AD Master Class 2

How AD works

Viktor Mosenkis
viktor.mosenkis@nag.co.uk

6 August 2020

AD Masterclass Schedule and Remarks

AD Masterclass Schedule

- 30 July 2020 | Why the need for Algorithmic Differentiation?
- 6 August 2020 | How AD works
- 13 August 2020 | Testing and validation
- 20 August 2020 | Pushing performance using SIMD vectorization
- 27 August 2020 | Bootstrapping validated adjoints on real-world codes

Remarks

- Please submit your questions via the questions panel at any time during this session, these will be addressed at the end.
- A recording of this session, along with the slides will be shared with you in a day or two.

Dialogue

We want this webinar series to be interactive (even though it's hard to do)

- We want your feedback, we want to adapt material to your feedback
- Please feel free to contact us via email to ask questions at any time
- We'd love to reach out offline, discuss what's working, what to spend more time on
- For some orgs, may make sense for us to do a few bespoke sessions

Blog:
https://www.nag.com/blog/algorithmic-differentiation-masterclass-1

Outcomes

- Understand the basic idea behind the two AD models
  - tangent-linear and
  - adjoint model

- Learn how to apply dco/c++ to your code

- Learn how to write a correct driver with dco/c++ for
  - tangent-linear and
  - adjoint model

- Learn to decide which AD model should be used

- Understand the restrictions/problem that arise when using AD

Algorithmic Differentiation: Tangent-Linear Model

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

- **Tangent-Linear Model (TLM)** $\dot{F}$ **(forward mode)**
  $\dot{F}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \underbrace{F'(\boldsymbol{x})}_{\in \mathbf{R}^{m \times n}} \cdot \underbrace{\dot{\boldsymbol{x}}}_{\in \mathbf{R}^n} = \dot{\boldsymbol{y}}$

  - $F'(x)$ at $O(n) \cdot Cost(F)$
  - exact derivatives
  - $\frac{Cost(\dot{F})}{Cost(F)} \approx 2$

Inside TLM

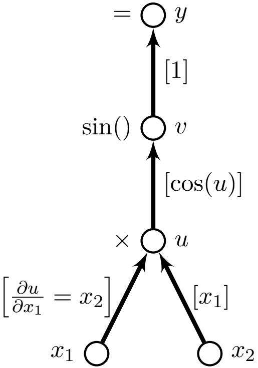
$$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$$

Single Assignment Code

$$u = x_1 \cdot x_2$$

$$v = \sin(u)$$

$$y = v$$



$$\dot{y} = F'(x) \cdot \dot{x} = \frac{\partial y}{\partial x} \cdot \dot{x} = \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial x} \cdot \dot{x}$$

Inside TLM

$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$

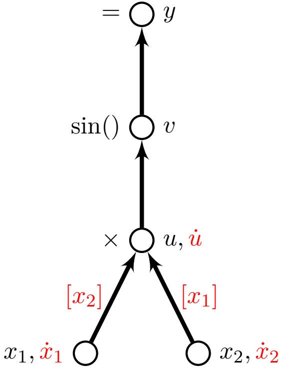
Tangent linear code

$\dot{u} = x_2 \cdot \dot{x}_1 + x_1 \cdot \dot{x}_2$

$u = x_1 \cdot x_2$

$v = sin(u)$

$y = v$



$$\dot{y} = F'(x) \cdot \dot{x} = \frac{\partial y}{\partial x} \cdot \dot{x} = \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \underbrace{\frac{\partial u}{\partial x} \cdot \dot{x}}_{=\dot{u}}$$

Inside TLM

$$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$$

Tangent linear code

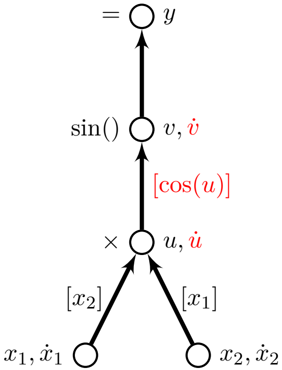
$\dot{u} = x_2 \cdot \dot{x}_1 + x_1 \cdot \dot{x}_2$

