Chapter 5

Multivalue or Multistep Methods

5.1 Introduction

One-step methods only require information about the solution at one time, say \( t = t_{n-1} \) to compute the solution at an advanced time \( t = t_n \). After integrating away from the initial point, we have several solution values that can be used to predict the solution at an advanced point. In this Chapter, we use these past solution values to construct “multistep” methods that potentially have fewer function evaluations per time step than one-step methods. To begin, let us define a linear multistep method for the scalar IVP

\[
y' = f(t, y), \quad t > 0, \quad y(0) = y_0
\]

as

\[
\sum_{i=0}^{k} \alpha_i y_{n-i} = h \sum_{i=0}^{k} \beta_i f_{n-i}, \quad (5.1.2a)
\]

where

\[
f_j = f(t_j, y_j). \quad (5.1.2b)
\]

In applying (5.1.2), we assume that approximate solutions \( y_{n-1}, y_{n-2}, \ldots, y_{n-k} \) are known and that we seek to calculate \( y_n \) (Figure 5.1.1). In particular, \( y_0, y_1, \ldots, y_{k-1} \) must be known in order to start using the method. We’ll discuss methods for obtaining these “starting values;” however, for the moment, assume that they are calculated by a one-step method. Some other assumptions and observations concerning (5.1.2) follow.
Figure 5.1.1: Domain of a \( k \)-step linear multistep method.

1. The step size \( h \) has been assumed to be constant for all steps. Designing variable step formulas and changing step sizes will be more complex than with one-step methods. We’ll return to this subject later.

2. We normalize (5.1.2b) by selecting
\[
\alpha_0 = 1
\]
and also assume that at least one other \( \alpha_i, i = 1, 2, \ldots, k \), or \( \beta_k, i = 0, 1, \ldots, k \), is nonzero.

3. If \( \beta_0 = 0 \) the method is explicit, otherwise it is implicit. For an explicit method, \( y_n \) may be directly obtained in terms of \( y_{n-i} \) and \( f_{n-i}, i = 1, 2, \ldots, k \). An implicit method generally requires the solution of a nonlinear problem.

We’ll discuss systematic approaches to constructing multistep methods in the next two sections; however, let us illustrate the approach using the method of undetermined coefficients. With this technique we

1. assume a particular form of the general formula (5.1.2) by, possibly, restricting some of the coefficients and
2. determine the remaining coefficients of (5.1.2) such that they match terms of a Taylor’s series expansion of the exact ODE solution to as high a degree as possible. Equivalently, the coefficients can be determined so that (5.1.2) produces the exact ODE solution when \( y(t) \) is a polynomial to as high a degree as possible.

Here’s an example.

**Example 5.1.1.** Consider a multistep method of the form

\[
y_n + \alpha_1 y_{n-1} + \alpha_2 y_{n-2} = h \beta_1 f_{n-1}.
\]

This explicit two-step formula has three undetermined coefficients \((\alpha_1, \alpha_2, \beta_1)\) and we’ll determine them so that the numerical method is exact when \( y(t) \) is an arbitrary quadratic polynomial. Since the multistep method is linear, it suffices to make the formula exact when \( y(t) \) is 1, \( t \), and \( t^2 \). If \( y(t) = 1 \) then \( f(t, y) = 0 \) and (5.1.2) yields

\[
1 + \alpha_1 + \alpha_2 = 0. \tag{5.1.3a}
\]

When \( y(t) = t \), \( f(t, y) = 1 \) and (5.1.2) yields

\[
t_n + \alpha_1 (t_n - h) + \alpha_2 (t_n - 2h) = h \beta_1.
\]

Using (5.1.3a),

\[
-\alpha_1 - 2\alpha_2 = \beta_1. \tag{5.1.3b}
\]

When \( y(t) = t^2 \), \( f(t, y) = 2t \) and (5.1.2) yields

\[
t_n^2 + \alpha_1 (t_n - h)^2 + \alpha_2 (t_n - 2h)^2 = 2h \beta_1 (t_n - h).
\]

This may be simplified by (5.1.3a,b) to

\[
\alpha_1 + 4\alpha_2 = -2\beta_1. \tag{5.1.3c}
\]

The solution of (5.1.3a,b,c) is

\[
\alpha_1 = 0, \quad \alpha_2 = -1, \quad \beta_1 = 2;
\]
Thus, the method is

\[ y_n = y_{n-2} + 2hf_{n-1}, \quad n = 2, 3, \ldots \]  

(5.1.4a)

This scheme is called the “leap frog” scheme. It has only one function evaluation per step and, once started (with \(y_0\) and \(y_1\)), is as simple as the explicit Euler method.

Without having introduced a formal definition of the local discretization error for multistep methods, let’s use our experience with one-step methods to define it for the leap frog method as

\[ \tau_n = \frac{y(t_n) - y(t_{n-2})}{2h} - f(t_{n-1}, y(t_{n-1})). \]

Since (5.1.4a) is exact when \(y(t)\) is a quadratic polynomial, we may either infer or a Taylor’s series expansion to show that

\[ \tau_n = Ch^2 y''(\xi_n), \quad \xi_n \in (t_{n-2}, t_n). \]

The numerical constant \(C\) may also be determined by the method of undetermined coefficients. To do this, we select the simplest ODE solution for which the numerical method does not produce an exact solution. In the case of the leap frog method, this would be any cubic polynomial, and we select \(y(t) = t^3\). Substitution into the two expressions for the local discretization error yields

\[ \frac{t_3^3 - (t_n - 2h)^3}{2h} - 3(t_n - h)^2 = 6Ch^2. \]

The location of the point \(\xi_n\) is irrelevant for this exact solution and we find \(C = 1/3\). Thus,

\[ \tau_n = \frac{h^2}{3} y''(\xi_n), \quad \xi_n \in (t_{n-2}, t_n). \]  

(5.1.4b)

**Problems**

1. Using the method of undetermined coefficients, we can make some general observations regarding the coefficients of (5.1.2). Show, for example, that

\[ \sum_{i=0}^{k} \alpha_i = 0 \]  

(5.1.5a)
if (5.1.2) is exact when $y(t) = 1$. Additionally, if (5.1.2) is exact when $y(t) = t^q$, $q = 1, 2, \ldots$, show

$$
\sum_{i=1}^{k} i^q \alpha_i + q \sum_{i=0}^{k} i^{q-1} \beta_i = 0, \quad q = 1, 2, \ldots
$$

(5.1.5b)

These are order conditions for the linear multistep method.

### 5.2 Newton Divided-Difference Polynomials

Specific multistep formulas will be derived by approximating $y(t)$ or $f(t, y)$ by interpolating polynomials and, respectively, differentiating or integrating these polynomials. We’ve already seen examples where one-step methods were constructed by using interpolating polynomials with collocation. Thus, in this section, we’ll review polynomial interpolation in an abstract setting that is removed from our primary task of solving differential equations.

The interpolation problem consists of finding a polynomial $P_k(t)$ of degree $k$ that interpolates a function $f(t)$ at $k + 1$ distinct points $t_0, t_1, \ldots, t_k$, i.e.,

$$
f(t_j) = P_k(t_j), \quad j = 0, 1, \ldots, k.
$$

(5.2.1)

In practice, we express the polynomial in a convenient basis for the application. The obvious basis of powers of monomial terms

$$
P_k(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_k t^k = \sum_{i=0}^{k} a_i t^i
$$

(5.2.2)

leads to the linear algebraic system

$$
f(t_0) = a_0 + a_1 t_0 + a_2 t_0^2 + \ldots + a_k t_0^k,$n$$

$$
f(t_1) = a_0 + a_1 t_1 + a_2 t_1^2 + \ldots + a_k t_1^k,$n$$

$$
\vdots
$$

$$
f(t_k) = a_0 + a_1 t_k + a_2 t_k^2 + \ldots + a_k t_k^k,$n$$

5
and is rarely convenient. The Lagrange basis

\[
L_i(t) = \prod_{j=1,j \neq i}^{k} \frac{t-t_j}{t_i-t_j} = \frac{(t-t_0)(t-t_1)\ldots(t-t_{i-1})(t-t_{i+1})\ldots(t-t_k)}{(t_i-t_0)(t_i-t_1)\ldots(t_i-t_{i-1})(t_i-t_{i+1})\ldots(t_i-t_k)}
\]

\[i = 0,1,\ldots,k,
\]

leads to the more direct solution

\[
P_k(t) = \sum_{i=0}^{k} f(t_i) L_i(t).
\]

A disadvantage of the Lagrange basis is that the computation of the basis elements \(L_i(t), i = 0,1,\ldots,k\), must be repeated when changing the polynomial degree by adding or deleting an interpolation point. As discussed in Chapter 4, the Aitken-Neville recursion remedies this by obtaining \(P_k(t)\) as a combination of polynomials of degree \(k-1\). Thus, we define the family of interpolants

\[
P_j^i(t) = (t-t_{i+j})P_j^{i-1}(t) + (t_i-t)P_{j-1}^{i+1}(t)i_j - t_j + j,
\]

\[i = 0,1,\ldots,k-j,
\]

\[j = 0,1,\ldots,k,
\]

where \(P_j^i(t)\) is a polynomial of degree \(j\) satisfying the interpolation conditions

\[
P_j^i(t_m) = f(t_m), \quad m = i,i+1,\ldots,i+j.
\]

The desired interpolating polynomial is \(P_k^0(t)\). The Aitken-Neville algorithm simplifies degree changes, but its complexity is slightly larger than necessary ([6], Chapter 3). The Newton form of the interpolating polynomial has a lower complexity and, like the Aitken-Neville procedure, is hierarchical. Again we’ll define a sequence of polynomials of increasing degree with

\[
P_0(t) = f(t_0).
\]

The degree one polynomial is obtained by adding a correction to \(P_0(t)\) in the form

\[
P_1(t) = P_0(t) + a_1(t - t_0).
\]

First, we easily verify that

\[
P_1(t_0) = P_0(t_0) = f(t_0).
\]
The coefficient \( a_1 \) is determined so that

\[ P_1(t_1) = f(t_1); \]

thus, using (5.2.5),

\[ f(t_1) = f(t_0) + a_1(t_1 - t_0) \]

and

\[ a_1 = \frac{f(t_1) - f(t_0)}{t_1 - t_0}. \]

The notation is simplified by defining the first divided difference at the points \( t_j \) and \( t_l \) as

\[ f[t_j, t_l] = \frac{f(t_j) - f(t_l)}{t_j - t_l}. \] (5.2.6a)

Then

\[ P_1(t) = P_0(t) + f[t_0, t_1](t - t_0). \] (5.2.6b)

In a similar manner, the second-degree polynomial is written in the form

\[ P_2(t) = P_1(t) + a_2(t - t_0)(t - t_1). \]

By construction, \( P_2(t) \) satisfies the interpolation requirements at \( t_0 \) and \( t_1 \). Satisfaction of the interpolation condition \( P_2(t_2) = f(t_2) \) determines \( a_2 \) as

\[ a_2 = f[t_0, t_1, t_2] \]

where \( f[t_j, t_t, t_m] \) is the second divided difference at the points \( t_j, t_t, t_m \)

\[ f[t_j, t_t, t_m] = \frac{f[t_j, t_t] - f[t_t, t_m]}{t_j - t_m}. \] (5.2.7a)

Thus, the second divided difference is a divided difference of first divided differences. The second-degree interpolating polynomial is

\[ P_2(t) = P_1(t) + f[t_0, t_1, t_2](t - t_0)(t - t_1). \] (5.2.7b)

Continuing, we construct the \( k \) th-degree polynomial as

\[ P_k(t) = P_{k-1}(t) + f[t_0, t_1, \ldots, t_k](t - t_0)(t - t_1) \ldots (t - t_k - 1) \] (5.2.8a)
where

\[ f[t_0, t_1, \ldots, t_k] = \frac{f[t_0, t_1, \ldots, t_{k-1}] - f[t_1, t_2, \ldots, t_k]}{t_0 - t_k} \quad (5.2.8b) \]

is the \( k \) th divided difference at the points \( t_0, t_1, \ldots, t_k \).

The intermediate polynomials can be eliminated from (5.2.8a) to write the Newton divided-difference polynomial in the more explicit form

\[ P_k(t) = \sum_{i=0}^{k} f[t_0, t_1, \ldots, t_i] \prod_{j=0}^{i-1} (t - t_j) \quad (5.2.9) \]

where the zeroth divided difference is \( f[t_j] = f(t_j) \).

The Newton divided-difference representation is the traditional interpolating polynomial to be used when developing multistep formulas. It had some advantages for hand computation and the divided differences furnish approximations of solution derivatives that may, e.g., be used for error estimation. Before specializing the approximation (5.2.9) to our ODE application, let us note that the interpolation problem has a unique solution as indicated by the following theorem. The different bases just simplify the interpolation problem for specific applications.

**Theorem 5.2.1.** There is one and only one polynomial of degree \( k \) that interpolates a function \( f(t) \) at \( k + 1 \) distinct points.

**Proof.** Suppose that there are two polynomials \( P_k(t) \) and \( Q_k(t) \) of degree \( k \) that interpolate \( f(t) \) at the points \( t_0, t_1, \ldots, t_k \). Subtract the two polynomials and define

\[ R(t) = P_k(t) - Q_k(t). \]

Now \( R(t) \) is also a polynomial of degree \( k \) that satisfies

\[ R(t_j) = P_k(t_j) - Q_k(t_j) = 0, \quad j = 0, 1, \ldots, k. \]

This, however, is impossible since a polynomial of degree \( k \) can only have \( k \) roots. Thus, the interpolating polynomial is unique. \( \square \)

We may suspect that divided differences are related to derivatives and the following Lemma shows that this is the case.
Lemma 5.2.1. Let $f(t) \in C^k[a,b]$ and let $t_0, t_1, \ldots, t_k$ be $k + 1$ distinct points on $[a, b]$, then

$$f[t_0, t_1, \ldots, t_k] = \frac{f^{(k)}(\xi)}{k!}, \quad \xi \in (a, b).$$  \hfill (5.2.10)

Proof. Consider the function

$$g(t) = f(t) - P_k(t).$$

Since $f(t_j) = P_k(t_j)$, $j = 0, 1, \ldots, k$, the function $g(t)$ has $k + 1$ distinct zeros in $[a, b]$. Thus, according to Rolle’s theorem, $g'(t)$ vanishes at $k$ distinct points on $(a, b)$ (Figure 5.2.1). Similarly, $g''(t)$ vanishes at $k - 1$ points on $(a, b)$ and, continuing in this manner, $g^{(k)}$ will vanish at one point on $(a, b)$. Let us call this point $\xi$. Thus,

$$g^{(k)}(\xi) = f^{(k)}(\xi) - P_k^{(k)}(\xi) = 0$$

According to (5.2.8a) or (5.2.9), the $k$th derivative of $P_k(t)$ is

$$P_k^{(k)}(t) = k! f[t_0, t_1, \ldots, t_k]$$

Combining the above two results establishes the result. \hfill \square

With Lemma 5.2.1, we can obtain formulas for interpolation errors.

