Source-to-source automatic differentiation using PSyclone

A prototype implementation

AIRSEA team meeting
September 5, 2023

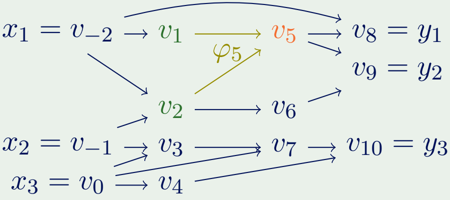
Julien REMY (M1 student - Martin's trainee)
julien.remy@grenoble-inp.org

Source code: https://github.com/JulienRemy/PSyclone
Documentation: https://psyclone-autodiff.readthedocs.io/ (not complete yet)

Automatic differentation

Program

Suppose we have a program computing return values of variables y_j from values of arguments x_i through intermediate values v_k, where each value is obtained from its direct predecessors through *elemental* operations $(+, \times, /, \exp, \text{etc.})$.



Notations

Relation: $i \prec j$ if v_j depends on v_i, *eg.* $1 \prec 5$.

Predecessors: $u_j := (v_i)_{i \prec j}$ *eg.* $u_5 = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$

Operation: $\varphi_j : u_j \mapsto v_j$ *eg.* $v_5 = \varphi_5(u_5)$

AD: Obtain $\dfrac{\partial y_j}{\partial x_i}(x_1, \dots, x_n)$ automatically, by differentiating operations $\varphi_k : u_k \mapsto v_k$.

Forward mode Tangent

Derivatives

For **one *independent variable*** z chosen among the **argument** variables, say $z = x_1$,

let $$\dot{v}_i = \frac{\partial v_i}{\partial z}$$

$$\overleftarrow{\dot{v}_{k+2} = \frac{\partial v_{k+2}}{\partial v_{k+1}} \underbrace{\left(\frac{\partial v_{k+1}}{\partial v_k} \dot{v}_k \right)}_{\dot{v}_{k+1}}}$$

The chain rule gives

Easy to implement.

$$\dot{v}_j = \sum_{i \prec j} \frac{\partial \varphi_j}{\partial v_i}(u_j) \dot{v}_i$$

Initialize $\dot{z} = 1$,
and $\dot{v}_k = 0, \forall v_k \neq z$,
get $\left(\frac{\partial y_j}{\partial z}(x_1, \dots x_n) \right)_j$

Issue

We obtain $J((x_i)_i)(0 \dots \dot{z} \dots 0)^T$: all the derivatives $(\dot{y}_j)_j$ wrt a **single** $z \in (x_i)_i$.
Interesting if $\#\{x_i\}_i \ll \#\{y_j\}_j$ but usually there are many arguments, few results.

Reverse mode Adjoint

Adjoints

For **one *dependent variable*** z chosen among the **return** variables, say $z = y_1$, denote

$$\bar{v}_i = \frac{\partial z}{\partial v_i}$$

$$\overrightarrow{\underbrace{\left(\bar{v}_k \frac{\partial v_k}{\partial v_{k-1}}\right)}_{\bar{v}_{k-1}} \frac{\partial v_{k-1}}{\partial v_{k-2}}} = \bar{v}_{k-2}$$

The chain rule gives

In practice, increment.

$$\bar{v}_i = \sum_{j \succ i} \bar{v}_j \frac{\partial \varphi_j}{\partial v_i}(\overset{?}{u_j})$$

Initialize $\bar{z} = 1$,
and $\bar{v}_k = 0, \forall v_k \neq z$,
get $\left(\frac{\partial z}{\partial x_i}(x_1, \ldots, x_n)\right)_i$

Remark

We obtain the adjoints $(\bar{x}_i)_i$ of **all arguments** x_i for a single differentiated $z \in (y_j)_j$.
Compute $\nabla^T z(x_1, \ldots, x_n) = (0 \ldots \bar{z} \ldots 0) J(x_1, \ldots, x_n)$ in one evaluation!

Reverse mode AD A simple math example, with non linearities

x_1

3

x_2

$v_1 = x_1^2$

$v_2 = 3 * x_1$

$v_3 = \exp(x_1^2)$

$v_4 = 3 * x_1 + x_2$

$y_1 = \exp(x_1^2) - 3 * x_1 + x_2$

$y_2 = x_2 * (3 * x_1 + x_2)$

Initialize with $\forall i, \bar{x}_i = 0, \quad \forall k, \bar{v}_k = 0$ and choose $(\bar{y}_1 = 1, \bar{y}_2 = 0)$ **or** $(\bar{y}_1 = 0, \bar{y}_2 = 1)$

$\bar{y}_1$

$\bar{y}_2$

$\bar{v}_3 += \bar{y}_1 * 1$

$\bar{v}_4 += \bar{y}_1 * (-1)$

$\bar{v}_4 += \bar{y}_2 * x_2$

$\bar{v}_1 += \bar{v}_3 * v_3$

$\bar{v}_2 += \bar{v}_4 * 1$

$\bar{x}_1 += \bar{v}_1 * 2 * x_1$

$\bar{x}_1 += \bar{v}_2 * 3$

$\bar{x}_2 += \bar{v}_4 * 1$

$\bar{x}_2 += \bar{y}_2 * v_4$

What about programs? First issue

Overwrites: taping

Recording routine

```
b = sin(a)
tape(i) = a
a = x + 1 ! overwrite a
c = cos(a)
```

