MATH3511/6111: Scientific Computing

05. Numerical Differentiation

Lindon Roberts

Semester 1, 2022

Office: Hanna Neumann Building #145, Room 4.87
Email: lindon.roberts@anu.edu.au

Based on lecture notes written by S. Roberts, L. Stals, Q. Jin, M. Hegland, K. Duru.

Motivation

Basic Problem

If we can only evaluate $f(x)$, how can we approximate $f'(x)$?

Why is this useful?

- We have already seen algorithms which require $f'(x)$: Newton's method for rootfinding, Hermite interpolation.
- Also an essential part of solving differential equations and optimisation (maximise/minimise a function by solving $f'(x) = 0$).

Why not just differentiate $f(x)$ by hand (or using Wolfram Alpha)?

- Good idea if $f(x)$ is simple, but can make mistakes for complicated $f(x)$
- Sometimes don't have an explicit formula for $f(x)$ (e.g. result of a simulation)

Motivation

The basic approach for approximating derivatives goes back to the definition:

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

Based on this, if h is close to zero, we would expect

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

This is a good first attempt, but we will study some more accurate approximations.

Truncation Error

What is the error in this approximation?

Taylor series:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2,$$

for some ξ between x and $x+h$. Therefore,

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{f''(\xi)}{2}h \approx f'(x) + \frac{f''(x)}{2}h.$$

So as $h \to 0$, the truncation error is

$$\left| f'(x) - \frac{f(x+h)-f(x)}{h} \right| \approx \left| \frac{f''(x)}{2}h \right| = \mathcal{O}(h) \to 0.$$

This is called the truncation error as the error comes from truncating the Taylor series for $f(x)$.

Reminder: Big-O Notation

Definition

We say $f(x) = \mathcal{O}(g(x))$ as $x \to a$ if there exists $\delta > 0$ and $C > 0$ such that

$$|f(x)| \leq Cg(x), \qquad \text{for all } x \in [a-\delta, a+\delta].$$

Alternatively, we say $f(x) = \mathcal{O}(g(x))$ as $x \to \infty$ if there exists $M > 0$ and $C > 0$ such that

$$|f(x)| \leq Cg(x), \qquad \text{for all } x \geq M.$$

For example, $8x^2 = \mathcal{O}(x)$ as $x \to 0$, and $3x^2 + 2x + \log x = \mathcal{O}(x^2)$ as $x \to \infty$.

Rough idea: "$f \leq g$, ignoring constants".

Little-o notation: $f(x) = o(g(x))$ as $x \to a$ if $\lim_{x \to a} \frac{f(x)}{g(x)} = 0$.

Roughly, "$f < g$, ignoring constants". For example, $8x^2 = o(x)$ as $x \to 0$,
$3x^2 + 2x + \log x = o(x^3)$ as $x \to \infty$.

Example

Using this method, estimate $f'(1)$ for $f(x) = x \sin x$ for different values of h.

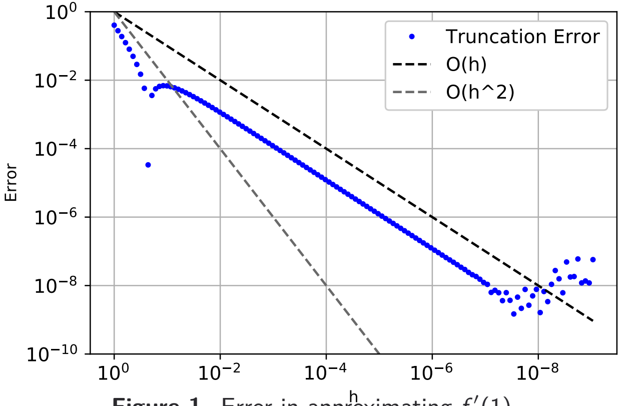

Figure 1. Error in approximating $f'(1)$.

Error decreases like $\mathcal{O}(h)$ until $h \approx 10^{-7}$, then it increases again! This is due to roundoff error.

Forward and Backward Differences

Using the Taylor series

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \mathcal{O}(h^4), \\ f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \mathcal{O}(h^4), \end{aligned}$$

we get the forward difference formula (same as before)

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

and the backward difference formula

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

In both cases, the truncation error is $\mathcal{O}(h)$, so these are called first-order methods.