Theorem 5.2.2. Let $f(t) \in C^{k+1}[a,b]$ and let $t_0, t_1, \ldots, t_k$ be $k + 1$ distinct points on $[a, b]$, then there exists a point $\xi = \xi(t) \in (a, b)$ such that

$$E_k(t) = f(t) - P_k(t) = \frac{f^{(k+1)}(\xi)}{(k + 1)!} \prod_{j=0}^{k} (t - t_j).$$ \hfill (5.2.11)

Proof. Construct a polynomial $P_{k+1}(t)$ that interpolates $f(t)$ at $t_0, t_1, \ldots, t_k$ and the additional point $\tau$. According to (5.2.8a),

$$P_{k+1}(t) = P_k(t) + f[t_0, t_1, \ldots, t_k, \tau] \prod_{j=0}^{k} (t - t_j).$$

Since $P_{k+1}(\tau) = f(\tau)$, we have

$$f(\tau) = P_k(\tau) + f[t_0, t_1, \ldots, t_k, \tau] \prod_{j=0}^{k} (\tau - t_j).$$
Figure 5.2.1: Zeros and extrema of the function \( g(t) = f(t) - P_k(t) \).

Thus,

\[
E_k(\tau) = f(\tau) - P_k(\tau) = f[t_0, t_1, \ldots, t_k, \tau] \prod_{j=0}^{k} (\tau - t_j).
\]
Using (5.2.10)
\[ E_k(\tau) = \frac{f^{(k+1)}(\xi)}{(k+1)!} \prod_{j=0}^{k} (\tau - t_j). \]
Since \( \tau \) is arbitrary we have established the result.

The points in the divided-difference polynomial have to be distinct but do not have to be uniform or ordered. However, the computation simplifies greatly when the interpolation points are ordered and uniformly spaced. This will be our main interest when developing formulas for ODEs, so we’ll restrict
\[ t_i = t_0 + ih, \quad i = 0, 1, \ldots, k. \]
With this, let us define the first forward and backward difference operators as
\[ \Delta f_i = f_{i+1} - f_i, \quad (5.2.13a) \]
\[ \nabla f_i = f_i - f_{i-1}. \quad (5.2.13b) \]
Although both operators are used, backward differences better suit our current needs (because multistep methods will be interpolating with data at prior times). Thus, using (5.2.6a) and (5.2.13b), let us write the first divided difference with uniform spacing as
\[ f[t_{i-1}, t_i] = \frac{f_{i-1} - f_i}{t_{i-1} - t_i} = \frac{\nabla f_i}{h}. \quad (5.2.14) \]
Higher-order operators follow by iteration; thus, using (5.2.7a)
\[ f[t_{i-2}, t_{i-1}, t_i] = \frac{f[t_{i-2}, t_{i-1}] - f[t_{i-1}, t_i]}{t_{i-2} - t_i} = \frac{\nabla f_{i-1} - \nabla f_{i-1}}{2h^2} = \frac{f_i - 2f_{i-1} + f_{i-2}}{2h^2}. \]
In a similar manner, we define the higher-order backward differences recursively as
\[ \nabla^m f_i = \nabla^{m-1} f_i - \nabla^{m-1} f_i - 1 \quad (5.2.15a) \]
with the understanding that
\[ \nabla^0 f_i = f_i. \quad (5.2.15b) \]
Thus,
\[ f[t_{i-2}, t_{i-1}, t_i] = \frac{\nabla^2 f_i}{2h^2}. \quad (5.2.15c) \]

The \( k \) th divided difference and \( k \) th backward difference operator are related by (Problem 1)
\[ f[t_{i-k}, t_{i-k+1}, \ldots, t_i] = \frac{\nabla^k f_i}{k!h^k}. \quad (5.2.15d) \]

We can easily write the divided-difference polynomial (5.2.9) in terms of backward differences; however, our intended use with ODEs will call for polynomials proceeding from an advanced point \((t_{n-1} \text{ or } t_n)\) into the past. Hence, it will be convenient to re-index the points in (5.2.9) to account for this. For the present, we’ll illustrate this by reversing the indexing; thus, let \( t_0 \) become \( t_k \), \( t_1 \) become \( t_{k-1} \) etc. Then, (5.2.9) becomes
\[ P_k(t) = \sum_{i=0}^{k} f[t_k, t_{k-1}, \ldots, t_{k-i}] \prod_{j=1}^{i-1} (t - t_{k-j}) \]
or
\[ P_k(t) = f[t_k] + f[t_k, t_{k-1}] (t - t_k) + \ldots + f[t_k, t_{k-1}, \ldots, t_0] (t - t_k) \ldots (t - t_1). \]

Now, for uniform spacing, we use (5.2.15) to obtain
\[ P_k(t) = \sum_{i=0}^{k} \frac{\nabla^i f_k}{i!h^i} \prod_{j=0}^{i-1} (t - t_{k-j}) \quad (5.2.16a) \]
or
\[ P_k(t) = f_k + \frac{\nabla f_k}{h} (t - t_k) + \ldots + \frac{\nabla^k f_k}{k!h^k} (t - t_k) \ldots (t - t_1). \quad (5.2.16b) \]

Since our primary purpose is the development of multistep methods for ODEs, we won’t illustrate any interpolation examples at this point but will illustrate use of (5.2.16) in subsequent sections.

**Problems**

1. **Problem 1.** Derive the identity (5.2.15d) using, e.g., an induction argument.
5.3 Explicit Methods: Adams-Bashforth Methods

The first multistep formulas that we consider are explicit methods called Adams-Bashforth methods. These are derived by integrating the ODE

\[ y' = f(t, y) \]  

(5.3.1)

on the interval \((t_{n-1}, t_n)\) to obtain

\[ \int_{t_{n-1}}^{t_n} y' \, dt = \int_{t_{n-1}}^{t_n} f(t, y(t)) \, dt \]

or

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

(5.3.2)

Specific numerical techniques are obtained by approximating \(f(t, y(t))\) in (5.3.2) by an interpolating polynomial and integrating the result. If, for example, \(f(t, y(t))\) were approximated by (the constant interpolating polynomial) \(f(t, y(t_{n-1}))\) then (5.3.2) would produce Euler’s method

\[ y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}). \]

More generally, we’ll interpolate \(f(t, y(t))\) by a \(k - 1\)st degree polynomial passing through \(t_{n-1}, t_{n-2}, \ldots, t_{n-k}\). We’ll use the Newton backward-difference form of the interpolating polynomial. For this application, we identify interpolation point \(t_k\) in (5.2.16) with \(t_{n-1}, t_{k-1} \) with \(t_{n-2}, \ldots,\) and \(t_1\) with \(t_{n-k}\). Then, (5.2.16) becomes

\[ P_{k-1}(t) = \sum_{i=0}^{k-1} \frac{\nabla_i f_{n-1}}{i!h^i} \prod_{j=0}^{i-1}(t - t_{n-1-j}). \]  

(5.3.3a)

Using (5.2.11) with \(k\) replaced by \(k - 1\) and re-indexing the interpolation points as described above, we determine the error of this interpolation as

\[ E_{k-1}(t) = f(t, y(t)) - P_{k-1}(t) = \frac{f^{(k)}(\xi, y(\xi))}{k!} \prod_{j=0}^{k-1}(t - t_{n-1-j}), \quad \xi \in (t_{n-k}, t_{n-1}). \]  

(5.3.3b)

Let’s expand both (5.3.3a) and (5.3.3b) to reveal their structure

\[ P_{k-1}(t) = f_{n-1} + \frac{\nabla f_{n-1}}{h}(t - t_{n-1}) + \frac{\nabla^2 f_{n-1}}{2!h^2}(t - t_{n-1})(t - t_{n-2}) \]

\[ + \cdots + \frac{\nabla^{k-1} f_{n-1}}{(k-1)!h^{k-1}}(t - t_{n-1})(t - t_{n-2}) \cdots (t - t_1). \]
+ \ldots + \frac{\nabla^{k-1} f_{n-1}}{(k-1)! h^{k-1}} (t - t_{n-1})(t - t_{n-2}) \ldots (t - t_{n-k+1})

and

\[ E_{k-1}(t) = \frac{f^{(k)}(\xi, y(\xi))}{k!} (t - t_{n-1})(t - t_{n-2}) \ldots (t - t_{n-k}). \]

Using (5.2.14, 5.2.15), the first few divided differences are

\[ \nabla^0 f_{n-1} = f_{n-1}, \quad \nabla f_{n-1} = f_{n-1} - f_{n-2}, \]

\[ \nabla^2 f_{n-1} = \nabla f_{n-1} - \nabla f_{n-2} = f_{n-1} - 2f_{n-2} + f_{n-3}. \]

The integration (5.3.2) can be simplified by the change of variables

\[ \tau = \frac{t - t_{n-1}}{h} \] (5.3.4a)

Since \( t_{n-1-j} = t_{n-1} - j h \), we have

\[ t - t_{n-1-j} = (\tau + j) h \]

and

\[ \frac{1}{h^i} \prod_{j=0}^{i-1} (t - t_{n-1-j}) = \prod_{j=0}^{i-1} (\tau + j) = \tau(\tau + 1) \ldots (\tau + i - 1). \]

Recall the combination symbol

\[ \binom{\tau}{i} = \frac{\tau!}{i!(\tau - i)!} = \frac{\tau(\tau - 1) \ldots (\tau - i + 1)}{i!}, \quad \binom{\tau}{0} = 1. \] (5.3.4b)

This formula makes sense when \( \tau \) is negative, i.e.,

\[ \binom{-\tau}{i} = \frac{-\tau(-\tau - 1) \ldots (-\tau - i + 1)}{i!} = (-1)^i \frac{\tau(\tau + 1) \ldots (\tau + i - 1)}{i!}. \]

Thus,

\[ \frac{1}{h^i} \prod_{j=0}^{i-1} (t - t_{n-1-j}) = (-1)^i \binom{-\tau}{i}. \] (5.3.4c)

Substituting (5.3.4c) into (5.3.3a,b) yields

\[ P_{k-1}(t) = \sum_{i=0}^{k-1} (-1)^i \binom{-\tau}{i} \nabla^i f_{n-1} \] (5.3.5a)
and, using (5.3.1),

\[ E_{k-1}(t) = (-1)^k h^k \left( \frac{-\tau}{k} \right) y^{(k+1)}(\xi). \]  

(5.3.5b)

Replacing \( f(t, y(t)) \) in (5.3.2) by \( P_{k-1}(t) + E_{k-1}(t) \) and transforming the integrals from \( t \) to \( \tau \) using (5.3.4a) yields

\[
y(t_n) = y(t_{n-1}) + h \int_0^1 \sum_{i=0}^{k-1} (-1)^i \left( \frac{-\tau}{i} \right) \nabla^i f_{n-1} d\tau + h \int_0^1 (-1)^k h^k \left( \frac{-\tau}{k} \right) y^{(k+1)}(\xi) d\tau.
\]

Let

\[
\gamma_m = (-1)^m \int_0^1 \left( \frac{-\tau}{m} \right) d\tau.
\]

(5.3.6a)

Then we have

\[
y(t_n) = y(t_{n-1}) + h \sum_{i=0}^{k-1} \gamma_i \nabla^i f_{n-1} + (-1)^k h^{k+1} \int_0^1 \left( \frac{-\tau}{k} \right) y^{(k+1)}(\xi) d\tau.
\]

(5.3.6b)

Dropping the error term yields the \( k \)-step Adams-Bashforth method

\[
y_n = y_{n-1} + h \sum_{i=0}^{k-1} \gamma_i \nabla^i f_{n-1}.
\]

(5.3.7a)

The local error of this formula is

\[
d_n = (-1)^k h^{k+1} \int_0^1 \left( \frac{-\tau}{k} \right) y^{(k+1)}(\xi) d\tau.
\]

(5.3.7b)

Comparing (5.3.7a) with the general multistep formula (5.1.2a), we see that \( \alpha_0 = 1 \), \( \alpha_1 = -1 \), and \( \alpha_2 = \alpha_3 = \ldots = \alpha_k = \beta_0 = 0 \). Thus, (5.3.7a) uses the one solution value \( y_{n-1} \) and the \( k \) function values \( f_{n-1}, f_{n-2}, \ldots, f_{n-k} \) to obtain the \( y_n \) (cf. Figure 5.3.1). We’ll call (5.3.7a) a \( k \)-step method since it uses solution and function values at the \( k \) points \( t_{n-1}, t_{n-2}, \ldots, t_{n-k} \); however, other terminology is also used [10].

From (5.3.7b) we see that the local error of (5.3.7a) is \( O(h^{k+1}) \). Thus, the order of a \( k \)-step Adams-Bashforth method is \( k \). The method (5.3.7) has only one function evaluation per step. This is contrasted to the minimum of \( k \) function evaluations that are needed for a \( k \)-stage Runge-Kutta method.
The representation of (5.3.7a) in terms of backward differences will be useful to us in some circumstances, but generally we prefer a formula written in terms of function values at preceding points. This can be done using

$$\nabla^q f_{n-1} = \sum_{j=0}^{q} (-1)^j \binom{q}{j} f_{n-1-j}. \quad (5.3.8a)$$

Then, using (5.3.7a)

$$y_n = y_{n-1} + h \sum_{i=0}^{k-1} \gamma_i \sum_{j=0}^{i} (-1)^j \binom{i}{j} f_{n-1-j}. \quad (5.3.8b)$$

By rearranging the order of the computation (5.3.8b) can be written in the form (Problem 1)

$$y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j f_{n-j}. \quad (5.3.8c)$$

where

$$\beta_j = \sum_{i=j-1}^{k-1} \gamma_i \binom{i}{j-1}. \quad (5.3.8d)$$

The formula (5.3.8d) now has the form of (5.1.2a).

