Contents

1 Things You Should Know  1
1.1 Notation .............................................. 1
  1.1.1 Vectors and Matrices .......................... 1
  1.1.2 Functions ........................................ 2
1.2 Basic Calculus and Taylor’s Theorem .................. 2
  1.2.1 The Mean Value Theorem ....................... 2
  1.2.2 Taylor’s Theorem ................................. 3
  1.2.3 Other Mean Value Theorems .................... 4
1.3 Initial Value Problems ................................ 5
  1.3.1 Higher order systems ............................ 5
1.4 Exercises ............................................. 7

2 Nonlinear Equations  8
2.1 Fixed Point Iteration .................................. 8
2.2 Newton’s Method for Systems of Nonlinear Equations .... 10
2.3 Nonlinear Least Squares Problems ..................... 10
2.4 Notes .................................................. 10
2.5 Exercises ............................................. 10

3 Interpolation Theory  11
3.1 Polynomial Interpolation ............................... 11
3.2 The Error in the Interpolating Polynomial .............. 12
3.3 The Newton Form for the Interpolating Polynomial .... 12
  3.3.1 Computing the Coefficients of the Newton Form ... 13
3.4 Hermite Interpolation .................................. 15
3.5 Piecewise Polynomial Approximations .................. 17
  3.5.1 Piecewise Linear Splines ....................... 17
  3.5.2 Hermite Splines .................................. 18
  3.5.3 $C^2$ cubic splines .............................. 18
3.6 Error Estimation and Grid Refinement Studies .......... 21
3.7 Exercises ............................................. 23
### 4 Quadrature Rules

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Interpolatory Quadrature Rules</td>
<td>25</td>
</tr>
<tr>
<td>4.1.1</td>
<td>The Midpoint Rule</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>The Trapezoid Rule</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Simpson’s Rule</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Newton-Cotes Rules</td>
<td>29</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Closed Newton-Cotes Rules</td>
<td>29</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Open Newton-Cotes Rules</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Extrapolation and Romberg Integration</td>
<td>31</td>
</tr>
<tr>
<td>4.6</td>
<td>Gaussian Quadratures</td>
<td>32</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Roots of Orthogonal Polynomials</td>
<td>33</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Construction of Orthogonal Polynomials</td>
<td>34</td>
</tr>
<tr>
<td>4.7</td>
<td>Error Estimation</td>
<td>34</td>
</tr>
<tr>
<td>4.8</td>
<td>Exercises</td>
<td>34</td>
</tr>
</tbody>
</table>

### 5 Initial Value Problems

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Existence and Uniqueness</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Explicit and Implicit Methods</td>
<td>37</td>
</tr>
<tr>
<td>5.3</td>
<td>One Step methods</td>
<td>38</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Convergence and 0-stability</td>
<td>39</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Asymptotic Stability of One-step Methods</td>
<td>40</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Examples of one-step methods</td>
<td>40</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Explicit Runge-Kutta Methods</td>
<td>42</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Stepsize Control and Error Estimation for One-Step Methods</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>Linear Multistep Methods</td>
<td>46</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Stability of Linear Multistep Methods</td>
<td>46</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Adams Methods</td>
<td>47</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Predictor-Corrector Methods</td>
<td>49</td>
</tr>
<tr>
<td>5.4.4</td>
<td>BDF Methods</td>
<td>50</td>
</tr>
<tr>
<td>5.5</td>
<td>Using Codes</td>
<td>51</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Input</td>
<td>51</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Output</td>
<td>51</td>
</tr>
<tr>
<td>5.5.3</td>
<td>A third order IVP</td>
<td>52</td>
</tr>
<tr>
<td>5.6</td>
<td>Notes</td>
<td>53</td>
</tr>
<tr>
<td>5.7</td>
<td>Exercises</td>
<td>53</td>
</tr>
</tbody>
</table>
Preface

These notes are a work-in-progress. Please report errors directly to me at
timKelley@ncsu.edu

These notes are designed for MA780. I assume that the reader has recently completed MA580, using a book such as [4, 15]. I will use ideas from graduate-level numerical linear algebra under the assumption that the reader is in command of that material.

In addition to a first semester course in numerical analysis, at the level of [4], say, you’ll need a good command of elementary calculus and differential equations to understand most of the material in this course. §1.2 and §1.3 are short reviews of those things, but you’ll almost certainly need to consult a calculus or differential equations text at some point during the course.

You should have good programming skills. MA780 uses the MATLAB\textsuperscript{1} environment, which should be easy to learn if you have good programming skills in FORTRAN, C, or C++. MATLAB has good on-line help and [7, 14] are good introductory books.

\textsuperscript{1}Matlab is a registered trademark of The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760, USA, tel. 508-647-7001, info@mathworks.com, http://www.mathworks.com.
Chapter 1

Things You Should Know

1.1 Notation

1.1.1 Vectors and Matrices

We let $\mathbb{R}^N$ denote the space of $N$-dimensional vectors. Vectors will be denoted by bold faced lower case letters and matrices will be denoted by bold faced capital letters. We will write all vectors as column vectors. If $\mathbf{x}$ is a vector, we will denote its $i$th component by $x_i$. So

$$
\mathbf{x} = \begin{pmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_N
\end{pmatrix}.
$$

We will denote matrices with upper case letters and let $\mathbb{R}^{M \times N}$ be the space of $M \times N$ matrices. If $\mathbf{A} \in \mathbb{R}^{M \times N}$ we will let $a_{i,j}$ be the $ij$th entry. We will sometimes omit the comma in the subscript when on confusion will result. The transpose of $\mathbf{A}$, $\mathbf{A}^T$ is the the $m \times n$ matrix with entries $a_{ji}$. Consistent with this notation, we may express column vectors by

$$
\mathbf{x} = (x_1, \ldots, x_n)^T.
$$

That is a column vector is the transpose of a row vector.

$\mathbb{R}^1 = \mathbb{R}^{1 \times 1} = \mathbb{R}$ will denote the real numbers.

We will use the common $l^p$, $(1 \leq p < \infty)$ and $l^\infty$ norms for vectors

$$
\|\mathbf{x}\|_p = \left( \sum_{i=1}^N |x_i|^p \right)^{1/p}
$$

and

$$
\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq N} |x_i|.
$$

The $l^2$ norm is the standard Euclidean norm.
1.1.2 Functions

Let \( \Omega \subset \mathbb{R}^N \). Let \( C(\Omega) = C^0(\Omega) \) be the space of continuous functions and \( C^k(\Omega) \) be the space of \( k \) times continuously differentiable functions on \( \Omega \). In the special case where \( I = [a, b] \) is an interval, we will omit the parentheses and write, for example, \( C[a, b] \).

We will use several different norms to measure the size of functions. The reader who does not know measure theory can assume that all functions are continuous. If \( f \) is a measurable function on a set \( \Omega \subset \mathbb{R}^N \), define the \( L^p(\Omega) \) norm for \( 1 \leq p < \infty \) as

\[
\|f\|_p = \left( \int_{\Omega} |f(x)|^p \, dx \right)^{1/p}.
\]

The \( L^\infty \) norm is

\[
\|f\|_\infty = \text{esssup}_{x \in \Omega} |f(x)|.
\]

If \( \Omega \) is closed and \( f \in C(\Omega) \), then

\[
\|f\|_\infty = \max_{x \in \Omega} |f(x)|.
\]

Similarly, \( C^k(\Omega : \mathbb{R}^N) \) and \( L^p(\Omega : \mathbb{R}^N) \) are spaces of \( \mathbb{R}^N \)-valued functions on \( \Omega \). The norms are

\[
\|f\|_p = \left( \int_{\Omega} \|f(x)\|^p \, dx \right)^{1/p},
\]

for \( L^p(\Omega : \mathbb{R}^N) \), and

\[
\|f\|_\infty = \max_{x \in \Omega} \|f(x)\|,
\]

for \( C(\Omega : \mathbb{R}^N) \). One can use any norm \( \| \cdot \| \) on \( \mathbb{R}^N \), and we will use the Euclidean norm, unless we state otherwise.

1.2 Basic Calculus and Taylor’s Theorem

1.2.1 The Mean Value Theorem

Most calculus texts state several mean value theorems near the beginning. Some of these theorems are used heavily in this course, and we will state and prove the most important.

**Theorem 1.2.1 (Intermediate Value Theorem)** Let \( f \in C[a, b] \) and let \( z \) be between \( f(a) \) and \( f(b) \). Then there is \( \xi \in [a, b] \) such that \( z = f(\xi) \).

Rolle’s theorem is the first application one sees of the intermediate value theorem.

**Theorem 1.2.2 (Rolle’s Theorem)** Let \( f \in C[a, b] \) be differentiable in \( (a, b) \). Assume that

\[
f(a) = f(b) = 0. \tag{1.1}
\]

Then there is \( \xi \in (a, b) \) such that \( f'(\xi) = 0 \).
Proof. Either $f$ is constant on $[a, b]$ or $f$ has a maximum or a minimum at $\xi \in (a, b)$. In the former case, $f'(x) = 0$ for all $x \in [a, b]$. In the latter case $f'(\xi) = 0$. □

The mean value theorem follows directly from the Rolle’s theorem.

**Theorem 1.2.3 (Mean Value Theorem)** Let $f \in C[a, b]$ be differentiable in $(a, b)$. Then there is $\xi \in (a, b)$ such that

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}.$$ 

**Proof.** The proof, as do many others in the course, constructs an auxiliary function and then applies a previous result, in this case the intermediate value theorem, to complete the proof.

Consider the function

$$F(z) = f(z) - f(a) - \left( \frac{f(b) - f(a)}{b - a} \right) (z - a).$$

Clearly $F(a) = F(b) = 0$, so Rolle’s theorem implies that there is $\xi \in (a, b)$ such that

$$F'(\xi) = f'(\xi) - \frac{f(b) - f(a)}{b - a} = 0,$$

as asserted. □

### 1.2.2 Taylor’s Theorem

Taylor’s theorem says that if you know $n + 1$ values of sufficiently smooth function $f$ and its derivatives, in this case the first $n + 1$ derivatives at a point $x_0$, then you can construct a polynomial approximation to $f$ that is $O(h^{n+1})$ accurate in the interval $(x_0 - h, x_0 + h)$ for sufficiently small $h$.

Chapter 3 contains other results of this type.

**Theorem 1.2.4 (Taylor’s Theorem)** Let $f \in C^n[a, b]$ and let $f^{(n+1)}$ exist in $(a, b)$. Then for all $x$ in $[a, b]$ there is $\xi$ such that

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!} (x - a)^k + \frac{f^{(n+1)}(\xi)}{(n + 1)!} (x - a)^{n+1}. \quad (1.2)$$

**Proof.** Fix $x \in [a, b]$. Define $C$ by

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!} (x - a)^k + C(x - a)^{n+1}.$$ 

The plan of the proof is to compute $C$.

Define an auxiliary function

$$F(z) = f(z) - \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!} (z - a)^k - C(z - a)^{n+1}.$$
Clearly \( F(a) = 0 \), \( F(x) = 0 \) by the definition of \( C \). Rolle’s theorem then implies that there is \( \xi_1 \in (a, x) \) such that \( F'(\xi_1) = 0 \).

However, \( F'(a) = 0 \), so there is \( \xi_2 \in (a, \xi_1) \) such that \( F''(\xi_2) = 0 \). We can continue this process for all derivatives up to \( n + 1 \), and conclude that there is \( \xi \) such that

\[
F^{(n+1)}(\xi) = f^{(n+1)}(\xi) - C(n + 1)! = 0
\]

(1.3)

Solving (1.3) for \( C \) completes the proof. \( \square \)

The polynomial expansion of \( f \) in (1.2) is called the **Taylor polynomial** at \( x \) about \( a \). The difference between \( f \) and the Taylor polynomial is called the **remainder**.

One should be aware that the point \( \xi \) depends on both \( a \), the center of the expansion, and \( x \), the point at which the Taylor polynomial is evaluated. One must take care when using the error in Taylor’s theorem in a proof for this reason.

We will restate Taylor’s theorem in a way that makes it easy to compare it to the results in Chapter 3.

**Theorem 1.2.5 (Taylor’s Theorem)** Let \( x_0 \) and \( h > 0 \) be given and let \( f \in C^n[x_0 - h, x_0 + h] \). Let \( f^{(n+1)} \) exist in \( (x_0 - h, x_0 + h) \). Then for all \( x \in [x_0 - h, x_0 + h] \) there is \( \xi \) such that

\[
f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + \frac{f^{(n+1)}(\xi)}{(n + 1)!} (x - x_0)^{n+1}.
\]

(1.4)

Theorem 1.2.5 says that the remainder is \( O(h^{n+1}) \).

### 1.2.3 Other Mean Value Theorems

**Theorem 1.2.6 (Integral Mean Value Theorem)** Let \( f \in C[a, b] \) and let \( g \geq 0 \) be defined on \( [a, b] \). Then there is \( \xi \in (a, b) \) such that

\[
f(\xi) \int_a^b g(x) \, dx = \int_a^b f(x)g(x) \, dx.
\]

(1.5)

**Proof.** By this point you should not be surprised to see an auxiliary function

\[
F(z) = f(z) \int_a^b g(x) \, dx - \int_a^b f(x)g(x) \, dx.
\]

Since \( f \) is continuous, \( f \) attains both its maximum and minimum. So there are \( x_{\text{max}} \) and \( x_{\text{min}} \) such that

\[
f(x_{\text{min}}) \leq f(x) \leq f(x_{\text{max}}) \text{ for all } x \in [a, b].
\]

Therefore, since \( g \geq 0 \),

\[
F(x_{\text{min}}) = \int_a^b (f(x_{\text{min}}) - f(x))g(x) \, dx \leq 0
\]

and

\[
F(x_{\text{max}}) = \int_a^b (f(x_{\text{max}}) - f(x))g(x) \, dx \geq 0.
\]

Hence, the intermediate value theorem implies that there is \( \xi \) such that \( F(\xi) = 0 \), proving (1.5). \( \square \)
Theorem 1.2.7 (Summation Mean Value Theorem) Let $f \in C[a, b]$, let $g \geq 0$ be defined on $[a, b]$, and let $\{x_i\}_{i=1}^n \subset [a, b]$. Then there is $\xi \in (a, b)$ such that

$$f(\xi) \sum_{i=1}^n g(x_i) = \sum_{i=1}^n f(x_i)g(x_i).$$

(1.6)

1.3 Initial Value Problems

All undergraduate differential equations courses cover initial value problems. We will formulate all such problems as systems of first order initial value problems. A first order initial value problem consists of a first order system of differential equations

$$y'(t) = f(t, y(t)), \quad (1.7)$$

and an initial condition

$$y(t_0) = y_0. \quad (1.8)$$

The unknown $y$ is a $\mathbb{R}^M$-valued function of $t$. $y'$, and denotes differentiation with respect to $t$. We will make the simplifying assumption that $f$ is uniformly Lipschitz continuous in $[t_0, \infty) \times \mathbb{R}^N$. By this we mean that $f$ is continuous and that there is $L$, called the Lipschitz constant of $f$, such that

$$||f(t, y_1) - f(t, y_2)|| \leq L||y_1 - y_2||, \quad (1.9)$$

for all $t \in [t_0, \infty)$ and $y_1, y_2 \in \mathbb{R}^M$. In (1.9) the norm, unless explicitly stated otherwise, will be the Euclidean norm.

Many of the essential ideas in this book can be expressed in terms of scalar equations ($M = 1$). We will do that when we can, and avoid vector notation in the analysis. We will return to vectors in examples.

1.3.1 Higher order systems

You should understand the relationship between high-order initial value problems and first order systems of initial value problems. The reason for this is that algorithms for initial value problems are expressed in terms of first-order systems and you must be able to perform the conversion.

