sub-)program
- $\widehat{\textsc{Mem}}$ sharp upper bound on persistent memory available

Generic Scenario for Intervention

$$y = F(x) = f(x, h(x), g(x, h(x))) : \ \mathbb{R}^n \to \mathbb{R}^m$$

s.t. $h = h(x)$, $g = g(x, h)$, and

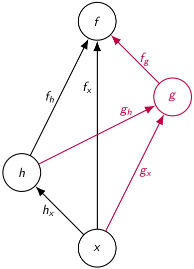


$$h : \mathbb{R}^n \to \mathbb{R}^{m_h}$$
$$g : \mathbb{R}^{n_g} \to \mathbb{R}^{m_g}$$
$$f : \mathbb{R}^{n_f} \to \mathbb{R}^m \ .$$

This generic scenario is used to illustrate the various modes of intervention applied to g.

Jacobian / Chain Rule:

$$\mathbb{R}^{m\times n} \ni F' \equiv \frac{dF}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial h}\cdot\frac{\partial h}{\partial x} + \frac{\partial f}{\partial g}\cdot\left(\frac{\partial g}{\partial x} + \frac{\partial g}{\partial h}\cdot\frac{\partial h}{\partial x}\right) = f_x + f_h\cdot h_x + f_g\cdot(g_x + g_h\cdot h_x)$$



Adjoint:

$$\mathbb{R}^{\bar{m}\times n} \ni \bar{X} = \bar{Y}\cdot\frac{dF}{dx} = \bar{Y}\cdot(f_x + f_h\cdot h_x + f_g\cdot(g_x + g_h\cdot h_x))$$

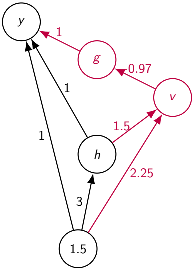
Toy Example

```
template<typename T>
T h(const T& x) { return pow(x,2); }

template<typename T>
T g(const T& x, const T& h) {
  T v=x*h;
  return sin(v);
}

template<typename T, typename Tg>
T f(const T& x, const T& h, const Tg& g) { return x+h+g; }

template<typename T>
T F(const T& x) {
  T _h=h(x);
  return f(x,_h,g(x,_h));
}
```



While in the following all types of intervention are illustrated in the context of adjoint AD by overloading with dco/c++ analogous techniques can be applied to hand-coding or to AD solutions based on other tools.

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$). Let $\mathcal{R} = 3$.

The elapsed run time evolves as

$$
\begin{array}{lll}
1: & \textrm{ERT}[\bar{F}] = 0 & \\
2: & \textrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|) & (0 + 3 \cdot (2 - 1 + 1) = 6) \\
3: & \textrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^g| - n^g + |E^g|) & (6 + 3 \cdot (4 - 2 + 3) = 21) \\
4: & \textrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^f| - n^f + |E^f|) & (21 + 3 \cdot (4 - 3 + 3) = 33) \\
5: & \textrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^f| & (33 + 1 \cdot 3 = 36) \\
6: & \textrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^g| & (36 + 1 \cdot 3 = 39) \\
7: & \textrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h| & (39 + 1 \cdot 1 = \mathbf{40})
\end{array}
$$

and the persistent memory requirement as ...

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^h| \cdot 16 \qquad (0 + 1 \cdot 16 = 16)$

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^g| \cdot 16 \qquad (16 + 3 \cdot 16 = 64)$

4: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^f| \cdot 16 \qquad (64 + 3 \cdot 16 = 112)$

5: $\textsc{Mem}[\bar{F}] \mathrel{+}= |V^h \cup V^g \cup V^f| \cdot 8 \qquad (112 + 5 \cdot 8 = \mathbf{152})$

6: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^f| \cdot 16 \qquad (152 - 3 \cdot 16 = 104)$

7: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^g| \cdot 16 \qquad (104 - 3 \cdot 16 = 56)$

8: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^h| \cdot 16 \qquad (56 - 1 \cdot 16 = 40)$

9: $\textsc{Mem}[\bar{F}] \mathrel{-}= |V^h \cup V^g \cup V^f| \cdot 8 \qquad (40 - 5 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 152 bytes in line 5.

Default Adjoint AD with dco/c++

```
#include "F.hpp"
#include "dco.hpp"
using AM=dco::ga1s<double>;
using AT=AM::type;

int main() {
  AT x=1.5, y;
  AM::global_tape=AM::tape_t::create(); // enable recording of tape
  AM::global_tape->register_variable(x); // register independent variables
  y=F(x); // record tape
  dco::derivative(y)=1; // seed
  AM::global_tape->interpret_adjoint(); // interpret tape
  cout << dco::value(y) << ' ' << dco::derivative(x) << endl; // harvest
  AM::tape_t::remove(AM::global_tape); // remove tape
  return 0;
}
```

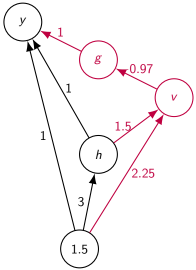
Default Adjoint AD with dco/c++: Behind the Scenes of Toy Example

Tape:

```
1 5: 4, 1
2 5: 2, 1
3 5: 1, 1
4 4: 3, −0.972884
5 3: 2, 1.5
6 3: 1, 2.25
7 2: 1, 3
```

Adjoints:

```
1 0 // unused
2 −2.56697
3 −0.459326
4 −0.972884
5 1
6 1
```



Default Adjoint AD: Exercise

Analyse elapsed run time and persistent memory requirement of the default EARLY RECORDING AND LATE BACK-PROPAGATION pattern for the computation of the Jacobian of F with

$$\begin{aligned}
\dot{n} = n = n^h &= 10 \\
m^h = n^g &= 3 \\
m^g &= 1 \\
n^f &= 4 \\
m^f = m = \bar{m} &= 2 \\
\mathcal{R} &= 3 \\
|V^\phi| &= 100 \\
|E^\phi| &= 1000
\end{aligned}$$

and for $\phi \in \{h, g, f\}$.

Solution

1: $\mathrm{ERT}[\bar{F}] = 0$

2: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[h] \qquad (0 + 3 \cdot (100 - 10 + 1.000) = 3.270)$

3: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[g] \qquad (3.270 + 3 \cdot (100 - 3 + 1.000) = 6.561)$

4: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[f] \qquad (6.561 + 3 \cdot (100 - 4 + 1.000) = 9.849)$

5: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[f] \qquad (9.849 + 2 \cdot 1.000 = 11.849)$

6: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[g] \qquad (11.849 + 2 \cdot 1.000 = 13.849)$

7: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[h] \qquad (13.849 + 2 \cdot 1.000 = \mathbf{15.849})$

Solution

$$
\begin{array}{lll}
1: & \mathrm{MEM}[\bar{F}] = 0 & \\
2: & \mathrm{MEM}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{MEM}}[h] & (0 + 1.000 \cdot 16 = 16.000) \\
3: & \mathrm{MEM}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{MEM}}[g] & (16.000 + 1.000 \cdot 16 = 32.000) \\
4: & \mathrm{MEM}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{MEM}}[f] & (32.000 + 1.000 \cdot 16 = 48.000) \\
5: & \mathrm{MEM}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{MEM}}[F] & (48.000 + 2 \cdot 300 \cdot 8 = \mathbf{52.800}) \\
6: & \mathrm{MEM}[\bar{F}] \mathrel{-}= \overrightarrow{\mathrm{MEM}}[f] & (52.800 - 1.000 \cdot 16 = 36.800) \\
7: & \mathrm{MEM}[\bar{F}] \mathrel{-}= \overrightarrow{\mathrm{MEM}}[g] & (36.800 - 1.000 \cdot 16 = 20.800) \\
8: & \mathrm{MEM}[\bar{F}] \mathrel{-}= \overrightarrow{\mathrm{MEM}}[h] & (20.800 - 1.000 \cdot 16 = 4.800) \\
9: & \mathrm{MEM}[\bar{F}] \mathrel{-}= \overleftarrow{\mathrm{MEM}}[F] & (4.800 - 2 \cdot 300 \cdot 8 = 0)
\end{array}
$$

The objective of early intervention is a lower elapsed run time

$$\textsc{Ert}(\bar{F}) \rightarrow \min$$

and / or a reduction of the maximum resident set size

$$\textsc{Mem}(\bar{F}) \rightarrow \ (\leq \widehat{\textsc{Mem}})$$

of the adjoint through

- Early Back-Propagation
- Early Preaccumulation
- Early Preaccumulation and Back-Propagation

Use EARLY BACKPROPAGATION whenever $\bar{G}$ is known prior to the evaluation of $\bar{f}$ implying that $\bar{g}$ can be evaluated before recording the tape of f.

Consequently, T^g and T^f can be held in overlapping memory.

Moreover, the tape of f does not need to record dependence on g as the latter becomes passive which we denote as $f(\tilde{g})$.

The elapsed run time of the Early Back-Propagation pattern evolves as

1: $\textsc{Ert}[\bar{F}] = 0$

2: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[h]$ (record T^h)

3: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[g]$ (record T^g)

4: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[g]$ (interpret T^g)

5: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[f(\tilde{g})]$ (record $T^{f(\tilde{g})}$)

6: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[f(\tilde{g})]$ (interpret $T^{f(\tilde{g})}$)

7: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[h]$ (interpret T^h)

subject to $\textsc{Mem}[\bar{F}] \leq \widehat{\textsc{Mem}}$.

Note that the above implies the algorithmic definition of the Early Back-Propagation pattern.

Lemma

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] \\
+ \max\{ \\
\overrightarrow{\textsc{Mem}}[g] + \overleftarrow{\textsc{Mem}}[g], \overrightarrow{\textsc{Mem}}[f(\tilde{g})] + \max\{\overleftarrow{\textsc{Mem}}[g], \overleftarrow{\textsc{Mem}}[f(\tilde{g})]\} \\
\}
\end{aligned}
$$

Proof

$$
\begin{array}{rll}
1: & \textsc{Mem}[\bar{F}] = 0 & \\
2: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[h] & \text{(record } T^h\text{)} \\
3: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[g] & \text{(record } T^g\text{)} \\
4: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g] & \text{(allocate adjoints)} \\
5: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[g] & \text{(interpret } T^g\text{)} \\
6: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[f(\tilde{g})] & \text{(record } T^{f(\tilde{g})}\text{)} \\
7: & \textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[f(\tilde{g})] - \overleftarrow{\textsc{Mem}}[g], 0\} & \text{(grow memory for adjoints?)} \\
8: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[f(\tilde{g})] & \text{(interpret } T^{f(\tilde{g})}\text{)} \\
9: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[h] & \text{(interpret } T^h\text{)} \\
10: & \textsc{Mem}[\bar{F}] = 0 & \text{(free adjoints)} \quad .
\end{array}
$$

Proof

Note that $\text{MEM}[\bar{F}]$ takes local maxima just prior to decrements, that is, in lines 4 and 7. Hence, the above becomes equal to

$$
\begin{aligned}
\text{MEM}[\bar{F}] = \max\{ & \\
& \overrightarrow{\text{MEM}}[h] + \overrightarrow{\text{MEM}}[g] + \overleftarrow{\text{MEM}}[h] + \overleftarrow{\text{MEM}}[g] \ , \\
& \overrightarrow{\text{MEM}}[h] + \overleftarrow{\text{MEM}}[h] + \overleftarrow{\text{MEM}}[g] \\
& \quad + \overrightarrow{\text{MEM}}[f(\tilde{g})] + \max\{\overleftarrow{\text{MEM}}[f(\tilde{g})] - \overleftarrow{\text{MEM}}[g], 0\} \\
\} &
\end{aligned}
$$

which simplifies to the equation stated in the lemma.

■

Early Back-Propagation: Toy Example

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$) for $\mathcal{R} = 3$.

The elapsed run time evolves as

$$
\begin{array}{lll}
1: & \mathrm{ERT}[\bar{F}] = 0 & \\
2: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|) & (0 + 3 \cdot (2 - 1 + 1) = 6) \\
3: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^g| - n^g + |E^g|) & (6 + 3 \cdot (4 - 2 + 3) = 21) \\
6: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^g| & (21 + 1 \cdot 3 = 24) \\
4: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^f| - n^f + |E^f|) & (24 + 3 \cdot (4 - 3 + 3) = 36) \\
5: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^f| & (36 + 1 \cdot 3 = 39) \\
7: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h| & (39 + 1 \cdot 1 = 40)
\end{array}
$$

and the persistent memory requirement as ...