The parameters $\gamma_j$ of (5.3.6a) and $\beta_j$ of (5.3.8d) are independent of the problem and can be evaluated in advance of the computation. Thus, using (5.3.6a)

$$\gamma_0 = \int_0^1 \begin{pmatrix} -\tau \\ 0 \end{pmatrix} d\tau = 1,$$
\[ \gamma_1 = \int_0^1 -\left( -\frac{\tau}{1} \right) d\tau = \int_0^1 \tau d\tau = \frac{1}{2}, \]
\[ \gamma_2 = \int_0^1 \left( -\frac{\tau}{2} \right) d\tau = \int_0^1 \frac{-\tau(-\tau-1)}{2} d\tau = \frac{5}{12}. \]

Continuing in this manner and using the results in (5.3.7a) yields
\[ y_n = y_{n-1} + h \left( 1 + \frac{1}{2} \nabla + \frac{5}{12} \nabla^2 + \frac{3}{8} \nabla^3 + \frac{251}{720} \nabla^4 + \frac{95}{288} \nabla^5 + \ldots + \gamma_{k-1} \nabla^{k-1} \right) f_{n-1}. \]

(5.3.9a)

Additional results, given in Hairer et al. [12], Section III.1, are reproduced in Table 5.3.1.

<table>
<thead>
<tr>
<th>( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_j )</td>
<td>1</td>
<td>2</td>
<td>5/12</td>
<td>3/8</td>
<td>251/720</td>
<td>95/288</td>
<td>525/1280</td>
<td>1049/3280</td>
<td>1718/3280</td>
</tr>
</tbody>
</table>

Table 5.3.1: Coefficients \( \gamma_j \) for Adams-Bashforth methods ([12], Section III.1).

You may recall the second mean value theorem for integrals which states that if \( p(\tau) \in C^0[a, b] \) and \( q(\tau) \) is integrable and does not change sign on \([a, b]\) then
\[ \int_a^b p(\tau)q(\tau)d\tau = p(\eta) \int_a^b q(\tau)d\tau, \quad \eta \in (a, b). \]

Examining (5.3.4b), reveals that
\[ \left( -\frac{\tau}{k} \right) = \frac{\tau(\tau+1)\ldots(\tau+k-1)}{k!} \]
does not change sign for \( \tau \in [0, 1] \). Thus, we can use the second mean value theorem with (5.3.7b) and write the local error in the more explicit form
\[ d_n = (-1)^k h^{k+1} y^{(k+1)}(\eta) \int_0^1 \left( -\frac{\tau}{k} \right) d\tau = \gamma_k h^{k+1} y^{(k+1)}(\eta), \quad \eta \in (t_{n-1}, t_n). \]

(5.3.9b)

Thus, the error coefficient \( \gamma_k \) is also known from, e.g., Table 5.3.1. Contrast this with discretization error formulas for Runge-Kutta methods that were extremely complex.

Formulas and their local discretization errors for \( k = 1, 2, 3, 4 \), follow.

- \( k = 1 \) : Euler’s method

\[ y_n = y_{n-1} + hf_{n-1}, \]

(5.3.10a)

\[ \tau_n = \frac{h}{2} y''(\xi), \]

(5.3.10b)
$k = 2 : \text{Two-step Adams-Bashforth formula}$

$$y_n = y_{n-1} + h(f_{n-1} + \nabla f_{n-1}/2)$$

or

$$y_n = y_{n-1} + \frac{h}{2}(3f_{n-1} - f_{n-2}), \quad (5.3.11a)$$

$$\tau_n = \frac{5}{12}h^2 y''(\xi). \quad (5.3.11b)$$

$k = 3 : \text{Three-step Adams-Bashforth formula}$

$$y_n = y_{n-1} + \frac{h}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3}), \quad (5.3.12a)$$

$$\tau_n = \frac{3}{8}h^3 y^{iv}(\xi). \quad (5.3.12b)$$

$k = 4 : \text{Four-step Adams-Bashforth formula}$

$$y_n = y_{n-1} + \frac{h}{24}(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}), \quad (5.3.13a)$$

$$\tau_n = \frac{251}{720}h^4 y^{v}(\xi). \quad (5.3.13b)$$

The coefficients of the Adams-Bashforth methods (5.3.8c) and the local error coefficient according to (5.3.9b) are repeated in Table 5.3.2 for $k = 1, 2, \ldots, 6$ [1].

<table>
<thead>
<tr>
<th>$k$</th>
<th>$j = 1$</th>
<th>$j = 2$</th>
<th>$j = 3$</th>
<th>$j = 4$</th>
<th>$j = 5$</th>
<th>$j = 6$</th>
<th>$\gamma_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\beta_j$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2$\beta_j$</td>
<td>3</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12$\beta_j$</td>
<td>23</td>
<td>-16</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24$\beta_j$</td>
<td>55</td>
<td>-59</td>
<td>37</td>
<td>-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>720$\beta_j$</td>
<td>1901</td>
<td>-2774</td>
<td>2616</td>
<td>-1274</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1440$\beta_j$</td>
<td>4277</td>
<td>-7923</td>
<td>9982</td>
<td>-7298</td>
<td>2877</td>
<td>-475</td>
</tr>
</tbody>
</table>

Table 5.3.2: Coefficients of the Adams-Bashforth method (5.3.8c) and the local error (5.3.9b) for orders one through six [1].

**Problems**
1. Show that (5.3.8b) can be written in the form
\[ y_n = y_{n-1} + h \sum_{j=0}^{k-1} (-1)^j f_{n-1-j} \sum_{i=j}^{k-1} \gamma_i \left( \begin{array}{c} i \\ j \end{array} \right) \]
and that this leads to (5.3.8c).

5.4 Implicit Methods: Adams-Moulton Methods

Implicit Adams formulas, called Adams-Moulton methods, are derived in the same manner as the explicit formulas of Section 5.3. In this case, we approximate \( f \) by a \( k-1 \) st degree interpolating polynomial \( P_{k-1}(t) \) that satisfies
\[ P_{k-1}(t_{n-i}) = f(t_{n-i}, y(t_{n-i})), \quad i = 0, 1, \ldots, k-1. \] (5.4.1)

A formula for \( P_{k-1}(t) \) and its error follow from (5.3.3a) and (5.3.3b) upon replacement of \( n-1 \) by \( n \). This yields
\[ P_{k-1}(t) = \sum_{i=0}^{k-1} \frac{\nabla^i f_n}{i!h^i} \prod_{j=0}^{i-1} (t - t_{n-j}) \] (5.4.2a)
\[ E_{k-1}(t) = \frac{f^{(k)}(\xi, y(\xi))}{k!} \prod_{j=0}^{k-1} (t - t_{n-j}). \] (5.4.2b)

Again letting \( \tau = (t - t_{n-1})/h \) and substituting (5.4.2) into (5.3.2), we find
\[ y(t_n) = y(t_{n-1}) + h \int_0^1 \sum_{i=0}^{k-1} (-1)^i \left( \begin{array}{c} -\tau + 1 \\ i \end{array} \right) \nabla^i f_n d\tau + h \int_0^1 (-1)^k h^k \left( \begin{array}{c} -\tau + 1 \\ k \end{array} \right) y^{(k+1)}(\xi) d\tau. \]

Letting
\[ \gamma^*_i = \int_0^1 (-1)^i \left( \begin{array}{c} -\tau + 1 \\ i \end{array} \right) d\tau. \] (5.4.3a)
we find the Adams-Moulton formula as
\[ y_n = y_{n-1} + h \sum_{i=0}^{k-1} \gamma^*_i \nabla^i f_n. \] (5.4.3b)
Figure 5.4.1: Information needed for a $k$-value Adams-Moulton method.

The local error is, as usual, a bit more complex with implicit formulas. We’ll skirt this by using the local discretization error

$$
\tau_n = \frac{y(t_n) - y(t_{n-1})}{h} - \sum_{i=0}^{k-1} \gamma_i^k \nabla_i f_n
$$

or

$$
\tau = (-1)^k h^k \int_0^1 \left( -\frac{\tau + 1}{k} \right) y^{(k+1)}(\xi) d\tau = \gamma_k^k h^k y^{(k+1)}(\eta), \quad \eta \in (t_{n-1}, t_n). \quad (5.4.3c)
$$

As shown in Figure 5.4.1, the solution $y_{n-1}$ and function values $f_n, f_{n-1}, \ldots, f_{n-k+1}$ are needed to compute $y_n$. Unlike the Adams-Bashforth formula, the index $k$ does not indicate the number of steps of the method. Methods with $k = 1$ and 2 are one-step methods and those with $k \geq 3$ are $k - 1$-step methods. We might call the $k$th Adams-Moulton method a $k$-value method since it involves $k$ values of $f$. Like a $k$-step Adams-Bashforth methods, a $k$-value Adams-Moulton method is of order $k$ (cf. (5.4.3c)).

For linear problems, (5.4.3a,b) only requires one function evaluation per step; however, iteration will generally be needed for nonlinear problems.

Expanding (5.4.3b) by evaluating $\gamma_i^k$, $i = 0, 1, \ldots, k - 1$, according to (5.4.3a), we find

$$
y_n = y_{n-1} + h \left( 1 - \frac{1}{2} \nabla - \frac{1}{12} \nabla^2 - \frac{1}{24} \nabla^3 - \frac{19}{720} \nabla^4 - \frac{3}{160} \nabla^5 + \right.
$$
\[ \ldots + \gamma_{k-1}^* \nabla^k - 1 \) f_n. \] (5.4.4)

Additional results, given in Hairer et al. [12], Section III.1, are reproduced in Table 5.4.1.

<table>
<thead>
<tr>
<th>( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma_j^* )</td>
<td>1</td>
<td>(-\frac{1}{2})</td>
<td>(-\frac{1}{12})</td>
<td>(-\frac{1}{720})</td>
<td>(-\frac{1}{40320})</td>
<td>(-\frac{1}{3628800})</td>
<td>(-\frac{1}{3628800})</td>
<td>(-\frac{1}{3628800})</td>
<td>(-\frac{1}{3628800})</td>
</tr>
</tbody>
</table>

Table 5.4.1: Coefficients \( \gamma_j^* \) for implicit Adams methods ([12], Section III.1).

Expanding the backward differences in (5.4.4) using (5.2.15a,b), we find specific formulas for given choices of \( k \). Those formulas for \( k = 1, 2, 3, 4 \), and their local discretization errors follow.

- **\( k = 1 \):** Backward Euler method

  \[ y_n = y_{n-1} + hf_n, \] (5.4.5a)

  \[ \tau_n = -\frac{h}{2} y''(\xi). \] (5.4.5b)

- **\( k = 2 \):** Trapezoidal rule

  \[ y_n = y_{n-1} + h(f_n - \frac{1}{2} \nabla f_n) \] or

  \[ y_n = y_{n-1} + \frac{h}{2} (f_n + f_{n-1}); \] (5.4.6a)

  \[ \tau_n = -\frac{1}{12} h^2 y''(\xi). \] (5.4.6b)

- **\( k = 3 \):** Three-value (two-step) Adams-Moulton formula

  \[ y_n = y_{n-1} + \frac{h}{12} (5f_n + 8f_{n-1} + f_{n-2}); \] (5.4.7a)

  \[ \tau_n = -\frac{1}{24} h^3 y''(\xi). \] (5.4.7b)
• $k = 4$: Four-value (three-step) Adams-Moulton formula

$$y_n = y_{n-1} + \frac{h}{24}(9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3}), \quad (5.4.8a)$$

$$\tau_n = -\frac{19}{720}h^4 y''(\xi). \quad (5.4.8b)$$

Once again, we present the coefficients of the Adams-Moulton methods

$$y_n = y_{n-1} + h \sum_{j=0}^{k-1} \beta_j f_{n-j} \quad (5.4.9)$$

and their error coefficients (5.4.3c) for $k = 1, 2, \ldots, 6$, in Table 5.4.2.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$\beta_j$</th>
<th>$j = 0$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>$\gamma_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\beta_j$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-$\frac{1}{7}$</td>
</tr>
<tr>
<td>2</td>
<td>$2\beta_j$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-$\frac{7}{12}$</td>
</tr>
<tr>
<td>3</td>
<td>$12\beta_j$</td>
<td>5</td>
<td>8</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td>-$\frac{21}{20}$</td>
</tr>
<tr>
<td>4</td>
<td>$24\beta_j$</td>
<td>9</td>
<td>19</td>
<td>-5</td>
<td>1</td>
<td></td>
<td></td>
<td>-$\frac{3}{2}$</td>
</tr>
<tr>
<td>5</td>
<td>$720\beta_j$</td>
<td>251</td>
<td>646</td>
<td>-264</td>
<td>106</td>
<td>-19</td>
<td></td>
<td>-$\frac{360}{663}$</td>
</tr>
<tr>
<td>6</td>
<td>$1440\beta_j$</td>
<td>475</td>
<td>1427</td>
<td>-798</td>
<td>482</td>
<td>-173</td>
<td>27</td>
<td>$\frac{863}{660}$</td>
</tr>
</tbody>
</table>

Table 5.4.2: Coefficients of the Adams-Moulton method (5.4.9) and their local error coefficients (5.4.3c) for orders one through six [1].

**Example 5.4.1.** Comparing Tables 5.3.2 and 5.4.2, we see that the error coefficient of the Adams-Moulton method of order $k$ is smaller than that of the Adams-Bashforth method of the same order. Thus, with comparable derivatives, the implicit methods should produce more accuracy than the explicit methods. Let’s examine this possibility using the simple example (cf. Burden and Faires [6], Chapter 5)

$$y' = y - t^2 + 1, \quad 0 < t \leq 2, \quad y(0) = \frac{1}{2},$$

which has the exact solution

$$y(t) = (t + 1)^2 - \frac{e^t}{2}.$$  

Burden and Faires [6] calculate solutions using the fourth order Adams-Bashforth (5.3.13a) and Adams-Moulton (5.4.8a) formulas. Since this problem is linear, the Adams-Moulton solution may be obtained without iteration. Results with $h = 0.2$ are shown in
Table 5.4.3. The four starting values \((y_0, y_1, y_2, y_3)\) that are needed for (5.3.13a) and the three \((y_0, y_1, y_2)\) that are needed for (5.4.8) are obtained from the exact solution. The global errors found when using the classical fourth-order Runge-Kutta method (3.2.4) with the same step size are also shown in Table 5.4.3.