In general terms if

$$w^{(n)} = f(t, w, w', \ldots, w^{(k-1)})$$

$$w(t_0) = w_0, w'(t_0) = w'_0, w''(t_0) = w''_0, \ldots, w^{(k-1)}(0) = w^{(k-1)}_0$$

(1.10)

is a $k$th-order initial value problem for a scalar function $w \in C^k$, and

$$y = (y_1, \ldots, y_k)^T = (w, w', w'', \ldots, w^{(k-1)})^T$$

is the column vector whose elements are $w$ and its first $k - 1$ derivatives, then

$$y' = F(t, y), y(t_0) = y_0, \quad (1.12)$$

5
where
\[ F(t, y) = (y_2, y_3, \ldots, y_{k-1}, f(t, y_1, \ldots, y_{k-1}))^T \]  
(1.13)

and
\[ y_0 = (w_0, w_0', \ldots, w_0^{(k-1)})^T. \]

For example, the first order system for
\[ w''' + w' + tw = e^t, w'''(0) = 1, w'(0) = 0, w(0) = 3 \]
is
\[ y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w \\ w' \\ w'' \end{pmatrix}, \quad F(t, y) = \begin{pmatrix} y_2 \\ y_3 \\ e^t - y_2 - ty_1 \end{pmatrix}, \quad \text{and} \quad y_0 = \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix}. \]
1.4 Exercises

1.4.1. Prove the intermediate value theorem.

1.4.2. Prove Theorem 1.2.7.

1.4.3. Convert the 3rd order initial value problem

\[ y''' + 2ty' + y^2 = e^t, \quad y(0) = 1, \quad y'(0) = 0, \quad y''(0) = 1, \]

into a first order system.
Chapter 2

Nonlinear Equations

Nonlinear solvers (both human and algorithmic) seek solutions of systems of nonlinear equations. We express these equations as

\[ F(x) = 0 \]  

(2.1)

In (2.1),

\[ F = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{pmatrix} : \mathbb{R}^N \rightarrow \mathbb{R}^N. \]

We will refer to any solution \( x^* \) of (2.1) as a root of \( f \).

Solutions of nonlinear equations need not be unique. Quadratic polynomials, for example, generally have two solutions. Algorithms for nonlinear equations will, when working well, find a root, which may not be the one you want.

Nonlinear solvers are almost always iterative. An iterative method produces a sequence of approximations of a solution. You should be familiar with iterative methods, such as GMRES and CG, for linear equations.

2.1 Fixed Point Iteration

Many nonlinear equations are naturally formulated as fixed point problems

\[ x = K(x) \]  

(2.2)

where \( K \), the fixed point map, may be nonlinear. A solution \( x^* \) of (2.2) is called a fixed point of the map \( K \).

In this section we analyze convergence of fixed point iteration,

\[ x_{n+1} = K(x_n). \]  

(2.3)

Fixed point iteration is also called nonlinear Richardson iteration, Picard iteration, or the method of successive substitution.

Before discussing convergence of fixed point iteration we make two definitions.
Definition 2.1.1 Let $\Omega \subset \mathbb{R}^N$ and let $G: \Omega \to \mathbb{R}^M$. $G$ is Lipschitz continuous on $\Omega$ with Lipschitz constant $L_G$ if 
$$\|G(x) - G(y)\| \leq L_G \|x - y\|$$
for all $x, y \in \Omega$.

Definition 2.1.2 Let $\Omega \subset \mathbb{R}^N$. $K: \Omega \to \mathbb{R}^N$ is a contraction mapping on $\Omega$ if $K$ is Lipschitz continuous on $\Omega$ with Lipschitz constant $L_K < 1$.

The standard result for fixed point iteration is the Contraction Mapping Theorem [2].

Theorem 2.1.3 Let $\Omega$ be a closed subset of $\mathbb{R}^N$ and let $K$ be a contraction mapping on $\Omega$ with Lipschitz constant $L_K < 1$ such that $K(x) \in \Omega$ for all $x \in \Omega$. Then there is a unique fixed point of $K$, $x^* \in \Omega$, and, for all $x_0 \in \Omega$, the iteration defined by (2.3) converges $x^*$. Moreover

$$\|x_{n+1} - x^*\| \leq L_K \|x_n - x^*\|. \quad (2.4)$$

Proof. The important part of the proof is a demonstration that the sequence of iterates $\{x_n\}$ is a Cauchy sequence, i.e.

$$\lim_{n,k \to \infty} \|x_{n+k} - x_n\| = 0.$$ 

Cauchy sequences in $\mathbb{R}^N$ converge [11], and hence the fixed point iteration will converge.

Let $x_0 \in \Omega$. Note that $\{x_n\} \subset \Omega$ because $x_0 \in \Omega$ and $K(x) \in \Omega$ whenever $x \in \Omega$. So the sequence of iterates $\{x_n\}$ remains bounded since for all $n \geq 1$

$$\|x_{i+1} - x_i\| = \|K(x_i) - K(x_{i-1})\| \leq L_K \|x_i - x_{i-1}\| \leq \sum_{i=0}^{n-1} L_K \|x_1 - x_0\|,$$

and therefore

$$\|x_n - x_0\| = \|\sum_{i=0}^{n-1} x_{i+1} - x_i\|$$

$$\leq \sum_{i=0}^{n-1} \|x_{i+1} - x_i\| \leq \|x_1 - x_0\| \sum_{i=0}^{n-1} L_K^i$$

$$\leq L_K \|x_1 - x_0\|/(1 - L_K).$$

So, for all $n, k \geq 0$,

$$\|x_{n+k} - x_n\| = \|K(x_{n+k-1}) - K(x_{n-1})\|$$

$$\leq L_K \|x_{n+k-1} - x_{n-1}\|$$

$$\leq L_K \|K(x_{n+k-2}) - K(x_{n-2})\|$$

$$\leq L_K^2 \|x_{n+k-2} - x_{n-2}\| \leq \ldots \leq L_K^n \|x_k - x_0\|$$

$$\leq L_K^n \|x_1 - x_0\|/(1 - L_K).$$
Hence
\[\lim_{n,k \to \infty} \|x_{n+k} - x_n\| = 0\]
and therefore the sequence \(\{x_n\}\) is a **Cauchy sequence**.

Having proved convergence, proving uniqueness and the convergence estimate (2.4) is easy. If there are two solutions \(x^*\) and \(y^*\), then
\[\|x^* - y^*\| = \|K(x^*) - K(y^*)\| \leq L_K \|x^* - y^*\|\]
which is impossible if \(x^* \neq y^*\), because \(L_K < 1\). Now, since \(x_{n+1} = K(x_n)\) and \(x^* = K(x^*)\),
\[\|x_{n+1} - x^*\| = \|K(x_n) - K(x^*)\| \leq L_K \|x_n - x^*\|,\]
which proves (2.4).

\[\square\]

### 2.2 Newton’s Method for Systems of Nonlinear Equations

### 2.3 Nonlinear Least Squares Problems

### 2.4 Notes

Three books [5, 9, 10] on nonlinear solvers to into far more depth than this chapter.

Cite Peter’s new book.

### 2.5 Exercises

2.5.1. A numerical analyst writes a best-selling textbook, and buys an Italian sports car with the royalties. He puts $50,000 down and borrows the remaining $85,000. If the monthly payments are $2202.32, what is the interest rate?
Chapter 3

Interpolation Theory

In this chapter we describe various strategies for interpolation.

**Definition 3.0.1** A function \( p(x) \) **interpolates** the data \( \{x_i, y_i\}_{i=0}^{n} \) if
\[
p(x_i) = y_i.
\]
The points \( \{x_i\} \) are called the **nodes** or **abscissae** of the interpolation and the points \( \{y_i\} \) the **nodal values**. If \( y_i = f(x_i) \) for some function \( f \), we say \( p \) **interpolates** \( f \) at the **nodes**.

### 3.1 Polynomial Interpolation

The simplest form of interpolation is to use a polynomial to interpolate the data. If there are \( n + 1 \) data pairs \( (x_i, y_i) \), there is at most enough data to create a polynomial of degree \( n \), which has \( n + 1 \) coefficients. If the nodes are distinct \( (x_i \neq x_j \text{ for } i \neq j) \), then one can construct the interpolating polynomial directly.

We let \( P_n \) be the space of \( n \)th degree polynomials.

**Lemma 3.1.1** Let the nodes \( \{x_i\} \) be distinct. Then there is a unique polynomial \( L_n \) of degree \( n \) which interpolates the nodes.

**Proof.** Define the sequence of **Lagrange polynomials** \( \{l_i\}_{i=0}^{n-1} \) by
\[
l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.
\]
Clearly \( l_i \in P_n \) and \( l_i(x_j) = \delta_{ij} \). Therefore
\[
L_n(x) = \sum_{i=0}^{n} y_i l_i(x) \in P_n
\]
interpolates the data.

If \( q \in P_n \) interpolates the data, then \( L_n - q \) is a polynomial of degree \( n \) with \( n + 1 \) roots. The fundamental theorem of algebra implies that \( L_n = q \), and hence \( L_n \) is unique. \( \square \)
3.2 The Error in the Interpolating Polynomial

**Theorem 3.2.1** Let $f \in C^{(n)}[a, b]$ and assume that $f^{(n+1)}$ exists in $[a, b]$. Let the nodes $\{x_i\}_{i=0}^n \subset [a, b]$ be distinct and let $L_n$ be the polynomial interpolant of $f$ at the nodes. Then for each $x \in [a, b]$ there is $\xi_x \in (a, b)$ such that

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

(3.3)

**Proof.** Let $x \in [a, b]$ be given. If $x$ is one of the nodes, then (3.3) trivially holds. If $x$ is not a node, we may define

$$E(x) = f(x) - L_n(x), \quad \Omega(x) = \prod_{i=0}^n (x - x_i),$$

and the auxiliary function

$$G(z) = f(z) - L_n(z) - \frac{E(x)\Omega(z)}{\Omega(x)}.$$

$G$ has $n + 2$ distinct zeros, namely $x$ and the nodes. We may therefore apply Rolle’s theorem repeatedly to conclude that $G^{(n+1)}$ has a zero $\xi_x \in (a, b)$.

$L_n$ has degree $n$, so $L_n^{(n+1)} = 0$, hence

$$G^{(n+1)}(\xi_x) = f^{(n+1)}(\xi) - E(x)(n+1)!/\Omega(x).$$

Solving for $E(x)$ completes the proof. □

As was the case with the error in Taylor’s theorem, the point $\xi$ depends on $x$, and one must take care when several interpolation errors at different points must be considered in an application.

3.3 The Newton Form for the Interpolating Polynomial

The Newton form of the interpolating polynomial is motivated by Horner’s method for evaluation of a general polynomial

$$p(z) = \sum_{i=0}^n a_i z^i$$

by the recursion

$$q_0 = a_n; \quad q_1 = q_0 z + a_{n-1}; \quad \ldots \quad q_k = q_{k-1} z + a_{n-k} \ldots p = q_n = q_{n-1} z + a_0.$$

(3.4)

This method of evaluation of $p(z)$ requires only $n$ adds and $n$ multiplies for $z \in R$.

A similar form for the interpolating polynomial would be

$$p(z) = \sum_{i=0}^n a_i \prod_{j=0}^{i-1} (z - x_i).$$

(3.5)
Horner’s method then becomes

\[ q_0 = a_n; q_1 = q_0(z - x_{n-1}) + a_{n-1}; \ldots \]
\[ \ldots q_k = q_{k-1}(z - x_{n-k}) + a_{n-k} \ldots p = q_n = q_{n-1}(z - x_0) + a_0. \]  

(3.6)

The cost of evaluation of the interpolating polynomial at \( z \in R \) is exactly \( n \) adds and \( n \) multiplies, **once the coefficients \( a_n \) have been computed.** Contrast this with the cost of evaluation of the Lagrange form. Evaluation of \( y_i * l_i(z) \) requires \( O(n) \) adds and multiplies. The Lagrange form sums \( n + 1 \) terms which cost \( O(n) \) work to evaluate, for a total cost of \( O(n^2) \) work. If one wants to evaluate the polynomial at many points (so \( z \in R^N \) is a vector), one must then do \( O(N n^2) \) work for the entire evaluation.

Contrast this with the Newton form. One must compute the coefficients, which we will show is \( O(n^2) \) work in \( \S \) 3.3.1. But then the evaluation for a scalar \( z \) is \( O(n) \) work. So if \( z \in R^N \), the cost of evaluation of \( p(z) \) is \( O(nN + n^2) \) floating point operations.

### 3.3.1 Computing the Coefficients of the Newton Form

**Theorem 3.3.1** The coefficients of the Newton form are the divided differences

\[ a_i = f[x_0, \ldots, x_i]. \]

Here

\[ f[x_0] = f(x_0), \]  

(3.7)

and, for \( n \geq 1 \), The \( n \)th divided difference is

\[ f[x_0, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}. \]  

(3.8)

**Proof.** The coefficients of the Newton form depend only on the order of the nodes and the \( i \)th coefficient depends only on \( \{x_j\}_{j=0}^i \). Therefore the notation

\[ a_i = f[x_0, \ldots, x_i], \]

is adequate.

By definition of the Newton form

\[ L_n(x) = L_{n-1}(x) + f[x_0, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i). \]  

(3.9)

Let

\[ Q_n(x) = \sum_{i=1}^{n} f[x_1, \ldots, x_i] \prod_{j=1}^{i-1} (x - x_j), \]

be the polynomial of degree \( n - 1 \) that interpolates \( f \) at \( x_1, \ldots, x_n \). Clearly

\[ Z(x) = \frac{(Q_n(x) - L_{n-1}(x))(x - x_0)}{x_n - x_0}. \]
is an \( n \)th degree polynomial that vanishes at \( \{ x_i \}_{i=0}^{n-1} \). Moreover

\[
Z(x_n) = Q_n(x_n) - L_{n-1}(x_n) = f(x_n) - L_{n-1}(x_n).
\]

So, \( Z \) and \( L_n - L_{n-1} \) agree at \( \{ x_i \}_{i=0}^{n-1} \), and therefore

\[
Z(x) = \frac{(Q_n(x) - L_{n-1}(x))(x - x_0)}{x_n - x_0} = a_n \prod_{i=0}^{n-1} (x - x_i).
\]

This means that the coefficients of \( x^n \) in the two polynomials must be the same, i.e.

\[
f[x_0, \ldots , x_n] = \frac{f[x_1, \ldots , x_n] - f[x_0, \ldots , x_{n-1}]}{x_n - x_0},
\]

as asserted. \( \square \)

The best way to organize the computation of the coefficients is to mimic the construction of the divided difference table. The table is the triangular array

\[
\begin{array}{cccccc}
  & f[x_0] & & & & \\
  f[x_1] & f[x_0, x_1] & & & & \\
  f[x_2] & f[x_1, x_2] & f[x_0, x_1, x_2] & & & \\
  f[x_3] & f[x_2, x_3] & f[x_1, x_2, x_3] & f[x_0, x_1, x_2, x_3] & & \\
    & \vdots & \ddots & \ddots & \ddots & \ddots \\
\end{array}
\]

(3.10)

The entries in (3.10) are computed using (3.8), with \( x_0 \) and \( x_n \) replaced by the nodes for the specific row and column. Specifically, for \( 0 \leq j \leq i \leq n \), we have

\[
f[x_i, x_i] = f[x_i] = f(x_i)
\]

and, for \( j < i \),

\[
f[x_j, \ldots , x_i] = \frac{f[x_{j+1}, \ldots , x_i] - f[x_j, \ldots , x_{i-1}]}{x_i - x_j}.
\]

(3.11)

The coefficients \( \{ a_i \} \) are the diagonal elements of the array. The computation of the array proceeds down the rows and then across the columns. For example, to compute \( f[x_1, x_3] \) using formula (3.8),

\[
f[x_1, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1}
\]

you need the \( f[x_1, x_2] \) (previous row and column) and \( f[x_2, x_3] \) (previous column, same row).

