Lab 4

Due: June 9, 2023 at the end of class

1 Lab 4 Specification

1.1 Overview

Continued fractions are fractions that follow a nested form, given two lists of terms, a and k:

$$a_0 + \cfrac{k_0}{a_1 + \cfrac{k_1}{a_2 + \cfrac{k_2}{a_3 + \ddots}}}$$

A finite continued fraction terminates after n iterations, which is great for computing the result in the real world:

$$a_0 + \cfrac{k_0}{a_1 + \cfrac{k_1}{\ddots + \cfrac{k_n}{a_n}}}$$

It is not immediately clear that this formulation has any use, but it turns out to allow quick approximations for some popular mathematical constants. The value of the Golden Ratio, for example, is a continued fraction with both $a = [1, 1, 1, \dots]$ and $k = [1, 1, 1, \dots]$. Continuing fractions can also approximate trig functions like $\sin x$ and $\tan^{-1} x$, as well as logarithms. In this lab, we will write concise F# functions to approximate the two most prevalent non-integer constants.

Since continuing fractions are a nested repeating function, the functional concept of folding applies to them very well.

1.2 Implementation

The skeleton code at `https://classroom.github.com/a/DyKE5CC5` is the start of this week's lab. Write and test the following functions. If you forget the values for the target constants, there are references on the internet. For this lab, the builtin `List.foldBack`, `List.map`, `List.filter`, and `List.fold` functions are available if you want.

1.2.1 Goals

1. Practice writing and running higher order F# functions
2. Practice modifying a math function to use as a foldback parameter

1.2.2 The constant π

One way to express a continuing fraction to approximate π is:

$$\pi \approx 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \ddots}}}}$$

The lists a and k in this case should be $[6, 6, 6, \ldots]$ and $[1^2, 3^2, 5^2, \ldots]$, respectively. (There is that weird 3 right at the beginning of the a sequence. We will account for it as a special case.)

Before we can start folding anything, we need to generate the list k. Complete the function `oddSquares` that has one input parameter – the number of terms in the list, and one output parameter – the list k that has type **double list**. (Use **double** instead of **float** for better precision.)

It should behave like:

```
> oddSquares 6;;
val it : double list = [1.0; 9.0; 25.0; 49.0; 81.0; 121.0]
```

Now that we can generate the list k, we need to write a folding function to incorporate k in the continued fraction. The continued fraction follows the `List.foldBack` formulation: $f(k_1, f(k_2, \ldots f(k_{n-1}, f(k_n, c)) \ldots))$

Determine what the function f should be, based on the definition of a continued fraction. For this fraction, we don't need a list for a – its value is always 6. In continued fractions, the value of c has a negligible effect on the overall value. You can just use 1.0 as the seed value.

Define the `piApprox` function with a `List.foldBack` call. `piApprox` should have one `int` input parameter: the number of terms to use in the approximation. That parameter should be passed to the `oddSquares` function that generates an appropriately sized list to fold.

You should get results roughly as follows:

```
> piApprox 10;;
val it : double = 6.141287133
> piApprox 20;;
val it : double = 6.141558104
```

π should really be closer to 3 than that. Looking back at the continued fraction for π, it looks like a_0 should have been 3 instead of 6. Fix that offset by adding a constant of -3.0 to the result of the `foldBack` call.

With a small number of terms, the approximation won't be very close. As the number of terms increases, your approximation of π should be more and more accurate. Within about 500 terms, the size of the data container should be the limiting factor in our precision.

1.2.3 The constant e

The constant e has a startling habit of popping up in many branches of mathematics, as well as equations that describe phenomena in the real world. e has a continuing fraction approximation:

$$e \approx 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{2}{3 + \cfrac{3}{4 + \ddots}}}}$$

Examining the pattern for lists a and k for this approximation, we can see that both a_n and k_n are just n. (Once again, there's something inconsistent with the initial value of '$2 + \ldots$'. We will return to that later.) Since the two lists are the same, we really only need one list parameter, which is great! `List.foldBack` only takes one list parameter to fold.

To generate the list of values, we can use the shortcut formulation `[1.0 .. 5.0]` to generate a list `[1.0; 2.0; 3.0; 4.0; 5.0]`. Define `eApprox` to take one input parameter, which is the number of terms to include in the list. Determine the function to fold, and use a seed value of 1.0 again.

With a list of values starting at 1, we didn't calculate the full approximation. Instead, we've calculated something like:

$$\text{eApprox} = 1 + \cfrac{1}{2 + \cfrac{2}{3 + \cfrac{3}{4 + \ddots}}}$$

Modify this incomplete calculation to get the correct value:

$$\text{eApprox} = 2.0 + \frac{1.0}{\text{List.foldBack result}}$$

Once `eApprox` is fully defined, you should see results like:

```
> eApprox 5;;
val it : double = 2.717770035
> eApprox 50;;
val it : double = 2.718281828
```

1.3 piApprox Contest [Extra Credit]

In addition to getting correct answers from `piApprox`, try to write a version with as little code as possible. It should calculate π with the same continued fraction equation as the first function.

I will count the number of characters for this contest, so you should get rid of extra whitespace. If you define any helper functions, they count towards the full function length as well.

Restrictions: the function must be named `piShort` and the type must be `int -> float` or `int -> double`.

(If I get lots of copies of the same answer, those won't count towards the contest.)

2 Grading

Be professional. Make results easy to understand and grade. Include only those parts to be graded, as well as their helper functions. Leave comments where necessary, especially if it aids in grading.

Each approximation function is worth 50% of the lab grade.