Lecture Notes on Numerical Analysis

Song Wang
School of Mathematics & Statistics
The University of Western Australia
swang@maths.uwa.edu.au
Contents

1 Computer Arithmetic ........................................... 4
  1.1 Floating point numbers ...................................... 4
  1.2 Overflow and underflow ...................................... 5
  1.3 Floating-point arithmetic .................................... 7
  1.4 Computing sums .............................................. 8
  1.5 Perturbation analysis ......................................... 9
  1.6 Cancellation .................................................. 10
  1.7 Algorithms and convergence ................................ 11

2 Nonlinear Equations of One Variable .......................... 14
  2.1 Bisection method ............................................. 14
  2.2 Fixed point method .......................................... 16
  2.3 Newton’s method ............................................. 18
  2.4 The secant method ........................................... 22
  2.5 Quasi-Newton method ........................................ 23
  2.6 Mülle’s method ............................................... 24

3 Interpolation & Polynomial Approximation ...................... 25
  3.1 Lagrange Polynomial .......................................... 25
  3.2 Divided difference & Newton’s polynomial .................... 27
  3.3 Hermite interpolation ........................................ 31
  3.4 Cubic spline interpolation ................................... 33

4 Numerical Integration & Numerical Differentiation ............ 36
  4.1 Trapezoidal rule .............................................. 37
  4.2 Simpson’s rule .............................................. 38
  4.3 Newton-Cote formulas ........................................ 39
  4.4 Composite rules ............................................. 40
    4.4.1 Composite Simpson’s rule ................................ 41
    4.4.2 Composite trapezoidal and mid-point rules .............. 42
  4.5 Gauss quadrature ............................................ 42
  4.6 First derivatives ............................................ 45
  4.7 Second derivatives .......................................... 45
  4.8 Computational errors ........................................ 46
## Numerical Solution of Ordinary Differential Equations (ODEs)

5.1 Euler’s method .................................................. 47
5.2 Higher-order Taylor methods .................................. 49
5.3 Runge-Kutta and Other Methods .............................. 51
5.4 Multi-step Methods ................................................ 53
5.5 Implicit Methods ................................................... 55
5.6 Stability of One-Step Methods ................................. 56
5.7 Stability of Multi-step Methods ............................... 57

## Least Squares Approximation

6.1 Discrete case .................................................... 60
6.2 Fitting by exponential functions .............................. 62
6.3 Orthogonal polynomials & least-squares approximation ... 62

## Solution of Nonlinear Systems of Equations

7.1 Fixed point iterations .......................................... 64
7.2 Newton’s method ................................................ 67
7.3 Quasi-Newton methods ........................................ 67
7.4 The steepest descent method .................................. 68
Chapter 1

Computer Arithmetic

1.1 Floating point numbers

Base 10

Computer memory can only store a finite number of digits. Therefore, a question becomes apparent: given a fixed number of digits how can we define a representation so that it gives the largest coverage of real numbers? An obvious method to use is the \textit{scientific notation}, i.e., a number of very large or very small magnitude is represented as a truncated number multiplied by an appropriate power of 10. For example, $2.597 - 03$ represents $2.597 \times 10^{-3}$.

Example. Given 4 digits, the minimum and maximum positive decimal numbers can be expressed by the conventional method are .0001 and 9999 respectively (assume that decimal point does not take a space). However, if we are ready to sacrifice some accuracy, we can use two digits to represent the power of 10 and its sign and the rest to represent the first two non-zero digits and the sign of a number, then, the minimum and maximum positive numbers are respectively .01 $\times 10^{-9}$ and 99 $\times 10^{+9}$. Note that we may lose some accuracy. For example 2.345 can only be expressed as .23 $\times 10^{+1}$.

In general we have

$$x = s \, 0.m \cdot 10^e,$$

where $s$ denotes the sign, $m$ is called Mantissa (or $M$ bits) and $e$ is the exponent (or $E$ bits).

Base $\beta$

A base $\beta$ floating point number consists of a fraction $f$ containing the significant figure of the number and exponent $e$. The value of the number is

$$f \cdot \beta^e.$$ 

A floating-point number $a = f \cdot \beta^e$ is said to be normalised if

$$\beta^{-1} \leq 1$$

(For base 10, $\beta^{-1} = 0.1$). In other words, $a$ is normalised if the base-$\beta$ representation of its fraction has the form

$$f = 0.x_1x_2 \cdots, \quad \text{with} \quad x_1 \neq 0.$$
Obviously, $2.597 \cdot 10^{-3}$ is not normalized, while $0.2587 \cdot 10^{-3}$ is.

Commonly used bases:

- **binary** — base 2, used by most of computer systems.
- **decimal** — base 10, used in most of hand calculators.
- **hex** — base 16, IBM mainframes and clones.

**IEEE Standard (32-binary digits or bits)**

In this standard, the first bit is for the sign, bits 2–9 for exponent and bits 10–32 for fraction.

\[
\begin{array}{cccc}
0 & 1 & 9 & 32 \\
\hline
s & exponent & fraction \\
8 & 23 \\
\end{array}
\]

Numerical range: approximately $10^{-38}$ to $10^{38}$, or it represents 7 significant decimal digits. This is the so-called *single precision*. We may extend it to double precision, i.e., 64-bit storage, which has roughly a numerical range from $10^{-307}$ to $10^{307}$.

### 1.2 Overflow and underflow

The real number set is finite, but the representation is finite. This limitation (that \( e \) is finite) leads to overflow or underflow. Collectively, we have *exponent exception*. There are a few cases as listed below:

- When a number is too large to be presented by the finite exponent range, the result is said to have overflow. For example, $10^{60}$ can be expressed by a 32-bit memory block.
- Similarly, we have underflow, e.g. $10^{-60}$.
- Overflow is a fatal error as it normally cause the system to stop. The system normally displays NaN – Not a Number!.
- Underflow is normally not harmful as it can be replaced be zero.

Overflow can sometimes be avoided by scaling.

Example: Consider \( c = \sqrt{a^2 + b^2} \) with \( a = 10^{60} \) and \( b = 1 \). On a 5 digit decimal computer, assuming

- Digit 1 — sign
- Digits 2 and 3 — exponent
- Digits 4 and 5 — magnitude.
The computation will overflow when computing

\[(10^{60})^2 = 10^{120}\]

as 120 has three digits, exceeding the length of exponent. However, we can scale the problem by a parameter \(l\), i.e.,

\[c = l \sqrt{\left(\frac{a}{7}\right)^2 + \left(\frac{b}{7}\right)^2}.\]

If we choose \(l = \max\{|a|, |b|\} = 10^{60}\), then

\[c = 10^{60} \sqrt{1^2 + \left(\frac{1}{10^{60}}\right)^2}.\]

This should work, though \(\frac{1}{10^{60}}\) will cause underflow. This is because it can be set to zero and so \(c \approx 10^{60} \sqrt{1^2} = 10^{60}\).

Errors

- Rounding: round off. eg. \(2.6457513 \rightarrow 2.6458\).
- Chopping/truncation. \(2.6457513 \rightarrow 2.6457\).
- Absolute and relative errors: If \(p^*\) is an approximation to \(p\), then \(|p - p^*|\) and \(\frac{|p - p^*|}{|p|}\) are called absolute and relative errors respectively, providing that \(p \neq 0\).

Bounds on the relative error for rounding-off

Consider \(a = x.xxxxy\) rounded to \(b = x.xxxz\). If \(y \geq 5\), round up, otherwise, round down. Therefore,

\[|b - a| \leq 5 \cdot 10^{-5}.\]

On the other hand, we assume that \(x \neq 0\). Then \(|a| \geq 1\) and so

\[\frac{|b - a|}{|a|} \leq 5 \cdot 10^{-5} = \frac{1}{2} \cdot 10^{-4}.\]

From this we have the following general theory: rounding \(a\) to \(t\) decimal digits gives a number \(b\) satisfying

\[\frac{|b - a|}{|a|} \leq \frac{1}{2} \cdot 10^{-t+1}.\]

Bounds on the relative error for chopping

Similarly, if \(b\) is obtained by chopping \(a\) \((t\) digits), then

\[\frac{|b - a|}{|a|} \leq 10^{-t+1}.\]
Bounds for \( t \)-digit binary numbers

\[
\frac{|b - a|}{|a|} \leq \begin{cases} 2^{-t}, & \text{rounding} \\ 2^{-t+1}, & \text{chopping}. \end{cases}
\]

Let \( b = \text{fl}(a) \) denote the result of rounding or chopping \( a \) on a particular machine, and \( \varepsilon_M > 0 \) the upper bound on the relative error. If we set

\[
\varepsilon = \frac{b - a}{a}, \quad a \neq 0
\]

then \( b = a(1 + \varepsilon) \) and \( |\varepsilon| \leq \varepsilon_M \). In other words,

\[
\text{fl}(a) = a(1 + \varepsilon), \quad |\varepsilon| \leq \varepsilon_M.
\]

This \( \varepsilon_M \) is characteristic of the floating point arithmetic of the machine in question. It is called the rounding unit for the machine or *machine epsilon*. We sometimes refer it to as *machine accuracy*.

### 1.3 Floating-point arithmetic

In general, a combination (i.e., +, −, ×, /) of floating-point numbers will not representable as a floating number of the same size. For example, product of two 5-digit numbers will generally require 10 digits for its representation. Thus, the result of a floating-point operation can be represented only approximately.

Let \( \odot \) denote an operation (+, −, ×, /), then

\[
\text{fl}(a \odot b) = (a \odot b)(1 + \varepsilon), \quad |\varepsilon| \leq \varepsilon_M,
\]

by the above theory. This is ideal. However, some systems can return a difference larger then the relative error. Here is one example.

Example: Consider the computation of difference 1 − 0.999999 in 6-digit decimal arithmetic. If it is done by a 7-digit machine, we have 0.000001 or 0.100000 \( \cdot 10^{-5} \). However, on a 6-digit machine,

\[
0.999999 \rightarrow 0.99999 \quad \text{(chopping)}.
\]

So,

\[
1.00000 - 0.99999 = 0.1 \cdot 10^{-4}.
\]

The relative error is

\[
\text{rel. error} = \frac{0.1 \cdot 10^{-4} - 0.1 \cdot 10^{-5}}{0.1 \cdot 10^{-5}} = 9.
\]

Note that in the above we first approximate \( b \) by \( \tilde{b} \) and then evaluate \( a - \tilde{b} \). In reality we may need to approximate both \( a \) and \( b \), so that

\[
\text{fl}(a \pm b) = a(1 + \varepsilon_a) \pm b(1 + \varepsilon_b)
\]

with \( |\varepsilon_a|, |\varepsilon_b| \leq \varepsilon_M \).
1.4 Computing sums

Consider the computation of

\[ s_n = \text{fl}(x_1 + x_2 + \cdots + x_n). \]

When \( n = 2 \), we have

\[ s_2 = \text{fl}(x_1 + x_2) \]
\[ = (x_1 + x_2)(1 + \varepsilon_1) \]
\[ = x_1(1 + \varepsilon_1) + x_2(1 + \varepsilon_1), \quad |\varepsilon_1| \leq \varepsilon_M. \]

Similarly,

\[ s_3 = \text{fl}(x_1 + x_2 + x_3) \]
\[ = (s_2 + x_3)(1 + \varepsilon_2) \]
\[ = [x_1(1 + \varepsilon_1) + x_2(1 + \varepsilon_1)](1 + \varepsilon_2) + x_3(1 + \varepsilon_2), \]
\[ = x_1(1 + \varepsilon_1)(1 + \varepsilon_2) + x_2(1 + \varepsilon_1)(1 + \varepsilon_2) + x_3(1 + \varepsilon_2) \]

where \(|\varepsilon_1|, |\varepsilon_2| \leq \varepsilon_M\). Continue with this process we have

\[ s_n = \text{fl}(s_{n-1} + x_n) \]
\[ = x_1(1 + \varepsilon_1)(1 + \varepsilon_2) \cdots (1 + \varepsilon_{n-1}) \]
\[ + x_2(1 + \varepsilon_1)(1 + \varepsilon_2) \cdots (1 + \varepsilon_{n-1}) \]
\[ + x_3(1 + \varepsilon_2) \cdots (1 + \varepsilon_{n-1}) \]
\[ \vdots \]
\[ + x_n(1 + \varepsilon_{n-1}) \]
\[ =: x_1(1 + \eta_1) + x_2(1 + \eta_2) + \cdots + x_n(1 + \eta_n), \]

where

\[ 1 + \eta_i = (1 + \varepsilon_{i-1})(1 + \varepsilon_i) \cdots (1 + \varepsilon_{n-1}) \]

with \(|\varepsilon_i| \leq \varepsilon_M\) for all \( i = 1, 2, \ldots, n - 1 \) and \( \varepsilon_0 = 0 \). Let us consider

\[ 1 + \eta_{n-1} = (1 + \varepsilon_{n-2})(1 + \varepsilon_{n-1}) \]
\[ = 1 + (\varepsilon_{n-2} + \varepsilon_{n-1}) + \varepsilon_{n-2}\varepsilon_{n-1}. \]

Now, \(|\varepsilon_{n-2} + \varepsilon_{n-1}| \leq 2\varepsilon_M\) and \(|\varepsilon_{n-2}\varepsilon_{n-1}| \leq \varepsilon_M^2\). From these we have

\[ \eta_{n-1} \approx \varepsilon_{n-2} + \varepsilon_{n-1} \quad \Rightarrow \quad |\eta_{n-1}| \leq 2\varepsilon_M. \]

In general, we have that approximately,

\[ |\eta_1| \leq (n - 1)\varepsilon_M \]
\[ |\eta_i| \leq (n - i + 1)\varepsilon_M. \]
More precisely, for all \( n = 1, 2, \ldots \),

\[
1 + \eta = (1 + \varepsilon_1)(1 + \varepsilon_2) \cdots (1 + \varepsilon_n)
\]

\[
= 1 + \sum_{i=1}^{n} \varepsilon_i + \sum_{i \neq j} \varepsilon_i \varepsilon_j + \cdots + \varepsilon_1 \varepsilon_2 \cdots \varepsilon_n
\]

\[
\leq 1 + n \varepsilon_M + \frac{n(n + 1)}{2} \varepsilon_M^2 + \cdots + \varepsilon_M^n.
\]

It can be shown that

\[
\eta \leq 1.06 \varepsilon_M =: n \varepsilon'_M,
\]

where \( \varepsilon'_M \) is called the adjusted rounding error. So, the modified expressions are

\[
|\eta_1| \leq (n - 1) \varepsilon'_M
\]

\[
|\eta_i| \leq (n - i + 1) \varepsilon'_M.
\]

Example: Let \( \varepsilon'_M = 10^{-15} \) and assume that we have a computer with an addition operation rate of \( 1 \mu \text{sec} = 10^{-6} \text{ sec} \). Let us calculate how long it will take before the error is accumulated to 0.1 on this computer.

From \( |\eta_1| \leq (n - 1) \varepsilon'_M = 0.1 \) we get \( n \approx 10^{14} \), i.e., it takes about \( 10^{14} \) additions before the accumulated error becomes 0.1. The time needed by this machine is

\[
10^{14} \cdot 10^{-6} = 10^{8} \text{ secs} \approx 3.2 \text{ years}.
\]

Therefore, it takes about 3.2 years before the error is accumulated to 0.1.

**Backward analysis**

The expression

\[
s_n = x_1(1 + \eta_1) + \cdots + x_n(1 + \eta_n)
\]

along with the bounds all \( \eta_i \) is called a backward error analysis because the rounding errors made in the course of computation are projected backward onto the original data. An algorithm that has such an analysis is called *stable* or *backward stable*.

### 1.5 Perturbation analysis

Consider the evaluation of

\[
\sigma = x_1 + x_2 + \cdots + x_n.
\]

We suppose there is a perturbation in \( x_i \), i.e., \( x_i \to \bar{x}_i \) with

\[
\bar{x}_i = x_i(1 + \mu_i), \quad |\mu_i| \leq \varepsilon
\]

and look for a bound on the error in

\[
\bar{\sigma} = \bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_n.
\]
(This is called a perturbation analysis.) Clearly,

\[ |\bar{\sigma} - \sigma| = |x_1\mu_1 + x_2\mu_2 + \cdots + x_n\mu_n| \]

\[ \leq \sum_{i=1}^{n} |x_i||\mu_i| \]

\[ \leq \varepsilon \sum_{i=1}^{n} |x_i|. \]

Dividing by |\sigma|, we have

\[ \frac{|\bar{\sigma} - \sigma|}{|\sigma|} \leq \frac{\varepsilon \sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|} =: \varepsilon \kappa, \]

where \( \kappa = \frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|} \geq 1. \) This \( \kappa \) is a magnification factor and serves as a condition number for the problem.

**Example:** Consider the sum

\[ s = 5.00 \cdot 10^8 - 4.99 \cdot 10^8 + 1.00, \]

where all the numbers are experimental data with 3 significant digits. In this case, \( \varepsilon = 10^{-2}. \) So,

\[ \kappa = \frac{5 \cdot 10^8 + 4.99 \cdot 10^8 + 1.00}{(5 - 4.99) \cdot 10^8 + 1} \]

\[ \approx \frac{9.99 \cdot 10^8}{0.01 \cdot 10^8} = 9.99 \cdot 10^2. \]

Therefore, the relative error equals \( \varepsilon \kappa = 9.99, \) not small!

When all \( x_i \)'s are positive (or negative), \( \kappa = 1. \) In this case the problem is called perfectly conditioned, and the errors will not accumulate or accumulate slowly. However, if \( \kappa >> 1, \) the problem is said to be ill-conditioned.

### 1.6 Cancellation

Let us calculate

\[ \text{fl}(37654 + 25.784 - 37679) = 0.874 \]

on a 5-digit machine (or in 5-digit floating-point). This gives

\[ \text{fl}(37654 + 25.784) = 37680 \]

and

\[ \text{fl}(37680 - 37679) = 1. \]

This does not agree with 0.874. The result has only one significant digit. This is called *cancellation*. In fact, cancellation does not cause any problem as \( \text{fl}(37680 - 37679) = 1 \) is exact. The trouble comes from the first step, i.e.,

\[ \text{fl}(37654 + 25.784) = 37680. \]
So, we have only two significant digits in the second number. This is why the result is not accurate.

