A brief note on linear multistep methods

Alen Alexanderian*

Abstract

We review some key results regarding linear multistep methods for solving initial value problems. Moreover, we discuss some common classes of linear multistep methods.

1 Basic theory

Here we provide a summary of key results regarding stability and consistency of linear multistep methods. We focus on the initial value problem,

$$y' = f(x, y), \quad x \in [a, b], \quad y(a) = y_0. \tag{1.1}$$

Here $y(x) \in \mathbb{R}$ is the state variable, and all throughout, we assume that f satisfies a uniform Lipschitz condition with respect to y.

Definition of a multistep method. The generic description of a linear multistep method is as follows: define nodes $x_n = a + nh$, $n = 0, \ldots, N$, with $h = (b-a)/N$. The general form of a multistep method is

$$y_{n+1} = \sum_{j=0}^{p} \alpha_j y_{n-j} + h \sum_{j=-1}^{p} \beta_j f(x_{n-j}, y_{n-j}), \quad n \geq p. \tag{1.2}$$

Here $\{\alpha_i\}_{i=0}^{p}$, and $\{\beta_i\}_{i=-1}^{p}$ are constant coefficients, and $p \geq 0$. If either $\alpha_p \neq 0$ or $\beta_p \neq 0$, then the method is called a $p+1$ step method. The *initial values*, $y_0, y_1, \ldots, y_p$, must be obtained by other means (e.g., an appropriate explicit method). Note that if $\beta_{-1} = 0$, then the method is explicit, and if $\beta_{-1} \neq 0$, the method is implicit. Also, here we use the convention that $y(x_n)$ is the exact value of y at x_n and y_n is the numerically computed approximation to $y(x_n)$; i.e., $y_n \approx y(x_n)$.

Truncation error and accuracy. The local trunction error of the method is defined by

$$T_n(y; h) = \frac{1}{h}\left[y(x_{n+1}) - \left(\sum_{j=0}^{p} \alpha_j y(x_{n-j}) + h \sum_{j=-1}^{p} \beta_j f(x_{n-j}, y(x_{n-j})) \right) \right]. \tag{1.3}$$

Note that in many texts, $\tau_n(y)$ is used to denote the $T_n(y; h)$ defined above. The present notation is adapted from [2]. Note also that we can rewrite the expression for the local truncation error as,

$$T_n(y; h) = \frac{1}{h}\left[y(x_{n+1}) - \left(\sum_{j=0}^{p} \alpha_j y(x_{n-j}) + h \sum_{j=-1}^{p} \beta_j y'(x_{n-j}) \right) \right],$$

which defines the local truncation error for integrating $y'(x)$.

Next, we define,

$$\tau(h) := \max_{x_p \leq x_n \leq b} |T_n(y; h)|.$$

We say the method is consistent if

$$\tau(h) \to 0, \quad \text{as } h \to 0. \tag{1.4}$$

*North Carolina State University, Raleigh, NC, USA. E-mail: alexanderian@ncsu.edu

Last revised: April 27, 2022.

The speed of convergence of the numerical solution to the true solution is related to the speed of convergence in (1.4). Hence, we need to understand the conditions under which

$$\tau(h) = \mathcal{O}(h^m), \tag{1.5}$$

as $h \to 0$. The largest value of m for which this holds is called the order of the method (1.2).

Theorem 1.1 ([1, Theorem 6.5])**.** *Let $m \geq 1$ be an integer. For (1.4) to hold for all continuously differentiable functions $y(x)$, that is for the method (1.2) to be consistent, it is necessary and sufficient that*

$$\sum_{j=0}^{p} \alpha_j = 1, \quad -\sum_{j=0}^{p} j\alpha_j + \sum_{j=-1}^{p} \beta_j = 1. \tag{1.6}$$

For (1.5) to hold for all $y(x)$ that are $m+1$ times continuously differentiable, it is necessary and sufficient that (1.6) hold and that

$$\sum_{j=0}^{p} (-j)^i \alpha_j + i \sum_{j=-1}^{p} (-j)^{i-1} \beta_j = 1, \quad i = 2, 3, \ldots, m.$$

Examples. The following is an example of a linear multistep method, known as the midpoint method:

$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n), \quad n \geq 1$$

Notice that y_1 needs to be computed before the iterations can begin. This can be done via one step of explicit Euler: $y_1 = y_0 + hf(x_0, y_0)$.

