ME5704: Numerical Methods in Mechanical Engineering and Robotics

Dr. Danielle Tan

mpetds@nus.edu.sg

Dept. of Mechanical Engineering, E1-05-30,

National University of Singapore

Contents

Recommended Reading

- **Steven C. Chapra & Raymond P. Canale, *Numerical Methods for Engineers*, McGraw-Hill, 1998, 2002, 2006, 2010.**
- Raman S. Esfandiari, *Numerical Methods for Engineers and Scientists Using Matlab*, CRC Press, 2013.
- Amos Gilat & Vish Subramaniam, *Numerical Methods for Engineers and Scientists*, John Wiley & Sons, Inc., 2011, 2014.

Other books on numerical methods (approximation, optimisation, integration, differentiation etc.) can be found in the Central Library under call numbers **QA 297**.

1 Introduction

Overview

This course is meant to introduce you to a variety of numerical methods and techniques which are applicable in mechanical engineering, ranging from situations with data sets to systems of equations and equations which have integrals and/or derivatives. We will not be learning how to deal with them analytically, as that is a whole different branch of mathematics; rather we will be exploring how to manipulate them in order to obtain a form that is easier to work with (or programme), or to determine an approximate solution (especially in cases where the analytical solution may not be difficult to determine).

The main topics we will be covering are:

- Curve-Fitting & Fourier Transforms

  This may be one of the more familiar of the topics, in that the idea of curve-fitting is not new, and many of you would have Excel's 'trendline' tool before when plotting. Here we will delve into how we can fit such trendlines for situations more complex than a proportional relationship.

  Specifically, we will look at:

  - least-squares regression (linear and nonlinear);
  - interpolation methods such as Newton's divided difference, Lagrange interpolation polynomials, splines and multidimensional interpolation; and
  - Fourier approximation (sinusoidal curve-fitting, DFT & FFT).

- Root-Finding Methods

  Typically these are iterative methods meant to arrive at the exact solution of an algebraic expression, based on an initial guess (or guesses). There are many numerical techniques that fall under this umbrella, some of which you may have come across or used before. We will cover some commonly-used methods:

  - bracketing methods (e.g. bisection and false-position methods);
  - open methods (e.g. Newton-Raphson and inverse quadratic interpolation);

  - methods for linear systems such as Jacobi & Gauss-Seidel iteration methods; and
  - methods for polynomials (e.g. Müller's and Bairstow's methods).

- Optimisation Methods

  Similar to root-finding, this topic typically involves iteration with the goal of finding the value(s) corresponding to a function's local maxima or minima. In practical situations there may also be some conditions that have to be satisfied simultaneously, so we will look at both unconstrained (no conditions) and constrained optimisation.

  The techniques we will cover are:

  - 1D unconstrained optimisation methods such as parabolic interpolation & Newton's method;
  - direct & gradient methods for multidimensional unconstrained optimisation; and
  - constrained optimisation methods for linear systems.

- Numerical Integration & Differentiation

  This topic covers techniques and formulae used to approximate integrals or derivatives, used mostly in situations where the function is unknown or the analytical solution might not be easy or possible to achieve. We will explore:

  - Newton-Cotes integration formulae (such as trapezoidal rule, Simpson's rules, multiple integrals and improper integrals);
  - integration of functions (e.g. using Romberg integration, adaptive quadrature and Gauss quadrature); and
  - differentiation techniques such as high-accuracy differentiation formulae and Richardson's extrapolation.

- Differential Equations

  Integration is one way of solving a differential equation. We will introduce other methods here:

  - for Ordinary Differential Equations (ODEs)

    - Runge-Kutta methods;
    - adaptive methods; and
    - multistep & stiffness methods.
  - for Partial Differential Equations (PDEs)
    - elliptic problems (including boundary value problems);
    - explicit & implicit methods for parabolic and hyperbolic problems;
    - alternating-direction implicit method for multi-dimension parabolic problems.

Unsurprisingly, these methods are all very programmable and relatively easy to implement in Matlab, Excel or similar computing software. For in-class exercises, we will mostly use regular calculators but you are welcome to use anything you prefer.

2 Curve-Fitting & Fourier Transforms

2.1 Introduction

In this chapter we will look at some topics that appear to be unrelated at first. However they all arise from a common motivation – typically we have a set of data, from which we would like to extract some information.

In engineering, we most commonly need to perform curve-fitting in order to do a **trend analysis** – that is, making predictions using the existing data – or **hypothesis testing** – determining whether or not a particular pattern is significant by comparing a specified mathematical model (equation) to the data.

How good is our data? This is one of the main differences between the topics. If we assume that there is some error or uncertainty in the data, then we do not expect our curve to pass through every single point. Instead we require that the overall pattern of the data is captured by the curve. Alternatively it could be that we are making a guess at the function of the curve, and thus expect that it might not be exact (and therefore unable to pass through every single data point). In both these scenarios what we would like is for the curve to pass closely to as many points as possible, with no huge differences. This is what we term **least-squares regression**. On the other hand, if we expect our data to be good, then correspondingly we should be able to obtain a curve (or a series of curves) to fit exactly through every single data point. **Interpolation** is when we follow this up by estimating values between the known data points.

The second main difference is the type of relationship that we are seeing: is it cyclic? Typically this is more likely to be a concern when we are looking at time-varying functions, i.e. time is one of the independent variables. If it is periodic, then we could use a finite **Fourier series**. If it is non-periodic and we are more interested in analysing it in the frequency domain (e.g. it might be the signal corresponding to a short blast of sound) rather than the time domain then we would be using **Fourier transforms**.

2.2 Least-Squares Regression

2.2.1 Linear Regression

We have all, at some point in school, performed linear regression. Whether it was using a ruler to draw a straight line through a set of points or simply by choosing to fit a trendline in computation software like Excel. With the ruler it is obvious that this line was decided visually – which as you can expect, differs depending on who is looking – but how does it work in Excel and other software?

Generally the basic idea is to minimise the total 'error' – that is, the (vertical) difference between the fitted line and the data points should be as small as possible. To illustrate this concept, let's consider a case where we only have 2 variables, x and y. y is the dependent variable which varies with the independent variable x. So by taking measurements at discrete values of x (x_1, x_2, x_3, ... , x_N) we get a set of discrete data points (y_1, y_2, y_3, ... , y_N).

Now ideally, our line will pass through every single data point (x_i, y_i). However realistically this is not going to happen, in which case we can expect that there will be a difference between the data point and the corresponding value on the line $(x_i, f(x_i))$.

We shall call this difference the error (also known as the **residual**):

$$e(x_i) = e_i \equiv y_i - f(x_i)\,.$$

Then logically, the line that fits best should be the one for which the total error (i.e. we sum up all the e_i) is the least. However, looking at Fig. 1, we see that it is possible to have both positive and negative error. Thus instead of summing the error as is, we sum the *squared* values of the error:

$$S_r \equiv \sum_i^N e_i^2 = \sum_i^N \left(y_i - f\left(x_i\right)\right)^2 .$$

The next step is then to determine which line results in the smallest S_r, in other words the fit for which the sum of the squared error is the least – hence the name 'least-squares regression'.

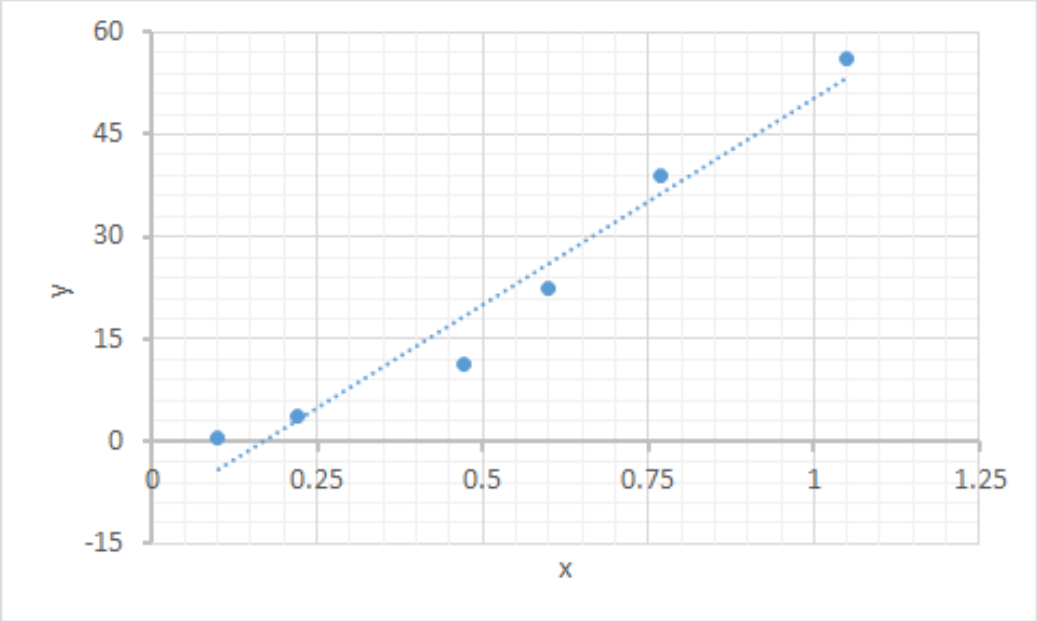


Figure 1: Data points (x_i, y_i) and linear fit $y = f(x)$.

The general equation for a straight line is

$$y = a_1 x + a_0 \,.$$

There may be circumstances under which we fix either the gradient or the intercept (e.g. if we set $a_0 = 0$ and thus are left with only $y = a_1 x$), but a lot of the time both coefficients a_1 and a_0 are left unknown.

So if we substitute this expression into our total squared-error:

$$S_r = \sum_i^N e_i^2 = \sum_i^N \left(y_i - (a_1 x_i + a_0)\right)^2 .$$

Following our previous logic, the best-fit line will be the one for which the combination of a_0 and a_1 results in a local minimum. In other words, we want:

$$\frac{\partial S_r}{\partial a_0} = 0$$

and

$$\frac{\partial S_r}{\partial a_1} = 0 \,.$$

The relevant derivatives are:

$$\begin{aligned}
\frac{\partial S_r}{\partial a_1} &= \frac{\partial}{\partial a_1} \sum_i^N \left(y_i - a_1 x_i - a_0\right)^2 \\
&= \sum_i^N \frac{\partial}{\partial a_1} \left(y_i - a_1 x_i - a_0\right)^2 \\
&= \sum_i^N 2\left(y_i - a_1 x_i - a_0\right)\left(-x_i\right) \\
&= \sum_i^N 2\left(a_1 x_i^2 + a_0 x_i - x_i y_i\right) \\
&= 2a_1 \sum_i^N x_i^2 + 2a_0 \sum_i^N x_i - 2\sum_i^N x_i y_i \,,
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial S_r}{\partial a_0} &= \frac{\partial}{\partial a_0} \sum_i^N \left(y_i - a_1 x_i - a_0\right)^2 \\
&= \sum_i^N \frac{\partial}{\partial a_0} \left(y_i - a_1 x_i - a_0\right)^2 \\
&= \sum_i^N 2\left(y_i - a_1 x_i - a_0\right)\left(-1\right) \\
&= \sum_i^N 2\left(a_1 x_i + a_0 - y_i\right) \\
&= 2a_1 \sum_i^N x_i + 2Na_0 - 2\sum_i^N y_i \,.
\end{aligned}$$

Setting both derivatives to zero, we get a pair of simultaneous equations:

$$\begin{aligned}
a_1 \sum_i^N x_i^2 + a_0 \sum_i^N x_i - \sum_i^N x_i y_i = 0 \,, \\
a_1 \sum_i^N x_i + Na_0 - \sum_i^N y_i = 0 \,.
\end{aligned}$$

These can then be solved to obtain:

$$\begin{aligned}
a_1 &= \frac{N\sum_i^N x_i y_i - \sum_i^N x_i \sum_i^N y_i}{N\sum_i^N x_i^2 - \left(\sum_i^N x_i\right)^2} \,, \\
a_0 &= \frac{1}{N}\left(\sum_i^N y_i - a_1 \sum_i^N x_i\right) \,.
\end{aligned}$$

Some of you might cleverly wonder whether this might actually result in something other than a local minimum. Rest assured: we know that it will never be a local maximum because $S_r \geq 0$, and a really awful fit (e.g. nearly vertical line) would have $S_r \to \infty$.

Exercise

Using the table below, perform a linear regression for a set of data points.

x_i	y_i	x_i^2	$x_i y_i$

If you wish, you can check your answer against Excel's trendline. Are they the same?

How much error is there in the linear regression? We have a measure of that from S_r, our total squared-error. We can also express the 'goodness of fit' using a 'standard deviation' of sorts:

$$s_{y/x} \equiv \sqrt{\frac{S_r}{N-2}},$$

where we have essentially averaged out the total squared-error over the number of data points we have (minus two, because we already estimated two unknowns – the coefficients

a_0 and a_1 from the data), and taken the square-root. This term is known as the **standard error of the estimate**.

When we use Excel and other statistical software, we often come across a parameter R, which we are told is a **correlation coefficient** that determines the 'goodness of fit'. R is actually related to our S_r and a new term S_t which is defined as the sum of the differences between the dependent variable y_i and the arithmetic mean $\overline{y}$. Specifically:

$$\begin{aligned} \overline{y} &= \frac{\sum_i^N y_i}{N}\,, \\ S_t &= \sum_i^N \left(y_i - \overline{y}\right)^2, \end{aligned}$$

and

$$R^2 = \frac{S_t - S_r}{S_t}\,.$$

Mathematically R^2 - known as the **coefficient of determination** - is the normalised reduction (improvement) in error for the regression fit compared to just the average value. For most of us though, the main takeaway would be that: *the closer R is to 1, the better the fit; the closer R is to 0, the worse the fit.*

2.2.2 Polynomial Regression

In the previous section, we conducted a least-squares regression for linear relationships, that is, straight lines. Logically, we should also be able to do so for nonlinear relationships, particularly if we cannot transform the predicted relationship into a straight line. Here, we will look at **polynomial** regression, i.e. regression for a polynomial relationship of the form:

$$y = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m\,.$$

Let's start with the simplest polynomial function:

$$y = a_0 + a_1 x + a_2 x^2\,.$$

As before, the sum of the squares of the residuals is

$$S_r = \sum_i^N e_i^2 = \sum_i^N \left(y_i - \left(a_0 + a_1 x_i + a_2 x^2\right)\right)^2.$$

Following our previous logic, the best-fit curve will be the one for which the combination of a_0, a_1 and a_2 results in a local minimum. In other words, we want:

$$\frac{\partial S_r}{\partial a_0} = \frac{\partial S_r}{\partial a_1} = \frac{\partial S_r}{\partial a_2} = 0\,.$$

The relevant derivatives are:

$$\begin{aligned}
\frac{\partial S_r}{\partial a_0} &= \frac{\partial}{\partial a_0}\sum_i^N \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \\
&= \sum_i^N \frac{\partial}{\partial a_0}\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \\
&= \sum_i^N 2\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)(-1) \\
&= \sum_i^N 2\left(a_0 + a_1 x_i + a_2 x_i^2 - y_i\right) \\
&= 2Na_0 + 2a_1\sum_i^N x_i + 2a_2\sum_i^N x_i^2 - 2\sum_i^N y_i\,,
\end{aligned}$$

$$\begin{aligned}
\frac{\partial S_r}{\partial a_1} &= \frac{\partial}{\partial a_1}\sum_i^N \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \\
&= \sum_i^N \frac{\partial}{\partial a_1}\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \\
&= \sum_i^N 2\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)(-x_i) \\
&= \sum_i^N 2\left(a_0 x_i + a_1 x_i^2 + a_2 x_i^3 - x_i y_i\right) \\
&= 2a_0\sum_i^N x_i + 2a_1\sum_i^N x_i^2 + 2a_2\sum_i^N x_i^3 - 2\sum_i^N x_i y_i\,,
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial S_r}{\partial a_2} &= \frac{\partial}{\partial a_0}\sum_i^N \left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \\
&= \sum_i^N \frac{\partial}{\partial a_0}\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)^2 \\
&= \sum_i^N 2\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)\left(-x_i^2\right) \\
&= \sum_i^N 2\left(a_0 x_i^2 + a_1 x_i^3 + a_2 x_i^4 - x_i^2 y_i\right) \\
&= 2a_0\sum_i^N x_i^2 + 2a_1\sum_i^N x_i^3 + 2a_2\sum_i^N x_i^4 - 2\sum_i^N x_i^2 y_i\,.
\end{aligned}$$

Setting all derivatives to zero, we get a set of simultaneous equations:

$$\begin{aligned}
Na_0 + a_1\sum_i^N x_i + a_2\sum_i^N x_i^2 - \sum_i^N y_i &= 0\,, \\
a_0\sum_i^N x_i + a_1\sum_i^N x_i^2 + a_2\sum_i^N x_i^3 - \sum_i^N x_i y_i &= 0\,, \\
a_0\sum_i^N x_i^2 + a_1\sum_i^N x_i^3 + a_2\sum_i^N x_i^4 - \sum_i^N x_i^2 y_i &= 0\,.
\end{aligned}$$

These can then be solved using the data points (x_i, y_i).

Exercise

Using the data set from the example in the previous section, determine the equation of the best-fitting quadratic curve which is plotted in Fig. 2. Do you get the same answer as in the figure caption?

x_i	y_i	x_i^2	x_i^3	x_i^4	$x_i y_i$	$x_i^2 y_i$
0.10	0.5					
0.22	3.8					
0.47	11.4					
0.60	22.3					
0.77	39.1					
1.05	56.2					

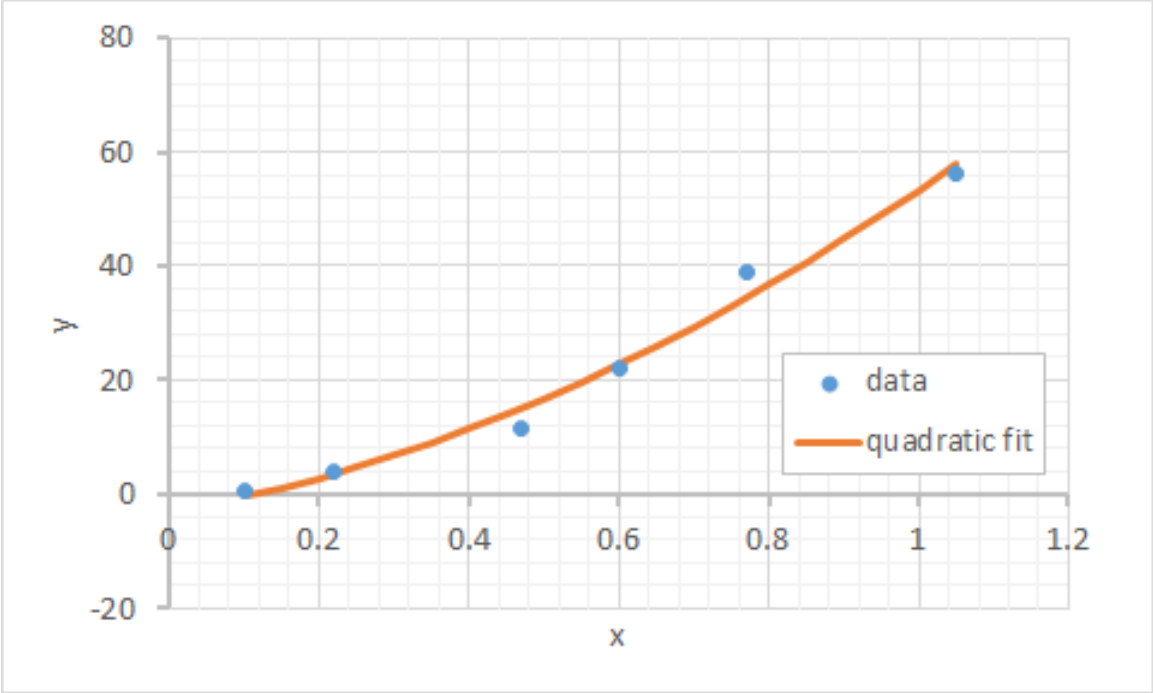


Figure 2: Data points (x_i, y_i) and quadratic fit $y = a_0 + a_1 x + a_2 x^2$, where $a_0 = -3.07$, $a_1 = 23.0$ and $a_2 = 33.3$.

We can of course, extend this logic to higher-order polynomials. So for an m-th order polynomial

$$y = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m \,,$$

we would have $m+1$ derivatives to set to 0, leading to a set of $m+1$ simultaneous equations to solve for the $m + 1$ unknown coefficients.

Correspondingly, the standard error for this polynomial regression would be

$$s_{y/x} = \sqrt{\frac{S_r}{N - (m+1)}} \,.$$

Essentially, the algorithm for least-squares regression is as follows:

1. Determine form of predicted relationship.

2. Linearise the relationship if needed.

3. Write out the expression for sum of squared residuals, S_r.

4. Differentiate S_r with respect to all the unknown coefficients a_0, a_1, a_2 etc.

5. Setting the derivatives to zero, obtain the set of simultaneous equations to be solved.

6. Solve the simultaneous equations (e.g. using Gaussian elimination).

7. Plug the now-determined coefficients back into predicted relationship $\Rightarrow$ best-fit curve.

This algorithm can also be applied to cases of higher dimensions, that is for determining a relationship for $y = f(x_1, x_2)$. For instance, consider the predicted relationship:

$$z = a_0 + a_1 x + b_1 y + b_2 y^2 .$$

What we will be fitting is actually a plane, rather than a curve. However the basic concepts of least-squares regression still apply.

The sum of squared-residuals is:

$$S_r = \sum_i^N \left[z_i - \left(a_0 + a_1 x_i + b_1 y_i + b_2 y_i^2 \right) \right]^2 .$$

For the best fit, we would need the combination of a_0, a_1, b_1 and b_2 that results in a local minimum for S_r.

The derivatives to set to zero would be:

$$\begin{aligned} \frac{\partial S_r}{\partial a_0} &= 2 \sum_i^N \left(z_i - a_0 - a_1 x_i - b_1 y_i - b_2 y_i^2 \right) (-1) \\ &= -2 \sum_i^N z_i + 2N a_0 + 2a_1 \sum_i^N x_i + 2b_1 \sum_i^N y_i + 2b_2 \sum_i^N y_i^2 , \end{aligned}$$

$$\begin{aligned} \frac{\partial S_r}{\partial a_1} &= 2 \sum_i^N \left(z_i - a_0 - a_1 x_i - b_1 y_i - b_2 y_i^2 \right) (-x_i) \\ &= -2 \sum_i^N x_i z_i + 2a_0 \sum_i^N x_i + 2a_1 \sum_i^N x_i^2 + 2b_1 \sum_i^N x_i y_i + 2b_2 \sum_i^N x_i y_i^2 , \end{aligned}$$

$$\begin{aligned}\frac{\partial S_r}{\partial b_1} &= 2\sum_i^N \left(z_i - a_0 - a_1 x_i - b_1 y_i - b_2 y_i^2\right)(-y_i) \\ &= -2\sum_i^N y_i z_i + 2a_0 \sum_i^N y_i + 2a_1 \sum_i^N x_i y_i + 2b_1 \sum_i^N y_i^2 + 2b_2 \sum_i^N y_i^3 \,,\end{aligned}$$

and

$$\begin{aligned}\frac{\partial S_r}{\partial b_2} &= 2\sum_i^N \left(z_i - a_0 - a_1 x_i - b_1 y_i - b_2 y_i^2\right)(-y_i^2) \\ &= -2\sum_i^N y_i^2 z_i + 2a_0 \sum_i^N y_i^2 + 2a_1 \sum_i^N x_i y_i^2 + 2b_1 \sum_i^N y_i^3 + 2b_2 \sum_i^N y_i^4 \,.\end{aligned}$$

Thus we have 4 simultaneous equations to solve for the 4 unknown coefficients, which would determine the best-fit plane.

Whether applying least-squares regression to higher order polynomials, or higher dimensions, it is generally easier to do so using a computer. Fortunately the algorithm outlined above is easy to implement, especially in Matlab which is built to manipulate matrices.

2.2.3 Nonlinear Regression

Linearisation

Let's now consider what happens when we have data that points to a relationship that is very obviously *non*linear, for instance:

$$y = c_1 e^{c_2 x} \,, \quad y = c_1 x^{c_2} \,, \quad \text{or} \quad y = \frac{c_1 x}{c_2 + x} \,.$$

One trick is to rewrite it in a form similar to $y = a_1 x + a_0$, with the uncoefficients c_1 and c_2 appearing as part of the linear expression coefficients a_1 and a_0.

Example 1

To illustrate, let's consider the exponential expression above:

$$\begin{aligned} y &= c_1 e^{c_2 x} \\ \ln y &= \ln\left(c_1 e^{c_2 x}\right) \\ \ln y &= \ln c_1 + c_2 x \end{aligned}$$

Comparing this to the general equation for a straight line, we see that if we plot $\ln y$ against x, this will be a linear relationship. The linear regression we derived for general straight lines can be applied to solve for the coefficients, where now:

$$\begin{aligned} \ln c_1 &= a_0 \quad \Rightarrow \quad c_1 = e^{a_0}\,, \quad \text{and} \\ c_2 &= a_1\,. \end{aligned}$$

Example 2

Let's try to linearise the power equation:

$$\begin{aligned} y &= c_1 x^{c_2} \\ \lg y &= \lg\left(c_1 x^{c_2}\right) \\ \lg y &= \lg c_1 + c_2 \lg x \end{aligned}$$

So if we plot $\lg y$ against $\lg x$, this will again be a linear relationship. The linear regression we derived for general straight lines can be applied to solve for the coefficients, where now:

$$\begin{aligned} \lg c_1 &= a_0 \quad \Rightarrow \quad c_1 = 10^{a_0}\,, \quad \text{and} \\ c_2 &= a_1\,. \end{aligned}$$

Incidentally, this is exactly what happens when we choose to plot a function on log-log axes in hopes of getting a straight line. ;)

Exercise 1

What would a linearised form of the nonlinear expression, $y = \frac{c_1 x}{c_2 + x}$ be? Determine c_1 and c_2 in terms of the linear regression coefficients a_0 and a_1.

Exercise 2

A set of data (shown in the table) was obtained. It is unclear whether a power law or an exponential fit would be better. Perform a linear regression for both fits, and plot them on the relevant axes (Figs. 3 and 4). Which fit do you think is more appropriate?

x_i	y_i	'new' x_i	'new' y_i	'new' x_i^2	'new' $x_i y_i$
0.10	0.5				
0.22	3.8				
0.47	11.4				
0.60	22.3				
0.77	39.1				
1.05	56.2				

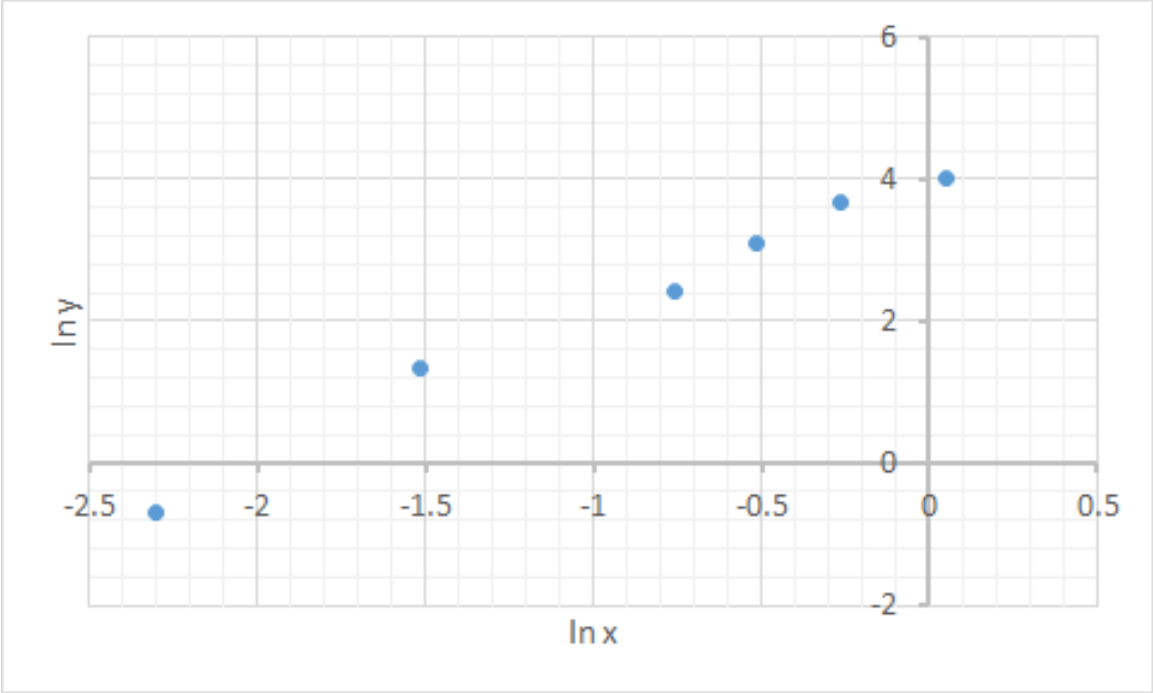


Figure 3: Data points (x_i, y_i) and power law fit $y = c_1 x^{c_2}$.

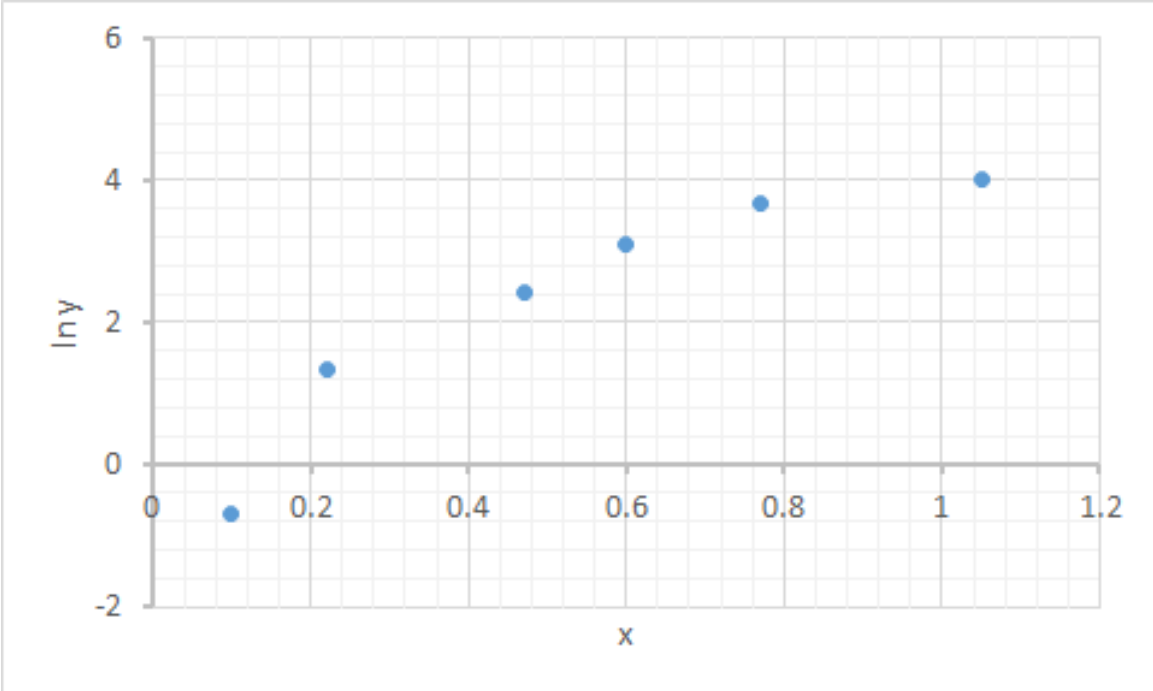


Figure 4: Data points (x_i, y_i) and exponential fit $y = c_1 e^{c_2 x}$.

Gauss-Newton Method

Now let's consider situations where linearisation of the function is not feasible or straightforward, for e.g. $f(x) = a_0\left(1 - e^{-a_1 x}\right) + e$. We are still going to minimise the residuals, but first we need to transform the nonlinear function into something more manageable. There exist many algorithms for doing so, but in this course we will only introduce the Gauss-Newton method which uses the Taylor series expansion of the nonlinear function. It involves making an initial guess for the unknown coefficients, and an iterative process to improve them.

The first step is to expand the aforementioned nonlinear function to the 1st order derivative (so it is linear with respect to the unknown coefficients e.g. a_0 and a_1). If we define j as our current guess (for a_0 and a_1) and $j+1$ as the prediction/improved guess, then:

$$f(x)_{j+1} = f(x)_j + \left.\frac{\partial f(x)}{\partial a_0}\right|_j \Delta a_0 + \left.\frac{\partial f(x)}{\partial a_1}\right|_j \Delta a_1 \,,$$

where

$$\begin{aligned} \Delta a_0 &= a_{0,j+1} - a_{0,j} \,, \\ \Delta a_1 &= a_{1,j+1} - a_{1,j} \,. \end{aligned}$$

In this form it is now possible to proceed with least-squares regression as per usual, except that where we previously had a_0 and a_1, we now have Δa_0 and Δa_1. The sum of squared-residuals can be written as:

$$\begin{aligned} S_r &= \sum_i^N e_i^2 = \sum_i^N \left(y_i - f(x_i)_{j+1}\right)^2 \\ &= \sum_i^N \left(y_i - \left(f(x_i)_j + \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \Delta a_0 + \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \Delta a_1\right)\right)^2 \\ &= \sum_i^N \left(y_i - f(x_i)_j - \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \Delta a_0 - \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \Delta a_1\right)^2 . \end{aligned}$$

Note that now the derivatives to set to zero are:

$$\begin{aligned} \frac{\partial S_r}{\partial \Delta a_0} &= 2\sum_i^N \left(y_i - f(x_i)_j - \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \Delta a_0 - \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \Delta a_1\right)\left(-\left.\frac{\partial f(x_i)}{\partial a_0}\right|_j\right) \\ &= 2\left[-\sum_i^N \left(y_i - f(x_i)_j\right)\left.\frac{\partial f(x_i)}{\partial a_0}\right|_j + \Delta a_0 \sum_i^N \left(\left.\frac{\partial f(x_i)}{\partial a_0}\right|_j\right)^2 + \Delta a_1 \sum_i^N \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j\right] , \end{aligned}$$

and

$$\begin{aligned}\frac{\partial S_r}{\partial \Delta a_1} &= 2\sum_i^N \left(y_i - f(x_i)_j - \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \Delta a_0 - \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \Delta a_1 \right) \left(- \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \right) \\ &= 2\left[-\sum_i^N (y_i - f(x_i)_j) \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j + \Delta a_0 \sum_i^N \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j + \Delta a_1 \sum_i^N \left(\left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \right)^2 \right] .\end{aligned}$$

From these we can solve for Δa_0 and Δa_1 in terms of various summations. Our new guess for the coefficients $a_{0,j+1}$ and $a_{1,j+1}$ can then be determined.

As mentioned earlier this is not a one-step process, because nearly all the summations need to be (re-)evaluated for each guess $a_{0,j}$ and $a_{1,j}$. We will keep cycling through the steps of calculating the summations, solving for Δa_0 and Δa_1, and obtaining the new guess... until the guesses have converged i.e. Δa_0 and Δa_1 are smaller than some threshold that we set.

In this demonstration there were only 2 unknowns a_0 and a_1. The process is the same even if there were more unknown coefficients, just that the 1st order Taylor series will have additional terms (which carry on into the derivatives) and there would be more equations to solve simultaneously.

2.2.4 General Matrix Formulation

You may be thinking that all this seems like a lot of calculation that needs to be done even before a programme can be written, and the data needs to undergo a lot of manipulation before it can be used to solve for the unknown coefficients. Fortunately there is a general formulation that works very well for programming purposes.

If applying least-squares regression to a polynomial of order p, the sum of squared-residuals can be expressed as

$$S_r = \sum_i^N \left(y_i - a_0 - a_1 x_i - a_2 x_i^2 - ... a_p x_i^p \right)^2 .$$

The set of simultaneous equations that we obtain after setting all the relevant derivatives

to zero, when expressed in matrix form, is:

$$\begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \\ \vdots \\ \sum y_i x_i^p \end{bmatrix} = \begin{bmatrix} N & \sum x_i & \sum x_i^2 & \dots & \sum x_i^p \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \dots & \sum x_i^{p+1} \\ \vdots & \vdots & \vdots & & \vdots \\ \sum x_i^p & \sum x_i^{p+1} & \sum x_i^{p+2} & \dots & \sum x_i^{2p} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_p \end{bmatrix},$$

where all the $\sum_i^N$ have been written as $\sum$ for overall clarity.

If we define the following column vectors (using the given set of data points):

$$\mathrm{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \mathrm{Z} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & \dots & x_2^p \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^p \end{bmatrix} \quad \text{and} \quad \mathrm{A} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_p \end{bmatrix},$$

then the matrix equation can be expressed as

$$\begin{aligned} \mathrm{Z}^T\mathrm{Y} &= \left(\mathrm{Z}^T\mathrm{Z}\right)\mathrm{A} \\ \Rightarrow \quad \mathrm{A} &= \left(\mathrm{Z}^T\mathrm{Z}\right)^{-1}\left(\mathrm{Z}^T\mathrm{Y}\right) \end{aligned}$$

which is much easier to write a programme for, especially if using software like Matlab which processes matrices easily.

This works even in the Gauss-Newton method. For the demonstrated example with unknown coefficients a_0 and a_1 where

$$S_r = \sum_i^N \left(y_i - f(x_i)_j - \left.\frac{\partial f(x_i)}{\partial a_0}\right|_j \Delta a_0 - \left.\frac{\partial f(x_i)}{\partial a_1}\right|_j \Delta a_1 \right)^2,$$

we can define the column vectors:

$$\mathrm{Y} = \begin{bmatrix} y_1 - f(x_1) \\ y_2 - f(x_2) \\ \vdots \\ y_N - f(x_N) \end{bmatrix}, \quad \mathrm{Z} = \begin{bmatrix} \frac{\partial f(x_1)}{\partial a_0} & \frac{\partial f(x_1)}{\partial a_1} \\ \frac{\partial f(x_2)}{\partial a_0} & \frac{\partial f(x_2)}{\partial a_1} \\ \vdots & \vdots \\ \frac{\partial f(x_N)}{\partial a_0} & \frac{\partial f(x_N)}{\partial a_1} \end{bmatrix}_j \quad \text{and} \quad \Delta\mathrm{A} = \begin{bmatrix} \Delta a_0 \\ \Delta a_1 \end{bmatrix}.$$

The least-squares regression process will then boil down to (again):

$$\begin{aligned} \mathrm{Z}^T\mathrm{Y} &= \left(\mathrm{Z}^T\mathrm{Z}\right)\Delta\mathrm{A} \\ \Rightarrow \quad \Delta\mathrm{A} &= \left(\mathrm{Z}^T\mathrm{Z}\right)^{-1}\left(\mathrm{Z}^T\mathrm{Y}\right). \end{aligned}$$

2.3 Interpolation

There are many ways of interpolating - that is, using known data points to estimate the value at some intermediate point. The simplest and most common method is to assume a linear relationship between two points. For instance when calibrating a thermometer, we obtain the position of the mercury thread at 0° and 100° using steam and ice water respectively. Then, by assuming that the change in the position of the mercury is proportional to the change in temperature, we are able to mark out the corresponding positions for any temperature between 0° and 100°.

We know, however, that a linear relationship might not be good enough in many situations and thus a nonlinear relationship might be better. Accordingly there have been many methods developed to account for that nonlinearity, and thus we shall also look at such methods in the following subsections.

2.3.1 Newton's Divided-Difference Interpolating Polynomials

As mentioned briefly, the most obvious way to interpolate is to assume a relationship between the data points. However it would be unreasonable to think that our chosen relationship satisfies *all* the data points (unless we are fantastic at guessing), as we have seen from the previous section on regression. What we can do, however, is to assume that our chosen relationship applies within a small range of the data, that includes the intermediate point(s) we are interested in.

If we assume a linear relationship – as in the thermometer example – then we only require two data points. Therefore it makes sense to pick the two that immediately bracket our intermediate point of interest. If we assume a quadratic relationship, we need three data points. Cubic, four data points etc. This is not unexpected as these are the minimum number of points required to define such relationships. We simply have the freedom of choosing which data points these are going to be. As we can guess, even this choice of data points may affect the interpolation function. Essentially,

For an assumed relationship of order n – i.e. an n-th order polynomial – we require at least $n+1$

data points.

So what do we do with our $n+1$ data points? Because we have assumed that our chosen relationship satisfies this range of data points, then we can obtain $n+1$ equations from them, simply by substituting each data point x_i in the relationship and setting it equal to the measured functional value y_i:

$$\begin{aligned} y_1 &= c_0 + c_1 x_1 + c_2 x_1^2 + c_3 x_1^3 + c_4 x_1^4 + ... + c_n x_1^n \\ y_2 &= c_0 + c_1 x_2 + c_2 x_2^2 + c_3 x_2^3 + c_4 x_2^4 + ... + c_n x_2^n \\ y_3 &= c_0 + c_1 x_3 + c_2 x_3^2 + c_3 x_3^3 + c_4 x_3^4 + ... + c_n x_3^n \\ \vdots &= \vdots \\ y_n &= c_0 + c_1 x_n + c_2 x_n^2 + c_3 x_n^3 + c_4 x_n^4 + ... + c_n x_n^n \end{aligned}$$

We then use this set of $n+1$ simultaneous equations to solve for the currently unknown coefficients c_0, c_1, ... c_n.

Clearly, for a linear relationship $f_1(x) = c_0 + c_1 x$, we would only have 2 equations:

$$\begin{aligned} y_1 &= c_0 + c_1 x_1 \\ y_2 &= c_0 + c_1 x_2 \end{aligned}$$

This is easy for us to solve, and we obtain the expected expressions for slope c_1 and intercept c_0:

$$\begin{aligned} c_1 &= \frac{y_2 - y_1}{x_2 - x_1}\,, \\ c_0 &= y_1 - c_1 x_1 = y_1 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right) x\,. \end{aligned}$$

Substituting back into our relationship, we obtain the final form of our interpolation function:

$$f_1(x) = y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1)\ .$$

As we can imagine, the higher the order of our relationship, the more equations there are to solve... and this is where **Newton's divided-difference interpolating polynomials** come in to make our lives a bit easier.

Here, the 'divided-difference' refers to the fractional coefficient in front of the $(x - x_1)$ term. Let's now consider what it looks like for a 2nd order (quadratic) interpolation function

$$f_2(x) = c_0 + c_1 x + c_2 x^2$$

which can also be written as:

$$f_2(x) = b_0 + b_1 (x - x_1) + b_2 (x - x_1)(x - x_2) \ ,$$

for a set of 3 data points (x_1, y_1), (x_2, y_2) and (x_3, y_3). In this form it is relatively simple to obtain the coefficients: substitute in first $x = x_1$ and then $x = x_2$ to solve for b_0, b_1 and finally b_2 in that order.

$$\begin{aligned} b_0 &= y_1 \, , \\ b_1 &= \frac{y_2 - y_1}{x_2 - x_1} \, , \\ b_2 &= \frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1} \, . \end{aligned}$$

Note that:

- b_0 and b_1 are exactly the same as for the 1st order (linear) interpolation $f_1(x) = b_0 + b_1 (x - x_1)$, as we derived earlier.
- b_1 is called the **first finite divided difference**, and is the difference in y divided by the difference in x.
- b_1 is the subtracted 2nd term in the numerator of b_2.
- b_2 is called the **second finite divided difference**, and is the difference in two first finite divided differences divided by the difference in x.

Now that we have seen Newton's divided-difference interpolating polynomials for both 1st and 2nd order, we can introduce the general form for a polynomial of order n which fits $n + 1$ data points:

$$f_n(x) = b_0 + b_1(x - x_1) + b_2(x - x_1)(x - x_2) + \cdots + b_n(x - x_1)(x - x_2) \ldots (x - x_n) \, ,$$

where the coefficients are:

$$
\begin{aligned}
b_0 &= f_n(x_1) = y_1 \,, \\
b_1 &= f[x_2, x_1] = \frac{y_2 - y_1}{x_2 - x_1} \,, \\
b_2 &= f[x_3, x_2, x_1] = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1} \,, \\
&\vdots \\
b_n &= f[x_{n+1}, x_n, x_{n-1}, \ldots, x_1] = \frac{f[x_{n+1}, x_n, \ldots x_2] - f[x_n, x_{n-1}, \ldots, x_1]}{x_{n+1} - x_1} \,.
\end{aligned}
$$

Basically each successive finite divided difference involves an additional data point, as well as the difference of two of the previous divided differences, eventually involving all of the $n + 1$ data points. Hence in a computer programme, a good approach would be to calculate all the finite divided differences starting from the 1st order, then proceeding to the 2nd order and so on.

Exercise

Obtain estimates for $f(x)$ at $x = 0.5$ using Newton's divided-difference interpolating polynomials of the 1st and 2nd orders, given that $f(0) = 0$, $f\left(\frac{1}{3}\right) = \frac{1}{9}$ and $f(1) = 1$.

2.3.2 Lagrange Interpolating Polynomials

If you found Newton's divided-difference interpolating polynomials a bit unwieldy, then perhaps Lagrange interpolating polynomials would be up your alley as they are much more concise.

The Lagrange interpolating polynomial is a sum defined as:

$$f_n(x) = \sum_{i=0}^{n} L_i(x)\, y_i\,,$$

where

$$L_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\,.$$

Before we proceed, let's break down what this expression is saying. On the LHS, $f_n(x)$ is the value that we are interested in, that we are interpolating to obtain. The n in the subscript denotes the order of the interpolation, for instance if we are assuming a linear $(n = 1)$ or quadratic $(n = 2)$ function for the interpolation. x is the independent variable, for which we have (presumably) taken measurements at known points x_0, x_1, x_2, etc. The functional values y_0, y_1, y_2 etc. are thus considered to be known.

Now on the RHS, we have a sum of the products $L_i(x)$ and the known functional values y_i. So in other words, we can interpret this as a weighted sum of the known data points, where $L_i(x)$ is the weight applied to each data point. Note that the number of data points involved depend on n, the order of interpolation!

Let's now look at this 'weight function' $L_i(x)$. $\prod$ means 'product of', so this means that it is actually a product of several fractions with the general form $(x - x_j)\,/\,(x_i - x_j)$. We should have a total of $n + 1$ fractions multiplied together ($j = 0$ to $j = n$), except that if we had $j = i$ then one of those fractions would become undefined. Thus we leave out that particular case, as seen in the subscript of the $\prod$.

Moving in one step closer, we now look at the fractions themselves - both the numerator and the denominator are simple differences. Specifically, in the numerator we have the difference between the intermediate point of interest x and the j-th data point x_j; and in the denominator we have the difference between the i-th and $j-$th data points x_i and x_j. x_i

is significant in that it is the point for which this weighting function L_i is being calculated.

So as a whole what we have is a weighted sum of the known functional values, where each of the weights depend on how far away the point of interest x and the weighted data point x_i are from each of the other data points x_j.

Let's expand one of the Lagrange interpolating polynomials, to make it clearer. A Lagrange interpolating polynomial of the 1st order means using a linear interpolation, so $n = 1$:

$$f_1(x) = L_0(x)\, y_0 + L_1(x)\, y_1 .$$

$f_1(x)$ is what we want to find, while y_0 and y_1 are known. Using the definition earlier, we have:

$$L_0(x) = \prod_{j=0,\, j\neq 0}^{1} \frac{x - x_j}{x_0 - x_j} = \frac{x - x_1}{x_0 - x_1} ,$$

and

$$L_1(x) = \prod_{j=0,\, j\neq 1}^{1} \frac{x - x_j}{x_0 - x_j} = \frac{x - x_0}{x_1 - x_0} .$$

Therefore,

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 .$$

Compare this to what we obtained earlier, for our manual derivation of the linear relationship $f_1(x) = c_0 + c_1 x$.You should find that they are equivalent! :)

Exercise 1

Write out the Lagrange interpolating polynomial of the 2nd order, i.e. $n = 2$.

Exercise 2

Obtain estimates for $f(x)$ at $x = 0.5$ using Lagrange interpolating polynomials of the 1st and 2nd orders, given that $f(0) = 0$, $f\left(\frac{1}{3}\right) = \frac{1}{9}$ and $f(1) = 1$.

2.3.3 Spline Interpolation

In Excel if we choose the option to connect all the data points using straight lines, that would be an example of using splines. Specifically we are assuming linear (1st order) interpolations between every adjacent pair of data points. Sometimes it gives us a pretty good idea of the general trend of the data, but it is not necessarily the full and/or correct picture. For example consider if we had a set of three data points $(0, 0), (\pi/2, 1), (\pi, 0)$. You can see how using linear splines would make calculation fairly simple, but in terms of accuracy a quadratic function fit to all three points would probably be more accurate.

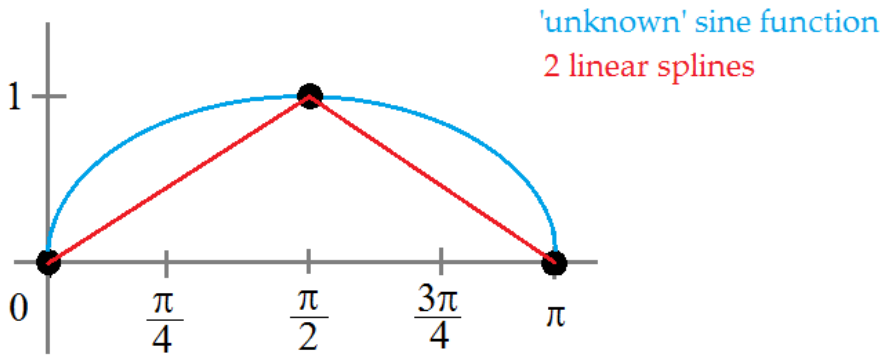


Figure 5: Illustration of linear spline interpolation, applied to an unknown function (sine curve).

So why would we do spline interpolation? Put simply, spline interpolation is when we use multiple lower order polynomials (not necessarily straight lines!) for small sections of the full data set. In other words, if we had a data set consisting of $n + 1$ points, instead of using a single n-th order polynomial to connect/use all the data points we instead use polynomials of orders less than n and apply them to adjacent data subsets. *Chapra & Canale* have a very good illustration of this:

As seen from Fig. 6, while increasing the order of the interpolation polynomial from 3 to 7 does qualitatively improve the approximation (blue line) to the step function (the black line), there is also a lot of oscillations being introduced as a result. This could lead to huge errors. On the other hand, the (linear) spline interpolation in the last row is obviously

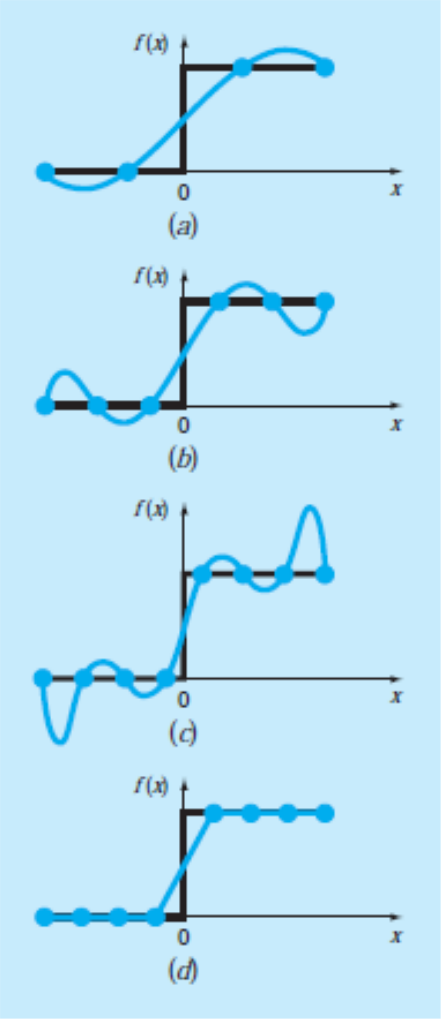


Figure 6: Illustration of spline interpolation taken from *Chapra & Canale*, Fig. 18.14 on pg 510, comparing polynomials of increasing order (top 3 rows) to spline interpolation (last row) when applied to a step function.

superior to the 7th order polynomial interpolation directly above it, even though they use the same data points. The reason is because the spline interpolation does not utilise all 8 of the data points together - there are instead at least 3 linear interpolations involved.

Linear Splines

In both the Excel example and the sine function example, each point is connected by a straight line to its neighbour. This is the easiest version - 1st order (linear) splines. For a

general group of data, these linear splines can be defined as follows:

$$\begin{aligned}
f(x) &= f(x_0) + m_0(x - x_0) && x_0 \leq x \leq x_1 \\
f(x) &= f(x_1) + m_1(x - x_1) && x_1 \leq x \leq x_2 \\
f(x) &= f(x_2) + m_2(x - x_2) && x_2 \leq x \leq x_3 \\
&\vdots && \vdots \\
f(x) &= f(x_{n-1}) + m_{n-1}(x - x_{n-1}) && x_{n-1} \leq x \leq x_n
\end{aligned}$$

In other words, we can define an approximation to the data using a set of linear expressions, each with slope $m_{i-1} \equiv (f(x_i) - f(x_{i-1})) / (x_i - x_{i-1})$. The function can thus be estimated at any point by identifying the interval within which the point of interest lies, and then applying the appropriate equation.

Looking at the linear spline approximation in Fig. 5, we can see straightaway that although the approximation is rather crude, it does give a fairly good idea of the function. However one immediate drawback is that the approximated function is *not smooth* – at each data point where two splines connect, there is a jump in slope. This means that we would have issues when trying to approximate the first derivative (let alone any higher derivatives) at these points.

Quadratic Splines

One way to overcome the issue of discontinuity in derivatives (because of the lack of smoothness when using linear splines, as mentioned previously) is to use higher order splines. Specifically, if we use quadratic splines then we can have continuity in the 1st order derivatives $(\partial f/\partial x)$ but not higher; and if we use cubic splines then we can have continuity in both 1st and 2nd order derivatives $(\partial f/\partial x)$ and $(\partial^2 f/\partial x^2)$ but not higher. Usually this is sufficient for our purposes, so cubic splines are most commonly used in practice.

