1 Ordinary Differential Equations

1.0 Mathematical Background

1.0.1 Smoothness

Definition 1.1 *A function f defined on $[a, b]$ is* continuous at $x_0 \in [a, b]$ *if* $\lim\limits_{x \to x_0} f(x) = f(x_0)$.

Remark Note that this implies existence of the quantities on both sides of the equation.

f is *continuous on* $[a, b]$, i.e., $f \in C[a, b]$, if f is continuous at every $x \in [a, b]$.
If $f^{(n)}$ is continuous on $[a, b]$, then $f \in C^{(n)}[a, b]$.
Alternatively, one could start with the following ε-δ definition:

Definition 1.2 *A function f defined on $[a, b]$ is continuous at $x_0 \in [a, b]$ if for every $\varepsilon > 0$ there exists a $\delta_\varepsilon > 0$ (that depends on x_0) such that $|f(x) - f(x_0)| < \varepsilon$ whenever $|x - x_0| < \delta_\varepsilon$.*

FIGURE

Example (A function that is continuous, but not uniformly continuous) $f(x) = \frac{1}{x}$ with FIGURE.

Definition 1.3 *A function f is* uniformly continuous on $[a, b]$ *if it is continuous with a uniform δ_ε for all $x_0 \in [a, b]$, i.e., independent of x_0.*

Important for ordinary differential equations is

Definition 1.4 *A function f defined on $[a, b]$ is* Lipschitz continuous on $[a, b]$ *if there exists a number λ such that $|f(x) - f(y)| \le \lambda |x - y|$ for all $x, y \in [a, b]$. λ is called the* Lipschitz constant.

Remark

1. In fact, any Lipschitz continuous function is *uniformly* continuous, and therefore continuous.

2. For a differentiable function with bounded derivative we can take

$$\lambda = \max_{x \in [a,b]} |f'(x)|,$$

and we see that Lipschitz continuity is "between" continuity and differentiability.

3. If the function f is Lipschitz continuous on $[a, b]$ with Lipschitz constant λ, then f is almost everywhere differentiable in $[a, b]$ with $|f'(x)| \le \lambda$. In other words, Lipschitz continuous functions need not be differentiable everywhere in $[a, b]$.

4. See also Assignment 1.

1.0.2 Polynomials

A real *polynomial of degree at most* n is of the form

$$p(x) = \sum_{j=0}^{n} a_j x^j, \qquad x \in \mathbb{R},$$

with coefficients $a_j \in \mathbb{R}$. Notation: We denote the space of (univariate) polynomials of degree at most m by $p \in \mathbb{P}_n$.

Theorem 1.5 *(Taylor's Theorem) If $f \in C^n[a,b]$ and $f^{(n+1)}$ exists on (a,b), then for any $x \in [a,b]$*

$$f(x) = \underbrace{\sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x_0)(x-x_0)^k}_{\textit{Taylor polynomial}} + \underbrace{\frac{1}{(n+1)!} f^{(n+1)}(\xi_x)(x-x_0)^{n+1}}_{\textit{error term}},$$

where ξ_x is some point between x and x_0.

FIGURE

An alternate form commonly used is

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

Remark Note that the information about f is provided locally, at x_0 only. If the information is spread out, i.e., when we are given distinct points $x_0 < x_1 < \ldots < x_n \in [a,b]$ and associated values $f(x_0), f(x_1), \ldots, f(x_n)$, then we will see below that there exists a *unique interpolation polynomial* p_n of degree at most n

$$p_n(x) = \sum_{j=0}^{n} \ell_j(x) f(x_j)$$

with

$$\ell_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}, \qquad j = 0, 1, \ldots, n,$$

such that $p_n(x_i) = f(x_i)$, $i = 0, 1, \ldots, n$. The ℓ_j, $j = 0, 1, \ldots, n$, are called *Lagrange functions*, and the polynomial p_n is said to be in *Lagrange form*.

Remark See Assignment 1 for an illustration of Taylor polynomials vs. interpolation polynomials.

1.1 Polynomial Interpolation

In the following two subsections we will discuss the problem of fitting data given in the form of discrete points (e.g., physical measurements, output from a differential equations solver, design points for CAD, etc.) with an appropriate function s taken from some (finite-dimensional) function space $\mathcal{S}$ of our choice. Throughout most of our discussion the data will be *univariate*, i.e., of the form (x_i, y_i), $i = 0, \ldots, n$, where the $x_i \in \mathbb{R}$ are referred to as *data sites* (or *nodes*), and the $y_i \in \mathbb{R}$ as *values* (or *ordinates*). Often we will assume $y_i = f(x_i)$ for some (unknown) function f. Later on we will also briefly consider *multivariate* data where $\boldsymbol{x}_i \in \mathbb{R}^d$, $d > 1$.

The data can be fitted either by *interpolation*, i.e., by satisfying

$$s(x_i) = y_i, \qquad i = 0, \ldots, n, \tag{1}$$

or by *approximation*, i.e., by satisfying

$$\|\boldsymbol{s} - \boldsymbol{y}\| < \epsilon,$$

where $\boldsymbol{s}$ and $\boldsymbol{y}$ have to be considered as vectors of function or data values, and $\|\cdot\|$ is some discrete norm on $\mathbb{R}^{n+1}$.

We will focus our attention on interpolation.

If $\{b_0, \ldots, b_n\}$ is some basis of the function space $\mathcal{S}$, then we can express s as a linear combination in the form

$$s(x) = \sum_{j=0}^{n} a_j b_j(x).$$

If we are given $m+1$ data points (x_i, y_i), $i = 0, 1, \ldots, m$, then the interpolation conditions (1) lead to

$$\sum_{j=0}^{n} a_j b_j(x_i) = y_i, \qquad i = 0, \ldots, m.$$

This represents a system of linear algebraic equations for the expansion coefficients a_j of the form

$$B\boldsymbol{a} = \boldsymbol{y},$$

where the $m \times n$ system matrix B has entries $b_j(x_i)$. We are especially interested in those function spaces and bases for which the case $m = n$ yields a unique solution.

Function spaces studied in this class include polynomials, piecewise polynomials, trigonometric polynomials, and radial basis functions.

We will begin by studying polynomials. There are several motivating factors for doing this:

- Everyone is familiar with polynomials.
- Polynomials can be easily and efficiently evaluated using Horner's algorithm (i.e., nested multiplication).

- We may have heard of the Weierstrass Approximation Theorem which states that any continuous function can be approximated arbitrarily closely by a polynomial (of sufficiently high degree).
- A lot is known about polynomial interpolation, and serves as starting point for other methods.

Formally, we are interested in solving the following

Problem 1.6 *Given data (x_i, y_i), $i = 0, \ldots, n$, find a polynomial p of minimal degree which matches the data in the sense of (1), i.e., for which*

$$p(x_i) = y_i, \qquad i = 0, \ldots, n.$$

We illustrate this problem with some numerical examples in the Maple worksheet `478578_PolynomialInterpolation.mws`. The numerical experiments suggest that $n+1$ data points can be interpolated by a polynomial of degree n. Indeed,

Theorem 1.7 *Let $n+1$ distinct real numbers $x_0, x_1, \ldots, x_n$ and associated values $y_0, y_1, \ldots, y_n$ be given. Then there exists a unique polynomial p_n of degree at most n such that*

$$p_n(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

Proof Assume

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n.$$

Then the interpolation conditions (1) lead to the linear system $B\boldsymbol{a} = \boldsymbol{y}$ with

$$B = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix}, \quad \boldsymbol{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \text{ and } \boldsymbol{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \ldots \\ y_n \end{bmatrix}.$$

For general data $\boldsymbol{y}$ this is a nonhomogeneous system, and we know that such a system has a unique solution if and only if the associated homogeneous system has only the trivial solution. The homogeneous system corresponds to

$$p_n(x_i) = 0, \qquad i = 0, 1, \ldots, n.$$

Thus, p_n has $n+1$ zeros. Now, the Fundamental Theorem of Algebra states that any nontrivial polynomial of degree n has n (possibly complex) zeros. Therefore p_n must be the zero polynomial, i.e., the homogeneous linear system has only the trivial solution $a_0 = a_1 = \ldots = a_n = 0$, and by the above comment the general nonhomogeneous problem has a unique solution. ∎

Remark The matrix B in the proof of Theorem 1.7 is referred to as a *Vandermonde matrix.*

Next, we illustrate how such a (low-degree) interpolating polynomial can be determined.

Example Let's assume we have the following data:

x	0	1	3
y	1	0	4

.

Now, since we want a square linear system, we pick an approximation space of dimension three, i.e., with three basis functions. This means we use quadratic polynomials, and as basis we take the monomials $b_0(x) = 1$, $b_1(x) = x$, and $b_2(x) = x^2$. Therefore, the interpolating polynomial will be of the form

$$p_2(x) = a_0 + a_1 x + a_2 x^2.$$

In order to determine the coefficients a_0, a_1, and a_2 we enforce the interpolation conditions (1), i.e., $p_2(x_i) = y_i$, $i = 0, 1, 2$. This leads to the linear system

$$\begin{array}{lccccccc} (p_2(0) =) & a_0 & & & & & = & 1 \\ (p_2(1) =) & a_0 & + & a_1 & + & a_2 & = & 0 \\ (p_2(3) =) & a_0 & + & 3a_1 & + & 9a_2 & = & 4 \end{array}$$

whose solution is easily verified to be $a_0 = 1$, $a_1 = -2$, and $a_2 = 1$. Thus,

$$p_2(x) = 1 - 2x + x^2.$$

Of course, this polynomial can also be written as

$$p_2(x) = (1 - x)^2.$$

So, had we chosen the basis of *shifted monomials* $b_0(x) = 1$, $b_1(x) = 1 - x$, and $b_2(x) = (1 - x)^2$, then the coefficients (for an expansion with respect to this basis) would have come out to be $a_0 = 0$, $a_1 = 0$, and $a_2 = 1$.

In general, use of the monomial basis leads to a Vandermonde system as listed in the proof of the theorem above. This is a classical example of an ill-conditioned system, and thus should be avoided. We will look at other bases later.

We now provide a second (constructive) proof of Theorem 1.7.

Constructive Proof: First we establish uniqueness. To this end we assume that p_n and q_n both are n-th degree interpolating polynomials. Then

$$r_n(x) = p_n(x) - q_n(x)$$

is also an n-th degree polynomial. Moreover, by the Fundamental Theorem of Algebra r_n has n zeros (or is the zero polynomial). However, by the nature of p_n and q_n we have

$$r_n(x_i) = p_n(x_i) - q_n(x_i) = y_i - y_i = 0, \qquad i = 0, 1, \ldots, n.$$

Thus, r_n has $n + 1$ zeros, and therefore must be identically equal to zero. This ensures uniqueness.

Existence is constructed by induction. For $n = 0$ we take $p_0 \equiv y_0$. Obviously, the degree of p_0 is less than or equal to 0, and also $p_0(x_0) = y_0$.

Now we assume p_{k-1} to be the unique polynomial of degree at most $k-1$ that interpolates the data (x_i, y_i), $i = 0, 1, \ldots, k-1$. We will construct p_k (of degree k) such that

$$p_k(x_i) = y_i, \qquad i = 0, 1, \ldots, k.$$

We let

$$p_k(x) = p_{k-1}(x) + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1})$$

with c_k yet to be determined. By construction, p_k is a polynomial of degree k which interpolates the data (x_i, y_i), $i = 0, 1, \ldots, k-1$.

We now determine c_k so that we also have $p_k(x_k) = y_k$. Thus,

$$(p_k(x_k) =) \quad p_{k-1}(x_k) + c_k(x_k - x_0)(x_k - x_1) \ldots (x_k - x_{k-1}) = y_k$$

or

$$c_k = \frac{y_k - p_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1) \ldots (x_k - x_{k-1})}.$$

This is well defined since the denominator is nonzero due to the fact that we assume distinct data sites. ∎

The construction used in this alternate proof provides the starting point for the *Newton form* of the interpolating polynomial.

