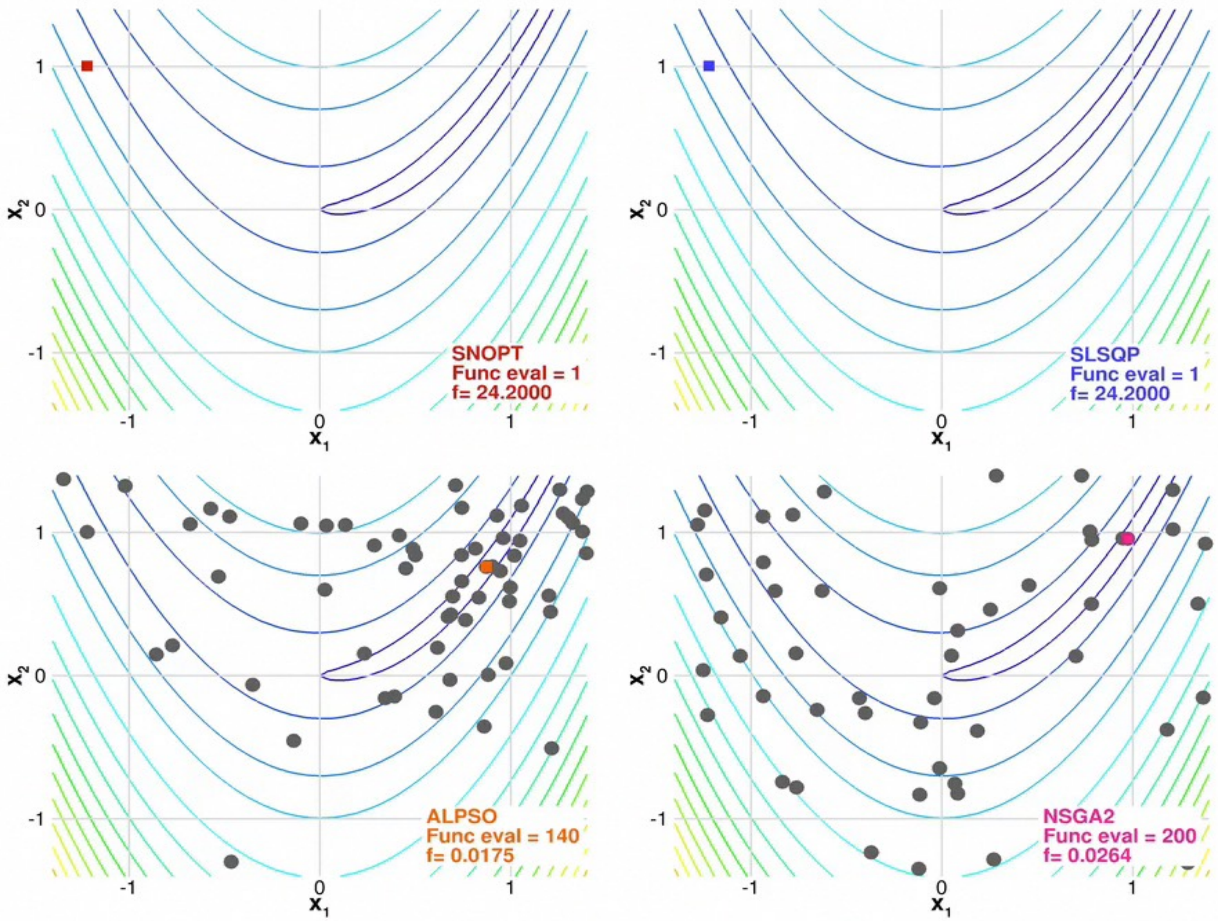
On sensibility

Gradient, Hessian and many more?

