Computational Physics (PHYS6350)

Lecture 8: Numerical Derivatives

- Finite differences
- Automatic differentiation

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x+h)-f(x)}{h}$$

February 18, 2025

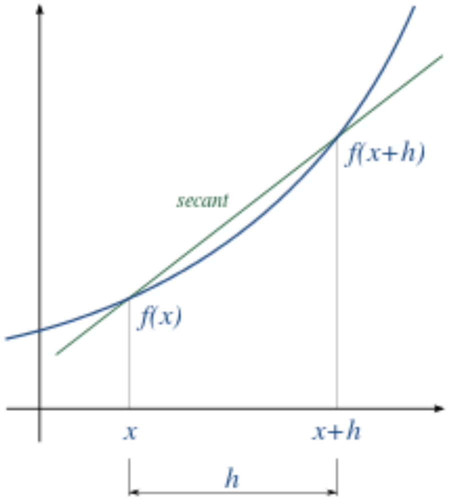
Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics/tree/spring2025

Numerical differentiation

Generic problem: evaluate

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$



We need numerical differentiation when

- Function f is known at a discrete set of points
- Too expensive/cumbersome to do directly
    - For example, when f(x) itself is a solution to a complex system of non-linear equations, calculating f'(x) explicitly will require rewriting all the equations

References: Chapter 5 of *Computational Physics* by Mark Newman

https://autodiff.github.io/

Forward difference

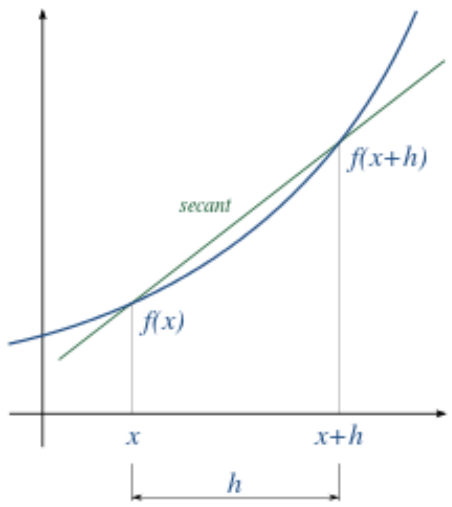
Simply approximate

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

by

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x+h) - f(x)}{h}$$

where h is finite



Taylor theorem:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \dots$$

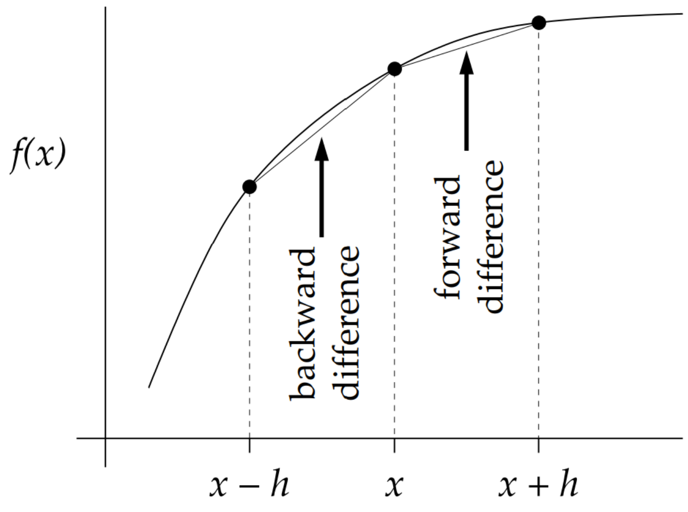
gives the approximation error estimate of

$$R_{\text{forw}} = -\frac{1}{2}hf''(x) + \mathcal{O}(h^2)$$

Backward difference

Backward difference

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x) - f(x-h)}{h}$$



Taylor theorem:

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

gives the approximation error estimate of

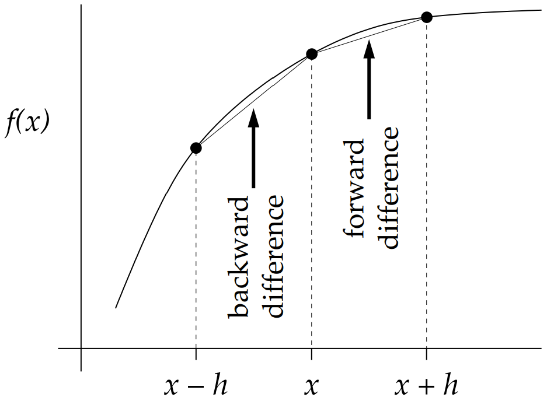
$$R_{\text{back}} = \frac{1}{2} h f''(x) + \mathcal{O}(h^2)$$

Central difference

Recall the forward and backward difference and their errors

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x+h)-f(x)}{h} \qquad \frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x)-f(x-h)}{h}$$

$$R_{\mathrm{forw}} = -\frac{1}{2}hf''(x) + \mathcal{O}(h^2) \qquad R_{\mathrm{back}} = \frac{1}{2}hf''(x) + \mathcal{O}(h^2)$$



