AD semantics, pitfalls, and the level of abstraction

Jan Hückelheim

jhueckelheim@anl.gov

Overview

- This is work in progress
- Content from an upcoming review paper on AD pitfalls
- Based on discussions with William Moses, Harshitha Menon, Paul Hovland, Bruce Christianson, Laurent Hascoët

How did this start?

- We know: ”AD differentiates programs. Unlike symbolic differentiation, it can handle large computations with loops and branches. Unlike FD, it is accurate.”

How did this start?

- We know: ”AD differentiates programs. Unlike symbolic differentiation, it can handle large computations with loops and branches. Unlike FD, it is accurate.”
- Except if your code has parametric integrals.

How did this start?

- We know: ”AD differentiates programs. Unlike symbolic differentiation, it can handle large computations with loops and branches. Unlike FD, it is accurate.”
- Except if your code has parametric integrals.
- Or linear solvers.

How did this start?

- We know: ”AD differentiates programs. Unlike symbolic differentiation, it can handle large computations with loops and branches. Unlike FD, it is accurate.”
- Except if your code has parametric integrals.
- Or linear solvers.
- Or fixed point iterations.

How did this start?

- We know: ”AD differentiates programs. Unlike symbolic differentiation, it can handle large computations with loops and branches. Unlike FD, it is accurate.”
- Except if your code has parametric integrals.
- Or linear solvers.
- Or fixed point iterations.
- Or Monte Carlo sampling.

What is going on here?

- Does AD just break randomly for some programs?
- Can we make sense of the failure modes?

Before we talk about pitfalls and AD problems…

Let us talk about what AD is supposed to compute.

"Frequently we have a program that calculates numerical values for a function, and we would like to obtain accurate values for derivatives of the function as well."[1]

[1] Griewank A, Walther A. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Second ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics; 2008.

Before we talk about pitfalls and AD problems…

Let us talk about what AD is supposed to compute.

Wishful thinking, or state of the art?

“Frequently we have a program that calculates numerical values for a function, and we would like to obtain accurate values for derivatives of the function as well.”[1]

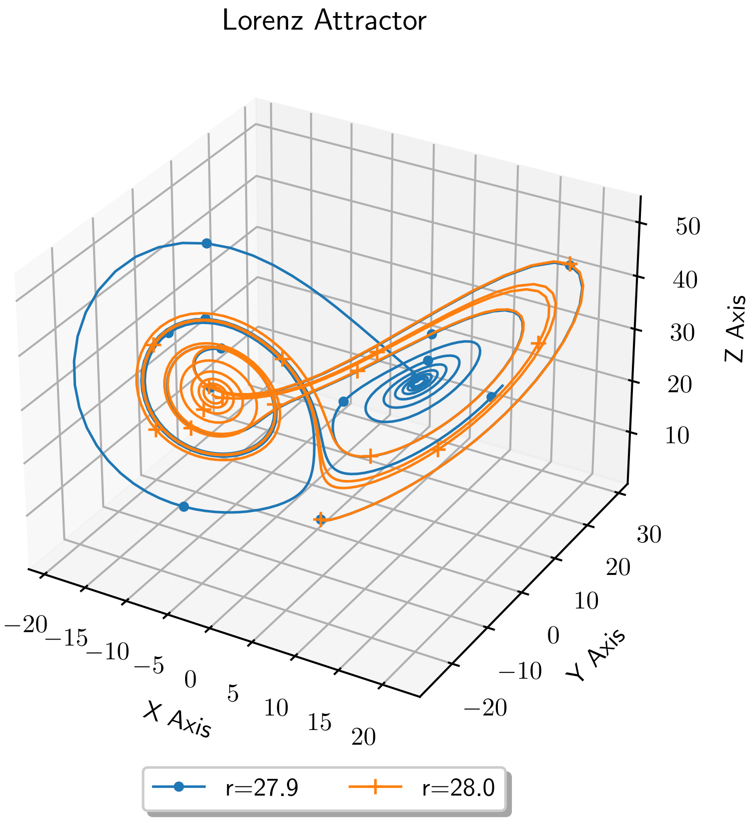
[1] Griewank A, Walther A. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Second ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics; 2008.