The truncation error for this method is given by:

$$T_n(y; h) = \frac{1}{h}\left[y(x_{n+1}) - y(x_{n-1}) - 2hy'(x_n)\right]. \tag{1.7}$$

To compute this, we use Taylor expansion of $y(x_{n+1})$ and $y(x_{n-1})$:

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + hy'(x_n) + \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y'''(x_n) + O(h^4), \\ y(x_{n-1}) &= y(x_n) - hy'(x_n) + \frac{h^2}{2}y''(x_n) - \frac{h^3}{6}y'''(x_n) + O(h^4). \end{aligned}$$

Substituting Taylor expansions of $y(x_{n+1})$ and $y(x_{n-1})$ in (1.7), we have

$$\begin{aligned} T_n(y; h) = \frac{1}{h}\Bigg[&y(x_n) + hy'(x_n) + \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y'''(x_n) + O(h^4) \\ &-y(x_n) + hy'(x_n) - \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y'''(x_n) + O(h^4) - 2hy'(x_n)\Bigg] \\ = \frac{h^2}{3}&y'''(x_n) + O(h^3). \end{aligned}$$

This shows, in particular, that the method is second order.

The following is another example of a linear multistep method:

$$y_{n+1} = 3y_n - 2y_{n-1} + \frac{h}{2}[f(x_n, y_n) - 3f(x_{n-1}, y_{n-1})], \quad n \geq 1. \tag{1.8}$$

Using a similar approach as above, we can compute the truncation error for this method:

$$T_n(y; h) = \frac{7}{12}h^2 y'''(x_n) + O(h^3).$$

Thus, this is also a second order method.

A simple but useful convergence result. The following convergence result is Theorem 6.6 in [1]. While this is not the most general convergence result for linear multistep methods, it applies to a wide class of such methods.

Theorem 1.2. *Consider solving the initial value problem (1.1) using the linear multistep method (1.2) with a step size h to obtain $\{y_n\}_{n=0}^N$ that approximate $\{y(x_n)\}_{n=0}^N$. Let the initial errors satisfy,*

$$\eta(h) := \max_{0\le n\le p} |y(x_n) - y_n| \to 0, \quad \text{as } h \to 0.$$

Assume that the method is consistent, and assume that the coefficients $\{\alpha_j\}_{j=0}^p$ in (1.2) are non-negative. Then, the method is convergent, and

$$\max_{a\le x_n\le b} |y(x_n) - y_n| \le c_1\eta(h) + c_2\tau(h),$$

where c_1 and c_2 are suitable constants. If the method is of order m, and if the initial errors satisfy $\eta(h) = \mathcal{O}(h^m)$, then the speed of convergence is $\mathcal{O}(h^m)$.

Remark 1.3. *The above result states that to ensure a rate of convergence of $O(h^m)$ for the method (1.2), it is necessary that the method be of order m, i.e, each step has an error of $O(h^{m+1})$ (recall truncation error (1.3) is this error divided by h). However, the initial values $y_0, \ldots, y_p$ need to be computed with only an accuracy of $O(h^m)$. Thus, the initial values can be computed with a lower order method. This is also noted in [1, p. 361]. As an example, and as mentioned before, the midpoint method, which is of second order, can be initialized with explicit Euler, which is only first order accurate.*

General stability and convergence theory. To obtain a more general convergence result, without the non-negativity assumption on α_j, we need to investigate the stability of the method (1.2) more closely. The general theory is rather involved and is discussed in texts such as [1, 2, 3]. The stability of (1.2) is linked to the polynomial

$$\rho(r) = r^{p+1} - \sum_{j=0}^{p} \alpha_j r^{p-j}.$$

We note immediately that by the consistency condition, we have $\rho(1) = 0$. Let $r_0, r_1, \ldots, r_p$ denote roots of ρ, repeated according to their multiplicity, and let $r_0 = 1$. The method (1.2) satisfies the *root condition* if the following hold:

$$|r_j| \le 1, \quad j = 0, 1, \ldots, p \tag{1.9}$$

$$|r_j| = 1 \Rightarrow \rho'(r_j) \ne 0. \tag{1.10}$$

We note that (1.9) requires all roots of ρ lie inside the unit circle $\{z \in \mathbb{C} : |z| \le 1\}$, and (1.10) states that the roots on the boundary of the unit circle are to be simple roots (i.e., of multiplicity one) of $\rho(r)$.

