A new solution for computing quick and accurate numerical derivatives

Results from the working paper:
Kostyrka, A. V. (2024). What are you doing, step size:
Fast computation of accurate numerical derivatives with finite precision.

Andreï V. Kostyrka

UNIVERSITÉ DU LUXEMBOURG
Department of Economics
and Management (DEM)

Brown-bag seminar series
Faculty of Law, Economics, and Finance (FDEF)
University of Luxembourg
28th of May 2024

Presentation structure

1. Motivation and empirical applications
2. Approximations of analytical derivatives
3. Error sources in numerical derivatives
4. Approaches to step size selection
5. Showcase of the new package

Motivation and empirical applications

Contribution

I extend the existing software ecosystem and numerical-methods literature by:

1. Creating an open-source R package for fast, parallelised numerical differentiation
    - First open-source parallel Jacobians, Hessians and higher-order-accurate gradients
2. Deriving analytical error bounds and optimal step-size rules for higher-order-accurate derivatives and second-order-accurate Hessians
3. Implementing previously proposed algorithms of step-size estimation, benchmarking their relative performance, and suggesting improved modifications

Motivation and research question

- Researchers rely on optimisers, algorithms, black boxes etc. to 'solve' their models and carry out inference
- The end result is highly dependent on the solver quality
- Most popular modern optimisation techniques use numerical derivatives for minimisation or maximisation

However, most software implementation yield **inaccurate** and **slow** numerical derivatives.

How can we attain the hardware-dependent accuracy bound for numerical derivatives?

Consequences of inaccurate derivatives

- Inexact solutions, values not at the optimum
- Wrong asymptotic-approximation-based inference
  - Wrong standard errors and *p* values in non-linear models
- Worst case: negative Hessian-based variances

Example from a financial application

Simple AR(1)-GARCH(1, 1) model for NASDAQ log-returns, 1990–1994:

$$r_t = \mu + \rho r_{t-1} + \sigma_t U_t, \quad \sigma_t^2 = \omega + \alpha U_{t-1}^2 + \beta \sigma_{t-1}^2$$

Coefficient	Est.	t-stat (rugarch)	t-stat (fGarch)	t-stat (manual)
μ	0.0007	2.34	2.31	2.33
ρ	0.24	7.77	7.73	7.73
$\omega \times 10^3$	0.0098	NaN or 65 (default / fallback)	3.09	3.08
α	0.13	11.1	4.27	4.26
β	0.73	39.6	10.9	11.0

Gradients, Jacobians, Hessians in economics

- Gradient: marginal effects and causal interpretation
  - It is common to numerically estimate the response of Y to a small change X in large systems of interdependent equations
- Hessian: standard errors in semi-parametric and parametric models (non-linear least squares, GMM, maximum likelihood: probit, logit, heckit…)
- Jacobian: must be supplied in constrained-optimisation problems (optimisation subject to $g(\theta) = 0$, $h(\theta) \geq 0$)
- Numerical optimisation with steepest-descent / hill-climbing methods

Necessary in any model that is not linear in parameters.

You have encountered numerical algorithms

12 heckman — Heckman selection model

```
. use https://www.stata-press.com/data/r18/twopart

. heckman yt x1 x2 x3, select(z1 z2) nonrtol

Iteration 0:  Log likelihood = -111.94996
Iteration 1:  Log likelihood = -110.82258
Iteration 2:  Log likelihood = -110.17707
Iteration 3:  Log likelihood = -107.70663  (not concave)
Iteration 4:  Log likelihood = -107.07729  (not concave)
  (output omitted)
Iteration 36: Log likelihood =  -104.0825

Heckman selection model                         Number of obs     =        150
(regression model with sample selection)              Selected    =         63
                                                      Nonselected =         87

                                                Wald chi2(3)      =   8.84e+08
Log likelihood = -104.0825                      Prob > chi2       =     0.0000
```

yt	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
yt						
x1	.8974192	.0002164	4146.52	0.000	.896995	.8978434
x2	-2.525303	.0001244	-2.0e+04	0.000	-2.525546	-2.525059
x3	2.855786	.0002695	1.1e+04	0.000	2.855258	2.856314
_cons	.6975003	.0907873	7.68	0.000	.5195604	.8754402

Existing literature / software

- Gerber & Furrer (2019). `optimParallel`: An R Package Providing a Parallel Version of the L-BFGS-B Optimization Method. *The R Journal* 11 (1). `cran.r-project.org/package=optimParallel`
  - Limited to the built-in `optim(..., method = "L-BFGS-B")`
- Textbooks on linear algebra, calculus, and numerical analysis
- Papers on computer algorithms from the 1970s
- Hong, Mahajan & Nekipelov, (2015, *JoE*). Extremum estimation and numerical derivatives.

Non-existent literature / software

- Most modern articles focus on ultra-high-dimensional numerical gradients with much fewer evaluations
  - Only one (!) paper (Mathur 2012, Ph. D. thesis) with a comprehensive treatment of the classical case useful for low-dimensional models
- Existing algorithms (Curtis & Reid 1974, Dumontet & Vignes 1977, Stepleman & Winarsky 1979) lack open-source implementations
  - Popular software packages implement very rough rules and do not refer to any optimality results in the literature
- Most implementations of higher-order and cross-derivatives are through repeated differencing
  - Slower and less accurate than the best solution

Partial solutions

- R packages *numDeriv* and *optimParallel*
  - *numDeriv*: the most full-featured arsenal in terms of accuracy, but slow; `optimParallel`: speed gains but no focus on accuracy
- Python's *numdifftools*
  - Discusses Richardson extrapolation; no error analysis
- MATLAB's *Optimisation Toolbox*
  - Focuses on parallel evaluation, not accuracy
- Stata's `deriv`
  - Implements a step-size search to obtain 8 accurate digits

Derivatives in linear models

$$FUELSALES = \beta_0 + \beta_1 P_{Lux} + \beta_2 P_{abroad} + \beta_3 COMMUTERS + \beta_4 LOCKDOWN + U$$

- Exogeneity assumption: $\mathbb{E}(U \mid P_{Lux}, P_{abroad}, COMMUTERS, LOCKDOWN) = 0$
- $\frac{\partial}{\partial P_{abroad}} \mathbb{E}[FUELSALES \mid P_{Lux}, P_{abroad}, \ldots] = \beta_2$ by exogeneity
- **Causal interpretation:** if the foreign fuel price changes by 1 €, fuel sales will change by β_2 units *ceteris paribus* (including U)

Derivatives in non-linear models

Economic vulnerability model for women over 50:

$$Y^* = \alpha_0 + \gamma_1 EducYears + \gamma_2 NonWhite + \gamma_3 EducYears \times NonWhite + X'\beta_0 + U := \tilde{X}'\theta_0 + U$$

$$Y := \begin{cases} 1, & Y^* > 0, \\ 0, & Y^* \le 0, \end{cases} \quad \mathbb{P}(Y = 1 \mid \tilde{X}) = F_U(\tilde{X}'\theta_0), \quad U \sim \mathcal{N}, \Lambda, \ldots$$

$$\frac{\partial \mathbb{P}(Y = 1 \mid \tilde{X})}{\partial EducYears} = f_U(\tilde{X}'\theta_0) \cdot (\gamma_1 + \gamma_3\, NonWhite)$$

$$\frac{\partial \mathbb{P}(Y = 1 \mid \tilde{X})}{\partial NonWhite} = f_U(\tilde{X}'\theta_0) \cdot (\gamma_2 + \gamma_3\, EducYears)$$

Inference on γ_3 is not intuitive.

Inference in non-linear models

Policy-makers are interested in the effects due to changes in *explanatory variables*, not parameters.

Average partial effect of the k^{th} variable: $\mathbb{E}\frac{\partial}{\partial X^{(k)}}\mathbb{P}(Y = 1 \mid \tilde X)$.

Its straightforward estimator is $\frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial X^{(k)}}\hat{\mathbb{P}}(Y_i = 1 \mid \tilde X_i)$.

Embarrassingly parallel task: a problem that can be split into smaller problems that can be solved in parallel with no communication between the processes.

- Computing the n-dimensional derivative vector $\left\{\frac{\partial}{\partial X_i^{(k)}}\hat{\mathbb{P}}(Y_i = 1 \mid \tilde X_i)\right\}_{i=1}^{n}$ is embarrassingly parallel
- Inference on θ_0 based on the Hessian of the log-likelihood is embarrassingly parallel

Complications in non-linear models

- F_U is often confined to a specific family (Poisson, exponential, Gaussian, logistic etc.)
  - This parametric assumption could be wrong
  - A more flexible approximation of the true distribution of U may not have a manageable closed-form derivative
- Most data-generating process in economics are highly non-linear and hard-to-formalise
  - Non-linear high-dimensional models tend to have a better explanatory power and yield more accurate forecasts
  - Loss of parameter interpretability
  - Numerical derivatives are often the only solution

Approximations of analytical derivatives

Derivative of a function

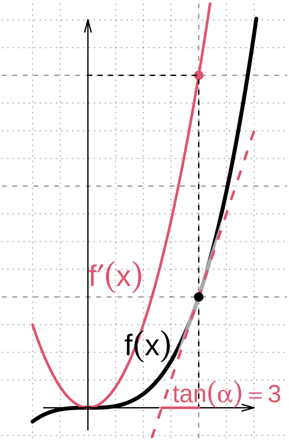
Derivative: The instantaneous rate of change of a function.

