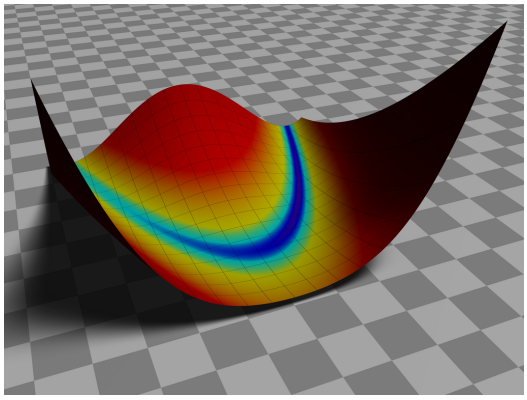
Finite Difference and Complex Step

Lecture 9



ME EN 575
Andrew Ning
aning@byu.edu

Outline

Computing Gradients

$$\begin{aligned} \text{minimize} \quad & f(x_i) \\ \text{w.r.t} \quad & x_i \qquad\qquad i = 1, 2, \ldots, n \\ \text{subject to} \quad & c_j(x_i) \geq 0, \quad j = 1, 2, \ldots, m \end{aligned}$$

If using a gradient-based method, what gradients do we need?

- Gradient of objective: $g = \nabla f(x) = \partial f / \partial x_i$ $(n \times 1)$
- Gradients of constraints: $\partial c_j / \partial x_i$ $(m \times n)$

Gradients useful not just for optimization, but also for sensitivity analysis, uncertainty quantification, ODE simulations, etc.

Finite Difference

Perform a Taylor's series expansion of $f(x+h)$

$$f(x+h) = f(x)+hf'(x)+\frac{h^2}{2!}f''(x)+\frac{h^3}{3!}f'''(x)+\ldots,$$

Solve for $f'(x)$

$$f'(x) = \frac{f(x+h)-f(x)}{h} + \mathcal{O}(h),$$

Foward Difference:

$$\boxed{f'(x) \approx \frac{f(x+h)-f(x)}{h}}$$

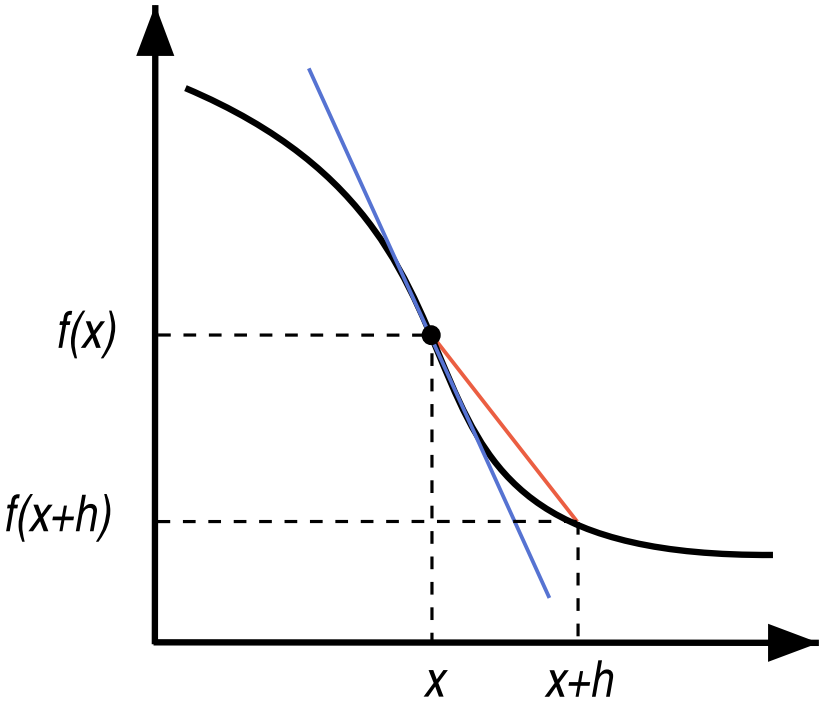
Generalize to ND:

$$\boxed{\frac{\partial f}{\partial x_i} \approx \frac{f(x+h\hat{e}_i)-f(x)}{h}}$$

Must be repeated for each $i = 1 \ldots n$

Can also do backwards difference:

$$\boxed{f'(x) \approx \frac{f(x) - f(x-h)}{h}}$$



We can get also get a second-order accurate gradient with just function values.

$$f(x+h) = f(x)+hf'(x)+\frac{h^2}{2!}f''(x)+\frac{h^3}{3!}f'''(x)+\ldots,$$

$$f(x-h) = f(x)-hf'(x)+\frac{h^2}{2!}f''(x)-\frac{h^3}{3!}f'''(x)+\ldots,$$

Subtract these two equations

$$f'(x) = \frac{f(x+h)-f(x-h)}{2h} + \mathcal{O}(h^2).$$

Central-Difference:

$$\boxed{f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}}$$

ND:

$$\boxed{\frac{\partial f}{\partial x_i} \approx \frac{f(x+h\hat{e}_i)-f(x-h\hat{e}_i)}{2h}}$$

Must be repeated $2n$ times.

What step size (h) should I use?