Central Differences

We can be more clever. Subtracting the two Taylor series gives

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{f'''(x)}{3}h^3 + \mathcal{O}(h^4),$$

and the $f''(x)h^2$ terms have cancelled!

This gives the central difference formula

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

which is a second-order method.

Central Differences

Comparing forward, backward and central differences for estimating $f'(1)$ with $f(x) = x \sin x$.

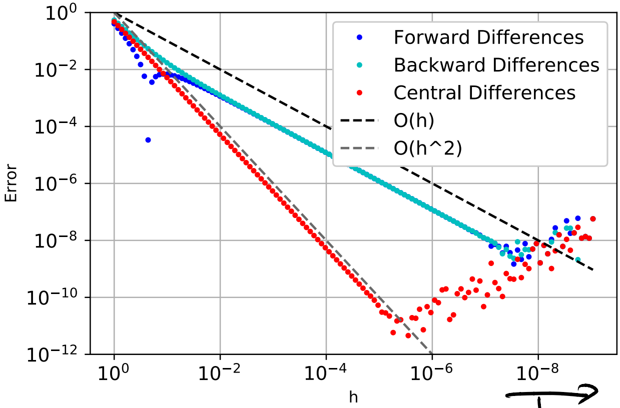

Figure 2. Error in approximating $f'(1)$.

$h \to 0$

choose h
related to
machine ε

Higher Derivatives

If we want to approximate higher derivatives, we need to take larger Taylor series expansions:

$$\begin{aligned}
f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \frac{f''''(x)}{24}h^4 + \mathcal{O}(h^5), \\
f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \frac{f''''(x)}{24}h^4 + \mathcal{O}(h^5),
\end{aligned}$$

Now, we add both equations (to cancel the $f'(x)$ terms), leaving

$$f(x+h) + f(x-h) = 2f(x) + f''(x)h^2 + \frac{f''''(x)}{12}h^4 + \mathcal{O}(h^5).$$

Rearranging, we get a second-order approximation for $f''(x)$:

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

Method of Undetermined Coefficients

To approximate $f'(x)$ or $f''(x)$ using these formulae, we need to evaluate f at $x-h$, x and/or $x+h$.

If evaluating f is slow (e.g. running a simulation), we might want to only use evaluations of f that we already have.

New Problem

Given some evaluations $f(x_0), \ldots, f(x_n)$, how can we approximate $f'(x_0)$ using only this data?

(we choose to approximate $f'(x_0)$ but the same idea applies to approximating $f'(x)$ at other x_i)

Our method is to approximate $f(x)$ by a polynomial $p_n(x)$, and calculate $p_n'(x_0)$.

Method of Undetermined Coefficients

Given nodes $x_0, \ldots, x_n$, we know the unique polynomial of degree (at most) n which interpolates $f(x)$ at these nodes is (using the Lagrange form)

$$f(x) \approx p_n(x) = f(x_0)L_0(x) + \cdots + f(x_n)L_n(x).$$

Therefore, we take our approximation to be

$$f'(x_0) \approx p_n'(x_0) = f(x_0)L_0'(x_0) + \cdots + f(x_n)L_n'(x_0).$$

However, we can calculate $p_n'(x_0)$ without finding the full interpolant $p_n(x)$! We write

$$\boxed{f'(x_0) \approx a_0 f(x_0) + \cdots a_n f(x_n)}$$

dont depend on f

where $a_i = L_i'(x_0)$, and try to calculate the a_i directly. This is called the method of undetermined coefficients.

Method of Undetermined Coefficients

How to calculate a_i directly?

Key idea: if $f(x) = p(x)$ is a polynomial of degree $\leq n$, then the interpolation is exact

$$p(x) = p_n(x). \quad \Rightarrow \text{ same derivative}$$

Therefore the derivative approximation is also exact:

derive approx.

$$p'(x_0) = p_n'(x_0) = a_0 p(x_0) + \cdots + a_n p(x_n).$$

If we choose any $n+1$ polynomials $p(x)$ of degree $\leq n$, this gives $n+1$ linear equations with unknowns $a_0, \ldots, a_n$. n+1 unknowns

$1, x, x^2, \ldots \rightarrow$ basis for polynomials

It will be easiest to choose the $n+1$ polynomials to have increasing degrees and have $\underline{p(x_0) = 0}$: except 1

$$p(x) = 1, \quad p(x) = x - x_0, \quad p(x) = (x - x_0)^2, \ldots$$

Since these form a basis for all polynomials of degree $\leq n$, the linear system will have unique solution.