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

Assume that f is differentiable and therefore continuous.

$f'(x)$ is the slope of the tangent line to the graph at x.

Illustration: $f(x) := x^3$, $f'(x) = 3x^2$. $f(1) = 1$, $f'(1) = 3$. The tangent equation at $x = 1$ is $3x - 2$.



Naïve numerical derivatives

In the definition

$$f'(x) := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

remove the limit to obtain a **forward difference**:

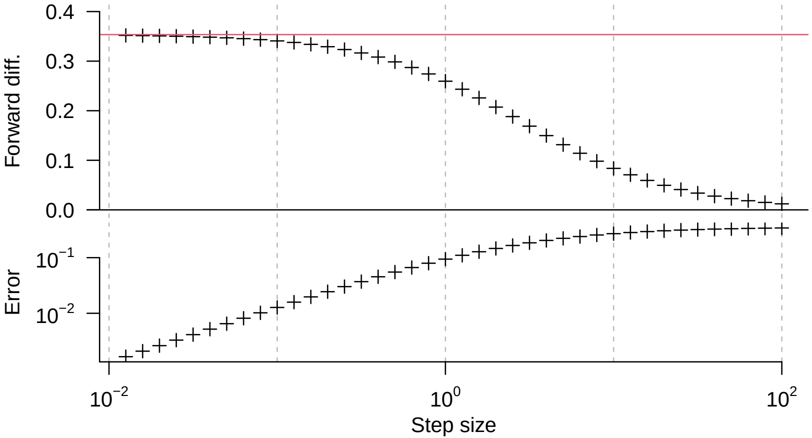
$$f'_{\mathrm{FD}}(x, h) := \frac{f(x+h) - f(x)}{h}$$

Choose a sequence of decreasing step sizes h_i (e. g. $\{0.1, 0.01, 0.001, \ldots\}$), observe the sequence $f'_{\mathrm{FD}}(x, 0.1), f'_{\mathrm{FD}}(x, 0.01), f'_{\mathrm{FD}}(x, 0.001), \ldots$ converge to f'.

We revisit this expression in Slide 25.

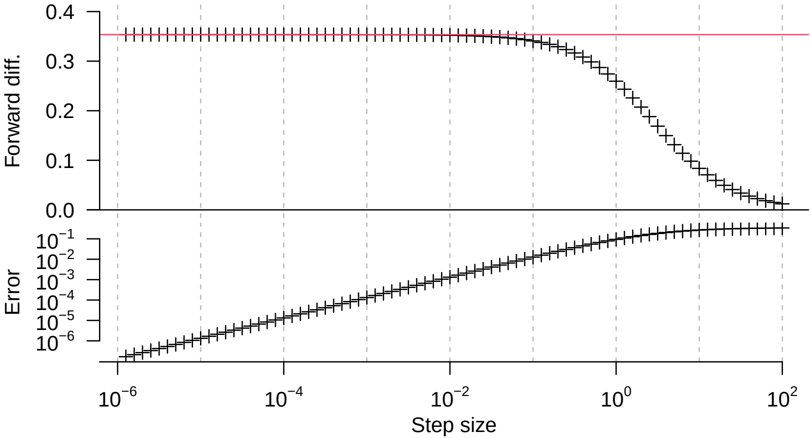
Naïve numerical derivatives in practice

Mathematically, $f'_{\mathrm{FD}}(x, 0.1), f'_{\mathrm{FD}}(x, 0.01), f'_{\mathrm{FD}}(x, 0.001), \ldots$ converges to $f'(x)$.



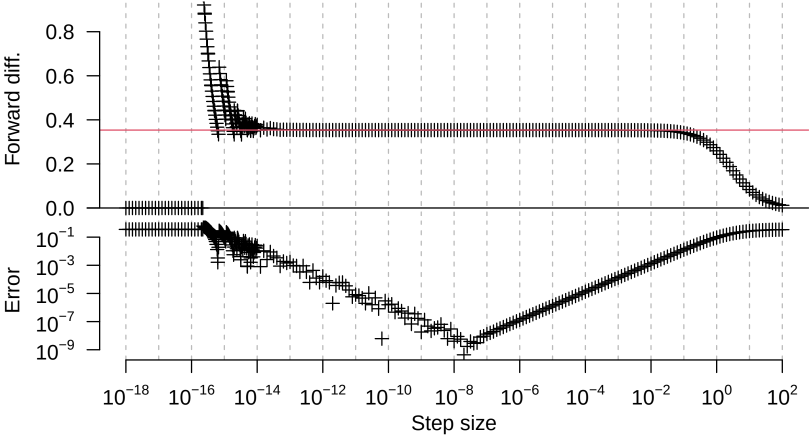
Naïve numerical derivatives in practice

Mathematically, $f'_{\mathrm{FD}}(x, 0.1), f'_{\mathrm{FD}}(x, 0.01), f'_{\mathrm{FD}}(x, 0.001), \ldots$ converges to $f'(x)$.



Naïve numerical derivatives in practice

Mathematically, $f'_{\mathrm{FD}}(x, 0.1), f'_{\mathrm{FD}}(x, 0.01), f'_{\mathrm{FD}}(x, 0.001), \ldots$ converges to $f'(x)$. **But not true in practice!**



Gradient of a function

Gradient: column vector of partial derivatives of a differentiable scalar function.

$$\nabla f(x) := \begin{pmatrix} \frac{\partial f}{\partial x^{(1)}}(x) \\ \vdots \\ \frac{\partial f}{\partial x^{(d)}}(x) \end{pmatrix}$$

- Vector input x + scalar output f = vector ∇
- At any point x, the gradient – the d-dimensional slope – is the **direction and rate of the steepest growth** of f

‘A source of anxiety for non-mathematics students.’
J. Nash, ‘Nonlinear Parameter Optimization’ (2014).

Visualisation of a gradient

Jacobian of a function

Jacobian: Matrix of gradients for a vector-valued function f.

If $\dim x = d$, $\dim f = k$,

$$\nabla f(x) := \left(\frac{\partial f}{\partial x^{(1)}}(x) \quad \cdots \quad \frac{\partial f}{\partial x^{(d)}}(x) \right)_{k \times d} = \begin{pmatrix} \nabla^T f^{(1)}(x) \\ \vdots \\ \nabla^T f^{(k)}(x) \end{pmatrix}_{k \times d}$$

- Vector input x + vector output f = matrix ∇
- In constrained problems, most solvers (e. g. NLopt) for $\min_x f(x)$ s. t. $g(x) = 0$ require an explicit $\nabla g(x)$

  Including incorrectly computed derivatives (mostly gradients or Jacobian matrices) <...> explains almost all the 'failures' of optimisation codes I see. (Idem.)

Hessian of a function

Hessian: Square matrix of second-order partial derivatives of a twice-differentiable scalar function.

$$\nabla^2 f(x) := \left\{ \frac{\partial^2 f}{\partial x^{(i)} \partial x^{(j)}} \right\}_{i,j=1}^{d} = \begin{pmatrix} \frac{\partial^2 f}{\partial x^{(1)} \partial x^{(1)}} & \cdots & \frac{\partial^2 f}{\partial x^{(1)} \partial x^{(d)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x^{(d)} \partial x^{(1)}} & \cdots & \frac{\partial^2 f}{\partial x^{(d)} \partial x^{(d)}} \end{pmatrix} (x)$$

The Hessian is the transpose Jacobian of the gradient:

$$\nabla^2 f(x) = \nabla^T [\nabla f(x)]$$

- Vector input x + scalar output f = matrix ∇^2
- If ∇f is differentiable, ∇_f^2 is symmetric

Taylor series

$$f(x \pm h) = \sum_{i=0}^{\infty} \frac{1}{i!} \frac{\mathrm{d}^i}{\mathrm{d}x^i} f(x) \cdot (\pm h)^i$$
$$= f(x) \pm \frac{f'(x)}{1!} h + \frac{f''(x)}{2!} h^2 \pm \frac{f'''(x)}{3!} h^3 + \ldots$$

The a^{th}-order approximation of f at x is a polynomial of degree a. The discrepancy between f and its approximation is the **remainder**. For some $\delta \in [0, 1]$,

$$f(x \pm h) - \sum_{i=0}^{a} \frac{1}{i!} \frac{d^i f(x)}{dx^i} (\pm h)^i = \frac{f^{(a+1)}(x \pm \delta h)}{(a+1)!} (\pm h)^{a+1}$$

For small h ($h < 1$, $h \to 0$), $h^{a+1} \xrightarrow{a \to \infty} 0$.