1:	$\textsc{Mem}[\bar{F}] = 0$	
2:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \|E^h\| \cdot 16$	$(0 + 1 \cdot 16 = 16)$
3:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \|E^g\| \cdot 16$	$(16 + 3 \cdot 16 = 64)$
4:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \|V^h \cup V^g\| \cdot 8$	$(64 + 4 \cdot 8 = 96)$
5:	$\textsc{Mem}[\bar{F}] \mathrel{-}= \|E^g\| \cdot 16$	$(96 - 3 \cdot 16 = 48)$
6:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \|E^{f(\tilde{g})}\| \cdot 16$	$(48 + 2 \cdot 16 = 80)$
7:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\|V^h \cup V^f\| - \|V^h \cup V^g\|, 0\} \cdot 8$	$(80 + \max\{4 - 4, 0\} \cdot 8 = 80)$
8:	$\textsc{Mem}[\bar{F}] \mathrel{-}= \|E^{f(\tilde{g})}\| \cdot 16$	$(80 - 2 \cdot 16 = 48)$
9:	$\textsc{Mem}[\bar{F}] \mathrel{-}= \|E^h\| \cdot 16$	$(48 - 1 \cdot 16 = 32)$
10:	$\textsc{Mem}[\bar{F}] \mathrel{-}= \|V^h \cup V^g\| \cdot 8$	$(32 - 4 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 96 bytes in line 4.

Early Back-Propagation with dco/c++

```
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
AM::tape_t::position_t pos=AM::global_tape->get_position();
vg=g(x,vh);
dco::derivative(vg)=1; // seed known \bar{G}
AM::global_tape->interpret_adjoint_to(pos);
AM::global_tape->reset_to(pos);
double pvg=dco::value(vg);
y=f(x,vh,pvg);
dco::derivative(y)=1; // seed given \bar{Y}
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' ' << dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

Compare the default Early Recording and Late Back-Propagation pattern with Early Back-Propagation for the computation of the Jacobian of F with

$$\dot{n} = n = n^h = 10$$
$$m^h = n^g = 3$$
$$m^g = 1$$
$$n^f = 4$$
$$m^f = m = \bar{m} = 2$$
$$\mathcal{R} = 3$$
$$|V^\phi| = 100$$
$$|E^\phi| = 1000$$

and for $\phi \in \{h, g, f\}$.

Solution: $\textsc{Ert}$

1: $\textsc{Ert}[\bar{F}] = 0$

2: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[h]$ $\quad (0 + 3 \cdot (100 - 10 + 1.000) = 3.270)$

3: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[g]$ $\quad (3.270 + 3 \cdot (100 - 3 + 1.000) = 6.561)$

4: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[g]$ $\quad (6.561 + 2 \cdot 1.000 = 8.561)$

5: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[f(\tilde{g})]\ (\approx \overrightarrow{\textsc{Ert}}[f])$ $\quad (8.561 + 3 \cdot (100 - 4 + 1.000) = 11.849)$

6: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[f(\tilde{g})]\ (\approx \overleftarrow{\textsc{Ert}}[f])$ $\quad (11.849 + 2 \cdot 1.000 = 13.849)$

7: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[h]$ $\quad (13.849 + 2 \cdot 1.000 = \mathbf{15.849})$

Solution: $\textsc{Mem}$

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[h]$ $\quad (0 + 1.000 \cdot 16 = 16.000)$

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[g]$ $\quad (16.000 + 1.000 \cdot 16 = 32.000)$

4: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g]$ $\quad (32.000 + 2 \cdot 200 \cdot 8 = 35.200)$

5: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[g]$ $\quad (35.200 - 1.000 \cdot 16 = 19.200)$

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[f(\tilde{g})]\ (\approx \overrightarrow{\textsc{Mem}}[f])$ $\quad (19.200 + 1.000 \cdot 16 = \mathbf{35.200})$

7: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[f(\tilde{g})] - \overleftarrow{\textsc{Mem}}[g], 0\}$ $\quad (35.200 + \max\{2 \cdot 100 \cdot 8 - 1.600, 0\} = 35.200)$

8: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[f(\tilde{g})]\ (\approx \overrightarrow{\textsc{Mem}}[f])$ $\quad (35.200 - 1.000 \cdot 16 = 19.200)$

9: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[h]$ $\quad (19.200 - 1.000 \cdot 16 = 3.200)$

10: $\textsc{Mem}[\bar{F}] = 0$ $\quad (3.200 - 2 \cdot 200 \cdot 8 = 0)$.

Use Early Preaccumulation whenever a relevant decrease in tape size and/or run time can be expected.

The elapsed run time of the EARLY PREACCUMULATION pattern evolves as

1: $\mathrm{ERT}[\bar{F}] = 0$

2: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[h]$ (record T^h)

3: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \mathrm{ERT}[g^-]$ (preaccumulate T^{g^-})

4: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[T^{g^-}]$ (record T^{g^-})

5: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[f]$ (record T^f)

6: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[f]$ (interpret T^f)

7: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[T^{g^-}]$ (interpret T^{g^-})

8: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[h]$ (interpret T^h)

subject to $\mathrm{MEM}[\bar{F}] \leq \widehat{\mathrm{MEM}}$.

Lemma

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] + \max\{ \\
& \overrightarrow{\textsc{Mem}}[g^{-}] + \overleftarrow{\textsc{Mem}}[g^{-}] \; , \\
& \overrightarrow{\textsc{Mem}}[T^{g^{-}}] + \overrightarrow{\textsc{Mem}}[f] + \max\{\overleftarrow{\textsc{Mem}}[g^{-}], \overleftarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[T^{g^{-}}]\} \\
\} \; .
\end{aligned}
$$

Proof

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[h]$ (record T^h)

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[g^-]$ (record for preaccumulation)

4: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-]$ (allocate adjoints)

5: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[g^-]$ (preaccumulate T^{g^-})

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[T^{g^-}]$ (record T^{g^-})

7: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[f]$ (record T^f)

8: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[T^{g^-}] - \overleftarrow{\textsc{Mem}}[g^-], 0\}$ (grow memory for adjoints?)

9: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[f]$ (interpret T^f)

10: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[T^{g^-}]$ (interpret T^{g^-})

11: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[h]$ (interpret T^h)

12: $\textsc{Mem}[\bar{F}] = 0$ (free adjoints)

Proof

$\textsc{Mem}[\bar{F}]$ takes local maxima in lines 4 and 8. Hence, the above becomes equal to

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = \max\{ & \\
& \overrightarrow{\textsc{Mem}}[h] + \overrightarrow{\textsc{Mem}}[g^-] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-] \,, \\
& \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-] \\
& \quad + \overrightarrow{\textsc{Mem}}[T^{g^-}] + \overrightarrow{\textsc{Mem}}[f] + \max\{\overleftarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[T^{g^-}] - \overleftarrow{\textsc{Mem}}[g^-], 0\} \\
\} &
\end{aligned}
$$

which simplifies to the equation stated in the lemma.

■

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$) for $\mathcal{R} = 3$.

The elapsed run time evolves as

$$
\begin{array}{rll}
1: & \textsc{Ert}[\bar{F}] = 0 & \\
2: & \textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|) & (0 + 3 \cdot (2 - 1 + 1) = 6) \\
3a: & \textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^g| - n^g + |E^g|) & (6 + 3 \cdot (4 - 2 + 3) = 21) \\
3b: & \textsc{Ert}[\bar{F}] \mathrel{+}= m^g \cdot |E^g| & (21 + 1 \cdot 3 = 24) \\
4: & \textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot n^g \cdot m^g & (24 + 3 \cdot 2 \cdot 1 = 30) \\
5: & \textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^f| - n^f + |E^f|) & (30 + 3 \cdot (4 - 3 + 3) = 42) \\
6: & \textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^f| & (42 + 1 \cdot 3 = 45) \\
7: & \textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot n^g \cdot m^g & (45 + 1 \cdot 2 \cdot 1 = 47) \\
8: & \textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h| & (47 + 1 \cdot 1 = \mathbf{48})
\end{array}
$$

and the persistent memory requirement as ...

Early Preaccumulation in Adjoint Mode

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^h| \cdot 16$ $\quad (0 + 1 \cdot 16 = 16)$

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^g| \cdot 16$ $\quad (16 + 3 \cdot 16 = 64)$

4: $\textsc{Mem}[\bar{F}] \mathrel{+}= |V^h \cup V^g| \cdot 8$ $\quad (64 + 4 \cdot 8 = 96)$

5: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^g| \cdot 16$ $\quad (96 - 3 \cdot 16 = 48)$

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^{g'}| \cdot 16$ $\quad (48 + 2 \cdot 16 = 80)$

7: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^f| \cdot 16$ $\quad (80 + 3 \cdot 16 = \mathbf{128})$

8: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{|V^f \cup V^{g'}| - |V^g|, 0\} \cdot 8$ $\quad (128 + \max\{4 - 4, 0\} \cdot 8 = 128)$

9: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^f| \cdot 16$ $\quad (128 - 3 \cdot 16 = 80)$

10: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^{g'}| \cdot 16$ $\quad (80 - 2 \cdot 16 = 48)$

11: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^h| \cdot 16$ $\quad (48 - 1 \cdot 16 = 32)$

12: $\textsc{Mem}[\bar{F}] \mathrel{-}= |V^h \cup V^g| \cdot 8$ $\quad (32 - 4 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 128 bytes in line 8.

Early Preaccumulation in Adjoint Mode with dco/c++

```
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
AM::tape_t::position_t pos=AM::global_tape->get_position();
vg=g(x,vh);
double pvg=dco::value(vg);
// seed \bar{G} as identity for full preaccumulation in adjoint mode
dco::derivative(vg)=1;
AM::global_tape->interpret_adjoint_to(pos);
AM::global_tape->reset_to(pos);
vg=x*dco::derivative(x)+vh*dco::derivative(vh);
dco::derivative(x)=0; dco::derivative(vh)=0;
dco::value(vg)=pvg;
y=f(x,vh,vg);
dco::derivative(y)=1; // seed \bar{Y}
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' ' << dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

$\textsc{Early Preaccumulation}$ in Tangent Mode

Let g^- implement full preaccumulation in tangent mode, that is, $\dot{n}^g = n^g$ and $\textsc{Ert}[g^-] = \dot{\textsc{Ert}}[g]$.

The elapsed run time evolves as

1: $\textsc{Ert}[\bar{F}] = 0$

2: $\textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|)$ $\quad (0 + 3 \cdot (2 - 1 + 1) = 6)$

3: $\textsc{Ert}[\bar{F}] \mathrel{+}= |V^g| - n^g + |E^g| + n^g \cdot |E^g|$ $\quad (6 + 4 - 2 + 3 + 2 \cdot 3 = 17)$

4: $\textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot n^g \cdot m^g$ $\quad (17 + 3 \cdot 2 \cdot 1 = 23)$

5: $\textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^f| - n^f + |E^f|)$ $\quad (23 + 3 \cdot (4 - 3 + 3) = 35)$

6: $\textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^f|$ $\quad (35 + 1 \cdot 3 = 38)$

7: $\textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot n^g \cdot m^g$ $\quad (38 + 1 \cdot 2 \cdot 1 = 40)$

8: $\textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h|$ $\quad (40 + 1 \cdot 1 = \mathbf{41})$

and the persistent memory requirement as ...

$\textsc{Early Preaccumulation}$ in Tangent Mode

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^h| \cdot 16$ $\qquad (0 + 1 \cdot 16 = 16)$

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= 0$ $\qquad (16 + 0 = 16)$

(Lines 4 and 5 are not present in tangent mode.)

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^{g'}| \cdot 16$ $\qquad (16 + 2 \cdot 16 = 48)$

7: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^f| \cdot 16$ $\qquad (48 + 3 \cdot 16 = 96)$

8: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{|V^f \cup V^{g'} \cup V^h|, 0\} \cdot 8$ $\qquad (96 + 4 \cdot 8 = \mathbf{128})$

9: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^f| \cdot 16$ $\qquad (128 - 3 \cdot 16 = 80)$

10: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^{g'}| \cdot 16$ $\qquad (80 - 2 \cdot 16 = 48)$

11: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^h| \cdot 16$ $\qquad (48 - 1 \cdot 16 = 32)$

12: $\textsc{Mem}[\bar{F}] \mathrel{-}= |V^f \cup V^{g'} \cup V^h| \cdot 8$ $\qquad (32 - 4 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 128 bytes in line 8.

