Constructive Mathematics

Developed by Andy Wathen,
minor edits by Raphael Hauser

May 5, 2022

Introduction

Much of Mathematics is abstract. However a surprisingly large number of algorithms - that is procedures which one can carry out in practice or in principle - are used throughout the subject. In Pure Mathematics, a procedure which can be guaranteed to yield a particular structure for a particular problem statement may provide an existence proof. In Statistics, algorithms are widely used on carefully collected data sets to yield statistics, that is numbers which have some representative meaning. In Applied Mathematics, the solutions of problems which model the physical world are very often found approximately using computational procedures.

Some constructions of use in algebra, analysis and more broadly in diverse applications will be introduced and analysed in this course. The ability to actually carry out these procedures is quite often provided only by computers; humans are not always so good at repetitive or routine procedures especially where a lifetime of 'hand' calculations can be done in milliseconds by an appropriate algorithm implemented on an electronic computer!

The Division Algorithm and Euclid's Algorithm

Consider the natural numbers

$$\mathbb{N} = \{0, 1, 2, \ldots\}.$$

A basic operation we learned in early school is to divide: if $a \in \mathbb{N}$ and $b \in \mathbb{N}$ with $a > b > 0$ then we can always find $q \in \mathbb{N} - \{0\}$ and $r \in \mathbb{N}$ with $0 \leq r < b$ such that

$$a = q \times b + r. \tag{1}$$

Thus $42 = 2 \times 15 + 12$ or $151 = 13 \times 11 + 8$. Note the clear stipulation that q is a positive integer and that the *remainder*, r is a non-negative integer which is less than b: we would introduce non-uniqueness (of q, r) were this not so.

This simple procedure is known as *The Division Algorithm* on the natural numbers.

A simple MATLAB implementation would be the function

```
function[q,r]=division(a,b)
  q=floor(a/b);r=a-q*b;
end
```

(which one would store in the file `division.m`).

We will not usually denote multiplication by $\times$, but rather, as is common, write qb for the product of q and b; in MATLAB it is `q*b`.

Armed with such a procedure, it is natural to explore what can be achieved by repeated application: since $b > r$ it is clear that the Division Algorithm can be applied to the pair (b, r) as it was to (a, b) provided $r > 0$. Since we might otherwise run out of letters, let us introduce subscripts and rewrite (1) as

$$a = q_1 b + r_1. \tag{2}$$

Further applying the Division Algorithm to (b, r_1), there must be $q_2 \in \mathbb{N} - \{0\}$ and $r_2 \in \mathbb{N}$ with $0 \leq r_2 < r_1$ such that

$$b = q_2 r_1 + r_2.$$

Repeating we obtain altogether

$$\begin{aligned}
a &= q_1 b + r_1 \\
b &= q_2 r_1 + r_2 \qquad (3) \\
r_1 &= q_3 r_2 + r_3 \\
r_2 &= q_4 r_3 + r_4 \\
&\vdots \\
r_{j-3} &= q_{j-1} r_{j-2} + r_{j-1} \qquad (4) \\
r_{j-2} &= q_j r_{j-1} + r_j \qquad (5) \\
&\vdots
\end{aligned}$$

with $a > b > r_1 > r_2 > r_3 > r_4 > \ldots > r_j > \ldots > 0$. One quickly realises that since a, b and the r_j for every j are members of $\mathbb{N}$ then this repeated procedure must terminate at some finite stage with $r_j = 0$ since the Division Algorithm can continue to be applied whenever the remainder is positive. Thus if it is r_j that is the first remainder to be zero, (5) must be

$$r_{j-2} = q_j r_{j-1} \qquad (6)$$

and our repeated procedure must stop: the Division Algorithm is not defined for application to $(r_{j-1}, 0)$.

Now (6) tells us that r_{j-1} is a factor of r_{j-2}. Thus the right hand side of the equals sign in (4) must also be exactly divisible (in the sense of the natural numbers: no fractions allowed here!) by r_{j-1}. Therefore r_{j-3} (as the left hand side) is also exactly divisible by r_{j-1}. Continuing to apply this argument back through the 'equations' we find, by induction, that $r_{j-1} > 0$ is a factor of each of the natural numbers $r_{j-2}, r_{j-3}, \ldots, r_2, r_1, b, a$.

Thus the repeated use of the Division Algorithm described here yields a (positive integer) common factor of a and b. That this procedure, widely known as *Euclid's Algorithm*, in fact yields the highest common factor of a and b is proved by the following inductive argument. Suppose $d > r_{j-1}$ is a larger common factor of a and b, then since (2) may be restated as

$$a - q_1 b = r_1,$$

it follows that d also must be a factor of r_1. Similarly using (3), d must divide r_2 and inductively d must divide r_{j-1}. That is, we cannot have a common factor of a, b which is greater than r_{j-1}.

The outcome is that Euclid's Algorithm must always compute the highest common factor, $hcf(a, b)$, of a and b as the last non-zero remainder.

Example.

$$\begin{aligned} 1099 &= 2 \times 525 + 49 \\ 525 &= 10 \times 49 + 35 \\ 49 &= 1 \times 35 + 14 \\ 35 &= 2 \times 14 + 7 \\ 14 &= 2 \times 7 + 0 \end{aligned}$$

hence 7 is the highest common factor of 1099 and 525. □

Note that if we divide each of 1099 and 525 by 7 to get 157 and 75, then the two numbers are necessarily *coprime*; that is the highest common factor of 157 and 75 is 1.

In case the highest common factor of larger natural numbers are required, a simple MATLAB implementation is something like

```
function[h]=euclid(a,b)
% a>b>0 are expected to be integers otherwise this will not work!
oldr = a; newr = b;
while newr>0, [q,r]=division(oldr,newr); oldr=newr; newr=r; end
h=oldr, end
```

(After storing this in the file `euclid.m` you might wish to experiment: I've just discovered that $hcf(987654321, 123456789) = 9$ which might be obvious, but I didn't know it! MATLAB starts to automatically convert very large integers to real (floating point) numbers when they are bigger than 2147483647, so you'll have to use some other computational environment if you want to go bigger than this.)

The basis of currently used cryptography and hence data security (whether you insert your cash card into a cash machine or order a book online) is the factorization of large integers. This is simply because factorization of a single natural number is a very computationally intensive task, requiring much more arithmetic than finding the highest common factor of two natural numbers by Euclid's Algorithm. Current crytographic methods (RSA, named after Rivest, Shamir and Adleman is the most widely used) would become insecure if a fast algorithm was found for integer factorization.

Simple Linear Diophantine Equations

We are used to solving linear equations over $\mathbb{R}$ (or indeed over $\mathbb{Q}$): thus $3x+2 = 4$ yields $x = 2/3$ and

$$\begin{aligned} 2x + 5y &= 3 \\ 7x + 3y &= 5 \end{aligned}$$

yields $x = 16/29, y = 11/29$. An interesting and different set of issues arise if we seek only solutions in the integers, $\mathbb{Z}$. To make this sensible we must have more variables than equations; $3x + 2 = 4$ will clearly have no solution in the integers, for example!

