Forward Mode AD

Recall Our Simple Function That We Will Use to Illustrate AD

Our simple function is

$$f(x_1, x_2) = x_1^2 + x_2 \sin\left(x_1^2\right).$$

```matlab
function [f] = func(x1, x2)
% compute a simple function value
v1 = x1.^2;
v2 = x2.*sin(v1);
f = v1 + v2;
end
```

We Need to Distinguish Between Total and Partial Derivatives

Consider $g(x, t) = e^x + t^2$.

- Clearly $\partial g / \partial t = 2t$.
- But, what if x itself is a function of t?

partial derivative: Denoted by $\partial g / \partial t$ here, these are nonzero only if g depends explicitly on t. They assume x is independent of t.

total derivative: Denoted by dg/dt here, these accounts for all dependencies on t.

Apply the Chain Rule to Our Simple Function

$$\begin{aligned} \frac{df}{dx_1} &= \\ &= \\ &= \\ &= \end{aligned}$$

How Do We Implement the Chain Rule in the Code?

- A key observation is that the total derivative of the intermediate values with respect to x_1 can be found sequentially as we progress through the function
- That is, we compute the values of dv_j/dx_1 together with the value v_j

How Do We Implement the Chain Rule in the Code? (cont.)

Applying the above reasoning, we obtain a new function that computes `f` and `dfdx1`, the total derivative df/dx_1.

```matlab
function [f, dfdx1] = dfunc(x1, x2)
% compute a simple function value and its derivative w.r.t. x1
v1 = x1.^2;
dv1dx1 = 2.0.*x1;
v2 = x2.*sin(v1);
dv2dx1 = x2.*cos(v1).*dv1dx1;
f = v1 + v2;
dfdx1 = dv1dx1 + dv2dx1;
end
```

This Example Illustrates the Forward Mode of AD

- We have used an implementation of forward mode called source-code transformation.
- In object-oriented languages, the same result can be achieved by defining a new data type that tracks both the function values and their derivatives.

Pros & Cons of Forward-mode AD

- ✓ no truncation error and no *h* to worry about!
- ✓ the cost of evaluating the differentiated function is only a small factor more than the original code (approximately twice the cost)
- ✗ requires access to the source code
- ✗ computational cost still scales with the # of design variables

Cost is still proportional to the number of design variables, so the advantages over complex-step are minimal.

> **Kenobi:** *That [method] was our last hope.*
>
> **Yoda:** *No. There is another.*