Chapter 1

Nonlinear Equations

Many scientific problems result in a **root finding problem** or an **optimization problem**.

1.1 Root Finding

Root finding problem is to find a solution(**root** or **zero**) x^* of $f(x) = 0$. Or a solution of a system of equations:

$$\mathbf{F}(\mathbf{x}^*) = 0 \tag{1.1}$$

where

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2, \cdots, x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) \end{pmatrix}$$

What kind of f ?

(1) Linear polynomial $f(x) = ax + b$, Quadratic polynomial $f(x) = ax^2 + bx + c$

(2) Cubic, Quartic equations, ... , nonfactorizable polynomials

(3) Non polynomials(still smooth or analytic functions)

(4) Continuous functions possibly nonsmooth

(5) A function given by discrete data: Suppose the values $(f(u_1), f(u_2), \cdots, f(u_k))$ are given, and we are asked to find the zero of $f(x)$. What can we do?

(6) A function is given implicitly, say by the solution of a differential equation.

We assume f is continuous defined on an interval from now on.

Exhaustive Search

Suppose we have found $f(a)f(b) < 0$. The interval $[a, b]$ is called a **bracket**. To find all the zeros, we divide the interval into

$$x_0, x_1 = x_0 + h, x_2 = x_0 + 2h, \cdots,$$

and search the value of f on each subintervals. Here $h > 0$ is called the **step size, grid size**. This is a time consuming and may miss certain zeros. There is no guarantee to find a root.

Bisection Method

Assume we have found a bracket $[x_0, x_1]$, i.e, $f(x_0)f(x_1) < 0$. Then let

$$x_2 = \frac{x_0 + x_1}{2}.$$

If $f(x_2) = 0$, we are done. Otherwise, either $f(x_0)f(x_2) < 0$ or $f(x_2)f(x_1) < 0$. In the first case, $[x_0, x_2]$ is a new bracket and in the latter case $[x_2, x_1]$ is a new bracket. In either case, the size of a new bracket is halved. Repeat this process until some **stopping criterion** is satisfied.

Stopping criterion

We have to stop the program if any of the following conditions are met.

(1) the size of bracket is less than certain specified TOL ($\epsilon_1 = 10^{-12}$).

(2) $|f(x_i)| < \epsilon_2$.

(3) the number of iteration is too large.(We do not want the program run indefinitely!)

Algorithm Bisection

$u \leftarrow f(a)$
$v \leftarrow f(b)$

$e \leftarrow b - a$
If $u = v$, stop
for $k = 1, 2, \cdots, maxit$ do
  $e \leftarrow e/2$
  $c \leftarrow a + e$
  $w \leftarrow f(c)$
  If $|e| < \epsilon_1$ or $|w| < \epsilon_2$, stop
  If sgn $(w) \neq$ sgn (u) then
    $b \leftarrow c$
    $v \leftarrow w$
  else
    $a \leftarrow c$
    $u \leftarrow w$
  endif
enddo

The bisection method is guaranteed to produce a root. But rather slow.

(1) What is a reasonable stopping criterion?

(2) Under what case does this algorithm work well or go wrong?(or behave bad)

(3) Is there a condition under which the convergence can be guaranteed?

Inverse Linear Method-Regula Falsi or False Position

Given a bracket $[x_0, x_1]$, we interpolate f by a linear function and let the zero of the linear function be the approximate root of f. Using slope-intercept form, we see the linear function approximating f is

$$y = L(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) + f(x_0). \tag{1.2}$$

Solving $L(x) = 0$, we obtain an approximate zero which we define as x_2:

$$x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0). \tag{1.3}$$

We repeat the same process until some criterion is met. This is again guaranteed to converge: Eq (1.2) can be rewritten as a function of y:

$$x = \frac{x_1 - x_0}{f(x_1) - f(x_0)}(y - f(x_1)) + x_1. \tag{1.4}$$

Things to think about:

MATLAB 1.1

Try bisection on $f(x) = \sin(e^x)$.

```
>> x=0: 0.1 : 3
```

to make a vector whose entries ranges from 0 to 3 in steps of 0.1. Then

```
>> y= sin (exp(x));
```

assigns values of $sin(exp(x))$ to a vector y. Here the semi-colon suppresses the printing of result. The arguments of sin are assumed to be in radians.

```
>> y, plot(x,y)
```

to see the plot. Change the step size and redo to see a finer graph. Try a bisection and then say 'clear' in the matlab window and start over by inverse linear method.

