**Newton Iteration**

Newton's method is a simple method for finding approximations of roots of non-linear real functions. The goal is to develop a generic framework for this that can be instantiated for particular functions and connecting it with Isabelle's existing packages for interval arithmetic and Taylor models.

Advisor: [Manuel Eberl](https://www21.in.tum.de/~eberlm)

20.10 Reading IIA until mid-chapter 6.

Review on proofs in other proof assistants

https://math.stackexchange.com/questions/2963302/finding-libraries-of-formalized-mathematics

21.10 Reading IIA until chapter 8. page 116.

Questions for meeting.

1. Do you know the proof archives of other systems? (connection with math.stack.exchange)

2. to develop a generic framework for this that can be instantiated for particular functions

2.1 this implies building an interval analysis framework (examples in Isabelle)

2.2 particular functions (given by Isabelle libraries)

2.3 and then proving theorems (essentially theorem and lemma 8.1)

**Mail Immler (18/10)**  
Resources for interval arithmetic:

|  |
| --- |
| theory Decision\_Procs.Approximation |
| <http://home.in.tum.de/~hoelzl/> |
| Proving Real-Valued Inequalities by Computation in Isabelle/HOL |
| Proving Inequalities over Reals with Computation in Isabelle/HOL |
| Introduction to Interval Analysis:  http://www-sbras.nsc.ru/interval/Library/InteBooks/IntroIntervAn.pdf |

Proposed directions:

|  |
| --- |
| 1. a one-dimensional (using the library) |
| 2. extend it to vectors and matrices (interval arithmetic library isn't really well developed, and  there might be some useful things in the Affine\_Arithmetic) |

Bibliography ot explore

http://webcache.googleusercontent.com/search?q=cache:hVCYk2MaVB4J:perso.ens-lyon.fr/nathalie.revol/publis/Revol03.ps+&cd=1&hl=es&ct=clnk&gl=de

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.427.4628&rep=rep1&type=pdf

https://github.com/NAMEhzj/Newton-Algorithm-and-Midpoint-Method-for-Intervals

Gappa: https://www.lri.fr/~melquion/

29/18

Moore defines midpoint([a;b]) = [floor((a+b)/2); ceil((a+b)/2)]. Note that the midpoint (a + b) / 2 itself can be represented exactly as a floating point number in Isabelle.

The only problem is that each time we take the midpoint, the bit width can increase, so if we never round the midpoints, we will have to compute with higher and higher precision.  
  
For the derivative, use the theorem DERIV\_approx:  
  
⟦

n < length xs;

bounded\_by xs vs;

isDERIV\_approx prec n f vs;

Some (l, u) = approx prec (DERIV\_floatarith n f) vs

⟧  
  ⟹

∃x. x ≥ real\_of\_float l ∧ x ≤ real\_of\_float u ∧  
         ((λx. interpret\_floatarith f (xs[n := x])) has\_real\_derivative  
x) (at (xs ! n))  
  
If you have an expression f on some variables vs with certain bounds, then calling "approx" on the expression given by DERIV\_floatarith gives you an enclosure [l;u] in which the derivative of f w.r.t. the given variable x lies.

There is also the "isDERIV\_approx" assumption, which is a (computable) predicate on the input expression that ensures that the expression is actually differentiable w.r.t. the given variable in the given bounds. This will return "False" if e.g. you have something like "abs x" in there, or "1 / x" and x can be 0.