From the proof we have

$$\begin{aligned} p_k(x) &= p_{k-1}(x) + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1}) \\ &= p_{k-2}(x) + c_{k-1}(x - x_0)(x - x_1) \ldots (x - x_{k-2}) + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1}) \\ &\vdots \\ &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1}). \end{aligned}$$

Thus, the Newton form of the interpolating polynomial is given by

$$p_n(x) = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (x - x_i). \tag{2}$$

This notation implies that the empty product (when $j - 1 < 0$) is equal to 1.

The proof above also provides a formula for the coefficients in the Newton form:

$$c_j = \frac{y_j - p_{j-1}(x_j)}{(x_j - x_0)(x_j - x_1) \ldots (x_j - x_{j-1})}, \qquad p_0 \equiv c_0 = y_0. \tag{3}$$

So the Newton coefficients can be computed recursively. This leads to a first algorithm for the solution of the interpolation problem.

Algorithm

```
Input $n$, $x_0, x_1, \ldots, x_n$, $y_0, y_1, \ldots, y_n$

$c_0 = y_0$

for $k = 1 : n$
```

```
        $d = x_k - x_{k-1}$
        $u = c_{k-1}$
        for $i = k-2 : -1 : 0$        % build $p_{k-1}$
            $u = u(x_k - x_i) + c_i$        % Horner
            $d = d(x_k - x_i)$        % accumulate denominator
        end
        $c_k = \frac{y_k - u}{d}$
    end
    Output $c_0, c_1, \ldots, c_n$
```

Remark A more detailed derivation of this algorithm is provided on page 310 of the Kincaid/Cheney book. However, the standard, more efficient, algorithm for computing the coefficients of the Newton form is based on the use of divided differences and will not be discussed here.

Example We now compute the Newton form of the polynomial interpolating the data

x	0	1	3
y	1	0	4

.

According to (2) and (3) we have

$$p_2(x) = \sum_{j=0}^{2} c_j \prod_{i=0}^{j-1} (x - x_i) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1)$$

with

$$c_0 = y_0 = 1, \text{ and } c_j = \frac{y_j - p_{j-1}(x_j)}{(x_j - x_0)(x_j - x_1)\ldots(x_j - x_{j-1})}, \qquad j = 1, 2.$$

Thus, we are representing the space of quadratic polynomials with the basis $b_0(x) = 1$, $b_1(x) = x - x_0 = x$, and $b_2(x) = (x - x_0)(x - x_1) = x(x - 1)$.

We now determine the two remaining coefficients. First,

$$c_1 = \frac{y_1 - p_0(x_1)}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} = \frac{0 - 1}{1 - 0} = -1.$$

This gives us

$$p_1(x) = c_0 + c_1(x - x_0) = 1 - x.$$

Next,

$$c_2 = \frac{y_2 - p_1(x_2)}{(x_2 - x_0)(x_2 - x_1)} = \frac{4 - (1 - x_2)}{3 \cdot 2} = 1,$$

and so

$$p_2(x) = p_1(x) + c_2(x - x_0)(x - x_1) = 1 - x + x(x - 1).$$

A third representation (after the monomial and Newton forms) of the same (unique) interpolating polynomial is of the general form

$$p_n(x) = \sum_{j=0}^{n} y_j \ell_j(x). \tag{4}$$

Note that the coefficients here are the data values, and the polynomial basis functions ℓ_j are so-called *cardinal* (or *Lagrange*) *functions*. We want these functions to depend on the data sites $x_0, x_1, \dots, x_n$, but not the data values $y_0, y_1, \dots, y_n$. In order to satisfy the interpolation conditions (1) we require

$$p_n(x_i) = \sum_{j=0}^{n} y_j \ell_j(x_i) = y_i, \qquad i = 0, 1, \dots, n.$$

Clearly, this is ensured if

$$\ell_j(x_i) = \delta_{ij}, \tag{5}$$

which is called a *cardinality* (or *Lagrange*) *condition*.

How do we determine the Lagrange functions ℓ_j?

We want them to be polynomials of degree n and satisfy the cardinality conditions (5). Let's fix j, and assume

$$\begin{aligned} \ell_j(x) &= c(x-x_0)(x-x_1)\dots(x-x_{j-1})(x-x_{j+1})\dots(x-x_n) \\ &= c\prod_{\substack{i=0\\ i\neq j}}^{n}(x-x_i). \end{aligned}$$

Clearly, $\ell_j(x_i) = 0$ for $j \neq i$. Also, ℓ_j depends only on the data sites and is a polynomial of degree n. The last requirement is $\ell_j(x_j) = 1$. Thus,

$$1 = c\prod_{\substack{i=0\\ i\neq j}}^{n}(x_j - x_i)$$

or

$$c = \frac{1}{\prod_{\substack{i=0\\ i\neq j}}^{n}(x_j - x_i)}.$$

Again, the denominator is nonzero since the data sites are assumed to be distinct. Therefore, the Lagrange functions are given by

$$\ell_j(x) = \prod_{\substack{i=0\\ i\neq j}}^{n} \frac{x - x_i}{x_j - x_i}, \qquad j = 0, 1, \dots, n,$$

and the Lagrange form of the interpolating polynomial is

$$p_n(x) = \sum_{j=0}^{n} y_j \prod_{\substack{i=0\\ i\neq j}}^{n} \frac{x - x_i}{x_j - x_i}. \tag{6}$$

Remark

1. We mentioned above that the monomial basis results in an interpolation matrix (a Vandermonde matrix) that is notoriously ill-conditioned. However, an advantage of the monomial basis representation is the fact that the interpolant can be evaluated efficiently using Horner's method.

2. The interpolation matrix for the Lagrange form is the identity matrix and the coefficients in the basis expansion are given by the data values. This makes the Lagrange form ideal for situations in which many experiments with the same data sites, but different data values need to be performed. However, evaluation (as well as differentiation or integration) is more expensive.

3. A major advantage of the Newton form is its efficiency in the case of adaptive interpolation, i.e., when an existing interpolant needs to be refined by adding more data. Due to the recursive nature, the new interpolant can be determined by updating the existing one (as we did in the example above). If we were to form the interpolation matrix for the Newton form we would get a triangular matrix. Therefore, for the Newton form we have a balance between stability of computation and ease of evaluation.

Example Returning to our earlier example, we now compute the Lagrange form of the polynomial interpolating the data

x	0	1	3
y	1	0	4

.

According to (4) we have

$$p_2(x) = \sum_{j=0}^{2} y_j \ell_j(x) = \ell_0(x) + 4\ell_2(x),$$

where (see (6))

$$\ell_j(x) = \prod_{\substack{i=0\\ i\neq j}}^{2} \frac{x - x_i}{x_j - x_i}.$$

Thus,

$$\ell_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-1)(x-3)}{(-1)(-3)} = \frac{1}{3}(x-1)(x-3),$$

and

$$\ell_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{x(x-1)}{(3-0)(3-1)} = \frac{1}{6}x(x-1).$$

This gives us

$$p_2(x) = \frac{1}{3}(x-1)(x-3) + \frac{2}{3}x(x-1).$$

Note that here we have represented the space of quadratic polynomials with the (cardinal) basis $b_0(x) = \frac{1}{3}(x-1)(x-3)$, $b_2(x) = \frac{1}{6}x(x-1)$, and $b_1(x) = \ell_1(x) = -\frac{1}{2}x(x-3)$ (which we did not need to compute for this example since its coefficient $y_1 = 0$).

Summarizing our example, we have found four different forms of the quadratic polynomial interpolating our data:

monomial	$p_2(x) = 1 - 2x + x^2$
shifted monomial	$p_2(x) = (1 - x)^2$
Newton form	$p_2(x) = 1 - x + x(x - 1)$
Lagrange form	$p_2(x) = \frac{1}{3}(x-1)(x-3) + \frac{2}{3}x(x-1)$

The basis polynomials for these four cases are displayed in Figures 1 and 2.

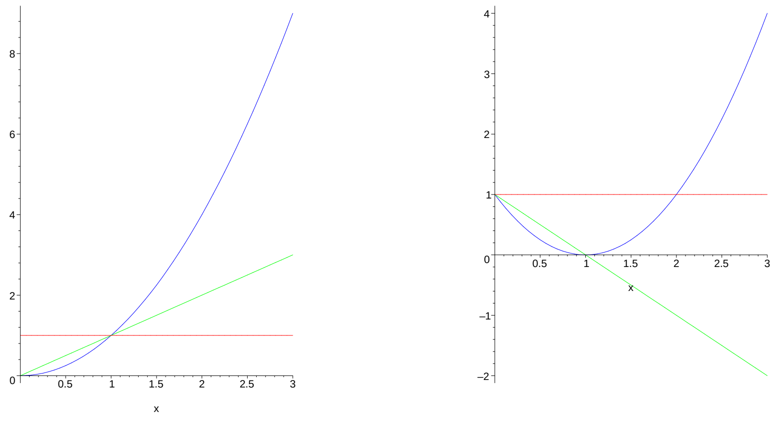

Figure 1: Monomial (left) and shifted monomial (right) bases for the interpolation example.

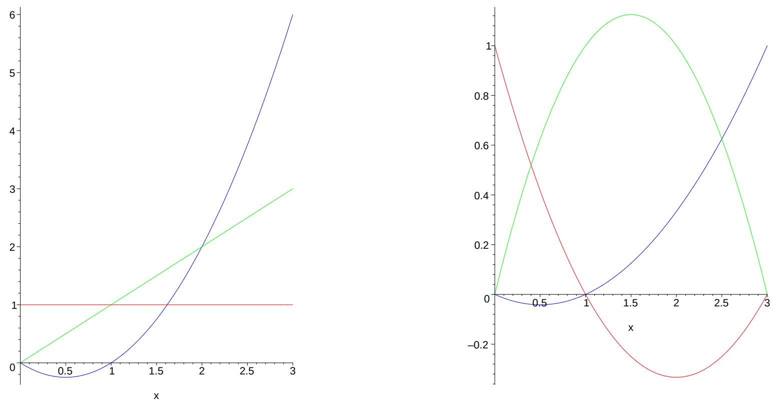

Figure 2: Basis polynomials for the Newton (left) and Lagrange form (right).

1.1.1 Error in Polynomial Interpolation

In the introduction we mentioned the Weierstrass Approximation Theorem as one of the motivations for the use of polynomials. Here are the details.

Theorem 1.8 *Let $f \in C[a, b]$ and $\epsilon > 0$. Then there exists a polynomial p of sufficiently high degree such that*

$$|f(x) - p(x)| < \epsilon$$

for all $x \in [a, b]$.

We will not prove this theorem. A proof can, e.g., be found in the book by Kincaid and Cheney. Now we only point out that — as nice as this theorem is — it does not cover interpolation of (values of) f by p. For this problem we have

Theorem 1.9 *Let $f \in C^{n+1}[a,b]$ and p be a polynomial of degree at most n which interpolates f at the $n+1$ distinct points $x_0, x_1, \ldots, x_n$ in $[a,b]$ (i.e., $p(x_i) = f(x_i)$, $i = 0, 1, \ldots, n$). Then, for each $x \in [a,b]$ there exists a number $\xi_x \in (a,b)$ such that*

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{i=0}^{n} (x - x_i). \tag{7}$$

In order to prove this result we need to recall Rolle's Theorem:

Theorem 1.10 *If $f \in C[a,b]$ and f' exists on (a,b), and if $f(a) = f(b) = 0$, then there exists a number $\xi \in (a,b)$ such that $f'(\xi) = 0$.*

Proof (of Theorem 1.9) If x coincides with one of the data sites x_i, $i = 0, 1, \ldots, n$, then it is easy to see that both sides of equation (7) are zero.