Example: Taylor series for CRRA utility

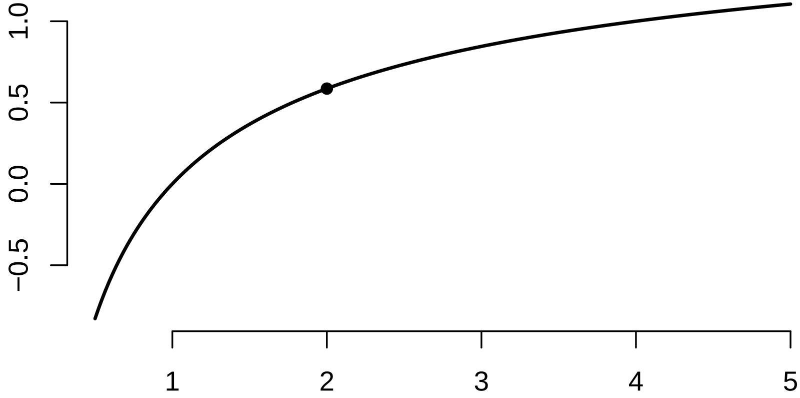
Linear approximation of CRRA utility with risk aversion η:

$$f(x) = \frac{x^{1-\eta}}{1-\eta}, \quad f'(x) = x^{-\eta}, \quad f''(x) = -\eta x^{-\eta-1}, \quad \dots$$

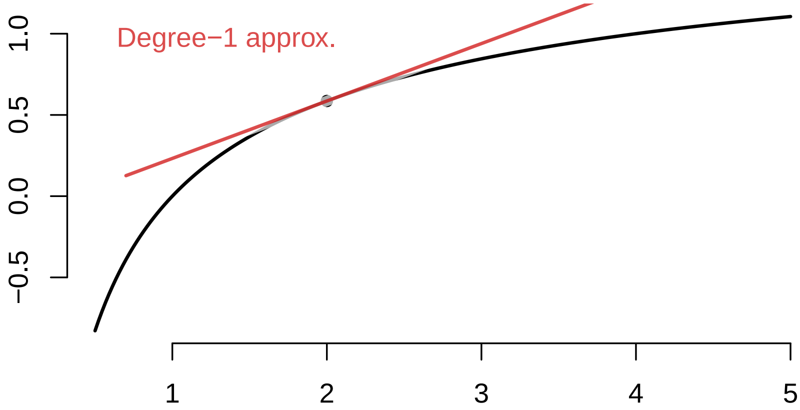
Assume $\eta = 1.5$, approximate f around $x_0 = 2$.

$$\begin{aligned} f(2+h) &\approx f(x_0) + f'(x_0)h = 0.59 + 0.35h = P_1(h) \\ &\approx P_1(h) + \frac{f''(x_0)}{2!}h^2 = 0.59 + 0.35h - 0.13h^2 = P_2(h) \\ &\approx P_2(h) + \frac{f'''(x_0)}{3!}h^3 = 0.59 + 0.35h - 0.27h^2 + 0.06h^3 \\ &\approx 0.59 + 0.35h - 0.27h^2 + 0.06h^3 - 0.02h^4 \approx \dots \end{aligned}$$

