Lecture II
Approximate Newton and Evaluation of Elementary Functions

THIS IS A SUPPLEMENT TO LECTURE II OF THE BOOK OF Mehlhorn-Yap.

This note is an elaboration of Brent's work on the complexity of approximating elementary functions such as $\exp x, \log x, \sin x, \arcsin x$, etc. Algebraic functions such as $\sqrt{x}$ and zeros of polynomials can be included among these elementary functions. The key to these results lie in the use of Newton iteration.

The main restriction of Brent's work is that the argument x of functions are required to lie in some fixed but arbitrary range $[a, b]$. Removing this restriction may sometimes require non-trivial techniques. If this can be done, then Brent's approach appears to yield the fastest method available for such functions.

§1. Introduction

In a series of papers from the 1970's, Richard Brent has demonstrated the effectiveness of Newton-type methods for approximating various elementary functions. We shall mainly review his results in [3]. The basic conclusion is that all the common elementary functions can be evaluated at arguments in some fixed but arbitrary range $0 \notin [a, b]$ to s precisions in time $O(M(s) \log s)$ where $M(s)$ is the complexity of multiplying two s-bit integers. The class of "elementary functions" is usually not a precise notion. Following Shanks [5], we may define an **elementary function** if as a complex function that can be built up by a finite composition of constant functions, field operations, algebraic functions, exponential and logarithmic functions. An algebraic function $f(x)$ is one that satisfies an equation $A(f(x)) = 0$ for some integer polynomial $A(X)$. E.g., $f(x) = \sqrt{x}$. We need to treat these functions as complex functions so that trigonometric functions and their inverses can be captured as well. However, in our treatment below, we restrict our elementary functions to real arguments.

In [2], Brent discussed the inherent complexity of evaluating such functions. In particular, it can be shown $\exp(x)$ and $\sin(x)$ can simulate multiplication, and hence must have complexity at least $M(s)$ for evaluation to precision s. In other words, the $O(M(n) \log n)$ upper bound for evaluating elementary functions is tight to a factor of $\log s$.

It is highly desirable to extend Brent's results to unrestricted range. It would also be nice to general the results to hypergeometric functions, as the evaluation of elementary functions can be reduced to evaluation of hypergeometric functions. The basic approach to achieving unrestricted range is to reduce the computation of $f(x)$ to a computation of $f(x/2^n)$ such that $x/2^n$ lies in some fixed range $[a, b]$. This could be done for particular functions such as $\log(x)$ or $\exp(x)$. See [4, Appendix]. For the trigonometric functions, this requires approximating $x \mod \pi$, a problem known as "argument reduction". But it is not clear how to do this reduction in general, or say, for the hypergeometric functions.

§2. Complexity Model for Bigfloats

The arithmetic model of Brent [2] is variable precision floating point numbers (which we simply call "bigfloats"). This is a very natural extension of the "fixed precision" arithmetic normally used in numerical analysis. In this section, we describe this model in some detail.

Let $M(n)$ denote the complexity of multiplying two n-bit binary integers on multitape Turing machines. This definition of $M(n)$ applies only to positive integers n. To simplify our statements, we shall define $M(x)$ for any real number x: if $x \leq 0$, $M(x) = 0$ and otherwise let $M(x) = M(\lceil x \rceil)$. The Schönhage-Strassen bound says that $M(n) = O(n \log n \log\log n)$. This bound can be achieved in the (multitape) **Turing machine model**. In general, our complexity bounds will be in this Turing model, but such bounds can be tedious to show directly. So what we normally show is a weaker result: we simply count the number of bit operations and this established the complexity in the the so-call :Boolean Circuit model. These two models are closely related: in particular, if a problem has time complexity $T(n)$ in the Turing model, then it has $T(n)$ in the Boolean in the Boolean model.

In our applications, we do not explicitly use the Schönhage-Strassen bound, but leaves $M(n)$ as a parameter in our the complexity statements. We only appeal only to some general assumptions on $M(n)$:
1. Superlinear: $M(n) \geq n$.
2. Monotone: $M(n)$ is monotone nondecreasing.
3. Regularity: $M(n)$ satisfies the "regularity condition" that for all $\alpha \in (0, 1)$,

$$\alpha^2 M(n) \leq M(\alpha n) \leq \alpha M(n) \text{ (ev. } n). \tag{1}$$

where "$\cdots$ (ev. n)" means that condition "$\cdots$" will **eventually hold**, *i.e.*, when n is sufficiently large.

This regularity condition is from [1] and is slightly different then Brent. These assumptions are satisfied by the Schönhage-Strassen bounds for $M(n)$, but is also satisfied by the naive bound $M(n) = O(n^2)$ or the Karatsuba bound $M(n) = O(n^{\lg 3})$. In this way, the complexity bound we derive can be used even if we use the alternative algorithms.