The simplest linear Diophantine equation (named after Diophantus of Alexandria who lived in the 3rd century) is of the form

$$ax + by = c, \tag{7}$$

where $a, b, c \in \mathbb{N}$ are given and it is desired to find all solutions $x, y \in \mathbb{Z}$. If $a = b$ then the question of existence of solutions simply reduces to whether a is a factor of c, or not, and there is some arbitrariness in x and y since only $x + y$ is essentially determined even if this condition is satisfied. Thus we consider (7) only when the two 'coefficients' are different and, without loss of further generality, when $a > b$.

First consider the situation where c is not divisible by $h = hcf(a, b)$, the highest common factor of a and b. The left hand side is then divisible by h for all integer values of x, y, but the right hand side is not: the conclusion must be that there can be no integer solutions in this case. Thus $1099x + 525y = 12$ can have no integer solutions since 12 is not divisible by $7 = hcf(1099, 525)$.

Next, if c is divisible by $hcf(a, b)$ then the whole equation can be divided by $hcf(a, b)$; thus $1099x+525y = 28$ has the same integer solutions as $157x+75y = 4$. We may thus assume without loss of generality that a and b are co-prime (that is, $hfc(a, b) = 1$). To find the set of all such solutions in this case, we can first find a particular solution and then add the general solution of the associated homogeneous solution:

Lemma. If (x_p, y_p) is a particular solution of (7), then $(x, y) \in \mathbb{Z}^2$ is a solution of (7) if and only if it is of the form $(x, y) = (x_p + x_h, y_p + y_h)$, where $(x_h, y_h) \in \mathbb{Z}^2$ satisfies the homogeneous equation

$$ax + by = 0. \tag{8}$$

Proof
Let (x, y) be a solution of (7), and define $x_h = x_p - x$, $y_h = y_p - y$. Then $(x_h, y_h) \in \mathbb{Z}^2$ and $ax_h + by_h = (ax_h + by_h) - (ax + by) = 0$. The reverse implication follows similarly. □

To find a particular solution of (7), it is sufficient to find any integer solution $\hat{x}, \hat{y}$ of the equation

$$a\hat{x} + b\hat{y} = 1 \tag{9}$$

and recover a particular solution of (7) by setting $(x_p, y_p) = c(\hat{x}, \hat{y})$. For example, a particular solution of $157x + 75y = 4$ can be found by identifying a particular solution $(\hat{x}, \hat{y})$ of $157\hat{x} + 75\hat{y} = 1$, and by multiplying the result by 4, $(x_p, y_p) = (4\hat{x}, 4\hat{y})$. In what follows, we will see that Diophantine equations of the form (9) always have a solution when a and b are co-prime. The proof is

constructive and yields an algorithm for the computation of such a solution.

Examples.

i) $12x + 9y = 4$ has no integer solutions, because $3 = hcf(12, 9)$ does not divide into 4.

ii) $12x + 9y = 15$ has the same set of integer solutions as

$$4x + 3y = 5, \tag{10}$$

and a particular solution of the latter can be found by multiplying a particular solution of $4\hat{x} + 3\hat{y} = 1$ by 5. Taking for example $(\hat{x}, \hat{y}) = (1, -1)$, we find $(x, y) = (5, -5)$ as a particular solution of (10). Note however, that not all solutions of the latter equation can be obtained in this fashion. For example, $(2, -1)$ solves (10) but is not of the form $5(\hat{x}, \hat{y})$ for any $(\hat{x}, \hat{y}) \in \mathbb{Z}^2$.

□

Consider now Euclid's Algorithm applied to a, b as in (9), and assuming that $hcf(a, b) = 1$, as above. Since 1 is the highest common factor, the final equation in the algorithm (that is the one with the last non-zero remainder) must be

$$r_{j-2} = q_j r_{j-1} + 1$$

for some j (in case the remainder 1 arises at the first stage (2) or second stage (3) we could just call $a = r_{-1}$ and $b = r_0$). Thus

$$1 = r_{j-2} - q_j r_{j-1}. \tag{11}$$

The previous equation (4) similarly defines

$$r_{j-1} = r_{j-3} - q_{j-1} r_{j-2}$$

and hence r_{j-1} can be subtituted in so that (11) become an equality between 1 and the sum of multiples of r_{j-2} and r_{j-3}. This process can always be continued (with integers only) until we have an equality between 1 and the sum of multiples of b and a. An example demonstrates this best.

Example. Euclid's Algorithm applied to $(157, 75)$ gives

$$\begin{aligned} 157 &= 2 \times 75 + 7 \\ 75 &= 10 \times 7 + 5 \\ 7 &= 1 \times 5 + 2 \\ 5 &= 2 \times 2 + 1 \\ (2 &= 2 \times 1 + 0). \end{aligned}$$

Thus,

$$\begin{aligned} 1 &= 5 - 2 \times 2 \\ &= 5 - 2 \times (7 - 1 \times 5) = -2 \times 7 + 3 \times 5 \\ &= -2 \times 7 + 3 \times (75 - 10 \times 7) = 3 \times 75 - 32 \times 7 \\ &= 3 \times 75 - 32 \times (157 - 2 \times 75) = -32 \times 157 + 67 \times 75 \end{aligned}$$

so that $\hat{x} = -32, \hat{y} = 67$ is a solution of $157\hat{x} + 75\hat{y} = 1$. A solution of $157x + 75y = 4$, for example is then $x = -128, y = 268$. □

Notice the similarity between this example and the example in the Euclid Algorithm section above: the reduction to coprimality was not actually necessary, but has been introduced here as a simplifying device. This is a very common feature in general in the derivation and use of constructive methods: define an algorithm for the simplest case from which a whole range of problems can be solved.

The question arises as to whether we have found the only solution by the above procedure: in fact if solutions exist, then there are always infinitely many of which the above procedure has found only one. For consider if $\hat{x}, \hat{y}$ are the solution of (9) found by the procedure above (or in fact by any method) and $\xi, \eta \in \mathbb{Z}$ satisfy

$$a\xi + b\eta = 0 \tag{12}$$

then by addition (i.e. by linearity)

$$a(\hat{x} + \xi) + b(\hat{y} + \eta) = 1.$$

One solution of (12) is clearly $\xi = b, \eta = -a$ and since a and b are coprime it must be the case that $\xi = nb, \eta = -na, n \in \mathbb{Z}$ are the set of all solutions of the *homogeneous* equation (12). So the set of all solutions to (9) are $x = \hat{x} + nb, y = \hat{y} - na, n \in \mathbb{Z}$. If there were any other solution $\tilde{x}, \tilde{y}$ not of this form then $\tilde{x} - \hat{x}, \tilde{y} - \hat{y}$ must be a solution of the homogeneous equation (12) from which it follows that $\tilde{x} = \hat{x} + nb, \tilde{y} = \hat{y} - na$ for some $n \in \mathbb{Z}$.