Taking the average of the two cancels out the *O(h)* error term

central difference $$\frac{df}{dx} \simeq \frac{f(x+h)-f(x-h)}{2h}$$

Error: $$R_{\mathrm{cent}} = -\frac{f'''(x)}{6}h^2 + \mathcal{O}(h^3)$$

High-order central difference

To improve the approximation error, use more function evaluations, e.g.

$$\frac{df}{dx} \simeq \frac{Af(x+2h) + Bf(x+h) + Cf(x) + Df(x-h) + Ef(x-2h)}{h} + O(h^4)$$

Determine A, B, C, D, E using Taylor expansion to cancel all terms up to h^4

$$\frac{df}{dx} \simeq \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + \frac{h^4}{30} f^{(5)}(x)$$

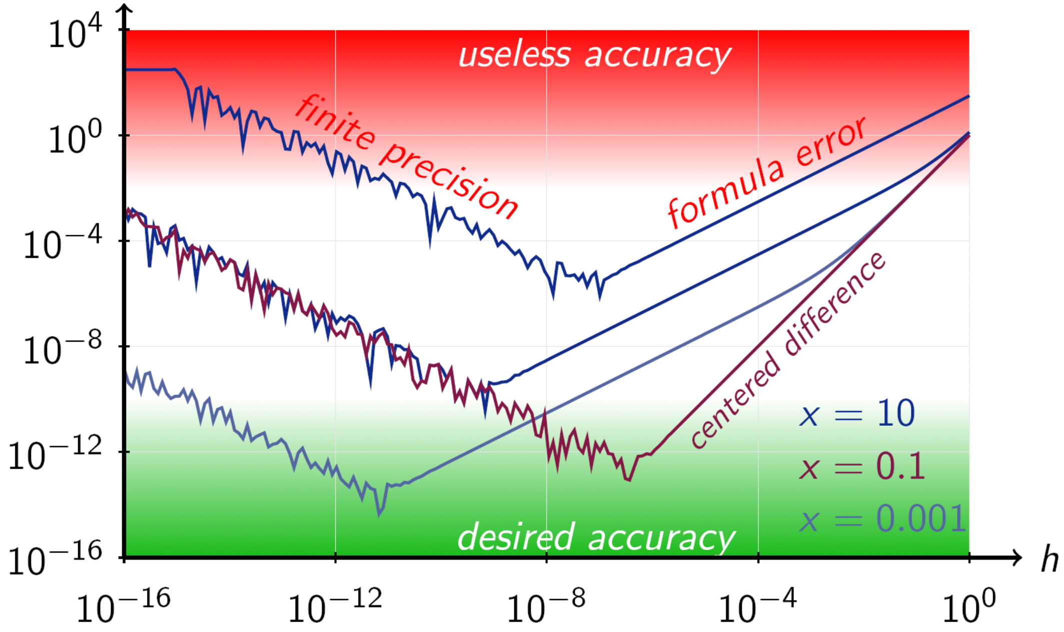
High-order terms:

Derivative	Accuracy	−5	−4	−3	−2	−1	0	1	2	3	4	5
1	2					−1/2	0	1/2				
	4				1/12	−2/3	0	2/3	−1/12			
	6			−1/60	3/20	−3/4	0	3/4	−3/20	1/60		
	8		1/280	−4/105	1/5	−4/5	0	4/5	−1/5	4/105	−1/280	

Balancing truncation and round-off errors

If h is too small, **round-off errors** become important

- cannot distinguish $x+h$ and x and/or $f(x+h)$ and $f(x)$ with enough accuracy



Credit: Wikipedia

Rule of thumb:

- if ε is machine precision and the truncation error is of order $O(h^n)$, then h should not be much smaller than $h \sim \sqrt[n+1]{\varepsilon}$

The higher the finite difference order is, the larger h should be

Balancing truncation and round-off errors

Consider central difference:

$$\frac{df}{dx} \simeq \frac{f(x+h) - f(x-h)}{2h}$$

Total error:

$$\text{error(df/dx)} = \underbrace{\varepsilon_m \frac{|f(x)|}{h}}_{\textit{round-off}} + \underbrace{\frac{|f'''(x)|}{6} h^2}_{\textit{truncation}}$$

ε_m - relative error in f(x)

$\varepsilon_m \sim 10^{-16}$

Best case: machine precision

Minimizing with respect to *h* gives optimal choice for the step size:

$$h = \sqrt[3]{6\,\varepsilon_m |f'''(x)|/|f(x)|} \sim \sqrt[3]{\varepsilon_m\, |f'''(x)|/|f(x)|}$$