Pitfall 1

- We will categorize our problems into “pitfalls”.
- First pitfall:

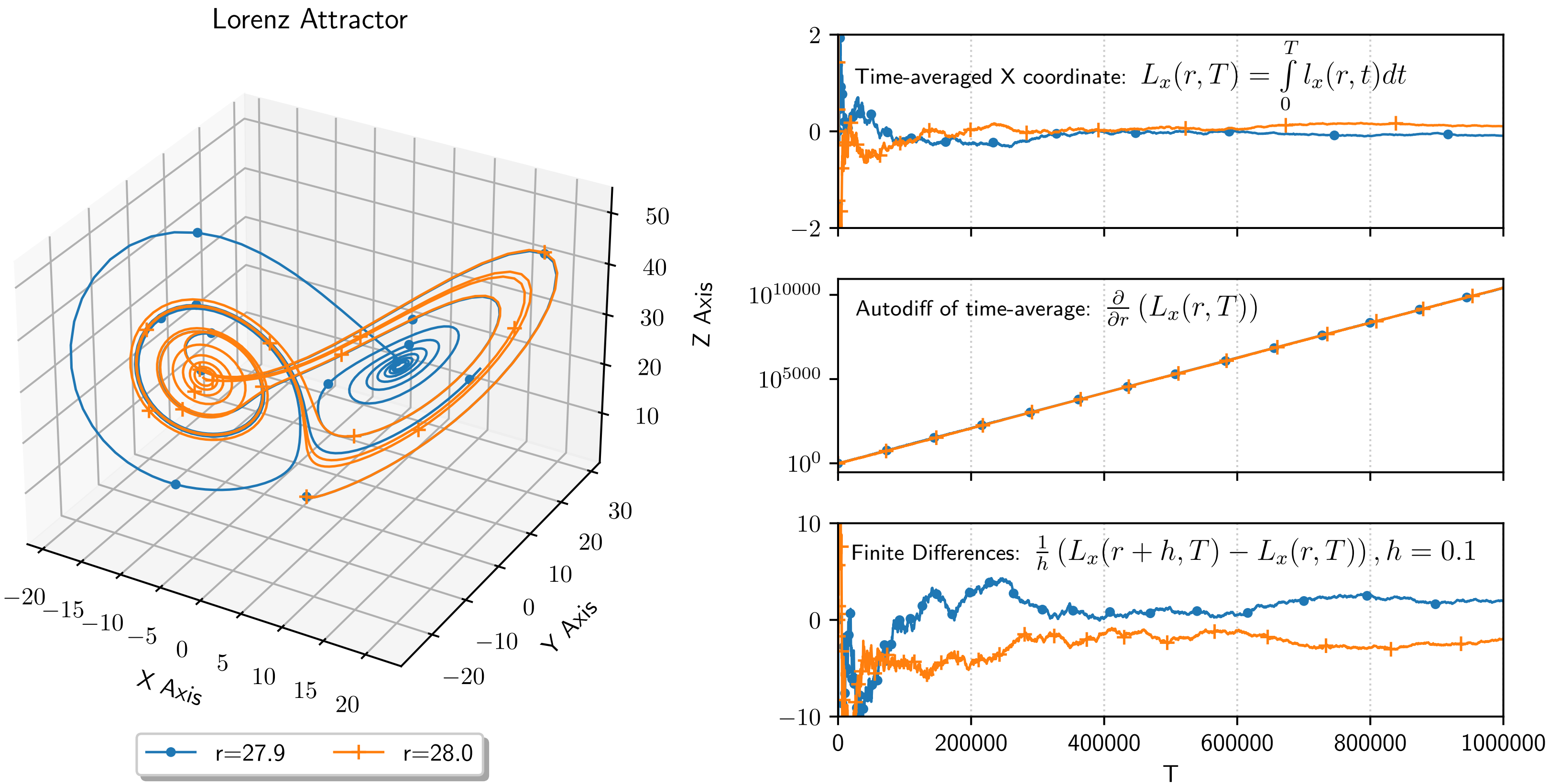
*Do you actually want the derivatives of **that** function?*

Known problem: Chaos

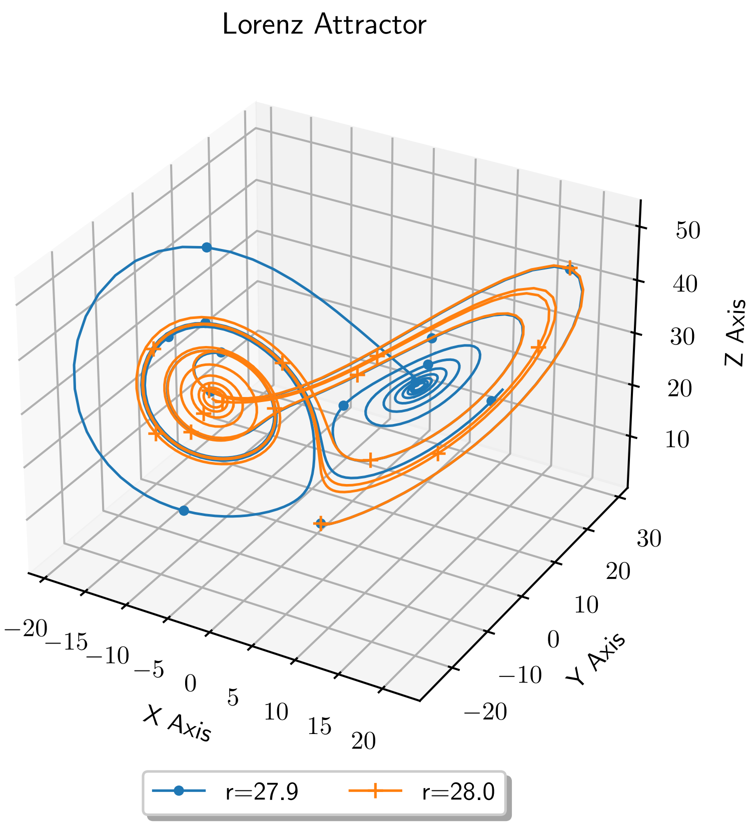


- Of course chaotic functions have crazy derivatives
- But what about their time-averaged behavior?