We need to explicitly discuss how to represent bigfloats. A (binary) **bigfloat** is a rational number of the form $n2^m$ where $n, m \in \mathbb{Z}$. We call it bigfloat to distinguish it from the usual programming concept of a "float" which has fixed precision (typically, this means $|n| < 2^t$ where t is the precision). If f is an integer, we write $\langle f \rangle$ for the value $f 2^{-\lfloor \lg |f| \rfloor}$. We may call $\langle f \rangle$ the "normalized value" of f. E.g., $\langle 1 \rangle = \langle 2 \rangle = \langle 4 \rangle = 1$, $\langle 3 \rangle = \langle 6 \rangle = 1.5$, $\langle 5 \rangle = 1.25$, $\langle 7 \rangle = 1.75$, etc. In general, for $f \neq 0$, we have $|\langle f \rangle| \in [1, 2)$. and $\langle 2^k f \rangle = \langle f \rangle$ for all $k \in \mathbb{N}$.

Let the binary representation of an integer $f \neq 0$ can be written as $f = \sigma(b_0, b_1, \ldots, b_t)_2$ where $b_i \in \{0, 1\}$, $b_0 \neq 0$ and $\sigma \in \{+, -\}$. A bigfloat can then be denoted by a sequence of bits with a binary point somewhere in this sequence: $\sigma(b_0 b_1 \cdots b_i . b_{i+1} \cdots b_t)$, and this represents the value $\sigma(b_0, \ldots, b_t)_2 2^{t-i}$. In this notation, $\langle f \rangle$ is equal to $\sigma(b_0 . b_1 b_2 \cdots b_t)_2$. In other words, the binary point in $\langle f \rangle$ is placed just to the right of the most significant bit, b_0. Thus $\langle f \rangle \in [1, 2)$.

A **bigfloat representation** x is a pair of binary integers, written $\langle e, f \rangle$ or $bigfloat(e, f)$. This representation has **value** equal to $f 2^{e - \lfloor \lg |f| \rfloor} = \langle f \rangle 2^e$. E.g., the value of $\langle \lfloor \lg |f| \rfloor, f \rangle$ is f. We say $\langle e, f \rangle$ is **normalized** if $e = f = 0$ or if f is odd. Clearly every bigfloat has a unique normalized representation.

When there is no confusion, it is convenient to treat x and its value interchangeably. For any real number, define the **most significant bit position** of x to be $\mathrm{msb}(x) := 1 + \lfloor \lg(|x|) \rfloor$. By definition, let $\mathrm{msb}(0) = 0$. When n is an integer, $\mathrm{msb}(n)$ is the length of the binary representation of n. For instance, $\mathrm{msb}(n) = n$ for $n = 0, 1, 2$ and $\mathrm{msb}(2^k - 1) = k$. $\mathrm{msb}(2^k) = k + 1$. Roughly speaking, $\mathrm{msb}(x)$ is the number of bits to the left of the most significant bit before we reach the binary point. Thus $\mathrm{msb}((11.111)_2) = 2$ If $f = \sigma(b_0 b_1 \cdots b_i . b_{i+1} \cdots b_t)_2$ then $\mathrm{msb}(f) = i + 1$.

> Let $[a, b]$ be a fixed but arbitrary range where $0 < a < b$. We shall be considering bigfloats in $[a, b] \cup [-b, -a]$. Such bigfloats are called "bounded" bigfloats.

Note that we may be able to assume that $b = 1/a$ then $c \in [a, b]$ iff $1/c \in [a, b]$.

Some Error Notations. For $x, \widetilde{x}, \ell \in \mathbb{R}$, then we say $\widetilde{x}$ is a **ℓ-bit absolute approximation** to x if $|x - \widetilde{x}| \le 2^{-\ell}$, and $\widetilde{x}$ is a **ℓ-bit relative approximation** to x if $|x - \widetilde{x}| \le 2^{-\ell}|x|$.

It is said that the art of error analysis consists of a good notation:

- First, a meta-notation: whenever we use the symbol "$\pm$", it is to be rewritten as "$+\theta$" where θ is a new variable satisfying $|\theta| \le 1$. E.g., $x \pm u$ is rewritten as $x + \theta u$. Each occurence of $\pm$ in a single expression will have a different implicit θ variable: $(x \pm u)(y \pm u)$ is the same as $(x + \theta u)(y + \theta' u)$ where $|\theta| \le 1$ and $|\theta'| \le 1$.
- Second, let "$[x]_n$" be a short hand for "$x(1 \pm 2^{-n})$". By our meta-notation, this refers to a number of the form $x(1 + \theta 2^{-n})$. Thus, this number is an an approximation to x with n relative bits of precision. As an application, we write

  $$[x + y]_n, \quad [x - y]_n, \quad [xy]_n, \ldots$$

  for the **truncated relative precision arithmetic** with n-bit relative precision. For any binary operation $\circ \in \{+, -, \times, \ldots\}$, we have the equation

