A Solution Routine for Singular Boundary Value Problems

Winfried Auzinger
Günter Kneisl
Othmar Koch
Ewa B. Weinmüller

Technical Report

ANUM Preprint No. 1/02

Technische
Universität
Wien

Vienna
University of
Technology

Institute for Applied Mathematics
and Numerical Analysis

Abstract

In this report, we discuss the implementation and numerical aspects of the MATLAB solver `sbvp` designed for the solution of two-point boundary value problems, which may include a singularity of the first kind,

$$\begin{aligned}&\mathbf{z}'(t) = \mathbf{f}(t, \mathbf{z}(t)) := \frac{1}{(t-a)} M(t) \cdot \mathbf{z}(t) + \mathbf{g}(t, \mathbf{z}(t)), \quad t \in (a, b),\\ &\mathbf{R}(\mathbf{z}(a), \mathbf{z}(b)) = \mathbf{0}.\end{aligned}$$

The code is based on collocation at either equidistant or Gaussian collocation points. For singular problems, the choice of equidistant nodes does not imply a loss of efficiency since no convergence order higher than the stage order can be expected in general.

An error estimate for the global error of the approximate solution is also provided. This estimate is obtained by a modification of the Defect Correction idea originally proposed by Zadunaisky in 1976. This estimate has been proven to be asymptotically correct and provides the basis for an adaptive mesh selection strategy. Here the grid is modified with the aim to equidistribute the global error. Most importantly, we observe that the grid is refined in a way reflecting merely the smoothness of the solution, unaffected by the singularity of f.

We discuss details of the efficient implementation in MATLAB 6 and illustrate the performance of the code by comparing it with the standard MATLAB solver `bvp4c` and the Fortran 90 code COLNEW.

The `sbvp` package is available from `http://www.math.tuwien.ac.at/~ewa`.

Contents

1 Introduction **6**

1.1 Problem statement 6

1.1.1 Motivation 6

1.1.2 Solution approach 7

1.2 Contents of the paper 7

2 Solution of the problem **8**

2.1 The method of collocation 8

2.1.1 General setting 8

2.1.2 Basis simplifications 9

2.1.3 Lagrange and Runge-Kutta basis 10

2.1.4 Linear indexing and Jacobian 10

2.1.5 Quasilinearization 11

2.2 The error estimate 12

2.2.1 The neighboring problem 13

2.2.2 The backward Euler method 14

2.3 Solution of nonlinear equations 14

2.3.1 Damped Newton iteration 14

2.3.2 Termination conditions 15

2.4 Bases and round-off errors 16

2.4.1 Evaluation of **F** 16

2.4.2 Simplifications for Lagrange and Runge-Kutta basis 17

2.4.3 Evaluation of DF 17

2.4.4 The Newton increment 18

2.4.5 Conclusions 19

3 Numerical tests **20**

3.1 The Newton iteration 20

3.2 The collocation solver 21

3.2.1 The conditioning of DF 21

3.2.2 Orders of convergence 23

3.2.3 Bases and round-off errors 24
3.3 The error estimate 25
3.3.1 The conditioning of DG 25
3.3.2 Order of convergence 25

4 The solution routines **27**
4.1 Description 27
4.1.1 Modules 27
4.1.2 Solver syntax 27
4.1.3 The `bvpfile` 28
4.1.4 Solution options 30
4.1.5 Zerofinder options 33
4.1.6 Output functions 33
4.1.7 Log-structs and test routines 34
4.2 Developing the solver 35
4.2.1 Efficient referencing 35
4.2.2 Vectorization 36
4.2.3 Indexing 37
4.2.4 Efficient storage of `sbvpcol` variables 37