Early Preaccumulation in Tangent Mode with dco/c++

```
using TT=dco::gt1v<double,2>::type;
...
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
TT xt=dco::value(x), vht=dco::value(vh);
dco::derivative(xt)[0]=1; dco::derivative(vht)[1]=1;
TT vgt=g(xt,vht);
vg=x*dco::derivative(vgt)[0]+vh*dco::derivative(vgt)[1];
dco::value(vg)=dco::value(vgt);
y=f(x,vh,vg);
dco::derivative(y)=1;
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' ' << dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

Compare the adjoint code design patterns discussed so far with Early Preaccumulation for the computation of the Jacobian of F with

$$
\begin{aligned}
\dot{n} = n = n^h &= 10 \\
m^h = n^g &= 3 \\
m^g &= 1 \\
n^f &= 4 \\
m^f = m = \bar{m} &= 2 \\
\mathcal{R} &= 3 \\
|V^\phi| &= 100 \\
|E^\phi| &= 1000
\end{aligned}
$$

and for $\phi \in \{h, g, f\}$.

Preaccumulation by local

- adjoint AD (see above; sparsity?)
- jacobian_preaccumulator_t ("convenience feature")
- tangent AD ($n_g \leq \frac{\text{ERT}(\bar{g}))}{\text{ERT}(\dot{g})} \cdot m_g$ or $|T \setminus T^f| > \overline{\text{MEM}}$; sparsity)
- finite difference (smoothing?)
- hand coding (better local performance?)
- elimination techniques (scarcity?)
- *different tool (e.g. not C++ or dco/map for GPGPU)*

$\rightarrow$ Live

Use Early Preaccumulation and Back-Propagation whenever Early Preaccumulation should be applied and if Early Back-Propagation turns out to be feasible.

The elapsed run time of the Early Preaccumulation and Back-Propagation pattern evolves as

$$
\begin{array}{lll}
1: & \textsc{Ert}[\bar{F}] = 0 & \\
2: & \textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[h] & (\text{record } T^h) \\
3: & \textsc{Ert}[\bar{F}] \mathrel{+}= \textsc{Ert}[g^-] & (\text{preaccumulate } T^{g^-}) \\
4: & \textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[T^{g^-}] & (\text{record } T^{g^-}) \\
5: & \textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[T^{g^-}] & (\text{interpret } T^{g^-}) \\
6: & \textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[f(\tilde{g})] & (\text{record } T^{f(\tilde{g})}) \\
7: & \textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[f(\tilde{g})] & (\text{interpret } T^{f(\tilde{g})}) \\
8: & \textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[h] & (\text{interpret } T^h)
\end{array}
$$

subject to $\textsc{Mem}[\bar{F}] \leq \widehat{\textsc{Mem}}$.

Lemma

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = {} & \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] \\
& + \max\{ \\
& \quad \overrightarrow{\textsc{Mem}}[g^-] + \overleftarrow{\textsc{Mem}}[g^-] \;, \\
& \quad \overleftarrow{\textsc{Mem}}[g^-] + \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}], \overleftarrow{\textsc{Mem}}[g^-]\} \;, \\
& \quad \overrightarrow{\textsc{Mem}}[f(\tilde{g}] + \max\{\overleftarrow{\textsc{Mem}}[T^{f(\tilde{g})}], \overleftarrow{\textsc{Mem}}[T^{g^-}], \overleftarrow{\textsc{Mem}}[g^-]\} \\
& \} \;.
\end{aligned}
$$

Proof I

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[h]$ (record T^h)

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[g^-]$ (record for preaccumulation)

4: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-]$ (allocate adjoints for preaccumulation)

5: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[g^-]$ (preaccumulate T^{g^-})

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[T^{g^-}]$ (record T^{g^-})

7: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}] - \overleftarrow{\textsc{Mem}}[g^-], 0\}$ (grow memory for adjoints?)

8: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[T^{g^-}]$ (interpret T^{g^-})

9: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[f(\tilde{g}]$ (record $T^{f(\tilde{g})}$)

Proof II

10: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[T^{f(\tilde{g})}]$

$- \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}], \overleftarrow{\textsc{Mem}}[g^-]\}, 0\}$ (grow memory for adjoints?)

11: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[f(\tilde{g})]$ (interpret $T^{f(\tilde{g})}$)

12: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[h]$ (interpret T^h)

13: $\textsc{Mem}[\bar{F}] = 0$ (free adjoints)

Proof

$\textsc{Mem}[\bar{F}]$ takes local maxima in lines 4, 7 and 10 yielding

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = \max\{ & \\
& \overrightarrow{\textsc{Mem}}[h] + \overrightarrow{\textsc{Mem}}[g^-] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-] \ , \\
& \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-] \\
& \quad + \overrightarrow{\textsc{Mem}}[T^{g^-}] + \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}] - \overleftarrow{\textsc{Mem}}[g^-], 0\} \ , \\
& \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[g^-] + \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}] - \overleftarrow{\textsc{Mem}}[g^-], 0\} \\
& \quad + \overrightarrow{\textsc{Mem}}[f(\tilde{g}] + \max\{\overleftarrow{\textsc{Mem}}[T^{f(\tilde{g})}] - \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}], \overleftarrow{\textsc{Mem}}[g^-]\}, 0\} \\
\} &
\end{aligned}
$$

which simplifies to the equation stated in the lemma.

■

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$) for $\mathcal{R} = 3$.

The elapsed run time evolves as

1: $\textsc{Ert}[\bar{F}] = 0$

2: $\textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|)$ $\quad (0 + 3 \cdot (2 - 1 + 1) = 6)$

3: $\textsc{Ert}[\bar{F}] \mathrel{+}= |V^g| - n^g + |E^g| + n^g \cdot |E^g|$ $\quad (6 + 4 - 2 + 3 + 2 \cdot 3) = 17)$

4: $\textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot n^g \cdot m^g$ $\quad (17 + 3 \cdot 2 \cdot 1 = 23)$

5: $\textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot n^g \cdot m^g$ $\quad (23 + 1 \cdot 2 \cdot 1 = 25)$

6: $\textsc{Ert}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^{f(\tilde{g})}| - n^{f(\tilde{g})} + |E^{f(\tilde{g})}|)$ $\quad (25 + 3 \cdot (3 - 2 + 2) = 34)$

7: $\textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot (|E^{f(\tilde{g})}|)$ $\quad (34 + 1 \cdot 2 = 36)$

8: $\textsc{Ert}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h|$ $\quad (36 + 1 \cdot 1 = \mathbf{37})$

and the persistent memory requirement as ...

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^h| \cdot 16$ $\quad (0 + 1 \cdot 16 = 16)$

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= 0$ $\quad (16 + 0 = 16)$

(Lines 4 and 5 are not present in tangent mode.)

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^{g'}| \cdot 16$ $\quad (16 + 2 \cdot 16 = 48)$

7: $\textsc{Mem}[\bar{F}] \mathrel{+}= |V^h \cup V^{g'}| \cdot 8$ $\quad (48 + 3 \cdot 8 = \mathbf{72})$

8: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^{g'}| \cdot 16$ $\quad (72 - 2 \cdot 16 = 40)$

9: $\textsc{Mem}[\bar{F}] \mathrel{+}= |E^{f(\tilde{g})}| \cdot 16$ $\quad (40 + 2 \cdot 16 = 72)$

10: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{|V^{f(\tilde{g})}| - |V^{g'}|, 0\} \cdot 8$ $\quad (80 + \max\{3 - 3, 0\} \cdot 8 = 72)$

11: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^{f(\tilde{g})}| \cdot 16$ $\quad (72 - 2 \cdot 16 = 40)$

12: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^h| \cdot 16$ $\quad (40 - 1 \cdot 16 = 24)$

13: $\textsc{Mem}[\bar{F}] \mathrel{-}= |V^h \cup V^{f(\tilde{g})}| \cdot 8$ $\quad (24 - 3 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 72 bytes in line 4.

Early Preaccumulation and Back-Propagation with dco/c++

```
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
AM::tape_t::position_t pos=AM::global_tape->get_position();
vg=g(x,vh);
double pvg=dco::value(vg);
dco::derivative(vg)=1; // seed identity
AM::global_tape->interpret_adjoint_to(pos);
AM::global_tape->reset_to(pos);
vg=x*dco::derivative(x)+vh*dco::derivative(vh);
dco::derivative(x)=0; dco::derivative(vh)=0;
dco::value(vg)=pvg;
dco::derivative(vg)=1;
AM::global_tape->interpret_adjoint_to(pos);
AM::global_tape->reset_to(pos);
y=f(x,vh,pvg);
dco::derivative(y)=1; // seed
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' ' << dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

Compare the adjoint code design patterns discussed so far with Early Preaccumulation and Back-Propagation for the computation of the Jacobian of F with

$$
\begin{aligned}
\dot{n} &= n = n^h = 10 \\
m^h &= n^g = 3 \\
m^g &= 1 \\
n^f &= 4 \\
m^f &= m = \bar{m} = 2 \\
\mathcal{R} &= 3 \\
|V^\phi| &= 100 \\
|E^\phi| &= 1000
\end{aligned}
$$

and for $\phi \in \{h, g, f\}$.

Consider the modified version of the SDE code in

`SDE/early/f.h` and `SDE/early/main.cpp`.

It groups Monte Carlo paths into buckets and Euler-Maruyama steps into chunks.

Apply

- Early Back-Propagation (pathwise adjoints using ga1s)
- Early Preaccumulation (using ga1s, jacobian_preaccumlator_t, gt1v)
- Early Preaccumulation and Back-Propagation (using gt1v)

to individual buckets.

Run experiments and compare performances.

Outline

Beyond Black-Box Adjoints: Late Intervention

Contents:

- Late Back-Propagation (default)
- Late Recording
- Late Preaccumulation

Use LATE RECORDING if EARLY RECORDING yields infeasible memory requirement and EARLY PREACCUMULATION [AND BACK-PROPAGATION] are not applicable.

1: $\textsc{Ert}[\bar{F}] = 0$

2: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[h]$ (record T^h)

3: $\textsc{Ert}[\bar{F}] \mathrel{+}= \textsc{Ert}{\downarrow}[g]$ (store inputs of g)

4: $\textsc{Ert}[\bar{F}] \mathrel{+}= \textsc{Ert}[g]$ (run g)

5: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[f]$ (record T^f)

6: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[f]$ (interpret T^f)

7: $\textsc{Ert}[\bar{F}] \mathrel{+}= \textsc{Ert}{\uparrow}[g]$ (restore inputs of g)

8: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Ert}}[g]$ (record T^g)

9: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[g]$ (interpret T^g)

10: $\textsc{Ert}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Ert}}[h]$ (interpret T^h)

subject to $\textsc{Mem}[\bar{F}] \leq \widehat{\textsc{Mem}}$.

Lemma

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = {} & \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] \\
& + \max\{ \\
& \qquad \textsc{Mem}{\downarrow}[g] + n^g + \overrightarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[f], \\
& \qquad \overrightarrow{\textsc{Mem}}[g] + \max\{\overleftarrow{\textsc{Mem}}[f], \overleftarrow{\textsc{Mem}}[g]\} \\
& \ \} \ .
\end{aligned}
$$

Proof

$$
\begin{array}{rll}
1: & \textsc{Mem}[\bar{F}] = 0 & \\
2: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[h] & \text{(record } T^h\text{)} \\
3: & \textsc{Mem}[\bar{F}] \mathrel{+}= \textsc{Mem}{\downarrow}[g] & \text{(store inputs of } g\text{)} \\
4: & \textsc{Mem}[\bar{F}] \mathrel{+}= \textsc{Mem}[g] & \text{(run } g\text{)} \\
5: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[f] & \text{(record } T^f\text{)} \\
6: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f] & \text{(allocate adjoints)} \\
7: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[f] & \text{(interpret } T^f\text{)} \\
8: & \textsc{Mem}[\bar{F}] \mathrel{-}= \textsc{Mem}{\downarrow}[g] & \text{(restore inputs of } g\text{)} \\
9: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[g] & \text{(record } T^g\text{)} \\
10: & \textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[g] - \overleftarrow{\textsc{Mem}}[f], 0\} & \text{(grow memory for adjoints?)} \\
11: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[g] & \text{(interpret } T^g\text{)} \\
12: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[h] & \text{(interpret } T^h\text{)} \\
13: & \textsc{Mem}[\bar{F}] = 0 & \text{(free adjoints)}
\end{array}
$$

Proof

$\textsc{Mem}[\bar{F}]$ takes local maxima just prior to decrements, that is in lines 6 and 10. Hence, the above becomes equal to

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = \max\{ \\
& \overrightarrow{\textsc{Mem}}[h] + \textsc{Mem}{\downarrow}[g] + \underbrace{\textsc{Mem}[g]}_{=0} + \overrightarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f]\ , \\
& \overrightarrow{\textsc{Mem}}[h] + \underbrace{\textsc{Mem}[g]}_{=0} + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f] \\
& \quad + \overrightarrow{\textsc{Mem}}[g] + \max\{\overleftarrow{\textsc{Mem}}[g] - \overleftarrow{\textsc{Mem}}[f], 0\} \\
\}
\end{aligned}
$$

which simplifies to the equation stated in the lemma.

