MATH 387 ASSIGNMENT 1

DUE THURSDAY FEBRUARY 11

1. In this and the following exercises, we will study standard multiplication and division algorithms in arbitrary precision floating point arithmetics. We consider those numbers that are represented in the format

$$a = \pm \sum_{k=0}^{\infty} a_k \beta^{k+e},$$

where $0 \leq a_k \leq \beta - 1$ are the digits (or "big digits") of the mantissa that are stored in a finite (!) array of integers, and $e \in \mathbb{Z}$ is the exponent, stored as a single integer variable (In practice, one rarely needs exponents that go out of the 32 or 64 bit integer range). Furthermore, $\beta \in \mathbb{N}$ is a fixed parameter called the base (or radix), e.g., $\beta = 1000$, chosen to be a fairly large integer, but not exceedingly large, because among other things, we need to be able to perform divisions on numbers of the form $b_0 + b_1\beta + b_2\beta^2$ by using the built-in CPU arithmetic.

It is clear that multiplication reduces to multiplication of two positive integers. To multiply two positive integers, we first define the *Cauchy product*

$$ab = \Big(\sum_{j=0}^{\infty} a_j \beta^j\Big) \cdot \Big(\sum_{i=0}^{\infty} b_i \beta^i\Big) = \sum_{k=0}^{\infty} \Big(\sum_{j=0}^{k} a_j b_{k-j}\Big)\beta^k = \sum_{k=0}^{\infty} p_k^* \beta^k, \tag{1}$$

where

$$p_k^* = \sum_{j=0}^{k} a_j b_{k-j}, \tag{2}$$

is the k-th *generalized digit* of ab. In general, p_k^* can be larger than $\beta - 1$, and so (1) is not the base-β expansion of the product ab. However, the proper digits $0 \leq p_k \leq \beta - 1$ of ab can be found by writing p_k^* in base-β and then performing the summation in the right hand side of (1) in base-β arithmetic. One way to find the base-β expansion of p_k^* would be to do the summation in (2) from the beginning in base-β arithmetic.

(a) Formulate a multiplication algorithm that is based on the double summation in (1). In other words, we do not want to compute the generalized digits p_k^* explicitly. Write a convincing argument (i.e., proof) that your algorithm terminates in a finite number of steps, and that it returns the correct answer. Suppose that n is the largest index for which $a_n \neq 0$, and that m is the analogous quantity for b. Basically, n and m measure how much storage each of a and b takes. Then estimate the number of

Date: Winter 2016.

built-in arithmetic operations needed to compute the digits of ab, in terms of n and m, when n and m are large.

(b) With the intent of saving resources, let us ignore the terms with $k < k^*$ in (1), with the truncation parameter k^*, i.e., we replace the product ab by

$$\tilde{p} = \sum_{k=k^*}^{\infty} \Big(\sum_{j=0}^{k} a_j b_{k-j} \Big) \beta^k.$$

Show that

$$0 \leq ab - \tilde{p} \leq ab \cdot \beta^{k^*+3-n-m},$$

where n and m are as in (a). What would be a good choice for the value of k^*?

2. Now we consider a division algorithm in the context of the preceding problem. We assume that a and b are positive integers. The goal is to compute the digits of $q \geq 0$ and $0 \leq r < b$, satisfying

$$a = qb + r.$$

Here and in what follows, unless otherwise specified, all variables are *integer* variables. The algorithm we are going to build is an adaptation of the usual long division algorithm we study in school. Recall that

$$n = \max\{k : a_k \neq 0\}, \qquad m = \max\{k : b_k \neq 0\}.$$

Without loss of generality, we can assume $n \geq m$ and $a > b \geq 2$.