One can recover the coefficients of the Newton form by computing the entries in the table using (3.11) by computing the entries of the matrix \( A \), where

\[
A_{ij} = f[x_{i-j}, \ldots , x_i]
\]

and then taking \( a_i = A_{ii} = f[x_0, x_i] \). In terms of \( A \), (3.8) becomes, for \( j > 0 \),

\[
A_{i,j} = \frac{A_{i,j-1} - A_{i-1,j-1}}{x_i - x_{i-j}} \text{ and } A_{i,0} = f(x_i).
\]
newton_coeff(A, x, y)
    A_{0,0} \leftarrow y_0
    \textbf{for} i = 1 \ldots n \textbf{do}
    \hspace{1em} A_{i,0} = y_i
    \textbf{for} j = 1 \ldots i \textbf{do}
    \hspace{2em} A_{i,j} = (A_{i,j-1} - A_{i-1,j-1})/(x_i - x_{i-j})
    \textbf{end for}
    \textbf{end for}

Here is a simple algorithm that computes the entire table from the nodes \(\{x_i\}_{i=0}^n\) and the abscissae \(\{y_i\}_{i=0}^n\).

To evaluate the interpolating polynomial at \(z\) (and \(z\) may be a vector!) you first compute \(a_i = A_{i,i}\) for \(0 \leq i \leq n\) and then compute \(p = L_n(z)\) with Algorithm poly_eval.

poly_eval(z, x, a)
    \(p = a_n\)
    \textbf{for} i = n - 1, \ldots, 0 \textbf{do}
    \hspace{1em} p \leftarrow p(z - x_i) + a_i
    \textbf{end for}

Here’s a simple example. We will compute the quadratic polynomial \(p\) which interpolates the data in Table 3.1.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
x & y & & \\
\hline
0 & 0 & & \\
1 & 2 & 2 & \\
2 & 8 & 6 & 2 \\
\hline
\end{tabular}
\caption{Data for interpolation and difference table}
\end{table}

So
\[ p(x) = (2 + 2(x - 1)) \ast x = 2x^2, \]
which is clearly correct.

### 3.4 Hermite Interpolation

The \textbf{Hermite interpolating polynomial} \(H_n\) of a differentiable function \(f\) is the polynomial of degree \(2n + 1\) that agrees with \(f\) and \(f'\) at the nodes \(\{x_i\}_{i=0}^n\). One can easily verify (and you should) that

\[ H_n(x) = \sum_{i=0}^n (f(x_i)(1 - 2l_i'(x_i)(x - x_i)) + f'(x_i)(x - x_i))l_i^2(x), \quad (3.12) \]
where \( l_i \) is the Lagrange polynomial defined in (3.1).

The error formula and its proof should come as no surprise at this point.

**Theorem 3.4.1** Let \( f \in C^{2n+1} \) and let \( f^{(2n+2)} \) exist in \([a, b]\). Let the nodes \( \{x_i\}_{i=0}^n \subset [a, b] \) be distinct and let \( H_n \) be the Hermite interpolant of \( f \) at the nodes. Then for each \( x \in [a, b] \) there is \( \xi \in (a, b) \) such that

\[
 f(x) = H_n(x) + \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \prod_{i=0}^n (x - x_i)^2. \tag{3.13}
\]

**Proof.** Let \( x \in [a, b] \) be given. If \( x = x_i \) for some \( i \), then (3.13) holds trivially. Otherwise, let

\[
 E(x) = f(x) - H_n(x) \quad \text{and} \quad \Omega(x) = \prod_{i=0}^n (x - x_i)^2.
\]

Once again we define an auxiliary function

\[
 G(z) = f(z) - H_n(z) - \frac{\Omega(z)E(x)}{\Omega(x)}. \tag{3.14}
\]

\( G(z) = 0 \) at the \( n + 2 \) distinct points \( \{x_i\}_{i=0}^n \) and \( x \). By Rolle’s theorem, there are \( \{\xi_i\}_{i=0}^n \) such that \( G' (\xi_i) = 0 \) and \( \xi_i \neq x_j \) for any \( i, j \). This means that \( G' \) has \( 2n + 2 \) distinct roots. Repeated application of Rolle’s theorem implies that there is \( \xi \in (a, b) \) such that

\[
 G^{(2n+2)}(\xi) = f^{(2n+2)}(\xi) - \frac{\Omega^{(2n+2)}(\xi)E(x)}{\Omega(x)} = 0.
\]

Since \( \Omega^{(2n+2)} = (2n + 2)! \), the proof is complete. \( \square \)

There’s a pattern here. For any of Taylor’s theorem, Lagrange interpolation, and Hermite interpolation, if you know \( N \) values of a function and its derivatives an an array of points, you can use those data to construct a polynomial of degree \( N - 1 \) and the error is the product of a \( N \)th degree polynomial and

\[
 \frac{f^{(N)}(\xi)}{N!}.
\]

The proofs all follow the same pattern. One builds an auxiliary function and uses Rolle’s theorem to argue that the \( N + 1 \)st derivative of that function vanishes at some point \( \xi \in [a, b] \).

**Theorem 3.4.2** is a general interpolation theorem, which includes all the others as special cases.

**Theorem 3.4.2** Let \( \{x_i\}_{i=0}^n \) be distinct points in \([a, b]\). Let nonnegative integers \( \{m_i\}_{i=0}^n \) be given and let

\[
 N = \sum_{i=0}^n (m_i + 1).
\]

Let \( f \in C^{(N-1)}[a, b] \) and assume that \( f^{(N)} \) exists in \((a, b)\). Then there is a unique polynomial \( B \) of degree \( N - 1 \) such that

\[
 B^{(m)}(x_i) = f^{(m)}(x_i) \quad \text{for all} \quad 0 \leq m \leq m_i \quad \text{and} \quad 0 \leq i \leq n.
\]
Moreover, there is $\xi \in (a, b)$ such that

\[ f(x) = B(x) + \frac{f^{(N)}(\xi)}{N!} \prod_{i=0}^{n} (x - x_i)^{m_i+1}. \]

For Taylor’s theorem, there is only one point, so $n = 0$ in Theorem 3.4.2, and $m_0$ is the order of the expansion, playing the role of $n$ in Theorem 1.2.4. To recover Lagrange interpolation, set $m_i = 0$ for all $i$ in Theorem 3.4.2. For Hermite interpolation, set $m_i = 1$ for all $i$.

### 3.5 Piecewise Polynomial Approximations

In general, polynomial interpolation is useful and accurate if $b - a$ is small and the degree of the polynomial is low. If one has many nodes, a better strategy is to interpolate low degree polynomials over small groups of nodes. The resulting piecewise polynomial approximations are called Splines of degree $k$ if they are polynomials in the intervals between the nodes and in $C^k[a, b]$.

Splines solve the problem that the interpolating polynomial may not converge to $f$ as the number of nodes increases. By limiting the degree, one obtains convergence because the size of the intervals over which the error estimates are used is decreasing.

In this section we set $a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b$. The length of the $i$th subinterval $I_i = [x_i, x_{i+1}]$ is $h_i = x_{i+1} - x_i$. We will set

\[ h = \max_i h_i. \]

#### 3.5.1 Piecewise Linear Splines

A piecewise linear spline approximates $f$ on $I_i$ by a linear polynomial interpolant. This means

\[ S_L(x) = \frac{f(x_i)(x_{i+1} - x) + (x - x_i)f(x_{i+1})}{h_i} \text{ for } x_i \leq x \leq x_{i+1}. \]

$S_L$ is clearly a continuous function. Theorem 3.2.1 implies that for each $x \in [a, b]$,

\[ f(x) = S_L(x) + \frac{f''(\xi)}{2} (x - x_i)^2, \text{ if } x \in I_i. \]

Clearly,

\[ \|f - S_L\|_\infty \leq \|f''\|_\infty h^2 / 2. \quad (3.15) \]

The $L^1$ and $L^2$ errors are also second order in $h$, but the constants differ.

\[ \|f - S_L\|_1 \leq \frac{\|f''\|_\infty}{2} \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} (x - x_i)^2 dx \]

\[ \leq \frac{\|f''\|_\infty}{6} \|f''\|_\infty \sum_{i=0}^{n-1} h_i^3 \leq \frac{\|f''\|_\infty h^2}{6} \sum_{i=0}^{n-1} h_i = \frac{\|f''\|_\infty (b-a)h^2}{6}. \]
As for the $L^2$ error
\[
\| f - S_L \|_2^2 \leq \frac{\| f'' \|_\infty^2}{4} \sum_{i=0}^{n-1} f''_{x_i}(x - x_i)^4 \, dx
\]
\[
\leq \frac{\| f'' \|_\infty^2}{4} \sum_{i=0}^{n-1} h_i^5 / 5, \frac{\| f'' \|_\infty^2 (b - a) h^4}{20}.
\]
So
\[
\| f - S_L \|_2 \leq \frac{\| f'' \|_\infty \sqrt{b - ah^2}}{\sqrt{20}}.
\]

### 3.5.2 Hermite Splines

If one approximates the solution of a differential equation $y' = F(y)$ by estimating the values of the solution at a set of nodal points, then, if $F$ is smooth, one knows $y'$ as well as $y$. This information can be used to approximate $y$ by a Hermite spline, $S_H$. In this special case, only the two nodes at the endpoints of any subinterval are used to construct the splines, so $n = 1$ and the approximation is 4th order.

\[
S_H(x) = (y_i + y'_i(x - x_i)) \left( \frac{x_{i+1} - x}{h_i} \right)^2 + (y_{i+1} + y'_{i+1}(x_{i+1} - x)) \left( \frac{x - x_i}{h_i} \right)^2, \text{ for } x_i \leq x \leq x_{i+1}.
\]

Theorem 3.4.1 implies that if $x \in I_i$,
\[
y(x) - S_H(x) = \frac{y^{(4)}(\xi)}{4!}(x - x_i)^2(x - x_{i+1})^2.
\]
Since
\[
(x - x_i)^2(x - x_{i+1})^2 \leq h_i^4 / 16,
\]
we have
\[
\| y - S_H \|_\infty \leq \| y^{(4)} \|_\infty h_i^4 / 16 \times 4!.
\]

In Exercise 2, you are asked to estimate the $L^1$ and $L^2$ errors.

### 3.5.3 $C^2$ cubic splines

One can still obtain a 4th order accurate approximation even if derivative information is not available. $C^2$ cubic spline approximations are cubic polynomials in the subintervals. The $4n$ unknown coefficients are constrained by

1. Interpolation: $S(x_i) = f(x_i)$ for $0 \leq i \leq n$.
2. Continuity at the interior nodes $\{x_i\}_{i=1}^{n-1}$
   (a) Continuity of $S$: $S$ is continuous at $x_i$ for $1 \leq i \leq n - 1$.
   (b) Continuity of $S'$: $S'$ is continuous at $x_i$ for $1 \leq i \leq n - 1$. 

18
(c) Continuity of \( S'' \): \( S'' \) is continuous at \( x_i \) for \( 1 \leq i \leq n - 1 \).

There are \( n + 1 \) interpolation constraints and \( 3n - 3 \) continuity constraints. We will see that these constraints can be expressed as \( 4n - 2 \) linear equations in the \( 4n \) unknown coefficients of the cubic polynomials on the subintervals.

Let

\[
S(x) = S_i(x) = \sum_{k=0}^{3} a_{ik}(x - x_i)^k / k! \text{ for } x \in I_i. \tag{3.16}
\]

Continuity of \( S'' \) implies that \( S'' \) is a piecewise linear spline. Let \( S''(x_i) = \sigma_i \). Then

\[
S''_i(x) = \sigma_i \frac{x_{i+1} - x}{h_i} + \sigma_{i+1} \frac{x - x_i}{h_i}.
\]

At this point we have used the continuity of \( S'' \) to replace the \( 2n \) unknown coefficients in the linear polynomials for \( S'' \) in the subintervals with the \( n + 1 \) unknowns \( \sigma_i \).

Integrate \( S''_i \) twice to obtain

\[
S'_i(x) = -\sigma_i \frac{(x_{i+1} - x)^2}{2h_i} + \sigma_{i+1} \frac{(x - x_i)^2}{2h_i} + C_i.
\]

and so, for \( 0 \leq i \leq n - 1 \), we may arrange the constants of integration so that

\[
S_i(x) = \sigma_i \frac{(x_{i+1} - x)^3}{6h_i} + \sigma_{i+1} \frac{(x - x_i)^3}{6h_i} + C_i (x - x_i) + D_i (x_{i+1} - x). \tag{3.17}
\]

We can solve for the constants of integration \( C_i \) and \( D_i \) in (3.17) in terms of the second derivatives using the interpolation constraint and the continuity of \( S \). The interpolation constraint implies that, for \( 0 \leq i \leq n - 1 \),

\[
f(x_i) = S_i(x_i) = \sigma_i h_i / 6 + D_i h_i,
\]

and hence, for \( 0 \leq i \leq n - 1 \),

\[
D_i = f(x_i) / h_i - \sigma_i h_i / 6. \tag{3.18}
\]

Continuity of \( S \) implies that, for \( 0 \leq i \leq n - 1 \),

\[
f(x_{i+1}) = S_i(x_{i+1}) = \sigma_{i+1} h_i^2 / 6 + C_i h_i
\]

and therefore

\[
C_i = f(x_{i+1}) / h_i - \sigma_{i+1} h_i / 6. \tag{3.19}
\]

At this point we have expressed the unknown coefficients of \( S_i \) completely in terms of the sequence of second derivatives \( \{ \sigma_i \}_{i=0}^{n-1} \) and the data,

\[
S_i(x) = \frac{(x-x_i)^3 \sigma_{i+1} + (x_{i+1} - x)^3 \sigma_i}{6h_i} + \frac{(x-x_i) f(x_{i+1}) + (x_{i+1} - x) f(x_i)}{h_i}
- \frac{h_i (x-x_i) \sigma_{i+1} + (x_{i+1} - x) \sigma_i}{6} \tag{3.20}
\]
We have not used the continuity constraint for $S'$. To do that we compute

$$S'_i(x) = \frac{(x-x_i)^2\sigma_{i+1} - (x_{i+1}-x_i)^2\sigma_i}{2h_i} + \frac{f(x_{i+1}) - f(x_i)}{h_i}$$

$$- \frac{h_i(\sigma_{i+1} - \sigma_i)}{6}$$

and

$$S'_{i-1}(x) = \frac{(x-x_{i-1})^2\sigma_i - (x_i-x)^2\sigma_{i-1}}{2h_{i-1}} + \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}$$

$$- \frac{h_{i-1}(\sigma_i - \sigma_{i-1})}{6}.$$ 

Continuity of $S'$ implies that

$$S'_{i-1}(x_i) = S'_i(x_i)$$

i. e.

$$\frac{h_{i-1}\sigma_i}{2} + \frac{f(x_i) - f(x_{i-1})}{h_{i-1}} - \frac{h_{i-1}(\sigma_i - \sigma_{i-1})}{6} = -\frac{h_i\sigma_i}{2} + \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{h_i(\sigma_{i+1} - \sigma_i)}{6}. $$

So, for $1 \leq i \leq n - 1$,

$$\frac{h_{i-1}\sigma_i}{6} - \frac{h_i + h_{i-1}}{3}\sigma_i + \frac{h_i}{6}\sigma_{i+1} = \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}.$$  \hspace{1cm} (3.21)

The system of equations (3.21) must be augmented with two more equations in order to close the system. The new equations, which will correspond to the first and last rows of the matrix, should preserve the tridiagonal, strictly diagonally dominant structure.

One easy way to do this is the **natural spline** conditions that $s''(a) = s''(b) = 0$. For this case, the 0th equation is $\sigma_0 = 0$ and the $n$th is $\sigma_n = 0$.