A key result on linear multistep methods says that (1.2) is stable if and only if the root condition holds. The general convergence result for linear multistep methods states that a consistent linear multistep method is convergent if and only if it satisfies the root condition; see [1, 2] for more details.

As an example, consider the midpoint method described above. For that method

$$\rho(r) = r^2 - 1,$$

which has roots $r = \pm 1$. Therefore, the method satisfies the root condition, and is thus stable. Next, we study stability of (1.8). The method has $\alpha_0 = 3$ and $\alpha_1 = -2$ and thus, for this method,

$$\rho(r) = r^2 - 3r + 2 = (r-1)(r-2),$$

which has roots $r_1 = 1$ and $r_2 = 2$ and thus violates the root condition. Therefore, the method (1.8) is unstable.

2 Some commonly used linear multistep methods.

We will focus on the following classes of methods:

1. Backward differentiation formulas (BDFs)
2. Adams methods
   (a) Adams–Bashforth (explicit)
   (b) Adams–Moulton (implicit)
3. Predictor corrector methods

2.1 Backward differentiation formulas

These methods are obtained based on numerical differentiation formulas.

Basic idea: Interpolate past values of $y(x)$, and then differentiate the interpolating polynomial to approximate y'.

Specifically, let $q(x)$ be the polynomial of degree $\geq p$ that interpolates $y(x)$ at the points $x_{n+1}, x_n, \ldots, x_{n-p+1}$, for some $p \geq 1$:

$$q(x) = \sum_{j=-1}^{p-1} y(x_{n-j})\ell_j(x), \tag{2.1}$$

where $\ell_j(x)$'s are elementary Lagrange polynomials. We then use,

$$q'(x_{n+1}) \approx y'(x_{n+1}) = f(x_{n+1}, y(x_{n+1})); \tag{2.2}$$

combine this with (2.1), and solve for $y(x_{n+1})$ to obtain a formula of the form

$$y(x_{n+1}) \approx \sum_{j=0}^{p-1} \alpha_j y(x_{n-j}) + h\beta f(x_{n+1}, y(x_{n+1})),$$

for approximating $y(x_{n+1})$. This leads to the definition of the p-step BDF formula:

$$y_{n+1} = \sum_{j=0}^{p-1} \alpha_j y_{n-j} + h\beta f(x_{n+1}, y_{n+1}).$$

The coefficients $\{\alpha_j\}$ and β can be obtained for specific choices of p, by using the appropriate numerical differentiation formula. Note that all we are doing is finding a numerical differentiation formula to approximate $y'(x_{n+1})$ using function values at $x_{n+1}, x_n, \ldots, x_{n-p+1}$. The useful cases are $p = 1, \ldots, 6$, coefficients for which can be found in [1, p. 411]. Note that p-step BDF formulas with $p \geq 7$ are unstable, and should not be used; see [4, p. 381].

Examples of BDF methods. We consider a couple of examples to help clarify matters. Consider the case of $p = 1$. In that case $q(x)$ is just the linear interpolating polynomial that interpolates y at the nodes x_n and x_{n+1} (drawing a picture might be helpful), for which $q'(x) = (y(x_{n+1}) - y(x_n))/h$. Substituting this in (2.2) gives,

$$\frac{y(x_{n+1}) - y(x_n)}{h} \approx f(x_{n+1}, y(x_{n+1})),$$

which leads to the approximation:

$$y(x_{n+1}) \approx y(x_n) + hf(x_{n+1}, y(x_{n+1})).$$

This gives the numerical method

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}),$$

which is none but backward Euler!