■

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$) for $\mathcal{R} = 3$.
The elapsed run time evolves as

$$
\begin{array}{rll}
1: & \mathrm{ERT}[\bar{F}] = 0 & \\
2: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|) & (0 + 3 \cdot (2 - 1 + 1) = 6) \\
3: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot n^g & (6 + 3 \cdot 2 = 12) \\
4: & \mathrm{ERT}[\bar{F}] \mathrel{+}= |V^g| - n^g & (12 + 4 - 2 = 14) \\
5: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^f| - n^f + |E^f|) & (14 + 3 \cdot (4 - 3 + 3) = 26) \\
6: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^f| & (26 + 1 \cdot 3 = 29) \\
7: & \mathrm{ERT}[\bar{F}] \mathrel{+}= n^g & (29 + 2 = 31) \\
8: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^g| - n^g + |E^g|) & (31 + 3 \cdot (4 - 2 + 3) = 46) \\
9: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^g| & (46 + 1 \cdot 3 = 49) \\
10: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h| & (49 + 1 \cdot 1 = \mathbf{50})
\end{array}
$$

and the persistent memory requirement as ...

1:	$\textsc{Mem}[\bar{F}] = 0$	
2:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \lvert E^h\rvert \cdot 16$	$(0 + 1 \cdot 16 = 16)$
3:	$\textsc{Mem}[\bar{F}] \mathrel{+}= n^g$	$(16 + 2 = 18)$
4:	$\textsc{Mem}[\bar{F}] \mathrel{+}= 0$	$(18 + 0 = 18)$

Note that the results of g need to be registered explicitly with the tape adding m^g entries in the following.

5:	$\textsc{Mem}[\bar{F}] \mathrel{+}= (\lvert E^f\rvert + m^g) \cdot 16$	$(18 + (3 + 1) \cdot 16 = 82)$
6:	$\textsc{Mem}[\bar{F}] \mathrel{+}= (\lvert V^h \cup V^f\rvert) \cdot 8$	$(82 + 4 \cdot 8 = \mathbf{114})$
7:	$\textsc{Mem}[\bar{F}] \mathrel{-}= (\lvert E^f\rvert + m^g\rvert) \cdot 16$	$(114 - (3 + 1) \cdot 16 = 50)$
8:	$\textsc{Mem}[\bar{F}] \mathrel{-}= n^g$	$(50 - 0 = 48)$
9:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \lvert E^g\rvert \cdot 16$	$(48 + 3 \cdot 16 = 96)$
10:	$\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\lvert V^g\rvert - \lvert V^f\rvert), 0\} \cdot 8$	$(96 + \max\{4 - 4, 0\} \cdot 8 = 96)$

11: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^g| \cdot 16$ $(96 - 3 \cdot 16 = 48)$

12: $\textsc{Mem}[\bar{F}] \mathrel{-}= |E^h| \cdot 16$ $(48 - 1 \cdot 16 = 32)$

13: $\textsc{Mem}[\bar{F}] = \ \mathrel{-}= (|V^h \cup V^f|) \cdot 8$ $(32 - 4 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 114 bytes in line 6. Neither x nor h are assumed to be persistent within F yielding the requirement for checkpointing in lines 3 and 8. The maximum persistent memory requirement would be equal to 112 bytes for persistent x and h.

```
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
dco::value(vg)=g(dco::value(x),dco::value(vh));
AM::tape_t::position_t pos1=AM::global_tape->get_position();
AM::global_tape->register_variable(vg);
AM::tape_t::position_t pos2=AM::global_tape->get_position();
y=f(x,vh,vg);
dco::derivative(y)=1;
AM::global_tape->interpret_adjoint_to(pos2);
double adjoint_vg=dco::derivative(vg);
AM::global_tape->reset_to(pos1);
vg=g(x,vh);
dco::derivative(vg)=adjoint_vg;
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' ' << dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

Compare the adjoint code design patterns discussed so far with LATE RECORDING for the computation of the Jacobian of F with

$$\dot{n} = n = n^h = 10$$
$$m^h = n^g = 3$$
$$m^g = 1$$
$$n^f = 4$$
$$m^f = m = \bar{m} = 2$$
$$\mathcal{R} = 3$$
$$|V^\phi| = 100$$
$$|E^\phi| = 1000$$

and for $\phi \in \{h, g, f\}$.

Use Late Preaccumulation whenever Late Recording is required and if local preaccumulation yields additional savings in terms of memory requirement and / or run time.

$$
\begin{array}{rll}
1: & \mathrm{ERT}[\bar{F}] = 0 & \\
2: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[h] & \text{(record } T^h\text{)} \\
3: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathrm{ERT}{\downarrow}[g] & \text{(store inputs of } g\text{)} \\
4: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathrm{ERT}[g] & \text{(run } g\text{)} \\
5: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[f] & \text{(record } T^f\text{)} \\
6: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[f] & \text{(interpret } T^h\text{)} \\
7: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathrm{ERT}{\uparrow}[g] & \text{(restore inputs of } g\text{)} \\
8: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \mathrm{ERT}[g^-] & \text{(preaccumulate } T^{g^-}\text{)} \\
9: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \overrightarrow{\mathrm{ERT}}[T^{g^-}] & \text{(record } T^{g^-}\text{)} \\
10: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[T^{g^-}] & \text{(interpret } T^{g^-}\text{)} \\
11: & \mathrm{ERT}[\bar{F}] \mathrel{+}= \overleftarrow{\mathrm{ERT}}[h] & \text{(interpret } T^h\text{)}
\end{array}
$$

subject to $\mathrm{MEM}[\bar{F}] \leq \widehat{\mathrm{MEM}}$.

Lemma

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = {} & \overrightarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[h] \\
& + \max\{ \\
& \qquad \textsc{Mem}\!\downarrow\![g] + \overrightarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[f] \;, \\
& \qquad \overrightarrow{\textsc{Mem}}[g^-] + \max\{\overleftarrow{\textsc{Mem}}[g^-], \overleftarrow{\textsc{Mem}}[f]\} \;, \\
& \qquad \overrightarrow{\textsc{Mem}}[T^{g^-}] + \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}], \overleftarrow{\textsc{Mem}}[g^-], \overleftarrow{\textsc{Mem}}[f])\} \\
& \} \; .
\end{aligned}
$$

Proof I

1: $\textsc{Mem}[\bar{F}] = 0$

2: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[h]$ (record T^h)

3: $\textsc{Mem}[\bar{F}] \mathrel{+}= \textsc{Mem}{\downarrow}[g]$ (store inputs of g)

4: $\textsc{Mem}[\bar{F}] \mathrel{+}= \textsc{Mem}[g]$ (run g)

5: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[f]$ (record T^f)

6: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f]$ (allocate adjoints)

7: $\textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[f]$ (interpret T^f)

8: $\textsc{Mem}[\bar{F}] \mathrel{-}= \textsc{Mem}{\downarrow}[g]$ (restore inputs of g)

9: $\textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[g^-]$ (record for preaccumulation)

10: $\textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[g^-] - \overleftarrow{\textsc{Mem}}[f], 0\}$ (grow memory for adjoints?)

Proof II

$$
\begin{array}{lll}
11: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[g^-] & \text{(preaccumulate } T^{g^-}\text{)} \\
12: & \textsc{Mem}[\bar{F}] \mathrel{+}= \overrightarrow{\textsc{Mem}}[T^{g^-}] & \text{(record } T^{g^-}\text{)} \\
13: & \textsc{Mem}[\bar{F}] \mathrel{+}= \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}] - \max\{\overleftarrow{\textsc{Mem}}[g^-], \overleftarrow{\textsc{Mem}}[f]\}, 0\} & \text{(grow memory for adjoints?)} \\
14: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[T^{g^-}] & \text{(interpret } T^{g^-}\text{)} \\
15: & \textsc{Mem}[\bar{F}] \mathrel{-}= \overrightarrow{\textsc{Mem}}[h] & \text{(interpret } T^{h}\text{)} \\
16: & \textsc{Mem}[\bar{F}] = 0 & \text{(free adjoints)}
\end{array}
$$

Proof III

$\textsc{Mem}[\bar{F}]$ takes local maxima just prior to decrements, that is in lines 6, 10 and 13. Hence, the above becomes equal to

$$
\begin{aligned}
\textsc{Mem}[\bar{F}] = \max\{ & \\
& \overrightarrow{\textsc{Mem}}[h] + \textsc{Mem}{\downarrow}[g] + \underbrace{\textsc{Mem}[g]}_{=0} + \overrightarrow{\textsc{Mem}}[f] + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f]\ , \\
& \overrightarrow{\textsc{Mem}}[h] + \underbrace{\textsc{Mem}[g]}_{=0} + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f] \\
& \quad + \overrightarrow{\textsc{Mem}}[g^-] + \max\{\overleftarrow{\textsc{Mem}}[g^-] - \overleftarrow{\textsc{Mem}}[f], 0\}\ , \\
& \overrightarrow{\textsc{Mem}}[h] + \underbrace{\textsc{Mem}[g]}_{=0} + \overleftarrow{\textsc{Mem}}[h] + \overleftarrow{\textsc{Mem}}[f] + \max\{\overleftarrow{\textsc{Mem}}[g^-] - \overleftarrow{\textsc{Mem}}[f], 0\} \\
& \quad + \overrightarrow{\textsc{Mem}}[T^{g^-}] + \max\{\overleftarrow{\textsc{Mem}}[T^{g^-}] - \max\{\overleftarrow{\textsc{Mem}}[g^-], \overleftarrow{\textsc{Mem}}[f]\}\}, 0\} \\
\} &
\end{aligned}
$$

which simplifies to the equation stated in the lemma.

■

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$) for $\mathcal{R} = 3$ and with preaccumulation of g' in tangent mode.

The elapsed run time evolves as

1:	$\text{ERT}[\bar{F}] = 0$	
2:	$\text{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (\lvert V^h \rvert - n^h + \lvert E^h \rvert)$	$(0 + 3 \cdot (2 - 1 + 1) = 6)$
3:	$\text{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot n^g$	$(6 + 3 \cdot 2 = 12)$
4:	$\text{ERT}[\bar{F}] \mathrel{+}= \lvert V^g \rvert - n^g$	$(12 + 4 - 2 = 14)$
5:	$\text{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (\lvert V^f \rvert - n^f + \lvert E^f \rvert)$	$(14 + 3 \cdot (4 - 3 + 3) = 26)$
6:	$\text{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot \lvert E^f \rvert$	$(26 + 1 \cdot 3 = 29)$
7:	$\text{ERT}[\bar{F}] \mathrel{+}= n^g$	$(29 + 2 = 31)$
8a:	$\text{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (\lvert V^g \rvert - n^g + \lvert E^g \rvert)$	$(31 + 3 \cdot (4 - 2 + 3) = 46)$
8b:	$\text{ERT}[\bar{F}] \mathrel{+}= \bar{m}^g \cdot \lvert E^g \rvert$	$(46 + 1 \cdot 3 = 49)$
9:	$\text{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot \lvert E^{g^-} \rvert$	$(49 + 3 \cdot 2 = 55)$
10:	$\text{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot \lvert E^{g^-} \rvert$	$(55 + 1 \cdot 2 = 57)$
11:	$\text{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot \lvert E^h \rvert$	$(57 + 1 \cdot 1 = 58)$

and the persistent memory requirement as ...

1: $\mathrm{MEM}[\bar{F}] = 0$

2: $\mathrm{MEM}[\bar{F}] \mathrel{+}= |E^h| \cdot 16$ $\quad (0 + 1 \cdot 16 = 16)$

3: $\mathrm{MEM}[\bar{F}] \mathrel{+}= n^g$ $\quad (16 + 2 = 18)$

4: $\mathrm{MEM}[\bar{F}] \mathrel{+}= 0$ $\quad (18 + 0 = 18)$

5: $\mathrm{MEM}[\bar{F}] \mathrel{+}= (|E^f| + m^g) \cdot 16$ $\quad (18 + (3 + 1) \cdot 16 = 82)$

6: $\mathrm{MEM}[\bar{F}] \mathrel{+}= (|V^h \cup V^f|) \cdot 8$ $\quad (82 + 4 \cdot 8 = 114)$

7: $\mathrm{MEM}[\bar{F}] \mathrel{-}= (|E^f| + m^g|) \cdot 16$ $\quad (114 - 4 \cdot 16 = 50)$

8: $\mathrm{MEM}[\bar{F}] \mathrel{-}= n^g$ $\quad (50 - 2 = 48)$

9: $\mathrm{MEM}[\bar{F}] \mathrel{+}= |E^g| \cdot 16$ $\quad (48 + 3 \cdot 16 = 96)$

10: $\mathrm{MEM}[\bar{F}] \mathrel{+}= \max\{|V^g| - |V^f|), 0\} \cdot 8$ $\quad (96 + \max\{4 - 4, 0\} \cdot 8 = 96)$

$$
\begin{array}{lll}
11: & \textsc{Mem}[\bar{F}] \mathrel{-}= |E^g| \cdot 16 & (96 - 3 \cdot 16 = 48) \\
12: & \textsc{Mem}[\bar{F}] \mathrel{+}= n^g \cdot m^g \cdot 16 & (48 + 2 \cdot 1 \cdot 16 = 80) \\
13: & \textsc{Mem}[\bar{F}] \mathrel{+}= \max\{n^g + m^g - \max\{|V^g|, |V^f|\}, 0\} \cdot 8 & (80 + \max\{2 + 1 - \max\{4, 4\}, 0\} \cdot 8 = 80) \\
14: & \textsc{Mem}[\bar{F}] \mathrel{-}= n^g \cdot m^g \cdot 16 & (80 - 2 \cdot 1 \cdot 16 = 48) \\
15: & \textsc{Mem}[\bar{F}] \mathrel{-}= |E^h| \cdot 16 & (48 - 1 \cdot 16 = 32) \\
16: & \textsc{Mem}[\bar{F}] = \; \mathrel{-}= (|V^h \cup V^f|) \cdot 8 & (32 - 4 \cdot 8 = 0)
\end{array}
$$

yielding the maximum persistent memory requirement of 114 bytes in line 6.

Late Preaccumulation in adjoint mode with dco/c++ I