More generally, for *O(h^n)* scheme one has

$$h \sim \sqrt[n+1]{\varepsilon_m\, |f^{(n+1)}(x)|/|f(x)|}$$

Balancing truncation and round-off errors

Let $f(x) = exp(x)$

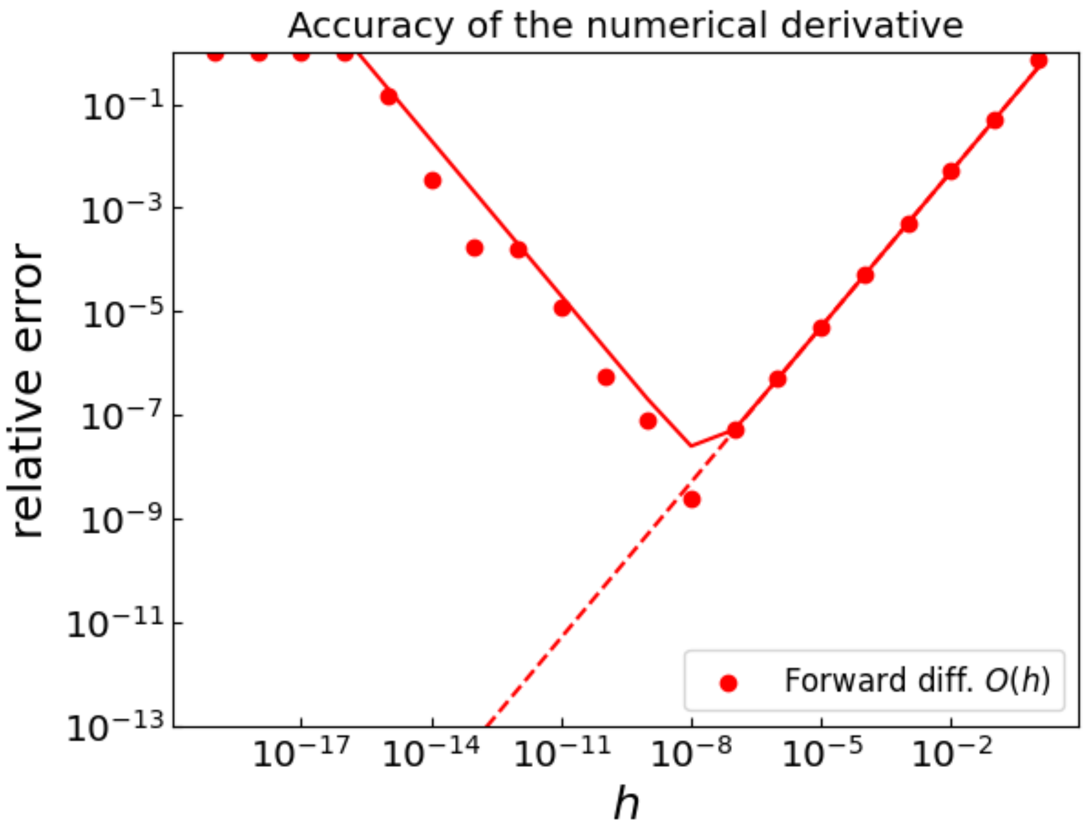
Calculate the derivatives at $x = 0$

```
def f(x):
    return np.exp(x)

def df(x):
    return np.exp(x)
```

Forward difference O(h):

Optimal $h \sim \sqrt[2]{10^{-16}} \sim 10^{-8}$



Balancing truncation and round-off errors

Let $f(x) = exp(x)$

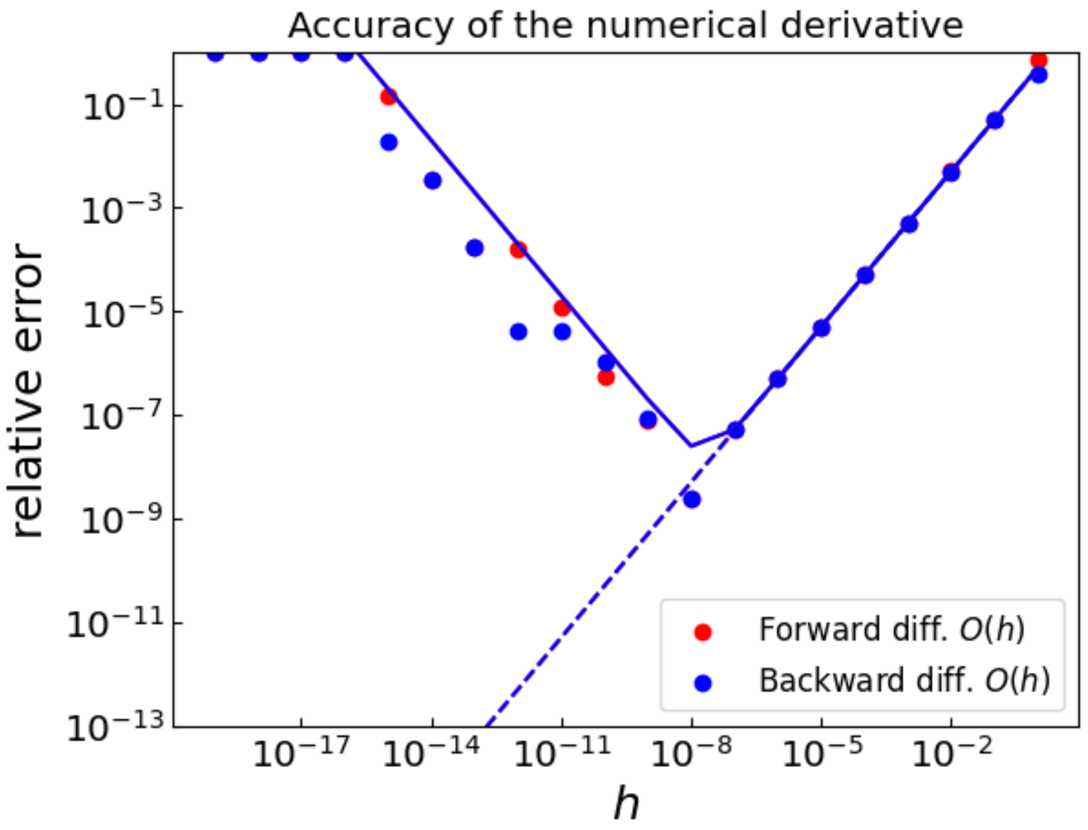
Calculate the derivatives at $x = 0$

```
def f(x):
    return np.exp(x)

def df(x):
    return np.exp(x)
```