Known problem: Chaos



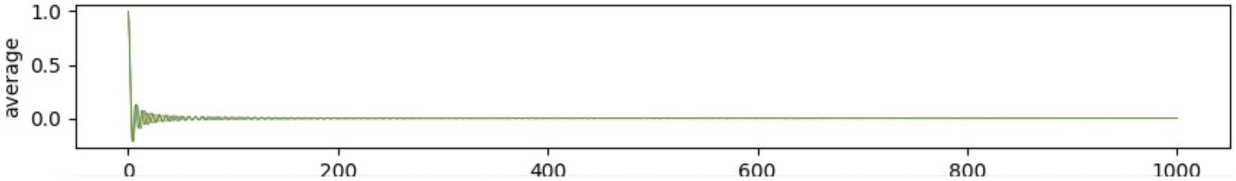
Known problem: Chaos



- Are you sure that your PDE does not have chaotic effects that you are averaging over?
- What happens if you increase the resolution?
- What happens if you switch from RANS to LES?

Another (less known?) problem: Oscillations

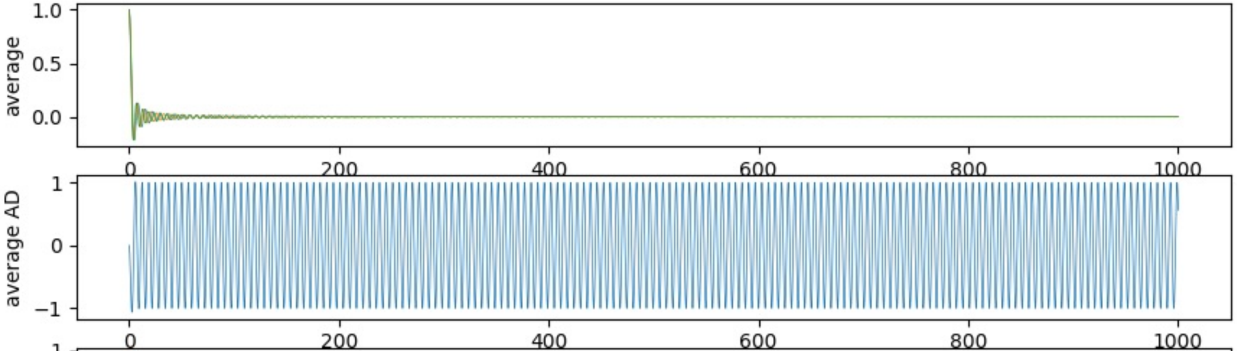
- We don’t need chaos to get bad derivatives. Just oscillations.
- What is the average of a cosine function over time?



- What if we differentiate the time average wrt. frequency?
- We expect convergence to zero!

Another (less known?) problem: Oscillations

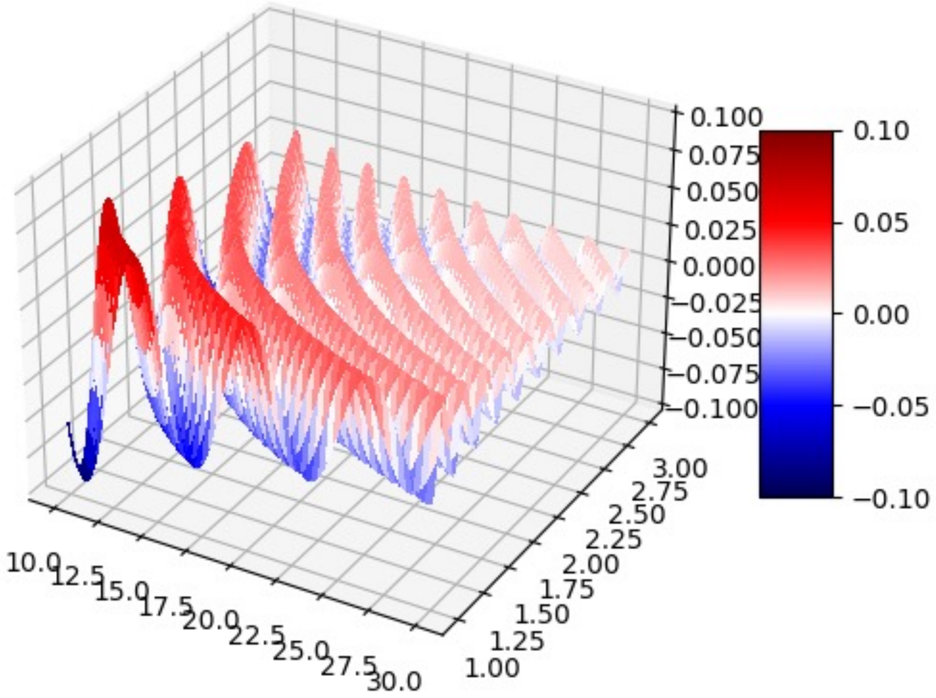
- We don't need chaos to get bad derivatives. Just oscillations.
- What is the average of a cosine function over time?