The **not-a-knot** closure enforces continuity of $S'''$ at $x_1$ and $x_{n-1}$. At $x_1$, for example, we ask that

$$S'''_0(x_1) = S'''_1(x_1),$$

i. e.

$$\frac{\sigma_1 - \sigma_0}{h_0} = \frac{\sigma_2 - \sigma_1}{h_1}.$$ 

So the 0th equation is

$$-h_0^{-1}\sigma_0 + (h_0^{-1} + h_1^{-1})\sigma_1 - h_1^{-1}\sigma_2 = 0.$$ \hspace{1cm} (3.22)

Set $i = 1$ in (3.21)

$$\frac{h_0}{6}\sigma_0 + \frac{h_1}{3}\sigma_1 + \frac{h_1}{6}\sigma_2 = \frac{f(x_2) - f(x_1)}{h_1} - \frac{f(x_1) - f(x_0)}{h_0}.$$ 

Eliminate $\sigma_0$ using (3.22) to get

$$\frac{h_0}{6}\sigma_0 = \frac{-1}{6} \left( (h_0^{-1} + h_1^{-1})\sigma_1 - h_1^{-1}\sigma_2 \right).$$

20
So the new first equation may be written as

\[ \alpha_1 \sigma_1 + \beta \sigma_2 = c \]

where

\[ \alpha = \frac{1}{6} \left( (h_0^{-1} + h_1^{-1}) - 2(h_1 + h_0) \right) \]
\[ \beta = -h_1^{-1} - h_1/6, \text{ and} \]
\[ \gamma = -\frac{f(x_2) - f(x_1)}{h_1} + \frac{f(x_1) - f(x_0)}{h_0}. \]

Diagonal dominance is preserved if \( h_i < \sqrt{6/25} \) for \( i = 0, 1 \). The case at the right endpoint is similar. MATLAB uses the not-a-knot condition.

Another common closure condition is specification of \( S' \) at the endpoints. At \( x_0 = a \),

\[ S'_0(x_0) = -h_0 \sigma_0/3 - h_0 \sigma_1/6 + \frac{f(x_1) - f(x_0)}{h_0}, \]

which implies that a constraint of the form \( S'(x_0) = a \) is diagonally dominant. Similarly for constraints of the form \( S'(x_n) = b \).

### 3.6 Error Estimation and Grid Refinement Studies

If \( f \) is sufficiently smooth, then the piecewise polynomial interpolation methods we have discussed in this chapter all satisfy formulae like

\[ E(h) = S(h) - f = O(h^d) \] (3.23)

where \( S(h) \) is the piecewise polynomial interpolant using intervals of length \( h \). If we assume a somewhat stronger condition

\[ E(h) = 2^d E(h/2) + O(h^{d+1}) \] (3.24)

then we can approximate the error by varying \( h \). In fact, (3.24) implies that

\[ S(h) - S(h/2) = E(h) - E(h/2) + O(h^{d+1}) = (2^d - 1)E(h/2) + O(h^{d+1}), \]

and therefore

\[ T(h) = (S(h) - S(h/2))/(2^d - 1) \approx E(h/2). \] (3.25)

There are two applications of (3.25). The first is to decide when an approximation is good enough. Suppose, for example, we wish to approximate \( f \) well at \( \{z_i\}_{i=1}^N \) by a piecewise polynomial. One way to quantify this objective is to reduce \( h \) until

\[ \max_{1 \leq i \leq N} |S(h)(z_i) - f(z_i)| \leq \tau, \] (3.26)
where $\tau$ is an error tolerance.

Equation (3.25) implies that we can replace (3.26) by

$$\max_{1 \leq i \leq N} |T(h)(z_i)| \leq \tau,$$  \hspace{1cm} (3.27)

and then accept $S(h/2)$ as the final result.

As second application is to verify that things are going well. If, for example, $S(h)$ is a cubic spline interpolant (so $d = 4$) and $f \in C^4[a, b]$, then (3.24) and (3.25) would imply that

$$T(h)/T(h/2) \approx 2^d = 16.$$  \hspace{1cm} (3.28)

One can test (3.28) for several values of $h$ in a grid refinement study. We would expect that (3.28) become more and more accurate as $h$ is reduced. If that does not happen, then it is likely that either the implementation of the spline interpolation has an error or $f$ is not $C^4$.

As an example we consider piecewise linear ($d = 2$), and the $C^2$ natural cubic spline interpolation for functions on $[0, 1]$. We let $z_i = (i - 1)/999$ for $1 \leq i \leq M = 1000$. We tabulate

$$\|E(h)\| = \max_{1 \leq i \leq N} |E(h)(z_i)|$$

the estimate

$$\|T(2h)\| = \max_{1 \leq i \leq N} |T(2h)(z_i)|$$

of $\|E(h)\|$, and the ratios

$$R(h) = \|T(2h)\|/\|T(h)\|.$$

Table 3.2 shows the result for $f(x) = \sin(x)$ on $[0, 1]$. This function is smooth and we see that $\|T(2h)\|$ agrees well with $\|E(h)\|$ and that the ratios $R(h)$ tend to the correct values of 4 for piecewise linear and 16 for the cubic spline interpolations. Even for this very easy problem, the agreement with the predicted values is not perfect, and one should not expect that.

<table>
<thead>
<tr>
<th>h</th>
<th>$|E(h)|$</th>
<th>$|T(2h)|$</th>
<th>$R(h)$</th>
<th>$|E(h)|$</th>
<th>$|T(2h)|$</th>
<th>$R(h)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>2.123e-02</td>
<td>7.196e-03</td>
<td></td>
<td>7.196e-03</td>
<td>4.793e-04</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>5.990e-03</td>
<td>7.056e-03</td>
<td>7.253e-05</td>
<td>7.253e-05</td>
<td>4.793e-04</td>
<td></td>
</tr>
<tr>
<td>1/8</td>
<td>1.574e-03</td>
<td>1.994e-03</td>
<td>3.54</td>
<td>5.277e-06</td>
<td>4.663e-06</td>
<td>02.79</td>
</tr>
<tr>
<td>1/32</td>
<td>1.016e-04</td>
<td>1.332e-04</td>
<td>3.91</td>
<td>2.218e-08</td>
<td>2.243e-08</td>
<td>15.19</td>
</tr>
<tr>
<td>1/64</td>
<td>2.553e-05</td>
<td>3.304e-05</td>
<td>4.03</td>
<td>1.400e-09</td>
<td>1.435e-09</td>
<td>15.63</td>
</tr>
</tbody>
</table>

In Table 3.3 we repeat the study with $f(x) = \sqrt{x}$, which is not differentiable at $x = 0$, much less $C^2$ or $C^4$. The predicted errors are far too optimistic, and the ratios converge to $\sqrt{2}$, not the predicted values of 4 and 16.
Table 3.3: Interpolation for $\sqrt{x}$

<table>
<thead>
<tr>
<th>h</th>
<th>$|E(h)|$</th>
<th>$|T(2h)|$</th>
<th>$R(h)$</th>
<th>$|E(h)|$</th>
<th>$|T(2h)|$</th>
<th>$R(h)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>1.768e-01</td>
<td></td>
<td></td>
<td>1.422e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>1.250e-01</td>
<td>4.877e-02</td>
<td></td>
<td>8.825e-02</td>
<td>6.704e-03</td>
<td></td>
</tr>
<tr>
<td>1/8</td>
<td>8.839e-02</td>
<td>3.448e-02</td>
<td>1.41</td>
<td>6.232e-02</td>
<td>3.472e-03</td>
<td>1.93</td>
</tr>
<tr>
<td>1/16</td>
<td>6.249e-02</td>
<td>2.424e-02</td>
<td>1.42</td>
<td>4.406e-02</td>
<td>2.446e-03</td>
<td>1.42</td>
</tr>
<tr>
<td>1/32</td>
<td>4.419e-02</td>
<td>1.714e-02</td>
<td>1.41</td>
<td>3.111e-02</td>
<td>1.729e-03</td>
<td>1.41</td>
</tr>
<tr>
<td>1/64</td>
<td>3.125e-02</td>
<td>1.190e-02</td>
<td>1.44</td>
<td>2.200e-02</td>
<td>1.223e-03</td>
<td>1.41</td>
</tr>
<tr>
<td>1/128</td>
<td>2.209e-02</td>
<td>8.413e-03</td>
<td>1.41</td>
<td>1.555e-02</td>
<td>8.645e-04</td>
<td>1.41</td>
</tr>
</tbody>
</table>

### 3.7 Exercises

3.7.1. Prove Theorem 3.4.2.

3.7.2. Estimate the $L^1$ and $L^2$ errors of piecewise Hermite splines.

3.7.3. Explain why the ratios of errors seem to be converging to $\sqrt{2}$ in Table 3.3.

3.7.4. One could use norms other than $l^\infty$ to measure $E(h)$ and $T(h)$ in the grid refinement studies in § 3.6. Try the scaled $l^1$ norm

$$
\|z\|_1^S = \frac{1}{N} \sum_{i=1}^{N} |z_i|
$$

and generate new versions of Tables 3.2 and 3.3. Why is the scaling (the multiplication by $1/N$) important? If the results are different from the tables, explain why.

3.7.5. Write a program that evaluates the interpolating polynomial at an points \(\{z_i\}_{i=1}^{m}\) using the Newton form. You’ll need to compute the divided difference table as part of this program.

3.7.6. Write a MATLAB program that takes as its input a function, the knots for the spline, and the points at which the spline should be evaluated and produces as output the values of the natural spline interpolant at the output points. Within this program you’ll need to obtain the coefficients for the natural spline interpolation. You’ll need to use a tridiagonal solver (which you may have written in ma580 or can get to in MATLAB with the sparse matrix operations) to solve the system (8.48) on page 350. Then compare your splines to the output of the MATLAB spline command and to the result from the program you wrote for Exercise 5.

- Use \(\{0, .1, .2, ..., .9, 1\}\) as knots.
- Use \(\{k/1000\}_{k=0}^{1000}\) as output points.
- Plot the function and the results from the three interpolants at the output points. Then plot the errors at the output points. Can you visually see any major differences?
Do this for the following functions

1. \( f(x) = e^x \)
2. \( f(x) = \cos(2\pi x) \)
3. \( f(x) = \cos(20\pi x) \)
4. \( f(x) = \sqrt{x} \)

If anything strange happens, explain it!
Chapter 4

Quadrature Rules

A Quadrature rule is a formula for approximating the definite integral
\[ I(f) = \int_a^b f(x) \, dx \]
by a weighted sum of the form
\[ I_m(f, a, b) = \sum_{i=0}^{m} f(x_i)w_i. \] (4.1)
The points \( \{x_i\}_{i=0}^{m} \) are called the nodes of the quadrature rule and \( \{w_i\}_{i=0}^{m} \) are the weights. We will simply write \( I_m(f) \) in cases where \( a \) and \( b \) are fixed.

In this chapter we will assume that \( a \) and \( b \) are finite.

Depending on the context, the sum in (4.1) may begin with \( i = 1 \) or \( i = 0 \). The subscript \( m \) may refer to the number of quadrature points or, if \( [a, b] \) has been subdivided into small intervals, \( m \) may refer to the number of those subintervals.

We will denote the error in the quadrature by
\[ E_m(f) = I(f) - I_m(f). \]
The precision of a quadrature rule is the largest \( k \) for which \( E_m(x^k) = 0 \).

4.1 Interpolatory Quadrature Rules

An interpolatory quadrature rule approximates \( f \) by an interpolating polynomial at a set of equally spaced points on \( [a, b] \) and then approximates the integral with the integral of the polynomial.

Interpolatory rules alone are poor choices, because there is no guarantee that the interpolating polynomial will converge to \( f \) as the degree increases. Composite rules address this in the following way. One subdivides \( [a, b] \) into subintervals, and then applies the interpolatory rule on each subinterval. In this way, similarly to piecewise polynomial approximation of functions, the degree of the interpolating polynomial is held constant, and one improves the accuracy by decreasing the size of the interval on which the interpolation is done.
4.1.1 The Midpoint Rule

The **midpoint rule** uses the single node \( c = (a + b) / 2 \), approximates \( f \) by the constant function

\[
p_0(x) = f(c),
\]

and the integral \( I(f) \) by

\[
I_0(f) = (b - a) f(c).
\]

By Taylor’s theorem, for each \( x \) there is \( \xi_x \) between \( x \) and \( c \) such that

\[
f(x) = f(c) + f'(c)(x - c) + f''(\xi_x)(x - c)^2 / 2
\]

for some \( \xi_x \in (a, b) \). We integrate (4.2) on \([a, b]\), and observe that

\[
\int_a^b (x - c) \, dx = 0,
\]

to obtain

\[
I(f) = I_0(f) + \int_a^b f''(\xi_x)(x - c)^2 / 2 \, dx.
\]

The integral mean value theorem implies that there is \( \xi \in (a, b) \) such that

\[
I(f) = I_0(f) + f''(\xi)(b - c)^3 - (a - c)^3)
\]

(4.3)

To build the **composite midpoint rule** we divide \([a, b]\) into \( n \) subintervals

\[
[a, b] = \bigcup_{i=0}^{n-1} I_i, \text{ where } I_i = [z_i, z_{i+1}],
\]

and apply the midpoint rule on each interval. So

\[
I_n^M(f, a, b) = \sum_{i=0}^{n-1} I_0(f, x_i, x_{i+1}) = \sum_{i=0}^{n-1} h_i f \left( \frac{z_i + z_{i+1}}{2} \right),
\]

where \( h_i = z_{i+1} - z_i \).

The weights and nodes of the composite midpoint rule are

\[
x_i = \frac{z_i + z_{i+1}}{2} \text{ and } w_i = h_i \text{ for } 0 \leq i \leq n - 1.
\]

The error in the composite midpoint rule is the sum of the **local errors** on \( I_i \)

\[
E_i = \frac{f''(\xi_i)(h_i^3)}{24} \text{ where } \xi_i \in I_i.
\]
So, using the summation mean value theorem

\[
E_M = \sum_{i=0}^{n-1} \frac{f''(\xi_i) h_i^3}{24} = \frac{f''(\xi)}{24} \sum_{i=0}^{n-1} h_i^3.
\]

Hence

\[
|E_M| \leq \frac{\|f''\| h^2}{24} \sum_{i=0}^{n-1} h_i = \frac{\|f''\|(b - a) h^2}{24},
\]

where \( h = \max h_i \).

### 4.2 The Trapezoid Rule

The trapezoid rule integrates

\[
p(x) = f(a) \frac{b - x}{b - a} + f(b) \frac{x - a}{b - a}
\]
over \([a, b]\) to approximate \(I(f)\). The approximation is

\[
I_1(f, a, b) = \frac{b - a}{2} (f(a) + f(b))
\]

The interpolation error is

\[
f(x) - p(x) = \frac{f''(\xi)}{2} (x - a)(x - b)
\]
for \( \xi \in (a, b) \). Since \((x - a)(x - b)\) does not change signs in \([a, b]\), we can use the integral mean value theorem to show that the error in the integral is

\[
E_1 = \int_a^b \frac{f''(\xi)}{2} (x - a)(x - b) \, dx
\]

\[
= \frac{f''(\xi)}{2} \int_a^b (x - a)(x - b) \, dx = -\frac{f''(\xi)(b - a)^3}{12}.
\]

We construct the composite trapezoid rule as we did the composite midpoint rule. Let

\[
[a, b] = \bigcup_{i=0}^{n-1} I_i, \text{ where } I_i = [z_i, z_{i+1}].
\]

If we apply the trapezoid rule to each subinterval we obtain

\[
I_n^T(f, a, b) = \sum_{i=0}^{n-1} I_1(f, x_i, x_{i+1}) = \sum_{i=0}^{n-1} h_i \left( \frac{f(z_i) + f(z_{i+1})}{2} \right),
\]

where \( h_i = z_{i+1} - z_i \). So, the nodes of the composite trapezoid rule are \( \{x_i\} = \{z_i\} \), the endpoints of the intervals. The weights are

\[
w_0 = h_0/2, \ w_i = \frac{h_i + h_{i+1}}{2} \text{ for } 1 \leq i \leq n - 1, \text{ and } w_n = h_n/2.
\]
The error in the composite trapezoid rule is the sum of the local errors

\[ E_n^T = -\sum_{i=0}^{n-1} \frac{f''(\xi_i)h_i^3}{12} \]

where \( \xi_i \in (z_i, z_{i+1}) \). We can apply the summation mean value theorem to obtain

\[ E_n^T = -\frac{f''(\xi)}{12} \sum_{i=0}^{n-1} h_i^3, \]

for some \( \xi \in (a, b) \). If the intervals are of equal size, we obtain the classical result,

\[ E_n^T = -\frac{f''(\xi)(b-a)h^2}{12}. \]

### 4.3 Simpson’s Rule

Simpson’s rule integrates the quadratic polynomial that interpolates \( f \) at \( a, b \), and the midpoint \( c = (a+b)/2 \). So

\[ I^S(f, a, b) = \frac{b-a}{6} (f(a) + 4f(c) + f(b)). \]

As for the error, we can use the same trick as we used for the midpoint rule. The interpolation error is

\[ \frac{f'''(\xi_x)}{6} (x-a)(x-c)(x-b), \]

where \( \xi_x \in [a, b] \). The error in the integral is

\[ E^S = \int_a^b \frac{f'''(\xi_x)}{6} (x-a)(x-c)(x-b) \, dx. \]

Now, if

\[ f(x) = x^3, \]

then \( f'''(\xi_x) = 6 \) and

\[ E^S = 6 \int_a^b (x-a)(x-c)(x-b) \, dx = 0. \]

Hence, the precision of Simpson’s rule is 3.