Examples.

i) 157 and 75 are coprime, and the equation $157\hat{x} + 75\hat{y} = 1$ is satisfied for $\hat{x} = -32, \hat{y} = 67$. A particular solution of

$$157x + 75y = 4 \tag{13}$$

is thus $(-128, 268) = 4(-32, 67)$, and the general solution of (13) is given by $x = -128 + 75n$, $y = 268 - 157n$, $(n \in \mathbb{Z})$.

ii) We saw previously that Equation (10) has $(5, -5)$ as particular solution. The set of all solutions is thus given as $\{(5 + 3n, -5 - 4n) : n \in \mathbb{Z}\}$. In particular, the solution $(2, -1)$ is found in this set by choosing $n = -1$.

□

The above constructions (do more than) provide a proof of the following result.

Corollary. (Bézout's Lemma)
If a, $b \in \mathbb{N}$ are both non-zero and $d = hcf(a, b)$ then there exist $x, y \in \mathbb{Z}$ satisfying $ax + by = d$. □

Note, in particular, that if a and b are coprime then there are integer solutions of $ax + by = 1$.

As a historical note, it was in the margin of his copy of a Latin translation of Diophantus' book Arithmetica that Fermat scribbled the now famous observation that he had a beautiful proof of what became known as Fermat's Last Theorem, but that there was not enough room in the margin to write it down!

Polynomials

There is another context in which the Division Algorithm might have been encountered at school: if one considers the set of real polynomials then division is possible. We will not cover this rigorously here – it will appear in a later algebra course – however a couple of examples indicate the idea.

Examples.

$$x^4 + 2x^3 - 2x^2 + 3x = \left(\frac{1}{2}x^2 + \frac{1}{2}\right)\left(2x^2 + 4x - 6\right) + (x + 3),$$
$$2x^3 + 3x^2 + (\pi - 2)x - \frac{\pi}{2} = \left(x - \frac{1}{2}\right)\left(2x^2 + 4x + \pi\right) + 0,$$

that is, in both cases we have

$$a(x) = q(x)b(x) + r(x),$$

where $b(x)$ has degree lower than or equal than the degree of $a(x)$, and $r(x)$ is a polynomial of degree strictly less than the degree of $b(x)$. □

Note that the (ordered) 'size' of a number is what is important for $\mathbb{N}$, but it is the degree which determines 'size' in this context: $a(x) > b(x)$ would here have to mean that the degree of the polynomial a is greater than the degree of the polynomial b. The coefficients of all of the polynomials are in general real and not integers.

If this division can always be done (it can), then Euclid's Algorithm can also be applied since it is simply repeated application of the Division Algorithm. This repeated application must eventually lead to a remainder polynomial of degree zero i.e. to a constant. If this constant is zero then a common factor

– indeed the highest common factor – of two polynomials has been found as the previous remainder. If the constant is not zero then there are no common factors other than (arbitrary) constants. Any constant is a common factor of any pair of polynomials just as 1 is a common factor of any two natural numbers.

Example.

$$x^6 - 4x^5 + x^4 - 4x^3 - 5x^2 + 24x + 3 = (x-4)\left(x^5 + x^3 - 6x\right) + \left(x^2 + 3\right)$$
$$x^5 + x^3 - 6x = \left(x^3 - 2x\right)\left(x^2 + 3\right) + 0$$

so that (any scalar multiple of) $x^2 + 3$ is the highest common factor of the polynomials $x^6 - 4x^5 + x^4 - 4x^3 - 5x^2 + 24x + 3$ and $x^5 + x^3 - 6x$. □

Example.

$$x^4 + 1 = (x)\left(x^3 + 1\right) - (x - 1)$$
$$x^3 + 1 = \left(-x^2 - x - 1\right)(-x + 1) + 2$$

so there are no common factors of $x^4 + 1$ and $x^3 + 1$ (other than constants). □

Thus there is in principle a constructive method to find common factors and hence, common roots of two polynomials. Just as with the natural numbers, however, the problem of factoring a single polynomial, that is finding its roots, is rather more difficult. We consider this problem next: it has a rather more analytic and less algebraic flavour and the methods generally apply to smooth enough functions and not just to polynomials.

Root Finding: The Bisection Algorithm

If one is given a function which is continuous on a real interval and two points a, b within this interval for which the values $f(a)$ and $f(b)$ are of opposite sign, then the Intermediate Value Theorem immediately guarantees the existence of at least one point α between a and b where $f(\alpha) = 0$.

This observation can be turned into a procedure for finding such an α to any desired accuracy, $\epsilon > 0$. The simplest way is to calculate the point mid way between a and b, to check the sign of f at that point, to identify an interval of half the length which must contain a root and to repeat:

Algorithm.

Starting from $a = a_0$, $b = b_0$
for $k = 0, 1, 2, \ldots$
  calculate $c_k = \frac{1}{2}(a_k + b_k)$
  if $f(c_k)$ is of the same sign as $f(a_k)$ then set $a_{k+1} = c_k$ and $b_{k+1} = b_k$
  otherwise set $a_{k+1} = a_k$ and $b_{k+1} = c_k$
repeat

If this procedure is stopped when $|a_k - b_k| < 2\epsilon$ then $|c_k - \alpha| < \epsilon$ and one has the desired accuracy.

Analysing this procedure, one has

$$|c_k - \alpha| \leq \frac{1}{2}|b_k - a_k| = \frac{1}{2}\left(\frac{1}{2}|b_{k-1} - a_{k-1}|\right) = \ldots = \frac{1}{2^{k+1}}|b_0 - a_0| = \frac{1}{2^{k+1}}|b-a|$$

and a required accuracy ϵ is guaranteed when $2^{k+1}\epsilon \geq |b-a|$.

A simpler implementation just redefines a, b at each *iteration*:

Algorithm.

while $|a-b| > \epsilon$, do the following
set $m = \frac{1}{2}(a+b)$
if $f(m)f(a) < 0$, redefine $b = m$
else redefine $a = m$
repeat

This algorithm is implemented in the following MATLAB function

```
function[m]=bisection(a,b,tol)
  while b-a>tol, m=(a+b)/2;
    if f(m)*f(a) <0, b=m, else a=m; end
  end
end
```

which should be stored in the file `bisection.m`. The extra MATLAB function

```
function[value]=f(x)
  value = x^2 -1;
end
```

(stored in the file `f.m` and set to $x^2 - 1$ here) must simply define (that is, evaluate) the function f.

This procedure is called the *Bisection Method* since one is bisecting (cutting in half) the length of the interval containing a root at each stage. It is perhaps the simplest root finding method. It illustrates the important idea that if one has a procedure which generates a convergent sequence, then one can stop at some finite stage to achieve an approximate solution to any desired accuracy: with such a method, more computational work is required for better accuracy.