Backward difference O(h):

Optimal $h \sim \sqrt[2]{10^{-16}} \sim 10^{-8}$



Balancing truncation and round-off errors

Let $f(x) = exp(x)$

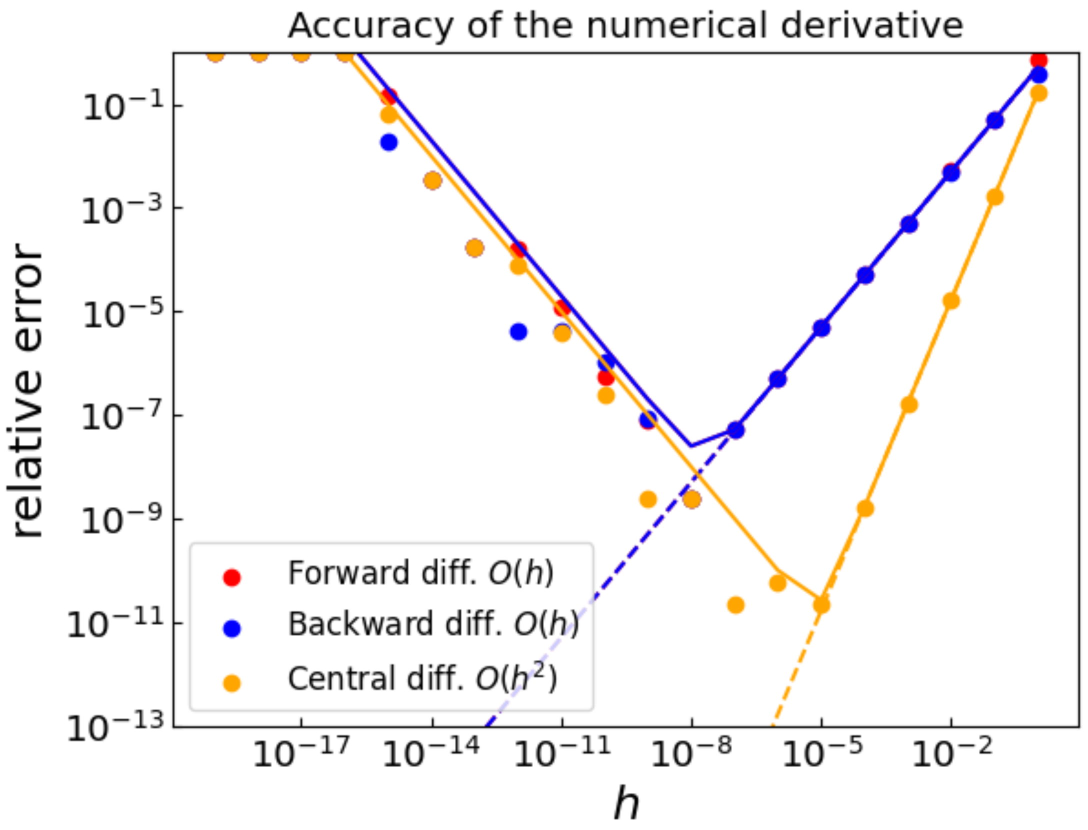
Calculate the derivatives at $x = 0$

```
def f(x):
    return np.exp(x)

def df(x):
    return np.exp(x)
```

Central difference O(h²):