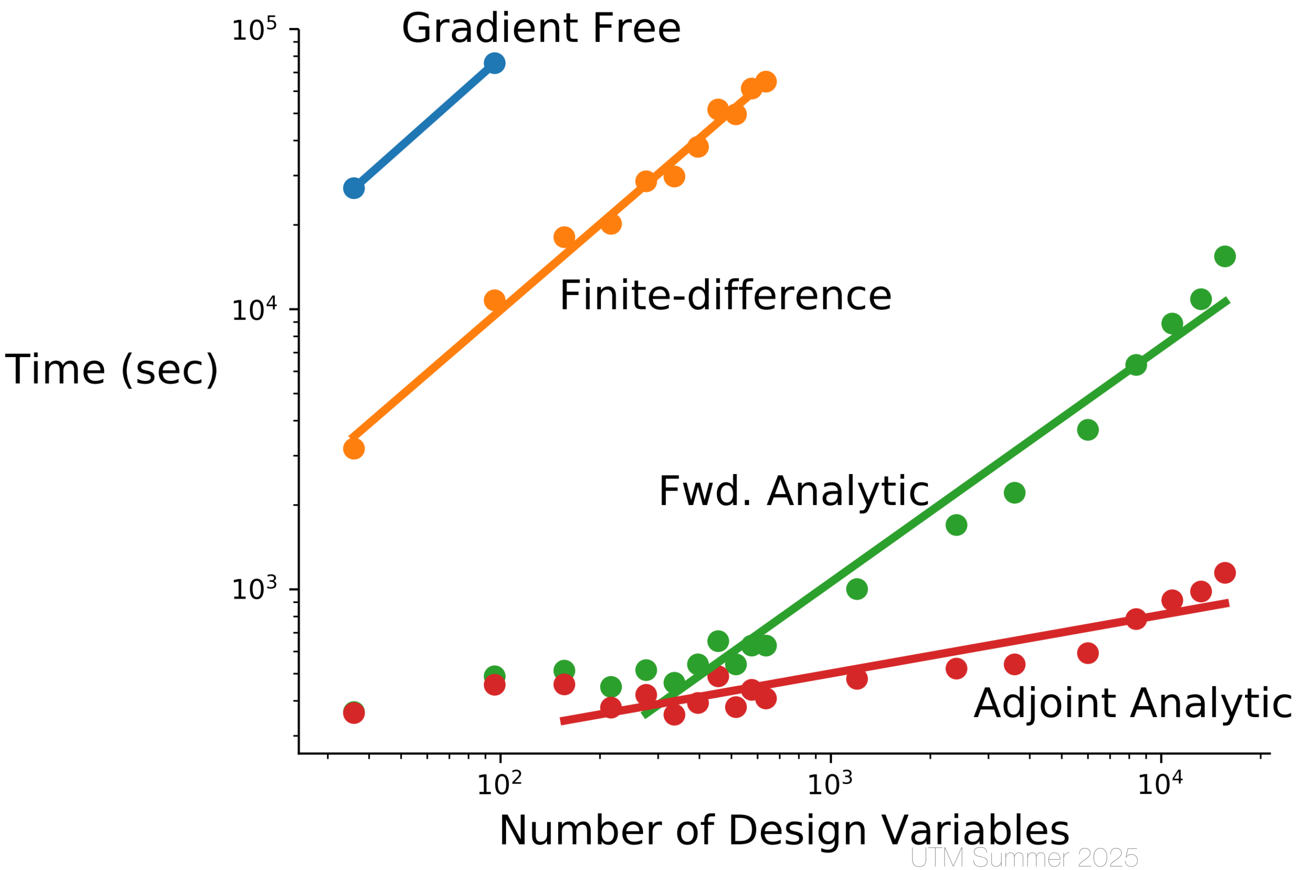
Gradient-based methods take a more direct path to … the optimum

Gradient-based

Gradient-free



Gradient-based optimization with analytic derivatives is our only hope for large-scale problems



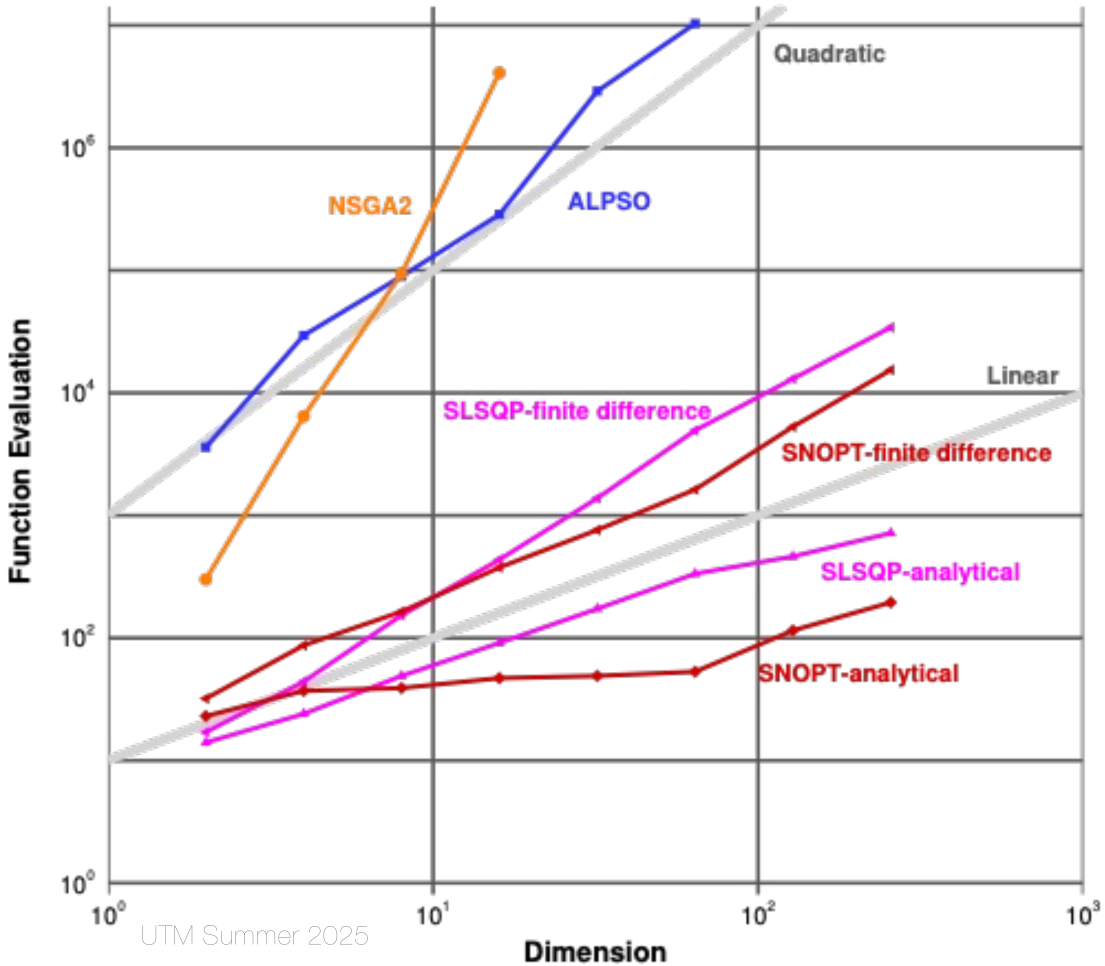
100x-10,000x speedup for aerodynamic shape optimization vs. gradient-free[1]

