

SOLUTION of (8-2.e):

Denote the zero of $f(z) := e^z - x$ with z^* , write for the derivative $f'(z) = e^z$, and set $e^{(n)} := z^{(n)} - z^*$ for the iteration errors.



As in [Lecture → § 8.4.2.5] and the proof of [Lecture → Lemma 8.3.2.15] we

- rewrite the Newton iteration as a **fixed-point iteration**,
- use a Taylor expansion of its iteration function around z^* ,
- and drop a powers $(e^{(n)})^p$ for $p > 2$ (asymptotic perspective).

As in [Lecture → (8.4.2.6)] the Newton iteration can be rewritten as

$$z^{(k+1)} = \Phi(z^{(k)}) \quad \text{with} \quad \Phi(z) = z - \frac{f(z)}{f'(z)}. \quad (8.2.5)$$

As in [Lecture → (8.4.2.7)] we find by the quotient rule

$$\frac{d\Phi}{dz}(z) = 1 - \frac{(f'(z))^2 - f(z)f''(z)}{(f'(z))^2} = \frac{f(z)f''(z)}{(f'(z))^2}.$$

Plugging in the concrete function f with $f(z) = e^z - x$, $f'(z) = e^z$, $f''(z) = e^z$, we get

$$\Phi'(z) := \frac{d\Phi}{dz}(z) = \frac{(e^z - x)e^z}{(e^z)^2} = 1 - xe^{-z},$$

and by straightforward differentiation

$$\Phi''(z) := \frac{d^2\Phi}{dz^2}(z) = xe^{-z}.$$

For the iteration error of a fixed-point iteration like (8.2.5) we have

$$e^{(k+1)} = z^{(k+1)} - z^* = \Phi(z^{(k)}) - \Phi(z^*) = \Phi(z^* + e^{(k)}) - \Phi(z^*). \quad (8.2.6)$$

Recall the **Taylor expansion** [Lecture → (1.5.4.28)] of a function $g \in C^3([a, b])$ around $t^* \in]a, b[$:

$$g(t^* + h) = g(t^*) + g'(t^*)h + \frac{1}{2}g''(t^*)h^2 + O(h^3) \quad \text{for } h \rightarrow 0. \quad (8.2.7)$$

Apply this formula to the right side of (8.2.6) with $g := \Phi$, $t^* := z^*$, $h := e^{(k)}$:

$$\blacktriangleright \quad e^{(k+1)} = \Phi'(z^*)e^{(k)} + \frac{1}{2}\Phi''(z^*)(e^{(k)})^2 + O((e^{(k)})^3) \quad \text{for } |e^{(k)}| \approx 0.$$

Since $e^{z^*} = x$, we immediately compute

$$\Phi'(z^*) = 0 \quad , \quad \Phi''(z^*) = 1,$$

which implies

$$e^{(k+1)} = \frac{1}{2}(e^{(k)})^2 + O((e^{(k)})^3).$$

If we ignore higher powers of the iteration error, which is justified if the initial guess is already close to the solution,

$$|e^{(n+1)}| \doteq \frac{1}{2}|e^{(n)}|^2 \doteq \dots \doteq \left(\frac{1}{2}\right)^{1+2+\dots+2^n} (e^{(0)})^{2^{n+1}} \doteq 2 \cdot \left(\frac{1}{2}\right)^{2^{n+1}} (e^{(0)})^{2^{n+1}} \doteq 2 \cdot \left(\frac{1}{2}|e^{(0)}|\right)^{2^{n+1}},$$

where $\doteq$ expresses equality “up to terms of size $O((e^{(k)})^3)$ ”. Further, the initial error is

$$e^{(0)} = z^{(0)} - z^* = x - 1 - \log(x), \quad x \in [\sqrt{2}, 2\sqrt{2}].$$

By discussing the simple function $x \mapsto x - 1 - \log(x)$ we find that the initial error $e^{(0)}$ attains its maximum value for $x = 2\sqrt{2}$:

$$|e^{(0)}| < \epsilon_{\max} := 2\sqrt{2} - 1 - \log(2\sqrt{2}) \approx 0.7887.$$

We can determine the minimal required number n of iteration steps by finding the smallest integer satisfying

$$\left| \frac{e^{(n)}}{\log x} \right| \leq \text{EPS}.$$

We are on the safe side, when replacing $|\log(x)|$ with its smallest possible value $\frac{1}{2} \log 2$, as $x \in [\sqrt{2}, 2\sqrt{2}]$. This leads to the requirement

$$|e^{(n)}| \leq \frac{1}{2} \log 2 \text{ EPS}.$$

Then we can plug in the expression for $e^{(n)}$ found above:

$$2 \left(\frac{1}{2} e^{(0)} \right)^{2^n} \leq \frac{1}{2} \log 2 \text{ EPS}.$$

Next, we replace $e^{(0)}$ with its upper bound and get

$$\left(\frac{1}{2} \epsilon_{\max} \right)^{2^n} \leq \frac{1}{4} \log 2 \text{ EPS},$$

from which we conclude that we need at least

$$n_{\min} := \lceil \log_2 \left(\frac{-\log\left(\frac{1}{4} \log 2 \text{ EPS}\right)}{-\log\left(\frac{1}{2} \epsilon_{\max}\right)} \right) \rceil$$

steps, where $\lceil \xi \rceil$ means the smallest integer $\geq \xi$.

The code in Code 8.2.8 demonstrates the use of this prediction for the number of required Newton iterations.

C++ code 8.2.8: Modified myfunction.

```
2 double myfunction_modified(double x) {
3     constexpr double dy = 0.693147180559945; // = std::log(2)
```

```

4  double y = 0.;
5  while (x > 2. * std::sqrt(2.)) {
6      x /= 2.;
7      y += dy;
8  }
9  while (x < std::sqrt(2.)) {
10     x *= 2.;
11     y -= dy;
12 }
13 double z = x - 1.;
14 double dz = x * std::exp(-z) - 1.;
15 // TODO: (8-2.e) Write a for-loop that achieves the same accuracy
16 // as the third while-loop of myfunction(). I.e. fix the number of
17 // iterations before looping.
18 // START
19 const double e_max = 2. * std::sqrt(2.) - 1. - std::log(2. * std::sqrt(2.));
20 constexpr double eps = std::numeric_limits<double>::epsilon();
21
22 const int k_min = std::ceil(log2((-std::log(0.25 * std::log(2.) * eps)) /
23                                (-std::log(0.5 * std::abs(e_max))))));
24
25 for (int i = 1; i < k_min; ++i) {
26     z = z + dz;
27     dz = x * std::exp(-z) - 1.;
28 }
29 // END
30 return y + z + dz;
31 }

```

Get it on  GitLab (codequiz.hpp).