Chapter 1

Preliminaries

1.1 Review of calculus

Theorem 1.1 (Mean value theorem) *If $f \in C[a,b]$ is differentiable in (a,b), then there exists a point $c \in (a,b)$ such that*

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

Theorem 1.2 (Mean value theorem for integrals) *Let $f \in C[a,b]$ and let g be integrable on $[a,b]$ and having constant sign. Then, there exists a point $c \in (a,b)$ such that*

$$\int_a^b f(x)g(x)\,dx = f(c)\int_a^b g(x)\,dx.$$

If, in particular, $g(x) = 1$, then there exists a point where f equals to its average on the interval.

Theorem 1.3 (Taylor's theorem) *let $f \in C^n[a,b]$ with $f^{(n+1)}$ existing on $[a,b]$ (but not necessarily differentiable). Let $x_0 \in [a,b]$. Then, for every $x \in [a,b]$ there exists a point $\xi(x)$ between x_0 and x such that*

$$f(x) = P_n(x) + R_n(x),$$

where

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

is the n***-th Taylor polynomial of*** f ***about*** x_0*, and*

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}$$

is the ***remainder term****.*

Comment: It is often useful to think of x as x_0+h; we know the function and some of its derivatives at a point x_0 and we want to estimate it at another point at a distance h. Then,

$$f(x_0 + h) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}h^k + \frac{f^{(n+1)}(x_0 + \theta(h)h)}{(n+1)!}h^{n+1},$$

where $0 < \theta(h) < 1$. Often, we approximate the function f by its n-th Taylor polynomial, in which case we refer to the remainder as the ***truncation error***.

✎ *Exercise 1.1* (a) Approximate the function $f(x) = \cos x$ at the point $x = 0.01$ by its second and third Taylor polynomials about the point $x_0 = 0$. Estimate the error. (b) Use the third Taylor polynomial to estimate $\int_0^{0.1} \cos x\, dx$. Estimate the error.

Theorem 1.4 (Multi-dimensional Taylor theorem) TO COMPLETE

✎ *Exercise 1.2* Let k be a positive integer and let $0 < \alpha < 1$. To what class of functions $C^n(\mathbb{R})$ does the function $x^{k+\alpha}$ belong?

✎ *Exercise 1.3* For small values of x it is standard practice to approximate the function $\sin x$ by x itself. Estimate the error by using Taylor's theorem. For what range of x will this approximation give results accurate to six decimal places?

✎ *Exercise 1.4* Find the first two terms in the Taylor expansion of $x^{1/5}$ about the point $x = 32$. Approximate the fifth root of 31.999999 using these two terms in the series. How accurate is your answer?

✎ *Exercise 1.5* The ***error function*** defined by

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$$

gives the probability that a trial value will lie within x units of the mean, assuming that the trials have a standard normal distribution. This integral cannot be evaluated in terms of elementary functions.

① Integrate Taylor's series for e^{-t^2} about $t = 0$ to show that

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)\, k!}$$

(more precisely, use the Taylor expansion for e^{-x}).

② Use this series to approximate $\operatorname{erf}(1)$ to within 10^{-7}.

1.2 Order of convergence

Definition 1.1 (Rate of convergence) *Let (x_n) be a converging sequence with limit L. Its rate of convergence is said to be (at least)* **linear** *if there exist a constant $C < 1$ and an integer N, such that for all $n \geq N$,*

$$|x_{n+1} - L| \leq C\, |x_n - L|.$$

The rate of convergence is said to be (at least) **superlinear** *if there exists a sequence $\epsilon_n \to 0$, such that for all $n \geq N$,*

$$|x_{n+1} - L| \leq \epsilon_n\, |x_n - L|.$$

The rate of convergence is said to be of order (at least) α if there exists a constant C such that

$$|x_{n+1} - L| \leq C\, |x_n - L|^{\alpha}.$$

Comment: Can be generalized for sequences in a normed vector space.

Example 1.1 ① The convergence of $(1 + 1/n)^n$ to e satisfies

$$\frac{|x_{n+1} - e|}{|x_n - e|} \to 1,$$

i.e., the rate of convergence is worse than linear.

② The sequence $2^{-n}/n$ is another example of a linear rate of convergence.