Next, let us consider $p = 2$. In this case, we can derive the numerical differentiation formula for $y'(x_{n+1})$ using function values at x_{n+1}, x_n, and x_{n-1}:

$$y'(x_{n+1}) \approx \frac{3y(x_{n+1}) - 4y(x_n) + y(x_{n-1})}{2h}.$$

Using this in (2.2), we obtain

$$\frac{3y(x_{n+1}) - 4y(x_n) + y(x_{n-1})}{2h} \approx f(x_{n+1}, y(x_{n+1})).$$

Following the same steps as outlined above, gives the BDF2 method:

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2h}{3}f(x_{n+1}, y_{n+1}), \quad n \geq 1.$$

Remark 2.1. *The BDF methods described above are implicit linear multistep methods that are suitable for stiff problems.*

2.2 Adams methods

These methods are derived by doing quadrature in

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t))\, dt.$$

Basic idea:

- In the case of Adams–Bashforth (AB): Replace $y'(t) = f(t, y(t))$ by the interpolating polynomial $p(t)$ that interpolates $y'(t)$ at

  $$x_{n-k+1}, \ldots, x_{n-1}, x_n, \quad \text{(for some } k \geq 1),$$

  and then use the approximation

  $$y(x_{n+1}) \approx y(x_n) + \int_{x_n}^{x_{n+1}} p(t)\, dt,$$

  This leads to formulas of the form

  $$y_{n+1} = y_n + h \sum_{j=0}^{k-1} \beta_j f(x_{n-j}, y_{n-j}).$$

- In the case of Adams–Moulton (AM), we proceed similarly to Adams–Bashforth, but interpolate $y'(t) = f(t, y(t))$ at the $k+1$ points,

  $$x_{n-k+1}, \ldots, x_{n-1}, x_n, x_{n+1}, \quad \text{(for some } k \geq 0).$$

  This leads to (implicit) formulas for the form:

  $$y_{n+1} = y_n + h \sum_{j=-1}^{k-1} \beta_j f(x_{n-j}, y_{n-j}).$$

Remark 2.2. *We record the following remarks regarding AB/AM methods.*

- *The naming convention for AB and AM methods is ABq and AMq, where q is the order of the method; e.g., a third order Adams–Bashforth method is referred to as AB3.*
- *AB methods are explicit, AM methods are implicit.*
- *p-step AB has order p.*
- *p-step AM has order $p+1$ (an exception is AM1, the backward Euler method, which is a one-step of order 1).*
- *The AM formulas have a considerably smaller truncation error than the AB formulas with comparable order; see [1, p. 389].*

Examples. It is straightforward to check (left as exercises) that AB1 which is forward Euler, AM1 is backward Euler, and AM2 is the trapezoidal rule. As an illustration, we derive the AB2 method. To this end, we interpolate $y'(x)$ at nodes x_{n-1} and x_n:

$$p_1(x) = y'(x_{n-1}) + \frac{y'(x_n) - y'(x_{n-1})}{h}(x - x_{n-1}).$$

Thus, we have

$$\begin{aligned} y(x_{n+1}) \approx y(x_n) + \int_{x_n}^{x_{n+1}} p_1(t)\, dt &= y(x_n) + h\left[\frac{3}{2}y'(x_n) - \frac{1}{2}y'(x_{n-1})\right] \\ &= y(x_n) + h\left[\frac{3}{2}f(x_n, y(x_n)) - \frac{1}{2}f(x_{n-1}, y(x_{n-1}))\right], \end{aligned}$$

from which we obtain the numerical scheme (AB2):

$$y_{n+1} = y_n + h\left[\frac{3}{2}f(x_n, y_n) - \frac{1}{2}f(x_{n-1}, y_{n-1})\right].$$

Remark 2.3. *Note that the derivation of the above methods also shows how to derive the respective truncation errors*[1] *, all going back to the truncation error for the interpolant used. Also, for a specific method you can derive the truncation error using the general strategy for linear multistep methods discussed earlier.*

[1] The expressions for these exist, and can be found in standard references.

2.3 Predictor corrector formulas

Adams methods are typically applied in pairs as follows: take an ABq (or AB(q-1)) step and use the result as the initial iterate for AMq, which is an implicit method. Then, take one Picard iteration for AMq. In this setting, the AB method acts as the predictor and the AM method the corrector. For example, AM2 with the AB1 as predictor is just the trapezoidal method with the forward Euler predictor. A less trivial example is the following method with AB3-AM4:

$$\hat{y}_{n+1} = y_n + \frac{h}{12}\left[23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + 5f(x_{n-2}, y_{n-2})\right] \quad \text{(AB3)}$$

$$y_{n+1} = y_n + \frac{h}{24}\left[9f(x_{n+1}, \hat{y}_{n+1}) + 19f(x_n, y_n) - 5f(x_{n-1}, y_{n-1}) + f(x_{n-2}, y_{n-2})\right] \quad \text{(AM4)}$$

The procedure just described is known as the PECE method (Predict-Evaluate-Correct-Evaluate). The procedure, for a pair of Adams methods—ABq and AMq—is summarized in Algorithm 1.