<table>
<thead>
<tr>
<th>(t_n)</th>
<th>(y(t_n))</th>
<th>Adams-Bashforth Error</th>
<th>Adams-Moulton Error</th>
<th>Runge-Kutta Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.50000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
</tr>
<tr>
<td>0.2</td>
<td>0.8292986</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.00000053</td>
</tr>
<tr>
<td>0.4</td>
<td>1.2140877</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.0000114</td>
</tr>
<tr>
<td>0.6</td>
<td>1.6489406</td>
<td>0.00000000</td>
<td>0.00000065</td>
<td>0.0000186</td>
</tr>
<tr>
<td>0.8</td>
<td>2.1272207</td>
<td>0.0000828</td>
<td>0.0000160</td>
<td>0.0000269</td>
</tr>
<tr>
<td>1.0</td>
<td>2.6408227</td>
<td>0.0002219</td>
<td>0.0000293</td>
<td>0.0000364</td>
</tr>
<tr>
<td>1.2</td>
<td>3.1798942</td>
<td>0.0004065</td>
<td>0.0000478</td>
<td>0.0000474</td>
</tr>
<tr>
<td>1.4</td>
<td>3.7323401</td>
<td>0.0006601</td>
<td>0.0000731</td>
<td>0.0000599</td>
</tr>
<tr>
<td>1.6</td>
<td>4.2834095</td>
<td>0.0010993</td>
<td>0.0001071</td>
<td>0.0000743</td>
</tr>
<tr>
<td>1.8</td>
<td>4.8150857</td>
<td>0.0014812</td>
<td>0.0001527</td>
<td>0.0000906</td>
</tr>
<tr>
<td>2.0</td>
<td>5.3053630</td>
<td>0.0021119</td>
<td>0.0002132</td>
<td>0.0001089</td>
</tr>
</tbody>
</table>

Table 5.4.3: Global errors for Example 5.4.1 obtained by fourth-order Adams-Bashforth, Adams-Moulton, and classical Runge-Kutta methods ([6], Chapter 5).

Errors with the Adams-Moulton method are about ten times smaller than those with the Adams-Bashforth method. The ratio of their error coefficients is 251/19 ≈ 13 (cf. (5.3.13b) and (5.4.8b)). Thus, the results are close to their expected values. Accuracy of the Runge-Kutta method is almost twice that of the Adams-Moulton method; however, the Runge-Kutta method requires approximately four times the work of the two Adams methods.

### 5.5 Implicit Methods: Backward-difference Methods

While we haven’t analyzed stability of multistep methods, our experience with Runge-Kutta methods indicates that implicit methods have better stability properties than explicit methods. Thus, with the objectives of developing implicit methods having good stability properties for stiff systems and of illustrating another class of methods, we’ll
examine \textit{backward-difference formulas}. These methods are derived by approximating $y(t)$ by a $k$ \textit{th}-degree interpolating polynomial $P_k(t)$ using the $k + 1$ points $t_{n-i}$, $i = 0, 1, \ldots, k$, and differentiating $P_k(t)$ to approximate $y'(t)$. Once again, we’ll use the Newton form of the interpolating polynomial which may be obtained from (5.3.3) by replacing $n - 1$ by $n$, $f$ by $y$, and $k - 1$ by $k$ to obtain

\begin{equation}
    P_k(t) = \sum_{i=0}^{k} \frac{\nabla^i y(t_n)}{i! h^i} \prod_{j=0}^{i-1} (t - t_{n-j}) \tag{5.5.1a}
\end{equation}

\begin{equation}
    E_k(t) = y(t) - P_k(t) = \frac{y^{(k+1)}(\xi)}{(k + 1)!} \prod_{j=0}^{k} (t - t_{n-j}), \quad \xi \in (t_{n-k}, t_n). \tag{5.5.1b}
\end{equation}

The backward-difference method will be obtained by collocating at $t = t_n$, i.e., by enforcing

\begin{equation}
    P_k'(t_n) = f(t_n, P_k(t_n)). \tag{5.5.2}
\end{equation}

Once again, it will be convenient to change variables by letting

\begin{equation}
    \tau = \frac{t - t_n}{h} \tag{5.5.3a}
\end{equation}

and use (5.3.4) to obtain

\begin{equation}
    \frac{1}{i! h^i} \prod_{j=0}^{i-1} (t - t_{n-j}) = \frac{\tau^{(i+1)} \cdots (\tau + i - 1)}{i!} = (-1)^i \binom{-\tau}{i}. \tag{5.5.3b}
\end{equation}

Then, (5.5.1) can be written as

\begin{equation}
    P_k(t) = \sum_{i=0}^{k} (-1)^i \binom{-\tau}{i} \nabla^i y(t_n) \tag{5.5.3b}
\end{equation}

and

\begin{equation}
    E_k(t) = (-1)^{k+1} h^{k+1} \binom{-\tau}{k+1} y^{(k+1)}(\xi). \tag{5.5.3c}
\end{equation}

Differentiating (5.5.3b)

\begin{equation}
    P_k'(t_n) = \frac{1}{h} \frac{dP_k(0)}{d\tau} = \frac{1}{h} \sum_{i=1}^{k} \delta_i \nabla^i y(t_n) \tag{5.5.4a}
\end{equation}
where

\[ \delta_i = (-1)^i \frac{d}{d\tau} \left( \begin{array}{c} -\tau \\ i \end{array} \right)_{\tau=0} = \frac{d}{d\tau} \left[ \frac{\tau(\tau+1)(\tau+2)\ldots(\tau+i-1)}{i!} \right]_{\tau=0}, \quad i \geq 1. \]

Differentiating the product yields

\[ \delta_i = \frac{1}{i}, \quad i \geq 1. \quad \text{(5.5.4b)} \]

Differentiation of the error expression (5.5.3b) proceeds in the same manner. Bear in mind that \( \xi \) is a function of \( t \) (or \( \tau \)); however,

\[ \left( \begin{array}{c} -\tau \\ k+1 \end{array} \right)_{\tau=0} = 0, \]

so

\[ E^l_k(t_n) = h^k \delta_{k+1} y^{(k+1)}(\xi(t_n)) = \frac{h^k}{k+1} y^{(k+1)}(\xi(t_n)). \quad \text{(5.5.4c)} \]

Replacing \( y(t_n) \) in (5.3.1) by \( P_k(t_n) + e_k(t_n) \) and using (5.5.4) yields

\[ \sum_{i=1}^{k} \frac{\nabla^i y(t_n)}{i} + \frac{h^{k+1}}{k+1} y^{(k+1)}(\xi) = hf(t_n, y(t_n)). \]

Neglecting the local discretization error yields the backward-difference formula

\[ \sum_{i=1}^{k} \frac{\nabla^i y_n}{i} = hf(t_n, y_n). \quad \text{(5.5.5a)} \]

Its local discretization error is

\[ \tau_n = \sum_{i=1}^{k} \frac{\nabla^i y(t_n)}{i} - hf(t_n, y(t_n)) = -\frac{h^k}{k+1} y^{(k+1)}(\xi), \quad \xi \in (t_{n-k}, t_n). \quad \text{(5.5.5b)} \]

The formula (5.5.5a) has the general form of (5.1.2a) with \( \beta_1 = \beta_2 = \ldots = \beta_k = 0 \), i.e.,

\[ \sum_{i=0}^{k} \alpha_i y_{n-i} = h\beta_0 f_n, \quad \text{(5.5.5c)} \]

The backward-difference formula with index \( k \) is a \( k \)-step method having order \( k \) (since the local discretization error is \( O(h^k) \)).

Backward difference methods and their local discretization errors for \( k = 1, 2, 3, 4, \ldots \) follow. Backward difference coefficients for \( k = 1, 2, \ldots, 6 \), appear in Table 5.5.1 [1].
• $k = 1$ : Backward Euler method

\[ y_n = y_{n-1} + hf_n, \]  
\[ \tau_n = -\frac{h}{2} y''(\xi). \]  
(5.5.6a)  
(5.5.6b)

• $k = 2$ : Two-step backward-difference formula

\[ \nabla y_n + \frac{\nabla^2 y_n}{2} = hf_n \]

or

\[ y_n = \frac{1}{3} (4y_{n-1} - y_{n-2} + 2hf_n), \]  
(5.5.7a)

\[ \tau_n = -\frac{h^2}{3} y'''(\xi). \]  
(5.5.7b)

• $k = 3$ : Three-step backward-difference formula

\[ y_n = \frac{1}{11} (18y_{n-1} - 9y_{n-2} + 2y_{n-3} + 6hf_n), \]  
(5.5.8a)

\[ \tau_n = -\frac{h^3}{4} y^{\prime\prime\prime}(\xi). \]  
(5.5.8b)

• $k = 4$ : Four-step backward-difference formula

\[ y_n = \frac{1}{23} (48y_{n-1} - 36y_{n-2} + 16y_{n-3} - 3y_{n-4} + 12hf_n), \]  
(5.5.9a)

\[ \tau_n = -\frac{h^4}{5} y^{\prime\prime\prime}(\xi). \]  
(5.5.9b)

Examples using backward-difference formulas will be presented in Section 5.7.
Table 5.5.1: Coefficients of the backward difference method (5.5.5c). for orders one through six [1]. Their local error coefficients are $1/(k + 1)$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$\delta$</th>
<th>$\delta \beta_0$</th>
<th>$\delta \alpha_0$</th>
<th>$\delta \alpha_1$</th>
<th>$\delta \alpha_2$</th>
<th>$\delta \alpha_3$</th>
<th>$\delta \alpha_4$</th>
<th>$\delta \alpha_5$</th>
<th>$\delta \alpha_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>-4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>6</td>
<td>11</td>
<td>-18</td>
<td>9</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>12</td>
<td>25</td>
<td>-48</td>
<td>36</td>
<td>-16</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>137</td>
<td>60</td>
<td>137</td>
<td>-300</td>
<td>300</td>
<td>-200</td>
<td>75</td>
<td>-12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>147</td>
<td>60</td>
<td>147</td>
<td>-360</td>
<td>450</td>
<td>-400</td>
<td>225</td>
<td>-72</td>
<td>10</td>
</tr>
</tbody>
</table>

5.6 Convergence, Accuracy, and Stability

Having several methods at our disposal, let us now study questions of consistency, convergence, and stability of linear multistep methods (LMMs) of the form (5.1.2) applied to the IVP (5.1.1). We begin by applying the usual definitions in our present situation.

**Definition 5.6.1.** Let $z(t) \in C^1[0, T]$, $t_n - ih \in [0, T]$, $i = 0, 1, \ldots, k$, and

$$L_h(z(t)) = \frac{\sum_{i=0}^{k} \alpha_i z(t_n - ih)}{h} - \sum_{i=0}^{k} \beta_i z'(t_n - ih).$$

(5.6.1a)

The local discretization error of the LMM (5.1.2) is

$$\tau_n = L_h(y(t_n)).$$

(5.6.1b)

**Definition 5.6.2.** The local error is

$$d_n = y(t_n) - y_n$$

assuming that no errors have occurred prior to $t_n$, i.e., $y_i = y(t_i)$, $i < n$.

Substituting exact solution values into (5.1.2) for $i = 0, 1, \ldots, n - 1$, using (5.6.2), and using the normalization $\alpha_0 = 1$ (cf. (5.1.2c)), we find

$$y(t_n) - y_n = y(t_n) + \sum_{i=1}^{k} \alpha_i y(t_{n-i}) - h \beta_0 f(t_n, y_n) - h \sum_{i=1}^{k} \beta_i f(t_{n-i}, y(t_{n-i})).$$

Using (5.6.1)

$$y(t_n) - y_n = h \beta_0 [f(t_n, y(t_n)) - f(t_n, y_n)] + h \tau_n.$$
Assuming that $f$ is continuously differentiable and using the mean value theorem

$$f(t_n, y(t_n)) - f(t_n, y_n) = f_y(t_n, \xi_n)[y(t_n) - y_n]$$

where $\xi_n$ is between $y_n$ and $y(t_n)$. Using (5.6.2), we have

$$d_n = \frac{h \tau_n}{1 - h/3 \beta_0 f_y(t_n, \xi_n)}.$$ 

Thus, the local error is proportional to the local discretization error for all LMMs and the local error is the product of the step size and the local discretization errors for all explicit LMMs ($\beta_0 = 0$).

**Definition 5.6.3.** A LMM is of order $p$ if $p$ is the largest integer for which $L_h(y(t_n)) = O(h^p)$.

**Definition 5.6.4.** A LMM is consistent if it is at least of order one.

From Definition 5.6.4, consistency implies that the LMM must be exact when $y(t)$ is a linear polynomial (Problem 5.1.1). Using (5.1.2) with $y(t) = 1$ and $y(t) = t$ yields the explicit consistency conditions

$$\sum_{i=0}^{k} \alpha_i = 0, \quad \sum_{i=1}^{k} i \alpha_i + \sum_{i=0}^{k} \beta_i = 0. \quad (5.6.3)$$

**Definition 5.6.5.** Consider solving (5.1.1) using (5.1.2) on $0 < t \leq T$ with a sequence of meshes $\{0 = t_0 < t_1 < \ldots < t_N = T\}$ such that $N \to \infty$ and $h = \max_{1 \leq n \leq N}(t_n - t_{n-1}) \to 0$. A LMM is convergent if $y_n \to y(t_n)$, $n \in [0, N]$ uniformly as $N \to \infty$ ($h \to 0$).

Stability and its relation to consistency and convergence is not as simple for LMMs as for one-step methods. While a convergent multistep method is necessarily consistent, consistency alone is not sufficient for convergence. Before tackling the general question of stability (absolute stability, etc.), let us consider a simple example.

**Example 5.6.1.** Consider the solution of the test problem

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

by the leap frog method

$$y_n = y_{n-2} + 2hf_{n-1} = y_{n-2} + 2h\lambda y_{n-1}, \quad y_0 = 1.$$
A solution, calculated on $0 < t \leq 4.5$ with $\lambda = -2$ and $h = 1/32$, is shown in Figure 5.6.1. The solution appears to be correct initially, but oscillations of increasing amplitude develop and, at the very least, the solution is not absolutely stable. The local discretization error of the leap frog scheme is $O(h^2)$, so it is certainly consistent. Since the test equation is linear, it suffices to forgo perturbations and examine solution of

$$y_n = y_{n-2} + 2h\lambda y_{n-1}, \quad y_0 = \delta.$$  

This is a second-order difference equation with constant (independent of $n$) coefficients. These difference equations have many similarities to second-order, constant-coefficient ODEs. In this case, the difference equation may be solved by assuming a solution of the form

$$y_n = c\xi^n.$$  

29
Substituting this assumed solution into the leap frog difference equation yields

\[ c\xi^n = c\xi^{n-2} + 2h\lambda c\xi^{n-1}. \]

Seeking non-trivial solutions, it is permissible to divide by \( c\xi^{n-2} \) to obtain

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

The roots of this quadratic equation are

\[ \xi_{1,2} = h\lambda \pm \sqrt{1 + (h\lambda)^2}. \]

The two roots satisfy the difference equation and the general solution is a linear combination of both of them; thus,

\[ y_n = c_1\xi_1^n + c_2\xi_2^n. \]

The values of \( c_1 \) and \( c_2 \) are determined from the initial condition \( y_0 = \delta \) and the starting value \( y_1 \); however, for our purposes, it won’t be necessary to explicitly calculate \( c_1 \) and \( c_2 \). If, however, \( y_0 \) and \( y_1 \) are \( O(\delta) \) then the \( c_1 \) and \( c_2 \) will also be \( O(\delta) \).