1.2 Newton's Method

Let us count the number of iterations in bisection method to reduce the accuracy by a factor of 10. So

$$\frac{\epsilon}{2^m} = \frac{\epsilon}{10}.$$

Thus

$$m = \log_2(10) \approx 3.3219.$$

This method is used in many hand held calculator because of its simplicity and guaranteed convergence. However, it is slow and we need an initial bracket to start with.

Secant Method

We may use the inverse linear iteration when $[x_0, x_1]$ do not form a bracket. This is an extrapolation outside the interval. Called the **secant method**.

See example 1.2.1. It shows better convergence: Disadvantage: This is not guaranteed to converge.

What happens if we let x_1 slide toward x_0?

$$x = x_0 - \lim_{x_1 \to x_0} \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0) = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{1.5}$$

This is a Newton's method: With an initial point x_0, we have

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}.$$

Newton method is fast, but still may diverge.(or oscillate)

Difficulties

If $f'(x^*) = 0$ the method breaks down.

(1) Can we find all the zeros?

(2) We do not know how many zeros there are in general.

(3) The case of densely packed zeros.

(4) The function is very flat in some domain.

(5) The function is positive in most part of the domain except on a very narrow region. In this case finding a bracket is difficult.

MATLAB 1.2

inline function. Let us try Newton's method to $f(x) = \frac{1}{2}xe^x - 2x^2$.

```
>> f=inline(' .5*x. exp(x)-2*x.^2')
>> fp=inline('.5*exp(x)+.5*x.*exp(x)-4*x')
```

Here $x.$ leave the possibility of using a vector. Try either of the two:

```
>> feval(f,3)
>> f(3)
```

Start with $x_0 = 0.4$.

```
>> x0=.4;
>> x0=x0-f(x0)/fp(x0)
>> f(x0)
```

You will get $x_1 = 0.3611$. Mover the cursor up and change the value of x_0 by 0.3611 you will get $x_2 = 0.3574$.

1.3 Fixed Point Theorem

An iteration of the form

$$x_k = g(x_{k-1}) \tag{1.6}$$

is called a **fixed point iteration** and any point x^* such that

$$x^* = g(x^*)$$

is called a **fixed point**.

If the Newton's method converges, the fixed point iteration is converging to a fixed point of N_f. If g is continuous, then the sequence x_k converges to a fixed point.

Theorem 1.3.1. *(Fixed Point Theorem) If g is continuous on $[a, b]$ and $g : [a, b] \to [a, b]$ then g has a fixed point in $[a, b]$. If, in addition g is differentiable, on (a, b) and there is a λ, $0 < \lambda < 1$ such that*

$$|g'(x)| \leq \lambda, \ \forall x \in (a, b),$$

then this fixed point is unique and the fixed point iteration $x_k = g(x_{k-1})$ converges to it for any $x_0 \in [a, b]$.

Proof. Let $h(x) = g(x) - x$. If $h(a) = 0$ or $h(b) = 0$ we have a fixed point. Thus we may assume $h(a) \neq 0$ and $h(b) \neq 0$. Then

$$h(a) = g(a) - a > 0$$

since $g(a) \in (a, b)$ and similarly

$$h(b) = g(b) - b < 0.$$

Thus by intermediate value theorem, there is a value $c \in (a, b)$ such that $g(c) = 0$.

Now suppose in addition that g is differentiable on (a, b) and there is a λ, $0 < \lambda < 1$ such that $|g'(x)| \leq \lambda$, $\forall x \in (a, b)$. If p, q are two distinct fixed points of g, then by MVT

$$\frac{g(p) - g(q)}{p - q} = g'(\theta)$$

for some $\theta \in (a, b)$. Since $g(p) = p$ and $g(q) = q$

$$\frac{g(p) - g(q)}{p - q} = g'(\theta) = \frac{p - q}{p - q} = 1.$$

This is a contradiction. Thus we must have $p = q$.