- We expect convergence to zero!
- But we don't get convergence.

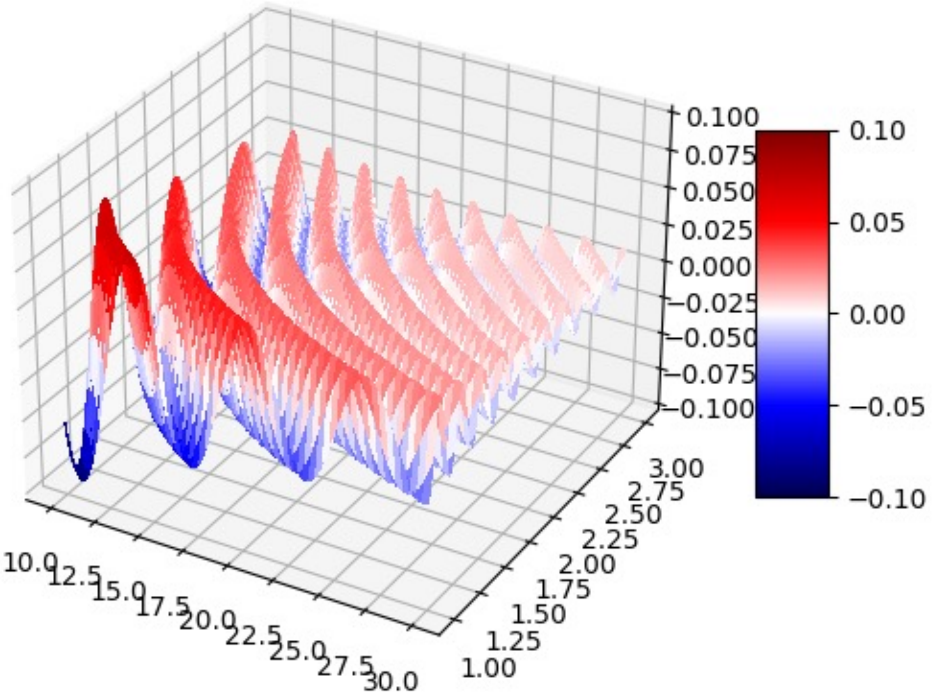
Why does the derivative not converge?

- The amplitude decreases over time
- But the oscillations wrt. frequency get faster
- The two effects balance each other out, derivatives stay constant



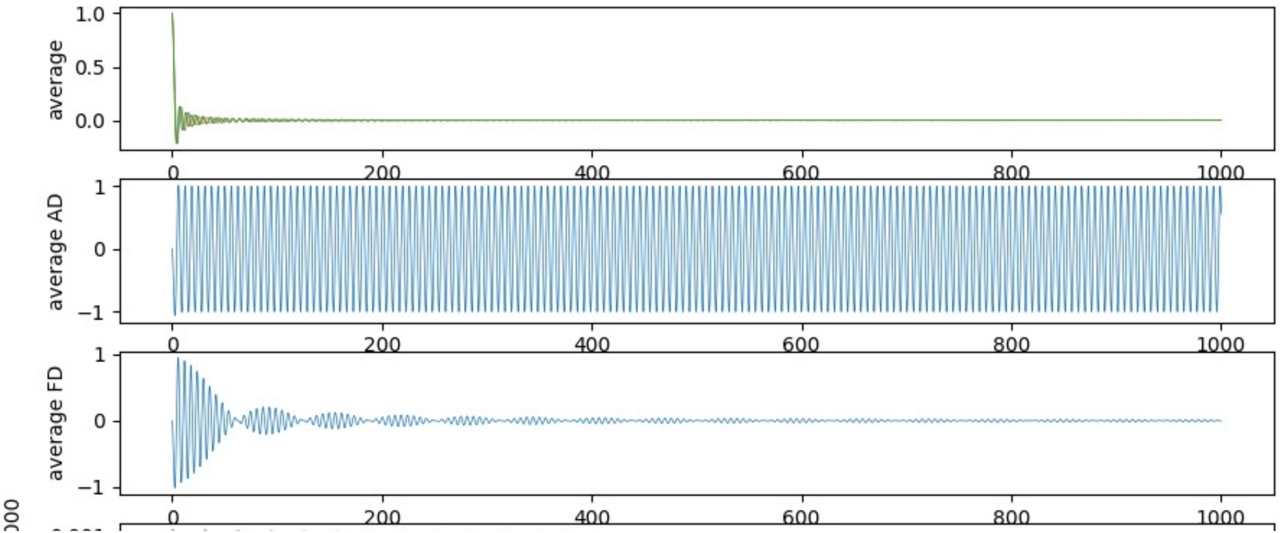
Why does the derivative not converge?