Let’s obtain more explicit information by assuming that \( |h\lambda| \ll 1 \). Then, expanding the square root,

\[ \xi_1 = 1 + h\lambda + O((h\lambda)^2), \quad \xi_2 = -[1 - h\lambda + O((h\lambda)^2)]. \]

Thus,

\[ y_n = c_1[1 + h\lambda + O((h\lambda)^2)]^n + (-1)^nc_2[1 - h\lambda + O((h\lambda)^2)]^n, \]

or

\[ y_n = c_1 e^{\lambda n} + (-1)^nc_2 e^{-\lambda n} + O(n(h\lambda)^2). \]

The first term on the right approximates the exact ODE solution which, of course, is

\[ y(t_n) = \delta e^{\lambda t_n} = \delta e^{\lambda n}. \]

The second solution is called “parasitic” or “extraneous” and arises because the first-order ODE has been approximated by a second-order difference equation. If \( \lambda > 0 \), the parasitic solution

\[ (-1)^nc_2 e^{-\lambda n} \]
decays as \( n \) increases and causes little harm. However, if \( \lambda < 0 \), the parasitic solution grows with increasing \( n \) and eventually dominates the approximation of the exact solution. The parasitic solution oscillates from time step to time step as indicated by the presence of the \((-1)^n\) factor (Figure 5.6.1).

Let’s continue our investigation of stability by re-stating the basic concept (Section 2.1) as it applies to a LMM.

**Definition 5.6.6.** A LMM is stable if there exists an \( \hat{h} \) for each \( f(t,y) \) such that a change in the starting values by a fixed amount produces a bounded change in the numerical solution for all \( h \in (0, \hat{h}) \) and all \( t \in [0, T] \).

This definition is, once again, too general for practical verification. As usual, we rely on the test equation

\[
y' = \lambda y, \quad y(0) = \delta. \tag{5.6.4}
\]

To conduct a stability analysis, we apply (5.6.4) to (5.1.2) to obtain

\[
\sum_{i=0}^{k} (\alpha_i y_{n-i} - h \lambda \beta_i y_{n-i}) = 0. \tag{5.6.5}
\]

This \( k \) th order constant-coefficient difference equation can be solved in the manner used for Example 5.6.1; thus, assume a solution of the form

\[
y_n = c \xi^n \tag{5.6.6}
\]

and substitute it into (5.6.5) to obtain

\[
\sum_{i=0}^{k} (\alpha_i c \xi^{n-i} - h \lambda \beta_i c \xi^{n-i}) = 0.
\]

Dividing by the common \( c \xi^{n-k} \) factor, we find that \( \xi \) is a root of the polynomial

\[
\rho(\xi) - h \lambda \sigma(\xi) = 0, \tag{5.6.7a}
\]

where

\[
\rho(\xi) = \sum_{i=0}^{k} \alpha_i \xi^{k-i}, \tag{5.6.7b}
\]
and
\[
\sigma(\xi) = \sum_{i=0}^{k} \beta_i \xi^{k-i}. \tag{5.6.7c}
\]

Equations (5.6.7b) and (5.6.7c) are called the first and second characteristic polynomials of the LMM, respectively. Using (5.6.3), we see that the LMM is consistent if and only if
\[
\rho(1) = 0, \quad \rho'(1) - \sigma(1) = 0. \tag{5.6.7d}
\]

Equation (5.6.7a) has \(k\) solutions, \(\xi_i, i = 1, 2, \ldots, k\). One solution, say \(\xi_1\), approximates the solution of the test problem (5.6.4). The remaining \(k-1\) solutions are parasitic. If the roots of (5.6.7a) are distinct then the general solution of (5.6.5) is
\[
y_n = \sum_{i=1}^{k} c_i \xi_i^n. \tag{5.6.8a}
\]
The assumed solution (5.6.6) must be modified when some of the roots are equal. Suppose, for example, that \(\xi_i\) is a root of multiplicity \(\mu\), then the \(\mu\) solutions corresponding to \(\xi_i\) are
\[
(c_i + c_{i+1}n + c_{i+2}n^2 + \ldots + c_{i+\mu-1}n^{\mu-1})\xi_i^n. \tag{5.6.8b}
\]
Note the similarities between this analysis and that of constant-coefficient ODE [3].

Since the exact solution of the test equation (5.6.4) is
\[
y(t_n) = \delta e^{h\lambda n} = \delta (e^{h\lambda})^n, \tag{5.6.9}
\]
the principal root \(\xi_1\) of (5.6.7) must be an approximation of \(e^{h\lambda}\). If the remaining roots \(|\xi_i| > 1, i = 2, 3, \ldots, k\), the corresponding parasitic solutions grow as \(n\) increases and the method, at the very least, cannot be absolutely stable. Let us emphasize that all roots \(\xi_i, i = 1, 2, \ldots, k\), of (5.6.7a-c) depend continuously on the parameter \(h\lambda\) by writing \(\xi_i = \xi_i(h\lambda), i = 1, 2, \ldots, k\). If \(|\xi_i(0)| < 1, i = 1, 2, \ldots, k\), there is an \(\hat{h}\) for any fixed \(\lambda\) such that \(|\xi_i(h\lambda)| < 1, i = 1, 2, \ldots, k\), for \(h < \hat{h}\). These arguments suggest that we can study the stability characteristics of the LMM by examining the roots of
\[
\rho(\xi) = \sum_{i=1}^{k} \alpha_i \xi_i = 0, \tag{5.6.10}
\]
which is the solution of (5.6.7a) when \( h \lambda = 0 \).

If the LMM is consistent then (5.6.7d) implies that unity is a root of (5.6.10). From (5.6.9) we see that this corresponds to the exact solution of the test equation, i.e.,

\[ \xi_1(0) = 1. \quad (5.6.11) \]

We have noted that roots \( \xi_i(0), i = 2, 3, \ldots, k \), that are less than unity will remain so for sufficiently small values of \( h \lambda \). Thus, the limiting stability (as \( h \lambda \to 0 \)) is determined by those roots that are greater than or equal to unity in modulus. Any root having larger than unit modulus will produce unbounded solutions and must be unstable according to Definition 5.6.6. It, therefore, remains to investigate the behavior of those roots that have unit modulus when \( h = 0 \). Let \( \xi_i \) be such a root and let us initially assume that it is simple. Supposing that \( |h\lambda| \ll 1 \), let us expand \( \xi_i(h\lambda) \) as the series in powers of \( h\lambda \)

\[ \xi_i(h\lambda) = \xi_i(0) + \gamma_i h\lambda + O((h\lambda)^2). \quad (5.6.12a) \]

Substituting (5.6.12a) into (5.6.7a)

\[ \rho(\xi_i(0) + \gamma_i h\lambda + O((h\lambda)^2)) - h\lambda \sigma(\xi_i(0) + \gamma_i h\lambda + O((h\lambda)^2)) = 0. \]

Expanding \( \rho \) and \( \sigma \)

\[ \rho(\xi_i(0)) + h\lambda[\gamma_i \rho'(\xi_i(0)) - \sigma(\xi_i(0))] + O((h\lambda)^2) = 0. \]

Now, \( \rho(\xi_i(0)) = 0 \) according to (5.6.10). Thus, to satisfy (5.6.7a) for sufficiently small \( h\lambda \), the coefficient of \( h\lambda \) in the above expression must vanish. This yields

\[ \gamma_i = \frac{\sigma(\xi_i(0))}{\rho'(\xi_i(0))}. \quad (5.6.12b) \]

Remark 1. The principal root \( \xi_1(0) = 1 \). This, with the consistency conditions (5.6.7d) further implies

\[ \gamma_1 = \frac{\sigma(1)}{\rho'(1)} = 1. \quad (5.6.12c) \]

The series expansion suggests the existence of a constant \( c \) such that

\[ |\xi_i(h\lambda)| \leq |\xi_i(0)|(1 + ch\lambda), \quad h < \hat{h}. \]
Taking the $n$th power and recalling that $|\xi_i(0)| = 1$ yields

$$|\xi_i^n(h\lambda)| \leq |(1 + ch\lambda)^n| \leq |e^{ch\lambda}| \leq |e^{\lambda T}| \leq C, \quad 0 \leq t \leq T, \quad h < \hat{h}. $$

Therefore, the parasitic solutions only grow by a bounded amount for finite times when roots of unit magnitude are simple. Hence, the LMM is stable (but not absolutely stable) for the test equation in this case.

It remains to examine those roots of (5.6.10) that have unit magnitude but are not simple. Suppose that $\xi_i(0)$ is such a root with multiplicity $\mu$, then, as indicated in (5.6.8b), the solution $y_n$ would contain the terms $n\xi_i^n, n^2\xi_i^n, \ldots, n^{\mu-1}\xi_i^n$. With $|\xi_i(0)| = 1$, these terms are unbounded as $n \to \infty$. Hence, the LMM cannot be stable.

The preceding arguments suggest that the stability of a LMM is linked to the following condition.

**Definition 5.6.7.** A LMM satisfies the root condition (or the condition of zero stability if the roots of $\rho(\xi) = 0$ are inside the unit circle or simple on the unit circle in the complex $\xi$-plane.

**Theorem 5.6.1.** Satisfaction of the root condition is necessary and sufficient for the stability of a LMM.

**Proof.** The key aspect of the proof have been presented. A more thorough analysis is given in Gear [10], Section 10.1. \qed

**Example 5.6.2.** The leap frog method of Example 5.1.1 has $\alpha_0 = 1 \alpha_1 = 0$, and $\alpha_2 = -1$ with $k = 2$; thus, using (5.6.7b)

$$\rho(\xi) = \xi^2 - 1. $$

The roots of this equation are $\xi_1(0) = 1$ and $\xi_2(0) = -1$. Both roots have unit modulus and are simple on the unit circle in the complex plane; hence, the leap frog scheme satisfies the root condition and is stable.

While the leap frog scheme is stable, it is clear from Example 5.1.1 that solutions grow rapidly when they should be decaying. Clearly a finer distinction is needed and this prompts the concepts of strong and weak stability.
Definition 5.6.8. A LMM is weakly stable if it is stable, but has more than one root of $\rho(\xi) = 0$ on the unit circle in the complex $\xi$-plane.

Definition 5.6.9. A LMM is strongly stable if all roots of $\rho(\xi) = 0$ are inside the unit circle in the $\xi$-plane except for the principal root $\xi_1(0) = 1$.

Example 5.6.3. The results of Example 5.6.2 confirm that the leap frog scheme is weakly stable. Parasitic solutions of a weakly stable scheme may either grow or decay for nonzero values of $h\lambda$ as $n$ increases, depending on whether they move outside or inside of the unit circle. In Example 5.6.1, we saw that the parasitic solution of the leap frog scheme grew when $h\lambda < 0$ and decayed when $h\lambda > 0$. All parasitic solutions of a strongly stable method decay for sufficiently small values of $h\lambda$ as $n$ increases, since there modulus is less than unity when $h\lambda = 0$.

Example 5.6.4. Let us examine the stability of the Adams-Bashforth and Adams-Moulton methods as $h \to 0$. Recall (Sections 5.3 and 5.4) that the Adams’ methods have the form (5.1.2) with $\alpha_0 = 1$, $\alpha_1 = -1$, and $\alpha_i = 0$, $i = 2, 3, \ldots, k$, i.e.,

$$y_n = y_{n-1} + h \sum_{i=0}^{k} \beta_i f_{n-i}.$$ 

The coefficient $\beta_0 = 0$ for the (explicit) Adams-Bashforth methods and is nonzero for the (implicit) Adams-Moulton methods.

Using the test equation (5.6.4)

$$y_n = y_{n-1} + h \lambda \sum_{i=0}^{k} \beta_i y_{n-i}.$$ 

Assuming that $y_n$ is proportional to $\xi^n$ leads to (5.6.7) which, in this case, is

$$\rho(\xi) - h\lambda \sigma(\xi) = 0,$$

$$\rho(\xi) = \xi^{k-1}(\xi - 1), \quad \sigma(\xi) = \sum_{i=0}^{k} \beta_i \xi^{k-i}.$$ 

Thus, the roots of the first characteristic polynomial ($\rho(\xi) = 0$) are

$$\xi_1 = 1, \quad \xi_2 = \xi_3 = \ldots = \xi_k = 0.$$
The parasitic roots $\xi_i(0), i = 2, 3, \ldots, k$, are all zero for both the Adams-Bashforth and Adams-Moulton methods; thus, these methods are strongly stable.

Computation is performed at finite values of $h$. The root condition and the notions of strong and weak stability all involve reasoning in the limit of vanishingly small step sizes. This shortcoming encourages us to further extend concepts of stability.

**Definition 5.6.10.** A LMM is absolutely stable for those values of $h\lambda$ where the roots of (5.6.7a) satisfy $|\xi(h\lambda)| \leq 1$ when applied to the test problem (5.6.4).

**Definition 5.6.11.** A LMM is relatively stable for those values of $h\lambda$ where the parasitic roots of (5.6.7) are less in magnitude than the principal root.

The boundary of the region of absolute stability of a LMM may be calculated by the *boundary locus method* as described in Chapter 3 for one-step methods. Complex values of $\xi$ having unit modulus are written in polar form

$$\xi = e^{i\theta}$$

and (5.6.7a) is imposed. The boundary of the region of absolute stability follows by calculating

$$h\lambda = \frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})}, \quad \theta \in [0, \pi].$$

The regions of absolute stability of the Adams-Bashforth, Adams-Moulton, and backward difference methods are illustrated in Figures 2, 3, and 4, respectively [1]. Some observations and conclusions follow.

1. The Adams-Moulton methods of orders 1 and 2 are the backward Euler method and the trapezoidal rule. The absolute stability regions of these methods are the exterior of a unit circle centered at $(1, 0)$ and the entire left-half of the $h\lambda$-plane, respectively. Neither region is shown in Figure 5.6.2. Ascher and Petzold’s [1] definition of $k$ differs from ours for Adams-Moulton methods. The regions labeled $k = 2$ to 4 in Figure 5.6.3 correspond to our $k = 3$ to 5.

2. Explicit methods always have a finite region of absolute stability.
Figure 5.6.2: Absolute stability regions for Adams-Bashforth methods of orders 1 to 4. A method is stable inside of its shaded region [1].

