Solution Of Stiff Systems Described By Ordinary Differential Equations Using Regression Backward Difference Formulae (RBDF)

Klaus Hermann

1995

STATEMENT BY AUTHOR

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

______________________________	__________________
Dr. François E. Cellier Professor of Electrical and Computer Engineering	Date

ACKNOWLEDGMENTS

I would like to express my special appreciation to Dr. François E. Cellier, my thesis advisor, for his guidance, valuable advice and for always devoting his time and patience to me during this research.

I should also like to thank my friend Michael Schweisguth who reviewed this thesis and helped me to find formulations in proper English.

Special thanks are due to Dr. Christoph Woernle, Dr. Parviz Nikravesh and the German Exchange Service (DAAD) for organizing the exchange program between the University of Stuttgart/Germany and the University of Arizona at Tucson/U.S.A., which allowed me to make all these precious experiences brought along by an academic year abroad.

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the past research has been done to solve stiff systems described by ordinary differential equations (ODEs). An important result are the famous *Backward Difference Formulae (BDF)* [10]. These methods are capable of solving stiff ODE-systems up to accuracy order six — accuracy order six means that the error made at each integration step is roughly proportional to the seventh power of the step-size. So far, no BDF algorithms of seventh order and higher, have been found that are stable.

This thesis proposes the *Regression Backward Difference Formulae (RBDF)* as new numerical solution methods for stiff systems described by first order ordinary differential equations. The RBDF algorithms derived in this thesis, by means of a new regression technique, will be of sixth and seventh order, and it will be shown that some of the sixth order RBDF algorithms compare favorably against the sixth order BDF.

The results for the new seventh order RBDF algorithms are shown, but not compared to BDF since no stable seventh order BDF technique exists.

It can be expected that RBDF methods of order higher than seven may be found by using the proposed regression approach.

In particular, celestial analysis demands highly accurate calculation and integration. Therefore, this might be one area where even higher order RBDF techniques than seventh order RBDF could be applied.

CHAPTER 1

Introduction

1.1 Ordinary Differential Equations (ODE)

A first order scalar ODE can be written as

$$\frac{dx}{dt} = \dot{x}(t) = f(x,u,t) \tag{1.1}$$

with initial condition

$$x(t_0) = x_0. \tag{1.2}$$

Here t is the independent variable (which usually, but not necessarily, denotes time), u is a given input, and the function f indicates any explicit functionality between the dependent variable x and the independent variable t.

Without loss of generality only systems, as shown in equation (1.3), are considered.

$$\dot{x}(t) = f(x,u), \tag{1.3}$$

$$x(t_0) = x_0. \tag{1.4}$$

i.e., systems, where the independent variable t does not appear explicitly in the equations.

In addition to the first-order scalar equation (1.3) it is possible to consider a set of simultaneous first-order equations, or an equivalent higher-order single equation. Thus we may write

$$\dot{x_i}(t) = f_i(u, x_1, \ldots x_{n_s}) \qquad i = 1, 2, \ldots n_s, \tag{1.5}$$

as representing a set of n_s simultaneous first-order ODEs with the corresponding initial conditions

$$x_i(t_0) = x_{0i}. \tag{1.6}$$

As long as the derivatives $\dot{x_1}, \ldots \dot{x_{n_s}}$ appear only on the left-hand side of the differential equations, then (1.5) is equivalent to one n_s^{th}-order equation. The equations (1.5) and (1.6) can be written in vector form:

$$\begin{aligned} \underline{\frac{dx}{dt}} &= \underline{\dot{x}}(t) = \underline{f}(\underline{x}, u), \\ \underline{x}(t_0) &= \underline{x}_0, \end{aligned} \tag{1.7}$$

These relations represent the autonomous initial value problem that this thesis will solve with RBDF techniques.

1.2 Numerical methods to solve ODEs

There is a variety of numerical techniques which may be applied to solve system (1.7). Two major approaches are used to solve these systems: The *single-step methods* and the *multistep methods.* Both techniques try to approximate a Taylor-Series expansion of the unknown solution around the current time instant.

In single-step methods, the higher order derivative information is discarded. This information is thrown out by only using the function value x_k at the preceding time instant, and maybe its derivative, $\dot{x}_k$, to calculate the unknown variable x_{k+1}. The multistep approach, however, preserves some of the higher order derivative information by using data of earlier values x_i and $\dot{x}_i$.

In general a multistep integration algorithm can be expressed as

$$\begin{aligned} x_{k+1} &= b_{-1}hf_{k+1} + a_0 x_k + b_0 hf_k + a_1 x_{k-1} + \ldots + a_{m-1}x_{k+1-m} + b_{m-1}hf_{k+1-m} \\ &= \sum_{i=0}^{m-1} a_i x_{k-i} + \sum_{i=-1}^{m-1} b_i hf_{k-i}, \end{aligned} \tag{1.8}$$

where $f_i = \dot{x}_i$ is the derivative of the system variable x at the time instant $t = t_0 + i\Delta t$. Note that this method (1.8) is implicit when $b_{-1} \neq 0$ and it is explicit when $b_{-1} = 0$. The number m denotes the number of steps, so the algorithm shown in (1.8) can be called an *m-step integration algorithm.*

Note also that the algorithm depicted in equation (1.8) only uses values of x and $\dot{x}$ and not higher derivatives, such as $\ddot{x}_k$. It is also required that (1.8) be applied with equispaced steps. These restrictions may limit the performance of solutions based on (1.8). However, (1.8) represents a very important and extensive class of formulas which will be used to derive RBDF methods.

The two most famous and frequently used multistep integration methods are the *Adams methods* and the *Backward Difference Formulae (BDF).*

The Adams algorithms use the function value x_k one time step back and the derivative values of several of the preceding function values such that they can be written in the form

$$x_{k+1} = b_{-1}hf_{k+1} + a_0 x_k + b_0 hf_k + b_1 hf_{k-1} + \ldots + b_{m-1}hf_{k+1-m}. \tag{1.9}$$

If $b_{-1} = 0$ then equation (1.9) is reduced to the explicit *Adams-Bashforth algorithms.* However, if $b_{-1} \neq 0$ then equation (1.9) reduces to the implicit *Adams-Moulton methods.*

Unlike the Adams methods, the BDF techniques are always implicit and take into account the current derivative $f_{k+1} = \dot{x}_{k+1}$ as well as the state values calculated at earlier time instants. Figures 1.1 and 1.2 show which data points the Adams methods and BDF use to compute the unknown value x_{k+1} of the function x.

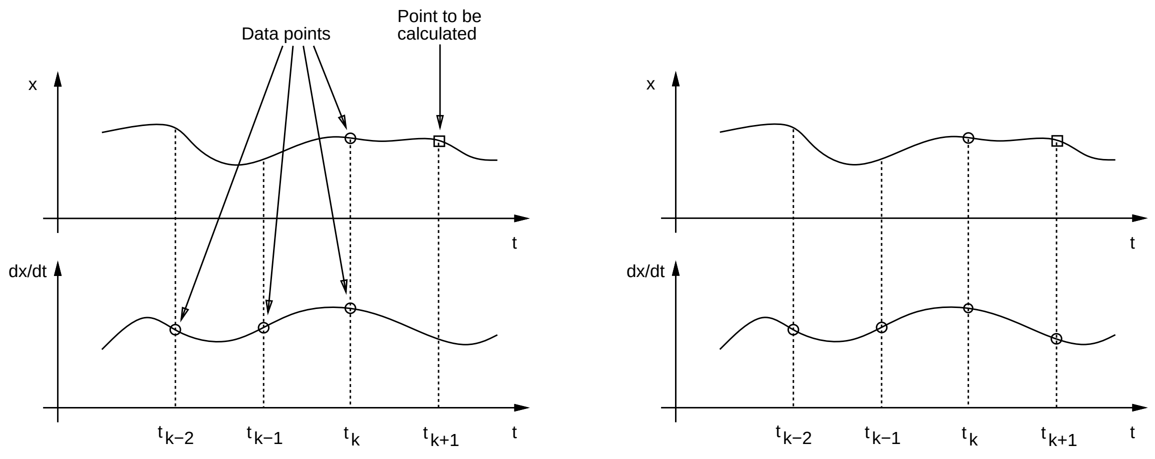


Figure 1.1: Data points used by the Adams-Bashforth algorithms (left) and the Adams-Moulton algorithms (right)

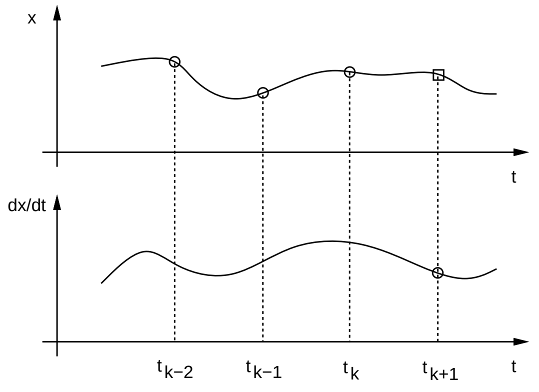


Figure 1.2: Data points used by the BDF algorithms

1.3 Stability versus accuracy

1.3.1 Introduction

In this section two key considerations will be discussed that are important for the analysis of numerical integration algorithms: These are *accuracy* and *stability*. Both considerations are strongly linked to the calculation error in a single step and the accumulation of errors in multiple steps. The issue of accuracy is detailed in section 1.3.2 where various types of calculation errors are discussed. The issue of stability is discussed in section 1.3.3. Finally, the relationship between so-called “spurious” or “extraneous” eigenvalues and stability/accuracy will be introduced in section 1.3.4, since this phenomenon occurs in multistep integration methods.

1.3.2 Errors incurred in numerical integration

The error introduced by approximating the differential equation by the difference equation (1.8) is termed *local truncation error (LTE)*. It can be shown [14] that the LTE, at time t_k, is given by

$$T(x,h) \quad = \quad C_{n+1}h^{n+1}x^{(n+1)}(t_k) + \mathcal{O}(h^{n+2}), \tag{1.10}$$

where the coefficient C_{n+1} is called the *error constant* of the method (see chapter 3 for additional details), h is the steplength of the method, $x^{(n+1)}$ is the $(n+1)^{th}$ derivative of the function at time t_k, and n is the order of the integration algorithm.

Moreover, errors also occur during integration which are due to the deviation of the numerical solution from the exact theoretical solution of the difference equation. Included in this class of errors are *round-off errors* and convergence errors which are incurred since (1.8) is implicit and must be solved by Newton-iteration.

Another class of errors is called the *startup error* and it is introduced because a multistep method requires values of m earlier steps. Typically, a single-step method is used to generate m starting steps. Hence, an error is introduced into the calculation because of the numerical approximation of the initial steps.

In work done by Cellier [4], it was shown that the effect of the startup error being accumulated during a simulation is eliminated if the system being integrated is analytically stable and if the integration method in use is numerically stable.

Hence, the approximated initial conditions of an integration step don't excessively affect the result of the overall simulation of an analytically stable system, i.e. the initial conditions of a m-step method that are produced by a single-step algorithm, are not of large significance for the resulting accuracy of the overall simulation.

Cellier [4] notes two other classes of errors, the *parametric model error* and the *structural model error*. Parametric model errors occur because the model parameters are inaccurately estimated, whereas structural model errors are due to the fact that the model fails to describe important dynamics of the real system.

Thus, these types of errors have nothing to do with the accuracy and stability of the numerical integration technique since they are present in the model already.

The LTE and the startup errors accumulate to produce the *global truncation error (GTE)* or *accumulated error* [14]. The GTE can be expressed as

$$\epsilon_k \quad = \quad x(t_k) - x_k, \tag{1.11}$$

where $x(t_k)$ is the exact function value of the analytical solution, and x_k is the numerical approximation of $x(t_k)$.

Note that the global truncation error cannot be calculated as the sum of the local errors. It must be computed as a solution to the linear k-step difference equation

$$\rho(E)\epsilon_k + h\sigma(E)\lambda_k\epsilon_k + T_k + \eta_k \quad = \quad 0,$$

which will be derived in section 1.3.4.

While the LTE is proportional to h^{n+1} for a n^{th} order algorithm, Lambert shows in [14] that the GTE is roughly proportional to h^n for analytically stable systems. Therefore, one power of h has been lost during the process of accumulation.

1.3.3 Stability of numerical methods

Dahlquist states in [7] that:

Definition 1.3.1 *A method is said to be* A-stable*, if the values $\hat{h} = h\lambda$ have negative real parts, where* h *is the steplength of the applied numerical method and λ denotes the (complex) eigenvalue of the test-system $\dot{x} = \lambda x$.*

The highest order of an A-stable linear multistep method is two [7]. The smallest truncation error is obtained by using the trapezoidal rule.

In order to be able to apply the definition of A-stability to higher order linear multistep techniques, the requirements of A-stability have to be relaxed:

The first step is to only regard a wedge in the left half side of the complex λh-plane. This motivates the definition [20]:

Definition 1.3.2 *A method is said to be* A(α)-stable*,* $\alpha \in (0, \pi/2)$ *if*

$$\mathcal{R}_A \supseteq \{\hat{h}| - \alpha < \pi - arg\{\hat{h}\} < \alpha\},$$

it is said to be A(0)-stable *if it is A(α)-stable for some* $\alpha \in (0, \pi/2)$*, where* $\mathcal{R}_A$ *is the region of absolute stability, and* $\hat{h} = h\lambda$*.*

Gear proposes in [10] an alternative way of relaxing the requirements of A-stability by using Cartesian rather than polar coordinates:

Definition 1.3.3 *A method is said to be* stiffly stable *if* $\mathcal{R}_A \supseteq \mathcal{R}_1 \cup \mathcal{R}_2$*, where* $\mathcal{R}_1 = \{\hat{h}|Re\{\hat{h}\} < -a\}$ *and* $\mathcal{R}_2 = \{\hat{h}| - a \leq Re\{\hat{h}\} < 0, -c \leq Im\{\hat{h}\} \leq c\}$*,* a *and* c *are positive real numbers and* $\hat{h} = h\lambda$*.*

It is characteristic for stiffly stable systems to have eigenvalues which lie far left in the left half plane. They represent the fast transients of the system. In order to eliminate these transients a large damping is required as the real part of the eigenvalues goes to $-\infty$. This results in another definition [2],[8]:

Definition 1.3.4 *A one-step method is said to be* L-stable *if it is A-stable and, in addition, when applied to the scalar test equation* $\dot{x} = \lambda x$*,* λ *a complex constant with* $Re\{\lambda\} < 0$*, it yields* $x_{k+1} = R(h\lambda)x_k$*, where* $|R(h\lambda)| \to 0$ *as* $Re\{h\lambda\} \to -\infty$*.*

Note that L-stable methods mustn't be applied to unstable systems since the unstable behavior would be removed such that the systems would appear stable [14].

Figure 1.3 illustrates the given definitions of stability:

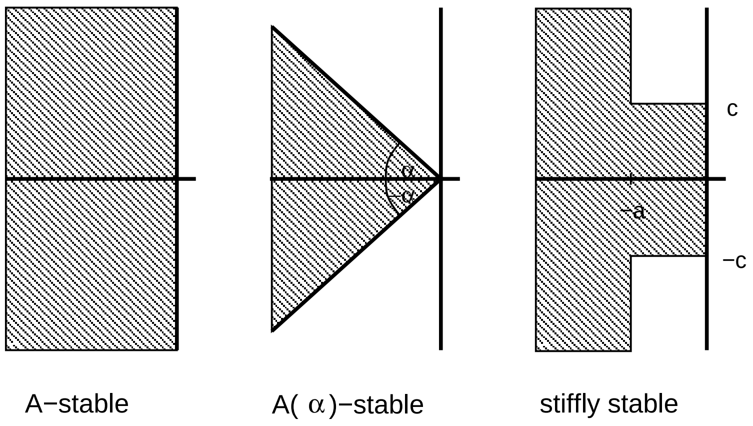


Figure 1.3: Stability definitions for numerical integration methods.

1.3.4 Spurious eigenvalues

Given the linear multistep method

$$x_{k+1} = \sum_{i=0}^{m-1} a_i x_{k-i} + \sum_{i=-1}^{m-1} b_i h f_{k-i}. \tag{1.12}$$

The generating polynomials can be introduced [15]

$$\rho(\xi) = -\xi^m + a_0\xi^{m-1} + \ldots + a_{m-1} \tag{1.13}$$

$$\sigma(\xi) = b_{-1}\xi^m + b_0\xi^{m-1} + \ldots + b_{m-1} \tag{1.14}$$

along with the shift operator $\mathcal{E}$,

$$\mathcal{E}^m x_k = x_{k+m}, \tag{1.15}$$

to reformulate equation (1.12) more compactly as

$$\rho(\mathcal{E})x_{k-m+1} + h\sigma(\mathcal{E})f_{k-m+1} = 0. \tag{1.16}$$

The true solution, $x(t_k)$, is substituted into equation (1.16) which results in

$$\rho(\mathcal{E})x(t_k) + h\sigma(\mathcal{E})f\left(t_k, x(t_k)\right) - T_k = 0, \tag{1.17}$$

where $T_k = T(t_k, h)$.

Note that the index of x has been changed from $k-m+1$ to k for convenience.

For the numerical solution the relationship

$$\rho(\mathcal{E})x_k + h\sigma(\mathcal{E})f(t_k, x_k) + \eta_k \quad = \quad 0 \tag{1.18}$$

holds, where η_k represents the error which results from not having solved the difference equation exactly.

Subtracting equation (1.17) from equation (1.18) and then applying the mean-value theorem

$$f(t_k, x_k) - f\left(t_k, x(t_k)\right) \quad = \quad f_{\bar{x}}(t_k, \bar{x})\left(x_k - x(t_k)\right), \tag{1.19}$$

where $x_k \leq \bar{x} \leq x(t_k)$, denoting $f_{\bar{x}}(t_k, \bar{x})$ as λ_k and introducing the accumulated error

$$\epsilon_k \quad = \quad x_k - x(t_k), \tag{1.20}$$

the m-step difference equation of the accumulated error ϵ_k

$$\rho(\mathcal{E})\epsilon_k + h\sigma(\mathcal{E})\lambda_k\epsilon_k + T_k + \eta_k \quad = \quad 0. \tag{1.21}$$

is obtained. Assuming that T_k, λ_k and η_k are constants, equation (1.21) yields

$$\rho(\mathcal{E})\epsilon_k + h\lambda\sigma(\mathcal{E})\epsilon_k + T + \eta \quad = \quad 0. \tag{1.22}$$

Therefore, the accumulated error ϵ_k obeys a linear, inhomogeneous, m-step difference equation with constant coefficients. By solving the characteristic equation of (1.22),

$$\rho(\mu) + h\lambda\sigma(\mu) \quad = \quad 0, \tag{1.23}$$

the characteristic roots $\mu_i, i = 1, 2, \ldots m$ are obtained.

When equation (1.22) is solved and the constant particular solution ϵ_{kp} is ignored, only the homogeneous solution ϵ_{kh} remains as shown in equation (1.24).

$$\begin{aligned} \epsilon_k &= \epsilon_{kh} + \epsilon_{kp} = \\ &\approx \epsilon_{kh} = \\ &= c_1\mu_1^k + c_2\mu_2^k + \ldots + c_m\mu_m^k. \end{aligned} \tag{1.24}$$

Since the numerical solution x_k obeys the same difference equation as the accumulated error ϵ_k, x_k can be expressed as

$$x_k = d_1\mu_1^k + d_2\mu_2^k + \ldots + d_m\mu_m^k. \tag{1.25}$$

The root μ_1, which is called the "*principal root*," approximates the Taylor Series expansion of the true solution. This approximation has a truncation error which corresponds to the order of the method.

The other $m-1$ roots are termed "*spurious*," "*parasitic*," or "*extraneous*" roots or eigenvalues.

Lapidus [10] shows that a multistep method, given by (1.12), is absolutely stable if, for $h < h_0$, where h_0 is a real constant, the extraneous solutions in (1.25) vanish as $k \to \infty$ [15].

Alternatively, the method is absolutely stable for those values of $h\lambda$ where both the principal root and the spurious roots are within the unit circle.

The spurious eigenvalues don't exist in the original system and have been introduced by the multistep algorithm, that substitutes a first-order differential equation by a m^{th} order difference equation. Since they bear no connection to the exact solution, they can cause numerical instability [15]. The impact of the extraneous eigenvalues on the stability properties of a multistep method can also be observed when regarding the stability domain of the method (see chap. 3).

1.4 Stiff systems

Many physical systems give rise to ordinary differential equations which have eigenvalues that vary greatly in magnitude. For instance, such situations can arise in studies of chemical kinetics, network analysis and simulation, CAD techniques, and the Method-of-Lines solution to parabolic partial differential equations.
Practical problems that exhibit such properties include the attitude control system of a rocket [18], and switched-mode power supplies [19].

An illustrating example of a stiff system (taken from [9]) shall now be discussed.
The analytical solution of the linear system $\underline{\dot{x}} = A\underline{x}$ with the system-matrix

$$A = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix}$$

and the initial conditions $x_1(0) = x_2(0) = 1$ is

$$x_1(t) = 4e^{-t} - 3e^{-1000t} \tag{1.26}$$

$$x_2(t) = -2e^{-t} + 3e^{-1000t}. \tag{1.27}$$

A plot of (1.26) and (1.27) is shown in figure 1.4.

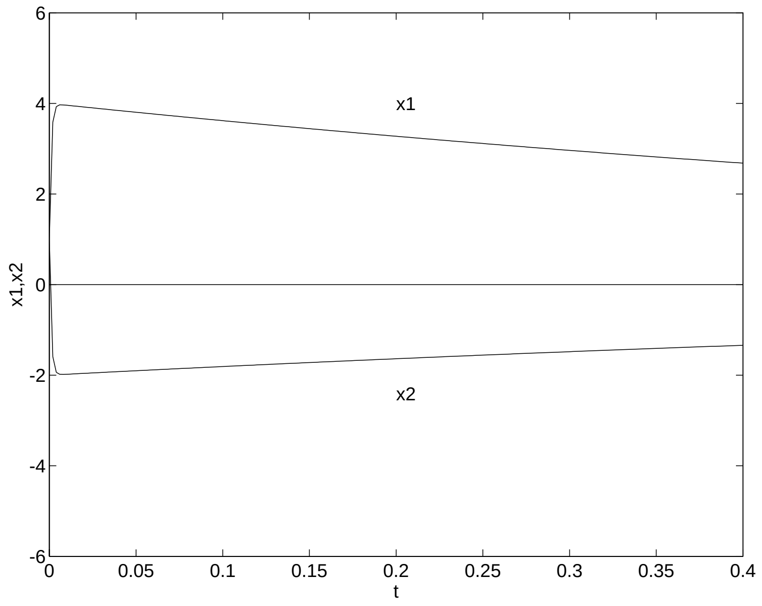


Figure 1.4: Simulation of a stiff linear system.

After a short time the solution can be closely approximated by the dominant terms as

$$x_1(t) = 4e^{-t} \tag{1.28}$$

$$x_2(t) = -2e^{-t}, \tag{1.29}$$

since the fast decaying component vanished. Therefore, an informal definition of a stiff problem is one in which the solution components of interest are slowly varying but solutions with rapidly changing components are possible [18].

Lambert [14] gives, among others, one essential definition of stiffness in his book:

Definition 1.4.1 (A system is called stiff if a) *numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a steplength which is excessively small in relation to the smoothness of the exact solution in that interval.*

Using this definition, an algorithm with finite region of absolute stability (such as explicit integration methods) used to integrate a stiff system has to apply a very small steplength which gives rise to long and expensive simulation runs. This is the real problem when integrating stiff ordinary differential equations.

The statement made above still doesn't define the term stiffness exactly because of the fuzzy expression "excessively small." Therefore, Cellier [4] further specifies the stiffness of a system in the following manner:

Definition 1.4.2 *A system is called stiff if, when integrated with any explicit RK*n *algorithm and a local error tolerance of* 10^{-n}*, the step-size of the algorithm is forced down to below a value indicated by the local error estimate due to constraints imposed on it by the limited size of the numerically stable region.*

This is probably the most exact definition of stiffness available. However, it still has an inherent drawback, namely that a system may be regarded as stiff when integrated with one RKn whereas it is not stiff when integrated with another [4].

Although explicit methods — like the explicit RKn algorithms — may be good for checking the stiffness of a system, they are not capable of integrating stiff systems because their stability domain in the left half plane is too small.

Dahlquist [14] states that "*an explicit linear multistep method cannot be A-stable*" and "*the order of an A-stable linear multistep method cannot exceed two.*" These statements are referred to as the "*second Dahlquist barrier.*" It can also be noted that an "*explicit method cannot be A(0)-stable*" [16],[20].

Because of these properties of explicit methods they don't lend themselves to the integration of stiff systems. Those should be integrated by a method which is at least $A(\alpha)$-stable. If the closed instability domain doesn't intersect with the negative real axis of the complex (λh)-plane, and if the system being integrated is stable and only has real eigenvalues, then any steplength may be used to scale the eigenvalues of the system without fear of causing numerical instability. In this case the steplength is not restricted by stability requirements, but instead by accuracy requirements. Since explicit methods don't have this feature, implicit techniques have to be used to simulate stiff systems [14]. However, a Newton iteration is required to solve the implicit algebraic equation, since fixed-point iteration again destroys the stability properties of the stiffly stable method [4]. This causes higher computation cost.

CHAPTER 2

Derivation of RBDF by regression

2.1 Introduction

In chapter 1 it has been shown that linear m-step integration algorithms of order n use p ($p \geq n + 1$) function values or derivatives to calculate the unknown value x_{k+1}. As a consequence, numerical integration by a multistep method can be considered as an interpolation procedure where a higher-order interpolation (extrapolation) polynomial is used to approximate a function g through p given data points and calculate the unknown point which is represented by x_{k+1}. Note that g is not identical with the function x being integrated by the multistep technique. It is instead formed by both x and its derivative f at special time instants.

If there are $p = n + 1$ (n = accuracy order of the integration method) distinct data points, then the interpolation problem has a unique solution and the resulting polynomial doesn't only approximate the function through the given data points but it interpolates the points that it passes through [6], [12].

An interesting question arises about using more data points, that is, $p > n+1$. In practice, this means that the multistep integration algorithm applies $p > n+1$ earlier function values or derivatives to calculate the unknown function value x_{k+1} with the accuracy order of n.

Thus, the order is not increased by using more data points. Instead, the interpolation results in an n^{th} order method which might have better properties because it uses more information.

As far as the interpolation problem is concerned, having more data points necessarily means that we are confronted with an overdetermined system which may be used to attain two different types of smoothing [6]:

1. A reduction of the effect of random errors in the values of the function.

2. A smoother shape between the net points (even when the function values are perfect).

The solution to this overdetermined system using a regression approach will be discussed in the next section.

2.2 Formulation and solution of the regression problem

An n^{th}-order multistep algorithm is defined through an n^{th}-order polynomial fitted through p points which can be either function values or derivatives of the function to be integrated.

By introducing the auxiliary variable

$$s = \frac{t - t_k}{h}, \tag{2.1}$$

such that $s = 1.0$ corresponds to $t = t_{k+1}$ and $s = 0.0$ corresponds to $t = t_k$, the interpolating n^{th}-order polynomial can be written in the form

$$p(s) = a_0 + a_1 s + a_2 s^2 + \ldots a_n s^n, \tag{2.2}$$

where the coefficients a_i are unknown.

The time derivative of equation (2.2) is given by

$$h\dot{p}(s) \quad = \quad a_1 + 2a_2 s + 3a_3 s^2 + \ldots + na_n s^{n-1}. \tag{2.3}$$

Since the polynomial p(s) interpolates the function x in the data points $x_{k-i}, i = 0, 1, \ldots, m$, relation (2.2) results in

$$\begin{array}{rcccl} p(0) & = & a_0 & = & x_k \\ p(-1) & = & a_0 - a_1 + a_2 - a_3 + \ldots + a_n(-1)^n & = & x_{k-1} \\ p(-2) & = & a_0 - 2a_1 + 4a_2 - 8a_3 + \ldots + a_n(-2)^n & = & x_{k-2} \\ \vdots & & \vdots & & \vdots \\ p(-m) & = & a_0 - ma_1 + m^2 a_2 + \ldots + a_n(-m)^n & = & x_{k-m}. \end{array} \tag{2.4}$$

Accordingly, it follows from (2.3) that

$$\begin{array}{rcccl} h\dot{p}(1) & = & a_1 + 2a_2 + 3a_3 + \ldots + na_n & = & hf_{k+1} \\ h\dot{p}(0) & = & a_1 & = & hf_k \\ h\dot{p}(-1) & = & a_1 - 2a_2 + 3a_3 + \ldots + na_n(-1)^{n-1} & = & hf_{k-1} \\ \vdots & & \vdots & & \vdots \\ h\dot{p}(-m) & = & a_1 - 2a_2 m + 3a_3 m^2 + \ldots + na_n(-m)^{n-1} & = & hf_{k-m}. \end{array} \tag{2.5}$$

Equations (2.4) and (2.5) can be combined such that they fit the matrix form

$$\begin{pmatrix} hf_{k+1} \\ x_k \\ hf_k \\ x_{k-1} \\ hf_{k-1} \\ \vdots \\ x_{k-m} \\ hf_{k-m} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 & 2 & 3 & \cdots & n \\ 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -1 & 1 & -1 & \cdots & (-1)^n \\ 0 & 1 & -2 & 3 & \cdots & n(-1)^{n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & -m & m^2 & & \cdots & (-m)^n \\ 0 & 1 & -2m & 3m^2 & \cdots & n(-m)^{n-1} \end{pmatrix}}_{H} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ \\ \\ \\ a_n \end{pmatrix}, \tag{2.6}$$

which can be abbreviated by

$$\underline{x} \;=\; H\underline{a}. \tag{2.7}$$

Hence, vector $\underline{x}$ has $2m+3$ elements for an m-step algorithm and contains all the information about the state vector and its derivative at the given data points. Vector $\underline{a}$ contains $n+1$ unknown coefficients (n = order of the integration algorithm) and H is the $[(2m+3)\times(n+1)]$ transformation matrix.

In the special case where $n+1 = 2m+3$, the matrix H will be quadratic and nonsingular. Therefore, H can be inverted and equation (2.7) has the unique solution $\underline{a} = H^{-1}\underline{x}$. However, the RBDF techniques derived in this thesis use more than $n+1$ data points such that the case $2m+3 > n+1$, has to be considered — note that $2m+3$ is the maximum number of data points. Consequently, equation (2.7) results in an overdetermined system. Since this system cannot be solved exactly any more, a vector $\underline{a}$ has to be found such that

$H\,\underline{a}$ is the "best" approximation to $\underline{x}$ [6]. The solution vector $\underline{a}$ is the least-squares solution of the overdetermined system.