③ Consider the sequence (x_n) defined recursively by

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n},$$

with $x_1 = 1$. Then

$$\begin{aligned} 2x_n x_{n+1} &= x_n^2 + 2 \\ 2x_n x_{n+1} - 2\sqrt{2}x_n &= (x_n - \sqrt{2})^2 \\ 2x_n(x_{n+1} - \sqrt{2}) &= (x_n - \sqrt{2})^2, \end{aligned}$$

i.e.,

$$x_{n+1} - \sqrt{2} = \frac{(x_n - \sqrt{2})^2}{2\,x_n}.$$

Clearly, if the distance of the initial value from $\sqrt{2}$ is less than $1/2$, then the sequence converges. The rate is by definition quadratic. The following table gives the distance of x_n from $\sqrt{2}$ for various n

n	$x_n - \sqrt{2}$
1	-0.41×10^{-1}
2	8.58×10^{-2}
3	2.5×10^{-3}
4	2.12×10^{-6}
5	1.59×10^{-12}

Definition 1.2 *Let (x_n) and (y_n) be sequences. We say that $x_n = O(y_n)$ if there exist C, N such that*

$$|x_n| \leq C|y_n|$$

for all $n \geq N$. We say that $x_n = o(y_n)$ if

$$\lim_{n\to\infty} \frac{x_n}{y_n} = 0.$$

Comments:

① Again generalizable for normed linear spaces.

② If $x_n = O(y_n)$ then there exists a $C > 0$ such that $\limsup x_n/y_n \leq C$.

③ $f(x) = O(g(x))$ as $x \to x_0$ means that there exists a neighborhood of x_0 in which $|f(x)| \leq C\,|g(x)|$. Also, $f(x) = o(g(x))$ if for every $\epsilon > 0$ there exists a neighborhood of x_0 where $|f(x)| \leq \epsilon |g(x)|$.

Example 1.2 ① Show that $x_n = O(z_n)$ and $y_n = O(z_n)$ implies that $x_n + y_n = O(z_n)$.

② Show that if $\alpha_n \to 0$, $x_n = O(\alpha_n)$ and $y_n = O(\alpha_n)$, then $x_n y_n = o(\alpha_n)$.

✎ *Exercise 1.6* Prove that if $x_n = O(\alpha_n)$ then $\alpha_n^{-1} = O(x_n^{-1})$. Prove that the same holds for the o-relation.

✎ *Exercise 1.7* Let n be fixed. Show that

$$\sum_{k=0}^{n} x^k = \frac{1}{1-x} + o(x^n)$$

as $x \to 0$.

1.3 Floating point arithmetic

A real number in scientific notation has the following representation,

$$\pm(\text{fraction}) \times (\text{base})^{(\text{exponent})}.$$

Any real number can be represented in this way. On a computer, the base is always 2. Due to the finiteness of the number of bits used to represent numbers, the range of fractions and exponents is limited. A ***floating point numbers*** is a number in scientific notation that fits the format of a computer word, e.g.,

$$-0.1101 \times 2^{-8}.$$

A floating point is called ***normalized*** if the leading digit of the fraction is 1.

Different computers have different ways of storing floating point numbers. In addition, they may differ in the way they perform arithmetic operations on floating point numbers. They may differ in

① The way results are rounded.

② The way they deal with numbers very close to zero (underflow).

③ The way they deal with numbers that are too big (overflow).

④ The way they deal with operations such as $0/0$, $\sqrt{-1}$.

The most common choice of floating point arithmetic is the IEEE standard.

Floating point numbers in the IEEE standard have the following representation,

$$(-1)^s\,(1+f)\times 2^{e-1023},$$

where the ***sign***, s, takes one bit, the ***fraction***, f, takes 52 bits, and the ***exponent***, e, takes 11 bits. Because the number is assumed normalized, there is no need to store its leading one. We note the following:

① The exponent range is between $2^{-1023}\approx 10^{-308}$ (the underflow threshold), and $2^{1024}\approx 10^{308}$ (the overflow threshold).

② Let x be a number within the exponential range and $\mathrm{fl}(x)$ be its approximation by a floating point number. The difference between x and $\mathrm{fl}(x)$ scales with the exponent. The ***relative representation error***, however, is bounded by

$$\frac{|x-\mathrm{fl}(x)|}{|x|}\le 2^{-53}\approx 10^{-16},$$

which is the relative distance between two consecutive floating point numbers. The bound in the relative representation error is known as the ***machine-ϵ***.