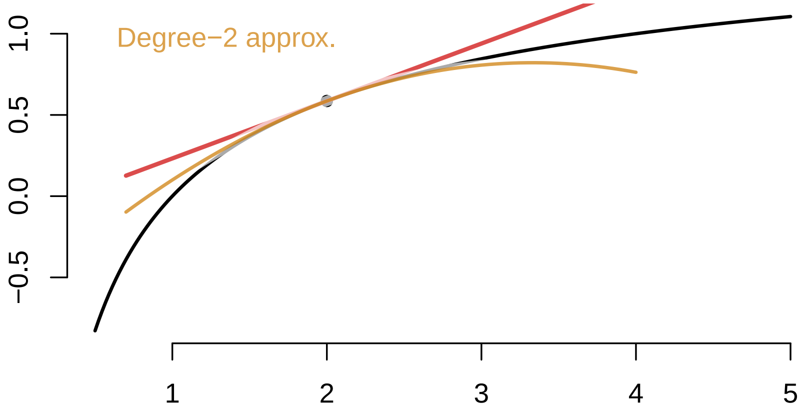
Example: CRRA utility visualisation



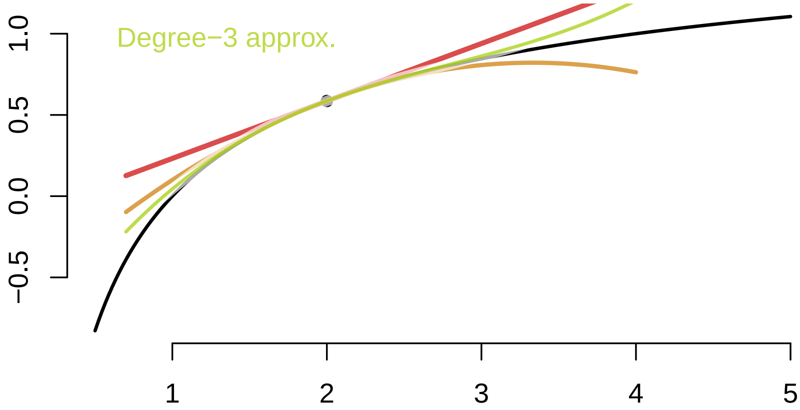
Example: CRRA utility visualisation



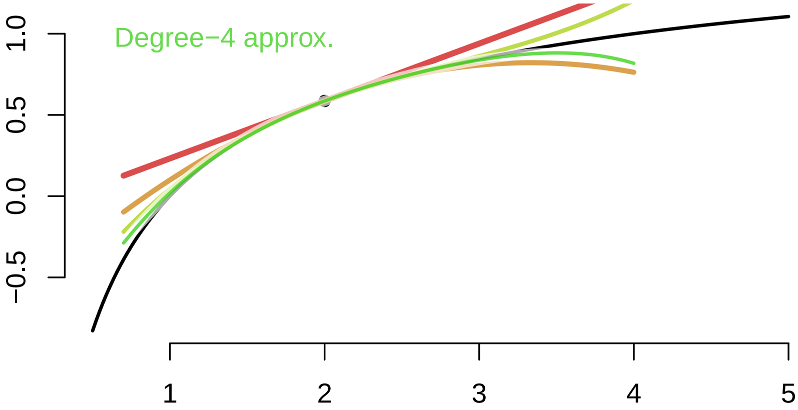
Example: CRRA utility visualisation



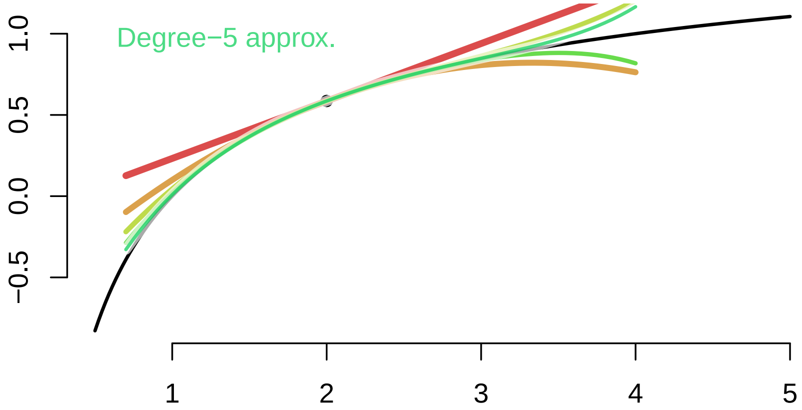
Example: CRRA utility visualisation



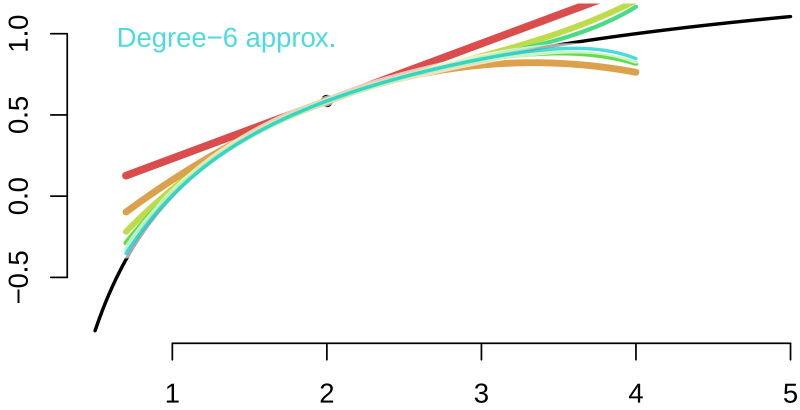
Example: CRRA utility visualisation



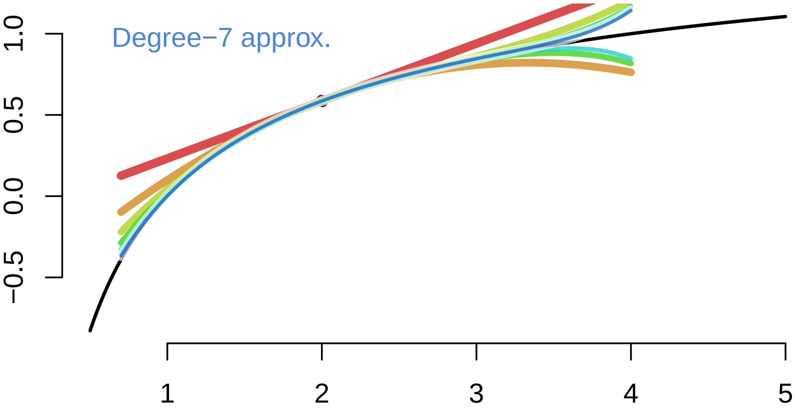
Example: CRRA utility visualisation



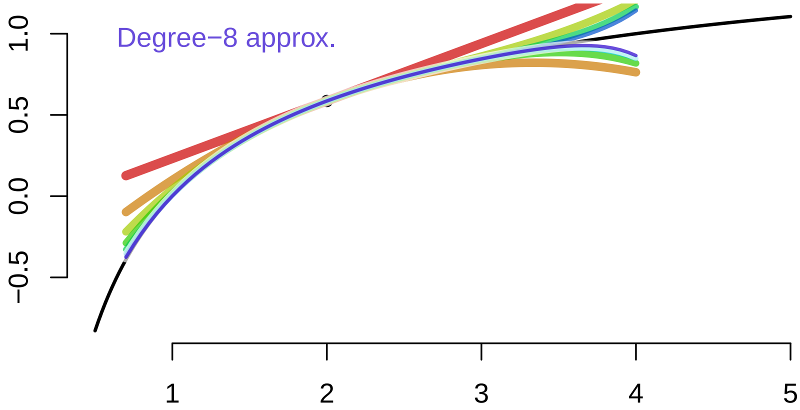
Example: CRRA utility visualisation



Example: CRRA utility visualisation



Example: CRRA utility visualisation



Reversing the Taylor series

- Knowing many derivative values allows one to approximate the function
- Do the opposite: use the function values to approximate any derivative

Derivatives through Taylor series

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x+\alpha h)}{2}h^2, \quad \alpha \in [0,1]$$

Subtract $f(x)$ and divide by h:

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

Therefore, assuming that $f''(x)$ is uniformly bounded*, $f'(x) = f'_{\mathrm{FD}}(x,h) + O(h) \approx f'_{\mathrm{FD}}(x,h) + \frac{f''(x)}{2}h$ (for small h), and $f'_{\mathrm{FD}}(x,h)$ is **first-order-accurate**.

This is the naïve approximation from Slide 15!

* $\exists M > 0\colon \sup_x |f''(x+\alpha h)| \le M < \infty$.

Symmetrical differences

To improve the accuracy, consider expansions at $x \pm h$:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x+\beta_1 h)}{6}h^3, \ \beta_1 \in [0,1]$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x-\beta_2 h)}{6}h^3, \ \beta_2 \in [0,1]$$

Subtract (2) from (1):

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{f'''(x+\beta_1 h) + f'''(x+\beta_2 h)}{6}h^3$$

Divide by $2h$ + generalised intermediate value theorem:

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{f'''(x+\beta h)}{3}h^2, \quad \beta \in [-1,1]$$

Second-order accuracy of derivatives

Central differences are symmetrical around *x*:

$$f'_{\mathrm{CD}}(x, h) := \frac{f(x+h) - f(x-h)}{2h}$$

f'_{CD} is more accurate than f'_{FD}:*

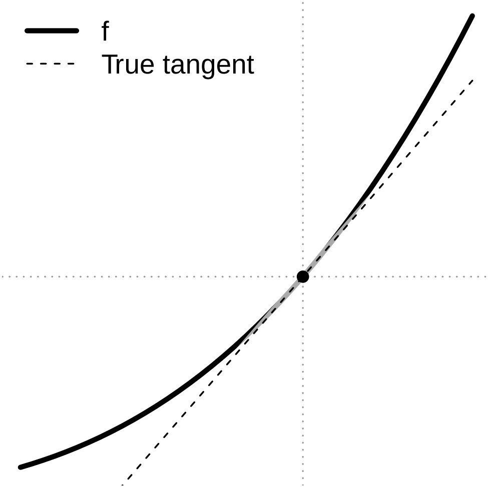
- $f'(x) - f'_{\mathrm{FD}}(x, h) = -\frac{f''(x+\alpha h)}{2} h \approx -\frac{f''(x)}{2} h = O(h)$
- $f'(x) - f'_{\mathrm{CD}}(x, h) = -\frac{f'''(x+\beta h)}{6} h^2 \approx -\frac{f'''(x)}{6} h^2 = O(h^2)$

If $f(x)$ has not been evaluated, computing f'_{FD} and f'_{CD} takes the same amount of time – use f'_{CD}.

If $f(x)$ is already known, CD requires 1 more computation than f'_{FD}, which is 2 times slower – use f'_{FD} for costly f.

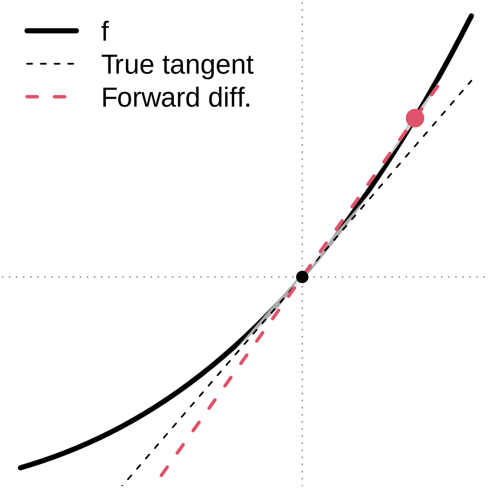
* Assuming f'' and f''' are uniformly bounded.

Graphical illustration of accuracy



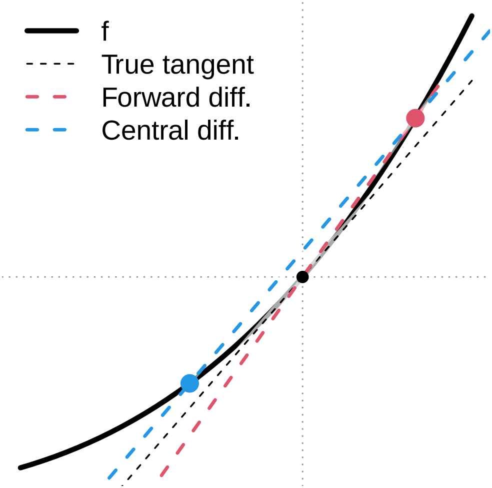
- $f(x) = x^3$, $x_0 = 1$
- $f'(x_0) = 3$

Graphical illustration of accuracy



- $f(x) = x^3$, $x_0 = 1$
- $f'(x_0) = 3$
- Step size $h = 0.2$
- $f'_{FD}(x_0, 0.2) = 3.64$
  Error ≈ 21%

Graphical illustration of accuracy



- $f(x) = x^3$, $x_0 = 1$
- $f'(x_0) = 3$
- Step size $h = 0.2$
- $f'_{FD}(x_0, 0.2) = 3.64$
  Error ≈ 21%
- $f'_{CD}(x_0, 0.2) = 3.04$
  Error ≈ 1.3%

Second derivatives via central differences

$$f''(x) := \frac{\mathrm{d}}{\mathrm{d}x} f'(x)$$

Find such a linear combination of $f(x-h)$, $f(x)$, $f(x+h)$ that the coloured terms should cancel out:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \frac{f''''(x+\gamma_1 h)}{24}h^4$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \frac{f''''(x-\gamma_2 h)}{24}h^4$$

This weighted sum is the solution:

$$f''_{\mathrm{CD}}(x,h) := \frac{f(x-h) - 2f(x) + f(x+h)}{h^2}$$

Accuracy of second derivatives

The error order is the same as with f'_{CD}:

$$f''(x) - f''_{\text{CD}}(x, h) \approx -\frac{f''''(x)}{12} h^2 = O(h^2)$$

However, the default implementation in many software products is **repeated differences**:

$$f''(x) \approx \frac{f'(x+h) + f'(x-h)}{2h} \approx \frac{f'_{\text{CD}}(x+h) + f'_{\text{CD}}(x-h)}{2h}$$

- Approximating $f''(x)$ via a 3-term f''_{CD} is **faster**: each f'_{CD} takes 2 evaluations
- **More accurate** with the optimal step size: the h^* that is optimal for f'_{CD} is too small for f''_{CD} (Slide 55)

Higher-order accuracy of derivatives

Better accuracy is achievable with more terms in the sum. Carefully choose the coefficients to eliminate the undesirable terms:

$$f' = \underbrace{\frac{-f(x-h) + f(x+h)}{2h}}_{f'_{\mathrm{CD},2}} + O(h^2)$$

$$f' = \underbrace{\frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h}}_{f'_{\mathrm{CD},4}} + O(h^4)$$

For the same h, the error of $f'_{\mathrm{CD},4}$ is generally* smaller $\Leftrightarrow$ large h for $f'_{\mathrm{CD},4}$ yields the same error as small h for $f'_{\mathrm{CD},2}$.