Definition 2.2.1 *The vector $\underline{a}$ is defined as the vector which minimizes the Euclidean length of the residual vector, i.e. minimizes*

$$\|\underline{r}\|_2 \;=\; (\underline{r}^T\underline{r})^{1/2}, \qquad \underline{r} = H\underline{a}. \tag{2.8}$$

In [1] it is stated that when the columns of H are linearly independent, i.e. H has full rank ($\rho(H) = n+1$), then the matrix H^TH is nonsingular and can be inverted.

Hence, in order to solve system (2.7) it first has to be shown that the columns of matrix H are linearly independent. This is done by defining two generating row vectors p and q shown in equations (2.9) and (2.10).

$$\underline{p} \;=\; [\; 1 \quad -s \quad s^2 \quad \ldots \quad (-s)^n \;], \tag{2.9}$$

$$\underline{q} \;=\; [\; 0 \quad 1 \quad -2s \quad 3s^2 \quad \ldots \quad n(-s)^{n-1} \;]. \tag{2.10}$$

The rows of H that correspond to x_{k-s} are formed by the components of vector $\underline{p}$ and and the rows corresponding to hf_{k-s} are formed by the components of vector $\underline{q}$. The components of these two row vectors (2.9) and (2.10) are elements of polynomials. Such elements are linearly independent [3] and thus, all the column vectors of H, that are formed by the components of the generating row vectors $\underline{p}$ and $\underline{q}$ when varying the parameter s, are linearly independent, too, which was to be shown.

Therefore, matrix H can be inverted and it follows from (2.7) that

$$H^T\underline{x} \;=\; H^TH\underline{a} \tag{2.11}$$

or

$$\underline{a} \quad = \quad (H^T H)^{-1} H^T \underline{x}. \tag{2.12}$$

Dahlquist [6] proves that this is the unique solution which solves the overdetermined system (2.7) in a least-squares sense.

In the literature the matrix

$$H^+ \quad = \quad (H^T H)^{-1} H^T \tag{2.13}$$

is referred to as the *Penrose-Moore Pseudoinverse.*

By using (2.12) to calculate $\underline{a}$, the elements of $\underline{a}$ can be substituted into (2.2) to determine x_{k+1}.

$$x_{k+1} \quad = \quad p(1) \quad = \quad a_0 + a_1 + \ldots + a_n \tag{2.14}$$

Note that the coefficients a_i are not constants. Each of them represents a linear combination of the elements of the vector $\underline{x}$, given in (2.6) and (2.7).

2.3 Development of an algorithm for the search of RBDF

In the preceding section it has been described how an integration algorithm can be derived if p data points are chosen out of the possible $2m + 3$ points.

Since m and p are free parameters, the search is (theoretically) not restricted, and to get an n^{th}-order integration algorithm, more data points may be used as long as $p \geq n + 1$ holds. In practice, the search is limited by efficiency considerations. Since the H-matrix becomes larger as m grows, the integration algorithm is likely to consume more execution time.

Thus there are two questions that have to be answered:

1. How many data points should be used?

2. Where should these data points lie to result in an integration algorithm with minimum stability properties?

In this context, having "minimum stability properties" means that the stability locus of an implicit method doesn't intersect with the negative real axis of the complex (λh)-plane (see chapter 3).

To answer the first question, multiple experiments have been performed with only a few data points. These experiments have shown that the probability of getting implicit algorithms with minimum stability properties shrinks as the number of employed data points grows. Hence, the search starts with $p_{min} = n + 2$ employed data points and it stops at $p_{max} = 2n$ data points. No decent RBDF algorithms have been found for values $p > 2n$.

Concerning the second question, a lot of calculations with a small number of data points have shown that most of the data points that yield decent integration algorithms lie within the time range $[t_{k-n-2}, t_{k+1}]$. The stability properties of the resulting RBDF algorithms worsen as the data points get farther away from the defined interval. For this reason the search is limited to the interval $[t_{k-n-2}, t_{k+1}]$.

This search for RBDF methods can be automated by testing all possible combinations of p data points within the range $[t_{k-n-2}, t_{k+1}]$ where p is incremented step by step from $n + 2$ to $2n$.

CHAPTER 3

Analysis of RBDF

3.1 Introduction

In this chapter some methods will be discussed to analyze integration methods, in particular RBDF.

These integration methods will be used in chapters 5 and 6 to assess the RBDF methods that will be derived.

The technique being discussed in the second section of this chapter is the analysis of the *stability domain* of the numerical integration method.

In the third section the *error constant* will be computed.

Another important method for investigating the quality of RBDF is the analysis of the *damping plot* which will be described in section four.

The *order star* will be introduced in the fifth section to make some additional statements about stability and accuracy. Finally, the *Bode plot* of the integration algorithm will be analyzed in section six to see how the algorithm behaves in the frequency domain.

3.2 Stability domain

3.2.1 Introduction

In chapter 1, some definitions of numerical stability are given. Chapter 1 also mentions that the stability properties of the integrator depend strongly on both the steplength used during the integration, and the eigenvalues of the system.

Before the numerical stability domain is determined, the analytical stability of the system being integrated can be determined using the following definition:

Definition 3.2.1 *The solution of the autonomous, time-invariant linear system*

$$\underline{\dot{x}} = A\underline{x} \tag{3.1}$$

with the initial conditions specified by $\underline{x}(t = t_0) = \underline{x}_0$ is called analytically stable *if all the eigenvalues of A have negative real parts.*

Figure 3.1 shows the domain of analytical stability in the λ-plane.

Suppose that the system (3.1) is analytically stable and that it is integrated by a linear

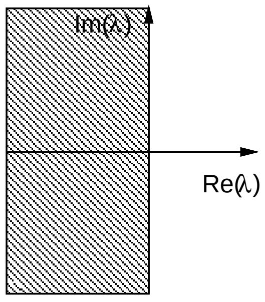


Figure 3.1: Domain of analytical stability

technique. Then, the domain of analytical stability changes and it becomes a function of the step-size h. Hence, the stability domain of a linear integration method (with fixed step-size h), when used to integrate the autonomous system (3.1) is defined as the region in the complex (λh)-plane where the eigenvalues of the equivalent discrete time system all lie within the unit circle. Note that all the eigenvalues have negative real parts. This is illustrated by means of the implicit *Backward Euler algorithm* expressed by:

$$\underline{x}_{k+1} = \underline{x}_k + h\underline{f}_{k+1}, \tag{3.2}$$

where $\underline{f}_{k+1} = \dot{\underline{x}}_{k+1}$.

Substituting equation (3.1) into (3.2), equation (3.2) results in

$$\underline{x}_{k+1} = \underline{x}_k + hA\underline{x}_{k+1},$$

or

$$\underline{x}_{k+1} = [I^{(n_s)} - Ah]^{-1}\underline{x}_k, \tag{3.3}$$

where n_s is the dimension of system (3.1) and $I^{(n_s)}$ is the ($n_s \times n_s$) identity matrix.

Consequently, the continuous linear system (3.1) has been converted into the equivalent discrete system (3.3) with the new system matrix

$$F = [I^{(n_s)} - Ah]^{-1}. \tag{3.4}$$

This discrete system is analytically stable if all of the eigenvalues of F are located within the unit circle [4].

Since matrix F depends on the steplength h, the eigenvalues of F depend on h, too. Therefore, the stability domain indeed depends on h. Figure 3.2 displays the stability region of the implicit Backward Euler integration algorithm. Figure 3.2 shows that a typical property of implicit methods is to have a region of instability in the right half plane,

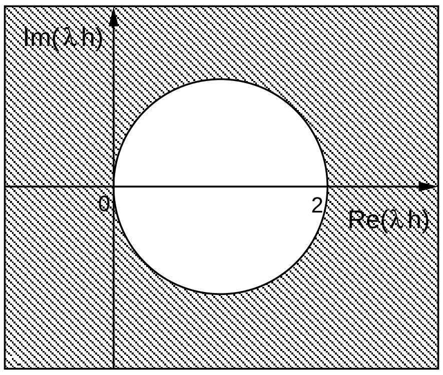


Figure 3.2: Stability domain of the BE-algorithm

close to the origin.

When integrating a stable system, whose eigenvalues are in the left half plane, the instability region shown in figure 3.2 shouldn't extend into the left half plane. However, this property, called *A-stability* in chapter 1, cannot be obtained by linear multistep methods of higher than second order.

At the very least an implicit integration algorithm must have an instability domain that doesn't include the origin. In other words, the stability locus, i.e. the border line of the stability domain, mustn't intersect with the negative real axis. Recall that this has been the main requirement for RBDF algorithms to be good candidates in chapter 2.

3.2.2 Stability domain of RBDF

In order to depict the stability domain, the F-matrix, the discrete time system that results from applying the RBDF method to system (3.1) is calculated.

The general RBDF algorithm is given by

$$\underline{x}_{k+1} = \sum_{i=0}^{m-1} a_i \underline{x}_{k-i} + \sum_{i=-1}^{m-1} b_i h \underline{f}_{k-i}. \tag{3.5}$$

By applying this algorithm to system (3.1), relation (3.5) can be rewritten in the form of

$$\underline{x}_{k+1} = b_{-1}Ah\underline{x}_{k+1} + a_0\underline{x}_k + b_0Ah\underline{x}_k + \ldots + a_{m-1}\underline{x}_{k-m+1} + b_{m-1}Ah\underline{x}_{k-m+1},$$

or

$$\begin{aligned}\underline{x}_{k+1} &= [I^{(n_s)} - b_{-1}Ah]^{-1}[(a_0I^{(n_s)} + b_0Ah)\underline{x}_k + (a_1I^{(n_s)} + b_1Ah)\underline{x}_{k-1} + \ldots + \\ &\quad +(a_{m-1}I^{(n_s)} + b_{m-1}Ah)\underline{x}_{k-m+1}],\end{aligned} \tag{3.6}$$

where n_s is the dimension of system (3.1) and $I^{(n_s)}$ is the $(n_s \times n_s)$ identity matrix.

Equation (3.6) is a m^{th}-order difference equation which can be transformed into m first order difference equations by applying the transformation

$$\begin{aligned}\underline{z}_1(t_k) &= \underline{x}(t_{k-m+1}) \\ \underline{z}_2(t_k) &= \underline{x}(t_{k-m+2}) \\ &\vdots \\ \underline{z}_m(t_k) &= \underline{x}(t_k).\end{aligned} \tag{3.7}$$

or

$$\begin{aligned}\underline{z}_1(t_{k+1}) &= \underline{x}(t_{k-m+2}) = \underline{z}_2(t_k) \\ &\vdots \\ \underline{z}_m(t_{k+1}) &= \underline{x}(t_{k+1}).\end{aligned} \tag{3.8}$$

By substituting $\underline{x}_{k+1}$ from equation (3.6) into (3.8), it follows that

$$\underline{z}(t_{k+1}) = F\underline{z}(t_k), \tag{3.9}$$

where the mn_s-vector $\underline{z}$ is given by

$$\underline{z}(t_i) = \begin{pmatrix} \underline{z}_1(t_i) \\ \underline{z}_2(t_i) \\ \vdots \\ \underline{z}_m(t_i) \end{pmatrix}, \qquad i = k,\ k+1, \tag{3.10}$$

and the $(mn_s \times mn_s)$-matrix F can be written as

$$F = \begin{pmatrix} O^{(n_s)} & I^{(n_s)} & O^{(n_s)} & \cdots & O^{(n_s)} \\ \vdots & O^{(n_s)} & I^{(n_s)} & \ddots & \vdots \\ & \vdots & O^{(n_s)} & \ddots & O^{(n_s)} \\ & & \vdots & \ddots & I^{(n_s)} \\ F_{m,1} & F_{m,2} & \cdots & & F_{m,m} \end{pmatrix}, \tag{3.11}$$

$$F_{m,1} = Q^{(n_s)}[a_{m-1}I^{(n_s)} + b_{m-1}Ah], \tag{3.12}$$

$$F_{m,2} = Q^{(n_s)}[a_{m-2}I^{(n_s)} + b_{m-2}Ah], \tag{3.13}$$

$$F_{m,m} = Q^{(n_s)}[a_0 I^{(n_s)} + b_0 Ah], \tag{3.14}$$

$$Q^{(n_s)} = [I^{(n_s)} - b_{-1}Ah]^{-1}. \tag{3.15}$$

(3.16)

As an example one of the RBDF methods that have been found, RBDF61, is considered:

$$\begin{aligned} x_{k+1} = {} & \frac{594}{1357}hf_{k+1} + \frac{977}{461}x_k - \frac{1612}{915}x_{k-1} + \frac{361}{943}x_{k-2} \\ & + \frac{1171}{1310}x_{k-3} - \frac{3199}{3212}x_{k-4} + \frac{257}{592}x_{k-5} - \frac{389}{5370}x_{k-6} \end{aligned} \tag{3.17}$$

Figure 3.3 displays the stability region of RBDF61.

There exist many different ways of assessing the stability domain of implicit methods. One criteria is the size of the stability region in the left half plane. Since most of the systems being integrated are stable, this region should be as large as possible. Therefore, the instability domain shouldn't extend too far into the left half plane.

By considering the definition of *A(α)*-stability given in chapter 1, the angle α should be large.

For stiff systems the definition of *stiffly stable* algorithms might be even more useful. This

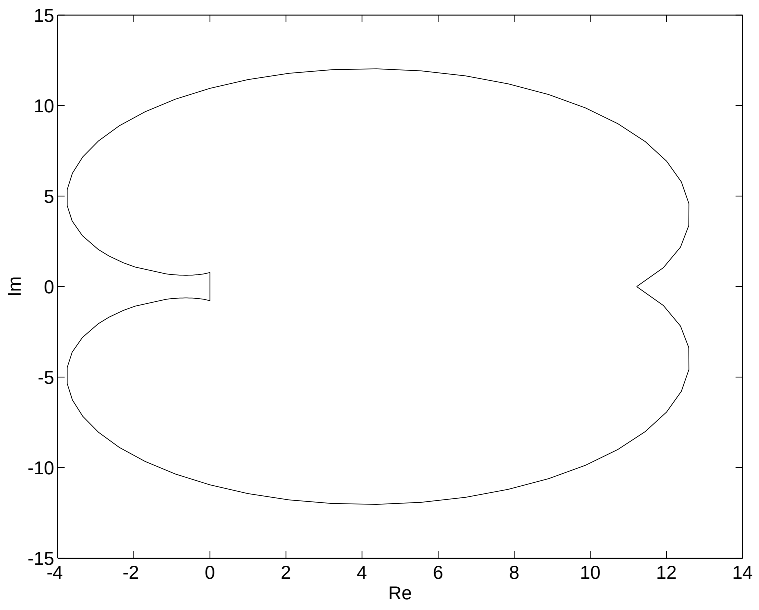


Figure 3.3: Stability region of RBDF61

definition of stability is shown once again in figure 3.4. In this case, the parameter a

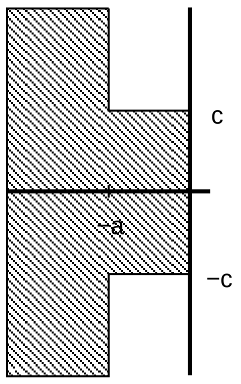


Figure 3.4: Stiffly stable system

should be as small as possible and c should be as large as possible.

It can also be observed that the *spurious eigenvalues* (discussed in chapter 1) may affect the shape of the stability region. The stronger this impact is the less smooth the stability locus becomes. As an example, figure 3.4 shows the stability domain of RBDF69, whose formula is given by:

$$\begin{aligned} x_{k+1} &= \frac{232}{533}hf_{k+1} + \frac{2734}{1241}x_k - \frac{414}{227}x_{k-1} - \frac{211}{1164}hf_{k-1} \\ &\quad + \frac{1007}{1440}x_{k-2} - \frac{985}{2009}x_{k-5} + \frac{871}{5716}hf_{k-5} + \frac{471}{1144}x_{k-6} + \frac{191}{1020}hf_{k-6} \end{aligned} \tag{3.18}$$

Figure 3.5 illustrates that the stability locus of (3.18) has some sharp bends which are

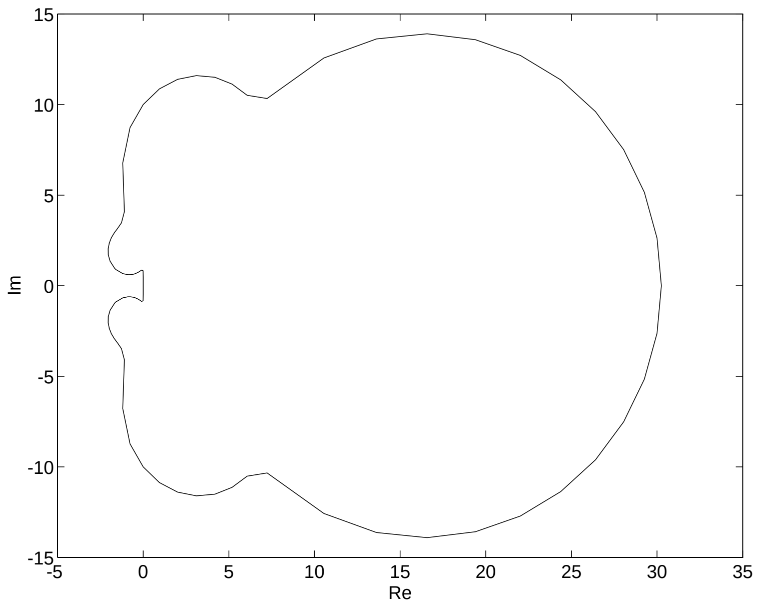


Figure 3.5: Stability locus of RBDF69

caused by the spurious eigenvalues described in chapter 1. As noted previously, they strongly disturb the behavior of integration algorithms in terms of stability and accuracy.

3.3 Error constant

In [14], the *Linear Difference Operator* $\mathcal{L}$, associated with the linear m-step method given in standard form as

$$\sum_{i=0}^{m} \alpha_i x_{k+i} = h \sum_{i=0}^{m} \beta_i f_{k+i}, \tag{3.19}$$

is defined as

$$\mathcal{L}[z(t);h] \;=\; \sum_{i=0}^{m}[\alpha_i z(t+ih) - h\beta_i \dot{z}(t+ih)], \qquad z(t) \in \mathcal{C}^1[a,b]. \tag{3.20}$$

If $z(t)$ is infinitely differentiable, then $z(t+ih)$ and $\dot{z}(t+ih)$ can be developed in a Taylor series around t as shown in the following equation

$$\mathcal{L}[z(t);h] \;=\; C_0 z(t) + C_1 h z^{(1)}(t) + \ldots + C_q h^q z^{(q)}(t) + \ldots, \tag{3.21}$$

where $z^{(q)}(t)$ is the q^{th} time derivative of $z(t)/$.

Now the following statement can be made [14]:

Definition 3.3.1 *The linear multistep method (3.19) and the associated difference operator $\mathcal{L}$ are said to be of order* n *if, in (3.21),*

$$C_0 = C_1 = \ldots = C_n = 0; \quad C_{n+1} \neq 0.$$

For the constants C_i the formulae

$$\begin{aligned} C_0 &= \sum_{i=0}^{m}\alpha_i \\ C_1 &= \sum_{i=0}^{m}(i\alpha_i - \beta_i) \\ &\vdots \\ C_q &= \sum_{i=0}^{m}\left(\tfrac{1}{q!}i^q\alpha_i - \tfrac{1}{(q-1)!}i^{q-1}\beta_i\right), \qquad q = 2,3,\ldots \end{aligned} \tag{3.22}$$

hold. Using the equations shown in (3.22), the error constant can be defined as:

Definition 3.3.2 *The linear multistep method of order* n *is said to have the* error constant *C_{n+1} given by (3.22).*

In the first chapter it is shown that the *local truncation error* $T(x,h)$ is strongly related to this error constant by

$$T(x,h) \quad = \quad C_{n+1}h^{n+1}x^{(n+1)}(t_k) + \mathcal{O}(h^{n+2}). \tag{3.23}$$

Hence, a multistep method may integrate more accurately than others if the absolute value of its error constant is smaller.

3.4 Damping plot

Cellier [4] introduces the *damping plot* as yet another tool to describe the accuracy of an integration algorithm. In order to derive the damping plot, the standard linear system $\dot{\underline{x}} = A\underline{x}, \quad \underline{x}(t_0) = \underline{x}_0$ is considered which has the analytical solution $\underline{x}_{anal} = e^{A(t-t_0)}\underline{x}_0$. This solution is true for any value of $\underline{x}_0$, and any value of t. Therefore, if the time instant $t = t_{k+1}$ is chosen as well as the initial conditions $t_0 = t_k$, and $\underline{x}_0 = \underline{x}_k$, the analytical result becomes

$$\underline{x}_{k+1} \quad = \quad e^{Ah}\underline{x}_k. \tag{3.24}$$

This discrete system has the analytical F-matrix $F_{anal} = e^{Ah}$ and the eigenvalues

$$\lambda_{dis} \quad = \quad eig\{F_{anal}\} \quad = \quad e^{eig\{A\}h} \quad = \quad e^{\lambda_i h}, \quad i = 1,\ldots,n_s. \tag{3.25}$$

The damping of an analytically stable system $\dot{\underline{x}} = A\underline{x}$ is defined as the smallest magnitude value of the real parts of its eigenvalues, or

$$\sigma \quad = \quad \min_i(|\sigma_i|) \quad = \quad \min_i(|Re\{\lambda_i\}|). \tag{3.26}$$

Since the eigenvalues λ_i are complex, i.e. $\lambda_i = -\sigma_i + j\omega_i$, the eigenvalue λ_{dis} can be rewritten in the form

$$\lambda_{dis} = e^{\lambda_i h} = e^{-\sigma_i h} e^{j\omega_i h}. \tag{3.27}$$

The damping of the discrete system (3.24) is defined as the largest magnitude value of the eigenvalues λ_{dis}. Therefore, it follows from equation (3.27) that the damping of the discrete system can be expressed as

$$\sigma_{dis} = \max_i (e^{-\sigma_i h}). \tag{3.28}$$

Thus, in the case of the continuous system $\underline{\dot{x}} = A\underline{x}$, the damping corresponds to the smallest distance of the eigenvalues from the imaginary axis in the λ-plane, whereas in the case of the corresponding discrete system $\underline{x}_{k+1} = e^{Ah}\underline{x}_k$, the damping refers to the largest distance of the eigenvalues from the origin in the $e^{\lambda h}$-plane. This is commonly known as the *z-domain*, where $z = e^{\lambda h}$. Cellier [4] introduces the discrete damping as

$$\sigma_d = h\sigma. \tag{3.29}$$

The relationship between σ_d and F_{anal} can be derived from the equations (3.25), (3.27) and (3.28):

$$\sigma_d = -log(\max_i |eig\{F_{anal}\}|). \tag{3.30}$$

In order to come up with an expression for the numerical damping, i.e. the damping of the numerical integration algorithm applied to the system $\underline{\dot{x}} = A\underline{x}$, the analytical F_{anal}-matrix needs to be approximated by the matrix F_{num} of the numerical integration method.

Thus, by substituting F_{anal} by F_{num}, equation (3.30) yields

$$\hat{\sigma}_d = -log(\max_i |eig\{F_{num}\}|), \tag{3.31}$$

where $\hat{\sigma}_d$ is the discrete damping of the numerical integration algorithm.

Cellier [4] also defines the damping plot as the curve of the function

$$\hat{\sigma}_d = \hat{\sigma}_d(\sigma_d). \tag{3.32}$$

This curve represents the relationship between the numerical and analytical damping. In particular, it reveals where these two damping values are approximately the same, and where they differ. As an example, figure 3.6 shows the damping plot of the BDF6 method. It can be seen in figure 3.6 that as soon as the analytical damping σ_d becomes larger than

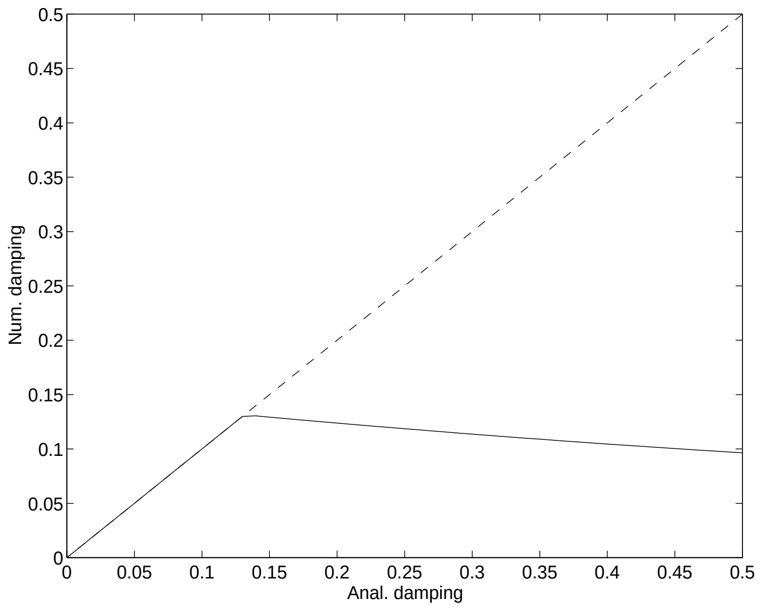


Figure 3.6: Damping plot of BDF6

$\sigma_{d,crit} = 0.13$, σ_d and the numerical damping $\hat{\sigma}_d$ start to diverge from each other. The spurious eigenvalues mentioned in chapter 1 are responsible for this behavior [4].

The value of the damping plot is that it can be used to assess an integration algorithm. If an integration method has a larger value $\sigma_{d,crit}$ than another integration algorithm, then the first method may integrate accurately in a larger range. The range of accurate integration is often referred to as the *asymptotic region* in the literature.

To illustrate the relationship between the damping plot and the stability domain of the integration method (sec. 3.2.2), figure 3.7 shows the modified damping plot for the BDF6 technique where the negative numerical damping $-\hat{\sigma}_d$ is a function of the negative analytical damping $-\sigma_d$. It can be seen in figure 3.7 that the negative numerical damping $-\hat{\sigma}_d$

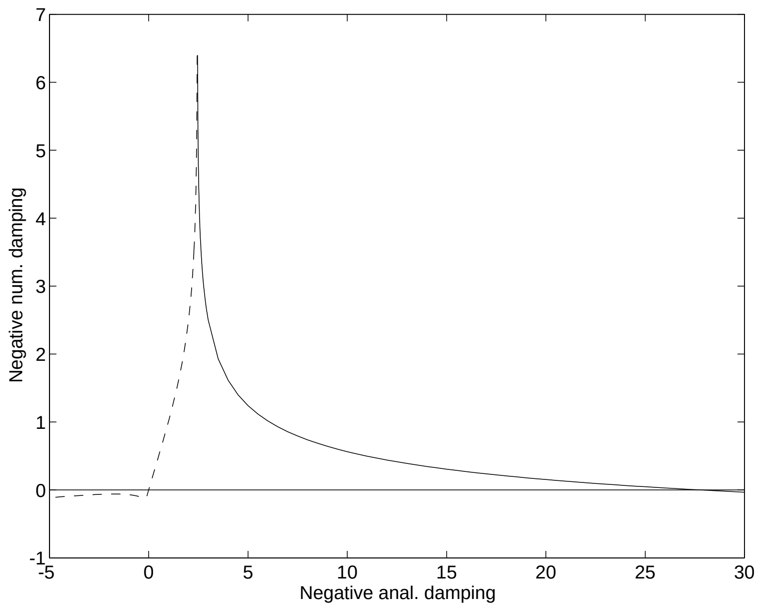


Figure 3.7: Modified damping plot of the BDF6 method

is zero at $-\sigma_d = 0$ and $-\sigma_d = 27.72$. These points can also be found in figure 3.8 which displays the stability domain of the BDF6 technique. There, these points are given by the intersection points of the stability locus with the real axis.

In order to state anything about the behavior of the numerical damping $\hat{\sigma}_d$ as $\sigma_d \to \infty$, Cellier [4] proposes to produce a *logarithmic damping plot.* Figure 3.9 displays the logarithmic damping plot of the BDF6 method.

Using this technique, if a numerical integration method is at least A(α)-stable and if $\hat{\sigma}_d$

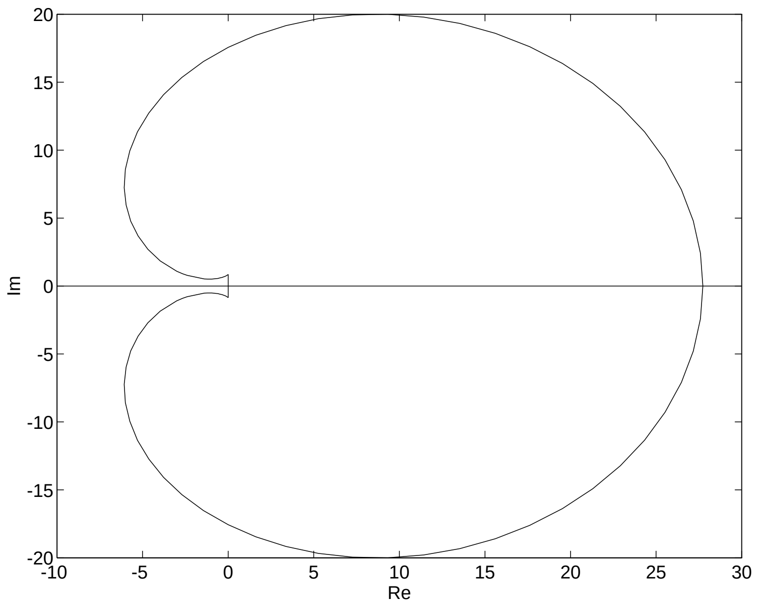


Figure 3.8: Stability domain of BDF6

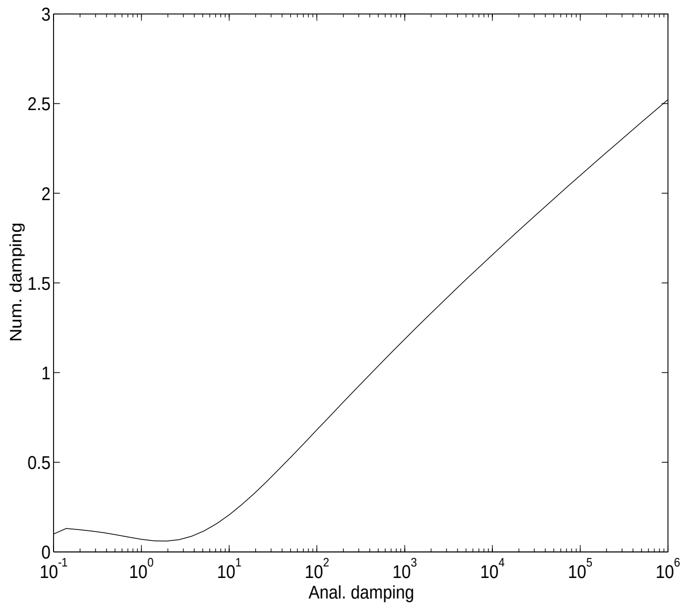


Figure 3.9: Logarithmic damping plot of BDF6

goes to infinity as σ_d goes to infinity, then it lends itself to integrate a stiff system because it is capable of damping out the fast transients. Note that it is not required that the numerical method integrating a stiff system be L-stable since L-stability is only possible if the integration technique is A-stable. For instance, the A(α)-stable BDF6 method can be used to integrate a stiff system although it it not L-stable.