$u = x_1 \cdot x_2$

$\dot{v} = \cos(u) \cdot \dot{u}$

$v = \sin(u)$

$y = v$



$$\dot{y} = F'(x) \cdot \dot{x} = \frac{\partial y}{\partial x} \cdot \dot{x} = \frac{\partial y}{\partial v} \cdot \underbrace{\frac{\partial v}{\partial u} \cdot \dot{u}}_{=\dot{v}}$$

Inside TLM

$$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$$

Tangent linear code

$$\dot{u} = x_2 \cdot \dot{x}_1 + x_1 \cdot \dot{x}_2$$

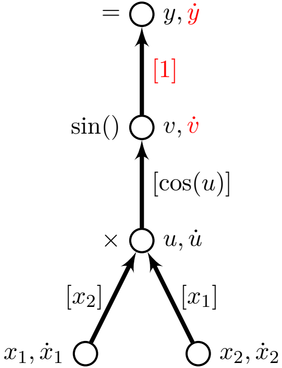
$$u = x_1 \cdot x_2$$

$$\dot{v} = \cos(u) \cdot \dot{u}$$

$$v = \sin(u)$$

$$\dot{y} = \dot{v}$$

$$y = v$$



$$\dot{y} = F'(x) \cdot \dot{x} = \frac{\partial y}{\partial x} \cdot \dot{x} = \underbrace{\frac{\partial y}{\partial v}}_{=\dot{y}} \cdot \dot{v}$$

Inside TLM

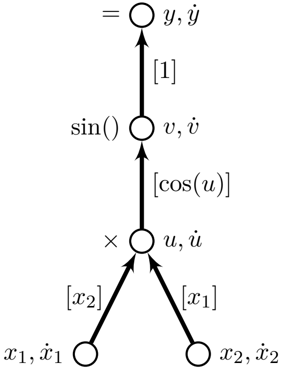
$$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$$

Tangent linear code

$$\dot{u} = x_2 \cdot \dot{x}_1 + x_1 \cdot \dot{x}_2$$
$$u = x_1 \cdot x_2$$

$$\dot{v} = \cos(u) \cdot \dot{u}$$
$$v = \sin(u)$$

$$\dot{y} = \dot{v}$$
$$y = v$$



Algorithmic Differentiation: Adjoint Model

$$F : \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad \boldsymbol{y} = F(\boldsymbol{x})$$

- **Adjoint Model (ADM) $\bar{F}$ (reverse mode)**

$$\bar{F}(\boldsymbol{x}, \bar{\boldsymbol{y}}) = \underbrace{\bar{\boldsymbol{y}}}_{\in \mathbf{R}^m} \cdot \underbrace{F'(\boldsymbol{x})}_{\in \mathbf{R}^{m \times n}} = F'(\boldsymbol{x})^T \cdot \bar{y} = \bar{\boldsymbol{x}}$$

  - ☐ $F'(x)$ at $O(m) \cdot Cost(F)$
  - ☐ exact derivatives
  - ☐ $\frac{Cost(\bar{F})}{Cost(F)} < 30$

Inside ADM

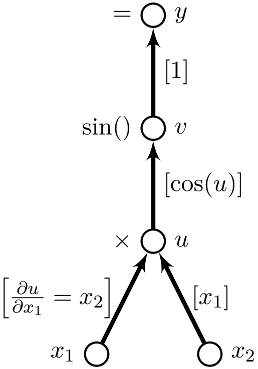
$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$

Single Assignment Code

$u = x_1 \cdot x_2$

$v = \sin(u)$

$y = v$



$\bar{x} = \bar{y} \cdot F'(x) = \bar{y} \cdot \frac{\partial y}{\partial x} = \bar{y} \cdot \frac{\partial y}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial x}$