It remains to show the fixed point iteration converges to the solution. Let $x^* = g(x^*)$ be the fixed point and let x_0 be any point. Then by the MVT, there exists some $\theta \in (a, b)$ and

$$\begin{aligned} |x_n - x^*| &= |g(x_{n-1}) - g(x^*)| \\ &= g'(\theta)(x_{n-1} - x^*) \\ &\leq |g'(\theta)||x_{n-1} - x^*| \\ &\leq \lambda |x_{n-1} - x^*| \\ &\leq \lambda^n |x_0 - x^*| \\ &\to 0. \end{aligned}$$

□

Example 1.3.2. Note that $x^2 = k$ is equivalent to $x = \frac{1}{2}(x + \frac{k}{x})$. Thus we can use fixed point iteration to $g(x) = \frac{1}{2}(x + \frac{k}{x})$ to find the square root of a positive number k. Let $x_0 = 1$.

$$\begin{aligned} x_1 &= \frac{1}{2}\left(x_0 + \frac{2}{x_0}\right) = 1.5 \\ x_2 &= \frac{1}{2}\left(x_1 + \frac{2}{x_1}\right) = 1.41666666 \\ x_3 &= \frac{1}{2}\left(x_2 + \frac{2}{x_2}\right) = 1.414215682 \\ &= \cdots \end{aligned}$$

Example 1.3.3. Next we let $f(x) = e^x + x^2 - 5x$. It has two zeros one near $x = 1/3$ another $x = 7/4$.

$$\begin{aligned} e^x + x^2 - 5x &= 0 \\ 5x &= e^x + x^2 \\ x &= \frac{1}{5}(e^x + x^2). \end{aligned}$$

Try the iteration with initial guess $x_0 = 1/3$ and $x_0 = 7/4$. What do you see and why?

Newton's method as a fixed point iteration

Newton's method

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} := g(x_{k-1})$$

$$\begin{aligned} g'(x) &= 1 - \left(\frac{f(x)}{f'(x)}\right)' \\ &= \frac{f(x)f''(x)}{[f'(x)]^2}. \end{aligned}$$

If x^* is a simple zero, $g'(x^*) = 0$. By continuity, $|g'(x)| < 1$ in a nhd of x^*.

1.4 Quadratic Convergence of Newton's Method

Stopping criteria: Kind of errors:

(1) absolute error : $|x^* - x_n|$

(2) relative error : $\frac{|x^* - x_n|}{|x^*|}$(when x^* is order of millions)

(3) residual error : $|f(x_n)|$

In practice we do not know exact solution. Thus it is reasonable to stop if $|x_{n+1} - x_n|$ or $\frac{|x_{n+1} - x_n|}{|x_{n+1}|}$ is sufficiently small. Smallness of residual error does not imply smallness in ther other errors. Thus using the residual error as stopping criteria is not recommended. It should be only used in conjunction with at least one other criterion.

Order of convergence

We say a sequence $\{x_n\}$ with limit x^* has order of convergence 1 if there is a number $0 < M < 1$ such that

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|} = M.$$

A fixed point iteration usually has linear convergence and

$$e_{n+1} \approx \lambda e_n.$$

In Newton's we have seen

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|} = 0.$$

IN this case it is called **superlinear**. If there is $p > 1$ such that

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^p} = M$$

for some $M > 0$(here we do not require $M < 1$).

Order of convergence of Newton's Method

Newton's method can be viewed as a fixed point iteration $x_{n+1} = g(x_n)$ where

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

We see

$$\begin{aligned} g'(x) &= 1 - \left(\frac{f(x)}{f'(x)}\right)' \\ &= \frac{f(x)f''(x)}{[f'(x)]^2} \\ &= \cdots \\ g''(x^*) &= \frac{f''(x^*)}{f'(x^*)} \end{aligned}$$

$$\begin{aligned} g(x_n) &= g(x^*) + g'(x^*)(x_n - x^*) + g''(\xi_n)\frac{(x_n - x^*)^2}{2} \\ &= g(x^*) + g''(\xi_n)\frac{(x_n - x^*)^2}{2} \\ &= x^* + g''(\xi_n)\frac{(x_n - x^*)^2}{2}. \end{aligned}$$

Hence

$$\begin{aligned} g(x_n) - x^* &= g''(\xi_n)\frac{(x_n - x^*)^2}{2} \\ x_{n+1} - x^* &= \frac{1}{2}g''(x_n)(x_n - x^*)^2 \\ \frac{x_{n+1} - x^*}{(x_n - x^*)^2} &= \frac{1}{2}g''(x_n). \end{aligned}$$

By the same way, one can see the order will be p if $g'(x^*) = \cdots = g^{p-1}(x^*) = 0$ and $g^p(x^*) \neq 0$.