3.5 Order star

Before we derive the order star of a numerical method, we make the following definition:

Definition 3.5.1 *A function f is called "essentially analytic" if it is analytical in the complex plane except at a finite set of singularities.*

Assuming that an essentially analytic function f is approximated by a rational function R, the rational function $\rho(z)$ can be introduced:

$$\rho(z) \quad = \quad \frac{R(z)}{f(z)}, \quad z \in \mathcal{C}. \tag{3.33}$$

In [13], the *order star* (of the first kind) is defined as the locus where $\rho(z) = 1$. Thus, a damping order star can be created by defining z as $z = \lambda h$ and $\rho(\lambda h)$ as

$$\rho(\lambda h) \quad = \quad \frac{\hat{\sigma}_d(\lambda h)}{\sigma_d(\lambda h)}, \tag{3.34}$$

where the numerical damping $\hat{\sigma}_d$ is an approximation for the analytical damping σ_d. In other words, the damping order star is the locus of the points in the complex (λh)-plane where $\hat{\sigma}_d(\lambda h) = \sigma_d$. In figure 3.10 the order star for the BDF6 algorithm is depicted.

Since the damping order star is produced by integrating a system with two complex eigenvalues, the function $\rho(\lambda h)$ is complex, too. Recall that the damping plot introduced in

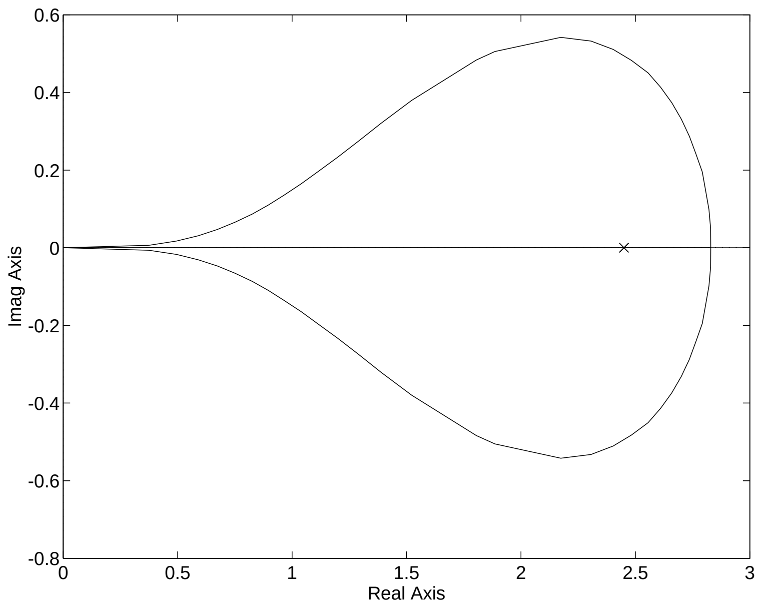


Figure 3.10: Order star of the BDF6 method

the preceding section is a real function because only systems with real eigenvalues are considered. Therefore, the real axis in the plot for the damping order star corresponds to the damping plot. To illustrate this, a modified damping plot for the BDF6 method is displayed in figure 3.11 where the negative numerical damping $-\hat{\sigma}_d$ is a function of the negative analytical damping $-\sigma_d$. Figure 3.11 shows that $-\hat{\sigma}_d$ and $-\sigma_d$ are equal when $-\sigma_d = 0$ or $-\sigma_d = 2.82$. These values can also be found in figure 3.10 by considering the points where the order star intersects with the real axis.

Furthermore, it can be seen in figure 3.11 that

$$-\hat{\sigma}_d(\lambda h) \quad \rightarrow \quad \infty \tag{3.35}$$

holds for $-\sigma_d = 2.45$. In figure 3.10 this point on the real axis is marked by a cross and it is referred to as the "*pole of the order star*".

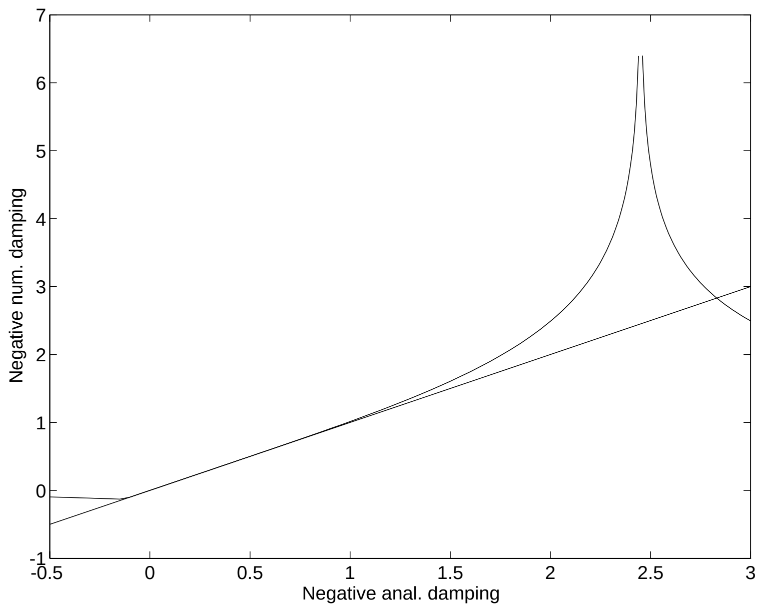


Figure 3.11: Modified damping plot of the BDF6 method

An order star may also have *zero points* where

$$\hat{\sigma}_d(z) \quad = \quad 0. \tag{3.36}$$

For a scalar system $\dot{x} = \lambda x$ the following theorems can be formulated:

Theorem 3.5.1 *The only pole of the order star of the* m*-step integration algorithm*

$$x_{k+1} \quad = \quad \sum_{i=0}^{m-1} a_i x_{k-i} + \sum_{i=-1}^{m-1} b_i h f_{k-i} \tag{3.37}$$

is that point in the complex (λh)-plane for which

$$\lambda h \quad = \quad \frac{1}{b_{-1}} \tag{3.38}$$

holds.

Proof: The system matrix F of the discrete system (3.37) can be expressed by

$$F = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ & & & \ddots & 0 \\ 0 & \cdots & & 0 & 1 \\ p_{m-1} & p_{m-2} & p_{m-3} & \cdots & p_0 \end{pmatrix}, \tag{3.39}$$

where

$$p_i = \frac{a_i + b_i q}{1 - b_{-1} q}, \qquad q = \lambda h. \tag{3.40}$$

The eigenvalues of F are calculated by solving the characteristic equation:

$$\begin{aligned} |\lambda I - F| &= \begin{vmatrix} \lambda & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & & & 0 \\ 0 & \cdots & 0 & \lambda & -1 \\ -p_{m-1} & -p_{m-2} & \cdots & -p_1 & \lambda - p_0 \end{vmatrix} \\ &= \lambda(\lambda A_{m-3} + B_{m-2}) + B_{m-1} \\ &\overset{!}{=} 0, \end{aligned} \tag{3.41}$$

where

$$A_i = \begin{vmatrix} \lambda & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & & & 0 \\ 0 & \cdots & 0 & \lambda & -1 \\ -p_i & \cdots & & -p_1 & \lambda - p_0 \end{vmatrix}, \tag{3.42}$$

$$B_i = \begin{vmatrix} 0 & -1 & 0 & \cdots & 0 \\ 0 & \lambda & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & & 0 \\ 0 & \cdots & 0 & \lambda & -1 \\ -p_i & -p_{i-2} & \cdots & -p_1 & \lambda - p_0 \end{vmatrix}. \tag{3.43}$$

The determinants A_i and B_i may be written as

$$A_i = \lambda A_{i-1} + B_i, \tag{3.44}$$

$$\begin{aligned} B_i &= \begin{vmatrix} 0 & -1 & 0 & \cdots & & 0 \\ 0 & \lambda & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & & & 0 \\ 0 & \cdots & 0 & & \lambda & -1 \\ -p_i & -p_{i-3} & -p_{i-4} & \cdots & -p_1 & \lambda - p_0 \end{vmatrix} \\ &= \ldots \\ &= -p_i. \end{aligned} \tag{3.45}$$

Hence it follows from (3.44) that

$$\begin{aligned} A_i &= \lambda A_{i-1} + B_i \\ &= \lambda(\lambda A_{i-2} + B_{i-1}) + B_i \\ &\vdots \\ &= \lambda^i A_0 + \lambda^{i-1} B_1 + \lambda^{i-2} B_2 + \ldots + \lambda B_{i-1} + B_i \end{aligned} \tag{3.46}$$

By applying (3.45), equation (3.46) can be formulated as

$$A_i = \lambda^i (A_0 - \sum_{j=1}^{i} p_j \lambda^{-j}). \tag{3.47}$$

Since $A_0 = \lambda - p_0$, equation (3.47) yields

$$A_i = \lambda^i(\lambda - \sum_{j=0}^{i} p_j \lambda^{-j}). \tag{3.48}$$

With (3.45) and (3.48), equation (3.41) results in

$$\begin{aligned} |\lambda I - F| &= \lambda[\lambda^{m-2}(\lambda - \sum_{j=0}^{m-3} p_j \lambda^{-j}) - p_{m-2}] - p_{m-1} \\ &= \lambda^{m-1}(\lambda - \sum_{j=0}^{m-3} p_j \lambda^{-j}) - \lambda p_{m-2} - p_{m-1} \\ &\overset{!}{=} 0, \qquad m \geq 3. \end{aligned} \tag{3.49}$$

Because of (3.31) and (3.35), a pole requires that the largest eigenvalue λ_{max} of F goes to infinity. Therefore, relation (3.49) can only be satisfied if $p_i \to \infty$ holds for $i = 0$ or $i = m-2$ or $i = m-1$.

From (3.40) it follows that this is identical with the requirement that $q = \frac{1}{b_{-1}}$. Since this is the only pole, the proof is complete.

Theorem 3.5.2 *For the zeros of the order star of the* m*-step integration method (3.37) the relation*

$$\max_i |\lambda_i| = 1, \qquad i = 0, 1, \ldots, m-1, \tag{3.50}$$

holds where λ_i are the eigenvalues of F *and can be calculated with (3.49).*

Proof: This theorem follows directly from (3.31) and (3.36).

Theorem 3.5.2 means that all the eigenvalues have to be within the unit circle in the (λh)-plane while it is required that at least one eigenvalue lies exactly on the unit circle.

Hence, the more steps the multistep method uses, the larger the F-matrix becomes, and the less probable it is that this condition can be satisfied.

Note that the unit circle represents the stability locus of a discrete system. Thus, theorem 3.5.2 shows that the zeros of the damping order star correspond to points on the stability locus.

To be able to use the order star of an integration method to compare one method with another, the region around the origin is of particular interest. Basically, the order star, that is the locus of all those points where the error

$$\Delta\sigma_d \quad = \hat{\sigma}_d - \sigma_d \tag{3.51}$$

is zero, is not displayed. Instead, the region is determined where

$$\Delta|\sigma_d| \quad \leq \quad \Delta\sigma_{d,lim}. \tag{3.52}$$

Statements about the size of the area can be made where an integration method works accurately with respect to a given error bound $\Delta\sigma_{d,lim}$.

Roughly speaking, the larger this area is, the more accurate is the integration method.

3.6 Bode plot

In this section an alternative way of evaluating an integration algorithm will be introduced. This time, the behavior of the integration method is analyzed in the frequency domain through the use of a *Bode plot.* To provide a motivation for this approach, the Bode plot of a first order system is considered as shown in figure 3.12.

Figure 3.12 illustrates that the gain $|P(i\omega)|$ is 1 at low frequencies, while it goes to smaller values when the frequency is increased. In other words, this system lets signals of lower

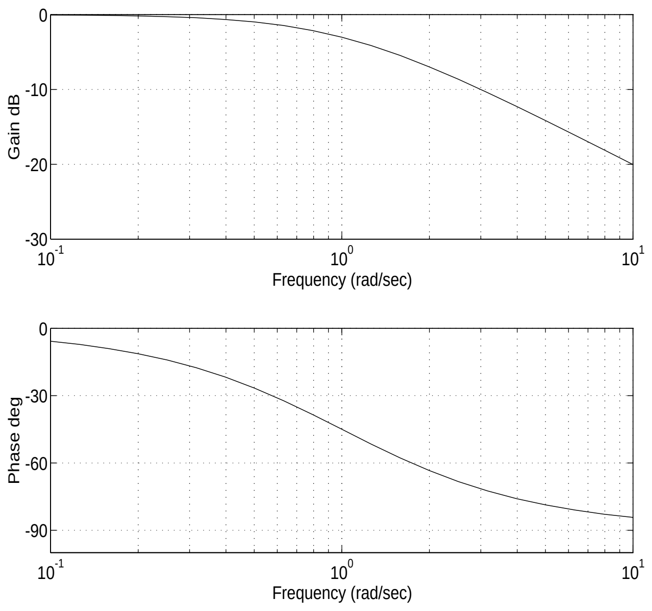


Figure 3.12: Bode plot of a first order system

frequencies pass through whereas it attenuates signals of higher frequencies. Hence, this system is called a *low-pass filter*.

When integrating a system whose input signal includes high frequency noise, an integration method might fail to control the step-size. Instead, the step-size control would follow the high frequencies such that the step-size would be reduced to an excessively small value. This effect can be avoided if the integration method is able to damp out high frequencies (i.e. functions as a low-pass filter).

To look for this property in a RBDF method, a Bode plot is used.

A key result of section 3.2.2 is that the general RBDF method which is formulated as

$$\underline{x}_{k+1} = \sum_{i=0}^{m-1} a_i \underline{x}_{k-i} + \sum_{i=-1}^{m-1} b_i h \underline{f}_{k-i} \tag{3.53}$$

can be considered as a discrete system

$$\underline{z}_{k+1} = F\underline{z}_k, \tag{3.54}$$

or more generally as

$$\begin{aligned} \underline{z}_{k+1} &= F\underline{z}_k + G\underline{u}_k, \\ \underline{x}_k &= H\underline{z}_k + I\underline{u}_k, \end{aligned} \tag{3.55}$$

where $\underline{u}_k = \underline{u}(t_k)$ is a given input variable.

This system has a transfer function which is called the *pulse transfer function* (PTF) in the discrete case.

For Single-Input Single-Output (SISO) systems, the pulse transfer function P results in

$$P(\zeta) = \frac{x(\zeta)}{u(\zeta)}, \tag{3.56}$$

where $\zeta = e^{-hs}$ (h = step-size of the discrete system). This function describes how an input value u is propagated to the output.

Equation (3.56) can be transformed from the ζ-domain to the frequency domain where the transfer function $P(i\omega)$ corresponds to $P(\zeta)$. Figure 3.13 illustrates how an integration algorithm can be regarded as a discrete system, and it gives an idea of what $P(i\omega) = x(i\omega)/u(i\omega)$ means.

By displaying both $|P(i\omega)|$ and $\arg\{P(i\omega)\}$ over a logarithmically scaled frequency axis a Bode plot is obtained.

Three linear test systems — stable, marginally stable, unstable — described by $\underline{\dot{x}} = A\underline{x}$, are chosen that keep their stability properties when being integrated by the considered RBDF algorithm.

This means that if this system is integrated by the method Φ using the step-size h then

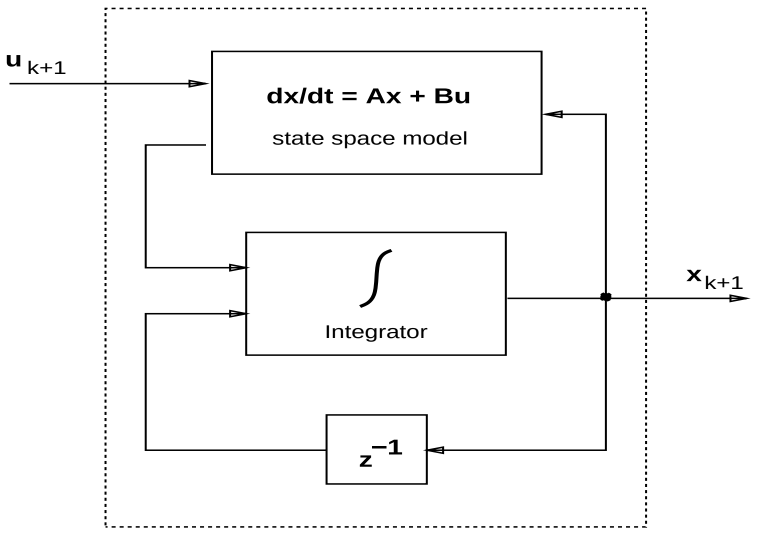


Figure 3.13: Integration method regarded as a discrete system

the eigenvalues $\lambda_i = eig\{A\}$ should satisfy

$$\begin{aligned} Re\{\lambda_i\} < 0 &\implies \lambda_i h \overset{!}{\in} \mathcal{S}(\Phi) \\ Re\{\lambda_i\} = 0 &\implies \lambda_i h \overset{!}{\in} \mathcal{M}(\Phi) \\ Re\{\lambda_i\} > 0 &\implies \lambda_i h \overset{!}{\in} \mathcal{I}(\Phi), \end{aligned} \tag{3.57}$$

where

$$\begin{aligned} \mathcal{S} &= \text{stability domain of method } \Phi \text{ in the } (\lambda h)\text{-plane}, \\ \mathcal{M} &= \text{stability locus of } \Phi \text{ in the } (\lambda h)\text{-plane}, \\ \mathcal{I} &= \text{instability domain of } \Phi \text{ in the } (\lambda h)\text{-plane} \end{aligned}$$

After the integration algorithm has been chosen such that the conditions (3.57) are satisfied, the integration technique is described as a discrete system as shown in (3.55). Then the Bode plot can be produced by using the MATLAB-command "DBODE." As an example, figure 3.6 shows the Bode plot when the stable system $\underline{\dot{x}} = A\underline{x}$ is integrated by the BDF6 technique, where

$$A = \begin{pmatrix} -10 & 0 \\ 0 & -10 \end{pmatrix}.$$

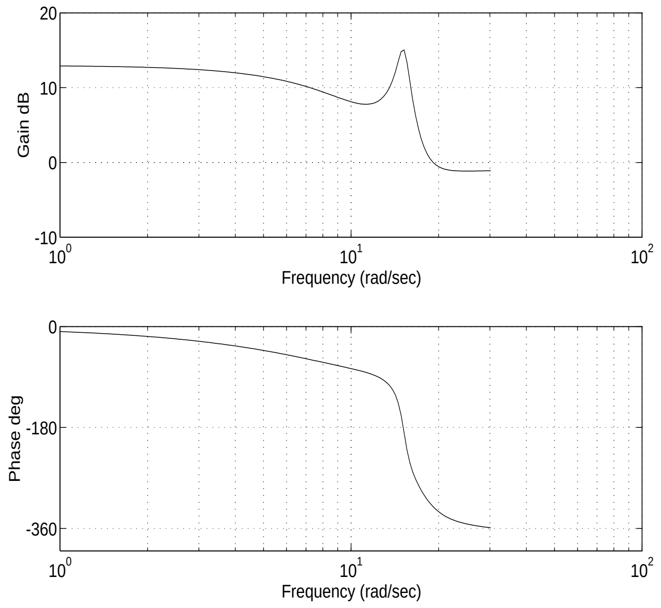


Figure 3.14: Bode plot of a stable system integrated by BDF6

CHAPTER 4

Implementation of RBDF

4.1 Startup problem

An m-step integration method needs m initial values of the state variable in order to get started. These starting values needn't be accurate as has been described in chapter 1. Gear [10] proposes to use Runge-Kutta methods for startup purposes. This is specified more in detail by Cellier [4] who applies $n-1$ steps with fixed step-size of a n^{th} order Runge-Kutta. Thus, the step-size should only be determined once at the beginning of the simulation. In order to calculate the initial step-size h quickly a *binary search technique* is employed that starts at a steplength h_0. This starting value of h_0 must guarantee a stable integration of a system with known eigenvalues by a Runge-Kutta algorithm of n^{th} order. At each step of the binary search, an estimate of the relative error, ϵ_{rel}, is made where,

$$\epsilon_{rel} \quad = \quad \frac{|x_1 - x_2|}{max(|x_1|, |x_2|, \delta)}. \tag{4.1}$$

Hence, x_1 is the value of the state variable x calculated by a n^{th} order RK-algorithm and x_2 is the value of the state variable x calculated by a $(n-1)^{th}$ order RK-algorithm. This result shown in equation (4.1) can be compared to a given error bound tolerance denoted as tol_{rel}:

If $0.9tol_{rel} \leq \epsilon_{rel} \leq tol_{rel}$, then the binary search is stopped; otherwise, either h is increased (if $\epsilon_{rel} < 0.9 tol_{rel}$) or h is decreased before the step is calculated again. If the initial values vanish ($\underline{x}(0) = \underline{0}$, $\underline{f}(0) = \underline{0}$) then the startup with RK-methods fails to compute the initial values for the multistep technique. In this case, a general RK-method described as

$$x_{k+1} = x_k + h \sum_{i=1}^{l} \beta_{li} \dot{x}^{P_{i-1}}, \tag{4.2}$$

where $\dot{x}^{P_{i-1}}$ are the predictors given by

$$\dot{x}^{P_{i-1}} = f(x^{P_{i-1}}, t_k + \alpha_{i-1} h), \tag{4.3}$$

will always yield the result $x = 0$. Therefore, relation (4.1) results in $\epsilon_{rel} = 0$, such that the step-size is never changed. Since the algorithm has been started at the maximum value of the steplength h_0 this might give rise to instability.

This problem can be solved by running one step with an RK-method of n^{th} order to modify the initial conditions. The time is set back to $t = t_0$ and one step of the startup is executed to determine the initial step-size h_0. By using h_0 and the original initial conditions of the system, the startup procedure is started again at $t = t_0$.

4.2 Step-size control

4.2.1 Introduction

Most of the integration methods can only be applied efficiently if the step-size is adjusted according to accuracy requirements. For example, when integrating stiff systems without step-size control a very small steplength would be applied during the whole integration (cf. chapter 1). Thus, the simulation would last excessively long.

Before the step-size control with multistep methods will be discussed, a mathematical tool, the so-called *Newton-Gregory polynomials*, is introduced.

Afterwards, it will be shown how this tool is used to calculate *Nordsieck vectors* of different order.

Finally, the algorithm for step-size control will be derived that applies the Nordsieck vector to update the state history vector if the step-size h is changed. This vector contains m earlier state variables x.

4.2.2 Newton-Gregory polynomials

To make the following calculations more convenient the *forward difference operator* Δ is introduced:

$$\begin{aligned} \Delta g_0 &= g_1 - g_0, \\ \Delta g_1 &= g_2 - g_1, \\ &\vdots \\ \Delta g_i &= g_{i+1} - g_i, \end{aligned} \tag{4.4}$$

where g_i denotes the value of a continuous function $g(t)$ at the point of time t_i.

By recursively applying the operator Δ, higher-order forward difference operators are obtained which, in the general case, are given by [4]

$$\begin{aligned} \Delta g_i^{p-1} &= \begin{pmatrix} p-1 \\ 0 \end{pmatrix} g_{i+p-1} - \begin{pmatrix} p-1 \\ 1 \end{pmatrix} g_{i+p-2} + \begin{pmatrix} p-1 \\ 2 \end{pmatrix} g_{i+p-3} \\ &\quad + \ldots \pm \begin{pmatrix} p-1 \\ p-1 \end{pmatrix} g_i, \end{aligned} \tag{4.5}$$

where p is the number of earlier data points.

After having defined the *backward difference operator* ∇ as

$$\nabla g_i = g_i - g_{i-1}, \tag{4.6}$$

we get the higher-order backward difference operators accordingly:

$$\begin{aligned}\nabla^{p-1} g_i &= \begin{pmatrix} p-1 \\ 0 \end{pmatrix} g_i - \begin{pmatrix} p-1 \\ 1 \end{pmatrix} g_{i-1} + \begin{pmatrix} p-1 \\ 2 \end{pmatrix} g_{i-2} \\ &+ \ldots \pm \begin{pmatrix} p-1 \\ p-1 \end{pmatrix} g_{i-p+1}.\end{aligned} \tag{4.7}$$

The task is to find a vector polynomial, i.e. a polynomial with scalar argument and vector coefficients, which interpolates the p distinct data points. This interpolant takes a particularly simple form when the time points t_i, where the function values are taken, are equally spaced [14], i.e.

$$t_{k-j} = t_k - jh, \qquad j = 0, 1, 2, \ldots, p \quad (h = \mathit{const.}). \tag{4.8}$$

At this point an auxiliary variable s is introduced as defined in [4],

$$s = \frac{t - t_0}{h}. \tag{4.9}$$

Hence, the interpolation polynomial can be depicted either as

$$g(s) \approx \begin{pmatrix} s \\ 0 \end{pmatrix} g_0 + \ldots + \begin{pmatrix} s \\ p-1 \end{pmatrix} \Delta^{p-1} g_0, \qquad (p = \text{number of data points}) \tag{4.10}$$

or

$$g(s) \approx g_0 + \begin{pmatrix} s \\ 1 \end{pmatrix} \nabla g_0 + \begin{pmatrix} s+1 \\ 2 \end{pmatrix} \nabla^2 g_0 + \ldots + \begin{pmatrix} s+p-2 \\ p-1 \end{pmatrix} \nabla^{p-1} g_0. \tag{4.11}$$

The expression in (4.10) is called the *Newton-Gregory forward polynomial* whereas (4.11) is known as the *Newton-Gregory backward polynomial* (see [4] for the derivation of (4.10) and (4.11)).

Note that the Newton-Gregory polynomials are only valid if the data points are evenly spaced, otherwise interpolating polynomials, such as the ones derived by Lagrange [14], have to be used.

The Newton-Gregory backward polynomial can be employed in a multistep integration by setting $t_k = t_0$ and $s = 1.0$. Furthermore, the back values need to be substituted by the corresponding function values $x(t_k), x(t_{k-1}), \ldots$ etc..

Thus, equation (4.11) automatically calculates an estimate $x(t_{k+1}) = g_1$ which is the unknown of the multistep algorithm.

4.2.3 Calculation of the Nordsieck vector

In the following it is assumed for convenience that the system being integrated is scalar, that is: $n_s = 1$.

With the results of the preceding subsection, the Newton-Gregory backward polynomial for $x(t)$ can be written as

$$x(t) = x_k + s\nabla x_k + \left(\frac{s^2}{2} + \frac{s}{2}\right)\nabla^2 x_k + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3}\right)\nabla^3 x_k + \ldots \tag{4.12}$$

This equation is differentiated with respect to time which yields

$$\dot{x}(t) = \frac{1}{h}[\nabla x_k + (s + \frac{1}{2})\nabla^2 x_k + (\frac{s^2}{2} + s + \frac{1}{3})\nabla^3 x_k + \ldots]. \tag{4.13}$$

The higher order derivatives are determined by recursively differentiating (4.13). By truncating the resulting expressions after the n_g^{th} term (n_g = order of the Nordsieck vector) and expanding the ∇-operator according to (4.4) and evaluating for $t = t_k$ $(s = 0.0)$, the expression

$$\underline{g} \quad = \quad T\underline{s}, \tag{4.14}$$

is obtained where

$$\underline{s} \quad = \quad \begin{pmatrix} x_k \\ x_{k-1} \\ \vdots \\ x_{k-m+1} \end{pmatrix}$$

is the state history vector that contains m state variables of m earlier time instants, T is a $[(n_g + 1) \times m]$ transformation matrix, and

$$\underline{g} \quad = \quad \begin{pmatrix} x_k \\ h\dot{x}_k \\ \vdots \\ \frac{h^{n_g}}{n_g!} x_k^{(n_g)} \end{pmatrix}$$

is the $(n_g + 1)$-vector which is similar to the Nordsieck vector $N = (x_k, h\dot{x}_k, \ldots, h^{n_g} x_k^{(n_g)})$ of n_g^{th}-order (cf. [4]). However, for convenience we refer to $\underline{g}$ as the Nordsieck vector in this section.

In the appendix some transformation matrices are shown for different values of n_g and m if the system being integrated is one-dimensional $(n_s = 1)$.

One might get the idea of using the given value $h\dot{x}_k$ as additional information in the state history vector $\underline{s}$. This would require one to substitute the expressions for the last element of $\underline{s}$ by the expressions dependent on $\dot{x}_k$. These expressions can be derived by solving

$$\dot{x} = \ldots + a_{k-m+1} x_{k-m+1}$$

for x_{k-m+1}. Hence, the last element of $\underline{s}$ can be rewritten as

$$x_{k-m+1} = a x_k + b h \dot{x}_k + c x_{k-1} + \ldots,$$

where the constants a, b, c, ... need to be determined.

However, multiple simulations which implemented the described step-size control have shown that the use of $h\dot{x}_k$ is not advantageous. The following quickly describes two key results:

1. The integration process becomes slower. One reason for this is that the transformation matrix T, which has to be inverted to calculate the new state history vector after a change of the step-size, becomes larger. This is explained by the fact that the formula of the integration algorithm contains the last element of the state history vector $\underline{s}$. Therefore, this element has to be calculated and can only be substituted by an expression of $\dot{x_k}$ when the Nordsieck vector is computed.

   Furthermore, it can be observed that the Nordsieck vector is calculated less accurately: An error is introduced by replacing $\dot{x}$ by $\dot{x}_k$. Therefore, the integration algorithm must use smaller steplengths to satisfy the accuracy requirements.

2. Although one might think that the integration would become more accurate by using additional information about the state space model within each integration step, this is not true. Simulations have proven without any exception that the remaining global

error becomes larger as soon as $h\dot{x}_k$ is used.

If $n_g + 1 = m$ then T is quadratic and nonsingular and can be quickly inverted. However, if the integration algorithm uses more than $n_g + 1$ steps, the vector $\underline{s}$ would have more elements than the Nordsieck vector $\underline{g}$. Therefore, T isn't quadratic any more. In the following section it will be shown how the step-size can be controlled in this case.

