Numerical Simulation of Dynamic Systems VI

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

March 12, 2013

Introduction

Until now, we have discussed only *single-step numerical integration algorithms*. The entire information that the algorithm required was obtained locally.

Introduction

Until now, we have discussed only *single-step numerical integration algorithms*. The entire information that the algorithm required was obtained locally.

The price that we paid for this choice was that higher-order single-step methods require multiple function evaluations during each integration step. We were talking about the different stages of the algorithm.

Introduction

Until now, we have discussed only *single-step numerical integration algorithms*. The entire information that the algorithm required was obtained locally.

The price that we paid for this choice was that higher-order single-step methods require multiple function evaluations during each integration step. We were talking about the different stages of the algorithm.

Maybe this way of solving the problem is inefficient. At the end of each step, we have available a lot of valuable information that could be used during the next step. Until now, we simply threw that information away and started from scratch.

Introduction

Until now, we have discussed only *single-step numerical integration algorithms*. The entire information that the algorithm required was obtained locally.

The price that we paid for this choice was that higher-order single-step methods require multiple function evaluations during each integration step. We were talking about the different stages of the algorithm.

Maybe this way of solving the problem is inefficient. At the end of each step, we have available a lot of valuable information that could be used during the next step. Until now, we simply threw that information away and started from scratch.

There exist other classes of higher-order numerical ODE solvers that preserve some of the information gathered during previous steps. As a consequence, they manage to get away with a single function evaluation in each step. These algorithms are called *linear multi-step integration algorithms*.

The Newton-Gregory Polynomials

For the analysis of *multi-step methods*, we require a set of mathematical tools that we shall now introduce.

The Newton-Gregory Polynomials

For the analysis of *multi-step methods*, we require a set of mathematical tools that we shall now introduce.

Given a function $f(t)$ equidistantly sampled over time, t_0, $t_1 > t_0$, $t_2 > t_1$, $\ldots$. The function assumes the values f_0, f_1, f_2, $\ldots$ at the sampling points.

The Newton-Gregory Polynomials

For the analysis of *multi-step methods*, we require a set of mathematical tools that we shall now introduce.

Given a function $f(t)$ equidistantly sampled over time, t_0, $t_1 > t_0$, $t_2 > t_1$, $\ldots$. The function assumes the values f_0, f_1, f_2, $\ldots$ at the sampling points.

We introduce the *forward difference operator*, Δ:

$$\begin{aligned}
\Delta f_0 &= f_1 - f_0 \\
\Delta^2 f_0 &= \Delta(\Delta f_0) = \Delta(f_1 - f_0) = \Delta f_1 - \Delta f_0 = f_2 - 2f_1 + f_0 \\
\Delta^3 f_0 &= \Delta(\Delta^2 f_0) = f_3 - 3f_2 + 3f_1 - f_0 \\
&\text{etc.}
\end{aligned}$$

The Newton-Gregory Polynomials

For the analysis of *multi-step methods*, we require a set of mathematical tools that we shall now introduce.

Given a function $f(t)$ equidistantly sampled over time, $t_0, t_1 > t_0, t_2 > t_1, \ldots$. The function assumes the values $f_0, f_1, f_2, \ldots$ at the sampling points.

We introduce the *forward difference operator*, Δ:

$$\begin{aligned}
\Delta f_0 &= f_1 - f_0 \\
\Delta^2 f_0 &= \Delta(\Delta f_0) = \Delta(f_1 - f_0) = \Delta f_1 - \Delta f_0 = f_2 - 2f_1 + f_0 \\
\Delta^3 f_0 &= \Delta(\Delta^2 f_0) = f_3 - 3f_2 + 3f_1 - f_0 \\
&\text{etc.}
\end{aligned}$$

In general:

$$\begin{aligned}
\Delta^n f_i &= f_{i+n} - n \cdot f_{i+n-1} + \frac{n(n-1)}{2!} \cdot f_{i+n-2} - \frac{n(n-1)(n-2)}{3!} \cdot f_{i+n-3} + \ldots \\
&= \binom{n}{0} f_{i+n} - \binom{n}{1} f_{i+n-1} + \binom{n}{2} f_{i+n-2} - \binom{n}{3} f_{i+n-3} + \cdots \pm \binom{n}{n} f_i
\end{aligned}$$

The Newton-Gregory Polynomials II

As we assumed *equidistant sampling*, we can write: $t_1 = t_0 + h$, $t_2 = t_0 + 2h$, $\ldots$, $t_n = t_0 + n \cdot h$.

The Newton-Gregory Polynomials II

As we assumed *equidistant sampling*, we can write: $t_1 = t_0 + h$, $t_2 = t_0 + 2h$, $\dots$, $t_n = t_0 + n \cdot h$.

We now introduce a *normalized time variable*, s:

$$s = \frac{t - t_0}{h}$$

Consequently:

$$t = t_0 \;\Leftrightarrow\; s = 0.0, t = t_1 \;\Leftrightarrow\; s = 1.0, \dots$$

The Newton-Gregory Polynomials II

As we assumed *equidistant sampling*, we can write: $t_1 = t_0 + h$, $t_2 = t_0 + 2h$, $\dots$, $t_n = t_0 + n \cdot h$.

We now introduce a *normalized time variable*, s:

$$s = \frac{t - t_0}{h}$$

Consequently:

$$t = t_0 \;\; \Leftrightarrow \;\; s = 0.0, t = t_1 \;\; \Leftrightarrow \;\; s = 1.0, \dots$$

It is possible to define an *interpolation polynomial* of order n that passes through the $n+1$ points f_0, f_1, $\dots$, f_n:

$$f(s) \approx \binom{s}{0} f_0 + \binom{s}{1} \Delta f_0 + \binom{s}{2} \Delta^2 f_0 + \dots + \binom{s}{n} \Delta^n f_0$$

The Newton-Gregory Polynomials II

As we assumed *equidistant sampling*, we can write: $t_1 = t_0 + h$, $t_2 = t_0 + 2h$, $\ldots$, $t_n = t_0 + n \cdot h$.

We now introduce a *normalized time variable*, s:

$$s = \frac{t - t_0}{h}$$

Consequently:

$$t = t_0 \;\Leftrightarrow\; s = 0.0, t = t_1 \;\Leftrightarrow\; s = 1.0, \ldots$$

It is possible to define an *interpolation polynomial* of order n that passes through the $n+1$ points f_0, f_1, $\ldots$, f_n:

$$f(s) \approx \binom{s}{0} f_0 + \binom{s}{1} \Delta f_0 + \binom{s}{2} \Delta^2 f_0 + \cdots + \binom{s}{n} \Delta^n f_0$$

This polynomial is called *forward Newton-Gregory interpolation polynomial*. It is trivial to show that this polynomial of order n passes through the $n+1$ points f_0, f_1, $\ldots$, f_n.

The Newton-Gregory Polynomials III

It is important to mention that the variable s is allowed to assume also *non-integer values*. For example:

$$\binom{s}{3}_{s=1.5} \equiv \left[\frac{s(s-1)(s-2)}{3!}\right]_{s=1.5} = -\frac{1}{16}$$

The Newton-Gregory Polynomials III

It is important to mention that the variable s is allowed to assume also *non-integer values*. For example:

$$\binom{s}{3}_{s=1.5} \equiv \left[\frac{s(s-1)(s-2)}{3!}\right]_{s=1.5} = -\frac{1}{16}$$

Sometimes it is more useful to work with a different interpolation polynomial:

$$f(s) \approx f_0 + \binom{s}{1}\Delta f_{-1} + \binom{s+1}{2}\Delta^2 f_{-2} + \binom{s+2}{3}\Delta^3 f_{-3} + \cdots + \binom{s+n-1}{n}\Delta^n f_{-n}$$

The Newton-Gregory Polynomials III

It is important to mention that the variable s is allowed to assume also *non-integer values*. For example:

$$\binom{s}{3}_{s=1.5} \equiv \left[\frac{s(s-1)(s-2)}{3!}\right]_{s=1.5} = -\frac{1}{16}$$

Sometimes it is more useful to work with a different interpolation polynomial:

$$f(s) \approx f_0 + \binom{s}{1}\Delta f_{-1} + \binom{s+1}{2}\Delta^2 f_{-2} + \binom{s+2}{3}\Delta^3 f_{-3} + \cdots + \binom{s+n-1}{n}\Delta^n f_{-n}$$

This polynomial is called *backward Newton-Gregory interpolation polynomial*. It is equally easy to demonstrate that this polynomial of order n passes through the $n+1$ points f_0, f_{-1}, ..., f_{-n}.

The Newton-Gregory Polynomials IV

We introduce now a second operator, the *backward difference operator*, ∇:

$$\nabla f_i = f_i - f_{i-1}$$
$$\nabla^2 f_i = \nabla(\nabla f_i) = \nabla(f_i - f_{i-1}) = \nabla f_i - \nabla f_{i-1} = f_i - 2\ f_{i-1} + f_{i-2}$$
$$\nabla^3 f_i = \nabla(\nabla^2 f_i) = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}$$

etc.