Also, check that $g''(x^*) \neq 0$ when x^* is not a multiple root.

Discussion on the behavior of Newton's method

Summary: Newton's Method

(1) converges quadratically near a simple zero if $f \in C^2$.

(2) If the zero is non simple, it still converges but linearly.

(3) Quadratic convergence is what we usually expect for a good method.

(4) How good is quadratic convergence? See example.

(5) No guarantee for convergence.(so safeguard combined with bisection method may be useful)

(6) In the first few steps, it may fluctuate but in most cases eventually converges.

(7) May oscillate for some initial point.

Suppose $M = 1$ and $e_n = 10^{-4}$. Then

$$e_{n+1} = e_n^2 = 10^{-8}.$$

Once the quadratic convergence begins, the Newton's methods doubles the number of correct significant figures in approximation at every iteration.

1.5 Variants of Newton's Method

A few **difficulties** for Newton's method

(1) It may not be possible to find a formula for f'.

(2) It may be possible to find a formula for f' but may be too expansive to compute.

(3) Newton's method may fail to converge.

Finite Differences

We may replace $f'(x_k)$ by

$$f'(x_k) \approx \frac{f(x_k + h) - f(x_k)}{h}$$

for some h.

Quasi Newton Method

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{s_k}.$$

(1) One choice is $s_k = \frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}$(secant method)

(2) another choice is to use **constant slope** $s_k = c$.

Damping

Failure of Newton's method may happen for example, when we overshoot the value: Try Newton's method for $f(x) = 1/x - 10$ with $x_0 = 10$. We will see it fails to converge.

Thus we use damping:

$$x_k = x_{k-1} - \mu_k \frac{f(x_{k-1})}{f'(x_{k-1})} \tag{1.7}$$

for $0 < \mu_k < 1$ and $\mu_k \to 1$ as $k \to \infty$.

Mixing Methods

Start with bisection (guaranteed) and after a few iteration **switch to Newton's method**(fast).

Safeguarded Newton's Method

Combine Newton's Method with bisection from the start. Start with a bracket $[x_0, x_1]$. If each Newton's step gives new iterate x_{new}, if $x_{new} \in [x_0, x_1]$ use either $[x_0, x_{new}]$ or $[x_{new}, x_1]$ (whichever is a bracket). Otherwise throw away x_{new} and use bisection.

Matlab 1.5

```
>> realmax
>> realmin
>> 1/0
```

Anything bigger than the 'realmax' is set to be Inf, (operation **overflow**), Anything smaller than the 'realmin' is set to be zero, (operation **underflow**), 'NaN' means not a number.

1.6 Brent method — Variation of Quadratic Methods

General advantages of Quadratic Methods:

(1) No need to compute derivative.

(2) Still fast enough, order approx. 1.84.

(3) Complex roots can be found if started with complex number.

Quadratic method

Suppose $[x_0, x_2]$ is a bracket of f. Choose any point x_1 in $[x_0, x_2]$ and use find a quadratic poly Q that interpolates f with these data. Now use the zeros of

Q. But Q has two roots, Which one to choose ? Let $Q(x) = \alpha(x - x_2)^2 + \beta(x - x_2) + \gamma$.

$$\begin{aligned} f(x_0) &= \alpha(x_0 - x_2)^2 + \beta(x_0 - x_2) + \gamma \\ f(x_1) &= \alpha(x_1 - x_2)^2 + \beta(x_1 - x_2) + \gamma \\ f(x_2) &= \alpha(x_2 - x_2)^2 + \beta(x_2 - x_2) + \gamma = \gamma. \end{aligned}$$

Solving for α and β, we get

$$\begin{aligned} \alpha &= \frac{(x_1 - x_2)(f(x_0) - f(x_2)) - (x_0 - x_2)(f(x_1) - f(x_2))}{(x_0 - x_1)(x_1 - x_2)(x_0 - x_2)} \\ \beta &= \frac{(x_0 - x_2)^2(f(x_1) - f(x_2)) - (x_1 - x_2)^2(f(x_0) - f(x_2))}{(x_0 - x_1)(x_1 - x_2)(x_0 - x_2)}. \end{aligned}$$

The roots of $Q(x)$ are

$$r_1, r_2 = x_2 + \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}.$$

Since $Q(x_0)Q(x_2) = f(x_0)f(x_2) < 0$, the quadratic must have at least one root in the interval $[x_0, x_2]$. Let x_3 be the root in the bracket. Now compare $[x_0, x_3]$ and $[x_3, x_2]$ to find a new bracket.

