Automating/avoiding derivative calculations (§11.4 in book)

- Many optimization algorithms need derivatives
- **Newton**: first and second derivatives needed
- **Steepest-descent, quasi-Newton**: first derivatives needed
- However, derivatives may not be available or useful:
  - Function values may be given by measurements or black-box simulation code
  - Function may be noisy
  - Function may possess numerous local minima
If function is very irregular or very noisy:

- Alt. 1: use some “filtering” strategy, e.g. “response surface” (fitting observations to a simplified formula such as a polynomial)
- Alt. 2: Use a stochastic method (e.g. genetic algorithms, evolutionary strategies). Does not use gradients.
- Stochastic methods generally needs many function evaluations: only feasible when function evaluations are not too expensive

If function not too noisy or too irregular (thus, derivatives can be used), but if exact derivatives are not available:

- Let the computer calculate derivatives:
  - Finite differences (approximate derivatives)
  - Automatic (or algorithmic) differentiation (exact derivatives)
- Use derivative-free methods such as (Multidirectional search, Nelder–Mead)
Computer-calculated derivatives I

Simplest strategy: Use **finite differences**

\[
\frac{\partial f(x)}{\partial x_i} \approx \begin{cases} 
\frac{f(x+e_i\delta) - f(x)}{\delta} & \text{Forward Euler} \\
\frac{f(x+e_i\delta) - f(x-e_i\delta)}{2\delta} & \text{Central difference}
\end{cases}
\]

- Available as an option in most optimization software
- “Non-intrusive”, can be used with black-box simulation code
- Cost: \(O(n)\) function evaluations for each gradient, \(O(n^2)\) function evaluations for each Hessian \((x \in \mathbb{R}^n)\)
- Selection of step \(\delta\) delicate:
  - Too large steps yields inaccurate derivatives
  - Too small steps yields cancelation of significant digits
Example

- Let \( f(x) = e^x \), \( f'(x) = e^x \).
- Study the error
  \[ |f'(1) - \frac{f(1+h)-f(1)}{h}| \]
  for \( h = 1, 0.1, 0.01, \ldots \)

For forward Euler: optimal value of \( h \sim \sqrt{\epsilon_M} \approx 10^{-8} \) for IEEE double precision
Computer-calculated derivatives II

Automatic differentiation (AD)

- Useable for function evaluation coming from a computer calculation (not measurements!)
- “Intrusive”: needs modifications of the computer source code
- Observation:
  - Each line in a computer program is easy to differentiate
  - Differentiation rules (e.g. product rule, chain rule) are completely mechanical
  - Let the computer analyze each row in the program and calculate the derivative simultaneously as the function
Assume we have a computer program computes the function 
\( f : \mathbb{R}^n \rightarrow \mathbb{R} \)

AD software turns this program into another program returning \( f(x) \) and \( \nabla f(x) \)

Two ways to implement AD:
- Source transformation (compiler technology)
- Operator overloading
AD with operator overloading

Convenient in languages such as C++ and Java
Redefines real variables and redefines arithmetic operations to include derivative information

- $u, v, w$: real variables (their values depend on input vector $x$)
- $\alpha, \beta$: constants (do not depend on $x$)
- Data structures for $u, v$:

$$
\mathbf{u} = \begin{pmatrix} du \\ u \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} dv \\ v \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} dw \\ w \end{pmatrix}
$$

- For simplicity, assume scalar $du, dv, dw$ (input vector $x$ scalar)
- Straightforward to extend to vector $du, dv, dw$
Examples

- Operation: $v = \alpha u$. In code: $v = \alpha \ast u$. Defined as
  \[ v = \begin{pmatrix} \alpha du \\ \alpha u \end{pmatrix} \]

- Operation: $w = uv$. In code: $w = u \ast v$. Defined as
  \[ w = \begin{pmatrix} u dv + v du \\ uv \end{pmatrix} \]

- Operation: $w = \frac{u}{v}$. In code: $w = u / v$. Defined as
  \[ w = \begin{pmatrix} \frac{du}{v} - \frac{1}{v^2} dv \\ \frac{u}{v} \end{pmatrix} \]
Change data type for each variable in program calculating $f$

Providing the input \( \frac{dx}{x} = \begin{pmatrix} 1 \\ x_0 \end{pmatrix} \) yields the output \( \frac{df}{f} = \begin{pmatrix} f'(x_0) \\ f(x_0) \end{pmatrix} \)

Yields exact derivatives (up to machine precision)

Computational effort grows linearly with the dimension of $x$

There are more efficient, but more memory-demanding versions of AD ("backward mode")