The Newton-Gregory Polynomials IV

We introduce now a second operator, the *backward difference operator*, ∇:

$$
\begin{aligned}
\nabla f_i &= f_i - f_{i-1} \\
\nabla^2 f_i &= \nabla(\nabla f_i) = \nabla(f_i - f_{i-1}) = \nabla f_i - \nabla f_{i-1} = f_i - 2\,f_{i-1} + f_{i-2} \\
\nabla^3 f_i &= \nabla(\nabla^2 f_i) = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3} \\
&\text{etc.}
\end{aligned}
$$

In general:

$$\nabla^n f_i = \binom{n}{0} f_i - \binom{n}{1} f_{i-1} + \binom{n}{2} f_{i-2} - \binom{n}{3} f_{i-3} + \cdots \pm \binom{n}{n} f_{i-n}$$

The Newton-Gregory Polynomials IV

We introduce now a second operator, the *backward difference operator*, ∇:

$$
\begin{aligned}
\nabla f_i &= f_i - f_{i-1} \\
\nabla^2 f_i &= \nabla(\nabla f_i) = \nabla(f_i - f_{i-1}) = \nabla f_i - \nabla f_{i-1} = f_i - 2\, f_{i-1} + f_{i-2} \\
\nabla^3 f_i &= \nabla(\nabla^2 f_i) = f_i - 3 f_{i-1} + 3 f_{i-2} - f_{i-3} \\
&\text{etc.}
\end{aligned}
$$

In general:

$$\nabla^n f_i = \binom{n}{0} f_i - \binom{n}{1} f_{i-1} + \binom{n}{2} f_{i-2} - \binom{n}{3} f_{i-3} + \cdots \pm \binom{n}{n} f_{i-n}$$

The *backward Newton-Gregory interpolation polynomial* can also be written in terms of the ∇ operator:

$$f(s) \approx f_0 + \binom{s}{1} \nabla f_0 + \binom{s+1}{2} \nabla^2 f_0 + \binom{s+2}{3} \nabla^3 f_0 + \cdots + \binom{s+n-1}{n} \nabla^n f_0$$

The Newton-Gregory Polynomials V

Another operator is also sometimes useful, namely the *displacement operator*, $\mathcal{E}$:

$$\begin{aligned}
\mathcal{E} f_i &= f_{i+1} \\
\mathcal{E}^2 f_i &= \mathcal{E}(\mathcal{E} f_i) = \mathcal{E}(f_{i+1}) = f_{i+2} \\
\mathcal{E}^3 f_i &= \mathcal{E}(\mathcal{E}^2 f_i) = \mathcal{E}(f_{i+2}) = f_{i+3} \\
&\text{etc.}
\end{aligned}$$

The Newton-Gregory Polynomials V

Another operator is also sometimes useful, namely the *displacement operator*, $\mathcal{E}$:

$$\begin{aligned} \mathcal{E}f_i &= f_{i+1} \\ \mathcal{E}^2 f_i &= \mathcal{E}(\mathcal{E}f_i) = \mathcal{E}(f_{i+1}) = f_{i+2} \\ \mathcal{E}^3 f_i &= \mathcal{E}(\mathcal{E}^2 f_i) = \mathcal{E}(f_{i+2}) = f_{i+3} \\ &\text{etc.} \end{aligned}$$

Evidently:

$$\begin{aligned} \Delta f_i &= \mathcal{E}f_i - f_i = (\mathcal{E} - 1)f_i \\ \nabla f_i &= f_i - \mathcal{E}^{-1}f_i = (1 - \mathcal{E}^{-1})f_i \\ \mathcal{E}(\nabla f_i) &= \mathcal{E}(f_i - f_{i-1}) = f_{i+1} - f_i = \Delta f_i \end{aligned}$$

The Newton-Gregory Polynomials V

Another operator is also sometimes useful, namely the *displacement operator*, $\mathcal{E}$:

$$\begin{aligned}
\mathcal{E} f_i &= f_{i+1} \\
\mathcal{E}^2 f_i &= \mathcal{E}(\mathcal{E} f_i) = \mathcal{E}(f_{i+1}) = f_{i+2} \\
\mathcal{E}^3 f_i &= \mathcal{E}(\mathcal{E}^2 f_i) = \mathcal{E}(f_{i+2}) = f_{i+3} \\
&\text{etc.}
\end{aligned}$$

Evidently:

$$\begin{aligned}
\Delta f_i &= \mathcal{E} f_i - f_i = (\mathcal{E} - 1) f_i \\
\nabla f_i &= f_i - \mathcal{E}^{-1} f_i = (1 - \mathcal{E}^{-1}) f_i \\
\mathcal{E}(\nabla f_i) &= \mathcal{E}(f_i - f_{i-1}) = f_{i+1} - f_i = \Delta f_i
\end{aligned}$$

By abstraction (a bit dangerous!):

$$\begin{aligned}
\Delta &= \mathcal{E} - 1 \\
\nabla &= 1 - \mathcal{E}^{-1} \\
\Delta &= \mathcal{E} \nabla
\end{aligned}$$

The Newton-Gregory Polynomials VI

The three operators Δ, ∇, and $\mathcal{E}$ are *linear operators*. Hence they can be used in algebraic expressions.

In particular:

$$\Delta^n = (\mathcal{E} - 1)^n = \mathcal{E}^n - n\mathcal{E}^{n-1} + \binom{n}{2}\mathcal{E}^{n-2} - + \cdots \pm \binom{n}{n-1}\mathcal{E} \mp 1$$

The Newton-Gregory Polynomials VI

The three operators Δ, ∇, and $\mathcal{E}$ are *linear operators*. Hence they can be used in algebraic expressions.

In particular:

$$\Delta^n = (\mathcal{E} - 1)^n = \mathcal{E}^n - n\mathcal{E}^{n-1} + \binom{n}{2}\mathcal{E}^{n-2} - + \cdots \pm \binom{n}{n-1}\mathcal{E} \mp 1$$

Making use of operator calculus, the derivation of the Newton-Gregory polynomials is much simplified:

$$f(s) \approx \mathcal{E}^s f_0 = (1 + \Delta)^s f_0 = \left[1 + \binom{s}{1}\Delta + \binom{s}{2}\Delta^2 + \binom{s}{3}\Delta^3 + \dots\right] f_0$$

and:

$$f(s) \approx (1 - \nabla)^{-s} f_0 = \left[1 + \binom{s}{1}\nabla + \binom{s+1}{2}\nabla^2 + \binom{s+2}{3}\nabla^3 + \dots\right] f_0$$

The Newton-Gregory Polynomials VII

Also *differentiation* is a *linear operation*. Therefore:

$$\begin{aligned}\dot{f}(t) &= \frac{d}{dt}f(t) = \frac{\partial}{\partial s}f(s) \cdot \frac{ds}{dt} \\ &\approx \frac{1}{h} \cdot \frac{\partial}{\partial s}\left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \ldots \right)\end{aligned}$$

In particular:

$$\dot{f}(t_0) \approx \frac{1}{h} \cdot \left(\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0 - \cdots \pm \frac{1}{n}\Delta^n f_0 \right)$$

The Newton-Gregory Polynomials VII

Also *differentiation* is a *linear operation*. Therefore:

$$\begin{aligned}\dot{f}(t) &= \frac{d}{dt}f(t) = \frac{\partial}{\partial s}f(s)\cdot\frac{ds}{dt} \\ &\approx \frac{1}{h}\cdot\frac{\partial}{\partial s}\left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \dots\right)\end{aligned}$$

In particular:

$$\dot{f}(t_0) \approx \frac{1}{h}\cdot\left(\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0 - \dots \pm \frac{1}{n}\Delta^n f_0\right)$$

It makes sense to introduce yet another operator, the *differentiation operator*, $\mathcal{D}$:

$$\mathcal{D} = \frac{1}{h}\cdot\left(\Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 - \dots \pm \frac{1}{n}\Delta^n\right)$$

The Newton-Gregory Polynomials VII

Also *differentiation* is a *linear operation*. Therefore:

$$\begin{aligned}\dot{f}(t) &= \frac{d}{dt}f(t) = \frac{\partial}{\partial s}f(s)\cdot\frac{ds}{dt} \\ &\approx \frac{1}{h}\cdot\frac{\partial}{\partial s}\left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \dots\right)\end{aligned}$$

In particular:

$$\dot{f}(t_0) \approx \frac{1}{h}\cdot\left(\Delta f_0 - \frac{1}{2}\Delta^2 f_0 + \frac{1}{3}\Delta^3 f_0 - \dots \pm \frac{1}{n}\Delta^n f_0\right)$$

It makes sense to introduce yet another operator, the *differentiation operator*, $\mathcal{D}$:

$$\mathcal{D} = \frac{1}{h}\cdot\left(\Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 - \dots \pm \frac{1}{n}\Delta^n\right)$$

Thus, the second derivative can be obtained in the following manner:

$$\begin{aligned}\mathcal{D}^2 &= \frac{1}{h^2}\cdot\left(\Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 - \dots \pm \frac{1}{n}\Delta^n\right)^2 \\ &= \frac{1}{h^2}\cdot\left(\Delta^2 - \Delta^3 + \frac{11}{12}\Delta^4 - \frac{5}{6}\Delta^5 + \dots\right)\end{aligned}$$

Linear Multi-step Integration Methods

All families of numerical linear multi-step integration methods used for the simulation of dynamic systems can be elegantly derived by means of Newton-Gregory polynomials.