Thus we now assume $x \neq x_i$ to be fixed. We start be defining

$$w(t) = \prod_{i=0}^{n} (t - x_i)$$

and

$$F = f - p - \alpha w$$

with α such that $F(x) = 0$, i.e.,

$$\alpha = \frac{f(x) - p(x)}{w(x)}.$$

We need to show that $\alpha = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x)$ for some $\xi_x \in (a,b)$.

Since $f \in C^{n+1}[a,b]$ we know that $F \in C^{n+1}[a,b]$ also. Moreover,

$$F(t) = 0 \quad \text{for} \quad t = x, x_0, x_1, \ldots, x_n.$$

The first of these equations holds by the definition of α, the remainder by the definition of w and the fact that p interpolates f at these points.

Now we apply Rolle's Theorem to F on each of the $n+1$ subintervals generated by the $n+2$ points $x, x_0, x_1, \ldots, x_n$. Thus, F' has (at least) $n+1$ distinct zeros in (a,b).

Next, by Rolle's Theorem (applied to F' on n subintervals) we know that F'' has (at least) n zeros in (a,b).

Continuing this argument we deduce that $F^{(n+1)}$ has (at least) one zero, ξ_x, in (a,b).

On the other hand, since

$$F(t) = f(t) - p(t) - \alpha w(t)$$

we have

$$F^{(n+1)}(t) = f^{(n+1)}(t) - p^{(n+1)}(t) - \alpha w^{(n+1)}(t).$$

However, p is a polynomial of degree at most n, so $p^{(n+1)} \equiv 0$. Since the leading coefficient of the $(n+1)$-degree polynomial w is 1 we have

$$w^{(n+1)}(t) = \frac{d^{n+1}}{dt^{n+1}} \prod_{i=0}^{n} (t - x_i) = (n+1)!.$$

Therefore,

$$F^{(n+1)}(t) = f^{(n+1)}(t) - \alpha(n+1)!.$$

Combining this with the information about the zero of $F^{(n+1)}$ we have

$$\begin{aligned} 0 = F^{(n+1)}(\xi_x) &= f^{(n+1)}(\xi_x) - \alpha(n+1)! \\ &= f^{(n+1)}(\xi_x) - \frac{f(x) - p(x)}{w(x)}(n+1)! \end{aligned}$$

or

$$f(x) - p(x) = f^{(n+1)}(\xi_x) \frac{w(x)}{(n+1)!}.$$

∎

Remark 1. The error formula (7) in Theorem 1.9 looks almost like the error formula for Taylor polynomials from Theorem 1.5:

$$f(x) - T_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)^{n+1}.$$

The difference lies in the fact that for Taylor polynomials the information is concentrated at one point x_0, whereas for interpolation the information is obtained at the points $x_0, x_1, \ldots, x_n$.

2. The error formula (7) will be used later to derive (and judge the accuracy of) methods for solving differential equations as well as for numerical integration.

If the data sites are equally spaced, i.e., $\Delta x_i = x_i - x_{i-1} = h$, $i = 1, 2, \ldots, n$, it is possible to derive the bound

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{1}{4} h^{n+1} n!$$

in the error formula (7). If we also assume that the derivatives of f are bounded, i.e., $|f^{(n+1)}(x)| \leq M$ for all $x \in [a, b]$, then we get

$$|f(x) - p(x)| \leq \frac{M}{4(n+1)} h^{n+1}.$$

Thus, the error for interpolation with degree-n polynomials is $\mathcal{O}(h^{n+1})$. We illustrate this formula for a fixed-degree polynomial in the Maple worksheet `478578_PolynomialInterpolationError.mws`.

Formula (7) tells us that the interpolation error depends on the function we're interpolating as well as the data sites (interpolation nodes) we use. If we are interested in minimizing the interpolation error, then we need to choose the data sites x_i, $i = 0, 1, \ldots, n$, such that

$$w(t) = \prod_{i=0}^{n}(t - x_i)$$

is minimized. Figure 3 shows the graph of the function $|w|$ for 10 equally spaced (red,dashed) and 10 optimally spaced (green,solid) data sites on the interval $[-1, 1]$. We will derive the locations of the optimally spaced points (known as *Chebyshev points*) below.

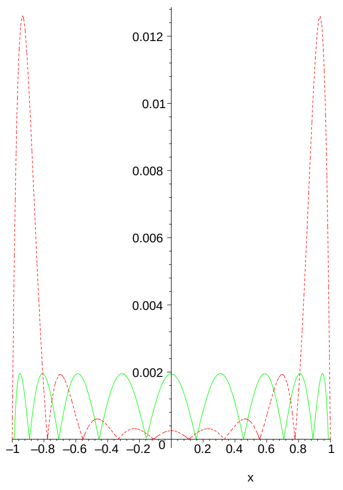

Figure 3: Graph of the function $|w|$ for 10 equally spaced (red,dashed) and 10 Chebyshev points (green,solid) on $[-1, 1]$.

Figure 3 shows that the error near the endpoints of the interval can be significant if we insist on using equally spaced points. A classical example that illustrates this phenomenon is the Runge function

$$f(x) = \frac{1}{1 + 25x^2}.$$

We present this example in the Maple worksheet `478578_Runge.mws`.

In order to discuss Chebyshev points we need to introduce a certain family of orthogonal polynomials called *Chebyshev polynomials*.

1.1.2 Chebyshev Polynomials

The Chebyshev polynomials (of the first kind) can be defined recursively. We have

$$T_0(x) = 1, \qquad T_1(x) = x,$$

and

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n \geq 1. \tag{8}$$

An explicit formula for the n-th degree Chebyshev polynomial is

$$T_n(x) = \cos(n \arccos x), \quad x \in [-1, 1], \quad n = 0, 1, 2, \ldots. \tag{9}$$

We can verify this explicit formula by using the trigonometric identity

$$\cos(A+B) = \cos A \cos B - \sin A \sin B.$$

This identity gives rise to the following two formulas:

$$\begin{aligned}\cos(n+1)\theta &= \cos n\theta \cos\theta - \sin n\theta \sin\theta \\ \cos(n-1)\theta &= \cos n\theta \cos\theta + \sin n\theta \sin\theta.\end{aligned}$$

Addition of these two formulas yields

$$\cos(n+1)\theta + \cos(n-1)\theta = 2\cos n\theta \cos\theta. \tag{10}$$

Now, if we let $x = \cos\theta$ (or $\theta = \arccos x$) then (10) becomes

$$\cos\left[(n+1)\arccos x\right] + \cos\left[(n-1)\arccos x\right] = 2\cos\left(n\arccos x\right)\cos\left(\arccos x\right)$$

or

$$\cos\left[(n+1)\arccos x\right] = 2x\cos\left(n\arccos x\right) - \cos\left[(n-1)\arccos x\right],$$

which is of the same form as the recursion (8) for the Chebyshev polynomials provided we identify T_n with the explicit formula (9).

Some properties of Chebyshev polynomials are

1. $|T_n(x)| \le 1, \quad x \in [-1,1]$.
2. $T_n\left(\cos\frac{i\pi}{n}\right) = (-1)^i, \quad i = 0,1,\ldots,n$. These are the extrema of T_n.
3. $T_n\left(\cos\frac{2i-1}{2n}\pi\right) = 0, \quad i = 1,\ldots,n$. This gives the zeros of T_n.
4. The leading coefficient of T_n, $n = 1,2,\ldots$, is 2^{n-1}, i.e.,

$$T_n(x) = 2^{n-1}x^n + \text{lower order terms}.$$

Items 1–3 follow immediately from (9). Item 4 is clear from the recursion formula (8). Therefore, $2^{1-n}T_n$ is a *monic* polynomial, i.e., its leading coefficient is 1. We will need the following property of monic polynomials below.

Theorem 1.11 *For any monic polynomial p of degree n on $[-1,1]$ we have*

$$\|p\|_\infty = \max_{-1\le x\le 1}|p(x)| \ge 2^{1-n}.$$

Proof A proof can be found on p. 317 of the Kincaid and Cheney book.

We are now ready to return to the formula for the interpolation error. From (7) we get on $[-1,1]$

$$\|f-p\|_\infty = \max_{-1\le x\le 1}|f(x)-p(x)|$$

$$\leq \frac{1}{(n+1)!} \max_{-1\leq x\leq 1} \left|f^{(n+1)}(x)\right| \max_{-1\leq x\leq 1} \Big|\underbrace{\prod_{i=0}^{n}(x-x_i)}_{=w(x)}\Big|.$$

We note that w is a monic polynomial of degree $n+1$ with zeros x_i, $i = 0, 1, \ldots, n$, and therefore

$$\max_{-1\leq x\leq 1} |w(x)| \geq 2^{-n}.$$

From above we know that $2^{-n}T_{n+1}$ is also a monic polynomial of degree $n+1$ with extrema $T_{n+1}\left(\cos\frac{i\pi}{n+1}\right) = \frac{(-1)^i}{2^n}$, $i = 0, 1, \ldots, n+1$. By Theorem 1.11 the minimal value of $\max_{-1\leq x\leq 1} |w(x)|$ is 2^{-n}. We just observed that this value is attained for T_{n+1}.

Thus, the zeros x_i of the optimal w should coincide with the zeros of T_{n+1}, or

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \qquad i = 0, 1, \ldots, n. \tag{11}$$

These are the Chebyshev points used in Figure 3 and in the Maple worksheet `478578_Runge.mws`.

If the data sites are taken to be the Chebyshev points on $[-1, 1]$, then the interpolation error (7) becomes

$$|f(x) - p(x)| \leq \frac{1}{2^n(n+1)!} \max_{-1\leq t\leq 1} \left|f^{(n+1)}(t)\right|, \qquad |x| \leq 1.$$

Remark The Chebyshev points (11) are for interpolation on $[-1, 1]$. If a different interval is used, the Chebyshev points need to be transformed accordingly.

We just observed that the Chebyshev nodes are "optimal" interpolation points in the sense that for any given f and fixed degree n of the interpolating polynomial, if we are free to choose the data sites, then the Chebyshev points will yield the most accurate approximation (measured in the maximum norm).

For equally spaced interpolation points, our numerical examples in the Maple worksheets `478578_PolynomialInterpolation.mws` and `478578_Runge.mws` have shown that, contrary to our intuition, the interpolation error (measured in the maximum norm) does not tend to zero as we increase the number of interpolation points (or polynomial degree).

The situation is even worse. There is also the following more general result proved by Faber in 1914.

Theorem 1.12 *For any fixed system of interpolation nodes $a \leq x_0^{(n)} < x_1^{(n)} < \ldots < x_n^{(n)} \leq b$ there exists a function $f \in C[a, b]$ such that the interpolating polynomial p_n does not uniformly converge to f, i.e.,*

$$\|f - p_n\|_\infty \not\to 0, \quad n \to \infty.$$

This, however, needs to be contrasted with the positive result (very much in the spirit of the Weierstrass Approximation Theorem) for the situation in which we are free to choose the location of the interpolation points.

Theorem 1.13 *Let $f \in C[a,b]$. Then there exists a system of interpolation nodes such that*

$$\|f - p_n\|_\infty \to 0, \quad n \to \infty.$$

Proof The proof uses the Weierstrass Approximation Theorem as well as the Chebyshev Alternation Theorem. (cf. Kincaid and Cheney).

Finally, if we insist on using the Chebyshev points as data sites, then we have the following theorem due to Fejér.

Theorem 1.14 *Let $f \in C[-1,1]$, and $x_0, x_1, \ldots, x_{n-1}$ be the first n Chebyshev points. Then there exists a polynomial p_{2n-1} of degree $2n-1$ that interpolates f at $x_0, x_1, \ldots, x_{n-1}$, and for which*

$$\|f - p_{2n-1}\|_\infty \to 0, \quad n \to \infty.$$

Remark The polynomial p_{2n-1} also has zero derivatives at x_i, $i = 0, 1, \ldots, n-1$.