**The quadratic equation**

Let us consider solving \( x^2 - bx + c = 0 \) which has the roots

\[
x = \frac{b \pm \sqrt{b^2 - 4c}}{2}.
\]

If we take \( b = 3.6778 \) and \( c = 0.0020798 \), then

\[
x_1 = 3.67723441190 \ldots, \quad x_2 = 0.0005658809.
\]

An attempt to calculate the smallest root in 5-digit arithmetic gives

1. \( b^2 - 1.3526 \cdot 10^1 \).
2. \( 4c - 8.3192 \cdot 10^{-3} \).
3. \( b^2 - 4c - 1.3518 \cdot 10^1 \).
4. \( \sqrt{b^2 - 4c} - 3.6767 \cdot 10^0 \).
5. \( b - \sqrt{b^2 - 4c} - 1.1000 \cdot 10^{-3} \).
6. \( (b - \sqrt{b^2 - 4c})/2 - 5.5000 \cdot 10^{-4} \).

The computed value differs from the true value \( 5.6558809 \cdot 10^{-4} \) in the second significant number. The reason of the cancellation at step 5, where 3 significant digits were cancelled when computing \( 3.6778 - 3.6767 \). Cancellation only reveals a loss of information that occurred earlier. The real trouble occurred at step 3.

\[
b^2 - 4c = \text{fl}(13.543 - 0.0083192) = 1.3518 \cdot 10^1.
\]

This cancellation corresponds to replacing the number 0.0083192 by 0.008 and perform the difference exactly.

Can anything be done to save it? It all depends.

### 1.7 Algorithms and convergence

**Definition 1.1 (algorithm)**: A numerical algorithms is the combination of

1. **Input variables**.

2. A sequence of steps which manipulates the input variables along with additional temporary variables.

3. **Output variables**.
Definition 1.2 (Stability of algorithm) An algorithm is stable if any small change in the initial data only results in a small change in the final data. Otherwise, it is unstable. Furthermore, if the stability is satisfied only for certain choice of initial data, then the algorithm is called conditionally stable.

Definition 1.3 Suppose $E_0 > 0$ denotes an initial error and $E_n$ represents the magnitude of an error after $n$ steps/operations. Then,

- $E_n \approx CnE_0$ — linear error growth ($C > 0$ is a constant).
- $E_n \approx C^nE_0$ — exponential error growth if $C > 1$.

Linear growth is unavoidable, but exponential growth is fatal.

Example: Consider a general recurrence relation

$$ax_{n+1} + bx_n + cx_{n-1} = 0, \quad n = 1, 2, \ldots, \quad (1.1.7.1)$$

with $x_0 = \alpha$ and $x_1 = \beta$. Let us find a solution of the form $x_n = p^n$ where $p$ is to be determined. Substituting $x_n$ into (1.1.7.1) gives

$$ap^{n+1} + bp^n + cp^{n-1} = 0,$$

or

$$ap^2 + bp + c = 0.$$ 

This has the solution

$$p = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$ 

In particular, if $a = 3$, $b = -13/3$ and $c = 4$, then we have two roots: $p_1 = 4$ and $p_2 = 1/3$.

The general solution to the difference equation is

$$x_n = A \cdot 4^n + B \cdot \left(\frac{1}{3}\right)^n,$$

where $A$ and $B$ are two arbitrary constants. Using the initial conditions $\alpha$ and $\beta$ we have

$$x_n = \frac{3\beta - \alpha}{11} \cdot 4^n + \frac{12\alpha - 3\beta}{11} \cdot \left(\frac{1}{3}\right)^n.$$ 

Now, if $\alpha = 1$ and $\beta = 1/3$, then $A = 0$ and $B = 1$, and the solution becomes

$$x_n = \frac{12 - 1}{11} \left(\frac{1}{3}\right)^n = \left(\frac{1}{3}\right)^n.$$ 

Let us consider the computation of this solution. If $A = 0$ is exact, but $B = 1 + \varepsilon$, then

$$\tilde{x}_n = (1 + \varepsilon) \left(\frac{1}{3}\right)^n \to 0$$
as \( n \to \infty \). The absolute error is

\[
E_n = |\tilde{x}_n - x_n| = \varepsilon \frac{1}{3^n} \to 0
\]
as \( n \to \infty \). This implies that the solution is stable with respect to \( B \).

If there is also an initial error \( \delta \) in \( A \), i.e., \( A \approx 1 + \delta \), then

\[
\tilde{x}_n = (1 + \delta)4^n + (1 + \varepsilon) \left(\frac{1}{3}\right)^n.
\]

So, the error becomes

\[
E_n = |x_n - \tilde{x}_n| = \delta 4^n + \frac{\varepsilon}{3^n} \to \infty
\]
as \( n \to \infty \). Therefore, it is not stable with respect to \( A \).

Consider \( x_n = \frac{1}{3^n} \) again. If 3 becomes \( 3 + \varepsilon \) due to truncation for a very small \( \varepsilon \), then

\[
\tilde{x}_n = \frac{1}{(3 + \varepsilon)^n}.
\]

The error is

\[
E_n = \quad |x_n - \tilde{x}_n| \\
= \quad \left| \frac{1}{3^n} - \frac{1}{(3 + \varepsilon)^n} \right| \\
\leq \quad \frac{1}{3^n} + \frac{1}{(3 + \varepsilon)^n} \\
\leq \quad \begin{cases} 
\frac{2}{3^n} & \varepsilon \geq 0 \\
\frac{2}{(3+\varepsilon)^n} & \varepsilon < 0
\end{cases}
\]

\[
\to 0, \quad \text{as } n \to \infty.
\]

So, it is stable. It is easy to show that \( x_n = 4^n \) is not stable with respect to 4.
Chapter 2

Nonlinear Equations of One Variable

How far a cannon ball can travel?
The motion of the ball satisfies
\[ y''(t) = -g, \quad y(0) = 0, \quad y'(0) = V_0 \sin \theta, \]
where \( y \) is the displacement along the vertical direction (height), \( V_0 \) is the initial speed, \( g \) is the gravitational acceleration constant and \( \theta \) is the angle (from the horizontal axis). The solution of this initial value problem is
\[ y(t) = V_0 t \sin \theta - \frac{1}{2} gt^2. \]

When the ball touches the ground again at \( t = T \), we have
\[ y(T) = 0 = V_0 T \sin \theta - \frac{1}{2} gT^2. \]

This has tow solutions
\[ T = 0 \quad \text{and} \quad T = \frac{2V_0 \sin \theta}{g}. \]

The distance travelled is
\[ d_{\text{max}} = V_0 \cos \theta T = \frac{2V_0 \sin \theta \cos \theta}{g}. \]

Question: How long it is needed for the ball to reach the height \( h_0 \)?
Obviously, we need to solve
\[ y_0 = V_0 t \sin \theta - \frac{1}{2} gt^2 \]
for \( t \). This is a nonlinear equation in \( t \). In general we need to consider the solution of \( f(x) = 0 \).

2.1 Bisection method

Let us quote the Intermediate Value Theorem from calculus.
Theorem 2.1 If $f$ is continuous on $[a, b]$ and $g$ lies between $f(a)$ and $f(b)$, then there exists a point $x \in [a, b]$ such that $g = f(x)$.

Now, for the solution of $f(x) = 0$ on $[a, b]$, if $f(a) \cdot f(b) < 0$, then we can find an approximation to $x$ in the following way.

Let $c = (a + b)/2$. There are three possibilities.

1. $f(c) = 0 \Rightarrow c$ is a solution.
2. $f(c) \neq 0$, and $f(c) \cdot f(b) < 0$. $f(x) = 0$ has a solution in $[c, b]$.
3. $f(c) \neq 0$, and $f(a) \cdot f(c) < 0$. $f(x) = 0$ has a solution in $[a, c]$.

Clearly, we either have a solution or the solution is in an interval of which the size (length) is half of that of $[a, b]$. We then repeat the above process.

Algorithm (bisection)

INPUT $a, b$, the tolerance TOL and the maximum number of iterations $N_0$.

step 1. set $i = 1$;  
$FA = f(a)$

step 2 While $i \leq N_0$ do steps 3–6

step 3 set $c = a + (b - a)/2$;  
$FC = f(c)$

step 4 If $c = 0$ or $(b - c)/2 < TOL$, then  
OUTPUT $C$; STOP.

step 5 set $i = i + 1$

step 6 If $FA \cdot FC > 0$ then set $a = c$;  
$FA = FC$
else set $b = c$.

step 7 OUTPUT(’Method failed after $N_0$ iterations’)  
STOP.

Question: How many iterations are needed in order that the interval length is less then $\varepsilon$?

Let $L_0 = b - a$. From the construction of the bisection method we see that after $k$ iterations, the length becomes

$$L_k = \frac{L_0}{2^k}.$$  

We require

$$L_k \leq \varepsilon \Rightarrow \frac{L_0}{2^k} \leq \varepsilon \Rightarrow k \leq \log_2 \frac{L_0}{\varepsilon}.$$  

We choose $k = \left\lceil \log_2 \frac{L_0}{\varepsilon} \right\rceil$, where $\lceil \cdot \rceil$ denotes the ceiling function.

Example: If $b - a = 1$ and $\varepsilon = 10^{-6}$, then $k = 20$.  

15
2.2 Fixed point method

**Definition 2.1 (fixed point)** A fixed point of a function \( g(x) \) is a real number \( p \) such that \( p = f(p) \).

Example. Let \( g(x) = x^2 - 2 \). Then the fixed points of \( g \) can be found from \( g(x) = x^2 - 2 = x \), or by solving \( x^2 - x - 2 = 0 \). Solving this gives two fixed points \( x_1 = -1 \) and \( x_2 = 2 \).

**Definition 2.2** The set of functions which are continuous on \([a, b]\) is denoted as \( C[a, b] \).

**Theorem 2.2** If \( g \in C[a, b] \) and \( g(x) \in [a, b] \), \( \forall x \in [a, b] \), then \( g \) has a fixed point in \([a, b]\).

**Proof.** Consider \( h(x) = g(x) - x \). Since \( g(x) \in [a, b] \) for all \( x \in [a, b] \), we have that \( a \leq g(a) \) and \( g(b) \leq b \).

Using these we have

\[
h(a) = g(a) - a \geq 0 \quad \text{and} \quad h(b) = g(b) - b \leq 0.
\]

By the Intermediate Value Theorem we see that \( \exists c \in [a, b] \) such that \( h(c) = 0 \), or \( g(c) = c \).

So, \( c \) is a fixed point of \( g \). \( \square \)

**Theorem 2.3** In addition to the assumption in the above theorem, if \( g'(x) \) exists on \((a, b)\) and there exists a positive constant \( k < 1 \) such that

\[
|g'(x)| \leq k < 1, \quad \forall x \in (a, b),
\]

then, the fixed point is unique.

**Proof.** We prove it by contradiction. Assume that \( g \) has two fixed points \( p_1, p_2 \in (a, b) \) and \( p_1 \neq p_2 \). Without loss of generality we assume that \( p_1 < p_2 \). By the Mean Value Theorem, \( \exists d \in (p_1, p_2) \) such that

\[
g'(d) = \frac{g(p_2) - g(p_1)}{p_2 - p_1}.
\]

But \( g(p_i) = p_i \) for \( i = 1, 2 \). We have from the above that

\[
g'(d) = \frac{p_2 - p_1}{p_2 - p_1} = 1.
\]

This is a contradiction as we have \( |g'(x)| \leq k < 1 \) for all \( x \in (a, b) \). Therefore, \( p_1 = p_2 \). \( \square \)

Example. Show that \( g(x) = \cos x \) has a unique fixed point in \([0, 1]\).

Clearly, \( g \in C[0, 1] \). \( \cos x \) is decreasing on \([0, 1]\), and \( \cos 0 = 1 \) and \( \cos 1 > 0 \), we have \( g(x) \in [0, 1] \). Furthermore, \( |g'(x)| = |\sin x| < \sin 1 < 1 \) for all \( x \in (0, 1) \). Therefore, \( g \) has a unique fixed point in \([0, 1]\).

Example. Consider the function \( g(x) = x - x^3 - 4x^2 + 10 \) on \([1, 2]\).
Since \( g(1) = 6 \) and \( g(2) = -12 \), \( g(x) \notin [1, 2] \) for \( x \in [1, 2] \). So, it is not known whether there is a fixed point in \([1, 2]\).

Example. Consider the function \( g(x) = \left( \frac{10}{4 + x} \right)^{1/2} \) on \([1, 2]\).

From the expression we see that \( g \) is decreasing in \([1, 2]\) and \( g(1) = \sqrt{2}, g(2) = \sqrt{5/3} \).

This implies that \( g(x) \in [1, 2] \) for any \( x \in [1, 2] \). Also,

\[
|g'(x)| = \left| \frac{-\sqrt{10}}{2(x + 4)^{3/2}} \right| \leq \frac{5}{10 \cdot 5^{3/2}} < 0.15.
\]

So, by the above theorem we have that \( g \) has a unique fixed point in \([1, 2]\).

**Definition 2.3 (fixed point iteration)** The iteration \( p_n = g(p_{n-1}) \) for \( n = 1, 2, \ldots \) is called fixed-point iteration.

**Theorem 2.4 (Fixed-point Theorem)** Let \( g \in C[a, b] \) be such that \( g(x) \in [a, b] \) for all \( x \in [a, b] \). Suppose, in addition, that \( g'(x) \) exists and there is a constant \( K \in (0, 1) \) such that

\[ g'(x) \leq K \quad \forall x \in (a, b). \]

Then, for any \( p_0 \in [a, b] \), the fixed point iteration

\[ p_n = g(p_{n-1}), \quad n = 1, 2, \ldots \]

converges to the unique fixed point \( p \in [a, b] \).

**PROOF.** For any positive integer \( n \), using the Mean Value Theorem we have

\[
|p_{n+1} - p_n| = |g(p_n) - g(p_{n-1})| \\
\leq |g' (\xi_n)||p_n - p_{n-1}| \\
\leq K |p_n - p_{n-1}| \\
\leq K^2 |p_{n-1} - p_{n-2}| \\
\vdots \\
\leq K^n |p_1 - p_0|.
\]

Then, for any positive integers \( m > n \),

\[
|p_m - p_n| = |p_m - p_{m-1} + p_{m-1} + \cdots + p_{n+1} + p_{n+1} - p_n| \\
\leq |p_m - p_{m-1}| + \cdots + |p_{n+1} - p_n| \\
\leq (K^{m-1} + K^{m-2} + \cdots + K^n)|p_1 - p_0| \\
= K^n(1 + K + \cdots + K^{m-1-n})|p_1 - p_0| \\
\leq K^n \cdot \frac{1}{1 - K} |p_1 - p_0| \to 0
\]

as \( n \to \infty \). Therefore, \( \{p_n\} \) is a Cauchy sequence in \([a, b]\) because \( a \leq g(x) \leq b \) for all \( x \in [a, b] \). We thus have

\[ p = \lim_{n \to \infty} p_n \in [a, b]. \]
Since
\[ \lim_{n \to \infty} p_n = \lim_{n \to \infty} g(p_{n-1}) \Rightarrow p = g(p), \]
we have that \( p \) is the unique fixed point of \( g \) in \([a, b]\). \( \square \)

Example. Consider again \( g(x) = \left( \frac{10}{4 + x} \right)^{1/2} \) on \([1, 2]\). We have shown above that it has a unique fixed point. This fixed point can be approximated by the following fixed point iteration
\[ x_n = \left( \frac{10}{4 + x_{n-1}} \right)^{1/2}, \quad n = 1, 2, \ldots \]
with any initial guess \( x_0 \in (1, 2) \).

### 2.3 Newton’s method

**Definition 2.4** The set of functions which together with their up to and including \( k \)th order derivatives are continuous on \([a, b]\) is denoted as \( C^k[a, b] \).

By this definition and that for \( C[a, b] \) we have that \( C[a, b] \equiv C^0[a, b] \).

**Construction of the method**

Let \( f \in C^2[a, b] \) and consider the solution of
\[ f(x) = 0. \quad (2.2.3.1) \]
If \( \bar{x} \in [a, b] \) is an approximation to a solution \( p \) to (2.2.3.1) in the sense that \( |\bar{x} - p| \) is sufficiently small, then \( f(x) \) can be expanded at \( \bar{x} \) as
\[ f(x) = f(\bar{x}) + f'(\bar{x})(x - \bar{x}) + \frac{1}{2} f''(\xi(x))(x - \bar{x})^2, \]
where \( \xi(x) \) is a point between \( x \) and \( \bar{x} \). Replacing all the \( x \) in the above with \( p \) and since \( f(p) = 0 \), we have
\[ 0 = f(\bar{x}) + f'(\bar{x})(p - \bar{x}) + \frac{1}{2} f''(\xi(p))(p - \bar{x})^2. \quad (2.2.3.2) \]
Now, if \( |\bar{x} - p| \) is sufficiently small,
\[ (\bar{x} - p)^2 \ll |\bar{x} - p|. \]
Then, (2.2.3.2) can be approximated by
\[ f(\bar{x}) + f'(\bar{x})(p - \bar{x}) = 0, \]
from which we get
\[ p = \bar{x} - \frac{f(\bar{x})}{f'(\bar{x})}. \quad (2.2.3.3) \]
Obviously, we need to assume that \( f'(p) \neq 0 \). If \( f(p) = 0 = f'(p) \), then \( p \) is a saddle point of \( f \) and the problem becomes more complicated.

Eq.(2.2.3.3) motivates us to define the following algorithm.

**Algorithm (Newton).** Given \( x_0 \in (a, b) \) sufficiently close to a solution \( p \) of (2.2.3.1), we define the sequence

\[
x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots
\]  

(2.2.3.4)

**Geometric explanation**

In the figure, the function

\[
y = f(x_k) + f'(x_k)(x - x_k)
\]

represents the tangent line of the curve \( y = f(x) \) at \( x = x_k \). The solution to

\[
f(x_k) + f'(x_k)(x - x_k) = 0
\]

gives an approximation to \( p \) better than \( x_k \).

