Numerical Solutions to Differential Equations

Lecture Notes #12
Linear Multistep Methods for Stiff Systems

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2015

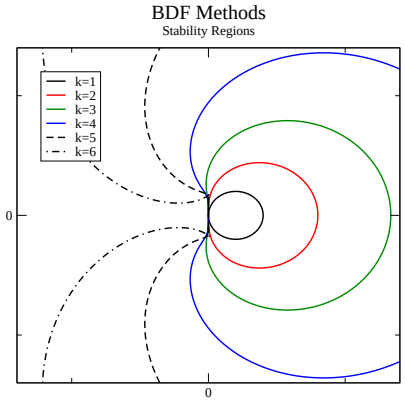
Outline

Today's Lecture

- Wrap up the discussion of solutions of stiff initial value problems.
- An overview of how to use linear multistep methods for stiff problems.
- How to build an efficient scheme for stiff ODEs.

Linear Multistep Methods and Stiffness...

Linear Multistep Methods tend to have small regions of absolute stability and are therefore not particularly well suited for stiff problems.
The **notable exception** are the Backward Differentiation Formula (BDF) methods.



The Second Dahlquist Barrier

Dahlquist (1963) quantified how difficult it is for Linear Multistep Methods to achieve A-stability:

Theorem (The Second Dahlquist Barrier)

1. *An explicit linear multistep method cannot be A-stable.*
2. *The order of an A-stable linear multistep method cannot exceed 2.*
3. *The second-order A-stable linear multistep method with smallest error constant is the Trapezoidal Rule.*

Implementing Trapezoidal Rule for Stiff Systems

Recall: Trapezoidal rule is not L-stable, and if we have an eigenvalue with large negative real part we may have a damped oscillatory behavior until the associated transient has decayed.

Real-world implementation of Trapezoidal rule for stiff systems usually employ 3 "tricks" —

1. A smoothing procedure to lessen the oscillatory behavior.
2. Extrapolation to raise the order to 4.
3. Local error estimation by Richardson Extrapolation.

Smoothing for the Trapezoidal Rule I/II

The oscillations for the fast transients can be alleviated in two ways —

1. by taking a smaller step initially (adaptive scheme), or
2. by smoothing the solution.

The smoothing idea was introduced by Lindberg (1971) — We replace $\mathbf{y_n}$ by

$$\widehat{y}_n = \frac{y_{n-1} + 2y_n + y_{n+1}}{4}$$

and then propagate the solution. This weighted average smooths out the oscillations.

Smoothing for the Trapezoidal Rule II/II

The smoothing procedure is active for the first couple of steps, while the fast transients are still alive, and/or introduced automatically whenever the solution exhibits lack of smoothness.

Of course, the smoothing affects the local truncation error; the complete analysis is in Lindberg's paper:

Reference

B. Lindberg, *On Smoothing and Extrapolation for the Trapezoidal Rule*, BIT, **11**, pp. 29–52, 1971.

BIT is a suitable permutation of the letters T, I, and B from *Nordisk* **T***idskrift för* **I***nformations* **B***ehandling*. (Obviously!)

Another Theorem for Linear Multistep Methods I/II

If we relax the requirement for complete A-stability, there are some options...

Theorem (Widlund, 1967)

1. *An* **explicit** *linear multistep method cannot be A(0)-stable.*
2. *There is only one A(0)-stable linear k-step method whose order exceeds k, the Trapezoidal Rule.*
3. *For all $\alpha \in [0, \pi/2)$ there exists A(α)-stable linear k-step methods of order p for which $k = p = 3$, and $k = p = 4$.*

If we relax the "for all" requirement in (3)...

Another Theorem for Linear Multistep Methods II/II

If we relax the "for all" requirement in (3)...

... then we can find k-step methods of order $k > 4$ that are A(α)-stable for **some** value of α — most notably the BDF methods for $k = 4, 5, 6$.

Recall that the BDF methods are zero-stable for $k \leq 6$.

BDF methods — The Choice for Stiff Problems

Recall: The stability polynomial for a linear multistep method is

$$\pi(r, \widehat{h}) = \rho(r) - \widehat{h}\sigma(r)$$

For a stiff system $|\widehat{h}|$ is very large, hence the stability properties would be dominated by $\sigma(r)$.