To this end, we either approximate the function itself by a Newton-Gregory polynomial and differentiate this polynomial with respect to time, or alternatively, we approximate the first time derivative by a Newton-Gregory polynomial and integrate this polynomial with respect to time.

The Explicit Adams-Bashforth Formulae

Let us formulate a *backward Newton-Gregory polynomial* of the first time derivative $\dot{\mathbf{x}}$ around the time instant t_k:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_k + \binom{s}{1}\nabla\mathbf{f}_k + \binom{s+1}{2}\nabla^2\mathbf{f}_k + \binom{s+2}{3}\nabla^3\mathbf{f}_k + \dots$$

with:

$$\mathbf{f}_k = \dot{\mathbf{x}}(t_k) = \mathbf{f}(\mathbf{x}(t_k), t_k)$$

The Explicit Adams-Bashforth Formulae

Let us formulate a *backward Newton-Gregory polynomial* of the first time derivative $\dot{\mathbf{x}}$ around the time instant t_k:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_k + \binom{s}{1}\nabla\mathbf{f}_k + \binom{s+1}{2}\nabla^2\mathbf{f}_k + \binom{s+2}{3}\nabla^3\mathbf{f}_k + \dots$$

with:

$$\mathbf{f}_k = \dot{\mathbf{x}}(t_k) = \mathbf{f}(\mathbf{x}(t_k), t_k)$$

We integrate this polynomial over the time interval $t \in [t_k, t_{k+1}]$:

$$\begin{aligned}
\int_{t_k}^{t_{k+1}} \dot{\mathbf{x}}(t)dt &= \mathbf{x}(t_{k+1}) - \mathbf{x}(t_k) \\
&= \int_{t_k}^{t_{k+1}} \left[\mathbf{f}_k + \binom{s}{1}\nabla\mathbf{f}_k + \binom{s+1}{2}\nabla^2\mathbf{f}_k + \binom{s+2}{3}\nabla^3\mathbf{f}_k + \dots\right] dt \\
&= \int_{0.0}^{1.0} \left[\mathbf{f}_k + \binom{s}{1}\nabla\mathbf{f}_k + \binom{s+1}{2}\nabla^2\mathbf{f}_k + \binom{s+2}{3}\nabla^3\mathbf{f}_k + \dots\right] \cdot \frac{dt}{ds} \cdot ds
\end{aligned}$$

The Explicit Adams-Bashforth Formulae II

Therefore:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h \int_0^1 \left[\mathbf{f}_k + s\nabla\mathbf{f}_k + \left(\frac{s^2}{2} + \frac{s}{2} \right) \nabla^2\mathbf{f}_k \right.$$
$$\left. + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3} \right) \nabla^3\mathbf{f}_k + \ldots \right] ds$$

and consequently:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h \left(\mathbf{f}_k + \frac{1}{2}\nabla\mathbf{f}_k + \frac{5}{12}\nabla^2\mathbf{f}_k + \frac{3}{8}\nabla^3\mathbf{f}_k + \ldots \right)$$

The Explicit Adams-Bashforth Formulae II

Therefore:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h \int_0^1 \left[\mathbf{f}_k + s\nabla\mathbf{f}_k + \left(\frac{s^2}{2} + \frac{s}{2} \right) \nabla^2\mathbf{f}_k \right.$$
$$\left. + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3} \right) \nabla^3\mathbf{f}_k + \dots \right] ds$$

and consequently:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h \left(\mathbf{f}_k + \frac{1}{2}\nabla\mathbf{f}_k + \frac{5}{12}\nabla^2\mathbf{f}_k + \frac{3}{8}\nabla^3\mathbf{f}_k + \dots \right)$$

If we truncate this infinite series after the quadratic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12} (23\mathbf{f}_k - 16\mathbf{f}_{k-1} + 5\mathbf{f}_{k-2})$$

which is the well-known *Adams-Bashforth third order algorithm (AB3)*.

The Explicit Adams-Bashforth Formulae II

Therefore:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h \int_0^1 \left[\mathbf{f}_k + s\nabla\mathbf{f}_k + \left(\frac{s^2}{2} + \frac{s}{2}\right)\nabla^2\mathbf{f}_k + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3}\right)\nabla^3\mathbf{f}_k + \dots \right] ds$$

and consequently:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h\left(\mathbf{f}_k + \frac{1}{2}\nabla\mathbf{f}_k + \frac{5}{12}\nabla^2\mathbf{f}_k + \frac{3}{8}\nabla^3\mathbf{f}_k + \dots\right)$$

If we truncate this infinite series after the quadratic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(23\mathbf{f}_k - 16\mathbf{f}_{k-1} + 5\mathbf{f}_{k-2}\right)$$

which is the well-known *Adams-Bashforth third order algorithm (AB3)*.

If we truncate the series only after the cubic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{24}\left(55\mathbf{f}_k - 59\mathbf{f}_{k-1} + 37\mathbf{f}_{k-2} - 9\mathbf{f}_{k-3}\right)$$

which is the *Adams-Bashforth fourth order algorithm (AB4)*.

The Explicit Adams-Bashforth Formulae III

- The *Adams-Bashforth algorithms* are *explicit ODE solvers*.

The Explicit Adams-Bashforth Formulae III

- The *Adams-Bashforth algorithms* are *explicit ODE solvers*.
- Every one of them uses a *single function evaluation* during each step.

The Explicit Adams-Bashforth Formulae III

- The *Adams-Bashforth algorithms* are *explicit ODE solvers*.
- Every one of them uses a *single function evaluation* during each step.
- The AB algorithms make use of *past values of time derivatives*. The AB algorithm of order n makes use of $n-1$ earlier time derivative values.

The Explicit Adams-Bashforth Formulae III

- The *Adams-Bashforth algorithms* are *explicit ODE solvers*.
- Every one of them uses a *single function evaluation* during each step.
- The AB algorithms make use of *past values of time derivatives*. The AB algorithm of order n makes use of $n-1$ earlier time derivative values.
- The AB algorithm of order n can only be used after $n-1$ previous steps. This means that, except for AB1, these algorithms are not self-starting.

The Explicit Adams-Bashforth Formulae III

- The *Adams-Bashforth algorithms* are *explicit ODE solvers*.
- Every one of them uses a *single function evaluation* during each step.
- The AB algorithms make use of *past values of time derivatives*. The AB algorithm of order n makes use of $n-1$ earlier time derivative values.
- The AB algorithm of order n can only be used after $n-1$ previous steps. This means that, except for AB1, these algorithms are not self-starting.
- The *step-size control* is complicated by the need to use information of the past. You may remember that the Newton-Gregory polynomials were developed on the basis of *equidistant sampling*.

The Explicit Adams-Bashforth Formulae III

- The *Adams-Bashforth algorithms* are *explicit ODE solvers*.
- Every one of them uses a *single function evaluation* during each step.
- The AB algorithms make use of *past values of time derivatives*. The AB algorithm of order n makes use of $n-1$ earlier time derivative values.
- The AB algorithm of order n can only be used after $n-1$ previous steps. This means that, except for AB1, these algorithms are not self-starting.
- The *step-size control* is complicated by the need to use information of the past. You may remember that the Newton-Gregory polynomials were developed on the basis of *equidistant sampling*.
- The AB formulae were derived under the *linearity assumption*. It is therefore not guaranteed that AB3 is a third-order accurate algorithm also when used in the simulation of non-linear systems.

The Explicit Adams-Bashforth Formulae IV

The AB algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivative values:

$$\alpha = \begin{pmatrix} 1 \\ 2 \\ 12 \\ 24 \\ 720 \\ 1440 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & -1 & 0 & 0 & 0 & 0 \\ 23 & -16 & 5 & 0 & 0 & 0 \\ 55 & -59 & 37 & -9 & 0 & 0 \\ 1901 & -2774 & 2616 & -1274 & 251 & 0 \\ 4277 & -7923 & 9982 & -7298 & 2877 & -475 \end{pmatrix}$$

Every row specifies the coefficients of one of these algorithms.