Example. Let \( f(x) = \frac{1}{x} - a, \ a \neq 0 \). We solve \( f(x) = 0 \). Obviously, the exact solution is \( p = 1/a \).

Differentiating \( f \) gives

\[
f'(x) = -\frac{1}{x^2}.
\]

So, Newton’s scheme becomes

\[
x_{k+1} = x_k - \frac{1/x_k - a}{-1/x_k^2} = 2x_k - ax_k^2.
\]

Choosing an \( x_0 \in (0, \infty) \), we can approximate \( a \) by the above iterative scheme.

Example. Let \( f(x) = x^2 - a, \ a > 0 \). The exact positive solution is \( p = \sqrt{a} \).

Applying the Newton method gives

\[
x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} = \frac{1}{2} \left( x_k + \frac{a}{x_k} \right).
\]

This is known as Babylonian approximation to \( \sqrt{a} \).
Convergence of Newton’s method

**Theorem 2.5** Let \( f \in C^2[a,b] \). If \( p \in [a,b] \) is such that \( f(p) = 0 \) and \( f'(p) \neq 0 \), then \( \exists \delta > 0 \) such that the Newton’s method generates a sequence \( \{x_k\} \) given in (2.2.3.4) converges to \( p \) for any \( x_0 \in [p-\delta, p+\delta] \).

**Proof.** Let \( g(x) = x - \frac{f(x)}{f'(x)} \). Clearly \( p \) is a fixed point of \( g \), as \( f(p) = 0 \) and \( f'(p) \neq 0 \).

Consider

\[
e_{k+1} = x_{k+1} - p = g(x_k) - g(p) = g'(\xi_k)(x_k - p) = g'(\xi_k)e_k,
\]

where \( \xi \) is a point between \( x_k \) and \( p \). Differentiating \( g \) gives

\[
g'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}.
\]

When \( x = p \),

\[
g'(p) = \frac{f(p)f''(p)}{(f'(p))^2} = 0.
\]

So, if \( x_0 \) is close to \( p \), say \( x_0 \in [p-\delta, p+\delta] \) for a sufficiently small \( \delta > 0 \), from the continuity of \( g' \) we have

\[
|g'(x_0)| \leq M < 1,
\]

where \( M \) is a positive constant. From this we have

\[
|e_1| = |g'(\xi_0)|(x_0 - p)| \leq M|e_0|.
\]

Similarly, we have

\[
|e_2| \leq M|e_1| \leq M^2|e_0| \\
\vdots \\
|e_k| \leq M|e_{k-1}| \leq \cdots \leq M^k|e_0| \to 0
\]

as \( k \to \infty \). Therefore, \( x_k \to p \) as \( k \to \infty \). \( \square \)

**Rate of convergence**

**Definition 2.5** Suppose \( x_n \to p \) as \( n \to \infty \) with \( x_n \neq p \) for all \( n \). If there is a \( \lambda > 0 \) and \( \alpha > 0 \) such that

\[
\lim_{n \to \infty} \frac{|x_{n+1} - p|}{|x_n - p|^\alpha} = \lambda,
\]

then, \( x_n \to p \) at a rate of order \( \alpha \) and with asymptotic error constant \( \lambda \). In particular,

- if \( \alpha = 1 \), the sequence is linearly convergent, and
• if $\alpha = 2$, it is quadratically convergent.

**Theorem 2.6 (Quadratic convergence of Newton’s method)** If $f \in C^3[a, b]$, then the Newton’s method is quadratically convergent.

**Proof.** From Theorem 2.5 we have

$$e_{k+1} = g(x_k) - g(p).$$

Now, expanding $g(x_k)$ at $p$ gives

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

$$= k g''(\xi_k)x_k^2$$

since $g'(p) = 0$. Therefore,

$$\frac{e_{k+1}}{e_k^2} = \frac{1}{2}g''(\xi_k).$$

Taking the limit,

$$\lim_{n \to \infty} \frac{e_{k+1}}{e_k^2} = \frac{1}{2}g''(p)$$

(2.2.3.5)

since $x_k \to p$ as $k \to \infty$. Note that

$$g' = \frac{-ff''}{(f')^2}$$

$$g'' = \frac{-(f'f'' + f'f''')(f')^2 + f''f'f'''}{(f')^4}$$

and so

$$g''(p) = \frac{-(f'(p))^3f''(p)}{(f'(p))^4} = \frac{f''(p)}{f'(p)} \neq \infty.$$ 

This, together with (2.2.3.5) imply that the rate is quadratic. \qed

**Slow death**

Though Newton’s method is quadratically convergent when $x_0$ is close to $p$, in practice, it may take a large number of iterations before it becomes quadratic. Here is an example.

**Example.** Consider $x_{k+1} = 2x_k - ax_k^2$. The fixed points for $2x - ax^2$ are $x = 0$ and $x = 1/a$. We choose $a = 10^{-10}$ and $x_0 = a$. Then

$$x_1 = 2x_0 - ax_0^2 = 2 \cdot 10^{-10} - 10^{-30} \approx 2 \cdot 10^{-10}$$

$$x_2 = 2x_1 - ax_1^2 = 4 \cdot 10^{-10} - 4 \cdot 10^{-30} \approx 4 \cdot 10^{-10}$$

Every iteration the value of $x_{k+1}$ is about $2x_k$, or

$$x_k \approx 2^k 10^{-10}.$$ 

If we want $2^k \cdot 10^{-10} = 1/a = 10^{10}$, we need above 66 iterations. This is a lot of iterations in practice.
2.4 The secant method

Recall the Newton’s algorithm

\[ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots \]

Sometimes \( f' \) is hard to derive. So, we may simply use an approximation as follows

\[ f'(x_k) \approx g_k := \frac{f(x_k + h_k) - f(x_k)}{h_k} \]

where \( h_k > 0 \) is a constant. There are two problems associated with this approximation. These are

1. The choice of \( h_k \) is tricky. If \( h_k \) is too large, \( g_k \) is an inaccurate approximation to \( f'(x_k) \). If \( h_k \) is too small, \( g_k \) is also inaccurate because of the rounding error.

2. The procedure requires one extra function evaluation \((f(x_k + h_k))\) per iteration. This is a serious problem in practice.

Now, starting from \( k = 1 \), we choose

\[ g_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}. \]

The secant method is then defined as

\[ x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})} \approx x_k - \frac{x_{k-1}f(x_k) - x_kf(x_{k-1})}{f(x_k) - f(x_{k-1})} \quad k = 1, 2, \ldots \]

where \( x_0 \) and \( x_1 \) are given.

This can be geometrically displayed by the figure.

Example. Use the secant method to solve \( f(x) = x - \cos x = 0 \).

The secant algorithm gives

\[ x_{k+1} = x_k - \frac{(x_{k-1} - \cos x_k)(x_k - x_{k-1})}{(x_k - x_{k-1}) - (\cos x_k - \cos x_{k-1})} \quad k = 1, 2, \ldots \]

where \( x_0 \) and \( x_1 \) are given.
2.5 Quasi-Newton method

In general a quasi-Newton method is of the form

\[ x_{k+1} = x_k - \frac{f(x_k)}{g_k}, \quad k = 0, 1, 2... \]

where \( g_k \) is an approximation to \( f'(x_k) \). The above secant method is a quasi-Newton’s method. Another choice is

\[ g_k = f'(x_0). \]

(Assume \( f'(x_0) \neq 0 \).) This is the constant slope method. Graphically, it is

\[ f \]

\[ a \rightarrow x \rightarrow x_0 \rightarrow b \]

Convergence of constant slope method

Assume that \( p \) satisfies \( f(p) = 0 \). Let

\[ e_{k+1} = x_{k+1} - p = g(x_k) - g(p), \]

where \( g(x) = x - \frac{f(x)}{f(x_0)} \). Using the Mean Value Theorem we have

\[ e_{k+1} = g'(\xi_k)(x_k - x) = g'(\xi_k)e_k \]

where \( \xi_k \) is a point between \( x_k \) and \( p \). But

\[ g'(x) = 1 - \frac{f'(x)}{f(x_0)} = \frac{f(x_0) - f(x)}{f'(x_0)}. \]

So, we need to assume that

\[ \left| \frac{f(x_0) - f(p)}{f(x_0)} \right| < K < 1. \]

In this case, when \( x_k \) is sufficiently close to \( p \) (so is \( \xi_k \), \( |g'(\xi_k)| < K < 1 \). Therefore,

\[ |e_{k+1}| \leq K|e_k| \leq \cdots \leq K^{k+1}|e_0| \rightarrow 0 \]

as \( k \rightarrow \infty \).
2.6 Müller’s method

While Newton’s method uses a local linear approximation to the function \( f \), Müller’s one is based on a quadratic approximation. This is done in the following two steps.

1. Given 3 points \( x_{k-2}, x_{k-1} \) and \( x_k \), find a quadratic function

\[
g(x) = a + bx + cx^2
\]

such that

\[
g(x_i) = f(x_i), \quad i = k - 2, k - 1, k.
\]

The coefficients \( a, b \) and \( c \) satisfy the system

\[
\begin{pmatrix}
1 & x_{k-2} & x_{k-2}^2 \\
1 & x_{k-1} & x_{k-1}^2 \\
1 & x_k & x_k^2
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
=
\begin{pmatrix}
f(x_{k-2}) \\
f(x_{k-1}) \\
f(x_k)
\end{pmatrix}
\]

2. Solve \( g(x) = 0 \) for \( x_{k+1} \) that lies nearest \( x_k \).

This is demonstrated graphically by the following figure.

**Quadratic fitting of a curve**

Given 3 points \((x_i, f(x_i))\), \( i = 0, 1, 2 \) on the curve \( y = f(x) \), we find a quadratic function of the form

\[
g(x) = a(x - x_2)^2 + b(x - x_2)^2 + c
\]

which passes through the three points. i.e., \( g \) satisfies

\[
\begin{align*}
g(x_0) &= a(x_0 - x_2)^2 + b(x_0 - x_2) + c = f_0 \\
g(x_1) &= a(x_1 - x_2)^2 + b(x_1 - x_2) + c = f_1 \\
g(x_2) &= c = f_2
\end{align*}
\]

where \( f_i = f(x_i) \) for \( i = 0, 1, 2 \). Solving this system gives

\[
\begin{align*}
c &= f_2 \\
b &= \frac{(x_0 - x_2)^2(f_1 - f_2) - (x_1 - x_2)^2(f_0 - f_2)}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)} \\
a &= \frac{(x_1 - x_2)^2(f_0 - f_2) - (x_0 - x_2)^2(f_1 - f_2)}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}
\end{align*}
\]
Chapter 3

Interpolation & Polynomial Approximation

Consider the following two questions:

- Given a set of points \((x_i, y_i), i = 0, 1, ..., n\) in a plane satisfying \(x_{i-1} < x_i\) for \(i = 1, 2, ..., n\). This determines a function relationship between \(x\) and \(y\). Can we find a systematic way to approximate the function value at any \(x \in (x_0, x_n)\)?

- If the answer to the above question is yes, how much error is involved in the approximation?

3.1 Lagrange Polynomial

Consider two points \((x_0, f_0)\) and \((x_1, f_1)\). We are to find a linear function \(f(x) = ax + bx\) passing through the points. This gives

\[
\begin{align*}
  f(x_0) &= a + bx_0 = f_0 \\
  f(x_1) &= a + bx_1 = f_1
\end{align*}
\]

Solving this system we have

\[
\begin{align*}
  a &= \frac{x_1 f_0 - x_0 f_1}{x_1 - x_0}, \\
  b &= \frac{f_0 - f_1}{x_0 - x_1}
\end{align*}
\]

Therefore,

\[
\begin{align*}
  f(x) &= \frac{x_1 f_0 - x_0 f_1}{x_1 - x_0} \cdot \frac{x}{x - x_0} + \frac{f_0 - f_1}{x_0 - x_1} \cdot x \\
  &= \left( \frac{x_1}{x_1 - x_0} + \frac{x}{x_0 - x_1} \right) f_0 + \left( \frac{-x_0}{x_1 - x_0} - \frac{x}{x_0 - x_1} \right) f_1 \\
  &= \frac{x - x_1}{x_0 - x_1} \cdot f_0 + \frac{x - x_0}{x_1 - x_0} \cdot f_1 \\
  &=: \ L_0(x) f_0 + L_1(x) f_1.
\end{align*}
\]
Therefore, it is a linear combination of \( L_0(x) \) and \( L_1(x) \). These \( L_0 \) and \( L_1 \) are called Lagrange interpolating polynomials of order 1. Graphically, \( L_0 \) is a line segment from \((x_0, 1)\) to \((x_1, 0)\), and \( L_1 \) is the one from \((x_0, 0)\) to \((x_1, 1)\) (cf. the figure).

In general, given \( n + 1 \) distinct nodes \( x_k, k = 0, 1, ..., n \), we can construct a unique polynomial \( L_{n,k}(x) \) of the form

\[
L_{n,k}(x) = \frac{(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}
\]  
(3.3.1.1)

It is easy to check that this polynomial satisfies

\[
L_{n,k}(x_i) = \begin{cases} 
0, & i \neq k, \\
1, & i = k. 
\end{cases}
\]  
(3.3.1.2)

\( L_{n,k} \) is called the \( n \)th Lagrange interpolating polynomial. Graphically it is demonstrated in the following figure.

**Theorem 3.1** If \( x_0, x_1, ..., x_n \) are \( n + 1 \) distinct numbers and \( f(x) \) is a function whose values are given at these points, then a unique polynomial \( P_n(x) \) of degree at most \( n \) exists with

\[
f(x_k) = P_n(x_k), \quad k = 0, 1, ..., n,
\]  
(3.3.1.3)

and \( P_n(x) \) is given by

\[
P_n(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)
\]  
(3.3.1.4)

where \( L_{n,k} \) is the polynomial defined in (3.3.1.1).

**PROOF.** The existence of such a polynomial is a consequence of (3.3.1.1), (3.3.1.2) and (3.3.1.4) because they are constructive.

To show the uniqueness of \( P_n(x) \), we need the use the fundamental theorem of algebra, i.e., a non-zero polynomial \( T(x) \) of degree \( \leq N \) has at most \( N \) roots. In order words, if \( T(x) \) is zero at \( N + 1 \) distinct nodes, it is identically zero.
Suppose there is another polynomial $Q_n(x)$ of order $\leq n$ satisfying (3.3.1.3). We let

$$T(x) = P_n(x) - Q_n(x).$$

Using (3.3.1.3) we have

$$T(x_k) = f(x_k) - f(x_k) = 0, \quad k = 0, 1, \ldots, n.$$ 

So, $T(x) \equiv 0$ and thus $P_n(x) = Q_n(x)$. \hfill \Box

Example. Consider $f(x) = \cos x$ over $[0, 1.2]$. Use the three nodes $x_0 = 0$, $x_1 = 0.6$ and $x_2 = 1.2$ to construct a quadratic interpolation polynomial $P_2(x)$.

The function values at the nodes are

$$f_0 = \cos 0 = 1, \quad f_1 = \cos 0.6 = 0.825336, \quad f_2 = \cos 1.2 = 0.362358.$$

So,

$$P_2(x) = f_0 \cdot \frac{(x - 0.6)(x - 1.2)}{(0 - 0.6)(0 - 1.2)} + f_1 \cdot \frac{(x - 0)(x - 1.2)}{(0.6 - 0)(0.6 - 1.2)} + f_2 \cdot \frac{(x - 0)(x - 0.6)}{(1.2 - 0)(1.2 - 0.6)}$$

$$= 1.38889(x - 0.6)(x - 1.2) - 2.292599x(x - 1.2) + 0.503275x(x - 0.6).$$

### 3.2 Divided difference & Newton’s polynomial

In the Lagrange interpolation (3.3.1.4), the expression $(x_0 - x_k) \cdots (x_{k-1} - x_k)(x_{k+1} - x_k) \cdots (x_n - x_k)$ often causes overflow or underflow. A better form is

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0) \cdots (x - x_{n-1}).$$

(3.3.2.5)

But we need to determine the coefficients $a_i$, $i = 0, 1, \ldots, n$. Clearly

$$a_0 = P_n(x_0) = f(x_0).$$

When $x = x_1$ in (3.3.2.5), we have

$$f(x_0) + a_1(x_1 - x_0) = P_n(x_1) = f(x_1).$$

This has the solution

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Similarly, setting $x = x_2$ in (3.3.2.5) gives

$$a_2 = \frac{f(x_2) - (a_0 + a_1(x_2 - x_0))}{(x_2 - x_0)(x_2 - x_1)}$$

$$= \frac{f(x_2) - (f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0})(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

$$= \left( \frac{f(x_2) - f(x_0)}{x_2 - x_0} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) / (x_2 - x_1).$$
For computational convenience, we rewrite the above as

\[ a_2 = \left( \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) / (x_2 - x_0). \]

The above motivates us to define the following divided differences

- **0th order:** \( f[x_i] = f(x_i) \)
- **1st order:** \( f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i} \)
- **2nd order:** \( f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \)
- **kth order:** \( f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+k+1}, \ldots, x_{i+k+1}] - f[x_i, x_{i+1}, \ldots, x_{k+k-1}]}{x_{i+k} - x_i} \)

Using this notation we have that the coefficients in (3.3.2.5) is given by

\[ a_k = f[x_0, x_1, \ldots, x_k], \quad k = 0, 1, \ldots, n, \]

and so (3.3.2.5) becomes

\[ P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0) \cdots (x - x_{k-1}). \]

This is called the Newton’s form of interpolant. The algorithmic description of the divided differences is

**Algorithm (Divided differences):**

**INPUT** \((x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)\) and let \(F_{k,0} = f_k\) for \(k = 0, 1, \ldots, n.\)

1. **Step 1.** For \(i = 1, 2, \ldots, n,\)
   - For \(j = 1, 2, \ldots, i\)
   - set \(F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-1}}.\)

2. **Step 2.** **OUTPUT**\((F_{0,0}, F_{1,1}, \ldots, F_{n,n})\) and
   \[ P(x) = \sum_{i=0}^{n} F_{i,i}\prod_{j=0}^{i-1}(x - x_j). \]

This case can also be written in the following table form

<table>
<thead>
<tr>
<th>(x_0)</th>
<th>(f[x_0])</th>
<th>(f[x_0, x_1])</th>
<th>(f[x_0, x_1, x_2])</th>
<th>(f[x_0, x_1, x_2, x_3])</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(f[x_1])</td>
<td>(f[x_1, x_2])</td>
<td>(f[x_1, x_2, x_3])</td>
<td>(f[x_1, x_2, x_3])</td>
</tr>
<tr>
<td>(x_2)</td>
<td>(f[x_2])</td>
<td>(f[x_2, x_3])</td>
<td>(f[x_2, x_3])</td>
<td>(f[x_2, x_3])</td>
</tr>
<tr>
<td>(x_3)</td>
<td>(f[x_3])</td>
<td>(f[x_3])</td>
<td>(f[x_3])</td>
<td>(f[x_3])</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
</tbody>
</table>
Theorem 3.2 Suppose that $f \in C^n[a, b]$ and $x_i, i = 0, 1,..., n$ are $n + 1$ distinct points in $[a, b]$. Then, $\exists \xi \in (a, b)$ such that

$$f[x_0, x_1, ..., x_n] = \frac{f^{(n)}(\xi)}{n!}.$$ 

PROOF. Let $g(x) = f(x) - P_n(x)$. Since $g_i(x) = 0$ for $i = 0, 1,..., n$, using the Generalised Rolle’s Theorem we have that $\exists \xi \in (a, b)$ such that

$$g^{(n)}(\xi) = 0, \quad \text{or} \quad f^{(n)}(\xi) - P_n^{(n)}(\xi) = 0. \quad (3.3.2.6)$$

Note that

$$P_n(x) = a_n x^n + \text{lower order terms} = f[x_0, x_1, ..., x_n] x^n + \text{lower order terms.}$$

Differentiating $P_n$ $n$ times gives

$$P_n^{(n)} = n! f[x_0, x_1, ..., x_n].$$

Therefore, from (3.3.2.6) we have

$$f[x_0, x_1, ..., x_n] = \frac{f^{(n)}}{n!}.$$ 

\[\square\]

In the case that the nodes are equally spaced, i.e.,

$$h = x_{i+1} - x_i \quad i = 0, 1,..., n - 1,$$

we can simplify the Newton’s form as follows.