When the integration algorithm uses one step more than necessary ($m = n_g + 2$), then the latter technique needn't be used. Instead, T is still a quadratic matrix which yields a Nordsieck vector that seems to have one element too much at first glance. However, it will be shown in the following section that this is not the case.

4.2.4 Step-size control algorithm

The idea is to calculate an estimate for the local error produced at each step and to compare it to a given error bound *tol*. Depending on the result of the comparison the steplength h will be changed.

The following shows how an estimate for the relative local error ϵ_{rel} can be obtained:

The last element g_{n_g} of the Nordsieck vector

$$\underline{g} = \begin{pmatrix} x_k \\ h\dot{x}_k \\ \vdots \\ \frac{h^{n_g}}{n_g!} x_k^{(n_g)} \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n_g} \end{pmatrix} \tag{4.15}$$

is given by

$$g_{n_g} = \frac{h^{n_g} x^{(n_g)}}{n_g!}. \tag{4.16}$$

According to [10], the change Δg_{n_g} of g_{n_g} is an estimate of

$$\Delta g_{n_g} = \frac{h^{n_g+1} x^{(n_g+1)}}{n_g!} \tag{4.17}$$

which can be shown by:

$$\begin{aligned} \Delta g_{n_g} &= h^{n_g} \frac{x_k^{(n_g)}}{n_g!} - h^{n_g} \frac{x_{k-1}^{(n_g)}}{n_g!} \\ &= \frac{h^{n_g}}{n_g!} h \underbrace{\left(\frac{x_k^{(n_g)} - x_{k-1}^{(n_g)}}{h} \right)}_{x_k^{(n_g+1)}}. \end{aligned} \tag{4.18}$$

As shown in chapter 1, the local truncation error can be estimated by

$$T = C_{n+1} h^{n+1} x_k^{(n+1)} + \mathcal{O}(h^{n+2}). \tag{4.19}$$

Only the principal term is considered, and that needs to be smaller than the given error bound *tol*:

$$C_{n+1} h^{n+1} x_k^{(n+1)} \le tol. \tag{4.20}$$

By choosing $n_g = n$, the equations (4.18) and (4.20) can be combined to formulate

$$C_{n+1} n! \Delta g_n \le tol. \tag{4.21}$$

As Gear [10] proposes, relation (4.21) should be tested first. If the test succeeds, the step is accepted; otherwise it is rejected.

The new step-size is calculated by

$$h_{new} = \alpha h_{old}, \tag{4.22}$$

where

$$C_{n+1} n! \alpha^{n+1} \Delta g_n = tol. \tag{4.23}$$

This can be shown by plugging (4.22) into (4.17) which yields

$$\begin{aligned} \Delta g_{n,new} &= \frac{\alpha^{n+1} h^{n+1}}{n!} \left(x_k^{(n+1)} \right)^T = \\ &= \alpha^{n+1} \Delta g_{n,old}. \end{aligned} \tag{4.24}$$

By plugging relation (4.24) into (4.21) and treating (4.21) as an equality, equation (4.23) is obtained.

Since Δg_n usually is not constant, a slightly smaller step-size is used in order that (4.21) can be expected to be satisfied. Thus α is determined by

$$\alpha = \frac{1}{1.2} \left(\frac{tol}{C_{n+1} n! \Delta g_n} \right)^{\frac{1}{n+1}}. \tag{4.25}$$

However, calculating α and changing the step-size at each step, the integration algorithm would be very slow. Therefore, some of the advises given in [11] should be considered:

1. Values for α are only accepted between 0.5 and 2.0.
2. If $1 \leq \alpha \leq 1.1$, the new step-size is chosen to be $h_{new} = 0.9 h_{old}$.
3. When the step has failed, the step-size is halved.

As soon as the step-size is changed the state history vector $\underline{s}$ will have to be adjusted accordingly. Otherwise the integration method would proceed with wrong initial values. The adjustment can easily be done by the Nordsieck technique:

The new Nordsieck vector is computed by

$$\underline{g}_{new} = H \underline{g}_{old}, \tag{4.26}$$

where

$$H = \begin{pmatrix} 1 & 0 & \cdots & & 0 \\ 0 & \alpha & \ddots & & \vdots \\ \vdots & \ddots & \alpha^2 & & \\ & & & \ddots & 0 \\ 0 & & & 0 & \alpha^{n_g} \end{pmatrix}. \tag{4.27}$$

In order to obtain the new state history vector $\underline{s}_{new}$, equation (4.26) is plugged into the relation

$$\underline{s}_{new} = T^{-1}\underline{g}_{new}, \tag{4.28}$$

where T is the transformation matrix introduced in (4.14).

So far the only case which has been regarded is where $n_g = n$. This means that the state history vector $\underline{s}$ has $n+1$ elements if T is quadratic. Therefore, the integration method uses $m = n+1$ steps.

If $m = n+2$ steps are applied, then the $[(n_g+1) \times m]$ transformation matrix T is still quadratic. Thus, the last element of the Nordsieck vector becomes

$$\frac{h^{n_g}}{n_g!}x_k^{n_g} = \frac{h^{n+1}}{(n+1)!}x_k^{(n+1)}, \tag{4.29}$$

and it can be directly used in (4.20) without making the approximation (4.17). This saves computation time.

If the integration algorithm employs more than $n+2$ steps then there are two options:

1. The transformation matrix T used in

$$\underline{g} = T\underline{s}$$

can still be quadratic. However, the Nordsieck vector will have too many elements since only the $(n_g+1)^{th}$ element of $\underline{g}$ has to be determined (g_{n_g}). The state history vector might be updated after a change of the steplength by applying the relation

$$\underline{s} \;=\; T^{-1}\underline{g}. \tag{4.30}$$

2. The $(n+2)$-Nordsieck vector $\underline{g}$ keeps its size. Therefore, $\underline{g}$ can still be computed by

   $$\underline{g} \;=\; T\underline{s},$$

   where $\underline{\mathrm{s}}$ is the state history vector containing m components $(m > n+2)$.

   However, to get the new vector $\underline{s}_{new}$ after a change of the step-size, which results in a change of $\underline{g}$, an overdetermined system has to be solved. As in chapter 2 this system is solved in a least-squares sense by calculating the Penrose-Moore Pseudoinverse:

   $$\underline{s} \;=\; \left(T^T T\right)^{-1} T^T \underline{g}. \tag{4.31}$$

In order to decide which of the two options is better the number of applied scalar multiplications (*sm*s) is estimated. To investigate the efficiency of the first option relation (4.14) has to be solved first where T is a $(m\times m)$-matrix $(m > n+2)$. This can be done with $q_1 = m^2$ multiplications. For the solution of the inverse problem (4.30) after the change of the step-size, it is assumed that the computation of the inverse of the asymmetric matrix T corresponds to q_2 multiplications. Therefore

$$q_3 \;=\; m^2 + q_2 \tag{4.32}$$

multiplications are needed to determine $\underline{s}$. Thus, after a change of the step-size

$$\begin{aligned} q_{o,1} &= q_1 + q_3 \\ &= 2m^2 + q_2 \end{aligned} \tag{4.33}$$

multiplications have to be executed when choosing the first option.

When choosing the second option, the $[(n+2) \times m]$-matrix T $(m > n+2)$ in equation (4.14) isn't quadratic any more. Therefore, equation (4.14) can be calculated with $q_4 = m(n+2)$ multiplications. Since the matrix

$$T^* = T^T T \tag{4.34}$$

is a $(m \times m)$-matrix, the calculation of its inverse corresponds to q_2 multiplications again. Hence, the inverse problem (4.28) is solved by

$$q_5 = q_2 + 2m^2(n+2) + m(n+2) \tag{4.35}$$

multiplications. Thus, by choosing the second option, the calculation of new vectors $\underline{g}$ and $\underline{s}$ corresponds to

$$\begin{aligned} q_{o,2} &= q_4 + q_5 \\ &= q_2 + 2m(n+2) + 2m^2(n+2) \end{aligned} \tag{4.36}$$

multiplications.

Since $m \geq n+2$, it follows from (4.33) and (4.36) that

$$q_{o,2} > q_{o,1}, \tag{4.37}$$

that is, the first option is faster.

4.3 Integration of nonlinear systems

4.3.1 Startup problem

In section 4.1 it has been described how a multistep integration method can be started quickly by using a binary search technique and RK methods.

In order to search in a binary fashion for the initial step-size h, a starting value h_0 is required. This value can be obtained by determining the largest magnitude $|\lambda_{max}|$ of the eigenvalues of the system and using the condition

$$|\lambda_{max}|h_0 \quad = \quad r_0, \tag{4.38}$$

where r_0 is the radius of the largest semicircle around the origin that fits completely into the part of the stability domain, of the applied n^{th}-order RK-method ($n > 2$), which is in the left half plane.

The eigenvalues can be calculated if the system being integrated is linear. However, in the nonlinear case the eigenvalues of the Jacobian of the system may be determined. The column vectors $\underline{J}_i$ can be computed as shown in equation (4.39) [4].

$$\underline{J}_i \quad = \quad \frac{\delta \underline{f}}{\delta x_i} \approx \frac{\underline{f}_{dev} - \underline{f}}{\Delta x_i}, \tag{4.39}$$

where $\underline{f}$ is the state derivative vector of the nominal state variables and the vector $\underline{f}_{dev}$ is the state derivative vector which results from the perturbation of the state variable x_i by Δx_i. Note that all of the other state variables are kept unchanged.

By determining the maximum eigenvalue λ_{max} of the Jacobian J and using relation (4.38) we obtain the initial step-size h_0. The value r_0 should be chosen slightly smaller since the Jacobian can only be approximated and a stable integration has to be ensured in any case.

4.3.2 Newton iteration

In this thesis only implicit linear multistep integration methods are of interest. They are given by

$$\underline{x}_{k+1} = \sum_{i=0}^{m-1} a_i \underline{x}_{k-i} + \sum_{i=-1}^{m-1} b_i h \underline{f}_{k-i}. \tag{4.40}$$

In the case of a linear homogeneous system

$$\dot{\underline{x}} = A\underline{x}, \quad \underline{x}(0) = \underline{0}, \tag{4.41}$$

equation (4.40) results in

$$\underline{x}_{k+1} = [I^{(n_s)} - b_{-1}hA]^{-1} \sum_{i=0}^{m-1} (a_i \underline{x}_{k-i} + b_i h \underline{f}_{k-i}). \tag{4.42}$$

However, the matrix inversion cannot be applied to a nonlinear problem. Cellier [4] proposes to rewrite (4.40) in the form

$$\underline{\mathcal{F}}(\underline{x}_{k+1}) = b_{-1}h\underline{f}(\underline{x}_{k+1}, t_{k+1}) - \underline{x}_{k+1} + \sum_{i=0}^{m-1} (a_i \underline{x}_{k-i} + b_i h \underline{f}_{k-i}) = \underline{0} \tag{4.43}$$

which can be solved by Newton iteration,

$$\underline{x}_{k+1}^{l+1} = \underline{x}_{k+1}^{l} - [\mathcal{H}^l]^{-1}[\underline{\mathcal{F}}^l], \tag{4.44}$$

where the *Hessian* $\mathcal{H}$ is given by:

$$\mathcal{H} = b_{-1}Jh - I^{(n_s)}. \tag{4.45}$$

In order to save computation time the Jacobian and the Hessian are only computed if it is necessary, e.g. when the state variables change drastically. This is discussed in more detail in [4].

4.4 Readout problem

At the end of a simulation the results of the simulation need to be displayed. Note that the points x_k calculated by the integration algorithm are most likely not evenly spaced because the step-size control always changes the step-size.

The problem is that the person who runs the simulation desires values at defined points called *communication points*. It is assumed for convenience that these points are equally spaced.

Within the startup period it is integrated right past the communication point, which is characterized by the time instant $t = t_{com}$. Then an interpolation is made between the calculated values to obtain the value of $\underline{x}_{com}$ at t_{com}.

As soon as the multistep integration algorithm is applied, the Nordsieck technique is used again: After having integrated right past t_{com}, a negative step-size is applied to reach the communication point. By calculating the Nordsieck vector $\underline{g}$ at t_{com} and adding up the components of $\underline{g}$, which represent terms in a Taylor series expansion, an estimate for

$$\underline{x}_{com} \quad = \quad \underline{x}(t_{com}).$$

is obtained (see [4] for more details).

From cellier@esaii.upc.esSun Jul 9 14:43:51 1995 Date: Sun, 9 Jul 1995 12:07:59 UTC+0200 From: Francois Cellier ¡cellier@esaii.upc.es¿ To: Klaus@GAS.UUG.Arizona.Edu Subject: Kapitel 5 jetzt in Ordnung (nur kleine Aenderungen vorgenommen)

CHAPTER 5

RBDF of 6th order (RBDF6)

5.1 Derivation of the algorithms

In this chapter RBDF techniques will be derived that are 6^{th}-order accurate (RBDF6 methods).

The order of 6 is chosen because this is the largest order for which a conventional stable BDF method can be found. Thus, it is possible to compare the performance of the RBDF6 methods with that of the BDF6 algorithm. The probability of finding RBDF methods increases with the accuracy order of the method due to the increased number of interpolation points available in the search. However, since there are more possible candidates, the search also takes longer.

The search method mentioned at the end of chapter 2 is applied to find RBDF6 algorithms that satisfy minimum stability requirements. In chapter 2 it is discussed that "minimum stability requirements" means that the stability locus doesn't intersect with the negative real axis of the complex (λh)-plane.

Table 5.1 displays the numbers of 6^{th}-order RBDF algorithms that have been found for different numbers of applied data points. Note that the data points are always given within the time interval $[t_{k-n-2}, t_{k+1}]$.

Number of data points	Number of RBDF6 algorithms
8	71
9	45
10	15
11	3
12	-

Table 5.1: Number of 6^{th} order RBDF algorithms satisfying minimum stability requirements

The stability domain of these methods is considered and compared to the stability domain of BDF6 depicted in figure 5.1.

Out of the 134 RBDF techniques, only 18 algorithms are found whose stability domains

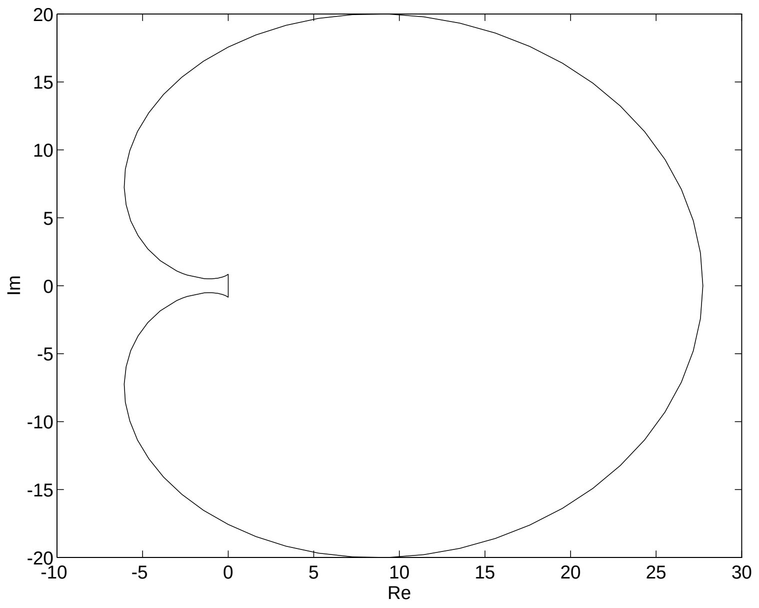


Figure 5.1: Stability domain of BDF6

look similar to the one of BDF6.

The next step is to analyze the damping plots of the new methods and compare them with the damping plot of BDF6 that is shown in figure 5.2.

Figure 5.2 illustrates that as soon as the magnitude of the analytical damping σ_d exceeds

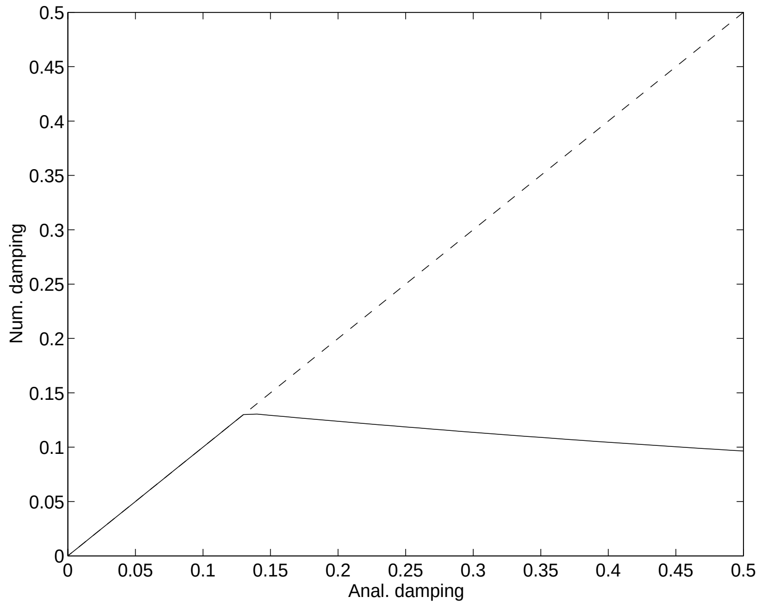


Figure 5.2: Damping plot of BDF6

$\sigma_{d,crit} = 0.13$, then the numerical damping $\hat{\sigma}_d$ of BDF6 starts to deviate from σ_d.

Only those methods which have a greater or at least only slightly smaller value $\sigma_{d,crit}$ than BDF6 are discussed in this section.

Regarding the damping plot shown in figure 5.3, one of the new methods can be seen to have poor damping qualities as expressed by the small value $\sigma_{d,crit} = 0.001$. This can be explained by the spurious eigenvalues which have a negative impact on the damping properties of a multistep technique (cf. chapter 3). They can also influence the numerical stability properties which can be observed in figure 5.4 where the stability domain of the considered RBDF method is depicted.

Figure 5.4 shows that the stability domain of the "bad" RBDF method has some sharp

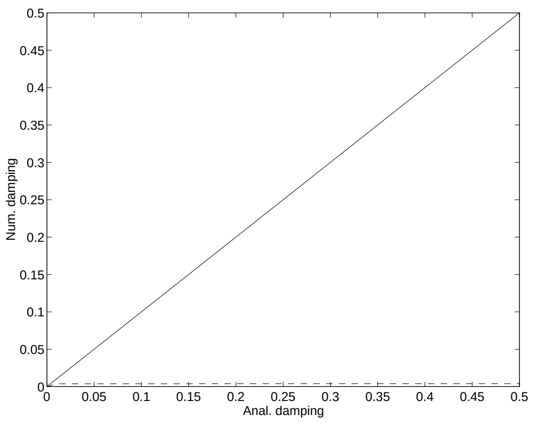


Figure 5.3: Damping plot of a "bad" RBDF

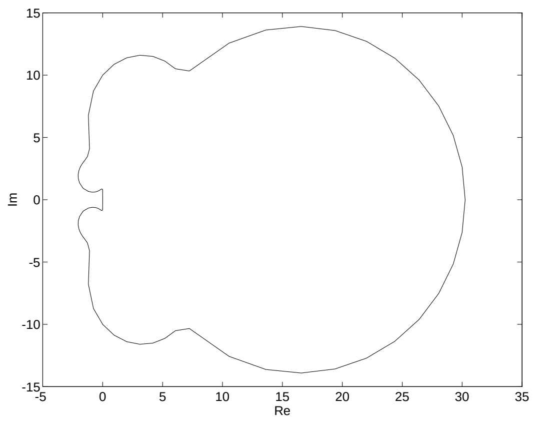


Figure 5.4: Stability domain of the "bad" RBDF

bends which are due to the spurious eigenvalues.

By comparing the damping plots of the RBDF6 methods to the damping plot of the BDF6 algorithm, the number of possible candidates is further reduced. Only 11 methods are left over by now.

Since it is required that the numerical damping $\hat{\sigma}_d$ goes to infinity as the analytical damping σ_d goes to infinity (cf. chapter 3), the logarithmic damping plot also needs to be analyzed. As a result of this analysis it is found that only one method satisfies this condition. This method will be referred to as RBDF61. All the other techniques have logarithmic damping plots where the magnitude of the numerical damping $\hat{\sigma}_d$ reaches a constant value $\hat{\sigma}_{d,lim}$ as the analytical damping σ_d goes to infinity. Those methods with a small value for $\hat{\sigma}_{d,lim}$ are eliminated.

Finally, there are eight methods left. They are listed in table 5.2. This table tabulates the number of applied data points, the RBDF formula and the error constant for each method by method name. Figures 5.5 and 5.6 show the corresponding stability domains, and figures

5.7 and 5.8 depict the damping plots.

The logarithmic damping plots are illustrated in figures 5.9 and 5.10.

Figures 5.11 and 5.12 depict the order stars of the methods. In order to make a comparison, the corresponding plots of BDF6 have been added to the figures.

Name	Number of data points	Formula	Error constant C_{err}
RBDF61	8	$x_{k+1} = \frac{594}{1357}hf_{k+1} + \frac{977}{461}x_k - \frac{1612}{915}x_{k-1} + \frac{361}{943}x_{k-2} + \frac{1171}{1310}x_{k-3} - \frac{3199}{3212}x_{k-4} + \frac{257}{592}x_{k-5} - \frac{389}{5370}x_{k-6}$	-0.1350
RBDF62	8	$x_{k+1} = \frac{582}{1331}hf_{k+1} + \frac{849}{392}x_k - \frac{2675}{1733}x_{k-1} - \frac{448}{2075}hf_{k-1} + \frac{504}{503}x_{k-3} - \frac{987}{1082}x_{k-4} + \frac{751}{2608}x_{k-5} + \frac{449}{14014}hf_{k-6}$	-0.1435
RBDF63	8	$x_{k+1} = \frac{139}{327}hf_{k+1} + \frac{671}{291}x_k - \frac{496}{249}x_{k-1} - \frac{704}{2707}hf_{k-1} + \frac{717}{1045}x_{k-2} + \frac{953}{3537}hf_{k-3} - \frac{649}{8310}hf_{k-5} + \frac{159}{6644}hf_{k-6}$	-0.1117
RBDF64	8	$x_{k+1} = \frac{440}{991}hf_{k+1} + \frac{1538}{747}x_k - \frac{982}{635}x_{k-1} + \frac{717}{842}x_{k-3} + \frac{1278}{4789}hf_{k-3} - \frac{955}{1969}x_{k-4} + \frac{659}{5445}x_{k-6} + \frac{418}{5011}hf_{k-6}$	-0.1612
RBDF65	8	$x_{k+1} = \frac{533}{1231}hf_{k+1} + \frac{4519}{2033}x_k - \frac{958}{509}x_{k-1} - \frac{587}{3222}hf_{k-1} + \frac{1543}{2037}x_{k-2} - \frac{326}{993}x_{k-5} + \frac{402}{1747}x_{k-6} + \frac{537}{4430}hf_{k-6}$	-0.1443
RBDF66	9	$x_{k+1} = \frac{731}{1692}hf_{k+1} + \frac{1733}{780}x_k - \frac{4758}{2779}x_{k-1} - \frac{609}{2605}hf_{k-1} + \frac{291}{1169}x_{k-2} + \frac{769}{942}x_{k-3} - \frac{1070}{1183}x_{k-4} + \frac{1265}{3196}x_{k-5} - \frac{596}{8993}x_{k-6}$	-0.1258
RBDF67	9	$x_{k+1} = \frac{607}{1385}hf_{k+1} + \frac{566}{265}x_k - \frac{1868}{1137}x_{k-1} - \frac{319}{3077}hf_{k-1} + \frac{247}{1239}x_{k-2} + \frac{879}{976}x_{k-3} - \frac{591}{677}x_{k-4} + \frac{943}{3367}x_{k-5} + \frac{428}{13581}hf_{k-6}$	-0.1433
RBDF68	9	$x_{k+1} = \frac{958}{2253}hf_{k+1} + \frac{3743}{1623}x_k - \frac{1121}{564}x_{k-1} - \frac{1389}{5230}hf_{k-1} + \frac{604}{859}x_{k-2} + \frac{895}{3989}hf_{k-3} - \frac{289}{2536}hf_{k-5} - \frac{87}{3995}x_{k-6} + \frac{180}{10001}hf_{k-6}$	-0.1125