Let \( p(x) \) be the cubic polynomial that interpolates \( f \) at \( a, b, c \) and \( f' \) at \( c \). By Theorem 3.4.2, there is \( \xi_x \in (a, b) \) such that

\[ f(x) = p(x) + \frac{f^{iv}(\xi_x)}{4!} (x-a)(x-c)^2(x-b). \]

Since Simpson’s rule has precision 3,

\[ I^S(f) = I^S(p) = I(f) - \int_a^b \frac{f^{iv}(\xi_x)}{4!} (x-a)(x-c)^2(x-b) \, dx, \]
and so, by the integral mean value theorem, there is \( \xi \in (a, b) \) such that

\[
E^S = -\frac{f^{iv}(\xi)}{4!} \int_a^b (x - a)(x - c)^2(x - b) \, dx = -\frac{f^{iv}(\xi)}{4!} \left( \frac{b-a}{2} \right)^5 (4/15)
\]

\[
= -\frac{f^{iv}(\xi)}{90} \left( \frac{b-a}{2} \right)^5.
\]

4.4 Newton-Cotes Rules

Newton-Cotes rules are interpolatory rules in which the nodes are equally spaced. A closed Newton-Cotes rule has nodes

\[
a = x_0, x_1 = x_0 + h \ldots x_n = b,
\]

with \( h = (b - a)/n \). Simpson’s rule and the trapezoid rule are closed Newton-Cotes rules.

An open Newton-Cotes rule has nodes at

\[
a + h = x_1, x_2 = x_1 + h, \ldots x_n = b - h,
\]

with \( h = (b - a)/(n + 1) \). The midpoint rule is an open Newton-Cotes rule.

4.4.1 Closed Newton-Cotes Rules

If \( n \) is odd, as is the case with the trapezoid rule, the precision of the rule is \( n \). However, if \( n \) is even, then the precision of the rule is \( n + 1 \), as happens with Simpson’s rule.

The proof of this assertion is easy. The error in interpolation is

\[
\frac{f^{n+1}(\xi)}{(n + 1)!} \prod_{i=0}^{n}(x - x_i).
\]

If \( n \) is even, then

\[
\int_a^b \prod_{i=0}^{n}(x - x_i) \, dx = 0,
\]

and hence \( p(x) = x^{n+1} \) is integrated exactly, since

\[
I_n(p) = I(p) - (n + 1)! \int_a^b \prod_{i=0}^{n}(x - x_i) \, dx = I(p).
\]

Hence the precision of the rule for even \( n \) is \( n + 1 \).

The product in the formula for the interpolation error changes signs if \( n > 1 \), so the proof of the error formula for the quadrature rule is more than a simple application of the integral mean value theorem [8].
Theorem 4.4.1 Let $E_n$ be the error in an $n$-point closed Newton-Cotes rule on $[a, b]$. Let $f \in C^{n+2}[a, b]$. Then there is $\xi \in (a, b)$ such that
\[
E_n = \frac{f^{(n+1)}(\xi)}{(n+1)!} \int_a^b \prod_{i=0}^{n} (x - x_i) \, dx \text{ if } n \text{ is odd}
\]
and
\[
E_n = \frac{f^{(n+2)}(\xi)}{(n+2)!} \int_a^b x \prod_{i=0}^{n} (x - x_i) \, dx \text{ if } n \text{ is even}.
\]

Table 4.1 contains the quadrature weights and errors for $1 \leq n \leq 5$. The convention used in the table is that
\[
I(f) = \sum_{i=0}^{n} w_i f(x_i) - E_n(f)
\]
and that
\[
h = \frac{b - a}{n}.
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$E_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$h/2$</td>
<td>$h/2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$-h^3 f''(\xi)/12$</td>
</tr>
<tr>
<td>2</td>
<td>$h/3$</td>
<td>$4h/3$</td>
<td>$h/3$</td>
<td></td>
<td></td>
<td></td>
<td>$-h^5 f^{(iv)}(\xi)/90$</td>
</tr>
<tr>
<td>3</td>
<td>$3h/8$</td>
<td>$9h/8$</td>
<td>$9h/8$</td>
<td>$3h/8$</td>
<td></td>
<td></td>
<td>$-3h^5 f^{(iv)}(\xi)/80$</td>
</tr>
<tr>
<td>4</td>
<td>$14h/45$</td>
<td>$64h/45$</td>
<td>$24h/45$</td>
<td>$64h/45$</td>
<td>$14h/45$</td>
<td></td>
<td>$-8h^7 f^{(v)}(\xi)/945$</td>
</tr>
<tr>
<td>5</td>
<td>$95h/288$</td>
<td>$375h/288$</td>
<td>$250h/288$</td>
<td>$250h/288$</td>
<td>$375h/288$</td>
<td>$95h/288$</td>
<td>$-275h^7 f^{(vi)}(\xi)/12096$</td>
</tr>
</tbody>
</table>

Closed Newton-Cotes rules with $n \geq 8$ have negative weights. This can result in numerical instability for composite rules. The most commonly used are the Trapezoid rule ($n = 1$) and Simpson’s rule ($n = 2$).

### 4.4.2 Open Newton-Cotes Rules

Open Newton-Cotes rules for $n \geq 2$ can have negative weights, are rarely used. One fourth-order rule ($n = 3$) has positive weights. The formula and error are
\[
I_{ONC}^4(f) = \frac{5(b - a)}{24} (11 f(x_0) + f(x_1) + f(x_2) + 11 f(x_3)),
\]
and
\[
E_{ONC}^4(f) = \frac{95(b - a)^5 f^{(iv)}(\xi)}{144}.
\]
4.5 Extrapolation and Romberg Integration

Suppose you have an approximation \( A(h) \) to a scalar \( A \), and that the error can be expanded in powers of \( h \). In most cases, the expansion is successively higher powers of \( h^d \), where \( d = 1 \) or \( d = 2 \). So, assume that

\[
A(h) = A + C_1 h^d + C_2 h^{2d} + \ldots + C_n h^{nd} + O(h^{d(n+1)}), \tag{4.6}
\]

where the coefficients \( \{C_i\} \) do not depend on \( h \).

**Extrapolation** is a simple technique that exploits the expansion to generate higher-order approximations. If we compare \( A(h) \) to \( A(h/2) \) we see that

\[
A_1(h) = \frac{2^d A(h/2) - A(h)}{2^d - 1} = A + O(h^{2d}), \tag{4.7}
\]

and we have double the order of the approximation. The cost of this doubling is that we must compute the low-order approximation twice. Moreover, \( A_1 \) also has an expansion like (4.6)

\[
A_1(h) = A + C_{2,1} h^{2d} + \ldots + C_{n,1} h^{nd} + O(h^{d(n+1)}),
\]

where, for example

\[
C_{2,1} = \frac{2^{-d} - 1}{2^d - 1} C_2.
\]

We may continue the process by using \( A(h/4) \) and \( A(h/2) \) to compute \( A_1(h/2) \). We can get an approximation of order \( 3d \) from

\[
A_2(h) = \frac{2^{2d} A_1(h/2) - A_1(h)}{2^{2d} - 1} = A + O(h^{3d}).
\]

The pattern should be clear. To obtain an approximation of the next higher order \( 4d \), one would need to compute \( A(h/8) \) and then use information already in hand to compute \( A_2(h/2) \) and \( A_3(h) \).

The process is easy to visualize in a table. In the left column of Table 4.2 are evaluations of \( A_0(2^{-p}h) = A(2^{-p}h) \) for \( p = 0, \ldots \). In the other columns are the evaluations of \( A_m(2^{-p}h) \) that use the evaluations of \( A \). The table is lower triangular because one needs all of

\[
A(h), \ldots, A(2^{-m}h)
\]

to compute \( A_m(h) \).

There are two ways to apply extrapolation. One way, **analytic extrapolation** is to apply extrapolation to an algorithm and obtain a new, higher-order, method. Consider the trapezoid rule, which satisfies (4.6) with \( d = 2 \).

We will apply extrapolation once to obtain a fourth-order method. Letting \( h = b - a \), we compute

\[
A(h) = \frac{h}{2} (f(a) + f(b)).
\]
Table 4.2: Extrapolation Table

<table>
<thead>
<tr>
<th>$A(h)$</th>
<th>$A_1(h)$</th>
<th>$A_2(h)$</th>
<th>$A_3(h)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(h/2)$</td>
<td>$A_1(h/2)$</td>
<td>$A_2(h/2)$</td>
<td>$A_3(h/2)$</td>
</tr>
<tr>
<td>$A(h/4)$</td>
<td>$A_1(h/4)$</td>
<td>$A_2(h/4)$</td>
<td>$A_3(h/4)$</td>
</tr>
<tr>
<td>$A(h/8)$</td>
<td>$A_1(h/8)$</td>
<td>$A_2(h/8)$</td>
<td>$A_3(h/8)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

To compute $A(h/2)$, we must apply the composite trapezoid rule on $[a, b]$ with two subintervals, $[a, c]$ and $[c, b]$, with $c = (b + a)/2$. The composite formula is

$$A(h/2) = \frac{h}{4}(f(a) + 2f(c) + f(b)).$$

Following formula (4.7) with $d = 2$, the fourth-order method is

$$A_1(h) = \frac{1}{3}(4A(h/2) - A(h))$$

$$= \frac{h}{3}(f(a)/2 + 2f(c) + f(b)/2) = \frac{h}{6}(f(a) + 4f(c) + f(b)).$$

We have rediscovered Simpson’s rule. SHAZAM!

Numerical extrapolation can go deeper into the extrapolation table than analytic extrapolation, because a computer does the work. In numerical extrapolation, one computes the first column of the table, say $m + 1$ rows, and then applies the extrapolation formula to complete the table and obtain $A_m(h)$. The algorithm depends on $A$, the power $d$, and the depth of the table.

```
extrap(A_0, h, d, m)
    for i = 0...m do
        Compute $A_0(2^{-m}h)$.
    end for
    for p = 1...m do
        for q = 0...m - i do
            $h_l = 2^{-q}h$
            $A_p(h_l) = (2^{pd}A_{p-1}(h_l/2) - A_{p-1}(h_l))/(2^{pd} - 1)$
        end for
    end for
```

Romberg integration is the numerical extrapolation of the composite trapezoid rule, which also satisfies (4.6) with $d = 2$.

### 4.6 Gaussian Quadratures

Gaussian quadratures have the maximal precision among $m$ point rules. These rules are for
integrals with weight functions $w$, and we seek to approximate
\[ \int_a^b f(x)w(x) \, dx. \]  
(4.8)

In (4.8) the function $w(x) \geq 0$ is positive almost everywhere.

Let the nodes and weights be $\{x_i\}_{i=0}^{m-1}$ and $\{w_i\}_{i=0}^{m-1}$. We seek to integrate
\[ \int_a^b x^k w(x) \, dx \] 
for $k = 0, \ldots, N$

exactly for the largest possible $N$. If either of $a$ or $b$ is infinite, we insist that $w(x)$ decay rapidly enough so that polynomials are absolutely integrable.

If $N = 2m - 1$, then the system of nonlinear equations
\[ \sum_{i=0}^{m-1} x_i^k w_i = \int_a^b x^k w(x) \, dx \] 
for $0 \leq k \leq N$,  
(4.9)

is square, having $2m$ equations and $2m$ unknowns.

### 4.6.1 Roots of Orthogonal Polynomials

We find the nodes and weights that satisfy (4.9) by a clever trick. First note that if we have computed the nodes, and they are distinct, then the weights are
\[ w_i = \int_a^b l_i(x)w(x) \, dx, \]
which can, in theory at least, be computed directly. Now, let
\[ p_m = \prod_{i=0}^{m-1} (x - x_i) \in \mathcal{P}_m. \]

The sequence $\{x_i\}$ are the nodes of a quadrature rule satisfying (4.9) if and only if
\[ 0 = \int_a^b p_m(x)q(x)w(x) \, dx, \] 
(4.10)

for all $q \in \mathcal{P}_{m-1}$, for then $p_m q \in \mathcal{P}_{2m-1}$.

Equation (4.10) is an orthogonality condition. It says that $p_m$ is orthogonal to all polynomials of lower degree with respect to the function-space inner product
\[ (f, g)_w = \int_a^b f(x)g(x)w(x) \, dx. \]  
(4.11)

In the special case where $w(x) = 1$, $(\cdot, \cdot)_w$ is the usual $L^2$ inner product, the subscript $w$ is usually omitted, and the rule is simply called Gaussian quadratures, with no reference to the weight function.

The sequence of polynomials generated in this way $\{p_m\}_{m=0}^\infty$ (with $p_0 = 1$) is called a sequence of orthogonal polynomials on $(a, b)$ with respect to the weight function $w$. The roots of the $m$th polynomial are the nodes of the $m$ point Gauss rule for $w$.  

4.6.2 Construction of Orthogonal Polynomials

One way to compute the weights and nodes for Gauss rules is to construct the sequence of orthogonal polynomials analytically, and then find the roots either analytically or numerically. This is, in fact, a useful method for small $m$. One can build the sequence with the Gram-Schmidt method, which you may have seen before as part of the GMRES and Arnoldi methods in numerical linear algebra [4, 9, 15].

Beginning with $p_0 = 1$, we define, for $m > 0$,

$$p_m = x^m - \sum_{i=0}^{m-1} a_{mi} p_i$$

where

$$a_{mi} = \frac{(x^m, p_i)_w}{(p_i, p_i)_w}.$$ 

Since $(p_i, p_j)_w = 0$ for $1 \leq i \neq j \leq m - 1$,

$$(p_m, p_i)_w = (x^m, p_i)_w - a_{mi}(p_i, p_i)_w = 0.$$ 

As an example, we will compute the Gaussian weights and nodes for $(a, b) = [-1, 1]$, $w(x) = 1$, and $m = 1, 2, 3$. For $m = 1$, we have

$$p_1(x) = x - (x, p_0)/(p_0, p_0) = x.$$ 

The single node is $x_0 = 0$ and the weight is $w_0 = 2$. For $m = 2$,

$$p_2(x) = x^2 - x(x^2, p_1)/(p_1, p_1) - (x^2, p_0)/(p_0, p_0) = x^2 - 1/3.$$ 

The nodes are $\pm 1/\sqrt{3}$ and the weights are $w_0 = w_1 = 1$. Finally, for $m = 3$, we have

$$p_3(x) = x^3 - p_2(x)(x^3, p_2)/(p_2, p_2) - x(x^3, p_1)/(p_1, p_1) - (x^3, p_0)/(p_0, p_0) = x^3 - 3x/5.$$ 

The nodes are $\pm \sqrt{3/5}$ and 0. The weights are $w_0 = w_2 = 1/3$ and $w_0 = 2/3$.