The Explicit Adams-Bashforth Formulae IV

The AB algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivative values:

$$\alpha = \begin{pmatrix} 1 \\ 2 \\ 12 \\ 24 \\ 720 \\ 1440 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & -1 & 0 & 0 & 0 & 0 \\ 23 & -16 & 5 & 0 & 0 & 0 \\ 55 & -59 & 37 & -9 & 0 & 0 \\ 1901 & -2774 & 2616 & -1274 & 251 & 0 \\ 4277 & -7923 & 9982 & -7298 & 2877 & -475 \end{pmatrix}$$

Every row specifies the coefficients of one of these algorithms.

The *algorithm AB1* is the algorithm:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{1}\,(1\mathbf{f}_k)$$

i.e., *AB1* $=$ *FE*.

The Stability Domain

We would like to draw the *stability domain of the AB3 algorithm*:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(23\mathbf{f}_k - 16\mathbf{f}_{k-1} + 5\mathbf{f}_{k-2}\right)$$

We apply this algorithm to our standard linear system:

$$\mathbf{x}(t_{k+1}) = \left[\mathbf{I}^{(\mathbf{n})} + \frac{23}{12}\mathbf{A}h\right] \cdot \mathbf{x}(t_k) - \frac{4}{3}\mathbf{A}h \cdot \mathbf{x}(t_{k-1}) + \frac{5}{12}\mathbf{A}h \cdot \mathbf{x}(t_{k-2})$$

The Stability Domain

We would like to draw the *stability domain of the AB3 algorithm*:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(23\mathbf{f}_k - 16\mathbf{f}_{k-1} + 5\mathbf{f}_{k-2}\right)$$

We apply this algorithm to our standard linear system:

$$\mathbf{x}(t_{k+1}) = \left[\mathbf{I}^{(\mathbf{n})} + \frac{23}{12}\mathbf{A}h\right] \cdot \mathbf{x}(t_k) - \frac{4}{3}\mathbf{A}h \cdot \mathbf{x}(t_{k-1}) + \frac{5}{12}\mathbf{A}h \cdot \mathbf{x}(t_{k-2})$$

By substitution:

$$\mathbf{z_1}(t_k) = \mathbf{x}(t_{k-2})$$
$$\mathbf{z_2}(t_k) = \mathbf{x}(t_{k-1})$$
$$\mathbf{z_3}(t_k) = \mathbf{x}(t_k)$$

Therefore:

$$\mathbf{z_1}(t_{k+1}) = \mathbf{z_2}(t_k)$$
$$\mathbf{z_2}(t_{k+1}) = \mathbf{z_3}(t_k)$$
$$\mathbf{z_3}(t_{k+1}) = \frac{5}{12}\mathbf{A}h \cdot \mathbf{z_1}(t_k) - \frac{4}{3}\mathbf{A}h \cdot \mathbf{z_2}(t_k) + \left[\mathbf{I}^{(\mathbf{n})} + \frac{23}{12}\mathbf{A}h\right] \cdot \mathbf{z_3}(t_k)$$

The Stability Domain II

Consequently, we can write:

$$\mathbf{z}(t_{k+1}) = \begin{pmatrix} \mathbf{O}^{(n)} & \mathbf{I}^{(n)} & \mathbf{O}^{(n)} \\ \mathbf{O}^{(n)} & \mathbf{O}^{(n)} & \mathbf{I}^{(n)} \\ \frac{5}{12}\mathbf{A}h & -\frac{4}{3}\mathbf{A}h & (\mathbf{I}^{(n)} + \frac{23}{12}\mathbf{A}h) \end{pmatrix} \cdot \mathbf{z}(t_k)$$

The Stability Domain II

Consequently, we can write:

$$\mathbf{z}(t_{k+1}) = \begin{pmatrix} \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} \\ \mathbf{O}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} \\ \frac{5}{12}\mathbf{A}h & -\frac{4}{3}\mathbf{A}h & (\mathbf{I}^{(\mathbf{n})} + \frac{23}{12}\mathbf{A}h) \end{pmatrix} \cdot \mathbf{z}(t_k)$$

i.e.:

$$\mathbf{z}(t_{k+1}) = \mathbf{F} \cdot \mathbf{z}(t_k)$$

with:

$$\mathbf{F} = \begin{pmatrix} \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} \\ \mathbf{O}^{(\mathbf{n})} & \mathbf{O}^{(\mathbf{n})} & \mathbf{I}^{(\mathbf{n})} \\ \frac{5}{12}\mathbf{A}h & -\frac{4}{3}\mathbf{A}h & (\mathbf{I}^{(\mathbf{n})} + \frac{23}{12}\mathbf{A}h) \end{pmatrix}$$

The Stability Domain II

Consequently, we can write:

$$\mathbf{z}(t_{k+1}) = \begin{pmatrix} \mathbf{O}^{(n)} & \mathbf{I}^{(n)} & \mathbf{O}^{(n)} \\ \mathbf{O}^{(n)} & \mathbf{O}^{(n)} & \mathbf{I}^{(n)} \\ \frac{5}{12}\mathbf{A}h & -\frac{4}{3}\mathbf{A}h & (\mathbf{I}^{(n)} + \frac{23}{12}\mathbf{A}h) \end{pmatrix} \cdot \mathbf{z}(t_k)$$

i.e.:

$$\mathbf{z}(t_{k+1}) = \mathbf{F} \cdot \mathbf{z}(t_k)$$

with:

$$\mathbf{F} = \begin{pmatrix} \mathbf{O}^{(n)} & \mathbf{I}^{(n)} & \mathbf{O}^{(n)} \\ \mathbf{O}^{(n)} & \mathbf{O}^{(n)} & \mathbf{I}^{(n)} \\ \frac{5}{12}\mathbf{A}h & -\frac{4}{3}\mathbf{A}h & (\mathbf{I}^{(n)} + \frac{23}{12}\mathbf{A}h) \end{pmatrix}$$

The F-matrix is three times larger than the A-matrix. Consequently, it contains three times as many eigenvalues.

Stability Domains of AB Algorithms

We are now ready to draw the *stability domains of the AB algorithms*.

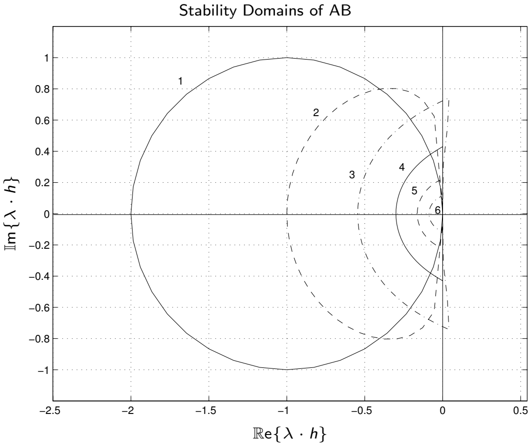


Figure: Stability domains of explicit AB algorithms

Stability Domains of AB Algorithms II

- Although the stability domains of the higher-order AB algorithms approximate the imaginary axis better in the proximity of the origin, the size of the numerical stability domains shrinks with growing orders of approximation accuracy instead of growing.

Stability Domains of AB Algorithms II

- Although the stability domains of the higher-order AB algorithms approximate the imaginary axis better in the proximity of the origin, the size of the numerical stability domains shrinks with growing orders of approximation accuracy instead of growing.
- There exist no stable AB algorithms of orders larger than six.

Stability Domains of AB Algorithms II

- Although the stability domains of the higher-order AB algorithms approximate the imaginary axis better in the proximity of the origin, the size of the numerical stability domains shrinks with growing orders of approximation accuracy instead of growing.
- There exist no stable AB algorithms of orders larger than six.
- We would like to use higher-order algorithms to improve the accuracy of the simulations, while still using larger integration step sizes. In reality, we have to reduce the step sizes due to problems with numerical stability.

Stability Domains of AB Algorithms II

- Although the stability domains of the higher-order AB algorithms approximate the imaginary axis better in the proximity of the origin, the size of the numerical stability domains shrinks with growing orders of approximation accuracy instead of growing.
- There exist no stable AB algorithms of orders larger than six.
- We would like to use higher-order algorithms to improve the accuracy of the simulations, while still using larger integration step sizes. In reality, we have to reduce the step sizes due to problems with numerical stability.
- Although the computational load associated with a single integration step is much lower for AB algorithms than for RK algorithms, we are forced to employ much smaller step sizes due to the reduced domains of numerical stability of these algorithms. For this reason, it is not at all clear that the AB algorithms in the end are more economical in their use than the RK algorithms.

Stability Domains of AB Algorithms III

What happened?

Stability Domains of AB Algorithms III

What happened?

We were looking for *interpolation polynomials* of higher orders passing through an extended number of equidistantly spaced points. Subsequently, we used these polynomials for an *extrapolation*.

Stability Domains of AB Algorithms III

What happened?