- Too large: truncation error
- Too small: subtractive cancellation error

Subtractive cancellation

$f(x+h)$	$+1.2345678901234\color{red}{31}$
$f(x)$	$+1.2345678901234\color{black}{56}$
Δ f	-0.000000000000025

What size to choose for h?

True answer: we don't know exactly. Requires testing.

Best estimate: for a well-scaled problem we can show that the best h is approximately the square root of the relative error of f.

Example:

- Function value accuracy: 1×10^{-12} (16 digits possible double precision)
- Step size: 1×10^{-6}

For central difference an estimate for step size scales with the $1/3$ power.

Example:

- Function value accuracy: 1×10^{-12} (16 digits possible double precision)
- Step size: 1×10^{-4}

However, gain in accuracy is not actually second order because of finite precision arithmetic.

Pros and Cons

- Easy. Code can be a black box. Just need function values.
- Expensive. Forward/backwards difference requires n function calls. Central difference requires $2n$ function calls. For large problems this is the bottleneck.
- Limited Accuracy. Consequence of finite precision arithmetic.

Example: fd.py

Complex Step

Consider a Taylor's series expansion where we take a step in the imaginary direction $f(x+ih)$:

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

Solve for $f'(x)$

$$f'(x) = \frac{\text{Im}\left[f(x+ih)\right]}{h} + h^2\frac{f'''(x)}{3!} + \dots$$

Complex step:

$$\boxed{\frac{\partial f}{\partial x_k} \approx \frac{\text{Im}\left[f\left(x+ih\hat{e}_k\right)\right]}{h}}$$

What is the advantage?

- Error is $\mathcal{O}(h^2)$, but that's not the big deal.
- Big deal: **no subtraction!**. This means no subtractive error. Can take arbitrarily small steps* and the error is effectively zero.

* Of course we can't take a number below the lowest floating point number. In practice 1×10^{-20} is a good number to use.

Revisit fd.py

Can we apply to an arbitrary algorithm?

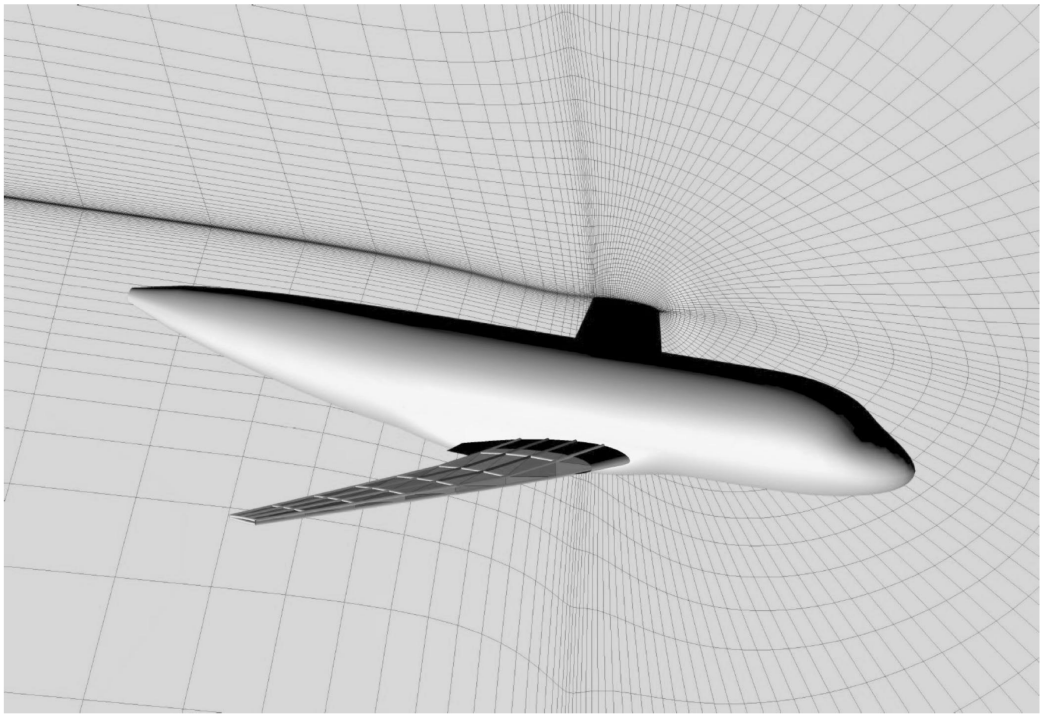
Yes, with some modifications.

- Change real type to complex.
- Some logical (e.g., if statements) may need to be modified to just compare real part.
- A few functions must be overloaded. For example:
  - *abs:* complex version is not analytic
  - *sin, arcsin*: uses subtraction for some compilers, but can be reformulated with trig identities to avoid subtraction

For several languages (Fortran, C, C++, Python, Matlab) there are script and header files to help with this.

In Matlab you also have to be careful with transpose (apostrophe) because Matlab does complex conjugate. Need to change to non-conjugate transpose (dot apostrophe).

Has been done with 3D CFD and FEM codes.



(Martins, Sturdza, Alonso)

Pros and Cons

- Accurate! Analytic accuracy.
- Computing gradients is slower (although optimization convergence is likely faster). Still requires n functions evaluations, and each function evaluation is approximately twice as expensive.

Next time: automatic differentiation