As a warm up, let us treat the special case $m = 0$ first. In this case, b has only one digit, i.e., $2 \leq b \leq \beta - 1$, so division can be performed in a straightforward digit-by-digit fashion. Thus the first step of the division algorithm would be to divide a_n by b, as

$$a_n = q_n b + r_n,$$

where $0 \leq r_n < b$ is the remainder, and $q_n \geq 0$ is the quotient. Obviously, $q_n \leq \beta - 1$ because $a_n \leq \beta - 1$. Computation of q_n and r_n should be performed in computer's built-in arithmetic. To proceed further, we combine r_n with the $(n-1)$-st digit of a, and divide it by b, that is,

$$r_n \beta + a_{n-1} = q_{n-1} b + r_{n-1},$$

where $0 \leq r_{n-1} < b$. Since $r_n < b$, we are guaranteed that $q_{n-1} \leq \beta - 1$. Computation of q_{n-1} and r_{n-1} should be performed in computer's built-in arithmetic, which imposes an upper bound on β. This procedure is repeated until we retrieve the last digit a_0, and we finally get

$$\begin{aligned} a &= a_n\beta^n + \ldots + a_0 = (q_nb + r_n)\beta^n + a_{n-1}\beta^{n-1} + \ldots + a_0 \\ &= q_nb\beta^n + (r_n\beta + a_{n-1})\beta^{n-1} + \ldots + a_0 \\ &= q_nb\beta^n + (q_{n-1}b + r_{n-1})\beta^{n-1} + \ldots + a_0 = \ldots \\ &= q_nb\beta^n + q_{n-1}b\beta^{n-1} + \ldots + q_0b + r_0 \\ &= b\sum_{k=0}^{n} q_k\beta^k + r_0, \end{aligned} \tag{3}$$

which shows that q_k is the k-th digit of q, and that $r = r_0$.

In the general case $m > 0$, the overall structure of the algorithm does not change, but there will be one essential new ingredient in the details. Before describing the algorithm, let us introduce a convenient new notation. For $0 \leq k \leq \ell$ let

$$a_{[k,\ell]} = a_k + a_{k+1}\beta + \ldots + a_\ell\beta^{\ell-k},$$

which is simply the number consisting of those digits of a that are numbered by $k, \ldots, \ell$. For example, when $\beta = 10$ and $a = 1532$, we have $a_{[2,4]} = 15$. The first step of our algorithm is to compute q_{n-m} and $0 \leq r_{n-m} < b$ satisfying

$$a_{[n-m,n]} = q_{n-m}b + r_{n-m}. \tag{4}$$

Since the number of digits of $a_{[n-m,n]}$ is the same as that of b, we have $q_{n-m} \leq \beta - 1$. Next, we compute q_{n-m-1} and $0 \leq r_{n-m-1} < b$ satisfying

$$r_{n-m}\beta + a_{n-m-1} = q_{n-m-1}b + r_{n-m-1}. \tag{5}$$

Since $r_{n-m} < b$, we are guaranteed that $q_{n-m-1} \leq \beta - 1$. We repeat this process until we retrieve the last digit a_0, and expect that q_k's give the digits of q.

This seems all well and good, except that there is a catch: In (4) and (5), we divide by b, which has $m+1$ digits, and we cannot rely on the built-in arithmetic since m can be large. We encounter the divisions (4) and (5) in each step of the paper-and-pencil long division method. There, what helps is intuition and the fact that in practice we usually have m not too large. Here, we need to replace intuition by a well defined algorithm. We shall consider here an approach that is based on a few crucial observations. The first observation is that since $r_{n-m} < b$ and $a_{n-m-1} < \beta$, we have

$$r_{n-m}\beta + a_{n-m-1} \leq (b-1)\beta + \beta - 1 = b\beta - 1,$$

so that the left hand side of (5) has at most $m+2$ digits. Noting that the left hand side of (4) has $m+1$ digits, we now see that (4) and (5) only require divisions of a number not exceeding $b\beta - 1$ by b. In other words, the original division problem a/b has been reduced to the case $a \leq b\beta - 1$ (and hence with $m \leq n \leq m+1$). This indeed helps, because if two numbers have roughly the same number of digits, then the first few digits of both numbers can be used to compute a very good approximation of the quotient. For instance, it turns out that under the assumption $a \leq b\beta - 1$, if