For stability, we want the roots of $\sigma(r)$ to be inside the unit circle.

The safest place would be the center of the unit circle!
— **With $\sigma(r) = r^k$ we achieve just that.**

$\sigma(r) = r^k$ & implicit method & maximal order $\Rightarrow$ BDF-methods!!!

Building an Efficient Algorithm for Stiff Problems I/III

We will use the BDF methods to construct a efficient algorithms for the solution of stiff ODEs.

Starting from the predictor-corrector P(EC)$^\mu$ framework:

1. The BDF-method will play the role of the corrector — C — in the Adams-Bashforth-Moulton predictor-corrector framework.
2. The fixed point iteration — (EC)$^\mu$ — is replaced by a Newton iteration pursued to convergence (thus the stability properties of the predictor does not influence the overall stability properties.)

Newton iteration — advantage, convergence
Newton iteration converges quadratically ($\approx$ doubling the number of accurate digits in each iteration), whereas fixed point iteration converges linearly.

Newton iteration — disadvantage, starting point
Unlike the fixed point iteration, the Newton iteration **does not** converge for arbitrary starting values — a good starting value is required.

Warning

An explicit predictor is likely **not** to give a good enough starting value.

3. We replace the Adams-Bashforth Predictor — P — by an extrapolation of previously computed y-values. For a k-step k-order method, the extrapolation must be based on the previous $(k+1)$ points $\{y_{n+k-1}, y_{n+k-2}, \ldots, y_{n-1}\}$:

$$y_{n+k}^{[0]} = \sum_{i=0}^{k} \nabla^i y_{n+k-1}.$$

This extrapolant-predictor has an error constant $C_{k+1}^* = 1$, hence the Milne's estimate for the principal part of the local truncation error is still available:

$$\text{LTE} \sim \frac{C_{k+1}}{1 - C_{k+1}} \left[y_{n+k} - y_{n+k}^{[0]} \right].$$

`[P]` Extrapolate to get an initial value for the Newton iteration:

$$y_{n+k}^{[0]} = \sum_{i=0}^{k} \nabla^i y_{n+k-1}.$$

`[(EC)`$^\infty$`]` Use the implicit BDF-method as the corrector, and solve **to convergence** using a quadratically convergent Newton solver.

`[(Err)]` Estimate the error using Milne's Error estimate:

$$\text{LTE} \sim \frac{C_{k+1}}{1 - C_{k+1}} \left[y_{n+k} - y_{n+k}^{[0]} \right].$$

`[(Tol)]` Is the error small enough? If not, either (1) reduce the step size, or (2) increase the order of the method.

Besides covering the Newton solver, which we need both for the BDF-based solver, and for the Implicit Runge-Kutta methods, we have covered the solution of the Initial Value Problem for ODEs in quite a bit of detail.

Before reviewing Newton's Method, lets summarize what we have found so far...

The Initial Value Problem

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

Taylor Series Methods

Best used when the Taylor expansion of $f(t, y(t))$ is available and cheap/easy to compute.

Stiffness: Small stability region. Step-size h very restrictive.

Runge-Kutta Methods

When the Taylor expansion of $f(t, y(t))$ is not easily computable (or prohibitively expensive), but multiple evaluation of $f(t, y(t))$ incur a reasonable amount of work, then RK-methods are a good choice.

Stiffness: When the problem is stiff, we have to use fully implicit RK-methods. We have seen that there are A-stable s-stage $2s$-order methods (Gauss-Legendre) for arbitrary s, as well as L-stable s-stage $(2s-1)$-order methods (Radau I-A, and II-A).

Linear Multistep Methods

Explicit LMMs only require one new function evaluation per step, making them very competitive (fast and cheap). Used in the predictor-corrector context P(EC)$^\mu$, only $(1+\mu)$ evaluations per step are required.

The main **drawback** is that LMMs are not self-starting, so we need an RK- or Taylor-series method (possibly with Richardson Extrapolation) to generate enough accurate starting information.

Stiffness: If/when we can live with an A(α)-stable method, implementing efficient LMM-based stiff solvers is quite straight forward (at least up to order 6...)