Let $x = x_0 + sh$ for $0 \leq s \leq n$. We have $x - x_i = (s - i)h$, since $x_i = x_0 + ih$. Thus

$$P_n(x) = P_n(x_0 + sh)$$

$$= f[x_0] + sh f[x_0, x_1] + s(s - 1)h^2 f[x_0, x_1, x_2]$$

$$+ \cdots + s(s - 1) \cdots (s - n + 1)h^n f[x_0, ..., x_n]$$

$$= \sum_{k=0}^{n} s(s - 1) \cdots (s - k + 1)h^k f[x_0, ..., x_k].$$

Using the binomial coefficient notation

$$\binom{s}{k} = \frac{s(s - 1) \cdots (s - k + 1)}{k!},$$

we have

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} k! h^k f[x_0, ..., x_k].$$

This is called the Newton’s forward divided-difference formula.
Error bound

We demonstrate it using the simplest case, i.e., the case that \( n = 1 \). Consider

\[
e = f(x) - P_1(x),
\]

where

\[
P_1(x) = f[x_0] + f[x_0, x_1](x - x_0) = f[x_0] + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0).
\]

Now, expressing \( f \) as a Taylor’s expansion at \( x_0 \),

\[
f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(\xi)}{2}(x - x_0)^2,
\]

where \( \xi \) is a point between \( x_0 \) and \( x \), we have that

\[
e = f(x) - P_1(x) \\
= f'(x_0)(x - x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) + \frac{f''(\xi)}{2}(x - x_0)^2 \\
= \left[f'(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0}\right](x - x_0) + \frac{f''(\xi)}{2}(x - x_0)^2.
\]

The Taylor’s expansion of \( f(x_1) \) at \( x_0 \) is

\[
f(x_1) = f(x_0) + f'(x_0)(x_1 - x_0) + \frac{f''(\eta)}{2}(x_1 - x_0)^2
\]

with \( \eta \) in between \( x_0 \) and \( x_1 \). Therefore,

\[
f'(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f''(\eta)}{2}(x_1 - x_0)^2.
\]

Substituting this into the expression for \( e \), we get

\[
|e| = \left|\frac{f''(\eta)(x_1 - x_0)^2}{2} + \frac{f''(\xi)}{2}(x - x_0)^2\right| \\
\leq M(x_1 - x_0)^2 \\
\leq Mh^2.
\]

if \( |f''(x)| \leq M \) for all \( x \in [x_0, x_1] \), where \( M \) is a positive constant. This implies that the method is of 2nd order accuracy. Note that this error bound is not sharp. A better estimate is as follows.

Since \( e(x_0) = 0 = e(x_1) \), we have

\[
e'(\xi) = f'(\xi) - f[x_0, x_1] = 0
\]

for a \( \xi \in (x_0, x_1) \). In this case, \( |e(x)| \) attains the maximum at \( \xi \). The Taylor’s expansion of \( f(x_0) \) at \( \xi \) is

\[
f(x_0) = f(\xi) + f'(\xi)(x_0 - \xi) + \frac{f''(\eta)}{2}(x_0 - \xi)^2.
\]
So,
\[ e(\xi) = f(\xi) - [f(x_0) + f[x_0, x_1](\xi - x_0)] \\
= f(\xi) - \left[ f(\xi) + f'(\xi)(x_0 - \xi) + \frac{f''(\eta)}{2}(x_0 - \xi)^2 + f[x_0, x_1](\xi - x_0) \right] \\
= [f'(\xi) - f[x_0, x_1](\xi - x_0)](x_0 - \xi) + \frac{f''(\eta)}{2}(x_0 - \xi)^2 \\
= \frac{f''(\eta)}{2}(x_0 - \xi)^2. \]

If \( |f''(x)| \leq M, \forall x \in (x_0, x_1) \), then from the above we have
\[ |e(x)| \leq |e(\xi)| \leq \frac{M}{2}(x_1 - x_0)^2. \]

This improves the previous error bound by a factor 2. In fact, we can show that
\[ |e(x)| \leq \frac{M}{8}(x_1 - x_0)^2. \]

Let \( g \) be defined as
\[ g(t) = f(t) - P_1(t) - \frac{(t - x_0)(t - x_1)}{(x - x_0)(x - x_1)}. \]

Then, it is easy to see that
\[ g(x_i) = 0, \quad i = 0, 1, \quad g(x) = 0. \]

Using the generalized Rolle’s Theorem, there exists a \( \xi \in (x_0, x_1) \) such that \( g''(\xi) = 0 \).

That is,
\[ f''(\xi) - [f(x) - P_1(x)] \cdot \frac{2}{(x - x_0)(x - x_1)} = 0. \]

From this we have
\[ |f(x) - P_1(x)| = |f''(\xi)(x - x_0)(x - x_1)/2| \leq \frac{M}{8}(x_1 - x_0)^2 \]

since \( |(x - x_0)(x - x_1)| \) attains its maximum at \( x = (x_1 - x_0)/2 \).

### 3.3 Hermite interpolation

In the Lagrange polynomial approximation we find \( P_n(x) \) such that
\[ P_n(x_i) = f(x_i), \quad i = 0, 1, \ldots, n. \]

For many cases, it is important to preserve the derivatives as well. For example, we need to keep the convexity of a curve unchanged.
Definition 3.1 Let $x_i, i = 0, 1, ..., n$ be $n + 1$ distinct points in $[a, b]$ and $m_i$ be a non-negative integer associated with $x_i$. Suppose $f \in C^m[a, b]$, where $m = \max_{0 \leq i \leq n} m_i$. The osculating polynomial approximating $f$ is the polynomial $P(x)$ of least degree such that

$$\frac{d^k P(x_i)}{dx^k} = \frac{d^k f(x_i)}{dx^k}, \quad i = 0, 1, ..., n, \quad k = 0, 1, ..., m_i.$$

There are two special cases:

- When all $m_i = 0$, we have the Lagrange polynomial.

- When all $m_i = 1$, the resulting polynomial is called the Hermite polynomial.

Let us consider the approximation of $f(x)$ on $[x_0, x_1]$ by the Hermite polynomial. Since there are 4 degrees of freedom/conditions, i.e., $(x_i, f(x_i))$ and $(x_i, f'(x_0))$ for $i = 0, 1$, we let

$$H(x) = a + bx + cx^2 + dx^3.$$

Using the four conditions we get

$$H(x_i) = a + bx_i + cx_i^2 + dx_i^3 = f(x_i)$$

$$H'(x_i) = b + 2cx_i + 3dx_i^2 = f'(x_i)$$

for $i = 0, 1$. This linear system determines the 4 constants $a, b, c$ and $d$, and thus the Hermite interpolant.

In general we have

Theorem 3.3 If $f \in C^1[a, b]$ and $\{x_i\}_{0}^{n}$ are distinct points in $[a, b]$, the unique Hermite polynomial approximating $f$ is

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x),$$

where

$$H_{n,j}(x) = [1 - 2(x - x_j)L'_{n,j}(x_j)]L^2_{n,j}(x)$$

$$\hat{H}_{n,j}(x) = (x - x_j)L^2_{n,j}(x)$$

and $L_{n,j}(x)$ is the Lagrange polynomial defined in (3.3.1.1).

Example. Find the Hermite polynomial approximating

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$f(x_k)$</th>
<th>$f'(x_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>0.620</td>
<td>-0.522</td>
</tr>
<tr>
<td>1.6</td>
<td>0.455</td>
<td>-0.540</td>
</tr>
<tr>
<td>1.9</td>
<td>0.282</td>
<td>-0.581</td>
</tr>
</tbody>
</table>
First, let’s find the Lagrange polynomials and their derivatives:

\[
L_{2,0}(x) = \frac{50}{9}x^2 - \frac{175}{9}x + \frac{152}{9},
\]

\[
L'_{2,0}(x) = \frac{100}{9}x - \frac{175}{9},
\]

\[
L_{2,1}(x) = -\frac{100}{9}x^2 + \frac{320}{9}x - \frac{247}{9},
\]

\[
L'_{2,1}(x) = -\frac{200}{9}x + \frac{320}{9},
\]

\[
L_{2,2}(x) = \frac{50}{9}x^2 + \frac{145}{9}x + \frac{104}{9},
\]

\[
L'_{2,2}(x) = \frac{100}{9}x + \frac{145}{9}.
\]

Now, the Hermite basis functions are

\[
H_{2,0}(x) = [1 - 2(x - 1.3)(-5)]L_{2,0}^2(x)
\]

\[
= (10x - 12)\left(\frac{50}{9}x^2 - \frac{175}{9}x + \frac{152}{9}\right)^2
\]

\[
H_{2,1}(x) = \left(\frac{-100}{9}x^2 + \frac{320}{9}x - \frac{247}{9}\right)^2
\]

\[
H_{2,2} = 10(2 - x)\left(\frac{50}{9}x^2 + \frac{145}{9}x + \frac{104}{9}\right)^2
\]

\[
\hat{H}_{2,0}(x) = (x - 1.3)\left(\frac{50}{9}x^2 - \frac{175}{9}x + \frac{152}{9}\right)^2
\]

\[
\hat{H}_{2,1}(x) = (x - 1.6)\left(\frac{-100}{9}x^2 + \frac{320}{9}x - \frac{247}{9}\right)^2
\]

\[
\hat{H}_{2,2}(x) = (x - 1.9)\left(\frac{50}{9}x^2 + \frac{145}{9}x + \frac{104}{9}\right)^2
\]

Finally, the Hermite polynomial is

\[
H_5(x) = 0.620H_{2,0}(x) + 0.455H_{2,1}(x) + 0.282H_{2,2}(x)
\]

\[
-0.522\hat{H}_{2,0}(x) - 0.570\hat{H}_{2,1}(x) - 0.581\hat{H}_{2,2}(x).
\]

### 3.4 Cubic spline interpolation

The Lagrange interpolation is normally used as a piecewise polynomial approximation. This is because the error in the approximation is bounded by \(C(b-1)^p\) with \(p > 0\). Only when \(b-1 < 1\), the method converges. Therefore, the Lagrange interpolation is always used to approximate a function locally. This will cause the problem of 'un-smoothness' as the approximate curve may not be smooth at some of the mesh points. For example, a curve obtained by piecewise linear interpolants is not smooth at all the mesh points. Therefore, we need to find an interpolant which is piecewise polynomial and smooth.
Let \((x_i, f(x_i)), i = 0, 1, ..., n\) be a set of \(n + 1\) distinct points, and consider the interpolation of the segment \((x_k, f(x_k))\) by \(S(x)\) consisting of three parts \(S_{j-1}(x), S_j(x)\) and \(S_{j+1}(x)\) defined on the three subinterval \((x_k, x_{k+1}), k = j - 1, j, j + 1\), respectively. We require that the polynomial \(S(x)\) and its 1st and 2nd derivatives are continuous on \((x_{j-1}, x_{j+2})\). In particular, \(S(x)\) and the derivatives should be continuous at the points \(x_j\) and \(x_{j+1}\), or

\[
S_j^{(k)}(x_j) = S_{j-1}^{(k)}(x_j), \\
S_j^{(k)}(x_{j+1}) = S_{j+1}^{(k)}(x_{j+1})
\]

for \(k = 0, 1, 2\) and

\[
S_j(x_j) = f(x_j), \quad S_j(x_{j+1}) = f(x_{j+1}), \\
S_{j-1}(x_{j-1}) = f(x_{j-1}), \quad S_{j+1}(x_{j+2}) = f(x_{j+2}).
\]

The number of equations in the above is 10. We may also impose either the free or natural boundary condition

\[
S''_{j-1}(x_{j-1}) = 0 = S''_{j+1}(x_{j+2})
\]

or the clamped boundary condition

\[
S'_{j-1}(x_{j-1}) = f'(x_{j-1}), \quad S'_{j+1}(x_{j+2}) = f'(x_{j+2}).
\]

Therefore, altogether we have 12 degrees of freedom. This implies that \(S(x)\) can have up to 12 unknown constants. We now define

\[
S_k(x) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3
\]

for \(k = j - 1, j, j + 1\). There are 12 unknown constants which can be determined by the above 12 equations. Clearly, setting \(x = x_k\), we get

\[
a_k = S_k(x_k) = f(x_k)
\]

for \(k = j - 1, j, j + 1\).

The above idea can be extended to the set of \(n + 1\) nodes \((x_j, f(x_j)), j = 0, 1, ..., n\) so that the constant \(a_k\) is given by

\[
a_k = S_k(x_k) = f(x_k), \quad k = 0, 1, ..., n - 1
\]

(3.3.4.7)

From the condition \(S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_{j+1}\) we have

\[
a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 = a_{j+1},
\]

(3.3.4.8)

where \(h_j = x_{j+1} - x_j\). Differentiating \(S_j\) gives

\[
S'_j(x_j) = [b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2]_{x=x_j} = b_j.
\]

From \(S'_j(x_{j+1}) = S'_{j+1}(x_{j+1})\) we have

\[
b_n = S'(x_n) = \frac{b_{n+1} - b_{n-1}}{h_n} = \frac{b_{n+1} - b_{n-1}}{2h_n}
\]

(3.3.4.9)

\[
b_{j+1} = b_{j} + 2c_{j} h_{j} + 3d_{j} h_{j}^2
\]

(3.3.4.10)
for \( j = 0, 1, \ldots, n - 1 \). Continuity of \( S''(x) \) gives

\[
S''(x_n)/2 = c_n, \quad S''(x_j) = 2c_j,
\]

and \( S''_j(x_{j+1}) = S''_{j+1}(x_{j+1}) \) yields

\[
c_{j+1} = c_j + 3d_j h_j.
\]

From this we have

\[
d_j = \frac{c_{j+1} - c_j}{3h_j}. \tag{3.3.4.11}
\]

Substituting this into (3.3.4.8) and (3.3.4.10) we have

\[
a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}) \tag{3.3.4.12}
\]

\[
b_{j+1} = b_j + h_j(c_j + c_{j+1}). \tag{3.3.4.13}
\]

Solving (3.3.4.12) for \( b_j \) gives

\[
b_j = \frac{a_{j+1} - a_j}{h_j} - \frac{h_j}{3}(2c_j + c_{j+1}). \tag{3.3.4.14}
\]

Substituting this into (3.3.4.13) and re-arranging we get

\[
h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_jc_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_i - a_{j-1}) \tag{3.3.4.15}
\]

for \( j = 1, 2, \ldots, n - 1 \). This is a tri-diagonal system if \( c_0 \) and \( c_n \) are given. We thus have

- Solution of (3.3.4.15) gives \( c_j \).
- Solution of (3.3.4.14) gives \( b_j \).
- Solution of (3.3.4.11) gives \( d_j \).
- \( a_j \) is given by (3.3.4.7).

**Theorem 3.4** If \( f \) is defined at \( a = x_0 < x_1 < \cdots < x_n = b \), then \( f \) has a unique natural spline interpolant \( S(x) \) on the notes \( x_j, j = 0, 1, \ldots, n \), i.e., \( S(x) \) satisfies \( S''(a) = S''(b) = 0 \).

**Proof.** From \( S''(a) = S''(b) = 0 \) we have that \( c_0 = c_n = 0 \). Other coefficients can be uniquely determined by (3.3.4.15), (3.3.4.14), (3.3.4.11) and (3.3.4.7).

Similarly, the spline satisfying the clamped boundary condition is also uniquely defined.
Chapter 4

Numerical Integration & Numerical Differentiation

Numerical Integration/Numerical Quadrature Rules

Many integrals in practice cannot be or really hard to be done exactly. For example, the integrals
\[ \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \quad \text{and} \quad \int_a^b \sin x^2 dx. \]
The first is the error function which cannot be evaluated exactly for any finite \( x > 0 \). In this case, only numerical value of such an integral can be obtained. In fact, the definition of definite integral provides the simplest numerical quadrature rule as demonstrated below.