Fixed Point Methods

Given $f : \mathbb{R} \to \mathbb{R}$ the problem of finding a value α such that $f(\alpha) = 0$ can be recast in many different ways as the problem of finding a value α such that

$$\alpha = g(\alpha) \tag{14}$$

for some g.

Examples.

(i) $x^5 - 3x + \frac{1}{2} = 0 \leftrightarrow x = \frac{1}{3}\left(x^5 + \frac{1}{2}\right)$

(ii) $x^5 - 3x + \frac{1}{2} = 0 \leftrightarrow x = -1/(2x^4 - 6)$

(iii) $\sin x - x\cos x = 0 \leftrightarrow x = \tan x$

(iv) $f(x) = 0 \leftrightarrow x = x - \mu f(x)$ for any $\mu \in \mathbb{R}, \mu \neq 0$. □

A value α satisfying (14) is called a *fixed point* of the function g.

The associated *fixed point iteration* is to select some particular $x_0 \in \mathbb{R}$ and then to iteratively define (i.e. compute!)

$$x_k = g(x_{k-1}) \quad \text{for } k = 1, 2, \ldots. \tag{15}$$

Clearly if this iteration defines a convergent sequence, so that $x_k \to \alpha$ as $k \to \infty$, then also $x_{k-1} \to \alpha$ as $k \to \infty$, so that the limit, α, must be a fixed point of g.

Our first consideration is sufficient conditions for existence of fixed points. We will say that $g : [a, b] \to [a, b]$ is a continuous function on $[a, b]$ if g is continuous on (a, b) as well as $g(a) = \lim_{x \searrow a} g(x)$ and $g(b) = \lim_{x \nearrow b} g(x)$. This is all that is needed for the existence of a fixed point to be guaranteed:

Lemma. (Existence of a fixed point)

If $g : [a, b] \to [a, b]$ is continuous on $[a, b]$ then g has a fixed point $\alpha \in [a, b]$.

Proof

If $g(a) = a$ or $g(b) = b$ then we are done. Assuming $a \neq g(a)$ and $b \neq g(b)$ then if we set

$$h(x) = x - g(x)$$

we have

$$a \leq g(x) \leq b \Rightarrow \begin{cases} h(a) = a - g(a) < 0 \\ h(b) = b - g(b) > 0 \end{cases}$$

Now h is necessarily a continuous function on $[a, b]$ since g is, so applying the Intermediate Value Theorem, there exists $\alpha \in (a, b)$ with $h(\alpha) = 0$, i.e., $\alpha = g(\alpha)$. □

You may like to convince yourself that the above Lemma gives sufficient but not necessary conditions for the existence of fixed points and also that uniqueness cannot hold without further conditions. Such conditions are given now:

Lemma. (Uniqueness of a fixed point)

If $g : [a, b] \to [a, b]$ is continuous on $[a, b]$, differentiable on (a, b) and if there exists $\gamma \in \mathbb{R}$ with $0 \leq \gamma < 1$ such that

$$|g'(s)| \leq \gamma \quad \text{for all } s \in (a, b)$$

then there is a unique fixed point of g in $[a, b]$.

Proof

Assume g has 2 fixed points $c \neq d, c, d \in [a, b]$ then we can employ the Mean Value Theorem (or Taylor's Theorem if you prefer) which guarantees the existence of $\xi \in \mathbb{R}$ strictly between c and d with

$$g(c) - g(d) = (c - d)g'(\xi).$$

But c and d are fixed points of g, so $g(c) = c$ and $g(d) = d$, thus

$$c - d = (c - d)g'(\xi)$$

and on taking absolute values we have

$$|c - d| = |c - d||g'(\xi)| \leq \gamma|c - d| < |c - d|$$

which provides a contradiction and hence, completes the proof of uniqueness. □

This Lemma is rather more important than one might expect. Consider if a function g did have 2 fixed points, then the fixed point iteration (15) might compute a sequence which simply oscillated between them and so not converge or it might compute an even more complicated non-convergent (hence divergent) sequence. This a major consideration for constructive methods: problems which have non-unique solutions are generally fraught with difficulties, whereas if any solution is unique then at least there is a candidate which might be computed or approximated by a constructive method.

In the context of fixed points and the fixed point iteration, precisely the same conditions which guarantee existence and uniqueness of a fixed point also guarantee convergence of the fixed point iteration:

Lemma. (Convergence of fixed point iteration)

If $g : [a, b] \to [a, b]$ is continuous on $[a, b]$, differentiable on (a, b) and if there exists $\gamma \in \mathbb{R}$ with $0 \leq \gamma < 1$ such that

$$|g'(s)| \leq \gamma \quad \text{for all } s \in (a, b)$$

then the fixed point iteration

$$x_k = g(x_{x-1}), \quad k = 1, 2, \ldots$$

gives a sequence $\{x_k, k = 0, 1, 2, \ldots\}$ which converges to the unique fixed point in $[a, b]$ for any $x_0 \in [a, b]$.

Proof

$x_0 \in [a, b] \Rightarrow x_1 = g(x_0) \in [a, b]$ and inductively $x_k = g(x_{k-1}) \in [a, b]$ for $k = 0, 1, 2, \ldots$.

If α is the fixed point in $[a, b]$ then

$$x_k - \alpha = g(x_{k-1}) - g(\alpha). \tag{16}$$

By Taylor's Theorem (or the Mean Value Theorem) there exists ξ_k strictly between x_{k-1} and α such that

$$g(x_{k-1}) = g(\alpha) + (x_{k-1} - \alpha)g'(\xi_k)$$

and using this in (16) gives

$$x_k - \alpha = (x_{k-1} - \alpha)g'(\xi_k).$$

On taking absolute values we have

$$|x_k - \alpha| = |x_{k-1} - \alpha|\,|g'(\xi_k)| \leq \gamma|x_{k-1} - \alpha| \leq \gamma^2|x_{k-2} - \alpha| \leq \ldots \leq \gamma^k|x_0 - \alpha|. \tag{17}$$

Thus $x_k \to \alpha$ as $k \to \infty$ because $\gamma < 1 \Rightarrow \gamma^k \to 0$ as $k \to \infty$. □

These results establish checkable conditions for fixed points and convergence of fixed point iterations. Note that the proof here is non-constructive (indeed the existence Lemma used the Intermediate Value Theorem which is non-constructive), but the result allows for the reliable use of a constructive method for the calculation of fixed points.

Example.

Determine if $f(x) = x^5 - 3x + \frac{1}{2}$ has any roots in $[0, \frac{1}{2}]$ and if so, find them.