At least 5x-10x speedup vs. finite-difference[2]

[1] Lyu et al. ICCFD8-2014-0203
[2] Gray et al. Aviation 2014-2042

Gradient-based optimization is the only hope
for large numbers of design variables

Need accurate derivative



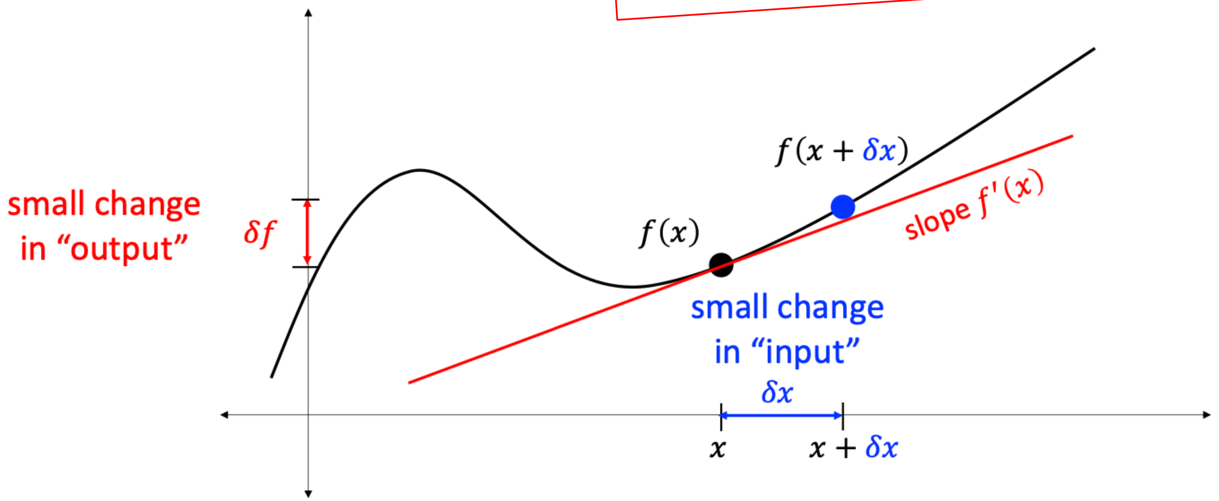
[Lyu et al. ICCFD8-2014-0203]

Derivatives

- Derivatives tell us which direction to search for a solution

The essence of derivative

The essence of a derivative is linearization: predicting a small change δf in the output f(x) from a small change δx in the input x, to first order in δx.



$$\delta f = f(x + \delta x) - f(x) = \underbrace{f'(x)\delta x}_{\text{linear term}} + \underbrace{o(\delta x)}_{\text{higher-order terms}}$$

The essence of derivative

1.2 First Derivatives

The derivative of a function of one variable is itself a function of one variable– it simply is (roughly) defined as the linearization of a function. I.e., it is of the form $(f(x) - f(x_0)) \approx f'(x_0)(x - x_0)$. In this sense, "everything is easy" with scalar functions of scalars (by which we mean, functions that take in one number and spit out one number).

There are occasionally other notations used for this linearization:

- $\delta y \approx f'(x)\delta x$,
- and $df = f'(x)dx$.

This last one will be the preferred of the above for this class. One can think of dx and dy as "really small numbers." In mathematics, they are called infinitesimals, defined rigorously via taking limits. Note that here we do not want to divide by dx. While this is completely fine to do with scalars, once we get to vectors and matrices you can't always divide!

Example

The numerics of such derivatives are simple enough to play around with. For instance, consider the function $f(x) = x^2$ and the point $(x_0, f(x_0)) = (3, 9)$. Then, we have the following numerical values near $(3, 9)$:

$$\begin{aligned} f(3.\mathbf{0001}) &= 9.\mathbf{0006}0001 \\ f(3.\mathbf{00001}) &= 9.\mathbf{00006}00001 \\ f(3.\mathbf{000001}) &= 9.\mathbf{000006}000001 \\ f(3.\mathbf{0000001}) &= 9.\mathbf{0000006}0000001. \end{aligned}$$

Here, the bolded digits on the left are Δx and the bolded digits on the right are Δy. Notice that $\Delta y = 6\Delta x$. Hence, we have that

$$f(3 + \Delta x) = 9 + \Delta y = 9 + 6\Delta x \implies f(3 + \Delta x) - f(3) = 6\Delta x \approx f'(3)\Delta x.$$

Therefore, we have that the linearization of x^2 at $x = 3$ is the function $f(x) - f(3) \approx 6(x - 3)$.

$(\nabla f)^T$, so that df is the dot ("inner") product of dx with the gradient

$$df = \nabla f \cdot dx = \underbrace{(\nabla f)^T}_{f'(x)} dx \text{ where } dx = \begin{pmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_n. \end{pmatrix}$$

This is perfectly consistent with the viewpoint of the gradient that you may remember from multivariable calculus, in which the gradient was a vector of components

$$df = f(x + dx) - f(x) = \nabla f \cdot dx$$
$$= \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \cdots + \frac{\partial f}{\partial x_n} dx_n$$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Example (Interesting for TopOpt with A=K)

Consider $f(x) = x^T A x$ where $x \in \mathbb{R}^n$ and A is a square $n \times n$ matrix, and thus $f(x) \in \mathbb{R}$. Compute $df, f'(x)$, and ∇f.