We were looking for *interpolation polynomials* of higher orders passing through an extended number of equidistantly spaced points. Subsequently, we used these polynomials for an *extrapolation*.

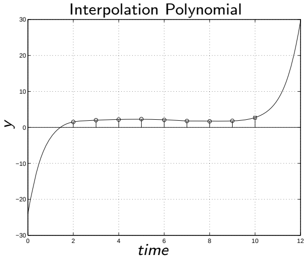


Figure: Interpolation polynomial used for extrapolation

Stability Domains of AB Algorithms III

What happened?

We were looking for *interpolation polynomials* of higher orders passing through an extended number of equidistantly spaced points. Subsequently, we used these polynomials for an *extrapolation*.

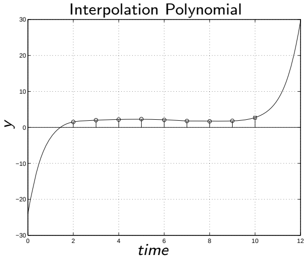


Figure: Interpolation polynomial used for extrapolation

This doesn't work very well.

The Implicit Adams-Moulton Formulae

If we use *implicit methods* instead of explicit algorithms, we are able to *interpolate* instead of extrapolating. This may help.

The Implicit Adams-Moulton Formulae

If we use *implicit methods* instead of explicit algorithms, we are able to *interpolate* instead of extrapolating. This may help.

We now formulate a *backward Newton-Gregory interpolation polynomial* of the first time derivative $\dot{\mathbf{x}}$ around the time instant t_{k+1}:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{k+1} + \binom{s}{1}\nabla\mathbf{f}_{k+1} + \binom{s+1}{2}\nabla^2\mathbf{f}_{k+1} + \binom{s+2}{3}\nabla^3\mathbf{f}_{k+1} + \dots$$

The Implicit Adams-Moulton Formulae

If we use *implicit methods* instead of explicit algorithms, we are able to *interpolate* instead of extrapolating. This may help.

We now formulate a *backward Newton-Gregory interpolation polynomial* of the first time derivative $\dot{\mathbf{x}}$ around the time instant t_{k+1}:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{k+1} + \binom{s}{1} \nabla \mathbf{f}_{k+1} + \binom{s+1}{2} \nabla^2 \mathbf{f}_{k+1} + \binom{s+2}{3} \nabla^3 \mathbf{f}_{k+1} + \dots$$

We integrate this polynomial over the time interval $t \in [t_k, t_{k+1}]$. This time around, this corresponds to the interval $s \in [-1.0, 0.0]$.

The Implicit Adams-Moulton Formulae

If we use *implicit methods* instead of explicit algorithms, we are able to *interpolate* instead of extrapolating. This may help.

We now formulate a *backward Newton-Gregory interpolation polynomial* of the first time derivative $\dot{\mathbf{x}}$ around the time instant t_{k+1}:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{k+1} + \binom{s}{1}\nabla\mathbf{f}_{k+1} + \binom{s+1}{2}\nabla^2\mathbf{f}_{k+1} + \binom{s+2}{3}\nabla^3\mathbf{f}_{k+1} + \ldots$$

We integrate this polynomial over the time interval $t \in [t_k, t_{k+1}]$. This time around, this corresponds to the interval $s \in [-1.0, 0.0]$.

There results the family of formulae:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + h\left(\mathbf{f}_{k+1} - \frac{1}{2}\nabla\mathbf{f}_{k+1} - \frac{1}{12}\nabla^2\mathbf{f}_{k+1} - \frac{1}{24}\nabla^3\mathbf{f}_{k+1} + \ldots\right)$$

The Implicit Adams-Moulton Formulae II

If we truncate this infinite time series after the quadratic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(5\mathbf{f}_{k+1} + 8\mathbf{f}_k - \mathbf{f}_{k-1}\right)$$

which is the well-known *third-order accurate Adams-Moulton algorithm (AM3)*.

The Implicit Adams-Moulton Formulae II

If we truncate this infinite time series after the quadratic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(5\mathbf{f}_{k+1} + 8\mathbf{f}_k - \mathbf{f}_{k-1}\right)$$

which is the well-known *third-order accurate Adams-Moulton algorithm (AM3)*.

If we truncate the series only after the cubic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{24}\left(9\mathbf{f}_{k+1} + 19\mathbf{f}_k - 5\mathbf{f}_{k-1} + \mathbf{f}_{k-2}\right)$$

i.e., the *fourth-order accurate Adams-Moulton algorithm (AM4)*.

The Implicit Adams-Moulton Formulae II

If we truncate this infinite time series after the quadratic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(5\mathbf{f}_{k+1} + 8\mathbf{f}_k - \mathbf{f}_{k-1}\right)$$

which is the well-known *third-order accurate Adams-Moulton algorithm (AM3)*.

If we truncate the series only after the cubic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{24}\left(9\mathbf{f}_{k+1} + 19\mathbf{f}_k - 5\mathbf{f}_{k-1} + \mathbf{f}_{k-2}\right)$$

i.e., the *fourth-order accurate Adams-Moulton algorithm (AM4)*.

The *algorithm AM1* is already known under the name *BE algorithm*.

The Implicit Adams-Moulton Formulae II

If we truncate this infinite time series after the quadratic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{12}\left(5\mathbf{f}_{k+1} + 8\mathbf{f}_k - \mathbf{f}_{k-1}\right)$$

which is the well-known *third-order accurate Adams-Moulton algorithm (AM3)*.

If we truncate the series only after the cubic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \frac{h}{24}\left(9\mathbf{f}_{k+1} + 19\mathbf{f}_k - 5\mathbf{f}_{k-1} + \mathbf{f}_{k-2}\right)$$

i.e., the *fourth-order accurate Adams-Moulton algorithm (AM4)*.

The *algorithm AM1* is already known under the name *BE algorithm*.

All AM algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivative values:

$$\alpha = \begin{pmatrix} 1 \\ 2 \\ 12 \\ 24 \\ 720 \\ 1440 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 5 & 8 & -1 & 0 & 0 & 0 \\ 9 & 19 & -5 & 1 & 0 & 0 \\ 251 & 646 & -264 & 106 & -19 & 0 \\ 475 & 1427 & -798 & 482 & -173 & 27 \end{pmatrix}$$

Stability Domains of AM Algorithms

We can draw the *stability domains of the AM algorithms*.

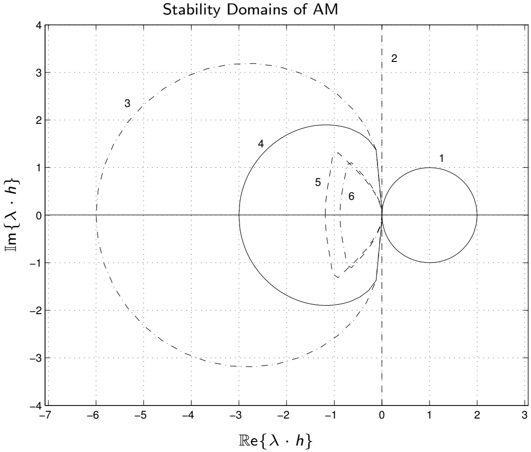


Figure: Stability domains of implicit AM algorithms

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.
- The algorithm **AM2=TR** is F-stable.

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.
- The algorithm **AM2=TR** is F-stable.
- The numerical stability domains of the higher-order AM algorithms loop in the left-half complex plane. These algorithms, in spite of being implicit, are not useful for the simulation of stiff systems.

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.
- The algorithm **AM2=TR** is F-stable.
- The numerical stability domains of the higher-order AM algorithms loop in the left-half complex plane. These algorithms, in spite of being implicit, are not useful for the simulation of stiff systems.
- We are still facing the same problem as before. The stability domains of the higher-order AM algorithms, although larger than those of the corresponding AB algorithms, shrink with growing orders instead of growing.

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.
- The algorithm **AM2=TR** is F-stable.
- The numerical stability domains of the higher-order AM algorithms loop in the left-half complex plane. These algorithms, in spite of being implicit, are not useful for the simulation of stiff systems.
- We are still facing the same problem as before. The stability domains of the higher-order AM algorithms, although larger than those of the corresponding AB algorithms, shrink with growing orders instead of growing.
- There exist no stable AM algorithms of orders larger than six.

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.
- The algorithm **AM2=TR** is F-stable.
- The numerical stability domains of the higher-order AM algorithms loop in the left-half complex plane. These algorithms, in spite of being implicit, are not useful for the simulation of stiff systems.
- We are still facing the same problem as before. The stability domains of the higher-order AM algorithms, although larger than those of the corresponding AB algorithms, shrink with growing orders instead of growing.
- There exist no stable AM algorithms of orders larger than six.
- On average, we need three Newton iterations per integration step. Consequently, we perform on average three function evaluations during each step.