Optimal $h \sim \sqrt[3]{10^{-16}} \sim 10^{-5}$



Balancing truncation and round-off errors

Let $f(x) = exp(x)$

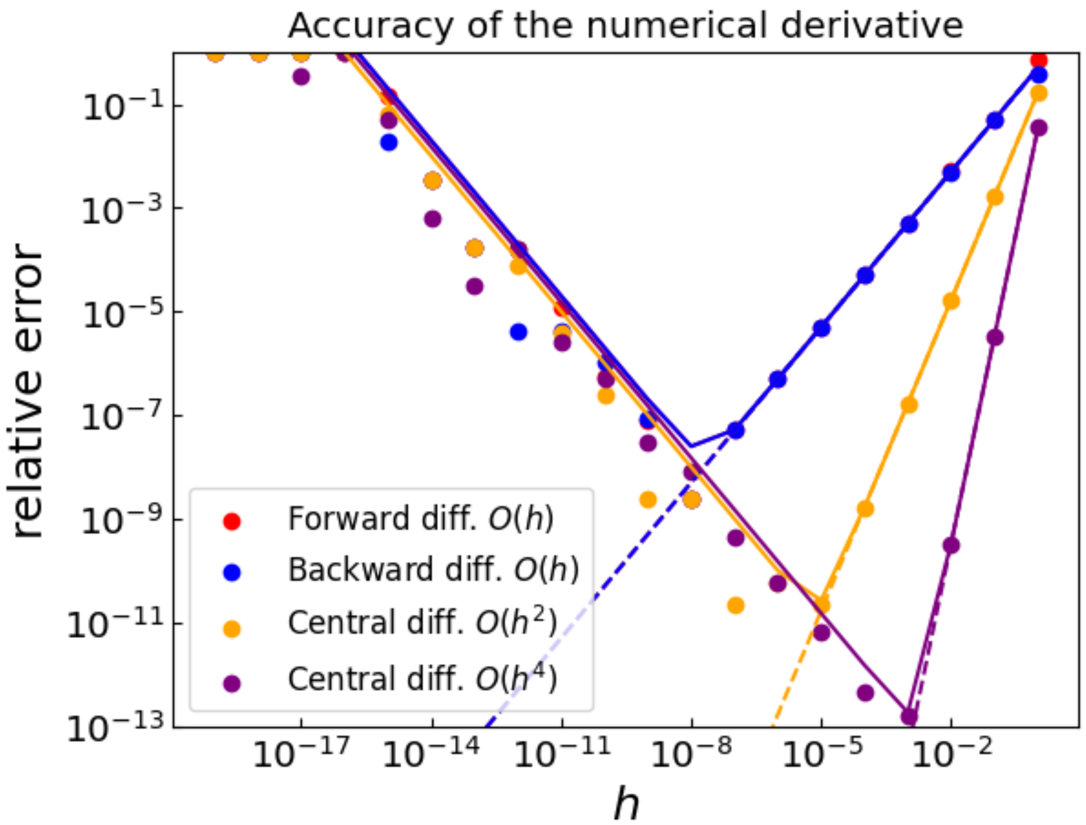
Calculate the derivatives at $x = 0$

```
def f(x):
    return np.exp(x)

def df(x):
    return np.exp(x)
```

Central difference O(h⁴):

Optimal $h \sim \sqrt[5]{10^{-16}} \sim 10^{-3}$



High-order derivatives

Central difference

$$\frac{df}{dx}(x) \simeq \frac{f(x+h/2) - f(x-h/2)}{h}$$

Now apply the central difference again to *f'(x+h/2)* and *f'(x-h/2)*

$$\begin{aligned} f''(x) &\simeq \frac{f'(x+h/2) - f'(x-h/2)}{h} \\ &= \frac{[f(x+h) - f(x)]/h - [f(x) - f(x-h)]/h}{h} \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \end{aligned}$$

General formula [to order O(h^2)]

$$f^{(n)}(x) = \frac{1}{h^n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} f\left[x + (n/2 - k)h\right] + O(h^2)$$

Second derivative

```python
def d2f_central(f,x,h):
    return (f(x+h) - 2*f(x) + f(x-h)) / (h**2)
```

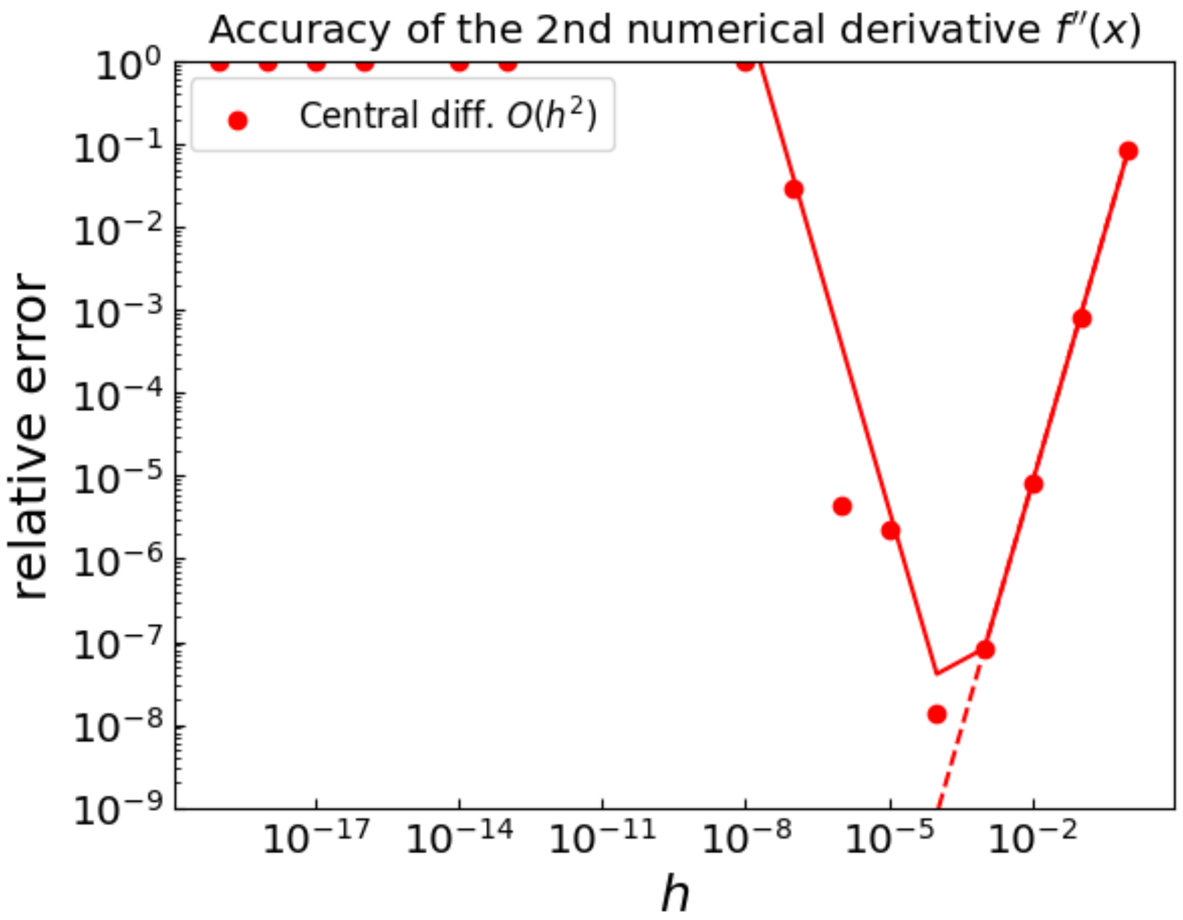
$$f(x) = exp(x)$$

```python
def f(x):
    return np.exp(x)

def df(x):
    return np.exp(x)

def d2f(x):
    return np.exp(x)
```