Fig. 7 illustrates this concept - the spline function becomes visually smoother as the order of the splines increase. However note that the cubic spline function is *not the same* as the cubic interpolation function plotted in the same graph! The reason for this is that instead of applying a single cubic function to all 4 points (cubic interpolation function), we are still

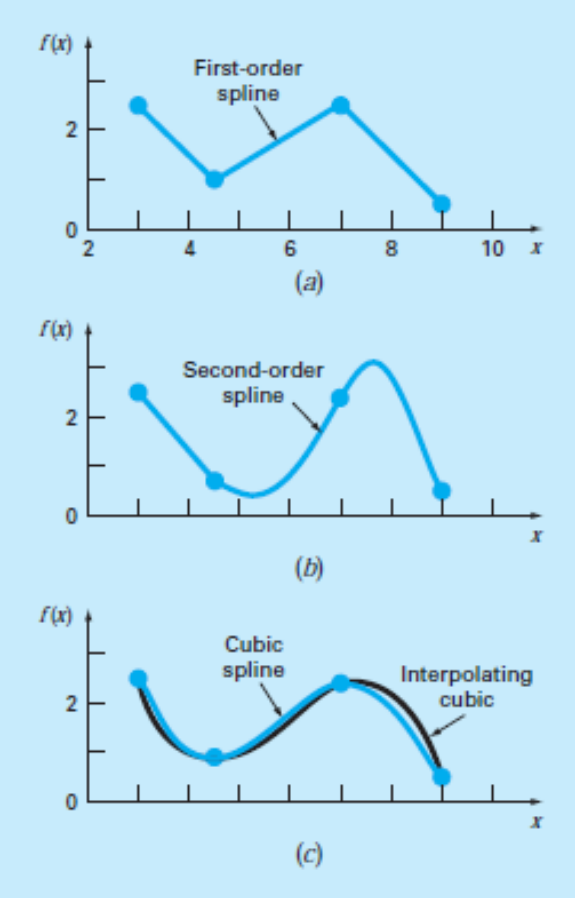


Figure 7: Illustration of spline interpolation taken from *Chapra & Canale*, Fig. 18.16 on pg512, comparing splines of increasing order (from top to bottom) as well as a cubic interpolation (last row) when applied to a set of 4 data points.

sticking to pairs of adjacent data points when we apply the spline functions. But – think about it – if we do this, then how does this resolve the issue of smoothness at the data points? This will become clearer when we go through the 'rules' of formulating the spline functions. For simplicity, we will look at quadratic splines first.

For quadratic splines, we want to have a 2nd order polynomial between each pair of data points. Thus the function in each interval $[x_{i-1},\, x_i]$ can be generally written as:

$$f_i(x) = a_i x^2 + b_i x + c_i \,.$$

If we have $n + 1$ points, we will have n intervals, and therefore a total of $3n$ unknown coefficients. We can determine these using these rules:

1. *The polynomial of each interval must pass through the data points defining it.*
   In other words, for an interval i between data points x_{i-1} and x_i, we will obtain 2 equations:

$$f_i(x_{i-1}) = a_i x_{i-1}^2 + b_i x_{i-1} + c_i$$
$$f_i(x_i) = a_i x_i^2 + b_i x_i + c_i$$

   Since we have n intervals, this gives us $2n$ equations.

2. *The first derivatives of adjacent functions must be equal at interior data points.*
   This is the condition we require to ensure continuity of the first derivative $\partial f/\partial x$. So at each interior point x_i we have:

$$\left.\frac{\partial f_i}{\partial x}\right|_{x_i} = \left.\frac{\partial f_{i+1}}{\partial x}\right|_{x_i}$$

$$\Rightarrow \quad 2a_i x_i + b_i = 2a_{i+1} x_i + b_{i+1}$$

   Thus we have an additional $n-1$ equations.

3. *Arbitrary decision: assume that the second derivative is zero at the first data point.*
   From the previous 2 rules, we have already obtained $3n-1$ equations, so we are one short. This last condition is usually obtained from any other information we have regarding the derivatives of the actual function. However if we have no other information, then the usual choice is to assume that $\partial^2 f_1/\partial x^2 = 0$ at x_0. This means that $a_1 = 0$, and the first 2 data points are connected by a straight line rather than a quadratic function like the other data points.

It may be easier to think of these rules as the same way that we determine the boundary conditions for say, a bending problem. The beam equation which describes the amount of deformation of a bending beam is 4th order, and often we have point loads and such applied at different locations along the beam, which means that we have to solve the same equation for each interval. However we always have to make sure that the deformation is continuous at those points (same as Rule 1 above), and that the slope itself is smooth (Rule 2 above). The equivalent of Rule 3 would be derived from what we know of the beam's exact conditions - whether it is a cantilever or a pin joint at the end etc.

Example

Let's revisit the sine function example, where our known data points are $(0, 0)$, $(\pi/2, 1)$ and $(\pi, 0)$. We shall now determine the set of quadratic splines for this small data set.

Because we have 3 data points, we will thus have 2 intervals and therefore only 2 quadratic splines:

$$\begin{aligned} f_1(x) &= a_1 x^2 + b_1 x + c_1, & 0 \le x \le \pi/2 \\ f_2(x) &= a_2 x^2 + b_2 x + c_2, & \pi/2 \le x \le \pi \end{aligned}$$

We have 6 unknown coefficients, for which Rules 1 and 2 above give us 5 equations with which to solve:

$$\begin{aligned} f_1(0) &= a_1(0)^2 + b_1(0) + c_1 \\ f_1\left(\frac{\pi}{2}\right) &= a_1\left(\frac{\pi}{2}\right)^2 + b_1\left(\frac{\pi}{2}\right) + c_1 \\ f_2\left(\frac{\pi}{2}\right) &= a_2\left(\frac{\pi}{2}\right)^2 + b_2\left(\frac{\pi}{2}\right) + c_2 \\ f_2(\pi) &= a_2(\pi)^2 + b_2(\pi) + c_2 \\ 2a_1\left(\frac{\pi}{2}\right) + b_1 &= 2a_2\left(\frac{\pi}{2}\right) + b_2 \end{aligned}$$

For the last condition, we can go with the usual as outlined in Rule 3, that is $\frac{\partial^2 f_1}{\partial x^2} = 0$ at the first data point ($x = 0$). However if we happen to remember that $\frac{\pi}{2}$ is a local maximum - so the first derivative there is zero - then we can use that as our final condition:

$$\left.\frac{\partial f_1}{\partial x}\right|_{\pi/2} = 0\,.$$

Thus our full set of equations, after simplifying, are now:

$$\begin{aligned} 0 &= c_1 \\ 1 &= a_1\frac{\pi^2}{4} + b_1\frac{\pi}{2} + c_1 \\ 1 &= a_2\frac{\pi^2}{4} + b_2\frac{\pi}{2} + c_2 \\ 0 &= a_2\pi^2 + b_2\pi + c_2 \\ a_1\pi + b_1 &= 0 \\ a_2\pi + b_2 &= 0 \end{aligned}$$

Substituting the 1st and 5th equations into the 2nd, we get $a_1 = -\frac{4}{\pi^2}$, $b_1 = \frac{4}{\pi}$ and $c_1 = 0$. Substituting the 6th equation into the 4th gives us $c_2 = 0$, which in conjunction with the 3rd and 6th equations leads to $a_2 = -\frac{4}{\pi^2}$ and $b_2 = \frac{4}{\pi}$. The 2 quadratic splines are thus:

$$\begin{aligned} f_1(x) &= -\tfrac{4}{\pi^2}x^2 + \tfrac{4}{\pi}x + c_1, && 0 \le x \le \pi/2 \\ f_2(x) &= -\tfrac{4}{\pi^2}x^2 + \tfrac{4}{\pi}x + c_2, && \pi/2 \le x \le \pi \end{aligned}$$

In this case, it turns out that the same quadratic spline can fit both intervals, so we can approximate this 3-point data set using a single quadratic function

$$f(x) = -\frac{4}{\pi^2}x^2 + \frac{4}{\pi}x + c_1 \,.$$

Cubic Splines

Given the procedure above for quadratic splines, can you logically extend that to cubic splines? In this case, we have a 3rd order polynomial between each pair of data points, for which the general equation is:

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \,.$$

If we have $n + 1$ points, we will have n intervals, and therefore a total of $4n$ unknown coefficients. The 'rules' will thus be:

1. *The polynomial of each interval must pass through the data points defining it.*

   In other words, for an interval i between data points x_{i-1} and x_i, we will obtain 2 equations:

   $$\begin{aligned} f_i\left(x_{i-1}\right) &= a_i x_{i-1}^3 + b_i x_{i-1}^2 + c_i x_{i-1} + d_i \\ f_i\left(x_i\right) &= a_i x_i^3 + b_i x_i^2 + c_i x_i + d_i \end{aligned}$$

   Since we have n intervals, this gives us $2n$ equations.

2. *The first derivatives of adjacent functions must be equal at interior data points.*

   This is the condition we require to ensure continuity of the first derivative $\partial f/\partial x$. So at each interior point x_i we have:

   $$\left.\frac{\partial f_i}{\partial x}\right|_{x_i} = \left.\frac{\partial f_{i+1}}{\partial x}\right|_{x_i}$$

   $$\Rightarrow \qquad 3a_i x_i^2 + 2b_i x_i + c_i = 3a_{i+1} x_i^2 + 2b_{i+1} x_i + c_{i+1}$$

Thus we have an additional $n-1$ equations.

3. *The **second** derivatives of adjacent functions must be equal at interior data points.*
   This is the condition we require to ensure continuity of the second derivative $\partial^2 f/\partial x^2$. So at each interior point x_i we also have:

$$\left.\frac{\partial f_i^2}{\partial x^2}\right|_{x_i} = \left.\frac{\partial f_{i+1}^2}{\partial x^2}\right|_{x_i}$$

$$\Rightarrow \qquad 6a_i x_i + 2b_i = 6a_{i+1}x_i + 2b_{i+1}$$

   Thus we have yet another $n-1$ equations.

4. *Arbitrary decision: assume that the second derivative is zero at the first and last data points.*
   From the previous 2 rules, we have already obtained $4n-2$ equations, so we are two short. As with the cubic splines, these last conditions are usually obtained from any other information we have regarding the derivatives of the actual function. However if we have no other information, then the usual choice is to assume that $\partial^2 f/\partial x^2 = 0$ at both x_0 and x_n. This means that $a_1 = 0$ and $a_n = 0$, and both the first 2 and last 2 data points are connected by straight lines rather than a cubic function like the other data points.

2.3.4 Multidimensional Interpolation

Logically you may expect that multidimensional interpolation could be simply an extension of the 1D interpolation methods we covered previously. Indeed that is probably the most intuitive way to consider it, and here we will outline one of the basic approaches – **bilinear interpolation**.

Let's consider a function $z = f(x, y)$ which is dependent on two variables x and y. If we wanted to do an interpolation it would thus be within a range of both x and y – which means that we would be looking at a surface connecting 4 points (x_1, y_1, z_{11}), (x_2, y_1, z_{21}), (x_1, y_2, z_{12}) and (x_2, y_2, z_{22}). This is illustrated in Fig. 8 with the equivalent in 1D for comparison. The next step would be the choice of function: the simplest would be a linear function in both x and y, and thus this function is what we call **bilinear**.

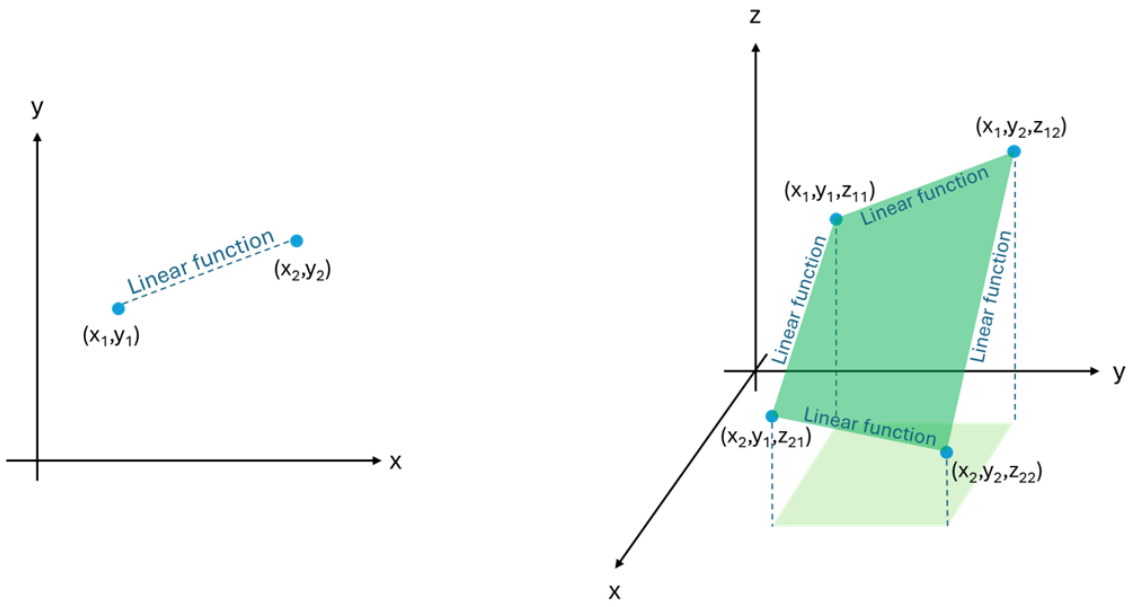


Figure 8: Illustration of 1D linear interpolation and bilinear interpolation.

Bilinear interpolation is, at its core, still the linear interpolation we are familiar with. Just that we apply it three times:

- at $y = y_1$, interpolate between x_1 and x_2
- at $y = y_2$, interpolate between x_1 and x_2
- using the above 2 expressions, interpolate between y_1 and y_2

For instance if we used Lagrange interpolating functions, this would be:

$$\begin{aligned} f(x, y_1) &= \frac{x - x_2}{x_1 - x_2} z_{11} + \frac{x - x_1}{x_2 - x_1} z_{21} \\ f(x, y_2) &= \frac{x - x_2}{x_1 - x_2} z_{12} + \frac{x - x_1}{x_2 - x_1} z_{22} \\ f(x, y) &= \frac{y - y_2}{y_1 - y_2} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \end{aligned}$$

We could, of course, combine all of it into a single equation:

$$\begin{aligned} f(x, y) = &\left(\frac{y - y_2}{y_1 - y_2}\right)\left(\frac{x - x_2}{x_1 - x_2}\right) z_{11} + \left(\frac{y - y_2}{y_1 - y_2}\right)\left(\frac{x - x_1}{x_2 - x_1}\right) z_{21} + \left(\frac{y - y_1}{y_2 - y_1}\right)\left(\frac{x - x_2}{x_1 - x_2}\right) z_{12} \\ &+ \left(\frac{y - y_1}{y_2 - y_1}\right)\left(\frac{x - x_1}{x_2 - x_1}\right) z_{22} \,. \end{aligned}$$

Exercise

Using the 4 data points below, interpolate for the functional value at the centre.

$z = f(x, y)$	$y_1 =$ $y_2 =$
$x_1 =$ $x_2 =$	

Here we interpolated the first two times holding y constant, but we can of course do it for x first; the combined equation would be identical so it is simply a matter of convenience. Note also that although we only introduced bilinear interpolation here, it is possible to use higher-order polynomials or splines for the interpolation as well as extend to three dimensions.

2.4 Fourier Approximation

Previously we were mostly concentrating on polynomials for both the least-squares regression and interpolation. We will now consider sinusoidal functions, which can also represent a lot of the relationships in engineering systems.

2.4.1 Sinusoidal Curve-Fitting

A general equation for a sinusoidal function is

$$f(t) = A + C\cos(\omega t + \theta) \ ,$$

where A is the time-averaged value (or the average height above the horizontal time axis), C and ω are the amplitude and angular frequency of the sinusoid respectively, and θ is the phase angle or phase shift (the angle in radians at which the function is leading a standard trigonometrical function of similar frequency but with $\theta = 0$. Fig. 9 illustrates these four parameters.

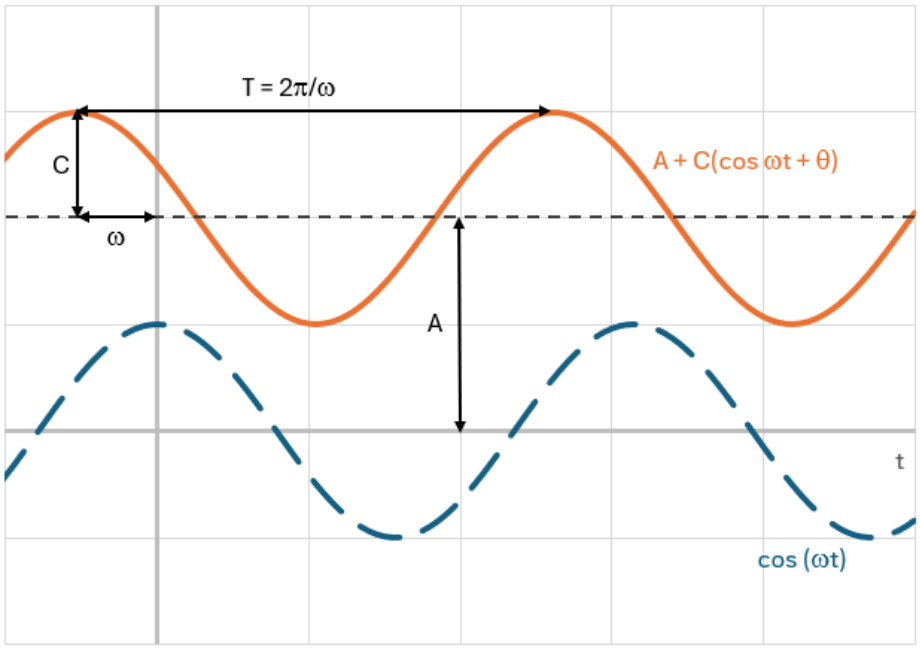


Figure 9: General sinusoid $A + C\cos(\omega t + \theta)$ (orange solid line) versus $\cos(\omega t)$ (blue dashed line).

An equivalent and perhaps more useful form is

$$f(t) = A_0 + A_1 \cos(\omega t) + B_1 \sin(\omega t)\,,$$

which still has four main parameters but is now a linear sum of a cosine and sine function. In this form, it is possible to use the least-squares regression concepts covered in Section 2.2.

The sum of squared-residuals would be:

$$S_r = \sum_i^N \left[y_i - (A_0 + A_1 \cos(\omega t) + B_1 \sin(\omega t))\right]^2,$$

and the set of simultaneous equations that we obtain after setting all the relevant derivatives to zero, when expressed in matrix form, is:

$$\begin{bmatrix} \sum y_i \\ \sum y_i \cos(\omega t_i) \\ \sum y_i \sin(\omega t_i) \end{bmatrix} = \begin{bmatrix} N & \sum \cos(\omega t_i) & \sum \sin(\omega t_i) \\ \sum \cos(\omega t_i) & \sum \cos^2(\omega t_i) & \sum \cos(\omega t_i)\sin(\omega t_i) \\ \sum \sin(\omega t_i) & \sum \cos(\omega t_i)\sin(\omega t_i) & \sum \sin^2(\omega t_i) \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ B_1 \end{bmatrix},$$

where all the $\sum_i^N$ have been written as $\sum$ for overall clarity. We can then proceed to solve for the unknown coefficients A_0, A_1 and B_1 as per usual.

However if we have the luxury of choosing the set of data points such that they are equally spaced over a single period of the sinusoid (i.e. N data points with spacing T/N; the $(N+1)$-th point is not included because it's assumed to be the same as the 1st point), then the following relationships would be valid:

$$\begin{aligned} \frac{\sin(\omega t_i)}{N} &= 0\,, & \qquad \frac{\cos(\omega t_i)}{N} &= 0\,, \\ \frac{\sin^2(\omega t_i)}{N} &= \frac{1}{2}\,, & \qquad \frac{\cos^2(\omega t_i)}{N} &= \frac{1}{2}\,, \\ \frac{\cos(\omega t_i)\sin(\omega t_i)}{N} &= 0 \end{aligned}$$

The matrix equation would thus be simplified to:

$$\begin{bmatrix} \sum y_i \\ \sum y_i \cos(\omega t_i) \\ \sum y_i \sin(\omega t_i) \end{bmatrix} = \begin{bmatrix} N & 0 & 0 \\ 0 & \frac{N}{2} & 0 \\ 0 & 0 & \frac{N}{2} \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ B_1 \end{bmatrix},$$

and therefore

$$\begin{aligned}
\begin{bmatrix} A_0 \\ A_1 \\ B_1 \end{bmatrix} &= \begin{bmatrix} N & 0 & 0 \\ 0 & \frac{N}{2} & 0 \\ 0 & 0 & \frac{N}{2} \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i \\ \sum y_i \cos(\omega t_i) \\ \sum y_i \sin(\omega t_i) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{N} & 0 & 0 \\ 0 & \frac{2}{N} & 0 \\ 0 & 0 & \frac{2}{N} \end{bmatrix} \begin{bmatrix} \sum y_i \\ \sum y_i \cos(\omega t_i) \\ \sum y_i \sin(\omega t_i) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{N} \sum y_i \\ \frac{2}{N} \sum y_i \cos(\omega t_i) \\ \frac{2}{N} \sum y_i \sin(\omega t_i) \end{bmatrix} .
\end{aligned}$$

Exercise

The table below contains data from a single cycle of the function $y = 2 + \cos t - \sin t$. Does the least-squares fit for $\omega = 1$ match the original function?

t_i	y_i	$y_i \cos(\omega t_i)$	$y_i \sin(\omega t_i)$
0	3		
$\frac{\pi}{2}$	1		
π	1		
$\frac{3\pi}{2}$	3		
$\sum$	8		

Given that the above analysis was for a general sinusoid $f(t) = A_0 + A_1 \cos(\omega t) + B_1 \sin(\omega t)$, you might wonder if we could extend this further to periodic functions that were not quite the usual sinusoid. Fig. 10 illustrates how summing harmonic waves gradually approaches a rectangular wave form. In that case, the function for a periodic (but not necessarily sinusoidal) function could be described by:

$$f(t) = A_0 + A_1 \cos(\omega t) + B_1 \sin(\omega t) + A_2 \cos(2\omega t) + B_2 \sin(2\omega t) + \cdots + A_m \cos(m\omega t) + B_m \sin(m\omega t) .$$

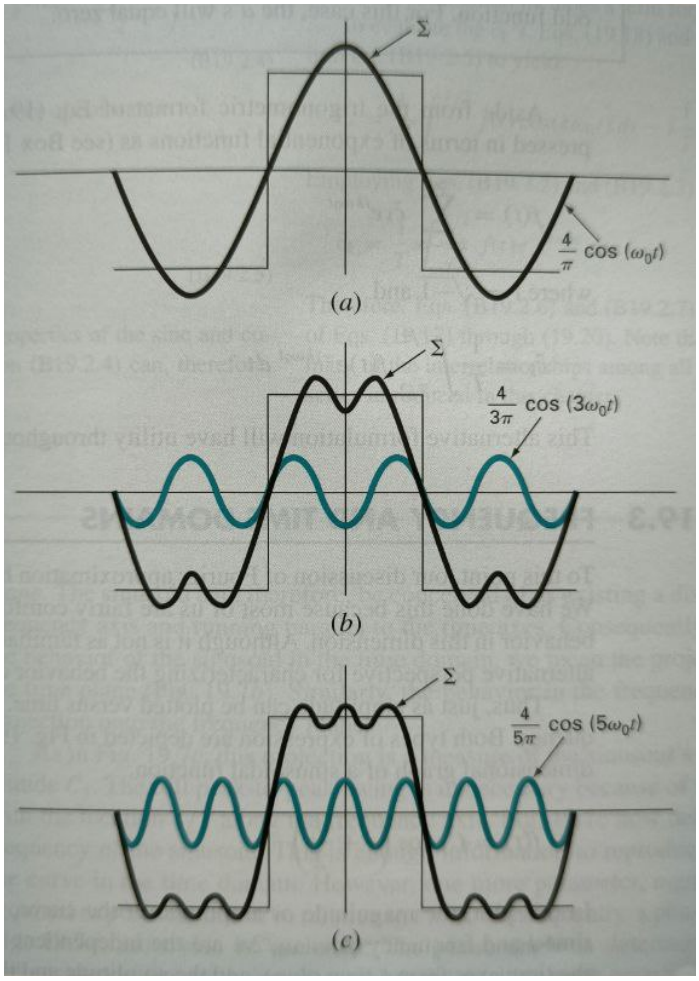


Figure 10: Illustration showing the summation of (a) one, (b) two and (c) three sinusoidal functions increasingly approximate a rectangular waveform, from *Chapra & Canale*, Fig. 19.6 on pg533.

Applying the least-squares regression analysis similarly we find that (again for equally-spaced data) the coefficients are:

$$\begin{aligned} A_0 &= \frac{1}{N}\sum y_i\,, \\ A_k &= \frac{2}{N}\sum y_i \cos(k\omega t_i)\,, \\ B_k &= \frac{2}{N}\sum y_i \sin(k\omega t_i)\,, \end{aligned}$$

for $k = 1, 2, ..., m$.

2.4.2 Fourier Transforms

In the previous section we discussed periodic functions. But what about functions that are not periodic? We can achieve such a function by allowing the period T to approach infinity, such that the function never repeats. You can imagine that in that case, we would need an infinite series of harmonic waves to achieve that signal. And considering that frequency $\omega = T/2\pi$, when $T \to \infty$ then $\omega \to 0$. Thus altogether, a non-periodic function in the time domain would be made up of what looks like a continuous function in the frequency domain.

So first let us consider a continuous periodic function. As demonstrated by Fourier, an arbitrary periodic function can be modelled using an *infinite* series of sinusoids with harmonic frequencies. Specifically a function with period T can be written as a **continuous Fourier series** as follows:

$$\begin{aligned} f(t) &= a_0 + a_1 \cos(\omega t) + b_1 \sin(\omega t) + a_2 \cos(2\omega t) + b_2 \sin(2\omega t) + \dots \\ &= a_0 + \sum_{k=1}^{\infty} \left[a_k \cos(k\omega t) + b_k \sin(\omega t) \right], \end{aligned}$$

where $\omega = 2\pi/T$ is the **fundamental frequency** and all the multiples $(k\omega)$ are its **harmonics**. Similarly to the previous section, the coefficients of the Fourier series are:

$$\begin{aligned} a_0 &= \frac{1}{T} \int_0^T f(t) dt \,, \\ a_k &= \frac{2}{T} \int_0^T f(t) \cos(k\omega t)\, dt \,, \text{ and} \\ b_k &= \frac{2}{T} \int_0^T f(t) \sin(k\omega t)\, dt \,. \end{aligned}$$

By expressing the sine and cosine functions in exponential form and redefining the coefficients (the full working can be found in *Chapra & Canale* p535), the Fourier series can be written in the exponential form:

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{ik\omega t} \,,$$

where

$$c_k = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-ik\omega t} dt \,.$$

Recall that in the complex form $Re^{i\theta}$, R is the magnitude of the wave which has phase $\theta = \omega t$. So the c_k here is the magnitude of the $k-$th sinusoidal wave (which has angular frequency of $k\omega$) in the Fourier series.

Now if $T \to \infty$ (we will not include the full working here), the Fourier series becomes

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(i\omega) e^{i\omega t} d\omega \,,$$

and the coefficients are a continuous function of ω:

$$F(i\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \,.$$

We can see that $F(i\omega)$ is also the magnitude of the sinusoidal waves making up the function, except that now it has itself become a continuous function with respect to angular frequency ω.

Mathematically $F(i\omega)$ is the **Fourier integral** or **Fourier transform** of $f(t)$, and conversely $f(t)$ is the **inverse Fourier transform** of $F(i\omega)$. Together, they are called the **Fourier transform pair**. We can think of them as the same non-periodic function but in either the time domain or frequency domain.

Discrete Fourier Transform (DFT)

Now that we have introduced the Fourier transform pair for non-periodic functions, let us now look at how this would work for a set of data - in other words, we only have discrete values of $f_n = f(t_n)$ available, but we want to look at the data in the frequency domain. In this situation, we would utilise the **discrete Fourier transform** (aka DFT).

For a data set of $N + 1$ points equally-spaced in time, (subscripts n=0, 1, 2, ... N-1, N), we use the first N points in the DFT (the last point at $t_n = T$ must be excluded[1]):

$$F_k = \sum_{n=0}^{N-1} f_n e^{-ik\omega n} \qquad k = 0, \ldots, N-1 \,,$$

to generate a set of discrete points in the frequency domain with $\omega = 2\pi/N$. Specifically F_k is the magnitude of the k-th sinusoid with angular frequency $k\omega$. The corresponding

[1]Detailed rationale can be found in Ramirez, R. W.., *The FFT, Fundamentals and Concepts*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

inverse Fourier transform is:

$$f_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k e^{ik\omega n} \qquad n = 0, \ldots, N-1\,.$$

Note that the coefficient $1/N$ in the inverse transform is actually just a scale factor that can appear in either transform but not both. If it were included in the Fourier transform, then $F_0 = \sum f_n/N$ which is mathematically the same as A_0 in our least-squares regression analysis for sinusoidal curve-fitting.

Looking at both transforms, we observe that there are N operations needed for each of the N data points, so a total of N^2 operations are involved. Hence a computer algorithm is strongly recommended.

Example

Perform a DFT for the data provided below.

t	0	$\frac{\pi}{2}$	π	$\frac{3\pi}{2}$	2π
$f(t)$	0	1	1	1	0

Here there are 5 ($= N + 1$) data points, which means that the fundamental frequency $\omega = 2\pi/N = \pi/2$. Recall that we do not use the last data point, so we will only be using the first 4 data points i.e. subscripts $n = 0$, 1, 2, 3.

We use DFT to generate a corresponding data set $\{F_0, F_1, F_2, F_3\}$ in the frequency domain:

$$\begin{aligned}
F_0 &= \sum_{n=0}^{N-1} f_n e^0 = f_0 + f_1 + f_2 + f_3 = 3\,,\\
F_1 &= \sum_{n=0}^{N-1} f_n e^{-i\omega n} = f_0 e^0 + f_1 e^{-i\omega} + f_2 e^{-i2\omega} + f_3 e^{-i3\omega} = e^{-i\pi/2} + e^{-i\pi} + e^{-i3\pi/2} = -1\,,\\
F_2 &= \sum_{n=0}^{N-1} f_n e^{-i2\omega n} = f_0 e^0 + f_1 e^{-i2\omega} + f_2 e^{-i4\omega} + f_3 e^{-i6\omega} = e^{-i\pi} + e^{-i2\pi} + e^{-i3\pi} = -1\,,\\
F_3 &= \sum_{n=0}^{N-1} f_n e^{-i3\omega n} = f_0 e^0 + f_1 e^{-i3\omega} + f_2 e^{-i6\omega} + f_3 e^{-i9\omega} = e^{-i3\pi/2} + e^{-i3\pi} + e^{-i9\pi/2} = -1\,.
\end{aligned}$$

Fig. 11 shows the outcome of our DFT calculation in both the frequency (left) and time (right) domains.

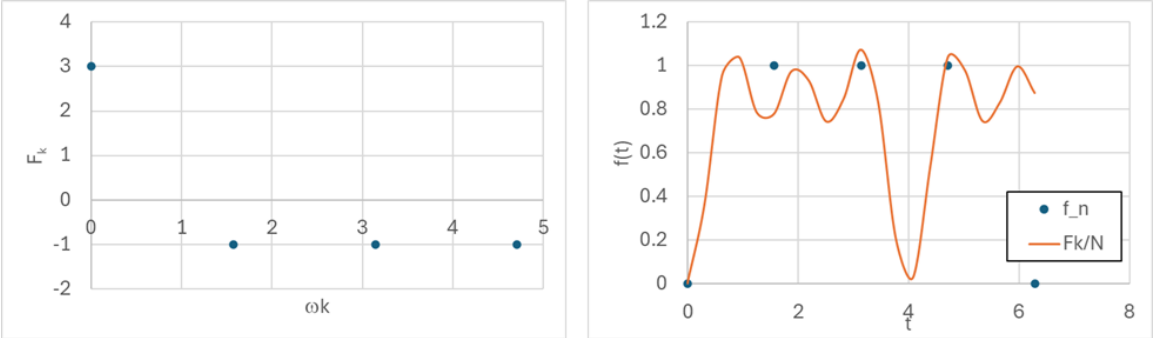


Figure 11: Left: F_k values plotted in the frequency domain; right: Fourier approximation obtained using $\frac{F_k}{N}$ (solid line) with the original data set (circles).

We can see that although only 4 data points were used (based on the data, it could be a rectangular function), the sum of the 4 sinusoids with coefficients determined by the DFT was not too far off.

Fast Fourier Transform (FFT)

The **Fast Fourier Transform** (aka FFT) was developed as a more economical version of DFT - instead of N^2 operations, it requires approximately $N \log_2 N$ operations only. It may not be much faster for small data sets (for $N = 10$, FFT requires 33 operations compared to DFT's 100) but at moderate sizes (for $N = 50$, FFT requires ≈280 operations while DFT needs 2500) the time saving is significant, not to mention for large data sets.

Basically, FFT works by decomposing or 'decimating' the original DFT of N points, into DFTs of successively smaller lengths. There are two general techniques: 'decimation-in-time' (e.g. **Cooley-Tukey algorithm**) and 'decimation-in-frequency' (e.g. **Sande-Tukey algorithm**). We will first introduce the Sande-Tukey algorithm.

Note that although this technique works with a data set of any length N, it is best (and easiest) if N is a power of 2, i.e. $N = 2^M$, where M is a non-zero positive integer. This is because we will keep dividing the data set in half.

Recall that the DFT is:

$$F_k = \sum_{n=0}^{N-1} f_n e^{-ik\omega n} \qquad k = 0, \ldots, N-1\,.$$

We will split this summation into half, one from $n = 0$ to $n = \frac{N}{2} - 1$, and the other from

$n = \frac{N}{2}$ to $n = N - 1$:

$$\begin{aligned} F_k &= \sum_{n=0}^{N/2-1} f_n e^{-ik\omega n} + \sum_{n=N/2}^{N-1} f_n e^{-ik\omega n} \\ &= \sum_{n=0}^{N/2-1} f_n e^{-ik\omega n} + \sum_{n=0}^{N/2-1} f_{n+N/2}\, e^{-ik\omega(n+N/2)} \\ &= \sum_{n=0}^{N/2-1} \left(f_n e^{-ik\omega n} + f_{n+N/2}\, e^{-ik\omega(n+N/2)} \right) \\ &= \sum_{n=0}^{N/2-1} \left(f_n + f_{n+N/2}\, e^{-ik\omega(N/2)} \right) e^{-ik\omega n}\,. \end{aligned}$$

Recall that $\omega = 2\pi/N$, so

$$e^{ik\omega(N/2))} = e^{ik\pi} = \begin{cases} 1 & k \text{ is even} \\ -1 & k \text{ is odd} \end{cases}$$

Thus we can write out two versions of the F_k expression depending on whether k is even or odd, as follows:

$$F_{\text{even } k} \quad = F_{2j} \quad = \sum_{n=0}^{N/2-1} \left(f_n + f_{n+N/2} \right) e^{-i2j\omega n} = \sum_{n=0}^{N/2-1} \left(f_n + f_{n+N/2} \right) e^{-ij\left(\frac{2\pi}{N/2}\right)n}$$

$$F_{\text{odd } k} \quad = F_{2j+1} \quad = \sum_{n=0}^{N/2-1} \left(f_n - f_{n+N/2} \right) e^{-i(2j+1)\omega n} = \sum_{n=0}^{N/2-1} \left(f_n - f_{n+N/2} \right) e^{-ij\left(\frac{2\pi}{N/2}\right)n}\, e^{-i\left(\frac{2\pi}{N}\right)n}$$

for $j = 0, 1, 2, ..., \frac{N}{2} - 1$.

Comparing to the original DFT, we can observe that:

- for even k, it is the DFT of a function $g_n = f_n + f_{n+N/2}$
- for odd k, it is the DFT of a function $h_n = \left(f_n - f_{n+N/2} \right) e^{-i(2\pi/N)n}$.

and the length is now halved i.e. $n = 0, 1, \ldots, \frac{N}{2} - 1$. The exponential term in the odd k expression is a weight W^n, sometimes called a 'twiddle factor'.

So essentially, instead of one DFT of length N (i.e. DFT on a data set of N points), we now have 2 DFTs of length $N/2$. Thus the number of computations we have to do will drop from N^2 to $2 \times \left(\frac{N}{2}\right)^2$ plus some corresponding to the formation of g_n and h_n.

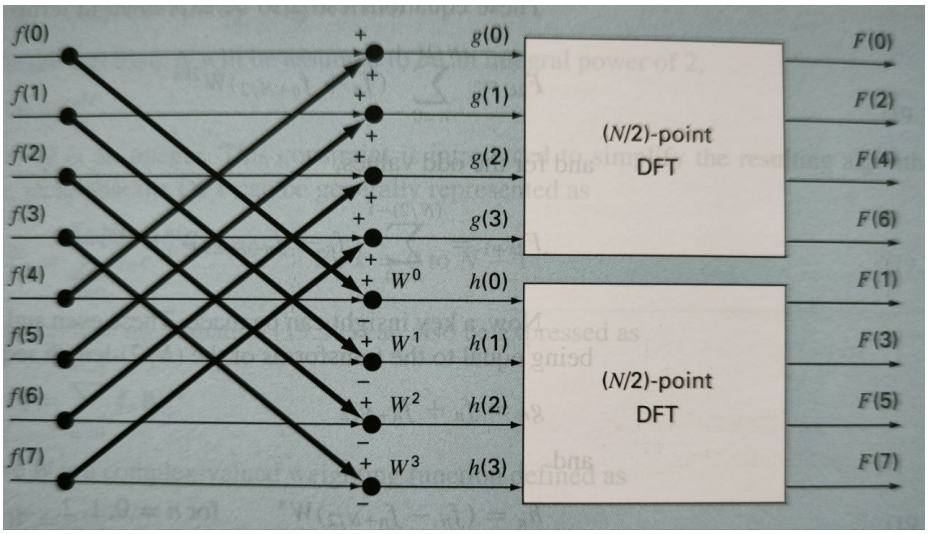


Figure 12: Flowchart showing first stage of decomposition of an 8-point DFT using Sande-Tukey algorithm, from *Chapra & Canale*, Fig. 19.15 on pg546.

Fig. 12 shows this decomposition for an 8-point DFT. The g_n and h_n functions are determined first, and then two DFTs are performed (one for the set of g_n and the other for the set of h_n). Note that, due to splitting up the odd and even k, the output points are no longer in sequence (for instance if we did the DFT for g_n first followed by the DFT for h_n, we will obtain the output for even k before odd k).

As mentioned in the beginning we can continue with this decomposition and split the data sets further until we are satisfied and then conduct the required number of DFTs. This process is illustrated in Fig. 13, which includes and continues the decomposition for the same 8-point DFT in the previous figure. Observe that the output sequence is further scrambled compared to before.

Fortunately, the scrambling is easy to deal with - if the subscripts are translated into binary form (e.g. 0 is 000, 1 is 001, 2 is 010 and so on), then upon reversing the order (e.g. 011 $\rightarrow$ 110) and translating back, the order will be corrected. This process is called **bit reversal**.

The Cooley-Tukey algorithm, as mentioned earlier in this section, is a 'decimation-in-time' technique. Here the algorithm is reversed compared to the Sande-Tukey algorithm: the data set is split into odd- and even-numbered points first and then the decomposition is carried out; the final output will be ordered correctly. This is illustrated in Fig. 14; ob-

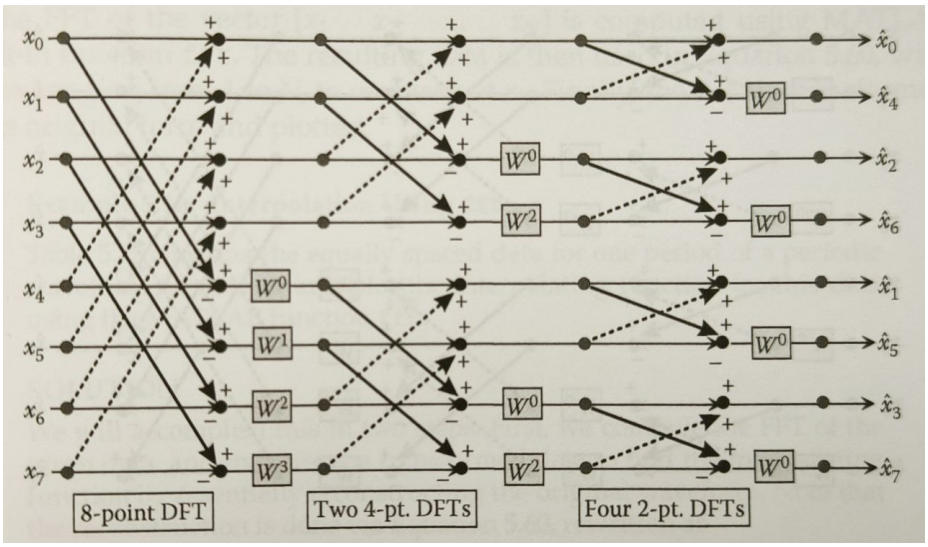


Figure 13: Flowchart showing full decomposition of an 8-point DFT using Sande-Tukey algorithm, from *Esfandiari*, Fig. 5.30 on pg237. Observe that the initially ordered data points (x_0 to x_7) are scrambled at the end of the decomposition.

serve how the input data is scrambled in the same way the output from the Sande-Tukey algorithm was.

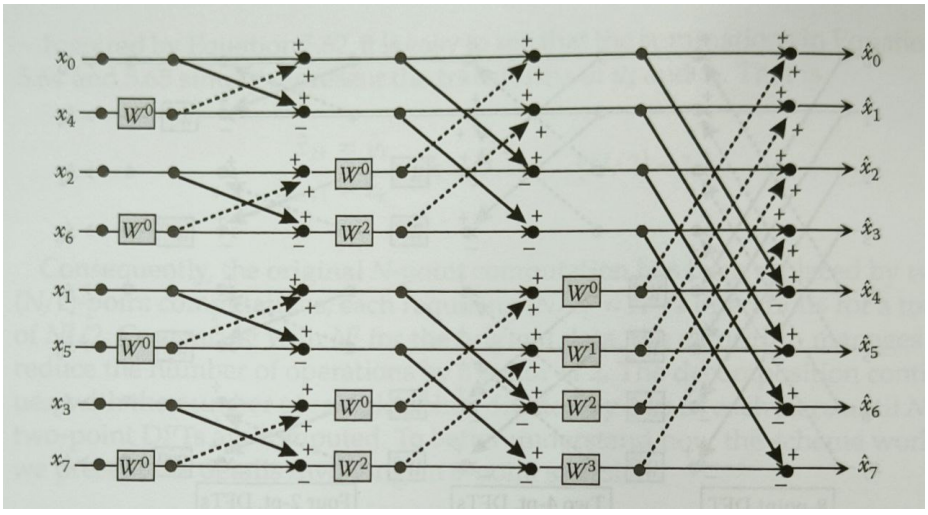


Figure 14: Flowchart showing full decomposition of an 8-point DFT using Cooley-Tukey algorithm, from *Esfandiari*, Fig. 5.31 on pg238. Observe that the data points are initially scrambled but emerge ordered at the end.

Example

Let's use the same example from the previous section on DFT, but instead perform an FFT using the Sande-Tukey algorithm.

t	0	$\frac{\pi}{2}$	π	$\frac{3\pi}{2}$	2π
$f(t)$	0	1	1	1	0

Here the original DFT is of length 4 (we don't use the last data point). So we expect to perform the decomposition twice.

At the first stage, $N = 4$. Thus we calculate the functions g_n and h_n for $n = 0, 1$:

$$\begin{aligned} g_0 &= f_0 + f_2 = 1 & h_0 &= (f_0 - f_2)\, e^{-i(2\pi/4)0} = -1 \\ g_1 &= f_1 + f_3 = 2 & h_1 &= (f_1 - f_3)\, e^{-i(2\pi/4)1} = 0 \end{aligned}$$

This is the first decomposition. Next we decompose each of the two sets further. For the set of g_n (corresponding to F_0 and F_2) we calculate functions gg_n (corresponding to F_0) and hg_n (corresponding to F_2), where now $n = 0$ only. Note that now $N = 2$.

$$gg_0 = g_0 + g_1 = 3 \qquad hg_0 = (g_0 - g_1)\, e^{-i(2\pi/2)0} = -1$$

For the set of h_n (corresponding to F_1 and F_3) we calculate functions gh_n (corresponding to F_1) and hh_n (corresponding to F_3), where again $n = 0$ only:

$$gh_0 = h_0 + h_1 = -1 \qquad hh_0 = (h_0 - h_1)\, e^{-i(2\pi/2)0} = -1$$

We can do our DFTs now, but since there is only one term in each data set, then $F_k = f_0 e^{-ik\omega 0} = f_0$. In other words, the values we obtained at the final round of decomposition are already what we want. That is,

$$\begin{aligned} F_0 &= gg_0 = 3 \\ F_2 &= hg_0 = -1 \\ F_1 &= gh_0 = -1 \\ F_3 &= hh_0 = -1 \end{aligned}$$

which is the same as we got for the DFT previously. Here the order is quite easy to unscramble, so we will not expound on it. You can, of course, try out the bit reversal for yourself (gg_0 is point 00, hg_0 is point 01 and so on).

2.5 Summary

Least-squares regression is a method for **fitting a curve to a set of data**. The basic idea is to determine the set of coefficients (in the predicted curve) that result in the smallest sum of squared-residuals. Whether this is for a linear, nonlinear, polynomial, or multi-variable relationship, the general procedure remains the same:

1. Determine form of predicted relationship (eyeballing the data, or from some previous reasoning).
2. Linearise the relationship either by manipulation or using 1st order Taylor series (Gauss-Newton method) if needed.
3. Write out the expression for sum of squared residuals, S_r.
4. Differentiate S_r with respect to all the unknown coefficients a_0, a_1, a_2 etc.
5. Set the derivatives to zero.
6. Solve the resulting simultaneous equations (e.g. using Gaussian elimination).
7. Plug the now-determined coefficients back into predicted relationship $\Rightarrow$ best-fit curve.

If using a computer programme, then the matrix formulation would allow an easy way of obtaining the unknown coefficients. For a relationship of order p and a data set of N points, the coefficient vector would be:

$$\mathrm{A} = \left(\mathrm{Z}^T\mathrm{Z}\right)^{-1}\left(\mathrm{Z}^T\mathrm{Y}\right) ,$$

where

$$\mathrm{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} , \quad \mathrm{Z} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & \dots & x_2^p \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^p \end{bmatrix} \quad \text{and} \quad \mathrm{A} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_p \end{bmatrix} .$$

For the Gauss-Newton method, it would be

$$\Delta\mathrm{A} = \left(\mathrm{Z}^T\mathrm{Z}\right)^{-1}\left(\mathrm{Z}^T\mathrm{Y}\right) ,$$

where

$$\mathrm{Y} = \begin{bmatrix} y_1 - f(x_1) \\ y_2 - f(x_2) \\ \vdots \\ y_N - f(x_N) \end{bmatrix}, \quad \mathrm{Z} = \begin{bmatrix} \frac{\partial f(x_1)}{\partial a_0} & \frac{\partial f(x_1)}{\partial a_1} & \cdots & \frac{\partial f(x_1)}{\partial a_p} \\ \frac{\partial f(x_2)}{\partial a_0} & \frac{\partial f(x_2)}{\partial a_1} & \cdots & \frac{\partial f(x_2)}{\partial a_p} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f(x_N)}{\partial a_0} & \frac{\partial f(x_N)}{\partial a_1} & \cdots & \frac{\partial f(x_N)}{\partial a_p} \end{bmatrix}_j \quad \text{and} \quad \Delta\mathrm{A} = \begin{bmatrix} \Delta a_0 \\ \Delta a_1 \\ \vdots \\ \Delta a_p \end{bmatrix}.$$

For interpolation, we do not particularly require the relationship describing the data; rather we want to know either (1) the **value at a specific point**, or (2) the **intermediate point which corresponds to a given value**. What we have covered here is polynomial interpolation, for which the typical procedure is:

1. Choose the order n of the polynomial interpolation function $f_n(x)$.
2. Choose the $n+1$ data points with which to define the interpolation function.
3. Obtain a set of $n+1$ simultaneous equations by substituting the chosen data points (x_i, y_i) into the function.
4. Solve for the $n+1$ unknown coefficients c_0, c_1, ... c_n.
5. Solve for the item of interest, with the now-fully-determined $f_n(x)$.

If we have a high order n, we can consider using Newton's divided-difference or Lagrange interpolating polynomials to shortcut the process. Instead of steps 3 and 4 above, we use the data points to evaluate the coefficients in the chosen interpolating polynomial.

For Newton's divided-difference interpolating polynomials, we need to calculate the finite divided differences from 1st order to n-th order:

$$\begin{aligned} b_0 &= f_n(x_1) = y_1\,, \\ b_1 &= f[x_2, x_1] = \frac{y_2 - y_1}{x_2 - x_1}\,, \\ b_2 &= f[x_3, x_2, x_1] = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1}\,, \\ &\vdots \\ b_n &= f[x_{n+1}, x_n, x_{n-1}, \ldots, x_1] = \frac{f[x_{n+1}, x_n, \ldots x_2] - f[x_n, x_{n-1}, \ldots, x_1]}{x_{n+1} - x_1}\,, \end{aligned}$$

as required in

$$f_n(x) = b_0 + b_1(x - x_1) + b_2(x - x_1)(x - x_2) + \cdots + b_n(x - x_1)(x - x_2)\ldots(x - x_n)\,.$$

For Lagrange interpolating polynomials, directly substitute in the data points (x_i, y_i) as per the formula:

$$f_n\left(x\right) = \sum_{i=0}^{n} L_i\left(x\right) y_i\,,$$

where

$$L_i\left(x\right) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\,.$$

An alternative method to interpolate is to use splines - that is, we apply polynomial functions between *pairs* of data points, with the understanding that

1. the polynomial functions must pass through both the data points defining the interval.
2. higher order derivative(s) are set equal at the interior data points to ensure smoothness. For quadratic splines this is the 1st order derivative; for cubic splines this is both the 1st and 2nd order derivatives.
3. additional information is required to determine the remaining one (or two, in the case of cubic splines) condition. Otherwise, an assumption is made regarding a higher order derivative (e.g. $\partial^2 f/\partial x^2 = 0$ at the 1st data point).

The above techniques can be extended to multi-dimensions. For instance the process in bilinear interpolation for a function $z = f(x, y)$ is:

- at $y = y_1$, interpolate between x_1 and x_2;
- at $y = y_2$, interpolate between x_1 and x_2;
- using the above 2 expressions, interpolate between y_1 and y_2.

Higher-order polynomials or splines can also be used instead of a linear relationship, for which the process is largely similar.

Fourier approximations and transforms are considered usually when we have time-varying functions. Fourier approximations are used when we have a periodic function, i.e. we have equally-spaced data and we want to fit a sinusoidal curve of equation

$$f(t) = A_0 + A_1 \cos(\omega t) + B_1 \sin(\omega t) + A_2 \cos(2\omega t) + B_2 \sin(2\omega t) + \cdots + A_m \cos(m\omega t) + B_m \sin(m\omega t)\,.$$

then from least-squares regression analysis the coefficients are:

$$\begin{aligned} A_0 &= \frac{1}{N} \sum y_i\,, \\ A_k &= \frac{2}{N} \sum y_i \cos(k\omega t_i)\,, \\ B_k &= \frac{2}{N} \sum y_i \sin(k\omega t_i)\,, \end{aligned}$$

for $k = 1$, 2, ..., m. Remember that the last data point is not included, as it is assumed to have the same functional value as the first.

If we are more interested in the frequency domain (which is usually the case for non-periodic functions), then a Fourier transform can be used. As with the Fourier approximation, the last data point corresponding to $t_n = T$ is not included. The Discrete Fourier Transform (DFT) and its inverse are:

$$F_k = \sum_{n=0}^{N-1} f_n e^{-ik\omega n} \qquad k = 0, \ldots, N-1\,,$$

where F_k is the magnitude of the k-th sinusoid with angular frequency $k\omega$ and $\omega = 2\pi/N$; and

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{ik\omega n} \qquad n = 0, \ldots, N-1\,.$$

The coefficient $1/N$ in the inverse transform is actually just a scale factor that can appear in either transform but not both.

The Fast Fourier Transform (FFT) is a technique which decomposes the original DFT into several smaller DFTs. In the Sande-Tukey algorithm, the process is as follows:

1. Calculate two new functions/data sets:

   - $g_n = f_n + f_{n+N/2}$
   - $h_n = \left(f_n - f_{n+N/2}\right) e^{-i(2\pi/N)n}$

for $n = 0, 1, 2, ... \frac{N}{2} - 1$.

2. Repeat above step for each of the new (smaller) data sets (with halved N!) until $\frac{N}{2}$ sets of 2 data points are obtained.

3. Perform DFT for each of those $\frac{N}{2}$ data sets.

4. Unscramble the output.

Alternatively, the Cooley-Tukey algorithm requires scrambling the input data points first before decomposition (as per the above) and finally DFT of the eventual $\frac{N}{2}$ data sets.

3 Root-Finding Methods

3.1 Introduction

Recall in the previous chapter one of the goals of curve-fitting was to establish a mathematical relationship. In the process of using that relationship we sometimes need to solve for the root(s), e.g. in order to determine which intermediate data point x corresponded to a given functional value $f(x)$. In this chapter we will look at various methods with which we can obtain close estimates of the roots of equations, if we are unable to solve for them directly.

One of the most straightforward methods – that we have likely done before in labs – is to plot a graph of the function, and then see where it crosses the horizontal axis (assuming that we are solving for $f(x) = 0$). In other cases we may have had to plot two different functions and looked at where they intersected. This family of techniques is called **graphical methods**. While easy to implement, unfortunately the roots obtained this way are not necessarily very precise. However we can use them as initial guesses for more accurate methods such as those we will cover here.

There are two main groups of root-finding techniques we will introduce, called **bracketing methods** and **open methods**. Bracketing methods are not quite self-explanatory, but it is not difficult to imagine what might be involved. Open methods on the other hand, tend to be favoured for their simplicity in execution. Both groups of techniques though, work by **converging** (hopefully) onto the root(s), beginning from a chosen starting point.

