Know what you are differentiating

SIAM CSE 2021

Know what you are differentiating

AD tools, like compilers, do not *know* your code.
$\Rightarrow$ therefore they apply general methods

The AD tool may fail on a code that hides its maths (or on non-smooth math)

Knowing your code, the tool may apply a better method, ... sometimes the only one that works.

The **A** in **AD** could stand for “Assisted”
$\Rightarrow$ *You* assist the tool!

Codes that hide their maths

Implementation/discretization may hide mathematical meaning e.g.

- solving $f(x, y) = 0$ by bisection
- discrete special cases: `(a==1 ? b : a*b)`

Code not "chainrule-differentiable" needs rewriting.

Code can also be non-smooth:

```
rainfall = MAX(rainfall, 0)
```

Useful AD on this is an active research field, with little tool support.

When assistance is welcome

An AD tool cannot detect in general,
but can do a good job when indicated.
Examples:

- Black boxes
- Linear solvers
- Independent iterations
- Fixed-point iterations
- ...

Black boxes

Compiled libraries, lookup tables
$\rightarrow$ no source, or source not differentiable

- replace black box with a dummy function
- differentiate code around black-box
- differentiate black-box mathematically
- replace AD-differentiated dummy with hand-written diff black-box

In other words, Differentiate-then-Discretize

Loops with Independent Iterations

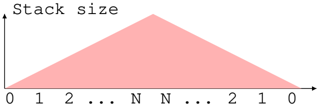
```
for (i=0 ; i<=N ; ++i) {
  iteration i
}
```

Suppose no loop-carried dependency
(except possible sum-reduction)

Loops with Independent Iterations

Plain reverse AD builds:

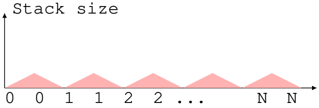
```
for (i=0 ; i<=N ; ++i) {
  forward sweep of iteration i
}
for (i=N ; i>=0 ; --i) {
  backward sweep of iteration i
}
```



Loop #2 also has independent iterations

Reverse iterations, fuse with loop #1:

```
for (i=0 ; i<=N ; ++i) {
  forward sweep of iteration i
  backward sweep of iteration i
}
```



$\Rightarrow$ Reduces peak stack size dramatically!

Linear Solvers (forward mode)

A perfect black box example:
either no source or poorly differentiable.

Go back to math instead:

$$Ax = b$$

Tangent AD:

$$\dot{A}x + A\dot{x} = \dot{b}$$

$$A\dot{x} = \dot{b} - \dot{A}x$$

Linear Solvers (forward mode)

Hand-written SOLVE_D:

```
SOLVE_D(A,Ad,x,xd,b,bd) {
  SOLVE(A,x,b)
  bdcopy = bd
  DGEMV(-1,Ad,x,1,bdcopy)
  SOLVE(A,xd,bdcopy)
}
```

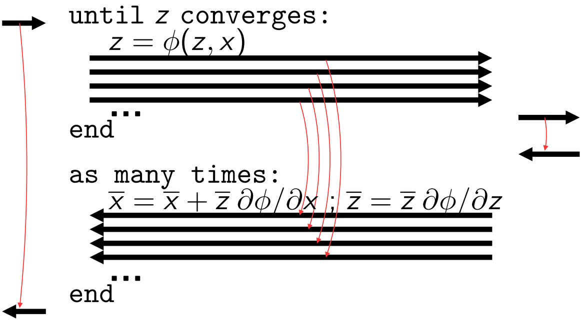
Linear Solvers (reverse mode)

(Similarly, with a little more effort. . .)

Hand-written SOLVE_B:

```
SOLVE_B(A,Ab,x,xb,b,bb) {
  At = TRANSPOSE(A)
  SOLVE(At,tmp,xb)
  bb[:]  = bb[:]  + tmp[:]
  SOLVE(A,x,b)
  for each i and each j {
    Ab[i,j] = Ab[i,j] - x[j]*tmp[i]
  }
  xb[:]  = 0.0
}
```

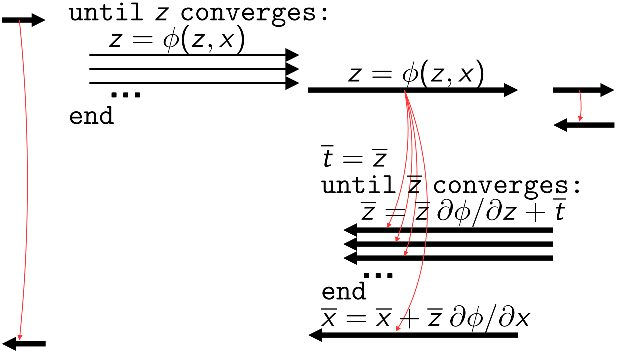
Fixed point iterations (reverse mode)



You should not do that!

- all values from intermediate iterations are stored
- poor convergence guarantees of the adjoint sweep

Fixed point Two-Phases Adjoint



- Only the converged primal iteration is stored, then is used several times.
- The adjoint iteration has its own convergence control
- Converges in one step if primal has quadratic convergence

Thank you