Optimal $h \sim \sqrt[4]{10^{-16}} \sim 10^{-4}$



Partial derivatives

Let us have a function of two variables: *f(x,y)*
Use central difference to calculate first-order derivatives

$$\frac{\partial f}{\partial x} = \frac{f(x+h/2,y) - f(x-h/2,y)}{h}$$
$$\frac{\partial f}{\partial y} = \frac{f(x,y+h/2) - f(x,y-h/2)}{h}$$

Reapply the central difference to calculate $\partial^2 f(x,y)/\partial x \partial y$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{f(x+h/2,y+h/2) - f(x-h/2,y+h/2) - f(x+h/2,y-h/2) + f(x-h/2,y-h/2)}{h^2}$$

Finite differences: Summary

- **Forward/backward differences**
  - Useful when we are given a grid of function values
  - Need $f'(x)$ at the same point as x
  - Have limited accuracy (error is linear in h)

- **Central difference**
  - More precise than forward/backward differences (error is quadratic in h)
  - Gives $f'(x)$ estimate at the midpoint of function evaluation points

- **Higher-order formulas are obtained by using more than two function evaluations**
  - Can be used when limited number of function evaluations available

- **Straightforwardly extendable to high-order and partial derivatives**

- **Balance between truncation and round-off error must be respected**
  - h should not be taken too small

Automatic differentiation

Automatic differentiation (or **algorithmic differentiation**) is a computational technique to evaluate derivatives of a function specified by a computer program

It is based on the fact that every computer calculation executes a sequence of

- Elementary arithmetic operations (+,-,*,/)
- Elementary functions (exp, log, sin, …)

Calculation of the derivatives then proceeds via the **chain rule**

View computer calculation as evaluating a composite function: $y = f(g(h(x)))$

Numerical value: $y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$

Derivative (gradient): $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial f(w_2)}{\partial w_2}\frac{\partial g(w_1)}{\partial w_1}\frac{\partial h(w_0)}{\partial x}$

Resulting calculating is in theory exact

Automatic differentiation: Example

Numerical value | **Derivative (gradient)**

$$w_0 = x$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial f(w_2)}{\partial w_2}\frac{\partial g(w_1)}{\partial w_1}\frac{\partial h(w_0)}{\partial x}$$

Keep track not only of intermediate function values w_i but also of gradients $\dot{w}_i = \partial w_i / \partial x$

Step 0:	$w_0 = x$	$\dot{w}_0 = 1$
Step 1:	$w_1 = h(w_0)$	$\dot{w}_1 = h'(w_0) * \dot{w}_0$
Step 2:	$w_2 = g(w_1)$	$\dot{w}_2 = g'(w_1) * \dot{w}_1$
Step 3:	$w_3 = f(w_2) = y$	$\dot{w}_3 = f'(w_2) * \dot{w}_2 = dy/dx$

w_i can be a function of multiple predecessors w_j:

$$\dot{w}_i = \sum_{j \in \{\text{predecessors of i}\}} \frac{\partial w_i}{\partial w_j} \dot{w}_j$$