Figure 5.6.3: Absolute stability regions for Adams-Moulton methods of orders 3 to 5. A method is stable inside of its shaded region [1].
Figure 5.6.4: Absolute stability regions for backward difference formulas of orders 1 to 3 (top) and 4 to 6 (bottom). Methods are stable outside of the shaded regions [1].
3. Implicit methods have a larger region of absolute stability than a corresponding explicit method of the same order.

4. The absolute stability region typically becomes smaller as order increases.

5. The backward difference formulas shown in Figure 5.6.4 have unbounded absolute stability regions as $Re(h\lambda) \to -\infty$.

6. We have noted that the error coefficients of Adams-Moulton methods (5.4.5 - 5.4.8) are smaller than those of Adams-Bashforth methods (5.3.10 - 5.3.13) having the same order. Figures 5.6.2 and 5.6.3 indicate that the absolute stability regions of the Adams-Moulton methods are larger than those of the Adams-Bashforth methods of the same order. Therefore, the Adams-Moulton methods can be used with much larger step sizes than required for the Adams-Bashforth methods having the same accuracy. Typically, this increase in step size offsets the cost of solving an implicit difference equation.

The maximal order of a $k$-step LMM is $2k$ since there are $2k+1$ free parameters in the general formula (5.1.2); however, these maximal-order methods are typically unstable.

**Theorem 5.6.2.** No $k$-step LMM satisfying the root condition can have order exceeding $k + 1$ when $k$ is odd or order $k + 2$ when $k$ is even.

**Proof.** cf. Hairer et al. [12].

The $k$-step methods of order $k + 2$ are called optimal-order methods. All roots of $\rho(\xi)$ are on the unit circle, so they are only weakly stable.

**Example 5.6.5.** The highest order two-step LMM

$$y_n = -4y_{n-1} + 5y_{n-2} + 2h(2f_{n-1} + f_{n-2})$$

has order four. Using (5.6.7b), the first characteristic polynomial for this method is

$$\rho(\xi) = \xi^2 + 4\xi - 5.$$
The roots of this polynomial, \( \xi_1 = 1 \) and \( \xi_2 = -5 \), do not satisfy the root condition; hence, the method is unstable. Let us illustrate this by applying the method to the very simple equation
\[
y' = 0, \quad y(0) = 0.
\]
The exact solution of this IVP is, of course, the trivial solution \( y(t) = 0 \). For starting values, let us use
\[
y_0 = 0, \quad y_1 = \epsilon,
\]
where \( \epsilon \) represents a small rounding error.

Successive solutions satisfy
\[
y_2 = -4y_1 + 5y_0 = -4\epsilon, \\
y_3 = -4y_2 + 5y_1 = 21\epsilon, \\
y_4 = -4y_3 + 5y_2 = -104\epsilon, \\
\ldots
\]

The step size \( h \) does not explicitly appear in the method for this IVP; hence, the solution cannot be bounded as \( h \to 0 \) and, therefore, the method is unstable. The method also does not converge as \( h \to 0 \); thus, consistency alone is not sufficient for convergence. Necessary and sufficient conditions for convergence are given in the following theorem which is due to G. Dahlquist.

**Theorem 5.6.3.** The necessary and sufficient conditions for a LMM to be convergent are that it be consistent and satisfy the root condition.

**Proof.** Complete proofs of this important theorem are given in Gear [10], Chapter 10, and Henrici [14], Chapter 5. We’ll prove that convergence implies stability and consistency.

Applying the LMM (5.1.2) to the IVP \( y' = 0, \ y(0) = 0 \) yields
\[
\sum_{i=0}^{k} a_i y_{n-i} = 0
\]
Seeking to use a contradiction argument, suppose that \( \rho(\xi) \) has a root \( \xi_2 \) with \( |\xi_2| > 1 \) and/or a root \( \xi_3 \) on the unit circle whose multiplicity exceeds unity. Further suppose
that the starting values are $O(h)$ (so that the starting values converge to $y_0 = 0$). The two solutions of the difference equation corresponding to $\xi_2$ and $\xi_3$ are

$$c_2 h \xi_2^n, \quad c_3 h n \xi_3^n.$$ (The coefficients $c_2$ and $c_3$ have been scaled to reflect the $O(h)$ size of the solution.) The analysis is simplified by letting $Y(t)$ be a continuous function that interpolates to $y_j$, $j = 0, 1, \ldots$. Thus,

$$Y_2(t) = c_2 h \xi_2^{t/h}, \quad Y_3(t) = c_3 h \xi_3^{t/h}, \quad t = nh.$$ For fixed $t$, the solutions $Y_2(t)$ and $Y_3(t)$ are unbounded as $h \to 0$; thus, the solution cannot be convergent and we have a contradiction. Therefore, the LMM must be stable.

Next consider the IVP $y' = 0$, $y(0) = 1$, which has the solution $y(t) = 1$. With the present notation, the difference equation (5.1.2) becomes

$$\sum_{i=0}^{k} \alpha_i Y(t - ih) = 0.$$ Convergence implies that

$$\lim_{h \to 0} Y(t - ih) = 1$$ or, using (5.6.7b), that $\rho(1) = 0$.

Finally, consider the IVP $y' = 1$, $y(0) = 0$, which has the solution $y(t) = t$. The difference equation (5.1.2) for this problem is

$$\sum_{i=0}^{k} \alpha_i Y(t - ih) = h \sum_{i=0}^{k} \beta_i.$$ Convergence would imply that

$$Y(t - ih) = t - ih.$$ Substituting this function into the difference equation and using (5.1.2b) yields

$$\rho(1) - \sigma(1) = 0,$$ which is the consistency condition (5.6.7d). Thus, both consistency conditions within (5.6.7d) are satisfied. \qed

41
A summary of the relationships between the basic concepts of consistency, convergence, and stability is given in Figure 5.6.5. Thus, from left to right, stability implies that the root condition is satisfied, which together with consistency implies convergence. Reading from right to left, convergence implies that the method is consistent and satisfies the root condition, which implies stability.

Let us recall the definition of A-stability introduced in Chapter 3.

**Definition 5.6.12.** A LMM is A-stable if its region of absolute stability contains the left-half plane $\text{Re}(h\lambda) \leq 0$.

Unfortunately, A-stability is very difficult to obtain for LMMs.

**Theorem 5.6.4.** (i) No explicit LMM is A-stable, (ii) the order of an A-stable LMM cannot exceed two, and (iii) the second-order A-stable LMM with the smallest error coefficient is the trapezoidal rule.

*Proof.* cf. Dahlquist [8]. A different proof of (ii) and (iii) appears in Wanner et al. [17].

This a major negative result of numerical ODEs. No such restriction exists for implicit Runge-Kutta methods and arbitrarily high-order A-stable methods may be constructed.

In order to obtain high orders of accuracy with LMMs, we will have to relax our stability demands and give up a portion of the left-half of the $h\lambda$-plane.

**Definition 5.6.13.** A numerical method is $A(\alpha)$-stable, $\alpha \in [0, \pi/2)$ if its region of absolute stability contains the infinite wedge

$$W_\alpha = \{h\lambda \mid -\alpha < \pi - \arg(h\lambda) < \alpha\}.$$
The notion of $A(\alpha)$-stability was introduced by Widlund [18] and is discussed in Gear [10], Chapter 11 and Hairer and Wanner [13], Section V.2. An example of the wedge $W_\alpha$ is shown in Figure 5.6.6. $A(0^+)$-stable methods have regions of absolute stability that contain the negative real axis. $A(\pi/2)$-stable methods are $A$-stable. For a given $\lambda$ with $Re(\lambda) < 0$, the point $h\lambda$ either lies inside $W_\alpha$ for all positive $h$ or outside it for all positive $h$. If we know that all of the eigenvalues of a stiff system lie inside of a certain wedge $W_{\alpha^*}$, then an $A(\alpha^*)$-stable method can be used without any stability restriction on $h$.

Widlund [18] showed that $A(\alpha)$-stable LMMs of orders one through four exist with $\alpha$ arbitrarily close to $\pi/2$. Grigorieff and Schroll [11] showed that $k$-step LMMs of order $k$ also exist for all $k$ (cf. [13], Section V.2).

Gear [10], Chapter 11, describes another means of relaxing $A$-stability by introducing the notion of stiff stability, which combines both stability and accuracy.

**Definition 5.6.14.** A numerical method is stiffly stable if it is absolutely stable in the region $R_1 = \{h\lambda \mid Re(h\lambda) \leq D_L\}$ and accurate in the region $R_2 = \{h\lambda \mid D_L < Re(h\lambda) < D_R, \ |Im(h\lambda)| < D_I\}$.

The regions $R_1$ and $R_2$ are shown in Figure 5.6.7. Those eigenvalues $\lambda$ that represent rapidly decaying terms in the transient solution of the ODE may be solved with step...
sizes \( h \) such that \( h\lambda \in R_1 \). These components would be approximated stably but not accurately. Eigenvalues of the system representing slowly varying parts of the solution are smaller and their product with \( h \) would place them in \( R_2 \). Values of \( h\lambda \) that are outside of \( R_1 \cup R_2 \) should not be used since either the growth of the solution is too fast (when \( \text{Re}(h \lambda) > 0 \)) to be approximated accurately or the solution is too oscillatory (when \( |\text{Im}(h \lambda)| > D_I \)) to follow accurately. In both cases, \( h \) should be reduced so that \( h\lambda \in R_2 \).

**Example 5.6.7.** From Figure 5.6.4, we see that the backward difference formulas of orders one through six are stiffly stable. They are also \( \Lambda(\alpha) \)-stable, but \( \alpha \) decreases with increasing order.

**Example 5.6.8.** From (5.5.5), we recall that the backward difference formulas have the form

\[
\sum_{i=0}^{k} \alpha_i y_{n-i} = \beta_0 f_n.
\]

Applying this method to the test equation (5.6.4) yields

\[
\sum_{i=0}^{k} \alpha_i y_{n-i} - h\lambda \beta_0 y_n = 0.
\]
Assuming a solution of the form $\xi^n$ gives

$$\rho(\xi) - h\lambda \sigma(\xi) = 0,$$

with

$$\rho(\xi) = \sum_{i=0}^{k} \alpha_i \xi^{k-i}, \quad \sigma(\xi) = \beta_0 \xi^k.$$

As $\text{Re}(h\lambda) \to -\infty$, the roots $\xi$ satisfy

$$\sigma(\xi) = 0.$$

The roots of $\sigma(\xi) = 0$ are $\xi_i = 0$, $i = 1, 2, \ldots, k$. So all solutions of the backward difference formulas decay as $\text{Re}(h\lambda) \to -\infty$.

### 5.7 Implementation: Error and step size control

There are several implementation issues to discuss, including (i) starting values, (ii) iteration for implicit methods, (iii) error estimation, (iv) step-size variation, and (iv) method order variation. We’ll begin with the solution of implicit difference equations, which are essential for stiff systems, but also preferred for non-stiff ODEs because of their smaller error coefficients and larger regions of absolute stability. Consider an implicit LMM of the form (5.1.2) and write it as

$$y_n + \sum_{i=1}^{k} \alpha_i y_{n-i} = h\beta_0 f(t_n, y_n) + h \sum_{i=1}^{k} \beta_i f_{n-i},$$

(5.7.1)

where $\alpha_0 = 1$ (as a normalization), $\beta_0 \neq 0$, and $y_{n-i}$ and $f_{n-i}$, $i = 1, 2, \ldots, k$, are assumed known. Equation (5.7.1) has a unique solution for $h$ sufficiently small that may be computed by functional iteration

$$y_n^{(\nu)} = -\sum_{i=1}^{k} \alpha_i y_{n-i} + h\beta_0 f(t_n, y_n^{(\nu-1)}) + h \sum_{i=1}^{k} \beta_i f_{n-i}, \quad \nu = 1, 2, \ldots,$$

(5.7.2a)

with $y_n^{(0)}$ being an initial guess for $y_n$. Convergence of the iteration occurs when $h$ satisfies

$$hL|\beta_0| < 1$$

(5.7.2b)
where $L$ is a Lipscitz constant for $f(t,y)$. If $f \in C^1$ then we may take
\[ L = \max_{y \in I} |f_y(t_n,y)|, \quad I = [y_n - \delta, y_n + \delta]. \quad (5.7.2c) \]

For non-stiff problems, where $L$ is small, the step size is usually determined by accuracy conditions rather than by (5.7.2b). Thus, functional iteration will give acceptable performance. However, for stiff problems, where $L$ is large, the step size is severely restricted by (5.7.2b). Newton's iteration is typically provides better performance in these cases. With Newton's method, we find the roots of
\[ F(y_n) = y_n + \sum_{i=1}^{k} \alpha_i y_{n-i} - h \beta_0 f(t_n, y_n) - h \sum_{i=1}^{k} \beta_i f_{n-i}. \quad (5.7.3a) \]
As noted in Section 2.2, Newton's method is obtained by linearizing (5.7.3a) about an iterate $y_n^{(\nu-1)}$; thus,
\[ 0 = F(y_n^{(\nu)}) = F(y_n^{(\nu-1)}) + F_y(y_n^{(\nu-1)})(y_n^{(\nu)} - y_n^{(\nu-1)}). \]
Using (5.7.3a) to calculate the derivative yields
\[ [1 - h \beta_0 f_y(t_n, y_n^{(\nu-1)})](y_n^{(\nu)} - y_n^{(\nu-1)}) = -F(y_n^{(\nu-1)}), \quad \nu = 1, 2, \ldots, \quad (5.7.3b) \]
Newton iteration converges when the initial guess $y_n^{(0)}$ is sufficiently close to $y_n$ and/or $h$ is sufficiently small. Unfortunately, it is difficult to get bounds like (5.7.2b,c) for Newton’s method; however, typically $h$ does not have to be as small as required for convergence with functional iteration (5.7.2). (Problem 1 contains a simple example.)

Newton’s iteration usually converges when the Jacobian $f_y$ is not reevaluated after each iteration. Thus, for example, we may use the same Jacobian for the entire iteration or even for iterations at several times.

With either functional or Newton iteration, a function evaluation must be made at each iteration. For vector systems, it is important to minimize the number of function evaluations and, thus, important to select an accurate initial guess $y_n^{(0)}$. Initial guess are usually chosen by a separate explicit method called a predictor. The implicit method is then called a corrector. For example, the Adams-Bashforth methods furnish predictors for use with Adams-Moulton correctors.