Inside ADM

$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$

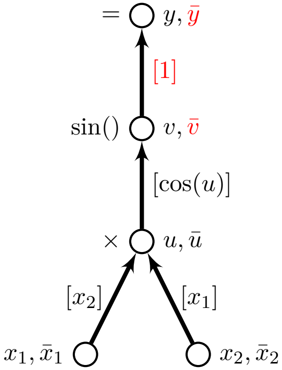
Adjoint Code

$u = x_1 \cdot x_2$

$v = \sin(u)$

$y = v$

$\bar{v} = \bar{y}$



$$\bar{x} = \bar{y} \cdot F'(x) = \bar{y} \cdot \frac{\partial y}{\partial x} = \underbrace{\bar{y} \cdot \frac{\partial y}{\partial v}}_{=\bar{v}} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Inside ADM

$$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$$

Adjoint Code

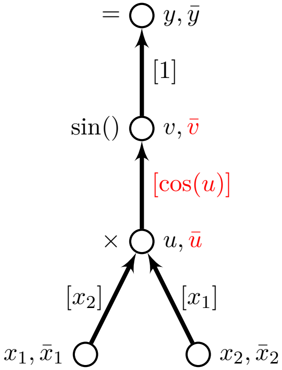
$u = x_1 \cdot x_2$

$v = \sin(u)$

$y = v$

$\bar{v} = \bar{y}$

$\bar{u} = \cos(u) \cdot \bar{v}$



$$\bar{x} = \bar{y} \cdot F'(x) = \bar{y} \cdot \frac{\partial y}{\partial x} = \underbrace{\bar{v} \cdot \frac{\partial v}{\partial u}}_{=\bar{u}} \cdot \frac{\partial u}{\partial x}$$

Inside ADM

$$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$$

Adjoint Code

$u = x_1 \cdot x_2$

$v = \sin(u)$

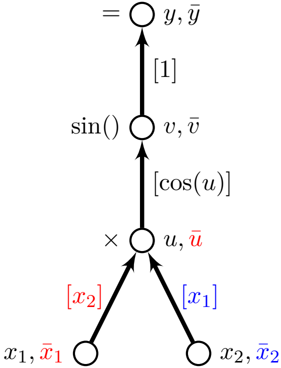
$y = v$

$\bar{v} = \bar{y}$

$\bar{u} = \cos(u) \cdot \bar{v}$

$\bar{x}_1 = x_2 \cdot \bar{u} \qquad \bar{x}_2 = x_1 \cdot \bar{u}$

$$\bar{x} = \bar{y} \cdot F'(x) = \bar{y} \cdot \frac{\partial y}{\partial x} = \underbrace{\bar{u} \cdot \frac{\partial u}{\partial x}}_{=\bar{x}}$$



Inside ADM

$F : \mathbb{R}^2 \to \mathbb{R}, \quad y = \sin(x_1 \cdot x_2)$

Adjoint Code

Forward Sweep (primal computation)

$u = x_1 \cdot x_2$

$v = \sin(u)$

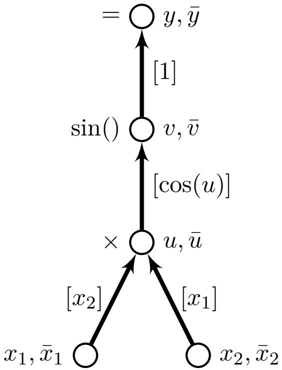
$y = v$

Reverse Sweep (adjoint computation)

$\bar{v} = \bar{y}$

$\bar{u} = \cos(u) \cdot \bar{v}$

$\bar{x}_1 = x_2 \cdot \bar{u} \qquad \bar{x}_2 = x_1 \cdot \bar{u}$



The control flow of the programm must be reversed, intermediate results are required for the reverse sweep!