```
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
dco::value(vg)=g(dco::value(x),dco::value(vh));
AM::tape_t::position_t pos1=AM::global_tape->get_position();
AM::global_tape->register_variable(vg);
AM::tape_t::position_t pos2=AM::global_tape->get_position();
y=f(x,vh,vg);
dco::derivative(y)=1; // seed
AM::global_tape->interpret_adjoint_to(pos2);
double adjoint_vg=dco::derivative(vg);
double adjoint_vh=dco::derivative(vh); dco::derivative(vh)=0;
double adjoint_x=dco::derivative(x); dco::derivative(x)=0;
AM::global_tape->reset_to(pos1);
vg=g(x,vh);
dco::derivative(vg)=1;
AM::global_tape->interpret_adjoint_to(pos1);
AM::global_tape->reset_to(pos1);
vg=x*dco::derivative(x)+vh*dco::derivative(vh);
```

Late Preaccumulation in adjoint mode with dco/c++ II

```
dco::derivative(x)=adjoint_x;
dco::derivative(vh)=adjoint_vh;
dco::derivative(vg)=adjoint_vg;
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' '<< dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

Consider the computation of the Jacobian of F in adjoint mode ($\bar{m} = m = 1$) for $\mathcal{R} = 3$ and with preaccumulation of g' in tangent mode.
The elapsed run time evolves as

1: $\mathrm{ERT}[\bar{F}] = 0$
2: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^h| - n^h + |E^h|)$ $\quad (0 + 3 \cdot (2 - 1 + 1) = 6)$
3: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot n^g$ $\quad (6 + 3 \cdot 2 = 12)$
4: $\mathrm{ERT}[\bar{F}] \mathrel{+}= |V^g| - n^g$ $\quad (12 + 4 - 2 = 14)$
5: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot (|V^f| - n^f + |E^f|)$ $\quad (14 + 3 \cdot (4 - 3 + 3) = 26)$
6: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^f|$ $\quad (26 + 1 \cdot 3 = 29)$
7: $\mathrm{ERT}[\bar{F}] \mathrel{+}= n^g$ $\quad (29 + 2 = 31)$
8: $\mathrm{ERT}[\bar{F}] \mathrel{+}= |V^g| - n^g + |E^g| + n^g \cdot |E^g|)$ $\quad (31 + 4 - 2 + 3 + 2 \cdot 3) = 42)$
9: $\mathrm{ERT}[\bar{F}] \mathrel{+}= \mathcal{R} \cdot |E^{g^-}|$ $\quad (42 + 3 \cdot 2 = 48)$

Late Preaccumulation in Tangent Mode for Toy Example II

10: $\text{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^{g^-}|$ $(48 + 1 \cdot 2 = 50)$

11: $\text{ERT}[\bar{F}] \mathrel{+}= \bar{m} \cdot |E^h|$ $(50 + 1 \cdot 1 = \mathbf{51})$

and the persistent memory requirement as ...

1: $\text{MEM}[\bar{F}] = 0$

2: $\text{MEM}[\bar{F}] \mathrel{+}= |E^h| \cdot 16$ $(0 + 1 \cdot 16 = 16)$

3: $\text{MEM}[\bar{F}] \mathrel{+}= n^g$ $(16 + 2 = 18)$

4: $\text{MEM}[\bar{F}] \mathrel{+}= 0$ $(18 + 0 = 18)$

5: $\text{MEM}[\bar{F}] \mathrel{+}= (|E^f| + m^g) \cdot 16$ $(18 + (3 + 1) \cdot 16 = 82)$

6: $\text{MEM}[\bar{F}] \mathrel{+}= (|V^h \cup V^f|) \cdot 8$ $(82 + 4 \cdot 8 = \mathbf{114})$

7: $\text{MEM}[\bar{F}] \mathrel{-}= (|E^f| + m^g|) \cdot 16$ $(114 - 4 \cdot 16 = 50)$

8: $\text{MEM}[\bar{F}] \mathrel{-}= n^g$ $(50 - 2 = 48)$

9: $\textrm{MEM}[\bar{F}] \mathrel{+}= 0$ $(48 + 0 = 48)$

10: $\textrm{MEM}[\bar{F}] \mathrel{+}= n^g \cdot m^g \cdot 16$ $(48 + 2 \cdot 1 \cdot 16 = 80)$

11: $\textrm{MEM}[\bar{F}] \mathrel{+}= \max\{n^g + m^g - \max\{|V^g|, |V^f|\}, 0\} \cdot 8$ $(80 + \max\{2 + 1 - \max\{4, 4\}, 0\} \cdot 8 = 80)$

12: $\textrm{MEM}[\bar{F}] \mathrel{-}= n^g \cdot m^g \cdot 16$ $(80 - 2 \cdot 1 \cdot 16 = 48)$

(Lines 13 and 14 are not present in tangent mode.)

15: $\textrm{MEM}[\bar{F}] \mathrel{-}= |E^h| \cdot 16$ $(48 - 1 \cdot 16 = 32)$

16: $\textrm{MEM}[\bar{F}] = \mathrel{-}= (|V^h \cup V^f|) \cdot 8$ $(32 - 4 \cdot 8 = 0)$

yielding the maximum persistent memory requirement of 114 bytes in line 6.

Late Preaccumulation in Tangent Mode with dco/c++ I

```
AT x=1.5, vh, vg, y;
AM::global_tape=AM::tape_t::create();
AM::global_tape->register_variable(x);
vh=h(x);
dco::value(vg)=g(dco::value(x),dco::value(vh));
AM::tape_t::position_t pos1=AM::global_tape->get_position();
AM::global_tape->register_variable(vg);
AM::tape_t::position_t pos2=AM::global_tape->get_position();
y=f(x,vh,vg);
dco::derivative(y)=1; // seed
AM::global_tape->interpret_adjoint_to(pos2);
double adjoint_vg=dco::derivative(vg);
double adjoint_vh=dco::derivative(vh); dco::derivative(vh)=0;
double adjoint_x=dco::derivative(x); dco::derivative(x)=0;
AM::global_tape->reset_to(pos1);
TT xt=dco::value(x), vht=dco::value(vh);
dco::derivative(xt)[0]=1; dco::derivative(vht)[1]=1;
TT vgt=g(xt,vht);
vg=x*dco::derivative(vgt)[0]+vh*dco::derivative(vgt)[1];
dco::derivative(x)=adjoint_x;
```

LATE PREACCUMULATION in Tangent Mode with dco/c++ II

```
dco::derivative(vh)=adjoint_vh;
dco::derivative(vg)=adjoint_vg;
AM::global_tape->interpret_adjoint();
cout << dco::value(y) << ' '<< dco::derivative(x) << endl;
AM::tape_t::remove(AM::global_tape);
```

Compare the adjoint code design patterns discussed so far with LATE PREACCUMULATION in tangent and adjoint modes for the computation of the Jacobian of F with

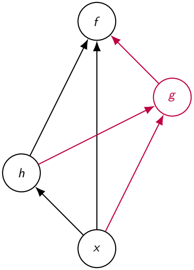
$$\dot{n} = n = n^h = 10$$
$$m^h = n^g = 3$$
$$m^g = 1$$
$$n^f = 4$$
$$m^f = m = \bar{m} = 2$$
$$\mathcal{R} = 3$$
$$|V^\phi| = 100$$
$$|E^\phi| = 1000$$

and for $\phi \in \{h, g, f\}$.

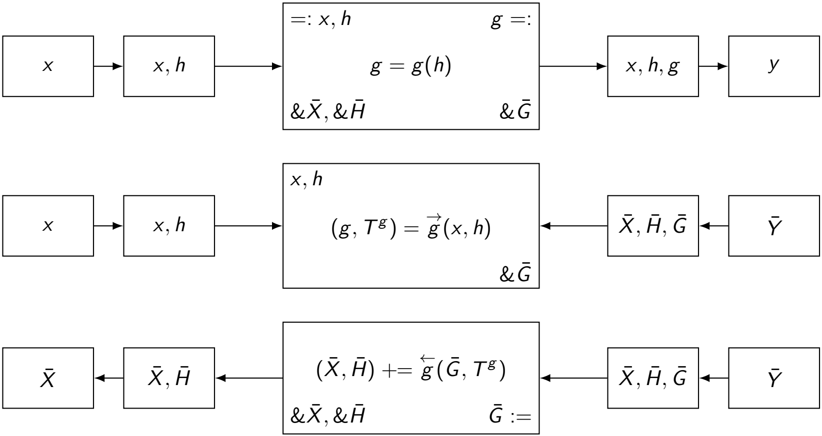
Outline

- External adjoints interface of dco/c++ illustrated with Toy
- Case study: SDE
  - External adjoint buckets
  - External adjoint chunks
  - External adjoint chunks inside of external adjoint buckets

Recall



External Adjoints (e.g. Late Recording)



(f1) create EA

(f2) store data required by adjoint $\Rightarrow$ e.g. $=: x, h$

(f3) store address of adjoint of active input $\Rightarrow \& \bar{X}, \& \bar{H}$

(f4) run local primal $g = g(x, h) \Rightarrow g =:$

(f5) register active output with tape and store address of its adjoint $\Rightarrow \& \bar{G}$

(f6) register adjoint callback with EA and insert EA into tape

(r1) read data required by adjoint $\Rightarrow$ e.g. x, h

(r2) run local augmented primal $(g, T^g) = \overrightarrow{g}(x, h)$

(r3) get adjoint output $\Rightarrow \bar{G} :=$

(r4) run local adjoint $\overleftarrow{g}(\bar{G}, T^g)$

(r5) increment adjoint input $\Rightarrow += (\bar{X}, \bar{H})$

External Adjoint Interface of dco/c++

- write_data / read_data to store / recover data required for context-free evaluation of the adjoint target sub-program
- register_input / register_output to establish pairs of twins representing one a the same program variable in the target sub-program and in the context.
- get_output_adjoint to extract adjoints of the outputs of the target sub-program from the context
- increment_adjoint_input to transfer adjoints of the inputs of the target sub-program to the context

Live: External Adjoints with dco/c++ (Augmented Primal)

```
template<typename T>
typename dco::ga1s<T>::type g(const typename dco::ga1s<T>::type &x, const typename
    dco::ga1s<T>::type &h) {
  using AM=typename dco::ga1s<T>; using A=typename AM::type;
  typename AM::external_adjoint_object_t* EA=AM::global_tape->template
      create_callback_object<typename AM::external_adjoint_object_t>();
  EA->register_input(x);
  EA->register_input(h);
  T xv=dco::value(x); EA->write_data(xv);
  T hv=dco::value(h); EA->write_data(hv);
  T yv=g(xv,hv);
  A gv=EA->register_output(yv);
  AM::global_tape->template insert_callback<typename AM::external_adjoint_object_t>(ag
      <T>,EA);
  return gv;
}
```

Live: External Adjoints with dco/c++ (Adjoint)

```
template <typename T>
void ag(typename dco::ga1s<T>::external_adjoint_object_t *EA) {
  using AM=typename dco::ga1s<T>; using A=typename AM::type;
  A x=EA->template read_data<T>();
  A h=EA->template read_data<T>();
  AM::global_tape->register_variable(x);
  AM::global_tape->register_variable(h);
  typename AM::tape_t::position_t pos=AM::global_tape->get_position();
  A gv=g(x,h);
  dco::derivative(gv)=EA->get_output_adjoint();
  AM::global_tape->interpret_adjoint_and_reset_to(pos);
  EA->increment_input_adjoint(dco::derivative(x));
  EA->increment_input_adjoint(dco::derivative(h));
}
```

Live: External Adjoints with dco/c++ (Driver)

```
int main() {
  using P=double;
  using AM=dco::ga1s<P>;
  using A=AM::type;
  AM::global_tape=AM::tape_t::create();
  A x=1.5,hv,gv,y;
  AM::global_tape->register_variable(x);
  hv=h(x);
  gv=g<P>(x,hv);
  y=f(x,hv,gv);
  dco::derivative(y)=1;
  AM::global_tape->interpret_adjoint();
  cout << dco::value(y) << " "
       << dco::derivative(x) << endl;
  AM::tape_t::remove(AM::global_tape);
  return 0;
}
```

Consider the modified version of the SDE code in

`SDE/external/f.h` and `SDE/external/main.cpp`.