  $$[x \circ y]_n = (x \circ y)(1 \pm 2^{-n}).$$

  There are corresponding truncated unary operations.
- In addition to the notation $[x]_n$ for relative error, we have a similar notation for absolute error,

  $$\{x\}_n = x \pm 2^{-n} = x + \theta 2^{-n}$$

  and the corresponding **truncated absolute error arithmetic**,

  $$\{x + y\}_n, \quad \{x - y\}_n, \quad \{xy\}_n, \ldots.$$

Theorem 1 *Let x, y be bounded bigfloats, and n be a positive natural number.*

1. *We can compute $[x]_n$ in $O(n)$ time.*
2. *We can compute $[xy]_n$ in $O(M(n))$ time.*
3. *We can compute $[x + y]_n$ in $O(n)$ time provided $xy \ge 0$ or $|x| > 2|y|$ or $|x| < |y|/2$. In general, $[x + y]_n$ only has the bound $O(n_x + n_y)$ where n_x, n_y are the respective precision of x, y.*
4. *An analogous statement holds for $[x - y]$, where we replace $xy \ge 0$ by $xy \le 0$.*

Proof. Let $x = \langle e_x, f_x \rangle$.

1. To compute $[x]_n$, we just have just have to truncate f_x to at most $(n+1)$-bits. This is done by repeated decrements of a binary counter whose initial value is n. With each decrement, we copy the next bit of f_x. The bits are copied starting from the most significant bit. We stop when we reach the least significant bit of f_x or when the counter reaches 0. The complexity of decrementing a binary counter from n to 0 is well-known to be $O(n)$ time (the obvious upper bound is $O(n \lg n)$). We return at most $n+1$ bits of f_x that have been copied.

2. To compute $[xy]_n$, we compute compute $[x]_{n+2}$ and $[y]_{n+2}$ and multiply their factional parts together in $O(M(n))$ time. To see that this is correct, let $\widetilde{x} = [x]_{n+2} = x(1+\theta 2^{-n-2})$ and $\widetilde{y} = [y]_{n+2} = y(1+\theta' 2^{-n-2})$ where $|\theta| \le 1$ and $|\theta'| \le 1$. Then $\widetilde{x} \cdot \widetilde{y} = xy(1+\theta 2^{-n-2})(1+\theta' 2^{-n-2})$ which is easily seen to be equal to $xy(1+\theta'' 2^{-n})$ for some $|\theta''| \le 1$.

3. To compute $[x+y]_n$, we first compute $[x]_{n+1}$ and $[y]_{n+1}$. This takes $O(n)$ time. We now compare $e_x : e_y$ in $O(1)$ time. Without loss of generally, assume $e_x \ge e_y$ We now align the two $n+1$-bit fractional parts so that their binary points coincide and carry out the standard addition. This amounts to computing $e_x - e_y$, and putting this into a binary counter. Now, as above, we decrement this counter to get to the bit in $[y]_{n+1}$ that corresponds to the least significant bit of $[x]_{n+1}$. Now we can begin the addition of these two aligned fractional parts.

Under the assumption that $xy \ge 0$ or $\mathrm{msb}(x) \ne \mathrm{msb}(y)$, we are guaranteed that $\mathrm{msb}(x+y)$ is

To compute $[xy]_n$, we first compute $[x]_{2n}$ and $[y]_{2n}$ and then apply the algorithm for integer multiplication to the fractional parts. **Q.E.D.**

There are two notable facts about this lemma:
1. The actual precision of x and y does not figure into the complexity except for one case. As usual, the above are only shown in Boolean complexity model. To achieve such sublinear bounds on a Turing machine requires care, for instance, we should ensure that the the tape head for each input bigfloat representation $(sign, exponent, faction)$ is positioned at the beginning of the representation.
2. We cannot generally achieve precision n in addition or substraction in $O(n)$ time, because of the possibility of cancellation. In the worst case, we may have to look at all the bits of the fractional part in x and y.

EXERCISE:
(1) Verify the regularity condition when $M(n) = Cn \lg n \lg \lg n$, and when $M(n) = Cn^c$ (you must determine the range of the constant c in the later case).
(2) Redo the above theorem for unbounded bigfloats.

§3. Reciprocal

To warm up, we consider how to efficiently compute the reciprocal of a bigfloat number $c \ne 0$.

Lemma 2 *For $\alpha \in (0,1)$, we have*

$$\sum_{j=0}^{\infty} M(\alpha^j n) = O(M(n)).$$

Proof. By regularity, we have $M(\alpha^j n) \le \alpha^j M(n)$ and so

$$\begin{aligned}\sum_{j\ge 0} M(\alpha^j n) &\le M(n)\sum_{j\ge 0}\alpha^j \\ &= \frac{M(n)}{1-\alpha}.\end{aligned}$$

Q.E.D.

