

ETH Lectures 401-0663-00L Numerical Methods for Computer Science  
401-2673-00L Numerical Methods for CSE

# Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022  
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

## 0.1 General Information

### 0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

### 0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

### 0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

**Problem 8-2: Code quiz**

A frequently encountered drudgery in scientific computing is the use and modification of poorly documented code. This makes it necessary to understand the ideas behind the code first. Now we practice this in the case of a simple iterative method.

Related to [Lecture → Section 8.4.2.1], involves implementation in C++.

(8-2.a)  (20 min.) What is the purpose of the following C++ code?

**C++ code 8.2.1: Undocumented function**

```

2  double myfunction(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     while (std::abs(dz / z) > std::numeric_limits<double>::epsilon()) {
16         z += dz;
17         dz = x * std::exp(-z) - 1.;
18     }
19     return y + z + dz; //
20 }
```



HINT 1 for (8-2.a): First try to understand what is accomplished in Lines 5–12 of this code. Recall the properties of logarithms. 

HINT 2 for (8-2.a): Line 15–18 realize the Newton iteration for finding a zero of a real valued function. 

SOLUTION of (8-2.a):

The C++ code computes  $y = \log(x)$  for a given  $x$ . The loop in lines 15–18 realizes the Newton iterations for finding the zero of the function

$$f(z) = e^z - x, \quad z \in \mathbb{R}, \quad (8.2.2)$$

where  $z$  is the independent variable and  $x$  is given. We compute  $f'(x) = e^z$  and obtain the Newton iteration

$$z^{(k+1)} = z^{(k)} - \frac{\exp(z^{(k)}) - x}{\exp(z^{(k)})} = z^{(k)} - s, \quad s = 1 - \frac{x}{\exp(z^{(k)})}. \quad (8.2.3)$$

**C++ code 8.2.4: The documented function**

```

2  double myfunction(double x) {
3      // We require that x > 0.
```

```

4  assert(x > 0);
5  // First we factorize x as  $x = 2^{n+r} = 2^n * x_{new}$ 
6  // with integer n and  $r \in [0.5, 1.5]$ .
7  // Set  $y = n \log(2)$  and  $z = \log(x_{new})$ ,
8  // then  $\log(x) = y + z$ .
9  constexpr double dy = 0.693147180559945; // =  $\text{std::log}(2)$ 
10 double y = 0.;
11 // If  $x = 2^{n+r}$  with positive n:
12 while (x > 2. * std::sqrt(2.)) {
13     x /= 2.;
14     y += dy;
15 } //
16 // If  $x = 2^{n+r}$  with negative n:
17 while (x < std::sqrt(2.)) {
18     x *= 2.;
19     y -= dy;
20 } //
21 // Now, we have found y, and we use Newton iteration for the
22 // function  $f(z) = \exp(z) - x_{new}$  to find  $z = \log(x_{new})$ .
23 // Initial guess:
24 double z = x - 1.; //
25 // The update is  $dz = -f(z)/f'(z)$ 
26 double dz = x * std::exp(-z) - 1.;
27 while (std::abs(dz / z) > std::numeric_limits<double>::epsilon()) {
28     z += dz;
29     dz = x * std::exp(-z) - 1.;
30 }
31 return y + z + dz; //
32 }

```

Get it on  GitLab (codequiz.hpp).

**(8-2.b)**  (10 min.) Explain the rationale behind the first two initial **while** loops in the code Code 8.2.1, Lines 5–12, preceding the main iteration.

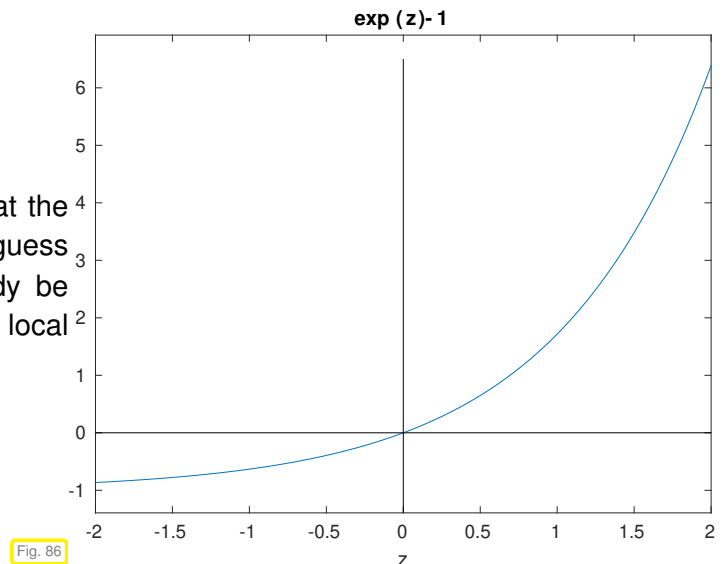
SOLUTION of (8-2.b):

The **while** loops rely on the identities

$$\log(x) = \log(2^{x/2}) = \log 2 + \log(x/2) \quad , \quad \log(x) = \log\left(\frac{1}{2} \cdot 2x\right) = -\log 2 + \log(2x) \quad .$$

to reduce the problem to the computation of  $\log(x)$  with  $x \in [\sqrt{2}, 2\sqrt{2}]$ .

The rationale for this can be seen by looking at the graph of  $f$  as defined in (8.2.2). An initial guess  $z^{(0)} := x - 1$  for  $x$  in that interval will already be close to the zero of  $f$ , so that we can expect local quadratic convergence of Newton's method.



**(8-2.c)** (10 min.) Explain what the **while** loop in lines 13–18 of Code 8.2.1 does.

SOLUTION of (8-2.c):

The **while**-loop computes the zero of the function  $z \mapsto f(z)$  from (8.2.2) using Newton iterations, see also (8.2.3):

$$\begin{cases} z^{(0)} &:= x - 1, \\ z^{(n+1)} &:= z^{(n)} - \frac{e^{z^{(n)}} - x}{e^{z^{(n)}}}, \quad n \geq 1 \end{cases}.$$

**(8-2.d)** (10 min.) Explain the conditional expression in the last **while** loop.

SOLUTION of (8-2.d):

This is a correction based termination criterium, see [Lecture → § 8.2.3.4]. It is meant to ensure that the relative error of the final iterate is of the size of the machine precision **EPS**, because in C++ **EPS** for a floating point data type **T** is available as **std::numeric\_limits<T>::epsilon()**.

**(8-2.e)** (60 min.) [ depends on Sub-problem (8-2.b) ]

Write a C++ function

```
double myfunction_modified(double x);
```

where you replace the last **while**-loop with a fixed number of iterations that, nevertheless, guarantee that the result has a relative accuracy **EPS**. Derive that required minimal number of iterations!

HINT 1 for (8-2.e): Using **Taylor expansion** and the fixed-point form of the Newton iteration as in [Lecture → § 8.4.2.5], establish a recursion for the **iteration errors**  $e^{(n)} := z^{(n)} - z^*$ . In that recursion,

drop all powers of the iteration error with an exponent  $> 2$ , because you may assume that the iteration error is always small for a good initial guess. ┘

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 → Eq. (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 → Eq. (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 → Eq. (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

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

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

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

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 \(codequiz.hpp\)](#).



End Problem 8-2 , 110 min.