We will also discuss more complex cases like when we have simultaneous equations (which probably mean multiple independent variables and/or multiple roots) as well as the special case of polynomials.

3.2 Bracketing Methods

Have you played guessing games where you only receive one of two answers - 'hot' and 'cold', or 'higher' and 'lower'? In the game, we implicitly have a range of answers avail-

able, for instance if the goal is to guess the number of jelly beans in a jar, then obviously the answer is going to be a real number between zero (the lower bound) and infinity (the upper bound). The person who knows the answer can give hints by telling you how your guess compares to the correct answer, and you thus adjust for your next guess.

Bracketing methods work somewhat similarly to the game above. As per the name, they tend to work by bracketing the real root. In fact, they typically take advantage of the fact that the sign of the function changes in the vicinity of the root (the functional value is zero at the root). The general idea is to begin with a range which the root is in, and then gradually narrow it down.

3.2.1 Bisection Method

The bisection method works by halving the interval with each iteration. As mentioned earlier, there is assumed to be a sign change in the vicinity of the root, so there should be a change in sign within the chosen interval. If there is, then the midpoint of the interval is evaluated. The basic algorithm is as follows:

1. Choose upper and lower points for the root, x_u and x_l respectively, such that there is a change in the sign of the function within the interval $[x_l,\, x_u]$.
2. Obtain the midpoint $x_r = \frac{x_l + x_u}{2}$, and the corresponding functional value $f\,(x_r)$.
3. Make a decision on refining the interval:
   - If $f\,(x_r)\, f\,(x_l) < 0$, then the new interval is $[x_l,\, x_r]$.
   - If $f\,(x_r)\, f\,(x_u) < 0$, then the new interval is $[x_r,\, x_u]$.
   - If $f\,(x_r)\, f\,(x_l) = 0$, then the root is x_r.
4. If the interval was redefined, then repeat from Step 1.

When do we stop refining? One possibility is to end when the error has fallen below a certain level, say 0.1%. However for us to determine the error we would have to know the true value *a priori*... in which case there would not be a need to estimate the root.

So instead, the typical criteria is to use a **relative error** – that is, when the relative difference between estimated roots becomes very small:

$$\epsilon = \left| \frac{x^{\text{new}} - x^{\text{old}}}{x^{\text{new}}} \right| \times 100\% \,.$$

Example

Given a polynomial equation $0.04x^2 - 0.3x - 0.78 = 0$, which has analytical roots $x = 9.54$ and $x = -2.04$, let's see if we can obtain the same root in the range $8 \leq x \leq 10$ for a relative error of $< 1\%$.

First we need to obtain some initial guesses. The easiest would be to use the extremes of the data range i.e. $x_l = 8$ and $x_u = 12$. Before we proceed we should check that this is a valid interval:

$$\begin{aligned} f(x_l) &= f(8) = 0.04(8)^2 - 0.3(8) - 0.78 = -0.62 \\ f(x_u) &= f(12) = 0.04(12)^2 - 0.3(12) - 0.78 = 1.38 \end{aligned}$$

$$\Rightarrow f(x_l)\ f(x_u) < 0\,,$$

which means that there is a sign change in the interval, and therefore this interval is valid. We can now proceed to refine the interval.

$$\begin{aligned} x_r &= \frac{x_l + x_u}{2} = \frac{8 + 12}{2} = 10 \\ f(x_r) &= 0.04(10)^2 - 0.3(10) - 0.78 = 0.22 \\ \epsilon &= \left| \frac{10 - 8}{10} \right| \times 100\% = 20\% \end{aligned}$$

Looking at the sign of the functional value, we will replace the old upper bound with this midpoint. So now the interval is $[8, 10]$.

$$\begin{aligned} x_r &= \frac{x_l + x_u}{2} = \frac{8 + 10}{2} = 9 \\ f(x_r) &= 0.04(9)^2 - 0.3(9) - 0.78 = -0.24 \\ \epsilon &= \left| \frac{9 - 10}{9} \right| \times 100\% = 11.1\% \end{aligned}$$

Looking at the sign of the functional value, we will replace the old lower bound with this

midpoint. So now the interval is $[9, 10]$.

$$\begin{aligned} x_r &= \frac{x_l + x_u}{2} = \frac{9+10}{2} = 9.5 \\ f(x_r) &= 0.04(9.5)^2 - 0.3(9.5) - 0.78 = -0.02 \\ \epsilon &= \left|\frac{9.5-9}{9.5}\right| \times 100\% = 5.26\% \end{aligned}$$

We can see that $f(x_r)$ is quite close to 0 already (and correct too, compared to the analytical root!), but the relative error is not yet below our threshold so we will continue. Looking at the sign of the functional value, we will replace the old lower bound with this midpoint. So now the interval is $[9.5, 10]$.

$$\begin{aligned} x_r &= \frac{x_l + x_u}{2} = \frac{9.5+10}{2} = 9.75 \\ f(x_r) &= 0.04(9.75)^2 - 0.3(9.75) - 0.78 = 0.0975 \\ \epsilon &= \left|\frac{9.75-9.5}{9.75}\right| \times 100\% = 2.56\% \end{aligned}$$

Looking at the sign of the functional value, we will replace the old upper bound with this midpoint. So now the interval is $[9.5, 9.75]$.

$$\begin{aligned} x_r &= \frac{x_l + x_u}{2} = \frac{9.5+9.75}{2} = 9.625 \\ f(x_r) &= 0.04(9.625)^2 - 0.3(9.625) - 0.78 = 0.0381 \\ \epsilon &= \left|\frac{9.625-9.75}{9.625}\right| \times 100\% = 1.30\% \end{aligned}$$

Looking at the sign of the functional value, we will replace the old upper bound with this midpoint. So now the interval is $[9.5, 9.625]$.

$$\begin{aligned} x_r &= \frac{x_l + x_u}{2} = \frac{9.5+9.625}{2} = 9.5625 \\ f(x_r) &= 0.04(9.5625)^2 - 0.3(9.5625) - 0.78 = 0.00891 \\ \epsilon &= \left|\frac{9.5625-9.625}{9.5625}\right| \times 100\% = 0.654\% \end{aligned}$$

The relative error is now below our designated 1% tolerance, so we terminate the iterations here with our estimated root as 9.5625. Compared to the analytical root (9.54), we see that our estimate is pretty good for a relative error $< 1\%$.

Technically, we did not need to use the values of $f(x_r)$ and were only concerned with the sign. However seeing the value approach zero is a good check that we are on the right track.

Exercise

Using the bisection method, try to obtain the second root ($x = -2.04$) of the polynomial in the Example. Using the range $-5 \leq x \leq -1$, how good is your estimate after five iterations? How many more iterations do you think you will need to achieve a relative error $< 1\%$?

Ans: $x_r = -2.125$, $\epsilon = 5.9\%$, 3 more iterations.

3.2.2 Method of False-Position

The **false-position** method is similar to the bisection method, but does not halve the interval each iteration. Instead it makes a much more dramatic 'cut' based on the relative magnitudes of the upper and lower bounds. Consider the sketch in Fig. 15:

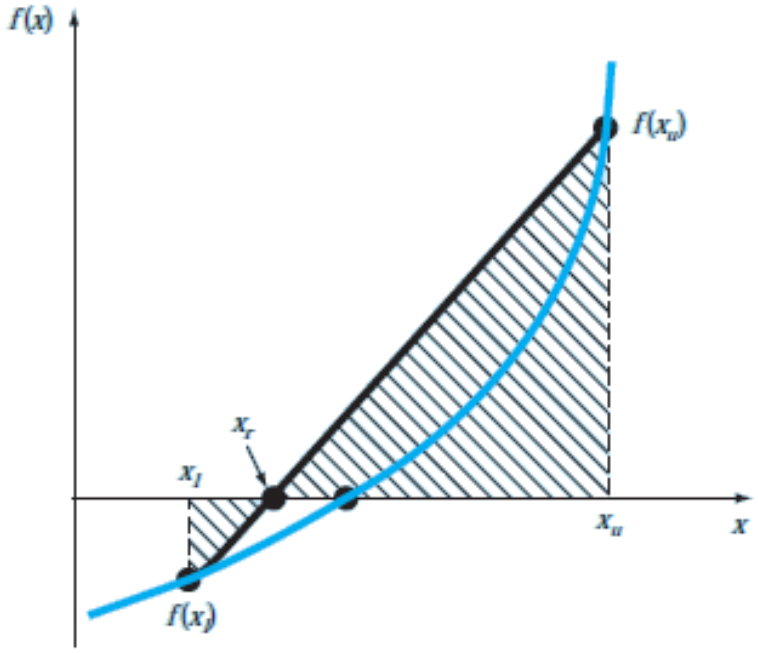


Figure 15: Sketch illustrating the method of false position, from *Chapra & Canale*, Fig. 5.12 on pg132. A linear interpolation is used to obtain a closer estimate of the root than a simple bisection. Similar triangles (shaded) are used to formulate the expression for this root estimate.

As you can see, the functional value at the upper bound is obviously larger than that at the lower bound, so the root is going to be located closer to the lower bound than the upper. Halving the intervals as we do with the bisection method will still get us a good estimate of the root, but logically it would be faster if we could obtain estimates that are biased towards the closer bound.

In Fig. 15, we essentially use a linear interpolation (the straight line) to obtain a (better) estimate of the root. By doing so, we obtain a *false position* for the root – which leads to

the name of this technique. However the use of linear interpolation has resulted in this method also being called **linear interpolation method**.

Regardless of the name, when we use linear interpolation (or similar triangles, essentially) we can express the root estimate in terms of the upper and lower bounds and their respective functional values:

$$\frac{0 - f(x_l)}{x_r - x_l} = \frac{f(x_u) - 0}{x_u - x_r},$$

which simplifies to:

$$x_r = x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)}.$$

Apart from this, the algorithm for the false-position method is similar to that of the bisection method. So the procedure would be:

1. Choose upper and lower points for the root, x_u and x_l respectively, such that there is a change in the sign of the function within the interval $[x_l,\ x_u]$.
2. Obtain the false position root estimate $x_r = x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)}$, and the corresponding functional value $f(x_r)$.
3. Calculate the relative error $\epsilon = \left|\frac{x^{\text{new}} - x^{\text{old}}}{x^{\text{new}}}\right| \times 100\%$.
4. If the relative error is not yet within specified tolerances, make a decision on refining the interval:
   - If $f(x_r)\, f(x_l) < 0$, then the new interval is $[x_l,\ x_r]$.
   - If $f(x_r)\, f(x_u) < 0$, then the new interval is $[x_r,\ x_u]$.
   - If $f(x_r)\, f(x_l) = 0$, then the root is x_r.
5. If the interval was redefined, then repeat from Step 1.

Example

Let's try the earlier example from the section on the bisection method again. We had a polynomial equation $0.04x^2 - 0.3x - 0.78 = 0$ with analytical roots $x = 9.54$ and $x = -2.04$. Using the bisection method we obtained a estimate of $x_r = 9.5625$ with a relative error of 0.654% after six iterations.

For a good comparison, we shall use the same initial guesses for upper and lower bounds, i.e. $x_l = 8$ and $x_u = 12$. As before we confirm that this is a valid interval:

$$\begin{aligned} f(x_l) &= f(8) = 0.04(8)^2 - 0.3(8) - 0.78 = -0.62 \\ f(x_u) &= f(12) = 0.04(12)^2 - 0.3(12) - 0.78 = 1.38 \end{aligned}$$

$$\Rightarrow f(x_l)\ f(x_u) < 0\,,$$

which means that there is a sign change in the interval, and therefore this interval is valid. We can now proceed to refine the interval.

$$\begin{aligned} x_r &= x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)} = 12 - \frac{1.38(12-8)}{1.38 + 0.62} = 9.24 \\ f(x_r) &= 0.04(9.24)^2 - 0.3(9.24) - 0.78 = -0.1369 \\ \epsilon &= \left|\frac{9.24 - 8}{9.24}\right| \times 100\% = 13.4\% \end{aligned}$$

Looking at the sign of the functional value, we will replace the old lower bound with this 'midpoint'. So now the interval is $[9.24, 12]$.

$$\begin{aligned} x_r &= x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)} = 12 - \frac{1.38(12-9.24)}{1.38 + 0.1369} = 9.489 \\ f(x_r) &= 0.70.04(9.489)^2 - 0.3(9.489) - 0.78 = -0.02502 \\ \epsilon &= \left|\frac{9.489 - 9.24}{9.489}\right| \times 100\% = 2.62\% \end{aligned}$$

Looking at the sign of the functional value, we will replace the old lower bound with this 'midpoint'. So now the interval is $[9.489, 12]$.

$$\begin{aligned} x_r &= x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)} = 12 - \frac{1.38(12-9.489)}{1.38 + 0.02502} = 9.534 \\ f(x_r) &= 0.04(9.534)^2 - 0.3(9.534) - 0.78 = -0.00441 \\ \epsilon &= \left|\frac{9.534 - 9.489}{9.534}\right| \times 100\% = 0.47\% \end{aligned}$$

The relative error is now below our designated 1% tolerance, so we terminate the iterations here with our estimated root as 9.534. Comparing this answer and the number of iterations taken to get here, to that of the bisection method as well as the analytical root, what are your thoughts?

Exercise 1

Using the false-position method, try to obtain the second root of the polynomial in Example 1. How good is your estimate after five iterations? And how does this answer compare to that obtained from bisection method?

Ans: $x_r = -2.041$, $\epsilon = 0.076\%$

Exercise 2: Food for thought!

1. Did you notice how, when we did the example on the false-position method (for the root ≈ 3), the lower bound remained the same with every iteration?

    - Why do you think this was so?

    - In what case would it be that the upper bound remains unchanged?

    - In what case would both bounds change?

- If one bound doesn't change, does it mean this method is less accurate compared to bisection method, where the interval size is halved every iteration?

- Can you think of a way to deal with an unchanging bound?

2. In all the above examples, we've very conveniently had cases where our interval only bracketed one root (we chose so, in fact).

   - Is it possible for there to be more than one root in the bracket?

   - What would happen, if this is the case?

3. We've covered two bracketing methods here - bisection and false-position. For what reasons would you choose one method over another?

3.3 Open Methods

Unlike **bracketing methods**, **open methods** do not hone in on the root by narrowing down the guess interval. Instead, a formula is used that often requires only one or two **starting values**. These starting values are guesses at the value of the root, *not* guesses of the bounds.

These are also the methods which are more easily implemented using Excel or a calculator, as we will see later. Within this section we shall introduce some of the more popular methods.

3.3.1 Simple Fixed-Point Iteration

The name '**fixed-point iteration**' is perhaps a bit ambiguous, but another name for this method is '**successive substitution**' which gives a better indication of what this method may entail.

Basically, the idea is to *rearrange the function involving* x such that we have a single x on the left hand side, and an expression (which still has x) on the right hand side. We will use this expression on the right, to come up with new guesses for the root x.

For instance, let's say we have the function $f(x) = ax^2 + bx + c = 0$. We can rearrange this to get:

$$x = -\frac{ax^2 + c}{b}\,.$$

This is going to be the formula we use for getting a series of (hopefully) converging guesses. Let's call the first guess x_1, for which we choose a value based on some logical reasoning (maybe by inspecting the graph, if it is available). Then the second guess would be:

$$x_2 = -\frac{ax_1^2 + c}{b}\,.$$

We can substitute x_2 into the right hand side again to obtain the third guess, and so on. In

other words we will be doing this:

$$\begin{aligned} x_2 &= -\frac{ax_1^2 + c}{b} \\ x_3 &= -\frac{ax_2^2 + c}{b} \\ x_4 &= -\frac{ax_3^2 + c}{b} \\ \vdots &= \vdots \end{aligned}$$

Eventually we stop when we achieve the required relative error

$$\epsilon = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \times 100\% \,,$$

as with the bracketing methods previously.

Graphically, what we are looking for is the intersection between $y = x$ and $y = -\frac{ax^2+c}{b}$. Our guesses, substituted into the right hand side, are essentially vertical lines drawn up from the horizontal axis to $y = -\frac{ax^2+c}{b}$. From here we then obtain a horizontal line leading to the vertical axis – our next guess. We then repeat in the hopes of getting closer to the intersection with $y = x$.

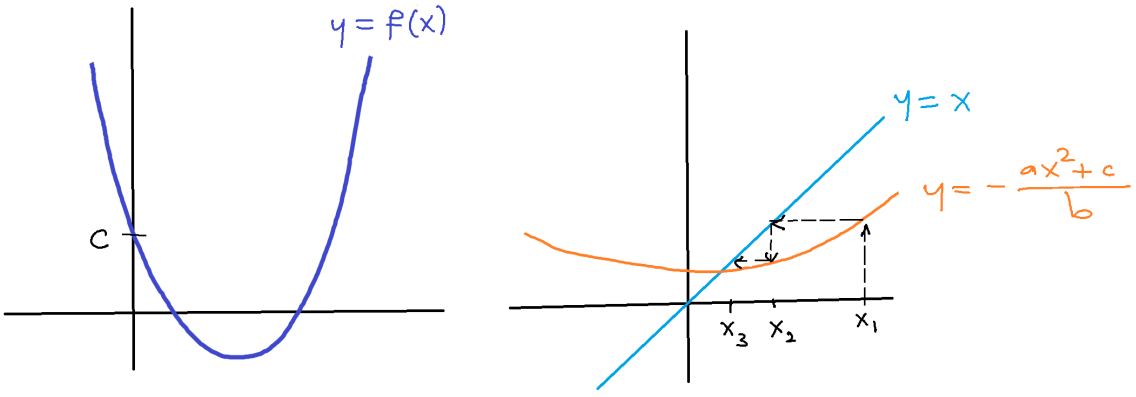


Figure 16: Sketch illustrating the process of simple fixed-point iteration, for a quadratic function $f\,(x) = ax^2 + bx + c$.

Example

Let's go with a quadratic expression for simplicity, say

$$f\,(x) = 2x^2 - 5x + 3 = 0\,.$$

We know that the two roots of this expression are $x = 3/2$ and $x = 1$, so we will be able to tell whether our fixed-point iteration is progressing well.

As before, we rearrange the expression to get a single x on the left hand side:

$$x = \frac{2x^2 + 3}{5}\,.$$

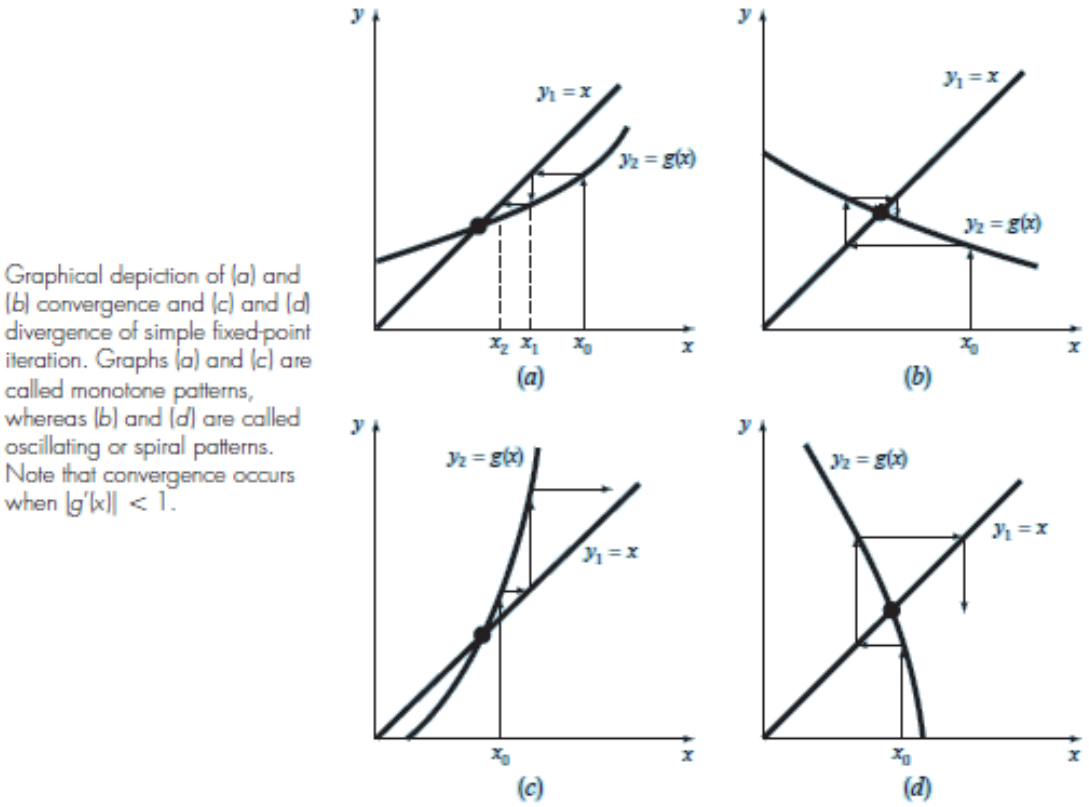
Let's use $x_1 = 0$ as the first guess, and stop after we obtain $< 1\%$ relative error. So now:

$$\begin{aligned}
x_2 &= \frac{2x_1^2 + 3}{5} = 0.6\,, & \epsilon &= \left|\frac{0.6 - 0}{0.6}\right| \times 100\% = 100\% \\
x_3 &= \frac{2x_2^2 + 3}{5} = 0.744\,, & \epsilon &= \left|\frac{0.744 - 0.6}{0.744}\right| \times 100\% = 19.4\% \\
x_4 &= \frac{2x_3^2 + 3}{5} = 0.8214\,, & \epsilon &= \left|\frac{0.8214 - 0.744}{0.8214}\right| \times 100\% = 9.4\% \\
x_5 &= \frac{2x_4^2 + 3}{5} = 0.8699\,, & \epsilon &= \left|\frac{0.8699 - 0.8214}{0.8699}\right| \times 100\% = 5.6\% \\
x_6 &= \frac{2x_4^2 + 3}{5} = 0.9027\,, & \epsilon &= \left|\frac{0.9027 - 0.8699}{0.9027}\right| \times 100\% = 3.6\% \\
x_7 &= \frac{2x_4^2 + 3}{5} = 0.9259\,, & \epsilon &= \left|\frac{0.9259 - 0.9027}{0.9259}\right| \times 100\% = 2.5\% \\
x_8 &= \frac{2x_4^2 + 3}{5} = 0.9429\,, & \epsilon &= \left|\frac{0.9429 - 0.9259}{0.9429}\right| \times 100\% = 1.8\% \\
x_9 &= \frac{2x_4^2 + 3}{5} = 0.9557\,, & \epsilon &= \left|\frac{0.9557 - 0.9429}{0.9557}\right| \times 100\% = 1.3\% \\
x_{10} &= \frac{2x_4^2 + 3}{5} = 0.9653\,, & \epsilon &= \left|\frac{0.9653 - 0.9557}{0.9653}\right| \times 100\% = 0.99\%
\end{aligned}$$

Since the relative error is less than 1%, we can stop the iterations here with the 10-th guess for the root $x_{10} = 0.9653$. We see that it is quite a good approximation for the true value $x = 1$.

This method is fairly straightforward, but there are some hidden issues. An obvious one is the choice of initial guess - logically, the closer the guess is to the true value (that is, its 'goodness') then the shorter the series of iterations should be. However this is dependent on the form of the iterative expression, as there is no guarantee that it would converge in the 'right' direction to the true value.

Fig. 17 shows some sketches of different functions $y = g(x)$, where $x = g(x)$ is the rearranged form of the function for which we want estimates of the root. In two of the four



Graphical depiction of (a) and (b) convergence and (c) and (d) divergence of simple fixed-point iteration. Graphs (a) and (c) are called monotone patterns, whereas (b) and (d) are called oscillating or spiral patterns. Note that convergence occurs when $|g'(x)| < 1$.

Figure 17: Sketches illustrating the concept of convergence and divergence in the simple fixed-point iteration method, from *Chapra & Canale*, Fig. 6.3 on pg146.

cases there, **convergence** occurs – that is, the iterations move towards the intersection of the two graphs. In the other two cases, **divergence** occurs – the iterations move away from the intersection. Note also that the way each of those iterations change is different - they can 'spiral' or they can move in a zigzag or oscillatory pattern. Typically what we find is that:

convergence occurs when the local slope of $y = g(x)$ is between -1 and 1.

This condition is not always easily satisfied especially if the graph is not monotonous. If you are interested, Chapra & Canale give a short derivation on this convergence criteria.

In the case of Fig. 17(a) it looks like there could potentially be a second root (larger than x_0), so if that were the value we were aiming for then our iteration would actually be diverging

from it. This actually highlights another issue - that of multiple roots. There is always the possibility that even though we are 'aiming' for one root, our iteration converges to another.

Exercise

Consider a function $f(x) = x - g(x)$, where $g(x) = \sin x$ is plotted in Fig. 18. Obtain an estimate for the root close to zero (analytical root is 0). Hint: consider the graph of $g'(x)$ to check for convergence before proceeding to iterate!

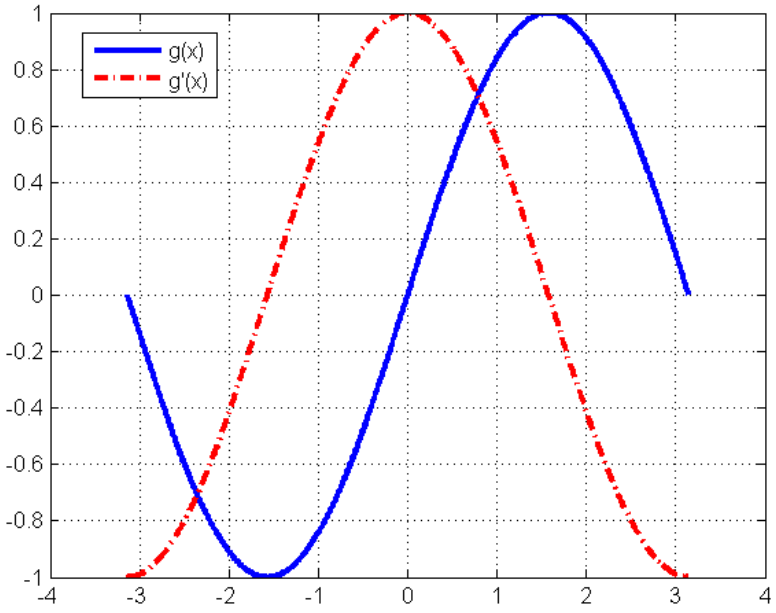


Figure 18: Plot of $g(x) = \sin x$ and $g'(x)$, for $-\pi \leq x \leq \pi$.

If the function were instead $f(x) = x - x \sin x$, do you think it would still converge?

3.3.2 Newton-Raphson Method

The Newton-Raphson method is a bit more complicated than the simple fixed-point iteration, but more commonly used. It uses the slope of the function $f(x)$ at the guessed value to obtain the next guess. Fig. 19 shows an illustration of this concept.

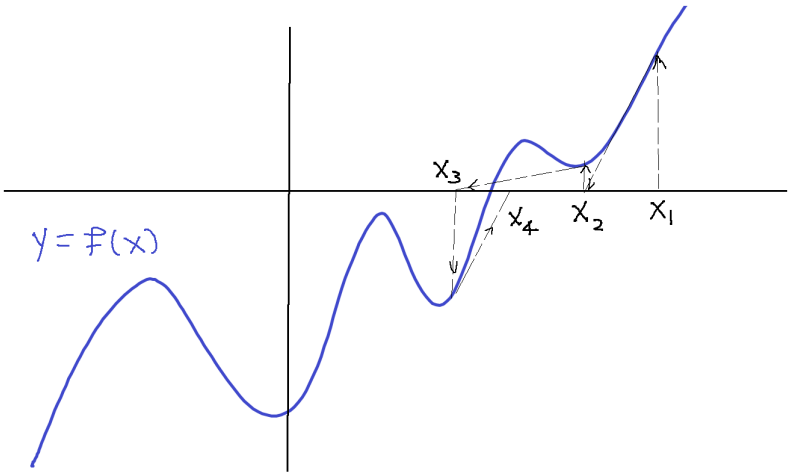


Figure 19: Sketch illustrating the process of the Newton-Raphson method, for a function $f(x) = 0$.

Basically, we extend a tangent from the spot corresponding to the i-th guess, i.e. $(x_i,\ f(x_i))$ until it intersects the horizontal axis. This horizontal intersect is the next guess, x_{i+1}. Mathematically, this is:

$$\begin{aligned} f'(x_i) &= \frac{f(x_i) - 0}{x_i - x_{i+1}} \\ \Rightarrow \qquad x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \end{aligned}$$

This is called the **Newton-Raphson formula**.

Example

Consider the function $e^x + 3x - 7 = 0$, for which we wish to locate the root(s). We first consider graphical methods to determine how many roots there will be – the intersections of $y = e^x$ and $y = 7 - 3x$ will be the roots. In this case, from a quick sketch (see Fig. 20) we know that there will be only one root, and it will be positive. Looking at the plot, $x_0 = 1$ is a reasonable first guess.

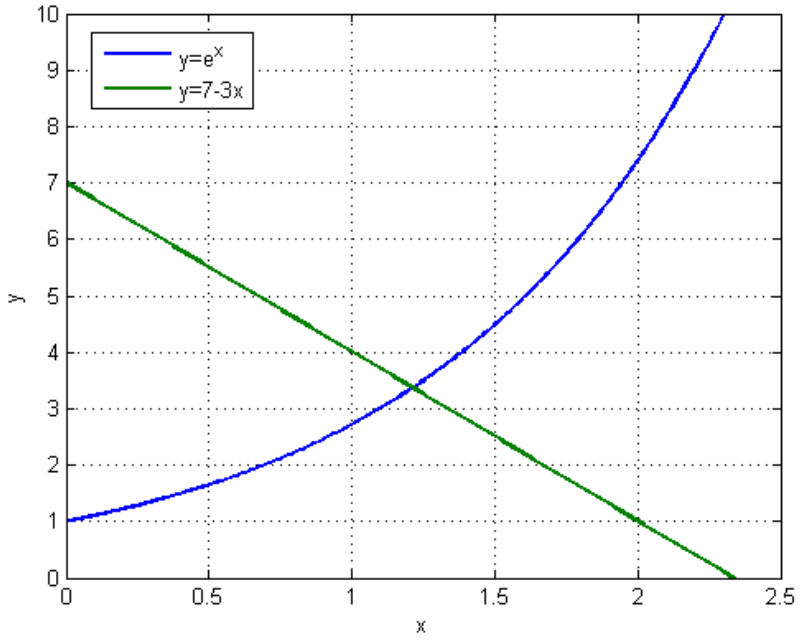


Figure 20: Plots of $y = e^x$ and $y = 7 - 3x$, for $0 \leq x \leq 2.5$.

The next step is to determine the slope expression:

$$f'(x) = e^x + 3\,,$$

as this features in the Newton-Raphson formula.

We now proceed to iterate until we achieve a specified relative tolerance, say $< 0.1\%$.

$$x_1 = 1 - \frac{e^1+3\times(1)-7}{e^1+3} = 1.22414\,, \qquad \epsilon = \left|\frac{1.22414-1}{1.22414}\right| \times 100\% = 18.3\%$$

$$x_2 = 1.22414 - \frac{e^{1.22414}+3\times(1.22414)-7}{e^{1.22414}+3} = 1.21263\,, \qquad \epsilon = \left|\frac{1.21263-1.22414}{1.21263}\right| \times 100\% = 0.95\%$$

$$x_3 = 1.21263 - \frac{e^{1.21263}+3\times(1.21263)-7}{e^{1.21263}+3} = 1.21260\,, \qquad \epsilon = \left|\frac{1.21260-1.21263}{1.21260}\right| \times 100\% = 0.002\%$$

Since the relative error is less than 0.1%, we can stop the iterations here. We see that it is quite a good approximation for the true value just by looking at the plot in Fig. 20. Notice also that the relative error decreases very dramatically - this is very different from the previous methods we discussed, and also one of the reasons why the Newton-Raphson method is so popular.

The Newton-Raphson method is not without potential difficulties. As you can imagine, having multiple roots would be an issue (which we will discuss later). Another is the choice of initial guess, as well as the curvature of the function. We can see that using the slope of the function at the guess value can lead to large jumps from guess to guess - and this is amplified when the slope gets close to zero. A 'worst case' scenario would be if by chance, one of the guesses is at a stationary point of the function, whereby the slope is zero and thus no possibility of obtaining a next guess. There is also the possibility of the curvature being such that convergence is slow (see Example 6.5 in *Chapra & Canale*, pg151), or that the iterations diverge from the true root.

Exercise 1

Consider a function $g(x) = e^x \sin x$, which is plotted in Fig. 21. Obtain an estimate for the local maximum point (analytically: $x = \frac{3}{4}\pi$) in the range $-\pi \leq x \leq \pi$.

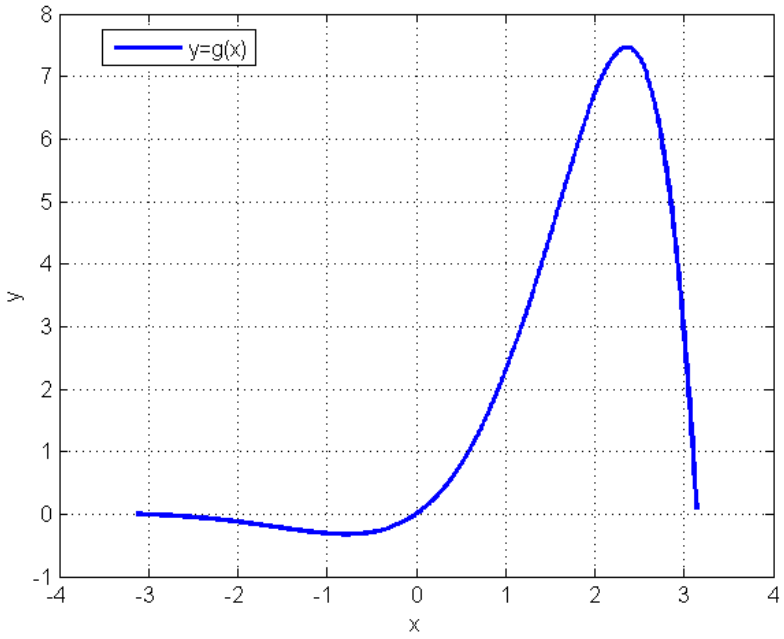


Figure 21: Plot of $g(x) = e^x \sin x$, for $-\pi \leq x \leq \pi$.

Exercise 2

Let's revisit the function $g(x) = e^x \sin x$ from Exercise 1 again. See if you can obtain both roots for $g(x) = 0$. Hint: the analytical roots are at $x = -\pi$, 0 and π.

3.3.3 Secant Method

Recall that for the Newton-Raphson method, the slope of the function at the guess is used to obtain the next guess. But not all functions are easy to differentiate (and thus obtain an expression for the slope)! So we can use a finite difference scheme to approximate the slope, in those cases:

$$f'(x_i) \approx \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}.$$

Substituting this estimate of the slope into the Newton-Raphson formula we get:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}.$$

This is the formula for the **secant method**.

In a way, this is similar to the linear interpolation that we used for the false position method, but instead of using the two bracketing values, we use the current and previous

guesses to estimate the slope. Fig. 22 shows a simple illustration of this process.

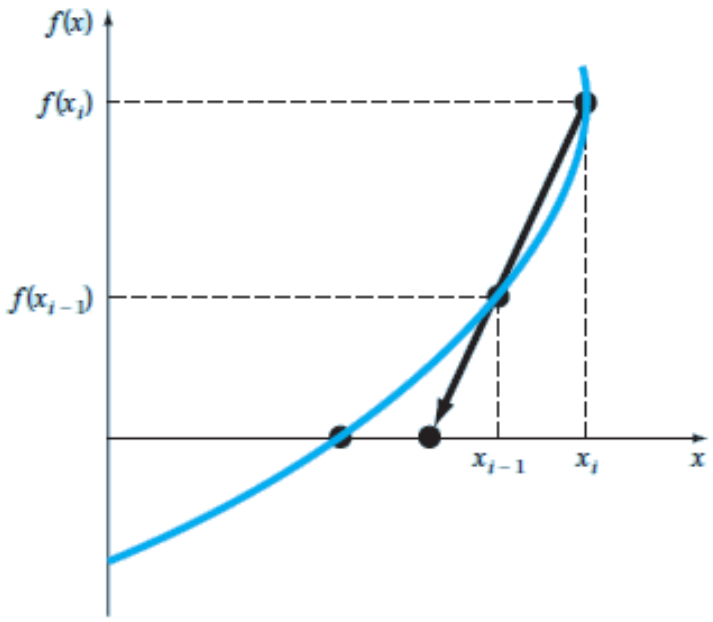


Figure 22: Sketch illustrating the concept behind the secant method, from *Chapra & Canale*, Fig. 6.7 on pg154.

However we can see straightaway that there are two obvious issues - (1) if the two successive guesses x_{i-1} and x_i are not close together then there is a high possibility of the slope estimate being very inaccurate; and (2) we would need *two* initial guesses to even begin this whole iterative process.

Example

Let's revisit the function $e^x + 3x - 7 = 0$, for which we wish to locate the root. From Fig. 20, we see that there is only one intersection of $y = e^x$ and $y = 7 - 3x$, and therefore only one root. We will need two initial guesses, so let's use say, $x_{-1} = 1$ and $x_0 = 1.5$.

We can now proceed to iterate, until we reach a relative error of say $< 0.1\%$ for comparison

with the Newton-Raphson method example.

$$\begin{aligned} x_1 &= (1.5) - \frac{\left(e^{1.5} + 3 \times (1.5) - 7\right)(1 - 1.5)}{(e^1 + 3 \times (1) - 7) - (e^{1.5} + 3 \times (1.5) - 7)} = 1.1964\,, \\ \epsilon &= \left|\frac{1.1964 - 1.5}{1.1964}\right| \times 100\% = 25.38\%\,. \end{aligned}$$

The relative tolerance is larger than 0.1%, so we iterate again but now using $x_2 = 1.5$ and $x_3 = 1.1964$ in the formula to obtain the next guess, x_4:

$$\begin{aligned} x_2 &= (1.1964) - \frac{\left(e^{1.1964} + 3 \times (1.1964) - 7\right)(1.5 - 1.1964)}{(e^{1.5} + 3 \times (1.5) - 7) - (e^{1.1964} + 3 \times (1.1964) - 7)} = 1.2113\,, \\ \epsilon &= \left|\frac{1.2113 - 1.1964}{1.2113}\right| \times 100\% = 1.24\%\,. \end{aligned}$$

Again, the relative tolerance is larger than 0.1%, so we iterate again but now using $x_3 = 1.1964$ and $x_4 = 1.2113$ in the formula to obtain the next guess, x_5:

$$\begin{aligned} x_3 &= (1.2113) - \frac{\left(e^{1.2113} + 3 \times (1.2113) - 7\right)(1.1964 - 1.2113)}{(e^{1.1964} + 3 \times (1.1964) - 7) - (e^{1.2113} + 3 \times (1.2113) - 7)} = 1.2126\,, \\ \epsilon &= \left|\frac{1.2126 - 1.2113}{1.2126}\right| \times 100\% = 0.10\%\,. \end{aligned}$$

The relative tolerance has reached below our specified level, so we can stop iterating here.

Compared to the estimated value from Newton-Raphson method ($x_3 = 1.21260$), our estimate of 1.2126 was obtained with the same number of iterations, but a larger relative tolerance. What do you think is causing the difference?

Exercise

Determine the left-most root of $f(x) = x^7 + 3x^4 + \frac{1}{2}x^3 - x^2 - \cos(x)$, within the range $-1.5 \leq x \leq 1$. The graph is plotted in Fig. 23 to aid you.

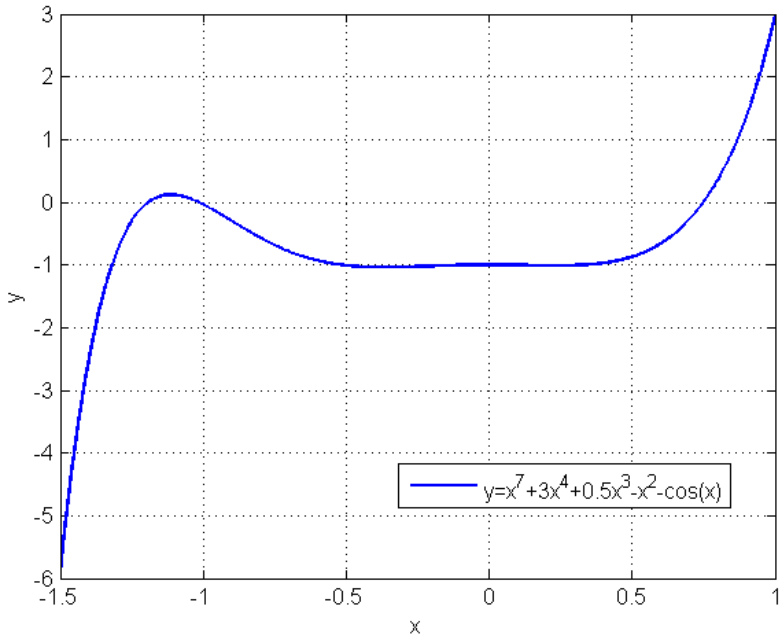


Figure 23: Plot of $f(x) = x^7 + 3x^4 + \frac{1}{2}x^3 - x^2 - \cos(x)$, for $-1.5 \leq x \leq 1$.

Ans: if $x_{-1} = -1.5$ and $x_0 = -1.3$, then $x_4 = -1.19466$ and $\epsilon = 0.84\%$.

3.3.4 Inverse Quadratic Interpolation

Inverse quadratic interpolation is, as the name sounds, a method that involves a quadratic interpolation but instead of looking for the functional value $f(x)$ we are interested in obtaining the x – hence the 'inverse'. In a way this is similar to the secant method - which can be considered an inverse linear interpolation - but perhaps more accurate. Fig. 24 illustrates both methods applied to the same function: because of the curve, the inverse quadratic interpolation clearly gives a better estimate of the root.

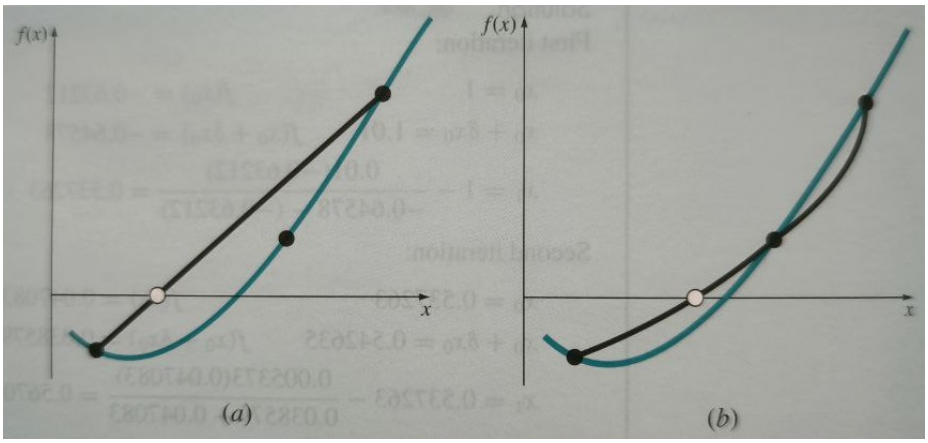


Figure 24: Sketch of (a) secant method and (b) inverse quadratic interpolation applied to the same function (blue line). The estimated root (white circle) is closer to the true root for (b) compared to (a). This figure was taken from *Chapra & Canale*, Fig. 6.10 on pg160.

The problem with using a quadratic function though, is that there is a possibility that it does not intersect the horizontal axis i.e. it has complex instead of real roots. We can preempt this by using a quadratic function in y (the independent variable) instead. As shown in Fig. 25, such a function will definitely intersect the horizontal axis.

We can use a concept we learned earlier - the Lagrange interpolating polynomial - to form a quadratic function $x = g(y)$ using 3 points (x_{i-2}, y_{i-2}), (x_{i-1}, y_{i-1}) and (x_i, y_i):

$$g(y) = \frac{(y - y_{i-1})(y - y_i)}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)} x_{i-2} + \frac{(y - y_{i-2})(y - y_i)}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)} x_{i-1} + \frac{(y - y_{i-2})(y - y_{i-1})}{(y_i - y_{i-2})(y_i - y_{i-1})} x_i \,.$$

The estimated root x_{i+1} will correspond to $y = 0$, so substituting it in we get the iteration

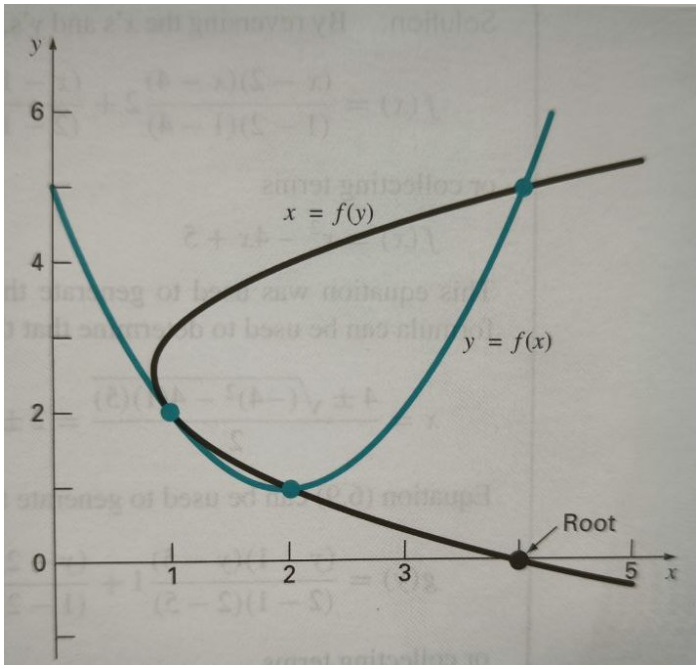


Figure 25: Two quadratic functions fit to the same 3 points: $y = f(x)$ (blue line) does not intersect the x-axis and thus has complex roots; $x = f(y)$ (black line) has a single real root. This figure was taken from *Chapra & Canale*, Fig. 6.11 on pg161.

formula:

$$x_{i+1} = \frac{y_{i-1}y_i}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)}x_{i-2} + \frac{y_{i-2}y_i}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)}x_{i-1} + \frac{y_{i-2}y_{i-1}}{(y_i - y_{i-2})(y_i - y_{i-1})}x_i \,.$$

Thus we will always require 3 initial guesses (as compared to secant method's 2 initial guesses and Newton-Raphson's 1 initial guess) for the iteration process. At each iteration, we will remove one of the old points and use the new guess (and its corresponding functional value): that is, the old y_{i-1} and y_i now become y_{i-2} and y_{i-1} respectively, while x_{i+1} becomes the new x_i. You can, of course, use any other logical reason to choose the points e.g. removing the point which is furthest away from the new guess. The final converged root will be the same, just that there may be minor differences during the intermediate iterations.

Note that on the off-chance a pair of the y values are identical (e.g. $y_i = y_{i-1}$) then we should swap to the secant method using only the non-identical root estimates (in this case x_i and $x_i - 2$).

Example

For comparison with the previous two methods, we will again use the same function $e^x + 3x - 7 = 0$. We will need three initial guesses, so let's use say, $(2, 6.389)$, $(1.5, 1.982)$ and $(1, -1.282)$. We can now proceed to iterate, until we reach a relative error of $< 0.1\%$.

$$\begin{aligned} x_1 &= \frac{1.1982 \times -1.282}{(6.389 - 1.982)(6.389 + 1.282)} \times 2 + \frac{6.389 \times -1.282}{(1.982 - 6.389)(1.982 + 1.282)} \times 1.5 \\ &\quad + \frac{6.389 \times 1.982}{(-1.282 - 6.389)(-1.282 - 1.982)} \times 1 = 1.20955\,, \\ y_1 &= e^{1.20955} + 3 \times 1.20955 - 7 = -0.0194\,, \\ \epsilon &= \left|\frac{1.20955 - 1}{1.20955}\right| \times 100\% = 17.32\%\,. \end{aligned}$$

Compared to the outcome after 1 iteration for the secant method ($x_1 = 1.1964$), we can see that this guess is closer to the converged value ($x = 1.2126$). However since the relative tolerance is larger than 0.1% we will iterate again. Now, we will use $(1.5, 1.982)$, $(1, -1.282)$ and the new point $(1.20955, -0.0194)$.

$$\begin{aligned} x_2 &= \frac{-1.282 \times -0.0194}{(1.982 + 1.282)(1.982 + 0.0194)} \times 1.5 + \frac{1.982 \times -0.0194}{(-1.282 - 1.982)(-1.282 + 0.0194)} \times 1 \\ &\quad + \frac{1.982 \times -1.282}{(-0.0194 - 1.982)(-0.0194 + 1.282)} \times 1.20955 = 1.21261\,, \\ y_2 &= e^{1.21261} + 3 \times 1.21261 - 7 = 6.25 \times 10^{-5}\,, \\ \epsilon &= \left|\frac{1.21261 - 1.20955}{1.21261}\right| \times 100\% = 0.25\%\,. \end{aligned}$$

There is now a steep decrease in relative error (unsurprising, since the first guess was already quite close) and the functional value y_2 is very close to zero too. However we have not met the convergence criteria yet so we iterate once more using $(1, -1.282)$, $(1.20955, -0.0194)$ and the new point $(1.21261, 6.25 \times 10^{-5})$:

$$\begin{aligned} x_3 &= \frac{-0.0194 \times 6.25 \times 10^{-5}}{(-1.282 + 0.0194)(-1.282 - 6.25 \times 10^{-5})} \times 1 + \frac{-1.282 \times 6.25 \times 10^{-5}}{(-0.0194 + 1.282)(-0.0194 - 6.25 \times 10^{-5})} \times 1.20955 \\ &\quad + \frac{-1.282 \times -0.0194}{(6.25 \times 10^{-5} + 1.282)(6.25 \times 10{-}5 + 0.0194)} \times 1.21261 = 1.21260\,, \\ y_3 &= e^{1.21260} + 3 \times 1.21261 - 7 = 1.95 \times 10^{-9}\,, \\ \epsilon &= \left|\frac{1.21260 - 1.21261}{1.21260}\right| \times 100\% = 0.00081\%\,. \end{aligned}$$

Technically we can see that we already reached the same root as both the Newton-Raphson and secant methods after 2 iterations. The relative error also decreases very rapidly each iteration. Computationally however it is definitely more expensive.

3.3.5 Müller's method

Müller's method is one of the popular methods developed for polynomials (although there is no reason stopping us from using it on other functions), and is able to solve for both real and complex roots. In other words, it is not affected by whether or not the function intersects the horizontal axis.

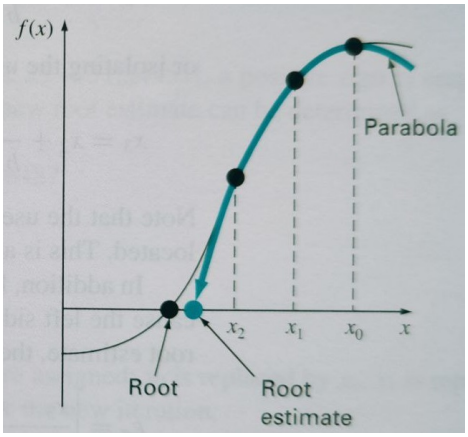


Figure 26: Illustration of Müller's method applied to a polynomial, from *Chapra & Canale*, Fig. 7.3b on pg181.

Fig. 26 illustrates the main concept. It is somewhat similar to inverse quadratic interpolation in the sense that both use a quadratic function, except that for Müller's method we do not need to write it in terms of the dependent variable y. Instead, assuming that we have 3 initial points/guesses x_{i-2}, x_{i-1} and x_i, we can fit this specific parabola:

$$f(x) = a\,(x - x_i)^2 + b\,(x - x_i) + c\,.$$

We can form a set of 3 simultaneous equations by substituting in the 3 points (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\begin{aligned} y_{i-2} &= a\,(x_{i-2} - x_i)^2 + b\,(x_{i-2} - x_i) + c \\ y_{i-1} &= a\,(x_{i-1} - x_i)^2 + b\,(x_{i-1} - x_i) + c \\ y_i &= c \end{aligned}$$

As you can see, the form of this quadratic function allows us to obtain c straightaway. The

remaining two coefficients can also be obtained easily:

$$\begin{aligned} a &= \frac{(x_{i-1}-x_i)(y_{i-2}-y_i)-(x_{i-2}-x_i)(y_{i-1})-y_i)}{(x_{i-1}-x_i)(x_{i-2}-x_i)(x_{i-2}-x_{i-1})}\,, \\ b &= \frac{y_{i-2}-y_i}{x_{i-2}-x_i} - a(x_{i-2}-x_i)\,. \end{aligned}$$

Now that we have the coefficients for the quadratic function, we can obtain the guess for the root x_{i+1} by solving $f(x) = 0$:

$$x_{i+1} - x_i = \frac{-b \pm \sqrt{b^2-4ac}}{2a}\,.$$

There is, however, the risk of error in the numerator if $b^2 >> 4ac$. So rather than the conventional quadratic formula we use an alternative formulation (obtained by multiplying top and bottom by $-b \mp \sqrt{b^2-4ac}$):

$$\begin{aligned} x_{i+1} - x_i &= \frac{-2c}{b \pm \sqrt{b^2-4ac}} \\ \Rightarrow \quad x_{i+1} &= x_i - \frac{2c}{b \pm \sqrt{b^2-4ac}}\,. \end{aligned}$$

You may notice that because we used a quadratic function, there will now be two roots (if complex, a conjugate pair); this is borne out by the $\pm$ sign in the denominator. However we are only interested in one particular root, so we will pick the sign to follow that of b. Doing so will result in a larger denominator, and thus x_{i+1} will be closer to x_i than the alternative.

Bear in mind that just as with the inverse quadratic interpolation, the 3 points used will be updated at each iteration. This means that the coefficients a, b and c also need to be determined each time. In other words, at each iteration <u>four</u> variables need to be calculated.

<u>Example</u>

Consider the function $y = x^3 - 13x - 12$ which has roots at x = -3, -1 and 4. Let's see if we can obtain the positive root ($x = 4$) for a relative error $< 1\%$.