Method of Undetermined Coefficients: Example

Example

Find an approximation

$$f'(x_0) \approx a_0 f(x_0) + a_1 f(x_1) + a_2 f(x_2) + a_3 f(x_3),$$

where $x_i = x_0 + ih$ for some grid size $h > 0$.

We choose $a_0, \ldots, a_3$ so that the approximation is exact for the polynomials

$$f_0(x) = 1, \quad f_1(x) = x - x_0, \quad f_2(x) = (x - x_0)^2, \quad f_3(x) = (x - x_0)^3,$$

which together form a basis for all cubic polynomials.

Method of Undetermined Coefficients: Example

Therefore, we require $a_0, \dots, a_3$ such that

$$f_i'(x_0) = a_0 f_i(x_0) + a_1 f_i(\underset{x_0+h}{x_1}) + a_2 f_i(\underset{x_0+2h}{x_2}) + a_3 f_i(\underset{x_0+3h}{x_3}), \qquad \text{for all } i = 0, \dots, 3.$$

(note we have $=$ not $\approx$). This gives the equations

$$\begin{aligned}
f_0(x):& \qquad 0 = a_0 + a_1 + a_2 + a_3, \\
f_1(x):& \qquad 1 = ha_1 + 2ha_2 + 3ha_3, \\
f_2(x):& \qquad 0 = h^2 a_1 + 4h^2 a_2 + 9h^2 a_3, \\
f_3(x):& \qquad 0 = h^3 a_1 + 8h^3 a_2 + 27h^3 a_3.
\end{aligned}$$

Order?

This linear system has the unique solution

$$a_0 = \frac{-11}{6h}, \quad a_1 = \frac{3}{h}, \quad a_2 = \frac{-3}{2h}, \quad a_3 = \frac{1}{3h}.$$

Taylor Series

So our approximation is

$$f'(x_0) \approx -\frac{11}{6h} f(x_0) + \frac{3}{h} f(x_1) - \frac{3}{2h} f(x_2) + \frac{1}{3h} f(x_3).$$

Method of Undetermined Coefficients: Example

Comparing this approxiation with forward and central differences for estimating $f'(1)$ with $f(x) = x \sin x$.

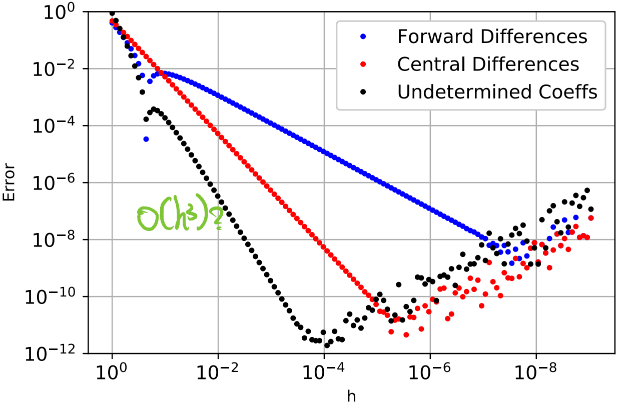

Figure 3. Error in approximating $f'(1)$.