1.2 Spline Interpolation

One of the main disadvantages associated with polynomial interpolation were the oscillations resulting from the use of high-degree polynomials. If we want to maintain such advantages as simplicity, ease and speed of evaluation, as well as similar approximation properties, we are naturally led to consider *piecewise polynomial interpolation* or *spline interpolation.*

Definition 1.15 *A* spline function S of degree k *is a function such that*

a) S is defined on an interval $[a,b]$,

b) $S \in C^{k-1}[a,b]$,

c) there are points $a = t_0 < t_1 < \ldots < t_n = b$ (called knots) such that S is a polynomial of degree at most k on each subinterval $[t_i, t_{i+1})$.

Example The most commonly used spline function is the piecewise linear ($k = 1$) spline, i.e., given a knot sequence as in Definition 1.15, S is a linear function on each subinterval with continuous joints at the knots.

If we are given the data

x	0	1	2	3
y	1	0	-1	3

,

then we can let the knot sequence coincide with the data sites, i.e.,

$$t_i = i, \qquad i = 0, 1, 2, 3.$$

The piecewise linear spline interpolating the data is then given by the "connect-the-dots" approach, or in formulas

$$S(x) = \begin{cases} 1 - x & 0 \le x < 1, \\ 1 - x & 1 \le x < 2, \\ 4x - 9 & 2 \le x < 3. \end{cases}$$

This spline is displayed in Figure 4

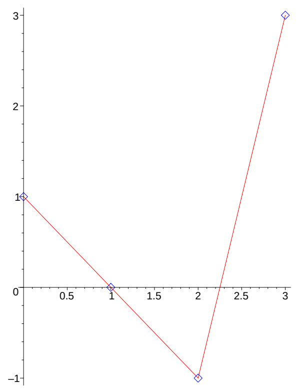

Figure 4: Graph of the linear interpolating spline of Example 1.

Remark Definition 1.15 does not make any statement about the relation between the location of the knots and the data sites for interpolation. We just observe that — for linear splines — it works to let the knots coincide with the data sites. We will not discuss the general problem in this class.

Example We use the same data as above, i.e.,

x	0	1	2	3
y	1	0	-1	3

,

but now we want to interpolate with a quadratic (C^1) spline. Again, we let the knots and data sites coincide, i.e., $t_i = x_i$, $i = 0, 1, 2, 3$.

Now it is not so obvious what S will look like. From Definition 1.15 we know that S will be of the form

$$S(x) = \begin{cases} S_0(x) = a_0x^2 + b_0x + c_0, & 0 \le x < 1, \\ S_1(x) = a_1x^2 + b_1x + c_1, & 1 \le x < 2, \\ S_2(x) = a_2x^2 + b_2x + c_2, & 2 \le x < 3. \end{cases}$$

Since we have three quadratic pieces there are 9 parameters (a_j, b_j, c_j, $j = 0, 1, 2$) to be determined. To this end, we collect the conditions we have on S. Obviously, we have the 4 interpolation conditions

$$S(x_i) = y_i, \qquad i = 0, 1, 2, 3.$$

Moreover, S needs to satisfy the C^1 (and C^0) smoothness conditions of Definition 1.15 at the interior knots. That leads to four more conditions:

$$\begin{aligned} S_0(t_1) &= S_1(t_1), \\ S_1(t_2) &= S_2(t_2), \\ S_0'(t_1) &= S_1'(t_1), \\ S_1'(t_2) &= S_2'(t_2). \end{aligned}$$

Thus, we have a total of 8 conditions to determine the 9 free parameters. This leaves at least one undetermined parameter (provided the above conditions are linearly independent).

Hoping that the conditions above are indeed independent, we add one more (arbitrary) condition $S_0'(0) = -1$ to obtain a square linear system. Therefore, we need to solve

$$\begin{aligned}
S_0(0) &= 1, \\
S_0(1) &= 0, \\
S_1(1) &= 0, \\
S_1(2) &= -1, \\
S_2(2) &= -1, \\
S_2(3) &= 3, \\
S_0'(1) - S_1'(1) &= 0, \\
S_1'(2) - S_2'(2) &= 0, \\
S_0'(0) &= -1.
\end{aligned}$$

Note that we have implemented the C^0 conditions at the two interior knots by stating the interpolation conditions for both adjoining pieces. In matrix form, the resulting linear system is

$$\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 4 & 2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 9 & 3 & 1 \\
2 & 1 & 0 & -2 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 1 & 0 & -4 & -1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \end{bmatrix}
=
\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ -1 \\ 3 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

The solution of this linear system is given by

$$\begin{gathered}
a_0 = 0,\ b_0 = -1,\ c_0 = 1, \\
a_1 = 0,\ b_1 = -1,\ c_1 = 1, \\
a_2 = 5,\ b_2 = -21,\ c_2 = 21,
\end{gathered}$$

or

$$S(x) = \begin{cases} 1 - x, & 0 \le x < 1, \\ 1 - x, & 1 \le x < 2, \\ 5x^2 - 21x + 21, & 2 \le x < 3. \end{cases}$$

This example is illustrated in the Maple worksheet `478578_SplineInterpolation.mws` and a plot of the quadratic spline computed in Example 2 is also provided in Figure 5.

Remark In order to efficiently evaluate a piecewise defined spline S at some point $x \in [t_0, t_n]$ one needs to be able to identify which polynomial piece to evaluate, i.e., determine in which subinterval $[t_i, t_{i+1})$ the evaluation point x lies. An algorithm (for linear splines) is given in on page 350 of the Kincaid/Cheney book.

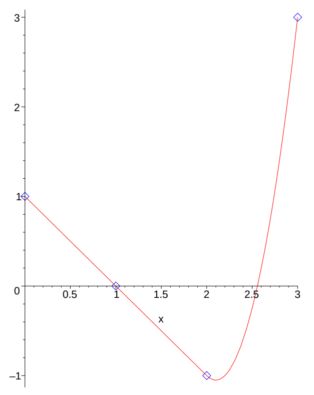

Figure 5: Graph of the quadratic interpolating spline of Example 2.

1.2.1 Cubic Splines

Another very popular spline is the (C^2) cubic spline. Assume we are given $n+1$ pieces of data (x_i, y_i), $i = 0, 1, \ldots, n$ to interpolate. Again, we let the knot sequence $\{t_i\}$ coincide with the data sites. According to Definition 1.15 the spline S will consist of n cubic polynomial pieces with a total of $4n$ parameters.

The conditions prescribed in Definition 1.15 are

$n+1$ interpolation conditions,

$n-1$ C^0 continuity conditions at interior knots,

$n-1$ C^1 continuity conditions at interior knots,

$n-1$ C^2 continuity conditions at interior knots,

for a total of $4n-2$ conditions. Assuming linear independence of these conditions, we will be able to impose two additional conditions on S.

There are many possible ways of doing this. We will discuss:

1. $S''(t_0) = S''(t_n) = 0$, (so-called *natural* end conditions).

2. Other boundary conditions, such as

$$S'(t_0) = f'(t_0), \quad S'(t_n) = f'(t_n)$$

   which lead to *complete* splines, or

$$S''(t_0) = f''(t_0), \quad S''(t_n) = f''(t_n).$$

   In either case, f' or f'' needs to be provided (or estimated) as additional data.

3. The so-called "not-a-knot" condition.

1.2.2 Cubic Natural Spline Interpolation

To simplify the notation we introduce the abbreviation

$$z_i = S''(t_i), \qquad i = 0, 1, \dots, n,$$

for the value of the second derivative at the knots. We point out that these values are *not* given as data, but are parameters to be determined.

Since S is cubic S'' will be linear. If we write this linear polynomial on the subinterval $[t_i, t_{i+1})$ in its Lagrange form we have

$$S_i''(x) = z_i \frac{t_{i+1} - x}{t_{i+1} - t_i} + z_{i+1} \frac{x - t_i}{t_{i+1} - t_i}.$$

With another abbreviation $h_i = t_{i+1} - t_i$ this becomes

$$S_i''(x) = \frac{z_i}{h_i}(t_{i+1} - x) + \frac{z_{i+1}}{h_i}(x - t_i). \tag{12}$$

Remark By assigning the value z_i to both pieces joining together at t_i we will automatically enforce continuity of the second derivative of S.

Now, we obtain a representation for the piece S_i by integrating (12) twice:

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 + C(x - t_i) + D(t_{i+1} - x). \tag{13}$$

The interpolation conditions (for the piece S_i)

$$S_i(t_i) = y_i, \quad S_i(t_{i+1}) = y_{i+1}$$

yield a 2×2 linear system for the constants C and D. This leads to

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x - t_i) + \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(t_{i+1} - x). \tag{14}$$

(Note that it is easily verified that (14) satisfies the interpolation conditions.)

Once we have determined the unknowns z_i each piece of the spline S can be evaluated via (14). We have not yet employed the C^1 continuity conditions at the interior knots, i.e.,

$$S_{i-1}'(t_i) = S_i'(t_i), \qquad i = 1, 2, \dots, n - 1. \tag{15}$$

Thus, we have $n - 1$ additional conditions. Since there are $n + 1$ unknowns z_i, $i = 0, 1, \dots, n$, we fix the second derivative at the endpoints to be zero, i.e.,

$$z_0 = z_n = 0.$$

These are the so-called *natural end conditions*. Differentiation of (14) leads to

$$S_i'(x) = -\frac{z_i}{2h_i}(t_{i+1} - x)^2 + \frac{z_{i+1}}{2h_i}(x - t_i)^2 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right) - \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right).$$

Using this expression in (15) results in

$$z_{i-1}h_{i-1}+2(h_{i-1}+h_i)z_i+z_{i+1}h_i = \frac{6}{h_i}(y_{i+1}-y_i)-\frac{6}{h_{i-1}}(y_i-y_{i-1}), \quad i=1,2,\dots,n-1.$$

This represents a symmetric tridiagonal system for the unknowns $z_1,\dots,z_{n-1}$ of the form

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & 0 & \dots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & \dots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n-2} \\ r_{n-1} \end{bmatrix}.$$

Here $h_i = t_{i+1}-t_i$ and

$$r_i = \frac{6}{h_i}(y_{i+1}-y_i)-\frac{6}{h_{i-1}}(y_i-y_{i-1}), \quad i=1,2,\dots,n-1.$$

Remark The system matrix is even *diagonally dominant* since

$$2(h_{i-1}+h_i) > h_{i-1}+h_i$$

since all $h_i > 0$ due to the fact that the knots t_i are distinct and form an increasing sequence. Thus, a very efficient tridiagonal version of Gauss elimination without pivoting can be employed to solve for the missing z_i (see the textbook on page 353 for such an algorithm).

1.2.3 "Optimality" of Natural Splines

Among all smooth functions which interpolate a given function f at the knots $t_0, t_1, \dots, t_n$, the natural spline is the smoothest, i.e.,

Theorem 1.16 *Let $f \in C^2[a,b]$ and $a = t_0 < t_1 < \dots < t_n = b$. If S is the cubic natural spline interpolating f at t_i, $i = 0,1,\dots,n$, then*

$$\int_a^b \left[S''(x)\right]^2 dx \le \int_a^b \left[f''(x)\right]^2 dx.$$

Remark The integrals in Theorem 1.16 can be interpreted as bending energies of a thin rod, or as "total curvatures" (since for small deflections the curvature $\kappa \approx f''$). This optimality result is what gave rise to the name *spline*, since early ship designers used a piece of wood (called a spline) fixed at certain (interpolation) points to describe the shape of the ship's hull.