The infinite summation in this lemma has only $O(\lg n)$ non-zero terms. Naively, the sum is upper bounded by $O(M(n)\lg n)$, but regularity of $M(n)$ allows us to remove the $\lg n$ factor.

Lemma 3 *For a bounded bigfloat c, we can compute $1/c$ to precision n in time $O(M(n))$.*

Proof. Let $f(x) = \frac{1}{x} - c$. Then Newton's iterator for this function is $N(x) = x - f(x)/f'(x) = x(2 - cx)$. Thus our iteration is

$$x_{i+1} = x_i(2 - cx_i). \tag{2}$$

If $x_i = (1-\varepsilon_i)/c$ or $cx_i = 1-\varepsilon_i$ then substituting into (2) gives $x_{i+1} = (1-\varepsilon_i^2)/c$. Hence $\varepsilon_{i+1} = \varepsilon_i^2$. Assuming $|\varepsilon_0| < 1/2$, we conclude that $|\varepsilon_i| < 2^{-2^i}$ for all $i \ge 0$. Let $y = 1/c$ and $\widetilde{y} = x_k$ where $k = \lceil \lg n\rceil$. Then $|y - \widetilde{c}| = \varepsilon_k/c < 2^{-n}/c = O(2^{-n})$ (since c is bounded).

The above analysis assumes error-free computation (in fact, we even have $\varepsilon_i \ge 0$ for $i > 0$). To extend this analysis to include error in each iteration, we must ensure (at least) that the last step is done with precision n. Now, it turns out that Newton iteration is self-correcting so that the previous step only requires about precision $n/2$, etc. Suppose that the i-th iteration (2) is carried out with precision 2^{i+1}. The algorithm has $\lg n$ iterations. Then the complexity of our algorithm is bounded by $\sum_{i=0}^{\lg n} M(n2^{-i}) = O(M(n))$, by the Lemma 2.

The details could be slightly messy, so to see the big outline, you could initially only pay attention to the first order terms (we underline the second order terms). Let $c\widetilde{x}_i = 1 - \delta_i$ where

$$\delta_i = 2^{2^i-1}3^{-2^i}\theta_i = \frac{1}{2}(2/3)^{2^i}\theta_i, \qquad (\text{for some } |\theta_i| \le 1). \tag{3}$$

Suppose we compute the operations in (2) to precision $2^{-2^{i+1}}$. For our analysis, let us break (2) into three steps:

```
ITERATION $\widetilde{x}_i \to \widetilde{x}_{i+1}$:
1.  $y_i \leftarrow [c\widetilde{x}_i]_{2^{i+1}}$
2.  $z_i \leftarrow 2 - y_i$
3.  $\widetilde{x}_{i+1} \leftarrow [\widetilde{x}_i z_i]_{2^{i+1}}$
```

Note that we assume Step 2 is error-free. It is possible to do this without affecting our basic premise that the ith iteration in (2) takes time $O(M(2^i))$. Step 1 gives

$$\begin{aligned} y_i &= c\widetilde{x}_i(1 + 2^{-2^{i+1}}\theta), \qquad |\theta| \le 1. \\ &= (1-\delta_i)(1 + 2^{-2^{i+1}}\theta) \\ &= 1 - \delta_i + 2^{-2^{i+1}}\theta(1-\delta_i). \end{aligned}$$

Thus $z_i = 1 + \delta_i - 2^{-2^{i+1}}\theta(1-\delta_i)$. Note that

$$|1 - y_i| = \left|\delta_i - 2^{-2^{i+1}}\theta(1-\delta_i)\right| \le |\delta_i| + 2^{1-2^{i+1}} \le \frac{1}{2}\left(\frac{2}{3}\right)^{2^i} + 2(1/4)^{2^i} < 1.$$

Hence

$$\begin{aligned} c\widetilde{x}_{i+1} &= c\widetilde{x}_i z_i(1 + 2^{-2^{i+1}}\theta'), \qquad |\theta'| \le 1. \\ &= (1-\delta_i)(1+\delta_i - 2^{-2^{i+1}}\theta(1-\delta_i))(1+2^{-2^{i+1}}\theta') \\ &= (1-\delta_i^2 - (1-\delta_i)^2\theta 2^{-2^{i+1}})(1+2^{-2^{i+1}}\theta') \\ &= 1 - \delta_{i+1} \end{aligned}$$

where

$$\begin{aligned} \delta_{i+1} &= \delta_i^2 + (1-\delta_i)^2\theta 2^{-2^{i+1}} + \underline{2^{-2^{i+1}}\theta\left(\delta_i^2 + \theta'(1-\delta_i)^2 2^{-2^{i+1}}\right)} \\ |\delta_{i+1}| &\le \frac{1}{4}\left(\frac{2}{3}\right)^{2^{i+1}} + \left(\frac{1}{2}\right)^{2^{i+1}} \\ &\le \frac{1}{2}\left(\frac{2}{3}\right)^{2^{i+1}} \end{aligned}$$

for $i \ge 1$. Since this last inequality requires $i \ge 1$, we must start this algorithm with an approximate value x satisfying $|x - (1/c)| \le |\delta_1| \le \frac{1}{2}(2/3)^{2^1} = 2/9$. **Q.E.D.**