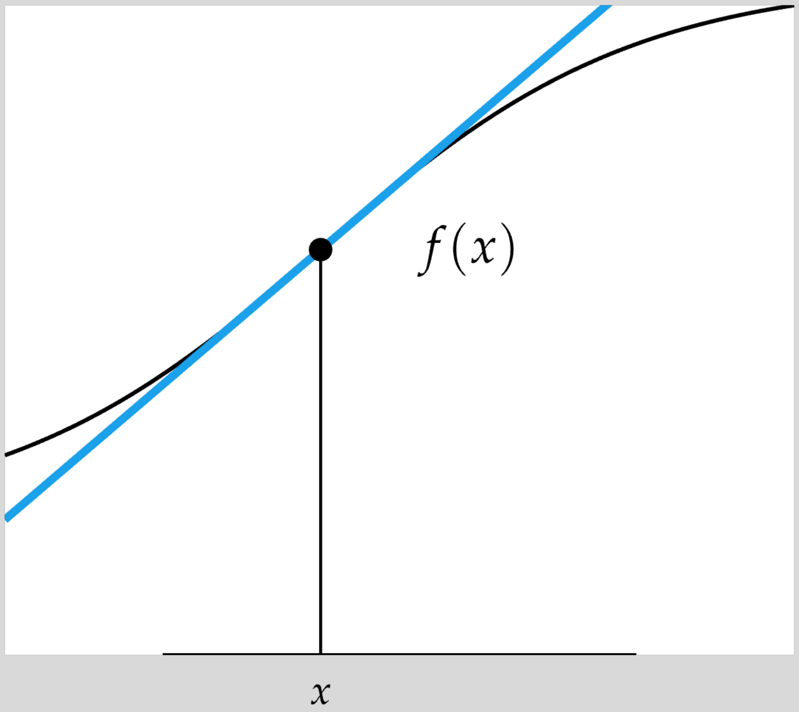
$$
\begin{aligned}
df &= f(x+dx) - f(x) \\
&= \left(x + dx\right)^T A(x+dx) - x^T A x \\
&= x^T A x + dx^T A x + x^T A dx + dx^T A dx - x^T A x \\
&= \underbrace{x^T (A + A^T)}_{f'(x) = (\nabla f)^T} dx \Longrightarrow \nabla f = (A + A^T) x.
\end{aligned}
$$

(which simplifies to $2Ax$ if A is symmetric $A = A^T$).