Proof (of optimality theorem) Define an auxiliary function $g = f - S$. Then $f = S + g$ and $f'' = S'' + g''$ or

$$\left(f''\right)^2 = \left(S''\right)^2 + \left(g''\right)^2 + 2S''g''.$$

Therefore,

$$\int_a^b \left(f''(x)\right)^2 dx = \int_a^b \left(S''(x)\right)^2 dx + \int_a^b \left(g''(x)\right)^2 dx + \int_a^b 2S''(x)g''(x)dx.$$

Obviously,

$$\int_a^b \left(g''(x)\right)^2 dx \geq 0,$$

so that we are done is we can show that also

$$\int_a^b 2S''(x)g''(x)dx \geq 0.$$

To this end we break the interval $[a,b]$ into the subintervals $[t_{i-1}, t_i]$, and get

$$\int_a^b S''(x)g''(x)dx = \sum_{i=1}^n \int_{t_{i-1}}^{t_i} S''(x)g''(x)dx.$$

Now we can integrate by parts (with $u = S''(x)$, $dv = g''(x)dx$) to obtain

$$\sum_{i=1}^n \left\{ \left[S''(x)g'(x)\right]_{t_{i-1}}^{t_i} - \int_{t_{i-1}}^{t_i} S'''(x)g'(x)dx \right\}.$$

The first term is a telescoping sum so that

$$\sum_{i=1}^n \left[S''(x)g'(x)\right]_{t_{i-1}}^{t_i} = S''(t_n)g'(t_n) - S''(t_0)g'(t_0) = 0$$

due to the natural end conditions of the spline S.

This leaves

$$\int_a^b S''(x)g''(x)dx = -\sum_{i=1}^n \int_{t_{i-1}}^{t_i} S'''(x)g'(x)dx.$$

However, since S_i is a cubic polynomial we know that $S'''(x) \equiv c_i$ on $[t_{i-1}, t_i)$. Thus,

$$\begin{aligned} \int_a^b S''(x)g''(x)dx &= -\sum_{i=1}^n c_i \int_{t_{i-1}}^{t_i} g'(x)dx \\ &= -\sum_{i=1}^n c_i \left[g(x)\right]_{t_{i-1}}^{t_i} = 0. \end{aligned}$$

The last equation holds since $g(t_i) = f(t_i) - S(t_i)$, $i = 0, 1, \ldots, n$, and S interpolates f at the knots t_i. ∎

Remark Cubic natural splines should *not be considered the natural choice for cubic spline interpolation.* This is due to the fact that one can show that the (rather arbitrary) choice of natural end conditions yields an interpolation error estimate of only $\mathcal{O}(h^2)$, where $h = \max\limits_{i=1,\ldots,n} \Delta t_i$ and $\Delta t_i = t_i - t_{i-1}$. This needs to be compared to the estimate of $\mathcal{O}(h^4)$ obtainable by cubic polynomials as well as other cubic spline methods.

1.2.4 Cubic Complete Spline Interpolation

The derivation of cubic complete splines is similar to that of the cubic natural splines. However, we impose the additional end constraints

$$S'(t_0) = f'(t_0), \qquad S'(t_n) = f'(t_n).$$

This requires additional data (f' at the endpoints), but can be shown to yields an $\mathcal{O}(h^4)$ interpolation error estimate.

Moreover, an energy minimization theorem analogous to Theorem 1.16 also holds since

$$\begin{aligned}\sum_{i=1}^{n} \left[S''(x)g'(x)\right]_{t_{i-1}}^{t_i} &= S''(t_n)g'(t_n) - S''(t_0)g'(t_0) \\ &= S''(t_n)\left(f'(t_n) - S'(t_n)\right) - S''(t_0)\left(f'(t_0) - S'(t_0)\right) = 0\end{aligned}$$

by the end conditions.

1.2.5 Not-a-Knot Spline Interpolation

One of the most effective cubic spline interpolation methods is obtained by choosing the knots *different from the data sites.* In particular, if the data is of the form

x_0	x_1	x_2	$\ldots$	x_{n-1}	x_n
y_0	y_1	y_2	$\ldots$	y_{n-1}	y_n

,

then we take the $n-1$ knots as

x_0	x_2	x_3	$\ldots$	x_{n-2}	x_n
t_0	t_1	t_2	$\ldots$	t_{n-3}	t_{n-2}

,

i.e., the data sites x_1 and x_{n-1} are "not-a-knot".

The knots now define $n-2$ cubic polynomial pieces with a total of $4n-8$ coefficients. On the other hand, there are $n+1$ interpolation conditions together with three sets of $(n-3)$ smoothness conditions at the interior knots. Thus, the number of conditions is equal to the number of unknown coefficients, and no additional (arbitrary) conditions need to be imposed to solve the interpolation problem.

One can interpret this approach as using only one cubic polynomial piece to represent the first two (last two) data segments.

The cubic not-a-knot spline has an $\mathcal{O}(h^4)$ interpolation error, and requires no additional data. More details can be found in the book "A Practical Guide to Splines" by Carl de Boor.

Remark 1. If we are given also derivative information at the data sites x_i, $i = 0, 1, \ldots, n$, then we can perform piecewise cubic Hermite interpolation. The resulting function will be C^1 continuous, and one can show that the interpolation error is $\mathcal{O}(h^4)$. However, this function is *not* considered a spline function since it does not have the required smoothness.

2. There are also piecewise cubic interpolation methods that *estimate* derivative information at the data sites, i.e., no derivative information is provided as data. Two such (local) methods are named after Bessel (yielding an $\mathcal{O}(h^3)$ interpolation error) and Akima (with an $\mathcal{O}(h^2)$ error). Again, they are not spline functions as they are only C^1 smooth.

1.3 Numerical Integration Based on Interpolation

We now turn to approximate integration (or *quadrature*). The simplest numerical integration methods are the left/right endpoint and the midpoint rules studied in calculus. We will focus on methods based on polynomial interpolation. The idea is a simple one which is also frequently used for numerical differentiation: interpolate the integrand at $n+1$ points, and then (exactly) integrate the degree n polynomial. Using the Lagrange form of the interpolating polynomial this means

$$\begin{aligned}\int_a^b f(x)dx &\approx \int_a^b p(x)dx = \int_a^b \sum_{i=0}^n f(x_i)\ell_i(x)dx \\ &= \sum_{i=0}^n f(x_i)\int_a^b \ell_i(x)dx \\ &= \sum_{i=0}^n A_i f(x_i) \qquad (16)\end{aligned}$$

with *weights*

$$A_i = \int_a^b \ell_i(x)dx. \qquad (17)$$

Depending on the number of points (degree of the polynomial) used we have a different quadrature rule. Formulas of the type (16) and (17) are collectively known as *Newton-Cotes* formulas.

Example (Trapezoid Rule) In the case $n = 1$ we get

$$\begin{aligned}\int_a^b f(x)dx &\approx A_0 f(x_0) + A_1 f(x_1) \\ &= A_0 f(a) + A_1 f(b).\end{aligned}$$

where

$$A_0 = \int_a^b \ell_0(x)dx = \int_a^b \frac{x-b}{a-b}dx = \frac{1}{a-b}\left.\frac{(x-b)^2}{2}\right|_a^b = \frac{b-a}{2},$$

and

$$A_1 = \int_a^b \ell_1(x)dx = \int_a^b \frac{x-a}{b-a}dx = \frac{1}{b-a}\left.\frac{(x-a)^2}{2}\right|_a^b = \frac{b-a}{2}.$$

Together we get

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left[f(a)+f(b)\right].$$

This is the well-known *trapezoid rule*. We can improve the accuracy by using a piecewise linear spline interpolation, i.e., we subdivide the interval $[a, b]$ into subintervals

$[x_0, x_1], [x_1, x_2], \ldots, [x_{N-1}, x_N]$, and form a linear polynomial interpolant on each subinterval. Then

$$\int_a^b f(x)dx \approx \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} p_i(x)dx.$$

Using the trapezoid rule on each subinterval we have

$$\int_a^b f(x)dx \approx \sum_{i=0}^{N-1} \frac{x_{i+1} - x_i}{2} \left[f(x_i) + f(x_{i+1})\right].$$

To simplify this expression we assume that the subintervals are of equal length h, i.e., $x_i = a + ih$, $i = 0, 1, \ldots, N$, with $h = \frac{b-a}{N}$. This results in the *composite trapezoid rule*

$$\int_a^b f(x)dx \approx T_N f = \frac{b-a}{2N} \left[f(a) + 2\sum_{i=1}^{N-1} f(a + ih) + f(b)\right].$$

The error of the composite trapezoid rule is the subject of

Theorem 1.17 *If $f \in C^2[a, b]$ and $h = \frac{b-a}{N}$ are used for T_N then the error*

$$E_{T_N} f = \int_a^b f(x)dx - T_N f = -\frac{b-a}{12} h^2 f''(\xi) = \mathcal{O}(h^2),$$

where $\xi \in (a, b)$.

Proof We start with the error for a single subinterval (i.e., for the basic trapezoid rule). Without loss of generality we let $[a, b] = [0, h]$. Then, using the error formula for polynomial interpolation (see Theorem 1.9),

$$\begin{aligned}
E_{T_1} f &= \int_0^h \left[f(x) - p_1(x)\right] dx \\
&= \int_0^h \frac{1}{2} f''(\xi_x) \prod_{i=0}^{1} (x - x_i)dx \\
&\overset{x_0=0, x_1=h}{=} \int_0^h \frac{1}{2} f''(\xi_x) x(x - h)dx \\
&= \frac{1}{2} f''(\xi) \int_0^h x(x - h)dx,
\end{aligned}$$

where we have used the Mean-Value Theorem for integrals of continuous functions in the last step. The last integral can of course be evaluated, and we obtain

$$E_{T_1} f = -\frac{1}{12} h^3 f''(\xi), \quad \xi \in (a, b).$$

For the composite rule we now have

$$E_{T_N} f = \sum_{i=1}^{N} E_{T_i} f = \sum_{i=1}^{N} -\frac{1}{12} h^3 f''(\xi_i).$$

By rewriting one copy of h as $\frac{b-a}{N}$ we get

$$E_{T_N}f = -\frac{b-a}{12}h^2\sum_{i=1}^{N}\frac{f''(\xi_i)}{N}.$$

Now, for some $\xi \in (a,b)$ we have

$$\sum_{i=1}^{N}\frac{f''(\xi_i)}{N} = f''(\xi),$$

and therefore we are done. ∎

Example (Simpson's Rule) The Newton-Cotes formula in the case $n = 2$ (quadratic polynomial interpolant on a single interval $[a,b]$) yields (the details are omitted here)

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right], \tag{18}$$

where $h = \frac{b-a}{2}$. Equation (18) is known as *Simpson's rule.*

In order to formulate a *composite Simpson rule* we need an even number, N, of subintervals. Then

$$\begin{aligned}\int_a^b f(x)dx &= \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \ldots + \int_{x_{N-2}}^{x_N} f(x)dx \\ &\approx \sum_{i=1}^{N/2}\int_{x_{2i-2}}^{x_{2i}} p_i(x)dx \\ &= \sum_{i=1}^{N/2}\frac{h}{3}\left[f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right],\end{aligned}$$

where we have used Simpson's rule (18) on each subinterval $[x_{2i-2}, x_{2i}]$. We can rewrite the previous formula more efficiently as

$$S_N f = \frac{h}{3}\left[f(a) + 2\sum_{i=2}^{N/2} f(x_{2i-2}) + 4\sum_{i=1}^{N/2} f(x_{2i-1}) + f(b)\right] \tag{19}$$

since N is even. This is the composite Simpson rule. For the error we have

Theorem 1.18 *If $f \in C^4[a,b]$ and $h = \frac{b-a}{N}$ with N even are used for S_N, then*

$$E_{S_N}f = -\frac{b-a}{180}h^4 f^{(4)}(\xi) = \mathcal{O}(h^4),$$

where $\xi \in (a,b)$.

Remark Note that the accuracy corresponds to that expected for *cubic* polynomials even though only quadratic interpolating polynomials were used. We get one order of h "for free".

Proof (of Simpson error formula) Parts can be found in the book by Kincaid and Cheney on page 484. See also the discussion of Peano kernels below, and Homework Assignment 2. ∎

Remark 1. An error estimate for the general Newton-Cotes formulas (on a single interval) can be found in the Kincaid/Cheney book on pages 486-487.

2. The error is minimized by taking the interpolation nodes as the zeros of the Chebyshev polynomials *of the second kind*.