Two possible termination criteria for the corrector iteration are:
1. Iterate the corrector to convergence, e.g., terminate when
\[ |y^{(\nu^*)}_n - y^{(\nu^*-1)}_n| \leq \epsilon, \]

where \( \epsilon \) is on the order of the unit round off error of the computer. The local
discretization and stability of this predictor-corrector combination are determined
by the properties of the corrector alone; however, more function evaluations than
necessary may be required.

2. Iterate the corrector a fixed number of times. This procedure reduces the number of
function evaluations, but now the discretization error and stability characteristics
of the result contain a mixture of the properties of both the predictor and corrector.

A compromise between these two strategies is to iterate the corrector a fixed number
of times and to repeat the step with a smaller step size if adequate convergence was not
obtained.

In describing predictor-corrector methods, we will use the notation

- \( P \) to denote an application of the predictor step,
- \( C \) to denote an application of the corrector step, and
- \( E \) to denote a function evaluation, i.e., an evaluation of \( f \).

\section*{Example 5.7.1.} A method that uses a predictor to evaluate \( y^{(0)}_n \), does a function
evaluation \( f^{(0)}_n = f(t_n, y^{(0)}_n) \), and one corrector iteration to obtain \( y_n = y^{(1)}_n \) is a \( PEC \)
method. If \( \nu \) corrector iterations are performed and each iteration requires one evaluation
of \( f \), then the method is a \( P(EC)^\nu \) method. In this case, the last function evaluation
\( f(t_n, y^{(\nu-1)}_n) \) is saved as \( f_n \) for the next time step. Since we save \( y_n = y^{(\nu)}_n \), it may be
preferable to make an additional function evaluation and save \( f_n = f(t_n, y^{(\nu)}_n) \). This
method then becomes a \( P(EC)^\nu E \) method.

The orders of the predictor and corrector formulas need not be the same. Assume
that the predictor has an order \( p \) and the corrector has an order \( q \). Thus, the predictor satisfies

\[ y^{(0)}_n = y(t_n) + O(h^{p+1}) \]
and the corrector satisfies

\[ y_n = y(t_n) + O(h^{q+1}). \]

The \( \nu \)th iteration of the corrector satisfies

\[ y_n^{(\nu)} = y(t_n) + O(h^{p+1+\nu}) + O(h^{q+1}). \]

Thus, each corrector iteration decreases the effect of the predicted solution by one until the order of the corrector formula is reached. Since corrector iterations should be minimized, the order of the predictor is typically chosen as either \( q \) or \( q - 1 \).

When the predictor and corrector formulas both have the same order \( q \), the difference between them can be used to estimate the local error. Neglecting all errors prior to \( t_n \) and assuming the solution of the ODE to be sufficiently differentiable, the local error of the predicted and corrected solutions satisfy

\[ y(t_n) - y_n^{(0)} = C_q h^{q+1} y^{(q+1)}(t_n) + O(h^{q+2}) \]

and

\[ y(t_n) - y_n^{(\nu)} = C_q h^{q+1} y^{(q+1)}(t_n) + O(h^{q+2}). \]

The constants \( C_q^* \) and \( C_q \) are known error coefficients of the predictor and corrector, respectively. Subtracting the above two equations to eliminate the exact solution \( y(t_n) \) yields

\[ (C_q^* - C_q) h^{q+1} y^{(q+1)}(t_n) = y_n^{(\nu)} - y_n^{(0)} + O(h^{q+2}). \]

Thus, for example, the local error of the corrected solution may be estimated as

\[ C_q^* h^{q+1} y_n^{(q+1)} \approx \frac{C_q}{C_q^* - C_q} (y_n^{(\nu)} - y_n^{(0)}). \]  \hspace{1cm} (5.7.4)

**Example 5.7.2.** Consider the use of the fourth-order Adams-Bashforth method (5.3.13),

\[ y_n^{(0)} = y_{n-1} + \frac{h}{24} (55 f_{n-1} - 59 f_{n-2} + 37 f_{n-3} - 9 f_{n-4}), \quad d_n = \frac{251}{720} h^5 y^{(5)}(t_n) + O(h^6). \]

as a predictor and the fourth-order Adams-Moulton method (5.4.8),

\[ y_n^{(\nu)} = y_{n-1} + \frac{h}{24} (9 f_n^{(\nu-1)} + 19 f_{n-1} - 5 f_{n-2} + f_{n-3}), \quad d_n = \frac{19}{720} h^5 y^{(5)}(t_n) + O(h^6). \]
With \( p = q = 4 \), \( C_4^* = 251/720 \) and \( C_4 = -19/720 \). Using (5.7.4), the local error of the Adams-Moulton corrector may be estimated as

\[
y(t_n) - y_n^{(\nu)} \approx -\frac{19}{270}(y_n^{(\nu)} - y_n^{(0)}).
\]

Error estimates obtained by (5.7.4) are used to control step sizes so that specified local error tolerances are satisfied. Chosen step sizes must also satisfy appropriate (relative or absolute) stability conditions and ensure the convergence of the iterative scheme.

There is a second method of estimating the local errors of LMMs that does not rely on the predictor and corrector formulas having the same order. The development will also illustrate the appropriate information to be saved between time steps. We’ll present the results in general but illustrate the approach when the third-order Adams-Bashforth method is used as a predictor and when the fourth-order Adams-Moulton method is as a corrector. Thus, from (5.3.12a) and (5.4.8a), we have

\[
y_n^{(0)} = y_{n-1} + \frac{h}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3}) \quad (5.7.5a)
\]

\[
y_n^{(\nu)} = y_{n-1} + \frac{h}{24}(9f_n^{(\nu-1)} + 19f_{n-1} - 5f_{n-2} + f_{n-3}), \quad (5.7.5b)
\]

Let us write these methods in a vector form by introducing

\[
y_{n-1} = \begin{bmatrix} y_{n-1} \\ hy'_{n-1} \\ hy''_{n-2} \\ hy'''_{n-3} \end{bmatrix}, \quad y_n = \begin{bmatrix} y_n \\ hy'_n \\ hy''_{n-1} \\ hy'''_{n-2} \end{bmatrix}, \quad (5.7.6a)
\]

where \( y'_n = f_n, etc \). The vector \( y_{n-1} \) contains the information needed to calculate \( y_n \) and \( y_n \) contains the information that is transferred to the next step. With this notation, the predicted solution (5.7.5a) is

\[
\begin{bmatrix} y_n^{(0)} \\ hy''^{(0)} \\ hy''_{n-1} \\ hy''_{n-2} \end{bmatrix} = \begin{bmatrix} 1 & 23/12 & -16/12 & 5/12 \\ 0 & 3 & -3 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_{n-1} \\ hy'_{n-1} \\ hy''_{n-1} \\ hy'''_{n-3} \end{bmatrix} \quad (5.7.6b)
\]

or

\[
y_n^{(0)} = By_{n-1}. \quad (5.7.6c)
\]

49
The first row of (5.7.6b) is just the predictor formula (5.7.5a). The third and fourth rows are obvious identities. The second row can be developed from the Newton interpolation formula (5.3.5a), which we repeat here for convenience,

\[ P_{k-1}(t) = \sum_{i=0}^{k-1} (-1)^i \binom{-\tau}{i} \nabla^i y_{n-1} \]  \hspace{1cm} (5.7.7a)

where

\[ \tau = \frac{t - t_{n-1}}{h} \]  \hspace{1cm} (5.7.7b)

and \( k = 3 \) for the present example. Recall that \( P_{k-1}(t) \) provides an approximation of \( y'(t) = f(t, y(t)) \) on the interval \([t_{n-k}, t_n]\); thus, setting \( \tau = 1 \) \( (t = t_n) \) in (5.7.7a) yields

\[ y_n^{(0)} = P_{k-1}(t_n) = \sum_{i=0}^{k-1} (-1)^i \binom{-1}{i} \nabla^i y_{n-1} \]

But (cf. (5.3.4c,d))

\[ (-1)^i \binom{-1}{i} = 1; \]

thus,

\[ y_n^{(0)} = \sum_{i=0}^{k-1} \nabla^i y_{n-1}. \]

Setting \( k = 3 \) for the example at hand

\[ y_n^{(0)} = y_{n-1} + (y'_{n-1} - y'_{n-2}) + (y'_{n-2} - 2y'_{n-3}) + y'_{n-3} \]

or

\[ y_n^{(0)} = 3y'_{n-1} - 3y'_{n-2} + y'_{n-3}. \]

In a similar manner, we write the corrector as

\[ y_n^{(\nu)} = y_n^{(\nu-1)} + cg(y_n^{(\nu-1)}), \hspace{1cm} \nu = 1, 2, \ldots \]  \hspace{1cm} (5.7.8a)

The vector \( c \) and the scalar function \( g(y_n^{(\nu-1)}) \) must be determined so that (5.7.8) agrees with the original corrector formula. We’ll illustrate the procedure for the third-order corrector (5.7.5b). Setting \( \nu = 1 \) in (5.7.5b) and subtracting (5.7.5a) yields

\[ y_n^{[1]} = y_n^{[0]} + \frac{3h}{8} [f(t_n, y_n^{[0]}) - 3y_{n-1} + 3y_{n-2} - y_{n-3}]. \]
From the above predicted interpolant, we have

\[ y_n^{(0)} - 3y_{n-1} + 3y_{n-2} - y_{n-3} = 0. \]

Let us define \( y_n^{(1)} = f(t_n, y_n^{(0)}) \) and add this to both sides of this relation to obtain

\[ y_n^{(1)} = y_n^{(0)} + f(t_n, y_n^{(0)}) - 3y_{n-1} + 3y_{n-2} - y_{n-3}. \]

The previous relations may be used for all iterates and not just for the first correction; thus, we obtain (5.7.8a) with

\[ c = \begin{bmatrix} 3/8 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \] (5.7.8b)

\[ g(y_n^{(\nu-1)}) = h[f(t_n, y_n^{(\nu-1)}) - 3y_{n-1} + 3y_{n-2} - y_{n-3}]. \] (5.7.8c)

The form of the predictor-corrector pair given by (5.7.6c) and (5.7.8) avoids saving \( y_{n-1}, y_{n-2}, \) and \( y_{n-3} \). Functional iteration is assumed. The formulas would have to be modified for Newton iteration. Our development does not reveal the generality of (5.7.6c) and (5.7.8). Gear [10], Chapters 7 and 9, provides additional details. The final converged iterates are passed to the next step; thus, if convergence occurs after \( \nu \) corrector iterations, \( y_n = y_n^{(\nu)} \).

The representation (5.7.6c) and (5.7.8) may not be best when automatic step and order changes are involved. Formulas involving backward differences may, for example, be best for these cases. Having the general representation (5.7.6c) and (5.7.8), we can switch to another one by a linear transformation. Thus, let

\[ a_n = Ty_n \] (5.7.9a)

and use (5.7.6b) and (5.7.8) to get a new predictor-corrector pair

\[ a_n^{(0)} = Aa_{n-1}, \] (5.7.9b)

\[ a_n^{(\nu)} = a_n^{(\nu-1)} + 1P(a_n^{(\nu)}), \] (5.7.9c)
where
\[
A = \mathbf{TBT}^{-1}, \quad \mathbf{l} = \mathbf{Tc}, \quad P(\mathbf{a}_n) = g(\mathbf{T}^{-1}\mathbf{a}_n).
\] (5.7.9d)

**Example 5.7.3.** Let us develop the backward-difference form of the Adams predictor-corrector pair (5.7.5a,b). Let

\[
\mathbf{a}_n = \begin{bmatrix}
y_n \\
hy'_n \\
\nabla hy'_n \\
\nabla^2 hy'_n
\end{bmatrix}.
\]

Using the definition of the backward-difference operators (5.2.15)

\[
\begin{bmatrix}
y_n \\
hy'_n \\
\nabla hy'_n \\
\nabla^2 hy'_n
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & -1 & 0 \\
0 & 1 & -2 & 1
\end{bmatrix} \begin{bmatrix}
y_n \\
hy'_n \\
hy'_{n-1} \\
hy'_{n-2}
\end{bmatrix} = \mathbf{Ty}_n.
\]

Nordsieck [15] suggested representing the solution in terms of scaled derivatives at \(t = t_n\), i.e., as

\[
\mathbf{a}_n = [y_n, hy'_n, \frac{h^2}{2!}y''_n, \ldots, \frac{h^{k-1}}{(k-1)!}y^{(k-1)}_n]^T.
\] (5.7.10)

Step size variation is simplified with this representation. The transformation between representations may be obtained from the Newton backward-difference polynomial (5.3.3a)

\[
P_{L-1}(t) = \sum_{i=0}^{L-1} (-1)^i \binom{-\tau + 1}{i} \nabla^i y'_n.
\] (5.7.11)

(The relationship between the order \(k\) and the degree of the Newton polynomial \(l-1\) will become clear shortly.) Since \(P_{L-1}(t)\) is an approximation of \(y'(t)\), differentiating (5.7.11) and setting \(\tau = 1\) gives a relation between the Nordsieck and backward-difference representations. A similar linear relationship between the backward-difference and standard representation (5.7.6b) can be used to relate the Nordsieck and the standard variables.