Corollary 4 *Computing by x/c for a bounded c to precision n has complexity $O(M(n))$.*

Proof. We compute $[x/c]_n$ using the steps

```
0.   $z_0 \leftarrow [x]_{n+2}$
1.   $z_1 \leftarrow [1/c]_{n+2}$
2.   $z_2 \leftarrow z_0 z_1$
3.   Return($z_2$)
```

Each step takes $O(M(n))$. The result is seen to have precision n. **Q.E.D.**

EXERCISE:

(1) Extend the above result to an unbounded bigfloat c.
(2) Give an analogous analysis for the quadratic convergence of Newton's iteration for square root of a bounded bigfloat c.
(3) Extend question (2) to unbounded c.
(4) We said that to implement the approximate Newton iteration for computing $[1/c]_n$, we must begin with x_0 such that $|x_0 - (1/c)| < 2/9$. How to you achieve this in practice.

§4. Contraction Maps

The Newton iterator is basically a contraction map. Let us investigate some basic properties of contractions.

Let

$$f : S \to S$$

be any continuous function where $S \subseteq \mathbb{C}$ is any closed set.

We say f is **Lipschitz** if there is a constant $K > 0$ such that for all $x, y \in S$, $|f(x) - f(y)| \le K|x - y|$. K is called a **Lipschitz constant**. When $K < 1$, then f is called a **contraction map**.

Lemma 5 *If $f : S \to S$ is a contraction map with Lipschitz constant $K < 1$ then there is a unique fixed point $x^* \in S$ such that for all $x \in S$, $f^n(x) \to x^*$ as $n \to \infty$. Moreover, $f(x^*) = x^*$ and*

$$\frac{|f(x) - x|}{1 + K} \le |x - x^*| \le \frac{|f(x) - x|}{1 - K}. \tag{4}$$

Proof. We have

$$|f^{n+1}(x) - f^n(x)| \le K|f^n(x) - f^{n-1}(x)| \le \cdots \le K^n|f(x) - x|.$$

Then

$$\begin{aligned} |f^{m+n}(x) - f^n(x)| &\le \sum_{i=0}^{m-1} |f^{i+1+n}(x) - f^{i+n}(x)| \\ &\le \sum_{i=0}^{m-1} K^{i+n}|f(x) - x| \\ &< \frac{K^n}{1 - K}|f(x) - x|. \end{aligned}$$

Thus the sequence $x, f(x), f^2(x), \ldots$ is a Cauchy sequence with a limit $x^* \in S$. By continuity, we know that $f(x^*) = x^*$.

We must show that if $y \in S$, then $y, f(y), f^2(y), \ldots$ also converge to x^*. That is because $|f^n(y) - f^n(x)| \le K^n|y - x|$.

To show the first inequality in (4), we note that

$$\begin{aligned} |f(x) - x| &\le |f(x) - x^*| + |x^* - x| \\ &\le |f(x) - f(x^*)| + |x^* - x| \\ &\le (K + 1)|x^* - x|. \end{aligned}$$

The second inequality follows from the above bound $|f^m(x) - x| < \frac{1}{1-K}|f(x) - x|$. In other words, $f^m(x)$ lies in the ball $B_r(x)$ where $r = \frac{1}{1-K}|f(x) - x|$. Thus $x^* = \lim_{m\to\infty} f^m(x)$ lies in $B_r(x)$, which proves $|x^* - x| \le \frac{1}{1-K}|f(x) - x|$. **Q.E.D.**

Let us relate the Lipschitz constant to derivatives:

Lemma 6 *Suppose S is convex and $|f'(x)| \le K$ for all $x \in S$. Then f is Lipschitz with constant K.*

Proof. We use the fact that for $x \neq y$,

$$\frac{f(x) - f(y)}{x - y} = f'(\theta)$$

where $\theta \in [x, y]$. Since S is convex, $\theta \in S$ and so $|f'(\theta)| \leq K$. **Q.E.D.**

Let $N_f(x) = x - f(x)/f'(x)$ be the Newton iterator. In order to ensure that N_f is a contraction map, we what to bound $|N_f'(x)|$. The following quantities defined by Smale is useful to this end: assume $f^{(k)}(x)$ exists for all $x \in S$ and $k \geq 0$. If $f'(x) \neq 0$, we define the quantities:

- $\gamma(f, x) := \sup_{k \geq 1} \left| \frac{f^{(k)}(x)}{k! f'(x)} \right|^{1/(k-1)}$.
- $\beta(f, x) := \left| \frac{f(x)}{f'(x)} \right|$.
- $\alpha(f, x) := \beta(f, x)\gamma(f, x)$.