- The amplitude decreases over time
- But the oscillations wrt. frequency get faster
- The two effects balance each other out, derivatives stay constant
- This example is contrived, but what about PDE solvers and time-averaged cost functions?



Annoyingly: FD gets it right!

- As the oscillations get faster, they fall below FD resolution eventually.



Pitfall 2

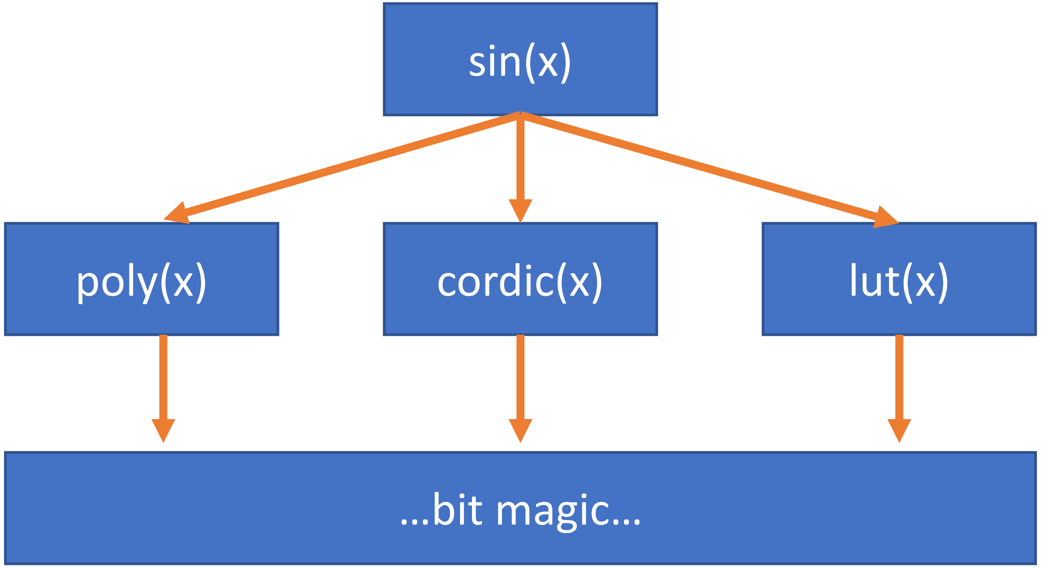
*Do you actually want to differentiate at **that** abstraction level?*

Pitfall 2

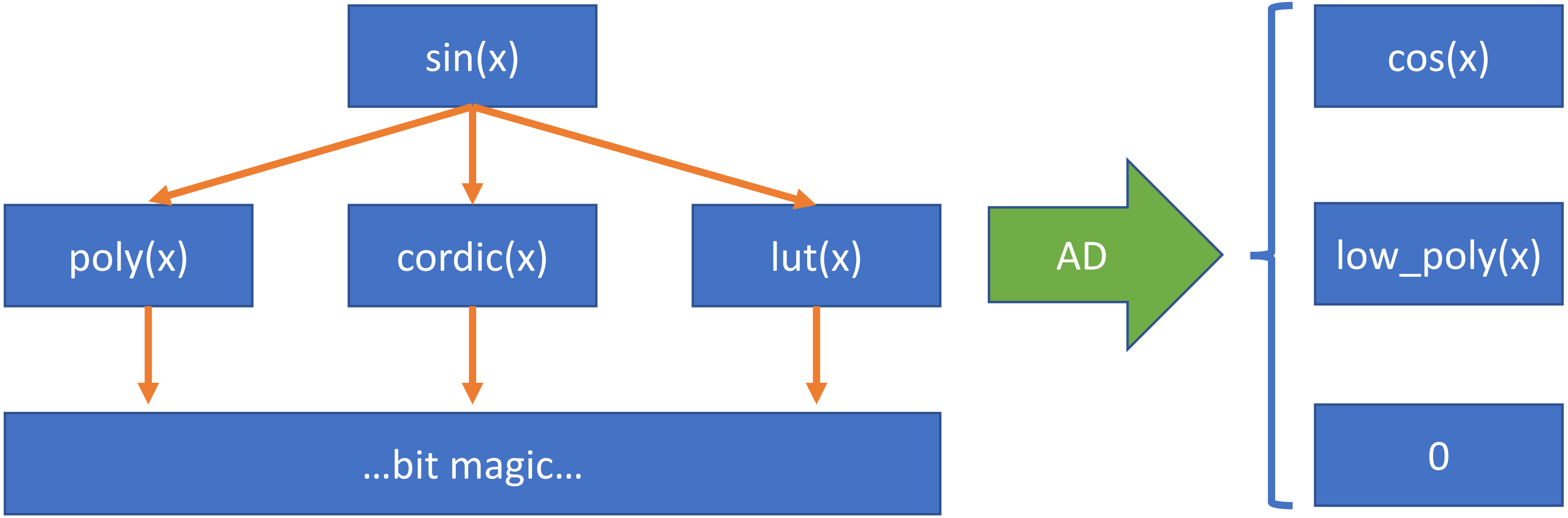
"AD differentiates what you implement![...] Which occasionally differs from what you think you implement!"[2]