Table 5.2: RBDF6 methods

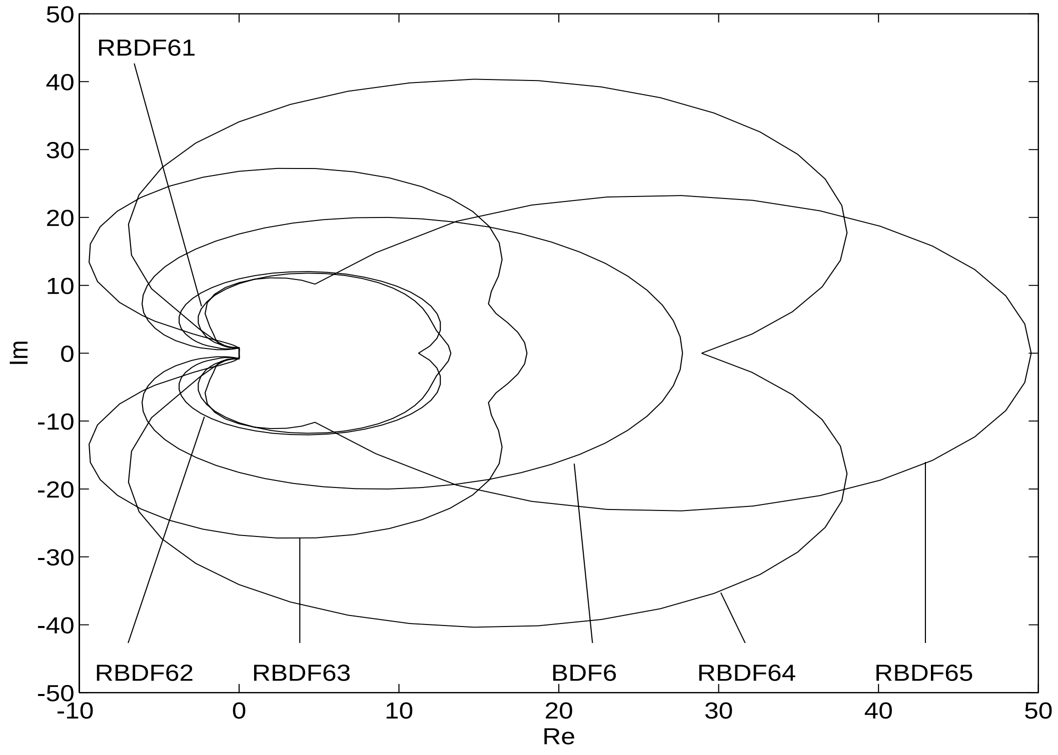


Figure 5.5: Stability domains of RBDF6 with 8 data points

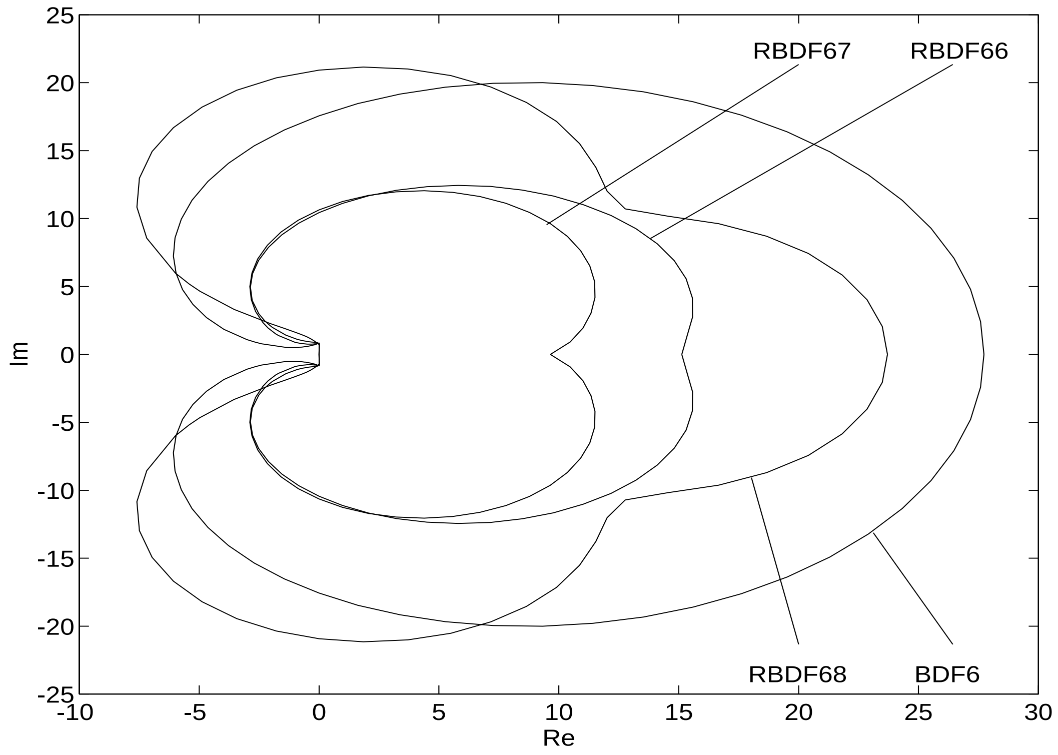


Figure 5.6: Stability domains of RBDF6 with 9 data points

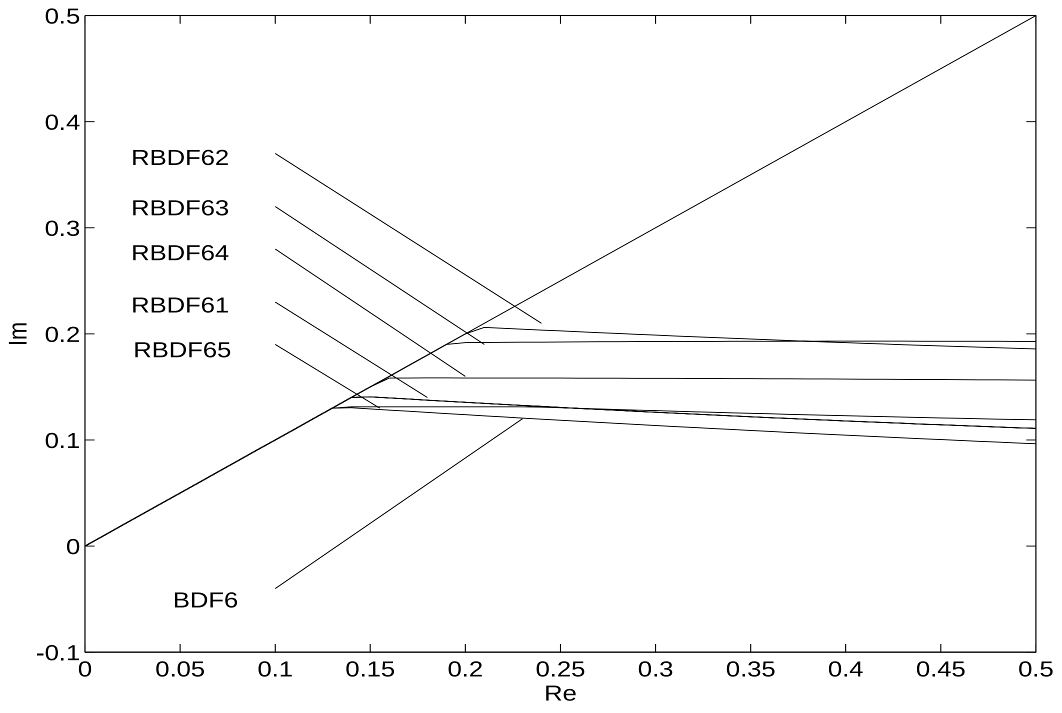


Figure 5.7: Damping plots of RBDF6 with 8 data points

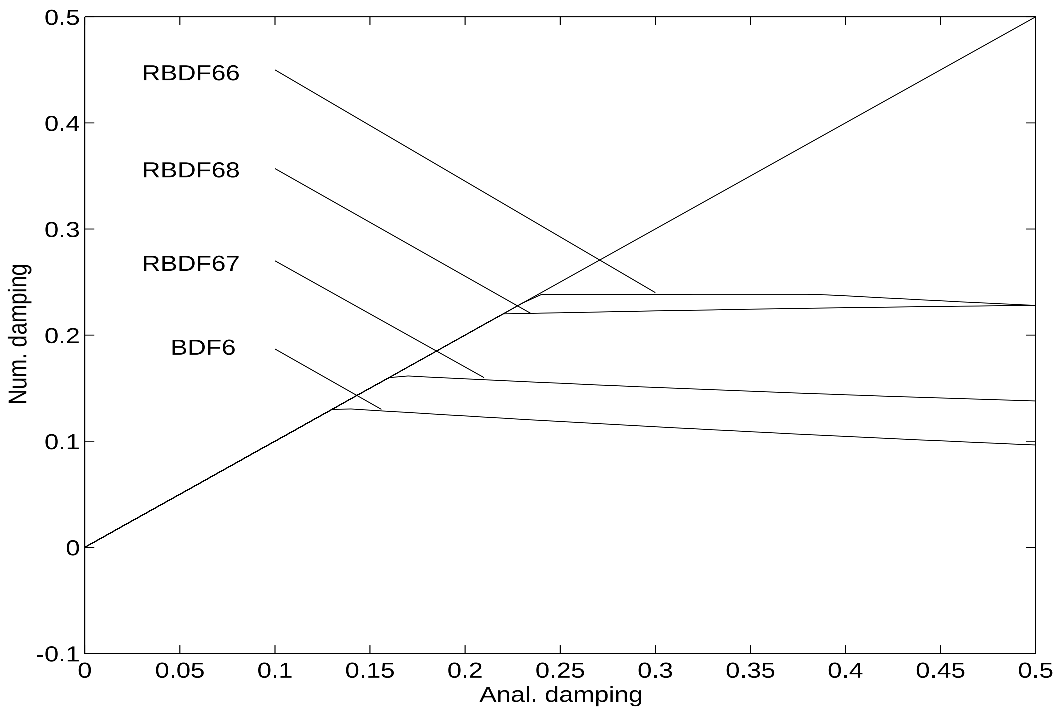


Figure 5.8: Damping plots of RBDF6 with 9 data points

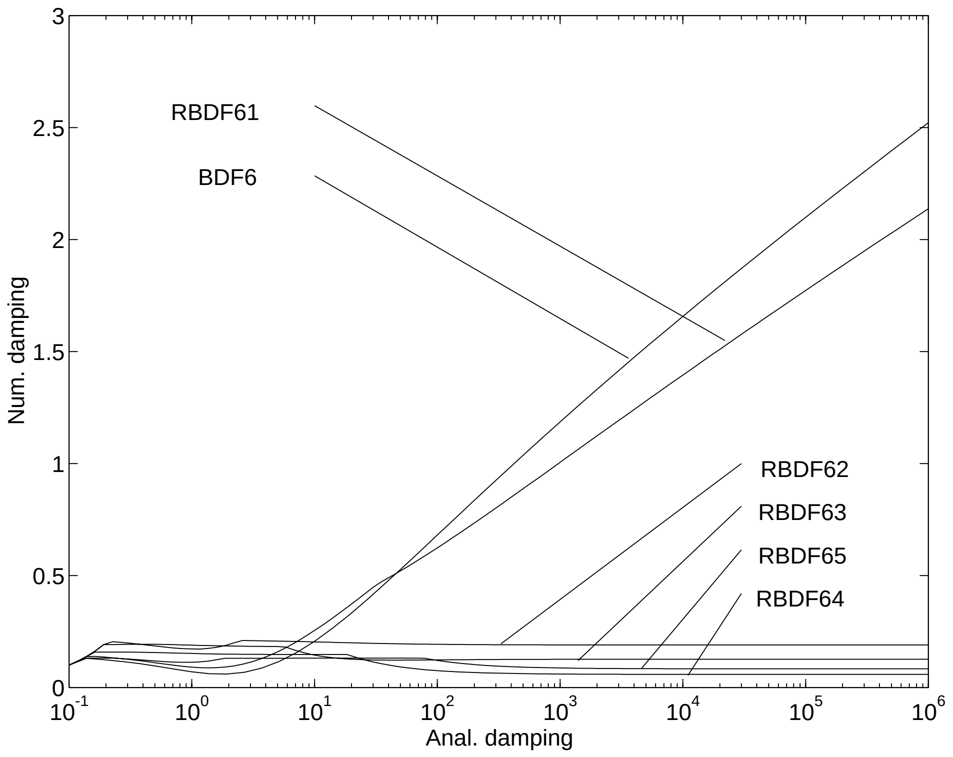


Figure 5.9: Logarithmic damping plots of RBDF6 with 8 data points

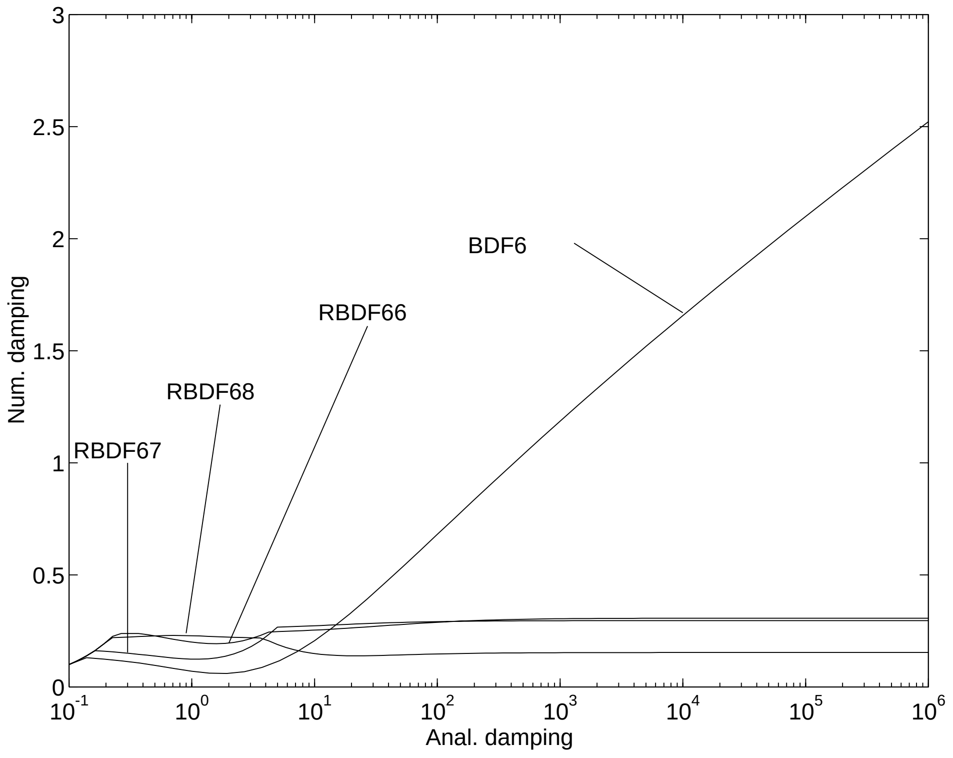


Figure 5.10: Logarithmic damping plots of RBDF6 with 9 data points

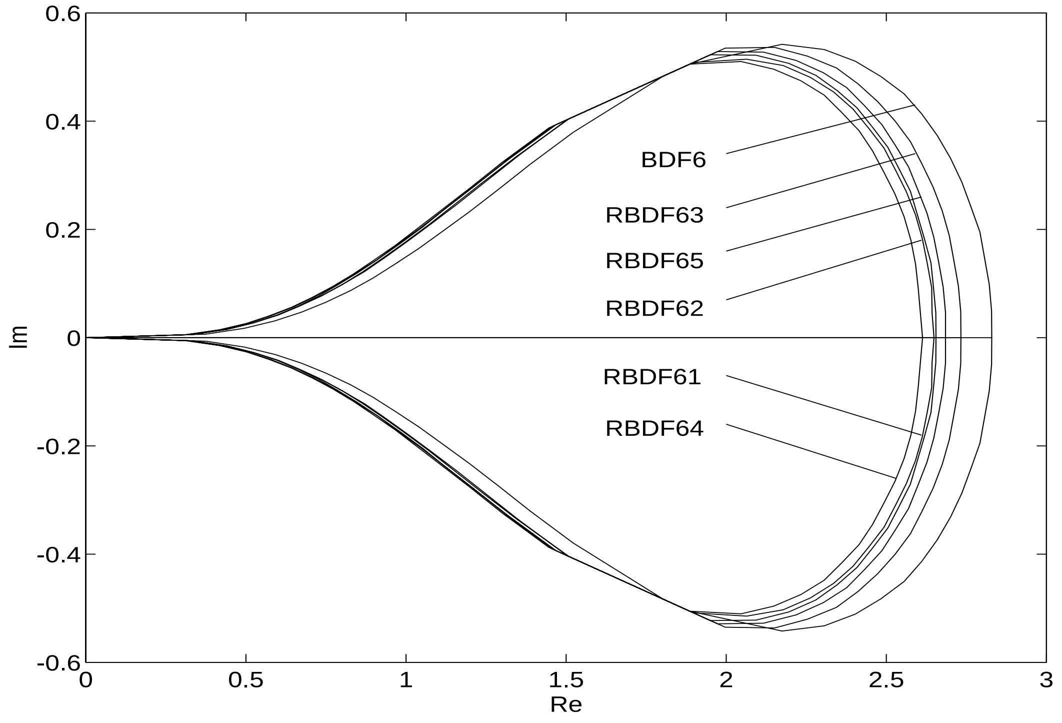


Figure 5.11: Order stars of RBDF6 with 8 data points

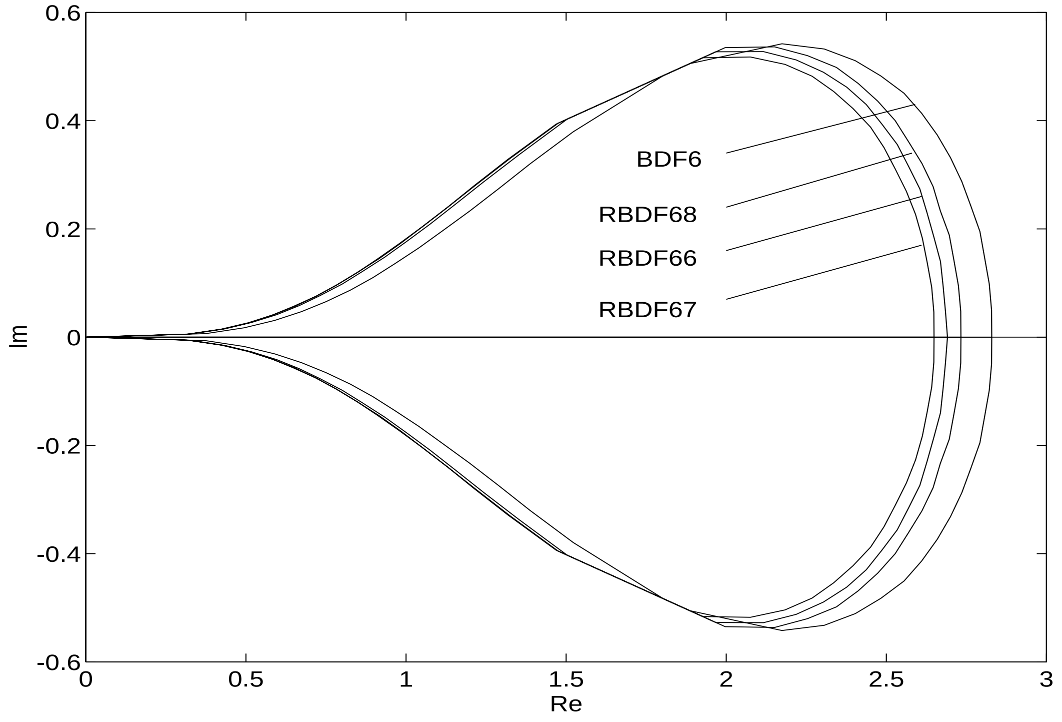


Figure 5.12: Order stars of RBDF6 with 9 data points

Name	Number of data points	Formula	Error constant C_{err}
BDF6	7	$x_{k+1} = \frac{20}{49}hf_{k+1} + \frac{120}{49}x_k - \frac{150}{49}x_{k-1} + \frac{400}{147}x_{k-2} - \frac{75}{49}x_{k-3} + \frac{24}{49}x_{k-4} - \frac{10}{147}x_{k-5}$	-0.0583

Table 5.3: BDF6 method

5.2 Analysis

The RBDF methods were already analyzed while they were being derived to make sure that they are good candidates. However, their properties are now discussed in more detail in order to assess their characteristics.

After applying the definition of stiffly stable systems (see figure 1.3) to the stability domain of BDF6 and RBDF6 it can be seen that only RBDF61, RBDF62, RBDF65, RBDF66 and RBDF67 can compete against BDF6, i.e. they have a smaller value a and a larger value c. The damping plots of all eight RBDF6 methods look better than the one of BDF6. However, the logarithmic damping plot is always worse. Only for the BDF6 method and the RBDF61 technique, $\hat{\sigma}_d \to \infty$ holds as $\sigma_d \to \infty$. Thus, the other RBDF methods cannot dampen fast transients of a system sufficiently well, which might give rise to longer integration runs.

Table 5.2 shows that the error constant of every RBDF6 algorithm is larger in magnitude than the error constant of BDF6 (cf. table 5.3). This might cause higher computation cost since the step-size control will suggest smaller step sizes to be used.

The order stars of the new methods look very similar to the order star of BDF6. However, no conclusions can be drawn from this since the order star itself doesn't tell us much about numerical stability and accuracy (cf. chapter 3). Therefore, the region is considered where the absolute damping error

$$\Delta\sigma_d \quad = \quad |\hat{\sigma}_d - \sigma_d| \tag{5.1}$$

is smaller than a given absolute error bound $\Delta\sigma_{d,lim}$. Since a 6th-order integration technique is used it makes sense to choose $\Delta\sigma_{d,lim} = 1E - 6$.

The accuracy properties of the integration method are determined by the area close to the origin. Therefore, this origin-close region should be regarded in detail as shown in figure 5.13 where the accuracy regions for BDF6, RBDF61 and RBDF66 are displayed.

Figure 5.13 shows that the accuracy regions of BDF6, RBDF61 and RBDF66 are almost

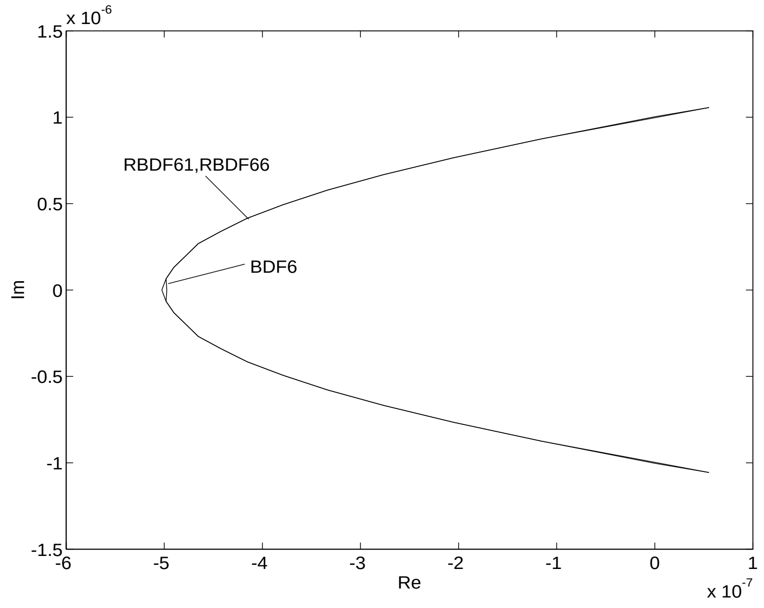


Figure 5.13: Accuracy region of BDF6, RBDF61 and RBDF66

identical in the left half plane. However, near the real axis, the border of the accuracy region of BDF6 is slightly closer to the origin than the ones of RBDF61 and RBDF66. Therefore, RBDF61 and RBDF66 can use a slightly larger step-size than BDF6 when a stable system with real eigenvalues is integrated and a given accuracy condition has to be satisfied.

5.3 Simulation results and comparison of the RBDF6 methods with BDF6

In this section RBDF6 methods will be used to integrate some stiff systems. The results will be compared to those obtained when integrating the same systems with BDF6. Although seven stiff systems have been simulated, the results of only two of these systems are discussed here since they are representative of the others. One of the two systems is linear, the other one is nonlinear:

- System 1 (taken from [6]):

$$\underline{\dot{x}} = \begin{pmatrix} 0 & 1 \\ -1000 & -1001 \end{pmatrix} \underline{x}, \tag{5.2}$$

$$\underline{x}(0) = \begin{pmatrix} 1 \\ -1 \end{pmatrix}. \tag{5.3}$$

$$\tag{5.4}$$

Eigenvalues: $\lambda_1 = -1000$; $\lambda_2 = -1$

Analytical solution:

$$\begin{aligned} x_1(t) &= e^{-t} \\ x_2(t) &= -e^{-t} \end{aligned} \tag{5.5}$$

- System 2:

$$\ddot{x} + x + 0.01\left(\dot{x} - \frac{\dot{x}^3}{3}\right) = 0, \tag{5.6}$$

$$x(0) = 0.01, \tag{5.7}$$

$$\dot{x}(0) = -4.999875E-5. \tag{5.8}$$

Analytical solution:

$$x(t) = a(t)\cos t, \tag{5.9}$$

where

$$a(t) = \frac{0.01e^{0.005t}}{\left(1+0.000025(e^{0.01t}-1)\right)^{\frac{1}{2}}}. \tag{5.10}$$

Note that system 2 is a special case of the famous *Rayleigh-equation* and its analytical solution has been derived by using the *Krylov-Bogoliubov-Formulae* [?].

All of the eight RBDF6 methods displayed in table 5.1 are tested using these two systems. Although all of these algorithms perform fairly well in simulating these systems, only the results of RBDF61 and RBDF66 will be shown. These two algorithms yield the best results overall. This might be explained by the better damping properties of the two methods (see chapter 3). Therefore, RBDF61 and RBDF66 are the best RBDF6 methods to compete against BDF6.

The relative error bound for the step-size control is chosen to be $tol_{rel} = 0.001$. Hence, the relative local truncation error (LTE) is compared to this value of tol_{rel} and, the result of this comparison determines the need for step-size change.

Of course the value of tol_{rel} is chosen to be much smaller in a real simulation —

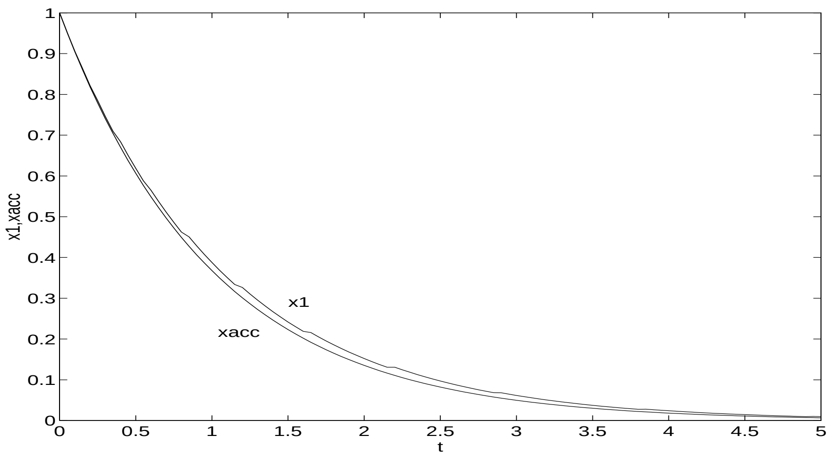
e.g. $tol_{rel} = 1E-6$. However, to be able to more easily compare the different integration methods, the error is made larger.

In order to assess the simulation results the time functions of the numerical solution x and the global error $\epsilon = x - x_{anal}$, will be recorded where x_{anal} is the analytical solution of the system. Furthermore, the time behavior of the step-size h will be displayed.

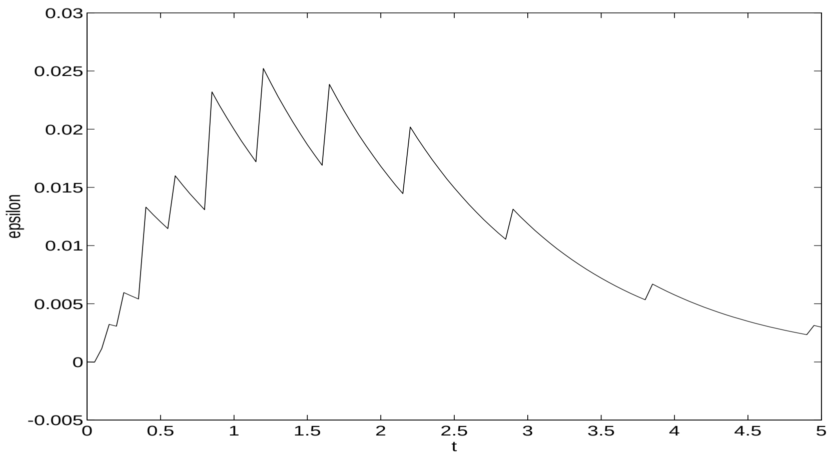
Eventually, the number of floating-point operations (*fop*) will be considered to compare the efficiency of the integration techniques.

At first, system 1 is integrated by BDF6. The simulation results are shown in figure 5.14. The MATLAB-simulation over 5 time units with BDF6 takes 59026 floating point operations (*fop*s). Recall that system 1 is stable — both eigenvalues have negative real parts. Figure 5.14 shows that the integration by BDF6 is stable, too, since the global error is decreasing with time. This seems to be clear because both eigenvalues being multiplied with an arbitrary step-size h still lie within the stability domain of BDF6.

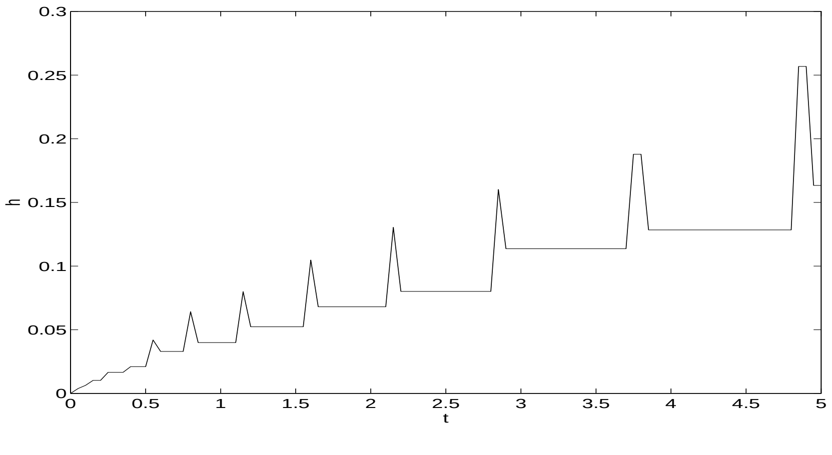
Now the same system is simulated with RBDF61 and RBDF66. Figures 5.15 and 5.16 display the results.



Analytical ($x_{1,anal}$) and numerical solution (x_1)

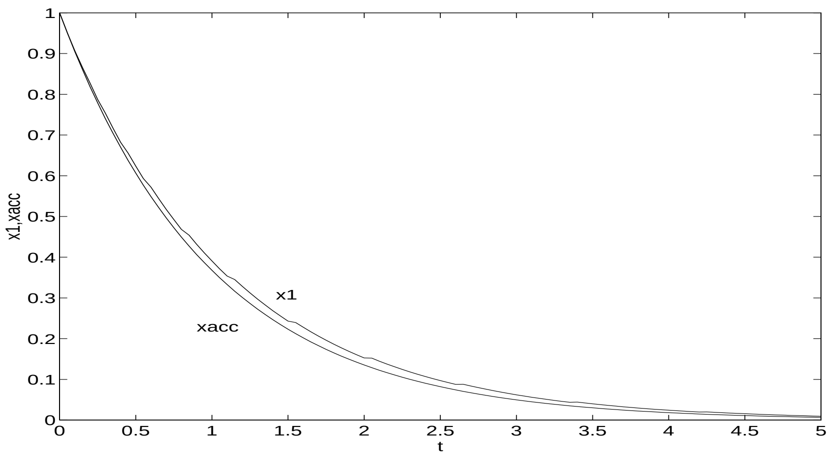


Global error

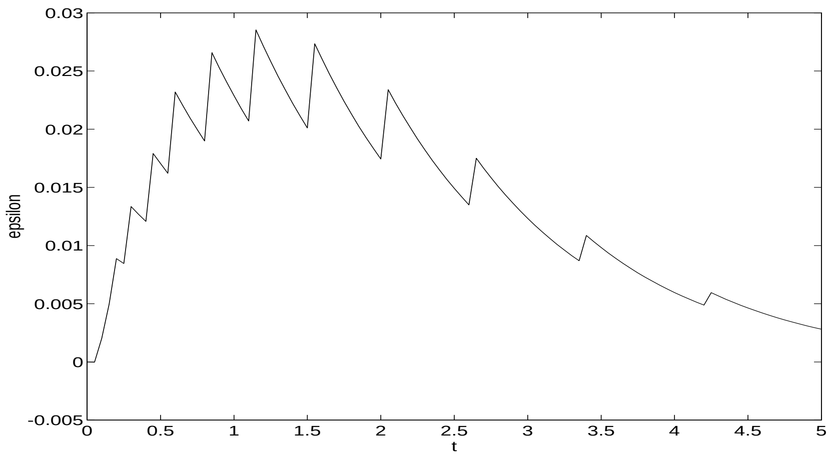


Step-size h

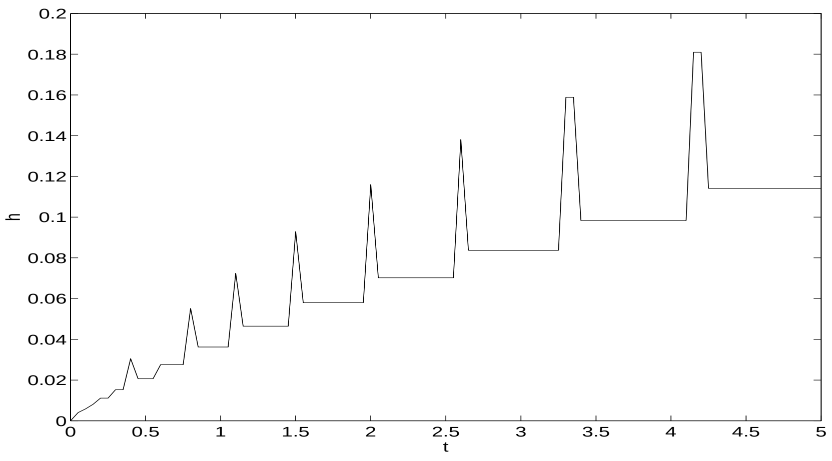
Figure 5.14: Integration of system 1 by BDF6



Analytical ($x_{1,anal}$) and numerical solution (x_1)

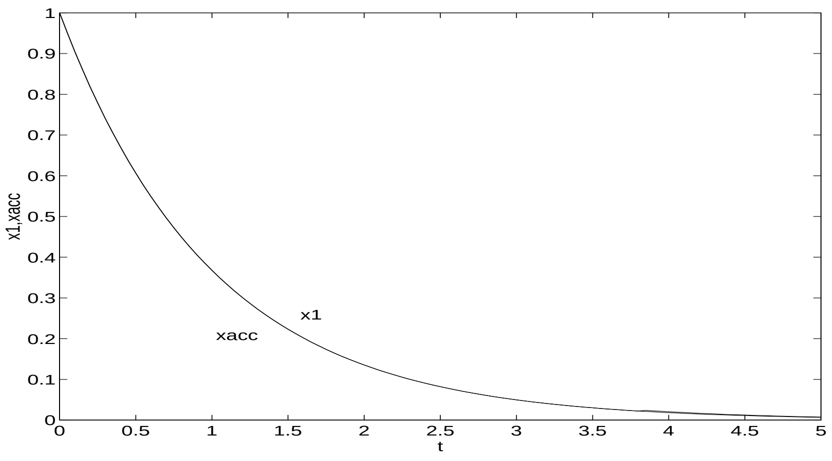


Global error

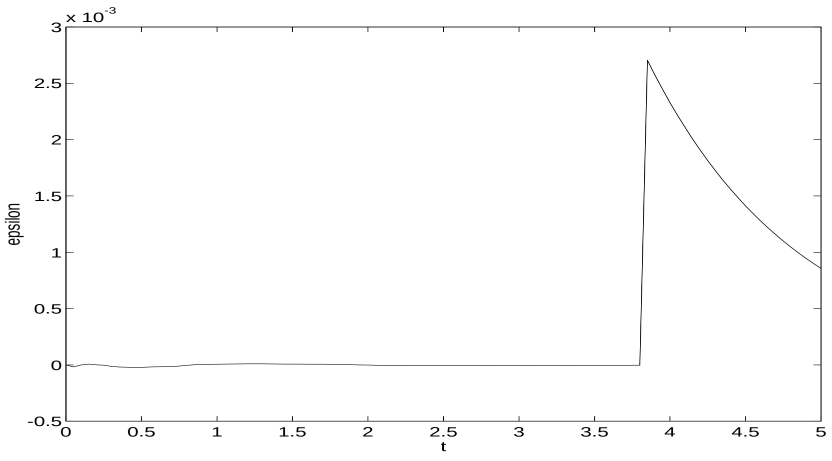


Step-size h

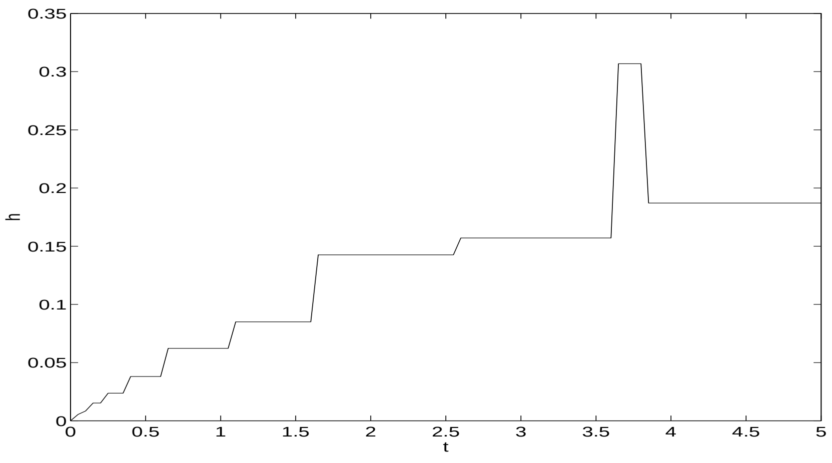
Figure 5.15: Integration of system 1 by RBDF61