IEEE arithmetic also handles $\pm\infty$ and NaN with the rules

$$\frac{1}{0}=\infty,\quad \infty+\infty=\infty,\quad \frac{x}{\pm\infty}=0,$$

and

$$\infty-\infty=\mathrm{NaN},\quad \frac{\infty}{\infty}=\mathrm{NaN},\quad \sqrt{-1}=\mathrm{NaN},\quad x+\mathrm{NaN}=\mathrm{NaN}.$$

Let $\odot$ be any of the four arithmetic operations, and let a, b be two floating point numbers. After the computer performs the operation $a\odot b$, the result has to be stored in a computer word, introducing a ***roundoff error***. Then,

$$\frac{a\odot b-\mathrm{fl}(a\odot b)}{a\odot b}=\delta,$$

where $|\delta| \leq \epsilon$. That is

$$\mathrm{fl}(a \odot b) = a \odot b\,(1+\delta).$$

1.4 Stability and condition numbers

Condition numbers Let X, Y be normed linear spaces and $f : X \mapsto Y$. Suppose we want to compute $f(x)$ for some $x \in X$, but we may introduce errors in x and compute instead $f(x+\delta x)$, where $\|\delta x\|$ is "small". A function is called ***well-conditioned*** if small errors in its input result in small errors in its output, and it is called ***ill-conditioned*** otherwise.

Suppose that f is differentiable. Then, under certain assumptions,

$$f(x+\delta x) \approx f(x) + Df(x)\,\delta x,$$

or,

$$\|f(x+\delta x) - f(x)\| \approx \|Df(x)\|\|\delta x\|.$$

The absolute output error scales like the absolute input error times a multiplier, $\|Df(x)\|$, which we call the ***absolute condition number of*** f ***at*** x. In addition,

$$\frac{\|f(x+\delta x) - f(x)\|}{\|f(x)\|} \approx \frac{\|Df(x)\|\|x\|}{\|f(x)\|} \cdot \frac{\|\delta x\|}{\|x\|}.$$

Here we call the multiplier of the relative input and output errors the ***relative condition number of*** f ***at*** x. When the condition number is infinite the problem (i.e., the function) is called ***ill-posed***. *The condition number is a characteristic of the problem, not of an algorithm.*

Backward stability Suppose next that we want to compute a function $f(x)$, but we use an approximating algorithm which yields instead a result $\mathrm{alg}(x)$. We call $\mathrm{alg}(x)$ a ***backward stable algorithm for*** f, if there exists a "small" δx such that

$$\mathrm{alg}(x) = f(x+\delta x).$$

I.e., $\mathrm{alg}(x)$ gives the exact solution for a slightly different problem. If the algorithm is backward stable, then

$$\mathrm{alg}(x) \approx f(x) + Df(x)\delta x,$$

i.e.,

$$\| \operatorname{alg}(x) - f(x)\| \approx \|Df(x)\|\|\delta x\|,$$

so that the output error is small provided that the problem is well-conditioned. To conclude, *for an algorithm to gives accurate results, it has to be backward stable and the problem has to be well-conditioned.*

Example 1.3 Consider polynomial functions,

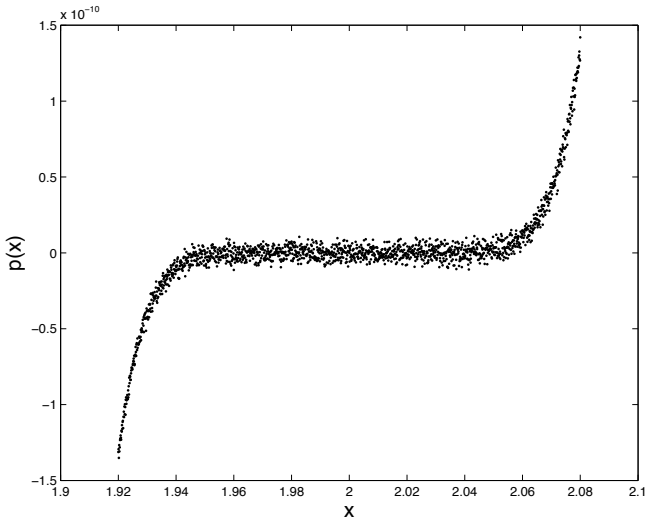
$$p(x) = \sum_{i=0}^{d} a_i x^i,$$

which are evaluated on the computer with ***Horner's rule***:

Algorithm 1.4.1: POLYNOMIAL EVALUATION(x)

$p = a_d$
for $i = d - 1$ **downto** 0
  do $p = x * p + a_i$
return (p)

The graph below shows the result of such a polynomial evaluation for $x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512 = (x-2)^9$, on the interval $[1.92, 2.08]$.