Müller's method

This is similar to quadratic method. But we do not necessarily start from a bracket. Among the two roots, we usually take the one with smaller in modulus, (i.e. the one closer to x_2). (Reason:Assume three consecutive points lie on a smooth convex curve, it is reasonable to expect smaller value is a good approximation.)

Brent method

If we observe a poor convergence in quadratic method, we can combined with bisection method. It is called Brent method.

1.6.1 Inverse quadratic method

Use $x = Q(y)$ instead of $y = Q(x)$. Finding the approx root is just obtained by setting $x = Q(0)$. (See fig. 1.20). advantage: We can use any high degree

polynomial, since it is not necessary to solve $P(x) = 0$ for (usually high degree) interpolating polynomial.

disadvantage: The approximation may not give a bracket.

Remark 1.6.1. (1) Müller's method finds complex root also.

(2) Newton's method also finds complex root if started with complex numbers.

(3) To find the root of a polynomial, it is suggested to use Companion matrix. Note that

$$\det \begin{bmatrix} -\lambda & 1 & 0 & \\ 0 & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & 1 \\ -a_n & \dots & & -a_1 - \lambda \end{bmatrix} = 0$$

if and only if $p(\lambda) = \lambda^n + a_1\lambda^{n-1} + \cdots + a_{n-1}\lambda + a_n = 0$.

(4) Order of convergence for Müller's method. $e_{n+1} \doteq e_n^{1.839}$.

(5) Müller's method may not converge for triple root. In this case use perturbation.

Example 1.6.2. Consider a cubic interpolating poly with data:

$$(0, 1.2), (1, 1), (2, 1.5), (3, .3)$$

Lagrange form $p_n(x) = \sum_{i=0}^{n} f(x_i) \prod_{j \neq i} \frac{(x-x_j)}{(x_i-x_j)}$ gives

$$p(x) = 1.2\frac{(x-1)(x-2)(x-3)}{-6} + 1\frac{x(x-2)(x-3)}{1\cdot(-1)(-2)} + 1.5\frac{x(x-1)(x-3)}{2\cdot 1(-1)} + .3\frac{x(x-1)(x-2)}{6}$$

But its inverse cubic $x = \tilde{p}(y)$ is

$$\tilde{p}(y) = 0 + 1\frac{(y-1.2)(y-1.5)(y-.3)}{(-.2)(-.5)(.7)} + 2\frac{(y-1.2)(y-1)(y-.3)}{.3\cdot .5(1.2)} + 3\frac{(y-1.2)(y-1)(y-1.5)}{(-.9)(-.7)(-1.2)}$$

Thus

$$p(x) = -0.4x^3 + 1.55x^2 - 1.35x + 1.2, \quad \tilde{p}(y) = 21.42857y^3 - 55.95238y^2 + 40.09524y - 4.57142857.$$

Graph below is shift by $-(0, 0.5)$.

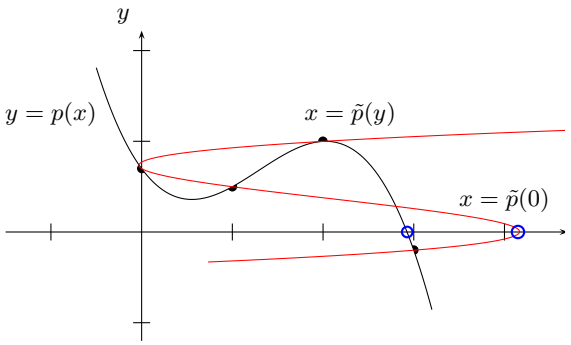

Figure 1.1: Cubic and inverse cubic interpolation

1.7 Computer arithematic

Floating-point numbers

Expressing real number: In decimal system

$$325.7 = 3 \times 10^2 + 2 \times 10^1 + 5 + 7 \times 10^{-1}.$$

In the most computer they use binary such as :