First we choose 3 initial guesses: $x_0 = 4.5$, $x_1 = 5$ and $x_2 = 5.5$. The corresponding

functional values are $y_0 = 20.625$, $y_1 = 48$ and $y_2 = 82.875$. Then,

$$\begin{aligned}
a &= \frac{(5-5.5)(20.625-82.875)-(4.5-5.5)(48-82.875)}{(5-5.5)(4.5-5.5)(4.5-5)} = 15\,,\\
b &= \frac{20.625-82.875}{4.5-5.5} - 15(4.5-5.5) = 77.25\,,\\
c &= 82.875\,,\\
x_3 &= 5.5 - \frac{2\times 82.875}{77.25+\sqrt{77.25^2-4\times 15\times 82.875}} = 3.9765\,,\\
\epsilon &= \left|\frac{3.9765-5.5}{3.9765}\right|\times 100\% = 38.3\%\,.
\end{aligned}$$

If we choose to update sequentially, then we will be using x_1, x_2 and x_3 in the next iteration:

$$\begin{aligned}
y_3 &= 3.9765^3 - 13(3.9765) - 12 = -0.81633\,,\\
a &= \frac{(5.5-3.9765)(48+0.81633)-(5-3.9765)(82.875+0.81633)}{(5.5-3.9765)(5-3.9765)(5-5.5)} = 14.476\,,\\
b &= \frac{48+0.81633}{5-3.9765} - 14.476(5-3.9765) = 32.878\,,\\
c &= -0.81633\,,\\
x_4 &= 3.9765 - \frac{2\times -0.81633}{32.878+\sqrt{32.878^2-4\times 14.476\times -0.81633}} = 4.0011\,,\\
\epsilon &= \left|\frac{4.0011-3.9765}{4.0011}\right|\times 100\% = 0.62\%\,.
\end{aligned}$$

Actually both this and the previous guess are already very close to analytical root but we have not yet fulfilled the convergence criteria. Now using x_2, x_3 and x_4:

$$\begin{aligned}
y_4 &= 4.0011^3 - 13(4.0011) - 12 = 0.036781\,,\\
a &= \frac{(3.9765-4.0011)(82.875-0.036781)-(5.5-4.0011)(-0.81633-0.036781)}{(3.9765-4.0011)(5.5-4.0011)(5.5-3.9765)} = 13.478\,,\\
b &= \frac{82.875-0.036781}{5.5-4.0011} - 13.478(5.5-4.0011) = 35.062\,,\\
c &= 0.036781\,,\\
x_5 &= 4.0011 - \frac{2\times 0.036781}{35.062+\sqrt{35.062^2-4\times 13.478\times 0.036781}} = 4.0000\,,\\
\epsilon &= \left|\frac{4.0000-4.0011}{4.0000}\right|\times 100\% = 0.026\%\,.
\end{aligned}$$

The relative error is now $< 1\%$ so we can stop iterating here with our final guess of $x_5 = 4.0000$.

Exercise

Use Müller's method to obtain the other 2 roots (-3 and -1) for the same polynomial function above.

3.4 Multiple Roots

In some of the previous sketches and examples we implied that when there are more than one root, we may have additional complications. This affects both the bracketing and open methods. You may have some idea of what these are.

Exercise: Think about this!
The most common issues are listed below. Why do these complications arise, and how might we bypass them?

- The criteria for a good interval for bracketing methods fails when there is an even number of roots included.

- If by chance both the function and its first derivative approach zero at the root, both the Newton-Raphson and the secant methods will fail.

- Depending on the choice of initial guess and the function's curvature, sometimes it may not be possible to get the targeted root quickly, or at all.

One aspect we did not mention above is the case of multiple (or repeated) roots, e.g. $f(x) = (x-1)^2$ has 2 identical roots $x = 1$. Graphically this function is tangential to the horizontal

axis at $x = 1$. For multiple roots like these – in particular, when there are an even number of repeated roots – bracketing methods will not work because there is no sign change in the vicinity of the root.

The tangential function also means that both the function and its first derivative go to zero at the root, as we thought about earlier. Fortunately, it has been proven that typically the function will go to zero *before* its first derivative[2], and hence including a check for convergence of the functional value is a good way to avoid this issue.

Alternatively we can make use of the fact that both $f(x)$ and $f'(x)$ are zero at the root, by searching instead for the root of the function

$$u(x) = \frac{f(x)}{f'(x)} \, .$$

That is, $u(x)$ has the same roots as $f(x)$, but does not have the same issue of its derivative going to zero. Hence we will have a modified Newton-Raphson method for multiple roots:

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

Exercise

Estimate the roots of $f(x) = (x-1)(x-2)^2$ using both the standard and modified Newton-Raphson methods. You can use $x = \frac{1}{2}$ and $x = 3$ as the initial guesses.

[2]Details in Ralston, A., and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd Ed., McGraw-Hill, New York, 1978.

Compared to the standard Newton-Raphson method, the modified version requires more computational effort but is definitely more efficient for the repeated root. However for a simple root, it is slightly less efficient.

3.5 Systems of Equations

Thus far we have dealt with solving for a single root of a function which is dependent solely on one parameter. What do we do if instead we have a *set of simultaneous equations* like below, where we would need to solve the combination of several parameters?

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ \vdots &= 0 \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

It turns out that we can extend the open methods we learned about to these systems, especially when these systems are nonlinear. There exist many analytical methods for solving linear systems which you have likely learned before, and are easily translated into computer scripts, so we will only briefly describe these and focus a bit more on the iterative methods.

3.5.1 Nonlinear Equations

Although we covered quite a few open methods, we will only extend two of them to systems of nonlinear equations: fixed-point iteration and Newton-Raphson method.

Fixed-Point Iteration

The underlying concept of fixed-point iteration that we learnt before remains the same - we will rearrange the equation such that we can obtain our next guess based on the previous guesses. However because we now have more than one equation and more than one parameter, we try to rearrange such that each equation will give the next guess for a

different parameter. Let's use an example of two simultaneous equations to illustrate this process.

Example 1

Consider the set of simultaneous equations:

$$\begin{aligned} x - y^2 &= 0 \\ x^2 - y &= 0 \end{aligned}$$

Analytically, there are two sets of solutions for this: $x = 0$, $y = 0$ (this is the **trivial solution**), and $x = 1$, $y = 1$. Let's see if we can obtain the non-trivial solution.

First we will rearrange the simultaneous equations to get a formula for x and y each, as we did for the single equations previously:

$$\begin{aligned} x &= y^2 \\ y &= x^2 \end{aligned}$$

At this point we only need one initial guess (for either x or y). Let's go with $x_1 = 2$, and substitute that into the second equation. This leads us to our first guess for y:

$$y_1 = x_1^2 = 4\,.$$

Substituting y_1 into the first equation, we get our second guess for x:

$$x_2 = y_1^2 = 16\,.$$

Substituting x_2 into the second equation, we get our second guess for y:

$$y_2 = x_2^2 = 256\,.$$

We can continue on this vein, until we reach a specified relative error. Or in this case, because we can see at this point that the guesses are getting bigger and bigger, and most definitely diverging from the true value of the roots, we can stop and change tack.

Note that in Example 1, we only required one initial guess. This is not typical, normally (and intuitively) we would expect to have an initial guess for all the independent parameters involved; the only reason it turned out the way it did was because of how the formulae

we developed for x and y in Example 1 turned out. Example 2 below will demonstrate that even for a more typical case involving initial guesses for both parameters, the process is not much different.

Example 2

$$\begin{aligned} x - y^2 + xy &= 0 \\ x^2 - y + xy &= 0 \end{aligned}$$

We first rearrange these as per normal:

$$\begin{aligned} x &= y^2 - xy \\ y &= x^2 + xy \end{aligned}$$

Let's use $x_1 = 0.5$ and $y_1 = 0.5$ as initial guesses. We substitute these into the first equation to obtain a second guess for x:

$$x_2 = y_1^2 - x_1 y_1 = 0\,.$$

Now we substitute $x_2 = 0$ and $y_1 = 0.5$ into the second equation to obtain a second guess for y:

$$y_2 = x_2^2 + x_2 y_1 = 0\,.$$

We substitute both second guesses for x and y into the first equation to obtain a third guess for x:

$$x_3 = y_2^2 - x_2 y_2 = 0\,.$$

Note that $x_3 = x_2 = 0$. This shows that we have possibly already converged, but we will still have to iterate for the other parameter y. We substitute $x_3 = 0$ and $y_2 = 0$ into the second equation to obtain a third guess for y:

$$y_3 = x_3^2 + x_2 y_2 = 0\,.$$

We see that as with the x, $y_3 = y_2 = 0$, i.e. we have converged here too. Therefore, we can stop the iterating process here. If you check these values ($x = 0$ and $y = 0$) against the original equations, we find that this is actually the trivial solution.

Note that when we calculated y_2, we used the newest value of x (x_2) and the previous value of y (y_1). We could also have just gone with using only the previous values (x_1 and y_1), which you can guess will still lead to the same outcome but perhaps a little slower.

Newton-Raphson Method

Recall that we used the slope of the function at the guesses to obtain the next guess – this was simply $\frac{df}{dx}$. However now that we have more than one equation and more than one independent parameter to consider, the expression for the slope is no longer as straightforward.

Technically-speaking, the idea of using the slope came from a first order Taylor series expansion:

$$f\left(x_{i+1}\right) = f\left(x_i\right) + \left(x_{i+1} - x_i\right) f'\left(x_i\right) \ ,$$

where x_i is the current (old) guess and x_{i+1} is the new guess, where the slope intercepts the horizontal axis (and thus $f\left(x_{i+1}\right) = 0$ by definition).

So for a system with more than one parameter, we can use the multivariable form of the Taylor series expansion instead:

$$f_{i+1} = f_i + \left(x_{i+1} - x_i\right) \frac{\partial f_i}{\partial x} + \left(y_{i+1} - y_i\right) \frac{\partial f_i}{\partial y} + \left(z_{i+1} - z_i\right) \frac{\partial f_i}{\partial z} + \ldots \ .$$

Remember that we have more than one equation, so we also have to do the same for the second (and third etc.) equation:

$$g_{i+1} = g_i + \left(x_{i+1} - x_i\right) \frac{\partial g_i}{\partial x} + \left(y_{i+1} - y_i\right) \frac{\partial g_i}{\partial y} + \left(z_{i+1} - z_i\right) \frac{\partial g_i}{\partial z} + \ldots \ .$$

For simplicity, we will consider a system with only 2 parameters:

$$\begin{aligned} f\left(x, y\right) &= 0 \\ g\left(x, y\right) &= 0 \end{aligned}$$

The first order Taylor series expansions are thus:

$$\begin{aligned} f_{i+1} &= f_i + \left(x_{i+1} - x_i\right) \frac{\partial f_i}{\partial x} + \left(y_{i+1} - y_i\right) \frac{\partial f_i}{\partial y} \\ g_{i+1} &= g_i + \left(x_{i+1} - x_i\right) \frac{\partial g_i}{\partial x} + \left(y_{i+1} - y_i\right) \frac{\partial g_i}{\partial y} \end{aligned}$$

As stated earlier, $f_{i+1} = 0$ and $g_{i+1} = 0$ by definition (the intercepts with horizontal axis), so we can rearrange and manipulate the set of two equations to obtain the form 'new guess

= function involving old guess'. This turns out to be:

$$\begin{aligned} x_{i+1} &= x_i - \frac{f_i \frac{\partial g_i}{\partial y} - g_i \frac{\partial f_i}{\partial y}}{\frac{\partial f_i}{\partial x}\frac{\partial g_i}{\partial y} - \frac{\partial f_i}{\partial y}\frac{\partial g_i}{\partial x}} \\ y_{i+1} &= y_i - \frac{g_i \frac{\partial f_i}{\partial x} - f_i \frac{\partial g_i}{\partial x}}{\frac{\partial f_i}{\partial x}\frac{\partial g_i}{\partial y} - \frac{\partial f_i}{\partial y}\frac{\partial g_i}{\partial x}} \end{aligned}$$

This is the **two-equation version of the Newton-Raphson formula**. We utilise it the same way we do the simpler version.

Example 3

Let's apply this modified Newton-Raphson formula to the system used in a previous example:

$$\begin{aligned} f(x,y) &= x - y^2 &= 0\,, \\ g(x,y) &= x^2 - y &= 0\,. \end{aligned}$$

We first obtain the expressions for the partial derivatives:

$$\begin{aligned} \frac{\partial f}{\partial x} &= 1\,, & \frac{\partial f}{\partial y} &= -2y\,, \\ \frac{\partial g}{\partial x} &= 2x\,, & \frac{\partial g}{\partial y} &= -1\,. \end{aligned}$$

Substituting these expressions and the functions f and g into the modified Newton-Raphson formula, we get:

$$\begin{aligned} x_{i+1} &= x_i - \frac{\left(x_i - y_i^2\right)(-1) - \left(x_i^2 - y_i\right)(-2y_i)}{(1)(-1) - (-2y_i)(2x_i)} \\ &= x_i + \frac{\left(x_i - y_i^2\right) - 2y_i\left(x_i^2 - y_i\right)}{4x_i y_i - 1} \end{aligned}$$

and

$$\begin{aligned} y_{i+1} &= y_i - \frac{\left(x_i^2 - y_i\right)(1) - \left(x_i - y_i^2\right)(2x_i)}{(1)(-1) - (-2y_i)(2x_i)} \\ &= y_i - \frac{\left(x_i^2 - y_i\right) - 2x_i\left(x_i - y_i^2\right)}{4x_i y_i - 1}\,. \end{aligned}$$

Let our initial guesses be $x_1 = 0.1$, $y_1 = 0.1$. Substituting these into each equation, we

obtain the second guesses:

$$\begin{aligned} x_2 &= 0.1 + \frac{\left(0.1 - 0.1^2\right) - 2 \times 0.1 \left(0.1^2 - 0.1\right)}{4 \times 0.1 \times 0.1 - 1} \\ &= -0.0125\,, \\ y_2 &= 0.1 - \frac{\left(0.1^2 - 0.1\right) - 2 \times 0.1 \left(0.1 - 0.1^2\right)}{4 \times 0.1 \times 0.1 - 1} \\ &= -0.0125\,. \end{aligned}$$

Substitute the second guesses into the formulae to obtain the third guesses:

$$\begin{aligned} x_3 &= -0.0125 + \frac{\left(-0.0125 - (-0.0125)^2\right) - 2 \times -0.0125 \left((-0.0125)^2 - (-0.0125)\right)}{4 \times -0.0125 \times -0.0125 - 1} \\ &= -0.00015244\,, \\ y_3 &= -0.0125 - \frac{\left((-0.0125)^2 - (-0.0125)\right) - 2 \times -0.0125 \left(-0.0125 - (-0.0125)^2\right)}{4 \times -0.0125 \times -0.0125 - 1} \\ &= -0.00015244\,. \end{aligned}$$

As we can see, the guesses are approaching the trivial solution ($x = 0$, $y = 0$) very rapidly.

3.5.2 Linear Equations

Although our focus was largely on the more general topic of nonlinear equations, we could just as easily come across a system of linear equations e.g. when solving for the voltages and/or currents in an electrical circuit. Assuming there are n unknown variables, the system of linear equations can be rewritten as a single matrix equation $\mathbf{A}\vec{x} = \vec{b}$:

$$\begin{bmatrix} A_{11} & A_{12} & \dots & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & & A_{2n} \\ \vdots & \vdots & \ddots & & \vdots \\ A_{n1} & A_{n2} & \dots & & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Direct Methods

As mentioned earlier in the beginning of this section on systems of linear equations, there are many analytical methods for solving such systems of linear equations. These are so-called direct methods, which we will only go over briefly since you have most likely learned them before.

1. Gauss Elimination

   Here the goal is to manipulate the matrix equation $\mathbf{A}\vec{x} = \vec{b}$ until **A** has become an upper triangular matrix. The variables are solved for in reverse order: the last variable x_n is obtained first, after which it is substituted into the row above to obtain x_{n-1}. Both are then used to solve x_{n-2} in the row above, and so on.

2. Gauss-Jordan Elimination

   This is similar to Gauss Elimination except that the goal is to obtain a diagonal matrix.

3. LU Decomposition

   This is a 3-step process. First, **A** is rewritten as the product of a lower triangular matrix **L** and an upper triangular matrix **U**. The two most common ways to obtain these are via Gauss elimination and Crout's method - the only difference between the two methods is whether **L** or **U** has a leading diagonal of '1's.

   The idea behind the decomposition is that $\mathbf{A}\vec{x} = \mathbf{LU}\vec{x} = \mathbf{L}\vec{y}$. Thus the second step is to solve for $\vec{y}$ in $\mathbf{L}\vec{y} = \vec{b}$.

   Finally, solve for $\vec{x}$ in $\mathbf{U}\vec{x} = \vec{y}$.

These 3 methods can be translated into computer scripts fairly easily. The only issue that may come up is the need to rearrange rows (this is known as 'pivoting') in cases where a diagonal matrix entry is 0 or relatively small compared to other entries in the same column.

Iterative Methods

There is also an indirect (iterative) method to solve systems of linear equations, which is in essence the fixed-point iteration that we used for nonlinear systems. That is, we would rewrite the system of equations to be:

$$\begin{aligned} x_1 &= (A_{12}x_2 + A_{13}x_3 + A_{14}x_4 + \dots)/A_{11} \\ x_2 &= (A_{21}x_1 + A_{23}x_3 + A_{24}x_4 + \dots)/A_{22} \\ x_3 &= (A_{31}x_1 + A_{32}x_2 + A_{34}x_4 + \dots)/A_{33} \\ \vdots &= \vdots \end{aligned}$$

Given the (usually) large number of equations, it would be helpful to know whether the iterative process will converge. Luckily we can be guaranteed of such if the matrix is

diagonally dominant – that is, if

$$|A_{ii}| > \sum_{j=1,\, j\neq i}^{j=n} |A_{ij}|\,,$$

where A_{ii} is the i-th diagonal entry of matrix **A**. Note that this is a sufficient but not strictly necessary criteria; in other words it is possible for convergence to occur even if this criteria is not satisfied.

You may recall from before that there is the possibility of using the new values of the variables in the equations instead of the old ones if they have already been obtained in the same iteration; this is the main difference between the **Jacobi** and **Gauss-Seidel** iterative methods. We will illustrate this using a system involving 3 unknowns x, y and z:

$$\begin{aligned} x &= c_1 y + c_2 z \\ y &= c_3 x + c_4 z \\ z &= c_5 x + c_6 y \end{aligned}$$

The Jacobi iterative method updates all the variables at the end of each iteration, meaning that the iterative formulae are:

$$\begin{aligned} x_{i+1} &= c_1 y_i + c_2 z_i \\ y_{i+1} &= c_3 x_i + c_4 z_i \\ z_{i+1} &= c_5 x_i + c_6 y_i \end{aligned}$$

We will therefore need an initial guess for all the variables.

On the other hand, the Gauss-Seidel iterative method uses the newest value of the variables as soon as they are determined and hence the iterative formulae could be:

$$\begin{aligned} x_{i+1} &= c_1 y_i + c_2 z_i \\ y_{i+1} &= c_3 x_{i+1} + c_4 z_i \\ z_{i+1} &= c_5 x_{i+1} + c_6 y_{i+1} \end{aligned}$$

Here, initial guesses are needed for all the variables except the first one to be calculated (x in this example). As you may expect, the Gauss-Seidel method is faster than the Jacobi method because of the earlier usage of the new values. It also uses less memory because it does not have to store the old and new values separately.

3.6 Summary

There are two families of methods covered in this chapter. The first are bracketing methods - bisection and false-position. For both of these methods, the algorithm is:

1. Choose upper and lower points for the root, x_u and x_l respectively, such that there is a change in the sign of the function within the interval $[x_l,\, x_u]$.
2. Obtain the root estimate x_r , and the corresponding functional value $f\,(x_r)$.
3. Determine the relative error:
$$\epsilon = \left| \frac{x^{\text{new}} - x^{\text{old}}}{x^{\text{new}}} \right| \times 100\% \,.$$
4. If the relative error is larger than the specified tolerance, make a decision on refining the interval:
   - If $f\,(x_r)\, f\,(x_l) < 0$, then the new interval is $[x_l,\, x_r]$.
   - If $f\,(x_r)\, f\,(x_u) < 0$, then the new interval is $[x_r,\, x_u]$.
   - If $f\,(x_r)\, f\,(x_l) = 0$, then the root is x_r.
5. If the interval was redefined, then repeat from Step 1.

The main difference between the bisection and false-position methods is in how the root estimate is obtained:

$$x_r = \begin{cases} \frac{x_l + x_u}{2}\,, & \text{bisection method} \\ x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)}\,, & \text{false-position method} \end{cases}$$

We introduced a few open methods, which involve choosing one or more starting value(s) and then using a formula to obtain successive guesses.

- Simple Fixed-Point Iteration
  For this method, we rearrange the expression $f\,(x) = 0$ into the form $x = g\,(x)$. Each old guess x_i is substituted into the RHS to obtain the next guess x_{i+1}.

- Newton-Raphson Method

  The Newton-Raphson formula is utilises the tangent at each guess:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} .$$

- Secant Method

  Instead of the analytical expression for the slope, a finite difference using the current and previous guess is used to obtain an estimate. This leads to the formula:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} .$$

- Inverse Quadratic Interpolation

  Unlike the Newton-Raphson and Secant methods which essentially use a linear interpolation, here a quadratic function $x = g(y)$ is used instead. The formula is:

$$x_{i+1} = \frac{y_{i-1}y_i}{(y_{i-2} - y_{i-1})(y_{i-2} - y_i)} x_{i-2} + \frac{y_{i-2}y_i}{(y_{i-1} - y_{i-2})(y_{i-1} - y_i)} x_{i-1} + \frac{y_{i-2}y_{i-1}}{(y_i - y_{i-2})(y_i - y_{i-1})} x_i ,$$

  and **three** distinct initial guesses are required to start.

- Müller's method

  Similarly to inverse quadratic interpolation, three points are used to fit a quadratic function $y = f(x - x_i)$. The iteration formula is:

$$x_{i+1} = x_i - \frac{2c}{b \pm \sqrt{b^2 - 4ac}} ,$$

  where the $\pm$ sign in the denominator will follow the sign of b, and

$$\begin{aligned} a &= \frac{(x_{i-1} - x_i)(y_{i-2} - y_i) - (x_{i-2} - x_i)(y_{i-1}) - y_i)}{(x_{i-1} - x_i)(x_{i-2} - x_i)(x_{i-2} - x_{i-1})} , \\ b &= \frac{y_{i-2} - y_i}{x_{i-2} - x_i} - a(x_{i-2} - x_i) , \\ c &= y_i . \end{aligned}$$

  This method can solve for both complex and real roots.

- modified Newton-Raphson Method

  If there are repeated roots, the ratio of $f(x)$ to its first derivative $f'(x)$ is used instead. This formula is:

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{(f'(x_i))^2 - f(x_i)f''(x_i)} .$$

For systems of equations, we deal with them slightly differently depending on if the equations are linear or nonlinear. For nonlinear equations we extend two of the open methods:

- Fixed-Point Iteration

  We rearrange each of the equations in the set such that each has a different parameter's next guess on the LHS, while the RHS involves the current/old guesses of the full parameter set.

- Newton-Raphson Method

  From the first order Taylor series, we obtain an expression involving the first partial derivatives. For a 2-equation (2 parameters) system, the modified Newton-Raphson formula is:

$$\begin{aligned} x_{i+1} &= x_i - \frac{f_i \frac{\partial g_i}{\partial y} - g_i \frac{\partial f_i}{\partial y}}{\frac{\partial f_i}{\partial x}\frac{\partial g_i}{\partial y} - \frac{\partial f_i}{\partial y}\frac{\partial g_i}{\partial x}}\,, \\ y_{i+1} &= y_i - \frac{g_i \frac{\partial f_i}{\partial x} - f_i \frac{\partial g_i}{\partial x}}{\frac{\partial f_i}{\partial x}\frac{\partial g_i}{\partial y} - \frac{\partial f_i}{\partial y}\frac{\partial g_i}{\partial x}}\,. \end{aligned}$$

For systems of linear equations, they can be combined into the matrix equation $\mathbf{A}\vec{x} = \vec{b}$. Direct methods of solution include:

- Gauss Elimination - form an upper triangular matrix,
- Gauss-Jordan Elimination - form a diagonal matrix, or
- LU decomposition - split **A** into the product **LU**, where

$$\begin{aligned} \mathbf{L}\vec{y} &= \vec{b}\,, \\ \mathbf{U}\vec{x} &= \vec{x}\,. \end{aligned}$$

Indirect (iterative) methods are essentially fixed-point iteration, with slight differences in when the variables are updated:

- Jacobi Iterative Method - all variables updated at the end of each iteration.
- Gauss-Seidel Iterative Method - updated variables are used as soon as they are determined within the iteration.

A sufficient, but not necessary, convergence criteria is

$$|A_{ii}| > \sum_{j=1,\, j\neq i}^{j=n} |A_{ij}|\,,$$

where A_{ii} is the i-th diagonal entry of matrix $\mathbf{A}$.

4 Optimisation Methods

4.1 Introduction

As mentioned briefly in the overview, this topic is fundamentally similar to the previous one on finding roots. The only difference is that instead of finding the value(s) of x that give us $f(x) = 0$, we are instead more interested in the x that gives the maximum (or minimum) value of $f(x)$. In terms of practical engineering, usually we are looking for the (combination of) parameters that will give us the optimal (best) outcome – this is either a minimum in error/waste output/production time/costs, or a maximum in product quantity/process efficiency etc. Hence the name **'optimisation'**.

Logically we might expect that the methods used for optimisation are also going to be similar to what we use for root-finding, and certainly this is true. One way would be by recognising that the slope of $f(x)$ is zero at these local stationary points, and thus turn the problem into a root-finding problem (as in one of the previous examples). Unfortunately there are times when the slope cannot be found easily by analytical means, and thus techniques like finite differences are used to obtain estimates of the slope. Additionally, there are other techniques that are available to use when doing optimisation which are not suited to root-finding, and they tend to make our problem a lot easier particularly when dealing with multiple dimensions (parameters).

One marked difference from root-finding techniques, is that sometimes we also have **constraints** in optimisation problems. Basically these are restraints or limitations on the parameters, which also need to be accounted for when optimizing. In engineering applications this may be something a limit on the maximum feed rate of one component, or that production of the end result must be above a minimum temperature for the chemical reactions to take place etc.

In this section we shall look at a few groups of optimisation techniques: line searches for one-dimensional unconstrained optimisation, direct and gradient methods for multi-dimensional unconstrained optimisation, and linear programming for constrained optimisation of linear systems.

4.2 1D Unconstrained Optimisation

Just as with the root-finding methods, there are both bracketing and open methods available to us for locating maxima and minima. Here we will look at **parabolic interpolation** which is a bracketing method, and **Newton's method** which is an open method we have encountered before.

4.2.1 Parabolic Interpolation

Parabolic interpolation, as you can guess from the name, uses a 2nd order polynomial as part of the approximation to estimate the function near the desired maxima/minima. This method is popular because it allows for rapid convergence (if it *does* converge).

We know that only 3 points are required to define a parabola, that is a quadratic function. So if those 3 points bracket an optimum – the same way they do in the golden-section search – then we can use those same 3 points to approximate a 2nd order polynomial fit. You can see how easy it would then be to estimate the maxima/minima using that quadratic polynomial!

Let's say x_0, x_1 and x_2 are the 3 points within which there is a maxima/minima. Using the concepts we learned for polynomial interpolation, we obtain 3 equations for the fit $f(x) = a + bx + cx^2$:

$$\begin{aligned} a + bx_0 + cx_0^2 &= f_0 \\ a + bx_1 + cx_1^2 &= f_1 \\ a + bx_2 + cx_2^2 &= f_2 \end{aligned}$$

We can solve for the 3 unknown coefficients using these 3 equations. Recall that we want the maxima/minima, which would be at $\frac{df}{dx} = b + 2cx = 0$, or essentially at $x = -\frac{b}{2c}$.

Our next guess (x_3, to follow the numbering sequence) will thus be:

$$x_3 = \frac{f_0\left(x_1^2 - x_2^2\right) + f_1\left(x_2^2 - x_0^2\right) + f_2\left(x_0^2 - x_1^2\right)}{2f_0\left(x_1 - x_2\right) + 2f_1\left(x_2 - x_0\right) + 2f_2\left(x_0 - x_1\right)}.$$

Note that the subscripts are cycling in the order 0, 1, 2, 0, 1, 2, ... in both the numerator and denominator!

With a new guess, there are 2 ways we can proceed.

1. **Repeat the process but using the latest 3 points**.
   In other words, $x_1 \rightarrow x_0$, $x_2 \rightarrow x_1$ and $x_3 \rightarrow x_2$ (and similarly for the functional values) in the formula above.

2. **Choose the lower 3 or upper 3 points to proceed.**
   We could compare the functional values and pick the points within which there is a guaranteed minima/maxima. E.g. if $x_1 < x_2 < x_3 < x_4$ and $f_2 > f_1$, f_3 and f_4, then there is a definite maxima in the range x_1 to x_3 and so we can discard x_4.

Regardless of which option we choose to proceed with, there is the possibility that one of our limits remains unchanged (like in the false-position method). This may result in convergence being slow. We can try to ameliorate that by using careful strategies when selecting the 3 points we keep for successive iterations.

Example

Let's find the minima of $f(x) = x^3 - x^2$, in the range $0 < x \leq 1$. The analytical answer is $f\left(x = \frac{2}{3}\right) = -\frac{4}{27}$.

We need 3 points to start, so let's use $x_0 = 0.1$, $x_1 = 0.5$ and $x_2 = 1$. The corresponding functional values are $f_0 = -0.009$, $f_1 = -0.125$ and $f_2 = 0$.

Using the derived formula, our next guess will be:

$$\begin{aligned} x_3 &= \frac{-0.009\left(0.5^2 - 1^2\right) - 0.125\left(1^2 - 0.1^2\right) + 0\left(0.1^2 - 0.5^2\right)}{2\left(-0.009\right)\left(0.5 - 1\right) + 2\left(-0.125\right)\left(1 - 0.1\right) + 2\left(0\right)\left(0.1 - 0.5\right)} \\ &= 0.5417 \end{aligned}$$

For simplicity, we shall go with option 1 to proceed. So now $x_0 = 0.5$, $x_1 = 1$ and $x_2 = 0.5417$, and the functional values are $f_0 = -0.125$, $f_1 = 0$ and $f_2 = -0.1345$. The next guess will thus be:

$$\begin{aligned} x_3 &= \frac{-0.125\left(1^2 - 0.5417^2\right) + 0\left(0.5417^2 - 0.5^2\right) - 0.1345\left(0.5^2 - 1^2\right)}{2\left(-0.125\right)\left(1 - 0.5417\right) + 2\left(0\right)\left(0.5417 - 0.5\right) + 2\left(-0.1345\right)\left(0.5 - 1\right)} \\ &= 0.6301 \end{aligned}$$

At the next iteration, $x_0 = 1$, $x_1 = 0.5417$ and $x_2 = 0.6301$, and the functional values are $f_0 = 0$, $f_1 = -0.1345$ and $f_2 = -0.1469$. The next guess will thus be:

$$\begin{aligned} x_3 &= \frac{0\left(0.5417^2 - 0.6301^2\right) - 0.1345\left(0.6301^2 - 1^2\right) - 0.1469\left(1^2 - 0.5417^2\right)}{2\,(0)\,(0.5417 - 0.6301) + 2\,(-0.1345)\,(0.6301 - 1) + 2\,(-0.1469)\,(1 - 0.5417)} \\ &= 0.6457 \end{aligned}$$

For the 4th iteration, $x_0 = 0.5417$, $x_1 = 0.6301$ and $x_2 = 0.6457$, and the functional values are $f_0 = -0.1345$, $f_1 = -0.1469$ and $f_2 = -0.1477$. The next guess will thus be:

$$\begin{aligned} x_3 &= \frac{-0.1345\left(0.6301^2 - 0.6457^2\right) - 0.1469\left(0.6457^2 - 0.5417^2\right) - 0.1477\left(0.5417^2 - 0.6301^2\right)}{2\,(-0.1345)\,(0.6301 - 0.6457) + 2\,(-0.1469)\,(0.6457 - 0.5417) + 2\,(-0.1477)\,(0.5417 - 0.6301)} \\ &= 0.6679 \end{aligned}$$

Although the math can get a bit messy, it is actually quite straightforward and we see that we converge to an answer fairly quickly. If you wish to, you can try this again but using option 2 for selecting the 3 points – do you reach the converged answer faster?

Exercise

Find the 3 stationary points of the function, $f(x) = 3x^4 + x^3 - x^2$, for a relative error of $< 1\%$.

Hint: the two minima are at $x = -0.552$ and $x = 0.302$, and the maxima is at $x = 0$.

4.2.2 Newton's Method

In the section on root-finding, the 1st order Taylor series for $f(x)$ at the current guess x_i was used to derive the Newton-Raphson formula:

$$x_{i+1} = x_i - \frac{f_i}{f_i'} \,.$$

Here, because we are looking for a maxima or minima, we are essentially trying to find the root of $g(x) \equiv f'(x)$. So we can rewrite the Newton-Raphson formula for locating maxima/minima as:

$$x_{i+1} = x_i - \frac{g_i}{g_i'} = x_i - \frac{f_i'}{f_i''} \,.$$

This means that unlike root-finding, we need both the 1st and 2nd derivatives to iterate.

Just as with root-finding, we only require a single guess to start the iterative process unlike the bracketing methods we covered previously (both the golden-section search and parabolic interpolation required at least 2 initial points). We can also intuit logically that this method can converge rapidly, but has the possibility of diverging.

In the case of multiple stationary points, because we do not begin with initial guesses that bracket the value we are targeting, we will need to check the sign of the 2nd derivative to be sure that we are converging towards the desired stationary point.

Example

We shall again reuse an example from before, for simplicity: find the minima of $f(x) = x^3 - x^2$, within the range $0 < x \leq 1$, for which we know the analytical answer is $x = \frac{2}{3}$.

The required derivatives are $f'(x) = 3x^2 - 2x$ and $f''(x) = 6x - 2$, which we can substitute into the Newton-Raphson formula for locating stationary points to get:

$$x_{i+1} = x_i - \frac{3x_i^2 - 2x_i}{6x_i - 2} = \frac{3x_i^2}{6x_i - 2} \,.$$

We need the second derivative to be positive for us to get a minima, and this is true for $x > \frac{1}{3}$.

Let's use $x_0 = 1$ as the initial guess, which lies within our range for a positive 2nd deriva-

tive. The iterations will proceed as follows:

$$\begin{aligned} x_1 &= \frac{3(1)^2}{6(1)-2} = 0.75 \\ x_2 &= \frac{3(0.75)^2}{6(0.75)-2} = 0.675 \\ x_3 &= \frac{3(0.675)^2}{6(0.675)-2} = 0.6668 \\ x_4 &= \frac{3(0.668)^2}{6(0.668)-2} = 0.6667 \end{aligned}$$

As expected, we converged on the minima very quickly!

4.3 Multidimensional Unconstrained Optimisation: Direct Methods

There are two classes of methods used for solving multidimensional unconstrained optimisation problems: **gradient** and **non-gradient** methods. You can guess from the names that one class requires the function's 1st derivative to locate the maxima/minima, while the other does not. We will first look at the non-gradient methods, which are also known as **direct** methods.

4.3.1 Random Search Method

As the name sounds, the random search method entails forming a data set by evaluating the given function at randomly selected locations; the idea is that if the number of data points are large enough, the maxima (or minima) will eventually be found. In other words, this is like using trial and error.

In a computer programme, we might use a random number generator or similar algorithm to select the locations at which to evaluate the function, and constantly update the maximum (or minimum) value. If the number of independent variables increase – resulting in a large parameter space – the search process will become very tedious.

As you can imagine, this is not a very efficient method because it does not take into account the function's behaviour. Additionally, the outcome will always be the global maximum/minimum; we cannot determine any local maxima or minima.

On the other hand, the method is simple and does not require much memory space (especially if we are only storing the latest maximum/minimum value). It also works regardless of whether the function is discontinuous or non-differentiable.

4.3.2 Univariate & Pattern Searches

A more systematic approach would be the univariate search method, which only varies one parameter while keeping all others constant (rather like how we are advised to do science experiments). In this manner, the multidimensional problem then becomes a series

of one-dimensional line searches, which we have seen previously.

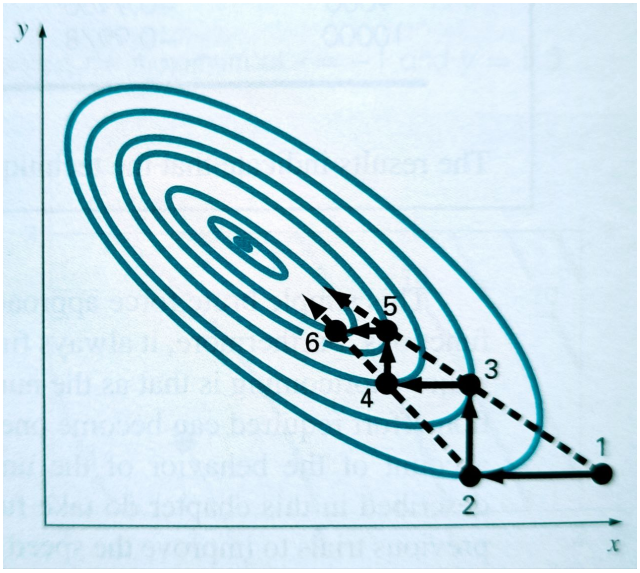


Figure 27: Sketch illustrating the concept of the univariate search method for a 2D function, from *Chapra & Canale*, Fig. 14.3 on pg370.

Fig. 27 illustrates how the univariate search method may play out for a two-dimensional function $f(x, y)$: starting from point 1 y is held constant while a line search is conducted in the x-direction. After an optimum point is found at point 2, a line search is now conducted in the y-direction but now x is held constant. This is repeated at each new optimum point found, changing the parameter being varied, until eventually the local optimum point is reached.

Example

Use the univariate search method to optimise the function $f(x, y) = (x-1)^2 + (y-1)^2$ with an initial guess of $x = 0$ and $y = 0$.

First let's hold y constant at $y = y_0$: we will thus be optimising

$$f(x, y_0) = g(x) = (x-1)^2 + (0-1)^2 = (x-1)^2 + 1\,.$$

Since this is a simple quadratic function, we can easily solve for the optimum point:

$$\begin{aligned} g' &= 2(x-1) = 0 && \rightarrow x = 1 = x_1 \\ g'' &= 2 > 0 && \rightarrow \text{local minimum} \end{aligned}$$

Or if we decided to use Newton's method, our iterative formula would be

$$x_{i+1} = x_i - \frac{g'(x_i)}{g''(x_i)} = x_i - \frac{2(x_i - 1)}{2} = 1\,,$$

so the optimal point would be at $x_1 = 1$.

Next we hold x constant at $x = x_1$: now we will be optimising

$$f(x_1, y) = g(x) = (1-1)^2 + (y-1)^2 = (y-1)^2\,.$$

Without going through the full working, we will find the optimal point to be at $y_1 = 1$.

Now, $y = y_1$ and optimise

$$f(x, y_1) = g(x) = (x-1)^2 + (1-1)^2 = (x-1)^2\,.$$

We find that $x_2 = x_1 = 1$, so we have converged in x. Since the$x$ value is unchanged, it means that we will also be achieving the same outcome for y. Thus the optimal point is $x = 1$, $y = 1$ which is the same as the analytical answer.

If we look again at Fig. 27, we notice that if we extend a dashed line through some of the points they seem to point towards the local optimum point. These trajectories that look like shortcuts, are in fact what we would call **pattern directions**, and they can be made use of in algorithms to make the search process more efficient. One such algorithm we will expand on is **Powell's method**.

The main idea underlying Powell's method is this: if we did a line search along a certain direction but from two different starting points, we would get two different ending points. Then:

- the line joining these two points will then point towards the local maximum/minimum; and
- this line and the search direction will form a pair of **conjugate directions**.

For example in Fig. 27, both points 2 and 4 were obtained by searching along the same direction but from different starting points. Hence the search direction (x-direction) and

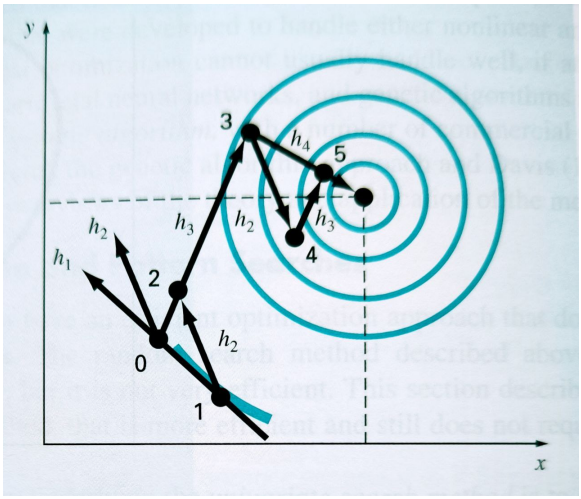


Figure 28: Sketch illustrating the concept of Powell's method for a 2D function, from *Chapra & Canale*, Fig. 14.5 on pg372.

the dashed line joining points 2 and 4 are conjugate directions. Similarly the y-direction and the dashed line joining points 3 and 5 are another pair of conjugate directions.

Fig. 28 demonstrates Powell's method for a function $f(x, y)$. The process begins at point 0 with 2 chosen directions $\vec{h}_1$ and $\vec{h}_2$ (which are not necessarily conjugate directions).

1. A search is done along $\vec{h}_1$, the outcome of which is point 1.
2. Point 2 is located by searching from point 1 along $\vec{h}_2$.
3. A new direction $\vec{h}_3$ is defined by joining points 0 and 2. This will be one half of a pair of conjugate directions.
4. From point 2, search along $\vec{h}_3$ to find point 3.
5. From point 3, search along $\vec{h}_2$ to find point 4.
6. From point 4, search along $\vec{h}_3$ to find point 5.
7. Both points 3 and 5 were found by searching along $\vec{h}_3$, so the line joining them ($\vec{h}_4$) will form the other half of the pair of conjugate directions.
8. From point 5, search along $\vec{h}_4$ to find the local maximum/minimum.

Example

Use Powell's method to optimise the same function $f(x,y) = (x-1)^2 + (y-1)^2$ with an initial guess of $x = 0$ and $y = 0$. The initial search directions are $\vec{h}_1 = (1,0)$ and $\vec{h}_2 = (1,1)$.

The first step is to do a search along $\vec{h}_1$, starting from point 0 (0,0). So we expect that point 1 will be some distance away from point 0 that is a multiple α of $\vec{h_1}$. In other words, the coordinates would be:

$$\begin{aligned} x_1 &= x_0 + \alpha(1) = \alpha\,, \\ y_1 &= y_0 + \alpha(0) = 0\,. \end{aligned}$$

We substitute these into $f(x,y)$ to get a 1D function for optimising:

$$g(\alpha) \equiv f(x_1, y_1) = (\alpha - 1)^2 + (0-1)^2 = (\alpha - 1)^2\,.$$

Analytically, the optimal point is at $\alpha = 1$, hence point 1 is at (1,0).

The second step is to do a search along $\vec{h}_2$, starting from point 1. Point 2 will therefore be at:

$$\begin{aligned} x_2 &= x_1 + \alpha(1) = 1 + \alpha\,, \\ y_2 &= y_1 + \alpha(1) = \alpha\,. \end{aligned}$$

We substitute these into $f(x,y)$ to get a 1D function for optimising:

$$g(\alpha) \equiv f(x_2, y_2) = (1 + \alpha - 1)^2 + (\alpha - 1)^2 = 2\alpha^2 - 2\alpha + 1\,.$$

Analytically, the optimal point is at $\alpha = \frac{1}{2}$, hence point 2 is at $\left(\frac{3}{2}, \frac{1}{2}\right)$.

We obtain a new direction $\vec{h}_3$ by joining points 0 and 2:

$$\vec{h}_3 = \vec{x}_2 - \vec{x}_0 = (\frac{3}{2}, \frac{1}{2}) = \frac{1}{2}(3,1)\,.$$

Since it is the ratio of components that is important, not the exact values, we will use a convenient multiple: $\vec{h}_3 = (3,1)$.

Now we do a search along $\vec{h}_3$, starting from point 2. Point 3 will be at:

$$\begin{aligned} x_3 &= x_2 + \alpha(3) = \frac{3}{2} + 3\alpha\,, \\ y_3 &= y_2 + \alpha(1) = \frac{1}{2} + \alpha\,. \end{aligned}$$

We substitute these into $f(x, y)$ to get a 1D function for optimising:

$$g(\alpha) \equiv f(x_3, y_3) = (\frac{3}{2} + 3\alpha - 1)^2 + (\frac{1}{2} + \alpha - 1)^2 = 10\alpha^2 + 2\alpha + \frac{1}{2}\,.$$

Analytically, the optimal point is at $\alpha = -\frac{1}{10}$, hence point 3 is at $\left(\frac{3}{2} - \frac{3}{10}, \frac{1}{2} - \frac{1}{10}\right) = \left(\frac{6}{5}, \frac{2}{5}\right)$.

Next we search along $\vec{h}_2$, starting from point 3. Point 4 will be at:

$$\begin{aligned} x_4 &= x_3 + \alpha(1) = \frac{6}{5} + \alpha\,, \\ y_4 &= y_3 + \alpha(1) = \frac{2}{5} + \alpha\,. \end{aligned}$$

The 1D function for optimising is:

$$g(\alpha) \equiv f(x_4, y_4) = (\frac{6}{5} + \alpha - 1)^2 + (\frac{2}{5} + \alpha - 1)^2 = 2\alpha^2 - \frac{4}{5}\alpha + \frac{2}{5}\,.$$

Analytically, the optimal point is at $\alpha = \frac{1}{5}$, hence point 4 is at $\left(\frac{6}{5} + \frac{1}{5}, \frac{2}{5} + \frac{1}{5}\right) = \left(\frac{7}{5}, \frac{3}{5}\right)$.

We do another search along $\vec{h}_3$ from point 4, to obtain point 5:

$$\begin{aligned} x_5 &= x_4 + \alpha(3) = \frac{7}{5} + 3\alpha\,, \\ y_5 &= y_4 + \alpha(1) = \frac{3}{5} + \alpha\,. \end{aligned}$$

The 1D function for optimising is:

$$g(\alpha) \equiv f(x_5, y_5) = (\frac{7}{5} + 3\alpha - 1)^2 + (\frac{3}{5} + \alpha - 1)^2 = 10\alpha^2 + \frac{8}{5}\alpha + \frac{8}{25}\,.$$

Analytically, the optimal point is at $\alpha = -\frac{2}{25}$, hence point 5 is at $\left(\frac{7}{5} - \frac{6}{25}, \frac{3}{5} - \frac{2}{25}\right) = \left(\frac{29}{25}, \frac{13}{25}\right)$.

The second-last step is to define a direction $\vec{h}_4$ by joining points 3 and 5, which is conjugate to $\vec{h}_3$:

$$\vec{h}_4 = \vec{x}_5 - \vec{x}_3 = \left(\frac{29}{25}, \frac{13}{25}\right) - \left(\frac{6}{5}, \frac{2}{5}\right) = \left(\frac{-1}{25}, \frac{3}{25}\right)\,.$$

Since it is the ratio of components that is important, not the exact values, we will use a convenient multiple: $\vec{h}_4 = (-1, 3)$.

Finally we do a search along $\vec{h}_4$ from point 5 to obtain point 6:

$$\begin{aligned} x_6 &= x_5 + \alpha(-1) = \frac{29}{25} - \alpha\,, \\ y_6 &= y_5 + \alpha(3) = \frac{13}{25} + 3\alpha\,. \end{aligned}$$

The 1D function for optimising is:

$$g(\alpha) \equiv f(x_6, y_6) = (\frac{29}{25} - \alpha - 1)^2 + (\frac{13}{25} + 3\alpha - 1)^2 = 10\alpha^2 - \frac{16}{5}\alpha + \frac{32}{125}\,.$$

Analytically, the optimal point is at $\alpha = \frac{4}{25}$, hence point 6 is at $\left(\frac{29}{25} - \frac{4}{25}, \frac{13}{25} + \frac{12}{25}\right) = (1, 1)$.

Powell's method will bring us directly to the optimum if the function is quadratic, and close by if it is not. Hence for our function we expect that this is already the optimum; we can check by calculating the gradient at this location:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = \left(2\left(x_6 - 1\right), 2\left(y_6 - 1\right)\right) = (0, 0)\,.$$

The gradient is zero and hence we can confirm that we have reached the optimum point.

4.4 Multidimensional Unconstrained Optimisation: Gradient Methods

As mentioned previously, **gradient methods** make use of the function's 1st derivative (slope). We have already encountered **Newton's method** in the previous section on one-dimensional unconstrained optimisation, which can be classified as a gradient method. Since we are already familiar with it, we shall look at the multidimensional version of the Newton's method, as well as the **Steepest Ascent Method** and the **Marquardt method**.

Unlike one-dimensional functions, differentiation of multidimensional functions are no longer as straightforward because of all the *partial* derivatives involved. This can get messy when cross-derivatives come into play, which occurs when we compute the second derivative to check whether our stationary point is a maxima or a minima.

Here we will be encountering something called the **Hessian** rather frequently. This is a matrix which is composed of partial derivatives.

For a 2D function $f = f(x, y)$, the Hessian is defined as:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix},$$

and its determinant is:

$$\det(\mathbf{H}) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2.$$

The determinant of the Hessian tells us the nature of the stationary point:

- If $\det(\mathbf{H}) > 0$ and $\frac{\partial^2 f}{\partial x^2} > 0$, we have a local minimum.
- If $\det(\mathbf{H}) > 0$ and $\frac{\partial^2 f}{\partial x^2} < 0$, we have a local maximum.
- If $\det(\mathbf{H}) < 0$, we have a saddle point.

Logically we can guess what the Hessian matrix will look like for a 3D system and higher, but the rule that we use for determining whether we have a local maxima, local minima or a stationary point remains unchanged.

4.4.1 Steepest Ascent Method

The steepest ascent method is perhaps the most intuitive method for multidimensional functions. Let's say we place a ball on a slope. Assuming that it had no initial velocity, the ball will move along the steepest downward direction. If the slope remains the steepest all the way down in a straight line, then the ball will reach the valley rapidly.

However what happens if the surface that the ball is on is not so simple? In that case, the path the ball is moving on will eventually no longer be the steepest direction. We know that if the ball were moving slowly enough and the change in slope is dramatic enough, the ball will change direction. However that is not always the case and thus the ball will not reach the valley in the shortest time possible.

If we could directly control the motion of the ball, then we could evaluate the steepest downwards slope every now and then. This will help us (and the ball) to take the quickest path to the valley. Conceptually, this is still fairly straightforward.

The question now is how often we have to make this evaluation. Obviously we would need to evaluate very frequently so that we can adjust rapidly. But this is an inefficient process, and computationally expensive. So one option could be to move along the first 'steepest slope' without changing direction until the slope becomes flat. Then we determine the new 'steepest slope' and move again. Admittedly this is going to be slower than the previous approach in terms of directness, but we would save in computation time.

Fig. 29 illustrates this method, where the rings are contours of $f(x, y)$. The goal is to reach the single dot in the centre of the rings, which is either a local maxima or local minima. Our starting point is labelled '0', where the steepest slope is in the direction of vector $\vec{h_0}$. We move along that vector until we hit a contour tangentially – this means that the slope is now zero. Thus we stop here (this point is labelled '1'), and reevaluate. $\vec{h_1}$ is now the direction of steepest slope, so we proceed along in this direction until we once again encounter a contour tangentially (this occurs at point '2'). As you can see, by following the steepest ascent method, we will trace a somewhat zigzag path to the local maxima/minima.

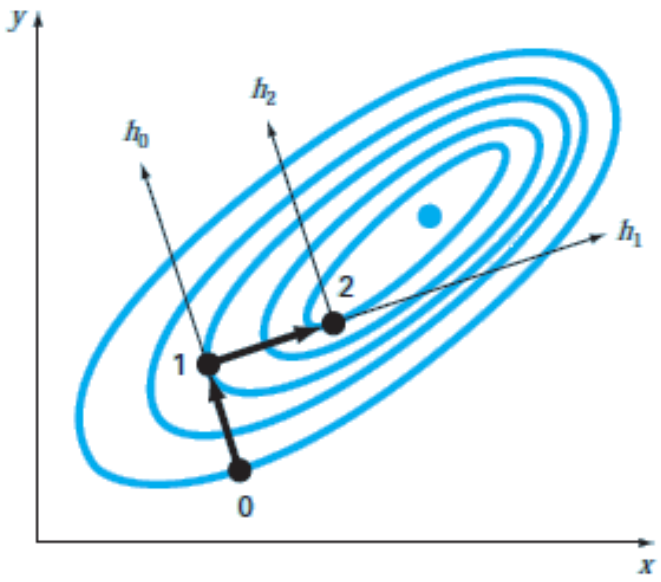


Figure 29: Sketch illustrating the concept of the steepest ascent method for a 2D function, from *Chapra & Canale*, Fig. 14.9 on pg379.

Our steepest slope is the **gradient vector**:

$$\nabla f = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j}\,.$$

As we move along the gradient vector, our coordinates will change as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + h \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix},$$

or

$$\begin{aligned} x &= x_0 + \left.\frac{\partial f}{\partial x}\right|_0 h\,, \\ y &= y_0 + \left.\frac{\partial f}{\partial y}\right|_0 h\,. \end{aligned}$$

where h times the gradient vector is the displacement (vector). We can thus think of h as the parameter controlling how far along the gradient vector we travel.

By substituting in the values of x_0, y_0 and the partial derivatives making up the gradient vector, we can rewrite our function $f(x, y)$ as a one-dimensional function $g(h)$ when travelling along this particular gradient vector. Recall that we are supposed to travel along this

vector until the slope is flat; this means that all we need to do now is determine the local maxima/minima for $g(h)$. We know how to deal with such a one-dimensional function from the previous section!

