DSLsofMath 2021: Assignment 2

Optimisation using Newton's method

This assignment is based on the lectures from weeks 3 and 4 (the *FunExp* type, *eval*, *derive*, *D*, tupling, homomorphisms, *FD*, *apply*, ...) so it pays off to work through those notes carefully.

1. The evaluation of the second derivative is given by

$$eval'' = eval' \circ derive = eval \circ derive \circ derive$$

   (a) Show that $eval''$ is not a homomorphism from *FunExp* to $FunSem = \mathbb{R} \to \mathbb{R}$.

   (b) Given the following types

$$\begin{array}{ll}
\textbf{type}\ \mathit{Tri}\ a & = (a, a, a) \\
\textbf{type}\ \mathit{TriFun}\ a = \mathit{Tri}\ (a \to a) & \text{-- } = (a \to a, a \to a, a \to a) \\
\textbf{type}\ \mathit{FunTri}\ a = a \to \mathit{Tri}\ a & \text{-- } = a \to (a, a, a)
\end{array}$$

   Define instances of the classes *Additive*, *AddGroup*, *Multiplicative*, *MulGroup*, *Algebraic*, and *Transcendental*, for *Tri a* and define a homomorphism *evalDD* from *FunExp* to *FunTri a* (for any type *a* in *Field*). You don't need to prove that it is a homomorphism in this part.

   (c) Show that *evalDD* is a homomorphism for the case of multiplication.

2. Newton's method allows us to find zeros of a large class of functions in a given interval. The following description of Newton's method follows Bird and Wadler [1988], page 23:

$$\begin{array}{l}
\textbf{type}\ \mathbb{R} = \mathit{Double} \\
\mathit{newton} :: (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R} \\
\mathit{newton}\ f\ \epsilon\ x = \textbf{if}\ \mathit{abs}\ \mathit{fx} < \epsilon \\
\qquad\qquad\qquad \textbf{then}\ x \\
\qquad\qquad\qquad \textbf{else if}\ \mathit{fx}' \neq 0\ \textbf{then}\ \mathit{newton}\ f\ \epsilon\ \mathit{next} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ \ \mathit{newton}\ f\ \epsilon\ (x + \epsilon) \\
\quad \textbf{where}\ \mathit{fx} \quad = f\ x \\
\qquad\qquad \mathit{fx}' \quad = \mathit{undefined} \quad \text{-- } f'\ x\ \text{(derivative of } f \text{ at } x\text{)} \\
\qquad\qquad \mathit{next} = x - (\mathit{fx}\ /\ \mathit{fx}')
\end{array}$$

   (a) Implement Newton's method, using $\mathit{Tri}\ \mathbb{R} \to \mathit{Tri}\ \mathbb{R}$ for the type of the first argument. In other words, use the code above to implement

$$\mathit{newtonTri} :: (\mathit{Tri}\ \mathbb{R} \to \mathit{Tri}\ \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}$$

   in order to obtain the appropriate value for $f'\ x$.

(b) Test your implementation on the following functions:

$$\begin{array}{ll}
\textit{test0}\ x = x\hat{}2 & \text{-- one (double) zero, in zero} \\
\textit{test1}\ x = x\hat{}2 - \textit{one} & \text{-- two zeros, in +-one} \\
\textit{test2}\ x = \textit{sin}\ x & \text{-- many, many zeros (in } n * \pi \text{ for all } n :: \mathbb{Z}) \\
\textit{test3}\ n\ x\ y = y\hat{}n - \textit{constTri}\ x & \text{-- } \textit{test3}\ n\ x\text{, has zero in "nth roots of } x\text{"} \\
\quad \text{-- where } \textit{constTri} \text{ is the embedding of } \textit{Const} &
\end{array}$$

For each of these functions, apply Newton's method to a number of starting points from a sensible interval. For example:

$$\textit{map}\ (\textit{newton test1}\ 0.001)\ [-2.0, -1.5 \mathinner{..} 2.0]$$

but be aware that the method might not always converge!

For debugging is advisable to implement *newton* in terms of *newtonList*, a minor variation which returns a list of the approximations encountered on the way to the final answer:

$$\begin{array}{l}
\textit{newton}\ f\ \epsilon\ x = \textit{last}\ (\textit{newtonList}\ f\ \epsilon\ x) \\
\textit{newtonList}\ f\ \epsilon\ x = x : \mathbf{if}\ \ldots\ \mathbf{then}\ [\,]\ \mathbf{else}\ \ldots
\end{array}$$

3. We can find the optima of a twice-differentiable function on an interval by finding the zeros of its derivative on that interval, and checking the second derivative. If $f'\ x_0$ is zero, then

- if $f''\ x_0 < 0$, then x_0 is a maximum
- if $f''\ x_0 > 0$, then x_0 is a minimum
- if $f''\ x_0 = 0$, then, if $f''\ (x_0 - \epsilon) * f''\ (x_0 + \epsilon) < 0$ (i.e., f'' changes its sign in the neighbourhood of x_0), x_0 is an inflection point (not an optimum)
- otherwise, we don't know

Use Newton's method to find the optima of the test functions from point 2. That is, implement a function

$$\textit{optim} :: (\textit{Tri}\ \mathbb{R} \to \textit{Tri}\ \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \textit{Result}\ \mathbb{R}$$

so that $\textit{optim}\ f\ \epsilon\ x$ uses Newton's method to find a zero of f' starting from x. If y is the result (i.e. $f'\ y$ is within ϵ of 0), then check the second derivative, returning $\textit{Maximum}\ y$ if $f''\ y < 0$, $\textit{Minimum}\ y$ if $f''\ y > 0$, and $\textit{Dunno}\ y$ if $f'' = 0$.

As before, use several starting points.

Hint: you might want to modify the code you've written for Newton's method at point 2.

Formalities

Submission: Assignments are to be submitted via Canvas

Deadline: 2021-03-05

Grading: Discussions with each of the teams during one of the slots 2021-03-08.

Skeleton code

Here is some useful skeleton Haskell code to start from, and the *Algebra* and *FunExp* modules are also available on github.

```
{-# LANGUAGE FlexibleContexts, FlexibleInstances, TypeSynonymInstances #-}
module A2_Skeleton where
import Prelude hiding ((+), (-), (*), (/), negate, recip, (^),
                       sin, pi, cos, exp, fromInteger, fromRational)
import DSLsofMath.Algebra
import DSLsofMath.FunExp

type Tri a     = (a, a, a)
type TriFun a = Tri (a -> a)   -- = (a -> a, a -> a, a -> a)
type FunTri a = a -> Tri a     -- = a -> (a, a, a)

instance Additive a       => Additive (Tri a)       where (+) = addTri; zero = zeroTri
instance (Additive a, Multiplicative a)
                          => Multiplicative (Tri a) where (*) = mulTri; one  = oneTri
instance AddGroup a       => AddGroup (Tri a)       where negate = negateTri
instance (AddGroup a, MulGroup a)
                          => MulGroup (Tri a)       where recip  = recipTri

(addTri, zeroTri, mulTri, oneTri, negateTri, recipTri) = undefined

instance Transcendental a => Transcendental (Tri a) where
  pi = piTri; sin = sinTri; cos = cosTri; exp = expTri

(piTri, sinTri, cosTri, expTri) = undefined
```

References

R. Bird and P. Wadler. *Introduction to Functional Programming, 1988.* Prentice-Hall, Englewood Cliffs, NJ, 1988.