Derivatives

- Slope of Tangent Line

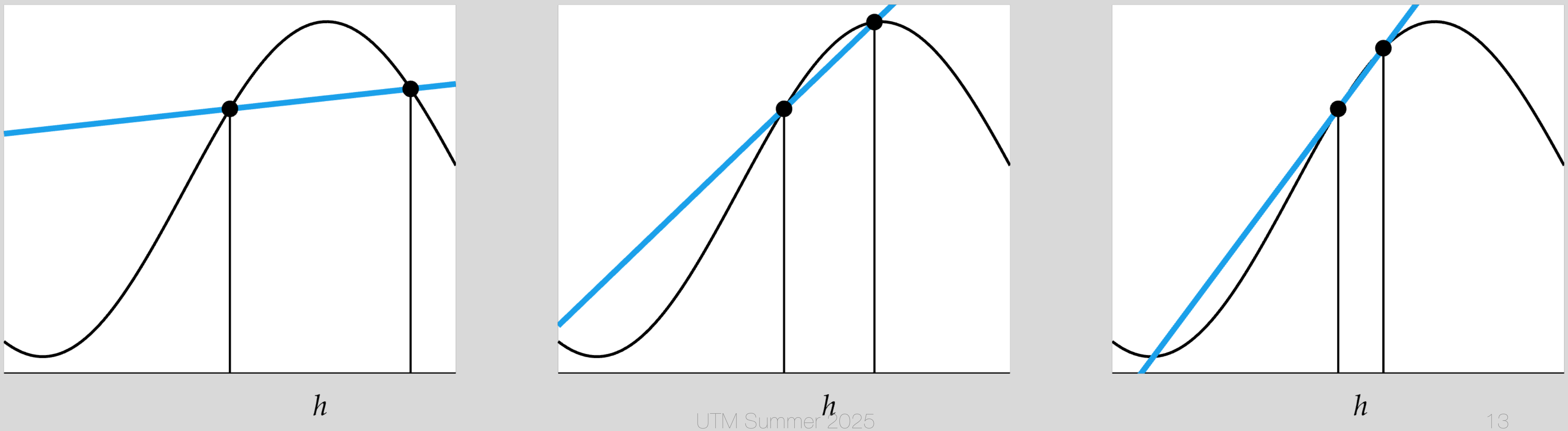
$$f'(x) \equiv \frac{df(x)}{dx}$$



Derivatives

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x$$

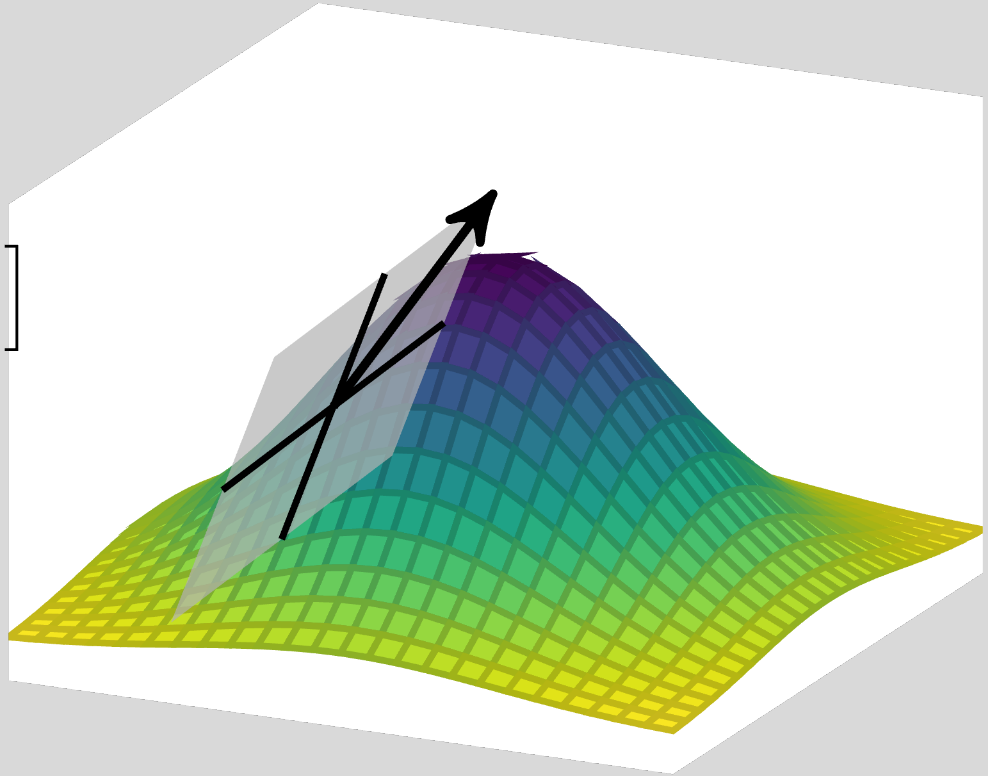
$$f'(x) = \frac{\Delta f(x)}{\Delta x}$$



Derivatives in Multiple Dimensions

- Gradient Vector

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \quad \frac{\partial f(\mathbf{x})}{\partial x_2}, \quad \ldots, \quad \frac{\partial f(\mathbf{x})}{\partial x_n}\right]$$



- Hessian Matrix

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ & \vdots & & \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_n} \end{bmatrix}$$

Derivatives in Multiple Dimensions

- Directional Derivative

$$\nabla_{\mathbf{s}} f(\mathbf{x}) \equiv \underbrace{\lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{s}) - f(\mathbf{x})}{h}}_{\text{forward difference}}$$

$$= \underbrace{\lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{s}/2) - f(\mathbf{x} - h\mathbf{s}/2)}{h}}_{\text{central difference}}$$

$$= \underbrace{\lim_{h \to 0} \frac{f(\mathbf{x}) - f(\mathbf{x} - h\mathbf{s})}{h}}_{\text{backward difference}}$$

Analytical sensitivities

If the objective function is known in closed form, we can often compute the gradient vector(s) in closed form (analytically):

Example

Example: $$J(x_1, x_2) = x_1 + x_2 + \frac{1}{x_1 \cdot x_2}$$

$x_1 = x_2 = 1$

$J(1,1) = 3$

$$\nabla J(1,1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

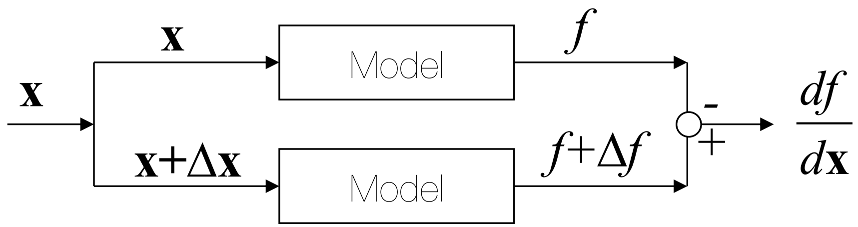
Analytical Gradient: $$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial x_1} \\ \frac{\partial J}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{x_1^2 x_2} \\ 1 - \frac{1}{x_1 x_2^2} \end{bmatrix}$$

Minimum

For complex systems analytical gradients are rarely available

Sensitivity analysis approaches

Simpler approach (*default with fmincon*):



GFD

Matlab

How to proceed with PDE such as Kq=f?

Discrete

Nastran SOL200

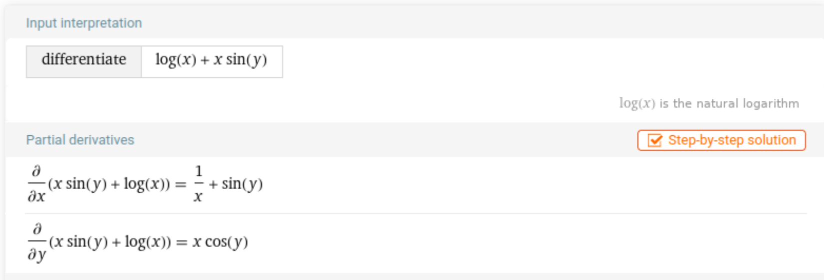
Symbolic differentiation

- Use symbolic mathematics programs
- e.g. MATLAB®, Maple®, Mathematica®

construct a symbolic object

```
» syms x1 x2
» J=x1+x2+1/(x1*x2);
» dJdx1=diff(J,x1)
dJdx1 =1-1/x1^2/x2
» dJdx2=diff(J,x2)
dJdx2 = 1-1/x1/x2^2
```

difference operator



Symbolic differentiation using WolframAlpha.