Consider the integral \( \int_a^b f(x) dx \), and divide \([a, b] \) into \( n \) equally spaced subintervals with nodes
\[ x_i = a + \frac{i}{n} \cdot (b - a), \quad i = 0, 1, ..., n. \]
By definition, the integral is equal to
\[ \int_a^b f(x) dx = \lim_{n \to \infty} \sum_{i=1}^{n} f(\xi_i) \cdot \frac{b-a}{n}. \]
where \( \xi_i \in (x_{i-1}, x_i) \) is arbitrary. Therefore, when \( n \) is large, we have
\[ \int_a^b f(x) dx \approx \frac{1}{n} \sum_{i=1}^{n} f(\xi_i). \]
More generally, we let \( x_i, i = 0, 1, ..., n \) be \( n + 1 \) distinct points in \([a, b]\) with \( x_0 = a \) and \( x_n = b \), and approximate \( f(x) \) on \([a, b]\) by the Lagrange interpolant
\[ f(x) \approx P_n(x) = \sum_{i=0}^{n} f(x_i)L_{n,i}(x). \]
Then, the integral can be approximated by

\[
\int_a^b f(x) \approx \int_a^b P_n(x)dx = \sum_{i=0}^n f(x_i) \int_a^b L_{n,i}(x)dx. \tag{4.4.0.1}
\]

The last integral can be evaluated exactly as \(L_{n,i}\) is a polynomial of order \(n\).

### 4.1 Trapezoidal rule

Consider the integral \(\int_a^b f(x)dx\) and let \(n = 2\) in (4.4.0.1). Without loss of generality, we assume that \(a = 0\) and \(b = h\). Approximating \(f(x)\) by the linear Lagrange interpolation gives

\[
\int_0^h f(x)dx \approx \int_0^h \left[ f(0) + \frac{f(h) - f(0)}{h} x \right] dx = \frac{f(0) + f(h)}{2} \cdot h =: T(f).
\]

For the integral of \(f\) on a general interval \([a, b]\), we let \(h = b - a\) and the transformation \(y = x - a\) transforms \([a, b]\) to \([0, h]\).

#### Error bound

We consider upper bound for

\[
\left| \int_0^h f(x)dx - T(f) \right| = \left| \int_0^h [f(x) - P_1(x)]dx \right|. \tag{4.4.1.2}
\]

Let

\[
g(t) = f(t) - P_1(t) - [f(x) - P_1(x)] \cdot \frac{(t-x_0)(t-x_1)}{(x-x_0)(x-x_1)}
\]

for an \(x \in (x_0, x_1)\). (Note that \(x_0 = 0\) and \(x_1 = h\) in the case here.) Then, we have \(g(x_0) = 0 = g(x_1)\). Furthermore,

\[
g(x) = f(x) - P_1(x) - [f(x) - P_1(x)] \cdot \frac{(x-x_0)(x-x_1)}{(x-x_0)(x-x_1)} = 0.
\]

Using the generalized Rolle’s theorem we have

\[
g''(\xi) = f''(\xi) - P_1''(\xi) - [f(x) - P_1(x)] \cdot 2 = 0
\]

for an \(\xi \in (a, b)\). But \(P_1'' = 0\). So,

\[
f(x) - P_1(x) = \frac{1}{2} f''(\xi)(x-x_0)(x-x_1).
\]

Applying this to (4.4.1.2) gives

\[
\left| \int_0^h f(x)dx - T(f) \right| \leq \frac{1}{2} \int_0^h \left| f''(\xi) x(x-h) \right| dx \\
= \frac{1}{2} \left| f''(\eta) \int_0^h x(x-h) dx \right| \\
\leq \frac{M}{12} h^3,
\]

where \(M > 0\) is such that \(|f''(x)| \leq M\) for all \(x \in [0, h]\). In the above we used the weighted mean value theorem since \(x(x-h)\) does not change sign in \((a, b)\). Therefore, the Trapezoidal’s quadrature rule is of 3rd order accuracy.
4.2 Simpson’s rule

We approximate $f$ on $[a, b]$ by a quadratic function

$$P_2(x) = \frac{(x-x_1)(x-x_2)}{h_1 h_2}f(x_0) + \frac{(x-x_0)(x-x_2)}{-h_1 h_2}f(x_1) + \frac{(x-x_0)(x-x_1)}{(h_1 + h_2) h_2}f(x_2),$$

where $h_1 = x_1 - x_0$ and $h_2 = x_2 - x_1$. Then, we have

$$\int_a^b f(x)dx \approx \int_a^b P_2(x)dx.$$

This is certainly more accurate the the Trapezoidal’s rule. But we can derive a quadrature rule in a different way – i.e., the undetermined coefficients. This gives the Simpson’s rule.

For simplicity and without loss of generality, we let $x_0 = 0$, $x_1 = 1/2$ and $x_2 = 1$, so that $h = 1/2$. Assume that the quadrature rule is of the form

$$\int_0^1 f(x)dx \approx \int_0^1 L(x)dx = A_0f(0) + A_1 f\left(\frac{1}{2}\right) + A_2 f(1),$$

where $A_0, A_1$ and $A_2$ are constants to be determined. To determine these constants, we require that

$$\int_0^1 f(x)dx = \int_0^1 L(x)dx$$

holds for $f(x) = 1, x$ and $x^2$.

Case 1: $f(x) = 1$

$$A_0 + A_1 + A_2 = \int_0^1 dx = 1 \quad (4.4.2.3)$$

Case 2: $f(x) = x$

$$A_0 \cdot 0 + A_1 \cdot \frac{1}{2} + A_2 \cdot 1 = \int_0^1 xdx = \frac{1}{2}. \quad (4.4.2.4)$$

Case 3: $f(x) = x^2$

$$A_0 \cdot 0 + A_1 \cdot \frac{1}{4} + A_2 \cdot 1 = \int_0^1 x^2dx = \frac{1}{3}. \quad (4.4.2.5)$$

Solving these $3 \times 3$ system gives

$$A_0 = A_2 = \frac{1}{6}, \quad A_1 = \frac{2}{3}.$$

So,

$$\int_0^1 f(x)dx \approx \frac{1}{6}f(0) + \frac{2}{3}f\left(\frac{1}{2}\right) + \frac{1}{6}f(1) = \frac{h}{3} \left[ f(0) + 4f\left(\frac{1}{2}\right) + f(1) \right].$$

In general we have

$$\int_{x_0}^{x_2} f(x)dx \approx \frac{h}{3} \left[ f(x_0) + 4f(x_1) + f(x_2) \right].$$

This is the Simpson’s rule. It can be shown that the error is given by

$$\int_{x_0}^{x_2} f(x) - \frac{h}{3} \left[ f(x_0) + 4f(x_1) + f(x_2) \right] = -\frac{h^5}{90} f^{(4)}(\xi)$$

where $\xi \in (x_0, x_2)$. 

38
**Definition 4.1 (Degree of accuracy or precision)** The degree of accuracy of a quadrature formula is the largest positive integer \( n \) such that the formula is exact for \( x^k, k = 0, 1, ..., n \).

Clearly we have

- Trapezoidal rule has a degree of accuracy 1.
- Simpson’s rule has a degree of accuracy 2.

### 4.3 Newton-Cote formulas

**\((n + 1)\)-point closed Newton-Cotes rule**

Let \( x_k, k = 0, 1, ..., n \), be \( n + 1 \) distinct points in \([a, b]\) with \( x_0 = a \) and \( x_n = b \). We wish to find \( a_k, k = 0, 1, ..., n \) such that

\[
\int_a^b f(x)dx = \sum_{j=0}^{n} a_j f(x_j)
\]

holds for all polynomial \( f(x) \) or order \( \leq n \). Let

\[
L_{n,i}(x) = \prod_{i=0,i\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)}
\]

be the Lagrange polynomial satisfying \( L_i(x_j) = \delta_{ij} \), where \( \delta_{ij} \) denotes the Kronecker delta. Then, replacing \( f \) by \( L_i \) in the above expression we have

\[
\int_a^b L_{n,i}(x)dx = \sum_{j=0}^{n} a_j L_{n,i}(x_j) = a_i
\]

for \( i = 0, 1, ..., n \). The (closed) Newton-Cotes formula is

\[
\int_a^b f(x)dx \approx \sum_{j=0}^{n} f(x_j) \int_a^b L_{n,j}(x)dx.
\]

**Theorem 4.1** Suppose that \( \sum_{j=0}^{n} a_j f(x_j) \) denotes the \((n+1)\)-point Newton Cotes formula with \( a = x_0 \) and \( b = x_n \), and \( h = (b - a)/n \). Then \( \exists \xi \in (a, b) \) such that

\[
\int_a^b f(x)dx = \sum_{j=0}^{n} a_j f(x_j) + \frac{h^{n+2} f^{(n+2)}(\xi)}{(n+2)!} \int_0^n t^2(t-1)\cdots(t-n)dt
\]

if \( n \) is even and \( f \in C^{n+2}[a, b] \), and

\[
\int_a^b f(x)dx = \sum_{j=0}^{n} a_j f(x_j) + \frac{h^{n+2} f^{(n+1)}(\xi)}{(n+1)!} \int_0^n t(t-1)\cdots(t-n)dt
\]

if \( n \) is odd and \( f \in C^{n+1}[a, b] \).
Examples are

- \( n = 1 \): Trapezoidal rule.
- \( n = 2 \): Simpson’s rule.
- \( n = 3 \): Simpson’s Three-Eighths rule

\[
\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8} [f_0 + 3f_1 + 3f_2 + f_3] - \frac{3h^5}{80} f^{(4)}(\xi).
\]

- \( n = 4 \):

\[
\int_{x_0}^{x_4} f(x)dx = \frac{2h}{45} [7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4] - \frac{8h^7}{945} f^{(6)}(\xi).
\]

\((n + 1)\)-point open Newton-Cotes rule

Let \( x_i, i = -1, 0, 1, ..., n + 1 \) be points satisfying

\[ x_{-1} = a < x_0 < x_1 < \cdots < x_n < x_{n+1} = b \]

and \( h = (b - a)/(n + 2) \). Then, \( \exists \xi \in (a, b) \) such that

\[
\int_a^b f(x)dx = \sum_{j=0}^{n} \int_a^b L_j(x)dx + \frac{h^{n+3} f^{(n+2)}(\xi)}{(n + 2)!} \int_{-1}^{n+1} t^2(t-1) \cdots (t-n)dt
\]

if \( n \) is even and \( f \in C^{n+2}[a, b] \), and

\[
\int_a^b f(x)dx = \sum_{j=0}^{n} \int_a^b L_j(x)dx + \frac{h^{n+2} f^{(n+1)}(\xi)}{(n + 1)!} \int_{-1}^{n+1} t(t-1) \cdots (t-n)dt
\]

if \( n \) is odd and \( f \in C^{n+1}[a, b] \).

A typical example is the case that \( n = 0 \). It is the mid-point rule

\[
\int_{x_{-1}}^{x_1} f(x)dx = 2hf(x_0) + \frac{h^3}{3} f''(\xi), \quad x_0 = \frac{a + b}{2} = \frac{x_{-1} + x_1}{2}.
\]

### 4.4 Composite rules

From the above discussion we see that \( h \) need to be small in order that the approximate error is small (normally bounded by \( h^p \) for a positive number \( p \)). Therefore, if \( b - a \) is large we have to divide it into a number of subintervals and apply a quadrature rule within each of the subintervals. This gives a composite quadrature rule.
4.4.1 Composite Simpson’s rule

Let \( x_i = a + ih \) for \( i = 0, 1, \ldots, n \) with \( h = (b-a)/n \), and denote \( f_i = f(x_i) \) for \( i = 0, 1, \ldots, n \).

On \([x_i, x_{i+2}]\), the Simpson’s rule is

\[
\int_{x_i}^{x_{i+2}} f(x) \, dx \approx \frac{h}{3} (f_i + 4f_{i+1} + f_{i+2}).
\]

Now, if \( n \) is even, we group the intervals into

\([x_0, x_2], [x_2, x_4], \ldots, [x_{n-2}, x_n]\)

and so

\[
\frac{3}{h} \int_a^b f(x) \, dx \approx f_0 + 4f_1 + f_2 + 4f_3 + f_4 + f_6 + f_8 + \cdots + f_n.
\]

Therefore,

\[
\int_a^b f(x) \, dx \approx \frac{h}{3} [f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n].
\]

**Theorem 4.2** Let \( f \in C^4[a, b] \), \( n \) be even, \( h = (b-a)/n \) and \( x_j = a + jh \) for \( j = 0, 1, \ldots, n \). Then, \( \exists \mu \in (a, b) \) such that

\[
\int_a^b f(x) \, dx = \frac{h}{3} \left[ f(a) + 2 \sum_{j=1}^{n/2-1} f(x_{2j}) + 4 \sum_{j=1}^{n/2} f(x_{2j-1}) + f(b) \right] - \frac{b-a}{180} h^4 f^{(4)}(\mu).
\]

The following is an algorithm for realizing Simpson’s rule.

**Algorithm:**

INPUT \( a, b \), even positive integer \( n \).

Step 1 \( h = (b-a)/n \).

Step 2 \( XI_0 = f(a) + f(b) \).  
\( XI_1 = 0 \)  
\( XI_2 = 0 \).

Step 3 For \( i = 1, 2, \ldots, n-1 \), do \( X = a + ih \)  
If \( i \) is even, then \( XI_2 = XI_2 + f(x) \), else \( XI_1 = XI_1 + f(x) \).

Step 4 \( XI = h(XI_0 + 2 \ast XI_1 + 4 \ast XI_2)/3 \).

Step 5 OUTPUT(XI).  
STOP.
4.4.2 Composite trapezoidal and mid-point rules

Let \( h = (b - a)/n \) and \( x_j = a + jh \) for \( j = 0, 1, \ldots, n \). Then,

\[
\frac{2}{h} \int_a^b f(x) \, dx \approx f_0 + \frac{f_1}{2} + f_2 + \frac{f_{n-2}}{2} + f_{n-1} + f_n
\]

\[
= f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n.
\]

From this we have

\[
\int_a^b f(x) \, dx \approx \frac{h}{2} \left[ f(a) + 2 \sum_{j=1}^{n-1} f_j + f(b) \right].
\]

This is the composite trapezoidal rule. The error term is

\[
E_n = -\frac{b - a}{12} h^2 f''(\mu)
\]

for \( \mu \in (a, b) \).

Similarly, the composite mid-point rule is

\[
\int_a^b f(x) \, dx \approx 2h \sum_{j=0}^{n/2} f(x_{2j}) + \frac{b - a}{6} h^2 f''(\mu),
\]

where \( n \) is a very positive integer, \( h = (b - 1)/(n+2) \) and \( x_j = a + (j+1)h \). The mid-point rule can also be expressed as

\[
\int_a^b f(x) \, dx \approx h \sum_{j=0}^{n-1} f\left(\frac{x_j + x_{j+1}}{2}\right)
\]

4.5 Gauss quadrature

Consider

\[
\int_{-1}^{1} f(x) \, dx \approx c_1 f(x_1) + c_2 f(x_2)
\]

where \(-1 \leq x_1, x_2 \leq 1\). Clearly, we can determine the Trapezoidal rule by forcing the above formula to be exact for \( f = 1 \) and \( f = x \), assuming \( x_1 \) and \( x_2 \) are fixed. This gives the degree of precision 1. Since \( x_1 \) and \( x_2 \) are also arbitrary, we may determine up to 4 unknown constants. Therefore, let us consider the case that (4.4.5.6) is exact for
\( f = 1, x, x^2 \) and \( x^3 \). This gives the following four equations.

\[
\begin{align*}
c_1 + c_2 &= \int_{-1}^{1} dx = 2 \\
c_1 x_1 + c_2 x_2 &= \int_{-1}^{1} x dx = 0 \\
c_1 x_1^2 + c_2 x_2^2 &= \int_{-1}^{1} x^2 dx = \frac{2}{3} \\
c_1 x_1^3 + c_2 x_2^3 &= \int_{-1}^{1} x^3 dx = 0.
\end{align*}
\]

Solving this \( 4 \times 4 \) non-linear system gives

\[
c_1 = c_2 = 1, \quad x_1 = -\frac{\sqrt{3}}{3}, \quad x_2 = \frac{\sqrt{3}}{3}.
\]

Therefore,

\[
\int_{-1}^{1} f(x) dx \approx f \left( -\frac{\sqrt{3}}{3} \right) + f \left( \frac{\sqrt{3}}{3} \right).
\]

This is the 2-point Gauss quadrature rule.

**Definition 4.2 (Legendre polynomials)** Polynomials \( \{P_0(x), P_1(x), \ldots, P_n(x), \ldots\} \) are said to be Legendre polynomials if

1. for each \( n \), \( P_n \) is a polynomial of order \( n \), and
2. \( \int_{-1}^{1} P(x) P_n(x) = 0 \) whenever \( P(x) \) is a polynomial of degree \( < n \).