To sum it up, what we have done is to change a multidimensional optimisation to a series of one-dimensional optimisations. The general procedure for one iteration is as follows:

1. Determine the gradient vector ∇f.
2. Determine the Hessian matrix $\mathbf{H}$.
   (a) Calculate the determinant of the Hessian matrix.
   (b) Check whether the stationary point we are interested in is a local maximum, local minimum, or a saddle point.
3. Choose a starting point (x_0, y_0) and evaluate the corresponding gradient vector there.
4. Determine the 1D function $g(h)$ along the gradient vector by substituting
   $$x = x_0 + \left.\frac{\partial f}{\partial x}\right|_0 h$$
   and
   $$y = y_0 + \left.\frac{\partial f}{\partial y}\right|_0 h$$
   into $f(x, y)$.
5. Locate h^*, the value of h that results in a local maxima/minima of $g(h)$.
   - Use the golden-section search, or
   - use parabolic interpolation, or
   - use Newton's method, or
   - solve analytically if simple enough.
6. Calculate the next starting point (x_1, y_1) by plugging h^* back into our expressions for x and y:
   $$x_1 = x_0 + \left.\frac{\partial f}{\partial x}\right|_{x_0} h^*,$$
   and
   $$y_1 = y_0 + \left.\frac{\partial f}{\partial y}\right|_{y_0} h^*.$$

Example

Optimize the function $f(x,y) = xy - x^2 - y^2$, using initial guesses $x = 1$ and $y = 1$.

The relevant partial derivatives for this function are:

$$\begin{aligned} \frac{\partial f}{\partial x} &= y - 2x \\ \frac{\partial^2 f}{\partial x^2} &= -2 \\ \frac{\partial^2 f}{\partial x \partial y} &= 1 \\ \frac{\partial f}{\partial y} &= x - 2y \\ \frac{\partial^2 f}{\partial y^2} &= -2 \end{aligned}$$

Thus the gradient vector is:

$$\nabla f = (y - 2x)\,\vec{i} + (x - 2y)\,\vec{j}\,,$$

and the determinant of the Hessian matrix $\mathbf{H}$ is:

$$\det(\mathbf{H}) = (-2)(-2) - (1)^2 = 3 > 0\,.$$

Since $\frac{\partial^2 f}{\partial x^2} < 0$, the optimized point is a local maximum.

We can solve analytically for this saddle point by setting the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ to zero and solving for x and y – the local maximum will be at $x = 0$, $y = 0$.

Using the initial guesses $x_0 = 1$ and $y_0 = 1$, we obtain the first 'steepest slope':

$$\nabla f = (1-2)\,\vec{i} + (1-2)\,\vec{j} = -\vec{i} - \vec{j}\,.$$

We use this to obtain the one-dimensional function $g(h)$:

$$\begin{aligned} f(x,y) &= f\left(x_0 + \left.\frac{\partial f}{\partial x}\right|_0 h,\; y_0 + \left.\frac{\partial f}{\partial y}\right|_0 h\right) \\ &= f(1-h, 1-h) \\ &= (1-h)(1-h) - (1-h)^2 - (1-h)^2 \\ &= -1 + 2h - h^2 \\ \Rightarrow \qquad g(h) &= -1 + 2h - h^2\,. \end{aligned}$$

Since this is a simple enough function we can solve it analytically, by setting $\frac{dg}{dh} = 0$. This gives us $h^* = 1$, and therefore our next guesses for x and y at the end of this iteration are:

$$\begin{aligned} x_1 &= 1 - h^* = 0 \\ y_1 &= 1 - h^* = 0 \end{aligned}$$

For the second iteration, we need to reevaluate the gradient vector:

$$\nabla f = (0 - 0)\,\vec{i} + (0 - 0)\,\vec{j} = \vec{0}\,.$$

Since the gradient vector is the zero vector, we know that we have already arrived at the local maximum.

In general we will require more than 1 iteration to reach the optimized point. What we have here – requiring only 1 iteration – is what we would call the **optimal steepest ascent**.

Exercise

Optimize the function $f(x, y) = (x - 1)^3 + (y - x)^2$, using initial guesses $x = y = 2$.

Ans: (1,1)

4.4.2 Newton's Method

Do you recall how we extended the Newton-Raphson method for root-finding in 1D for a set of coupled multidimensional functions? For a system with n functions and n independent variables, we expanded the Taylor series of each of those Taylor series to the first order (i.e. up to the first partial derivative of each independent variable). In this manner we obtained n equations for n unknowns (the next guesses) in terms of known values (the current guesses as well as the functional values and first derivatives).

We will do something similar here. What we have is a *single* function f depending on n independent variables (x, y, z, ...). In order to optimize f, we essentially are setting:

$$\begin{aligned} \frac{\partial f}{\partial x} &= 0 \\ \frac{\partial f}{\partial y} &= 0 \\ \frac{\partial f}{\partial z} &= 0 \\ &\vdots \end{aligned}$$

So we now have n coupled equations for which we require the root(s).

Since it is the 1st order derivative that we are finding roots for, we need to expand the Taylor series for the 1st order derivatives:

$$\nabla f_{i+1} = \nabla f_i + \mathbf{H}_i \left(\vec{x}_{i+1} - \vec{x}_i \right) .$$

$\vec{x}_i \equiv (x_i, y_i, z_i, \ldots)^T$ is the vector formed using the n independent variables, while ∇f_i and $\mathbf{H}_i$ are respectively the gradient vector and Hessian matrix evaluated at position $\vec{x}_i$.

Since the gradient vector is zero at the optimized point, the Taylor series expansion becomes:

$$\vec{0} = \nabla f_i + \mathbf{H}_i \left(\vec{x}_{i+1} - \vec{x}_i \right) .$$

Now if the Hessian matrix is non-singular (that is, an inverse exists), then we can solve for $\vec{x}_{i+1}$:

$$\vec{x}_{i+1} = \vec{x}_i - \mathbf{H}_i^{-1} \nabla f_i \, .$$

Unsurprisingly, this is very similar to the Newton's method formula for optimizing one-dimensional functions:

$$x_{i+1} = x_i - \frac{f_i'}{f_i''}\,,$$

except that it has vectors and matrices.

Example

Let's optimize the same function from before, $f(x, y) = xy - x^2 - y^2$, using the same initial guesses $x = 1$ and $y = 1$.

We will require both the gradient vector and Hessian matrix, so the partial derivatives of 1st and 2nd order need to be calculated:

$$\begin{aligned}
\frac{\partial f}{\partial x} &= y - 2x \\
\frac{\partial^2 f}{\partial x^2} &= -2 \\
\frac{\partial^2 f}{\partial x \partial y} &= 1 \\
\frac{\partial f}{\partial y} &= x - 2y \\
\frac{\partial^2 f}{\partial y^2} &= -2
\end{aligned}$$

Thus the gradient vector is:

$$\nabla f = (y - 2x)\,\vec{i} + (x - 2y)\,\vec{j}\,,$$

and the determinant of the Hessian matrix $\mathbf{H}$ is:

$$\det(\mathbf{H}) = (-2)(-2) - (1)^2 = 3 > 0\,.$$

Since $\frac{\partial^2 f}{\partial x^2} < 0$, the optimized point is a local maximum.

We also require the inverse of the Hessian matrix:

$$\begin{aligned}
\mathbf{H}^{-1} &= \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}^{-1} \\
&= -\frac{1}{3}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}
\end{aligned}$$

Fortunately for us the Hessian matrix (and hence its inverse) are constants, independent of x and y, so we will not have to recalculate at each iteration.

We now proceed to iterate:

$$\begin{aligned}
\vec{x}_1 &= \vec{x}_0 - \mathbf{H}^{-1}\nabla f_0 \\
&= \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 1-2\,(1) \\ 1-2\,(1) \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} -3 \\ -3 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}$$

$x = 0$, $y = 0$ is the correct answer as we know, but we can confirm this by calculating the gradient vector at this point - we will obtain the zero vector.

It is not obvious here, but the Newton's method outperforms the steepest ascent method in terms of speed. It has a slightly less involved procedure, and it also converges more rapidly (if it does not diverge). However it does have the major drawback of (typically) needing to calculate the second derivatives for the Hessian matrix's inverse at every iteration. As a result this method may not be the best choice if a large number of independent variables are involved as the Hessian matrix will get larger. The previously mentioned drawback is also present - that if the initial guess is not close enough, Newton's method may not converge.

4.4.3 Marquardt Method

The Marquardt method is actually a combination of both the steepest ascent and Newton's methods: the steepest ascent method works very well when far away from the target, while Newton's method converges quickly (and more efficiently) when nearby. In order to combine these two, the Marquardt method uses a slightly modified formulation of the Hessian in the usual Newton's method formula:

$$\vec{x}_{i+1} = \vec{x}_i - \mathbf{H}_i^{-1}\nabla f\,,$$

where

$$\tilde{\mathbf{H}}_i = \mathbf{H}_i + \alpha_i \mathbf{I}\,,$$

and α_i is a positive constant.

Initially, α_i is set of be a large value, such that the term $\alpha_i \mathbf{I}$ dominates in the expresssion for $\tilde{\mathbf{H}}_i$ and hence $\tilde{\mathbf{H}}_i^{-1} \approx \frac{1}{\alpha_i}\mathbf{I}$. This is similar to the steepest ascent method's formulation.

Then as the iterations increase (and the optimised location is approached), the value of α_i decreases towards zero – thus moving towards Newton's method.

In practice, the initial value of α_i is either an educated guess or a random value (e.g. 10^4). At the end of each iteration, the function f is compared to the previous value to determine whether the search process is getting closer or farther; if closer then α_i is decreased and vice versa. In this manner the Marquardt method allows for flexibility in shifting between the steepest ascent and Newton's methods adaptively as the iterations proceed, so overshooting is not a concern. How α_i is decreased is up to the user, for instance a constant factor of 0.8 for decreasing and a factor of 2 for increasing.

Example

Let's the Marquardt method to optimise the same function $f(x, y) = xy - x^2 - y^2$ with an initial guess of $x = 1$ and $y = 1$. We can let our initial value of α be 2 (since we know the analytical root of (0,0) is quite close).

We already know the gradient vector, Hessian matrix and its determinant from before:

$$\nabla f = \begin{bmatrix} y - 2x \\ x - 2y \end{bmatrix}\,,$$

$$\begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}\,,$$

and

$$\det(\mathbf{H}) = (-2)(-2) - (1)^2 = 3 > 0\,.$$

Since $\frac{\partial^2 f}{\partial x^2} < 0$, the optimized point is a local maximum.

The modified Hessian is thus

$$\begin{bmatrix} \alpha - 2 & 1 \\ 1 & \alpha - 2 \end{bmatrix}$$

and its inverse

$$\frac{1}{(\alpha-2)^2 - 1} \begin{bmatrix} \alpha - 2 & -1 \\ -1 & \alpha - 2 \end{bmatrix}.$$

The functional value at our starting point, $f_0 = f(1,1) = -1$.

We now proceed to iterate:

$$\begin{aligned} \vec{x}_1 &= \vec{x}_0 - \tilde{\mathbf{H}}_0^{-1} \nabla f_0 \\ &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{1}{(2-2)^2 - 1} \begin{bmatrix} 2-2 & -1 \\ -1 & 2-2 \end{bmatrix} \begin{bmatrix} 1 - 2\,(1) \\ 1 - 2\,(1) \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned}$$

At (0,0), the gradient vector is zero so we have already reached the optimum point. If we had not, then we would compare $f_1 = f(0,0)$ against f_0 to see whether we are getting closer to the local maximum and decide how to update α for the next iteration.

4.5 Constrained Optimisation

In real life situations, we are more likely to encounter constrained optimisation problems rather than unconstrained optimisation problems as in the previous section. The constraints are usually inequalities relating to physical reality (e.g. parameters like mass cannot be negative) or describing a limited amount of resources (e.g. the total time used for tasks in a single day cannot exceed 24 hours). Mathematically this means that on top of optimising the objective function, we also need to take into account the constraints – these can be represented as additional unknown variables or equations. Hence if we used the techniques for unconstrained optimisation to obtain solutions, if they do not satisfy the constraints these solutions would not be **feasible**.

4.5.1 Linear Programming

As you might imagine, **linear programming** is the family of numerical methods used to optimise constrained linear systems. That is, both the objective function $Z(x_1, x_2, ...x_n)$ and the constraints are linear functions of the independent variables x_1, x_2, ... x_n. In other words:

$$Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

is the objective function for which $x_i \geq 0$ $(1 \leq i \leq n)$, and there may exist m constraints of the form

$$a_{j1}x_1 + a_{j2}x_2 + \ldots a_{jn}x_n \leq b_j\,, \qquad 1 \leq j \leq m\,.$$

The 'programming' in the name has nothing to do with computer scripts, but rather has the meaning of 'scheduling'. So linear programming is a way of manipulating or (re)arranging the independent variables such that the objective function reaches its optimal value.

One of the more intuitive ways to solve a constrained optimisation problem is to look at it graphically: the extreme case of each constraint can be plotted in the n-dimensional domain space, and the area bounded by all the constraints contain all the feasible solutions of the objective function. We then only need to search in that bounded area for the optimal solution.

For instance let's consider a 2D function $Z(x, y) = 6x + 8y$, which has the constraints

$$\begin{aligned} x &\geq 0 \\ y &\geq 0 \\ 5x + 2y &\leq 40 \\ 6x + 6y &\leq 60 \\ 2x + 4y &\leq 32 \end{aligned}$$

The limiting case (i.e. '=' instead of '$\leq$' or '$\geq$') for all these form a set of lines which are plotted in Fig. 30. The shaded zone ABCDE is the portion of the domain in which all feasible solutions of $Z(x, y)$ exist.

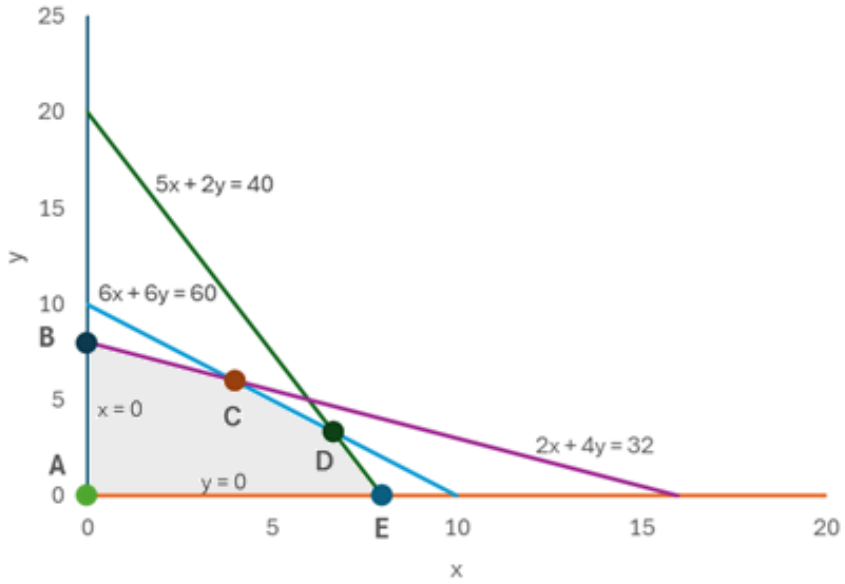


Figure 30: Graph showing the zone ABCDE bordered by the set of constraints.

Since we are interested in optimising Z, then we would look at the extremes of the zone - in other words the vertices A(0,0), B(0,8), C(4,6), D(6.67, 3.33) and E(8,0). After evaluating Z at each of these points, we find that the minimum is 0 at point A and 72 at point C.

You may realise that if we do not have the luxury of plotting a graph, or if there are multiple constraints in multiple dimensions, then it will be difficult to obtain a graphical solution. How would we handle this then?

First let's remove the inequalities from the 'resource-type' constraints, by introducing a **slack variable** S_j. You can think of this as the leftover resources that make up the maximum amount b_j. Our general constraint will then be written as:

$$a_{j1}x_1 + a_{j2}x_2 + \ldots a_{jn}x_n + S_j = b_j\,, \qquad 1 \leq j \leq m\,.$$

If $S_j = 0$, we get the equation corresponding to the extreme case of the constraint, which is also the function that we plotted to obtain the graphical solution.

However we now have a total of $m + n$ unknowns (n independent variables x_i and m slack variables S_j) but only m equations (from the constraints), which makes this system underdetermined. One way to deal with this is to remove n variables by setting them to zero. We call these non-zero variables **basic variables** and the zeroed variables **nonbasic variables**.

If we go back to the earlier example which is a 2D system with 3 constraints, we could choose to set two of the slack variables to be zero. Physically that means that we would be pushing two of the constraints to the limit (hence there would be no slack), and our solution would depend on the remaining 3 variables (x, y and one slack variable). Graphically that means we would get a solution which is at the intersection of the lines corresponding to the nonbasic variables. E.g. point C in Fig. 30 is the basic solution obtained when the slack variables corresponding to the constraints $6x + 6y \leq 60$ and $2x + 4y \leq 32$ are set to zero.

However notice that there are other intersections outside of the shaded zone ABCDE, e.g. (10,0) which is the location of the basic solution obtained when y and the slack variable in $6x + 6y \leq 60$ are set to zero. We can see that it does not satisfy one of the other constraints; hence this solution would be not be considered **feasible**.

Thus that means that in order to find the optimal solution, we would need to compare only the basic feasible solutions. For a n-dimensional problem with m 'resource-type' constraints, the total number of basic solutions is

$$C_m^{n+m} = \frac{(n+m)!}{m!\,n!} = C_n^{n+m}\,,$$

of which typically only a small portion is feasible. We can expect reasonably that obtaining the optimal solution is going to be quite inefficient, especially if m and n are large.

Simplex Method

The Simplex method is a technique that avoids the non-feasible basic solutions, and works in a manner reminiscient of the univariate search method. The general procedure is as follows:

1. Write out m+1 equations (the objective function Z and m constraint equations containing the slack variables S_j) in matrix/table form.
2. Identify the **entering** and **leaving** variables.
3. Apply Gaussian elimination: using the row corresponding to the **leaving** variable as the pivot row, eliminate all the elements in other rows that are in the same column as the **entering** variable (i.e. pivot element).
4. Repeat the previous two steps until no nonbasic variables are left that will optimise Z further.

We will illustrate this process using the example of $Z = 6x + 8y$ and corresponding constraints. Since there are 3 constraints, there are a total of $m + 1 = 3 + 1 = 4$ equations:

$$\begin{aligned} Z - 6x - 8y &= 0 \\ 5x + 2y + S_1 &= 40 \\ 6x + 6y + S_2 &= 60 \\ 2x + 4y + S_3 &= 32 \end{aligned}$$

Note that we have 5 independent variables (x, y, S_1, S_2 and S_3) but only 3 equations (the constraint equations) to determine them. The objective function Z is not included among them because we will be using that to determine the dependent function Z. So essentially we will need to select 2 variables to be the nonbasic variables (set to 0) and the remaining 3 will be the basic variables (assumed non-zero).

We first express the equations in matrix form:

$$\begin{bmatrix} 1 & -6 & -8 & 0 & 0 & 0 \\ 0 & 5 & 2 & 1 & 0 & 0 \\ 0 & 6 & 6 & 0 & 1 & 0 \\ 0 & 2 & 4 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Z \\ x \\ y \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 40 \\ 60 \\ 32 \end{bmatrix}$$

.

Note that if we were to select x and y as the nonbasic variables, then substituting in $x = 0$ and $y = 0$ into these equations will allow us to directly obtain $Z = 0$, $S_1 = 40$, $S_2 = 60$ and $S_3 = 32$. So we can view this as our starting point or an initial basic feasible solution from which we will proceed to incrementally optimise Z. In Fig. 30 this would be point A.

An alternative, perhaps more concise, way of writing these equations would be in a table, as follows:

	Z	x	y	S_1	S_2	S_3	soln
Z	1	-6	-8	0	0	0	0
S_1	0	5	2	1	0	0	40
S_2	0	6	6	0	1	0	60
S_3	0	2	4	0	0	1	32

The first column contains the basic variables, each corresponding to the equation which directly solves for them when the nonbasic variables (currently x and y) are 0, and the solutions are the last column. As we implement the simplex method, we should find the solution for Z improving each time towards the optimal solution.

Now we need to identify the **entering** and **leaving** variables. Essentially we are going to exchange one of the basic and nonbasic variables, such that Z becomes more optimised. This is done by considering what the goal is - in this case, we want to find the maximum of Z, thus the manipulation of variables should result in Z increasing.

For the entering variable, we choose the nonbasic variable in the current version of the

objective function (i.e. row 1 of the matrix/table) which has the most negative coefficient. Thus we will obtain the largest increase in Z when the selected variable goes from zero to non-zero (it will become positive, because we know that all the independent variables have to be positive). In our example, y has the most negative coefficient (-8) and thus this will be the entering variable.

For the leaving variable, we consider how the existing basic variables change as the entering variable becomes non-zero. Specifically, we need to know what the value of the entering variable is when the potential leaving variable becomes zero. This is easily determined from each row of the matrix/table: it would be the 'solution' (RHS column) divided by the coefficient of the entering variable. We need to select the one which gives us the least positive value. Why? Firstly because as mentioned before, all the independent variables are positive; and secondly the 'innermost' constraints form the bounds of the feasible solution space (you can refer back to Fig. 30). Basically, as the entering variable increases, the first constraint that it encounters will be a basic feasible solution while all the successive ones will be nonfeasible.

In our example, row 2 gives us $y = 20$, row 3 gives $y = 10$ and in row 4 $y = 8$. Hence we will choose S_3 (corresponding to row 4) to be the leaving variable. In Fig. 30, these correspond to the points where the three constraint equations intersect the y-axis; $S_3 = 8$ is point B while the other two intercepts are clearly outside of the feasible solution space.

Step 3 of the process is to use the leaving variable's row as the pivot row and apply Gaussian elimination to zero all other elements in the same column as the entering variable. So for us we will be using row 4 as the pivot row and zero out the elements in column 3 from other rows. The result will be:

	Z	x	y	S_1	S_2	S_3	soln
Z	1	-2	0	0	0	2	64
S_1	0	4	0	1	0	$-\frac{1}{2}$	24
S_2	0	3	0	0	1	$-\frac{3}{2}$	12
$S_3 \rightarrow y$	0	$\frac{1}{2}$	1	0	0	$\frac{1}{4}$	8

Here, we can see that if we set the current nonbasic variables (x and S_3) to zero, we obtain a new feasible basic solution ($Z = 64$, $S_1 = 24$, $S_2 = 12$ and $y = 8$). Z has indeed increased! In Fig. 30, we have essentially moved from point A to point B.

We now choose a new entering variable and leaving variable: in row 1, there is only one negative coefficient so we will choose that (x) as the entering variable. Dividing the 'solution' column entries by the corresponding coefficients of x in column 2, we find that row 2 gives $x = 6$, row 3 gives $x = 4$ and row 4 gives $x = 16$. Hence S_2 (corresponding to row 3) will be the leaving variable.

Thus we do Gaussian elimination using row 3 as the pivot row and column 2 as the pivot element:

	Z	x	y	S_1	S_2	S_3	soln
Z	1	0	0	0	$\frac{2}{3}$	1	72
S_1	0	0	0	1	$-\frac{4}{3}$	$\frac{3}{2}$	8
$S_2 \rightarrow x$	0	1	0	0	$\frac{1}{3}$	$-\frac{1}{2}$	4
y	0	0	1	0	$-\frac{1}{6}$	$\frac{1}{2}$	6

Now our solution is $Z = 72$, $S_1 = 8$, $x = 4$ and $y = 6$. This is point C in Fig. 30. We do not need to go further, because in row 1 of our table (i.e. the expression for Z) there are no nonbasic variables with negative coefficients left. Thus the maximum value of Z is our current solution (72).

Notice that right now, S_1 is still a basic variable with a non-zero value. This means that the constraint this corresponds to ($5x + 2y \leq 40$) *does not* limit Z's optimisation. In other words, it is the other constraints ($6x + 6y \leq 60$ and $2x + 4y \leq 32$) that were limiting the solution. We can also see that in Fig. 30 - point C is the vertex formed by the intersection of the aforementioned two constraints.

Exercise

Maximise the function $Z = 150\,x_1 + 175\,x_2$, subject to the constraints $7\,x_1 + 11\,x_2 \leq 77$, $10\,x_1 + 8\,x_2 \leq 80$, $0 \leq x_1 \leq 9$ and $0 \leq x_2 \leq 6$.

Ans: max $Z = 1414$ at $x_1 = 4.889$, $x_2 = 3.889$.

4.5.2 Nonlinear Systems

In the previous section we discussed linear programming, which was for a linear objective function along with linear constraints. However it is more likely that we will encounter nonlinear systems - either the objective function is nonlinear, the constraints are nonlinear or both. Unfortunately nonlinear programming is a very very large field of maths, and there are several methods used for each of the many branches.

For instance **quadratic programming** is - as you may have guessed - the family of numerical methods that deal with a quadratic objective function and a set of *linear* constraints. If both the objective function and the set of constraints are quadratic, then the solution techniques are called **quadratically constrained quadratic programming**. If the objective function is to be maximised (i.e. function is concave) or minimised (function is convex) and the set of constraints are convex, this comes under **convex programming**. If the objective function is the ratio of a convex and a concave function and its constraints are convex, then **fractional programming** techniques are applied. There is even a branch called **integer programming** for the cases where the variables are integers (i.e. not continuous)!

In general, the numerical methods used for nonlinear programming all iterate for the optimal solution using a relatively similar procedure: identify an initial point (usually a feasible basic solution) and then move towards the optimal solution using a rule or algorithm. There are, of course, many possibilities for these rules, but they tend to fall into 3 groups:

- zero-order routines: only the values of the objective function and its constraints are used
- 1st-order routines: the values and gradients of the objective function and its constraints are used
- 2nd-order routines: the values, gradients and Hessians of the objective function and its constraints are used

The simplex method uses a zero-order routine, where the 'rule' is to identify a pair of entering and leaving variables such that the objective function moved closer to the optimum

condition. Higher order routines are theoretically possible, but not typically used because the significantly higher computational cost would not be worth the slight benefit.

If you are interested in finding out more, you could check out Cornell University's open-source textbook on optimisation: https://optimization.cbe.cornell.edu/ . It contains both explanations and examples of a variety of optimisation techniques including, but not limited to, nonlinear programming. The research articles containing the detailed derivations of the techniques are also referenced if you prefer to go straight to the source.

4.6 Summary

For one-dimensional unconstrained optimisation, we have both bracketing and open methods (just like for root-finding). We covered one each: **parabolic interpolation** (which requires more than one initial guess) and **Newton's method** (which only requires one initial guess).

- Parabolic interpolation
  1. Choose 3 initial guesses, x_0, x_1 and x_2.
  2. Calculate their functional values f_0, f_1 and f_2.
  3. The next guess is given by:
  $$x_3 = \frac{f_0\left(x_1^2 - x_2^2\right) + f_1\left(x_2^2 - x_0^2\right) + f_2\left(x_0^2 - x_1^2\right)}{2f_0\left(x_1 - x_2\right) + 2f_1\left(x_2 - x_0\right) + 2f_2\left(x_0 - x_1\right)}\,.$$
  4. Calculate the functional value f_3.
  5. Choose 3 points for the next iteration by:
     - updating the subscripts, or
     - inspection (like for the golden-section search).
- Newton's method

  We optimize the function essentially by finding the root of the 1st derivative, so the procedure is similar to the Newton-Raphson's method covered before.
  1. Choose an initial guess x_0.
  2. Calculate the next guess(es) using the formula:
  $$x_{i+1} = x_i - \frac{f_i'}{f_i''}\,.$$

For the bracketing methods we calculate a relative error:

$$\epsilon = (1 - R)\left|\frac{x_u - x_l}{x_{\text{opt}}}\right| \times 100\%\,,$$

where $R = \frac{\sqrt{5}-1}{2}$ is the golden ratio, x_u and x_l are our upper and lower limits, and x_{opt} is the intermediate value that is our maxima/minima at the current iteration.

For the Newton's method we use the relative error between successive guesses:

$$\epsilon = \left| \frac{x_{\text{new}} - x_{\text{old}}}{x_{\text{new}}} \right| \times 100\% \,.$$

For multidimensional unconstrained optimisation, we introduced both direct and gradient methods.

- Random Search Method

  Evaluate the function at randomly chosen locations; the optimum point will be reached as the number of data points increase.

- Univariate Search Method

  Repeatedly vary a single parameter (i.e. do a 1D search for that parameter) while all other parameters are held constant.

- Powell's Method

  1. Choose a starting point 0 and two initial search directions h_1 and h_2.
  2. Locate point 1 by searching along h_1 from point 0.
  3. Locate point 2 by searching along h_2 from point 1.
  4. Form direction h_3 by joining points 0 and 2.
  5. Locate point 3 by searching along h_3 from point 2.
  6. Locate point 4 by searching along h_2 from point 3.
  7. Locate point 5 by searching along h_3 from point 4.
  8. Form direction h_4 by joining points 3 and 5.
  9. Locate optimum point by searching along h_4 from point 5.

- Steepest ascent method

  1. Determine the gradient vector ∇f.
  2. Determine the Hessian matrix **H**.
  3. Choose a starting point (x_0, y_0) and evaluate the corresponding gradient vector there.

4. Determine the 1D function $g(h)$ along the gradient vector by substituting

$$x = x_0 + \left.\frac{\partial f}{\partial x}\right|_0 h$$

and

$$y = y_0 + \left.\frac{\partial f}{\partial y}\right|_0 h$$

into $f(x, y)$.

5. Locate h^*, the value of h that results in a local maxima/minima of $g(h)$.
   - Use the golden-section search, or
   - use parabolic interpolation, or
   - use Newton's method, or
   - solve analytically if simple enough.
6. Calculate the next starting point (x_1, y_1) by plugging h^* back into our expressions for x and y:

$$x_1 = x_0 + \left.\frac{\partial f}{\partial x}\right|_0 h^* ,$$

and

$$y_1 = y_0 + \left.\frac{\partial f}{\partial y}\right|_0 h^* .$$

- Newton's method
  1. Determine the gradient vector ∇f.
  2. Determine the Hessian matrix $\mathbf{H}$ (and its inverse, if convenient).
  3. Choose an initial guess.
  4. Evaluate the next guess, using the iteration formula:

$$\vec{x}_{i+1} = \vec{x}_i - \mathbf{H}_i^{-1} \nabla f_i .$$

- Marquardt method
  1. Determine the gradient vector ∇f.
  2. Determine the modified Hessian matrix $\tilde{\mathbf{H}}_i = \mathbf{H}_i + \alpha_i \mathbf{I}$ (and its inverse, if convenient).
  3. Choose an initial guess.

4. Set an initially large value for α_i and an algorithm for updating it.
5. Evaluate the next guess, using the iteration formula:

$$\vec{x}_{i+1} = \vec{x}_i - \tilde{\mathbf{H}}_i^{-1} \nabla f_i \,.$$

6. Evaluate $f(\vec{x}_{i+1})$, and update α_i accordingly.

For constrained optimisation, we introduced the Simplex method for linear systems:

1. Write out m+1 equations (the objective function Z and m constraint equations containing the slack variables S_j) in matrix/table form.
2. Identify the **entering** and **leaving** variables:
   - Entering variable is the one in the Z equation with the most negative (positive) coefficient, so that it increases (decreases) Z the most.
   - Leaving variable is the one for which when it is zero, the entering variable has the smallest positive value, assuming that all independent variables are positive.
3. Apply Gaussian elimination: using the row corresponding to the **leaving** variable as the pivot row, eliminate all the elements in other rows that are in the same column as the **entering** variable (i.e. pivot element).
4. Repeat the previous two steps until no nonbasic variables are left that will optimise Z further.

5 Numerical Integration & Differentiation

5.1 Introduction

As mentioned briefly in the overview, this topic is focused on formulae for approximating integrals and derivatives. Although you would have already learned a lot of analytical methods for differentiating and integrating directly, there is always the possibility that the function is too difficult or perhaps is discontinuous. Further we might only have a set of data instead of an actual function. We have already learned some techniques from the chapter on curve-fitting which will allow us to generate a function from the set of data (and hence proceed to differentiate and/or integrate as required), so in this chapter we will introduce alternative methods that allow us to estimate the integral or derivative directly or build on those ideas previously discussed.

5.2 Newton-Cotes Integration Formulae

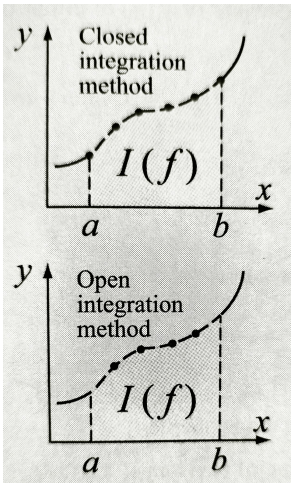


Figure 31: Sketch of an integral approximated using a (top) closed integration method and (bottom) open integration method, from *Gilat & Subramaniam*, Fig. 9-6 on pg343.

Integration methods generally fall into two categories, **closed** and **open**. Fig. 31 shows a sketch of the same integral approximated using a closed and an open integration method - the main difference is that closed integration methods make use of the points at the limits

(a and b in the figure), while open integration methods do not. In a sense open integration methods are extrapolating based on the information from the interior points. Thus, closed integration methods are typically used for definite integrals while open integration methods are mostly used for improper integrals.

In this section we will introduce the closed forms of the Newton-Cotes integration formulae. The Newton-Cotes integration formulae are the most popular schemes used for integration, probably due to their simple formulations and intuitive underpinnings. They were derived essentially by replacing (or approximating) the actual function to be integrated with a much simpler function – typically a polynomial. As you may expect, the accuracy of the approximation increases with the order of the polynomial used.

5.2.1 Rectangle & Midpoint methods

We will start with the simplest possible approximation - a constant. Basically we will treat the function as having a constant value between the specified limits; the integral will thus be the product of the constant value and the difference between the limits. Fig. 32 illustrates this idea, comparing two such possibilities against the exact integral. It should be obvious why this method is called the **Rectangle** method.

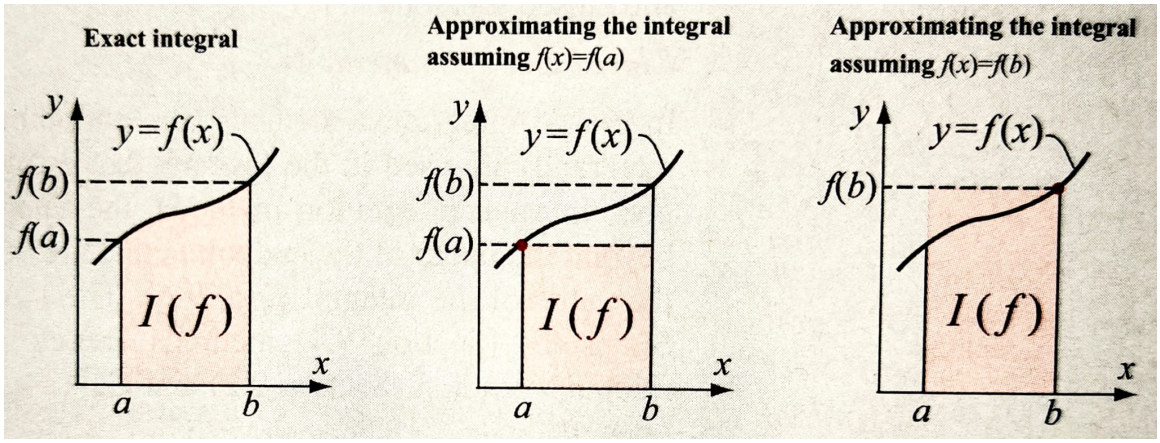


Figure 32: Illustration from *Gilat & Subramaniam*, Fig. 9-8 on pg344, of (left) an exact integral, (middle and right) approximations using two versions of the rectangle method.

Using the rectangle method, the integral is approximated as:

$$\int_a^b f(x)\,dx \approx (b-a)f(a)\,,$$

if the functional value at the lower limit $x = a$ is taken as the constant value; or

$$\int_a^b f(x)\,dx \approx (b-a)f(b)\,,$$

if the functional value at the upper limit $x = b$ is taken as the constant value instead.

Obviously we can see that the error can be quite large, and hence we could divide the domain $[a, b]$ into smaller intervals each approximated with a rectangle (this may sound familiar to you from high school when first learning about integration). We call this the **multiple-application** or **composite** rectangle method.

If the domain is split into N smaller intervals of equal width $\Delta x \equiv \frac{b-a}{N}$, then the integral would be approximated as:

$$\int_a^b f(x)\,dx \approx \Delta x \sum_{i=1}^{N} f(x_i)\,.$$

If the domain is split up by points $x_0 = a$, x_1, x_2, $\ldots$ $x_N = b$ then the above formula would be when the rectangles' heights are equal to the functional value at the end of each interval. Conversely if the numbering is from 1 to $N + 1$, then the rectangles' heights are equal to the functional value at the beginning of the interval.

You may have already thought that an obvious improvement would be to use the middle of the interval for the rectangle's height rather than either of the ends because the underestimation and overestimation at either side of the middle may offset each other somewhat. This would be the **Midpoint** method, illustrated in Fig. 33.

The midpoint method approximates the integral as:

$$\int_a^b f(x)\,dx \approx (b-a)f\left(\frac{a+b}{2}\right)\,,$$

and thus the composite midpoint method would be:

$$\int_a^b f(x)\,dx \approx \Delta x \sum_{i=0}^{N-1} f\left(\frac{x_i + x_{i+1}}{2}\right)\,,$$

for points $x_0 = a$, x_1, x_2, $\ldots$ $x_N = b$.

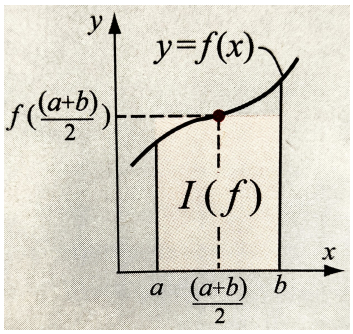


Figure 33: Sketch of an integral approximated using the midpoint method, from *Gilat & Subramaniam*, Fig. 9-10 on pg345.

5.2.2 Trapezoidal Rule

Some of you may have heard of the Trapezoidal rule before: instead of a rectangle, the integral (or area under the curve) is approximated using a trapezoid. This is usually the first Newton-Cotes integration formula introduced, and it approximates the function with a straight line (1st order polynomial).

Visually, the straight line is formed by joining the points $(a, f(a))$ and $(b, f(b))$. The area under the curve (i.e. the integral) is thus:

$$\int_a^b f(x)\,dx \approx \frac{1}{2}(b-a)\left(f(a)+f(b)\right) .$$

Again we can improve the accuracy by using the multiple-application or composite trapezoidal rule by splitting the domain into N intervals of equal width $\Delta x \equiv \frac{b-a}{N}$:

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx \sum_{i=0}^{N-1} \frac{1}{2}\Delta x\left[f(x_i)+f(x_{i+1})\right] \\
&= \frac{1}{2}\Delta x\left[f(x_0)+f(x_1)+f(x_1)+f(x_2)+f(x_2)+f(x_3)+\cdots+f(x_{N-1})+f(x_N)\right] \\
&= \frac{1}{2}\Delta x\left[f(x_0)+2\sum_{i=1}^{N-1} f(x_i)+f(x_N)\right] \\
&= \frac{b-a}{2N}\left[f(x_0)+2\sum_{i=1}^{N-1} f(x_i)+f(x_N)\right]
\end{aligned}$$

where $x_0 = a$ and $x_N = b$.

Exercise

Approximate the integral $I = \int_0^1 0.5 + 2x - 10x^2 + 15x^3\,dx$ by dividing into 5 equally-spaced intervals and applying the rectangle, midpoint and trapezoid methods. Compare your answers to the analytical answer ($I = 23/12 \approx 1.917$).

Ans: $I = 1.3$ (rectangle height = left functional value), 2.7 (rectangle height = right functional value), 1.875 (midpoint), 2 (trapezoid)

5.2.3 Simpson's Rules

We can take things a notch higher than a linear function by using a single higher order function to approximate an integral. We will therefore require more than just the two limits $x = a$ and $x = b$ to define the chosen polynomial. Fig. 34 shows the same integral

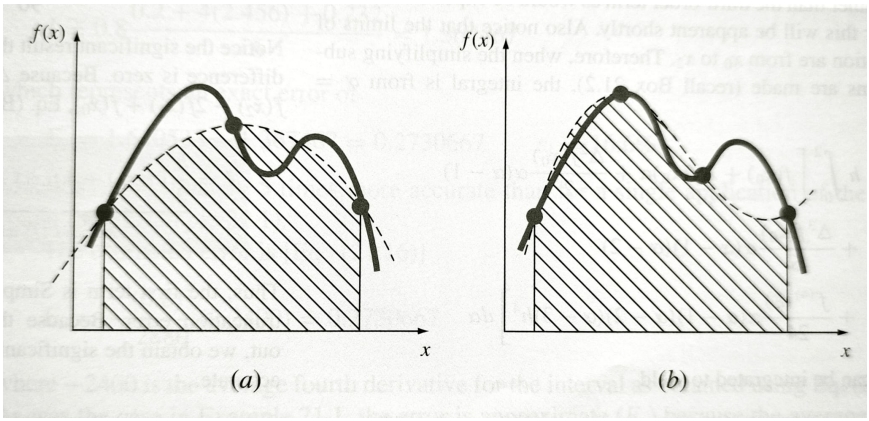


Figure 34: Sketch of an integral approximated using (left) Simpson's 1/3 rule and (right) Simpson's 3/8 rule, from *Chapra & Canale*, Fig. 21.10 on pg613.

approximated using first a quadratic polynomial - which leads to **Simpson's 1/3 rule** - and next a cubic polynomial - which leads to **Simpson's 3/8 rule**. The quadratic polynomial requires a total of 3 points while the cubic requires 4; typically these points are chosen to be equally-spaced.

Simpson's 1/3 Rule

We already covered in Section 2.3 how to fit a polynomial to given data points, for example a 2nd order Lagrange interpolating polynomial for the quadratic function:

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx \int_a^b f_2(x)\,dx \\
&= \int_a^b \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)\,dx \\
&\vdots \\
&= \frac{b-a}{6}\left[f(x_0) + 4f(x_1) + f(x_2)\right] \\
&= \frac{\Delta x}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] ,
\end{aligned}$$

where $\Delta x \equiv \frac{b-a}{2}$ is the interval between each of the points.

We could choose to do a multiple-application of Simpson's 1/3 rule, which we expect would have higher accuracy compared to the multiple-application trapezoidal rule. However since 3 points are needed for each application, then we would end up with an even number of segments (odd number of points) as illustrated in Fig. 35.

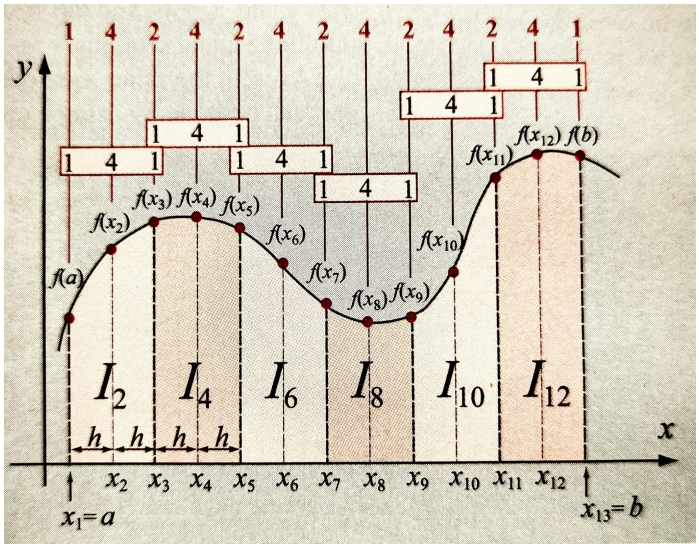


Figure 35: Sketch of an integral approximated using multiple application of Simpson's 1/3 rule, from *Gilat & Subramaniam*, Fig. 9-16 on pg353.

If we ensure that all the $N+1$ points (x_0, x_1, ... x_N) are equally-spaced, then for $\Delta x \equiv \frac{b-a}{N}$:

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx \sum_{i=1,3,5}^{N-1} \frac{\Delta x}{3}\left[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})\right] \\
&= \frac{\Delta x}{3}\left[f(x_0) + \sum_{i=1,3,5}^{N-1}\left(4f(x_i) + 2f(x_{i+1})\right) + f(x_N)\right] \\
&= \frac{\Delta x}{3}\left[f(x_0) + 4\sum_{i=1,3,5}^{N-1} f(x_i) + 2\sum_{j=2,4,6}^{N-2} f(x_j) + f(x_N)\right] .
\end{aligned}$$

The coefficient $\frac{1}{3}$ in front is what gives this rule its name.

Simpson's 3/8 Rule

Simpson's 3/8 rule is somewhat opposite to the 1/3 rule in that it needs an even number of points and odd number of segments, because 4 points are required to define a cubic polynomial (see Fig. 34). Again if we use equally-spaced points and a 3rd order Lagrange

interpolating polynomial:

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx \int_a^b f_3(x)\,dx \\
&= \int_a^b \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} f(x_0) + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-2)(x_1-x_3)} f(x_1) \\
&\quad + \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} f(x_2) + \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} f(x_3)\,dx \\
&\vdots \\
&= \frac{b-a}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] \\
&= \frac{3}{8}\Delta x\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right],
\end{aligned}$$

where $\Delta x \equiv \frac{b-a}{3}$ is the interval between each of the points. This time it is the $\frac{3}{8}$ coefficient that earned the method's name.

Just as before, we could do a multiple-application of the Simpson's 3/8 rule, as illustrated in Fig. 36.

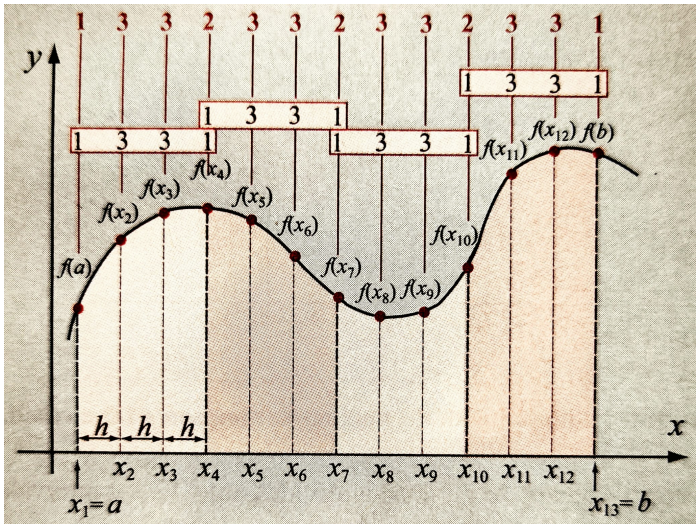


Figure 36: Sketch of an integral approximated using multiple application of Simpson's 3/8 rule, from *Gilat & Subramaniam*, Fig. 9-18 on pg354.

However we do not usually do so, because although Simpson's 3/8 rule is slightly more accurate than the 1/3 rule, it is computationally more expensive and requires more points. Thus an alternative way to integrate over a set of even number of data points is to apply

Simpson's 3/8 rule to only the first (or last) 3 segments and Simpson's 1/3 rule to all the rest, as shown in Fig. 37.

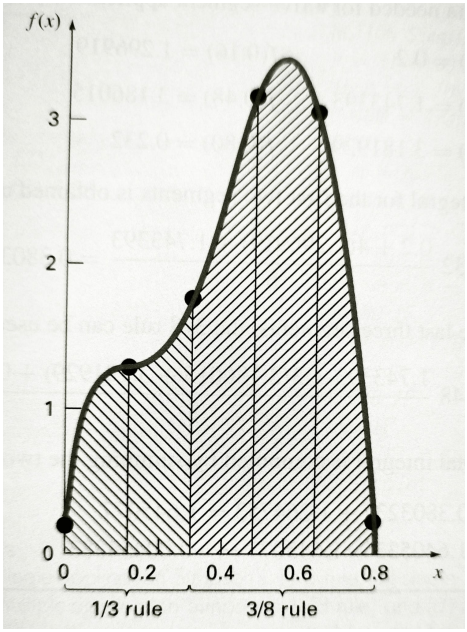


Figure 37: Sketch of an integral approximated using a combination of Simpson's 1/3 and 3/8 rules, from *Chapra & Canale*, Fig. 21.12 on pg619.

Whether the Simpson's 3/8 rule is applied to the first 3 segments or the last 3 segments is a matter of preference, since we would not know textita priori which portion is closer to a quadratic or cubic polynomial.

Finally, you may have noticed that so far we kept mentioning 'equally-spaced' points. What if they are not equally-spaced? In that case we could combine some of the other methods, for instance use Simpson's 1/3 rule only on the neighbouring segments which have equally-spaced points (if there are any) and the Trapezoidal rule on all the others.

Exercise

Approximate the integral $I = \int_0^1 0.5 + 2x - 10x^2 + 15x^3\, dx$ by dividing into (a) 4 segments and (b) 5 segments, and using Simpson's rules as appropriate. How do your estimates compare to the analytical answer ($I = 23/12 \approx 1.917$)?

Ans: $I = 1.917$ for both (a) and (b)

5.3 Integration of Functions

These techniques were meant explicitly for integrating functions, unlike the methods in the previous section which could handle both functions as well as data sets. We will look at three techniques: **Gauss quadrature**, **adaptive quadrature** and **Romberg integration**.

5.3.1 Gauss Quadrature

Think back to when we were discussing the Newton-Cotes integration formulae. In all of them, the points at which the function was evaluated were fixed - they were always the endpoints of the interval(s). In order to get more accurate results, we either increased the number of intervals (equivalent to decreasing the interval size) or increased the order of the polynomial used to approximate the function (e.g. using Simpson's $1/3$ rule instead of the trapezoidal rule). This results in a significant increase in computation. Thus the question arises: shouldn't it be possible to identify locations at which the error is minimised? Or in other words, can we shift the locations at which we evaluate the function so that just a single application is sufficient?

Fig. 38 illustrates this idea for the trapezoidal rule: the error in the approximation arises because the trapezoid does not fully match the function - some parts of the function will be underestimated and other parts will be overestimated. However if we do not restrict ourselves to using the domain end points, then it will be possible to define the trapezoid so that the over- and underestimated parts cancel out and hence we obtain an exact approximation. The class of techniques that arise from this idea is what we call **Gauss quadrature**.

Method of Undetermined Coefficients

Consider the form of the trapezoidal rule:

$$I = \int_a^b f(x)\,dx \approx \frac{1}{2}\,(b-a)\,[f(a)+f(b)]\ .$$

This is essentially a linear combination of $f(x)$ at two locations $x = a$ and $x = b$. It is reasonable to think that our 'exact approximation' could still be a linear combination of two functional values but at different x locations. In other words, we expect to have

$$I = \int_a^b f(x)\,dx \approx c_1 f(x_1) + c_2 f(x_2)\,,$$

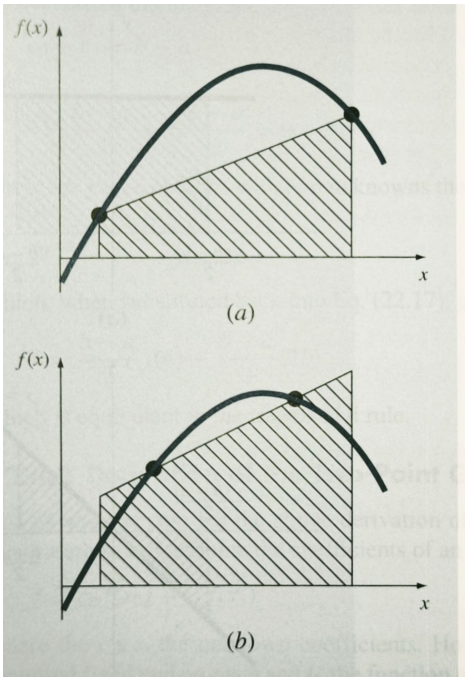


Figure 38: Sketch of a trapezoid defined using two different sets of points, from *Chapra & Canale*, Fig. 22.6 on pg641.

where the coefficients c_1 and c_2 are weights or weighting factors. The method of undetermined coefficients is an intuitive way to, as the name states, determine the unknown coefficients in this linear combination.

For instance, let's derive the coefficients c_1 and c_2 and locations x_1 and x_2 for the case where $f(x) = \alpha x + \beta$. Since we are aiming for an exact approximation, then

$$\begin{aligned} I = \int_a^b f(x)\,dx &= c_1 f(x_1) + c_2 f(x_2) \\ \int_a^b \alpha x + \beta\,dx &= c_1\left(\alpha x_1 + \beta\right) + c_2\left(\alpha x_2 + \beta\right) \\ \left[\frac{1}{2}\alpha x^2 + \beta x\right]_a^b &= \alpha\left(c_1 x_1 + c_2 x_2\right) + \beta\left(c_1 + c_2\right) \\ \frac{1}{2}\alpha\left(b^2 - a^2\right) + \beta\left(b - a\right) &= \alpha\left(c_1 x_1 + c_2 x_2\right) + \beta\left(c_1 + c_2\right) \end{aligned}$$

We can equate the coefficients of α and β on either side respectively to obtain 2 equations:

$$\begin{aligned} \alpha: \quad & \tfrac{1}{2}\left(b^2 - a^2\right) = c_1 x_1 + c_2 x_2 \\ \beta: \quad & b - a = c_1 + c_2 \end{aligned}$$

Unfortunately, there are 4 unknowns and only 2 equations here so we need to simplify the problem further. If we decide to keep the functional values at the end points i.e. $x_1 = a$ and $x_2 = b$, this gives us:

$$\begin{aligned} \alpha: \quad & \tfrac{1}{2}\left(b^2 - a^2\right) = c_1 a + c_2 b \\ \beta: \quad & b - a = c_1 + c_2 \end{aligned}$$

which can be solved to obtain $c_1 = c_2 = \frac{b-a}{2}$. You may notice that this is exactly the Trapezoidal rule!

Naturally if we had chosen other locations for x_1 and x_2, the weights c_1 and c_2 would have been different.