1.4 Peano Kernels

A useful (and rather beautiful) tool for error estimates (especially for numerical differentiation and integration problems) is the use of *Peano kernels* and the Peano kernel theorem.

A *linear functional* L on a linear space, e.g., $C^\nu[a,b]$, is a mapping that maps a function from this space onto a scalar and is linear, i.e., $L(\alpha f + \beta g) = \alpha Lf + \beta Lg$ for $\alpha, \beta \in \mathbb{R}$, $f, g \in C^\nu[a,b]$.

Example 1. Point evaluation functional:

$$Lf = f(x).$$

2. (Definite) Integration functional:

$$Lf = \int_a^b f(x)dx.$$

Note that linear combinations of linear functionals form new linear functionals. A fairly general linear functional is

$$Lf = \sum_{i=0}^{n} \left[\int_a^b \alpha_i(x) f^{(i)}(x)dx + \sum_{j=1}^{n} \beta_{ij} f^{(i)}(\xi_{ij}) \right]. \tag{20}$$

Here $\xi_{ij} \in [a,b]$, $f^{(i)}$ denotes the ith derivative of f, β_{ij} are real numbers, and the functions α_i are at least piecewise continuous on $[a,b]$. The function f should be in $C^n[a,b]$.

Furthermore, we say that a functional *annihilates polynomials* $\mathbb{P}_\nu$ if

$$Lp = 0, \qquad \text{for all } p \in \mathbb{P}_\nu.$$

The νth *Peano kernel* of L as in (20) is the function

$$k_\nu(\xi) = L\left[(x-\xi)_+^\nu\right], \qquad \xi \in [a,b],$$

where $\nu \geq n$ and

$$(x-\xi)_+^m = \begin{cases} (x-\xi)^m, & x \geq \xi \\ 0, & x < \xi, \end{cases}$$

is the *truncated power function*.

Example Let's compute the Peano kernel k_1 for the linear functional defined by

$$Lf = \int_0^\pi (\cos x) f'(x) dx.$$

By definition of the Peano kernel we have

$$\begin{aligned} k_1(\xi) &= L\left[(x-\xi)_+\right] \\ &= \int_0^\pi (\cos x)\frac{d}{dx}(x-\xi)_+ dx. \end{aligned}$$

Using the definition of the truncated power function we can rewrite the integral and then evaluate

$$\int_\xi^\pi (\cos x)\frac{d}{dx}(x-\xi) dx = \int_\xi^\pi \cos x dx = -\sin\xi.$$

Theorem 1.19 *(Peano Kernel Theorem) If a functional L of the form (20) annihilates polynomials $\mathbb{P}_\nu$, then for all $f \in C^{\nu+1}[a,b]$,*

$$Lf = \frac{1}{\nu!}\int_a^b k_\nu(\xi) f^{(\nu+1)}(\xi) d\xi$$

where $\nu \geq n$ and k_ν is the Peano kernel of L.

Remark The Peano kernel theorem allows estimates of the form

$$\begin{aligned} |Lf| &\leq \frac{1}{\nu!}\|k_\nu\|_1 \|f^{(\nu+1)}\|_\infty, \\ |Lf| &\leq \frac{1}{\nu!}\|k_\nu\|_\infty \|f^{(\nu+1)}\|_1, \\ |Lf| &\leq \frac{1}{\nu!}\|k_\nu\|_2 \|f^{(\nu+1)}\|_2, \end{aligned}$$

where we used the *norms*

$$\begin{aligned} \|f\|_1 &= \int_a^b |f(x)| dx, \\ \|f\|_2 &= \left(\int_a^b |f(x)|^2 dx\right)^{1/2}, \\ \|f\|_\infty &= \max_{x\in[a,b]} |f(x)|. \end{aligned}$$

Example Consider the integral

$$\int_0^1 f(\xi) d\xi$$

and find an approximate integration formula of the form

$$\int_0^1 f(\xi) d\xi \approx b_1 f(0) + b_2 f(\frac{1}{2}) + b_3 f(1)$$

that is exact if f is a polynomial in $\mathbb{P}_3$, and find its error.

To answer this question we consider the linear functional

$$Lf = \int_0^1 f(\xi)d\xi - b_1 f(0) + b_2 f(\frac{1}{2}) + b_3 f(1),$$

and first find b_1, b_2, and b_3 so that L annihilates $\mathbb{P}_3$.

If we let $f(x) = 1$, then we get the condition

$$0 = Lf = \int_0^1 1 d\xi - (b_1 + b_2 + b_3) = 1 - b_1 - b_2 - b_3.$$

For $f(x) = x$ we get

$$0 = Lf = \int_0^1 \xi d\xi - (\frac{1}{2}b_2 + b_3) = \frac{1}{2} - \frac{1}{2}b_2 - b_3,$$

for $f(x) = x^2$ we get

$$0 = Lf = \int_0^1 \xi^2 d\xi - (\frac{1}{4}b_2 + b_3) = \frac{1}{3} - \frac{1}{4}b_2 - b_3,$$

and for $f(x) = x^3$ we get

$$0 = Lf = \int_0^1 \xi^3 d\xi - (\frac{1}{8}b_2 + b_3) = \frac{1}{4} - \frac{1}{8}b_2 - b_3.$$

The unique solution of this system of 4 linear equations in 3 unknowns is

$$b_1 = \frac{1}{6},\ b_2 = \frac{2}{3},\ b_3 = \frac{1}{6},$$

and therefore

$$\int_0^1 f(\xi)d\xi \approx \frac{1}{6}\left[f(0) + 4f(\frac{1}{2}) + f(1)\right].$$

To estimate the error in this approximation we use the Peano kernel of L. It is given by

$$\begin{aligned}
k_3(\xi) &= L\left[(x-\xi)_+^3\right] \\
&= \int_0^1 (x-\xi)_+^3 dx - \frac{1}{6}\left[(0-\xi)_+^3 + 4(\frac{1}{2}-\xi)_+^3 + (1-\xi)_+^3\right] \\
&= \int_\xi^1 (x-\xi)^3 dx - \frac{1}{6}\left[4(\frac{1}{2}-\xi)_+^3 + (1-\xi)_+^3\right] \\
&= \frac{(1-\xi)^4}{4} - \frac{1}{6}\begin{cases}\left[4(\frac{1}{2}-\xi)^3 + (1-\xi)^3\right], & 0 \le \xi \le \frac{1}{2} \\ (1-\xi)^3, & \frac{1}{2} \le \xi \le 1.\end{cases} \\
&= \begin{cases} -\frac{1}{12}\xi^3(2-3\xi), & 0 \le \xi \le \frac{1}{2} \\ -\frac{1}{12}(1-\xi)^3(3\xi - 1), & \frac{1}{2} \le \xi \le 1.\end{cases}
\end{aligned}$$

Now the Peano kernel theorem says that

$$\int_0^1 f(\xi)d\xi - \frac{1}{6}\left[f(0) + 4f(\frac{1}{2}) + f(1)\right] = Lf = \frac{1}{3!}\int_0^1 k_3(\xi) f^{(4)}(\xi)d\xi,$$

and we can explicitly calculate estimates of the form

$$|Lf| \le \frac{1}{1152}\|f^{(4)}\|_1,\ |Lf| \le \frac{\sqrt{14}}{8064}\|f^{(4)}\|_2,\ |Lf| \le \frac{1}{2880}\|f^{(4)}\|_\infty \qquad f \in C^4[0,1]$$

since

$$\|k_3\|_1 = \frac{1}{480}, \quad \|k_3\|_2 = \frac{\sqrt{14}}{1344}, \quad \|k_3\|_\infty = \frac{1}{192}.$$

1.5 ODEs and the Lipschitz Condition

We consider the *system of first-order ODE IVP*

$$\begin{aligned} \boldsymbol{y}'(t) &= \frac{d\boldsymbol{y}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t)), \qquad t \ge t_0, & (21) \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0. & (22) \end{aligned}$$

Here

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix}, \quad \boldsymbol{y}_0 = \begin{bmatrix} y_{0,1} \\ \vdots \\ y_{0,d} \end{bmatrix}, \quad \boldsymbol{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_d \end{bmatrix}, \quad \in \mathbb{R}^d.$$

Remark This approach covers not only first-order ODEs, but also higher-order ODEs, since any d-th order ODE IVP can be converted to a system of d first-order IVPs (see Assignment 2).

If $\boldsymbol{f}(t, \boldsymbol{y}) = A(t)\boldsymbol{y} + \boldsymbol{b}(t)$ for some $d \times d$ matrix-valued function A and $d \times 1$ vector-valued function $\boldsymbol{b}$, then the ODE is *linear*, and if $\boldsymbol{b}(t) = \boldsymbol{0}$ it is linear and *homogeneous*. Otherwise it is *nonlinear*. If $\boldsymbol{f}$ is independent of t, the ODE is called *autonomous*, and if $\boldsymbol{f}$ is independent of $\boldsymbol{y}$, then the ODE system reduces to a (vector of) indefinite integral(s).

Theorem 1.20 *(Picard-Lindelöf: Existence and Uniqueness) Let $\mathbb{B}$ be the ball $\mathbb{B} = \{\boldsymbol{x} \in \mathbb{R}^d :\ \|\boldsymbol{x} - \boldsymbol{y}_0\| \le b\}$ and let $\mathbb{S}$ be the cylinder*

$$\mathbb{S} = \{(t, \boldsymbol{x}) :\ t \in [t_0, t_0 + a],\ \boldsymbol{x} \in \mathbb{B}\}$$

where $a, b > 0$. If $\boldsymbol{f}$ is continuous on $\mathbb{S}$ and $\boldsymbol{f}$ also satisfies the Lipschitz condition

$$\|\boldsymbol{f}(t, \boldsymbol{x}) - \boldsymbol{f}(t, \boldsymbol{y})\| \le \lambda \|\boldsymbol{x} - \boldsymbol{y}\|, \qquad \boldsymbol{x}, \boldsymbol{y} \in \mathbb{B},$$

then the IVP (21), (22) has a unique solution on $[t_0, t_0 + \alpha]$, where α is some constant that depends on a, b and $\boldsymbol{f}$. In fact,

$$\alpha = \min\left\{a, \frac{b}{\sup_{(t,\boldsymbol{x})\in\mathbb{S}} \|\boldsymbol{f}(t, \boldsymbol{x})\|}\right\}.$$

Note that in the system setting we need to measure differences of vectors in some appropriate *norm* instead of simple absolute value.

Remark The proof of this theorem is rather involved.

Example For a single equation, continuity of the partial derivative $\frac{\partial f(t,y)}{\partial y}$ on $\mathbb{S}$ guarantees Lipschitz continuity of f with

$$\lambda = \max_{\substack{t\in[t_0,t_0+a]\\ y\in\mathbb{B}}} \left|\frac{\partial f(t,y)}{\partial y}\right|.$$

For the initial value problem

$$y'(t) = 2t\,(y(t))^2\,, \qquad y(0) = 1,$$

we have

$$f(t,y) = 2ty^2, \qquad \frac{\partial f(t,y)}{\partial y} = 4ty,$$

which are both continuous on all of $\mathbb{R}^2$. The theorem above guarantees existence and uniqueness of a solution for t near $t_0 = 0$. In fact, it is given by

$$y(t) = \frac{1}{1-t^2}, \qquad -1 < t < 1.$$

However, we see that just because f and $\frac{\partial f(t,y)}{\partial y}$ are continuous on all of $\mathbb{R}^2$ we cannot expect existence or uniqueness of a solution y for all t.