$$(11.0101)_2 = 1 \cdot 2 + 1 + 1 \cdot 2^{-2} + 1 \cdot 2^{-4}.$$

So if we have a decimal expression, we have to change to the binary form:

$$(0.3)_{10} = (.a_1 a_2 a_3 \cdots)_2.$$

Multiplying by 2

$$0.6 = (a_1.a_2 a_3 \cdots)_2, \quad a_i = 0, \text{ or } 1$$

Thus $a_1 = 0$. Continuing we obtain $a_2 = 1$, $a_3 = 0$, etc.

N-ary number: We use numbers $0, 1, \cdots, N-1$ and write as $(\cdots)_N$

Rounding

(round up), (round), (chop, truncate). If $\tilde{x}$ is the rounded number of x after n-th decimal, then we have

$$|x - \tilde{x}| \leq 1/2 \times 10^{-n}.$$

If $\hat{x}$ a the chopped number after n-th decimal, then we have

$$|x - \hat{x}| < 10^{-n}$$

Example 1.7.1. (rounded) Let $n = 4$.

$$0.345649 - -- > 0.3456(error = 0.000049 < \frac{1}{2}10^{-4}).$$

$$0.345550 - -- > 0.3456(error = 0.00005 = \frac{1}{2}10^{-4}).$$

Example 1.7.2. (chopped) Let $n = 4$.

$$0.345699 - -- > 0.3456(error = 0.000099 \approx 10^{-4}).$$

Normalized scientific notation-real numbers

Right hand side the normalized expression.

$$\begin{aligned} 875.33432 &= 0.87533432 \times 10^3 \\ 0.00067 &= 0.67 \times 10^{-3}. \end{aligned}$$

In general in normalized notation, a nonzero real number is written as

$$x = \pm r \times 10^n, (\frac{1}{10} \leq r < 1).$$

In binary system, it is written as

$$x = \pm q \times 2^m, 1 \leq q < 2.$$

Here q is **mantissa**, m is **exponent**.

A hypothetical computer K32

In this computer, a word consists of 32 bits.

```
sign of the real number    1 bit
biased exponent(integer e) 8 bits
mantissa (real number f)  23 bits.
```

For example, assume

$$x = (-1)^s q \times 2^m,$$

where $(1 \le q < 2)$, $q = (1.f_1 f_2 \cdots)_2$ (f_i is 0 or 1) and $m = e - 127$. Since the first number of q is always 1, we do not store it. So the 23 bits are used to store the floating part f.

In the computer e is stored, however, in the actual computation the exponent $m = e - 127$ is used. The reason is to handle the **negative exponent**.

$$0 < e < (11111111)_2 = 2^8 - 1 = 255$$

$-126 \le m \le 127$.

1	exponent 8 bit(e)	normalized mantissa 23 bits

A word in K32

Since $2^{127} \approx 10^{38}$, K32 can deal with numbers in the range $10^{-38} \sim 10^{38}$.

The precision with 23 bits is $2^{-23} \approx 1.2 \times 10^{-7}$.

In the real number case we can treat numbers up to 10^{38}, it is not accurate. Since we have only seven effective digits.

(However, inside the registry, computer uses about 80 bits) after computation it stores using 32 bits only.

Example 1.7.3. It can deal with a small number such as $0.\underbrace{0000000\cdots0}_{37}1$, but cannot store $x = 10.0000302$ accurately. It will store x as $10.00003\overline{0}$ because the mantissa is 23 digits in binary(approximately 1.2×10^{-7}). If we are using a decimal system in the computer memory, we would store

	Exponent 1	1.0000003

Storage of 10.0000302

While $0.\underbrace{0000000\cdots0}_{37}1$ is stored as

	Exponent -38	1.0

Storage of $0.0000\cdots000001$

Integer

In case of integer we use all of 31 (except 1 bit for the sign) we can represent a number between $-(2^{31}-1)$ and $(2^{31}-1) \approx 2.1 \times 10^9$.