Gauss-Legendre Formulae

The above example was for a linear function but of course we are more likely to have to deal with a non-linear function. Let's apply the method of undetermined coefficients again but with the caveat that we want an 'exact approximation' for up to 3rd order polynomials. That is,

$$f(x) = \alpha + \beta x + \gamma x^2 + \delta x^3 \,.$$

The resulting expression would be:

$$\begin{aligned} \alpha\,(b-a) \quad &+ \quad \frac{1}{2}\beta\left(b^2 - a^2\right) + \frac{1}{3}\gamma\left(b^3 - a^3\right) + \frac{1}{4}\delta\left(b^4 - a^4\right) \\ &= \quad \alpha\,(c_1 + c_2) + \beta\,(c_1 x_1 + c_2 x_2) + \gamma\left(c_1 x_1^2 + c_2 x_2^2\right) + \delta\left(c_1 x_1^3 + c_2 x_2^3\right) \end{aligned}$$

We can equate the coefficients of α, β, γ and δ on both sides to obtain 4 equations:

$$\begin{aligned} \alpha: \quad & b - a = c_1 + c_2 \\ \beta: \quad & \tfrac{1}{2}(b^2 - a^2) = c_1 x_1 + c_2 x_2 \\ \gamma: \quad & \tfrac{1}{3}(b^3 - a^3) = c_1 x_1^2 + c_2 x_2^2 \\ \delta: \quad & \tfrac{1}{4}(b^4 - a^4) = c_1 x_1^3 + c_2 x_2^3 \end{aligned}$$

This time since we have exactly four equations we are able to solve for the four unknowns c_1, c_2, x_1 and x_2. However it will be much easier to solve if, instead of general limits a and

b, we have the integral be symmetrical i.e. $a = -1$ and $b = 1$. Then:

$$\begin{aligned} \alpha: &\qquad 2 = c_1 + c_2 \\ \beta: &\qquad 0 = c_1 x_1 + c_2 x_2 \\ \gamma: &\qquad \tfrac{2}{3} = c_1 x_1^2 + c_2 x_2^2 \\ \delta: &\qquad 0 = c_1 x_1^3 + c_2 x_2^3 \end{aligned}$$

We won't include the working here, but after solving these simultaneous equations we will find that $c_1 = c_2 = 1$, $x_1 = -\frac{1}{\sqrt{3}}$ and $x_2 = \frac{1}{\sqrt{3}}$. This is the **Two-Point Gauss-Legendre formula**:

$$I = \int_{-1}^{1} f(x)\,dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right).$$

But what about integrals with other limits? In that case, we just need to scale the integral using a linear transformation:

$$x = \frac{(b+a) + (b-a)y}{2},$$

for which $\frac{dx}{dy} = \frac{b-a}{2}$. We can then rewrite the integral into a form for which we can apply the Two-Point Gauss-Legendre formula:

$$\int_a^b f(x)\,dx = \int_{-1}^{1} f\left(\frac{(b+a) + (b-a)y}{2}\right) \cdot \left(\frac{b-a}{2}\right) dy\,.$$

Example

Let's approximate the previous exercise's integral $I = \int_0^1 0.5 + 2x - 10x^2 + 15x^3\,dx$ using the Two-Point Gauss-Legendre formula.

First, we need to do a transformation so that the integral limits are -1 and 1:

$$x = \frac{(b+a) + (b-a)y}{2} = \frac{(1+0) + (1-0)y}{2} = \frac{1+y}{2},$$

and $\frac{dx}{dy} = \frac{1}{2}$. Thus the transformed integral is:

$$\int_0^1 f(x)\,dx = \int_{-1}^{1} f\left(\frac{1+y}{2}\right) \cdot \frac{1}{2}\,dy = \frac{1}{2}\int_{-1}^{1} g\,(y)\;dy\,.$$

Applying the Two-Point Gauss-Legendre formula,

$$\begin{aligned} I = \frac{1}{2}\int_{-1}^{1} g(x)\,dx &\approx \frac{1}{2}\left[g\left(-\frac{1}{\sqrt{3}}\right) + g\left(\frac{1}{\sqrt{3}}\right)\right] \\ &= \frac{1}{2}\left[f\left(\frac{1-\frac{1}{\sqrt{3}}}{2}\right) + f\left(\frac{1+\frac{1}{\sqrt{3}}}{2}\right)\right] \\ &= \frac{1}{2}\left[f\left(\frac{1}{2}-\frac{1}{2\sqrt{3}}\right) + f\left(\frac{1}{2}+\frac{1}{2\sqrt{3}}\right)\right]. \end{aligned}$$

We evaluate the two required functional values:

$$\begin{aligned} f\left(\frac{1}{2}-\frac{1}{2\sqrt{3}}\right) &= \frac{1}{2} + 2\left(\frac{1}{2}-\frac{1}{2\sqrt{3}}\right) - 10\left(\frac{1}{2}-\frac{1}{2\sqrt{3}}\right)^2 + 15\left(\frac{1}{2}-\frac{1}{2\sqrt{3}}\right)^3 \\ &= 0.6176\,, \end{aligned}$$

and

$$\begin{aligned} f\left(\frac{1}{2}+\frac{1}{2\sqrt{3}}\right) &= \frac{1}{2} + 2\left(\frac{1}{2}+\frac{1}{2\sqrt{3}}\right) - 10\left(\frac{1}{2}+\frac{1}{2\sqrt{3}}\right)^2 + 15\left(\frac{1}{2}+\frac{1}{2\sqrt{3}}\right)^3 \\ &= 3.216\,. \end{aligned}$$

The approximate value for I is thus $\frac{1}{2}\left(0.6176 + 3.216\right) = 1.917$ (4sf).

This is very close to the analytical answer 23/12 (≈ 1.917)!

Higher-Point Formulae

We can, of course, have formulae using more than just two points. In general a formula involving would n points would have the form:

$$I = \int_{-1}^{1} f(x)\,dx \approx c_0 f(x_0) + c_1 f(x_1) + c_2 f(x_2) + \cdots + c_{n-1} f(x_{n-1})\,.$$

We are able to derive the required weights and locations in a similar manner to that for the Two-Point Gauss-Legendre formula.

For instance, consider a formula involving 3 points (which we expect would be more accurate). The goal, similarly, is that for some linear combination of the function $f(x)$ evaluated at 3 specific locations, we can get an 'exact approximation' of a high order polynomial.

Considering that there will be a total of 6 unknowns (3 x locations and 3 weights c), we would need 6 equations to obtain an exact solution. This set of 6 equations can be obtained if we assume that the 'exact approximation' will be up to 5th order polynomials only. That is,

$$f_5(x) = \alpha + \beta x + \gamma x^2 + \delta x^3 + \epsilon x^4 + \zeta x^5\,.$$

The formula would have the form

$$I = \int_{-1}^{1} f_5(x)\,dx \approx c_0 f(x_0) + c_1 f(x_1) + c_2 f(x_2)\,.$$

As before, after substituting in $f_5(x)$ and evaluating the integral, we can compare the coefficients of α, β, γ, δ, ϵ and ζ on both sides to obtain our set of equations (to be solved for the c's and x's).

The weights c and point locations x are laid out in the following Table 1. Notice that not only are the locations symmetrical, but the weights are also greatest at the centre and decrease outwards i.e. the points closest to the midpoint have the highest weights and vice versa.

Table 1: Points and weights for Gauss Quadrature

Points (n)	**Weights** (c)	**Locations** (x)
2	1	$\pm\frac{1}{\sqrt{3}} \approx \pm 0.577350$
3	$\frac{8}{9} \approx 0.888889$	0
	$\frac{5}{9} \approx 0.555556$	$\pm\sqrt{\frac{3}{5}} \approx 0.774597$
4	$\frac{1}{36}\left(18+\sqrt{30}\right) \approx 0.652145$	$\pm\frac{1}{35}\sqrt{525-70\sqrt{30}} \approx \pm 0.339981$
	$\frac{1}{36}\left(18-\sqrt{30}\right) \approx 0.347855$	$\pm\frac{1}{35}\sqrt{525+70\sqrt{30}} \approx \pm 0.861136$
5	$\frac{128}{225} \approx 0.568889$	0
	$\frac{1}{900}\left(322+13\sqrt{70}\right) \approx 0.478629$	$\pm\frac{1}{21}\sqrt{245-14\sqrt{70}} \approx \pm 0.538469$
	$\frac{1}{900}\left(322-13\sqrt{70}\right) \approx 0.236927$	$\pm\frac{1}{21}\sqrt{245+14\sqrt{70}} \approx \pm 0.906180$

Exercise

Obtain approximations for the integral $I = \int_1^2 x^7\,dx$ using the 2-point, 3-point and 4-point formulae. Do they get more accurate with increasing number of points?

Ans: $I \approx 31.201$ (2-point), 31.871 (3-point), 31.875 (4-point)

5.3.2 Romberg Integration

Aside from Gauss Quadrature, another method for obtaining more accurate approximations but without too much complicated calculation is Romberg Integration. It makes use of successive applications of the Trapezoidal rule (i.e. obtaining an estimate using the Trapezoidal rule more than once) in combination with **Richardson's Extrapolation**.

Richardson's Extrapolation

You may guess from the name that like the concepts covered in the chapter which discussed curve-fitting and interpolation, Richardson's extrapolation requires 'data' from which it can extrapolate to outside of the data's domain. Here, the 'data' is a set of estimates obtained using the Trapezoidal rule. Using these, a more accurate estimate of the integral is obtained. In this sense Richardson's extrapolation can also be thought of as similar to the corrector stage of the predictor-corrector methods we covered in the chapter on root-finding.

To explain how the extrapolation works we first need to have two estimates of the integral. Let's say we estimate an integral I twice: using intervals of size h_1 and h_2, where $h_1 < h_2$. Thus we obtain:

$$I \approx I(h_1) + \epsilon(h_1)$$

and

$$I \approx I(h_2) + \epsilon(h_2)\,,$$

where $\epsilon(h_1)$ and $\epsilon(h_2)$ are the errors associated with each of the two estimates. We can thus equate the two expressions:

$$I(h_1) + \epsilon(h_1) = I(h_2) + \epsilon(h_2)\,.$$

Generally the error in the Trapezoidal rule is $\sim O(h^2)$, so the ratio of the two errors is

approximately equal to $(\frac{h_1}{h_2})^2$. Substituting this in:

$$\begin{aligned} I(h_1) + \left(\frac{h_1}{h_2}\right)^2 \epsilon(h_2) &= I(h_2) + \epsilon(h_2) \\ \epsilon(h_2) &= \frac{I(h_2) - I(h_1)}{\left(\frac{h_1}{h_2}\right)^2 - 1} \\ \Rightarrow I &\approx I(h_2) + \frac{I(h_2) - I(h_1)}{\left(\frac{h_1}{h_2}\right)^2 - 1} \,. \end{aligned}$$

If we had halved the interval size, i.e. $h_1 = 2h_2$, then this simplifies to:

$$I \approx \frac{4}{3} I(h_2) - \frac{1}{3} I(h_1) \,,$$

which has an error $\sim O(h^4)$. The accuracy has increased by two orders of magnitude with just a simple linear combination of two 'ordinary' multiple-applications of Trapezoidal rule!

We do not discuss how the magnitude of the error is obtained in this course, but you can refer to *Chapra & Canale* (chp21) for the derivation on the Trapezoidal rule's estimate and *Ralston & Rabinowitz*[3] for proof regarding the increased accuracy.

Example

Let's use the integral $I = \int_1^2 x^7 \, dx$ from the exercise in Gauss quadrature. We need to obtain two estimates of the integral using the Trapezoidal rule, and the second estimate should use twice the number of intervals as the first estimate.

For the first estimate, let us use 2 intervals. We will thus have:

$$\begin{aligned} I(h_1) &= \frac{2-1}{2 \times 2} \left[f(x_0) + 2 \sum_{i=1}^{2-1} f(x_i) + f(x_2) \right] \\ &= \frac{1}{4} \left[1^7 + 2 \left(1.5^7 \right) + 2^7 \right] \\ &= 40.793 \,. \end{aligned}$$

[3] Ralston, A., and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd Ed., McGraw-Hill, New York, 1978.

The second estimate will then use twice as many intervals, i.e. 4 intervals:

$$\begin{aligned} I(h_2) &= \frac{2-1}{2\times 4}\left[f(x_0)+2\sum_{i=1}^{4-1} f(x_i)+f(x_4)\right] \\ &= \frac{1}{8}\left[1^7+2\left(1.25^7+1.5^7+1.75^7\right)+2^7\right] \\ &= 34.155\,. \end{aligned}$$

Now using Richardson's extrapolation, we obtain a final estimate:

$$I \approx \frac{4}{3}I(h_2)-\frac{1}{3}I(h_1) = 31.942\,.$$

Compared to the analytical answer of 31.875, we can see that the two Trapozoidal rule estimates are in the right ballpark but not fantastic. The estimate using Richardson's extrapolation, however, is much more accurate and obtained more easily than if we had used say, 20 intervals (which only gets the not-as-accurate 31.56). Even if a computer programme would be easy to set up for the Trapezoidal rule, it would still be more computationally expensive than using the Richardson's extrapolation because of the large number of operations required.

Romberg Integration Algorithm

There is one more step involved after using Richardson's extrapolation, which is the Romberg Integration Algorithm. Basically it's a process by which we can iteratively improve the accuracy of our estimate. For instance we can see from our example that our estimate is only good to 3 significant figures. But how do we get it more accurate than that?

One idea is that instead of using the estimate from 2 intervals, we should have used something more accurate. That is, we could have used applied Richardson's extrapolation to estimates from 4 intervals and 8 intervals. Indeed, the estimate using 8 intervals would be:

$$\begin{aligned} I(h_3) &= \frac{2-1}{2\times 8}\left[f(x_0)+2\sum_{i=1}^{8-1} f(x_i)+f(x_8)\right] \\ &= \frac{1}{16}\left[1^7+2\left(1.125^7+1.25^7+1.375^7+\cdots+1.875^7\right)+2^7\right] \\ &= 32.448\,. \end{aligned}$$

Then applying Richardson's extrapolation we get:

$$I \approx \frac{4}{3}I(h_3)-\frac{1}{3}I(h_2) = 31.879\,.$$

We can see that this is indeed more accurate than our previous estimate. But can we improve this further? For clarity, let's call the first estimate I_1 and the second estimate I_2. Logically, we can apply Richardson's extrapolation to these two again and an even more accurate estimate. Note that, because of the change in accuracy (it's now $\sim O(h^4)$), the formula we use will be different:

$$\begin{aligned} I_1 + \left(\frac{h_1}{h_2}\right)^4 \epsilon(h_2) &= I_2 + \epsilon(h_2) \\ \epsilon(h_2) &= \frac{I_2 - I_1}{\left(\frac{h_1}{h_2}\right)^4 - 1} \\ \Rightarrow I &\approx I_2 + \frac{I_2 - I_1}{\left(\frac{h_1}{h_2}\right)^4 - 1} \end{aligned}$$

And since $h_1 = 2h_2$,

$$I \approx \frac{16}{15} I_2 - \frac{1}{15} I_1 \,,$$

which has an accuracy of $\sim O(h^6)$.

Going back to our example, our estimate (to 5 s.f.) will thus be

$$I \approx \frac{16}{15} I_2 - \frac{1}{15} I_1 = 31.875 \,,$$

which is almost exactly equal to the analytical answer.

Through this process hopefully we have demonstrated that there are 2 ways in which we can obtain improved accuracy: (1) using more intervals, as we already know from before; and (2) using a series of Richardson's extrapolations. Using more intervals may not be the most ideal method if we are trying to save computational power, but applying Richardson's extrapolations multiple times requires more estimates of the Trapezoidal rule. That is, each of our 'first' estimates I_1 and I_2 only required two estimates of the Trapezoidal rule each, but the 'second' estimate required three estimates in total because it used both I_1 and I_2. Romberg's integration algorithm will allow us to progressively obtain estimates and only calculate as many Trapezoidal rule estimates as needed. This algorithm is illustrated in Fig. 39.

Note the subscripts used in Fig. 39: for an estimate $I_{j,k}$, j is a counter of sorts for which the higher the value the 'more accurate' the estimate (because more intervals would have

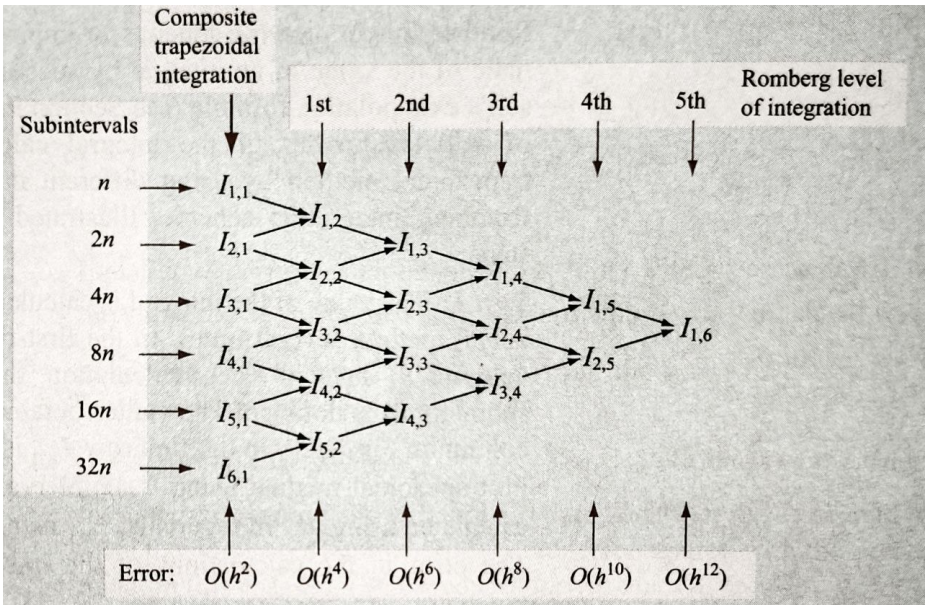


Figure 39: Illustration of integral estimates required for Romberg integration, from *Gilat & Subra*, Fig. 9-22 on pg370.

been used); and k represents 'level' of integration ('1' would mean just Trapezoidal rule, '2' would be Trapezoidal rule + 1 instance of Richardson's extrapolation etc.).

We would start at the top left corner: $I_{1,1}$ is obtained using the Trapezoidal rule and n intervals. A second estimate, $I_{2,1}$ is obtained using $2n$ intervals. We can use Richardson's extrapolation to extrapolate an estimate $I_{1,2}$ using $I_{1,1}$ and $I_{2,1}$.

If this were found to be not accurate enough (e.g. by comparing the relative error, as we did in the chapter on root-finding), then estimate $I_{3,1}$ would be found using $4n$ intervals. This, in conjunction with $I_{2,1}$, would lead to $I_{2,2}$. Applying Richardson's extrapolation to both $I_{2,1}$ and $I_{2,2}$ would lead to the even more accurate $I_{1,3}$.

The process thus continues, as long as any accuracy criterion is not met, each time adding on one more estimate using the Trapezoidal rule.

Although the process is relatively simple, we have to keep in mind that the formula used for Richardson's extrapolation changes with each 'level' (because of the increasing accuracy, as we demonstrated earlier). You may have noticed that the coefficients always sum to 1, and the 'heavier' weight is unsurprisingly always on the more accurate estimate. The

general form is:

$$I_{j,k} \approx \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}\,, \qquad \epsilon \sim O(h^{2k})\,,$$

where subscripts j and k are as defined previously. Note that the value of j does not have any significance apart from that $j+1$ denotes the more accurate estimate and j the less accurate estimate.

Exercise

Obtain an estimate for the integral $I = \int_0^1 0.5 + 2x - 10x^2 + 15x^3\,dx$ using Romberg integration, for a relative error of $< 0.1\%$. You can start with 2 intervals ($n = 2$).

Ans: 1.91667, needed 3 Trapezoidal rule estimates (analytical answer is $\frac{23}{12}$).

5.3.3 Adaptive Quadrature

When we were introducing Richardson's extrapolation, you may have questioned whether it was possible to use some method other than the Trapezoidal rule. Indeed we can! In fact, **adaptive quadrature** uses Simpson's $1/3$ rule instead of the Trapezoidal rule which grants us higher accuracy. In addition the algorithm employed also allows for decreasing the interval size only in regions where the function $f(x)$ is changing rapidly, unlike the Romberg integration algorithm which just halves the interval size uniformly over the integration domain. You may have seen this before in finite element analysis or computational fluid dynamics where the mesh is finer in regions where the results are expected to be more 'interesting', but coarser in regions where the results are mostly uniform or changing only very gradually.

The equivalent to Richardson's extrapolation, for adaptive quadrature, is **Boole's Rule**. Similarly to before, we first estimate an integral $I = \int_a^b f(x)\,dx$ twice using intervals of size h_1 and h_2, where $h_1 = b - a$ and $h_2 = h_1/2$. Thus we obtain:

$$I(h_1) = \frac{h_1}{6}\left(f(a) + 4f(c) + f(b)\right) ,$$

where $c = (a + b)/2$ is the midpoint between a and b and

$$I(h_2) = \frac{h_2}{6}\left(f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)\right) ,$$

where d and e are the midpoints between c and each of the endpoints a and b respectively.

As before if we use $\epsilon(h_1)$ and $\epsilon(h_2)$ to denote the errors associated with each of the two estimates, we can thus (again) equate the two expressions:

$$I(h_1) + \epsilon(h_1) = I(h_2) + \epsilon(h_2)\,.$$

Generally the error in Simpson's $1/3$ rule is $\sim O(h^4)$, so the ratio of the two errors is

approximately equal to $(\frac{h_1}{h_2})^4$. Substituting this in:

$$\begin{aligned} I(h_1) + \left(\frac{h_1}{h_2}\right)^4 \epsilon(h_2) &= I(h_2) + \epsilon(h_2) \\ \epsilon(h_2) &= \frac{I(h_2) - I(h_1)}{\left(\frac{h_1}{h_2}\right)^4 - 1} \\ &= \frac{I(h_2) - I(h_1)}{2^4 - 1} \\ &= \frac{I(h_2) - I(h_1)}{15} \\ \Rightarrow I &\approx \frac{16}{15} I(h_2) - \frac{1}{15} I(h_1)\,. \end{aligned}$$

The algorithm used for adaptive quadrature is quite simple and proceeds as follows:

1. For a given integral with limits a and b, obtain two estimates of Simpson's $1/3$ rule (the second estimate must use intervals half the size used for the first estimate).
2. Calculate the absolute error between the two estimates.
3. Compare the absolute error against a specified threshold.
    - If less than the specified threshold, use Boole's Rule to obtain a final estimate.
    - Otherwise, split the integral in two: one from a to the midpoint c and the second from c to b. Evaluate both of these subintegrals starting from step 1, before finally summing them up.

We know that if a function were 'simple' and did not vary much, then a small amount of large intervals is sufficient to obtain a good estimate (i.e. the estimate obtained using twice as many intervals would be very similar). On the other hand if the function had a lot of variation, then a large number of small intervals is necessary to reduce the amount of error. Hence by checking the absolute relative error between the two estimates of the integral, adaptive quadrature can decrease interval size as needed only in areas where the function varies rapidly while allowing for large intervals elsewhere. This is more efficient than applying a (very) small interval size to the entire domain, and thus allows adaptive quadrature to be as efficient as, if not more than, Romberg integration especially for functions which have both high and low variability.

Example

Let us apply adaptive quadrature to obtain an estimate for the integral $I = \int_1^2 x^7\,dx$, for a relative error of < 0.01.

We start with 1 interval: $h_1 = b - a = 1$.

$$\begin{aligned} I(h_1) &= \frac{1}{6}\left(f(1) + 4f(1.5) + f(2)\right) \\ &= \frac{1}{6}\left(1 + 4 \times 1.5^7 + 2^7\right) \\ &= 32.89063\,. \end{aligned}$$

The second estimate uses 2 intervals (half the size): $h_2 = h_1/2 = 0.5$.

$$\begin{aligned} I(h_2) &= \frac{0.5}{6}\left(f(1) + 4f(1.25) + 2f(1.5) + 4f(1.75) + f(2)\right) \\ &= \frac{1}{12}\left(1 + 4 \times 1.25^7 + 2 \times 1.5^7 + 4 \times 1.75^7 + 2^7\right) \\ &= 31.94214\,. \end{aligned}$$

The absolute relative error is 0.948 which is more than our threshold. Thus we shall split the integral into two:

$$I = \int_1^{1.5} x^7\,dx + \int_{1.5}^2 x^7\,dx\,.$$

Let's handle the first subintegral I_1 with limits 1 and 1.5. The two estimates are:

$$\begin{aligned} I(h_1) &= \frac{1.5-1}{6}\left(f(1) + 4f(1.25) + f(1.5)\right) \\ &= \frac{1}{12}\left(1^7 + 4 \times 1.25^7 + 1.5^7\right) \\ &= 3.096619\,, \end{aligned}$$

and

$$\begin{aligned} I(h_2) &= \frac{0.5/2}{6}\left(f(1) + 4f(1.125) + 2f(1.25) + 4f(1.375) + f(1.5)\right) \\ &= \frac{1}{24}\left(1^7 + 4 \times 1.25^7 + 2 \times 1.25^7 + 4 \times 1.375^7 + 1.5^7\right) \\ &= 3.079762\,. \end{aligned}$$

The absolute relative error is 0.0168 which is slightly more than our threshold. Thus we probably only need to split this integral once:

$$I_1 = \int_1^{1.25} x^7\,dx + \int_{1.25}^{1.5} x^7\,dx\,.$$

For the first of these smaller integrals I_{11} with limits 1 and 1.25, our two estimates are:

$$\begin{aligned} I(h_1) &= \frac{1.25-1}{6}\left(f(1)+4f(1.125)+f(1.25)\right) \\ &= \frac{1}{24}\left(1^7+4\times 1.125^7+1.25^7\right) \\ &= 0.620465\,, \end{aligned}$$

and

$$\begin{aligned} I(h_2) &= \frac{0.25/2}{6}\left(f(1)+4f(1.0625)+2f(1.125)+4f(1.1875)+f(1.25)\right) \\ &= \frac{1}{48}\left(1^7+4\times 1.0625^7+2\times 1.125^7+4\times 1.1875^7+1.25^7\right) \\ &= 0.620084\,. \end{aligned}$$

The absolute relative error is 0.0004 which is less than our threshold, so we can apply Boole's rule to these two estimates to obtain a more accurate estimate:

$$I_{11} \approx \frac{16}{15}I(h_2) - \frac{1}{15}I(h_1) = 0.620058\,.$$

For the other of these smaller integrals I_{12} with limits 1.25 and 1.5, our two estimates are:

$$\begin{aligned} I(h_1) &= \frac{1.5-1.25}{6}\left(f(1.25)+4f(1.375)+f(1.5)\right) \\ &= \frac{1}{24}\left(1.25^7+4\times 1.375^7+1.5^7\right) \\ &= 2.459297\,, \end{aligned}$$

and

$$\begin{aligned} I(h_2) &= \frac{0.25/2}{6}\left(f(1.25)+4f(1.3125)+2f(1.375)+4f(1.4375)+f(1.5)\right) \\ &= \frac{1}{48}\left(1.25^7+4\times 1.3125^7+2\times 1.375^7+4\times 1.4375^7+1.5^7\right) \\ &= 2.458602\,. \end{aligned}$$

The absolute relative error is 0.0007 which is less than our threshold, so we can apply Boole's rule to these two estimates to obtain a more accurate estimate:

$$I_{12} \approx \frac{16}{15}I(h_2) - \frac{1}{15}I(h_1) = 2.458555\,.$$

Now we handle the second subintegral I_2 with limits 1.5 and 2. The two estimates are:

$$\begin{aligned} I(h_1) &= \frac{2-1.5}{6}\left(f(1.5)+4f(1.75)+f(2)\right) \\ &= \frac{1}{12}\left(1.5^7+4\times 1.75^7+2^7\right) \\ &= 28.84552\,. \end{aligned}$$

and

$$\begin{aligned} I(h_2) &= \frac{0.5/2}{6}\left(f(1.5)+4f(1.625)+2f(1.75)+4f(1.875)+f(2)\right) \\ &= \frac{1}{24}\left(1.5^7+4\times 1.625^7+2\times 1.75^7+4\times 1.875^7+2^7\right) \\ &= 28.79949\,. \end{aligned}$$

The absolute relative error is 0.0460 which is more than our threshold of 0.01. Thus we shall split the integral into two:

$$I_2 = \int_{1.5}^{1.75} x^7\,dx + \int_{1.75}^{2} x^7\,dx\,.$$

For the first of these smaller integrals I_{21} with limits 1.5 and 1.75, our two estimates are:

$$\begin{aligned} I(h_1) &= \frac{1.75-1.5}{6}\left(f(1.5)+4f(1.625)+f(1.75)\right) \\ &= \frac{1}{24}\left(1.5^7+4\times 1.625^7+1.75^7\right) \\ &= 7.793096\,, \end{aligned}$$

and

$$\begin{aligned} I(h_2) &= \frac{0.25/2}{6}\left(f(1.5)+4f(1.5625)+2f(1.625)+4f(1.6875)+f(1.75)\right) \\ &= \frac{1}{48}\left(1.5^7+4\times 1.5625^7+2\times 1.625^7+4\times 1.6875^7+1.75^7\right) \\ &= 7.791949\,. \end{aligned}$$

The absolute relative error is 0.0011 which is less than our threshold, so we can apply Boole's rule to these two estimates to obtain a more accurate estimate:

$$I_{21} \approx \frac{16}{15}I(h_2) - \frac{1}{15}I(h_1) = 7.791872\,.$$

For the other of these smaller integrals I_{22} with limits 1.75 and 2, our two estimates are:

$$\begin{aligned} I(h_1) &= \frac{2-1.75}{6}\left(f(1.75)+4f(1.875)+f(2)\right) \\ &= \frac{1}{24}\left(1.75^7+4\times 1.875^7+2^7\right) \\ &= 21.00639\,, \end{aligned}$$

and

$$\begin{aligned} I(h_2) &= \frac{0.25/2}{6}\left(f(1.75)+4f(1.8125)+2f(1.875)+4f(1.9375)+f(2)\right) \\ &= \frac{1}{48}\left(1.75^7+4\times 1.8125^7+2\times 1.875^7+4\times 1.9375^7+2^7\right) \\ &= 21.00463\,. \end{aligned}$$

The absolute relative error is 0.0018 which is less than our threshold, so we can apply Boole's rule to these two estimates to obtain a more accurate estimate:

$$I_{12} \approx \frac{16}{15}I(h_2) - \frac{1}{15}I(h_1) = 21.00451\,.$$

Finally we obtain our estimate for I:

$$I \approx I_{11} + I_{12} + I_{21} + I_{22} = 31.875\,.$$

This is the same as the analytical answer, which we also got very close to using Romberg integration after 2 instances of Richardson's extrapolation (i.e. used 3 Trapezoidal rule estimates in total) and 4-point Gauss quadrature. Considering the computational effort for all three methods, which do you think you prefer?

5.4 Other Types of Integrals

5.4.1 Multiple Integrals

In the earlier sections we only discussed methods for an integral in one dimension. However it is highly likely we will encounter integrals of multiple dimensions also, e.g. area or volume integrals involve integration over more than one dimension.

The analytical way of handling a multiple integral such as $\int_a^b \int_c^d f(x,y)\,dxdy$, requires us to integrate in one dimension first, then integrate the result in the second dimension. The order of integration does not matter, i.e. $= \int_a^b \left[\int_c^d f(x,y)\,dx\right] dy = \int_c^d \left[\int_a^b f(x,y)\,dy\right] dx$.

This is the same approach that is used for numerical methods too: the inner integral is approximated using a numerical method with all non-relevant variables held constant, then the result is used for the numerical approximation of the outer integral. A bit more planning is required beforehand though, because as we saw in the earlier section numerical methods use the functional value at certain points. This means that the inner integral will need to be approximated at each of those points in order to be used for approximating the outer integral numerically. We will demonstrate this process using a simple example.

Example

Obtain a numerical approximation for I by applying the trapezoidal rule over 2 equally-spaced intervals in both x and y, where $I = \int_0^1 \int_0^\pi x \sin y\,dydx$. The analytical solution is $I = 1$.

If we do not change the order of the integrals, then we will be integrating over y first, and then integrate over x. So we can think of it as: $I = \int_0^1 G(x)\,dx$, where $G(x) = \int_0^\pi x \sin y\,dy$.

Since we are applying the trapezoidal rule over 2 equally-spaced intervals in both x and y, that means that we need to consider $x = 0$, $\frac{1}{2}$, 1 and $y = 0$, $\frac{\pi}{2}$, π. Thus our integral would be approximated as:

$$\begin{aligned} I = \int_0^1 G(x)\,dx &\approx \frac{1}{2}\Delta x\left(G(x_0) + 2G(x_1) + G(x_2)\right) \\ &= \frac{1}{4}\left(G(0) + 2G(\frac{1}{2}) + G(1)\right). \end{aligned}$$

Thus we need to approximate $G(x)$ three times. They would be:

$$\begin{aligned}
G(0) &= \int_0^\pi 0 \sin y \, dy = 0. \\
G\left(\frac{1}{2}\right) &= \int_0^\pi \frac{1}{2} \sin y \, dy \\
&\approx \frac{1}{2}\Delta y \left(\frac{1}{2}\sin(y_0) + 2 \times \frac{1}{2}\sin(y_1) + \frac{1}{2}\sin(y_2)\right) \\
&= \frac{\pi}{4}\left(\frac{1}{2}\sin 0 + 2 \times \frac{1}{2}\sin\frac{\pi}{2} + \frac{1}{2}\sin\pi\right) \\
&= \frac{\pi}{4}(0+1+0) \\
&= \frac{\pi}{4}. \\
G(1) &= \int_0^\pi 1 \sin y \, dy \\
&\approx \frac{1}{2}\Delta y \left(\sin(y_0) + 2\sin(y_1) + \sin(y_2)\right) \\
&= \frac{\pi}{4}\left(\sin 0 + 2\sin\frac{\pi}{2} + \sin\pi\right) \\
&= \frac{\pi}{4}(0+2+0) \\
&= \frac{\pi}{2}.
\end{aligned}$$

Thus,

$$I \approx \frac{1}{4}\left(0 + 2 \times \frac{\pi}{4} + \frac{\pi}{2}\right) = \frac{\pi}{4}.$$

We can also reverse the order of integration; that is we will have: $I = \int_0^\pi G(y)\, dy$, where $G(y) = \int_0^1 x \sin y \, dx$. Our integral will thus be approximated as:

$$\begin{aligned}
I = \int_0^\pi G(y)\, dy &\approx \frac{1}{2}\Delta y \left(G(y_0) + 2G(y_1) + G(y_2)\right) \\
&= \frac{\pi}{4}\left(G(0) + 2G\left(\frac{\pi}{2}\right) + G(\pi)\right).
\end{aligned}$$

Thus we need to approximate $G(y)$ three times. They would be:

$$\begin{aligned} G(0) &= \int_0^1 x \sin 0 \, dx = 0. \\ G\left(\frac{\pi}{2}\right) &= \int_0^1 x \sin \frac{\pi}{2} \, dx \\ &\approx \int_0^1 x \, dx \\ &= \frac{1}{2}\Delta x \left(x_0 + 2x_1 + x_2\right) \\ &= \frac{1}{4}\left(0 + 2 \times \frac{1}{2} + 1\right) \\ &= \frac{1}{2}\,. \\ G(\pi) &= \int_0^1 x \sin \pi \, dx = 0\,. \end{aligned}$$

Thus,

$$I \approx \frac{\pi}{4}\left(0 + 2 \times \frac{1}{2} + 0\right) = \frac{\pi}{4}\,,$$

which is the same answer as we achieved previously.

Regardless of the order of integration, you may have noticed that we essentially needed to evaluate the function $f(x, y) = x \sin y$ at all of the 9 points in the domain (0,0), (0,$\frac{\pi}{2}$), (0,π), ($\frac{1}{2}$,0), ($\frac{1}{2}$,$\frac{\pi}{2}$), ($\frac{1}{2}$,1), (1,0), (1,$\frac{\pi}{2}$) and (1,π). So it may be useful (in a computer programme) to evaluate all these functional values beforehand.

Exercise

Approximate the same integral $I = \int_0^1 \int_0^\pi x \sin y \, dydx$ using a single application of Simpson's 1/3 rule in both x and y.

Ans: $I \approx \frac{\pi}{3}$

5.4.2 Improper Integrals

Aside from multiple integrals, we also need to consider improper integrals: these include integrals with unbounded (or infinite) limits as well as those integrals over a domain containing singularities.

Integrals with Singularities

Sometimes the functions we want to integrate have singularities within the domain we wish to integrate over, e.g. $f(x) = \frac{1}{x}$ has a singularity at $x = 0$ and thus integrating over any range that contains $x = 0$ will go to infinity.

One approach to deal with this would be to split the integral into two integrals which are each integrated over ranges on either side of the singularity. For instance if we wanted to solve $\int_{-a}^{b} \frac{1}{x}\, dx$ where a and b are both positive, then we could rewrite the integral as:

$$\int_{-a}^{b} \frac{1}{x}\, dx = \int_{-a}^{0} \frac{1}{x}\, dx + \int_{0}^{b} \frac{1}{x}\, dx\,.$$

The reason we do so is to shift the singularity to one of the limits. In such a situation, there is a possibility that the integral might still have a finite value e.g. $\int_0^1 \frac{1}{\sqrt{x}}\, dx = 2$.

There are two ways we can then obtain a numerical solution for these integrals:

1. use an **open** integration method, or
2. use Gauss quadrature (discussed in Section 5.3.1).

We have mentioned open integration methods before in the beginning of this chapter - these are methods that do 'extrapolation' using the *interior* points rather than the limits. The midpoint method that we saw before is actually an open integration method since it utilises the functional at the middle of the interval rather than at the interval limits.

Table 2 below lists some of the Newton-Cotes open integration formulae (we have seen some of the closed ones like Trapezoidal rule and Simpson's rules previously). You may recognise the first one – it's the midpoint method.

Table 2: Newton-Cotes Open Integration Formulae

Intervals	**Interior Points**	**Newton-Cotes Open Integration Formula**
2	1	$(b-a)\,f(x_1)$
3	2	$(b-a)\,[f(x_1)+f(x_2)]/2$
4	3	$(b-a)\,[2f(x_1)-f(x_2)+2f(x_3)]/3$
5	4	$(b-a)\,[11f(x_1)+f(x_2)+f(x_3)+11f(x_4)]/24$
6	5	$(b-a)\,[11f(x_1)-14f(x_2)+26f(x_3)-14f(x_4)+11f(x_5)]/20$

Example

Let's use the first three Newton-Cotes open integration formulae to estimate $I=\int_0^1 \frac{1}{\sqrt{x}}\,dx$.

For the first one (the midpoint method), the range is split into 2 equally-spaced intervals of size $h=\frac{1-0}{2}=\frac{1}{2}$ and the interior point which we use for approximating the integral is $x_1=\frac{1}{2}$. From the formula,

$$I \approx (b-a)\,f(x_1) = (1-0)\,f\left(\frac{1}{2}\right) = \frac{1}{\sqrt{1/2}} = \sqrt{2} \approx 1.41\,.$$

For the second formula, the range is split into 3 equally-spaced intervals of size $h=\frac{1-0}{3}=\frac{1}{3}$. The two interior points which we use for approximating the integral are thus $x_1=\frac{1}{3}$ and $x_2=\frac{2}{3}$. Using them in the integration formula:

$$\begin{aligned} I &\approx \frac{b-a}{2}\,[f(x_1)+f(x_2)] \\ &= \frac{1-0}{2}\left[f\left(\frac{1}{3}\right)+f\left(\frac{2}{3}\right)\right] \\ &= \frac{1}{2}\left[\frac{1}{\sqrt{1/3}}+\frac{1}{\sqrt{2/3}}\right] \\ &= \frac{\sqrt{3}}{2}\left(1+\frac{1}{\sqrt{2}}\right) \\ &\approx 1.48. \end{aligned}$$

For the third formula, the range is split into 4 equally-spaced intervals of size $h=\frac{1-0}{4}=\frac{1}{4}$. The three interior points which we use for approximating the integral are thus $x_1=\frac{1}{4}$,

$x_2 = \frac{1}{2}$ and $x_3 = \frac{3}{4}$. Hence substituting into the formula:

$$\begin{aligned} I &\approx \frac{b-a}{3}\left[2f\left(x_1\right) - f\left(x_2\right) + 2f\left(x_3\right)\right] \\ &= \frac{1-0}{3}\left[2f\left(\frac{1}{4}\right) - f\left(\frac{1}{2}\right) + 2f\left(\frac{3}{4}\right)\right] \\ &= \frac{1}{3}\left[2\left(\frac{1}{\sqrt{1/4}}\right) - \left(\frac{1}{\sqrt{1/2}}\right) + 2\left(\frac{1}{\sqrt{3/4}}\right)\right] \\ &= \frac{1}{3}\left(4 - \sqrt{2} + \frac{4}{\sqrt{3}}\right) \\ &\approx 1.63. \end{aligned}$$

Compared to the analytical answer of 2, we can see that the approximations obtained from these open integration formulae are not too far off. And as expected, the more interior points are used, the more accurate the approximation is.

Integrals with Unbounded Limits

For an unbounded integral, one or both of the limits go to infinity and there is thus the possibility that the integral itself will also be undefined (going to infinity). However some unbounded integrals do have finite value – if so, then typically it is because the function has a finite value over some range of the domain but is very close to zero elsewhere.

Following from that characteristic, one reasonable way to handle an unbounded integral (assuming it has a finite value) is to first replace the unbounded limit(s) with finite limit(s) and integrate. Next the integral is iteratively evaluated while increasing the limit(s) until the value of the integral has converged i.e. does not change significantly as per some pre-defined criterion.

Alternatively, a transformation of the variables could be performed such that the resulting integral is bounded (has finite limits). Unfortunately this usually leads to the function being singular at one (or both) of the limits. In which case, we can then treat the integral as discussed in the earlier section.

Example

Let's obtain an approximation for the integral $I = \int_2^\infty \frac{1}{x(x+2)}\,dx$.

Although there are many logical ways to iterate this, let us approach this by using the

composite Trapezoidal rule and increasing the number of intervals. That is, we shall fix the interval size and calculate the additional 'area' as the upper limit increases (towards infinity).

For simplicity we will set the interval size as 1. Our initial upper limit will thus be 3. We will need the functional values at both ends of the interval:

$$\begin{aligned} f(2) &= \frac{1}{2\,(2+2)} = \frac{1}{8}\,, \\ f(3) &= \frac{1}{3\,(3+2)} = \frac{1}{15}\,. \end{aligned}$$

Thus the initial (obviously poor) approximation will be:

$$\int_2^3 \frac{1}{x\,(x+2)}\,dx \approx \frac{1}{2}\,(f(2)+f(3)) = 0.095833\,.$$

Now if we increase the upper limit to 4, then we will need to have another trapezoid (from x=3 to x=4):

$$\begin{aligned} f(4) &= \frac{1}{4\,(4+2)} = \frac{1}{24}\,, \\ \int_3^4 \frac{1}{x\,(x+2)}\,dx &\approx \frac{1}{2}\,(f(3)+f(4)) = 0.054167\,, \end{aligned}$$

and thus

$$\int_2^4 \frac{1}{x\,(x+2)}\,dx \approx 0.095833 + 0.054167 = 0.15\,.$$

We again increase the upper limit (to 5) and calculate the area of the new trapezoid (from x=4 to x=5):

$$\begin{aligned} f(5) &= \frac{1}{5\,(5+2)} = \frac{1}{35}\,, \\ \int_4^5 \frac{1}{x\,(x+2)}\,dx &\approx \frac{1}{2}\,(f(4)+f(5)) = 0.035119\,, \end{aligned}$$

and thus

$$\int_2^5 \frac{1}{x\,(x+2)}\,dx \approx 0.15 + 0.035119 = 0.185119\,.$$

Observe that each new trapezoid is decreasing in area as expected. We need to keep increasing the limit until there is no significant increase, which in our case means that the

area of the final trapezoid has to be smaller than some threshold. Let's set this threshold to be 0.001.

Then if we continue in this process until the 30th interval (from x=31 to 32), the newest trapezoid's area is 0.000948 (less than our threshold) and so we can terminate at our estimate of 0.324 (3sf). Compared to the analytical answer of $\frac{1}{2}\ln 2 \approx 0.3466$, we can see that this is not too far off.

Here we are adding trapezoids with unit width one by one. We could, of course, use small intervals – in which case we may obtain greater accuracy but will reach convergence slower; and/or we could add more trapezoids at a time to improve computational efficiency.

Now let's estimate the integral after using a transformation, $y = \frac{1}{x}$. With this, the integral becomes

$$I = \int_0^{1/2} \frac{1}{1+2y}\, dy\,.$$

To use Gauss quadrature we have to rescale it so that the limits are from -1 to 1, so we will use the transformation:

$$y = \frac{0+1/2}{2} + \frac{1/2-0}{2} z = \frac{1}{4}(1+z)\,.$$

Our integral thus becomes

$$\begin{aligned} I &= \int_{-1}^{1} \left[1 + 2\left(\frac{1}{4}(1+z)\right)\right]^{-1} \cdot \frac{1}{4}\, dz \\ &= \frac{1}{4}\int_{-1}^{1} \left(\frac{3}{2} + \frac{1}{2}z\right)^{-1} dz \\ &= \frac{1}{2}\int_{-1}^{1} (3+z)^{-1}\, dz\,. \end{aligned}$$

Then using the Two-Point Gauss-Legendre formula,

$$\begin{aligned} I &\approx \frac{1}{2}\left[\left(3 - \frac{1}{\sqrt{3}}\right)^{-1} + \left(3 + \frac{1}{\sqrt{3}}\right)^{-1}\right] \\ &= 0.3462\,. \end{aligned}$$

This is very close to the analytical answer $\frac{1}{2}\ln 2 \approx 0.3466$, and definitely a lot faster than the iterative procedure earlier... but of course this was only possible because the transformation was not difficult or impossible to do.

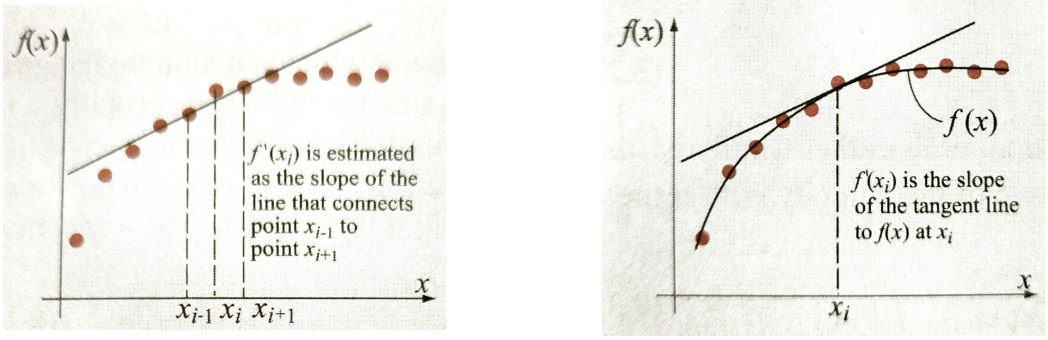
5.5 Numerical Differentiation

Now that we have gone through numerical integration, it is logical to ask whether there are numerical approaches to differentiation. Indeed there are, and you have certainly used at least one technique before!

In general there are two approaches to performing a differentiation numerically, especially if the only information available is a set of data points:

1. apply a fit using methods similar to what we covered in Section 2 and then differentiate as needed; or
2. estimate using the values near the point of interest.

These two approaches are illustrated simply in Figure 40: assuming that we were interested in determining the slope (i.e. 1st order derivative), then we could use nearby functional values to draw a slope or apply a fit to the data set and then differentiate the curve.



(a) using functional values

(b) using fitted curve

Figure 40: Illustration of slope obtained by (left) finite differences and (right) a fit, modified from *Gilat & Subra*, Fig. 8-2 on pg304.

There is no visual difference between the slopes approximated by both approaches; whichever approach is used depends on the amount of computational effort required, what information is available and what further information may be required from the data set. You may assume that this question is moot if the mathematical function were known, but that is of course assuming that we are able to easily differentiate said function.

We have already discussed methods to determine a fit for a set of data in Section 2 and so in this section we will concentrate on the other approach: using functional values near the point of interest to estimate derivatives (not just the slope!).

5.5.1 High-Accuracy Differentiation Formulae

As mentioned before, you have used functional values to differentiate before. Specifically, in some of our earliest science classes that had graph-drawing, we would have been tasked to calculate the slope – we did that by identifying two convenient points on the graph and drew a triangle. The slope would then be the difference between the functional values at those two points divided by the length of the triangle. This expression is what we call a **finite difference**:

$$\frac{df}{dx} \approx \frac{f(x_b) - f(x_a)}{x_b - x_a} .$$

If the graph were a straight line, then it would not matter which points we had chosen to create the finite difference; but if it were a curve then it would have made a huge difference. Intuitively (or perhaps visually) we can understand that the slope would look different if we picked points from the left or right of the point of interest, and that the closer the points we choose to the point of interest the better our estimate will be.

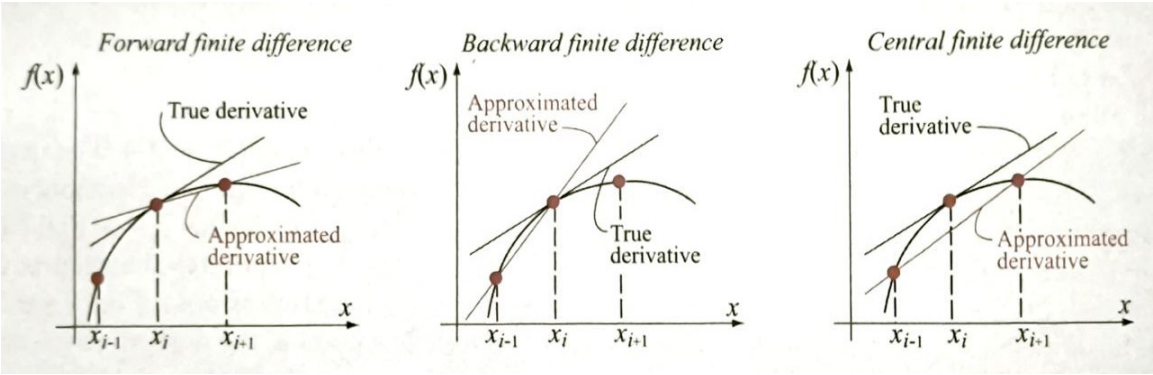


Figure 41: Illustration of true and approximated slopes using (left) forward, (middle) backward and (right) central finite differences, from *Gilat & Subra*, Fig. 8-5 on pg306.

Figure 41 shows three different sets of points used to estimate the slope at the point of interest x_i. The **forward** finite difference uses points to the right of x_i, while the **backward**

finite difference uses points to the left of x_i. Lastly, the **central** finite difference uses points from either side of x_i. The naming is rather intuitive, and generally helps differentiate between finite difference schemes in terms of accuracy as well as direction of information flow (which is more important if time were the independent variable rather than a space variable like x).

Taylor series expansion

One of the most popular ways to develop a finite difference scheme is using the Taylor series, because it allows some control over the accuracy. For a function $f(x)$, we can expand the Taylor series for the functional value at a point x_{i+s} in terms of $f(x)$ and its derivatives at the point x_i, that is,

$$f\left(x_i+s\right)=f\left(x_i\right)+f^{(1)}\left(x_i\right)\cdot s+\frac{1}{2!}f^{(2)}\left(x_i\right)\cdot s^2+\ldots+\frac{1}{k!}f^{(k)}\left(x_i\right)\cdot s^k+\ldots$$

where $f^{(k)}\left(x_i\right)$ represents the kth order derivative at the point x_i. It is noted that in the finite difference scheme, the *functional values are considered to be known values*, which are used to approximate the unknown derivatives. So, in the Taylor series expansion, the derivatives $f^{(1)}\left(x_i\right)$, $f^{(2)}\left(x_i\right)$, ... $f^{(k)}\left(x_i\right)$, ... are taken as unknowns.

Since the Taylor series is infinite, there are accordingly also an infinite number of unknown derivatives. We have to determine which term the Taylor series expansion should be truncated to. Once the truncated term is decided, the number of derivatives (unknowns) is also obtained. For example, if the Taylor series expansion is truncated after the term$f^{(K)}\left(x_i\right)\cdot s^K/K!$, the number of unknowns is K (unknowns are $f^{(1)}\left(x_i\right)$, $f^{(2)}\left(x_i\right)$, ..., $f^{(K)}\left(x_i\right)$).

The terms that are truncated are called the **truncation error**, and since generally the first term is the largest we would express the truncation error as being $\sim O(\Delta x^{K+1}$ and the **order of accuracy** is thus $K+1$.

Now to solve for the unknown derivatives, we need the *same number of equations*. On the other hand, we note that the application of Taylor series expansion at a single point provides one equation. **Since the number of unknowns is K, Taylor series expansion should be applied at K neighbouring points, giving us K equations.** This is a basic principle for the generation of a difference scheme. The value of K depends on the order

of the given derivative and the required order of accuracy.

Consider an nth order derivative of a function $f(x)$. We want to generate a finite difference scheme with pth order of accuracy at a grid point x_i. In other words, the required difference scheme is

$$f^{(n)}(x_i) = \text{finite difference expression} + O(s^p)$$

Notice that in the Taylor series expansion, the term involving $f^{(n)}(x_i)$ is $f^{(n)}(x_i) \cdot s^n/n!$. Thus, to get the expression for $f^{(n)}(x_i)$, each term in the Taylor series expansion should eventually be divided by s^n.

Now we want to match the accuracy requirement of the difference scheme, i.e. the truncation error term in the Taylor series expansion should be, *after* the division above, of the order s^p. In other words, *before* division, the truncation term was of the order s^{n+p}. From this analysis, we know that the Taylor series expansion should have terms *up to and including* $f^{(n+p-1)}(x_i) \cdot s^{n+p-1}/(n+p-1)!$.

As a result, **the number of unknowns (derivatives) in the Taylor series expansion is (*n* + *p* - 1)**. Therefore, Taylor series expansion has to be applied at $(n+p-1)$ neighbouring points, providing $(n+p-1)$ equations. In other words, **(*n* + *p* - 1) neighbouring points will be involved in the difference expression**.