Analytical ($x_{1,anal}$) and numerical solution (x_1)



Global error

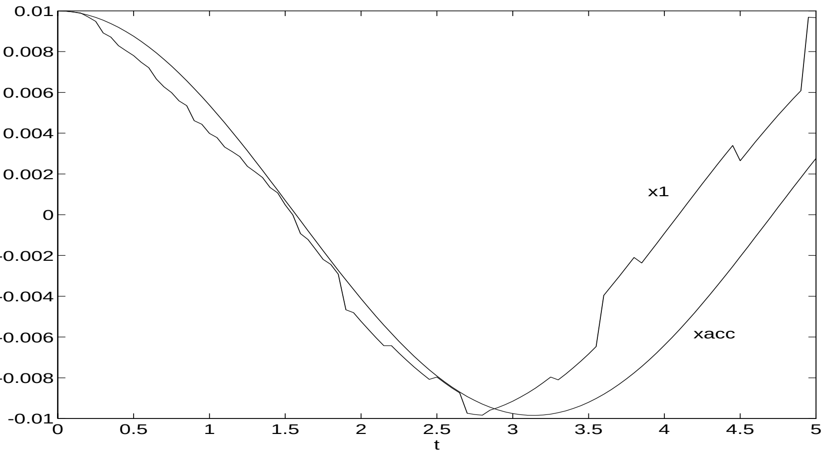


Step-size h

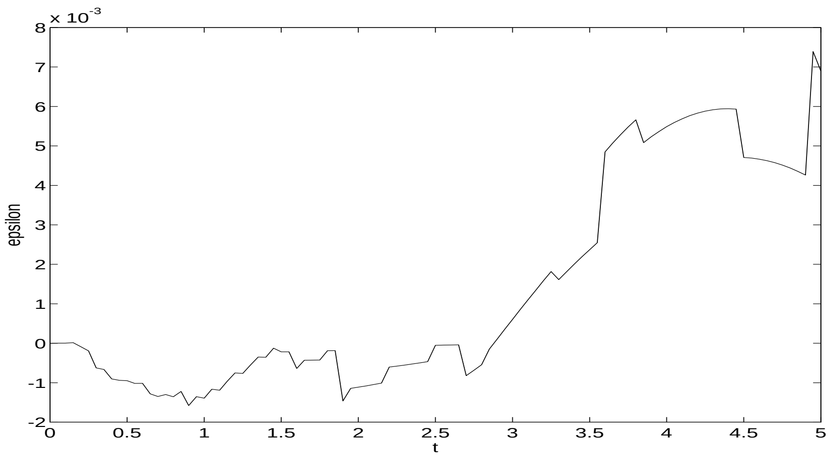
Figure 5.16: Integration of system 1 by RBDF66

When integrating system 1 with RBDF61 over 5 time units 68441 *fop*s are needed, whereas RBDF66 only needs 48634 *fop*s.
By comparing the results displayed in figures 5.15 and 5.16 it can be seen that RBDF66 integrates much more accurately than RBDF61 and BDF6. Moreover, RBDF66 needs 20% - 30% less floating point operations than its competitors.

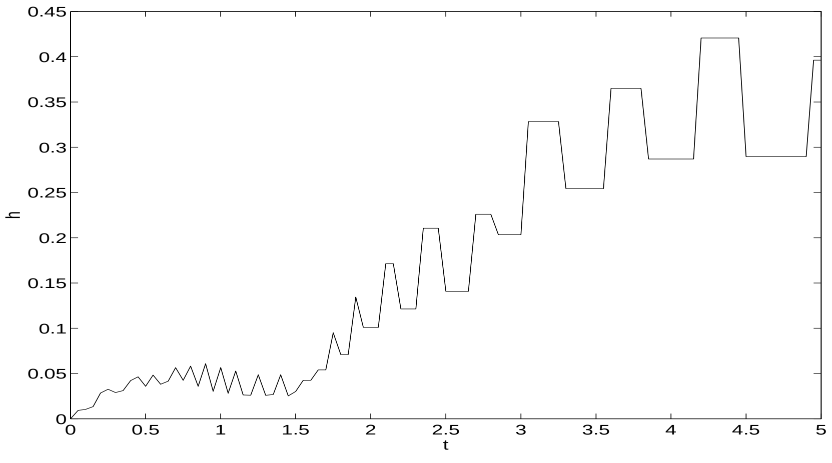
The nonlinear Rayleigh-equation (system 2) is integrated with BDF6, RBDF61 and RBDF66 which yields the results depicted in figures 5.17 through 5.19.



Analytical ($x_{1,anal}$) and numerical solution (x_1)

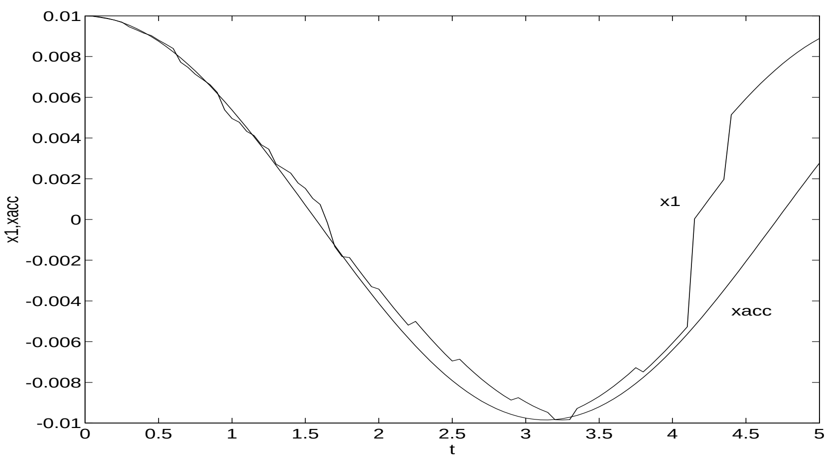


Global error

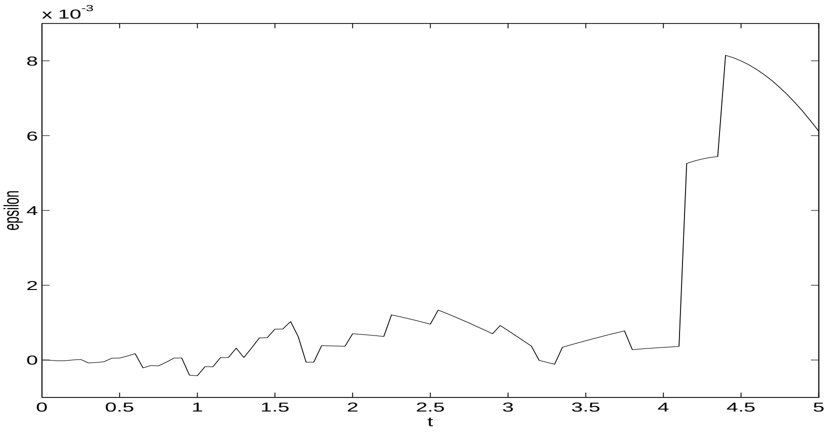


Step-size h

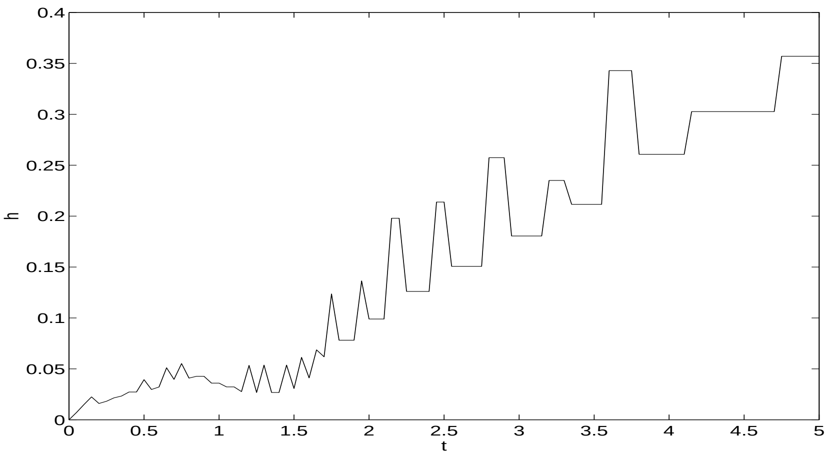
Figure 5.17: Integration of system 2 by BDF6



Analytical ($x_{1,anal}$) and numerical solution (x_1)

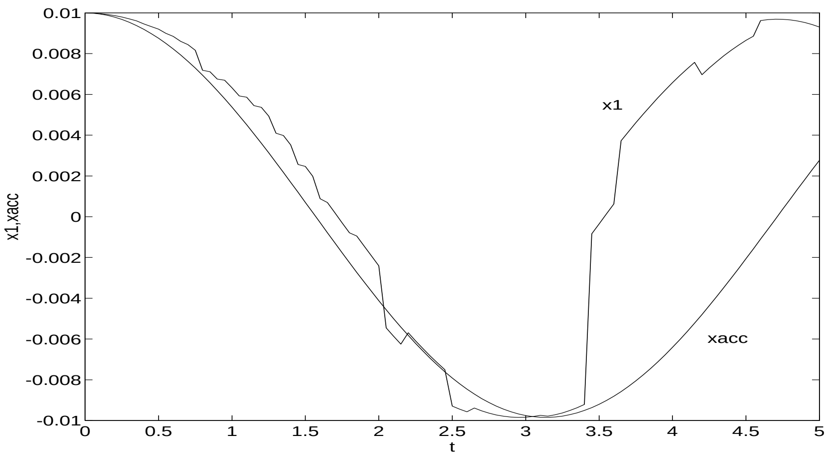


Global error

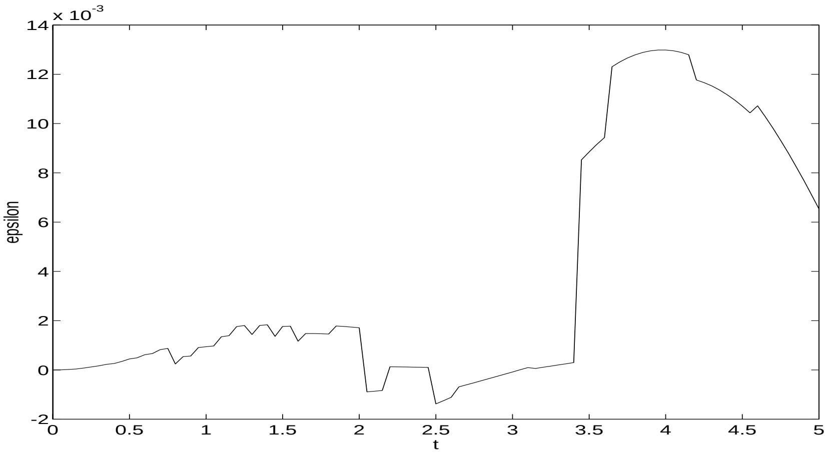


Step-size h

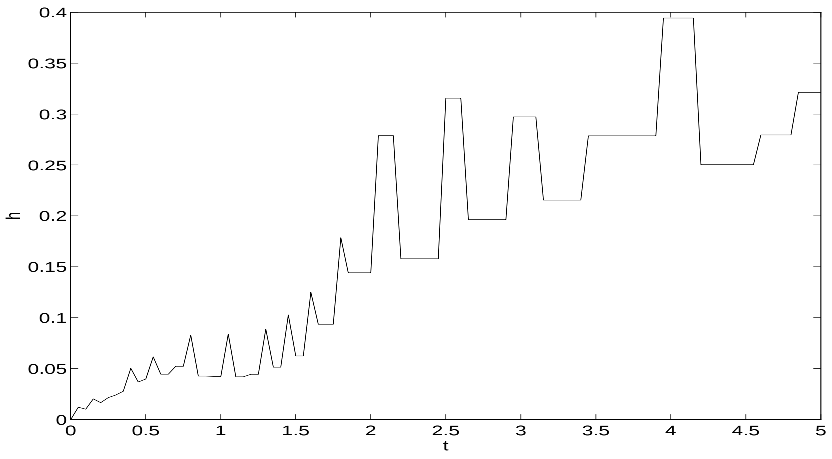
Figure 5.18: Integration of system 2 by RBDF61



Analytical ($x_{1,anal}$) and numerical solution (x_1)



Global error



Step-size h

Figure 5.19: Integration of system 2 by RBDF66

This time BDF6 needs 139229 *fop*s, RBDF61 uses 142983 *fop*s and RBDF66 applies 131570 *fop*s. It can be seen that RBDF66 is less accurate than the other two methods but it is still the fastest one of the three integrators.
These results might be explained by the better stability properties of RBDF61 and RBDF66 in comparison with BDF6 since in terms of all the other properties — e.g. damping, error constant — the two RBDF6 methods are worse. However, the following chapter will show that this assumption doesn't hold.
In figures 5.17 through 5.19 it can be seen that the integration is slightly unstable. This is due to the fact that the step-size in this implementation is usually changed only after each $(n+1)^{th}$ step, where n is the order of the integration method — recall that this has been done to save computation time. In addition, the bound tol_{rel} for the relative error has been chosen to be rather large.
Thus, the phenomenon of instability is a problem of the implementation, rather than a problem of the integration algorithm itself. The unstable behavior can easily be removed by choosing a smaller value for tol_{rel} and by changing the step-size more frequently.

Finally, the three methods are analyzed in the frequency domain by means of the Bode plot. As mentioned in chapter 3, three test systems (stable, marginally stable, unstable) of the kind $\dot{\underline{x}} = A\underline{x}$ are chosen that don't change their stability properties when integrated by the three methods.

The system matrix of the stable system is chosen to be

$$A = \begin{pmatrix} -10 & 0 \\ 0 & -10 \end{pmatrix}, \tag{5.11}$$

where the eigenvalues are $\lambda_{1,2} = -10$.

The marginally stable system has the system matrix

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \tag{5.12}$$

which corresponds to the eigenvalues $\lambda_1 = -i, \lambda_2 = i$.

The unstable system is characterized by the system matrix

$$A = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix} \tag{5.13}$$

with the eigenvalues $\lambda_{1,2} = 10$.

The stable system is integrated with the step-size $h = 1$. Note that $\lambda_i h = -10$ lies within the stability region of BDF6, RBDF61 and RBDF66.

Figure 5.20 shows the Bode plots that correspond to this stable integration. They have been produced using the MATLAB-command "DBODE."

Figure 5.20 shows that RBDF66 is better than RBDF61 with respect to low-pass filter characteristics and that RBDF61 is still better than BDF6.

The step-size $h = 0.01$ is chosen when simulating the marginally stable system. Figure 5.21 displays the corresponding Bode plots.

It can be seen in figure 5.21 that the gain has its maximum at the frequency $\omega = 1$ which is known as the *resonance frequency.*

Concerning filter properties this plot doesn't allow us to make clear statements.

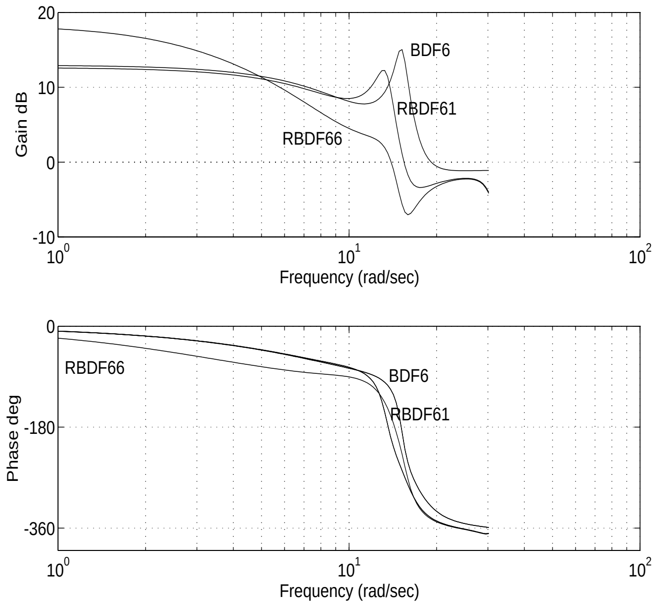


Figure 5.20: Bode plot of a stable system integrated by BDF6, RBDF61and RBDF66.

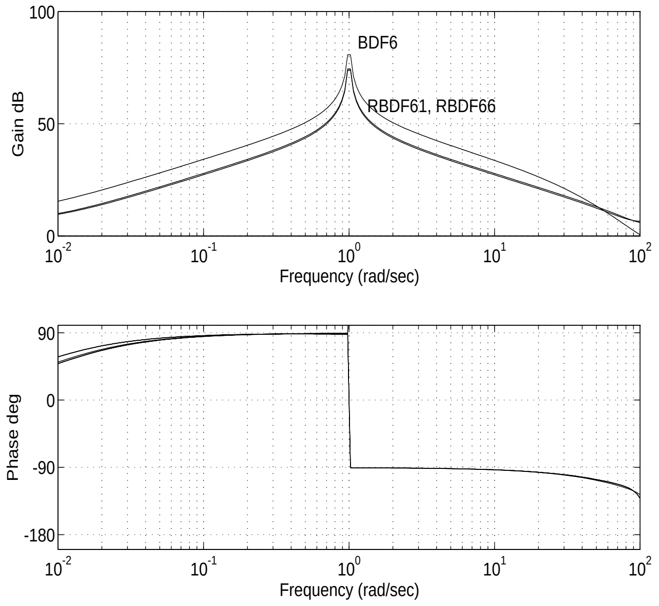


Figure 5.21: Bode plot of a marginally stable system integrated by BDF6, RBDF61 and RBDF66.

Finally, the unstable system is integrated with the step-size $h = 0.1$ which results in the Bode plots depicted in figure 5.22.

In this case BDF6 and RBDF61 behave like high-pass filters, i.e. they rather let high fre-

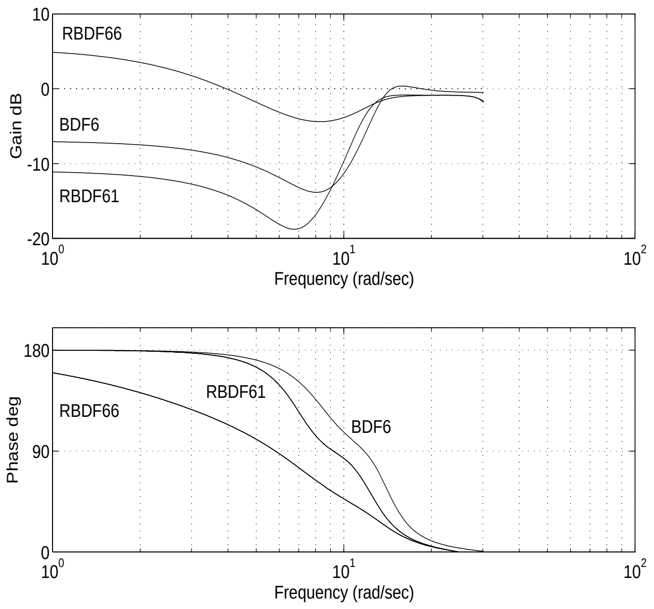


Figure 5.22: Bode plot of an unstable system integrated by BDF6, RBDF61 and RBDF66.

quent signals pass. RBDF66 still shows low-pass filter characteristics.

To sum this discussion up, the RBDF6 methods may lend themselves better to integrate systems with input noise than BDF6 since they show stronger low-pass filter behavior. Since inaccuracies (numerical errors) can be interpreted as high frequency noise, they shall be filtered out better by the RBDF6 methods.

CHAPTER 6

RBDF of 7th order (RBDF7)

6.1 Derivation of the algorithms

The procedure described in section 5.1 is applied to find 7^{th} order RBDF methods. Once again, the first step is to determine those algorithms that satisfy "minimum stability requirements."

Table 6.1 shows the number of RBDF7 methods that can be found for different numbers of applied data points.

Note that there are many more RBDF7 methods with the mentioned requirements than RBDF6 algorithms (cf. table 5.1).

Unlike the preceding chapter where BDF6 was the competitor, we don't have any comparison within the 7^{th} order range since BDF7 is unstable. Again, only those RBDF7 methods are accepted that are stiffly stable according to Gear's definition 1.3.3. 174 RBDF7 were determined that possess this property.

In chapter 5 it has been shown that it is more restrictive to require the numerical damping $\hat{\sigma}_d$ to go to infinity as σ_d goes to infinity than to require a large critical damping $\sigma_{d,crit}$. Therefore, the logarithmic damping plot is regarded first and the RBDF methods with the best damping curves are extracted. This results in 15 RBDF algorithms of 7^{th} order whose

Number of data points	Number of RBDF7 algorithms
9	268
10	278
11	189
12	77
13	14
14	-

Table 6.1: Number of 7th order RBDF algorithms satisfying minimum stability requirements

damping plots are displayed in figures 6.5 and 6.6.

The tables 6.2 and 6.3 tabulate these algorithms, their formulae and error constants.

The corresponding stability domains are depicted in figures 6.1 and 6.2, and the damping plots can be found in figures 6.3 and 6.4.

Figures 6.5 and 6.6 show the logarithmic damping plots.

6.2 Analysis

By considering again the definition of stiffly stable systems (figure 1.3) the values for a and c are measured for each RBDF7 algorithm. Moreover, we determine the value $\sigma_{d,crit}$ in the damping plot, where the numerical damping $\hat{\sigma}_d$ starts to deviate from the analytical damping σ_d, and $\hat{\sigma}_d^*(= \hat{\sigma}_d(\sigma_d = 1\text{E+}6))$ in the logarithmic damping plot and list all these values in table 6.4, together with the error constant C_{err}.

A good method would have a small value a, a large value c, both $\sigma_{d,crit}$ and σ_d^* would be large and $|C_{err}|$ would be small.

Name	Number of data points	Formula	Error constant C_{err}
RBDF71	9	$x_{k+1} = \frac{948}{2257}hf_{k+1} + \frac{454}{201}x_k - \frac{1476}{683}x_{k-1} + \frac{653}{869}x_{k-2} + \frac{1151}{1240}x_{k-3} - \frac{452}{357}x_{k-4} + \frac{1224}{2285}x_{k-5} - \frac{193}{3802}x_{k-7} + \frac{42}{10813}x_{k-9}$	-0.1765
RBDF72	9	$x_{k+1} = \frac{372}{869}hf_{k+1} + \frac{443}{204}x_k - \frac{3349}{1814}x_{k-1} + \frac{1345}{4324}x_{k-2} + \frac{703}{648}x_{k-3} - \frac{658}{715}x_{k-4} + \frac{298}{909}x_{k-6} - \frac{845}{6228}x_{k-7} + \frac{83}{12165}x_{k-9}$	-0.2249
RBDF73	9	$x_{k+1} = \frac{1363}{3123}hf_{k+1} + \frac{1022}{489}x_k - \frac{1165}{733}x_{k-1} + \frac{421}{4469}x_{k-2} + \frac{801}{1027}x_{k-3} - \frac{4831}{4944}x_{k-5} + \frac{810}{947}x_{k-6} - \frac{407}{1542}x_{k-7} + \frac{119}{10830}x_{k-9}$	-0.2851
RBDF74	9	$x_{k+1} = \frac{147}{344}hf_{k+1} + \frac{4431}{2014}x_k - \frac{1295}{636}x_{k-1} + \frac{584}{647}x_{k-2} + \frac{997}{3006}x_{k-4} - \frac{883}{967}x_{k-5} + \frac{1885}{2599}x_{k-6} - \frac{417}{1900}x_{k-7} + \frac{153}{16820}x_{k-9}$	-0.2433
RBDF75	9	$x_{k+1} = \frac{133}{305}hf_{k+1} + \frac{675}{323}x_k - \frac{1269}{808}x_{k-1} + \frac{1138}{1143}x_{k-3} - \frac{504}{1777}x_{k-4} - \frac{1065}{1417}x_{k-5} + \frac{2668}{3553}x_{k-6} - \frac{337}{1399}x_{k-7} + \frac{47}{4554}x_{k-9}$	-0.2780
RBDF76	9	$x_{k+1} = \frac{503}{1187}hf_{k+1} + \frac{322}{145}x_k - \frac{2357}{1161}x_{k-1} + \frac{665}{1142}x_{k-2} + \frac{1313}{1343}x_{k-3} - \frac{2125}{1834}x_{k-4} + \frac{1176}{2749}x_{k-5} - \frac{107}{3718}x_{k-8} + \frac{51}{5581}x_{k-9}$	-0.2052
RBDF77	9	$x_{k+1} = \frac{647}{1498}hf_{k+1} + \frac{2005}{939}x_k - \frac{2119}{1225}x_{k-1} + \frac{364}{1983}x_{k-2} + \frac{1268}{1165}x_{k-3} - \frac{1254}{1511}x_{k-4} + \frac{925}{4782}x_{k-6} - \frac{53}{931}x_{k-8} + \frac{33}{2065}x_{k-9}$	-0.2608
RBDF78	9	$x_{k+1} = \frac{473}{1069}hf_{k+1} + \frac{5326}{2621}x_k - \frac{709}{507}x_{k-1} - \frac{349}{2482}x_{k-2} + \frac{679}{778}x_{k-3} - \frac{7103}{8625}x_{k-5} + \frac{987}{1849}x_{k-6} - \frac{188}{1825}x_{k-8} + \frac{131}{4855}x_{k-9}$	-0.3424

Table 6.2: RBDF7 methods — 1. part

Name	Number of data points	Formula	Error constant C_{err}
RBDF79	9	$x_{k+1} = \frac{512}{1163}hf_{k+1} + \frac{1162}{565}x_k - \frac{970}{651}x_{k-1} + \frac{1263}{1612}x_{k-3} - \frac{172}{7307}x_{k-4} - \frac{699}{931}x_{k-5} + \frac{571}{1152}x_{k-6} - \frac{559}{5769}x_{k-8} + \frac{293}{11507}x_{k-9}$	-0.3288
RBDF710	9	$x_{k+1} = \frac{511}{1132}hf_{k+1} + \frac{2136}{1087}x_k - \frac{1049}{844}x_{k-1} - \frac{208}{1401}x_{k-2} + \frac{620}{1011}x_{k-3} - \frac{1735}{2353}x_{k-6} + \frac{1087}{1205}x_{k-7} - \frac{1586}{3723}x_{k-8} + \frac{847}{11393}x_{k-9}$	-0.4700
RBDF711	9	$x_{k+1} = \frac{601}{1348}hf_{k+1} + \frac{1077}{529}x_k - \frac{1311}{850}x_{k-1} + \frac{293}{685}x_{k-2} + \frac{455}{1401}x_{k-4} - \frac{1424}{1683}x_{k-6} + \frac{2381}{2466}x_{k-7} - \frac{1069}{2424}x_{k-8} + \frac{33}{437}x_{k-9}$	-0.4504
RBDF712	9	$x_{k+1} = \frac{938}{2101}hf_{k+1} + \frac{891}{443}x_k - \frac{1277}{920}x_{k-1} + \frac{151}{238}x_{k-3} - \frac{345}{2531}x_{k-4} - \frac{447}{812}x_{k-6} + \frac{460}{643}x_{k-7} - \frac{579}{1663}x_{k-8} + \frac{192}{3103}x_{k-9}$	-0.4221
RBDF713	9	$x_{k+1} = \frac{326}{751}hf_{k+1} + \frac{1016}{473}x_k - \frac{725}{381}x_{k-1} + \frac{587}{713}x_{k-2} + \frac{70}{891}x_{k-5} - \frac{1913}{3775}x_{k-6} + \frac{432}{733}x_{k-7} - \frac{3220}{11559}x_{k-8} + \frac{598}{12199}x_{k-9}$	-0.3424
RBDF714	9	$x_{k+1} = \frac{914}{2053}hf_{k+1} + \frac{931}{461}x_k - \frac{2045}{1456}x_{k-1} + \frac{918}{1465}x_{k-3} - \frac{2277}{7079}x_{k-5} - \frac{170}{903}x_{k-6} + \frac{397}{810}x_{k-7} - \frac{368}{1347}x_{k-8} + \frac{388}{7551}x_{k-9}$	-0.3993
RBDF715	9	$x_{k+1} = \frac{503}{1104}hf_{k+1} + \frac{857}{443}x_k - \frac{2192}{1837}x_{k-1} + \frac{209}{222}x_{k-4} - \frac{318}{419}x_{k-5} - \frac{811}{1867}x_{k-6} + \frac{2273}{2578}x_{k-7} - \frac{1175}{2599}x_{k-8} + \frac{285}{3518}x_{k-9}$	-0.5153