Stability Domains of AM Algorithms II

- The algorithm **AM1=BE=BRK1** is L-stable.
- The algorithm **AM2=TR** is F-stable.
- The numerical stability domains of the higher-order AM algorithms loop in the left-half complex plane. These algorithms, in spite of being implicit, are not useful for the simulation of stiff systems.
- We are still facing the same problem as before. The stability domains of the higher-order AM algorithms, although larger than those of the corresponding AB algorithms, shrink with growing orders instead of growing.
- There exist no stable AM algorithms of orders larger than six.
- On average, we need three Newton iterations per integration step. Consequently, we perform on average three function evaluations during each step.
- Because of the larger stability domains of the AM algorithms, we can use step sizes that are on average three times larger than those used with the corresponding AB algorithms. The efficiencies of the AB and AM algorithms is therefore quite similar.

Adams-Bashforth-Moulton Predictor-Corrector Formulae

Sometimes a method is used that combines a predictor stage of AB with a subsequent corrector stage of AM, e.g.:

$$\text{predictor:} \quad \begin{aligned} \dot{\mathbf{x}}_{\mathbf{k}} &= \mathbf{f}(\mathbf{x}_{\mathbf{k}}, t_k) \\ \mathbf{x}^{\mathbf{P}}_{\mathbf{k+1}} &= \mathbf{x}_{\mathbf{k}} + \frac{h}{12}(23\dot{\mathbf{x}}_{\mathbf{k}} - 16\dot{\mathbf{x}}_{\mathbf{k-1}} + 5\dot{\mathbf{x}}_{\mathbf{k-2}}) \end{aligned}$$

$$\text{corrector:} \quad \begin{aligned} \dot{\mathbf{x}}^{\mathbf{P}}_{\mathbf{k+1}} &= \mathbf{f}(\mathbf{x}^{\mathbf{P}}_{\mathbf{k+1}}, t_{k+1}) \\ \mathbf{x}^{\mathbf{C}}_{\mathbf{k+1}} &= \mathbf{x}_{\mathbf{k}} + \frac{h}{12}(5\dot{\mathbf{x}}^{\mathbf{P}}_{\mathbf{k+1}} + 8\dot{\mathbf{x}}_{\mathbf{k}} - \dot{\mathbf{x}}_{\mathbf{k-1}}) \end{aligned}$$

Adams-Bashforth-Moulton Predictor-Corrector Formulae

Sometimes a method is used that combines a predictor stage of AB with a subsequent corrector stage of AM, e.g.:

$$\begin{aligned}
\text{predictor:} \quad & \dot{\mathbf{x}}_\mathbf{k} = \mathbf{f}(\mathbf{x}_\mathbf{k}, t_k) \\
& \mathbf{x}^\mathbf{P}_\mathbf{k+1} = \mathbf{x}_\mathbf{k} + \frac{h}{12}(23\dot{\mathbf{x}}_\mathbf{k} - 16\dot{\mathbf{x}}_\mathbf{k-1} + 5\dot{\mathbf{x}}_\mathbf{k-2}) \\
\text{corrector:} \quad & \dot{\mathbf{x}}^\mathbf{P}_\mathbf{k+1} = \mathbf{f}(\mathbf{x}^\mathbf{P}_\mathbf{k+1}, t_{k+1}) \\
& \mathbf{x}^\mathbf{C}_\mathbf{k+1} = \mathbf{x}_\mathbf{k} + \frac{h}{12}(5\dot{\mathbf{x}}^\mathbf{P}_\mathbf{k+1} + 8\dot{\mathbf{x}}_\mathbf{k} - \dot{\mathbf{x}}_\mathbf{k-1})
\end{aligned}$$

These methods are called *Adams-Bashforth-Moulton predictor-corrector algorithms (ABM)*.

Adams-Bashforth-Moulton Predictor-Corrector Formulae

Sometimes a method is used that combines a predictor stage of AB with a subsequent corrector stage of AM, e.g.:

$$\text{predictor:} \quad \begin{aligned} \dot{\mathbf{x}}_\mathbf{k} &= \mathbf{f}(\mathbf{x}_\mathbf{k}, t_k) \\ \mathbf{x}^\mathbf{P}_\mathbf{k+1} &= \mathbf{x}_\mathbf{k} + \frac{h}{12}(23\dot{\mathbf{x}}_\mathbf{k} - 16\dot{\mathbf{x}}_\mathbf{k-1} + 5\dot{\mathbf{x}}_\mathbf{k-2}) \end{aligned}$$

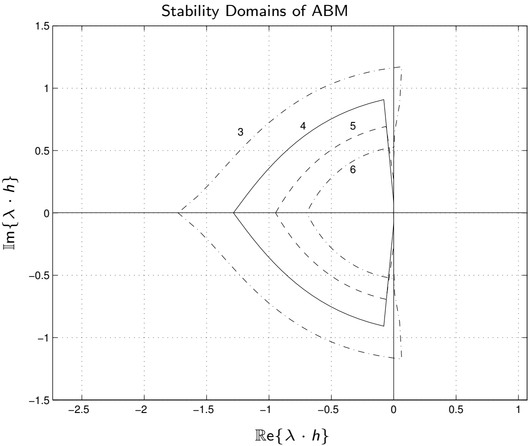
$$\text{corrector:} \quad \begin{aligned} \dot{\mathbf{x}}^\mathbf{P}_\mathbf{k+1} &= \mathbf{f}(\mathbf{x}^\mathbf{P}_\mathbf{k+1}, t_{k+1}) \\ \mathbf{x}^\mathbf{C}_\mathbf{k+1} &= \mathbf{x}_\mathbf{k} + \frac{h}{12}(5\dot{\mathbf{x}}^\mathbf{P}_\mathbf{k+1} + 8\dot{\mathbf{x}}_\mathbf{k} - \dot{\mathbf{x}}_\mathbf{k-1}) \end{aligned}$$

These methods are called *Adams-Bashforth-Moulton predictor-corrector algorithms (ABM)*.

The combined methods are *explicit algorithms*.

Stability Domains of ABM Algorithms

We can draw the *stability domains of the predictor-corrector ABM algorithms*.



Stability Domains of ABM Algorithms II

- Two function evaluations are required during each step, one for the predictor and the other for the corrector.

Stability Domains of ABM Algorithms II

- Two function evaluations are required during each step, one for the predictor and the other for the corrector.
- Due to the larger stability domains of the ABM algorithms in comparison with the AB algorithms, we can use steps during ABM simulations that are on average twice as large as those used during AB simulations. The efficiencies of the AB and ABM algorithms are very similar.

The Backward Difference Formulae

Until now, we have not encountered any family of formulae that could be used for the simulation of stiff systems. We shall introduce such a family now.

The Backward Difference Formulae

Until now, we have not encountered any family of formulae that could be used for the simulation of stiff systems. We shall introduce such a family now.

We formulate a *backward Newton-Gregory interpolation polynomial* of the state vector $\mathbf{x}$ around the time instant t_{k+1}:

$$\mathbf{x}(t) = \mathbf{x}_{k+1} + \binom{s}{1} \nabla \mathbf{x}_{k+1} + \binom{s+1}{2} \nabla^2 \mathbf{x}_{k+1} + \binom{s+2}{3} \nabla^3 \mathbf{x}_{k+1} + \dots$$

or:

$$\mathbf{x}(t) = \mathbf{x}_{k+1} + s \nabla \mathbf{x}_{k+1} + \left(\frac{s^2}{2} + \frac{s}{2} \right) \nabla^2 \mathbf{x}_{k+1} + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3} \right) \nabla^3 \mathbf{x}_{k+1} + \dots$$

The Backward Difference Formulae

Until now, we have not encountered any family of formulae that could be used for the simulation of stiff systems. We shall introduce such a family now.

We formulate a *backward Newton-Gregory interpolation polynomial* of the state vector $\mathbf{x}$ around the time instant t_{k+1}:

$$\mathbf{x}(t) = \mathbf{x}_{k+1} + \binom{s}{1} \nabla \mathbf{x}_{k+1} + \binom{s+1}{2} \nabla^2 \mathbf{x}_{k+1} + \binom{s+2}{3} \nabla^3 \mathbf{x}_{k+1} + \ldots$$

or:

$$\mathbf{x}(t) = \mathbf{x}_{k+1} + s \nabla \mathbf{x}_{k+1} + \left(\frac{s^2}{2} + \frac{s}{2} \right) \nabla^2 \mathbf{x}_{k+1} + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3} \right) \nabla^3 \mathbf{x}_{k+1} + \ldots$$

We differentiate this polynomial with respect to time, t:

$$\dot{\mathbf{x}}(t) = \frac{1}{h} \left[\nabla \mathbf{x}_{k+1} + \left(s + \frac{1}{2} \right) \nabla^2 \mathbf{x}_{k+1} + \left(\frac{s^2}{2} + s + \frac{1}{3} \right) \nabla^3 \mathbf{x}_{k+1} + \ldots \right]$$

The Backward Difference Formulae

Until now, we have not encountered any family of formulae that could be used for the simulation of stiff systems. We shall introduce such a family now.