As in the example above we can write this as the fixed point problem

$$x = \frac{1}{3}(x^5 + \frac{1}{2}) = g(x).$$

Now clearly for $0 \leq x \leq \frac{1}{2}$ we have $\frac{1}{6} \leq g(x) \leq \frac{1}{3}(\frac{1}{32} + \frac{1}{2})$ so $g : [0, \frac{1}{2}] \to [0, \frac{1}{2}]$ and roots must exists here because g is clearly continuous. Further

$$g'(x) = \frac{5}{3}x^4 \quad \Rightarrow \quad |g'(x)| \leq \frac{5}{48} < 1$$

on the open interval. Hence there exists a unique fixed point of g and thus a unique real root of f in $[0, \frac{1}{2}]$. Starting with $x_0 = 0.25$ we calculate

$$x_1 = 0.16699219, \quad x_2 = 0.16670995, \quad x_3 = 0.16670959 = x_4 = x_5 = \ldots$$

to the number of decimal places shown. □

Example.

On $[0, 3/2]$ verify if there are any roots of $\cos x - x = 0$ and calculate any such root to 3 decimal places.

Write as $g(x) = \cos x$ and note that $g(x) \in [0, 1] \subset [0, 3/2]$ (as $3/2 < \pi/2$) and g is certainly a continuous function on the given interval, so at least one fixed point in $[0, 3/2]$ exists. Also $|g'(x)| = |-\sin(x)| \leq \sin(3/2) < 0.9975$

for $x \in (0, 3/2)$ so there is a unique fixed point in this interval which we may compute by fixed point iteration: say we start with $x_0 = 0 \in [0, 3/2]$

$$\begin{aligned}
x_1 &= \cos(x_0) = 1 \\
x_2 &= \cos(x_1) = 0.5403 \\
x_3 &= \cos(x_2) = 0.8576 \\
x_4 &= \cos(x_3) = 0.6543 \\
x_5 &= \cos(x_4) = 0.7935 \\
&\vdots \\
x_{21} &= \cos(x_{20}) = 0.7392 \\
x_{22} &= \cos(x_{21}) = 0.7390 \\
x_{23} &= \cos(x_{22}) = 0.7391 = x_{24} = x_{25} = \ldots .
\end{aligned}$$

Given the value $\gamma = 0.9975$ here for which powers reduce pretty slowly, ($0.9975^{23} \approx 0.9441$) it is not so surprising that it takes many more iterations here to get convergence than in the example above. □

It is pretty clear from this last example that we can use 'mathematics by hand' to verify that there is a unique root in the required interval, and that may be all that we want to know. However if we wish to actually calculate this root, then using a computer is worthwhile. A simple piece of MATLAB code (stored in the file `fixedpt.m`) here will enable this:

```
function[x] = fixedpt(xin,tol)
xold=xin;x=g(xin),
while abs(x-xold)>tol,xold=x;x=g(xold), end
end
```

where the function g needs to be defined (and stored in the file `g.m`)

```
function[value]= g(x)
  value=cos(x);
end
```

If you don't want to see anything but the approximate root then one can simply amend the last three lines as

```
xold=xin;x=g(xin);
while abs(x-xold)>tol,xold=x;x=g(xold); end
x, end
```

Though this may be the more common situation, it is not actually necessary for the function g to be differentiable in order to guarantee uniqueness of fixed points and convergence of fixed point iterations:

Theorem. (The Contraction Mapping Theorem)
If $g : [a, b] \to [a, b]$ is continuous and satisfies

$$|g(x) - g(y)| \leq \gamma |x - y|$$

for some real value $0 \leq \gamma < 1$ and for all $x, y \in [a, b]$ then there exists a unique fixed point of g in $[a, b]$ and for any $x_0 \in [a, b]$ the fixed point iteration (15) will converge to it.

Proof. Similar to the above. □

Whether g is differentiable or not, the value of γ is important; if γ is only just smaller than 1 then γ^k reduces only slowly with increasing k, whereas if γ is rather smaller than 1 then the reduction is rapid and (17) guarantees rapid convergence to any desired accuracy of the corresponding fixed point iteration. In practice this is often a significant consideration: existence of $\gamma < 1$ is sufficient if existence of a unique fixed point is all one is interested in, but if one actually wants to compute accurate approximations of fixed points then speed of convergence of the fixed point iteration is important.

Newton's Method

It follows from (17) that

$$0 < \frac{|x_k - \alpha|}{|x_{k-1} - \alpha|} \leq \gamma < 1. \tag{18}$$

A convergent sequence $\{x_k\}$ satisfying (18) is said to *converge linearly* and the corresponding fixed point iteration to have *linear convergence*. The question arises as to whether more rapid convergence can be achieved with a fixed point iteration. For example, do there exist methods for which

$$0 < \frac{|x_k - \alpha|}{|x_{k-1} - \alpha|^2} \leq K \tag{19}$$

for some constant, K? Any such fixed point method is called *quadratically convergent*.

From Taylor's Theorem, provided $g \in C^2$ on some appropriate interval containing α, we have

$$g(x_k) = g(\alpha) + (x_k - \alpha)g'(\alpha) + \frac{1}{2}(x_k - \alpha)^2 g''(\xi_k), \quad \text{some } \xi_k$$

i.e.

$$x_{k+1} - \alpha = (x_k - \alpha)g'(\alpha) + \frac{1}{2}(x_k - \alpha)^2 g''(\xi_k)$$

and so quadratic convergence will be achieved if $g'(\alpha) = 0$. So to find a root α of f, look for $\phi(x)$ so that

$$x - \phi(x)f(x) = g(x)$$

satisfies $g'(\alpha) = 0$:

$$g'(\alpha) = 1 - \phi(\alpha)f'(\alpha) - \phi'(\alpha)f(\alpha) \Rightarrow \phi(x) = 1/f'(x)$$

(or indeed any sufficiently smooth function ϕ which happens to satisfy $\phi(\alpha) = 1/f'(\alpha)$) since $f(\alpha) = 0$. The resulting method is *Newton's method*:

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

It is widely used because of the quadratic convergence property, but it is only locally convergent: it is necessary that x_0 is sufficiently close to α to guarantee convergence to α. Also note that x_{k+1} is only defined (bounded) if $f'(x_k) \neq 0$. Rigorous conditions guaranteeing the convergence of (the iterates computed by) Newton's method are somewhat technical and will not be covered in this course: statements about the convergence of Newton's method should be assumed to implicitly include the condition: 'if $\{x_k\}$ converges'.

Example.

