

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 8-4: A derivative-free iterative scheme for finding zeros

[Lecture → Rem. 8.2.2.12] shows how to detect the order of convergence of an iterative method from a numerical experiment. In this problem we study the so-called **Steffensen's method**, which is a derivative-free iterative method for finding zeros of functions in 1D.

Related to [Lecture → Section 8.2.2] and requests simple C++ coding.

Let $f : [a, b] \mapsto \mathbb{R}$ be twice continuously differentiable with $f(x^*) = 0$ and $f'(x^*) \neq 0$. Consider the iteration defined by

$$x^{(n+1)} := x^{(n)} - \frac{f(x^{(n)})}{g(x^{(n)})}, \quad \text{where} \quad g(x) = \frac{f(x + f(x)) - f(x)}{f(x)}. \quad (8.4.1)$$

(8-4.a) ☐ (15 min.) Write a C++ function for the Steffensen's method:

```
template <class Function>
double steffensen(Function &&f, double x0)
```

f is a functor object or lambda function providing f , x_0 is the initial guess $x^{(0)}$. Terminate the iteration when the computed sequence of approximations becomes stationary, see [Lecture → Code 8.2.3.6] for an example.

SOLUTION of (8-4.a):

The implementation of (8.4.1) is straightforward. The finite-precision-based stopping rule can be implemented relying on the actual machine precision or by comparing two successive iterates.

C++-code 8.4.2: Solution of Sub-problem (8-4.a)

```
2  template <class Function>
3  double steffensen(Function &&f, double x0) {
4      double x_old = x0;
5      double x = x0;
6
7      // TODO: (8-4.a) Implement the Steffensen's method for a function f.
8      // START
9
10     double upd = 1;
11
12     // Get machine precision
13     const double eps = std::numeric_limits<double>::epsilon();
14
15     // Iterate until machine precision is reached
16     while (std::abs(upd) > eps * std::abs(x_old)) {
17         // Storing the old iterate
18         x_old = x;
19
20         // Only 2 evaluations of f at each step
21         const double fx = f(x);
22
23         if (fx != 0) {
24             // Compute update
25             upd = fx * fx / (f(x + fx) - fx);
26
27             // New iterate
28             x -= upd;
```

```

29     } else {
30         upd = 0;
31     }
32 }
33
34 // END
35
36 return x;
37 }

```

Get it on  GitLab (quadraticconvergence.hpp).



(8-4.b) (15 min.) [depends on Sub-problem (8-4.a)]

Write a C++ function

```
void testSteffensen();
```

that applies your implementation of `steffensen()` to find the zero of the function $f(x) = xe^x - 1$ (see [Lecture → Exp. 8.3.1.3]). Use $x^{(0)} = 1$ as initial guess.

SOLUTION of (8-4.b):

C++ code 8.4.3: Testing function for Steffensen's method

```

2  void testSteffensen() {
3      // TODO: (8-4.b) write a test of your implementation, that prints
4      // an estimate of the zero of  $f(x) = xe^x - 1$ 
5      // START
6
7      auto f = [](double x) { return x * std::exp(x) - 1; };
8      double x0 = 1.0;
9
10     const double x = steffensen(f, x0);
11
12     std::cout << "The iterative method converges to " << x << std::endl;
13
14     // END
15 }

```

Get it on  GitLab (quadraticconvergence.hpp).



(8-4.c) (10 min.) [depends on Sub-problem (8-4.a)]

We now want to investigate the order of Steffensen's method. To do so extend your implementation of `steffensen()` to

```

template <typename Function>
Eigen::VectorXd steffensen_log(Function &&f, double x0, LOGGER
    &&log = [](double) -> void {});

```

such that it logs each iterate using a callable `log()` as described in [Lecture → § 0.3.3.4]. Then use this new facility to implement a function

```
void orderSteffensen(void);
```

that tabulates values from which you can read of the order of Steffensen's method when applied to the test case of Sub-problem (8-4.b).

HINT 1 for (8-4.c): [Lecture → Def. 8.2.2.10] defines the order of convergence of an iteration for solving $f(x) = 0$, while [Lecture → Rem. 8.2.2.12] explains how to extract an estimate for the order from measured errors. ┘

SOLUTION of (8-4.c):