5 Comparisons **39**

List of symbols

$[a, b]$	interval where the BVP is defined	1.1
t	independent variable of the ODE	1.1
$\mathbf{y}$	dependent variable of the ODE	1.1
d	dimension of the problem	1.1
$\mathbf{f}(t, \mathbf{y})$	right-hand side of the ODE	1.1
$\mathbf{R}(\mathbf{y}_a, \mathbf{y}_b)$	function defining the boundary conditions	1.1
$\mathbf{y}^*(t)$	(exact) solution of the BVP	1.1
$N+1$	number of mesh points	2.1.1
τ_k	k-th mesh point	2.1.1
h_k	length of k-th interval, $\tau_{k+1} - \tau_k$	2.1.2
$h_{\min}$	$\min_{1 \le k \le N} h_k$	2.4.1
p	number of collocation points per interval $[\tau_k, \tau_{k+1}]$, maximal degree of basis polynomials	2.1.1
ρ^i	collocation points shifted to the interval $[0, 1]$	2.1.2
t_k^i	collocation points in the interval $[\tau_k, \tau_{k+1}]$	2.1.1
$\mathbf{y}(t)$	piecewise polynomial approximation of the solution $\mathbf{y}^*(t)$	2.1.1
$\mathbf{y}_k(t)$	collocation polynomial on $[\tau_k, \tau_{k+1}]$	2.1.1
$\varphi_j(t)$	j-th basis polynomial on the interval $[0, 1]$	2.1.1
$\varphi_{kj}(t)$	j-th basis polynomial shifted to the interval $[\tau_k, \tau_{k+1}]$	2.1.1
$\mathbf{a}_{kj}$	coefficient of $\mathbf{y}_k(t)$ w.r.t. the basis polynomial $\varphi_{kj}(t)$	2.1.1
$\mathbf{y}_k^i$	$\mathbf{y}(t_k^i)$	2.1.1
$\mathbf{y}_k'^i$	$\mathbf{y}'(t_k^i)$	2.1.1
$\mathbf{y}_k^-$	$\mathbf{y}(\tau_k)$	2.1.1
$\mathbf{y}_k^+$	$\mathbf{y}(\tau_{k+1})$	2.1.1
φ_j^i	$\varphi_{kj}(t_k^i) = \varphi_j(\rho^i)$	2.1.1
$\varphi_j'^i$	$\varphi_j'(\rho^i)$	2.1.1
$\varphi_{kj}'^i$	$\varphi_{kj}'(t_k^i)$	2.1.1
φ_j^-	$\varphi_{kj}(\tau_k)$	2.1.1
φ_j^+	$\varphi_{kj}(\tau_{k+1})$	2.1.1
$\mathbf{f}_k^i$	$\mathbf{f}(t_k^i, \mathbf{y}_k^i)$	2.1.1
$\mathbf{x}$	vector of the coefficients $(\mathbf{a}_{kj})$	2.1.4
$\mathbf{F}(\mathbf{x})$	residual of the coefficients w.r.t. the collocation scheme	2.1.4
$\mathrm{DF}(\mathbf{x})$	Jacobian of the discretization residual $\mathbf{F}$	2.1.4
Φ', Φ, Ψ	matrices in Jacobian decomposition $\mathrm{DF}(\mathbf{x}) = \Phi' - \Phi. * \Psi(\mathbf{x})$	2.1.4
$\mathbf{x}^*$	solution of the nonlinear system $\mathbf{F}(\mathbf{x}) = \mathbf{0}$	2.1.4
$\mathbf{x}^{(n)}$	n-th Newton approximation of $\mathbf{x}^*$	2.3.2
$\Delta\mathbf{x}^{(n)}$	n-th Newton increment	2.3.2
$\lambda^{(n)}$	n-th stepsize in the damped Newton iteration	3.1