Automatic differentiation: Forward and backward

Forward accumulation:

$$\begin{aligned}\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial w_{n-1}}\frac{\partial w_{n-1}}{\partial x}\\ &= \frac{\partial y}{\partial w_{n-1}}\left(\frac{\partial w_{n-1}}{\partial w_{n-2}}\frac{\partial w_{n-2}}{\partial x}\right)\\ &= \frac{\partial y}{\partial w_{n-1}}\left(\frac{\partial w_{n-1}}{\partial w_{n-2}}\left(\frac{\partial w_{n-2}}{\partial w_{n-3}}\frac{\partial w_{n-3}}{\partial x}\right)\right) = \cdots\end{aligned}$$

$$w_0 = x \qquad \dot{w}_0 = 1$$

$$\dot{w}_i = \sum_{j\in\{\text{predecessors of i}\}} \frac{\partial w_i}{\partial w_j}\dot{w}_j$$

Good for computing derivatives of many functions with respect to single variable

Reverse (adjoint) accumulation:

$$\begin{aligned}\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x}\\ &= \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x}\\ &= \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x}\\ &= \cdots\end{aligned}$$

$$\bar{w}_i = \frac{\partial y}{\partial w_i}$$

$$\bar{w}_i = \sum_{j\in\{\text{successors of i}\}} \bar{w}_j\frac{\partial w_j}{\partial w_i}$$

Good for computing derivatives of a single function with respect to many variables (neural networks)

Automatic differentiation: Implementation

Implementing automatic differentiation proceeds by replacing real numbers by **dual numbers** (value + derivative) and implementing dual number algebra

Python:

- **JAX:** https://docs.jax.dev/
- **MyGrad:** https://mygrad.readthedocs.io/en/latest/
- **TensorFlow, PyTorch, ...**

C++:

- **autodiff:** https://autodiff.github.io/
- **xad:** https://auto-differentiation.github.io/
- **...**

Automatic differentiation: Example

$$f(x) = x^3 - 2x^2 + x - 2$$

$$f'(x) = 3x^2 - 4x + 1$$

```python
def f(x):
    return x**3 - 2*x**2 + x - 1
```

```python
import jax.numpy as jnp
from jax import grad
from jax import jvp

# Autodiff derivative forward mode
def dfdx_auto_forward(func, x):
    x_val = jnp.array(x)
    dx = jnp.array(1.0)
    y, dy = jvp(func, (x_val,), (dx,))
    return dy

# Autodiff derivative reverse mode
def dfdx_auto_reverse(func, x):
    return grad(func)(x)
```

		analytic	forward AD	reverse AD
x	f(x)	df/dx_analyt	df/dx_ad_forw	df/dx_ad_reve
0.0	-1.0000	1.0000	1.0000	1.0000
0.2	-0.8720	0.3200	0.3200	0.3200
0.4	-0.8560	-0.1200	-0.1200	-0.1200
0.6	-0.9040	-0.3200	-0.3200	-0.3200
0.8	-0.9680	-0.2800	-0.2800	-0.2800
1.0	-1.0000	0.0000	0.0000	0.0000
1.2	-0.9520	0.5200	0.5200	0.5200
1.4	-0.7760	1.2800	1.2800	1.2800
1.6	-0.4240	2.2800	2.2800	2.2800
1.8	0.1520	3.5200	3.5200	3.5200
2.0	1.0000	5.0000	5.0000	5.0000
2.2	2.1680	6.7200	6.7200	6.7200
2.4	3.7040	8.6800	8.6800	8.6800
2.6	5.6560	10.8800	10.8800	10.8800
2.8	8.0720	13.3200	13.3200	13.3200

Automatic differentiation: A more involved example

Consider **Dawson function**

$$D_+(x) = e^{-x^2} \int_0^x e^{t^2} dt$$

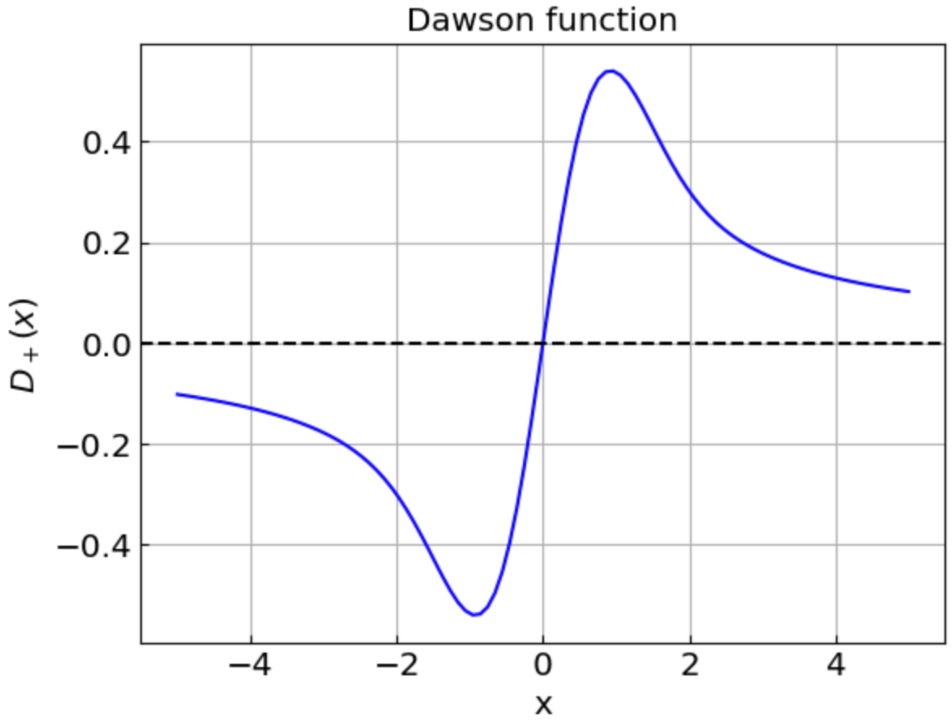
Compute using Gaussian quadrature

```
from IntegrateGauss import *

gaussxw32 = gaussxw(32)
def gaussxwab32(a,b):
    x,w = gaussxw32
    return 0.5*(b-a)*x+0.5*(b+a),0.5*(b-a)*w

from jax.numpy import exp

def DawsonF(x):
  def fint(t):
    return exp(t**2)
  x2 = x**2
  gaussx, gaussw = gaussxwab32(0,x)
  return exp(-x2) * integrate_quadrature(fint, (gaussx, gaussw))
```