Infinity, NaN, machine rounding

Overflow, underflow

If $m \geq 127$ it will say overflow or NaN(Not a number)

If a number is too small, say if $m \leq -127$ it says underflow and treat the number as 0

Floating point arithmetic

Assume a 5 machine having significant figures:

$$123.45, \quad 1.2345$$

In **floating point arithmetic**(Old computer)

$$0.00000005 \text{ is stored as } .5 \times 10^{-7}$$

In **fixed point arithmetic**, we would have $0.00000005 = 0$.

Let

$$x = .25689 \times 10^3, y = .75317 \times 10^6.$$

Then

$$xy = .193481841 \times 10^9 \rightarrow fl(xy) = .19348 \times 10^9.$$

Computers represent numbers with a fixed number of significant bits(precision) and a limited exponent.

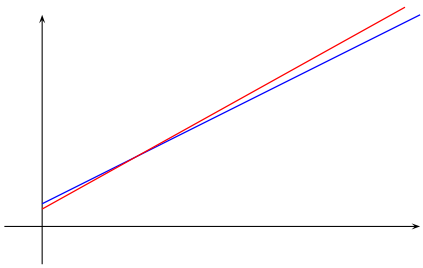

Figure 1.2: Intersection of almost parallel lines may induce large errors

Subtraction of nearly equal numbers

Suppose we compute $3 - 3.1415 = 0.1415$, we get exact value. However, suppose we compute $\pi - 3.1415$ in a five digit floating point arithmetic, the π will be stored as

$$0.31416 \times 10^1.$$

Thus

$$\pi - 3.1415 = 0.00001 = 0.1 \times 10^{-4},$$

while more accurate value is $3.1415926... - 3.1415 = 0.0000926....$ This is a large error. The reason is that by subtracting nearly equal numbers we have lost significant digits.

Similar phenomenon happens when one solves a system of equations when any two equations are nearly the same(the planes are almost parallel).

Machine ϵ

Among numbers of the form 2^{-k} the smallest machine number is a number such that $1.0 + \epsilon \neq 1.0$. One can compute it as follows:

input $s \leftarrow 1$
for $k = 1, 2, \cdots, 100$ do
  $s \leftarrow 0.5s$
  $t \leftarrow s + 1.0$
  if $(t \leq 1.0)$ then
    $s \leftarrow 2.0s$
    output $k - 1, s$
  stop
endif

1.8 Newton's Method for Systems

Let $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ be a differentiable map. We consider solving the roots of the equation:

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}.$$

Taylor expansion:

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}_0) + J(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + R(\mathbf{x}).$$

We replace $\mathbf{F}(\mathbf{x})$ by $\mathbf{F}(\mathbf{x}_0) + J(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$ and setting

$$\mathbf{F}(\mathbf{x}_0) + J(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \approx 0,$$

we obtain an approximate solution. Thus

$$J(\mathbf{x}_0)\mathbf{x} \approx J(\mathbf{x}_0)\mathbf{x}_0 - \mathbf{F}(\mathbf{x}_0).$$

$$\mathbf{x} \approx \mathbf{x}_0 - [J(\mathbf{x}_0)]^{-1}\mathbf{F}(\mathbf{x}_0).$$

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k - [J(\mathbf{x}_k)]^{-1}\mathbf{F}(\mathbf{x}_k).$$

Remark 1.8.1. To compute $[J(\mathbf{x}_0)]^{-1}\mathbf{F}(\mathbf{x}_0)$, it is not desirable to form the inverse $[J(\mathbf{x}_0)]^{-1}$. Instead we solve

$$J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\mathbf{F}(\mathbf{x}_k).$$

If $J(\mathbf{x}_k)$ is (close to) singular we expect a poor convergence. Stopping criterion

$$\rho_{k+1} = \frac{\|\mathbf{x}_{k+1} - \mathbf{x}_k\|}{\|\mathbf{x}_{k+1}\|},$$

where $\|\mathbf{x}\| = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$ is the norm of the vector.

Residual error.

Chord Method

Solving the system $J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\mathbf{F}(\mathbf{x}_k)$ is expensive. We thus replace it by

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k - [J(\mathbf{x}_0)]^{-1}\mathbf{F}(\mathbf{x}_k)$$

for fixed $\mathbf{x}_0$.