Example 1

Let's determine an FD scheme for the first derivative, to 1st order accuracy i.e.

$$\frac{\partial u}{\partial x} = \text{FD expression} + O(\Delta x) \ .$$

$\frac{\partial u}{\partial x} \Rightarrow n = 1$ and $O(\Delta x) \Rightarrow p = 1$. Hence we should include terms up to $(n+p-1) = 1$:

$$\begin{aligned} u(x+\Delta x) &= u(x) + u^{(1)} \cdot \Delta x + O\left(\Delta x^2\right) \\ u^{(1)} &= \frac{u(x+\Delta x) - u(x)}{\Delta x} + \frac{O(\Delta x)^2}{\Delta x} \\ \Rightarrow \left.\frac{\partial u}{\partial x}\right|_i &= \frac{u(x+\Delta x) - u(x)}{\Delta x} + O(\Delta x) \ . \end{aligned}$$

This FD scheme is known as the **explicit forward Euler** scheme, which has order of accuracy of 1. If we had chosen to use $u(x-\Delta x)$ instead, we would have gotten a similar

expression:

$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{u(x) - u\left(x - \Delta x\right)}{\Delta x} + O\left(\Delta x\right) \ ,$$

which is the **implicit backward Euler** scheme with an order of accuracy of 1 also.

Example 2

Now let's determine an FD scheme for the first derivative, but to 2nd order accuracy i.e.

$$\frac{\partial u}{\partial x} = \text{FD expression} + O\left(\Delta x\right)^2 \ .$$

$\frac{\partial u}{\partial x} \Rightarrow n = 1$ and $O\left(\Delta x\right)^2 \Rightarrow p = 2$. Hence we should include terms up to $(n + p - 1) = 2$:

$$u\left(x + \Delta x\right) = u\left(x\right) + u^{(1)} \cdot \Delta x + \frac{1}{2!} u^{(2)} \cdot \left(\Delta x\right)^2 + O\left(\Delta x\right)^3$$

Notice that there is now the 2nd derivative in the expression, which we do not want. So we shall use an additional Taylor series expression to eliminate it:

$$u\left(x - \Delta x\right) = u\left(x\right) - u^{(1)} \cdot \Delta x + \frac{1}{2!} u^{(2)} \cdot \left(\Delta x\right)^2 + O\left(\Delta x\right)^3$$

Using both of these expressions we can obtain:

$$\left.\frac{\partial u}{\partial x}\right|_i = \frac{u\left(x + \Delta x\right) - u\left(x - \Delta x\right)}{2\Delta x} + O\left(\Delta x\right)^2 \ .$$

This FD scheme is known as the **central difference** scheme, which has order of accuracy of 2. As expected, a central finite difference scheme is generally more accurate than either forward or backward schemes. Note that in these examples we have explicitly written out $x + \Delta x$ and so on for clarity, but you will probably find using subscripts like x_{i+1} etc. to be more compact.

Example 3

Let's consider the FD scheme for the 2nd derivative to 1st order of accuracy, i.e.

$$\frac{\partial^2 u}{\partial x^2} = \text{FD expression} + O\left(\Delta x\right) \ .$$

In this case, $n = 2$ and $p = 1$, so $(n + p - 1) = 2$. Due to the appearance of unwanted derivative(s), we will again need two Taylor series expansions. We will also include one

term extra, the reason for which will become clear later:

$$\begin{aligned} u(x+\Delta x) &= u(x) + u^{(1)} \cdot \Delta x + \frac{1}{2!}u^{(2)} \cdot (\Delta x)^2 + \frac{1}{3!}u^{(3)} \cdot (\Delta x)^3 + O(\Delta x)^4 \\ u(x-\Delta x) &= u(x) - u^{(1)} \cdot \Delta x + \frac{1}{2!}u^{(2)} \cdot (\Delta x)^2 - \frac{1}{3!}u^{(3)} \cdot (\Delta x)^3 + O(\Delta x)^4 \\ \Rightarrow \left.\frac{\partial^2 u}{\partial x^2}\right|_i &= \frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{(\Delta x)^2} + O(\Delta x)^2 \ . \end{aligned}$$

This FD scheme is also known as the **central difference** scheme, but for the 2nd order derivative. Note that the order of accuracy is 2, an order higher than expected! If we had not expanded to the 3rd order derivative we would have concluded that OOA = 1. This is only possible because the interval size is uniform, i.e. Δx is constant.

In general, if we want to have an increased accuracy, then we either have to (1) decrease the stepsize Δx, or (2) increase the number of neighbouring points in the finite difference scheme.

Tables 3-5 below list the forward, backward and central finite difference schemes and associated error for the first 4 derivatives. $h \equiv \Delta x$ is the interval size. Do you notice anything regarding the numerators and denominators?

Table 3: Forward finite difference schemes

Derivative	**FD scheme**	**Error**
$\frac{df}{dx}$	$\frac{1}{h}\left[f(x_{i+1}) - f(x_i)\right]$	O(h)
	$\frac{1}{2h}\left[-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)\right]$	O(h^2)
$\frac{d^2f}{dx^2}$	$\frac{1}{h^2}\left[f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)\right]$	O(h)
	$\frac{1}{h^2}\left[-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)\right]$	O(h^2)
$\frac{d^3f}{dx^3}$	$\frac{1}{h^3}\left[f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)\right]$	O(h)
	$\frac{1}{2h^3}\left[-3f(x_{i+4}) + 14f(x_{i+3}) - 24f(x_{i+2}) + 18f(x_{i+1}) - 5f(x_i)\right]$	O(h^2)
$\frac{d^4f}{dx^4}$	$\frac{1}{h^4}\left[f(x_{i+4}) - 4f(x_{i+3}) + 6f(x_{i+2}) - 4f(x_{i+1}) + f(x_i)\right]$	O(h)
	$\frac{1}{h^4}\left[-2f(x_{i+5}) + 11f(x_{i+4}) - 24f(x_{i+3}) + 26f(x_{i+2}) - 14f(x_{i+1}) + 3f(x_i)\right]$	O(h^2)

The choice of finite difference scheme is generally up to the user, but it must be noted that sometimes the location of the point of interest (x_i) and the number of points available can limit the user's options. For instance if we are trying to estimate a derivative at the left-

Table 4: Backward finite difference schemes

Derivative	FD scheme	Error
$\frac{df}{dx}$	$\frac{1}{h}\left[f(x_i) - f(x_{i-1})\right]$	O(h)
	$\frac{1}{2h}\left[3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})\right]$	O(h^2)
$\frac{d^2f}{dx^2}$	$\frac{1}{h^2}\left[f(x_i) - 2f(x_{i-1}) + f(x_{i-2})\right]$	O(h)
	$\frac{1}{h^2}\left[2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})\right]$	O(h^2)
$\frac{d^3f}{dx^3}$	$\frac{1}{h^3}\left[f(x_i) - 3f(x_{i-1}) + 3f(x_{i-2}) - f(x_{i-3})\right]$	O(h)
	$\frac{1}{2h^3}\left[5f(x_i) - 18f(x_{i-1}) + 24f(x_{i-2}) - 14f(x_{i-3}) + 3f(x_{i-4})\right]$	O(h^2)
$\frac{d^4f}{dx^4}$	$\frac{1}{h^4}\left[f(x_i) - 4f(x_{i-1}) + 6f(x_{i-2}) - 4f(x_{i-3}) + f(x_{i-4})\right]$	O(h)
	$\frac{1}{h^4}\left[3f(x_i) - 14f(x_{i-1}) + 26f(x_{i-2}) - 24f(x_{i-3}) + 11f(x_{i-4}) - 2f(x_{i-5})\right]$	O(h^2)

Table 5: Central finite difference schemes

Derivative	FD scheme	Error
$\frac{df}{dx}$	$\frac{1}{2h}\left[f(x_{i+1}) - f(x_{i-1})\right]$	O(h^2)
	$\frac{1}{12h}\left[-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})\right]$	O(h^4)
$\frac{d^2f}{dx^2}$	$\frac{1}{h^2}\left[f(x_{i+1}) - 2f(x_i) + f(x_{i-1})\right]$	O(h^2)
	$\frac{1}{12h^2}\left[-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})\right]$	O(h^4)
$\frac{d^3f}{dx^3}$	$\frac{1}{2h^3}\left[f(x_{i+2}) - 2f(x_{i+1}) + 2f(x_{i-1}) - f(x_{i-2})\right]$	O(h^2)
	$\frac{1}{8h^3}\left[-f(x_{i+3}) + 8f(x_{i+2}) - 13f(x_{i+1}) + 13f(x_{i-1}) - 8f(x_{i-2}) + f(x_{i-3})\right]$	O(h^4)
$\frac{d^4f}{dx^4}$	$\frac{1}{h^4}\left[f(x_{i+2}) - 4f(x_{i+1}) + 6f(x_i) - 4f(x_{i-1}) + f(x_{i-2})\right]$	O(h^2)
	$\frac{1}{6h^4}\left[-f(x_{i+3}) + 12f(x_{i+2}) - 39f(x_{i+1}) + 56f(x_i) - 39f(x_{i-1}) + 12f(x_{i-2}) - f(x_{i-3})\right]$	O(h^4)

most boundary (i.e. x_i corresponds to the first data point) then we can only use a forward finite difference scheme. Thus if we need the derivative across the entire data set, then for practical reasons we may choose to use different schemes at different regions.

5.5.2 Richardson's Extrapolation

If you are thinking the heading of this section looks familiar, you are correct. This is exactly the same name that was introduced in the section on Romberg integration. In fact the

underlying idea is also similar: that we can obtain a more accurate approximation (of a *derivative* this time) through a linear combination of two estimates of said derivative.

Let's use the central difference scheme (for $D = \frac{df}{dx}$) to derive this.

$$D = \frac{f(x_i + h_1) - f(x_i - h_1)}{2h_1} + \epsilon(h_1) = D(h_1) + \epsilon(h_1)\,,$$

where we use an interval of size h_1. We can obtain a second estimate using a different size h_2:

$$D = \frac{f(x_i + h_2) - f(x_i) - h_2}{2h_2} + \epsilon(h_2) = D(h_2) + \epsilon(h_2)\,.$$

Equating the two,

$$D(h_1) + \epsilon(h_1) = D(h_2) + \epsilon(h_2)\,.$$

We know that for the Euler forward, the error is O(h^2). Hence the ratio of the two errors $\epsilon(h_1)$ and $\epsilon(h_2)$ is approximately $\left(\frac{h_1}{h_2}\right)^2$. Thus:

$$\begin{aligned} D(h_1) + \left(\frac{h_1}{h_2}\right)^2 \epsilon(h_2) &= D(h_2) + \epsilon(h_2) \\ \epsilon(h_2) &= \frac{D(h_2) - D(h_1)}{\left(\frac{h_1}{h_2}\right)^2 - 1}\,. \end{aligned}$$

If we choose $h_1 = 2h_2$, then

$$\begin{aligned} \epsilon(h_2) &= \frac{D(h_2) - D(h_1)}{(2)^2 - 1} = \frac{1}{3}\left(D(h_2) - D(h_1)\right) \\ \Rightarrow D &\approx \frac{4}{3}D(h_2) - \frac{1}{3}D(h_1)\,, \end{aligned}$$

or more specifically,

$$D = \frac{4}{3}D\left(\frac{h}{2}\right) - \frac{1}{3}D(h) + O(h^4)\,,$$

which is the Richardson's extrapolation but for a derivative.

If you are unconvinced, you can also derive this using Taylor series expansions (you will need 4: one each at $x_i \pm h$ and $x_i \pm \frac{h}{2}$).

Observe that here, the result is unchanged by the exact form of the finite difference scheme – it is the magnitude of the truncation error that matters. So this extrapolation will work for any finite difference scheme that has a truncation error $\sim O(h^2)$. If we used something with a higher accuracy, e.g. O(h^4), then

$$D = \frac{16}{15}D\left(\frac{h}{2}\right) - \frac{1}{15}D(h) + O(h^6)\,.$$

Example

Let's approximate the derivative of the function $f(x) = \frac{3^x}{x}$ at the point $x = 1$.

We need a finite difference scheme with truncation error $\sim O(h^2)$, so let's use the more accurate forward scheme from Table 3:

$$\frac{df}{dx} = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2)\,.$$

For our first estimate we shall choose an interval size of $h = 0.2$:

$$\begin{aligned} D(h) &= \frac{-f(1+2h) + 4f(1+h) - 3f(1)}{2h} \\ &= \frac{-f(1.4) + 4f(1.2) - 3f(1)}{0.4} \\ &= \frac{1}{0.4}\left[-\left(\frac{3^{1.4}}{1.4}\right) + 4\left(\frac{3^{1.2}}{1.2}\right) - 3\left(\frac{3^1}{1}\right)\right] \\ &= 0.3298\,. \end{aligned}$$

The second estimate has to use half the interval size, so

$$\begin{aligned} D\left(\frac{h}{2}\right) &= \frac{-f(1+h) + 4f(1+h/2) - 3f(1)}{h} \\ &= \frac{-f(1.2) + 4f(1.1) - 3f(1)}{0.2} \\ &= \frac{1}{0.2}\left[-\left(\frac{3^{1.2}}{1.2}\right) + 4\left(\frac{3^{1.1}}{1.1}\right) - 3\left(\frac{3^1}{1}\right)\right] \\ &= 0.3078\,. \end{aligned}$$

Finally, we use Richardson's extrapolation to obtain a more accurate estimate:

$$D \approx \frac{4}{3}(0.3078) - \frac{1}{3}(0.3298) = 0.3005\,.$$

Comparing this to the analytical answer ($3ln3 - 3 \approx 0.2958$), we can see that it is indeed quite good!

5.5.3 Partial Derivatives

The way we treat partial derivatives is identical to how we treat regular derivatives. For instance, if we wanted to differentiate a 2D function $f(x, y)$ with respect to x, the Euler forward for the partial derivative would be:

$$\left.\frac{\partial f}{\partial x}\right|_{x_i, y_i} \approx \frac{f(x_{i+1}, y_i) - f(x_i, y_i)}{x_{i+1} - x_i}\,.$$

We would only have to ensure that the other independent variable(s) are held constant in this finite difference scheme.

For higher order partial derivatives, we may also encounter mixed partial derivatives. We would not be able to directly apply one of the higher-order finite difference schemes; instead we need to separate the partial derivative into stages corresponding to each independent variable. For example,

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial g}{\partial x}\,.$$

We can apply a finite difference scheme to the innermost derivative ($g \equiv \frac{\partial f}{\partial y}$), and then apply a finite difference scheme to the outer derivative ($\frac{\partial g}{\partial x}$).

Example

Write out the finite difference expression for $\frac{\partial^3 f}{\partial x \partial y^2}$, with second order accuracy in both Δx and Δy.

We first split the mixed partial derivative:

$$\frac{\partial^3 f}{\partial x \partial y^2} = \frac{\partial}{\partial x}\left(\frac{\partial^2 f}{\partial y^2}\right) = \frac{\partial g}{\partial x}\,,$$

where $g \equiv \frac{\partial^2 f}{\partial y^2}$. The 3-point central difference scheme is suitable for $\frac{\partial^2 f}{\partial y^2}$:

$$g(x_i, y_i) = \left.\frac{\partial^2 f}{\partial y^2}\right|_{x_i, y_i} \approx \frac{f(x_i, y_{i+1}) - 2f(x_i, y_i) + f(x_i, y_{i-1})}{(\Delta y)^2}\,.$$

And for $\frac{\partial g}{\partial x}$, we can use the 2-point central difference scheme:

$$\left.\frac{\partial g}{\partial x}\right|_{x_i, y_i} \approx \frac{g(x_{i+1}, y_i) - g(x_{i-1}, y_i)}{2\Delta x}\,.$$

We can now substitute in the finite difference expression for g, taking note of the subscripts of x and y:

$$g(x_{i+1}, y_i) = \frac{f(x_{i+1}, y_{i+1}) - 2f(x_{i+1}, y_i) + f(x_{i+1}, y_{i-1})}{(\Delta y)^2}\,,$$

and

$$g(x_{i-1}, y_i) = \frac{f(x_{i-1}, y_{i+1}) - 2f(x_{i-1}, y_i) + f(x_{i-1}, y_{i-1})}{(\Delta y)^2}\,,$$

so

$$\begin{aligned}\left.\frac{\partial g}{\partial x}\right|_{x_i, y_i} &\approx \frac{1}{2\Delta x}\left[\frac{f(x_{i+1}, y_{i+1}) - 2f(x_{i+1}, y_i) + f(x_{i+1}, y_{i-1})}{(\Delta y)^2} - \frac{f(x_{i-1}, y_{i+1}) - 2f(x_{i-1}, y_i) + f(x_{i-1}, y_{i-1})}{(\Delta y)^2}\right] \\ &= \frac{f(x_{i+1}, y_{i+1}) - 2f(x_{i+1}, y_i) + f(x_{i+1}, y_{i-1}) - f(x_{i-1}, y_{i+1}) + 2f(x_{i-1}, y_i) - f(x_{i-1}, y_{i-1})}{2\Delta x\,(\Delta y)^2}\,.\end{aligned}$$

Exercise

Repeat the above, but in the opposite order for x and y. Satisfy yourself that regardless of the sequence, you still obtain the same finite difference expression at the end.

5.6 Summary

For a one-dimensional integral with definite limits $\int_a^b f(x)\,dx$, where $f(x)$ is either a known function or a set of data $(x, f(x))$, we can use several numerical methods:

- Rectangle & midpoint methods

  $f(x)$ is approximated as a constant $\Rightarrow$ area under the curve $\approx$ a rectangle of width $(b-a)$ and height corresponding to the functional value at either of the limits or the midpoint.

- Trapezoidal rule

  $f(x)$ is approximated as a linear function $\Rightarrow$ area under the curve $\approx$ a trapezoid of width $(b-a)$.

- Simpson's rules

  $f(x)$ is approximated as a quadratic (cubic) function using the functional values at two (three) intervals of equal size.

Higher accuracy can be achieved by multiple application i.e. when the domain is split into N smaller intervals of equal width ($\Delta x \equiv \frac{b-a}{N}$) by points $x_0 = a, x_1, \ldots, x_N = b$.

Method	**Formula** (s: single; c: composite)
Rectangle method	s: $(b-a)f(a)$ or $(b-a)f(b)$
	c: $\Delta x \sum_{i=1}^{N} f(x_i)$
Midpoint method	s: $(b-a)f\left(\frac{a+b}{2}\right)$
	c: $\Delta x \sum_{i=0}^{N-1} f\left(\frac{x_i+x_{i+1}}{2}\right)$
Trapezoidal rule	s: $\frac{1}{2}(b-a)\left(f(a)+f(b)\right)$
	c: $\frac{1}{2}\Delta x\left[f(x_0)+2\sum_{i=1}^{N-1} f(x_i)+f(x_N)\right]$
Simpson's 1/3 rule	s: $\frac{1}{3}\Delta x\left[f(x_0)+4f(x_1)+f(x_2)\right]$
(odd #points)	c: $\frac{1}{3}\Delta x\left[f(x_0)+4\sum_{i=1,3,5}^{N-1} f(x_i)+2\sum_{j=2,4,6}^{N-2} f(x_j)+f(x_N)\right]$
Simpson's 3/8 rule	s: $\frac{3}{8}\Delta x\left[f(x_0)+3f(x_1)+3f(x_2)+f(x_3)\right]$
(even #points)	c: composite 1/3 rule + single 3/8 rule

The following methods are best used when $f(x)$ is a known function (i.e. not a set of data):

- Gauss Quadrature

  The integral is approximated as a linear combination of weighted functional values:

  $$I = \int_{-1}^{1} f(x)\,dx \approx c_0 f(x_0) + c_1 f(x_1) + c_2 f(x_2) + \cdots + c_{n-1} f(x_{n-1})\,.$$

  The weights and locations to evaluate the function are given in Table 1 (repeated below):

Points (n)	**Weights** (c)	**Locations** (x)
2	1	$\pm\frac{1}{\sqrt{3}} \approx \pm 0.577350$
3	$\frac{8}{9} \approx 0.888889$	0
	$\frac{5}{9} \approx 0.555556$	$\pm\sqrt{\frac{3}{5}} \approx 0.774597$
4	$\frac{1}{36}\left(18+\sqrt{30}\right) \approx 0.652145$	$\pm\frac{1}{35}\sqrt{525-70\sqrt{30}} \approx \pm 0.339981$
	$\frac{1}{36}\left(18-\sqrt{30}\right) \approx 0.347855$	$\pm\frac{1}{35}\sqrt{525+70\sqrt{30}} \approx \pm 0.861136$
5	$\frac{128}{225} \approx 0.568889$	0
	$\frac{1}{900}\left(322+13\sqrt{70}\right) \approx 0.478629$	$\pm\frac{1}{21}\sqrt{245-14\sqrt{70}} \approx \pm 0.538469$
	$\frac{1}{900}\left(322-13\sqrt{70}\right) \approx 0.236927$	$\pm\frac{1}{21}\sqrt{245+14\sqrt{70}} \approx \pm 0.906180$

- Romberg Integration

  Here, two estimates (obtained from Trapezoidal rule) are combined linearly to achieve a more accurate result. Richardson's extrapolation is used here:

  $$I \approx \frac{4}{3} I(h_2) - \frac{1}{3} I(h_1)\,,$$

  where $I(h_2)$ is the estimate obtained when using twice the number of intervals of $I(h_1)$.

  This result can be combined with another of similar accuracy (i.e. obtained from Trapezoidal rule estimates but using double the intervals) to achieve an even higher accuracy. This process is outlined in Romberg's integration algorithm:

  1. Obtain two 'level 1' estimates using Trapezoidal rule (the second must have twice the number of intervals as the first).

2. Use Richardson's extrapolation to obtain a 'level 2' estimate.
3. If a more accurate estimate is required, obtain another 'level 1' estimate using the Trapezoidal rule (double the number of intervals again).
4. Use Richardson's extrapolation on the 2 most accurate Trapezoidal rule estimates to obtain a new 'level 2' estimate.
5. Combine the two 'level 2' estimates using the appropriate form of Richardson's extrapolation to obtain a 'level 3' estimate.
6. Repeat the process (from step #4), ending with an increased 'level' each time.

The general form of Richardson's extrapolation is:

$$I_{j,k} \approx \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}\,, \qquad \epsilon \sim O(h^{2k})\,,$$

where $j+1$ denotes the more accurate estimate compared to j and k is the 'level' of integration.

- Adaptive Quadrature

  Boole's Rule is:

$$I \approx \frac{16}{15} I(h_2) - \frac{1}{15} I(h_1)\,,$$

  where $h_1 = 2h_2$. The algorithm used to refine interval size only when needed is as follows:

  1. Obtain two estimates using Simpson's $1/3$ rule (the second must have twice the number of intervals as the first).
  2. Calculate the absolute error between the two estimates.
     - If the absolute error is within threshold, obtain a final estimate using Boole's Rule.
     - If not, the integral is split in two and the algorithm is applied to each subintegral starting from step 1 and then finally added together.

For **multiple** integrals, each integral is approximated individually starting from the outermost integral. For instance,

$$\int_a^b \int_c^d f(x,y)\,dxdy = \int_a^b G(y)\,dy\,,$$

where $G(y) = \int_c^d f(x, y)\, dx$. These 'one-dimensional' integrals can be approximated using the methods above.

Unbounded integrals can:

- have their infinite bound(s) replaced with a finite bound which is increased gradually until the integral reaches a converged value; or
- be transformed so the infinite bound(s) are finite. This will likely then become a singularity.

For an integral containing **singularities**, they can be rewritten as a sum of smaller integrals such that any singularities are at the limit(s). Then either Gauss quadrature or an open integration method (Table 2 replicated below) can be applied.

Intervals (n)	**Newton-Cotes Open Integration Formula** ($x_i = a + i\left(\frac{b-a}{n}\right)$)
2	$(b-a)\, f(x_1)$
3	$(b-a)\left[f(x_1) + f(x_2)\right]/2$
4	$(b-a)\left[2f(x_1) - f(x_2) + 2f(x_3)\right]/3$
5	$(b-a)\left[11f(x_1) + f(x_2) + f(x_3) + 11f(x_4)\right]/24$
6	$(b-a)\left[11f(x_1) - 14f(x_2) + 26f(x_3) - 14f(x_4) + 11f(x_5)\right]/20$

For **numerical differentiation**, we either:

- determine a fit for the data set and then differentiate the function as required; or
- use a finite difference scheme.

Finite difference schemes can be derived using Taylor series expansions: if we want a finite difference expression for the n-th derivative of function $f(x)$ to have an OOA of p, then we will need:

- $n + p - 1$ neighbouring points,
- to generate $n + p - 1$ equations (Taylor series),

- the last term of which will have Δx raised to the power $n + p - 1$,
- to solve for the $n + p - 1$ unknown derivatives in the FD expression.

A more accurate approximation of a derivative can be obtained by applying Richardson's Extrapolation:

$$D = \frac{4}{3} D\left(\frac{h}{2}\right) - \frac{1}{3} D(h) + O(h^4)\,,$$

where $D(h)$ and $D\left(\frac{h}{2}\right)$ are the estimates obtained from a finite difference scheme with truncation error $\sim O(h^2)$.

6 Differential Equations

6.1 Introduction

Since we covered numerical methods for integration and differentiation in the previous chapter, it seems quite logical to move onto differential equations which involve derivatives. Generally we obtain differential equations from the governing equations of the system(s) we are studying. We will use ODEs to discuss the more common numerical methods, and then introduce the methods that are more suitable for multi-variable problems i.e. PDEs.

6.2 Ordinary Differential Equations (ODEs)

For ordinary differential equations, the numerical methods are broadly split into 'one-step' methods and 'multi-step' methods. 'One-step' methods – or the **Runge-Kutta** methods – are called so because only y_i, the functional value at the 'current' point, is needed (along with the differential equation, of course) to determine y_{i+1}, the functional value at the next point. 'Multi-step' methods, on the other hand, require functional values from more than the current location in order to proceed.

6.2.1 Runge-Kutta methods

Runge-Kutta methods all share the same general form:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2 + ... + a_n k_n)\, h\,,$$

where $y = y(x)$ is the function of interest, the polynomial in the brackets is called an **increment function** (we can think of it as something similar, but not necessarily equal to, the gradient), all the a's are non-zero constants, k's are **recurrence relationships** (because $k_2 = k_2(k_1)$, $k_3 = k_3(k_2)$ etc.) and n is the order of the Runge-Kutta method. In general, the higher-order Runge-Kutta methods are more accurate than lower-order methods, but they are also more computationally expensive.

1st order – Euler's method

The simplest is the 1st order Runge-Kutta method. It is known as Euler's method, Euler-Cauchy method or point-slope method:

$$y_{i+1} = y_i + f(x_i, y_i)h\,,$$

where $f(x, y) = \frac{dy}{dx}$. $a_1 = 1$ and $k_1 = f(x_i, y_i)$ in the general Runge-Kutta form. Here the increment function is equal to the slope at the beginning of the interval, as illustrated in Figure 42.

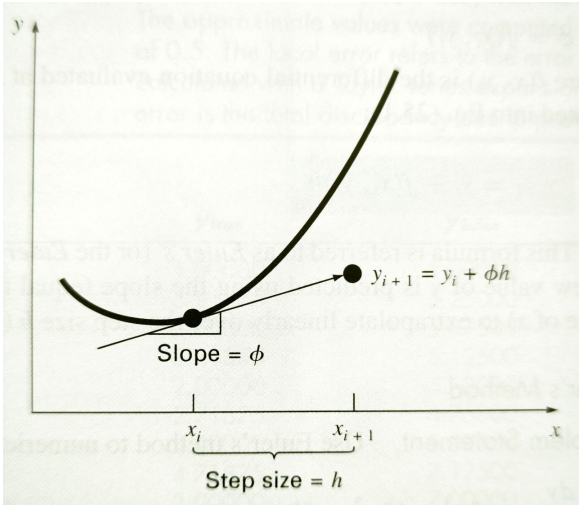


Figure 42: Illustration of Euler's method, from *Chapra & Canale*, Fig. 25.1 on pg707.

It is essentially a linear extrapolation using the slope at x_i. Note that if the ODE is 1D, that is if $\frac{dy}{dx} = f(x)$, then using Euler's method would look similar (graphically) to drawing a trapezoid. Of course we are typically interested in solving for more than just y_{i+1}, so we would be more akin to drawing a series of trapezoids each corresponding to successive applications of Euler's method.

Example

Use Euler's method to obtain a numerical solution $y(x)$ for $\frac{dy}{dx} = 3x^2 - 2x + 5$ from $x = 0$ to 10, given that $y = 0$ at $x = 0$.

We first need to choose a suitable step (or interval) size, for e.g. $h \equiv \Delta x = 2$. The initial values are $x_0 = 0$ and $y_0 = 0$. The increment function $f(x, y) = 3x^2 - 2x + 5$ is the slope.

We begin our calculations:

$$
\begin{aligned}
y_1 &= y_0 + f(x_0, y_0)\, h = 0 + \left(3(0)^2 - 2(0) + 5\right) \times 2 = 10 \\
y_2 &= y_1 + f(x_1, y_1)\, h = 10 + \left(3(2)^2 - 2(2) + 5\right) \times 2 = 36 \\
y_3 &= y_2 + f(x_2, y_2)\, h = 36 + \left(3(4)^2 - 2(4) + 5\right) \times 2 = 126 \\
y_4 &= y_3 + f(x_3, y_3)\, h = 126 + \left(3(6)^2 - 2(6) + 5\right) \times 2 = 328 \\
y_5 &= y_4 + f(x_4, y_4)\, h = 328 + \left(3(8)^2 - 2(8) + 5\right) \times 2 = 690
\end{aligned}
$$

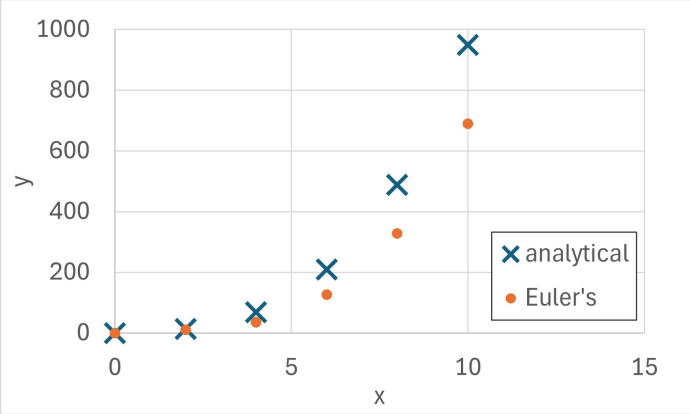


Figure 43: Comparison of analytical and numerical (from Euler's method) solutions.

The corresponding analytical answers are $y = 0$, 14, 68, 210, 488 and 950. By comparing the two solutions plotted in Figure 43, we can see that although the general trend of the numerical solution is correct, the error very quickly becomes quite significant. (Don't be fooled by the graph - the vertical axis has intervals of 200!) This can, of course, be improved by using smaller step sizes but it may be worth considering the higher accuracy afforded by higher-order methods.

Note: you may have noticed that if we rearrange the terms, we obtain the explicit Euler forward finite difference scheme from Section 5.5.1. That might lead you to think that we could also have used other finite difference schemes; indeed we can. In fact a lot of the methods that will be introduced in this section also pop up in texts on finite difference schemes (i.e. for approximating derivatives) because even if their goal appears to be different, they tend to arise from similar derivations (Taylor series).

2nd order - Heun's method & Midpoint method

We will not go into the derivation relating to the 2nd order Runge-Kutta methods (you can refer to *Chapra & Canale, p729*) and instead jump to the conclusion regarding the general form:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)\, h\,,$$

where

$$\begin{aligned} a_1 &= 1 - a_2 \\ k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2a_2}h, y_i + \frac{1}{2a_2}k_1 h\right)\,. \end{aligned}$$

Notice that a_2 has not been defined, which means there are an infinite possible 2nd order Runge-Kutta methods. (Recall that a_2 is non-zero; if it were then this would be reduced to the 1st order Euler's method.) Let us look at some of the more common versions: **Heun's method** ($a_2 = \frac{1}{2}$) and the **Midpoint method** ($a_2 = 1$).

For Heun's method, $a_2 = \frac{1}{2}$ leads to:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{2}(k_1 + k_2)\, h\,, \\ k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + h, y_i + k_1 h\right)\,. \end{aligned}$$

We know that k_1 is the slope at the start of the interval; this k_2 on the other hand is actually the slope at the end of the interval obtained using Euler's method. So essentially what happens in Heun's method is a 2-stage process (illustrated in Figure 44):

$$\begin{aligned} \hat{y}_{i+1} &= y_i + f(x_i, y_i) h\,, \\ y_{i+1} &= y_i + \frac{1}{2}\left(f(x_i, y_i) + f(x_{i+1}, \hat{y}_{i+1})\right) h\,. \end{aligned}$$

The first stage is called the 'predictor' stage, because we are predicting the functional value at x_{i+1}. The second stage is the 'corrector' stage, where we correct/improve our prediction. Here, specifically, we use the average of the slopes at the start and the end of the interval. We can understand how this would be an improvement on the Euler's method, because we are now trying to account for the change in curvature within the interval.

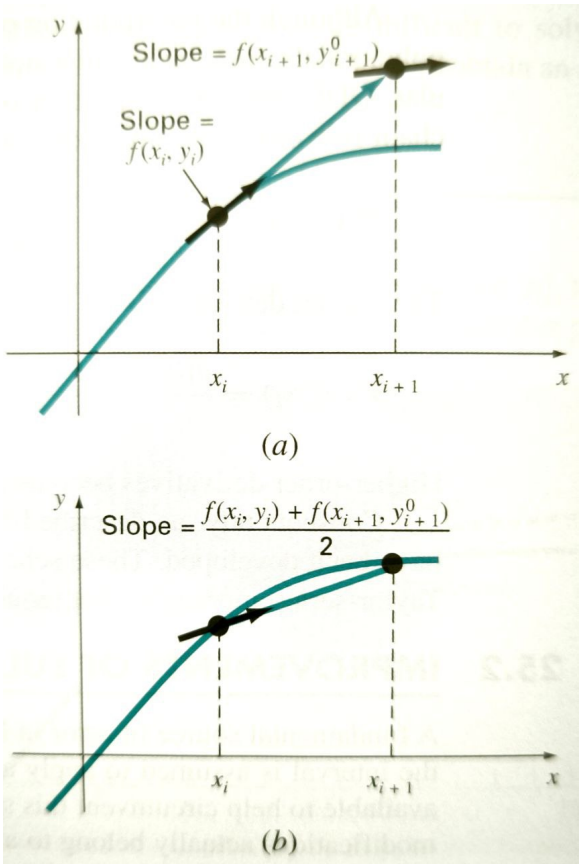


Figure 44: Illustration of Heun's method: (a) predictor and (b) corrector stages, from *Chapra & Canale*, Fig. 25.9 on pg720.

You may now be thinking 'why do we still consider this a 'one-step' method when there are 2 stages?' It does seem to be contradictory, but perhaps because both stages can be combined into a single expression (i.e. the predicted term $\hat{y}_{i+1}$ doesn't even have to appear) it is still classed as a 'one-step' numerical method.

Additionally, recall the iterative formulae we encountered in the root-finding topic (Section 3.3)? We can do so similarly here. That is, if we aren't satisfied with the accuracy of our estimate, we can attempt to improve it by iterating the corrector stage and updating the $\hat{y}_{i+1}$ with the previous y_{i+1} estimate.

Example

Use Heun's method to obtain a numerical solution $y(x)$ for $\frac{dy}{dx} = 3x^2 - 2x + 5$ from $x = 0$ to 10, given that $y = 0$ at $x = 0$.

Just as with the Euler's method, we first need to choose a suitable step (or interval) size. For ease of comparison let us use the same value i.e. $h \equiv \Delta x = 2$. We will denote the initial conditions as $(x_0, y_0) = (0, 0)$ and $f(x, y) \equiv \frac{dy}{dx} = 3x^2 - 2x + 5$. We apply Heun's method to estimate y_1 (at $x_1 = 2$):

$$\begin{aligned}
f(x_0, y_0) &= 3x_0^2 - 2x_0 + 5 = 3 \times 0^2 - 2 \times 0 + 5 = 5\,, \\
\hat{y}_1 &= y_0 + f(x_0, y_0)h = 0 + 5 \times 2 = 10\,, \\
f(x_1, \hat{y}_1) &= 3x_1^2 - 2x_1 + 5 = 3 \times 2^2 - 2 \times 2 + 5 = 13\,, \\
y_1 &= y_0 + \frac{1}{2}\left(f(x_0, y_0) + f(x_1, \hat{y}_1)\right)h = 0 + \frac{1}{2}(5 + 13)(2) = 18\,.
\end{aligned}$$

The process is the same for estimating y_2, y_3, y_4 and y_5.

$$\begin{aligned}
f(x_1, y_1) &= 3x_1^2 - 2x_1 + 5 = 3 \times 2^2 - 2 \times 2 + 5 = 13\,, \\
\hat{y}_2 &= y_1 + f(x_1, y_1)h = 18 + 13 \times 2 = 44\,, \\
f(x_2, \hat{y}_2) &= 3x_2^2 - 2x_2 + 5 = 3 \times 4^2 - 2 \times 4 + 5 = 45\,, \\
y_2 &= y_1 + \frac{1}{2}\left(f(x_1, y_1) + f(x_2, \hat{y}_2)\right)h = 18 + \frac{1}{2}(13 + 45)(2) = 76\,.
\end{aligned}$$

$$\begin{aligned}
f(x_2, y_2) &= 3x_2^2 - 2x_2 + 5 = 3 \times 4^2 - 2 \times 4 + 5 = 45\,, \\
\hat{y}_3 &= y_2 + f(x_2, y_2)h = 76 + 45 \times 2 = 166\,, \\
f(x_3, \hat{y}_3) &= 3x_3^2 - 2x_3 + 5 = 3 \times 6^2 - 2 \times 6 + 5 = 101\,, \\
y_3 &= y_2 + \frac{1}{2}\left(f(x_2, y_2) + f(x_3, \hat{y}_3)\right)h = 76 + \frac{1}{2}(45 + 101)(2) = 222\,.
\end{aligned}$$

$$\begin{aligned}
f(x_3, y_3) &= 3x_3^2 - 2x_3 + 5 = 3 \times 6^2 - 2 \times 6 + 5 = 101\,, \\
\hat{y}_4 &= y_3 + f(x_3, y_3)h = 222 + 101 \times 2 = 424\,, \\
f(x_4, \hat{y}_4) &= 3x_4^2 - 2x_4 + 5 = 3 \times 8^2 - 2 \times 8 + 5 = 181\,, \\
y_4 &= y_3 + \frac{1}{2}\left(f(x_3, y_3) + f(x_4, \hat{y}_4)\right)h = 222 + \frac{1}{2}(101 + 181)(2) = 504\,.
\end{aligned}$$

$$\begin{aligned}
f(x_4, y_4) &= 3x_4^2 - 2x_4 + 5 = 3 \times 8^2 - 2 \times 8 + 5 = 181\,, \\
\hat{y}_5 &= y_4 + f(x_4, y_4)h = 504 + 181 \times 2 = 866\,, \\
f(x_5, \hat{y}_5) &= 3x_5^2 - 2x_5 + 5 = 3 \times 10^2 - 2 \times 10 + 5 = 285\,, \\
y_5 &= y_4 + \frac{1}{2}\left(f(x_4, y_4) + f(x_5, \hat{y}_5)\right)h = 504 + \frac{1}{2}(181 + 285)(2) = 970\,.
\end{aligned}$$

Comparing against the analytical answer ($y = 0$, 14, 68, 210, 488 and 950) we can see that these estimates are much better than those from the Euler's method, as expected for a higher-order method.

Note: You may also have noticed that the slopes of the predictor and corrector were identical – this is because the increment function in our example was independent of y and hence the different values of predictor and corrector did not affect the calculated slopes. Additionally because of this, iterating the corrector stage (to obtain higher accuracy) would have been ineffectual.

For the Midpoint method, $a_2 = 1$ leads to:

$$\begin{aligned} y_{i+1} &= y_i + k_2 h\,, \\ k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)\,. \end{aligned}$$

Since k_1 is the slope at the start of the interval, this k_2 is the slope at the middle of the interval (and hence the name!) obtained using Euler's method. So while Heun's method involves predicting the functional value at the end and using that slope, the Midpoint method predicts the functional value at the middle instead (illustrated in Figure 45). Unlike Heun's method though, only the slope at the midpoint is used to estimate y_{i+1}:

$$\begin{aligned} \hat{y}_{i+1/2} &= y_i + f(x_i, y_i)\frac{h}{2}\,, \\ y_{i+1} &= y_i + f(x_{i+1/2}, \hat{y}_{i+1/2})h\,. \end{aligned}$$

As with Heun's method, the Midpoint method is an improvement on Euler's method because the slope used is not solely from the start of the interval. We would thus expect that the estimates obtained from the Midpoint method would be fairly similar to that from Heun's method, but a definite improvement compared to Euler's method.

Example

Use the Midpoint method to obtain a numerical solution $y(x)$ for $\frac{dy}{dx} = 3x^2 - 2x + 5$ from $x = 0$ to 10, given that $y = 0$ at $x = 0$.

We will use the same interval size as before ($h = \Delta x = 2$). The process is also similar to

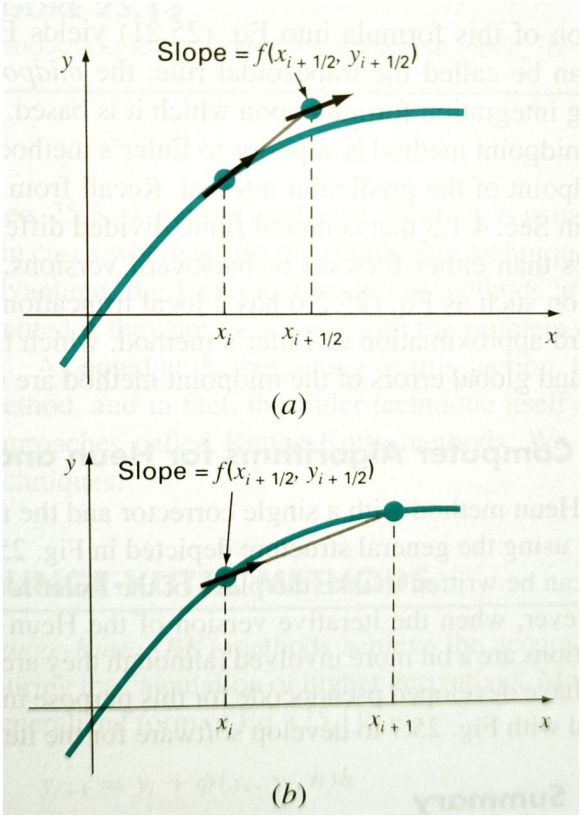


Figure 45: Illustration of the Midpoint method: (a) predictor and (b) corrector stages, from *Chapra & Canale*, Fig. 25.12 on pg725.

Heun's method.

$$
\begin{aligned}
f(x_0, y_0) &= 3x_0^2 - 2x_0 + 5 = 3 \times 0^2 - 2 \times 0 + 5 = 5\,, \\
\hat{y}_{1/2} &= y_0 + f(x_0, y_0)\frac{h}{2} = 0 + 5 \times \frac{2}{2} = 5\,, \\
f(x_{1/2}, \hat{y}_{1/2}) &= 3x_{1/2}^2 - 2x_{1/2} + 5 = 3 \times 1^2 - 2 \times 1 + 5 = 6\,, \\
y_1 &= y_0 + f(x_{1/2}, \hat{y}_{1/2})h = 0 + 6 \times 2 = 12\,.
\end{aligned}
$$

$$
\begin{aligned}
f(x_1, y_1) &= 3x_1^2 - 2x_1 + 5 = 3 \times 2^2 - 2 \times 2 + 5 = 13\,, \\
\hat{y}_{3/2} &= y_1 + f(x_1, y_1)\frac{h}{2} = 12 + 13 \times \frac{2}{2} = 25\,, \\
f(x_{3/2}, \hat{y}_{3/2}) &= 3x_{3/2}^2 - 2x_{3/2} + 5 = 3 \times 3^2 - 2 \times 3 + 5 = 26\,, \\
y_2 &= y_1 + f(x_{3/2}, \hat{y}_{3/2})h = 12 + 26 \times 2 = 64\,.
\end{aligned}
$$

$$
\begin{aligned}
f(x_2, y_2) &= 3x_2^2 - 2x_2 + 5 = 3 \times 4^2 - 2 \times 4 + 5 = 45\,, \\
\hat{y}_{5/2} &= y_2 + f(x_2, y_2)\frac{h}{2} = 64 + 45 \times \frac{2}{2} = 109\,, \\
f(x_{5/2}, \hat{y}_{5/2}) &= 3x_{5/2}^2 - 2x_{5/2} + 5 = 3 \times 5^2 - 2 \times 5 + 5 = 70\,, \\
y_3 &= y_2 + f(x_{5/2}, \hat{y}_{5/2})h = 64 + 70 \times 2 = 204\,.
\end{aligned}
$$

$$
\begin{aligned}
f(x_3, y_3) &= 3x_3^2 - 2x_3 + 5 = 3 \times 6^2 - 2 \times 6 + 5 = 101\,, \\
\hat{y}_{7/2} &= y_3 + f(x_3, y_3)\frac{h}{2} = 204 + 101 \times \frac{2}{2} = 305\,, \\
f(x_{7/2}, \hat{y}_{7/2}) &= 3x_{7/2}^2 - 2x_{7/2} + 5 = 3 \times 7^2 - 2 \times 7 + 5 = 138\,, \\
y_4 &= y_3 + f(x_{7/2}, \hat{y}_{7/2})h = 204 + 138 \times 2 = 480\,.
\end{aligned}
$$

$$
\begin{aligned}
f(x_4, y_4) &= 3x_4^2 - 2x_4 + 5 = 3 \times 8^2 - 2 \times 8 + 5 = 181\,, \\
\hat{y}_{9/2} &= y_4 + f(x_4, y_4)\frac{h}{2} = 480 + 181 \times \frac{2}{2} = 661\,, \\
f(x_{9/2}, \hat{y}_{9/2}) &= 3x_{9/2}^2 - 2x_{9/2} + 5 = 3 \times 9^2 - 2 \times 9 + 5 = 230\,, \\
y_5 &= y_4 + f(x_{9/2}, \hat{y}_{9/2})h = 480 + 230 \times 2 = 940\,.
\end{aligned}
$$

In terms of accuracy, the Midpoint method has a similar order of magnitude to Heun's method which you can observe in Figure 46.

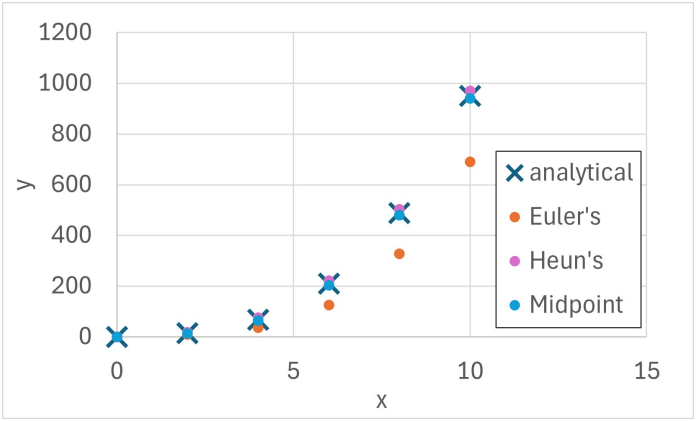


Figure 46: Comparison of analytical and numerical solutions from Euler's, Heun's & Midpoint methods.

Another popular 2nd-order Runge-Kutta method is **Ralston's method** (included in Figure 47) which is obtained when $a_2 = \frac{2}{3}$. Concept-wise it is not very different from Heun's

and the Midpoint methods, just that the locations of, and weightage for, the two slopes involved are different. Hence we will not go into detail about it here.

Exercise

Obtain a numerical solution $y(x)$ for $\frac{dy}{dx} = -2x^3 + 12x^2 - 20x + 8.5$ from $x = 0$ to 4 using a step size of 1, given the initial condition $y = 1$ at $x = 0$. Use Euler's method, Heun's method and the Midpoint method. Do you get the same answers as plotted in Figure 47?

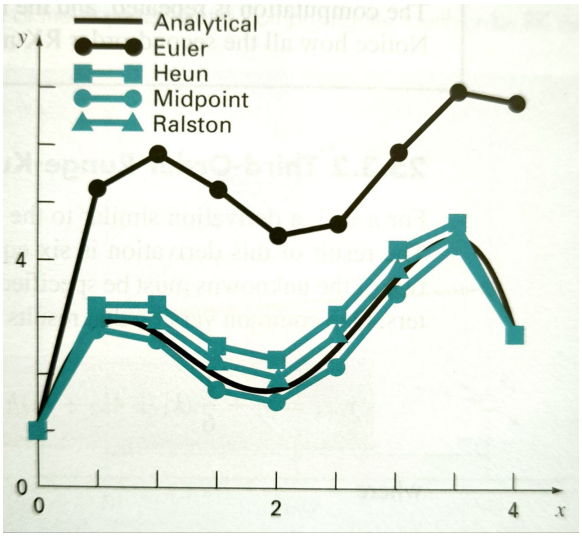


Figure 47: Analytical and numerical solutions, from *Chapra & Canale*, Fig. 25.14 on pg731.

Higher orders

If we are unsatisfied with the level of accuracy obtained by the 2nd-order Runge-Kutta methods, we can of course consider even higher orders. For instance the highly popular 4th-order Runge-Kutta method (also known as the classical version) is:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) h\,, \\ k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \\ k_4 &= f\left(x_i + h, y_i + k_3 h\right)\,. \end{aligned}$$

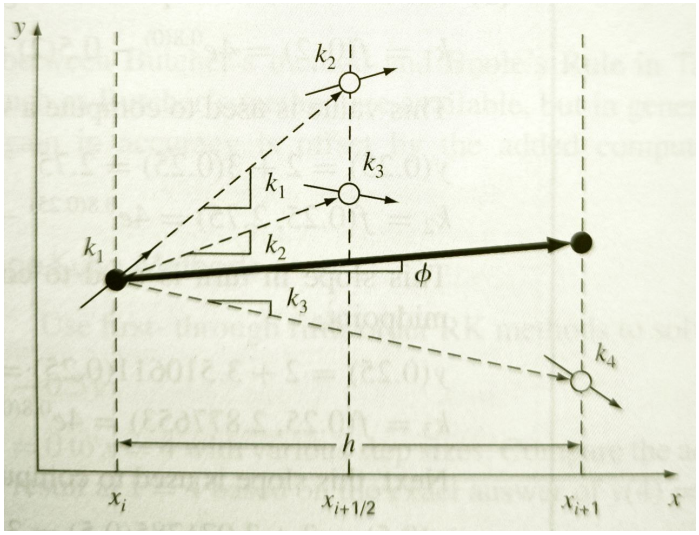


Figure 48: Sketch of the slopes involved in the classical 4th-order Runge-Kutta method, from *Chapra & Canale*, Fig. 25.15 on pg733.

This is a 4-stage process, involving 4 slopes as illustrated in Figure 48. The initial part is identical to the Midpoint method but then involves another two predictions before reaching the corrector stage:

1. The slope at the beginning of the interval (k_1) is used to predict the functional value at the midpoint ($\hat{y}^a_{i+1/2} = y_i + \frac{1}{2}k_1 h$).

2. The slope at the predicted midpoint (k_2) is used to predict the functional value at the midpoint again ($\hat{y}^b_{i+1/2} = y_i + \frac{1}{2}k_2 h$).

3. The slope at the 2nd midpoint prediction (k_3) is used to predict the functional value at the end of the interval ($\hat{y}^c_{i+1} = y_i + k_3 h$).

4. An estimate (the corrector) is obtained via a weighted average of the 4 slopes.

Exercise

Use the classical 4th-order Runge-Kutta method to obtain a numerical solution $y(x)$ for $\frac{dy}{dx} = 3x^2 - 2x + 5$ from $x = 0$ to 10, given that $y = 0$ at $x = 0$, with an interval size of 2.
Analytical ans: $y = 0$, 14, 68, 210, 488, 950.

6.2.2 Adaptive methods

Do you recall how we discussed using adaptive quadrature (previous chapter) as a way to improve computational efficiency? Given the similarity between solving a differential equation and performing an integration, you won't be surprised that we may also consider adaptive methods in this section. This is particularly important if we have regions of both gradual and rapid change, and hence need to change the interval size accordingly.

In general, since we do not know how exactly the function looks like (since that's what we are trying to solve for), we need to determine the local truncation error at each step/iteration of the numerical solution process in order to gauge how (or if) we should adjust the interval size. Of course we cannot obtain the exact truncation error (because again we do not know the true solution) so some measure of the local error has to be determined. This idea is similar to the relative error that we used often in the chapter on root-finding.

Adaptive Runge-Kutta method

This method is also known as the **Step-Halving method**. Here we obtain the local error by comparing two predictions of y_{i+1}:

- $\hat{y}^a_{i+1}$ from a single-step prediction;
- $\hat{y}^b_{i+1}$from two half-step predictions: i.e. estimate $y_{i+1/2}$ and then y_{i+1}.

The reason for the method's name should now be obvious. We know that the second estimate is likely to be more accurate – the interval size is not only smaller, it also looks similar to the Midpoint method which has higher accuracy compared to Euler's method. The difference between the two is used as a measure of the local truncation error. In fact, some researchers also use this to create a correction term which is then applied to one of the predictions e.g. $y_{i+1} = \hat{y}_{i+1} + \epsilon/15$, where $\epsilon \equiv \hat{y}^a_{i+1} - \hat{y}^b_{i+1}$.

Runge-Kutta Fehlberg method

Another popular way to gauge the local error is the Runge-Kutta Fehlberg method, also called the **embedded Runge-Kutta method**. Instead of comparing predictions from two interval sizes, we compare predictions from Runge-Kutta methods of different orders. The

idea is similar to the step-halving method, but could be more expensive computationally because as we know, higher orders of the Runge-Kutta method require evaluating more functional values and their corresponding slopes.