Automatic Differentiation

- Mathematical formulae are built from a finite set of basic functions, e.g. additions, sin x, exp x, etc.
- Using chain rule, differentiate analysis code: add statements that generate derivatives of the basic functions
- Tracks numerical values of derivatives, does not track symbolically as discussed before
- Outputs modified program = original + derivative capability
- e.g., ADIFOR (FORTRAN), TAPENADE (C, FORTRAN), TOMLAB (MATLAB), many more…
- Resources at http://www.autodiff.org/
- USE JULIA

https://sinews.siam.org/Details-Page/scientific-machine-learning-how-julia-employs-differentiable-programming-to-do-it-best

How Nastran
(a FE code)
is

doing this ?

- General case f is not depending on xi
- except for volumic force (i.e. gravity)

$$K(x)\,u(x) = f(x)$$

$$\frac{\partial K(x)}{\partial x_i}\,u(x) + K(x)\,\frac{\partial u(x)}{\partial x_i} = \frac{\partial f(x)}{\partial x_i}$$

$$K(x)\,\frac{\partial u(x)}{\partial x_i} = \frac{\partial f(x)}{\partial x_i} - \frac{\partial K(x)}{\partial x_i}\,u(x)$$

$$\widetilde{K}\,\widetilde{u} = \widetilde{f}$$

$$\widetilde{u} = \widetilde{K}^{-1}\,\widetilde{f} = K^{-1}\,\widetilde{f}$$

$$\frac{\partial u(x)}{\partial x_i} = K^{-1}\left\{\frac{\partial f(x)}{\partial x_i} - \frac{\partial K(x)}{\partial x_i}\,u(x)\right\}$$

$$= K^{-1}\left\{\frac{f(x+\Delta x) - f(x)}{\Delta x} - \frac{K(x+\Delta x) - K(x)}{\Delta x}\,u(x)\right\}$$

Numerical Differentiation

- Finite Difference Methods
- Complex Step Method

Numerical Differentiation: Finite Difference

- Derivation from Taylor series expansion

$$f(x+h) = f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \cdots$$

Numerical Differentiation: Finite Difference

- Neighboring points are used to approximate the derivative

$$f'(x) \approx \underbrace{\frac{f(x+h) - f(x)}{h}}_{\text{forward difference}} \approx \underbrace{\frac{f(x+h/2) - f(x-h/2)}{h}}_{\text{central difference}} \approx \underbrace{\frac{f(x) - f(x-h)}{h}}_{\text{backward difference}}$$

- h too small causes numerical cancellation errors

Numerical Differentiation: Finite Difference

- Error Analysis
  - Forward Difference: $O(h)$
  - Central Difference: $O(h^2)$

Numerical Differentiation: Complex Step

- Taylor series expansion using imaginary step

$$f(x+ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2!} - ih^3\frac{f'''(x)}{3!} + \cdots$$

$$f'(x) = \frac{\mathrm{Im}(f(x+ih))}{h} + O(h^2) \;\; \text{as } h \to 0$$

$$f(x) = \mathrm{Re}(f(x+ih)) + O(h^2)$$

Complex Step Derivative (see LIVESCRIPT ON LMS)

- Similar to finite differences, but uses an imaginary step

$$f'(x_0) \approx \frac{\mathrm{Im}\left[f(x_0 + i\Delta x)\right]}{\Delta x}$$

Second order accurate

Canx use very small step sizes e.g. $\Delta x \approx 10^{-20}$

Doesn't have rounding error, since it doesn't perform subtraction

Limited application areas

Code must be able to handle complex step values

J.R.R.A. MARTINS, I.M. KROO AND J.J. ALONSO, AN AUTOMATED METHOD FOR SENSITIVITY ANALYSIS USING COMPLEX VARIABLES, AIAA PAPER 2000-0689, JAN 2000