Algorithm 1 Computations at step n of a PECE method using an ABq–AMq pair. We use the notation $f_i = f(x_i, y_i)$.

$\hat{y}_{n+1} = y_n + h\sum_{j=0}^{q-1} \beta_j f_{n-j}$	{**P**redict}
$\hat{f}_{n+1} = f(x_{n+1}, \hat{y}_{n+1})$	{**E**valuate}
$y_{n+1} = y_n + h\left[\beta_{-1}\hat{f}_{n+1} + \sum_{j=0}^{q-2} \beta_j f_{n-j}\right]$	{**C**correct}
$f_{n+1} = f(x_{n+1}, y_{n+1})$	{**E**valuate}

Remark 2.4. *We record the following remarks:*

- *PECE methods make sense if the corrector is more accurate than the predictor, not necessarily in terms of order, but at least in terms of the size of the error coefficients. The benefit of using a PECE method is improved accuracy.*
- *Note that two evaluations of f are needed at each step.*
- *The second evaluation in the above algorithm is done in preparation for the next iteration.*

To illustrate further, we provide a basic Matlab implementation of the PECE method based on using AB3-AM4 in Appendix A.

3 Stiff problems, A-stability, and A(α) stability

When discussing A-stability, we consider the model IVP

$$y' = \lambda y, \quad x \geq 0, \quad Re\lambda < 0.$$

Definition 3.1. *A $p+1$ step linear multistep method is called A-stable if, when applied to the above model IVP, it produces a grid function $\{y_n\}_{n\geq 0}$ satisfying $y_n \to 0$, regardless of the choice of starting values $y_0, y_1, \ldots, y_p$.*

The following result, due to Dahlquist (see e.g., [5, p. 247]) shows that multistep methods have severe limitations as with regards to A-stability.

Theorem 3.2. *An A-stable multistep method must be of order $p \leq 2$.*

Identifying the stability regions for general linear multistep methods can become complicated in general but can be facilitated via computers; see [6, Chapter 7]. Let us consider the BDFk methods. It is known that BDF1 and BDF2 are A-stable (BDF1 is just implicit Euler). In Figure 1 (left), we show the stability regions for BDFk, $k = 1, \ldots, 6$. The stability regions in these plots are outside of the shaded regions. As can be seen, the stability regions of BDFk include the entire left half of the complex plane only for $k = 1$ and $k = 2$. However, we note that for $k = 3, \ldots, 6$, the stability regions includes all but a small part of the left half of the complex plane. This motivates the definition of weaker variants of the notion of A-stability. Specifically, we consider the so called A(α)-stable methods:

Definition 3.3. *A method is A(α)-stable, $\alpha \in (0, \pi/2)$, if its stability region contains the wedge-like region $W_\alpha = \{z \in \mathbb{C} : |\arg(-z)| < \alpha, z \neq 0\}$.*