It groups Monte Carlo paths into buckets and Euler-Maruyama steps into chunks.

Apply external adjoints to implement

- bucket-wise adjoints
- equidistant checkpointing of chunks
- bucket-wise adjoints with equidistant checkpointing of chunks.

Run experiments and compare performances.

Outline

The Hessian

$$F'' = F''(x) \equiv \frac{d^2F}{dx^2}(x) = \left(\frac{d^2F}{dx_i dx_j}(x)\right) \in \mathbb{R}^{m\times n\times n}$$

of a twice continuously differentiable multivariate vector function $F : \mathbb{R}^n \to \mathbb{R}^m$ can be approximated at a given point $x \in \mathbb{R}^n$ as a (central) finite difference approximation of the Jacobian of a (central) finite difference approximation of the Jacobian

$$F' = F'(x) \equiv \frac{dF}{dx}(x) = \left(\frac{dF}{dx_i}(x)\right) \in \mathbb{R}^{m\times n}$$

of F :

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

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

Similarly, for $f : \mathbb{R}^n \to \mathbb{R}$

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

A natural approach to implementing second-order finite differences by perturbing the Jacobian / gradient drivers follows immediately.

Accuracy suffers from the need to square (the small) Δx_j. A perturbation of

$$\Delta x_j = \begin{cases} \sqrt{\sqrt{\epsilon}} & x_j = 0 \\ \sqrt{\sqrt{\epsilon}} \cdot |x_j| & x_j \neq 0 \end{cases}$$

with machine epsilon ϵ dependent on the floating-point precision typically yields a reasonable compromise between accuracy and numerical stability.

Approximate Hessian by Finite Differences

`SDE/fd/Hessian/main.cpp`

- inspect
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)

Application of tangent AD to the first-order tangent

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

yields

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

as

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

During the i-th application of tangent AD overset dots are replaced by paranthesized superscripts (i). Multiple superscripts are written as comma-separated multi-indices.

AD of the first-order tangent

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

in tangent mode yields

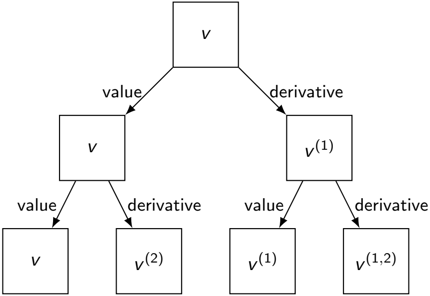
$$\begin{pmatrix} y^{(2)} \\ y^{(1,2)} \end{pmatrix} \equiv \frac{d \begin{pmatrix} y \\ y^{(1)} \end{pmatrix}}{d \begin{pmatrix} x \\ x^{(1)} \end{pmatrix}} \cdot \begin{pmatrix} x^{(2)} \\ x^{(1,2)} \end{pmatrix} = \begin{pmatrix} \frac{dy}{dx} \cdot x^{(2)} \left[+ \frac{dy}{dx^{(1)}} \cdot x^{(1,2)}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

implying with[3] $y^{(1)} = {x^{(1)}}^T \cdot f'(x)^T$ and $f''(x)^T = f''(x)$

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

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

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



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

`SDE/gt1s_gt1s/main.cpp`

- implement starting from primal `SDE/main.cpp`
- build (`Makefile`)
- run
- time (`/usr/bin/time -v`)
- compare with finite differences

Application of tangent AD to the first-order adjoint

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

yields

$$\begin{pmatrix} y & y^{(2)} & x_{(1)} & x_{(1)}^{(2)} \end{pmatrix}^T = f_{(1)}^{(2)}\left(x, x^{(2)}, y_{(1)}, y_{(1)}^{(2)}\right)$$

as

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

During the i-th application of adjoint AD overset bars are replaced by paranthesized subscripts (i). Multiple subscripts are written as comma-separated multi-indices.

Derivation

AD of the first-order adjoint

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

in tangent mode yields

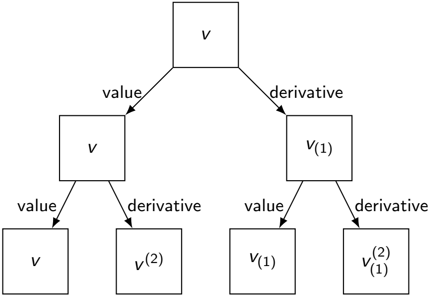
$$\begin{pmatrix} y^{(2)} \\ {x_{(1)}^{(2)}}^T \end{pmatrix} \equiv \frac{d\begin{pmatrix} y \\ x_{(1)}^T \end{pmatrix}}{d\begin{pmatrix} x \\ y_{(1)} \end{pmatrix}} \cdot \begin{pmatrix} x^{(2)} \\ y_{(1)}^{(2)} \end{pmatrix} = \begin{pmatrix} \frac{dy}{dx} \cdot x^{(2)} \left[+ \frac{dy}{dy_{(1)}}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

implying with[4] $x_{(1)}^T = f'(x)^T \cdot y_{(1)}$ and $f''(x)^T = f''(x)$

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

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

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



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

SDE/gt1s_ga1s/main.cpp

- implement starting from SDE/gt1s_gt1s/main.cpp
- build (Makefile)
- run
- time (/usr/bin/time -v)
- compare with finite differences

For the given implementation of the Heston Stochastic Volatility Model with Euler Discretization use dco/c++ to compute the Hessian in second-order

- tangent mode with dco/c++
- adjoint mode with dco/c++

Compare runtimes and memory requirements. solutions.

All previously developed solutions for early and late recording / preaccumulation / back-propagation of adjoints fit seamlessly into the second- (and higher-) order scenario, e.g,

- early back-propagation
- early preaccumulation in adjoint mode
- early preaccumulation in vector tangent mode
- early back-propagation based on preaccumulation in vector tangent mode
- (late) external adjoint buckets
- (late) external adjoint chunks
- (late) external adjoint chunks inside late adjoint buckets

... inspect, build, run, time, compare.

The remaining two combinations for second-order adjoint AD can also be implemented with dco/c++, i.e. adjoints of tangents (less common)

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

and adjoints of adjoints (advantageous for certain smaller problems)

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

Arbitrary combinations of tangent and adjoints are possible with dco/c++, e.g.

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

Note commutativity of multiplication in *index notation*.

Higher-Order Differential Invariants

Which of the following identities hold / do not?

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

$$x_{(2)} \cdot x^{(2)} = y_{(2)}^{(1)} \cdot y^{(1,2)}$$

$$x_{(2)} \cdot x^{(2)} = x^{(1,2)} \cdot y_{(1)}^{(2)}$$

$$x_{(3)} \cdot x^{(3)} = y_{(3)}^{(1,2)} \cdot y^{(1,2,3)}$$

$$x_{(3)} \cdot x^{(3)} = x_{(2,3)} \cdot x_{(2)}^{(3)}$$

$$x_{(3)} \cdot x^{(3)} = x_{(2,3)}^{(1)} \cdot x_{(1)}^{(2,3)}$$

$$x_{(6)} \cdot x^{(6)} = x_{(4,6)}^{(1,3,5)} \cdot x_{(4)}^{(1,3,5,6)}$$

more on

U. N.: *Differential Invariants.* arXiv:2101.03334 [math.NA]. Submitted.

Outline

BLAS

- axpy
- inner vector product
- matrix-vector product
- matrix-matrix product

The adjoint of the axpy operation $z = a \cdot x + y$ with active $z, x, y \in \mathbb{R}^n$ and $a \in \mathbb{R}$ is computed as

$$\begin{aligned} a_{(1)} &= \langle x, z_{(1)} \rangle \\ x_{(1)} &= a \cdot z_{(1)} \\ y_{(1)} &= z_{(1)} \end{aligned}$$

for $z_{(1)} \in \mathbb{R}$ yielding $a_{(1)}, x_{(1)}, y_{(1)} \in \mathbb{R}$. Multiple uses of a, x, y yield incremental adjoints as

$$\begin{aligned} a_{(1)} &= \langle x, z_{(1)} \rangle + a_{(1)} \\ x_{(1)} &= a \cdot z_{(1)} + x_{(1)} \\ y_{(1)} &= 1 \cdot z_{(1)} + y_{(1)} \end{aligned}$$

Inner Vector Product

The adjoint of an inner vector product

$$y = \langle \mathbf{a}, \mathbf{x} \rangle \equiv \mathbf{a}^T \cdot \mathbf{x} = \sum_{i=0}^{n-1} a_i \cdot x_i$$

with active inputs $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathbb{R}^n$ yielding the active output $y \in \mathbb{R}$ is computed as

$$\begin{aligned} \mathbf{a}_{(1)} &= \mathbf{x} \cdot y_{(1)} \\ \mathbf{x}_{(1)} &= \mathbf{a} \cdot y_{(1)} \end{aligned}$$

for $y_{(1)} \in \mathbb{R}$ yielding $\mathbf{a}_{(1)} \in \mathbb{R}^n$ and $\mathbf{x}_{(1)} \in \mathbb{R}^n$ as well as the corresponding incremental adjoint

$$\begin{aligned} \mathbf{a}_{(1)} &= \mathbf{x} \cdot y_{(1)} + \mathbf{a}_{(1)} \\ \mathbf{x}_{(1)} &= \mathbf{a} \cdot y_{(1)} + \mathbf{x}_{(1)} \ . \end{aligned}$$

The adjoint of a matrix-vector product

$$\mathbf{y} = A \cdot \mathbf{x} \equiv (\mathbf{a}_i \cdot \mathbf{x})_{i=0,\ldots,m-1}$$

with active inputs $A \in \mathbb{R}^{m \times n}$ and $\mathbf{x} \in \mathbb{R}^n$ yielding the active output $\mathbf{y} \in \mathbb{R}^m$ is computed as

$$\begin{aligned} \mathbf{x}_{(1)} &= A^T \cdot \mathbf{y}_{(1)} \\ A_{(1)} &= \mathbf{y}_{(1)} \cdot \mathbf{x}^T \end{aligned}$$

for $\mathbf{y}_{(1)} \in \mathbb{R}^m$ yielding $\mathbf{x}_{(1)} \in \mathbb{R}^n$ and $A_{(1)} \in \mathbb{R}^{m \times n}$ as well as the corresponding incremental adjoint

$$\begin{aligned} \mathbf{x}_{(1)} &= A^T \cdot \mathbf{y}_{(1)} + \mathbf{x}_{(1)} \\ A_{(1)} &= \mathbf{y}_{(1)} \cdot \mathbf{x}^T + A_{(1)} \; . \end{aligned}$$

The adjoint of a matrix-matrix product $Y = A \cdot X$ with active inputs $A \in \mathbb{R}^{m \times p}$, $X \in \mathbb{R}^{p \times n}$ yielding the active output $Y \in \mathbb{R}^{m \times n}$ is computed as

$$A_{(1)} = Y_{(1)} \cdot X^T$$
$$X_{(1)} = A^T \cdot Y_{(1)}$$

for $Y_{(1)} \in \mathbb{R}^{m \times n}$ yielding $A_{(1)} \in \mathbb{R}^{m \times p}$ and $X_{(1)} \in \mathbb{R}^{p \times n}$ as well as the corresponding incremental adjoint

$$A_{(1)} = Y_{(1)} \cdot X^T + A_{(1)}$$
$$X_{(1)} = A^T \cdot Y_{(1)} + X_{(1)} \ .$$

Implicit Functions

- Systems of Nonlinear Equations
- Systems of Linear Equations
- First-Order Optimality Conditions
- Case Study: SDE
- (First-Order) Error Analysis

Symbolic Adjoints of Implicit Functions (Systems of Nonlinear Equations)

Let $r = R(x(p), p) = 0$ with $R : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$ be continuously differentiable wrt. both x and p.