The first few Legendre polynomials:

\[
\begin{align*}
P_0(x) &= 1, \quad P_1(x) = x, \quad P_2(x) = x^2 - \frac{1}{3}, \\
P_3(x) &= x^3 - \frac{3}{5} x, \quad P_4(x) = x^4 - \frac{6}{7} x^2 + \frac{3}{35}.
\end{align*}
\]

**Theorem 4.3** Suppose that \( x_1, x_2, \ldots, x_n \) are the roots of the \( n \)th Legendre polynomial \( P_n(x) \) and let

\[
c_i = \int_{-1}^{1} \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j} dx = \int_{-1}^{1} L_{n-1,i}(x) dx,
\]

where \( L_{n-1,i} \) is the Lagrange polynomial constructed using \( x_i, i = 1, 2, \ldots, n \). If \( P(x) \) is any polynomial of degree \( < 2n \), then

\[
\int_{-1}^{1} P(x) dx = \sum_{i=1}^{n} c_i P(x_i).
\]
PROOF. If the degree of $P(x)$ is less than $n$, we have

$$P(x) = \sum_{i=1}^{n} L_{n-1,i}(x)P(x_i).$$

So,

$$\int_{-1}^{1} P(x)dx = \int_{-1}^{1} \sum_{i=1}^{n} L_{n-1,i}P(x_i)$$

$$= \sum_{i=1}^{n} P(x_i) \int_{-1}^{1} L_{n-1,i}(x)dx$$

$$= \sum_{i=1}^{n} c_i P(x_i).$$

When the degree of $P(x)$ is less than $2n$, we let

$$P(x) = Q(x)P_n(x) + R(x),$$

where $Q$ and $R$ are polynomials whose degrees $< n$. From the definition of Lagendre polynomials we have

$$\int_{-1}^{1} Q(x)P_n(x)dx = 0.$$

Also,

$$P(x_i) = Q(x_i)P_n(x_i) + R(x_i) = R(x_i)$$

since $P(x_i) = 0$ for all $i = 1, 2, ..., n$. So,

$$\int_{-1}^{1} P(x)dx = \int_{-1}^{1} [Q(x)P_n(x) + R(x)]dx$$

$$= \int_{-1}^{1} R(x)dx = \sum_{i=1}^{n} c_i R(x_i) = \sum_{i=1}^{n} c_i P(x_i).$$

The points and weights of the 2 and 3-point Gauss quadratures are

<table>
<thead>
<tr>
<th>$n$</th>
<th>$x_{n,i}$</th>
<th>$c_{n,i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5773502692</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>-0.5773502692</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0.7745966692</td>
<td>0.5555555556</td>
</tr>
<tr>
<td></td>
<td>0.0000000000</td>
<td>0.9999999999</td>
</tr>
<tr>
<td></td>
<td>-0.7745966692</td>
<td>0.5555555556</td>
</tr>
</tbody>
</table>

Numerical differentiation

From Calculus we have

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h).$$
Let us consider a linear combination of \( f(x-h), f(x), \) and \( f(x+h) \) of the form
\[
c_1 f(x+h) + c_2 f(x) + c_3 f(x-h),
\]
assuming \( h \) is small. Taylor’s expansions for these are, respectively,
\[
f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(x) + \cdots
\]
\[
f(x) = f(x)
\]
\[
f(x-h) = f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(x) + \cdots
\]

### 4.6 First derivatives

**Forward difference:** \( c_1 = 1, c_2 = -1, c_3 = 0. \)
\[
f(x+h) - f(x) = hf'(x) + \frac{h^2}{2} f''(x) + O(h^3).
\]
From this we have
\[
f'(x) = \frac{f(x+h) - f(x)}{h} + O(h).
\]

**Central difference:** \( c_1 = 1, c_2 = 0, c_3 = -1. \)
\[
f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3} f''(x) + O(h^5),
\]
or
\[
f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).
\]

**Backward difference:** \( c_1 = 0, c_2 = 1, c_3 = -1. \)
\[
f(x) - f(x-h) = hf'(x) + O(h^2),
\]
or
\[
f'(x) = \frac{f(x) - f(x-h)}{h} + O(h)
\]
Clearly, the accuracy of the central difference is one order higher than those of the other two.

### 4.7 Second derivatives

Using the weights \( c_1 = 1, c_2 = -2 \) and \( c_3 = 1, \) we have
\[
f(x+h) - 2f(x) + f(x-h) = h^2 f''(x) + O(h^4).
\]
From this we have
\[
f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2).
\]
This is the central difference for \( f''(x). \)
4.8 Computational errors

Let $D_h(f) = \frac{f(x + h) - f(x)}{h}$ and $D(f) = D_h(f) + O(h)$. Then,

$$|D(f) - D_h(f)| \leq O(h) = \frac{M}{2} h,$$  \hfill (4.4.8.7)  

where $M = \max_{a \leq x \leq b} |f''(x)|$. In practice, some computational errors are involved when evaluating $f$. We assume

$$\tilde{f}(t) = f(t) + e(t)$$

with $|e(t)| \leq \varepsilon$ for a given machine error $\varepsilon$. So, the bound for the computational error is

$$|D_h(\tilde{f}) - D_h(f)| \leq \left| \frac{e(t + h) - e(t)}{h} \right| \leq \frac{2\varepsilon}{h}.$$  

Combining this with the discretisation error (4.4.8.7) we have

$$|D_h(\tilde{f}) - D(f)| \leq \frac{2\varepsilon}{h} + \frac{M}{2} h.$$  

When $h$ is small, $2\varepsilon/h$ may be dominant. The optimal choice of $h$ is such that

$$\frac{2\varepsilon}{h} = \frac{M}{2} h,$$

This gives

$$h^2 = \frac{4\varepsilon}{M} \Rightarrow h = 2\sqrt{\frac{\varepsilon}{M}}.$$
Chapter 5

Numerical Solution of Ordinary Differential Equations (ODEs)

5.1 Euler’s method

Consider the following initial value problem (IVP)

\[
\frac{dy}{dt} = f(t,y), \quad a < t \leq b, \quad (5.5.1.1)
\]
\[
y(a) = \alpha, \quad (5.5.1.2)
\]

where \(a, b \in \mathbb{R}\) with \(a < b\). Let \([a,b]\) be divided into \(N\) equally spaced subintervals with break points

\[t_i = a + ih, \quad i = 0, 1, ..., N\]

and \(h = \frac{b - a}{N}\). For each \(i\), using Taylor’s expansion we have

\[y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi_i)\]

for any \(\xi_i \in (t_i, t_{i+1})\). Since \(y\) satisfies (5.5.1.1), we have

\[y(t_{i+1}) = y(t_i) + hf(t_i, y_i) + \frac{h^2}{2}y''(\xi_i).\]

Omitting the 2nd order term and using the initial condition (5.5.1.2) we get the following Euler’s algorithm for the approximation of (5.5.1.1) and (5.5.1.2)

\[w_{i+1} = w_i + hf(t_i, w_i), \quad i = 0, 1, ..., N - 1, \quad (5.5.1.3)\]
\[w_0 = \alpha. \quad (5.5.1.4)\]

A geometric explanation is given in the figure below.
Error bounds for Euler’s method

Lemma 5.1 For all $x \geq -1$ and any positive $m$, we have

$$0 \leq (1 + x)^m \leq e^{mx}.$$ 

**PROOF.** From Taylor’s theorem we have

$$e^x = 1 + x + \frac{x^2}{2} e^\xi$$

for a $\xi$ between 0 and $x$. Since the last term in the above is non-negative, we have

$$e^x \geq 1 + x \geq 0$$

if $x \geq -1$. From this we have the result. $\square$

Lemma 5.2 If $s$ and $t$ are positive real numbers, $\{a_i\}_0^k$ is a sequence satisfying $a_0 \geq -t/s$ and

$$a_{i+1} \leq (1 + s)a_i + t, \quad i = 0, 1, ..., k,$$

then,

$$a_{i+1} \leq e^{(i+1)s} \left( a_0 + \frac{t}{s} \right) - \frac{t}{s}.$$ 

The proof of this, which uses Lemma 5.1, is omitted here. For the error in the approximation of (5.5.1.1) by (5.5.1.3) we have the following theorem.

**Theorem 5.1** Suppose $f$ is continuous and satisfies the Lipschitz condition on $D = \{(t, y) : a \leq x \leq b, -\infty < y < \infty\}$, i.e.,

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|. \tag{5.5.1.5}$$

If there exists $M > 0$ such that

$$|y''(x)| \leq M, \quad \forall t \in [a, b], \tag{5.5.1.6}$$

then, we have

$$|y(t_i) - w_i| \leq \frac{hM}{2L} \left( e^{L(t_i-a)} - 1 \right),$$

where $\{w_i\}$ is the sequence from Euler’s method.
PROOF. By Taylor’s expansion we have

\[ y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2} y''(\xi_i). \]

From this and (5.5.1.3) we have

\[ |y_{i+1} - w_{i+1}| = \left| y_i - w_i + h[f(t_i, y_i) - f(t_i, w_i)] + \frac{h^2}{2} y''(\xi_i) \right| \]
\[ \leq |y_i - w_i| + h|f(t_i, y_i) - f(t_i, w_i)| + \frac{h^2}{2} |y''(\xi_i)| \]
\[ \leq (1 + hL) |y_i - w_i| + \frac{h^2 M}{2}. \]

In the above we have used (5.5.1.5) and (5.5.1.6). Let \( e_i = y_i - w_i \). The above becomes

\[ |e_{i+1}| \leq (1 + hL) |e_i| + \frac{h^2 M}{2}. \]

Using (5.2) we have

\[ |e_{i+1}| \leq e^{(i+1)hL} \left( |e_0| + \frac{h^2 M}{2hL} \right) - \frac{h^2 M}{2hL}. \]

But \( e_0 = w_0 - \alpha = 0 \) and \((i + 1)h = t_{i+1} - a\). Therefore, we get

\[ |e_{i+1}| \leq \frac{hM}{2L} \left( e^{(t_{i+1} - a)L} - 1 \right). \]

This completes the proof. \( \square \)

Example. Consider \( y' = y - t^2 + 1 \), \( 0 \leq t \leq 2 \) with the initial condition \( y(0) = 0.5 \).

For this example, the exact solution is

\[ y(t) = (1 + t)^2 - \frac{1}{2}t^2. \]

So,

\[ |y''| \leq 0.5e^2 - 2 =: M. \]

Also, \( |f(t, y_1) - f(t, y_2)| = |y_1 - y_2| \), implying \( L = 1 \). Thus,

\[ |e_i| \leq \frac{h(0.5e^2 - 2)}{2} \left( e^{(t_i - 0) - 1} - 1 \right) \]
\[ = \frac{h(0.5e^2 - 2)}{2} \left( e^{t_i} - 1 \right) \]

\section{Higher-order Taylor methods}

\textbf{Definition 5.1} The difference method

\[ w_0 = \alpha \]
\[ w_{i+1} = w_i + h\phi(t_i, w_i), \quad i = 0, 1, ..., N - 1 \]
has a local truncation error
\[ \tau_{i+1}(h) = \frac{y_{i+1} - (y_i + h\phi(t_i, y_i))}{h} = \frac{y_{i+1} - y_i}{h} - \phi(t_i, y_i) \]
for each \( i = 0, 1, ..., N - 1 \).

For Euler's method, \( \phi(t_i, y_i) = f(t_i, y_i) \), and so
\[
\tau_{i+1}(h) = \frac{y_{i+1} - y_i}{h} - f(t_i, y_i) = y'(t_i) - f(t_i, y_i) + \frac{h}{2} y''(\xi_i)
\]
\[
= \frac{h}{2} y''(\xi_i).
\]

In the case that \( y'' \) satisfies (5.5.1.6), we have
\[
|\tau_{i+1}(h)| \leq \frac{h}{2} M.
\]

Suppose (5.5.1.1) and (5.5.1.2) has a solution and \( y(t) \) has \( n + 1 \) continuous derivatives for a positive integer \( n \). At \( t = t_i \), Taylor's expansion gives
\[
y_{i+1} = y_i + hy'_i + \frac{h^2}{2} y''_i + \cdots + \frac{h^n}{n!} y^{(n)}_i + \frac{h^{n+1}}{(n+1)!} y^{(n+1)}(\xi_i)
\]
for a \( \xi_i \in (a, b) \). Differentiating (5.5.1.1) \( k - 1 \) times gives
\[
y^{(k)}(t) = f^{(k-1)}(t, y(t)), \quad k = 1, 2, ..., n.
\]

Therefore,
\[
y_{i+1} = y_i + h f(t_i, y_i) + \frac{h^2}{2} f'(t_i, y_i) + \cdots + \frac{h^n}{n!} f^{(n-1)}(t_i, y_i) + \frac{h^{n+1}}{(n+1)!} f^{(n)}(\xi_i, y(\xi_i))
\]
\[
= y_i + h f(t_i, y_i) + \frac{h}{2} f'(t_i, y_i) + \cdots + \frac{h^{n-1}}{n!} f^{(n-1)}(t_i, y_i) + \frac{h^{n+1}}{(n+1)!} f^{(n)}(\xi_i, y(\xi_i))
\]
\[
=: y_i + h T^{(n)}(t_i, y_i) + \frac{h^{n+1}}{(n+1)!} f^{(n)}(\xi_i, y(\xi_i)). \quad (5.5.2.7)
\]

Therefore, we have the Taylor's method of order \( n \) as follows
\[
w_0 = \alpha
\]
\[
w_{i+1} = w_i + h T^{(n)}(t_i, w_i), \quad i = 0, 1, ..., N - 1
\]
From (5.5.2.7) we have that the local truncation error bound is
\[
\tau_{i+1} = \frac{y_{i+1} - y_i}{h} - T^{(n)}(t_i, y_i)
\]
\[
= \frac{y_{i+1} - y_i}{h} - \frac{h^n}{(n+1)!} f^{(n)}(\xi_i, y(\xi_i)) = O(h^n).
\]
Example. Construct the Taylor’s methods of orders 2 and 4 for the following IVP
\[
y' = y - t^2 + 1, \quad 0 \leq t \leq 2
\]
y(0) = 0.5

Differentiating \( f \) repeatedly and using \( y' = y - t^2 + 1 \) we have
\[
f'(t, y) = y' - 2t = y - t^2 + 1 - 2t
\]
\[
f''(t, y) = y' - 2t - 2 = y - t^2 + 1 - 2t - 2
\]
\[
f'''(t, y) = y' - 2t - 2 = y - t^2 + 1 - 2t - 2
\]
\[
= y - t^2 - 2t - 1.
\]

Therefore,
\[
T^{(2)} = f(t_i, w_i) + \frac{h}{2} f'(t_i, w_i)
\]
\[
= w_i - t_i^2 + 1 + \frac{h}{2} [w_i - t_i^2 + 1 - 2t_i]
\]
\[
= (1 + \frac{h}{2})(w_i - t_i^2 + 1) - ht_i.
\]

The 2nd order Taylor’s method is
\[
w_0 = 0.5
\]
\[
w_{i+1} = w_i + h \left[ (1 + \frac{h}{2}) (w_i - t_i^2 + 1) - ht_i \right]
\]

Similarly, we have
\[
T^{(4)} = \left( 1 + \frac{h}{2} + \frac{h^2}{6} + \frac{h^3}{24} \right) (w_i - t_i^2) - \left( 1 + \frac{h}{3} \frac{h^2}{12} \right) + 1 + \frac{h}{2} - \frac{h^2}{6} - \frac{h^3}{24}.
\]

Therefore, the 4th order method is
\[
w_0 = 0.5
\]
\[
w_{i+1} = w_i + h T^{(4)}(t_i, w_i), \quad i = 0, 1, ..., N - 1.
\]

### 5.3 Runge-Kutta and Other Methods

Consider
\[
w_0 = 0.5
\]
\[
w_{i+1} = w_i + h T^{(2)}(t_i, w_i), \quad i = 0, 1, ..., N - 1,
\]

where
\[
T^{(2)}(t, y) = f(t, y) + \frac{h}{2} f'(t, y).
\]
This involves the 1st derivative of $f$. Using the chain rule we have

$$f'(t, y) = f_t(t, y) + f_y(t, y)y' = f_t(t, y) + f_y(t, y) \cdot f(t, y).$$

Substituting into $T^{(2)}$ gives

$$T^{(2)}(t, y) = f(t, y) + \frac{h}{2}f_t(t, y) + \frac{h}{2}f_y(t, y) \cdot f(t, y).$$

We now consider the approximation of $f_t$ and $f_y$ by linear combinations of $f$ at some points. Using Taylor’s expansion for two variables we have

$$f(t + \alpha_1, y + \beta_1) = f(t, y) + \alpha_1 f_t(t, y) + \beta_1 f_y(t, y) + O(\alpha_1^2 + \alpha_1 \beta_1 + \beta_1^2).$$

Matching the coefficients of the first order terms in $T^{(2)}$ and $f(t + \alpha_1, y + \beta_1)$ gives

$$\alpha_1 = \frac{h}{2} \quad \text{and} \quad \beta_1 = \frac{h}{2} f(t, y).$$

Therefore,

$$T^{(2)} = f(t, \frac{h}{2}, y + \frac{h}{2}f(t, y)) + O(h^2 + h^2 f^2).$$

We thus define the following 2nd order Runge-Kutta scheme

$$w_0 = \alpha$$

$$w_{i+1} = w_i + hf_t(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)), \quad i = 0, 1, ..., N - 1,$$

Obviously, from the above analysis we see that

$$\tau_{i+1}(h) = O(h^2).$$

But this method does not need $f'$. Consider the mean value theorem

$$\frac{y_{i+1} - y_i}{h} = y'(\xi) = f(\xi, y(\xi))$$

for a $\xi \in (t_i, t_{i+1})$. If we choose $\xi = t_i + \frac{h}{2}$, we have

$$\frac{y_{i+1} - y_i}{h} \approx f(t_i + \frac{h}{2}, y(t_i + \frac{h}{2})) = f(t_i + \frac{h}{2}, y(t_i) + y'(t_i) \frac{h}{2} + O(h^2)).$$

Therefore, we define the following \textit{Mid-point Method}

$$w_0 = \alpha$$

$$w_{i+1} = w_i + hf(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)), \quad i = 0, 1, ..., N - 1.$$
Let us approximate $y'(\xi)$ in (5.5.3.8) by
\[
y'(\xi) \approx \frac{y'(t_i) + y'(t_{i+1})}{2} = \frac{f(t_i, y_i) + f(t_{i+1}, y_{i+1})}{2}.
\]
But the resulting scheme is implicit:
\[
w_{i+1} = w_i + \frac{h}{2} [f(t_i, w_i) + f(t_{i+1}, w_{i+1})].
\]
To overcome this difficulty, we use the standard Euler’s method to approximate $y_{i+1}$ first and then plug it into the right-hand side of the above. This is the modified Euler’s method.
\[
w_0 = \alpha, \quad w_{i+1} = w_i + \frac{h}{2} [f(t_i, w_i) + f(t_{i+1}, w_{i+1})], \quad i = 0, 1, \ldots, N - 1.
\]
The local truncation error is $\tau_i(h) = O(h^2)$.