Remark In the system setting a sufficient condition for Lipschitz continuity of $\boldsymbol{f}$ is given by continuity of the *Jacobian matrix*

$$\frac{\partial \boldsymbol{f}(t,\boldsymbol{y})}{\partial \boldsymbol{y}} = \left[\frac{\partial f_i(t,y_1,\ldots,y_d)}{\partial y_j}\right]_{i,j=1}^{d}.$$

Remark Recall that a linear system

$$\boldsymbol{y}' = A\boldsymbol{y}, \qquad t \ge t_0, \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0$$

with $d\times d$ matrix A always has a unique solution. It is given by

$$\boldsymbol{y}(t) = \sum_{\ell=1}^{d} e^{\lambda_\ell (t-t_0)}\boldsymbol{\alpha}_\ell, \qquad t \ge t_0,$$

where $\lambda_1,\ldots,\lambda_d$ are the eigenvalues of A, and the $\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_d \in \mathbb{R}^d$ are vectors (eigenvectors if the eigenvalues are distinct).

1.6 Euler's Method

1.6.1 The basic Algorithm

Recall that we are interested in general first-order IVPs (21), (22) of the form

$$\begin{aligned} \boldsymbol{y}'(t) &= \boldsymbol{f}(t,\boldsymbol{y}(t)), \qquad t \ge t_0 \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0. \end{aligned}$$

It is our goal to derive numerical methods for the solution of this kind of problem. The first, and probably best known, method is called *Euler's method*. Even though this is one of the "original" numerical methods for the solution of IVPs, it remains important for both practical and theoretical purposes.

The method is derived by considering the approximation

$$\boldsymbol{y}'(t) \approx \frac{\boldsymbol{y}(t+h) - \boldsymbol{y}(t)}{h}$$

of the first derivative. This implies

$$\boldsymbol{y}(t+h) \approx \boldsymbol{y}(t) + h\boldsymbol{y}'(t),$$

which – using the differential equation (21) – becomes

$$\boldsymbol{y}(t+h) \approx \boldsymbol{y}(t) + h\boldsymbol{f}(t, \boldsymbol{y}(t)). \tag{23}$$

Introducing a sequence of points $t_0, t_1 = t_0 + h, t_2 = t_0 + 2h, \ldots, t_N = t_0 + Nh$, this immediately leads to an iterative algorithm.

Algorithm

```
Input $t_0$, $\boldsymbol{y}_0$, $\boldsymbol{f}$, $h$, $N$

$t = t_0$, $\boldsymbol{y} = \boldsymbol{y}_0$

for $n = 1$ to $N$ do

    $\boldsymbol{y} \leftarrow \boldsymbol{y} + h\boldsymbol{f}(t, \boldsymbol{y})$
    $t \leftarrow t + h$

end
```

Remark 1. Alternately, we can derive the formula for Euler's method via integration. Since the IVP gives us both an initial condition as well as the slope $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ of the solution, we can assume that the slope is constant on a small interval $[t_0, t_0 + h]$, i.e., $\boldsymbol{f}(t, \boldsymbol{y}(t)) \approx \boldsymbol{f}(t_0, \boldsymbol{y}(t_0))$ for $t \in [t_0, t_0 + h]$. Then we can integrate to get

$$\begin{aligned} \boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau \\ &\approx \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(t_0, \boldsymbol{y}(t_0))d\tau \\ &= \boldsymbol{y}(t_0) + (t - t_0)\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) \end{aligned}$$

— Euler's method.

2. Note that Euler's method yields a set of discrete points $(t_n, \boldsymbol{y}_n)$, $n = 1, \ldots, N$, which approximate the graph of the solution $\boldsymbol{y} = \boldsymbol{y}(t)$. In order to obtain a continuous solution one must use an interpolation or approximation method.

3. Euler's method is illustrated in the Maple worksheet `472_Euler_Taylor.mws`.

4. In principle, it is easy to use Euler's method with a variable step size, i.e.,

$$\boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h_n\boldsymbol{f}(t_n, \boldsymbol{y}_n),$$

but analysis of the method is simpler with a constant step size $h_n = h$.

1.6.2 Taylor Series Methods

An immediate generalization of Euler's method are the so-called general *Taylor series methods*. We use a Taylor expansion

$$\boldsymbol{y}(t+h) = \boldsymbol{y}(t) + h\boldsymbol{y}'(t) + \frac{h^2}{2}\boldsymbol{y}''(t) + \frac{h^3}{6}\boldsymbol{y}'''(t) + \ldots,$$

and therefore obtain the numerical approximation

$$\boldsymbol{y}(t+h) \approx \sum_{k=0}^{\nu} \frac{h^k \boldsymbol{y}^{(k)}(t)}{k!} \tag{24}$$

which is referred to as a ν-th order Taylor series method.

Remark 1. Obviously, Euler's method is a first-order Taylor method.

2. In order to program a Taylor method we need to pre-compute all higher-order derivatives of $\boldsymbol{y}$ required by the method since the differential equation only provides a representation for $\boldsymbol{y}'$. This implies that we will end up with code that depends on (and changes with) the IVP to be solved.

3. Computer software with symbolic manipulation capabilities (such as Maple or Mathematica) allows us to write code for Taylor methods for arbitrary IVPs.

We illustrate the traditional treatment of a second-order Taylor method in the following example.

Example Consider the initial value problem ($d = 1$)

$$\begin{aligned} y'(t) &= y(t) - t^2 + 1 \\ y(0) &= \frac{1}{2}. \end{aligned}$$

The second-order Taylor approximation is given by

$$y(t+h) \approx y(t) + hy'(t) + \frac{h^2}{2}y''(t).$$

Therefore, we need to express $y'(t)$ and $y''(t)$ in terms of y and t so that an iterative algorithm can be formulated.

From the differential equation

$$y'(t) = f(t, y(t)) = y(t) - t^2 + 1.$$

Therefore, differentiating this relation,

$$y''(t) = y'(t) - 2t,$$

and this can be incorporated into the following algorithm.

Algorithm

```
Input $t_0$, $y_0$, $f$, $h$, $N$
$t = t_0$, $y = y_0$
for $n = 1$ to $N$ do
    $y' = f(t, y)$
    $y'' = y' - 2t$
    $y \leftarrow y + hy' + \frac{h^2}{2} y''$
    $t \leftarrow t + h$
end
```

Remark 1. Two modifications are suggested to make the algorithm more efficient and numerically stable.

(a) Replace the computation of y by the nested formulation

$$y = y + h\left(y' + \frac{h}{2} y''\right).$$

(b) Advance the time t via $t = t_0 + nh$.

2. An example of a fourth-order Taylor method is given in the Maple worksheet `478578_Euler_Taylor.mws`.

1.6.3 Errors and Convergence

When considering errors introduced using the Taylor series or Euler approximation we need to distinguish between two different types of error:

- *local* truncation error, and
- *global* truncation error.

The local truncation error is the error introduced directly by truncation of the Taylor series, i.e., *at each time step* we have an error

$$E_\nu = \frac{h^{\nu+1}}{(\nu+1)!} y^{(\nu+1)}(t + \theta h), \qquad 0 < \theta < 1.$$

Thus, the ν-th order Taylor method has an $\mathcal{O}(h^{\nu+1})$ local truncation error.

The global truncation error is the error that results if we use a ν-th order Taylor method having $\mathcal{O}(h^{\nu+1})$ local truncation error to solve our IVP up to time $t = t_0 + t^*$. Since we will be performing

$$N = \left\lfloor \frac{t^*}{h} \right\rfloor$$

steps we see that one order of h is lost in the global truncation error, i.e., the global truncation error is of the order $\mathcal{O}(h^\nu)$.

Remark

- Of course, truncation errors are independent of roundoff errors which can add to the overall error.
- As we will see later, a method with $\mathcal{O}(h^{\nu+1})$ local accuracy need not be globally ν-th order. In fact, it need not converge at all. *Stability* will be the key to convergence.

A numerical method for the IVP (21), (22) is called *convergent* if for every Lipschitz function $\boldsymbol{f}$ and every $t^* > 0$ we have

$$\lim_{h\to 0^+} \max_{n=0,1,\ldots,N} \|\boldsymbol{y}_{n,h} - \boldsymbol{y}(t_n)\| = 0.$$

In other words, if the numerical solution approaches the analytic solution for increasingly smaller step sizes h.

For Euler's method we can establish convergence (and therefore the above heuristics regarding truncation errors are justified).

Theorem 1.21 *Euler's method is convergent.*

Proof To simplify the proof we assume that $\boldsymbol{f}$ (and therefore also $\boldsymbol{y}$) is analytic. We introduce the notation

$$\boldsymbol{e}_{n,h} = \boldsymbol{y}_{n,h} - \boldsymbol{y}(t_n)$$

fir the error at step n. We need to show

$$\lim_{h\to 0^+} \max_{n=0,1,\ldots,N} \|\boldsymbol{e}_{n,h}\| = 0.$$

Taylor's theorem for the analytic solution $\boldsymbol{y}$ gives us (since $t_{n+1} = t_n + h$)

$$\boldsymbol{y}(t_{n+1}) = \boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \mathcal{O}(h^2).$$

Replacing $\boldsymbol{y}'$ by the ODE (21) we have

$$\boldsymbol{y}(t_{n+1}) = \boldsymbol{y}(t_n) + h\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \mathcal{O}(h^2).$$

From Euler's method we have for the numerical solution

$$\boldsymbol{y}_{n+1,h} = \boldsymbol{y}_{n,h} + h\boldsymbol{f}(t_n, \boldsymbol{y}_{n,h}).$$

The difference of these last two expressions yields

$$\begin{aligned}
\boldsymbol{e}_{n+1,h} &= \boldsymbol{y}_{n+1,h} - \boldsymbol{y}(t_{n+1}) \\
&= [\boldsymbol{y}_{n,h} + h\boldsymbol{f}(t_n, \boldsymbol{y}_{n,h})] - \left[\boldsymbol{y}(t_n) + h\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \mathcal{O}(h^2)\right] \\
&= \boldsymbol{e}_{n,h} + h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_{n,h}) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\right] + \mathcal{O}(h^2).
\end{aligned}$$

Since $\boldsymbol{y}_{n,h} = \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}$ we have

$$\boldsymbol{e}_{n+1,h} = \boldsymbol{e}_{n,h} + h\left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\right] + \mathcal{O}(h^2).$$

Next we can apply norms and use the triangle inequality to obtain

$$\|\boldsymbol{e}_{n+1,h}\| \leq \|\boldsymbol{e}_{n,h}\| + h\|\boldsymbol{f}(t_n, \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\| + ch^2.$$

Here we also used the definition of $\mathcal{O}$-notation, i.e., $g(h) = \mathcal{O}(h^p)$ if $|g(h)| \leq ch^p$ for some constant c independent of h.

Now, note that $\boldsymbol{f}$ is Lipschitz, i.e., $\|\boldsymbol{f}(t, \boldsymbol{x}) - \boldsymbol{f}(t, \boldsymbol{y})\| \leq \lambda\|\boldsymbol{x} - \boldsymbol{y}\|$. Taking $\boldsymbol{x} = \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}$ and $\boldsymbol{y} = \boldsymbol{y}(t_n)$ we obtain

$$\begin{aligned} \|\boldsymbol{e}_{n+1,h}\| &\leq \|\boldsymbol{e}_{n,h}\| + h\lambda\|\boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h} - \boldsymbol{y}(t_n))\| + ch^2 \\ &= (1 + h\lambda)\|\boldsymbol{e}_{n,h}\| + ch^2. \end{aligned}$$

We can use induction to show that

$$\|\boldsymbol{e}_{n,h}\| \leq \frac{c}{\lambda} h \left[(1 + h\lambda)^n - 1\right], \qquad n = 0, 1, \ldots. \tag{25}$$

Finally, one can show that

$$(1 + h\lambda)^n < e^{nh\lambda} \leq e^{t^*\lambda} \tag{26}$$

so that

$$\|\boldsymbol{e}_{n,h}\| \leq \frac{c}{\lambda} h \left[e^{t^*\lambda} - 1\right], \qquad n = 0, 1, \ldots, N,$$

and

$$\lim_{h\to 0^+} \max_{n=0,1,\ldots,N} \|\boldsymbol{e}_{n,h}\| = \lim_{h\to 0^+} \underbrace{\frac{c}{\lambda}\left[e^{t^*\lambda} - 1\right]}_{\text{const}} h = 0.$$

∎

Remark The error estimate from the proof seems precise. In particular, since one can easily see (using the Peano kernel theorem) that $c = \max_{t\in[t_0,t_0+t^*]} \|\boldsymbol{y}''\|$ works. However, it grossly over-estimates the error in many cases. Thus, it is *useless for practical purposes*.