$$a_{[m-1,n]} = q^*b_{[m-1,m]} + r^*, \tag{6}$$

with $0 \leq r^* < b_{[m-1,m]}$, then

$$q \leq q^* \leq q + 1. \tag{7}$$

This means that the quotient of the number formed by the first 2 or 3 digits of a, divided by the number formed by the first 2 digits of b, is either equal to the quotient q of a divided by b, or off by 1. The cases $q^* = q + 1$ can easily be detected (and immediately corrected) by comparing the product $q^* b$ with a. The division (6) can be performed in the built-in arithmetic, because the number of digits of any of the operands therein does not exceed 3.

(a) Assuming that we have some means to carry out the divisions (4) and (5), demonstrate the correctness of the algorithm, that is, show that

$$a = r_0 + b \sum_{k=0}^{n-m} q_k \beta^k.$$

(b) Now let us focus on the divisions (4) and (5), that would occur in each iteration of the general division algorithm. Assume that $m \leq n \leq m + 1$ and $a \leq b\beta - 1$, and fix some (small) integer $p \geq 0$. Let q^* and $0 \leq r^* < b_{[m-p,m]}$ be defined by

$$a_{[m-p,n]} = q^* b_{[m-p,m]} + r^*. \tag{8}$$

Derive upper and lower bounds on q^* in terms of q. Then by using these bounds, show that

$$q \leq q^* \leq q + 1, \tag{9}$$

as long as $p \geq 1$.

(c) Describe a procedure to identify q, given that we have computed q^* as in (b) with $p = 1$. In particular, how do we tell $q^* = q$ or $q^* = q + 1$?

(d) What would be the maximum value of β that allows the division algorithm to work correctly, supposing that we want β to be a power of 10, and that the built-in arithmetic can handle 64 bit integers?

3. For $x > 0$ large, consider the computation of e^{-x} by truncating the Taylor series

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} + \ldots.$$

Suppose that we are given a tolerance parameter $\varepsilon > 0$, and we truncate the series once the current term becomes smaller than ε in absolute value. As the series is alternating, in exact arithmetic we would have the error of the truncated the series bounded by ε.

(a) Explain why you would expect cancellation of digits under floating point arithmetic. With the help of a calculator or a computer, produce a concrete and illustrative example where such a cancellation occurs.

(b) Come up with a method to compute e^{-x} without cancellation of digits. Show off the performance of your method by a concrete example.

4. Let $x_1, x_2, \ldots$ be a sequence of floating point numbers, and let $s_n = x_1 + \ldots + x_n$. Consider *Kahan's compensated summation algorithm*

$$\begin{aligned} y_n &= x_n + e_{n-1} \\ \tilde{s}_n &= \tilde{s}_{n-1} + y_n \\ e_n &= (\tilde{s}_{n-1} - \tilde{s}_n) + y_n, \qquad n = 1, 2, \ldots \end{aligned}$$

where each operation is performed in floating point arithmetic, and $\tilde{s}_0 = e_0 = 0$.

(a) Explain why you would expect the roundoff accuracy of this method to be better than that of the naive summation method.

(b) (*Note*: This part will not be graded.) Show that

$$|\tilde{s}_n - s_n| \leq [C\varepsilon + O(\varepsilon^2)] \sum_{k=1}^{n} |x_k|,$$

where C is some constant, and ε is the machine epsilon.