C++ code 8.4.4: Testing the order of Steffensen's method

```

2  void orderSteffensen() {
3      auto f = [](double x) { return x * std::exp(x) - 1; };
4
5      constexpr double x_star = 0.567143290409784; // use as exact value
6
7      // TODO: (8-4.c) Tabulate values from which you can read the
8      // order of Steffensen's method.
9      // Hint: To approximate the convergence rate, use the formula
10     // (log(ei) - log(ei-1)) / (log(ei-1) - log(ei-2))
11     // START
12
13     std::vector<double> store;
14
15     auto log = [&store](double x) -> void { store.push_back(x); };
16
17     steffensen_log(f, 1.0, log);
18
19     const unsigned n = store.size();
20
21     Eigen::VectorXd errs(n), log_errs(n);
22     for (unsigned i = 0; i < n; ++i) {
23         errs(i) = std::abs(store[i] - x_star);
24         log_errs(i) = std::log(errs(i));
25     }
26
27     Eigen::VectorXd ratios = Eigen::VectorXd::Zero(n);
28     for (unsigned i = 2; i < n; ++i) {
29         ratios(i) =
30             (log_errs(i) - log_errs(i - 1)) / (log_errs(i - 1) - log_errs(i - 2));
31     }
32
33     // Print output
34     std::cout.precision(10);
35     std::cout << std::setw(20) << "x" << std::setw(20) << "errors"
36         << std::setw(20) << "ratios" << std::endl;
37     std::cout << "
38         << std::endl;
39     for (unsigned i = 0; i < n; ++i) {
40         std::cout << std::setw(20) << store[i] << std::setw(20) << errs(i)
41             << std::setw(20) << ratios(i) << std::endl;
42     }
43
44     // END

```

45 }

Get it on  GitLab (quadraticconvergence.hpp).**C++ code 8.4.5: Steffensen's method with logging of iterates**

```

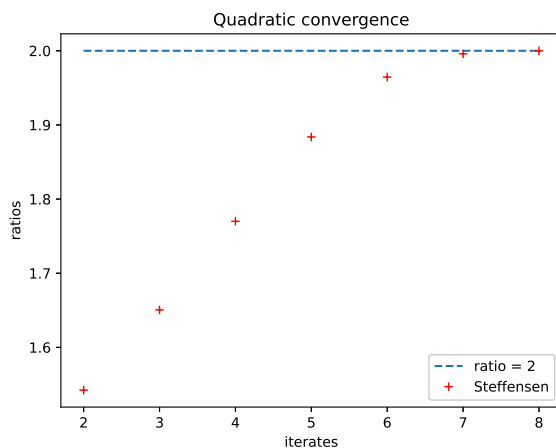
2  template <class Function, typename LOGGER>
3  double steffensen_log(Function &&f, double x0,
4      LOGGER &&log = [] (double) -> void {} ) {
5
6      double x = x0;
7
8      // TODO: (8-4.c) Modify the function steffensen: use the logger to
9      // save all the iterations x of the Steffensen's method.
10     // START
11
12     // Add x to the log. Note: This is an idle lambda function if none
13     is
14     provided. See the default value in the function signature.
15     log(x);
16
17     double upd = 1;
18     const double eps = std::numeric_limits<double>::epsilon();
19
20     // Iterate until machine precision is reached
21     while (std::abs(upd) > eps * x) {
22         // Only 2 evaluations of f at each step
23         const double fx = f(x);
24
25         if (fx != 0) {
26             // Compute update
27             upd = fx * fx / (f(x + fx) - fx);
28
29             // New iterate
30             x -= upd;
31
32             log(x);
33         } else {
34             upd = 0;
35         }
36     }
37
38     // END
39
40     return x;
41 }

```

Get it on  GitLab (quadraticconvergence.hpp).

The output is:

x	error e_n	$\frac{\log(e_n) - \log(e_{n-1})}{\log(e_{n-1}) - \log(e_{n-2})}$
1.0000000000000000	0.432856709590216	
0.923262600967822	0.356119310558038	
0.830705934728425	0.263562644318641	1.542345498206531
0.727518499997190	0.160375209587406	1.650553641703975
0.633710518522047	0.066567228112263	1.770024323911885
0.579846053882820	0.012702763473036	1.883754995643305
0.567633791946526	0.000490501536742	1.964598248590593
0.567144031581974	0.000000741172191	1.995899954235929
0.567143290411477	0.000000000001693	1.999927865685712
0.567143290409784	0.000000000000000	0.741551601040667



◁ quotients: $\frac{\log(e_n) - \log(e_{n-1})}{\log(e_{n-1}) - \log(e_{n-2})}$

The convergence is obviously quadratic, as can also be seen in Fig. 88. As in the experiments shown in class, roundoff affects the estimated order, once the iteration error approaches the machine precision.

Fig. 88

(8-4.d) (10 min.) For the test case of Sub-problem (8-4.b) the function $g(x)$ from (8.4.1) contains a term like e^{xe^x} . Therefore it grows very fast in x and the method cannot start for large $x^{(0)}$. How can you modify the function f (keeping the same zero) in order to allow the choice of a larger initial guess?

HINT 1 for (8-4.d): If f is a function and $h : [a, b] \rightarrow \mathbb{R}$ with $h(x) \neq 0, \forall x \in [a, b]$, then $(fh)(x) = 0 \Leftrightarrow f(x) = 0$.

SOLUTION of (8-4.d):

The choice $\tilde{f}(x) := (hf)(x) = e^{-x}f(x) = x - e^{-x}$ prevents the blow up of the function g and allows a larger set of positive initial points. Of course, $\tilde{f}(x) = 0$ exactly when $f(x) = 0$.

End Problem 8-4, 50 min.