AD with Source Transformation tool

Example: Live Coding TLM and ADM

We will now implement a tangent-linear and adjoint model for the following code, in a similar way a source transformation tool would do for the following function

```
void foo(int n, double *x, double &y){
  for (int i=0; i<n; i++){
    if (i == 0)
      y = sin(x[i]);
    else
      y *= x[i];
  }
}
```

Example: Reference TLM

```
void t1_foo(int n, double *x, double *t1_x, double &y,
      double &t1_y){
  for (int i=0; i<n; i++){
    if (i == 0) {
      t1_y = cos(x[i])*t1_x[i];
      y = sin(x[i]);
    }
    else {
      t1_y = x[i]*t1_y + y*t1_x[i];
      y *= x[i];
    }
  }
}
```

Example: Reference ADM

```
void a1_foo(int n, double *x, double *a1_x, double &y,
      double &a1_y){
  std::stack<double> ValStack;
  for (int i=0; i<n; i++){
    if (i == 0)  y = sin(x[i]);
    else {    ValStack.push(y);
      y = x[i]*y; }
  }
  //Reverse sweep
  for (int i = n-1; i>=0; i--){
    if (i == 0) a1_x[i] = cos(x[i])*a1_y;
    else {
      y = ValStack.top(); ValStack.pop();
      a1_x[i] = y*a1_y;     a1_y = x[i]*a1_y; }
  }
}
```

AD with Operator Overloading AD tool (dco/c++)

Tangent-linear Model with `dco/c++`

- Replace floating point variables with `dco::gt1s<double>::type`
- Write the driver

- Conceptually `dco::gt1s<double>::type` contains two components, both components are computed during normal execution
  - value
  - tangent
- Interface
  - `dco::value(DCO_TYPE)` - access to value component
  - `dco::derivative(DCO_TYPE)` - access to the tangent component

Tangent-Linear Model: Jacobian with dco/c++

```
template <typename T>
void foo(int& n, T* x, T& y){
  ...
}
int main(){
  DCO_TANGENT_TYPE *x, y;
  ...
  for (int i=0; i<n; i++) {
    dco::derivative(x[i]) = 1.0;
    foo(n, x, y);
    J[i] = dco::derivative(y);
    dco::derivative(x[i]) = 0.0;
  }
}
```

The tangent-linear model of the function `foo` is executed n times

Adjoint code with `dco/c++`: Concept

- Replace floating point variables with `dco::ga1s<double>::type`
- Write the driver

- Conceptually `dco::ga1s<double>::type` contains two components
  - value
  - adjoint

  During the execution of the function `dco/c++` computes the value component and records the computational graph (tape). Interpretation of the tape is needed to compute the adjoint components.

Adjoint code with dco/c++: Basic Interface

- Interface of `dco::ga1s<double>::type`
  - `dco::value(DCO_TYPE)` - access to value component
  - `dco::derivative(DCO_TYPE)` - access to the adjoint component

- Interface of the tape `DCO_MODE::tape_t`
  - `DCO_TAPE_TYPE::create()` - creates tape and returns pointer to it
  - `DCO_TAPE_TYPE::remove(DCO_TAPE_TYPE*)` - deallocates tape
  - `DCO_TAPE_TYPE::register_variable(DCO_TYPE)` - Marks variable as independent
  - `DCO_TAPE_TYPE::register_output_variable(DCO_TYPE)` - Marks variable as dependent
  - `DCO_TAPE_TYPE::interpret_adjoint()` - Runs tape interpreter

Adjoint Model: Jacobian with dco/c++

```
template <typename T>
void foo(int& n, T* x, T& y){ ... }

int main(){
  DCO_ADJOINT_TYPE *x, y;
  ...
  for (int i=0; i<n; i++)
    DCO_MODE::global_tape->register_variable(x[i]);
  foo(n, x, y);
  DCO_MODE::global_tape->register_output_variable(y);
  dco::derivative(y)=1.0;
  DCO_MODE::global_tape->interpret_adjoint();

  for (int i=0; i<n; i++){J[i] = dco::derivative(x[i]);
 }
```

The adjoint model of the function `foo` is executed only once

Types of AD Tools

- Source transformation (Compile time)
  - \+ efficient adjoint code
  - \+ small memory footprint (code optimization, store only required information)
  - \- does not support full language like C++ or Fortran > F90
  - \- significant changes to the primal code required (high development costs)
  - \- maintaining two source codes
- Operator overloading (Run time)
  - \+ support full language like (C++ or Fortran)
  - \+ almost no changes to the primal code are needed
  - \+ only one source code to maintain
  - \- less efficient adjoint code
  - \- higher memory requirements

Modern operator overloading AD tools such as (dco/c++) are trying to close the gap by using template meta programming!