$\varepsilon \approx 10^{-15}, 10^{-16}$

$O(h): h \approx \sqrt{\varepsilon}$

$O(h^2): h \approx \sqrt[3]{\varepsilon}$

Richardson Extrapolation

We can use our existing formulae to produce even better estimates. For example, the central difference formula is (with more error terms written out)

$$f'(x) \approx \underbrace{\frac{f(x+h)-f(x-h)}{2h}}_{=\phi(h)} \underbrace{- \frac{f'''(x)}{6}h^2 - \frac{f^{(5)}(x)}{120}h^4 + \mathcal{O}(h^6)}_{O(h^2)}$$

Now we use this formula with <u>both</u> h and $2h$:

$$\begin{aligned} f'(x) &= \phi(h) + a_2 h^2 + a_4 h^4 + \mathcal{O}(h^6), \\ f'(x) &= \phi(2h) + 4a_2 h^2 + 16 a_4 h^4 + \mathcal{O}(h^6). \end{aligned}$$

Subtract the second equation from $4\times$ the first equation (to cancel the h^2 terms), we get

$$3f'(x) = 4\phi(h) - \phi(2h) - 12 a_4 h^4 + \mathcal{O}(h^6),$$

and so

$$\boxed{f'(x) = \frac{4}{3}\phi(h) - \frac{1}{3}\phi(2h) + \mathcal{O}(h^4)}$$

Richard Extrapolation: Example

Comparing this approxiation with previous approaches for estimating $f'(1)$ with $f(x) = x \sin x$.

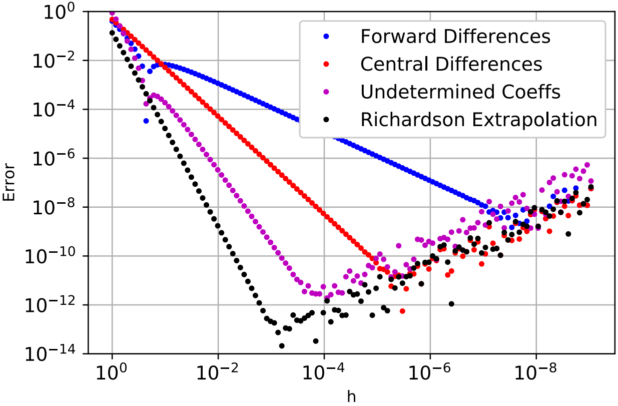

Figure 4. Error in approximating $f'(1)$.

Richardson Extrapolation: General Case

We have been able to use two central differences (with h and $2h$) to produce a new, better estimate. This idea is not specific to approximating derivatives, and is an example of a technique known as extrapolation.

Basic idea: using a sequence of (better) approximations to a quantity to predict the limiting value.

Example

I have been trying to approximate some number x^*. So far, my approximations are:

$$0.6,\ 1.2,\ 0.9,\ 1.05,\ 0.975,\ 1.0125, \ldots$$

Without computing any more information, what do you think x^* is?

Richardson Extrapolation: General Case

Richardson extrapolation (Lewis Fry Richardson, early 1900s) is a general technique for extrapolation when we have an approximation with power law behaviour:

$$\phi^* = \phi(h) + Ch^p + o(h^p).$$

We need to know p (e.g. $p = 2$ for central differences), but don't need to know C.

Then, we also have

$$\phi^* = \phi(2h) + 2^p Ch^p + o(h^p).$$

Combining these equations to eliminate the h^p term, we get

$$\phi^* = \frac{2^p \phi(h) - \phi(2h)}{2^p - 1} + o(h^p),$$

which is a better approximation than both $\phi(h)$ and $\phi(2h)$.

Richardson Extrapolation: Recursive

Now, we can keep going by "extrapolating the extrapolation". For example, for central differences we got

$$f'(x) \approx \underbrace{\frac{4}{3}\phi(h) - \frac{1}{3}\phi(2h)}_{=\phi_2(h)} - 4a_4 h^4 + \mathcal{O}(h^6).$$

Considering $\phi_2(h)$ and $\phi_2(2h)$ and cancelling the h^4 terms, we get the 6th order method

$$f'(x) \approx \frac{16\phi_2(h) - \phi_2(2h)}{15} + \mathcal{O}(h^6) = \underbrace{\frac{64}{45}\phi(h) - \frac{20}{45}\phi(2h) + \frac{1}{45}\phi(4h)}_{=\phi_3(h)} + \mathcal{O}(h^6).$$

Then we consider $\phi_3(h)$ and $\phi_3(2h)$, then cancel the h^6 terms to get an 8th-order method based on $\phi(h)$, $\phi(2h)$, $\phi(4h)$ and $\phi(8h)$, etc.