We see that the behavior of the function is quite unpredictable in the interval $[1.05, 2.05]$, and merits the name of ***noise***. In particular, try to imagine finding the root of $p(x)$ using the bisection algorithm.

Let's try to understand the situation in terms of condition numbers and backward stability. First, we rewrite Horner's rule as follows:

Algorithm 1.4.2: POLYNOMIAL EVALUATION(x)

$p = a_d$
for $i = d - 1$ **downto** 0
  do $p_i = x * p_{i+1} + a_i$
return (p_0)

And then, insert a multiplicative term of $(1 + \delta_i)$ each time a floating point operations is done:

Algorithm 1.4.3: POLYNOMIAL EVALUATION(x)

$p = a_d$
for $i = d - 1$ **downto** 0
  do $p_i = [x * p_{i+1}(1 + \delta_i) + a_i](1 + \delta_i')$
return (p_0)

What do we actually compute? The coefficients a_i are in fact $a_i(1 + \delta_i')$, and x is really $x(1 + \delta_i)(1 + \delta_i')$, so that

$$p_0 = \sum_{i=0}^{d} \left[(1 + \delta_i') \prod_{j=0}^{i-1} (1 + \delta_j)(1 + \delta_j') \right] a_i x^i.$$

This expression can be simplified,

$$p_0 = \sum_{i=0}^{d} (1 + \bar{\delta}_i) a_i x^i,$$

where

$$(1 + \bar{\delta}_i) = (1 + \delta_i') \prod_{j=0}^{i-1} (1 + \delta_j)(1 + \delta_j').$$

And we use the fact that

$$(1+\bar{\delta}_i) \le (1+\epsilon)^{1+2i} \le 1+2d\epsilon+O(\epsilon^2)$$
$$(1-\bar{\delta}_i) \ge (1-\epsilon)^{1+2i} \ge 1-2d\epsilon+O(\epsilon^2)$$

or $|\bar{\delta}_i| \le 2d\epsilon$.

Thus, our algorithm computes exactly a polynomial with slightly different coefficients $\bar{a}_i = (1+\bar{\delta}_i)a_i$, i.e., it is ***backward stable*** (the exact solution of a slightly different problem).

With that, we can compute the error in the computed polynomial:

$$\begin{aligned} |p(x)-p_0(x)| &= \left| \sum_{i=0}^{d} (1+\bar{\delta}_i)a_i x^i - \sum_{i=0}^{d} a_i x^i \right| \\ &= \left| \sum_{i=0}^{d} \bar{\delta}_i a_i x^i \right| \\ &\le 2d\epsilon \sum_{i=0}^{d} |a_i x^i|. \end{aligned}$$

This error bound is in fact attainable if the $\bar{\delta}_i$ have signs opposite to that of $a_i x^i$. The relative error (bound) in polynomial evaluation is

$$\frac{|p(x)-p_0(x)|}{|p(x)|} \le 2d\epsilon \, \frac{\sum_{i=0}^{d} |a_i x^i|}{|\sum_{i=0}^{d} a_i x^i|}.$$

Since $2d\epsilon$ is a measure of the input error, the multiplier $\sum_{i=0}^{d} |a_i x^i| / |\sum_{i=0}^{d} a_i x^i|$ is the relative condition number for polynomial evaluation. The relative error bound can be computed directly:

Algorithm 1.4.4: POLYNOMIAL EVALUATION ERROR(x)

$p = a_d$
$\hat{p} = |a_d|$
for $i = d-1$ **downto** 0
  do $\begin{cases} p = x * p + a_i \\ \hat{p} = |x| * \hat{p} + |a_i| \end{cases}$
return $(2d\epsilon\, \hat{p})$

From the relative error we may infer, for example, a lower bound number of correct digits,

$$n = -\log_{10} \frac{|p|}{\hat{p}}.$$

In the plot below we show this lower bound along with the actual number of correct digits. As expected, the relative error grows infinite at the root.