[2] Naumann U. The art of differentiating computer programs: an introduction to algorithmic differentiation, vol. 24. SIAM;
2012.

What does AD really differentiate?



What does AD really differentiate?



So what does AD compute?

- *“AD differentiates what is implemented at the abstraction to which you apply AD. Which occasionally differs from what you think you implement!”*

So what does AD compute?

- *"AD differentiates what is implemented at the abstraction to which you apply AD. Which occasionally differs from what you think you implement!"*

Note: abstraction level != language. For example, multiple levels within C or within MLIR.

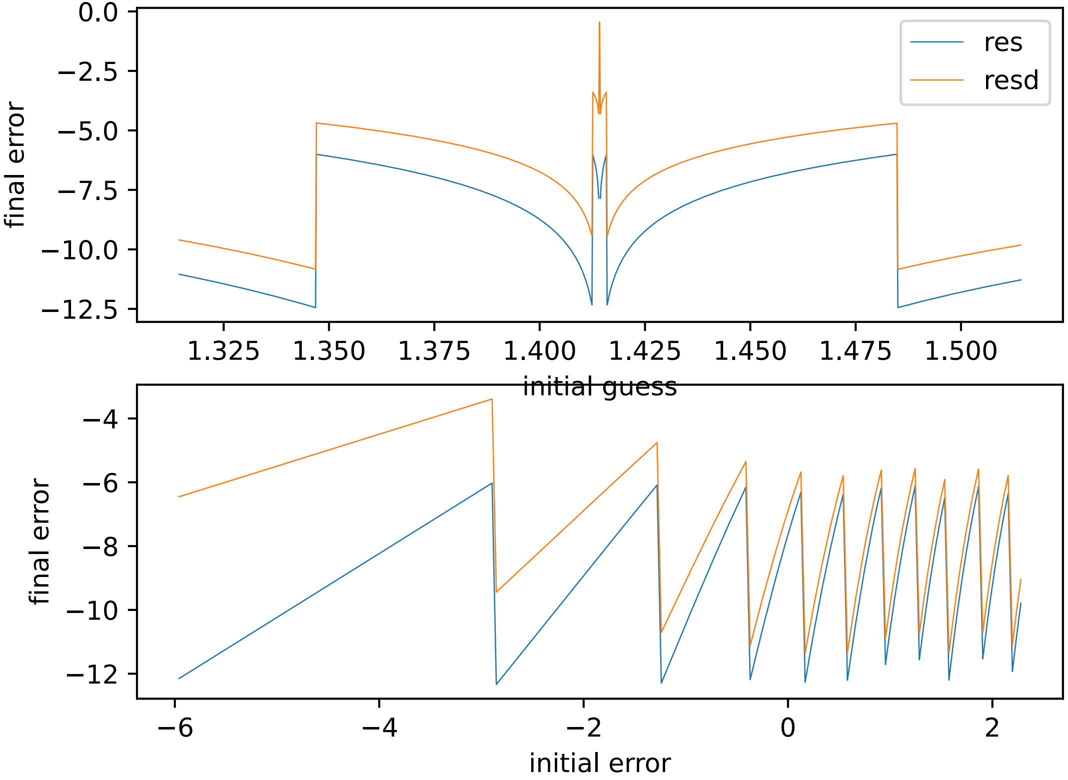
Pitfall 2: Iterative process example

```python
def f(a, x):
  return (a/x + x)*0.5

def heron(a, x0, n, tol=1e-6):
  x = x0
  for i in range(n):
    if(abs(x*x-2)<tol):
      break
    x = f(a, x)
  return x
```

- An old (but effective) method to find sqrt(x)
- We iterate until primal has converged to single precision
- What if we differentiate through this?