General solution

Stencil: strictly increasing sequence of real numbers: $b_1 < \ldots < b_n$. (Preferably symmetric around 0 for the best accuracy.) Example: $b = (-2, -1, 1, 2)$.

Derivatives of any order m with error $O(h^a)$ may be approximated as weighted sums of f evaluated on the **evaluation grid** for that stencil: $x + b_1 h, \ldots, x + b_n h$.

With enough points ($n > m$), one can find such weights $\{w_i\}_{i=1}^n$ that yield the a^{th}-order-accurate approximation of $f^{(m)}$, where $a \le n - m$:

$$\frac{\mathrm{d}^m f}{\mathrm{d}x^m}(x) = h^{-m} \sum_{i=1}^{n} w_i f(x + b_i h) + O(h^a)$$

Examples of stencils and weights

- $f'_{\mathrm{FD}} = \frac{f(x+h)-f(x)}{h} = h^{-1}[-1 \cdot f(x+0h) + 1 \cdot f(x+1h)]$
  - Stencil: $b = (0, 1)$, weights: $w = (-1, 1)$
- $f'_{\mathrm{CD}} = \frac{f(x+h)-f(x-h)}{2h} = h^{-1}[-\frac{1}{2}f(x-h) + \frac{1}{2}f(x+h)]$
  - Stencil: $b = (-1, 1)$ *(symmetric)*, weights: $w = \left(-\frac{1}{2}, \frac{1}{2}\right)$
- $f''_{\mathrm{CD}} = \frac{f(x-h)-2f(x)+f(x+h)}{h^2}$
  - Stencil: $b = (-1, 0, 1)$, weights: $w = \left(1, -2, 1\right)$
- $f'_{\mathrm{CD},4} = \frac{f(x-2h)-8f(x-h)+8f(x+h)-f(x+2h)}{12h}$
  - Stencil: $b = (-2, -1, 1, 2)$, weights: $w = \left(-\frac{1}{12}, \frac{8}{12}, -\frac{8}{12}, \frac{1}{12}\right)$

Example: finite diff. in the new R package

Use `fdCoef()` to obtain the coefficients that yield an approximation of the m^{th} derivative with error $O(h^a)$ on the **smallest sufficient stencil**.

```
fdCoef(deriv.order = 2, acc.order = 4)
# $stencil:   -2        -1           0        1        2
# $weights:  x-2h      x-1h          x      x+1h     x+2h
#         -0.08333  1.33333 -2.50000  1.33333 -0.08333
```

Arbitrary stencils are supported; the resulting coefficients yield the **maximum attainable accuracy**:

```
fdCoef(deriv.order = 1, stencil = c(-1, 0, 4))$weights
#  x-1h     x   x+4h
# -0.80  0.75  0.05  # Second-order accuracy
```

Numerical Hessians via central differences

Let $h_i := (0 \ \dots \ 0 \underbrace{h}_{i^{\text{th}} \text{ position}} 0 \ \dots \ 0)'$ and $x_{+-} := x + h_i - h_j$.

4 evaluations of f are required to approximate $\nabla^2_{ij} f$ via CD:

$$\nabla^2_{ij} f(x) := \left[\nabla^T \big(\nabla f(x)\big)\right]_{ij} := \nabla^2_{ij,\text{CD}} f(x) + O(h^2) =$$

$$= \frac{f(x_{++}) - f(x_{-+}) - f(x_{+-}) + f(x_{--})}{4h^2} + O(h^2)$$

- The 4-term sum is as **fast** as the 4-term $\frac{\nabla_i f(x+h_j) - \nabla_i f(x-h_j)}{2h_j}$, but guaranteed to be **symmetric**: $\hat\nabla^2_{ij,\text{CD}} = \hat\nabla^2_{ji,\text{CD}}$
  - Symmetric repeated differences require 8 terms
- Accuracy implications are being investigated

Efficient parallelisation of gradients

Example: $\nabla f(x)$, $\dim x = 3$, stencil $b = (-2, -1, 1, 2)$ for 4^{th}-order accuracy, same step size h. Total: 12 evaluations.

	$w_1 = \frac{1}{12}$	$w_2 = -\frac{8}{12}$	$w_3 = \frac{8}{12}$	$w_4 = -\frac{1}{12}$
$x^{(1)}$	$f(x - 2h_1)$	$f(x - h_1)$	$f(x + h_1)$	$f(x + 2h_1)$
$x^{(2)}$	$f(x - 2h_2)$	$f(x - h_2)$	$f(x + h_2)$	$f(x + 2h_2)$
$x^{(3)}$	$f(x - 2h_3)$	$f(x - h_3)$	$f(x + h_3)$	$f(x + 2h_3)$

- Create a list of length 12 containing $x + b_j h_i$
- Apply f **in parallel** to the list items, assemble $\left\{\{f(x + b_j h_i)\}_{i=1}^{3}\right\}_{j=1}^{4}$ in a matrix
- Compute weighted row sums

Error sources in numerical derivatives

Floating-point arithmetic

Computers convert inputs into **1**'s and **0**'s for processing.

Real numbers can be written with an **integer** mantissa (=significant digits) and an **integer** exponent (=magnitude):

$$1.8125 = \underbrace{18\,125}_{\text{integer mantissa}} \cdot \underbrace{10}_{\text{base}}{}^{\overbrace{-4}^{\text{integer exponent}}}$$

The number 18.125 has the same mantissa and a different exponent (–3). To multiply by 10 (the base), move the decimal point: $1.8125 \cdot 10 = 18.125$.

Such numbers are called **floating-point numbers**.

Available precision on 64-bit machines

1 bit sign	11 bits exponent	52 bits significant digits
0	10000001101	1100000011010001100110011001100110011001100110011001

Computing the number from bits:

$$(-1)^{\text{sign}} \cdot (1.\text{significand}) \cdot 2^{\text{exponent}-2^{10}+1} =$$

$$= 1.753198 \cdot 2^{1037-1023} = 28\,724.4$$

- 64-bit FP numbers represent $5 \cdot 10^{-324} \;\ldots\; 2 \cdot 10^{308}$
- Are 64-bit calculations relatively accurate up to 10^{-323}?
  No, only to $1/2^{52} = 2.2 \cdot 10^{-16}$**!**
- Precision beyond ≈16 decimal significant digits is lost

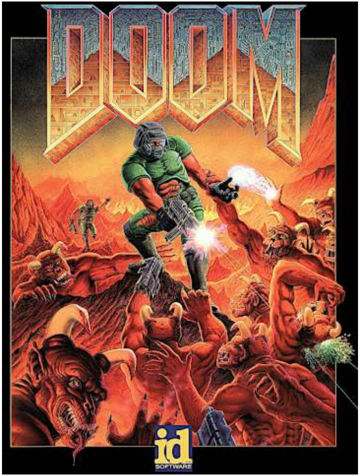
Computers have terrible precision

- **Machine epsilon** (ϵ_m): maximum relative step between two representable numbers, or $\epsilon_m := 2^{-52} \approx 2.2 \cdot 10^{-16}$
  - If $x = 2^i$ for integer i, the mantissa is 52 zeros: 000...000; when the least significant bit is flipped from 0 to 1, the mantissa becomes 000...00**1**, and $x \mapsto (1 + \epsilon_m)x$

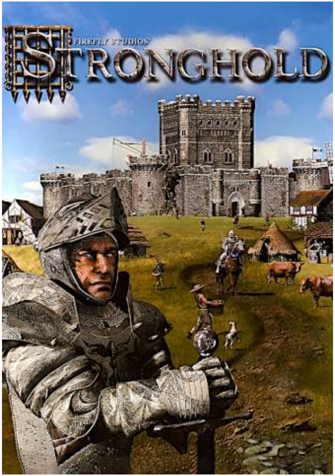
```
lm(formula = mpg ~ disp, data = mtcars)
Coef:        Estimate Std. Error t value Pr(>|t|)
(Intercept) 29.599855   1.229720  24.070  < 2e-16 ***
disp        -0.041215   0.004712  -8.747 9.38e-10 ***
```