4.7 Error Estimation

4.8 Exercises

4.8.1. Analyze the Hermite quadrature rule, which approximates $f$ by its Hermite interpolant on $[a, b]$ and has the form

$$a_0 f(a) + b_0 f'(a) + a_1 f(b) + b_1 f'(b).$$

Compute the weights. Show that this method is fourth order and obtain a formula for the error. Derive the composite rule and show that only the derivatives at the end points are needed.
4.8.2. Use the method from § 4.6.2 to compute the weights and nodes for the 4 point Gauss rule on \([-1, 1]\) by hand. Then compute the polynomial \(p_5\) and try to compute the nodes numerically.

4.8.3. Write codes for Romberg extrapolation \((m = 4)\), composite four point Gaussian quadratures, and the composite Simpson’s rule. Apply those codes with to the following integrals. Use a few values of \(h\) to perform a grid refinement study. Are you seeing the predicted orders of accuracy?

1. \(\int_0^\pi \sin(x) \, dx\)
2. \(\int_0^\pi \sin^2(10x) \, dx\)
3. \(\int_0^1 \sqrt{x} \, dx\)
4. \(\int_0^5 e^x \, dx\)

4.8.4. Write (4.9) as a nonlinear equation

\[ F(z) = 0, \quad \text{where } z = \begin{pmatrix} x \\ w \end{pmatrix}. \]

What conditions on the nodes and weights imply that \(F'(z)\) is nonsingular?
Chapter 5

Initial Value Problems

The methods in this chapter solve the initial value problem (1.7)–(1.8) on an interval \([t_0, t_0 + T]\), by computing a sequence of temporal sampling points \(\{t_n\}_{n=0}^N\), where \(t_N = T\), and vectors \(\{u_n\}_{n=0}^N\), so that

\[
u_n \approx y(t_n), \tag{5.1}\]

where \(y(t)\) is the solution of the initial value problem.

The \(n\)th time step \(h_n = t_{n+1} - t_n\). An integrator is a fixed step method if \(h_n = h\) is independent of \(n\). Otherwise the integrator is a variable step method.

The global error in the integration is

\[
E_G = \max_{1 \leq n \leq N} |u_n - y(t_n)|. \tag{5.2}\]

The creation of the sampling points is an important part of the algorithm. A fixed step method is almost certain to result in too much computational work, higher than expected errors, or both.

5.1 Existence and Uniqueness

The results in this section require concepts from function analysis. In particular, Theorem 5.1.1 is an application of the contraction mapping theorem on spaces of continuous functions.

**Theorem 5.1.1** Assume that \(f \in C([t_0, \infty), R^M)\) satisfies (1.9). Then the initial value problem (1.7)–(1.8) has a unique solution \(y \in C^1([t_0, \infty) : R^M)\). Moreover, if \(f \in C^k([t_0, \infty) : R^M)\), then \(y \in C^{k+1}([t_0, \infty) : R^M)\).

**Proof.** Let \(\tau = 1/(2L)\), where \(L\) is the Lipschitz constant of \(f\) as defined in (1.9). Define a nonlinear map \(K\) on \(C([t_0, t_0 + \tau] : R^M)\) by

\[
K(y)(t) = \int_{t_0}^t f(s, y(s)) \, ds + y_0.
\]

The fundamental theorem of calculus and (1.9) imply that if \(y = K(y)\), then \(y\) is a solution of the initial value problem.
We show that there is a solution of $y = K(y)$ on $C([t_0, t_0 + \tau] : \mathbb{R}^M)$ by the contraction mapping theorem. Let $y_1, y_2 \in C([t_0, t_0 + \tau] : \mathbb{R}^M)$. By (1.9)

$$\|K(y_1)(t) - K(y_2)(t)\| \leq \int_{t_0}^{t} \|f(s, y_1(s)) - f(s, y_2(s))\| \, ds$$

$$\leq \int_{t_0}^{t} L\|y_1(s) - f(s)\| \, ds$$

$$\leq (t - t_0)L\|y_1 - y_2\|_{\infty} \leq \|y_1 - y_2\|_{\infty}/2.$$

Hence

$$\|K(y_1) - K(y_2)\|_{\infty} \leq \|y_1 - y_2\|_{\infty}/2,$$

and therefore $K$ is a contraction on $C([t_0, t_0 + \tau] : \mathbb{R}^M)$. Hence there is a unique fixed point $y$, which is also a solution of the initial value problem on the interval $[t_0, t_0 + \tau]$.

We can repeat the above argument with the new map

$$K_1(y)(t) = \int_{t_0 + \tau}^{t} f(s, y(s)) \, ds + y(t_0 + \tau),$$

on $C([t_0 + \tau, t_0 + 2\tau] : \mathbb{R}^M)$ to continue the solution. By repeating the process indefinitely we conclude that there is a unique solution $y \in C([t_0, \infty) : \mathbb{R}^M)$ of the initial value problem and that

$$y(t) = \int_{t_0}^{t} f(s, y(s)) \, ds + y_0.$$

Now if $f \in C^k([t_0, \infty) : \mathbb{R}^M)$, then the fundamental theorem of calculus implies that $y \in C^{k+1}([t_0, \infty) : \mathbb{R}^M)$. This completes the proof. ☐

### 5.2 Explicit and Implicit Methods

A method is **explicit** if the advance in time is given by a direct formula. An example of an explicit method is the **forward (explicit) Euler method**. In this method we approximate (1.7) with the forward difference

$$\frac{u_{n+1} - u_n}{h_n} = f(t_n, u_n)$$

and solve for $u_{n+1}$ to obtain

$$u_{n+1} = u_n + hf(t_n, y_n).$$

Contrast this with an **implicit method**, for which an equation must be solved to advance in time. The **backward (implicit) Euler method** approximates (1.7) with the backward difference

$$\frac{u_{n+1} - u_n}{h_n} = f(t_{n+1}, u_{n+1}).$$

This cannot be solved directly for $u_{n+1}$. The method requires that the nonlinear equation

$$u_{n+1} = u_n + h_n f(t_{n+1}, u_{n+1})$$

be solved for $u_{n+1}$. This is the simplest example of an implicit method.
5.3 One Step methods

One step methods advance in time using only the value of the solution at the previous time step. The forward and backward Euler methods are examples of one-step methods.

A general formula for an explicit one step method is
\[ u_{n+1} = u_n + h_n \Phi(t_n, u_n), \]
and for an implicit one step method is
\[ u_{n+1} = u_n + h_n \Psi(t_{n+1}, u_{n+1}, t_n, u_n). \]

The function \( \Phi \) can, as we will see, be very complicated.

The local error in a one step method at time \( t_n \) is, for an explicit method,
\[ \tau_L(t, h) = y(t + h) - (y(t) + h \Phi(t, y(t))). \]

For an implicit method there is a subtle issue. We want to apply the concept of local error as the error in a single step of the method. That means that we would solve
\[ z = y(t) + h \Psi(t + h, z, t, y(t)) \]
and define the local error to be
\[ \tau_L^1(t, h) = y(t + h) - z. \]

However, the formulae (5.5)–(5.6) are not easy to use. It would be far more simple to use the error in the formula
\[ \tau_L(t, h) = y(t + h) - (y(t) + h \Psi(t + h, y(t + h), t, y(t))). \]

One can show with the contraction mapping theorem that if \( \Psi \) is Lipschitz continuous in its second argument, that (5.5) has a unique solution for all sufficiently small \( h \). In that case \( \tau_L \) and \( \tau_L^1 \) agree to leading order for sufficiently small \( h \). This means
\[ \tau_L(t, h) = (1 + O(h))\tau_L^1(t, h). \]

The local truncation error is the local error divided by the step size,
\[ \tau(t, h) = h^{-1} \tau_L(t, h). \]

The maximum local truncation error, or global truncation error,
\[ \tau_G(h) = \max_{t_0 \leq t \leq t_0 + T} \tau(t, h) \]
is a good estimate for the global error, as we show in Theorem 5.3.4.

**Definition 5.3.1** A one-step method is consistent on \([t_0, t_0 + T]\) if
\[ \lim_{h \to 0} \tau_G(h) = 0. \]

We say the method is consistent of order \( p \) or is of order \( p \) if
\[ \tau_G(h) = O(h^p). \]
5.3.1 Convergence and 0-stability

The first step in this error analysis is the discrete Gronwall inequality. We state this as Lemma 5.3.2 and will use it frequently in this chapter.

**Lemma 5.3.2** Let $\bar{\beta}, \alpha \geq 0$ and let $\{\beta_n\} \subset [0, \bar{\beta})$. Let $\epsilon_0$ be given and define

$$\epsilon_{n+1} = (1 + \alpha)\epsilon_n + \beta_n, \text{ for } n \geq 0.$$  

Then

$$\epsilon_n \leq e^{n\alpha}(\epsilon_0 + n\bar{\beta}). \quad (5.11)$$

**Proof.** From the definition of $\{\epsilon_n\}$ we have

$$\epsilon_n = (1 + \alpha)^n \epsilon_0 + \sum_{p=0}^{n-1} (1 + \alpha)^p \beta_{n-p}.$$  

Hence

$$\epsilon_n \leq (1 + \alpha)^n \left( \epsilon_0 + \bar{\beta} \sum_{p=1}^{n} (1 + \alpha)^{-p} \right)$$

$$\leq (1 + \alpha)^n \left( \epsilon_0 + n\bar{\beta} \right)$$

$$\leq e^{n\alpha} \left( \epsilon_0 + n\bar{\beta} \right),$$

as asserted. □

**Definition 5.3.3** A explicit one step method (5.3) is 0-stable if there is $h_0$ such that if

$$z_{n+1} = u_n + h_n (\Phi(t_n, z_n) + \delta_n),$$

then

$$\|u_n - z_n\| = O(\|u_0 - z_0\| + \sup_n \|\delta_n\|).$$

Now we can connect the global truncation error to the global error.

**Theorem 5.3.4** Let $\Phi(t, y)$ be Lipschitz continuous in the $y$ variable with Lipschitz constant $\Lambda$. Let the initial value problem (1.7)–(1.8) have a unique solution on the interval $[t_0, t_0 + T]$. Then the global error for the fixed-step integrator given by (5.3) satisfies

$$E_G \leq e^{\Lambda T} T \tau_G(h). \quad (5.12)$$

**Proof.** Let $e_n = u_n - y(t_n)$. From the formula

$$u_{n+1} = u_n + h\Phi(t_n, u_n)$$

39
we obtain
\[ e_{n+1} = u_n + h\Phi(t_n, u_n) - y(t_{n+1}) \]
\[ = u_n - y(t_n) + y(t_n) + h\Phi(t_n, y(t_n)) - y(t_{n+1}) + h\Phi(t_n, u_n) - h\Phi(t_n, y(t_n)) \]
\[ = e_n + h\tau(t_n) + h(\Phi(t_n, u_n) - \Phi(t_n, y(t_n))). \]

So,
\[ \|e_{n+1}\| \leq h\|\tau(t_n)\| + (1 + \Lambda h)\|e_n\|. \]  (5.13)

Equation (5.13) implies that
\[ \|e_{n+1}\| \leq \epsilon_n, \]
where \( \epsilon_0 = 0 \) and
\[ \epsilon_{n+1} = ((1 + \Lambda h)\epsilon_n + h\|\tau(t_n)\|). \]

Now let \( T/N = h \), we apply Lemma 5.3.2 with \( \alpha = \Lambda h, \epsilon_0 = 0, \beta_n = h\|\tau(t_n, h)\|, \bar{\beta} = h\tau_G(h), \) and \( Nh = T \) to obtain (5.12). \( \square \)

**Corollary 5.3.5** A consistent, 0-stable one step method is convergent.

### 5.3.2 Asyptotic Stability of One-step Methods

\[ y' = \lambda y \]  (5.14)

### 5.3.3 Examples of one-step methods

**The Euler Methods**

It is easy to compute the local truncation error, and hence the order, of the forward Euler method. From the formula
\[ u_{n+1} = u_n + hf(t_n, u_n), \]
we see that the local error is
\[ \tau_L(t, h) = y(t+h) - y(t) - h\Phi(t, y(t)). \]

If \( y'' \) is Lipschitz continuous, we can use the equation \( y'(t) = \Phi(t, y(t)) \) to obtain
\[ \tau_L(t, h) = y''(t)h^2/2 + O(h^3). \]

Hence the local truncation error is, to leading order,
\[ \tau(t, h) = y''(t)h/2 = O(h), \]
and the order of the method is one.
For the implicit method
\[ u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}), \]
we see that, if \( y'' \) is Lipschitz continuous
\[
\tau_L(t, h) = y(t + h) - hf(t + h, y(t + h)) - y(t)
\]
\[ = y(t + h) - hy'(t + h) - y(t) = -y''(t + h)h^2/2 + O(h^3) = -y(t)h^2/2 + O(h^3). \]
Hence the local truncation error is
\[
\tau(t, h) = -y''(\xi)h/2 = O(h),
\]
and the method has order one.

**Taylor Methods**

Taylor methods attempt to expand \( y \) about \( t_n \) and use the differential equation to compute the Taylor series coefficients in terms of values of \( y \) and partial derivatives of \( f \). The need to compute derivatives of \( f \) means that these are not general-purpose methods.

We begin with a \( p \)th order Taylor expansion of \( y(t + h) \) about \( t \)
\[
y(t + h) = y(t) + hy'(t) + h^2y''(t)/2 + \ldots + h^p y^{(p)}(t)/p! + O(h^{p+1}). \tag{5.15}
\]
Since \( y'(t) = f(t, y(t)) \) we have, for \( p = 1 \),
\[
y(t + h) = y(t) + hf(t, y(t)) + O(h^2),
\]
leading to the first-order Taylor method
\[
u_{n+1} = u_n + hf(t_n, u_n),
\]
which is the forward Euler method.

If we differentiate a second time we obtain
\[
y''(t) = f_y y' + f_t = f_y f + f_t
\]
where \( f_y = \partial f / \partial y \), the Jacobian matrix of \( f \) with respect to the vector variable \( y \), and \( f_t = \partial f / \partial t \). This leads to the second-order Taylor method
\[
u_{n+1} = u_n + hf(t_n, u_n) + h^2/2(f_y(t_n, u_n)f(t_n, u_n) + f_t(t_n, u_n)). \tag{5.16}
\]
Taylor methods of very high order are used for specialized applications.

The Runge-Kutta methods, which we discuss next, attempt to obtain high order without evaluating derivatives of \( f \).
5.3.4 Explicit Runge-Kutta Methods

Runge-Kutta methods attempt to replace the evaluations of derivatives of \( f \) in Taylor methods with evaluations of \( f \) at other points. The evaluations of \( f \) are called stages Stages of a Runge-Kutta method. The general form of an explicit \( s \)-stage Runge-Kutta method is

\[
\mathbf{u}_{n+1} = \mathbf{u}_n + h \sum_{i=1}^{s} b_i \mathbf{U}_i,
\]

where

\[
\mathbf{U}_1 = f(t_n, \mathbf{u}_n)
\]

and for \( 1 < i \leq s \),

\[
\mathbf{U}_i = f(t_n + c_i h, \mathbf{u}_n + h \sum_{j=1}^{i-1} a_{i,j} \mathbf{U}_j).
\]

The constants \( \{b_i\} \) and \( \{a_{i,j}\} \) determine the method. Consistency requires that

\[
c_i = \sum_{j=1}^{i-1} a_{i,j}.
\]

The Butcher array encodes the parameters of a Runge-Kutta method as,

\[
\begin{array}{c|c}
\mathbf{c} & \mathbf{A} \\
\hline
\end{array}
\begin{array}{c}
\mathbf{b}^T
\end{array}
\]

In (5.18), \( \mathbf{A} \) is \( s \times s \) and \( \mathbf{c}, \mathbf{b} \in \mathbb{R}^s \). For explicit methods, the matrix \( \mathbf{A} \) is strictly lower triangular.

We will derive the class of two-stage second-order explicit methods, show two examples, and show how one of them can be obtained by extrapolation of Euler’s method.