Pitfall 2: Iterative process example



Pitfall 2: More Examples

- Linear solvers
- Integrals with parametric discontinuities
- Iterative fixed point loops
- Discretizations (e.g. discrete vs continuous adjoint)
- Monte Carlo methods

- The ”linear solver fix”, “fixed point loop fix”, etc, are raising the abstraction level temporarily.
- They are not really a “fix”: Both answers are “correct”, they just answer different questions.

Pitfall 3

*Do you actually want to differentiate at **that** branch?*

Branches can have *wrong* derivatives

Definition	Code with fast path	Code with modified path

Function:
$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ x & \text{else} \end{cases}$
	`def f(x):`
	`  if x == 0:`
	`    return 0`
	`  else:`
	`    return x`
		`def f(x):`
		`  if x == 0:`
		`    return sin(x) # 0`
		`  else:`
		`    return x`

Derivative:
$\dot{f}(x) = \begin{cases} \dot{x} & \text{if } x = 0 \\ \dot{x} & \text{else} \end{cases}$
	`def f_d(x, xd=1.0):`
	`  if x == 0:`
	`    return 0 # wrong!`
	`  else:`
	`    return xd`
		`def f_d(x, xd=1.0):`
		`  if x == 0:`
		`    return cos(x)*xd # xd`
		`  else:`
		`    return xd`

What is the problem?

- The presence of branches?
- Discontinuities?
- Non-smoothness?
- If the first derivative is right, how about the rest?

Pitfall 3

- *“AD differentiates what is implemented at the abstraction to which you apply AD, **and in the branch that gets executed**. Which occasionally differs from what you think you implement!”*

Pitfall 4

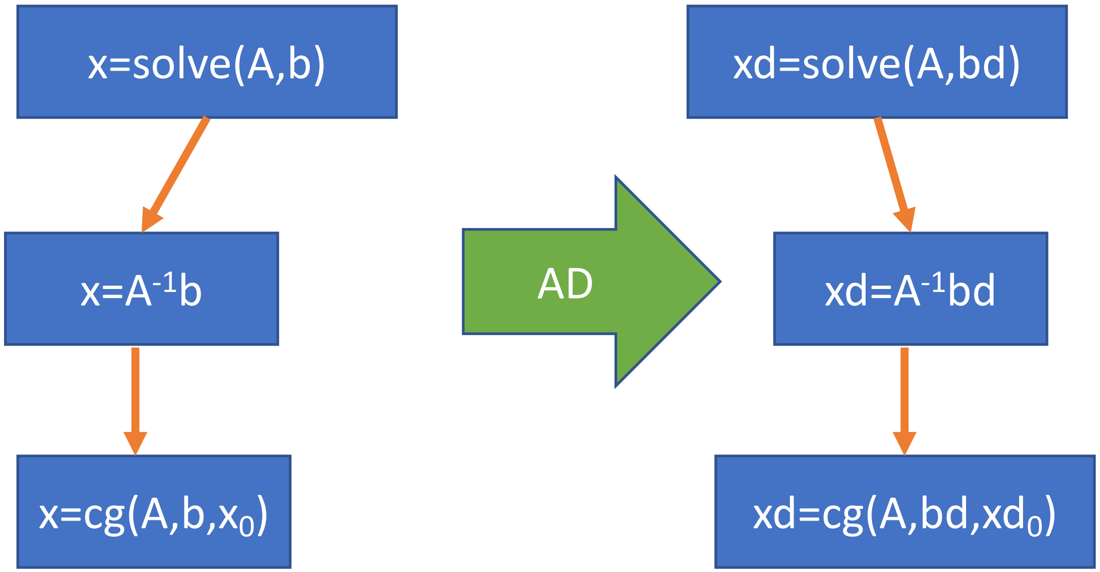
And what about roundoff?