It follows that

$$|N_f(x)| = \left| \frac{f(x) f''(x)}{f'(x)^2} \right| = 2 \left| \frac{f(x)}{f'(x)} \cdot \frac{f''(x)}{2! f'(x)} \right| \leq 2\alpha(f, x).$$

This suggest that if $\alpha(f, x)$ is sufficiently small, then N_f is a contraction map in the neighborhood of x. But how small? It seems that $\alpha(f, x) < 1/2$ is a necessary condition.

Smale proves the following stronger statement. Let $x_0 = x$ and $x_{i+1} = N_f(x_i)$. We call x a **(relative) approximate zero** of f if

$$|x_n - x^*| \leq \left(\frac{1}{2}\right)^{2^n - 1} |x_0 - x^*|, \qquad n \geq 0.$$

Theorem 7 (Smale) *If $\alpha(f, x) < 0.04$ then x is an approximate zero of f.*

This kind of result is called a **point estimate** because we can predict the convergence of Newton iteration at x based on a single value $\alpha(f, x)$. Traditional estimates are based on bounds on f or f' over a region (e.g., Lemma 6).

For proof and details, see the Thesis Proposal Survey of Vikram (Dec 2004).

EXERCISE: The following are open problems.

(1) Give a simple proof of Smale's theorem.
(2) Improve the point estimate of Smale.
(3) Give a point estimate for guaranteeing convergence of $x_0, x_1, \ldots$, where $x_{i+1} = N_f(x_i)$.

§5. Approximate Newton Iteration

In Newton iteration, we are trying to find a zero x^* of a function $f(x)$, *i.e.*, $f(x^*) = 0$. We convert $f(x)$ to a function $F(x)$ such that $f(x^*) = 0$ iff $x^* = F(x^*)$. I.e., we convert a zero of $f(x)$ to a fixed point of $F(x)$.

The general problem we face is how to derive $F(x)$ from $f(x)$. The standard transformation is the Newton iterator

$$F(x) = x - \frac{f(x)}{f'(x)} \tag{5}$$

where $f'(x)$ denotes the derivative with respect to x. This iterator is valid provided the zero x^* we are looking for is not a **critical point** of f, *i.e.*, $f'(x^*) \neq 0$.

Issues: what if x^* is a critical point? What if we do not know á priori whether x^* is critical? We shall return to some of these issues.

In general, the construction of $F(x)$ is an interesting one. For instance, suppose we want to compute the reciprocal of a constant c. What $f(x)$ should we pick? If you pick $f(x) = x - (1/c)$, and apply the Newton transformation, you get $F(x) = 1/c$. It is the "perfect" iterator, but totally useless for our intended application! The $F(x)$ we use in the previous section applies the Newton transformation to the function $f(x) = (1/x) - c$.

Assuming (5), we may verify that

$$\begin{aligned} F(x^* + h) &= F(x^*) + F'(x^*)h + x^* + F^{(2)}(x^*)\frac{h^2}{2!} + \cdots + F^{(n)}(x^* + \theta h)\frac{h^2}{2!}. \\ &= x^* + F^{(n)}(x^* + \theta h)\frac{h^n}{n!} \end{aligned}$$

where $\theta \in [0, 1]$ and n is the smallest positive integer such that $F^{(n)}(x^*) \neq 0$. Note that $n \geq 2$ because $F(x^*) = x^*$ and $F'(x) = (ff'')/(f')^2$ vanishes at $x = x^*$.

Convergence of the Ideal Newton Iteration. We now show the standard result that Newton iteration has order 2 convergence. In the derivation, we assume $n = 2$ for specificity, since the general case should be clear. Furthermore, we often need temporary constants in the unit interval $[0, 1]$ or in $[-1, 1]$; we denote these quantities by some variant of θ such as θ', θ_i, etc.

We consider the sequence $x_0, x_1, x_2, \ldots$ where

$$x_{i+1} = F(x_i). \tag{6}$$

Assume $F(x)$ is a **contraction map** in the ball $B_r(x_0)$ of radius $r > 0$ about x_0. Then the sequence converges to a unique solution x^*. Suppose

$$x_i = x^* + \varepsilon_i. \tag{7}$$

Then we see that

$$\begin{aligned} x_{i+1} - x^* &= F(x_i) - x^* \\ &= F^{(2)}(x^* + \theta\varepsilon_i)(\varepsilon_i)^2/2 \quad \text{for some} \quad 0 \leq \theta \leq 1. \end{aligned}$$