**Runge-Kutta Order 4 Method**

\[
w_0 = \alpha, \quad k_1 = hf(t_i, w_i), \quad k_2 = hf(t_i + \frac{h}{2}, w_i + \frac{1}{2} k_1), \quad k_3 = hf(t_i + \frac{h}{2}, w_i + \frac{1}{2} k_2), \quad k_4 = hf(t_{i+1}, w_i + k_3),
\]
\[
w_{i+1} = w_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4), \quad i = 0, 1, \ldots, N - 1.
\]
The local truncation error is $O(h^4)$.

### 5.4 Multi-step Methods

**Definition 5.2** An $m$-step multi-step method for solving $y' = f(t, y), a \leq t \leq b$, $y(a) = \alpha$ has a difference equation for finding the approximation $w_{i+1}$ at the mesh point $t_{i+1}$ represented by the following equation, where $m$ is an integer $> 1$:
\[
w_{i+1} = a_{m-1} w_i + a_{m-2} w_{i-1} + \cdots + a_0 w_{i+1-m} + h [b_m f(t_{i+1}, w_{i+1}) + b_{m-1} f(t_i, w_i) + \cdots + b_0 f(t_{i+1-m}, w_{i+1-m})]
\]
for $i = m - 1, m, \ldots, N - 1$, where $h = (b - a) / N$, $a_i, b_i, i = 0, 1, \ldots, m - 1$ are constants, and the starting values
\[
w_0 = \alpha, w_1 = \alpha_1, \ldots, w_{m-1} = \alpha_{m-1}
\]
are specified.
Integrating $y' = f$ from $t_i$ to $t_{i+1}$ gives

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(t, y)dt,$$

or

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y)dt.$$

Let $P(t)$ be an interpolating polynomial obtained using some of the previous data $(t_0, w_0), (t_1, w_1), ..., (t_i, w_i)$. We approximate $y_{i+1}$ by

$$y_{i+1} \approx y_i + \int_{t_i}^{t_{i+1}} P(t)dt.$$

For example, we let $P_{m-1}(t)$ denote the polynomial from the Newton’s backward difference formula on $(t_i, f(t_i, y_i)), ..., (t_{i+1-m}, f(t_{i+1-m}, y_{i+1-m}))$. Then, we have

$$f(t, y(t)) = P_{m-1}(t) + R_m(t),$$

where

$$R_m(t) = \frac{f^{(m)}(\xi, y(\xi))}{m!} (t - t_i) \cdots (t - t_{i+1-m}).$$

Introducing $t = t_i + sh$ for $s \in [0, 1]$, we have $dt = hds$, and so

$$\int_{t_i}^{t_{i+1}} f(t, y(t))dt = \int_{t_i}^{t_{i+1}} \sum_{k=0}^{m-1} (-1)^k \binom{s}{k} \nabla^k f(t_i, y(t_i)) dt + \int_{t_i}^{t_{i+1}} R_m(t)dt$$

$$= h \sum_{k=0}^{m-1} (-1)^k \nabla^k f(t_i, y(t_i)) \int_0^1 \binom{s}{k} ds + \int_{t_i}^{t_{i+1}} R_m(t)dt.$$

When $k = 3$, we have

$$(-1)^3 \int_0^1 \binom{s}{3} ds = - \int_0^1 \frac{(-s)(-s-1)(-s-2)}{3!} ds = \frac{3}{8}.$$

Other values of the integral for different $k$’s are given in the following table

<table>
<thead>
<tr>
<th>$k$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>integral</td>
<td>1</td>
<td>1/2</td>
<td>5/12</td>
<td>3/8</td>
<td>251/720</td>
<td>95/288</td>
</tr>
</tbody>
</table>

Therefore,

$$\int_{t_i}^{t_{i+1}} f(t, y)dt = h \left[ f(t_i, y_i) + \frac{1}{2} \nabla f(t_i, y_i) + \frac{5}{12} \nabla^2 f(t_i, y_i) + \cdots + \int_{t_i}^{t_{i+1}} R_m(s)ds \right].$$

Adam-Bashforth Method ($m = 3$)
When $m = 3$ we have
\[ y_{i+1} = y_i + h \left[ 1 + \frac{1}{2} \nabla + \frac{5}{12} \nabla^2 \right] f(t_i, y_i) \]
\[ = y_i + h \left\{ f(t_i, y_i) + \frac{1}{2} [f(t_i, y_i) - f(t_{i-1}, y_{i-1})] \right. \]
\[ + \frac{5}{12} [f(t_i, y_i) - 2f(t_{i-1}, y_{i-1}) + f(t_{i-2}, y_{i-2})] \} \]
\[ = y_i + \frac{h}{12} [23f(t_i, y_i) - 16f(t_{i-1}, y_{i-1}) + 5f(t_{i-2}, y_{i-2})] \].

From this we define the following algorithm.
\[
\begin{align*}
    w_0 &= \alpha, & w_1 &= \alpha_1, & w_2 &= \alpha_2, \\
    w_{i+1} &= w_i + \frac{h}{12} [23f(t_i, w_i) - 16f(t_{i-1}, w_{i-1}) + 5f(t_{i-2}, w_{i-2})], \quad i = 2, 3, \ldots, N - 1.
\end{align*}
\]

The local truncation error is $O(h^3)$.

**Adam-Bashforth 2-step Method** ($m = 2$)
\[
\begin{align*}
    w_0 &= \alpha, & w_1 &= \alpha_1, \\
    w_{i+1} &= w_i + \frac{h}{12} [3f(t_i, w_i) - f(t_{i-1}, w_{i-1})],
\end{align*}
\]

$i = 1, 2, \ldots, N - 1$. The local truncation error is $O(h^2)$.

### 5.5 Implicit Methods

**Adams-Moulton 2-step Method**
\[
\begin{align*}
    w_0 &= \alpha, & w_1 &= \alpha_1, \\
    w_{i+1} &= w_i + \frac{h}{12} [5f(t_{i+1}, w_{i+1}) + 8f(t_i, w_i) - f(t_{i-1}, w_{i-1})],
\end{align*}
\]

$i = 1, 2, \ldots, N - 1$.

**Adams-Moulton 3-step Method**
\[
\begin{align*}
    w_0 &= \alpha, & w_1 &= \alpha_1, & w_2 &= \alpha_2, \\
    w_{i+1} &= w_i + \frac{h}{24} [9f(t_{i+1}, w_{i+1}) + 19f(t_i, w_i) - 5f(t_{i-1}, w_{i-1}) + f(t_{i-2}, y_{i-2})],
\end{align*}
\]

$i = 2, 3, \ldots, N - 1$.

**Predictor-Corrector Methods**

Idea: Use an explicit method to obtain an intermediate approximation and then plug it into an implicit method to get an improved approximation. For example, the modified Euler’s method discussed before.
Combining Adams-Bashforth and Adams-Moulton 2-step methods we have

\[
\begin{align*}
    w_0 &= \alpha, \\
    w_1 &= \alpha_1, \\
    w_{i+1}^* &= w_i + \frac{h}{2} [3f(t_i, w_i) - f(t_{i-1}, w_{i-1})], \\
    w_{i+1} &= w_i + \frac{h}{12} [5f(t_{i+1}, w_{i+1}^*) + 8f(t_i, w_i) - f(t_{i-1}, w_{i-1})],
\end{align*}
\]

\(i = 1, 2, \ldots, N - 1\).

### 5.6 Stability of One-Step Methods

Consider a scheme of the form

\[
\begin{align*}
    w_0 &= \alpha, \\
    w_{i+1} &= w_i + h\phi(t_i, w_i, h), \quad i = 0, 1, \ldots, N - 1.
\end{align*}
\]

The local truncation error is denoted as \(\tau_i(h)\). We now define the consistency, convergence and stability of this scheme.

**Definition 5.3 (consistency)** The scheme is consistent if

\[
\lim_{h \to 0} \max_{1 \leq i \leq N} |\tau_i(h)| = 0.
\]

**Definition 5.4 (convergence)** The scheme is consistent if

\[
\lim_{h \to 0} \max_{1 \leq i \leq N} |w_i - y(t_i)| = 0.
\]

**Definition 5.5 (stability)** The scheme is stable if a small change/perturbation in \(\alpha\) results in only a small change in \(\{w_i\}\). In this case, we also say that the method depends continuously on the initial data.

**Theorem 5.2** Consider the above 1-step scheme. Suppose that \(\exists h_0 > 0\) such that \(\phi(t, w, h)\) is continuous and satisfies a Lipschitz condition in \(w\) with Lipschitz constant \(L\) on

\[
D = \{(t, w, h) : a \leq t \leq b, -\infty < w < \infty, 0 \leq h \leq h_0\}.
\]

Then

1. The method is stable.
2. The method is convergent iff it is consistent which is equivalent to

\[
\phi(t, y, 0) = f(t, y), \quad \forall t \in [a, b].
\]
3. If there exists a function \(\tau\) such that

\[
|\tau_i(h)| \leq \tau(h) \quad \forall i = 1, 2, \ldots, N \quad \text{and} \quad 0 \leq h \leq h_0,
\]

then

\[
|y(t_i) - w_i| \leq \frac{\tau(h)}{L} e^{L(t_i - a)}.
\]
The proof is omitted here.

Example. Consider the following Modified Euler’s Method when \( f \) satisfies the Lipschitz condition:

\[
\begin{align*}
    w_0 &= \alpha \\
    w_{i+1} &= w_i + \frac{h}{2} [f(t_i, w_i) + f(t_{i+1}, w_i + hf(t_i, w_i))], \quad i = 0, 1, ..., N - 1.
\end{align*}
\]

In this case

\[
\phi(t, w, h) = \frac{1}{2} f(t, w) + \frac{1}{2} f(t + h, w + hf(t, w)).
\]

So,

\[
\phi(t, w_1, h) - \phi(t, w_2, h) = \frac{1}{2} [f(t, w_1) - f(t, w_2)] + \frac{1}{2} [f(t + h, w_1 + hf(t, w_1)) - f(t + h, w_2 + hf(t, w_2))].
\]

Since \( f \) satisfies the Lipschitz condition, i.e.,

\[
|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|.
\]

we have

\[
|\phi(t, w_1, h) - \phi(t, w_2, h)| \leq \frac{L}{2}|w_1 - w_2| + \frac{L}{2}|w_1 + hf(t, w_1) - (w_2 + hf(t, w_2))|
\]

\[
\leq L|w_1 - w_2| + \frac{Lh}{2}|f(t, w_1) - f(t, w_2)|
\]

\[
\leq L|w_1 - w_2| + \frac{L^2h}{2}|w_1 - w_2|
\]

\[
= L(1 + \frac{Lh}{2})|w_1 - w_2|
\]

\[
= L'|w_1 - w_2|.
\]

Therefore, \( \phi \) also satisfies a Lipschitz condition. Clearly, \( \phi \) is continuous. Now,

\[
\phi(t, w, 0) = \frac{1}{2} f(t, w) + \frac{1}{2} f(t, w) = f(t, w).
\]

So, the method is stable and convergent. We can also show that

\[
|y(t_i) - w_i| \leq O(h^2) e^{L'(t_i-a)} \frac{e^{L(t_i-a)}}{L'}.
\]

### 5.7 Stability of Multi-step Methods

Consider

\[
\begin{align*}
    w_i &= \alpha_i, \quad i = 0, 1, ..., m - 1, \alpha_0 = \alpha \\
    w_{i+1} &= \alpha_{i+1} w_i + \alpha_{i+2} w_{i+1} + \cdots + \alpha_{i+m-1} w_{i+m-1} + \frac{h}{2} [f(t_{i+1}, w_{i+1}) + f(t_i, w_i) + \cdots + f(t_{i+1-m}, w_{i+1-m})]
\end{align*}
\]
for \( i = m - 1, m, \ldots, N - 1 \). The \textit{Characteristic Polynomial} associated with this scheme is defined as
\[
P(\lambda) = \lambda^m - a_{m-1}\lambda^{m-1} - a_{m-2}\lambda^{m-2} - \cdots - a_1\lambda - a_0.
\]
In the case that \( f(t, y) = 0 \), we have
\[
w_{i+1} - (a_{m-1}w_i + \cdots + a_0w_{i+1-m}) = 0.
\]
This is a difference equation. If \( \lambda_k \) satisfies \( P(\lambda_k) = 0 \), then \( w_n = \lambda_k^n \) for all \( k \), because
\[
\lambda_k^{i+1} - (a_{m-1}\lambda_k^i + \cdots + a_0\lambda_k^{i+1-m}) = \lambda_k^{i+1-m}(\lambda_k^m - (a_{m-1}\lambda_k^{m-1} + \cdots + a_0) = 0.
\]
So,
\[
w_n = \sum_{k=1}^{m} c_k\lambda_k^n
\]
for a set of constants \( \{c_k\} \). Using \( w_0 = \alpha \) we have
\[
w_n = \alpha + \sum_{k=2}^{m} c_k\lambda_k^n
\]
If everything is exact. \( c_k = 0 \) for all \( k = 2, 3, \ldots, m \). But, in practice, \( c_k \neq 0 \) because of the machine error. So, we need
\[
|\lambda_k| \leq 1
\]
in order that the calculation error does not grow.

\textbf{Definition 5.6 (root condition)} Let \( \lambda_i, i = 1, 2, \ldots, m \) be the roots of \( P(\lambda) = 0 \) associated with the multi-step scheme. If \( |\lambda_i| \leq 1 \) for each \( i = 1, 2, \ldots, m \) and all the roots with absolute value 1 are simple roots, then the multi-step scheme is said to satisfy the root condition.

\textbf{Theorem 5.3} The multi-step method is stable iff it satisfies the root condition. Moreover, if it is consistent with the difference equation, then the multi-step scheme is stable iff it is convergent.

The proof is omitted here.

Example 1. The 4th order Adams-Bashforth method
\[
w_{i+1} = w_i + hF(t_i, h, w_{i+1}, w_i, \ldots, w_{i-3}),
\]
where
\[
F = \frac{h}{24}[55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3})].
\]
In this case we have \( m = 4, a_0 = 0 = a_1 = a_2, a_3 = 1 \). So,
\[
P(\lambda) = \lambda^4 - \lambda^3 = \lambda^3(\lambda - 1) = 0.
\]
This gives \( \lambda = 0, 0, 0, 1 \), and so it is stable.
Example 2. The explicit multi-step method with

\[ w_{i+1} = w_{i-3} + \frac{4h}{3} [2f(t_i, w_i) - f(t_i, w_{i-1}) + 2f(t_{i-2}, w_{i-2})]. \]

In this case, \( m = 4, a_0 = 1, a_1 = a_2 = a_3 = 0. \) So,

\[ P(\lambda) = \lambda^4 - 1 = 0. \]

The roots are \( \lambda = \pm 1, \pm \pi i. \) The method is also stable.

**Definition 5.7** A multi-step method is

1. strongly stable if the roots condition is satisfied and \( \lambda = 1 \) is the only root satisfying \( |\lambda| = 1, \)
2. weakly stable if it is stable, but not strongly stable, and
3. unstable if it does not satisfy the root condition.

Example 3. Consider the method with

\[ w_{i+1} = \frac{1}{2}(w_i + w_{i-1}) + hF(t_i, h, w_i, w_{i-1}) \]

We have \( m = 2, a_0 = 1/2 = a_1. \) So,

\[ P(\lambda) = \lambda^2 - \frac{1}{2} \lambda - \frac{1}{2} = 0, \]

or

\[ 2\lambda^2 - \lambda - 1 = 0. \]

Solving this gives

\[ \lambda = \frac{1 \pm \sqrt{5}}{4} = 1 \text{ or } -\frac{1}{2}. \]

The method is strongly stable.
Chapter 6

Least Squares Approximation

6.1 Discrete case

The approximation of a given data set, \((x_i, y_i), i = 1, 2, ..., m\) by the Lagrange polynomial often yields large errors at points other than the data points \(x_i, i = 1, 2, ..., m\). This is because of the use of high order polynomials. Also, the 'curve' obtained by Lagrange type of interpolation does not agree with the practical situation. For example, the data set may show a linear, quadratic or exponential trend, but the Lagrange interpolant gives a curve with a complicated convexity property. Therefore, we need to find a better/alternative approximation method then/to the Lagrange interpolation method. A common practice is to minimize the 'mean square error' in the approximation as demonstrated below.