Richardson Extrapolation: General Recusrive Case

This idea works in general. Suppose we have an order $p = 2$ method, with error terms:

$$\phi^* = \phi(h) + a_2 h^2 + a_4 h^4 + a_6 h^6 + \cdots .$$

As we derived above, the first extrapolation step gives

$$\phi^* = \underbrace{\frac{4\phi(h) - \phi(2h)}{4-1}}_{=\phi_2(h)} - 4a_4 h^4 - 20 a_6 h^6 + \cdots$$

The next step (to cancel the h^4 terms) is

$$\phi^* = \underbrace{\frac{16\phi_2(h) - \phi_2(2h)}{16-1}}_{=\phi_3(h)} + 960 a_6 h^6 + \cdots$$

Richardson Extrapolation: General Recusrive Case

Continuing this we, the method $\phi_k(h)$ is of order h^{2k}:

$$\phi^* = \phi_k(h) + c_{2k} h^{2k} + \mathcal{O}(h^{2k+2}).$$

We get to the next method by cancelling the h^{2k} terms:

$$\phi^* = \underbrace{\frac{4^k \phi_k(h) - \phi_k(2h)}{4^k - 1}}_{=\phi_{k+1}(h)} + \mathcal{O}(h^{2k+2}).$$

To get our final answer $\phi_k(h)$, we need to calculate $\phi_{k-1}(h)$ and $\phi_{k-1}(2h)$, which depend on $\phi_{k-2}(h)$, $\phi_{k-2}(2h)$ and $\phi_{k-2}(4h)$, etc. Going all the way back to our original ϕ, we will ultimately need to calculate $\phi(h)$, $\phi(2h)$, ..., $\phi(2^k h)$.

For an error estimate, we can use our two best approximations:

$$|\phi^* - \phi_k(h)| \approx |\phi_{k-1}(h) - \phi_k(h)|.$$

Richardson Extrapolation: General Recusrive Case

All together, to compute $\phi_k(h)$ we need to compute (from left to right):

$$\begin{tikzcd}
\phi(h) \arrow[r] & \phi_2(h) \arrow[r] & \phi_3(h) & \cdots & \phi_{k-1}(h) \arrow[r] & \phi_k(h) \\
\phi(2h) \arrow[r] \arrow[ur] & \phi_2(2h) \arrow[r] \arrow[ur] & \phi_3(2h) & \cdots & \phi_{k-1}(2h) \arrow[ur] & \\
\phi(4h) \arrow[r] \arrow[ur] & \phi_2(4h) \arrow[r] \arrow[ur] & \phi_3(4h) & \iddots & & \\
\vdots \arrow[ur] & \vdots \arrow[ur] & \vdots & \iddots & & \\
\phi(2^{k-2}h) \arrow[r] & \phi_2(2^{k-2}h) \arrow[r] & \phi_3(2^{k-2}h) & & & \\
\phi(2^{k-1}h) \arrow[r] \arrow[ur] & \phi_2(2^{k-1}h) \arrow[ur] & & & & \\
\phi(2^k h) \arrow[ur] & & & & &
\end{tikzcd}$$

The top row gives the best estimates we have of ϕ^* ($\phi_k(h)$ is the best). The recursive formula is

$$\phi_{j+1}(2^i h) = \frac{4^j \phi_j(2^i h) - \phi_j(2^{i+1} h)}{4^j - 1}.$$

Richardson Extrapolation: Example

Let's use Richardson extrapolation to calculate $f'(1)$ for $f(x) = \sin x$, with $h = 0.25$:

$$\phi(h) = \frac{\sin(1.25) - \sin(0.75)}{0.5} \qquad \phi_2(h) = \frac{4\phi(h) - \phi(2h)}{4-1} \qquad \phi_3(h) = \frac{4^2\phi_2(h) - \phi_2(2h)}{4^2-1}$$

$$\phi(2h) = \frac{\sin(1.5) - \sin(0.5)}{1} \qquad \phi_2(2h) = \frac{4\phi(2h) - \phi(4h)}{4-1}$$

$$\phi(4h) = \frac{\sin(2) - \sin(0)}{2}$$

Numerically, this is

$$\phi(h) = 0.5346917 \qquad \phi_2(h) = 0.5402325 \qquad \phi_3(h) = 0.5403007$$

$$\phi(2h) = 0.5180694 \qquad \phi_2(2h) = 0.5392097$$

$$\phi(4h) = 0.4546487$$

The true value is $\cos(1) = 0.5403023059$. In practice, we calculate one column of values at a time: first $\phi(\cdot)$, then $\phi_2(\cdot)$, then $\phi_3(\cdot)$.