- Rounding errors (e. g. if numbers have different orders of magnitude), catastrophic cancellation, ill conditioning (high sensitivity to small input errors)
- Input errors, user mistakes, programmer and hardware bugs – *purgamenta intrant, purgamenta exeunt*

Example: low bit rates in early software

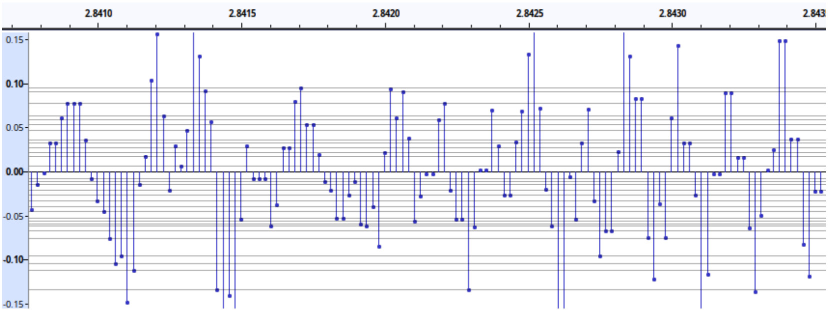


1993, **8-bit** audio,
11 025 Hz sampling



2001, **4-bit** audio,
44 100 Hz sampling

Example: 8-bit audio in the 1990s



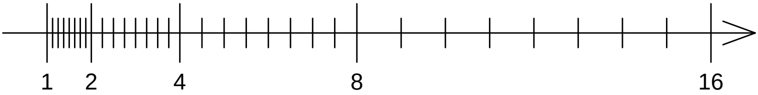
The vertical position of the wave can take any of the $2^8 = 256$ values; 1 point = 1 byte.

11 025 Hz = 11 kilobytes per second of audio.

Finite precision in digital data

- The vertical position of the sound wave intensity is digitally encoded as a number on a fixed grid:
  - 4 bits $\Rightarrow 2^4 = 16$ positions (very coarse)
  - 8 bits $\Rightarrow 2^8 = 256$ positions (coarse)
  - 16 bits $\Rightarrow 2^{16} = 65\,536$ positions (CD quality)
- 64-bit FP numbers use a similar grid to allow $\Rightarrow 2^{64} \approx 1.8 \cdot 10^{19}$ numbers **on the entire real line**
  - The amount of annual Internet traffic is $> 10^{21}$ bytes – already not enough even with positive integers
  - One is limited to 64 bits per number unless they use special libraries for arbitrary-precision arithmetic at the cost of extra memory and speed: GMP, MPFR…

Graphical representation of FP accuracy



- Intervals [1, 2], [2, 4], [4, 8], … are cut into $2^{52} \approx 4.5 \cdot 10^{15}$ equal intervals; all numbers are snapped to the edges
- The gap between two representable numbers is proportional to the number magnitude
  - The rounding error is **proportional** to the number
  - Relative rounding error range: $[0 \ \ldots \ 1.1 \cdot 10^{-16}]$
- Caution: `round(3.5) = 4`, but `round(4.5) = 4` due to rounding towards the nearest *even* number
  - **Worst case:** the 1992 precision loss in the Patriot missile control system ⇒ 28 soldiers died to a Scud missile

Insufficient precision example

```
a = 2^52      # 4 503 599 627 370 496, 1/macheps
b = a + 0.4
c = b + 0.3
d = c + 0.3

d - a         # Question: is equal to what?
```

Answer: **zero**. (At least in FP64 precision.)

- The next number after 2^{52} representable by the machine is $2^{52} + 1$
- Everything less than $2^{52} + 0.5$ is rounded down to 2^{52}
  - Sort the inputs or use Kahan's compensated summation to extend the precision
  - But `2^52+0.3+0.3+0.3+0.3+0.3+0.3+... = 2^52`!
- **Max. rel. error:** $\epsilon_m/2$, **max. abs. error:** $|y| \cdot \epsilon_m/2$

Base-conversion precision loss example

Only finite sums of integer powers of 2 up to 2^{52} are stored losslessly in computer memory:

$1/2 = 0.5_{10} = 0.1_2$ – fine.

$4/5 = 0.8_{10} = 0.1100\,1100\ldots_2 = 0.\overline{1100}_2$ – infinite period.

With 52 bits, one can represent only

$0.\underbrace{[1100]}_{\times 12}1100 = 0.8 - 2 \cdot 10^{-16}$ or

$0.\underbrace{[1100]}_{\times 12}1101 = 0.8 + 4 \cdot 10^{-17}$.

If 0.8 is saved as a number, it is read back as a **different** one: `print(0.8, 20) # 0.80000000000000004441`.

Real case #1: numerical derivative failure

- An economist is modelling some variable Y that is linear in the GDP: $Y := 1 \cdot GDP + g(\dots) + U$
  - $\partial \mathbb{E}(Y \mid \dots)/\partial GDP = 1$, but they use numerical derivatives
- Lux GDP is 80 bn € ⇒ the gap between two representable numbers is $8 \cdot 10^{10} / 2^{52} \approx 1.7 \cdot 10^{-5}$
- Step size: 10^{-8} (from the literature)

$$\nabla_{GDP} Y \Big|_{GDP_{\text{Lux}}} \approx \frac{[8 \cdot 10^{10} + 10^{-8}] - 8 \cdot 10^{10}}{10^{-8}}$$

- $[8 \cdot 10^{10} + 10^{-8}] = 8 \cdot 10^{10}$ because $10^{-8} < \frac{1}{2} \cdot 1.7 \cdot 10^{-5}$ ⇒ the numerator is zero (cf. Slide 16 plot)
  - Error: the computer returns $\widehat{\partial Y / \partial GDP} = 0$ instead of 1!

Real case #2: catastrophic cancellation

The causal effect of a 1-euro debt change on the probability of self-reported good health condition (GH) in the probit model $\mathbb{P}(GH = 1 \mid Debt, \dots) = \Phi(\gamma_0 Debt + \dots)$:

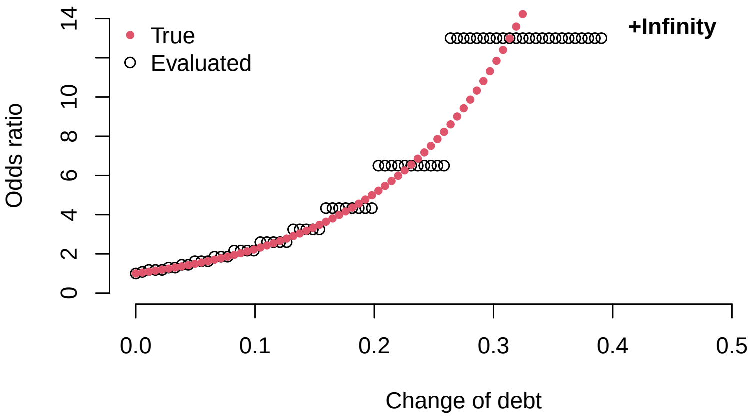
$$\frac{\partial \mathbb{P}(GH_i = 1)}{\partial Debt_i} \approx \frac{\Phi\big(\hat\gamma(Debt_i + 0.001) + \dots\big) - \Phi(\hat\gamma Debt_i + \dots)}{0.001}$$

If the argument of $\Phi(\cdot)$ is too large, probabilities close to 1 are predicted. If $\hat\gamma \cdot Debt_i + \dots = 8.3$, the relative error of $\frac{\partial \mathbb{P}(GH_i=0)}{\partial Debt_i}$ can be ≈ 17%.

Consequence: the error of the odds ratio is unbounded.

Illustration of catastrophic cancellation

Evaluated odds ratio $\frac{\mathbb{P}(GoodHealth_i=0|Debt_i)}{\mathbb{P}(GoodHealth_i=0|Debt_i+\text{change})}$.



Probit breaks at $X'\beta$ = 8.3; logit breaks at $X'\beta$ = 36.8.