Assuming $|\varepsilon_0| < 1$ and $|F^{(2)}(x)\varepsilon_0| < 2$ for all $x \in B_{\varepsilon_0}(x^*)$ (the ball about x^* of radius ε_0, we see that $F(x)$ is a contraction map in $B_{\varepsilon_0}(x^*)$ If $|F^{(2)}(x)|/2 \leq K$ in this ball, we conclude with the standard result about convergence of Newton iteration:

Theorem 8 *Let $x_{i+1} = F(x_i)$ and $x_i = x^* + \varepsilon_i$. If $|x^* - x_0| < 1$ and $|F^{(2)}(x)\varepsilon_0| < 2$ for all $x \in B(x^*, \varepsilon_0)$, then for all $i \geq 1$,*

$$0 < \varepsilon_i \leq K\varepsilon_0^{2^i}$$

It is worth nothing that ε_i is strictly positive for $i > 0$.

Convergence of Approximate Newton Iteration. The above iteration assumes exact operations. In practice, we do not compute the sequence $x_0, x_1, x_2, \ldots$, but some approximation

$$\widetilde{x}_0, \widetilde{x}_1, \widetilde{x}_2, \ldots$$

where each $\widetilde{x}_{i+1}$ is computed as in (6) except that the right-hand side is evaluated to a precision that depends on i. What is this precision?

It is clear that if we want to compute x^* to precision n then the last iteration must have at least precision n. Naively, we may require all previous iterations to also have precision n. This turns out to be enough, but even the correctness of this is not obvious. But we shall do much better – it suffices to about $cn/2$ precision in the computation of the previous iteration, and $cn/4$ precision in the iteration before that, etc. This remarkable property of the Newton iteration is usually described as "self-correcting".

Our plan is to begin with $\widetilde{x}_0$ sufficiently close to x^* (to be determined) and in the ith iteration, compute with precision $C2^i$ for some constant $C > 1$. Let $0 < \varepsilon < 1$ and $0 < c < 1$ be constants to be determined. Also, let $C = \lg(1/\varepsilon^2)$. Define our iteration by

$$\widetilde{x}_{i+1} = [F(\widetilde{x}_i)]_{C2^{i+1}}. \tag{8}$$

In other words, we evaluate $F(\widetilde{x}_i)$ to $C2^{i+1}$ bits of precision. We may express the error in $\widetilde{x}_i$ as

$$\begin{aligned} \delta_i &= \widetilde{x}_i - x^* \quad (i \ge 0) \\ &= c\theta_i \varepsilon^{2^i} \quad (|\theta_i| \le 1). \end{aligned}$$

Then we see that

$$\begin{aligned} \widetilde{x}_{i+1} - x^* &= F(\widetilde{x}_i)(1 \pm 2^{-C2^{i+1}}) - x^* \\ &= F(x^* + \delta_i)(1 \pm 2^{-C2^{i+1}}) - x^* \\ &= F^{(2)}(x^* \pm \delta_i)\frac{(\delta_i)^2}{2} \pm F(\widetilde{x}_i)2^{-C2^{i+1}} \\ &= F^{(2)}(x^* \pm \delta_i)\frac{(\theta_i c)^2 \varepsilon^{2^{i+1}}}{2} \pm F(\widetilde{x}_i)\varepsilon^{2^{i+2}} \qquad (\text{Since } 2^{-C} = \varepsilon^2) \\ &= c\varepsilon^{2^{i+1}}\left[F^{(2)}(x^* \pm \delta_i)\frac{(\theta_i)^2 c}{2} \pm \frac{F(\widetilde{x}_i)}{c}\varepsilon^{2^{i+1}}\right] \\ &= c\varepsilon^{2^{i+1}}\theta_{i+1} \end{aligned}$$

where the last equation serves as definition of θ_{i+1}.

It remains to choose c so that $|\theta_{i+1}| \le 1$. Since $F(\widetilde{x}_i) \to x^*$, we may assume pick c small enough that $|F(\widetilde{x}_i)|/c \le 1/2$ for all $i \ge 0$. Also, $F^{(2)}(x^* + \theta'\delta_i) \to F^{(2)}(x^*)$ as $i \to \infty$. So we can start at i large enough that $\left|F^{(2)}(x^* + \theta'\delta_i)\frac{(\theta_i)^2 c}{2}\right| \le 1/2$.

Note that we could choose $\varepsilon = 1/2$ and so $C = 1$.

EXERCISE:
(1) Implement this algorithm in Core Library where $F(x) = x - f(x)/f'(x)$ and $f(x)$ is any integer polynomial.
(2) Suppose $f(x)$ is a polynomial of degree d and coefficients are m-bit integers. What is the complexity of computing an n-bit relative approximation to a root of $f(x)$, assuming you can get an good enough initial approximation in constant time.
(3) Give a one-point estimate analogous to Smale in the case of approximate Newton iteration. More precisely, determine a global constant C_0 such that for $\alpha(f, x) < C_0$, the approximate Newton iteration starting at x yields quadratic convergence.