Richardson Extrapolation: Implementation

In Python, this looks like:

```python
import numpy as np

def richardson_central_differences(f, x, h, k):
    # Store all the values in a matrix R, where R[i,j] = phi_j(2**i * h)
    R = np.zeros((k+1,k+1))
    for i in range(k+1):
        # Create the first column of R
        # R[i,0] = central difference with step 2**i * h
        R[i,0] = (f(x + 2**i * h) - f(x - 2**i * h)) / (2 * 2**i * h)
    # Fill down each column recursively
    for j in range(1, k+1):
        for i in range(k-j+1):
            R[i,j] = (4**j * R[i,j-1] - R[i+1,j-1])/(4**j-1)
    return R[0,k]  # return value phi_k(h)
```

Richardson Extrapolation: Implementation

Again trying to calculate $f'(1)$ for $f(x) = \sin x$ with $h = 0.25$ and now $k = 5$, we get

```
[[ 0.534691719  0.540232476  0.540300661  0.540302217  0.540302294  0.540302302]
 [ 0.518069448  0.539209693  0.540202626  0.540282619  0.540294051  0.         ]
 [ 0.454648713  0.524315702  0.535163048  0.537367475  0.           0.         ]
 [ 0.245647748  0.361605509  0.396284125  0.           0.           0.         ]
 [-0.102225533 -0.158573734  0.           0.           0.           0.         ]
 [ 0.066819068  0.           0.           0.           0.           0.         ]]
```

The errors in these approximations are:

```
[[-0.005610587 -0.000069830 -0.000001645 -0.000000089 -0.000000012 -0.000000004]
 [-0.022232858 -0.001092613 -0.000099680 -0.000019687 -0.000008255             ]
 [-0.085653592 -0.015986604 -0.005139258 -0.002934831                          ]
 [-0.294654558 -0.178696797 -0.144018181                                       ]
 [-0.642527839 -0.698876040                                                    ]
 [-0.473483238                                                                 ]]
```

Algorithmic Differentiation

One final way to compute derivatives is using a completely different technique.

Algorithmic differentiation (AD) takes the source code for $f(x)$ and builds its own source code for $f'(x)$ using the chain rule.

It is widely used in optimisation and machine learning ("backpropagation"), but usually requires specialised software to work efficiently. If you have AD tools available, they can be very helpful, but it is very difficult to code yourself.

There are two modes: the forward mode is useful when you have many outputs to differentiate (and few input variables), the reverse mode is useful when you have few outputs to differentiate (but many input variables). The next slide shows an example of the forward mode of AD.

A related idea are adjoint equations, which can be used to calculate the derivative of the solution of a differential equation with respect to a parameter of the DE.

Historical note: most research on AD happened in the 1980s, and one of the leading figures was Andreas Griewank, a former ANU PhD student.

Algorithmic Differentiation: Example

Suppose we have code to evaluate $f(x) = (e^x - 1)/(x^2 + 1)$, broken down into the simplest parts:

```python
def f(x):
    u1 = math.exp(x)
    u2 = u1 - 1.0
    u3 = x**2
    u4 = u3 + 1.0
    u5 = u2 / u4
    return u5
```

AD software can differentiate these simple expressions, and so can generate code for $f'(x)$:

```python
def df(x):
    # ... assuming already have calculated u1, ..., u5
    du1 = math.exp(x)                        # u1'(x) = exp(x)
    du2 = du1                                # u2'(x) = du2/du1 * u1'(x)
    du3 = 2*x                                # u3'(x) = 2*x
    du4 = du3                                # u4'(x) = du4/du3 * u3'(x)
    du5 = (1/u4)*du2 + (-u2/u4**2) * du4     # u5'(x) = du5/du2 * u2'(x) + du5/du4 * u4'(x)
    return du5
```