Let us consider the approximation of \((x_i, y_i), i = 1, 2, ..., m\) by a polynomial

\[
P_n(x) = \sum_{j=0}^{n} a_j x^j,
\]

(6.6.1.1)

where \(m\) and \(n\) are positive integers. Normally we assume that \(n << m\). Thus, we look for \(\{a_j\}_0^n\) such that

\[
E_n = \sum_{i=1}^{m} (y_i - P_n(x_i))^2
\]

(6.6.1.2)

is minimized. Substituting (6.6.1.1) into (6.6.1.2) gives

\[
E_n = \sum_{i=1}^{m} \left( y_i - \sum_{j=0}^{n} a_j x_i^j \right)^2
\]

\[
= \sum_{i=1}^{m} \left[ y_i^2 - 2 y_i \sum_{j=0}^{n} a_j x_i^j + \left( \sum_{j=0}^{n} a_j x_i^j \right)^2 \right]
\]

\[
= \sum_{i=1}^{m} y_i^2 - 2 \sum_{j=0}^{n} a_j \left( \sum_{i=1}^{m} y_i x_i^j \right) + \sum_{j=0}^{n} \left( \sum_{i=1}^{m} a_j x_i^j \right)^2
\]

\[
= \sum_{i=1}^{m} y_i^2 - 2 \sum_{j=0}^{n} a_j \left( \sum_{i=1}^{m} y_i x_i^j \right) + \sum_{j,k=0}^{n} a_j a_k \left( \sum_{i=1}^{m} x_i^{j+k} \right).
\]
Differentiating $E_n$ with respect to $a_j$ and setting the derivative to zero we have

$$\frac{\partial E_n}{\partial a_j} = -2 \sum_{i=1}^{m} y_i x_i^j + 2 \sum_{k=0}^{n} a_k \left( \sum_{i=1}^{m} x_i^{j+k} \right) = 0,$$

or

$$\sum_{k=0}^{n} a_k \left( \sum_{i=1}^{m} x_i^{j+k} \right) = \sum_{i=1}^{m} y_i x_i^j$$

for $j = 0, 1, \ldots, n$. In the matrix form we have

$$\begin{pmatrix}
\sum_{i=1}^{m} x_i^0 & \sum_{i=1}^{m} x_i^1 & \cdots & \sum_{i=1}^{m} x_i^n \\
\sum_{i=1}^{m} x_i^1 & \sum_{i=1}^{m} x_i^2 & \cdots & \sum_{i=1}^{m} x_i^{n+1} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{i=1}^{m} x_i^n & \sum_{i=1}^{m} x_i^{n+1} & \cdots & \sum_{i=1}^{m} x_i^{2n}
\end{pmatrix}
\begin{pmatrix}
0 \\
a_1 \\
\vdots \\
a_n
\end{pmatrix}
= \begin{pmatrix}
\sum_{i=1}^{m} y_i x_i^0 \\
\sum_{i=1}^{m} y_i x_i^1 \\
\vdots \\
\sum_{i=1}^{m} y_i x_i^n
\end{pmatrix}.$$

When $n = 1$, i.e., $P_1(x) = a_0 + a_1 x$, we have

$$\begin{pmatrix}
\sum_{i=1}^{m} x_i^0 & \sum_{i=1}^{m} x_i^1 \\
\sum_{i=1}^{m} x_i^1 & \sum_{i=1}^{m} x_i^2
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1
\end{pmatrix}
= \begin{pmatrix}
\sum_{i=1}^{m} y_i x_i^0 \\
\sum_{i=1}^{m} y_i x_i^1
\end{pmatrix}.$$

Solving this $2 \times 2$ system exactly gives

$$a_0 = \frac{(\sum_{i=1}^{m} x_i^1)(\sum_{i=1}^{m} y_i) - (\sum_{i=1}^{m} y_i x_i)(\sum_{i=1}^{m} x_i)}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2},$$

$$a_1 = \frac{m(\sum_{i=1}^{m} y_i x_i) - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} y_i)}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2}.$$

Example. Find the linear fit to the data set given in the table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1.3</td>
<td>3.5</td>
<td>4.2</td>
<td>5.0</td>
<td>7.0</td>
<td>8.8</td>
<td>10.1</td>
<td>12.5</td>
<td>13</td>
<td>15.6</td>
</tr>
</tbody>
</table>

This can be solved by the following Matlab code:

```matlab
x=[1;2;3;4;5;6;7;8;9;10];
y=[1.3;3.5;4.2;5.0;7.0;8.8;10.1;12.5;13;15.6];
m=10;
X=sum(x);
Y=sum(y);
XX=dot(x,x);
XY=dot(x,y);
den=m*XX-X^2;
a0=(XX*Y-XY*X)/den;
a1=(m*XY-X*Y)/den;
```

Run this code will result in $a_0 = -0.36$ and $a_1 = 1.538$. 
6.2 Fitting by exponential functions

Consider fitting \((x_i, y_i), i = 1, 2, ..., m\) by \(y = f(x) = b \exp(ax)\), where \(a\) and \(b\) are to be determined. Taking \(\ln\) yields

\[
\ln y = \ln b + ax, \quad \text{or} \quad Y = B + ax.
\]

Therefore, the problem becomes how to fit the data set \((x_i, \ln y_i), i = 1, 2, ..., m\) by the above linear fit. The condition required is that \(y_i > 0\) for all \(i = 1, 2, ..., m\).

6.3 Orthogonal polynomials & least-squares approximation

Suppose \(f \in C[a, b]\). We look for \(P_n(x)\) of the form (6.6.1.1) so that

\[
E = \int_a^b [f(x) - P_n(x)]^2 dx
\]

is minimized. Substituting (6.6.1.1) into the above expression we have

\[
E = \left[ \int_a^b f^2(x) dx - 2 \int_a^b f(x) \sum_{k=0}^n a_k x^k dx + \int_a^b \left( \sum_{k=0}^n a_k x^k \right)^2 dx \right]
\]

\[
= \int_a^b f^2(x) dx - 2 \int_a^b f(x) \sum_{k=0}^n a_k x^k dx + \int_a^b \sum_{j,k=0}^n a_j a_k x^{j+k} dx.
\]

Differentiating with respect to \(a_j\) we have

\[
\frac{\partial E}{\partial a_j} = -2 \int_a^b x^j f(x) dx + 2 \sum_{k=0}^n a_k \int_a^b x^{j+k} dx = 0
\]

for all \(j = 0, 1, ..., n\). Therefore, we have following \((n+1) \times (n+1)\) linear system determining \(a_j, j = 0, 1, ..., n\).

\[
\sum_{k=0}^n a_k \int_a^b x^{j+k} dx = \int_a^b x^j f(x) dx
\]

for all \(j = 0, 1, ..., n\). In matrix form this is

\[
\begin{pmatrix}
\int_a^b dx & \int_a^b x dx & \cdots & \int_a^b x^n dx \\
\vdots & \vdots & \ddots & \vdots \\
\int_a^b x^n dx & \int_a^b x^{n+1} dx & \cdots & \int_a^b x^{2n} dx
\end{pmatrix}
\begin{pmatrix}
a_0 \\
\vdots \\
a_n
\end{pmatrix}
= \begin{pmatrix}
\int_a^b f dx \\
\vdots \\
\int_a^b x^n f dx
\end{pmatrix}
\]

The coefficient matrix is called a ‘Hilbert matrix’.
We may also use the weighted inner product to determine $P_n(x)$. More generally, we may find
\[ \phi(x) = \sum_{k=0}^{n} a_k \phi_k(x), \tag{6.6.3.3} \]
where $\{\phi_0(x), \phi_1(x), \ldots, \phi_n(x)\}$ is a set of linearly independent functions, so that the quadratic function
\[ E = \int_{a}^{b} w(x)[f(x) - \phi(x)]^2 dx \tag{6.6.3.4} \]
is minimized. In (6.6.3.4), $w(x) > 0$ is a weighting function. Substituting (6.6.3.3) into (6.6.3.4), differentiating $E$ with respect to $a_j$ and setting the result to zero, we have
\[ \sum_{k=0}^{n} a_k \int_{a}^{b} w(x) \phi_k(x) \phi_j(x) dx = \int_{a}^{b} w(x) f(x) \phi_j(x) dx \tag{6.6.3.5} \]
for $j = 0, 1, \ldots, n$. If we choose $\{\phi_k\}_{0}^{n}$ such that
\[ \int_{a}^{b} w(x) \phi_k(x) \phi_j(x) dx = \begin{cases} 0, & j \neq k, \\ \alpha_j > 0, & j = k, \end{cases} \tag{6.6.3.6} \]
then (6.6.3.5) has the solution
\[ a_j = \frac{1}{\alpha_j} \int_{a}^{b} w(x) f(x) \phi_j(x) dx \]
for $j = 0, 1, \ldots, n$. Functions satisfying (6.6.3.6) are called orthogonal functions.

Examples 1. The set $\{\sin(n\pi x), \cos(n\pi x)\}$ for $n = 0, 1, \ldots$ contains orthogonal functions on $(-\pi, \pi)$.

Example 2. The Legendre polynomials $\{P_n(x)\}$, where $P_0 = 1$, $P_1 = x$, $P_2 = x^2 - \frac{1}{3}$, $P_3 = x^3 - \frac{3}{2}x$, $P_4 = x^4 - \frac{6}{5}x^2 + \frac{3}{5}$, $P_5 = x^5 - \frac{10}{21}x^3 + \frac{5}{21}x$,... are a set of orthogonal polynomials on $[0, 1]$. 

63
Chapter 7

Solution of Nonlinear Systems of Equations

In this chapter we consider the solution of nonlinear algebraic systems of the form

\[ f(x) = 0, \quad (7.7.0.1) \]

where \( f = (f_1, f_2, ..., f_n)^T : \mathbb{R}^n \mapsto \mathbb{R}^n \) and \( x = (x_1, ..., x_n)^T \in \mathbb{R}^n \).

7.1 Fixed point iterations

**Theorem 7.1 (fixed point)** Let \( D \subset \mathbb{R}^n \) be a closed region and \( G \) be a mapping from \( D \) to itself satisfying the Lipschitz condition

\[ ||G(x) - G(y)|| \leq \gamma ||x - y||, \quad \forall x, y \in D, \]

where the Lipschitz constant \( \gamma < 1 \). Then, \( \exists \) a unique fixed point of \( G, p \in D \), and the fixed point iteration

\[ p_{n+1} = G(p_n), \quad n = 0, 1, ... \]

with \( p_0 \in D \) converges to \( p \).

**Proof.** Let \( p_0 \in D \). Then \( \{p_n\} \subset D \), because \( G \) maps \( D \) to \( D \). The difference between two consecutive iterates is

\[
\begin{align*}
||p_{n+1} - p_n|| & \leq ||G(p_n) - G(p_{n-1})|| \\
& \leq \gamma ||p_n - p_{n-1}|| \\
& \vdots \\
& \leq \gamma^n ||p_1 - p_0||.
\end{align*}
\]
Therefore,

\[
\|p_n - p_0\| = \left\| \sum_{i=0}^{n-1} (p_{i+1} - p_i) \right\| \\
\leq \sum_{i=0}^{n-1} \|p_{i+1} - p_i\| \\
\leq \|p_1 - p_0\| \sum_{i=0}^{n-1} \gamma^i \\
\leq \|p_1 - p_0\| \cdot \frac{\gamma}{1 - \gamma}
\]

for all \( n = 1, 2, \ldots \). Now, for any integers \( n, k > 0 \),

\[
\|p_{n+k} - p_n\| = \|G(p_{n+k-1}) - G(p_{n-1})\| \\
\leq \gamma \|p_{n+k-1} - p_{n-1}\| \\
: \\
\leq \gamma^n \|p_k - p_0\| \\
\leq \frac{\gamma^{n+1}}{1 - \gamma} \|p_1 - p_0\|.
\]

Taking the limit we have

\[
\lim_{n,k \to \infty} \|p_{n+k} - p_n\| = 0
\]

since \( 0 < \gamma < 1 \). So, \( \{p_n\} \) is a Cauchy sequence in \( D \), and it has a limit in \( D \) since \( D \) is closed.

Suppose \( \{p_n\} \) has two limits, say \( p \) and \( q \), both in \( D \). Then,

\[
\|p - q\| = \|G(p) - G(q)\| \leq \gamma \|p - q\|
\]

with \( \gamma < 1 \). This implies that \( \|p - q\| = 0 \). \( \square \)

**Corollary 7.1** Let \( G = (g_1, \ldots, g_n)^T \). If \( G \) satisfies

\[
\left| \frac{\partial g_i(x)}{\partial x_j} \right| \leq \frac{\gamma}{n}, \quad \forall x \in D
\]

for all \( i, j = 1, 2, \ldots, n \) with \( \gamma < 1 \), then

\[
p_{n+1} = G(p_n), \quad n = 0, 1, \ldots
\]

with \( p_0 \in D \) converges to the unique fixed point of \( G \) in \( D \).

**Proof.** (Scatch only) For any \( x, y \in D \), using a Taylor’s expansion

\[
G(x) - G(y) = \left( \sum_{j=1}^{n} \frac{\partial g_1(\xi)}{\partial x_j} (x_j - y_j), \ldots, \sum_{j=1}^{n} \frac{\partial g_n(\xi)}{\partial x_j} (x_j - y_j) \right)^T \\
= \begin{pmatrix}
\frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial g_n}{\partial x_1} & \cdots & \frac{\partial g_n}{\partial x_n}
\end{pmatrix}
\begin{pmatrix}
x_1 - y_1 \\
\vdots \\
x_n - y_n
\end{pmatrix}
\]

\[
= J(G)(x - y),
\]

65
where \( J(G) \) is called the Jacobian (Jacobi matrix) of \( G \). Taking the norm,

\[
\|G(x) - G(y)\| \leq \max_{1 \leq j \leq n} \sum_{i=1}^{n} \left| \frac{\partial g_i}{\partial x_j} \right| \|x - y\|
\]

\[
\leq \sum_{i=1}^{n} \frac{\gamma}{n} \|x - y\|
\]

\[
= \gamma \|x - y\|.
\]

Therefore, The Lipschitz condition is satisfied by \( G \).

\( \Box \)

Example. Consider the problem

\[
\begin{align*}
3x_1 - \cos(x_2 x_3) - \frac{1}{2} &= 0 \\
x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 &= 0 \\
e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0.
\end{align*}
\]

We rewrite these into

\[
\begin{align*}
x_1 &= \frac{1}{3} \cos(x_2 x_3) + \frac{1}{6} =: g_1 \\
x_2 &= \frac{1}{9} \sqrt{x_1^2 + \sin x_3 + 1.06} - 0.1 =: g_2 \\
x_3 &= -\frac{1}{20} \left( e^{x_1 x_2} - \frac{10\pi - 3}{3} \right) =: g_3.
\end{align*}
\]

It is easy to check that the conditions in the Corollary are satisfied. We check the first few below. When \( x_1, x_2, x_3 \in [-1, 1] \), we have

\[
\begin{align*}
\left| \frac{\partial g_1}{\partial x_1} \right| &= 0, \\
\left| \frac{\partial g_1}{\partial x_2} \right| &= \left| \frac{-1}{3} \sin(x_2 x_3) x_3 \right| < \frac{1}{3} \sin 1, \\
\left| \frac{\partial g_1}{\partial x_3} \right| &= \frac{1}{3} \left| \sin(x_2 x_3) x_2 \right| < \frac{1}{3} \sin 1, \\
\left| \frac{\partial g_2}{\partial x_1} \right| &= \frac{1}{9} \cdot \frac{1}{2} \left| \frac{2x_1}{\sqrt{x_1^2 + \sin x_3 + 1.06}} \right| \\
&\leq \frac{19}{\sqrt{1.06 + \sin x_3}} \\
&\leq \frac{|x_1|}{\sqrt{1.06 + \sin x_3}} \\
&\leq \frac{|x_1|}{\sqrt{0.06}} \\
&< 0.238.
\end{align*}
\]

Therefore, the fixed point iteration has a unique solution in \([0, 1]^3\).
7.2 Newton’s method

Assume that $x^*$ is a solution to (7.7.0.1), i.e., $f(x^*) = 0$. Then, when $x$ is sufficient close to $x^*$, a Taylor’s expansion gives

$$f(x^*) = f(x) + J(x)(x^* - x) + O(||x - x^*||^2),$$

where $J(x)$ is the Jacobian of $f$ defined by

$$J(x) = \begin{pmatrix}
\frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} 
\end{pmatrix}.$$

Since $f(x^*) = 0$, we have, omitting the second order term,

$$-f(x) \approx J(x)(x^* - x).$$

Therefore, we have

$$x^* \approx x - J^{-1}(x)f(x).$$

This motivates us to design the following algorithm:

Given $x_0$, for $k = 0, 1, \ldots$ until convergence, do

$$x_{k+1} = x_k - J^{-1}(x_k)f(x_k).$$

This is Newton’s method.

7.3 Quasi-Newton methods

The idea is to approximate $J$ by a finite difference. We have shown that in one-dimension, one choice is

$$f'(x_1) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0} \text{ or } f'(x_1)(x_1 - x_0) \approx f(x_1) - f(x_0).$$

In multi-dimensions, we have

$$J(x_1)(x_1 - x_0) \approx f(x_1) - f(x_0).$$

Now, assume that $x_0$ is given and $x_1$ is computed by the Newton’s method. We look for an approximation $A_1$ to $J(x_1)$ in the following way. Note

$$A_1(x_1 - x_0) = f(x_1) - f(x_0).$$

This shows that $A_1$ is a mapping defined for vectors parallel to $x_1 - x_0$. Note that $A_1$ is arbitrary for vectors perpendicular to $x_1 - x_0$. We define

$$A_1 z = J(x_0) z, \quad \forall z \perp (x_1 - x_0).$$
The above two equalities define $A_1$ uniquely as given below.

$$A_1 = J(x_0) + \frac{[f(x_1) - f(x_0) - J(x_0)(x_1 - x_0)](x_1 - x_0)^T}{||x_1 - x_0||^2}.$$  

Let us check this by right-multiplying the above by $(x_1 - x_0)$.

$$A_1(x_1 - x_0) = J(x_0)(x_1 - x_0) + \frac{[f(x_1) - f(x_0) - J(x_0)(x_1 - x_0)]||x_1 - x_0||^2}{||x_1 - x_0||^2} = f(x_1) - f(x_0),$$

and

$$A_1 z = J(x_0) z, \quad \forall z \perp (x_1 - x_0).$$

Therefore, we choose

$$x_2 = x_1 - A_1^{-1} f(x_1).$$

In general we have

$$A_i = A_{i-1} + \frac{y_i - A_{i-1} s_i}{||s_i||^2} s_i^T,$$

$$x_{i+1} = x_i - A_i^{-1} f(x_i),$$

where $y_i = f(x_i) - f(x_{i-1})$ and $s_i = x_i - x_{i-1}$. This is called Broyden’s method.

### 7.4 The steepest descent method

Consider the following minimization problem:

$$\min g(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} f_i^2(x_1, x_2, ..., x_n).$$

Obviously, if $f(x)$ has a root, then

$$g_{\text{min}} = 0 \quad \Rightarrow \quad f_i(x) = 0, \quad i = 1, 2, ..., n.$$  

Starting from an initial guess $x_0 = (x_1^0, ..., x_n^0)^T$, we find $\alpha_k^* \in \mathbb{R}$ such that $\alpha_k^*$ minimizes

$$h(\alpha) = g(x_k - \alpha \nabla g(x_k)).$$

(Note that this is a one-dimensional problem). We then let

$$x_{k+1} = x_k - \alpha_k^* \nabla g(x_k), \quad k = 0, 1, ...$$

This is the steepest descent method.