1.9 Broyden's Method

The Newton's method reads:

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k - [J(\mathbf{x}_k)]^{-1}\mathbf{F}(\mathbf{x}_k). \tag{1.8}$$

When n is large, the cost of computing $[J(\mathbf{x}_k)]^{-1}\mathbf{F}(\mathbf{x}_k)$ is high. So the idea is how to reduce the cost?

Quasi-Newton's Method

Replace (1.8) by

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k - B_k\mathbf{F}(\mathbf{x}_k),$$

where B_k is an approximation of $[J(\mathbf{x}_k)]^{-1}$. Typically $B_k \to [J(\mathbf{x}_k)]^{-1}$ as $k \to \infty$. Usually the convergence may be slower but the cost of calculating $B_k\mathbf{F}(\mathbf{x}_k)$ is cheaper so that the over all cost is cheap.

One of such method is the Broyden's Method, a secant like method.

Broyden's Method

Recall when $n = 1$ the secant method replaces $f'(x_k)$ by

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}. \tag{1.9}$$

But (1.9) is impossible when $n > 1$. However, observe that it is equivalent to

$$f'(x_k)(x_k - x_{k-1}) \approx (f(x_k) - f(x_{k-1})). \tag{1.10}$$

When $n > 1$ we apply an idea similar to this. We require

$$A(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}_{k-1}) \approx (\mathbf{F}(\mathbf{x}_k) - \mathbf{F}(\mathbf{x}_{k-1})). \tag{1.11}$$

Unfortunately this requirement does not define a unique matrix A_k. To define it we shall use A_{k-1} and require A_k satisfies (1.11) only along $\mathbf{x}_k - \mathbf{x}_{k-1}$ direction, while in the direction orthogonal to $\mathbf{x}_k - \mathbf{x}_{k-1}$, A_k has same effect as A_{k-1}, i.e.,

$$A_k\mathbf{z} = A_{k-1}\mathbf{z}, \tag{1.12}$$

for all $\mathbf{z}$ orthogonal to $\mathbf{x}_k - \mathbf{x}_{k-1}$. It can be verified that the matrix

$$A_k = A_{k-1} + \frac{(\mathbf{F}(\mathbf{x}_k) - \mathbf{F}(\mathbf{x}_{k-1}) - A_{k-1}(\mathbf{x}_k - \mathbf{x}_{k-1}))}{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|^2}(\mathbf{x}_k - \mathbf{x}_{k-1})^T \quad (1.13)$$

satisfies the requirement. This is summarized as Broyden's Method:

	Broyden's Method
1.	Set : $A_0 = J(\mathbf{x}_0)$.
2.	Compute $\mathbf{x}_1 = \mathbf{x}_0 - J^{-1}(\mathbf{x}_0)\mathbf{F}(\mathbf{x}_0)$
3.	Begin loop : $k = 1$
4.	Compute A_k using (1.11)
5.	Compute $\mathbf{x}_{k+1} = \mathbf{x}_k - A_k^{-1}\mathbf{F}(\mathbf{x}_k)$
6.	Repeat unless certain stopping criteria is met

This method is converges super linearly but takes much less time to converge. It is advised to reset A_k to $J(\mathbf{x}_k)$ periodically.

Look at (1.13). The formula is the form

$$A_k = A_{k-1} + \mathbf{u}\mathbf{v}^T,$$

where

$$\mathbf{u} = \frac{(\mathbf{F}(\mathbf{x}_k) - \mathbf{F}(\mathbf{x}_{k-1}) - A_{k-1}(\mathbf{x}_k - \mathbf{x}_{k-1}))}{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|^2}, \quad \mathbf{v} = (\mathbf{x}_k - \mathbf{x}_{k-1})^T.$$

The product of the form $\mathbf{u}\mathbf{v}^T$ is sometimes called **outer product** and

$$\text{rank } \mathbf{u}\mathbf{v}^T = 1.$$

The computation of A_k by (1.13) is called **rank-one update**. Implementing the Broyden's method by rank one update is called **Sherman-Morrison formula**.

Sherman-Morrison formula

If A is nonsingular and $\mathbf{v}^T A^{-1}\mathbf{u} \neq -1$ then $A + \mathbf{u}\mathbf{v}^T$ is nonsingular and

$$(A + \mathbf{u}\mathbf{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^T A^{-1}}{1 + \mathbf{v}^T A^{-1}\mathbf{u}}. \quad (1.14)$$