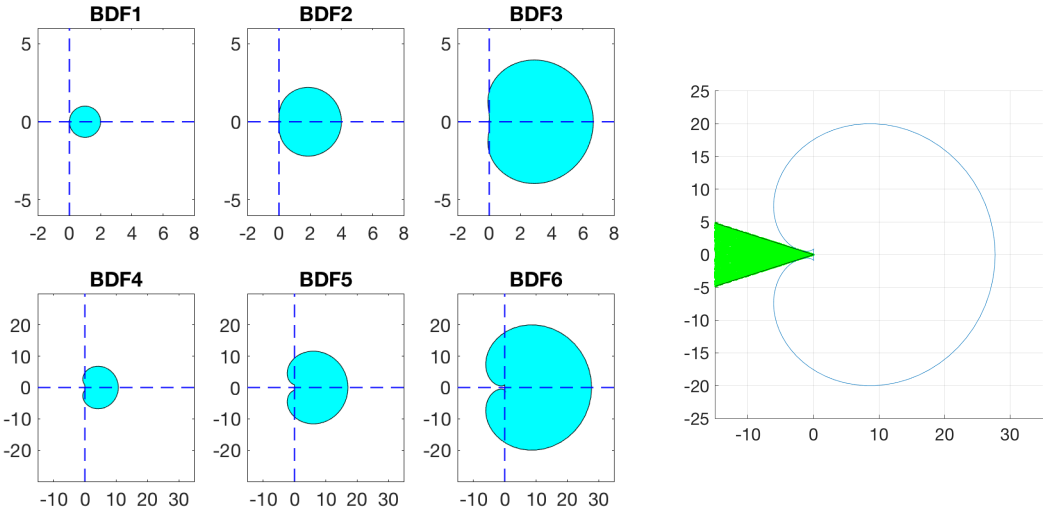

Figure 1: Left: stability regions for BDF1–BDF6. The regions of absolute stability are the exterior of the shaded regions. Right: stability region for BDF6, along with (a part of) the corresponding W_α region.

While BDF1 and BDF2 methods are A-stable, BDFk methods, with $k = 3, \ldots, 6$ are A(α)-stable, with $\alpha = 86.03°, 73.35°, 51.84°, 17.84°$, respectively. we illustrate the stability region for BDF6 and the corresponding W_α in Figure 1 (right).

References

[1] Kendall E. Atkinson. *An introduction to numerical analysis*. John Wiley & Sons Inc., New York, second edition, 1989.

[2] Gautschi, Walter. Numerical analysis. Springer Science & Business Media, 2011.

[3] Isaacson, Eugene, and Herbert Bishop Keller. Analysis of numerical methods. Courier Corporation, 1994.

[4] Hairer, Ernst and Nørsett, Syvert P and Wanner, Gerhard. Solving ordinary differential equations. I, volume 8 of Springer Series in Computational Mathematics. Springer, 1993.

[5] Hairer, E., and Wanner, G. Solving ordinary differential equations II. Stiff and differential-algebraic problems. Second revised edition, Springer Series in Computational Mathematics, 14. Springer-Verlag, Berlin, 2010.

[6] LeVeque, Randall J. Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems. SIAM, 2007.

A Matlab implementation of AB3-AM4 PECE method

```
function [t y] = ab3am4(f, a, b, y0, n)
% implementation of a 4th order Adams-Bashforth-Moulton PECE method for
%
%   y' = f(t, y),    y(0) = y0
%
% we use the 3-step Adams-Bashforth (AB3) method as a precdictor, and
% 3-step Adams-Moulton (AM4) method as a corrector;
% an RK3 method is used to initialize the starting values
%
% Input:
%     f:    the right hand side function f(t,y)
%     a,b:  end points of the integration time interval
%     y0:   initial condition
%     n:    number of sub-intervals in time grid
% Output:
%    t: vector of time steps
%    y: computed numerical solution

%
% initialization/memory pre-allocation
%
h = (b - a) / n;
p = 2;   % assuming n > 3
d = length(y0);      % dimension of state vector
t = zeros(1, n+1);
y = zeros(d, n+1);
F = zeros(d, n+1);

y(:,1) = y0;    % initial state
t(1) = a;

% compute y_1, y_2 using RK3
F(:, 1) = f(t(1), y(:, 1));    % initial f evaluation
for i = 1 : p
    t(i+1) = t(i) + h;
    k_1 = F(:, i);
    k_2 = f(t(i) + 0.5*h, y(:,i) + 0.5*h*k_1);
    k_3 = f(t(i) + h, y(:,i) + h*(-k_1 + 2*k_2));
    y(:,i+1) = y(:,i) + h * (1/6)*(k_1 + 4*k_2 + k_3);    % advance in time

    F(:,i+1) = f(t(i+1), y(:,i+1));    % keeping track of f evaluations for efficiency
end

%
% Main loop of PECE algorithm
%
for i = p+1 : n
    t(i+1) = t(i) + h;

    % Predict (via AB3 predictor)
    yhat = y(:,i) + (h/12) * (23 * F(:,i) - 16 * F(:,i-1) + 5 * F(:,i-2));

    % Evaluate
    fhat = f(t(i+1), yhat);

    % Correct (via AM4 corrector)
    y(:,i+1) = y(:,i) + (h/24) * (9 * fhat + 19 * F(:,i) - 5 * F(:,i-1) + F(:,i-2));

    % Evaluate
    F(:,i+1) = f(t(i+1), y(:,i+1));
end
```