One way to reduce this is to choose a pair of methods that have overlapping function evaluations i.e. some of the definitions for the k terms are identical. For instance, the **Cash-Karp** version of the Runge-Kutta Fehlberg method uses a pair of 4th and 5th order Runge-Kutta methods which have identical definitions for k_1 to k_5, and hence there is only one 'extra' evaluation (k_6 for the 5th order method).

Interval size control

The two methods above only suggest ways to determine a local error, but not the next step of modifying the interval size. Technically there is no official 'correct' way to do so; even 'trial and error' works but is of course inefficient. The general idea is to increase the interval size when the local relative error is small, and decrease the interval size when the local relative error is large. This of course means that there has to be some threshold(s) to determine 'small' and 'large'.

One suggestion for interval size control by Press et al.[4] is:

$$h_{new} = h_{current} \left| \frac{\epsilon_{crit}}{\epsilon} \right|^{\alpha}$$

where ϵ_{crit} is the specified threshold and

$$\alpha = \begin{cases} 0.2, & \text{if} \quad \epsilon \leq \epsilon_{crit} \\ 0.25, & \text{if} \quad \epsilon > \epsilon_{crit} \end{cases}$$

In this manner the change in interval size is dependent on the ratio of the local error to the specified threshold, and moderated by the power α.

Unfortunately, adaptive algorithms like this do not always work. In particular if the stability of the numerical solution depends on having a small interval size – we do not cover that here, but suffice to say that most explicit one-step methods do! – then we have to maintain a small interval size. Thus we are not able to achieve the hoped-for computational savings. Fortunately we have some options open to us, which we will discuss in the next section.

[4] W.H. Press, S.A. Teukolsky, W.T. Vetterling & B.P. Flannery, *Numerical recipes in C: The art of scientific computing*, 2nd Ed., Cambridge University Press, 1992.

6.2.3 Methods for Stiff Systems

'Stiff' systems are actually what we have described before - a function that has both rapidly and gradually changing components. One easy example to understand would be the transient (complementary) and steady-state (particular) components of a general solution – the transient component typically shows rapid fluctuations (with respect to time) but after a long period the steady-state component begins to dominate and thus the fluctuations are slower and more gradual. Figure 49 shows another example of the solution of a stiff ODE. Note that usually the independent variable here is time (t) rather than space (x). The interval size h will thus refer to timestep Δt in this section.

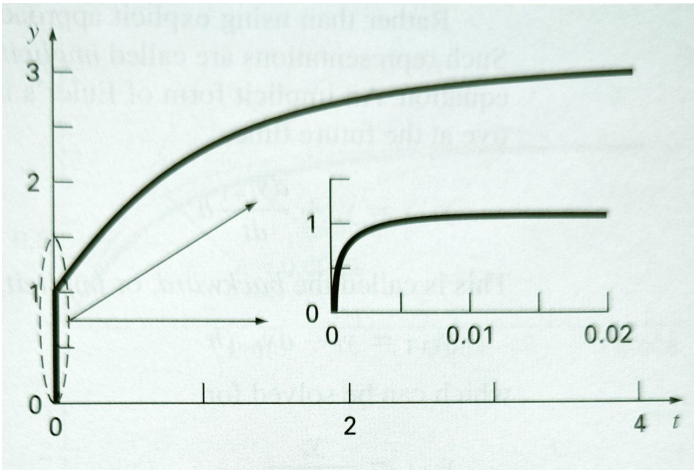


Figure 49: Sketch of the solution from a stiff ODE, from *Chapra & Canale*, Fig. 26.1 on pg753.

Implicit methods

As mentioned previously, for systems such as these, adaptive interval size algorithms do not work. Hence one option is to use **implicit** methods (as opposed to the previously introduced **explicit** methods). Mathematically the implicit methods do not look particularly different, except that the timestep h now tends to appear in the denominator of the y_i coefficient rather than the numerator.

Consider the **implicit Euler backward**, which can be obtained in a much similar way to

the **explicit Euler forward**:

$$y_{i+1} = y_i + hf\left(t_{i+1}, y_{i+1}\right) .$$

If the gradient $f(t,y) \equiv \frac{dy}{dt} = -ay$ (which is the general form for the homogeneous part of an ODE and corresponds to the complementary part of the solution; a is a positive constant), then

$$\begin{aligned} y_{i+1} &= y_i + h\left(-ay_{i+1}\right) \\ \Rightarrow y_{i+1} &= \frac{y_i}{1+ah} . \end{aligned}$$

We can see then, that regardless of the timestep size $y_{i+1} \leq y_i$. Hence as time goes to infinity ($i \to \infty$), y will always approach zero (as expected for a transient solution). This means that we have **unconditional stability**, which is one of the greatest advantages of implicit methods.

The implicit Euler backward has the same order of accuracy as the explicit Euler forward; if we wanted a higher level of accuracy then we could consider the popular **Crank-Nicolson method** which looks like Heun's method applied in the time domain but without the predictor stage:

$$y_{i+1} = y_i + \frac{1}{2}h\left[f\left(t_i, y_i\right) + f\left(t_{i+1}, y_{i+1}\right)\right] .$$

Multi-step methods

Another way to effectively handle stiff ODEs is to use multi-step methods. In the previous section we primarily used 'one-step' methods (even if they had multiple predictor-corrector stages); for multi-step methods, we need to utilise information from *before* the current step i. It's like predicting the weather not just from today's weather, but by considering the trend over the past few days. Mathematically the 'historical data', obtained from multiple steps before 'now', will allow us to have a better idea of the solution trajectory.

Note though that this makes it difficult to incorporate adaptive interval sizes, because then all the 'historical data' has to be recalculated at each timestep. One simple way of circumventing this, is to increase/decrease the step size by 2 so that some of the previous information can still be reused. This idea is illustrated in Figure 50.

Non-Self-starting Heun's method

Just as the implicit Euler backward is the implicit counterpart to the 1st order explicit

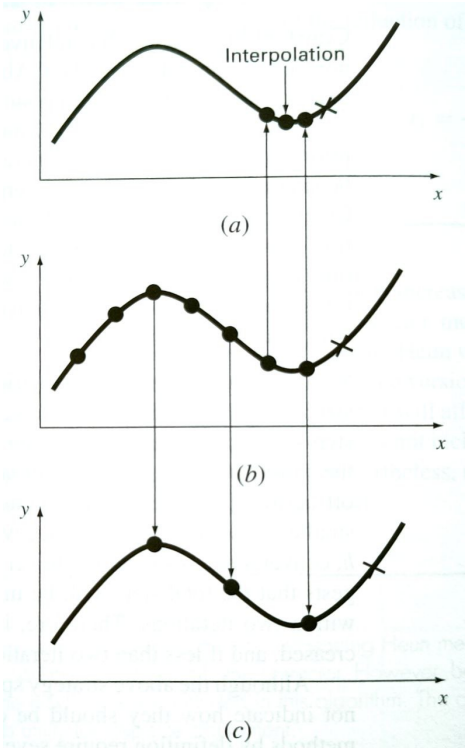


Figure 50: Illustration of how previously calculated information (b) can be reused if interval size is (a) halved or (c) doubled, from *Chapra & Canale*, Fig. 26.6 on pg766.

Euler forward method, we also have an implicit version of the 2nd order Heun's method, known as the non-self-starting Heun's method. 'Self-starting' simply refers to the fact that no additional information is required to complete the calculation process aside from the initial value. Thus 'non-self-starting' means that more information is required in addition to the initial value. In other words, we need more than one initial condition. We will see why, exactly, shortly.

The original Heun's method used the explicit Euler forward as a predictor and then the trapezoidal rule as a corrector. The implicit version's predictor uses information from a timestep prior to the 'current', while the corrector remains the same:

$$\begin{aligned} \hat{y}_{i+1} &= y_{i-1} + f(x_i, y_i)(2h)\,, \\ y_{i+1} &= y_i + \frac{1}{2}\left(f(x_i, y_i) + f(x_{i+1}, \hat{y}_{i+1})\right) h\,. \end{aligned}$$

Because of this change, this predictor works a bit more similarly to the midpoint method – the gradient at the current timestep is used to predict the change over 2 timesteps, from

t_{i-1} to t_{i+1}. Figure 51 As before, we can keep iterating the corrector stage to improve accuracy.

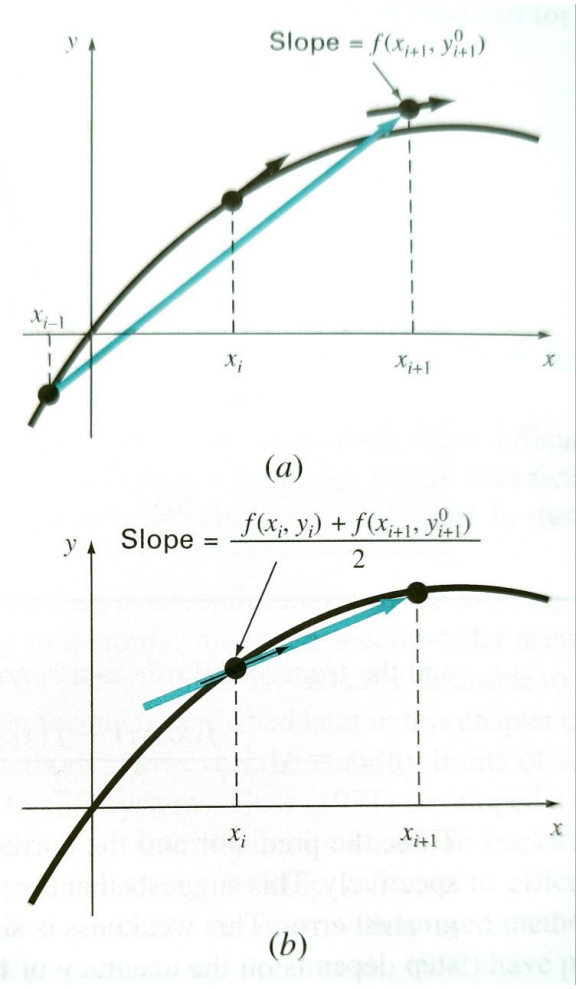


Figure 51: Illustration of the non-self-starting Heun's method, from *Chapra & Canale*, Fig. 26.4 on pg758.

Technically, either/both the predictor and corrector stages can be 'filled' using any of the finite difference schemes we have encountered before, which makes for very many versions of multi-step methods.

Adams formulae

We can also directly 'solve' for the next functional value using the integration methods we

covered before, for instance the Newton-Cotes formulae:

$$y_{i+1} = y_{i-n} + \int_{x_{i-n}}^{x_{i+1}} f_n(x)\,dx \qquad \text{open formula}$$

$$y_{i+1} = y_{i-n+1} + \int_{x_{i-n+1}}^{x_{i+1}} f_n(x)\,dx \qquad \text{closed formula}$$

However we will now introduce a new family of formulae – the **Adams** formulae – which may be preferable because they tend to focus on the last interval (from x_i to x_{i+1}) rather than involving a lot of 'previous' data. This difference is illustrated in Figure 52.

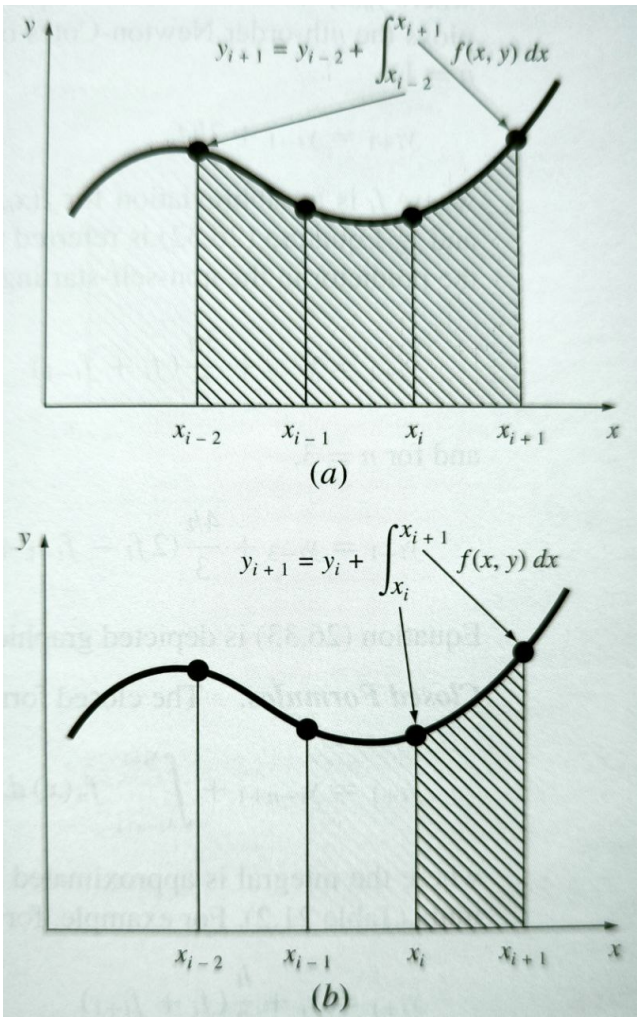


Figure 52: Comparison of Newton-Cotes closed formula versus closed Adams formula, from *Chapra & Canale*, Fig. 26.7 on pg767.

The open Adams formulae are called **Adams-Bashforth** formulae, and generally have the

form:

$$y_{i+1} = y_i + h\sum_{k=0}^{n-1} \beta_k f_{i-k} + O(h^{n+1}),$$

where n is the order of the formula and the coefficients β_k are listed in Table 6. Note: if $n = 1$ we get back Euler's method!

Table 6: Coefficients for Adams-Bashforth predictors

Order	β_0	β_1	β_2	β_3	β_4	β_5
1	1					
2	$\frac{3}{2}$	$-\frac{1}{2}$				
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$			
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$		
5	$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$	
6	$\frac{4277}{1440}$	$-\frac{7923}{1440}$	$\frac{9982}{1440}$	$-\frac{7298}{1440}$	$\frac{2877}{1440}$	$-\frac{475}{1440}$

The closed Adams formulae are called **Adams-Moulton** formulae, and generally have the form:

$$y_{i+1} = y_i + h\sum_{k=0}^{n-1} \beta_k f_{i+1-k} + O(h^{n+1}),$$

where n is the order of the formula and the coefficients β_k are listed in Table 7. Note: if $n = 2$ we get the trapezoidal rule!

Table 7: Coefficients for Adams-Moulton correctors

Order	β_0	β_1	β_2	β_3	β_4	β_5
2	$\frac{1}{2}$	$\frac{1}{2}$				
3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$			
4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$		
5	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$	
6	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$

You may have noticed that for both the Adams-Bashforth and Adams-Moulton formulae, the β_k coefficients sum to 1 and alternate their signs (except for β_0 which is always pos-

itive). Additionally, Adams-Bashforth formulae are typically used for predictors because they are open while Adams-Moulton formulae are used for correctors because they are closed.

6.2.4 Systems of ODEs

Previously we have been considering 1D problems). However we know that most engineering problems are more complex and typically involve at least 2 independent variables.

So the first issue is how to deal with a coupled set of ODEs. We may end up with these simultaneous ODEs either because we have both time t and space x variables, or perhaps because we have done some discretisation of the spatial domain and thus have variables x_1, x_2, x_3... etc. (we will discuss this in the section on elliptic equations). We handle these the same way we did the single ODEs before, using the same methods but on the individual ODEs one at a time. The only things we need to be careful of are:

- keep track of the subscripts of each variable; and
- for Runge-Kutta methods, the slopes (the k functions) should all be evaluated at the same location before being used for the next stage.

6.3 Partial Differential Equations (PDEs)

In this section we will consider PDEs, which we are more likely to encounter in engineering systems because most variables will vary with both time and space. Generally the behaviour (or type of equation) falls into three categories: **elliptic**, **parabolic** and/or **hyperbolic**. Elliptic equations typically describe steady-state behaviour, that is there is no variation in time. Parabolic and hyperbolic equations both have time-varying behaviour and thus usually describe propagation problems; the only difference is that the solutions of parabolic equations tend towards an equilibrium while solutions of hyperbolic equations are sinusoidal.

Aside from the type of differential equation, we also need to consider the initial/boundary conditions and how to incorporate them appropriately. These conditions are either **Dirichlet** (value given explicitly) or **Neumann** (a derivative of the function is given) or a combination of both.

6.3.1 Elliptic Equations

As mentioned in the introduction, elliptic equations describe steady-state problems. This is because time is not one of their independent variables, so any variation is only in the spatial domain. **Laplace's equation** is one of these, which you may have seen before describing the temperature distribution in head conduction problems, potential flow in fluid mechanics, or electric potential in an electric field without sources or sinks.

This type of PDE is essentially describing a **boundary value problem**, because time is not a variable so 'initial conditions' would not really make sense. Instead we have boundary conditions: the dependent variable (the function of interest) has specified values at certain spatial locations, typically the boundaries (hence the name). With initial value problems we just had to substitute in the given information at $x = 0$ as we did in the previous section; for BVPs we can consider these two options below.

Shooting method

This method 'converts' the boundary value problem into an initial value problem, and is

somewhat of a mix between trial and error and interpolation. The gist of it is as follows:

1. guess an intial value and integrate the ODE(s) to estimate the value at the boundary.
2. repeat the above but using a different guess for the initial value.
3. considering the two outcomes, perform an interpolation to determine what initial condition will achieve the correct boundary value.
4. solve the problem using the determined initial condition.

If the ODE is linear, then the functional values will also be linear. Hence a linear interpolation (or similar triangles, if you prefer) will work well to determine the 'correct' initial condition. If the ODE is nonlinear, then we could 'shoot' more and then apply other interpolation or root-finding methods.

Finite-difference methods

This method makes use of the high-accuracy differentiation formulae from the previous chapter to approximate the derivatives that appear in the ODE. Once we do so the ODE will then become a coupled set of ODE(s) for which we can directly incorporate the boundary conditions (because they are the specified values of some of the terms). We can then combine these into a matrix equation and solve using linear algebra.

This approach is good if we happen to be dealing with problems like vibration or elasticity, because we can then easily obtain the eigenvalues (which have some physical meaning in the problem context, e.g. modes of vibration).

In order to illustrate this more clearly, let us use the example of Laplace's equation in the context of temperature distribution on a square plate:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0\,.$$

Figure 53 shows how we could potentially divide up (**discretise**) the spatial domain as well as the specified boundary conditions – in this case, fixed temperatures along the plate boundaries.

The next step is to use a finite difference scheme to approximate the derivatives. For example we could choose the central difference scheme (for 2nd order derivative) for both

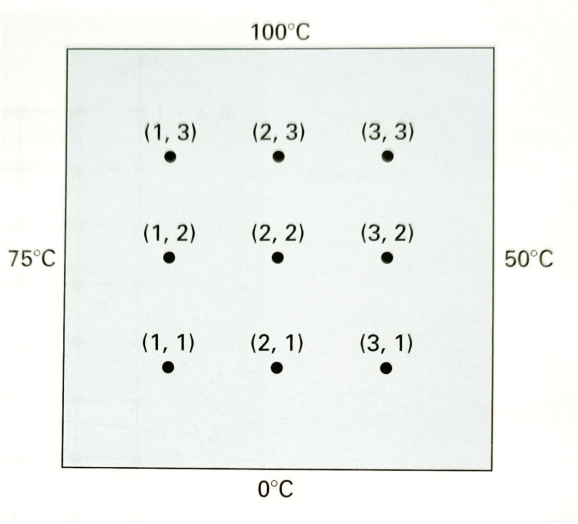


Figure 53: Sketch of discretised domain of square plate with Dirichlet boundary conditions, from *Chapra & Canale*, Fig. 29.4 on pg854.

derivatives, resulting in the algebraic expression:

$$\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x} + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta y} = 0\,,$$

where T_{ij} refers to the temperature T at the location (x_i, y_j). If we rearrange this such that $T_{i,j}$ is the subject on the LHS and all the other terms are on the RHS, then we can see that essentially it means the temperature at any location is a function of those immediately around it (You can refer to Figure 54 which illustrates these 5 grid points).

By substituting in all the values of i and j (for this example they both go from 1 to 3), we will obtain a set of algebraic equations:

$$\begin{aligned}
(1,1): &\quad \tfrac{T_{0,1}-2T_{1,1}+T_{2,1}}{\Delta x} + \tfrac{T_{1,0}-2T_{1,1}+T_{1,2}}{\Delta y} = 0\,,\\
(2,1): &\quad \tfrac{T_{1,1}-2T_{2,1}+T_{3,1}}{\Delta x} + \tfrac{T_{2,0}-2T_{2,1}+T_{2,2}}{\Delta y} = 0\,,\\
(3,1): &\quad \tfrac{T_{2,1}-2T_{3,1}+T_{4,1}}{\Delta x} + \tfrac{T_{3,0}-2T_{3,1}+T_{3,2}}{\Delta y} = 0\,,\\
(1,2): &\quad \tfrac{T_{0,2}-2T_{1,2}+T_{2,2}}{\Delta x} + \tfrac{T_{1,1}-2T_{1,2}+T_{1,3}}{\Delta y} = 0\,,\\
&\quad \vdots\\
(3,3): &\quad \tfrac{T_{2,3}-2T_{3,3}+T_{4,3}}{\Delta x} + \tfrac{T_{3,2}-2T_{3,3}+T_{3,4}}{\Delta y} = 0\,.
\end{aligned}$$

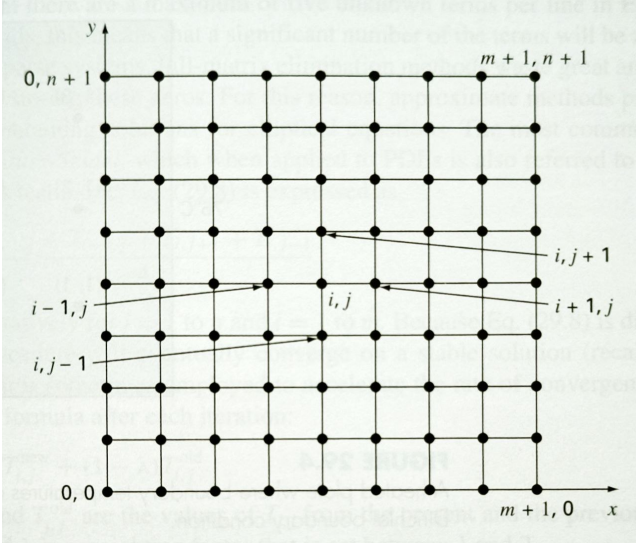


Figure 54: Illustration of 2D grid, from *Chapra & Canale*, Fig. 29.3 on pg853.

The Dirichlet boundary conditions ($T = 0$ at $y = 0$, $T = 75$ at $x = 0$ etc.) translate to:

$$\begin{aligned} T(i,0) &= 0 \\ T(i,4) &= 100 \\ T(0,j) &= 75 \\ T(4,j) &= 50 \end{aligned}$$

which can be substituted into the appropriate algebraic equations.

Finally this set of equations can be solved using linear algebra (if you prefer) or using the Gauss-Seidel method (recall that we introduced this in Section 3.5.2). When applied to PDEs, the Gauss-Seidel method is also called **Liebmann's method**. If we wished to speed up the convergence, we could also apply **overrelaxation**:

$$T_{i,j}^{new} = \lambda T_{i,j}^{new} + (1 - \lambda)\, T_{i,j}^{old},$$

where λ is a weighting factor between 1 and 2.

If you are familiar with heat transfer problems, you may also ask what if we have a boundary condition that corresponds to perfect insulation e.g. $\frac{\partial T}{\partial x} = 0$ at $x = 0$? In other words what if we had a Neumann boundary condition? In that case we just have to express the Neumann boundary condition as a finite difference too, the same way we dealt with the

PDE. The resulting algebraic expression will definitely have some boundary points in it, so we could rearrange the expression into the form of a Dirichlet boundary condition.

Exercise

Approximate the temperature distribution $T(x, y)$ for the square plate in Figure 53, assuming sides of unit length and 4 uniformly-distributed interior grid points (instead of 9). You can use initial values of 0°C, $\lambda = 1.1$ for the overrelaxation, and a maximum relative error of 1%.

Ans: $T_{1,1} = 46.9$°C, $T_{1,2} = 71.9$°C, $T_{2,1} = 40.6$°C, $T_{2,2} = 65.6$°C after 4 iterations.

6.3.2 Parabolic Equations

With parabolic equations, time is one of the independent variables. Typically there is no limit on time i.e. it could go to infinity. Thus we also have to consider the **stability** of the numerical solution – that is, whether or not the errors build up and approach infinity as the time variable increases. Hence we need to choose between explicit and implicit solution methods, the idea of which we have been introduced to before in Section 6.2.3.

We shall use the 1D time-varying heat conduction equation to illustrate the differences between using an explicit and an implicit method, specifically when dealing with the time derivative.

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2}\,,$$

where k is the coefficient of thermal conductivity. Since the spatial derivative is of 2nd order, we can use the central difference scheme; for the temporal derivative we shall use the explicit Euler forward. The resulting algebraic expression is:

$$T_i^{n+1} = T_i^n + \frac{k\Delta t}{(\Delta x)^2}\left(T_{i+1}^n - 2T_i^n + T_{i-1}^n\right)\,.$$

Figure 55 illustrates the grid points involved in this algebraic expression. Note that here we use subscripts n for the timestep, while the figure uses l.

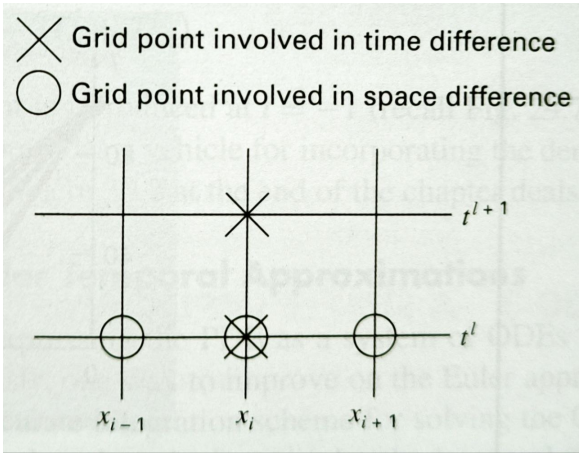


Figure 55: Sketch of grid points involved in spatial and temporal (explicit) finite difference schemes, from *Chapra & Canale*, Fig. 30.3 on pg873.

We expect that as both the timestep size and grid size get smaller (i.e. Δt and $\Delta x \to 0$) that the numerical solution will approach the true (unknown) solution. This is called **convergence**. It is linked to stability in the sense that if the solution is unstable, then by definition the errors will grow and thus the solution will not converge. We naturally hope that the errors will attentuate (go to zero) rather than amplify, but it is not a guarantee: even if the numerical solution is stable (the errors do not grow) and we achieve a convergent solution, there may still be non-zero errors present.

Carnahan et al.[5] derived a **stability criterion** for this case, grouping the constants into $\lambda \equiv \frac{k\Delta t}{(\Delta x)^2}$:

- if $\lambda \leq \frac{1}{2}$, then the solution will be convergent and stable. This translates to a maximum timestep $\Delta t = \frac{(\Delta x)^2}{2k}$.
- if $\lambda \leq \frac{1}{4}$, then the the numerical errors will not oscillate.
- if $\lambda = \frac{1}{6}$, then the truncation error is minimised.

This all looks very good but given that we need the grid size (Δx) to be small for accuracy, the stability criterion means that the timestep has to be even smaller. Thus if we need to obtain the solution over a long time period this could be very computationally expensive.

On top of that, if we take a bigger view of how the information is transferred forward in time (illustrated in Figure 56), we can see that only a portion of the grid points contribute to the point of interest, when some that intuitively should do not.

Let us now consider using an implicit method for the temporal derivative – the implicit Euler backward, which has a similar order of accuracy to the explicit Euler forward used previously. The resulting algebraic expression is now:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = k \, \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2} \, ,$$

where now nearly all the terms are from the 'future' timestep t^{n+1}. Figure 57 compares the grid points involved in this algebraic expression to that of the explicit temporal scheme: us-

[5] B. Carnahan, H.A. Luther & J.O. Wilkes, *Applied Numerical Methods*, Wiley, New York, 1969.

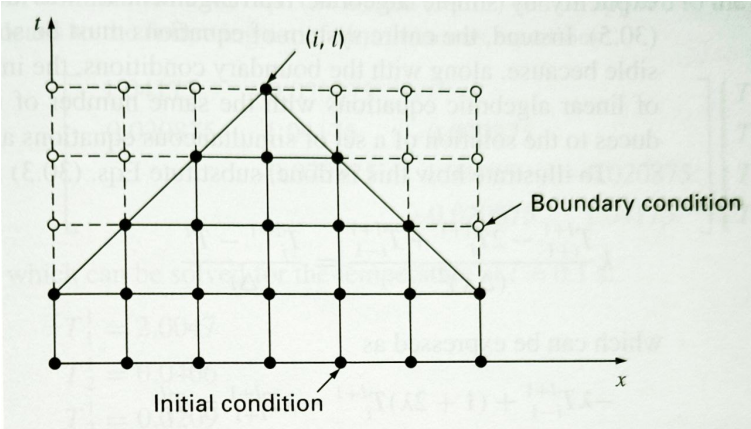


Figure 56: Sketch of grid points involved after a few timesteps for an explicit temporal scheme, from *Chapra & Canale*, Fig. 30.6 on pg877.

ing the implicit scheme, information from the immediate neighbours (spatially) contribute to the point of interest.

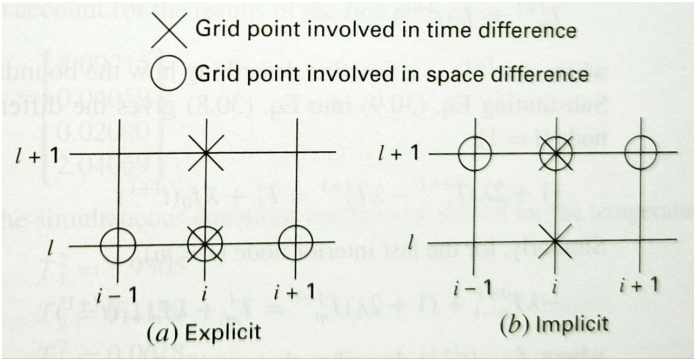


Figure 57: Comparison of grid points involved for (a) explicit and (b) implicit temporal finite difference schemes, from *Chapra & Canale*, Fig. 30.7 on pg877.

We cannot immediately calculate the functional value T_i^{n+1}, but we can solve the coupled set of algebraic expressions that result from applying this finite difference scheme to all the grid points at timestep t^{n+1}. One approach would be the Gauss-Seidel method from Section 3.5.2.

Note: if written as a matrix equation of the form $\mathbf{A}\vec{T}^{n+1} = \vec{T}^n$, we will find that the coefficient matrix $\mathbf{A}$ will be tridiagonal due to the central difference scheme for the spatial

derivative. This is not within our syllabus, but **A** is a type of special matrix for which there exist analytical methods to obtain eigenvalues and eigenvectors directly.

To this point we have only used temporal finite difference schemes with an order of accuracy of 1. If we wanted higher accuracy (at least to match that of the spatial derivative), we could use the Crank-Nicolson scheme which is also implicit but has an order of accuracy of 2. As mentioned before, in this scheme the 2nd order derivative at the middle of the interval is approximated as an average of the gradients at the beginning and end of the interval. In other words, the spatial derivative has to be approximated at both t_n and t^{n+1} and then averaged in order to approximate the temporal derivative at the middle of the timestep. The finite difference scheme would thus be:

$$T_i^{n+1} = T_i^n + \frac{1}{2} k\Delta t \left[\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} + \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2} \right] ,$$

which, after rearranging and again using $\lambda \equiv \frac{k\Delta t}{(\Delta x)^2}$, becomes:

$$-\lambda T_{i-1}^{n+1} + 2\,(1+\lambda)\, T_i^{n+1} - \lambda T_{i+1}^{n+1} = \lambda T_{i-1}^n + 2\,(1-\lambda)\, T_i^n + \lambda T_{i+1}^n \,.$$

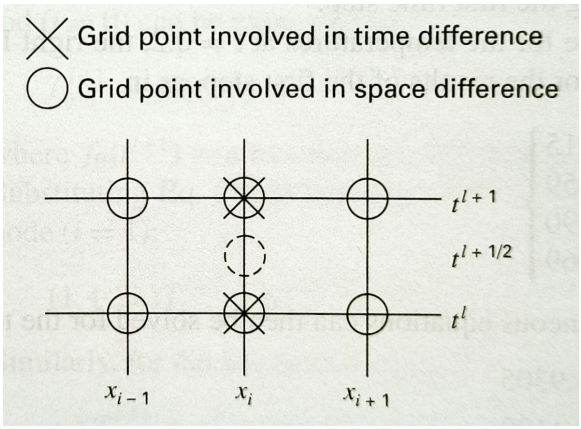


Figure 58: Sketch of grid points involved for Crank-Nicolson temporal scheme, from *Chapra & Canale*, Fig. 30.9 on pg880.

Figure 58 shows the grid points involved for the Crank-Nicolson scheme. We can see that even more neighbouring grid points are involved in the calculation, which accounts for the higher level of accuracy.

After applying the finite difference scheme to all the grid points as necessary we then

obtain a set of coupled algebraic expressions which we can handle as per usual. In matrix form, it would have the general form $\mathbf{A}\vec{T}^{n+1} = \mathbf{B}\vec{T}^{n}$.

Example

Obtain a numerical solution for Fourier's Law of Conduction from $x = 0$ to $x = 1$. The temperature $T = 75°C$ at $x = 0$ and $T = 50°C$ at $x = 1$. When $t = 0$, $T = 50°C$ for $0 < x \leq 1$. Use $k = 0.05$.

In this example we will have just 3 interior points, i.e. $x_0 = 0$, $x_4 = 1$ as the boundary points and $\Delta x = 0.25$. From the stability criteria for a convergent and stable solution, we know that the maximum timestep is $\Delta t = \frac{(\Delta x)^2}{2k} = \frac{0.25^2}{2\times 0.05} = 0.625$. Let's use $\Delta t = 0.5$ as a convenient timestep size that satisfies this criteria.

We shall use the finite difference scheme obtained by applying central difference spatially and Crank-Nicolson temporally:

$$-\lambda T_{i-1}^{n+1} + 2\,(1+\lambda)\,T_i^{n+1} - \lambda T_{i+1}^{n+1} = \lambda T_{i-1}^{n} + 2\,(1-\lambda)\,T_i^{n} + \lambda T_{i+1}^{n}\,,$$

where $\lambda \equiv \frac{k\Delta t}{(\Delta x)^2} = \frac{0.05\times 0.5}{0.25^2} = 0.4$.

At the 1st interior grid point ($i = 1$):

$$-\lambda T_0^{n+1} + 2\,(1+\lambda)\,T_1^{n+1} - \lambda T_2^{n+1} = \lambda T_0^{n} + 2\,(1-\lambda)\,T_1^{n} + \lambda T_2^{n}\,.$$

The terms T_0^n and T_0^{n+1} both refer to the leftmost boundary point, for which we are given the value. So we can just replace them with the constant T_0 (= 75):

$$-\lambda T_0 + 2\,(1+\lambda)\,T_1^{n+1} - \lambda T_2^{n+1} = \lambda T_0 + 2\,(1-\lambda)\,T_1^{n} + \lambda T_2^{n}\,.$$

At the 2nd interior grid point ($i = 2$):

$$-\lambda T_1^{n+1} + 2\,(1+\lambda)\,T_2^{n+1} - \lambda T_3^{n+1} = \lambda T_1^{n} + 2\,(1-\lambda)\,T_2^{n} + \lambda T_3^{n}\,.$$

At the 3rd interior grid point ($i = 3$):

$$-\lambda T_2^{n+1} + 2\,(1+\lambda)\,T_3^{n+1} - \lambda T_4^{n+1} = \lambda T_2^{n} + 2\,(1-\lambda)\,T_3^{n} + \lambda T_4^{n}\,.$$

The terms T_4^n and T_4^{n+1} both refer to the rightmost boundary point, for which we are given the value. So we can just replace them with the constant T_4 (= 50):

$$-\lambda T_2 + 2\,(1+\lambda)\,T_3^{n+1} - \lambda T_4 = \lambda T_2 + 2\,(1-\lambda)\,T_3^{n} + \lambda T_4\,.$$

Thus we now have a set of 3 simultaneous equations:

$$\begin{aligned} 2(1+\lambda)T_1^{n+1} - \lambda T_2^{n+1} &= 2(1-\lambda)T_1^n + \lambda T_2^n + 2\lambda T_0 \\ -\lambda T_1^{n+1} + 2(1+\lambda)T_2^{n+1} - \lambda T_3^{n+1} &= \lambda T_1^n + 2(1-\lambda)T_2^n + \lambda T_3^n \\ -\lambda T_2 + 2(1+\lambda)T_3^{n+1} &= \lambda T_2 + 2(1-\lambda)T_3^n + 2\lambda T_4 \end{aligned}$$

Alternatively if we prefer we can combine them into a matrix equation, with column vector $\vec{T} \equiv [T_1; T_2; T_3]$:

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 \\ -\lambda & 2(1+\lambda) & -\lambda \\ 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \vec{T}^{n+1} = \begin{bmatrix} 2(1-\lambda) & \lambda & 0 \\ \lambda & 2(1-\lambda) & \lambda \\ 0 & \lambda & 2(1-\lambda) \end{bmatrix} \vec{T}^n + \begin{bmatrix} 2\lambda T_0 \\ 0 \\ 2\lambda T_4 \end{bmatrix}.$$

At the moment this may look quite intimidating because there appears to be 6 unknowns (3 T's each at 2 timesteps). But remember that we are supposed to know all the information at the 'current' timestep t^n, which means that in reality there are only 3 unknowns (at timestep t^{n+1}. So in order for this to proceed, we need to begin with $n = 0$ i.e. the initial conditions which we know to be $T_i^0 = 50$ for $i = 1$,2, 3. For instance the matrix equation would be:

$$\begin{bmatrix} 2(1+\lambda) & -\lambda & 0 \\ -\lambda & 2(1+\lambda) & -\lambda \\ 0 & -\lambda & 2(1+\lambda) \end{bmatrix} \vec{T}^1 = \begin{bmatrix} 2(1-\lambda) & \lambda & 0 \\ \lambda & 2(1-\lambda) & \lambda \\ 0 & \lambda & 2(1-\lambda) \end{bmatrix} \vec{T}^0 + \begin{bmatrix} 2\lambda T_0 \\ 0 \\ 2\lambda T_4 \end{bmatrix},$$

where $\vec{T}^0 = [50; 50; 50]$. Now everything on the RHS is known so we can solve for $\vec{T}^1$ (the numerical solution at $t = 0.5$). Knowing $\vec{T}^1$, we can then solve for $\vec{T}^2$ (the numerical solution at $t = 1$), and so on.

6.3.3 Parabolic Equations in 2D

In the previous section we considered 1D parabolic equations. For 2D (and higher dimensions) we now have at least two spatial dimensions. For instance our heat conduction PDE would be:

$$\frac{\partial T}{\partial t} = k_x \frac{\partial^2 T}{\partial x^2} + k_y \frac{\partial^2 T}{\partial y^2}.$$

For a homogeneous material, $k_x = k_y = k$.

If we used an explicit temporal finite difference scheme, the stability criterion would then be (without derivation):

$$\Delta t \leq \frac{(\Delta x)^2 + (\Delta y)^2}{8k} ,$$

and if we used a grid with $\Delta x = \Delta y$, then this simplifies to

$$\Delta t \leq \frac{(\Delta x)^2}{4k} .$$

Compared to the 1D heat conduction equation, the maximum timestep here is half the size. This means that if we were to halve the grid size the maximum timestep is reduced by a factor of 4. In other words we will need 4 times as many timesteps as before, which translates to $4^2 = 16$ times the computational effort.

On the other hand if we used an implicit temporal finite difference scheme, we would need to solve $m \times n$ simultaneous equations (m grid points in x-direction; n grid points in $y-$ direction). In the matrix form, the coefficient matrix would not be tridiagonal and thus require more memory space and computational effort to solve it.

The **Alternating-Direction Implicit** (ADI) method is one of the most popular ways to deal with this issue. Basically, the calculation at each timestep is split into 2 stages:

1. Predict half a timestep ahead from t^n to $t^{n+1/2}$:
    - x-derivative(s): explicit scheme, i.e. approximated at t^n only
    - y-derivative(s): implicit scheme, i.e. use an approximation at $t^{n+1/2}$
2. Predict next half a timestep from $t^{n+1/2}$ to t^{n+1}:
    - x-derivative(s): implicit scheme, i.e. use an approximation at t^{n+1}
    - y-derivative(s): explicit scheme, i.e. approximated at $t^{n+1/2}$ only

In this manner, the number of 'unknown' terms in each equation are reduced to only 3 (assuming it is the central difference scheme is used for the spatial derivatives) instead of 6. We could then expect to obtain tridiagonal matrices later because there will only be at most 3 non-zero terms in the coefficient matrix.

For instance if we used the central difference scheme for the spatial derivatives and the explicit Euler forward/implicit Euler backward for the temporal derivative, then we would have:

$$\text{stage 1:} \quad \frac{T_{i,j}^{n+1/2} - T_{i,j}^{n}}{\Delta t/2} = k\left[\frac{T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n}}{(\Delta x)^2} + \frac{T_{i,j+1}^{n+1/2} - 2T_{i,j}^{n+1/2} + T_{i,j-1}^{n+1/2}}{(\Delta y)^2}\right]$$

$$\text{stage 2:} \quad \frac{T_{i,j}^{n+1} - T_{i,j}^{n+1/2}}{\Delta t/2} = k\left[\frac{T_{i+1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i-1,j}^{n+1}}{(\Delta x)^2} + \frac{T_{i,j+1}^{n+1/2} - 2T_{i,j}^{n+1/2} + T_{i,j-1}^{n+1/2}}{(\Delta y)^2}\right]$$

At stage 1, the unknown terms are those with the superscript $n + 1/2$; while at stage 2 the unknown terms are those with the superscript $n + 1$.

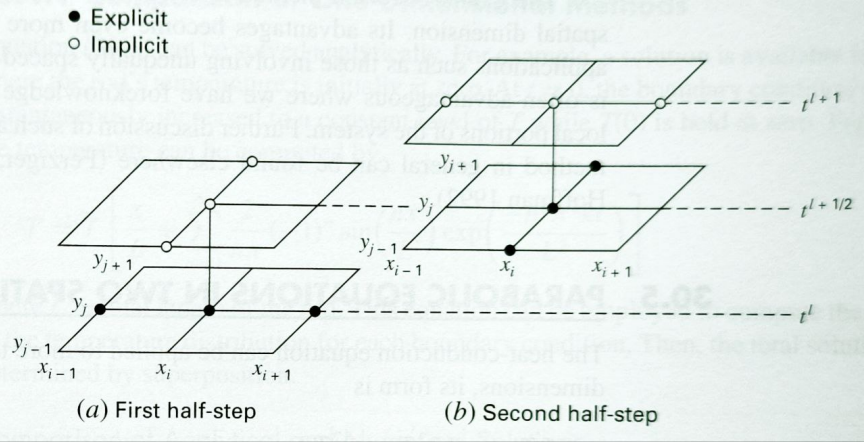


Figure 59: Sketch of grid points involved for ADI method, from *Chapra & Canale*, Fig. 30.10 on pg884.

Figure 59 illustrates the ADI method and the grid points involved in each stage. Because we have swapped between explicit and implicit schemes for both directions, any bias that was initially introduced at the first stage is partially corrected.

We can apply ADI to all of the relevant grid points to obtain the full set of simultaneous equations. But how do we obtain tridiagonal matrices? The trick is to only apply the ADI method in the direction that we are currently using an implicit scheme. In other words, during the first stage, we should group the grid points with the same x-value; in the second stage we should group those with the same y-value.

To make this even clearer: let's say we have a grid composed of 3 points in the x-direction and 4 points in the y-direction. During the first stage we would have three 4x4 tridiagonal

matrices:

- from (1,1), (1,2), (1,3) & (1,4);
- from (2,1), (2,2), (2,3) & (2,4); and
- from (3,1), (3,2), (3,3) & (3,4).

which would give us the functional values at timestep $t^{n+1/2}$. Then during the second stage we would have four 3x3 tridiagonal matrices:

- from (1,1), (2,1) & (3,1);
- from (1,2), (2,2) & (3,2); and
- from (1,3), (2,3) & (3,3);

which would give us the functional values at timestep t^{n+1}.

Example (from *Chapra & Canale*, Eg. 30.5)
Let us return to the square plate with 9 uniformly-spaced interior grid points that is sketched in Figure 53. If the plate is 40 x 40 cm, then the spatial step sizes $\Delta x = \Delta y = 10$ cm. Let the initial condition be $T = 0$ throughout, and that the boundary conditions as in the figure are instantaneously applied at time $t = 0$.

Assume the plate is made of aluminium, $k = 0.835$ cm^2/s. We shall use a timestep $\Delta t = 10$s (to make the math easier); the resulting constant $\lambda \equiv \frac{k\Delta t}{(\Delta x)^2} = 0.0835$.

We first predict the temperatures at half a timestep ahead ($t^{1/2} = 5$ s). The first tridiagonal matrix comes from $i = 1$.

$$\begin{aligned}
(1,1): \qquad -\lambda T_{1,0}^{1/2} + 2\,(1+\lambda)\,T_{1,1}^{1/2} - \lambda T_{1,2}^{1/2} &= \lambda T_{0,1}^{0} + 2\,(1-\lambda)\,T_{1,1}^{0} + \lambda T_{2,1}^{0} \\
(1,2): \qquad -\lambda T_{1,1}^{1/2} + 2\,(1+\lambda)\,T_{1,2}^{1/2} - \lambda T_{1,3}^{1/2} &= \lambda T_{0,2}^{0} + 2\,(1-\lambda)\,T_{1,2}^{0} + \lambda T_{2,2}^{0} \\
(1,3): \qquad -\lambda T_{1,2}^{1/2} + 2\,(1+\lambda)\,T_{1,3}^{1/2} - \lambda T_{1,4}^{1/2} &= \lambda T_{0,3}^{0} + 2\,(1-\lambda)\,T_{1,3}^{0} + \lambda T_{2,3}^{0}
\end{aligned}$$

Substituting in initial conditions and boundary conditions, and writing it as a matrix equa-

tion:

$$\begin{bmatrix} 2.167 & -0.0835 & 0 \\ -0.0835 & 2.167 & -0.0835 \\ 0 & -0.0835 & 2.167 \end{bmatrix} \begin{bmatrix} T_{1,1} \\ T_{1,2} \\ T_{1,3} \end{bmatrix}^{1/2} = \begin{bmatrix} 6.2625 \\ 6.2625 \\ 14.6125 \end{bmatrix},$$

which we can solve to get $T_{1,1}^{1/2} = 3.01597$, $T_{1,2}^{1/2} = 3.2708$ and $T_{1,3}^{1/2} = 6.8692$.

The second tridiagonal matrix is from $i = 2$, which allows us to obtain $T_{2,1}^{1/2} = 0.1274$, $T_{2,2}^{1/2} = 0.2900$ and $T_{2,3}^{1/2} = 4.1291$.

The third and last tridiagonal matrix is from $i = 3$, which allows us to obtain $T_{3,1}^{1/2} = 2.0181$, $T_{3,2}^{1/2} = 2.2477$ and $T_{3,3}^{1/2} = 6.0256$.

We now move to the second stage of ADI, that is to approximate the temperature at the next half a timestep ahead ($t^1 = 10$ s). This time the tridiagonal matrices will be obtained for the grid points at fixed j values. That is, we obtain the equations for the first tridiagonal matrix from $j = 1$:

$$\begin{aligned}
(1,1): \quad & -\lambda T_{0,1}^1 + 2(1+\lambda) T_{1,1}^1 - \lambda T_{2,1}^1 &=& \lambda T_{1,0}^{1/2} + 2(1-\lambda) T_{1,1}^{1/2} + \lambda T_{1,2}^{1/2} \\
(2,1): \quad & -\lambda T_{1,1}^1 + 2(1+\lambda) T_{2,1}^1 - \lambda T_{3,1}^1 &=& \lambda T_{2,0}^{1/2} + 2(1-\lambda) T_{2,1}^{1/2} + \lambda T_{2,2}^{1/2} \\
(3,1): \quad & -\lambda T_{2,1}^1 + 2(1+\lambda) T_{3,1}^1 - \lambda T_{4,1}^1 &=& \lambda T_{3,0}^{1/2} + 2(1-\lambda) T_{3,1}^{1/2} + \lambda T_{3,2}^{1/2}
\end{aligned}$$

which, after substituting in the initial and boundary conditions, becomes the matrix equation:

$$\begin{bmatrix} 2.167 & -0.0835 & 0 \\ -0.0835 & 2.167 & -0.0835 \\ 0 & -0.0835 & 2.167 \end{bmatrix} \begin{bmatrix} T_{1,1} \\ T_{2,1} \\ T_{3,1} \end{bmatrix}^{1} = \begin{bmatrix} 12.0639 \\ 0.2577 \\ 8.0619 \end{bmatrix},$$

which we can solve to get $T_{1,1}^1 = 5.5855$, $T_{2,1}^1 = 0.4782$ and $T_{3,1}^1 = 3.7388$.

The second tridiagonal matrix is from $j = 2$, which allows us to obtain $T_{1,2}^1 = 6.1683$, $T_{2,2}^1 = 0.8238$ and $T_{3,2}^1 = 4.2359$.

The third and last tridiagonal matrix is from $j = 3$, which allows us to obtain $T_{1,3}^1 = 13.1120$, $T_{2,3}^1 = 8.3207$ and $T_{3,3}^1 = 11.3606$.

We repeat this entire process for as long as we need.

6.4 Summary

Depending on the form of the differential equation, it can be treated as an integration problem (previous chapter) or solved using one-step or multi-step methods. These methods can be:

- explicit: more direct but need small intervals to avoid stability issues;
- implicit: mathematically more complex but usually unconditionally stable;
- adaptive: changing the interval size on the fly, based on the local relative error;

Adaptive, implicit and multi-step methods are good for stiff systems for different reasons.

One-step explicit Runge-Kutta methods

General form:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2 + ... + a_n k_n)\, h\,,$$

where all the a's are non-zero constants, k's are recurrence relationships ($k_2 = k_2(k_1)$, $k_3 = k_3(k_2)$ etc.) and n is the order of the Runge-Kutta method.

- explicit Euler forward ($n = 1$)

  $$y_{i+1} = y_i + f(x_i, y_i)h\,,$$

  where $f(x, y) = \frac{dy}{dx}$

- Heun's method ($n = 2$)

  $$\begin{aligned} \hat{y}_{i+1} &= y_i + f(x_i, y_i)h\,, \\ y_{i+1} &= y_i + \frac{1}{2}\left(f(x_i, y_i) + f(x_{i+1}, \hat{y}_{i+1})\right) h\,. \end{aligned}$$

- Midpoint method ($n = 2$)

  $$\begin{aligned} \hat{y}_{i+1/2} &= y_i + f(x_i, y_i)\frac{h}{2}\,, \\ y_{i+1} &= y_i + f(x_{i+1/2}, \hat{y}_{i+1/2})h\,. \end{aligned}$$

- classical 4th order Runge-Kutta ($n = 4$)

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) h\,, \\ k_1 &= f(x_i, y_i) = \frac{dy}{dx} \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \\ k_4 &= f\left(x_i + h, y_i + k_3 h\right)\,. \end{aligned}$$

One-step implicit Runge-Kutta methods

- implicit Euler backward ($n = 1$)

$$y_{i+1} = y_i + hf\left(t_{i+1}, y_{i+1}\right)\,.$$

- Crank-Nicolson method ($n = 2$)

$$y_{i+1} = y_i + \frac{1}{2}h\left[f\left(t_i, y_i\right) + f\left(t_{i+1}, y_{i+1}\right)\right]\,.$$

Multi-step methods

- Non-self-starting Heun's method ($n = 2$)

$$\begin{aligned} \hat{y}_{i+1} &= y_{i-1} + f(x_i, y_i)(2h)\,, \\ y_{i+1} &= y_i + \frac{1}{2}\left(f(x_i, y_i) + f(x_{i+1}, \hat{y}_{i+1})\right) h\,. \end{aligned}$$

- Adams-Bashforth (open; good as predictors)

$$y_{i+1} = y_i + h\sum_{k=0}^{n-1} \beta_k f_{i-k} + O(h^{n+1})\,,$$

  where β_k are listed in Table 6.

- Adams-Moulton (closed; good as correctors)

$$y_{i+1} = y_i + h\sum_{k=0}^{n-1} \beta_k f_{i+1-k} + O(h^{n+1})\,,$$

  where β_k are listed in Table 7.

Adaptive methods

The local relative error can be estimated by:

- adaptive Runge-Kutta a.k.a. step-halving method
  compare predictions from a single-step and two half-steps.
- Runge-Kutta Fehlberg a.k.a. embedded Runge-Kutta method
  compare predictions from two methods of different order. Best to use a pair with similar definitions of k.

The interval size is then modified as needed. For multi-step methods, it is recommended to change step size by a factor of 2 so that previously calculated information can be reused.

For boundary value problems, we can use either the

- **shooting method** (adjust the guess of inital condition to get the correct boundary value outcome) or
- **finite-difference methods** (replace the derivative with a finite difference scheme and then substitute in the boundary conditions directly).

For partial differential equations, we have slightly different methods depending on the type of equation. However for all of them we will use some finite difference method that is applied to all the interior grid points, resulting in a set of algebraic expressions.

- elliptic equations: boundary value problems; steady-state
  - Shooting method (described above)
  - Finite-difference methods
    Choose FD schemes for each derivative and after obtaining the set of algebraic expressions, use Liebmann's method to solve.
- parabolic & hyperbolic equations: time-varying problems; transient
  The spatial derivatives typically use a central difference scheme because of the higher accuracy. The temporal derivatives use:

- explicit methods

  Can solve directly for function at a particular location. Need to satisfy a stability criterion. May be computationally expensive. Not all available information is used which results in lower accuracy.

- implicit methods

  Need to solve a set of simultaneous equations. Typically unconditionally stable, but for accuracy the timestep is usually still small. Recommended methods are **Crank-Nicolson** (for 1D) and **Alternating-Direction Implicit** method (for 2D).

Note that for the ADI, tridiagonal matrices can be obtained by applying to the interior points in the same direction as the implicit scheme is applied. This are opposite directions in each of the 2 stages of ADI.