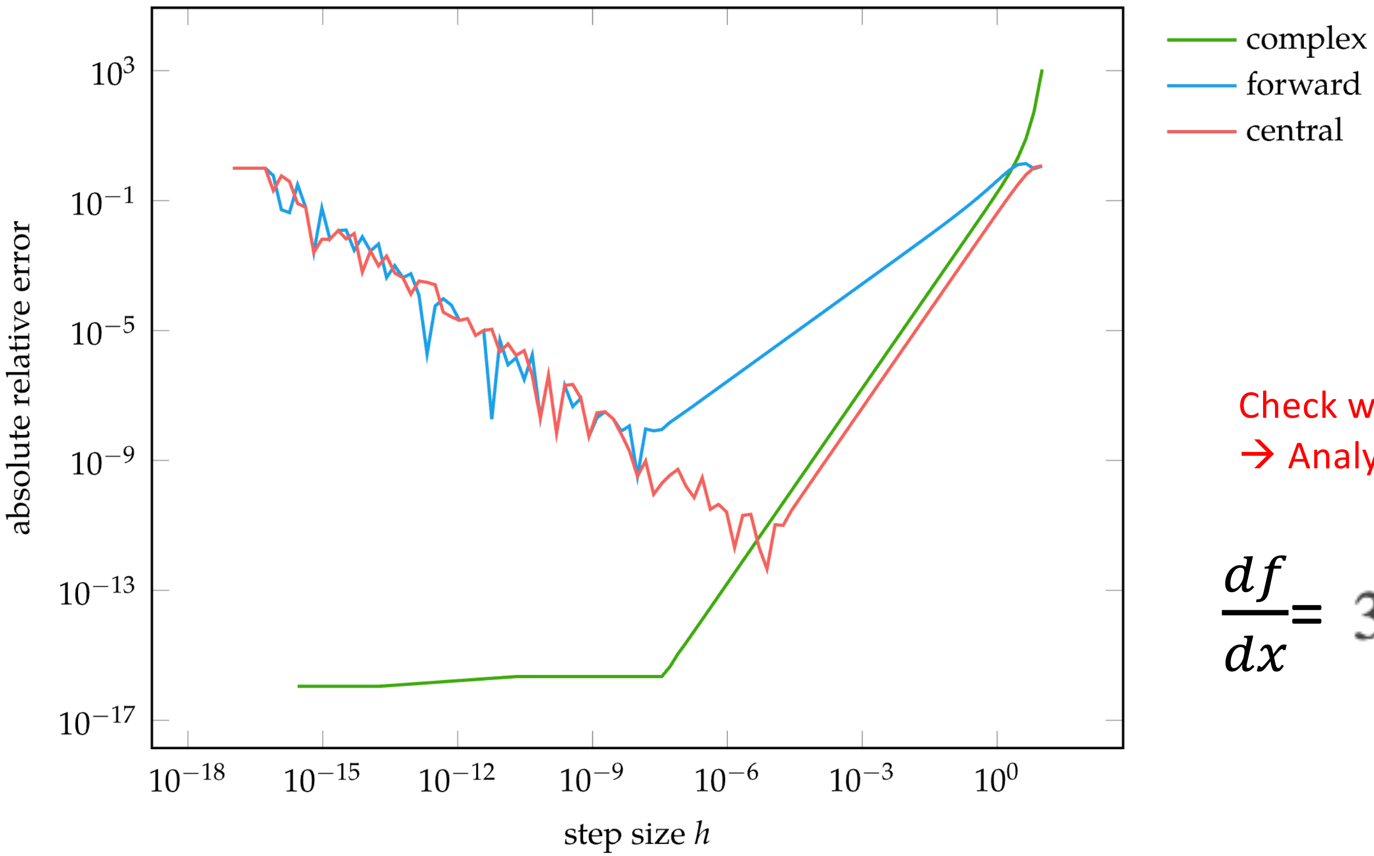
Numerical Differentiation

- Finite Difference Methods
- Complex Step Method

First Exercice

Read and finish the notebook called Complexstep_student.ipynb

Numerical Differentiation Error Comparison



Check with SYMPY
→ Analytical gradient

$$\frac{df}{dx} = 3\cos(3x)\log(x) + \frac{\sin(3x)}{x}$$

Numerical Differentiation

- AD?

Automatic Differentiation

- Evaluate a function and compute partial derivatives simultaneously using the chain rule of differentiation

$$\frac{d}{dx} f(g(x)) = \frac{d}{dx}(f \circ g)(x) = \frac{df}{dg}\frac{dg}{dx}$$

AD . . . is Computer Sciences

A program is composed of elementary operations like addition, subtraction, multiplication, and division.

Consider the function $f(a,b) = \ln(ab + \max(a,2))$. If we want to compute the partial derivative with respect to a at a point, we need to apply the chain rule several times:[9]

$$
\begin{aligned}
\frac{\partial f}{\partial a} &= \frac{\partial}{\partial a} \ln(ab + \max(a,2)) \\
&= \frac{1}{ab + \max(a,2)} \frac{\partial}{\partial a}(ab + \max(a,2)) \\
&= \frac{1}{ab + \max(a,2)} \left[\frac{\partial(ab)}{\partial a} + \frac{\partial \max(a,2)}{\partial a} \right] \\
&= \frac{1}{ab + \max(a,2)} \left[\left(b\frac{\partial a}{\partial a} + a\frac{\partial b}{\partial a} \right) + \left((2 > a)\frac{\partial 2}{\partial a} + (2 < a)\frac{\partial a}{\partial a} \right) \right] \\
&= \frac{1}{ab + \max(a,2)} [b + (2 < a)]
\end{aligned}
$$

One example

- Forward Accumulation is equivalent to expanding a function using the chain rule and computing the derivatives inside-out
- Requires n-passes to compute n-dimensional gradient