If \( s = 2 \), then the method must have the form

\[
\mathbf{u}_{n+1} = \mathbf{u}_n + h(b_1 \mathbf{U}_1 + b_2 \mathbf{U}_2)
\]

where, using \( c_1 = 0 \) and \( c_2 = a_{2,1} \),

\[
\mathbf{U}_1 = f(t_n, \mathbf{u}_n) \text{ and } \mathbf{U}_2 = f(t_n + c_2 h, \mathbf{u}_n + c_2 h \mathbf{U}_1).
\]

We will expand \( \mathbf{U}_2 \) about \( (t_n, \mathbf{u}_n) \), build the Runge-Kutta method, and compare the result to the 2nd order Taylor method. First

\[
\mathbf{U}_2 = f(t_n + c_2 h, \mathbf{u}_n + c_2 h \mathbf{U}_1)
\]

\[
= f(t_n, \mathbf{u}_n) + f_t(t_n, \mathbf{u}_n) c_2 h + f_y(t_n, \mathbf{u}_n) c_2 h \mathbf{U}_1 + O(h^2)
\]

\[
= f(t_n, \mathbf{u}_n) + c_2 h (f_t(t_n, \mathbf{u}_n) + f_y(t_n, \mathbf{u}_n) f(t_n, \mathbf{u}_n)) + O(h^2)
\]
So

\[ b_1 U_1 + b_2 U_2 = (b_1 + b_2) f(t_n, u_n) + b_2 c_2 h (f(t_n, u_n) + f_y(t_n, u_n) f(t_n, u_n)) + O(h^2). \]

Hence the leading terms in the Runge-Kutta method will agree with the second-order Taylor formula (5.16) if

\[ b_1 + b_2 = 1 \text{ and } b_2 c_2 = 1/2. \tag{5.19} \]

Equation (5.19) does not have a unique solution, and the various methods that satisfy (5.19) are all consistent, but can have very different stability properties.

Two well-known second-order Runge-Kutta methods are the **Modified Euler method** \((b_1 = 0, b_2 = 1, c_2 = 1/2)\)

\[
\begin{align*}
U_1 &= f(t_n, u_n), U_2 = f(t_n + h/2, u_n + h/2 U_1), \\
u_{n+1} &= u_n + h U_2 = u_n + h f(t_n + h/2, u_n + h/2 f(t_n, u_n)),
\end{align*}
\tag{5.20}
\]

and **Heun’s method** \((b_1 = b_2 = 1/2, c_2 = 1)\)

\[
\begin{align*}
U_1 &= f(t_n, u_n), U_2 = f(t_n + h, u_n + h U_1), \\
u_{n+1} &= u_n + h/2 (U_1 + U_2).
\end{align*}
\tag{5.21}
\]

**Extrapolation** can also be used to design Runge-Kutta methods. As an example we will derive the modified Euler method by extrapolation of Euler’s method. Assume that \(f\) is Lipschitz continuously differentiable.

The local error for Euler’s method is

\[ h \tau^E(t, h) = y''(t) h^2 / 2 + O(h^3) \]

If we take two steps of size \(h/2\) of of Euler’s method, then the local error is

\[ h \tau^{E2}(t, h) = y(t) + h/2 f(t, y(t)) + h/2 f(t + h/2, y(t) + h/2 f(t, y(t))) - y(t + h). \]

Since

\[ y(t) + h/2 f(t, y(t)) = y(t + h/2) + y''(t) h^2/8 + O(h^3) \]

and

\[ f(t + h/2, y(t) + h/2 f(t, y(t))) = f(t + h/2, y(t + h/2)) + O(h^2) \]

we have

\[
\begin{align*}
bf y(t) + h/2 f(t, y(t)) + h/2 f(t + h/2, y(t) + h/2 f(t, y(t)))
&= y(t + h/2) + y''(t) h^2/8 + f(t + h/2, y(t + h/2)) + y''(t) h^2/8 + O(h^3) \\
&= y(t + h) + y''(t) h^2/4 + O(h^3).
\end{align*}
\]
So,

\[ \tau_L^{E2}(t, h) = y''(t)h^2/4 + O(h^3) = \tau_L^E(t, h)/2 + O(h^3). \]

Hence if

\[ u_{n+1}^E = u_n + hf(t_n, u_n) \]

and

\[ u_{n+1}^{E2} = u_n + (h/2)f(t_n, u_n) + (h/2)f(t_n + h/2, u_n + (h/2)f(t_n, u_n)) \]

then

\[ u_n = 2u_{n+1}^{E2} - u_{n+1}^E \]

\[ = u_n + hf(t_n + h/2, u_n + h/2f(t_n, u_n)), \]

is exactly the modified Euler method, and the local error is \(O(h^3)\), implying that the local truncation error is second order.

**RK4** is the classical fourth order, four-stage Runge Kutta method.

\[ U_1 = hf(t_n, u_n); U_2 = hf(t_n + h/2, u_n + U_1/2); \]
\[ U_3 = hf(t_n + h/2, u_n + U_2/2); U_4 = hf(t_n + h, u_n + U_3); \]
\[ u_{n+1} = u_n + (U_1 + 2U_2 + 2U_3 + U_4)/6. \] (5.22)

### 5.3.5 Stepsize Control and Error Estimation for One-Step Methods

Fixed-step methods are almost never used in practice. The reason for this is that they are either too costly, taking far too many steps for the accuracy that is really needed, or too inaccurate, failing to detect important features of the solution, or both.

There are many ways to estimate the local error and adjust the stepsize as the integration progresses so that you obtain the error you want. The simplest way is to use two methods, one of higher order than the other. If the two methods have order \(p\) and \(p + 1\), then we can estimate the error in the low-order method and vary the step size as follows.

Given a step \(h_n\) and \(u_n\), compute \(u_{n+1}^p\) and \(u_{n+1}^{p+1}\) with the \(p\)th and \(p + 1\)st order methods. Assume that the local error in the \(p\)th order method is

\[ \tau_L(t_n, h_n) = C_Ly^{p+1}(t_n)h_{n+1}^{p+1} + O(h^{p+2}). \]

Even if, as is usually the case, you don’t know \(C_L\) or \(y^{p+1}(t_n)\), you can estimate the local error by

\[ C_Ly^{p+1}(t_n)h_{n+1}^{p+1} \approx u_n^p - u_{n+1}^p. \]

The objective in these methods is to change \(h\) so that the local error is smaller than a tolerance \(\epsilon\). Typically this is expressed as

\[ \|C_Ly^{p+1}(t_n)h_{n+1}^{p+1}\| \leq \sigma\epsilon \] (5.23)
where \( \sigma \) is a safety factor. In many codes \( \sigma = .9 \). If we set \( h_{n+1} = rh_n \) and replace (5.23) by

\[
  r^{p+1} \| u_n^p - u_{n+1}^p \| \leq \sigma \epsilon_r,
\]

(5.24)

we can solve for \( r \) and obtain a new stepsize.

This is implemented in the following way. After taking the step, one verifies that

\[
  \| u_n^p - u_{n+1}^p \| \leq \epsilon_r.
\]

(5.25)

If (5.25) is false, then one reduces \( h_n \) by a factor \( r \) that satisfies (5.24) and tries again. Only after (5.25) is satisfied does one use (5.24) to estimate \( h_{n+1} \) and continue the integration.

The cost of this is that you must do the work for both methods. If the differential equation is not stiff, it is standard to take \( u_{n+1}^{p+1} \), rather than \( u_{n+1}^p \). This practice is called order extrapolation.

Runge-Kutta methods can use some of the same function evaluations for the two methods. The ode45 code in MATLAB does this, obtaining a 4th and 5th order pair with only seven stages, instead of the nine you might expect. Table 5.1 is the Butcher array for the two methods. Note that \( A \) and \( c \) are the same for both, \( b \) is different.

<table>
<thead>
<tr>
<th>Table 5.1: Butcher Array for ode45 RK Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 )</td>
</tr>
<tr>
<td>( \frac{1}{5} )</td>
</tr>
<tr>
<td>( \frac{1}{5} )</td>
</tr>
<tr>
<td>( \frac{3}{10} )</td>
</tr>
<tr>
<td>( \frac{3}{40} )</td>
</tr>
<tr>
<td>( \frac{9}{40} )</td>
</tr>
<tr>
<td>( \frac{4}{5} )</td>
</tr>
<tr>
<td>( \frac{44}{45} )</td>
</tr>
<tr>
<td>( \frac{56}{15} )</td>
</tr>
<tr>
<td>( \frac{32}{9} )</td>
</tr>
<tr>
<td>( \frac{8}{9} )</td>
</tr>
<tr>
<td>( \frac{19372}{6561} )</td>
</tr>
<tr>
<td>( \frac{25360}{2187} )</td>
</tr>
<tr>
<td>( \frac{64448}{6561} )</td>
</tr>
<tr>
<td>( \frac{212}{729} )</td>
</tr>
<tr>
<td>( \frac{1}{1} )</td>
</tr>
<tr>
<td>( \frac{9017}{5168} )</td>
</tr>
<tr>
<td>( \frac{355}{33} )</td>
</tr>
<tr>
<td>( \frac{46372}{5427} )</td>
</tr>
<tr>
<td>( \frac{49}{176} )</td>
</tr>
<tr>
<td>( \frac{5103}{18656} )</td>
</tr>
<tr>
<td>( \frac{1}{1} )</td>
</tr>
<tr>
<td>( \frac{35}{84} )</td>
</tr>
<tr>
<td>( 0 )</td>
</tr>
<tr>
<td>( \frac{500}{1113} )</td>
</tr>
<tr>
<td>( \frac{125}{192} )</td>
</tr>
<tr>
<td>( \frac{2187}{6784} )</td>
</tr>
<tr>
<td>( \frac{11}{84} )</td>
</tr>
<tr>
<td>( \frac{1}{40} )</td>
</tr>
</tbody>
</table>

| \( \frac{5179}{57600} \)                       |
| \( 0 \)                                       |
| \( \frac{7571}{16695} \)                       |
| \( \frac{393}{640} \)                          |
| \( \frac{92097}{339200} \)                     |
| \( \frac{187}{2100} \)                         |

| \( \frac{35}{384} \)                          |
| \( 0 \)                                       |
| \( \frac{500}{1113} \)                         |
| \( \frac{125}{192} \)                          |
| \( \frac{2187}{6784} \)                        |
| \( \frac{11}{84} \)                            |
| \( 0 \)                                       |

45
5.4 Linear Multistep Methods

Linear multistep methods, so called because, unlike the RK methods, the formulae depend on the history of the integration and on evaluations of $f$ in a linear way, have the general form

$$u_{n+1} = \sum_{j=0}^{p} \alpha_j u_{n-j} + h \sum_{j=0}^{p} \beta_j f_{n-j} + h \beta_{-1} f_{n+1}. \quad (5.26)$$

In (5.26)

$$f_i = f(t_i, u_i).$$

Assuming that $\alpha_p$ and $\beta_p$ are not both zero, the method given by (5.26) is called a $p + 1$-step method. The method is explicit if $\beta_{-1} = 0$ and implicit otherwise.

Implicit in the formula is the assumption that the previous $k$ time steps are equally spaced. Multistep methods have the significant advantage over Runge-Kutta methods that high order can be achieved without evaluating $f$ many times at each time step. Instead one uses the history of the integration, the previous $k$ values of $f$ and $y$, and interpolation to obtain high order methods.

The local truncation error for a $p$-step $q$th order multistep method can be written

$$\tau(t, h) = C_{q+1} h^{q+1} y^{(q+1)}(t) + O(h^{q+1}). \quad (5.27)$$

The value of the constant $C_{q+1}$ depends on the method and can be used to compare methods of the same order.

5.4.1 Stability of Linear Multistep Methods

For the one-step methods we were able to describe stability and convergence in an easy way. Then we used the model problem (5.14) to discuss absolute stability. For the multistep methods we must take an extra step. The one-step methods we’ve discussed all solve the very simple scalar initial value problem

$$y' = 0, \quad y(0) = 1, \quad (5.28)$$

with no difficulties at all. Some multistep methods do not and these methods are unstable for all values of $h > 0$.

For (5.28), $f = 0$ and the method becomes

$$u_{n+1} = \sum_{i=0}^{p} \alpha_i u_{n-i}. \quad (5.29)$$

At a minimum, the method should be consistent, that is when the exact solution, $y \equiv 1$, is used for $y$ in (5.29), the equation should be true. This means that

$$\sum_{i=0}^{p} \alpha_i = 1. \quad (5.30)$$

In order for the method to have order $\geq 1$ (i.e. any accuracy at all) we must have

$$-\sum_{j=1}^{p} j \alpha_j + \sum_{j=-1}^{p} \beta_j = 0. \quad (5.31)$$
It is standard to express the two consistency conditions (5.30) and (5.31) in terms of the characteristic polynomials

$$\rho(\xi) = \sum_{i=0}^{p} \alpha_i \xi^{k-i} \quad \text{and} \quad \sigma(\xi) = \sum_{i=-1}^{p} \beta_i \xi^{k-i}. \quad (5.30)$$

(5.30) states that $$\rho(1) = 0$$ and (5.30) that $$\rho'(1) = \sigma(1)$$.

Moreover, any perturbation the constant solution should not lead to instability as the integration progresses. To understand this we try to find all solutions. If we try $$u_j = \xi^j$$, we get lucky. By the fundamental theorem of algebra there are exactly $$p$$ solutions of

$$\sum_{i=0}^{p} \alpha_i \xi^{n-i} = 0, \quad (5.32)$$

namely (after factoring out $$\xi^{n-p}$$) the roots of

$$\rho(\xi) = \sum_{i=0}^{p} \alpha_i \xi^{p-i}. \quad (5.33)$$

If any root of $$\rho$$ is larger than one in absolute value, then a perturbation of (5.28), say from floating point roundoff, can easily make the integration unstable. We fix this idea with a definition.

In some texts (e. g. [1]) 0-stability has a more technical definition and the equivalence of our definition is a theorem.

**Definition 5.4.1** A linear multistep method is **0-stable** if the roots $$\{\xi_i\}_{i=1}^{p}$$ of $$\rho$$ satisfy $$|\xi_i| \leq 1$$ and any root with $$|\xi_i| = 1$$ is simple.

It’s better if the roots (other than $$\xi = 1$$) have absolute value < 1, so that perturbations will decay.

### 5.4.2 Adams Methods

The Adams methods approximate the equation

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

by interpolating $$f(t, y(t))$$ at points $$t_{n-p}, \ldots, t_{n+1}$$ to obtain an interpolating polynomial $$Q$$ and then approximating the solution by

$$u_{n+1} - u_n = \int_{t_n}^{t_{n+1}} Q(t) \, dt. \quad (5.34)$$

This leads to the formula:

$$u_{n+1} = u_n + h \sum_{j=0}^{p} \beta_j f_{n+1-j}. \quad (5.35)$$

The method is explicit (**Adams-Bashforth**) if $$\beta_{-1} = 0$$ and implicit (**Adams-Moulton**) if $$\beta_{-1} \neq 0$$. The coefficients $$\{\beta_j\}$$ in (5.35) and the local error constant $$C_{p+1}$$ in (5.27) can be computed by
interpolation theory. We will refer to the Adams methods by their order, so AM1 is the first order Adams-Moulton method and AB3 is the third order Adams-Bashforth method.

AB1 approximates \( f \) by the constant polynomial that interpolates \( f(t, y(t)) \) at \( t = t_n \). Hence the formula is

\[
u_{n+1} = u_n + hf(t_n, u_n),
\]

which is the forward Euler method. We obtain \( \beta_0 = 1 \) and, from Chapter 5.3.3 the order is 1 and 
\( C_2 = 1/2 \).

AM1 approximates \( f \) by the constant polynomial that interpolates \( f(t, y(t)) \) at \( t = t_{n+1} \). So the formula is

\[
u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}),
\]

which is the backward Euler method. We obtain \( \beta_0 = 1 \) and \( C_2 = 1/2 \).