Automatic differentiation: A more involved example

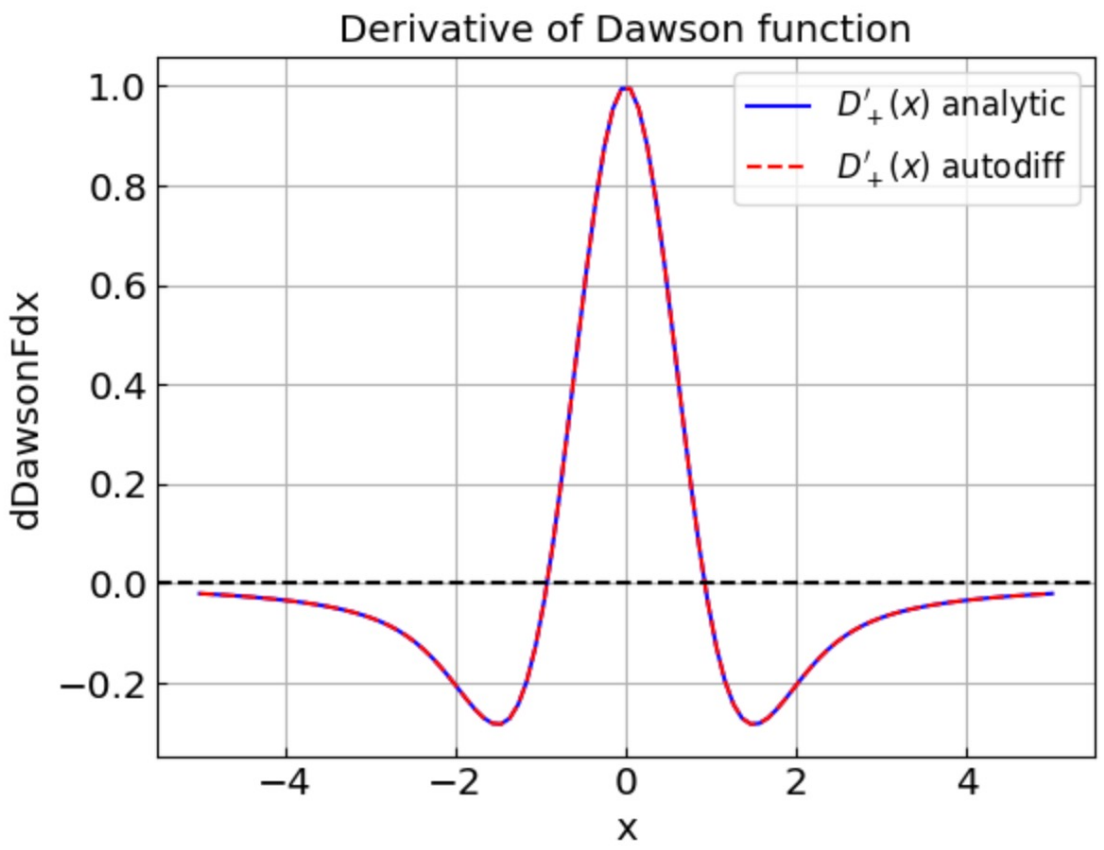
$$D_+(x) = e^{-x^2} \int_0^x e^{t^2}\, dt$$

Compute derivative with AD

```
# Forward mode AD
def dDawsonFdx_auto_forw(x):
    return dfdx_auto_forward(DawsonF, x)
# Reverse mode AD
def dDawsonFdx_auto_reve(x):
    return dfdx_auto_reverse(DawsonF, x)
```

Compare with the expected result

$$D'_+(x) = 1 - 2xD_+(x)$$



We combined numerical integration and automatic differentiation!

Automatic differentiation: Summary

Advantages:

- In theory exact calculation limited only by machine precision and by accuracy of the function calculation itself
- Efficient, often requiring comparable number of operations relative to the original calculation
- Works for implicit functions (such as those computed through Newton-Raphson method)
- Can be extended to high-order derivatives (gradient of a gradient)

Disadvantages:

- Requires adjustments to the existing code
- Can yield unexpected behavior for functions with noise of discontinuities
- Does not work well in the presence of branching [e.g. if bisection or golden section search is used to compute f(x)]