$f(x) = x^5 - 3x + \frac{1}{2}$ clearly has $f'(x) = 5x^4 - 3$ so Newton's method gives the following

(i) if $x_0 = 0$, then

$$\begin{aligned} x_1 &= 0.166666666666667 \\ x_2 &= 0.166709588805906 \\ x_3 &= 0.166709588834381 = x_4 = \ldots \end{aligned}$$

(ii) if $x_0 = 0.8$ then

$$\begin{aligned} x_1 &= -0.851596638655463 \\ x_2 &= 6.188323841208973 \\ x_3 &= 4.952617138936713 \\ x_4 &= 3.965882550556341 \\ x_5 &= 3.180014757669606 \\ x_6 &= 2.558042610355714 \\ x_7 &= 2.073148949750824 \\ x_8 &= 1.708602767234352 \\ x_9 &= 1.457779524622984 \\ x_{10} &= 1.319367836130980 \\ x_{11} &= 1.274944679385050 \\ x_{12} &= 1.270653002026494 \\ x_{13} &= 1.270615088695053 \\ x_{14} &= 1.270615085755746 = x_{15} = \ldots \end{aligned}$$

Notice how in (ii) it takes several iterations before the iterates 'settle down' and ultimately converge rapidly (in fact quadratically!) to another root of this quintic polynomial. □

Example.

$f(x) = \cos(x) - x$ has $f'(x) = -\sin(x) - 1$ so Newton's method gives

(i) if $x_0 = 0$, then

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 0.750363867840244 \\ x_3 &= 0.739112890911362 \\ x_4 &= 0.739085133385284 \\ x_5 &= 0.739085133215161 = x_6 = \ldots \end{aligned}$$

(ii) if $x_0 = -3$, then

$$\begin{aligned} x_1 &= -0.6597 \\ x_2 &= 3.0858 \\ x_3 &= -0.7829 \\ x_4 &= 4.2797 \\ x_5 &= -46.7126 \\ x_6 &= 29.5906 \\ x_7 &= -896.8049 \\ &\vdots \end{aligned}$$

and the sequence of iterates converges only after 131 iterations, after a phase of erratic behaviour. □

The simple MATLAB functions used to obtain these numerical results are

```
function[x] = newton(xin,tol)
  x=xin;funvalue = f(x),
while abs(funvalue)>tol,x=x-funvalue/fprime(x), funvalue = f(x), end
end

function[value]=f(x)
  value = x^5 -3*x + 0.5;
% value = cos(x)-x;
end

function[value]= fprime(x)
  value=5*x^4 - 3;
% value = -sin(x)-1;
end
```

For functions with enough derivatives, one can define iterative methods for which

$$0 < \frac{|x_k - \alpha|}{|x_{k-1} - \alpha|^p} \leq K \tag{20}$$

for some constant, K. Any such method would have *order of convergence* p.

In practice there is generally little to be gained from consideration of methods with higher order convergence than $p = 2$ and Newton's method (and its generalisations, some of which we will come to later) is by far the most important practical method for solving nonlinear equations when derivatives are available. If the derivative is not readily available, then the leading general, practical method is probably Brent's method which is essentially a clever combination of the Bisection method and the Secant method (see problem 5 on question sheet 3).

Horner's Method

When seeking roots of polynomials expressed in the form

$$p_n(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0, \quad a_n \neq 0, \tag{21}$$

Newton's method can be carried out in a particularly efficient and elegant way using *Horner's method* or *Horner's rule* which is sometimes also called the *nested multiplication* scheme. This is approached most easily by considering an efficient algorithm for evaluating a polynomial.

Theorem. Setting $b_n = a_n$ and for $r = n-1, n-2, \ldots 2, 1, 0$ setting

$$b_r = b_{r+1}\sigma + a_r,$$

then $b_0 = p_n(\sigma)$. Moreover, if

$$q_{n-1}(z) = b_n z^{n-1} + b_{n-1} z^{n-2} + \ldots + b_2 z + b_1,$$

then

$$p_n(z) = (z - \sigma) q_{n-1}(z) + b_0.$$

Proof

$$\begin{aligned}(z-\sigma)q_{n-1}(z) + b_0 &= (z-\sigma)(b_n z^{n-1} + b_{n-1} z^{n-2} + \ldots + b_2 z + b_1) + b_0 \\ &= b_n z^n + (b_{n-1} - b_n\sigma) z^{n-1} + \cdots + (b_r - b_{r+1}\sigma) z^r + \\ &\qquad \cdots + (b_1 - b_2\sigma) z + (b_0 - b_1\sigma).\end{aligned}$$

But $b_n = a_n$, and for each $r \in \{n-1, n-2, \ldots, 2, 1, 0\}, b_r = b_{r+1}\sigma + a_r$ which implies $a_r = b_r - b_{r+1}\sigma$, so that

$$(z-\sigma)q_{n-1}(z) + b_0 = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0 = p_n(z).$$

Setting $z = \sigma$ gives $b_0 = p_n(\sigma)$.□

Note that this nested multiplication requires only n multiplications and n additions.

Differentiating

$$p_n(z) = (z - \sigma)q_{n-1}(z) + b_0$$

gives

$$p_n'(z) = (z - \sigma)q_{n-1}'(z) + q_{n-1}(z)$$

so that $p_n'(\sigma) = q_{n-1}(\sigma)$. We may thus write Newton's method for a root of a polynomial as: choose x_0 and for $k = 0, 1, 2, \ldots$ set

$$x_{k+1} = x_k - p_n(x_k)/p_n'(x_k),$$

so with $x_k = \sigma$:

$$x_{k+1} = x_k - p_n(x_k)/q_{n-1}(x_k).$$

So perform nested multiplication on $p_n(x_k)$ to give

$$b_n = a_n \text{ and for } r = n-1, n-2, \ldots, 2, 1, 0 \text{ compute } b_r = b_{r+1}x_k + a_r$$

so that $b_0 = p_n(x_k)$, and then perform nested multiplication on $q_{n-1}(x_k)$ to give

$$c_{n-1} = b_n \text{ and for } r = n-2, n-3, \ldots, 2, 1, 0 \text{ compute } c_r = c_{r+1}x_k + b_{r+1}$$

so that $c_0 = q_{n-1}(x_k)$, and thus the next Newton iterate is

$$x_{k+1} = x_k - b_0/c_0.$$

For a computational algorithm, perform the two evaluations consecutively:

```
function[x] = horner(xin,a,n,maxit,tol)
  % n is degree of the polynomial, a is a vector of length n+1 containing
  % the coefficients with  constant term first, the coefficient of the
  % linear term second, then quadratic etc
  % xin is the starting value for the Newton iteration
x=xin;
for k=1:maxit,
    b=a(n+1);     % compute b_n
    c=b;          % compute c_{n-1}
    b=b*x + a(n);      %compute b_{n-1}
    for r=n-2:-1:0,       %steps of -1 down to zero
        c=c*x + b;    %compute c_r
        b=b*x + a(r+1);  %compute b_r
    end,
    if abs(b)<tol, return, end % convergence test
    x=x-b/c;  % Newton update
end
end
```

Notice that as MATLAB indexes vectors (and in fact all arrays) from 1 rather than from zero, in the above MATLAB function the given polynomial is

$$p_n(x) = a(1) + a(2)x + a(3)x^2 + \ldots + a(n)x^{n-1} + a(n+1)x^n.$$

Example.
For the cubic polynomial $6 - 5x - 2x^2 + x^3$,
`alpha=horner(0,[6,-5,-2,1],3,20,1.e-11)` gives `alpha = 1`,
`alpha=horner(20,[6,-5,-2,1],3,20,1.e-11)` gives `alpha = 3.000000000000099`,
and
`alpha=horner(0.2,[6,-5,-2,1],3,20,1.e-11)` gives `alpha = 0.999999999999997`.
□

Example.

```
maxit=20; a=[1,-2,-3,-4,-5,-6,7]; tolerance = 1.e-9;
alpha=horner(2.9,a,6,maxit,tolerance)
```

computes the approximate root

```
alpha = 1.634200239846092
```

of the equation $1 - 2x - 3x^2 - 4x^3 - 5x^4 - 6x^5 + 7x^6 = 0$. □

Example. Find the minimum value of the polynomial

$$1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5 + 7x^6 = p(x)$$

on the real line.