AD has restrictions

- Requires the knowledge of full source code. This problem can be resolved (vendor provides AD version of the code, symbolic adjoints, FD).
- AD differentiates the executed code not the underlying mathematical function. E.g. $y = \begin{cases} 3 \cdot x & x = 0 \\ 2 \cdot x & x \neq 0 \end{cases}$ implements the mathematical function $x \mapsto 2 \cdot x$. But AD will compute wrong result for $x = 0$.
- If your function is not differentiable you will get subgradients. E.g. $y = |x|$.
- sometimes partial derivatives of the language intrinsic are `NAN` or `inf`, although your primal computation is computing reasonable numbers. E.g. differentiation of $\sqrt{x}$ at $x = 0$.
- no smoothing for oscillating function as with FD.

Restrictions of the Adjoint Model

- Adjoint model assumes availability of a sufficient amount of memory to store the variables that are required for the data flow reversal (e.g., the tape). Can be resolved by
  - implementing a checkpointing scheme
  - use symbolic adjoint
  - handwritten adjoint code for numerical kernels
  - using disk drive
- Parallelization scheme must be reversed for tape interpretation step. Can be addressed E.g.
  - Adjoint MPI (AMPI) for MPI programms
  - Pathwise tape interpretation for Monte Carlo codes.
- Requires development time and good tool support to achieve good adjoint factors.

Tangents vs Adjoints

or when should I use tangent-linear and when the adjoint model?

The short answer is: If $n > m$ you should use adjoint model in all other cases the tangent-linear model.

But wait what about the adjoint factor $M = \frac{Cost(\bar{F})}{Cost(F)}$. So let's do the math.

- Jacobian with tangent-linear model $Cost(\dot{F}) = 2 \cdot n \cdot Cost(F)$
- Jacobian with the adjoint model $Cost(\bar{F}) = M \cdot m \cdot Cost(F)$
- Your speedup with adjoint model is $\frac{Cost(\dot{F})}{Cost(\bar{F})} = \frac{2 \cdot n \cdot Cost(F)}{M \cdot m \cdot Cost(F)} = \frac{2 \cdot n}{M \cdot m}$.

  So if $M \cdot m < 2 \cdot n$ you should use the adjoint model and else the tangent-linear model.

Tangents vs Adjoints

The adjoint factor M and $\frac{n}{m}$ decide whether you should use the adjoint model or the tangent-linear model to compute the Jacobian. M depends

- on the quality of your AD tool
- your problem
- amount of effort you invest to improve your adjoint model
  - exploitation of structure (e.g. Monte Carlo)
  - symbolic adjoints
  - hand writing numerical kernels

For small $\frac{n}{m}$ you should consider tangent-linear model as an option

- no problems with memory
- parallelization from the primal code can be reused
- computing several tangent simultaneously can speed up the computation significantly (Class 4)

Tangents vs Adjoints

There is much more to say on this topic, and we will have a more detailed discussion in Class 4.

Summary

In this Class we learned

- How AD works
- Type of AD tools and there advantages and disadvantages
  - source transformation
  - operator overloading
- How to write AD code with operator overloading tool
- Caveats that arise with the usage of AD
- How to choose the right model for your problem.

AD Master Class 3: Testing and validation

In the next part we will

- learn why is it hard to test AD codes
- learn how to incorporate AD testing into your test harness
- highlight common problems and pitfalls
- demonstrate software engineering implications for
  - code maintenance
  - build systems
- share available options and best practice for code structure

You will see a survey on your screen after exiting from this session.

We would appreciate your feedback.

We are now moving on the Q&A Session