In both implicit RK-methods and the BDF-based LMM methods for stiff problems we run into the problem of solving a non-linear equation

$$F(\tilde{\mathbf{y}}_{n+1}, \dots) = G(\tilde{\mathbf{y}}_{n+1}, \dots),$$

we can always rewrite this problem as

$$f(\tilde{\mathbf{y}}_{n+1}) = F(\tilde{\mathbf{y}}_{n+1}, \dots) - G(\tilde{\mathbf{y}}_{n+1}, \dots) = 0,$$

which means we are trying to solve

$$f(\tilde{\mathbf{y}}_{n+1}) = 0.$$

If our problem is scalar (one-dimensional), then $\tilde{\mathbf{y}}_{n+1} = y_{n+1}$ is a scalar, and we can use Newton's method as described in Math 541.

Newton's Method for Scalar Problems

We are trying to find the roots, y^* of the equation

$$f(y) = 0.$$

If/when we have a guess close to a root, *i.e.* $|y - y^*|$ is small, then we can formally Taylor expand around y and get

$$f(y^*) = f(y) + (y^* - y)f'(y) + \frac{(y^* - y)^2}{2} f''(\xi(y, y^*))$$
$$\xi(y, y^*) \in [\min(y, y^*), \max(y, y^*)].$$

Since $|y - y^*|$ is small, $|y - y^*|^2$ is even smaller, so we neglect the quadratic term in the expansion. Also $f(y^*) = 0$ by assumption, hence we have

$$0 \approx f(y) + (y^* - y)f'(y).$$

Newton's Method for Scalar Problems, II

We solve for y^* and get

$$y^* \approx y - \frac{f(y)}{f'(y)}.$$

A Newton (iterative) solver implements

$$y^{[\nu+1]} = y^{[\nu]} - \frac{f(y^{[\nu]})}{f'(y^{[\nu]})},$$

and converges quadratically as long as $f'(y^*) \neq 0$.

If, *a priori* we know that the derivative will be zero at the root, then we can implement the more costly version of Newton's method:

$$y^{[\nu+1]} = y^{[\nu]} - \frac{f(y^{[\nu]})f'(y^{[\nu]})}{[f'(y^{[\nu]})]^2 - f''(y^{[\nu]})f(y^{[\nu]})}.$$

Newton's Method for more Than One Dimension I/II

When we have n simultaneous ODEs

$$\begin{aligned} y_1'(t) &= g_1(t, y_1, \ldots, y_n) \\ &\vdots \\ y_n'(t) &= g_n(t, y_1, \ldots, y_n) \end{aligned}$$

by the same procedure (Taylor expansion for a vector-valued function of a vector-valued argument) we get

$$0 \approx \tilde{\mathbf{f}}(t, \tilde{\mathbf{y}}) - \underbrace{\left[\frac{\partial g_r}{\partial y_c}(t, \tilde{\mathbf{y}})\right]_{r,c=1,\ldots,n}}_{J(t,\tilde{\mathbf{y}})} (\tilde{\mathbf{y}}^* - \tilde{\mathbf{y}})$$

The matrix $J(t, \tilde{\mathbf{y}})$ is usually referred to as the "Jacobian."

Newton's Method for more Than One Dimension II/II

Again we solve for y^* and define our iterative scheme

$$\tilde{\mathbf{y}}^{[\nu+1]} = \tilde{\mathbf{y}}^{[\nu]} - [J(t, \tilde{\mathbf{y}}^{[\nu]})]^{-1} \tilde{\mathbf{f}}(t, \tilde{\mathbf{y}}^{[\nu]})$$

How to solve this iteration **efficiently** (especially for large systems) is a matter which will be covered in Math 693a (and there are some useful ideas in Math 543).

- We now have a pretty complete picture of how to solve the initial value problem, even when it fights back (stiff problems).
- The past few lectures on stiff problems have covered some pretty "mature" topics which put together quite a few ideas from (vector) calculus, complex analysis, previous knowledge of numerical methods, etc.

- Being a *"computational scientist"* means you have to understand how **your problem** fits into the numerical framework, and make sure your starting methods, error checking, and stability analysis are done right. — At some point you have to understand *every aspect of the problem* in enough detail that you can implement it in your favorite computer language.
- Given the complexity of the methods we have covered, it is a daunting task to try to give (non-silly) examples of all of them. However, in the next lecture(s) some examples will be given.