Simple calculus gives

$$p'(x) = 2 + 6x + 12x^2 + 20x^3 + 30x^4 + 42x^5$$

and using the `horner` function in MATLAB

```
alpha=horner(pi,[2,6,12,20,30,42],5,20,1.e-9)
```

for this and several other different values of x_0 gives always the approximate root `alpha =-0.508507238094213` for which `p(alpha) = 0.484106445983341`. □

Newton's Method for Systems of Nonlinear Equations

Having considered methods for the solution of the single equation $f(x) = 0$, a natural (and commonly arising) extension is to consider the nonlinear system of equations

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \qquad \text{where} \qquad \mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n,$$

that is, the set of equations

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad i = 1, 2, \ldots, n. \tag{22}$$

Fixed point methods can be applied in $\mathbb{R}^n$ and Newton's method and its variants are by far the most widely used.

If $\mathbf{f}(\boldsymbol{\alpha}) = \mathbf{0}$ then by Taylors theorem in several variables

$$\mathbf{0} = \mathbf{f}(\boldsymbol{\alpha}) = \mathbf{f}(\mathbf{x}) + J(\boldsymbol{\alpha} - \mathbf{x}) + \text{higher order terms}$$

where J is the *Jacobian* matrix:

$$J = \{J_{ij}, i, j = 1, 2, \ldots, n\}, \quad J_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{x})$$

evaluated at $\mathbf{x}$ and the higher order terms involve 2nd partial derivates and quadratic terms in $\alpha_j - x_j$. In component form as in (22) this is

$$0 = f_i(\alpha_1, \alpha_2, \ldots, \alpha_n) = f_i(x_1, x_2, \ldots, x_n) + \sum_{j=1}^{n} J_{ij}(\alpha_j - x_j) + \text{higher order terms}$$

for each $i = 1, 2, \ldots, n$. Now, if we suppose that the higher order terms can be ignored, then rearranging we obtain the (hopefully!) better estimate for $\boldsymbol{\alpha}$

$$\mathbf{x}_{new} = \mathbf{x} - J^{-1}\mathbf{f}(\mathbf{x}).$$

Certainly this should be a better estimate of $\boldsymbol{\alpha}$ if $\mathbf{x}$ is already reasonably close to $\boldsymbol{\alpha}$. Newton's method is therefore:
select $\mathbf{x}_0 \in \mathbb{R}^n$ and for $k = 0, 1, 2, \ldots$ until convergence compute

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

or in a form which is more useful for computation:
select $\mathbf{x}_0 \in \mathbb{R}^n$ and for $k = 0, 1, 2, \ldots$ until convergence solve the linear system

$$J\boldsymbol{\delta} = -\mathbf{f}(\mathbf{x}_k) \quad \text{and set} \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta} \tag{23}$$

where the Jacobian matrix is evaluated at $\mathbf{x}_k$.

Example.
Writing (x_1, x_2) as (x, y) for the system

$$\begin{aligned} f_1(x, y) &= xy + y^2 - 2 = 0 \\ f_2(x, y) &= x^3 y - 3x - 1 = 0, \end{aligned}$$

we have

$$J = \begin{bmatrix} y & x + 2y \\ 3x^2 y - 3 & x^3 \end{bmatrix}$$

so that starting at $x_0 = 0, y_0 = 1$ we compute the next iterate via

$$\begin{bmatrix} 1 & 2 \\ -3 & 0 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\text{giving } \boldsymbol{\delta} = \begin{bmatrix} -1/3 \\ 2/3 \end{bmatrix} \text{ so that } \mathbf{x}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \mathbf{x}_0 + \boldsymbol{\delta} = \begin{bmatrix} -1/3 \\ 5/3 \end{bmatrix}.$$

The next iterate is calculated from the solution of

$$\begin{bmatrix} \frac{5}{3} & 3 \\ \frac{-11}{45} & \frac{-1}{27} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \frac{-2}{9} \\ \frac{7}{81} \end{bmatrix}$$

leading to

$$\mathbf{x}_1 = \begin{bmatrix} -0.357668 \\ 1.606112 \end{bmatrix}.$$

Though one could continue with rationals (and 'hand' calculation), this is clearly a computational procedure; the MATLAB functions for this problem

```
function[x,y] = vecnewton(xin,yin,tol)
  x=xin;y=yin; funvalue = fvec(x,y);  %initial (vector) function value
  while norm(funvalue)>tol, % whilst the Euclidean length (norm) of the
                            % vector function value is too large
    delta=-jacobian(x,y)\funvalue; x=x+delta(1), y=y+delta(2),
                       % Newton iteration
                       % \ solves the linearised equation system
                       % J delta = -f
    funvalue = fvec(x,y),  %new (vector) function value
  end
end

function[vecval]=fvec(x,y)
  f1 = x*y+y^2-2; f2= x^3*y-3*x-1;
  vecval = [f1;f2];   % column vector
end

function[jac]=jacobian(x,y)
  jac=[ y, x+2*y ; 3*x^2*y-3, x^3];
end
```

compute further iterates:

$$\mathbf{x}_2 = \begin{bmatrix} -0.357838 \\ 1.604407 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -0.357838 \\ 1.604406 \end{bmatrix} = \mathbf{x}_4 = \mathbf{x}_5 = \ldots$$

to the number of decimal places shown.

I must confess that I guessed that I would see convergence to the more obviously checkable zero given by $x = -1, y = 2$ from these staring values, but it takes something like

```
vecnewton(-1.3,2.8,1.e-10);
```

to obtain this other solution of this system of nonlinear equations. □

Some comments are in order:
(i) as for a scalar equation, Newton's method is only convergent if the starting value is close enough to the desired zero (root).
(ii) the Newton iterates are only defined so long as the Jacobian remains non-singular when evaluated at the iterates.