Pitfall 4

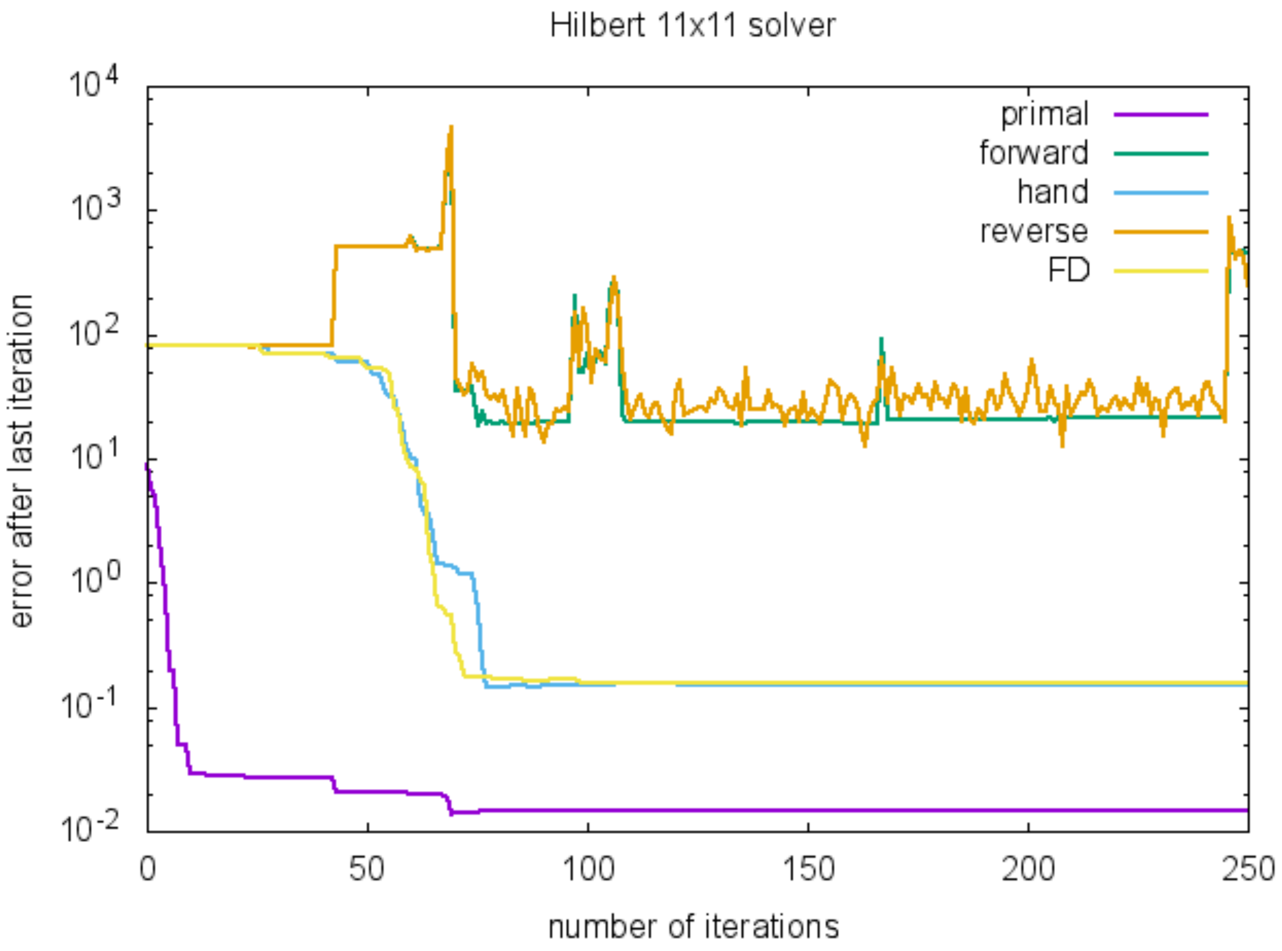
And what about roundoff?

And approximate operators?

Pitfall 4 example



- Does it converge?
- Does it converge well in both cases?
- How good is your initial guess for xd_0?



Hilbert 11x11 solver
error after last iteration
number of iterations
primal
forward
hand
reverse
FD
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
0
50
100
150
200
250

Pitfall 4

- *“AD differentiates what is implemented at the abstraction to which you apply AD, in the branch that gets executed, **and assuming that the operators used at that abstraction level are exact**. Which occasionally differs from what you think you implement!”*

Takeaway Messages, Part 1

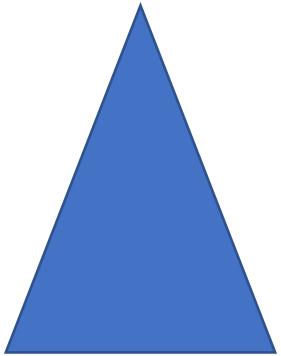
- Even if
  - tools could handle all language features and
  - performance was not a problem,
- AD is not a black-box method to “differentiate your program”.
- You need to understand your function, as implemented
- You need to be able to sanity-check your derivatives

“AD is for people who know the derivative of their code already.”

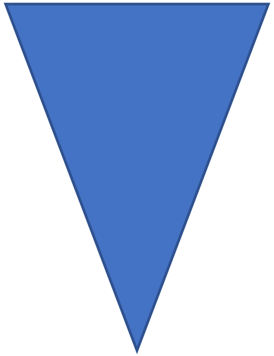
Takeaway Messages, Part 2

Who is responsible for problems?

- Pitfall 1: Weird functions
- Pitfall 2: Weird abstractions
- Pitfall 3: Weird branches
- Pitfall 4: Operator accuracy



Tool?



User?

Takeaway Messages, Part 3

- Enzyme is special!
- User API and AD engine are at different abstraction level
- What if the program semantics changed in between? Who is at fault?
  - Wait for users to report problems and fix case-by-case?
  - Automated checking?
  - Derivative-aware frontends?

Questions / Comments?

- Jan Hückelheim: jhueckelheim@anl.gov

Acknowledgements

The work is funded in part by support from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.