Table 6.3: RBDF7 methods — 2. part

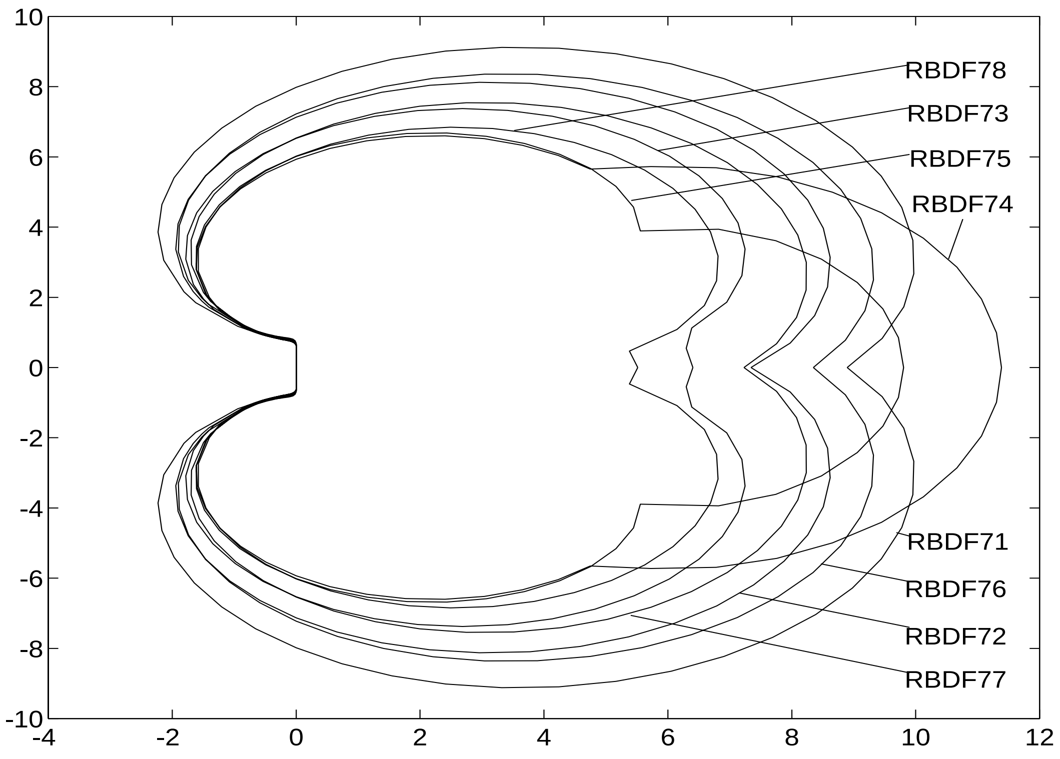


Figure 6.1: Stability domains of RBDF7 — 1. part

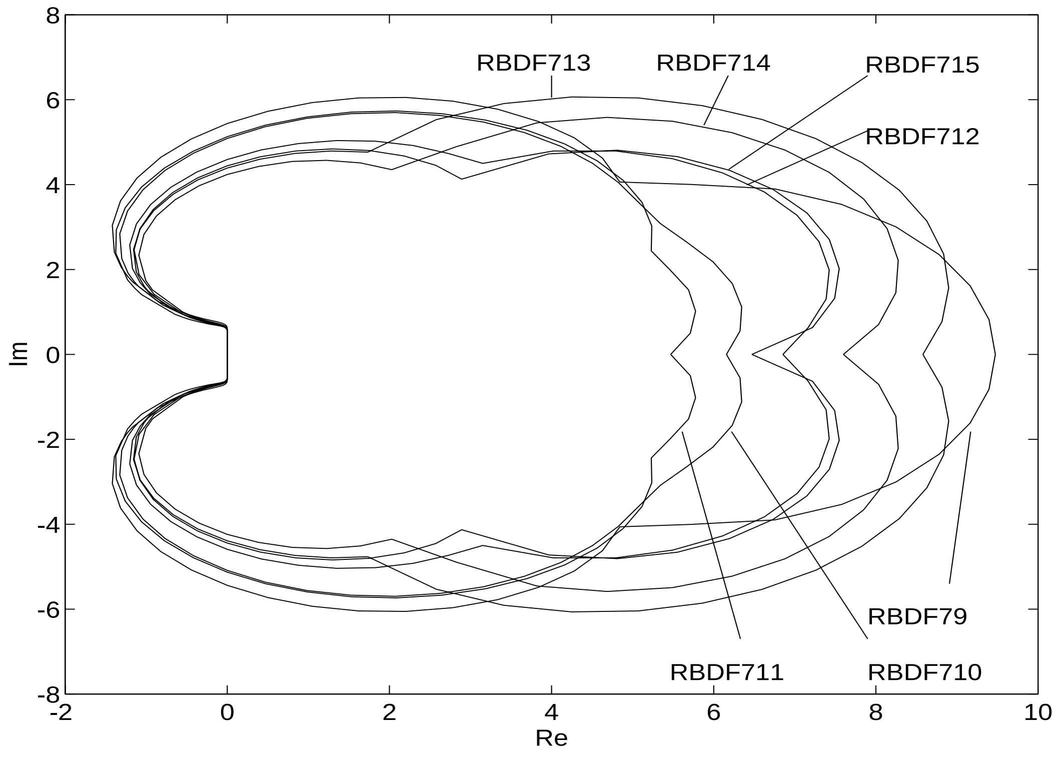


Figure 6.2: Stability domains of RBDF7 — 2. part

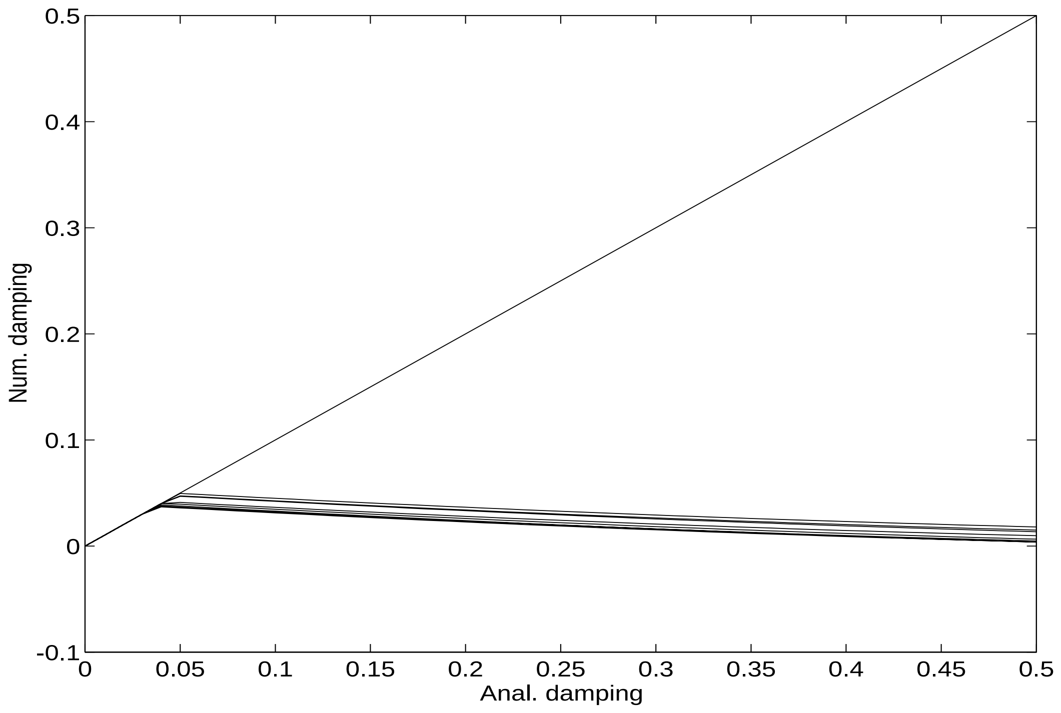


Figure 6.3: Damping plots of RBDF7 — 1. part (RBDF71 through RBDF78)

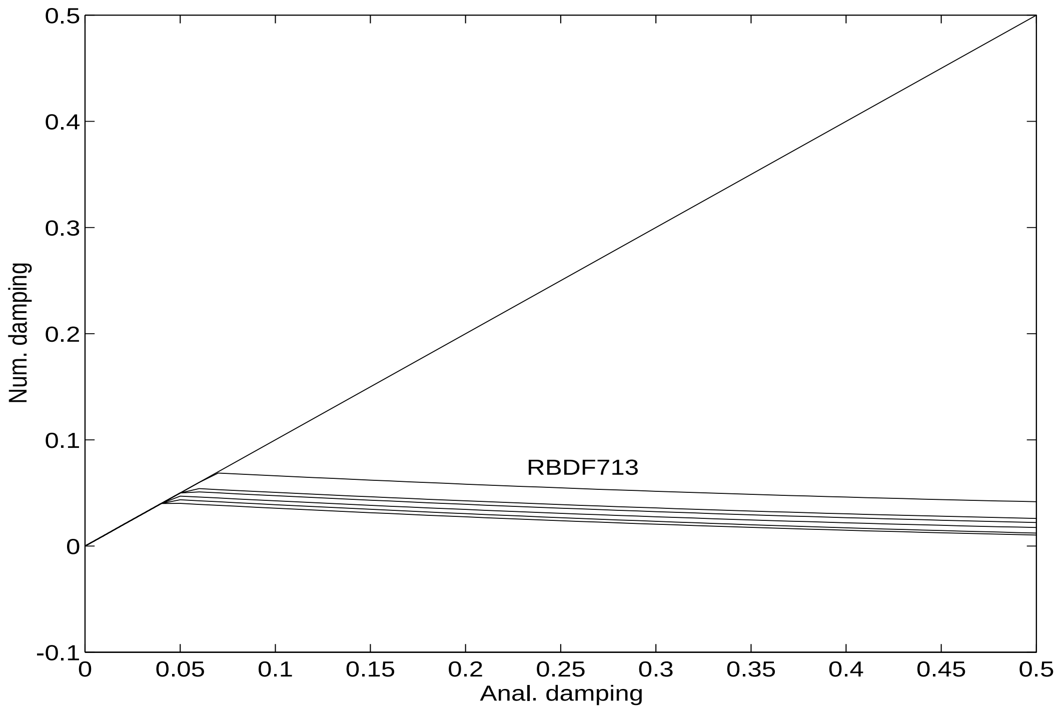


Figure 6.4: Damping plots of RBDF7 — 2. part (RBDF79 through RBDF715)

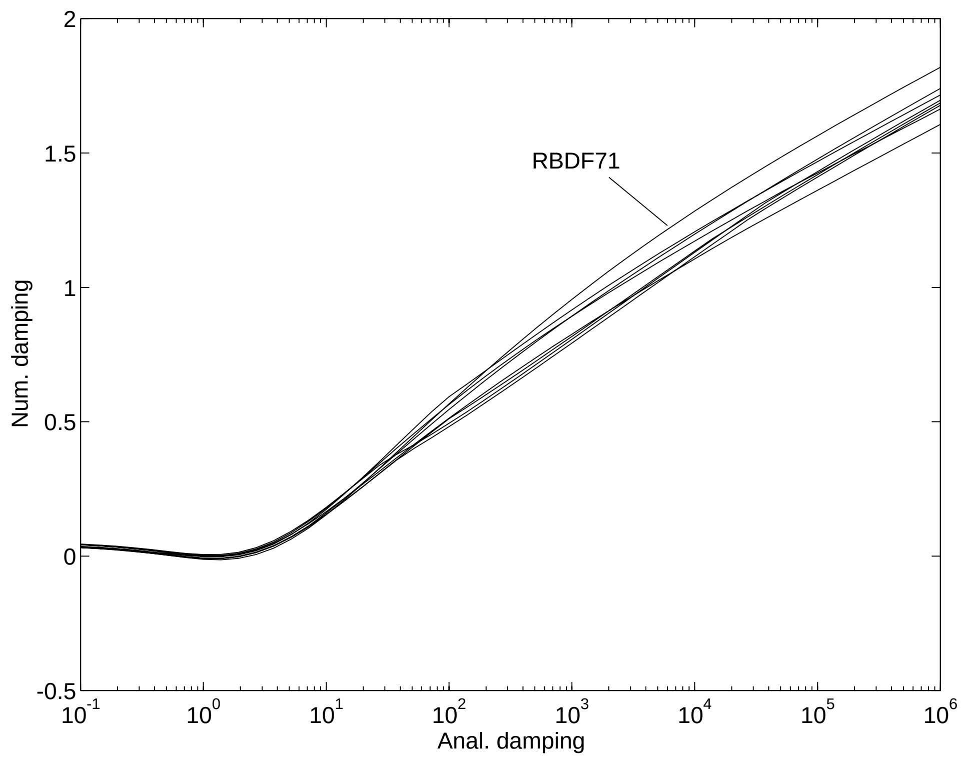


Figure 6.5: Logarithmic damping plots of RBDF7 — 1. part (RBDF71 through RBDF78)

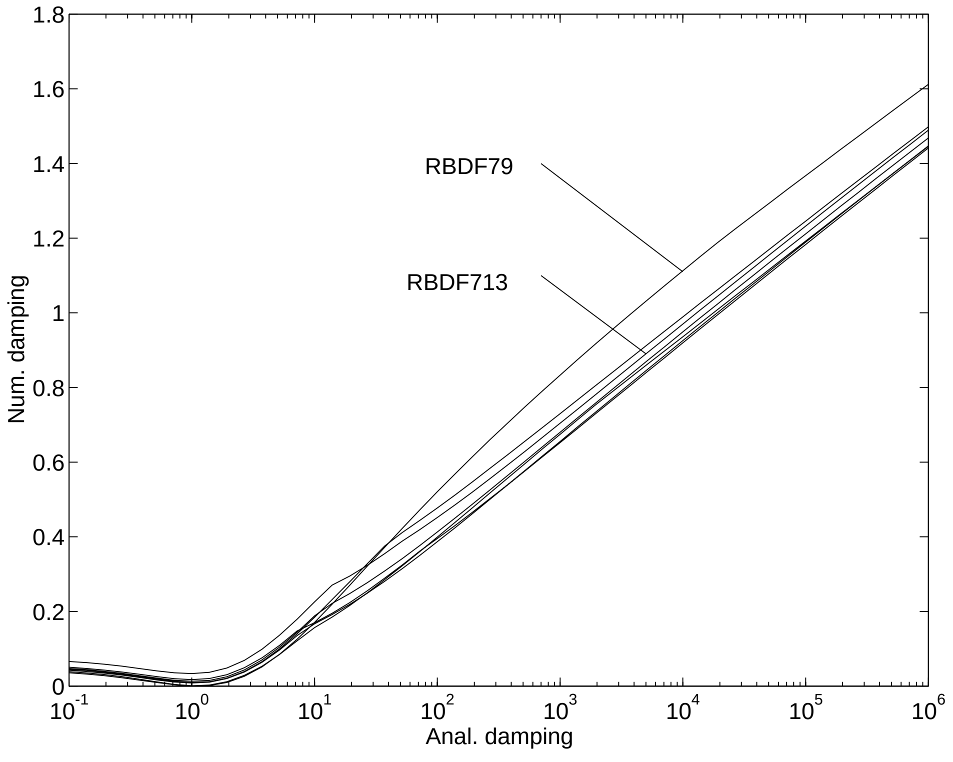


Figure 6.6: Logarithmic damping plots of RBDF7 — 2. part (RBDF79 through RBDF715)

Name	a	c	$\sigma_{d,crit}$	$\hat{\sigma}_d^*$	C_{err}
RBDF71	1.90	0.81	0.04	1.83	-0.1765
RBDF72	1.94	0.73	0.04	1.74	-0.2249
RBDF73	1.82	0.72	0.04	1.68	-0.2851
RBDF74	1.64	0.78	0.05	1.70	-0.2433
RBDF75	1.59	0.72	0.04	1.69	-0.2780
RBDF76	1.90	0.73	0.05	1.72	-0.2052
RBDF77	1.67	0.72	0.05	1.67	-0.2608
RBDF78	1.59	0.65	0.04	1.60	-0.3424
RBDF79	1.43	0.65	0.04	1.62	-0.3288
RBDF710	1.31	0.63	0.04	1.45	-0.4700
RBDF711	1.37	0.63	0.05	1.45	-0.4504
RBDF712	1.20	0.63	0.05	1.47	-0.4221
RBDF713	1.14	0.59	0.07	1.49	-0.3424
RBDF714	1.06	0.63	0.05	1.50	-0.3993
RBDF715	1.14	0.63	0.04	1.45	-0.5153

Table 6.4: Data of RBDF7 algorithms

No RBDF7 method stands out from the others with respect to stability and to accuracy. However, two methods are chosen that are better than the other techniques:

For RBDF71, the values c and $\hat{\sigma}_d^*$ are largest and $|C_{err}|$ is smallest.

RBDF713 is the best method among those that have good values for a and c because $\sigma_{d,crit}$ is largest for RBDF713, while $\hat{\sigma}_d^*$ and $|C_{err}|$ still have decent values.

6.3 Simulation results

The systems 1 and 2 introduced in chapter 5 are integrated with RBDF71 and RBDF713. To save computation time, the relative error tol_{rel} used for step-size control is assigned a value of 0.05.

Figures 6.7 and 6.8 display the results of the simulation of system 1 with RBDF71 and RBDF713.

It can be seen in these figures that RBDF71 integrates system 1 more accurately than RBDF713 does.

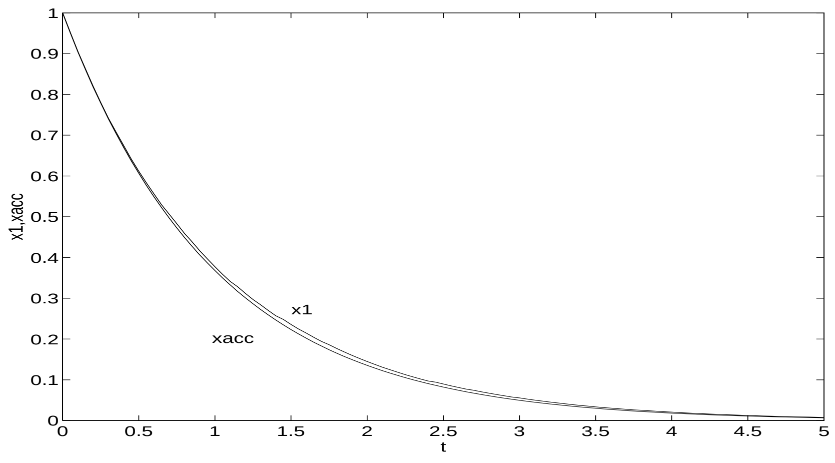
Moreover, RBDF71 integrates system 1 faster since it only uses 249355 *fop*s during the simulation while RBDF713 needs to execute 278368 *fop*s.

In figures 6.9 and 6.10 the results of the integration of the nonlinear system 2 are illustrated.

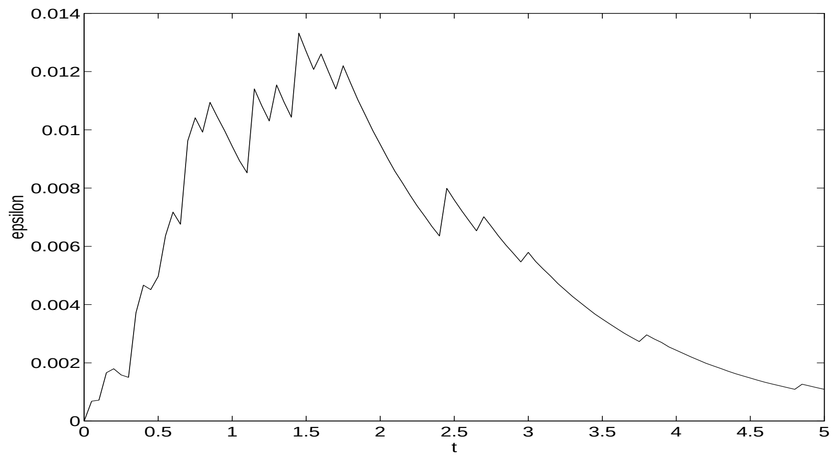
Again, these results show that RBDF71 behaves better than RBDF713, both with respect to accuracy and to efficiency — RBDF71 needs 599412 *fop*s and RBDF713 uses 672005 *fop*s.

One might be tempted to think that the values $\hat{\sigma}_d^*$ and C_{err} are responsible for this behavior since RBDF71 has better values $\hat{\sigma}_d^*$ and C_{err} than RBDF713 whereas the values a and $\sigma_{d,crit}$ are worse (see table 6.4).

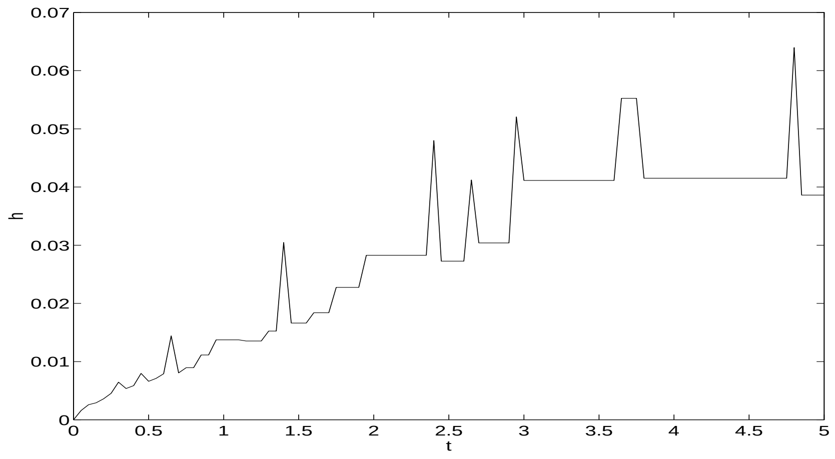
However, the results of chapter 5 don't confirm this assumption. Thus, only by considering all of the properties of a numerical integration method it can be stated that this method is better than another technique.



Analytical ($x_{1,anal}$) and numerical solution (x_1)

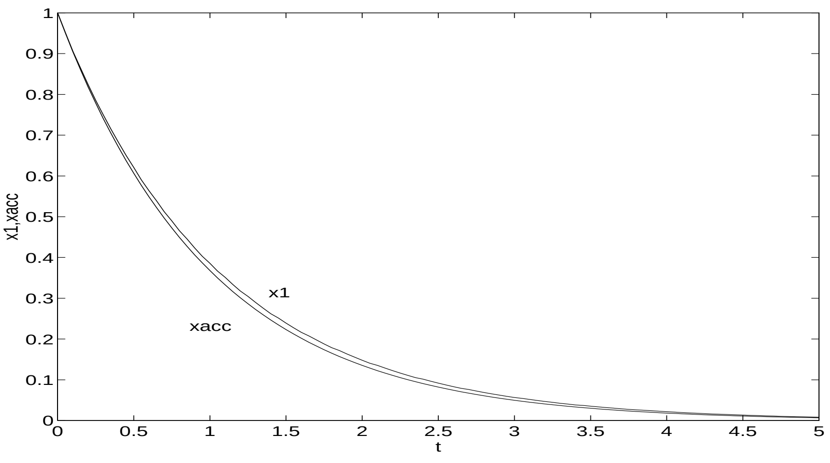


Global error

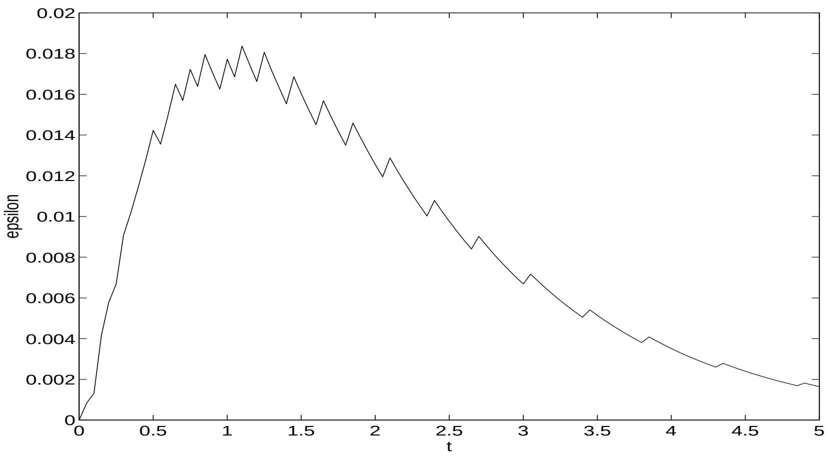


Step-size h

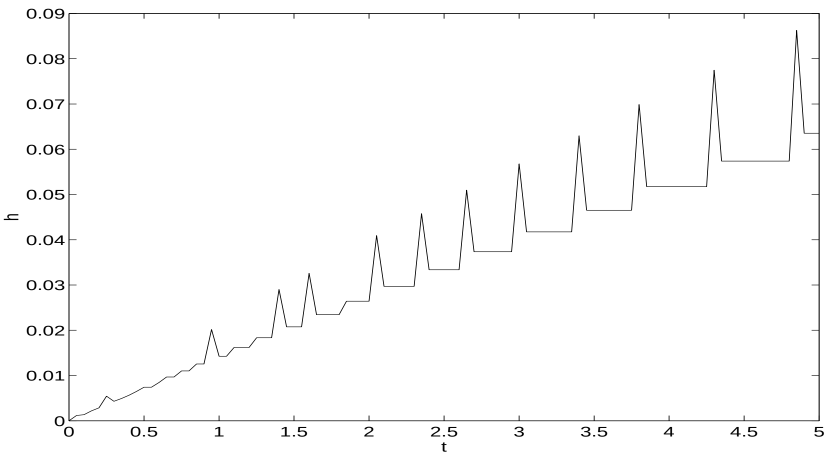
Figure 6.7: Integration of system 1 by RBDF71



Analytical ($x_{1,anal}$) and numerical solution (x_1)

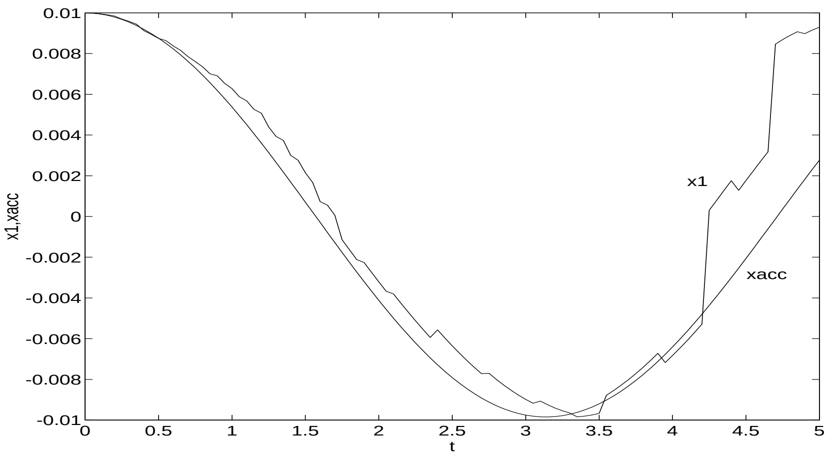


Global error

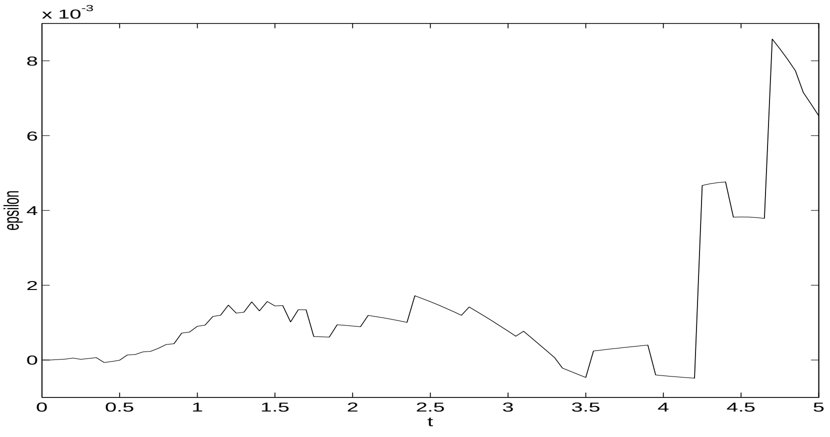


Step-size h

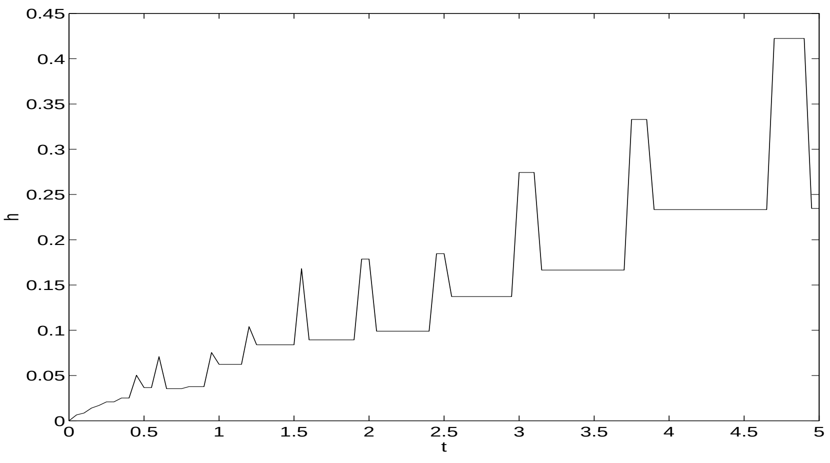
Figure 6.8: Integration of system 1 by RBDF713



Analytical ($x_{1,anal}$) and numerical solution (x_1)

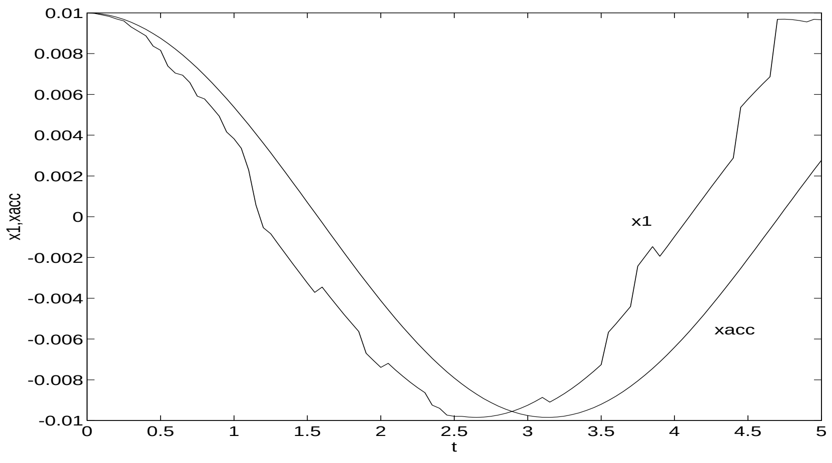


Global error

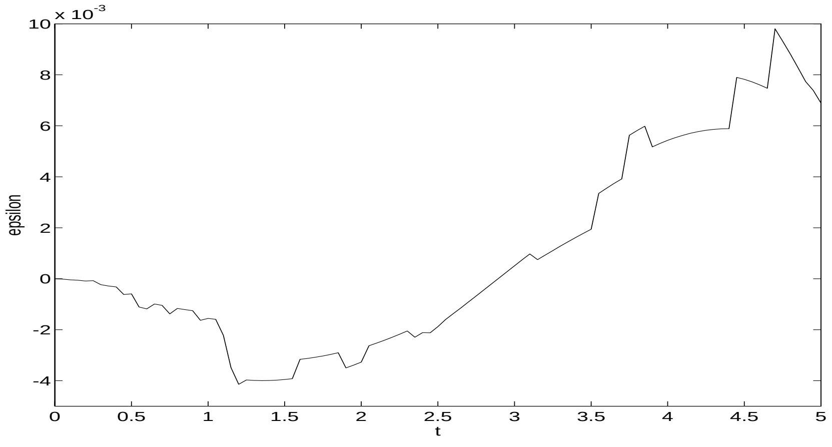


Step-size h

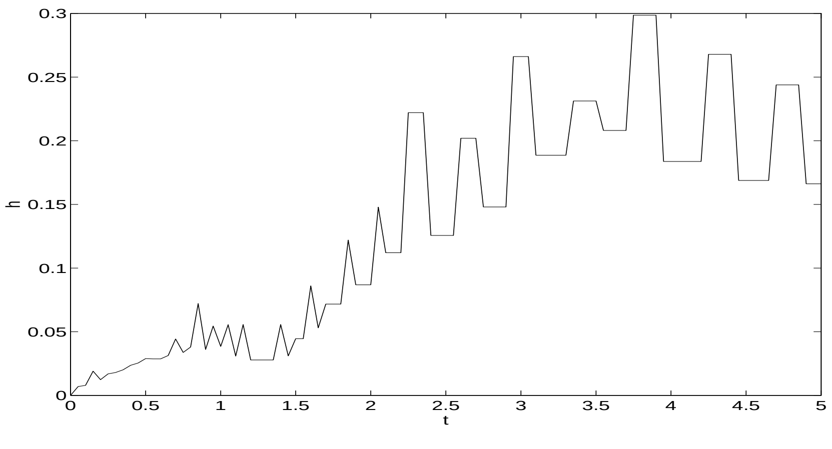
Figure 6.9: Integration of system 2 by RBDF71



Analytical ($x_{1,anal}$) and numerical solution (x_1)



Global error



Step-size h

Figure 6.10: Integration of system 2 by RBDF713

CHAPTER 7

Conclusions

In chapter 3, two new analysis methods for assessing numerical integration algorithms have been introduced:

1. Order star – accuracy region analysis

2. Bode plot analysis

The RBDF6 techniques presented in chapter 5 can compete with BDF6 in integrating stiff systems, a fact that has been verified in numerous simulations. In most of the cases considered in this thesis, the two algorithms RBDF61 and RBDF66 outperform BDF6, and because of the better shape of their Bode plot it may be concluded that they would do better than BDF6 when integrating a stiff system with noise input. However, this hasn't been proven so far and might be a topic of further research.