Optimization

One of the most significant applications of Newton's method is in optimization: one often desires to find the minimum (or at least a local minimum) value of a function $f : \mathbb{R}^n \to \mathbb{R}$. Provided the derivatives exist, the conditions for a stationary point are well known:

$$\mathbf{g}(\boldsymbol{\alpha}) = \nabla f(\boldsymbol{\alpha}) = \mathbf{0}.$$

The vector $\mathbf{g}$ with entries $g_i = \frac{\partial f}{\partial x_i}$ is the gradient. To guarantee that $\boldsymbol{\alpha}$ indeed gives minimality, we require some conditions of convexity of f, at least locally in a neighbourhood of $\boldsymbol{\alpha}$. Taylor's theorem is again useful provided that the partial derivatives exist:

$$\mathbf{0} = \mathbf{g}(\boldsymbol{\alpha}) = \mathbf{g}(\mathbf{x}) + H(\boldsymbol{\alpha} - \mathbf{x}) + O(\|\alpha - x\|^2), \tag{24}$$

where

$$H = [h_{ij}]_{i,j=1,2,\ldots,n},$$
$$h_{ij} = \frac{\partial g_i}{\partial x_j}(\mathbf{x}) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)(\mathbf{x}) = \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}), \quad (i, j = 1, \ldots, n).$$

Note that H is the Jacobian matrix of the (vector valued) function $\mathbf{g}(\mathbf{x})$, and this is also the Hessian matrix of the (real valued) function $f(\mathbf{x})$ at $\mathbf{x}$. Taking the Taylor development (24) around an approximate solution $\mathbf{x}_k$ and dropping the higher order terms, we obtain the system of linear equations

$$\mathbf{0} = \mathbf{g}(\mathbf{x}_k) + H(\mathbf{x}_{k+1} - \mathbf{x}_k),$$

where we now write $\mathbf{x}_{k+1}$ for $\boldsymbol{\alpha}$, as the solution

$$\mathbf{x}_{k+1} = \mathbf{x}_k - H^{-1}\mathbf{g}(\mathbf{x}_k) \tag{25}$$

of this approximate system merely gives another approximation of $\boldsymbol{\alpha}$, albeit a generally improved one. The updating rule (25) is the same as Newton updates for the nonlinear system of equations $\mathbf{0} = \mathbf{g}(\boldsymbol{\alpha})$.

Another way to motivate the method (25) is to approximate the objective function $f(x)$ by a quadratic model function

$$m_k(\mathbf{x}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^{\mathrm{T}}(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^{\mathrm{T}} B_k (\mathbf{x} - \mathbf{x}_k), \tag{26}$$

where B_k is an appropriate symmetric matrix, chosen so that $m_k(\mathbf{x}) \approx f(\mathbf{x})$ in a neighbourhood of $\mathbf{x}_k$, that is, $m_k(\mathbf{x})$ is a locally valid model for $f(\mathbf{x})$. Instead of minimising $f(\mathbf{x})$, it is simpler to minimise the model function. Its minimiser

$$\mathbf{x}_{k+1} = \arg\min m_k(\mathbf{x}) \tag{27}$$

is our updated approximation of a minimiser of $f(\mathbf{x})$. Setting up an updated model function $m_{k+1}(\mathbf{x})$ that is valid in a neighbourhood of $\mathbf{x}_{k+1}$, we obtain an iterative process. Note that B_k being symmetric, all its eigenvalues are real. When all the eigenvalues are positive, we say that B_k is *positive definite*, and it is then the case that $m_k(\mathbf{x})$ is a strictly convex function whose minimiser is found at the unique stationary point

$$bfx_{k+1} = \mathbf{x}_k - B_k^{-1}\nabla f(\mathbf{x}_k). \tag{28}$$

An obvious choice of quadratic model is to set $B_k = H = [\frac{\partial g_i}{\partial x_j}(\mathbf{x}_k)]$, so that $m_k(\mathbf{x})$ approximates $f(\mathbf{x})$ to second order at the current iterate $\mathbf{x}_k$. In this case, the *quasi-Newton update* (28) coincides with the Newton update (25).

The above discussion leads to the following observations:

i) When B_k is positive definite, the quasi-Newton update (25) moves to a point $\mathbf{x}_{k+1}$, where $f(\mathbf{x}_{k+1})$ is potentially smaller than $f(\mathbf{x}_k)$, since $\mathbf{x}_{k+1}$ is the minimiser of a model function that approximates $f(\mathbf{x})$ in a neighbourhood of $\mathbf{x}_k$. However, the descent condition

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k) \tag{29}$$

is not guaranteed in a situation where $\mathbf{x}_{k+1}$ lies far away from $\mathbf{x}_k$ where the model m_k is a bad fit for f. To impose the descent condition (29), one often combines the quasi-Newton method with a *line-search*: take a step

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \tau_k \mathbf{d}_k, \tag{30}$$

where $\mathbf{d}_k = -B_k^{-1}\nabla f(\mathbf{x}_k)$ is the quasi-Newton step used as a *search direction*, and $\tau_k > 0$ is an approximiate minimiser of the function

$$\phi(t) = f(\mathbf{x}_k + t\mathbf{d}_k).$$

Note that since B_k is positive definite, we have $\nabla f(\mathbf{x}_k)^{\mathrm{T}} B_k \nabla f(\mathbf{x}_k) > 0$, so that

$$phi'(0) = \nabla f(\mathbf{x}_k)^{\mathrm{T}}\mathbf{d}_k = -\nabla f(\mathbf{x}_k)^{\mathrm{T}} B_k \nabla f(\mathbf{x}_k) < 0,$$

showing that at least for small enough t, the descent condition $f(\mathbf{x}_k + t\mathbf{d}_k) < f(\mathbf{x}_k)$ is satisfied. The choice of τ_k thus guarantees that (29) holds true.

ii) Applied to the special case $B_k = H$, the above discussion shows that Newton's Method (25) can only be expected to converge to a local minimiser

$\mathbf{x}^*$ of f when the Hessian matrix of f at $\mathbf{x}^*$ is positive definite and the method is started sufficiently close to $\mathbf{x}^*$. Indeed, since Newton's Method is designed to solve $\mathbf{g}(\mathbf{x}) = 0$, it can also converge to a local maximiser of f when it is not started near a local minimiser. Line-searches prevent this phenomenon from occurring.

iii) Simple choices of B_k that guarantee that B_k is positive definite include the following, where σ_1 denotes the smalles eigenvalue of H,

$$B_k = H + \lambda \mathrm{I}, \text{ with } \lambda > -\sigma_1, \quad \text{(regularised Newton)},$$
$$B_k = \mathrm{I}, \quad \text{(steepest descent)}.$$

On large scale problems, the first of these two choices is computationally too expensive, while the second can lead to excessively slow convergence. Among numerous clever methods that were developed with the aim of combining the fast converence of the first of these methods with the low computational cost per iteration of the second, the so-called BFGS (Broyden– Fletcher–Goldfarb–Shanno) method is one of the most successful in practice.