From

$$\frac{dR}{dp} = \frac{\partial R}{\partial p} + \frac{dR}{dx} \cdot \frac{dx}{dp} = 0$$

follows

$$\underbrace{\frac{dx}{dp}}_{\in \mathbb{R}^{n_x \times n_p}} = - \underbrace{\frac{dR}{dx}^{-1}}_{\in \mathbb{R}^{n_x \times n_x}} \cdot \underbrace{\frac{\partial R}{\partial p}}_{\in \mathbb{R}^{n_x \times n_p}}$$

implying the adjoint

$$p_{(1)} \equiv x_{(1)} \cdot \frac{dx}{dp} = \underbrace{-x_{(1)} \cdot R_x^{-1}}_{R_x \cdot z_{(1)} = -x_{(1)}} \cdot R_p \; .$$

1. Solve the primal system $R(x(p), p) = 0$ to the required accuracy yielding an approximate root $x^* = S(R, x, p)$.
2. Compute the Jacobian R_x of the residual wrt. the state, e.g. using dco/c++ in tangent mode (dco::gt1s<T>::type).
3. Solve the system of linear equations $R_x \cdot z_{(1)} = -x_{(1)}$ for the given adjoint of the primal solution $x_{(1)}$ yielding $z_{(1)} \in \mathbb{R}^{1 \times n_x}$.
4. Evaluate the adjoint $p_{(1)} = z_{(1)} \cdot R_p$

Algorithmic differentiation of the iterative primal solver can be avoided at (or close to[5]) the primal solution.

[5] See below for first-order error analysis.

"Hello World" Example: Nonlinear Equation

$$
\begin{tikzcd}
R = R(x,p) \equiv x^2 - p = 0 \arrow[r, "x_{(1)} \cdot \frac{d}{dp}"] \arrow[d, "(x^*, T) = \overrightarrow{S}(R, p, x)"'] & x_{(1)} \cdot \left(2 \cdot x \cdot \frac{dx}{dp} - 1\right) = 2 \cdot x \cdot \underbrace{x_{(1)} \cdot \frac{dx}{dp}}_{=p_{(1)}} - x_{(1)} = 0 \arrow[d, "p_{(1)} = \tilde{S}(x, x_{(1)})"] \\
x^* \approx \sqrt{p} \arrow[r, "\overleftarrow{S}(T, x_{(1)})"'] & p_{(1)} = \frac{x_{(1)}}{2 \cdot x^*} \approx \frac{x_{(1)}}{2 \cdot x} \; ?
\end{tikzcd}
$$

Embedded Nonlinear Equation: Primal

$$x := x^2; \quad x^2 - p = 0; \quad x := e^{-x};$$

```
template<typename T>
void newton(T& x, const T& p, const T& eps) {
  do { // perform at least one iteration to fix data dependence
    x=x-(x*x-p)/(2*x);
  } while (fabs(x*x-p)>eps);
}

int main() {
  using T=double; T p=2, x=10; const T eps=1e-12;
  x=pow(x,2); newton(x,p,eps); x=exp(-x);
  std::cout << "x=" << x << std::endl;
  return 0;
}
```

Embedded Nonlinear Equation: Hand-Written Algorithmic Adjoint

```
int main() {
  using T=double;
  T p_v=2, x_v=10, p_a=0, x_a=1;
  const T eps=1e-12;
  std::stack<T> tbr_T;

  tbr_T.push(x_v);
  x_v=pow(x_v,2);
  newton_a(x_v,x_a,p_v,p_a,eps);
  x_v=exp(-x_v);
  std::cout << "x=" << x_v << std::endl;

  x_a=-x_v*x_a;
  newton_a(x_v,x_a,p_v,p_a,eps,true);
  x_v=tbr_T.top(); tbr_T.pop();
  x_a=2*x_v*x_a;
  std::cout << "dxdp=" << p_a << std::endl;
  // std::cout << "dxdx_0=" << x_a << std::endl; // vanishes

  return 0;
}
```

Embedded Nonlinear Equation: Hand-Written Algorithmic Adjoint (Cont'd)

```
template<typename T>
void newton_a(T& x_v, T& x_a, const T& p_v, T& p_a, const T& eps, bool a=false) {
  static std::stack<T> tbr_T; // to be recorded
  static int i=0; // iteration counter
  static T y=0; // persistent primal result
  if (!a) { // augmented primal
    do {
      tbr_T.push(x_v);
      x_v-=(x_v*x_v-p_v)/(2*x_v);
      i++;
    } while (fabs(x_v*x_v-p_v)>eps);
    y=x_v;
  } else { // adjoint
    for (int j=0;j<i;j++) {
      x_v=tbr_T.top(); tbr_T.pop();
      p_a+=x_a/(2*x_v);
      x_a-=(3./4.+p_v/(4*x_v*x_v))*x_a;
    }
    x_v=y;
  }
}
```

Embedded Nonlinear Equation: Hand-Written Symbolic Adjoint

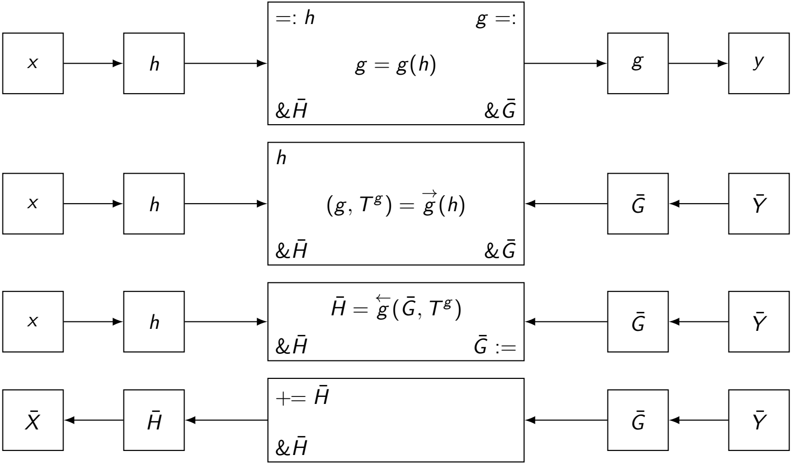
```
int main() {
  using T=double;
  T p_v=2, x_v=10, p_a=0, x_a=1;
  const T eps=1e−12;
  std::stack<T> tbr_T;
  tbr_T.push(x_v);
  x_v=pow(x_v,2);
  newton(x_v,p_v,eps);
  tbr_T.push(x_v);
  x_v=exp(−x_v);
  std::cout << "x=" << x_v << std::endl;
  x_a=−x_v*x_a;
  x_v=tbr_T.top(); tbr_T.pop();
  newton_a(x_v,x_a,p_v,p_a);
  x_v=tbr_T.top(); tbr_T.pop();
  x_a=2*x_v*x_a;
  std::cout << "dxdp=" << p_a << std::endl;
  // std::cout << "dxdx0=" << x_a << std::endl; // vanishes
  return 0;
}
```

Embedded Nonlinear Equation: Hand-Written Symbolic Adjoint (Cont'd)

```
template<typename T>
void newton(T& x, const T& p, const T& eps) {
  do { x=x-(x*x-p)/(2*x); } while (fabs(x*x-p)>eps);
}

template<typename T>
void newton_a(T &x_v, T &x_a, const T &p_v, T &p_a) {
  p_a+=x_a/(2*x_v);
  x_a=x_a-(2*x_v/(2*x_v)-(x_v*x_v-p_v)/(4*x_v*x_v))*x_a;
}
```

Recall:



Embedded Nonlinear Equation: dco/c++ Adjoint

```
template<typename T>
void dxdp_a(T& x_v, const T& p_v, const float& eps, T& dxdp) {
  using DCO_M=typename dco::ga1s<T>;
  using DCO_T=typename DCO_M::type;
  using DCO_TT=typename DCO_M::tape_t;
  DCO_M::global_tape=DCO_TT::create();
  DCO_T x=x_v,p=p_v;
  DCO_M::global_tape->register_variable(p);
  x=pow(x,2);
  // newton(x,p,eps); // algorithmic adjoint
  augmented_primal_newton<T>(x,p,eps); // symbolic adjoint
  x=exp(-x);
  x_v=dco::value(x);
  DCO_M::global_tape->register_output_variable(x);
  dco::derivative(x)=1;
  std::cerr << dco::size_of(DCO_M::global_tape) << "B" << std::endl;
  DCO_M::global_tape->interpret_adjoint();
  dxdp=dco::derivative(p);
  DCO_TT::remove(DCO_M::global_tape);
}
```

Embedded Nonlinear Equation: dco/c++ Adjoint (Cont'd)

```
template<typename T>
void adjoint_newton(typename dco::ga1s<T>::external_adjoint_object_t *D) {
  const T &x_v = D->template read_data<T>();
  T xa=D->get_output_adjoint();
  T pa=xa/(2*x_v);
  D->increment_input_adjoint(pa);
}

template<typename T>
void augmented_primal_newton(typename dco::ga1s<T>::type &x,
        const typename dco::ga1s<T>::type &p, const float& eps) {
  using DCO_M=typename dco::ga1s<T>;
  using DCO_EA=typename DCO_M::external_adjoint_object_t;
  DCO_EA *D=DCO_M::global_tape->template create_callback_object<DCO_EA>();
  T p_v=D->register_input(p);
  T x_v=dco::value(x);
  newton(x_v,p_v,eps);
  x=D->register_output(x_v);
  D->write_data(x_v);
  DCO_M::global_tape->template insert_callback<DCO_EA>(adjoint_newton<T>,D);
}
```

Systems of n linear equations

$$A \cdot \mathbf{x} = \mathbf{b}$$

with invertible $A \in \mathbb{R}^{n \times n}$ and right-hand side $\mathbf{b} \in \mathbb{R}^n$ define implicit functions $x = x(p)$ where $p = (A, \mathbf{b})$ as $A \cdot \mathbf{x} - \mathbf{b} = 0$.

Their adjoints can be evaluated at the primal solution $\mathbf{x} := A^{-1} \cdot \mathbf{b} \in \mathbb{R}^n$ as

$$\begin{aligned} \mathbf{b}_{(1)} &= A^{-T} \cdot \mathbf{x}_{(1)} \\ A_{(1)} &= -\mathbf{b}_{(1)} \cdot \mathbf{x}^T \end{aligned}$$

implying the incremental adjoint

$$\begin{aligned} \mathbf{v} &= A^{-T} \cdot \mathbf{x}_{(1)} \\ A_{(1)} &= -\mathbf{v} \cdot \mathbf{x}^T + A_{(1)} \\ \mathbf{b}_{(1)} &= \mathbf{v} + \mathbf{b}_{(1)} \ . \end{aligned}$$

1. Solve the primal system of linear equations $A \cdot x = b$; in case of a direct solver, store the factorization of the system matrix, e.g. $A = Q \cdot R$.