**Example 5.7.4.** We'll construct a relationship between the Nordsieck and standard representations for the three-step (fourth-order) Adams method (5.7.5b). The Nordsieck vector for this method is

\[
\mathbf{a}_n = [y_n, hy'_n, \frac{h^2}{2}y''_n, \frac{h^3}{6}y'''_n]^T.
\]
A second-degree polynomial suffices to get the necessary derivatives; thus, setting \( l = 3 \) in (5.7.11), we have
\[
P_2 = y'_n + (\tau - 1)\nabla y'_n + \frac{1}{2}(\tau - 1)\tau \nabla^2 y'_n.
\]
Differentiating while using (5.7.7b)
\[
h \frac{dP_2(t)}{dt} = \frac{dP_2(\tau)}{d\tau} = \nabla y'_n + \frac{1}{2}(2\tau - 1)\nabla^2 y'_n.
\]
Setting \( \tau = 1 \) \((t = t_n)\) gives an approximation of \( y''_n \) as
\[
h y''_n = \nabla y'_n + \frac{1}{2}\nabla^2 y'_n.
\]
Differentiating again
\[
h^2 \frac{d^2P_2(t)}{dt^2} = \frac{d^2P_2(\tau)}{d\tau^2} = \nabla^2 y'_n.
\]
Setting \( \tau = 1 \)
\[
h^2 y'''_n = \nabla^2 y'_n.
\]
Summarizing the results in matrix form
\[
\begin{bmatrix}
y_n \\
y'_n \\
y''_n \\
y'''_n
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1/2 & 1/4 \\
0 & 0 & 0 & 1/6
\end{bmatrix}
\begin{bmatrix}
y_n \\
y'_n \\
y''_n \\
y'''_n
\end{bmatrix}.
\]
Multiplying by the transformation of Example 5.7.3 gives the desired relationship between the Nordsieck and traditional forms as
\[
\begin{bmatrix}
y_n \\
y'_n \\
y''_n \\
y'''_n
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 3/4 & -1 & 1/4 \\
0 & 1/6 & -1/3 & 1/6
\end{bmatrix}
\begin{bmatrix}
y_n \\
y'_n \\
y''_n \\
y'''_n
\end{bmatrix}.
\]

Example 5.7.5. Storage was at a premium when the above transformations were developed. Now, with a third-degree interpolating polynomial and less pressure on storage, we may store an extra derivative. Considering (5.7.5b) once again, let us approximate the solution derivative more accurately using a third-degree polynomial. Thus, using (5.7.11) with \( l = k = 4 \)
\[
P_3 = y'_n + (\tau - 1)\nabla y'_n + \frac{1}{2}(\tau - 1)\tau \nabla^2 y'_n + \frac{1}{6}(\tau - 1)\tau(\tau + 1)\nabla^3 y'_n.
\]
Differentiating while using (5.7.7b)

\[
dP_3(t) \frac{d}{dt} = \frac{dP_3(\tau)}{d\tau} = \nabla y'_n + \frac{1}{2}(2\tau - 1) \nabla^2 y'_n + \frac{1}{6}(3\tau^2 - 1) \nabla^3 y'_n.
\]

Setting \( \tau = 1 \)

\[
h y''_n = \nabla y'_n + \frac{1}{2} \nabla^2 y'_n + \frac{1}{3} \nabla^3 y'_n.
\]

Differentiating again

\[
h^2 \frac{d^2 P_3(t)}{dt^2} = \frac{d^2 P_3(\tau)}{d\tau^2} = \nabla^2 y'_n + \tau \nabla^3 y'_n.
\]

Setting \( \tau = 1 \)

\[
h^2 y'''_n = \nabla^2 y'_n + \nabla^3 y'_n.
\]

Differentiating once again

\[
h^3 y''''_n = \nabla^3 y'_n.
\]

Thus, enlarging the Nordsieck vector by one, we have

\[
\begin{bmatrix}
y_n \\
y'_n \\
hy''_n/2 \\
h^2 y'''_n/3! \\
h^3 y''''_n/4!
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1/2 & 1/4 & 1/6 \\
0 & 0 & 0 & 1/6 & 1/6 \\
0 & 0 & 0 & 0 & 1/24
\end{bmatrix}
\begin{bmatrix}
y_n \\
y'_n \\
hy''_n \\
h^2 y'''_n/3! \\
h^3 y''''_n/4!
\end{bmatrix}.
\]

We’ve already seen that the local error of a \( k \)th-order multistep method has the form

\[
d_n = C_k h^{k+1} y^{(k+1)}(t_n) + O(h^{k+2}). \tag{5.7.12a}
\]

The Nordsieck vector (5.7.10) contains approximations of \( y_n \) and its first \( k-1 \) derivatives. However, by enlarging it to store the \( k \)th derivative (Example 5.7.5), we can construct an error estimate. Thus, let

\[
a_n = [y_n, \ h y'_n, \ \frac{h^2}{2!} y''_n, \ \ldots, \ \frac{h^k}{k!} y^{(k)}_n]^T \tag{5.7.12b}
\]

and take a backward difference at two consecutive time steps to obtain

\[
\nabla a_n = a_n - a_{n-1}.
\]
The last component of this vector $\nabla a_{n,k+1}$ is an approximation of $h^{k+1}y^{(k+1)}(t_n)/k!$. The local error can, thus, be approximated as

$$d_n \approx C_k k! \nabla a_{n,k+1}. \quad (5.7.12c)$$

**Example 5.7.6.** The local discretization error of the fourth-order Adams-Moulton method (5.7.5b) is given by (5.4.8b) as

$$d_n = -\frac{19}{720} h^5 y^v(t_n) + O(h^5).$$

Differencing the last component of the Nordsieck vector of Example 5.7.5 at $t_n$ and $t_{n-1}$ gives

$$\nabla a_{n,5} = \frac{h^4}{24} (y^w_{n} - y^w_{n-1}) = \frac{h^5}{24} \frac{y^w_n - y^w_{n-1}}{h} \approx \frac{h^5}{24} y^v(t_n).$$

Substituting into the discretization error formula

$$d_n \approx -\frac{19}{30} \nabla a_{n,5}.$$

Suppose we seek to keep $|d_n| \approx \epsilon$, then, either upon completion of a successful step or upon failure of a step, we can use (5.7.12a) to calculate a new step size $\alpha h$ such that

$$|C_k(\alpha h)^{k+1}y^{(k+1)}(t_n)| \approx \epsilon$$

or, using (5.7.12c),

$$|C_k k! \alpha^{k+1} \nabla a_{n,k+1}| \approx \epsilon.$$

Thus,

$$\alpha \approx \left( \frac{\epsilon}{|C_k k! \nabla a_{n,k+1}|} \right)^{1/(k+1)}. \quad (5.7.13a)$$

Knowing that the local error of the next (higher-order) method of a sequence of methods is

$$d_n = C_{k+1} h^{k+2} y^{(k+2)}(t_n) + O(h^{k+2}),$$

we can also calculate an approximation of $y^{(k+2)}(t_n)$ that can be used to change orders. Using second-order backward differences of the Nordsieck vector

$$\nabla^2 a_{n,k+1} \approx \frac{h^{k+2} y^{(k+2)}(t_n)}{k!}.$$
and
\[ d_n \approx C_{k+1} k! \nabla^2 a_{n,k+1}. \quad (5.7.13b) \]

Estimates of the error of the next lower-order method follow directly from the next-to-last entry in the Nordsieck array. Thus, suppose that the error formula for the \( k-1 \)-order method is
\[ d_n = C_{k-1} h^k y^{(k)}(t_n) + O(h^{k+1}). \]

Then, (cf. Example 5.7.5 and (5.7.12b))
\[ a_{n,k-1} = \frac{h^{k-1}}{(k-1)!} y^{(k-1)}[n] \]

and
\[ d_n \approx C_{k-1} (k-1)! h a_{n,k-1}. \quad (5.7.13c) \]

Formulas (5.7.13b,c) can be used to increase or decrease the method order by one. These order variations can be combined with step-size variations to produce a LMM code capable of both step and order adjustments. Most LMM codes do this according to the following guidelines.

1. Start the code with a first-order method. For the Adams methods, this would be the Euler-backward Euler pair. This avoids the need to incorporate separate one-step (Runge-Kutta) software in the code.

2. Change order before step size. Changing order is generally much more efficient than changing step size. Order increases and decreases in unit amounts can proceed by examining changes in the sequence of derivatives \( y_n^{(k-1)} \), \( y_n^{(k)} \), and \( y_n^{(k+1)} \). These derivatives may be computed as described for (5.7.13). If the sequence of derivatives are decreasing in magnitude then computational reductions can be achieved by increasing the method order from \( k \) to \( k+1 \). On the contrary, if the sequence of derivatives are increasing, a reduction of order may be appropriate.

3. Include heuristics to avoid increasing the order too often. One possibility is to do at least \( k \) steps with a method of order \( k \) before considering an order change.
Starting values when changing step size or order may be computed by a Taylor’s series when using the Nordsieck representation (5.7.10).

There are also variable step size LMMs ([12], Section III.5). The coefficients of these formulas are functions of $h$. The variable step formulas are generally more stable than the uniform step formulas but are less efficient.

Some available LMM codes are

1. *DEABM* is a modification of a code developed by Shampine and Gordon [16]. This code implements a variable step size divided difference representation of the Adams formulas. It uses a *PECE* strategy and includes order variation. Let’s go over the order variation scheme to illustrate the technique. After performing a step with an order $k$ method, compute estimates $d_{n}^{k-2}$, $d_{n}^{k-1}$, and $d_{n}^{k}$ of the local error of solutions with methods of order $k - 2$, $k - 1$, and $k$, respectively. Reduce the order to $k - 1$ if

$$\max[\| d_{n}^{k-2} \|, \| d_{n}^{k-1} \|] \leq \| d_{n}^{k} \|. \tag{5.7.14}$$

Increase the order when a step is successful, (5.7.14) is violated, and a constant step size is used. (Remember, these are variable step-size methods.) Estimates of local discretization errors are obtained using approaches similar to (5.7.13). Norms are used for vector systems.

2. *EPISODE*, developed by Byrne and Hindmarsh [7], is a variable step, variable order implementation of the constant-step Adams and BDF methods using the Nordsieck representation. For nonstiff problems, functional iteration uses a $P(EC)^{\nu}$ strategy. Newton’s method is used for stiff problems.

3. *LSODE* is another implementation of the constant-step Adams and BDF methods. It is similar to EPISODE.

4. *VODE* is a variable step, variable order code based on the variable-step Adams and BDF formulas. It was developed by Brown *et al.* [5] and is an extension of *EPISODE*.
Table 5.7.1: Legends for Figures 5.7.1 and 5.7.2 and code storage [12].

<table>
<thead>
<tr>
<th>Code</th>
<th>Symbol</th>
<th>Storage/eqn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEABM</td>
<td>- - - -</td>
<td>22</td>
</tr>
<tr>
<td>EPISODE</td>
<td>-...-</td>
<td>18</td>
</tr>
<tr>
<td>LSODE</td>
<td>-.-.-</td>
<td>17</td>
</tr>
<tr>
<td>D02CAF</td>
<td>......</td>
<td>19</td>
</tr>
<tr>
<td>DOPRI8</td>
<td>- - - -</td>
<td>9</td>
</tr>
</tbody>
</table>

5. DASSL, developed by Petzold (cf. [4]) is a backward difference code for stiff differential and differential algebraic systems. DASSL addresses differential equations in the implicit form

\[ \mathbf{f}(t, \mathbf{y}, \mathbf{y}') = 0. \]

It can also solve these systems when the Jacobian \( \mathbf{f}_y \) is not invertible. In these cases, \( \mathbf{y}' \) cannot be determined and the problem is a system of differential-algebraic equations. A simple two-dimensional example is

\[
y'_1 = a_{11}y_1 + a_{12}y_2 + b_1, \\
0 = a_{21}y_1 + a_{22}y_2 + b_2.
\]

Many of these codes have been incorporated into scientific subroutine libraries. For example, the IVPAG procedure in the IMSL Library is a modification of Gear’s [10] original Adam’s and BDF procedure. The procedures \textit{D02PCF} and \textit{D02EJF}, respectively, are variable order, variable step Adams and BDF codes in the NAG Library.

Example 5.7.7. Hairer et al. [12], Section III.7, compare several codes on a suite of six non-stiff problems. We report their results in Figures 5.7.1 and 5.7.2. Codes that have not already been identified include \textit{DOPRI8}, which is the eighth-order Dormand and Prince Runge-Kutta method [9], and \textit{D02CAF}, which is similar to \textit{LSODE} and \textit{EPISODE}, and contained in the NAG library. The six problems used for testing are described in Hairer et al. [12], Section II.10. The legends and storage requirements for each code are given in Table 5.7.1.

The Runge-Kutta code DOPRI8 generally uses more function evaluations but less CPU time than the LMM codes. The test problems are fairly simple, but the performance
Figure 5.7.1: Number of function evaluations for a six-problem suite of non-stiff problems [12].
Figure 5.7.2: CPU times (seconds) for a six-problem suite of non-stiff problems [12].
of the high-order Runge-Kutta method should not be ignored. Conditions, however, could be reversed with more complicated functions. Of the LMM codes, \textit{DEABM} seems to be the most efficient and \textit{EPISODE} the least. From Table 5.7.1, we see that DOPPI8 requires about half of the storage per equation as do the LMMs. Of the LMM codes, \textit{DEBEAM} has the greatest storage cost.

\textit{Example 5.7.8.} Hairer and Wanner [13], Section V.5, compare several codes for a suite of stiff problems. We report their results in Figures 5.7.3 and 5.7.4. The codes used for this test follow.

1. \textit{LSODE} and \textit{VODE} are the codes described previously with the BDF option set.

2. \textit{DEBDF} is a driver for \textit{LSODE} for stiff systems.

3. \textit{SPRINT}, developed by Berzins and Furzeland [2], contains several multistep methods and solution packages. The one used for the test is a blended multistep method that is A-stable to order four.

4. \textit{SECDER} and \textit{ROBER} are multistep codes that won’t be described further.

5. \textit{LADAMS} is \textit{LSODE} with the Adams methods selected.

6. \textit{RADAU5} is a fifth-order implicit Runge-Kutta method based on collocation at Radau points.

The results are scattered and difficult to interpret. The implicit Runge-Kutta method appears to do reasonably well on the small stiff problems of Figure 5.7.3 but less well on the larger problems of Figure 5.7.4. The Adams software did well on several of the smaller stiff problems but had difficulties with the larger problems. The opposite appears true for \textit{VODE}. The SPRINT software also did well on several of the larger problems.

Some final notes of comparison between Runge-Kutta and LMMs follow:

1. Runge-Kutta methods are preferred to Adams methods when function evaluations are inexpensive.

2. In general, the stability of an Adams predictor-corrector pair is better than that of an explicit Runge-Kutta method of the same order.
Figure 5.7.3: CPU time vs. error for a suite of small stiff problems [13].
Figure 5.7.4: CPU times vs. error for a suite of large stiff problems [13].
3. Estimation of local errors is less expensive with a LMM than with a Runge-Kutta method. As seen in Examples 5.7.7 and 5.7, however, Runge-Kutta methods remain competitive with LMMs as long as the function evaluations aren’t too expensive.

4. Fixed-order Runge-Kutta methods are easier to implement than LMMs. However, good software of both types exist.

5. For large stiff problems, BDFs are much more efficient than implicit Runge-Kutta methods unless the differential system has rapidly oscillating solutions. The solution of the BEAM problem of Example 5.7.4 is oscillatory and many BDF codes failed on it. The Runge-Kutta code RADAU5 was the second-most efficient method on this problem. The A-stable method within the SPRINT package was, by far, the most efficient technique.

Problems

1. Consider the solution of

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

which has the exact solution

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

This IVP is stiff when Re(\lambda) \ll 0. In this case, the solution behaves like \( e^{\lambda t} \), i.e., like the solution of the linear problem \( y' = \lambda y \). Suppose that this problem is solved by the backward Euler method.

1.1. Find the maximum step size \( h \) for which functional iteration (5.7.2) converges when \( \lambda = -10^4 \).

1.2. Show that Newton’s iteration converges for much larger step sizes.
Bibliography