Returning routine

```
! here a == x + 1
a_adj = a_adj + c_adj * (-sin(a));        c_adj = 0.0
x_adj = x_adj + a_adj * 1;                a_adj = 0.0
a = tape(i)
a_adj = a_adj + b_adj * cos(a);           b_adj = 0.0
```

Taping

Record and restore the values of overwritten variables to a "tape".
Usually a LIFO stack. But then we can't parallelize. Instead, I'm using (static) arrays.
Other possibility: recompute values, optional checkpointing.

Taping results of operations

Can also be used for the results of composed operations. Not implemented yet.
PSyclone: Operation nodes have no datatypes for now, especially shapes (for vectors).

What about programs? Second issue

Iterative assignments

Consider the following Fortran code and its naive and **wrong** reverse-mode AD:

```
a = 2 * a + x                a_adj = a_adj + a_adj * 2 ! wrong!
! now reverse                x_adj = x_adj + a_adj * 1 ! wrong too!
```

$\bar{a} \not\to 3\bar{a}$

Solution: deal with the LHS adjoint last

```
a = 2 * a + x                x_adj = x_adj + a_adj * 1 ! correct
! now reverse                a_adj = a_adj * 2 ! correct
```

Reversal schedules 1/2

Nested calls

Suppose we want to differentiate the subroutine foo, which calls bar and qux.

```
subroutine foo(x, y)
   call bar(x, y)
   call qux(x, y)
end subroutine foo
```

Reversal schedules

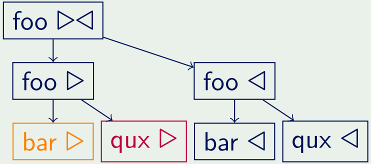
Let us call and denote :

- advancing $\square$ routine the original,
- recording $\triangleright$ routine the one recording values to the tape,
- returning $\triangleleft$ routine the one computing the adjoints,
- reversing $\triangleright\triangleleft$ routine the one combining the two preceding. Call it to differentiate.

Reversal schedules 2/2

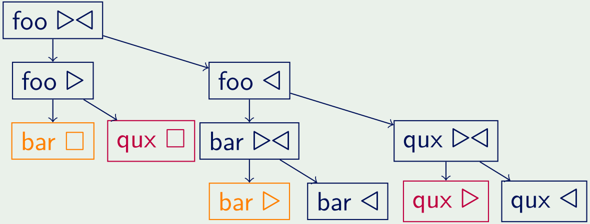
Split reversal

Larger tape in memory, but computations are only run once.



Joint reversal

Smaller tape but computations are repeated.



"Link" reversal

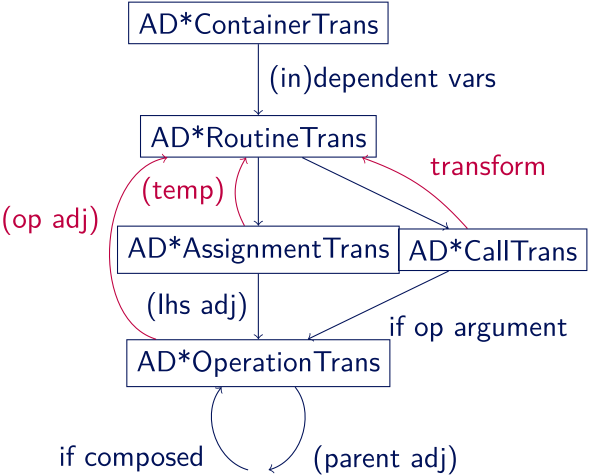
Specify whether to use a split or joint reversal for each (caller $\rightarrow$ called) pair.

AD prototype module in PSyclone - psyclone.autodiff

Forward- and reverse-mode:

- scalars only,
- subroutines only (no functions),
- calls to subroutines,
- reverse: joint, split and "link" reversals,
- adjoints and derivatives of fixed type and precision,
- quite naive implementation:
  - operations are always recomputed, differentiating $v = 1/u$ or $v = \exp(u)$ doesn't use the postvalue of v.
- simplification and substitution as a post-processing step.

Transformations: ($* \in \{$Forward, Reverse$\}$)



Test suite and planned features

Existing tests

- Unit tests for (most) elemental transformations.
- Transformations outputs tested by comparing to Tapenade (to be automated)
  - correct up to associativity/commutativity floating point errors in both modes.

Planned features

- Vectors and matrices,
- Activity ($\leftrightarrow$ dependence DAG) and TBR (to-be-recorded) analyses,
- Loops: need to be run backward in reverse-mode:
  - except if trivial: record the loop indices, run-time length of the tape
- Control flow: record the boolean values of conditions? Tag the branches?
- Functions? Programs?

Thank you for attending!

Questions? Remarks? Ideas?