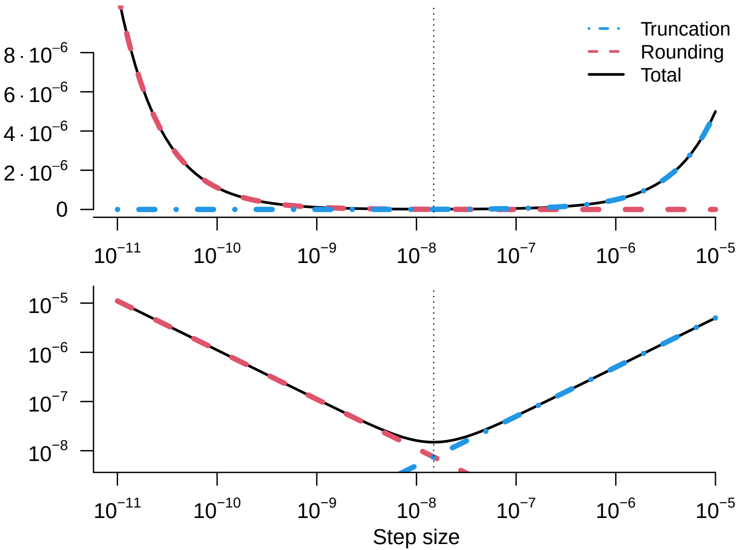
Total error in numerical derivatives

Step size selection is critical for accuracy:

- h too large → large **truncation error** from the truncated Taylor-series term (poor **mathematical** approximation)
- h too small → large **rounding error** (poor **numerical** approximation): catastrophic cancellation, division of something small by something small, machine accuracy always limited by ϵ_{m}

Finding the optimal h^* to balance these two errors is possible.

Visualisation of the error components



Total error function properties

On the log-log scale,

- The slope of the left branch is the differentiation order m (times -1)
  - The rounding error of the difference is divided by h^m
- The slope of the right branch is the accuracy order a
  - The truncation error is approximately $f''^{\cdots}/a!$ times h^a

Analytical error bounds for central diff.

Computing f results in a rounding error:

$$f(x+h) := \hat{f}_{\text{FP64}}(x+h) + e_+, \quad f(x-h) := \hat{f}_{\text{FP64}}(x-h) + e_-$$

$$\underbrace{\left[f(x+h) - f(x-h)\right]}_{\text{true difference}} - \underbrace{\left[\hat{f}_{\text{FP64}}(x+h) - \hat{f}_{\text{FP64}}(x-h)\right]}_{\text{computer evaluation}} = e_+ - e_-$$

$$\underbrace{f'(x) - \hat{f}'_{\text{CD}}(x,h)}_{\text{overall num. deriv. error}} \approx \underbrace{\frac{f'''(x)}{6}h^2}_{\text{truncation}} + \underbrace{\frac{0.5(e_+ - e_-)}{h}}_{\text{rounding}}$$

Rounding-error numerator bound:*

$$|e_+ - e_-| \le |e_+| + |e_-| \le 2\max\{|e_+|, |e_-|\} = |f(x)|\epsilon_{\text{m}}$$

* $f(x+h)$, $f(x-h)$ must have the same magnitude (binary exponent).

Optimal step size

Total-error function: conservative absolute bound (after several harmless simplifications).

$$E_{\mathrm{CD}}(x,h) := \frac{|f'''(x)|}{6} h^2 + 0.5|f(x)|\epsilon_{\mathrm{m}} h^{-1}$$

$$E_{\mathrm{FD}}(x,h) := \frac{|f''(x)|}{2} h + |f(x)|\epsilon_{\mathrm{m}} h^{-1}$$

Optimal step sizes that minimise it:

$$h^*_{\mathrm{CD}} = \sqrt[3]{\frac{1.5|f(x)|}{|f'''(x)|}\epsilon_{\mathrm{m}}}, \qquad h^*_{\mathrm{FD}} = \sqrt{\frac{2|f(x)|}{|f''(x)|}\epsilon_{\mathrm{m}}}$$

Therefore, $h^*_{\mathrm{CD}} \propto \epsilon_{\mathrm{m}}^{1/3}$ and $h^*_{\mathrm{FD}} \propto \epsilon_{\mathrm{m}}^{1/2}$ (machine-dependent).

General step-size selection

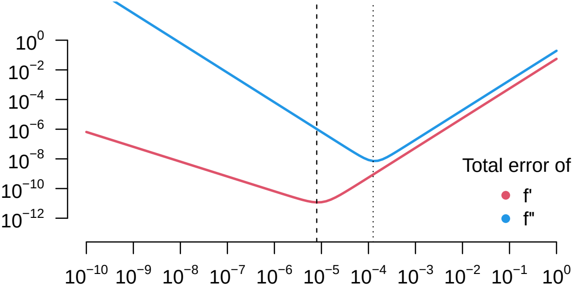
Result: a^{th}-order-accurate m^{th} numerical derivatives have:

- Optimal step size $h_* \propto \sqrt[a+m]{\epsilon_{\mathrm{m}}}$
- Approximation error $\propto \epsilon_{\mathrm{m}}^{a/(a+m)} \propto h_*^a \propto \epsilon_{\mathrm{m}}/h_*^m$ with equal order of truncation and rounding components
  - The total error at the optimal h^* is $O(\epsilon_{\mathrm{m}}^{1/2})$ for one-sided and $O(\epsilon_{\mathrm{m}}^{2/3})$ for central differences
  - In 64-bit precision, f'_{FD} is accurate only to ≈7–8 decimal digits, and f'_{CD} to ≈10–11 digits **at most**
  - Second derivatives and Hessians: $h_{\mathrm{CD}}^{**} \propto \epsilon_{\mathrm{m}}^{1/4}$
  - 4^{th}-order-accurate CD: $h_{\mathrm{CD},4}^{*} \propto \epsilon_{\mathrm{m}}^{1/5}$ (≈12–13 digits)
- Hard limit: impossible to have > 16 accurate decimal places on 64-bit machines without extra effort

Is repeated differencing dangerous?

Options for $f''(x)$: $\frac{f(x-h)-2x+f(x+h)}{h^2}$ or $\frac{f'_{\mathrm{CD}}(x+h)-f'_{\mathrm{CD}}(x-h)}{2h}$.

Surprisingly, both have the same maximum attainable accuracy, $O(\epsilon_{\mathrm{m}}^{1/2})$ (7–8 digits). However, using $h^*_{\mathrm{CD}} \propto \epsilon_{\mathrm{m}}^{1/3}$ results in an $O(\epsilon_{\mathrm{m}}^{1/3})$ error, i. e. only 5–6 accurate digits!



Rule of thumb: multiply h^*_{CD} by $\epsilon_{\mathrm{m}}^{1/4}/\epsilon_{\mathrm{m}}^{1/3} \approx 20$.

Approaches to step size selection

Paradigms for step-size search

1. Theoretical (plug-in expressions)
2. Empirical (finding the minimum of the total error)

My package, pnd, provides multiple algorithms **(currently under active feature implementation and testing)**.

Analogy: Silverman’s rule-of-thumb bandwidth vs. data-driven cross-validated bandwidth in non-parametric econometrics.

Using plug-in higher-order estimates

Since the optimal h^* for f'_{CD} depends on the true f''',

1. Compute $f'''_{\mathrm{CD}}(x, \tilde{h})$ using any reasonable $\tilde{h} \propto \epsilon_{\mathrm{m}}^{1/5}$ (e. g. naïve values 0.001 or 0.001x)
2. Compute $\hat{h}^*_{\mathrm{CD}} = \sqrt[3]{1.5|f(x)|\epsilon_{\mathrm{m}} / |f'''_{\mathrm{CD}}(x, \tilde{h})|}$

- Dumontet–Vignes (1977) proposed an iterative search algorithm for a reliable $\tilde{h}$
- Works for any orders m and a: take $\tilde{h} \propto \epsilon_{\mathrm{m}}^{1/(a+m)}$
- Reassemble the available values of f on $(\pm h, \pm 2h)$ into a 4^{th}-order-accurate $f'_{\mathrm{CD},4}$

```
Grad(func = CRRA, x = 2, h = "plugin", h0=0.01)
Grad(func = CRRA, x = 2, h = "DV")
```

Controlling the error ratio

Curtis & Reid (1974) proposed choosing such h that

$$\frac{\text{truncation error } e_\text{t}}{\text{rounding error } e_\text{r}} \in [10, 1000] \qquad \text{(aim for 100)}$$

Estimate the truncation and rounding errors separately:

- $\hat e_\text{t}(x, h) = |f'_\text{CD}(x, h) - f'_\text{FD}(x, h)|$
  - $\hat e_\text{t} = O(h)$ is too conservative because $e_\text{t} = O(h^2)$
- $\hat e_\text{r}(x, h) = \frac{0.5|f(x)|\epsilon_\text{m}}{h}$

Since $\hat e_\text{t}$ is over-estimated, this aim ensures that $e_\text{t} \approx e_\text{r}$.

```
Grad(func = CRRA, x = 2, h = "CR")
```

Controlling the error ratio, improved

The Curtis & Reid (1974) approach can be improved:

- Larger stencil + parallel evaluations = more accurate truncation estimate
  - $3 \to 4$ evaluations; modern machines have 4+ cores
  - I propose and solve a system of equations for better estimates of e_t with 4 or more evaluations
  - Eliminates the need for the *ad hoc* inflated target
- For f'_{CD}, e_t / e_r at the optimum is 0.5, not 1
- With 4 evaluations, f'_{CD} can be computed from existing values $\Rightarrow$ multiply the aim by $\epsilon_m^{-2/15} \approx 120$

```
Grad(func = CRRA, x = 2, h = "CRm")
Grad(func = CRRA, x = 2, h = "CRm", acc.order = 4)
```

Controlling the truncation-branch slope

Stepleman & Winarsky (1979) and Mathur (2012) proposed similar algorithms based on the idea of descending down the right slope of the estimated truncation error:

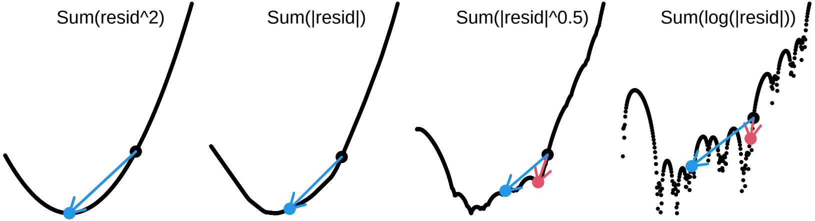
- The slope of the right branch of the total error is a
- Choose a large enough h_0, set $h_1 = 0.5 h_0$, get the truncation error estimate $\hat{e}_t(x) = \frac{f'_{CD}(x, h_1) - f'_{CD}(x, h_0)}{1 - 0.5^2}$
- Continue shrinking while the slope of $\hat{e}_t$ is $\approx a$, stop when it deviates due to the substantial round-off error
  - Never deals with the indeterminable round-off

Noisy functions

Noisy function: many local optima and strong abrupt changes of curvature.

In optimisation, accurate derivatives of noisy function are useless (local features obscure global optima).

Although $h^*_{\mathrm{CD}} = \sqrt[3]{1.5|f / f'''|\epsilon_{\mathrm{m}}} \propto 1/f'''$, use **larger** step sizes to guess a better trend.



Relative or absolute step?

- The optimal step size, $h^*_{\mathrm{CD}} = \sqrt[3]{\epsilon_{\mathrm{m}} \cdot 1.5|f(x)/f'''(x)|}$, depends on the value of x only through $f(x)/f'''(x)$
- However, **relative step** $x \cdot h^*_{\mathrm{CD}}$ is often used to eliminate the problems of **units of measurement** for large $|x|$
  - If $x = 10^{12}$ and $\tilde{h} = 10^{-4}$, **argument-representation errors** appear: $|[x + \tilde{h}]_{\mathrm{FP64}} - (x + \tilde{h})| = 2 \cdot 10^{-5} \neq 0$ (Slide 44)
  - If $x = 10^{-5}$ and $\tilde{h} = 10^{-4}$, $x - \tilde{h} < 0$; bad if $\operatorname{dom} f = \mathbb{R}^{++}$: $\log x$, $\sqrt{x}$... (Slide 4)
- The magnitude of x may be informative of the curvature change, $f'''(x)$
- Common practice: choose $x_{\min} = 10^{-5}$; for $|x| < x_{\min}$, use step size $\tilde{h}$ and for $|x| \geq x_{\min}$, use step size $|x|\tilde{h}$
  - Helps only with large x, not small x such that $|f'''(x)| \gg 0$

Showcase of the new package

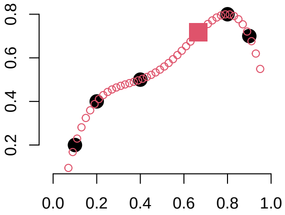
Finding approximations via interpolation

To calibrate η, you run thousands of simulations and compute the goodness of fit $f(\eta)$. You get $\eta = (0.1, 0.2, 0.4, 0.8, 0.9)$, $f(\eta) = (0.2, 0.4, 0.5, 0.8, 0.7)$, but you want to guess f and f' around $\eta_0 = 2/3$.

```
fdCoef(0, stencil = (n - n0))$weights %*% f
fdCoef(1, stencil = (n - n0))$weights %*% f
```

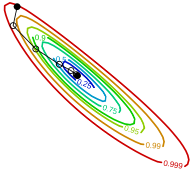
Weights for f: $(0.23, -0.56, 0.69, 0.98, -0.34) \Rightarrow f(2/3) \approx 0.71$.

Weights for f': $(-1.36, 3.51, -5.40, 3.30, -0.05) \Rightarrow f'(2/3) \approx 1.04$.



Computing gradients for optimisation

Smoothed empirical likelihood with missing endogenous variables (Cosma, Kostyrka, Tripathi, 2024). Problem: maximising SEL + computing ∇^2-based std. errors via BFGS on 4 CPU cores.



Method	Ord.	Time, s	$\|\nabla \text{SEL}\|$	Evals	Iters
built-in	2	21+3.8	$3.6 \cdot 10^{-4}$	46	10
new	2	13+1.5	$2.1 \cdot 10^{-7}$	37	10
new	4	16+2.9	$3.3 \cdot 10^{-8}$	32	10

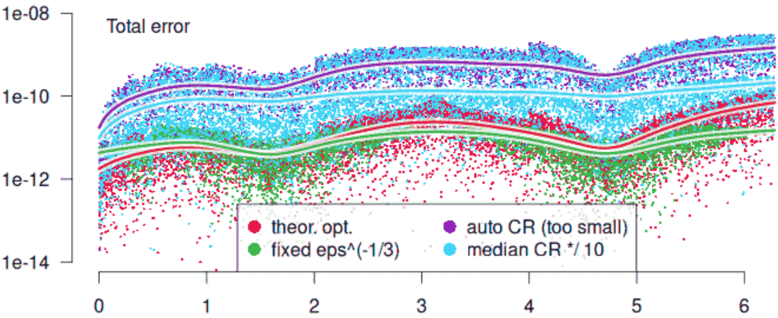
```
g4 <- function(x) pnd::Grad(SEL, x = x,
        acc.order = 4, cores = 4)
optim(par = c(1, 1), SEL, gr = g4, method = "BFGS")
```

Sensitivity of the error to the step size

Choosing a *slightly* sub-optimal step size is not as scary.

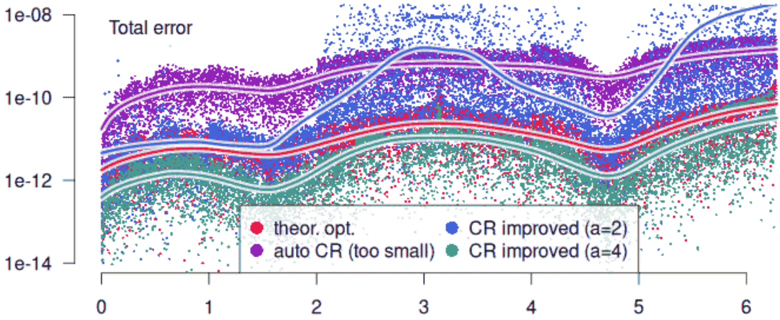
Example: for sin x, the optimal step size is $\propto$ tan x, which is unbounded – a fixed small h can work fine, too.

Or simply take the median of $f'_{\text{CD}}(x, \cdot)$ with $h \cdot 0.1, h, h \cdot 10$.



Improvements for the CR algorithm

1. Estimate the correct truncation error order with 4 parallel evaluations and using the correct target ratio
2. Obtain $f'_{\text{CD},4}$ with algorithmically chosen $h^*_{\text{CD},2}$ times 120
    - ≈ 3 times more accurate than theoretical



Demonstrations for another time

- Computing marginal effects in highly non-linear computationally heavy models with big data
- Computing accurate standard errors in conditional-volatility models (no more `NaN` in GARCH!)
- Choosing the optimal step size for complex multi-dimensional maximisation
- Handling f that are not accurate to the last digit

Practical recommendations

Do not:

- Trust the built-in numerical differences
  - Especially the step size
- Fix $h = 0.01$ because it 'feels right' / you interpret a 1-cent change
- Use FD when evaluating f is fast
- Believe that computers cannot be arbitrarily wrong

Do:

- Use all CPU cores
- Use optimal-step search or simply $h = \epsilon_m^{1/(a+m)}$
- For higher m, increase a to have the error $O(\sqrt{\epsilon_m})$
- Start gradient-based optimisation with a parallel CD2 gradient, restart with CD4
  - If no change, retrace towards x_{start} a bit

Further work

- Finish the formal part, test the suggested algorithm improvements
- Test the default parameters and upload the R package to CRAN as pnd
  - Currently under actively development on github.com/Fifis/pnd
- Add memoisation to reuse the cached evaluations in optimisation routines
- Add facilities to compute higher-order-accurate derivatives from previous candidate step sizes
- Implement complex derivatives