5. (a) Recall that the iteration (called the *Babylonian method* or *Heron's method*)

$$x_{n+1} = \frac{1}{2}\big(x_n + \frac{a}{x_n}\big),$$

for given $a > 0$ and $x_0 > 0$, converges to $x = \sqrt{a} > 0$. Show that

$$x_{n+1} - x = \frac{(x_n - x)^2}{2x_n}, \qquad \text{and} \qquad \varepsilon_{n+1} = \frac{\varepsilon_n^2}{2(1 + \varepsilon_n)},$$

where $\varepsilon_n = \frac{|x_n - x|}{x}$ is the relative error. Further, show that

$$\varepsilon_{n+1} \leq \frac{\min\{\varepsilon_n^2, \varepsilon_n\}}{2}.$$

In particular, we have $\varepsilon_{n+1} \leq \frac{\varepsilon_n}{2}$, meaning that any initial value $x_0 > 0$ leads to a sequence that converges to the correct value x. Such methods are called *globally convergent.* Moreover, methods with the property $\varepsilon_{n+1} \leq q\varepsilon_n$ for some constant $0 < q < 1$, are called *linearly convergent.* A linearly convergent method increases the number of significant digits by a fixed amount in each iteration. Obviously, the Babylonian method is linearly convergent, but much more is true. For some n large enough, we would have $\varepsilon_n < 1$, and from then on, the estimate $\varepsilon_{n+1} \leq \frac{\varepsilon_n^2}{2}$ goes into effect. This type of convergence is called *quadratic*, because for all n large, $\varepsilon_{n+1} \leq C\varepsilon_n^2$ for some constant C (We can take $C = \frac{1}{2}$ in the current case). During a quadratic convergence, the number of significant digits *doubles* in each iteration.

(b) Along the lines of the derivation we did in class for the Babylonian method, design a *quadratically convergent* iteration for computing $\sqrt[n]{a}$, where $n > 0$ is an integer, and $a > 0$. Derive *a priori* and *a posteriori* error estimators. Is the method globally convergent? Illustrate the performance of the method by a concrete example.

(c) Design a *quadratically convergent* iteration for computing $\frac{1}{a}$, that involves only multiplication and addition (Subtraction is a special case of addition). Derive *a priori* and *a posteriori* error estimators. Is the method globally convergent? Illustrate the performance of the method by a concrete example.

6. (a) Let us call the functions $\sin x$, $\arctan x$, e^x, and $\log x$ the *basic functions*. Then reduce the evaluation of $\cos x$, $\tan x$, $\arcsin x$, $\arccos x$, and x^a ($a \in \mathbb{R}$) into basic functions and elementary arithmetic operations. Here by elementary arithmetic operations we understand addition, subtraction, multiplication, division, and n-th root extraction $\sqrt[n]{x}$ (for $n > 0$ integer and $x > 0$ real), and all variables are real (in the sense that they are not complex variables).

   (b) Reduce the argument of $\arctan x$ into $[0, b]$, where $b < 1$ is to be chosen by you (Generally, $b \approx \frac{1}{2}$ would be considered satisfactory). In other words, express $\arctan x$ with $x \in \mathbb{R}$, in terms of $\arctan y$ with $0 \leq y \leq b$.

   (c) (*Note*: You may choose between (c) and (d).) Perform a detailed round-off error analysis on an algorithm that computes $\log y$ by employing the series
$$\log \frac{1+x}{1-x} = 2\big(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots\big).$$
   You can assume $-\frac{1}{2} \leq x \leq \frac{1}{2}$ in the analysis. Then describe a procedure to reduce the argument of $\log x$ into $-\frac{1}{2} \leq x \leq \frac{1}{2}$.

   (d) (*Note*: You may choose between (c) and (d).) Perform a detailed round-off error analysis on an algorithm that computes $\sin x$ ($0 < x \leq \frac{\pi}{4}$) by using its Maclaurin series. Describe a procedure to reduce the argument of $\sin x$ into $0 < x \leq \frac{\pi}{4}$.

Homework policy

You are welcome to consult each other provided (1) you list all people and sources who aided you, or whom you aided and (2) you write-up the solutions independently, in your own language. If you seek help from other people, you should be seeking general advice, not specific solutions, and must disclose this help. This applies especially to internet fora such as `MathStackExchange`.

Similarly, if you consult books and papers outside your notes, you should be looking for better understanding of or different points of view on the material, not solutions to the problems.