$$f(a,b) = \ln(ab + \max(a, 2))$$

$$\frac{\partial f}{\partial a}(3,2)$$

AD computational graphs

$$\frac{\partial f(3,2)}{\partial a}$$

- Forward Accumulation

$$f(a,b) = \ln(ab + \max(a,2))$$

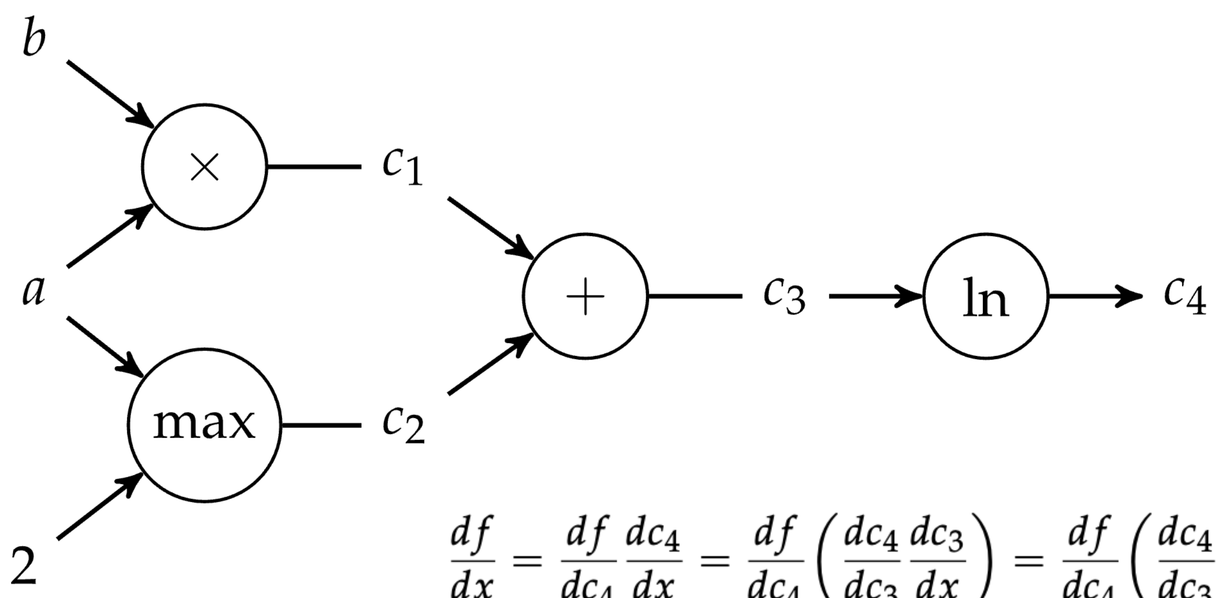
$b = 2$

$\dot{b} = 0$

$\frac{\partial b}{\partial a} =$

$a = 3$

$\dot{a} = 1$



$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\frac{dc_3}{dx}\right) = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\left(\frac{dc_3}{dc_2}\frac{dc_2}{dx} + \frac{dc_3}{dc_1}\frac{dc_1}{dx}\right)\right)$$

Automatic Differentiation

- Forward Accumulation

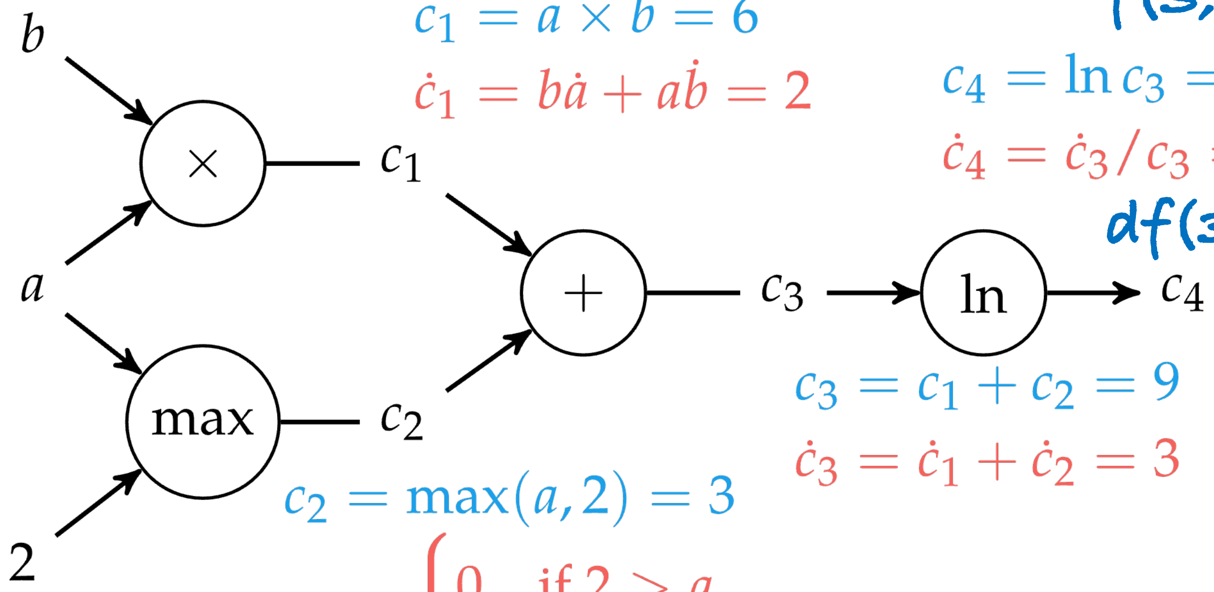
$$f(a,b) = \ln(\, ab + \max(a, 2))$$

$b = 2$

$\dot{b} = 0$

$a = 3$

$\dot{a} = 1$



$c_1 = a \times b = 6$

$\dot{c}_1 = b\dot{a} + a\dot{b} = 2$

$c_2 = \max(a, 2) = 3$

$$\dot{c}_2 = \begin{cases} 0 & \text{if } 2 > a \\ \dot{a} & \text{if } 2 < a \end{cases} = 1$$

$c_3 = c_1 + c_2 = 9$

$\dot{c}_3 = \dot{c}_1 + \dot{c}_2 = 3$

$f(3,2) = \ln 9$

$c_4 = \ln c_3 = \ln 9$

$\dot{c}_4 = \dot{c}_3 / c_3 = \frac{1}{3}$

$df(3,2)/da = 1/3$

In Julia

The `ForwardDiff.jl` package supports an extensive set of mathematical operations and additionally provides gradients and Hessians.

```
julia> using ForwardDiff
julia> a = ForwardDiff.Dual(3,1);
julia> b = ForwardDiff.Dual(2,0);
julia> log(a*b + max(a,2))
Dual{Nothing}(2.1972245773362196,0.3333333333333333)
```

In Julia

The `Zygote.jl` package provides automatic differentiation in the form of reverse-accumulation. Here the `gradient` function is used to automatically generate the backwards pass through the source code of `f` to obtain the gradient.

```
julia> import Zygote: gradient
julia> f(a, b) = log(a*b + max(a,2));
julia> gradient(f, 3.0, 2.0)
(0.3333333333333333, 0.3333333333333333)
```

So start… with CasADi

https://colab.research.google.com/drive/1IV_c2zB5kpaF2RLq6LC2YvAZkTTTL2oh#scrollTo=PaY36gax0iWn