2. Solve the adjoint system linear equations (with transposed system matrix) for the given right-hand side $x_{(1)}$ yielding $b_{(1)}$, e.g.,

$$b_{(1)} = A^{-T} \cdot \mathbf{x}_{(1)} = (R^{-1} \cdot Q^{-1})^{T} \cdot \mathbf{x}_{(1)} = Q \cdot R^{-T} \cdot \mathbf{x}_{(1)}.$$

3. Compute the (rank-1) adjoint of the system matrix or keep it in low-memory storage format.

Let $x^* = x^*(p) = \min_x f(x(p), p)$ for twice continuously differentiable $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}$ wrt. both x and p yielding

$$f_x(x^*) = 0 \quad \text{and} \quad f_{xx}(x^*) > 0 \; .$$

From

$$f_{xp}(x^*) = \frac{\partial f_x}{\partial p}(x^*) + f_{xx}(x^*) \cdot x_p(x^*) = 0$$

follows

$$\underbrace{x_p}_{\in \mathbb{R}^{n_x \times n_p}} = - \underbrace{f_{xx}^{-1}}_{\in \mathbb{R}^{n_x \times n_x}} \cdot \underbrace{\frac{\partial f_x}{\partial p}}_{\in \mathbb{R}^{n_x \times n_p}}$$

implying the adjoint

$$p_{(1)} \equiv x_{(1)} \cdot x_p = \underbrace{-x_{(1)} \cdot f_{xx}^{-1}}_{f_{xx} \cdot z_{(1)} = -x_{(1)}} \cdot \frac{\partial f_x}{\partial p} \; .$$

Example

The objective function $f(x,p) = e^x - p \cdot x$ has a minimum at

$$\frac{df}{dx}(x,p) = e^x - p = 0 \qquad \left(\frac{d^2 f}{dx^2}(x,p) = e^x > 0 \ \forall x \in \mathbb{R}\right) \; .$$

$$\begin{tikzcd}
\boxed{R = R(x,p) \equiv e^x - p = 0} \arrow[r, "x_{(1)} \cdot \frac{d}{dp}"] \arrow[d, "{(x^*, T) = \overrightarrow{S}(R,p,x)}"'] & \boxed{x_{(1)} \cdot \left(e^x \cdot \frac{dx}{dp} - 1\right) = e^x \cdot \underbrace{\left(x_{(1)} \cdot \frac{dx}{dp}\right)}_{=p_{(1)}} - x_{(1)} = 0} \arrow[d, "{p_{(1)} = \tilde{S}(x, x_{(1)})}"] \\
\boxed{x^* \approx \log p} \arrow[r, "{\overleftarrow{S}(T, x_{(1)})}"'] & \boxed{= p_{(1)} = \frac{x_{(1)}}{e^{x^*}} \approx \frac{x_{(1)}}{p} \ ?}
\end{tikzcd}$$

1. Solve the primal system $R(x(p), p) = f_x(x, p) = 0$ to the required accuracy yielding an approximate optimizer $x^* = S(R, x, p)$, e.g. using dco/c++ in adjoint mode (dco::ga1s<T>::type) for the computation of the gradient f_x of the objective.
2. Compute the Jacobian of the residual wrt. the state (Hessian f_{xx} of the objective), e.g. using dco/c++ in second-order adjoint mode (dco::ga1s<dco::gt1s<T>::type>::type).
3. Solve the system of linear equations $f_{xx} \cdot z_{(1)} = -x_{(1)}$ for the given adjoint of the primal solution $x_{(1)}$ yielding $z_{(1)} \in \mathbb{R}^{1 \times n_x}$.
4. Evaluate the adjoint $p_{(1)} = z_{(1)} \cdot R_p = z_{(1)} \cdot \frac{\partial f_x}{\partial p}$.

Algorithmic differentiation of the iterative primal optimizer can be avoided at (or close to[6]) the primal solution.

[6] See below for first-order error analysis.

We consider the minimization of the squared solution of the SDE

$$dx = f_1(x(p_1(t), t), p_1(t), t))dt + f_2(x(p_2(t), t), p_2(t), t)dW$$

representing calibration to a vanishing target solution for the given initial condition.

A simple gradient descent method with "learning rate" 0.1 yields the primal solution. The scalar adjoint mode of dco/c++ with multiple tapes dco::ga1sm<T>::type is used to compute the gradient inside of the gradient descent algorithm as well as the adjoint of the solution with respect to the parameters.

The symbolic adjoint is computed as outline above. Both $\frac{\partial f_x}{\partial p}$ and f_{xx} are computed is second-order adjoint mode (dco::ga1sm<dco::ga1sm<T>::type>::type).

Our SDE scenario $f : \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}$ yields

$$f_{xp}(x^*) = \frac{\partial f_x}{\partial p}(x^*) + f_{xx}(x^*) \cdot x_p(x^*) = 0 \, .$$

From

$$\underbrace{x_p}_{\in \mathbb{R}^{1 \times n_p}} = - \underbrace{f_{xx}^{-1}}_{\in \mathbb{R}^{1 \times n_x}} \cdot \underbrace{\frac{\partial f_x}{\partial p}}_{\in \mathbb{R}^{1 \times n_p}}$$

it follows that

$$p_{(1)} \equiv x_{(1)} \cdot x_p = -x_{(1)} \cdot f_{xx}^{-1} \cdot \frac{\partial f_x}{\partial p}$$

and, hence,

$$x_p = -\frac{\frac{\partial f_x}{\partial p}}{f_{xx}} \, .$$

Live:

`SDE/optimization/primal/main.cpp` with `SDE/f12.h` in configuration BC

- inspect
- build (`Makefile`)
- run, e.g. `./main.exe 100 1 1e-15` yields

  ```
  |dy/dx|=7.7715611723761e-16
  x=-0.0147719904586304
  y=0.999779639540333
  ```
- time (`/usr/bin/time -v`)

Live:

`SDE/optimization/ad/main.cpp` with `SDE/f12.h` in configuration BC

- inspect
- build (`Makefile`)
- run, e.g. `./main.exe 100 1 1e-15` yields

  ```
  dx/dp[0][0]=1.01477199045858
  dx/dp[1][0]=-0.0146634198662722
  x=-0.0147719904586304
  ```
- time (`/usr/bin/time -v`)

Live:

`SDE/optimization/sd/main.cpp` with `SDE/f12.h` in configuration BC

- inspect
- build (`Makefile`)
- run, e.g. `./main.exe 100 1 1e-15` yields

```
dx/dp[0][0]=1.01477199045863
dx/dp[1][0]=-0.0146634198662721
x=-0.0147719904586305
```

- time (`/usr/bin/time -v`)

Algorithmic Differentiation

np	ns	*epsilon*	TIME (s)	MEM (mb)
250	10	10^{-2}	0.4	439
500	10	10^{-2}	0.9	920
750	10	10^{-2}	1.3	1,358
1000	10	10^{-2}	1.7	$1,788$
1000	25	10^{-2}	9.1	$9,276$
1000	50	10^{-2}	-	$> 15,284$

Symbolic Differentiation

np	ns	ϵ	TIME (s)	MEM (mb)
10^3	10	10^{-2}	< 0.1	10
10^4	10	10^{-2}	0.6	65
10^4	100	10^{-2}	16	613
10^4	200	10^{-2}	32	1,223
10^4	300	10^{-2}	51	1,832
10^5	100	10^{-2}	177	6,103

Let np=100 and ns=1.

Algorithmic Differentiation

ϵ	x_r	x_σ
$1e-15$	1.01477199045858	-0.0146634198662722
$1e-10$	1.01477198950385	-0.0146634198790826
$1e-5$	1.01471758699072	-0.0146641119012473
$1e-3$	1.0113982382379	-0.0147037598529871

Symbolic Differentiation

ϵ	x_r	x_σ
$1e-15$	1.01477199045863	-0.0146634198662721
$1e-10$	1.01477199046099	-0.0146634198662386
$1e-5$	1.01477200283279	-0.0146634196901854
$1e-3$	1.01477115263426	-0.0146634317886887

We consider twice differentiable implicit functions

$$F : \mathbb{R}^m \to \mathbb{R}^n : \mathbf{p} \mapsto \mathbf{x} = F(\mathbf{p})$$

defined by the roots of residuals $R : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n : (\mathbf{x}, \mathbf{p}) \mapsto R(\mathbf{x}, \mathbf{p})$. R is referred to as the primal residual as opposed to tangent and adjoint residuals to be considered later. Primal roots of the residual satisfying

$$R(\mathbf{x}, \mathbf{p}) = 0$$

are assumed to be approximated by numerical solvers

$$S : \mathbb{R}^m \to \mathbb{R}^n : \mathbf{p} \mapsto \mathbf{x} + \Delta\mathbf{x} = S(\mathbf{p})$$

with an absolute error $\Delta\mathbf{x}$ yielding relative error $\delta\mathbf{x}$ of norm

$$\|\delta\mathbf{x}\| = \frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|S(\mathbf{p}) - F(\mathbf{p})\|}{\|F(\mathbf{p})\|} \; .$$

$$
\begin{aligned}
& A \cdot \mathbf{x} = \mathbf{b} \\
& \Rightarrow \quad \dot{\mathbf{x}} = \dot{\mathbf{x}}_A + \dot{\mathbf{x}}_{\mathbf{b}}, \quad \text{where} \quad A \cdot \dot{\mathbf{x}}_{\mathbf{b}} = \dot{\mathbf{b}} \quad \text{and} \quad A \cdot \dot{\mathbf{x}}_A = -\dot{A} \cdot \mathbf{x} \\
& \Rightarrow \quad A^T \cdot \bar{\mathbf{b}} = \bar{\mathbf{x}} \quad \text{and} \quad \bar{A} = -\bar{\mathbf{b}} \cdot \mathbf{x}^T \ ;
\end{aligned}
$$

see e.g. [Giles]

Btw., the above follows immediately from

$$\dot{Y} = A \cdot \dot{X} \cdot B \quad \Leftrightarrow \quad \bar{X} = A^T \cdot \bar{Y} \cdot B^T \ .$$

From

$$\Delta \mathbf{x} \approx \frac{dF}{d\mathbf{p}} \cdot \Delta \mathbf{p}$$

and

- scalar multiplication is numerically stable
- scalar addition is not

it follows that

- $\|\delta \mathbf{x}\| \approx \kappa(A) \cdot (\|\delta A\| + \|\delta \mathbf{b}\|)$
- $\|\delta \dot{\mathbf{x}}_{\mathbf{b}}\| \approx \kappa(A) \cdot (\|\delta A\| + \|\delta \dot{\mathbf{b}}\|)$ and $\|\delta \dot{\mathbf{x}}_A\| \approx \kappa(A) \cdot \kappa(\dot{A}) \cdot \|\delta \mathbf{x}\|$
- $\delta \bar{\mathbf{b}} \approx \kappa(A) \cdot (\delta A + \delta \bar{\mathbf{x}})$ and $\bar{A} = -\bar{\mathbf{b}} \cdot \mathbf{x}^T$ (stable!)

where $\kappa(A) \equiv \|A^{-1}\| \cdot \|A\|$.

Same for Tangents and Adjoints of Nonlinear Systems / Convex Objectives

Nonlinear Systems $R(\mathbf{x}, \mathbf{p}) = 0$

$$\|\delta\dot{\mathbf{x}}\| \approx \kappa(R_{\mathbf{x}}) \cdot \kappa(\Delta\dot{R}_{\mathbf{x}} + \Delta\dot{R}_{\mathbf{p}}) \cdot \|\delta\mathbf{x}\|$$

$$\|\delta\bar{\mathbf{p}}\| \approx \left(\kappa(\Delta\bar{R}_{\mathbf{p}}) + \kappa(R_{\mathbf{p}}) \cdot \kappa(R_{\mathbf{x}}) \cdot \kappa(\Delta\bar{R}_{\mathbf{x}})\right) \cdot \|\delta\mathbf{x}\|$$

Nonlinear Convex Objectives $f(\mathbf{x}, \mathbf{p}) \to \min$

$$\|\delta\dot{\mathbf{x}}\| \approx \kappa(f_{\mathbf{x},\mathbf{x}}) \cdot \kappa(\Delta\dot{f}_{\mathbf{x},\mathbf{x}} + \Delta\dot{f}_{\mathbf{x},\mathbf{p}}) \cdot \|\delta\mathbf{x}\|$$

$$\|\delta\bar{\mathbf{p}}\| \approx \left(\kappa(\Delta\bar{f}_{\mathbf{x},\mathbf{p}}) + \kappa(f_{\mathbf{x},\mathbf{p}}) \cdot \kappa(f_{\mathbf{x},\mathbf{x}}) \cdot \kappa(\Delta\bar{f}_{\mathbf{x},\mathbf{x}})\right) \cdot \|\delta\mathbf{x}\|$$

more on [arXiv]

The NAG AD Library implements AD on a variety of algorithms from the NAG Library. It builds on the functionality of dco/c++ and delivers first and second order derivatives via tangent and adjoint mode AD. A subset of the supported AD algorithms has been optimized by using symbolic differentiation, which gives substantial savings in runtime and memory consumption. The NAG AD Library comes with C++ interfaces which allow seamless use with dco/c++.

See

`www.nag.com/numeric/nl/nagdoc_latest/adhtml/genint/adintro.html`