We formulate a *backward Newton-Gregory interpolation polynomial* of the state vector $\mathbf{x}$ around the time instant t_{k+1}:

$$\mathbf{x}(t) = \mathbf{x}_{k+1} + \binom{s}{1}\nabla\mathbf{x}_{k+1} + \binom{s+1}{2}\nabla^2\mathbf{x}_{k+1} + \binom{s+2}{3}\nabla^3\mathbf{x}_{k+1} + \ldots$$

or:

$$\mathbf{x}(t) = \mathbf{x}_{k+1} + s\nabla\mathbf{x}_{k+1} + \left(\frac{s^2}{2} + \frac{s}{2}\right)\nabla^2\mathbf{x}_{k+1} + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3}\right)\nabla^3\mathbf{x}_{k+1} + \ldots$$

We differentiate this polynomial with respect to time, t:

$$\dot{\mathbf{x}}(t) = \frac{1}{h}\left[\nabla\mathbf{x}_{k+1} + \left(s + \frac{1}{2}\right)\nabla^2\mathbf{x}_{k+1} + \left(\frac{s^2}{2} + s + \frac{1}{3}\right)\nabla^3\mathbf{x}_{k+1} + \ldots\right]$$

We evaluate at $s = 0.0$:

$$\dot{\mathbf{x}}(t_{k+1}) = \frac{1}{h}\left[\nabla\mathbf{x}_{k+1} + \frac{1}{2}\nabla^2\mathbf{x}_{k+1} + \frac{1}{3}\nabla^3\mathbf{x}_{k+1} + \ldots\right]$$

The Backward Difference Formulae II

If we truncate this infinite series after the cubic term, we obtain:

$$h \cdot \mathbf{f}_{k+1} = \frac{11}{6}\mathbf{x}_{k+1} - 3\mathbf{x}_k + \frac{3}{2}\mathbf{x}_{k-1} - \frac{1}{3}\mathbf{x}_{k-2}$$

The Backward Difference Formulae II

If we truncate this infinite series after the cubic term, we obtain:

$$h \cdot \mathbf{f}_{k+1} = \frac{11}{6}\mathbf{x}_{k+1} - 3\mathbf{x}_k + \frac{3}{2}\mathbf{x}_{k-1} - \frac{1}{3}\mathbf{x}_{k-2}$$

We may solve this differential equation for the state variable at the time instant t_{k+1}:

$$\mathbf{x}_{k+1} = \frac{18}{11}\mathbf{x}_k - \frac{9}{11}\mathbf{x}_{k-1} + \frac{2}{11}\mathbf{x}_{k-2} + \frac{6}{11} \cdot h \cdot \mathbf{f}_{k+1}$$

The Backward Difference Formulae II

If we truncate this infinite series after the cubic term, we obtain:

$$h \cdot \mathbf{f}_{k+1} = \frac{11}{6}\mathbf{x}_{k+1} - 3\mathbf{x}_k + \frac{3}{2}\mathbf{x}_{k-1} - \frac{1}{3}\mathbf{x}_{k-2}$$

We may solve this differential equation for the state variable at the time instant t_{k+1}:

$$\mathbf{x}_{k+1} = \frac{18}{11}\mathbf{x}_k - \frac{9}{11}\mathbf{x}_{k-1} + \frac{2}{11}\mathbf{x}_{k-2} + \frac{6}{11} \cdot h \cdot \mathbf{f}_{k+1}$$

There results the well-known *third-order accurate backward difference formula (BDF3)*.

The Backward Difference Formulae II

If we truncate this infinite series after the cubic term, we obtain:

$$h \cdot \mathbf{f}_{k+1} = \frac{11}{6}\mathbf{x}_{k+1} - 3\mathbf{x}_k + \frac{3}{2}\mathbf{x}_{k-1} - \frac{1}{3}\mathbf{x}_{k-2}$$

We may solve this differential equation for the state variable at the time instant t_{k+1}:

$$\mathbf{x}_{k+1} = \frac{18}{11}\mathbf{x}_k - \frac{9}{11}\mathbf{x}_{k-1} + \frac{2}{11}\mathbf{x}_{k-2} + \frac{6}{11} \cdot h \cdot \mathbf{f}_{k+1}$$

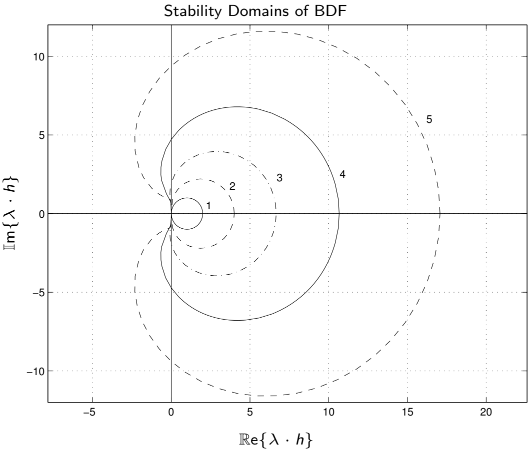
There results the well-known *third-order accurate backward difference formula (BDF3)*.

All BDF algorithms can be characterized by a vector α specifying the factor associated with the time derivative and by a matrix β that lists the weights of the past values of the state vector:

$$\alpha = \begin{pmatrix} 1 \\ 2/3 \\ 6/11 \\ 12/25 \\ 60/137 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 4/3 & -1/3 & 0 & 0 & 0 \\ 18/11 & -9/11 & 2/11 & 0 & 0 \\ 48/25 & -36/25 & 16/25 & -3/25 & 0 \\ 300/137 & -300/137 & 200/137 & -75/137 & 12/137 \end{pmatrix}$$

Stability Domains of BDF Algorithms

We can draw the *stability domains of the backward difference formulae (BDF)*.



Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.

Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.
- The algorithm **BDF2** is also L-stable.

Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.
- The algorithm **BDF2** is also L-stable.
- The higher-order BDF algorithms aren't L-stable. They are not even A-stable. A region of the complex plane to the left of the imaginary axis is unstable.

Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.
- The algorithm **BDF2** is also L-stable.
- The higher-order BDF algorithms aren't L-stable. They are not even A-stable. A region of the complex plane to the left of the imaginary axis is unstable.
- There exist no stable BDF algorithms for orders higher than six.

Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.
- The algorithm **BDF2** is also L-stable.
- The higher-order BDF algorithms aren't L-stable. They are not even A-stable. A region of the complex plane to the left of the imaginary axis is unstable.
- There exist no stable BDF algorithms for orders higher than six.
- Already the **BDF6** algorithm may only be used for the simulation of stiff systems without oscillatory behavior, such as thermal or chemical systems, because the unstable region to the left of the imaginary axis of the complex plane is too large.

Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.
- The algorithm **BDF2** is also L-stable.
- The higher-order BDF algorithms aren't L-stable. They are not even A-stable. A region of the complex plane to the left of the imaginary axis is unstable.
- There exist no stable BDF algorithms for orders higher than six.
- Already the **BDF6** algorithm may only be used for the simulation of stiff systems without oscillatory behavior, such as thermal or chemical systems, because the unstable region to the left of the imaginary axis of the complex plane is too large.
- The BDF algorithms are implicit methods. It is also possible to derive explicit BDF algorithms, but unfortunately, they are unstable everywhere.

Stability Domains of BDF Algorithms II

- The algorithm **BDF1=BE=BRK1=AM1** is L-stable.
- The algorithm **BDF2** is also L-stable.
- The higher-order BDF algorithms aren't L-stable. They are not even A-stable. A region of the complex plane to the left of the imaginary axis is unstable.
- There exist no stable BDF algorithms for orders higher than six.
- Already the **BDF6** algorithm may only be used for the simulation of stiff systems without oscillatory behavior, such as thermal or chemical systems, because the unstable region to the left of the imaginary axis of the complex plane is too large.
- The BDF algorithms are implicit methods. It is also possible to derive explicit BDF algorithms, but unfortunately, they are unstable everywhere.
- **Thanks to their suitability for the simulation of stiff systems and due to their simplicity, the BDF algorithms are among the most widely used numerical ODE solvers for the simulation of dynamic systems.**

The Explicit Nyström Formulae

We start with the same backward Newton-Gregory polynomial that we already used for the derivation of the AB algorithms:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_k + \binom{s}{1}\nabla\mathbf{f}_k + \binom{s+1}{2}\nabla^2\mathbf{f}_k + \binom{s+2}{3}\nabla^3\mathbf{f}_k + \dots$$

The Explicit Nyström Formulae

We start with the same backward Newton-Gregory polynomial that we already used for the derivation of the AB algorithms:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_k + \binom{s}{1}\nabla \mathbf{f}_k + \binom{s+1}{2}\nabla^2 \mathbf{f}_k + \binom{s+2}{3}\nabla^3 \mathbf{f}_k + \dots$$