§6. Approximate Discrete Newton Iteration.

We make two extensions of the above analysis. The first is that we want to replace the computation of $f'(x)$ by the discretized approximation, $f'(x) \approx (f(x+h) - f(x))/h$ for small $|h|$. Second, we want to replace $f(x)$ by its absolute approximation $\widetilde{f}(x;\ell)$ that satisfies the bound $|\widetilde{f}(x;\ell) - f(x)| \le 2^{-\ell}$.

First we consider the discretized version of Newton,

$$x_{i+1} = x_i - \frac{h_i f(x_i)}{f(x_i + h_i) - f(x_i)} \tag{9}$$

where

$$h_i = c 2^{-2^{i+1}} \tag{10}$$

for some $c > 0$. We shall determine how small c needs to be. Assuming that $f'(x)$ is Lipschitz with constant K, we obtain

$$f'(x+h) - f'(x) = Kh\theta, \qquad |\theta| \le 1.$$

Therefore

$$\begin{aligned} \frac{f(x+h) - f(x)}{h} &= f'(x + \theta h) \\ &= f'(x) + K\theta' h, \quad (|\theta'| \le 1). \\ x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i) + K\theta' h_i}. \end{aligned}$$

Suppose

$$x_i = x^* \pm 2^{-2^{i+1}}.$$

Then

$$f'(x_i) + K\theta' h_i = f'(x_i)\left[1 + \frac{1}{2}\theta'' 2^{-2^{i+1}}\right]$$

provided $cK \le |f'(x_i)|/2$. So

$$\begin{aligned} x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)\left(1 + \theta'' 2^{-1-2^{i+1}}\right)} \\ &= x_i - \frac{f(x_i)}{f'(x_i)}\left(1 + \theta'' 2^{-2^{i+1}}\right). \end{aligned}$$

Assume $x^* \neq 0$, and as $f(x_i)/f'(x_i) \to 0$, we can make $|x_i| \ge |x^*|/2$ and $|x^*|/2 \ge 2|f(x_i)/f'(x_i)|$. Hence

$$\begin{aligned} x_{i+1} &= \left(x_i - \tfrac{f(x_i)}{f'(x_i))}\right)\left[1 + \theta'' 2^{-2^{i+1}}\right] \quad \text{for some } |\theta''| \le 1. \\ &= [F(x_i)]_{2^{i+1}}. \end{aligned}$$

But this is just the iteration we analyzed under approximate Newton, and it has quadratic convergence.

Our final step is to replace the assumption that $f(x)$ and the operations of $F(x)$ is exact. We leave this as an exercise.

EXERCISE:
Provide the approximate version of the discretized Newton iteration (9). You must explicitly give the precision for evaluating each $f(x)$ and for performing each arithmetic operation. Prove a quadratic convergence using your iteration.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

[2] R. P. Brent. The complexity of multiple-precision arithmetic. In R. S. Anderssen and R. P. Brent, editors, *The Complexity of Computational Problem Solving*, pages 126–165. University of Queensland Press, Brisbane, 1976. Retyped and postscript added 1999.

[3] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *J. of the ACM*, 23:242–251, 1976.

[4] J. Choi, J. Sellen, and C. Yap. Approximate Euclidean shortest path in 3-space. *Int'l. J. Comput. Geometry and Appl.*, 7(4):271–295, 1997. Also: 10th ACM Symp. on Comp. Geom. (1994)pp.41–48.

[5] D. Shanks. *Solved and Unsolved Problems in Number Theory.* Dover Publications, Inc, New York, NY, 1993.

Added Notes

- There are 2 views of bigfloats: so far, they are viewed as exact numbers. In the exact view, the equation $\langle e, f\rangle = \langle e, 2f\rangle$ makes sense. On the other hand, we can view a bigfloat as implicitly representing an number whose list significant bit may off by 1/2. For instance, the bigfloat number $(1.010)_2$ represents the "implicit interval" $[(1.0011)_2, (1.0101)_2]$. The size of this interval is 2^{-3}. On the other hand, the value $(1.01)_2$ represents the implicit interval $[(1.001)_2, (1.011)_2$ whose size is 2^{-2}. Thus $\langle e, f\rangle$ and $\langle e, 2f\rangle$ represents different implicit intervals. In general, the size of the implicit interval is equal to $2^{1-\mathrm{msb}(f)}$. Hence we define the **precision** of a bigfloat representation $\langle e, f\rangle$ to be $\mathrm{msb}(x) - 1$. In other words, to have precision n, the fractional part f needs $n + 1$ bits. Thus "precision" here refers to relative precision.

- In Sutherland and Kim (2001), the current best bounds for finding the zeros of an integer polynomial are surveyed. But Brent's bounds, suitably generalized, appear to beat some of the newer bounds.