Remark The order $\mathcal{O}(h)$ convergence of Euler's method is demonstrated in the Matlab script `EulerDemo.m`.

Example Consider the simple linear decay problem $y'(t) = -100y(t)$, $y(0) = 1$ with exact solution $y(t) = e^{-100t}$.

Since $f(t, y) = -100y$, it is clear that f is Lipschitz continuous with $\lambda = 100$ (since $\frac{\partial f}{\partial y} = -100$).

On the other hand, $y''(t) = -100y'(t) = 100^2 y(t)$, so that $c = \max\|y''\| = 100^2 = \lambda^2$.

The error estimate from the proof is of the form

$$|e_{n,h}| \leq \frac{c}{\lambda} h \left[e^{t^*\lambda} - 1\right] = 100h\left[e^{100t^*} - 1\right].$$

If we limit ourselves to the interval $[0, 1]$ then $t^* = 1$ and

$$|e_{n,h}| \leq 100h\left[e^{100} - 1\right] \approx 2.6881 \times 10^{45} h.$$

On the other hand, Euler's method yields

$$\begin{aligned}
y_1 &= y_0 - h100y_0 = (1-100h)y_0 \\
y_2 &= y_1 - h100y_1 = (1-100h)y_1 = (1-100h)^2 y_0 \\
&\vdots \\
y_n &= (1-100h)^n y_0 = (1-100h)^n,
\end{aligned}$$

so that the true error is

$$|y_n - y(\underbrace{t_n}_{nh})| = \left|(1-100h)^n - e^{-100nh}\right| \ll 2.6881 \times 10^{45} h.$$

1.7 Trapezoidal Rule

Recall the derivation of Euler's method via integration:

$$\begin{aligned}
\boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau \\
&\approx \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(t_0, \boldsymbol{y}(t_0))d\tau \\
&= \boldsymbol{y}(t_0) + (t-t_0)\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)).
\end{aligned}$$

FIGURE

Note that this corresponds to the "left endpoint rule" for integration. A simple — but significant — improvement over the left endpoint rule is the trapezoidal rule (for numerical integration), where we use the average of the slopes at the endpoints of the interval.

FIGURE

This leads to an improved method to solve the IVP (21), (22)

$$\begin{aligned}
\boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau \\
&\approx \boldsymbol{y}(t_0) + \int_{t_0}^{t} \frac{\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) + \boldsymbol{f}(t, \boldsymbol{y}(t))}{2} d\tau \\
&= \boldsymbol{y}(t_0) + \frac{1}{2}(t-t_0)\left[\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) + \boldsymbol{f}(t, \boldsymbol{y}(t))\right].
\end{aligned}$$

This calculation motivates the *trapezoidal rule* (for IVPs):

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right]. \tag{27}$$

Remark The major difference between the trapezoidal rule and the Taylor/Euler methods studied earlier lies in the appearance of the "new" value of the approximate solution, $\boldsymbol{y}_{n+1}$, on *both* sides of the formula (27). This means that $\boldsymbol{y}_{n+1}$ is given only *implicitly* by equation (27), and therefore the trapezoidal rule is referred to as an *implicit* method. We will discuss one possible implementation of the trapezoidal rule later. Methods for which $\boldsymbol{y}_{n+1}$ appears only on the left-hand side of the formula are known as *explicit* methods.

The local truncation error for the trapezoidal rule can be derived by substituting the exact solution into the approximation formula (27). This leads to

$$\boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}(t_n) + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right]. \qquad (28)$$

To determine the approximation error in this formula we first rearrange (28) and use the ODE (21) to replace the terms involving $\boldsymbol{f}$ by first derivatives $\boldsymbol{y}'$, i.e.,

$$\begin{aligned}&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right] \\ &\quad = \boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left[\boldsymbol{y}'(t_n) + \boldsymbol{y}'(t_{n+1})\right].\end{aligned}$$

Next, we replace the terms involving t_{n+1} by Taylor expansions about t_n. This leads to

$$\begin{aligned}&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left\{\boldsymbol{y}'(t_n) + \boldsymbol{y}'(t_{n+1})\right\} \\ &= \left[\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{h^2}{2}\boldsymbol{y}''(t_n) + \mathcal{O}(h^3)\right] - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left\{\boldsymbol{y}'(t_n) + \left[\boldsymbol{y}'(t_n) + h\boldsymbol{y}''(t_n) + \mathcal{O}(h^2)\right]\right\} \\ &= \mathcal{O}(h^3),\end{aligned}$$

so that the local truncation error of the trapezoidal rule is of order $\mathcal{O}(h^3)$. To see that the trapezoidal rule is globally a second-order method we need to establish its convergence.

Theorem 1.22 *The trapezoidal rule (27) is convergent.*

Proof Similar to the proof of convergence for Euler's method. See Iserles book. ∎

As mentioned earlier, the trapezoidal rule is an implicit method, and therefore, additional computational effort is required to determine the approximate solution $\boldsymbol{y}_{n+1}$ at time t_{n+1}. There are various approaches to doing this.

1. One possibility is to use a *predictor-corrector* approach. Here an explicit method (such as Euler's method) is used to *predict* a preliminary value $\tilde{\boldsymbol{y}}_{n+1}$ for $\boldsymbol{y}_{n+1}$, and the trapezoidal rule is then used in the (explicit) form

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \tilde{\boldsymbol{y}}_{n+1})\right].$$

   We will study this general approach more carefully in the context of *multistep methods* later.

2. Another approach is to use *fixed-point iteration* to compute $\boldsymbol{y}_{n+1}$. Since the trapezoidal rule is given by

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right],$$

   and $\boldsymbol{f}$ can be a rather general (in particular nonlinear) function, the problem of finding $\boldsymbol{y}_{n+1}$ can be rephrased as a problem of finding a root of the (system of) nonlinear equation(s)

$$G(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{z} = \boldsymbol{0}.$$

In our case

$$\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \frac{h}{2}\boldsymbol{f}(t_{n+1}, \boldsymbol{z}).$$

Many techniques exist for solving such nonlinear equations such as *Newton* or *Newton-Raphson* iteration. The simplest approach is to use *functional iteration*

$$\boldsymbol{z}^{[k+1]} = \boldsymbol{g}(\boldsymbol{z}^{[k]}), \qquad k = 0, 1, 2, \ldots$$

with a good initial value $\boldsymbol{z}^{[0]}$. The famous *Banach fixed-point theorem* guarantees convergence of this approach provided the norm of the Jacobian of $\boldsymbol{g}$ is small enough, i.e.,

$$\|\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}\| < 1.$$

As a slightly weaker requirement, Lipschitz continuity of $\boldsymbol{g}$ is sufficient. Since here

$$\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} = \frac{h}{2}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}$$

we see that — depending on the function $\boldsymbol{f}$ — the stepsize h has to be chosen small enough.

Remark The specific predictor-corrector scheme suggested in 1. above (i.e., Euler's method as a predictor for the trapezoidal rule) is in fact a popular numerical IVP solver in its own right. It is known under many different names such as the *classical second-order Runge-Kutta method*, the *improved Euler method*, or *Heun's method*.

Remark An implementation of the fixed-point iteration approach for the trapezoidal rule is given in the Matlab function `Trapezoid.m`. As initial guess $\boldsymbol{z}^{[0]} = \boldsymbol{y}_{n+1}^{[0]}$ we use the most recent approximate solution from the previous time step $\boldsymbol{y}_n$. Pseudocode for the algorithm is
Algorithm

```
    Input $t_0$, $y_0$, $f$, $h$, $N$

    $t = t_0$, $y = y_0$, $w = y$

    for $n = 1$ to $N$ do

        $f_1 = f(t, w)$
        $t = t + h$
        for $k = 1, 2, \ldots$ do
            $f_2 = f(t, w)$
            $w = y + \frac{h}{2}(f_1 + f_2)$
        end
        $y = w$

    end
```

Remark The order $\mathcal{O}(h^2)$ convergence of the trapezoidal rule is demonstrated in the Matlab script `TrapezoidDemo.m`.

Example The problem

$$y'(t) = \ln(3)\left(y(t) - \lfloor y(t) \rfloor - \frac{3}{2}\right), \qquad y(0) = 0,$$

can be shown to have the solution

$$y(t) = -n + \frac{1}{2}\left(1 - 3^{t-n}\right), \qquad n \le t \le n+1, \quad n = 0, 1, \ldots.$$

However, since the function f in this example is not Lipschitz continuous we cannot expect our numerical solvers to perform as usual. The Matlab scripts `EulerFailDemo.m` and `TrapezoidFailDemo.m` show that we get about $\mathcal{O}(h^{0.8})$ convergence for *both* methods.

1.8 Theta Methods

Both Euler's method and the trapezoidal rule are included as special cases of the following formula:

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\left[\theta \boldsymbol{f}(t_n \boldsymbol{y}_n) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right], \quad n = 0, 1, \ldots. \tag{29}$$

Euler's method corresponds to the choice $\theta = 1$, and the trapezoidal rule to $\theta = 1/2$. In general, formula (29) for $\theta \in [0, 1]$ is known as *theta method*.

Remark The only explicit theta method is Euler's method ($\theta = 1$), all others are implicit. Moreover, the only second-order method is the trapezoid rule. All others are first-order.

To verify the order claim we determine the local truncation error for the general theta method. As before, we insert the exact solution $\boldsymbol{y}$ into the approximate formula (29). This yields

$$\boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}(t_n) + h\left[\theta \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right]. \tag{30}$$

To determine the approximation error in this formula we proceed analogously to what we did for the trapezoidal rule. First we rearrange (30) and use the ODE (21) to replace the terms involving $\boldsymbol{f}$ by first derivatives $\boldsymbol{y}'$, i.e.,

$$\begin{aligned} &\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h\left[\theta \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right] \\ &\quad = \boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h\left[\theta \boldsymbol{y}'(t_n) + (1-\theta)\boldsymbol{y}'(t_{n+1})\right]. \end{aligned}$$

Next, we replace the terms involving t_{n+1} by Taylor expansions about t_n. This leads to

$$\begin{aligned} &\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h\left\{\theta \boldsymbol{y}'(t_n) + (1-\theta)\boldsymbol{y}'(t_{n+1})\right\} \\ &= \left[\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{h^2}{2}\boldsymbol{y}''(t_n) + \tfrac{h^3}{6}\boldsymbol{y}'''(t_n) + \mathcal{O}(h^4)\right] - \boldsymbol{y}(t_n) \\ &-h\left\{\theta \boldsymbol{y}'(t_n) + (1-\theta)\left[\boldsymbol{y}'(t_n) + h\boldsymbol{y}''(t_n) + \tfrac{h^2}{2}\boldsymbol{y}'''(t_n) + \mathcal{O}(h^3)\right]\right\} \\ &= \left(\theta - \tfrac{1}{2}\right)h^2\boldsymbol{y}''(t_n) + \left(\tfrac{1}{2}\theta - \tfrac{1}{3}\right)h^3\boldsymbol{y}'''(t_n) + \mathcal{O}(h^4), \end{aligned}$$

so that the local truncation error of the general theta method is of order $\mathcal{O}(h^2)$. However, for $\theta = 1/2$ the first term on the right drops out, and we have a local truncation error of order $\mathcal{O}(h^3)$.

Convergence of the general theta method is established in the homework (see Assignment 2).

Remark The choice $\theta = 0$ yields the so-called *backward Euler method* which has particularly nice stability properties, and is often used to solve *stiff* equations. Other choices of θ are used less frequently. We will discuss the backward Euler method later.