CSE1729 - Introduction to Programming | February 10, 2022

Problem Set 4

1. Abstracting the summation of a series
   Consider the harmonic numbers $H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. Last week you wrote a recursive SCHEME function (named `harmonic`) which, given a number n, computes H_n. Revise your `harmonic` function, keeping the name `(harmonic n)`, to take advantage of the `sum` function seen in the textbook (Section 1.3.1) and shown below:

```
(define (sum term a next b)
    (if (> a b)
        0
        (+ (term a)
           (sum term (next a) next b))))
```

   Of course, your new and improved definition of harmonic should not be recursive itself and should rely on `sum` to do the hard work.

2. Abstracting the product of a series
   (a) The function `sum` in Question 1 is an abstraction of the sigma notation for summation:

$$\sum_{n=a}^{b} f(n) = f(a) + \cdots + f(b)$$

   Write a function `(product term a next b)` that abstracts the pi notation for the product of a series. You should name your function `(product term a next b)`.

$$\prod_{n=a}^{b} f(n) = f(a) \times \cdots \times f(b)$$

   (b) In mathematics, Wallis' product for π, documented in 1655 by John Wallis, states that

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \left(\frac{2n}{2n-1} \cdot \frac{2n}{2n+1} \right) = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdots$$

   Use your one of your product functions to define an function `(wallis-pi n)` that returns an approximation of π using the first n terms of the infinite product above. You will need to define appropriate functions for `term` and `next`.

3. For two functions f and g and an integer $n \geq 0$ we are interested in the sum of fractions of the form:

$$\frac{f(-n)}{g(-n)} + \cdots + \frac{f(n)}{g(n)}.$$

   (Note that this sum has $2n+1$ terms in it, as it evaluates the fraction $f(x)/g(x)$ at all points between $-n$ and n.) Note that a little care is required here, because the quantity is not even defined if $g(k) = 0$ for one of the $k \in \{-n, \ldots, n\}$. To work around this issue, we define

$$R(f, g; n) = q(-n) + \cdots + q(n)$$

where

$$q(k) = \begin{cases} 0 & \text{if } g(k) = 0, \text{ and} \\ \frac{f(k)}{g(k)} & \text{otherwise.} \end{cases}$$

Write a SCHEME program, called `frac-sum` so that `(frac-sum f g n)` returns the value $R(f, g; n)$ as defined above.

4. Recall the definition of the derivative of a function from calculus (you will not need to know any calculus to solve this problem):

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

By choosing very small values for h in the above equation, we can get a good approximation of $f'(x)$.

(a) Write a function (call it `der` for *derivative*) in SCHEME that takes a function f and a value for h as formal parameters and returns the function g defined by the rule

$$g(x) = \frac{f(x+h) - f(x)}{h}.$$

As mentioned above, for small `h`, g is a good approximation for the derivative of f.

Important note: Your function should take a *function* and a *number* as arguments (for example `(der f h)` where `f` is a function and `h` is a number) and return a *function*.

(b) Now take it a step further and write a function which takes three formal parameters f, n and h and returns an approximation of the n^{th} derivative of f. Call your function `der-n`; then `(der-n f n h)` is an example call to the function. Make use of the derivative function you just wrote. (Specifically, you wish to return the function obtained by applying `der` to your function n times.)

5. *Newton's Method* is an iterative method for finding successively better approximations to the roots (that is, the zeroes) of a real-valued function. To be more precise, given a function f, Newton's Method is an approach to find a value x for which $f(x) \approx 0$. Newton's Method requires an initial guess for the root (x_0) and determines a sequence of values $x_1, x_2, \ldots$ defined by the recursive rule:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

For many functions of interest, these "iterates" converge very quickly to a root.

For example, consider the polynomial $p(x) = x^2 - x - 1$. It turns out that the positive root of p is the number 1.618... that you computed in many ways on the last problem set (the golden ratio). Let's see Newton's method in action. It turns out that the derivative of p is the polynomial $p'(x) = 2x - 1$. (You don't need to know how to determine the explicit derivative of functions for this problem—you'll be using instead the code you generated in the last problem.) Starting with the guess $x_0 = 2$, we may run Newton's method forward to find that:

$$\begin{aligned} x_0 &= 2 \\ x_1 &= x_0 - \frac{p(x_0)}{p'(x_0)} = 2 - \frac{p(2)}{p'(2)} = \frac{5}{3} = 1.666\ldots \\ x_2 &= x_1 - \frac{p(x_1)}{p'(x_1)} = \frac{5}{3} - \frac{p(5/3)}{p'(5/3)} = 1\tfrac{13}{21} = 1.61904\ldots \\ &\ldots \end{aligned}$$

Write a SCHEME function that implements Newton's Method–call it `newton`. Your function should accept three formal parameters, a function f, an initial guess for the root x_0 and the number of iterations to run, n. You can make use of the derivative function from the previous problem (with `h` set to .01).

Specifically, the call `(newton f x n)` should return the nth iterate, as defined above, of Newton's method. (Thus `(newton f x 0)` should return x.)

To test your implementation, you can try the following in DrRacket.

(a) Use your implementation can find the root of the function $f(x) = 2x^2 - 1$. (start with an initial guess of 4).

(b) Use your solution to find a good approximation to the golden ratio by working with the polynomial $g(x) = x^2 - x - 1$ (start with the guess 2).

6. SICP, Problem 1.29 (Simpson's rule). Recall that the *integral* of a function f between a and b (with $a < b$) is the area underneath the function on the interval $[a, b]$. See Figure 1. Simpson's rule, described in your book, is a method for *approximating* this value.

   To begin with, define a function `(sum-term term a b)` that takes a function `term` and two integers a and b as arguments and returns the sum $\texttt{term}(a) + \texttt{term}(a+1) + \cdots + \texttt{term}(b)$. Use this in your solution by defining a function that computes the Simpson's rule terms and passing this to `sum-term`.

   Once `sum-term` is defined, as above, use this to construct a function `simpson-integrate` so that

   ```
   (simpson-integrate f a b n)
   ```

   returns the estimate to the integral of the function f, on the interval $[a, b]$ with n samples, according to Simpson's rule. Recall that Simpson's rule is simply:

$$\int_a^b f(x)dx = \frac{\Delta x}{3} \cdot (f(x_0) + 4 \cdot f(x_1) + 2 \cdot f(x_2) + 4 \cdot f(x_3) + 2 \cdot f(x_4) + \ldots + 4 \cdot f(x_{n-1}) + f(x_n))$$

   where $\Delta x = \frac{b-a}{n}$ and $x_i = a + i \cdot \Delta x$.

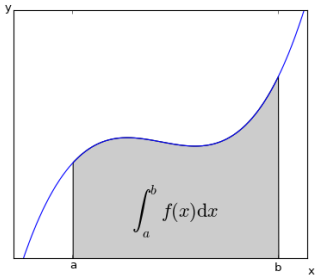

Figure 1: The integral of f from a to b is the area of the shaded region above.

To check your solution, you might try integrating the functions $f_1(x) = x$, $f_2(x) = x^2$, and $f_4(x) = x^4$ on the interval $[0, 1]$ (so $a = 0$ and $b = 1$). You should find that the integral of f_1 is $1/2$ (the area of the triangle), the integral of f_2 is $1/3$, and the integral of f_4 is $1/5$.