This time, we integrate the polynomial over the time interval $t \in [t_{k-1}, t_{k+1}]$, i.e., over the interval $s \in [-1.0, +1.0]$. We encounter the family of formulae:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_{k-1}) + h\left(2\mathbf{f}_k + \frac{1}{3}\nabla^2 \mathbf{f}_k + \frac{1}{3}\nabla^3 \mathbf{f}_k + \dots\right)$$

The Explicit Nyström Formulae

We start with the same backward Newton-Gregory polynomial that we already used for the derivation of the AB algorithms:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_k + \binom{s}{1} \nabla \mathbf{f}_k + \binom{s+1}{2} \nabla^2 \mathbf{f}_k + \binom{s+2}{3} \nabla^3 \mathbf{f}_k + \dots$$

This time, we integrate the polynomial over the time interval $t \in [t_{k-1}, t_{k+1}]$, i.e., over the interval $s \in [-1.0, +1.0]$. We encounter the family of formulae:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_{k-1}) + h \left(2\mathbf{f}_k + \frac{1}{3} \nabla^2 \mathbf{f}_k + \frac{1}{3} \nabla^3 \mathbf{f}_k + \dots \right)$$

If we truncate this infinite series after the cubic term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_{k-1}) + \frac{h}{3} (8\mathbf{f}_k - 5\mathbf{f}_{k-1} + 4\mathbf{f}_{k-2} - \mathbf{f}_{k-3})$$

This algorithm is called *fourth-order accurate Nyström algorithm (Ny4)*.

The Explicit Nyström Formulae II

The Nyström algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivatives:

$$\alpha = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 90 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 7 & -2 & 1 & 0 & 0 \\ 8 & -5 & 4 & -1 & 0 \\ 269 & -266 & 294 & -146 & 29 \end{pmatrix}$$

The Explicit Nyström Formulae II

The Nyström algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivatives:

$$\alpha = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 90 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 7 & -2 & 1 & 0 & 0 \\ 8 & -5 & 4 & -1 & 0 \\ 269 & -266 & 294 & -146 & 29 \end{pmatrix}$$

Unfortunately, all of the algorithms in the Nyström family are unstable.

The Explicit Nyström Formulae II

The Nyström algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivatives:

$$\alpha = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 90 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 7 & -2 & 1 & 0 & 0 \\ 8 & -5 & 4 & -1 & 0 \\ 269 & -266 & 294 & -146 & 29 \end{pmatrix}$$

Unfortunately, all of the algorithms in the Nyström family are unstable.

These algorithms can therefore not be used alone, but they may still be usable for *individual stages within blended or cyclic methods*.

The Implicit Milne Formulae

We start with the same backward Newton-Gregory polynomial that we had already used for the derivation of the AM algorithms:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{k+1} + \binom{s}{1}\nabla\mathbf{f}_{k+1} + \binom{s+1}{2}\nabla^2\mathbf{f}_{k+1} + \binom{s+2}{3}\nabla^3\mathbf{f}_{k+1} + \ldots$$

The Implicit Milne Formulae

We start with the same backward Newton-Gregory polynomial that we had already used for the derivation of the AM algorithms:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{k+1} + \binom{s}{1}\nabla\mathbf{f}_{k+1} + \binom{s+1}{2}\nabla^2\mathbf{f}_{k+1} + \binom{s+2}{3}\nabla^3\mathbf{f}_{k+1} + \dots$$

We integrate this polynomial over the time interval $t \in [t_{k-1}, t_{k+1}]$, i.e., over the interval $s \in [-2.0, 0.0]$. We encounter the family of formulae:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_{k-1}) + h\left(2\mathbf{f}_{k+1} - 2\nabla\mathbf{f}_{k+1} + \frac{1}{3}\nabla^2\mathbf{f}_{k+1} + 0\nabla^3\mathbf{f}_{k+1} + \dots\right)$$

The Implicit Milne Formulae

We start with the same backward Newton-Gregory polynomial that we had already used for the derivation of the AM algorithms:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{k+1} + \binom{s}{1}\nabla\mathbf{f}_{k+1} + \binom{s+1}{2}\nabla^2\mathbf{f}_{k+1} + \binom{s+2}{3}\nabla^3\mathbf{f}_{k+1} + \dots$$

We integrate this polynomial over the time interval $t \in [t_{k-1}, t_{k+1}]$, i.e., over the interval $s \in [-2.0, 0.0]$. We encounter the family of formulae:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_{k-1}) + h\left(2\mathbf{f}_{k+1} - 2\nabla\mathbf{f}_{k+1} + \frac{1}{3}\nabla^2\mathbf{f}_{k+1} + 0\nabla^3\mathbf{f}_{k+1} + \dots\right)$$

If we truncate this infinite series after the cubic (or rather quadratic) term, we obtain:

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_{k-1}) + \frac{h}{3}\left(\mathbf{f}_{k+1} + 4\mathbf{f}_k + \mathbf{f}_{k-1}\right)$$

This algorithm is called *fourth-order accurate Milne algorithm (Mi4)*.

The Implicit Milne Formulae II

The Milne algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivatives:

$$\alpha = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 90 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 29 & 124 & 24 & 4 & -1 \end{pmatrix}$$

The Implicit Milne Formulae II

The Milne algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivatives:

$$\alpha = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 90 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 29 & 124 & 24 & 4 & -1 \end{pmatrix}$$

Unfortunately, also all of the algorithms of the Milne family are unstable.

The Implicit Milne Formulae II

The Milne algorithms can be characterized by a vector α specifying the factor associated with the time step h and by a matrix β that lists the weights of the derivatives:

$$\alpha = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \\ 90 \end{pmatrix} \quad , \quad \beta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 29 & 124 & 24 & 4 & -1 \end{pmatrix}$$

Unfortunately, also all of the algorithms of the Milne family are unstable.

Also these algorithms are sometimes used for *individual stages within blended or cyclic algorithms*.

Conclusions

- In this presentation, we introduced a systematic way of developing all classes of *linear multi-step integration algorithms* for the simulation of dynamic systems.

Conclusions

- In this presentation, we introduced a systematic way of developing all classes of *linear multi-step integration algorithms* for the simulation of dynamic systems.
- We started out with the two families of *Adams algorithms*. There exists one family of *explicit Adams algorithms* and another of *implicit Adams algorithms*.

Conclusions

- In this presentation, we introduced a systematic way of developing all classes of *linear multi-step integration algorithms* for the simulation of dynamic systems.
- We started out with the two families of *Adams algorithms*. There exists one family of *explicit Adams algorithms* and another of *implicit Adams algorithms*.
- We analyzed their *numerical stability properties*. We demonstrated their shortcomings. In practical simulation, these methods aren't used much except for the simulation of linear non-stiff systems without discontinuities.

Conclusions

- In this presentation, we introduced a systematic way of developing all classes of *linear multi-step integration algorithms* for the simulation of dynamic systems.
- We started out with the two families of *Adams algorithms*. There exists one family of *explicit Adams algorithms* and another of *implicit Adams algorithms*.
- We analyzed their *numerical stability properties*. We demonstrated their shortcomings. In practical simulation, these methods aren't used much except for the simulation of linear non-stiff systems without discontinuities.
- **Matlab**'s lsim function makes use of the **AB3** algorithm.

Conclusions

- In this presentation, we introduced a systematic way of developing all classes of *linear multi-step integration algorithms* for the simulation of dynamic systems.
- We started out with the two families of *Adams algorithms*. There exists one family of *explicit Adams algorithms* and another of *implicit Adams algorithms*.
- We analyzed their *numerical stability properties*. We demonstrated their shortcomings. In practical simulation, these methods aren't used much except for the simulation of linear non-stiff systems without discontinuities.
- **Matlab**'s lsim function makes use of the **AB3** algorithm.
- Subsequently, we introduced the family of the *backward difference formulae*. These algorithms are widely used for the practical simulation of stiff systems. They are among the most widely used numerical ODE solvers in *modeling and simulation environments*.

Conclusions

- In this presentation, we introduced a systematic way of developing all classes of *linear multi-step integration algorithms* for the simulation of dynamic systems.

- We started out with the two families of *Adams algorithms*. There exists one family of *explicit Adams algorithms* and another of *implicit Adams algorithms*.

- We analyzed their *numerical stability properties*. We demonstrated their shortcomings. In practical simulation, these methods aren't used much except for the simulation of linear non-stiff systems without discontinuities.

- **Matlab**'s lsim function makes use of the **AB3** algorithm.

- Subsequently, we introduced the family of the *backward difference formulae*. These algorithms are widely used for the practical simulation of stiff systems. They are among the most widely used numerical ODE solvers in *modeling and simulation environments*.

- **Dymola** uses **DASSL**, an implementation of a variable-step, variable-order BDF algorithm as its default ODE solver.