Note that a \( p \)-step Adams-Bashforth method has order \( q = p \) and a \( p \)-step Adams-Moulton method has order \( q = p + 1 \). AMq is implicit and the nonlinear equation

\[
u_{n+1} - h\beta_0 f(t_n, u_{n+1}) = u_n + h\sum_{j=0}^{q-2} \beta_j f_{n-j} + h\beta_{q-1} f_{n+1}, \tag{5.36}
\]

must be solved to compute \( u_{n+1} \).

### Tables for Adams method coefficients

<table>
<thead>
<tr>
<th>( q )</th>
<th>( \beta_0 )</th>
<th>( \beta_1 )</th>
<th>( \beta_2 )</th>
<th>( \beta_3 )</th>
<th>( \beta_4 )</th>
<th>( \beta_5 )</th>
<th>( C_{q+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 1 )</td>
</tr>
<tr>
<td>2</td>
<td>( \frac{1}{2} )</td>
<td>( -\frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
<td></td>
<td></td>
<td></td>
<td>( 5 )</td>
</tr>
<tr>
<td>3</td>
<td>( \frac{23}{12} )</td>
<td>( -\frac{16}{12} )</td>
<td>( \frac{5}{12} )</td>
<td></td>
<td></td>
<td></td>
<td>( 3 )</td>
</tr>
<tr>
<td>4</td>
<td>( \frac{55}{24} )</td>
<td>( -\frac{59}{24} )</td>
<td>( \frac{37}{24} )</td>
<td>( -\frac{9}{24} )</td>
<td>( \frac{251}{720} )</td>
<td></td>
<td>( 8 )</td>
</tr>
<tr>
<td>5</td>
<td>( \frac{1901}{720} )</td>
<td>( -\frac{2774}{720} )</td>
<td>( \frac{2616}{720} )</td>
<td>( -\frac{1274}{720} )</td>
<td>( \frac{251}{720} )</td>
<td></td>
<td>( 95 )</td>
</tr>
<tr>
<td>6</td>
<td>( \frac{4277}{1440} )</td>
<td>( -\frac{7923}{1440} )</td>
<td>( \frac{9982}{1440} )</td>
<td>( -\frac{7298}{1440} )</td>
<td>( \frac{2877}{1440} )</td>
<td>( -\frac{475}{1440} )</td>
<td>( 19086 )</td>
</tr>
</tbody>
</table>

Note that the 0-step Adams-Moulton method (backward Euler) is included in the Table for consistency.
Table 5.3: AMq coefficients for $q \leq 6$

<table>
<thead>
<tr>
<th>$q$</th>
<th>$\beta_0^{AM}$</th>
<th>$\beta_1^{AM}$</th>
<th>$\beta_2^{AM}$</th>
<th>$\beta_3^{AM}$</th>
<th>$\beta_4^{AM}$</th>
<th>$\beta_5^{AM}$</th>
<th>$C_{p+2}^{AM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$-\frac{1}{2}$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>$-\frac{1}{12}$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>$-\frac{1}{12}$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{5}{12}$</td>
<td>$\frac{8}{12}$</td>
<td>$-\frac{1}{12}$</td>
<td>$\frac{9}{24}$</td>
<td>$\frac{19}{24}$</td>
<td>$-\frac{5}{24}$</td>
<td>$\frac{1}{720}$</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{9}{24}$</td>
<td>$\frac{19}{24}$</td>
<td>$-\frac{5}{24}$</td>
<td>$\frac{251}{1440}$</td>
<td>$\frac{646}{1440}$</td>
<td>$-\frac{264}{1440}$</td>
<td>$\frac{106}{1440}$</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{251}{720}$</td>
<td>$\frac{646}{720}$</td>
<td>$-\frac{264}{720}$</td>
<td>$\frac{106}{720}$</td>
<td>$\frac{720}{720}$</td>
<td>$-\frac{19}{720}$</td>
<td>$\frac{1}{160}$</td>
</tr>
<tr>
<td>6</td>
<td>$\frac{475}{1440}$</td>
<td>$\frac{1427}{1440}$</td>
<td>$-\frac{798}{1440}$</td>
<td>$\frac{482}{1440}$</td>
<td>$\frac{1440}{1440}$</td>
<td>$-\frac{173}{1440}$</td>
<td>$\frac{27}{60480}$</td>
</tr>
</tbody>
</table>

5.4.3 Predictor-Corrector Methods

Adams methods are typically applied in pairs in a **Predictor-Corrector** method. One first takes an Adams-Bashforth step, the **predictor**, and then uses that as an initial iterate for **one** fixed point iteration the **corrector** for the nonlinear equation (5.36) for the Adams-Moulton step.

To obtain a $q$th order AB/AM predictor-corrector, we take one step of AMq to obtain

$$
\hat{u}_{n+1} = u_n + h\sum_{j=0}^{q-1} \beta_j^{AB} f_{n-j}.
$$

(5.37)

The corrector step is a single successive substitution iteration for the nonlinear equation for the AMq step,

$$
u_{n+1} = u_n + h\beta_{-1}^{AM} f(t_{n+1}, \hat{u}_{n+1}) + h\sum_{j=0}^{q-2} \beta_j^{AM} f_{n-j}.
$$

(5.38)

The local truncation error of the ABq-AMq pair is the same as that of AMq. In this way we reduce the error constant of ABq without having to solve (5.36).

One could use a lower-order predictor or a higher-order corrector. The AB$q_B$ - AM$q_M$ pair has order $q_M$ if $q_B \geq q_M - 1$, and order $q_B$ otherwise. The standard implementation has $q_B = q_M$. In this case, the error constant of the combined methods is the same as that of Adams-Moulton, which is smaller than that of Adams-Bashforth.

These methods are sometimes called **PECE methods** because at each step of the integration one uses values of $f$ from previous time steps to **predict** (P) and obtain $\hat{u}_{n+1}$, then **evaluates** (E)
\( f(t_{n+1}, u_{n+1}) \) and uses that evaluation to correct (C) and obtain \( u_{n+1} \). Finally one evaluates (E) \( f(t_{n+1}, u_{n+1}) \) to prepare for the prediction at step \( n+2 \).

These are not implicit methods, the purpose of the corrector iteration is to reduce the error constant, not to enlarge the stability region.

### 5.4.4 BDF Methods

The Adams interpolate past values of \( f \) to approximate the derivative. The BDF (backward difference formula) methods interpolate past values of \( y \) and differentiate the resulting polynomial to approximate \( y' \). These methods are implicit and the formula for a \( p \)th order BDF method is

\[
    u_{n+1} = u_n + \sum_{j=1}^{p} \alpha_j u_{n-j} + h \beta_{-1} f_{n+1}.
\]

The Admas methods use the minimum number of values of \( y \) and the BDF methods use the minimum number of values of \( f \).

<table>
<thead>
<tr>
<th>( q )</th>
<th>( \beta_{-1} )</th>
<th>( \alpha_0 )</th>
<th>( \alpha_1 )</th>
<th>( \alpha_1 )</th>
<th>( \alpha_3 )</th>
<th>( \alpha_4 )</th>
<th>( \alpha_5 )</th>
<th>( C_{q+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X )</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td></td>
<td>( \frac{2}{3} )</td>
<td>( \frac{3}{3} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>18</td>
<td>-9</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td></td>
<td>( \frac{6}{11} )</td>
<td>( \frac{11}{11} )</td>
<td>( \frac{11}{11} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td>4</td>
<td>-12</td>
<td>48</td>
<td>-36</td>
<td>16</td>
<td>-3</td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td></td>
<td>( \frac{-12}{25} )</td>
<td>( \frac{25}{25} )</td>
<td>( \frac{25}{25} )</td>
<td>( \frac{25}{25} )</td>
<td></td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td>5</td>
<td>-60</td>
<td>300</td>
<td>-300</td>
<td>200</td>
<td>-75</td>
<td>12</td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td></td>
<td>( \frac{-60}{137} )</td>
<td>( \frac{137}{137} )</td>
<td>( \frac{137}{137} )</td>
<td>( \frac{137}{137} )</td>
<td>( \frac{137}{137} )</td>
<td></td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td>6</td>
<td>-60</td>
<td>360</td>
<td>-450</td>
<td>400</td>
<td>-225</td>
<td>72</td>
<td>-10</td>
<td>( X ) ( Y )</td>
</tr>
<tr>
<td></td>
<td>( \frac{-60}{147} )</td>
<td>( \frac{147}{147} )</td>
<td>( \frac{147}{147} )</td>
<td>( \frac{147}{147} )</td>
<td>( \frac{147}{147} )</td>
<td>( \frac{147}{147} )</td>
<td></td>
<td>( X ) ( Y )</td>
</tr>
</tbody>
</table>

BDF methods are intended to be used as implicit methods, so a nonlinear equation must be solved at each time step.
5.5 Using Codes

In this section we will discuss how codes for initial value problems operate. As an example we will use the MATLAB codes ode45 and ode15s. These codes are documented in the MATLAB online help information and more completely in [13].

These are variable stepsize codes. This means that the step $t_{n+1} - t_n$ is changed as the integration progresses in response to estimates of the local error. You can obtain values of the approximate solution an any sequence of time values you like with the appropriate input.

This information will help you design your own codes and evaluate their performance. Much of the input and output of the two codes is the same. You can get a good start with the calling sequence

$$[t_{mesh}, y_{sol}] = 	ext{ode45}(y_{fun}, t_{mesh}, y_0, \text{options});$$

5.5.1 Input

- $y_{fun}$ is the name of the MATLAB function of $t$ and $y$ that evaluates the right side of (1.7). $y_{fun}$ must return a column vector.
- $t_{mesh}$ is a vector of $t$ values for which a solution $y_{sol}$ will be reported.
- $y_0$ is the initial data from (1.8). $y_0$ must be a column vector.
- $\text{options}$ is an array of control parameters for the code. Using the options array is not simple. As a start, you might use it only to control the tolerances for the integration. To set both the relative and absolute local errors to $\text{tol}$, try

$$\text{options} = \text{odeset}('\text{RelTol}', \text{tol}, '\text{AbsTol}', \text{tol});$$

5.5.2 Output

Usually $t_{mesh}$ is the same as the input. $t_{mesh}$ can also be used to see what steps the code actually took.

$y_{sol}$ is the array of solution values at the given time points $t_{mesh}$. If

$$t_{mesh} = [t_0, \ldots, t_N]^T$$

and (1.7) is a system of $M$ equations. $y_{sol}$ is an $N \times M$ array, where the $k$th row is the approximation of $y(t_k)^T$ (here $y^T$ is the transpose of the column vector $y$). The MATLAB notation for $y^T$ is $y'$, which we use for derivative.

ode45 is a variable step Runge-Kutta code, ode15s is a variable step and variable order stiff integrator.
5.5.3 A third order IVP

We will solve the third order initial value problem

\[ x''' + x'' + x = \sin(t), \quad x(0) = 1, \quad x'(0) = 0, \quad x''(0) = -1 \]

with \texttt{ode45} by converting it into a third order system. Let \( y = (x, x', x'')^T \). \( y \) satisfies the first order system

\[
y' = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = f(t,y) = \begin{pmatrix} y_2 \\ y_3 \\ \sin(t) - y_1 - y_3 \end{pmatrix}
\]

The solution is \( x(t) = \cos(t) \), so

\[
y(t) = \begin{pmatrix} \cos(t) \\ -\sin(t) \\ -\cos(t) \end{pmatrix},
\]

and the initial data is

\[
y_0 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix},
\]

A MATLAB code that evaluates \( f \) and returns a \texttt{column} vector is:

```matlab
% sys1.m
% Convert y''' + y'' + y = \sin(t) into a system of three equations in the form y' = f(t,y);
function yp=sys1(t,y)
% make sure it's a column vector
yp=zeros(3,1);
yp(1) = y(2);
y(1) = y(3);
y(2) = \sin(t) - y(1) - y(3);
```

We’ll solve this initial value problem on the interval

\[
[t_0, t_{final}] = [0, 2],
\]

and report results at 200 equally spaced points for plotting.

Finally, here’s a MATLAB fragment that calls \texttt{ode45} and \texttt{ode15s} to solve the problem.
% Set the parameters for the codes. All I’ve done is set relative % and absolute error tolerances. There’s a lot more you can do. 
% 
% tol=1.d-1; 
for it=1:4 
  options=odeset(’RelTol’,tol,’AbsTol’,tol); 
%
% I want the values of the solution at 200 evenly spaced points for % plotting and comparison with the true solution 
%
% tmesh=0:.01:2; tmesh=tmesh’; 
%
% Make sure the initial vector is a column vector.  
%
% y0=[1,0,-1]’; 
%
% Now call the solver 
%
[tmesh, ysol]=ode45(’sys1’,tmesh,y0,options);

5.6 Notes

There are many good books on this topic. I particularly like [1] and [12]. [6], while somewhat out-of-date, is a classic. [3] is a clearly written advanced book.

5.7 Exercises

5.7.1. Prove (5.8).
Bibliography


Index

$p + 1$-step method, 46
$s$-stage Runge-Kutta method, 42
$0$-stable, 39, 47
Adams-Bashforth, 47
Adams-Moulton, 47
Analytic extrapolation, 31
BDF methods, 50
Butcher array, 42
Captain Marvel, 32
Cauchy sequence, 9, 10
Composite midpoint rule, 26
Composite quadrature rules, 25
Composite trapezoid rule, 27
Consistent method, 38
\hspace{1cm} order $p$, 38
Contraction mapping, 9
Contraction Mapping Theorem, 9
Convergence theorem
\hspace{1cm} contraction mapping, 9
Discrete Gronwall inequality, 39
Divided difference table, 14
Euler method
\hspace{1cm} backward, 37, 41
\hspace{1cm} explicit, 37, 40
\hspace{1cm} forward, 37, 40
\hspace{1cm} implicit, 37, 41
Explicit method, 37
Extrapolation, 31
\hspace{1cm} analytic, 31
\hspace{1cm} Euler’s method, 43
\hspace{1cm} numerical, 32
Extrapolation table, 31
Fixed point, 8
Fixed point iteration, 8
Fixed step method, 36
Gaussian quadratures, 32
Global error, 36
Global truncation error, 38
Gomer Pyle, 32
Grid refinement study, 22
Hermite interpolation, 15
Hermite quadrature rule, 34
Heun’s method, 43
Implicit method, 37
Initial condition, 5
Initial value problem
\hspace{1cm} first order systems, 5
Initial Value Problems
\hspace{1cm} first order systems, 5
\hspace{1cm} high order, 5
Intermediate Value Theorem, 2
Interpolation, 11
\hspace{1cm} abscissae, 11
\hspace{1cm} nodeal values, 11
\hspace{1cm} nodes, 11
Interpolatory quadrature rules, 25
Iterative method, 8
Lagrange Polynomials, 11
Linear multistep methods, 46
Linear multistep methods:characteristic polynomials, 47
Lipschitz constant, 5
Lipschitz continuous
\hspace{1cm} definition, 8
Local error, 38
Local truncation error, 38, 46
Mean Value Theorem, 3
Midpoint rule, 26
  composite, 26
Modified Euler method, 43
Natural spline, 20
Newton-Cotes rules, 29
Nonlinear equations, 8
Not-a-knot, 20

One step methods, 38
Order extrapolation, 45
Order of integrator, 38
Orthogonal polynomials, 33
Orthogonality condition, 33

PECE methods, 49
Picard iteration, 8
Predictor-Corrector methods, 49

Quadrature rule, 25
  composite, 25
  interpolatory, 25
  midpoint, 26
  nodes, 25
  precision, 25
  weights, 25

Richardson iteration
  nonlinear, 8
RK4, 44
Rolle’s Theorem, 2
Romberg integration, 32
Root, 8
Runge-Kutta Methods, 42
Runge-Kutta methods, 42

Simpson’s rule, 28, 32
Splines, 17
  $C^2$ cubic, 18
  piecewise Hermite, 18
  piecewise linear, 17
Successive substitution, 8

Taylor Polynomial, 4
Taylor’s Theorem, 3, 4
Temporal sampling points, 36
Time step, 36
Trapezoid rule, 27
  composite, 27

Variable step method, 36
Weight functions, 33

57