As seen in chapter 6, the RBDF7 methods offer a way to integrate stiff systems of ordinary differential equations with an accuracy order of 7 based on backward difference formulae. This hasn't been possible so far since there don't exist stable BDF methods of order higher than 6.

The regression approach introduced in this thesis to find stiffly stable integration algorithms of accuracy order seven, could be used to identify new stiffly stable RBDF methods of even higher accuracy orders.

Such methods would be of much interest for use in celestial dynamics where highly accurate integration of ordinary differential equations is required.

Appendix A

Nordsieck transformation matrices of different order

Appendix A contains transformation matrices **T** of different order that are used to calculate the Nordsieck vector.

1. $m = 2,\ n_g = 1$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}.$$

2. $m = 3,\ n_g = 2$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{2} & -2 & \frac{1}{2} \\ \frac{1}{2} & -1 & \frac{1}{2} \end{pmatrix}.$$

3. $m = 4,\ n_g = 3$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{11}{6} & -3 & \frac{3}{2} & -\frac{1}{3} \\ 1 & -\frac{5}{2} & 2 & -\frac{1}{2} \\ \frac{1}{6} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{6} \end{pmatrix}.$$

4. $m = 5,\ n_g = 4$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{25}{12} & -4 & 3 & -\frac{4}{3} & \frac{1}{4} \\ \frac{35}{24} & -\frac{13}{3} & \frac{19}{4} & -\frac{7}{3} & \frac{11}{24} \\ \frac{5}{12} & -\frac{3}{2} & 2 & -\frac{7}{6} & \frac{1}{4} \\ \frac{1}{24} & -\frac{1}{6} & \frac{1}{4} & -\frac{1}{6} & \frac{1}{24} \end{pmatrix}.$$

5. $m = 6,\ n_g = 5$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{137}{60} & -5 & 5 & -\frac{10}{3} & \frac{5}{4} & -\frac{1}{5} \\ \frac{15}{8} & -\frac{77}{12} & \frac{107}{12} & -\frac{13}{2} & \frac{61}{24} & -\frac{5}{12} \\ \frac{17}{24} & -\frac{71}{24} & \frac{59}{12} & -\frac{49}{12} & \frac{41}{24} & -\frac{7}{24} \\ \frac{1}{8} & -\frac{7}{12} & \frac{13}{12} & -1 & \frac{11}{24} & -\frac{1}{12} \\ \frac{1}{120} & -\frac{1}{24} & \frac{1}{12} & -\frac{1}{12} & \frac{1}{24} & -\frac{1}{120} \end{pmatrix}.$$

6. $m = 7,\ n_g = 6$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{49}{20} & -6 & \frac{15}{2} & -\frac{20}{3} & \frac{15}{4} & -\frac{6}{5} & \frac{1}{6} \\ \frac{203}{90} & -\frac{87}{10} & \frac{117}{8} & -\frac{127}{9} & \frac{33}{4} & -\frac{27}{10} & \frac{137}{360} \\ \frac{49}{48} & -\frac{29}{6} & \frac{461}{48} & -\frac{31}{3} & \frac{307}{48} & -\frac{13}{6} & \frac{15}{48} \\ \frac{35}{144} & -\frac{31}{24} & \frac{137}{48} & -\frac{121}{36} & \frac{107}{48} & -\frac{19}{24} & \frac{17}{144} \\ \frac{7}{240} & -\frac{1}{6} & \frac{19}{48} & -\frac{1}{2} & \frac{17}{48} & -\frac{2}{15} & \frac{1}{48} \\ \frac{1}{720} & -\frac{1}{120} & \frac{1}{48} & -\frac{1}{36} & \frac{1}{48} & -\frac{1}{120} & \frac{1}{720} \end{pmatrix}.$$

7. $m = 8,\ n_g = 7$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{363}{140} & -7 & \frac{21}{2} & -\frac{35}{3} & \frac{35}{4} & -\frac{21}{5} & \frac{7}{6} & -\frac{1}{7} \\ \frac{469}{180} & -\frac{223}{20} & \frac{879}{40} & -\frac{949}{36} & \frac{41}{2} & -\frac{201}{20} & \frac{1019}{360} & -\frac{7}{20} \\ \frac{967}{720} & -\frac{319}{45} & \frac{3929}{240} & -\frac{389}{18} & \frac{2545}{144} & -\frac{134}{15} & \frac{1849}{720} & -\frac{29}{90} \\ \frac{7}{18} & -\frac{111}{48} & \frac{71}{12} & -\frac{1219}{144} & \frac{22}{3} & -\frac{185}{48} & \frac{41}{36} & -\frac{7}{48} \\ \frac{23}{360} & -\frac{59}{144} & \frac{9}{8} & -\frac{247}{144} & \frac{113}{72} & -\frac{69}{80} & \frac{19}{72} & -\frac{5}{144} \\ \frac{1}{180} & -\frac{3}{80} & \frac{13}{120} & -\frac{25}{144} & \frac{1}{6} & -\frac{23}{240} & \frac{11}{360} & -\frac{1}{240} \\ \frac{1}{5040} & -\frac{1}{720} & \frac{1}{240} & -\frac{1}{144} & \frac{1}{144} & -\frac{1}{240} & \frac{1}{720} & -\frac{1}{5040} \end{pmatrix}.$$

8. $m = 9,\ n_g = 8$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{761}{280} & -8 & 14 & -\frac{56}{3} & \frac{35}{2} & -\frac{56}{5} & \frac{14}{3} & -\frac{8}{7} & \frac{1}{8} \\ \frac{29531}{10080} & -\frac{481}{35} & \frac{621}{20} & -\frac{2003}{45} & \frac{691}{16} & -\frac{141}{5} & \frac{2143}{180} & -\frac{103}{35} & \frac{363}{1120} \\ \frac{267}{160} & -\frac{349}{36} & \frac{18353}{720} & -\frac{2391}{60} & \frac{1457}{36} & -\frac{4891}{180} & \frac{561}{48} & -\frac{527}{180} & \frac{469}{1440} \\ \frac{1069}{1920} & -\frac{329}{90} & \frac{15289}{1440} & -\frac{268}{15} & \frac{10993}{576} & -\frac{1193}{90} & \frac{2803}{480} & -\frac{67}{45} & \frac{967}{5760} \\ \frac{9}{80} & -\frac{115}{144} & \frac{179}{72} & -\frac{213}{48} & \frac{179}{36} & -\frac{2581}{720} & \frac{13}{8} & -\frac{61}{144} & \frac{7}{144} \\ \frac{13}{960} & -\frac{73}{720} & \frac{239}{720} & -\frac{149}{240} & \frac{209}{288} & -\frac{391}{720} & \frac{61}{240} & -\frac{49}{720} & \frac{23}{2880} \\ \frac{1}{1120} & -\frac{1}{144} & \frac{119}{5040} & -\frac{231}{5040} & \frac{1}{18} & -\frac{217}{5040} & \frac{1}{48} & -\frac{29}{5040} & \frac{1}{1440} \\ \frac{1}{40320} & -\frac{1}{5040} & \frac{1}{1440} & -\frac{1}{720} & \frac{1}{576} & -\frac{1}{720} & \frac{1}{1440} & -\frac{1}{5040} & \frac{1}{40320} \end{pmatrix}.$$

9. $m = 10,\ n_g = 9$:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{7129}{2520} & -9 & 18 & -28 & \frac{63}{2} & -\frac{126}{5} & 14 & -\frac{36}{7} & \frac{9}{8} & -\frac{1}{9} \\ \frac{6515}{2016} & -\frac{4609}{280} & \frac{5869}{140} & -\frac{6289}{90} & \frac{6499}{80} & -\frac{265}{4} & \frac{6709}{180} & -\frac{967}{70} & \frac{3407}{1120} & -\frac{761}{2520} \\ \frac{4523}{2268} & -\frac{14139}{1120} & \frac{20837}{560} & -\frac{72569}{1080} & \frac{6519}{80} & -\frac{3273}{48} & \frac{84307}{2160} & -\frac{4101}{280} & \frac{1823}{560} & -\frac{29531}{90720} \\ \frac{95}{128} & -\frac{7667}{1440} & \frac{24901}{1440} & -\frac{4013}{120} & \frac{122249}{2880} & -\frac{5273}{144} & \frac{10279}{480} & -\frac{2939}{360} & \frac{10579}{5760} & -\frac{89}{480} \\ \frac{3013}{17280} & -\frac{7807}{5760} & \frac{6787}{1440} & -\frac{13873}{1440} & \frac{36769}{2880} & -\frac{32773}{2880} & \frac{9823}{1440} & -\frac{3817}{1440} & \frac{3487}{5760} & -\frac{1069}{17280} \\ \frac{5}{192} & -\frac{77}{360} & \frac{563}{720} & -\frac{401}{240} & \frac{3313}{1440} & -\frac{305}{144} & \frac{313}{240} & -\frac{373}{720} & \frac{347}{2880} & -\frac{1}{80} \\ \frac{29}{12096} & -\frac{413}{20160} & \frac{7}{90} & -\frac{31}{180} & \frac{353}{1440} & -\frac{67}{288} & \frac{53}{360} & -\frac{151}{2520} & \frac{287}{20160} & -\frac{91}{60480} \\ \frac{1}{8064} & -\frac{11}{10080} & \frac{43}{10080} & -\frac{7}{720} & \frac{41}{2880} & -\frac{1}{72} & \frac{91}{10080} & -\frac{19}{5040} & \frac{37}{40320} & -\frac{1}{10080} \\ \frac{1}{362880} & -\frac{1}{40320} & \frac{1}{10080} & -\frac{1}{4320} & \frac{1}{2880} & -\frac{1}{2880} & \frac{1}{4320} & -\frac{1}{10080} & \frac{1}{40320} & -\frac{1}{362880} \end{pmatrix}.$$

Appendix B

Implementation of RBDF61 in MATLAB to integrate (non-)linear ordinary differential equations of first order

Appendix B contains a complete listing of the MATLAB program "INT" that represents the implementation of the RBDF61 method .
By slightly modifying this program, other RBDF techniques introduced in this thesis can be implemented as well.

B.1 Main program INT

```
% Program to integrate ordinary differential equations of
% first order
%
flops(0)
define
% order of the integration algorithm
order = 6
steps = order + 1 % number of steps that we go backwards within the state
% history vector
all_points = 15
```

```
algor = 3
load searchmethod7
pick = picks(algor,:)
%
Ms = [ 0 1 2 3 4 5 6; 1 0 0 0 0 0 0; 0 1 0 0 0 0 0; 1 -1 1 -1 1 -1 1; 0 1 -2 3 -4 5 -6; 1 -2 4
-8 16 -32 64; 0 1 -4 12 -32 80 -192; 1 -3 9 -27 81 -243 729; 0 1 -6 27 -108 405 -1458; 1 -4 16
-64 256 -1024 4096; 0 1 -8 48 -256 1280 -6144; 1 -5 25 -125 625 -3125 15625; 0 1 -10 75 -500
3125 -18750; 1 -6 36 -216 1296 -7776 46656; 0 1 -12 108 -864 6480 -46656; 0 1 -12 108 -864
6480 -46656; 1 -7 49 -343 2401 -16807 117649; 0 1 -14 147 -1372 12005 -100842 ]
M = Ms(pick,:)
if rank(M) == order+1
% Penrose-Moore-Pseudoinverse
Minv = inv(M'*M)*M'
s = Minv(1,:)
for k=2:order+1
s = s + Minv(k,:)
end
ss = zeros(1,all_points)
ss(pick) = s
end
startup
[xvec,tvec,hvec] = mult(xvec,tvec,hvec,t,tsim,tcom,h,A,B,u,order,...
s,ss,all_points,xhist,tolrel,deltat,...
delta,pick,found,steps)
fop = flops % number of floating point operations
acc = exp(-tvec)
epsvec = xvec(:,1) – xacc
save 703mo1 xvec tvec hvec fop xacc epsvec
```

B.2 Procedure DEFINE

```
tolrelruku = 1E-6 % tolerance for the relative error produced by
% the Runge Kutta startup algorithm
tolrel = 1E-3 % tolerance of the relative error of the multistep algorithm
% stability domain of Ruku 5th order
stab_limit = 2.6
deltat = 0.05 % Length of communication interval
tsim = 5.0 % Simulation time
t = 0
tcom = 0
% Definition of the system
A =[ 0 1; -1000 -1001]
B = [0;0]
C = 0
D = 0
max_eig = max(abs(eig(A)))
```

```
% Input
u = 0
% Initial state vector x(0)
x = [1; -1]
n = size(x,1)
delta = ones(1,n)*1E-10
ECHO = 1
xvec = x'
tvec = tcom
hvec = 0
xcom = x' % state vector that is depicted graphically
xhist = x % whole state history matrix
```

B.3 Procedure STARTUP

```
bin_search_h0
if found == 1
x = x2
t = h
tcom = 0
if t >= tcom + deltat
% communication
comm
```

```
end
xhist = [x, xhist]
% order–1 times Runge Kutta 6th order with fixed step-size
for i=1:order–1
t = t + h
ruku6
x = x2
xhist = [x, xhist]
if t >= tcom + deltat
comm
end
end
end
```

B.4 Procedure BIN_SEARCH_H0

```
% This program calculates the initial step-size h for the
% multistep algorithm by means of a binary search
%
deltas = delta
ts = t
xs = x
hmin = 0
hmax = stab_limit/max_eig
```

```
maxcount = 30 % maximal number of steps of the binary search

count = 0

found = 0 % =1: Initial step-size h is found

% =0: Initial step-size h can't be found

xlast = x

terminate= -1

% Is hmax the step-size we are searching for?

h = hmax

ruku5

ruku6

eps1 = abs((x1 – x2)')

v = [x1';x2';delta]

epsrel= max(eps1 ./ max(abs(v)))

if epsrel < tolrelruku*1E-6 % too accurate —> change the initial values!

% First step with ruku6 to reevaluate the initial conditions

t = t + 0.5*(hmax+hmin)

ruku6

x = x2

delta = delta*1E-10

else

if epsrel <= tolrelruku

terminate = 1

found = 1
```

```
end
end
while terminate < 0
h = 0.5*(hmin + hmax)
ruku5
ruku6
eps1 = abs((x1 – x2)')
v = [x1';x2';delta]
epsrel= max(eps1 ./ max(abs(v)))
if epsrel > tolrelruku
hmax = h
else
hmin = h
if epsrel >= 0.9*tolrelruku
terminate = 1
found = 1
else
if count > maxcount
terminate = 1
else
count = count + 1
end
end
end
```

```
end

end

if delta < deltas

t = ts

x = xs

ruku6

end
```

B.5 Procedure RUKU5

```
% This program executes one step of a Runge Kutta 5th order

%

K1 = h*func(x,t,u)

K2 = h*func(x+0.125*K1,t,u)

K3 = h*func(x+0.25*K2,t,u)

K4 = h*func(x+0.5*K1–K2+K3,t,u)

K5 = h*func(x+0.1875*K1+0.5625*K4,t,u)

K6 = h*func(x+(–5*K1+4*K2+12*K3–12*K4+8*K5)/7,t,u)

x1 = x + (7*K1 + 32*K3 + 12*K4 + 32*K5 + 7*K6)/90
```

B.6 Procedure RUKU6

```
% This program executes one step of a Runge Kutta 6th order
%
K1 = h*func(x,t,u)
K2 = h*func(x+K1/9,t,u)
K3 = h*func(x+(K1+3*K2)/24,t,u)
K4 = h*func(x+(K1–3*K2+4*K3)/6,t,u)
K5 = h*func(x–0.625*K1+3.375*K2–3.0*K3+6.0*K4,t,u)
K6 = h*func(x+(221*K1–981*K2+867*K3–102*K4+K5)/9,t,u)
K7 = h*func(x+(-783*K1+678*K2–472*K3–66*K4+80*K5+3*K6)/48,t,u)
K8 = h*func(x+(761*K1–2079*K2+1002*K3+834*K4–454*K5–9*K6+72*K7)/82,t,u)
x2 = x + (41*K1 + 216*K2 + 27*K4 + 272*K5 + 27*K6 + 216*K7 + 41*K8)/840
```

B.7 Procedure COMM

```
% This procedure calculates the communication state vector that is
% used further to depict the state vector equidistantly
%
ncom = (t – tcom)/deltat – rem(t – tcom,deltat)
deltat1 = tcom + deltat – t + h
% Note that deltat1 > epscom is fulfilled!
deltax = (x–xlast)*deltat1/h
```

```
xcom = xlast + deltax

xvec = [xvec; xcom]

tcom = tcom + deltat

tvec = [tvec; tcom]

hvec = [hvec; h]

if ncom > 1

ncom = ncom – 1

deltax = (x – xcom)*deltat/h

for i = 1:ncom

xcom = xcom + deltax

tcom = tcom + deltat

xvec = [xvec; xcom]

tvec = [tvec; tcom]

hvec = [hvec; h]

end

end
```

B.8 Function FUNC

```
% Implementation of system 3 (Rayleigh equation)

%

function [fu] = func(x,t,u)

%
```

```
w = 1 – x(1)*x(1) – x(2)*x(2)

fu1 = 0.01*x(2) – x(1)*w

fu2 = -0.01*x(1) – x(2)*w

fu = [fu1;fu2]
```

B.9 Function MULT

```
% Multistep algorithm

%

% xhist is now a (n x m) matrix

%

function [xvec,tvec,hvec] = mult(xvec,tvec,hvec,t,tsim,tcom,h,u,n,order,...

s,ss,all_points,xhist,tolrel,deltat,...

delta,pick,found,steps)

%

I = eye(n)

toldelta = 0.1 % maximal value of the relative error that keeps the Jacobian unchanged

%

% Factorial of "order":

ordfac = 1

for i = 1:order

ordfac = ordfac*i

end
```

```
%
% Calculation of the error constant C_err
%
B1 = 0
p1 = size(pick)p1 = p1(2)
A1 = [zeros(1,p1)]
B1 = [zeros(1,p1)]
for k=p1:-1:1
kk = pick(k)
if (0.5*kk–0.5) == fix(0.5*kk)
B1(steps–0.5*kk+1.5) = s(k)
else
A1(steps–0.5*kk+1)= -s(k)
end
end
A1(steps+1) = 1
C_err = 0
for k = 1:p1,
k1 = k–1
Ak = A1(1,k)
Bk = B1(1,k)
C_err = C_err + (k1ˆ 7)*Ak/5040 – (k1ˆ 6)*Bk/720
end
```

```
%
factor = 1/1.2
p = 1/(order + 1)
if found == 1
% The first step of the multistep algorithm uses the old step-size
%
% Transformation matrix T to calculate the Nordsieck vector
%
T = [ 1 0 0 0 0 0 0; 2.45 -6 7.5 -20/3 3.75 -1.2 1/6; 20.3/9 -8.7 117/8 -127/9 33/4 -
2.7 137/360; 49/48 -29/6 461/48 -31/3 307/48 -13/6 5/16; 35/144 -31/24 137/48 -121/36
107/48 -19/24 17/144; 7/240 -1/6 19/48 -0.5 17/48 -2/15 1/48; 1/720 -1/120 1/48 -1/36
1/48 -1/120 1/720]
% Note: This matrix T is only valid for the order = 6!
if det(T’) == 0
else
Tinv = inv(T)
%
stepcount = 0
alpha = 1
N_old = T*xhist’
n_old = N_old(order+1,:)
h_old = h
chg_jacob = 1 % Jacobian is reevaluated
```

```
epsmultold = delta
x = xhist(:,1) % initial value of x_k+1 is x_k!
first_time = 1
while t < tsim
t = t + h
stepcount = stepcount + 1
finish_step = -1
errcount = 0
chgstep = 0
while finish_step < 0
one_mul = -1
while one_mul < 0
f = func(x,t,u)
if chg_jacob > 0
% reevaluation of the Jacobian
jacob = jac(x,t,u,f,n)
chg_jacob = 0
hess = ss(1)*h*jacob – I % Hesse matrix
inv_hess = inv(hess)
else
if chgstep > 0
chgstep = 0
hess = ss(1)*h*jacob – I
```

```
inv_hess = inv(hess)

end

end

% Modified Newton-Raphson iteration

[x,conver] = newton(all_points,x,t,u,inv_hess,h,conver,delta, ss,...

xhist,0.1*tolrel)

if conver > 0

% convergence of Newton-Raphson

xhist = [x, xhist]

xhist = xhist(:,1:order+1)

%

% Estimation of the relative error

N = T*xhist' % We have to take the transposed history matrix!

epsmult = C_err*ordfac*(N(order+1,:)–n_old*alphaˆ order)

epsmult = abs(epsmult)

v = [abs(x'); delta]

xmax = max(v)

epsmultrel = max(epsmult ./ xmax)

deltaeps = (epsmultrel – epsmultold)/epsmultold

epsmultold = epsmultrel

if deltaeps > toldelta

if first_time == 1

first_time = 0
```

```
one_mul = 1
else
chg_jacob = 1
end
else
one_mul = 1
end
else
x = xhist(:,1)
one_mul = 1
finish_step = 1
t = tsim
end % of if conver > 0
end % of while one_mul
if conver > 0
% Newton iteration worked fine
if epsmultrel < tolrel
finish_step = 1
else
% maybe change of the step-size
errcount = errcount + 1
if errcount >= 3
alpha = 0.5
```

```
stepcount = 0
chgstep = 1
h_old = h
else
stepcount = order + 1
end
end
if stepcount >= order + 1
stepcount = 0
alpha = factor*(tolrel/epsmultrel)^p
if alpha >= 1
if alpha > 1.1
chgstep = 1
alpha = min(2,alpha)
else
% step-size is not changed
chgstep = 0
end
else
chgstep = 1
if alpha > 0.9
alpha = 0.9
else
```

```
alpha = max(alpha,0.5)

end

end % of if alpha >= 1

end % of if stepcount >= order+1

if chgstep > 0

% change of the step-size

chgstep = 0

h = alpha*h_old

%Adjustment of the state history matrix

H = [1 0 0 0 0 0 0; 0 alpha 0 0 0 0 0; 0 0 alpha^ 2 0 0 0 0; 0 0 0 alpha^ 3 0 0 0; 0 0 0 0
alpha^ 4 0 0; 0 0 0 0 0 alpha^ 5 0; 0 0 0 0 0 0 alpha^ 6]

if epsmultrel < tolrel

N = H*N % Nordsieck matrix

else

N = H*N_old

end

xhist = Tinv*N

xhist = xhist' % We have to transpose the matrix again

end % of if chgstep

end % of if conver > 0

end % of while finish_step ..

N_old = N

n_old = N_old(order+1,:)
```

```
h_old = h
if t >= tcom + deltat
% communication point
[xvec,tvec,tcom,hvec] = comm_mult(tcom,deltat,t,N,order,h,xvec,...
tvec,hvec,tsim)
end
end % of while t < tsim
end end
```

B.10 Function NEWTON

```
% Modified Newton iteration to solve the implicit equation of the
% integration algorithm
%
function [x,conver] = newton(all_points,x,t,u,inv_hess,h,conver,delta,ss,...
xhist,tolnewton)
%
finish = -1
count = 0
maxcount = 30
conver = 0
while finish < 0
x_old = x
```

```
F = imp_func(all_points,ss,xhist,x,t,u,h)
x = x - inv_hess*F
epsx = abs((x - x_old)')
v = [abs(x');delta]
xmax = max(v)
epsrel = max(epsx ./ xmax)
if epsrel > tolnewton
if count > maxcount
conver = 0
finish = 1
else
count = count + 1
end
else
finish = 1
conver = 1
end
end
```

B.11 Function IMP_FUNC

```
%This function calculates the implicit function of the integration algorithm
function [F] = imp_func(all_points,ss,xhist,x,t,u,h)
```

```
%

F = 0

for i = 2:all_points % all_points = size(ss)

if ss(i) == 0

else

if 0.5*i–0.5 == fix(0.5*i)

F = F + ss(i)*h*func(xhist(:,0.5*i–0.5),t,u)

else

F = F + ss(i)*xhist(:,0.5*i)

end

end

end

F = F + ss(1)*h*func(x,t,u) – x
```

B.12 Function JAC

```
% This program calculates the Jacobian matrix for the Newton iteration

%

function [J] = jac(x,t,u,f,n)

%

for i = 1:n % columns

xnew = x

if x(i) == 0
```

```
deltax = 1E-10

else

deltax = x(i)*1E-7

end

xnew(i) = xnew(i) + deltax

fnew = func(xnew,t,u)

J(:,i) = (fnew – f)/deltax

end
```

B.13 Function COMM_MULT

```
% This function calculates the communication state vector if a

% multistep integration algorithm is used

%

function [xvec,tvec,tcom,hvec] = comm_mult(tcom,deltat,t,N,order,...

h,xvec,tvec,hvec,tsim)

finish = -1

while finish < 0

if tcom + deltat <= t

tcom = tcom + deltat % update tcom

if tcom > tsim

finish = 1

else
```

```
d = tcom – t % negative step-size!

N1 = N

for i = 2:order+1

N1(i,:) = ((d/h)ˆ (i – 1)) * N1(i,:)

end

xcom = N1(1,:)

for i = 2:order+1

xcom = xcom + N1(i,:)

end

xvec = [xvec; xcom]

tvec = [tvec; tcom]

hvec = [hvec; h]

end

else

finish = 1

end

end
```

REFERENCES

[1] Albert, A. (1972) *Regression and the Moore-Penrose Pseudoinverse.*, Academic Press,Inc., New York

[2] Axelsson, O. (1969) *A class of A-stable methods*, BIT

[3] Bronstein, I.N. and Semendjajew, K.A. (1987) *Taschenbuch der Mathematik*, Verlag Harri Deutsch, Frankfurt/Main

[4] Cellier, F.E. (1995) *Continuous System Simulation*, Springer-Verlag — in preparation

[5] Cryer, C.W. (1973) *A new class of highly stable methods: A_0-stable methods*, BIT

[6] Dahlquist, G. (1963) *A Special Stability Problem for Linear Multistep Methods*, MIT

[7] Dahlquist, G. and Björk, A. (1974) *Numerical Methods*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey

[8] Deif, A.S. (1982) *Advanced Matrix Theory for scientists and engineers*, John Wiley & Sons, New York

[9] Ehle, B.L. (1969) *On Padé approximations to the exponential function and A-stable methods for the numerical solution of initial value problems*, Research Rep. CSRR 2010, Dept. AACS, University of Waterloo

[10] Forsythe, G.E. (1977) *Computer Methods for Mathematical Computations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey

[11] Gear, C.W. (1971) *Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey

[12] Hu, L. (1991) *DBDF: An Implicit Numerical Differentiation Algorithm for Integrated Circuit Simulation*, Master Thesis at ECE, University of Arizona

[13] Isaacson, E. and Keller, H.B. (1966) *Analysis of Numerical Methods*, John Wiley & Sons, New York

[14] Iserles, A. and Nørsett, S.P. (1991) *Order Stars*, Chapman & Hall, New York

[15] Lambert, J.D. (1991) *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*, John Wiley & Sons, New York

[16] Lapidus L. and , Seinfeld, J.H. (1971) *Numerical Solution of Ordinary Differential Equations*, Academic Press, New York

[17] Mathworks, Inc. (1987) *Pro-MATLAB with System Identification Toolbox — User Manual*, South Natick, Mass.

[18] Nordsieck, A. (1962) *On numerical integration of ordinary differential equations*,Math. Comp.

[19] Shampine, L.F. and Gordon, M.K. (1975) *Computer Solution of Ordinary Differential Equations*, W.H. Freeman & Co., San Francisco

[20] Skelboe, S. (1984) *Multirate integration methods*, Research Report at the Institute of Datalogy, University of Copenhagen

[21] Vidyasagar M. (1978) *Nonlinear Systems Analysis* Prentice-Hall, Inc., Englewood Cliffs, N.J.

[22] Widlund, O.B. (1967) *A note on unconditionally stable linear multistep methods*, BIT