Martin Ziegler
Holger Thies

Issued on 2015-06-09

Computational Complexity in Analysis

SoSe 2015, Exercise Sheet #8

For this exercise you will need to program in the `C++` programming language using the `iRRAM` framework. You can download `iRRAM` from irram.uni-trier.de. The version on github is recommended for this exercise, but the current release (2013_01) should also be fine.

If you do not want to install `iRRAM` on your local computer, you can use `Secure Shell` to get remote access to a computer with a working `iRRAM` installation.

```
ssh irram@zieg.de
Password: TUDarmstadt
```

EXERCISE 13:
The logistic map is given by the recurrence relation

$$x_{n+1} = a \cdot x_n (1 - x_n) \tag{1}$$

for some x_0, $a \in \mathbb{R}$.
For this exercise we assume $x_0 = 0.4$ and $a = 3.8$.

a) Write a `C++` program that computes x_n for $n \in \{10, 20, 50, 100, 1000, 10000\}$.

   Use the data-type `float` for all variables holding real numbers.

b) Now rewrite your program from a) using `double` instead of `float`. Do the results differ? *

c) Write the same program using the `iRRAM` framework and the data-type `REAL` for real number computations. Make sure that your output is correct at least up to error 2^{-30}. How do the results compare to part a) and b)?

d) Use `iRRAM`'s debug mode to find the number of `iRRAM` iterations and the internal precision needed to compute x_n for each of the n in part c).

EXERCISE 14:
In the lecture we have seen the trisection method to compute the zero of a computable function $f : [0,1] \to \mathbb{R}$ such that $f(0) < 0$ and $f(1) > 0$ under the assumption that exactly one zero exists.

a) Write a function

   ```
   REAL approx_zero(const int p, const std::function<REAL(const REAL&)>& f)
   ```

   The function should give an approximation to a zero of f with error bounded by 2^p if the function f is of the above form.

*Depending on your compiler there might in fact be no difference between the `float` and `double` data-types.

b) Now write a function `REAL zero(REAL f(const REAL& f))` computing the zero of *f* exactly by making use of `iRRAM`'s limit operators.

   Using the limit operator on a function that has a function as input is a little tricky.

   Instead of the limit seen in the lecture, the following function can be used

   ```
   REAL limit (const FUNCTION<REAL,int> & f )
   ```

   It works the same way as `REAL limit(REAL f(int))` but the input is a `FUNCTION` object. `FUNCTION` is a class defined by `iRRAM`, that can be constructed from an `std::function` object `g` by using the function `from_algorithm(g)`.

   Now, to use this limit operator you have to apply partial application to bind f to the second parameter of the `approx_zero` function, i.e., define a function `REAL h(int p)` such that `h(p) = approx_zero(p,f)`.

c) Can you extend your program such that the function can have more than one zero?