MULTIRATE LINEAR MULTISTEP METHODS

C. W. GEAR and D. R. WELLS

Department of Computer Science,
University of Illinois at Urbana-Champaign,
1304 West Springfield Avenue,
Urbana, Illinois 61801, U.S.A.

IBM Corporation
DSC Division
Neighborhood Road
Kingston, NY 12401, U.S.A.

Dedicated to Professor Germund Dahlquist on the occasion of his 60th birthday

Abstract

The design of a code which uses different stepsizes for different components of a system of ordinary differential equations is discussed. Methods are suggested which achieve moderate efficiency for problems having some components with a much slower rate of variation than others. Techniques for estimating errors in the different components are analyzed and applied to automatic stepsize and order control. Difficulties, absent from non-multirate methods, arise in the automatic selection of stepsizes, leading to a suggested organization of the code that is counter-intuitive. An experimental code and some initial experiments are described.

1. Introduction.

The principal objective of a multirate method is to reduce the integration time by using larger stepsizes for those variables in a system that have a behavior which is slow compared to the fastest variables. The ordinary differential equation system

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

where \( f \in \mathbb{R}^N \times \mathbb{R} \rightarrow \mathbb{R}^N \) may be partitioned into the \( M \) coupled systems

\[ y'_i = f_i(y, t), \quad y_i(0) = y_{i0} \]

where \( f_i \in \mathbb{R}^N \times \mathbb{R}^N, 1 \leq i \leq M, \sum N_i = N, \) and the components of \( y_i \) are all to be treated with the same stepsize. For expository purposes in this paper, we will discuss the cases \( M = 2 \) and \( M = 3 \) using the systems

\[ y' = b(y, z, t) \]
\[ z' = c(y, z, t) \]

1 Supported in part by the Department of Energy under grant DOE DEAC0276ERO2383.

2 Work done while attending the University of Illinois.

Received December 1983. Revised May 1984.
and

\[
x' = a(x, y, z, t) \\
y' = b(x, y, z, t) \\
z' = c(x, y, z, t)
\]

respectively, where letters later in the alphabet represent faster components (thus, \(z\) is faster than \(y\)). The results extend to the general case.

A system could be partitioned in this way to permit different methods to be used for different components. This has been examined in [1] and [5]. Different methods might be used to improve the stability properties of the combined method, to handle different continuity properties of different components, or for other reasons such as the availability of higher derivatives of some but not all components. However, our purpose in partitioning is to use as large a stepsize as possible for each component to gain efficiency. We will restrict ourselves to the use of the same general type of method for all components (namely, a linear multistep method), although the order and choice of stiff (BDF) versus nonstiff (Adams) formulas is component dependent.

The objective of a modern code is to select automatically as many of the parameters of the method as possible, making the code available to users in other areas of expertise. The use of multirate methods increases the number of parameter choices. There are the usual choices of stepsize and order for each component. These choices can be made automatically using extensions of well-known techniques, although it is the automatic choice of stepsize which leads to the greatest difficulties in multirate methods. An approach to the resolution of these difficulties is the principal topic of this paper. There is the choice of a stiff versus nonstiff method for each component. In the experimental code we discuss later, this choice is not made by the program, although the techniques being used by various authors (for example [6] and [7]) could be added without difficulty. There is the choice of which components are fast and which are slow. For example, if a code were presented with the system (1.3), it could attempt to determine automatically which component is fast and which is slow. We have not attempted to do this in the code reported here for reasons which will become evident in later discussion. However, it does not appear to present an insurmountable difficulty. Finally, there is the choice of partitioning into subsystems. This can either be done statically before the start of the integration or dynamically during the integration. Neither has been attempted automatically for several reasons. Multirate methods introduce a fair amount of inefficiency at the code level. Dynamic repartitioning would increase this inefficiency considerably. Static partitioning could be done only on the basis of initial values, which may give a false picture of the behavior over the bulk of the interval. Hence it is of little value unless done by the user who has additional