List of symbols

Symbol	Explanation	Section
s_k	k-th point in collocation grid, $k = 1, \ldots, N(p+1)+1$	2.2.1
δ_k	$s_{k+1} - s_k$	2.2.2
$\mathbf{u}_k$	solution approximation on k-th grid point	2.2.2
$\mathbf{e}(t)$	truncation error $\mathbf{y}(t) - \mathbf{y}^*(t)$	2.2
$\mathbf{e}_k$	$\mathbf{e}(s_k)$	3.3.2
$\boldsymbol{\varepsilon}_k$	error estimate in s_k	3.3.2
$\mathbf{v}$	the solution components $\mathbf{u}_k$ arranged in linear succession	2.2.2
$\mathbf{G}(\mathbf{v})$	residual of the backward Euler discretization	2.2.2
$\mathrm{DG}(\mathbf{v})$	Jacobian of the discretization residual $\mathbf{F}$	2.2.2
I, J	matrices in Jacobian decomposition $\mathrm{DG}(\mathbf{v}) = I - J(\mathbf{v})$	2.2.2
$\chi_{[a,b]}$	characteristic function of the interval $[a, b]$	2.1.1
δ_{ij}	Kronecker delta	2.1.3
$\mathbf{1}_d$	d-dimensional identity matrix	2.1.4

Chapter 1

Introduction

1.1 Problem statement

We consider the boundary value problem for a system of ordinary differential equations which may contain a *singularity* of the first kind,

$$\mathbf{z}'(t) = \mathbf{f}(t, \mathbf{z}(t)) := \frac{1}{(t-a)} M(t) \cdot \mathbf{z}(t) + \mathbf{g}(t, \mathbf{z}(t)), \quad t \in (a, b), \tag{1.1a}$$

$$\mathbf{R}(\mathbf{z}(a), \mathbf{z}(b)) = \mathbf{0}, \tag{1.1b}$$

where $\mathbf{z}$, $\mathbf{g}$ and $\mathbf{R}$ are smooth vector-valued functions and M is a matrix which depends continuously on t. Note that a problem of this form is well-posed only under certain restrictions on the function $\mathbf{R}$. For a detailed discussion of the analytical properties of (1.1) see [12]. The goal is to find an approximation of the (locally) unique solution $\mathbf{y}^*(t)$ that satisfies a prescribed tolerance with as little computational effort as possible. This requires the use of an adaptive mesh selection strategy based on error estimation.

1.1.1 Motivation

Mathematical models of numerous applications from physics and chemistry take the form of systems of time-dependent partial differential equations subject to initial-boundary conditions, e.g. Ginzburg-Landau equations which arise in some physical contexts such as ferromagnetic systems, superconductivity models, models for unidirectional ring lasers and laser hydrodynamics.
For the investigation of stationary solutions many of these models can be reduced to singular systems of ordinary differential equations, especially when – due to symmetries in the geometry of the problem and the problem data – polar, cylindrical or spherical coordinates can be used.

The numerical computation of homoclinic and heteroclinic orbits is of interest in describing structural changes in dynamical systems. Such problems take the form of a boundary value problem posed on an infinite interval. If, instead of time, the arclength parametrization of the orbit is used, then the problem is posed on a finite interval, and techniques developed for singular problems can be applied.

Although there are various solution approaches suited for particular applications, professional software for at least wide subclasses of singular problems has not been available so far. Our aim was to provide such a code, implemented in MATLAB. This package called `sbvp` is now available from `http://www.math.tuwien.ac.at/~ewa`.

Standard codes like COLSYS, see [1], have often been used to solve singular problems numerically. In COLSYS, the underlying method does not involve function evaluations at the singular point, and so this code can be applied to solve the system (1.1). On the other hand, the new MATLAB 6 code `bvp4c` is based on a Lobatto scheme including function evaluations at the endpoints of the interval and therefore can be used for singular systems only after a certain prehandling of the problem data. A comparison of the performance of our code `sbvp`, COLNEW (a Fortran 90 version of COLSYS), and `bvp4c` when applied to singular test problems is given in §5.

1.1.2 Solution approach

We decided to use collocation for the numerical solution of the underlying boundary value problems. A collocation solution is a piecewise polynomial function which satisfies the given ODE at a finite number of nodes (collocation points). This approach shows advantageous convergence properties compared to other direct higher order methods (see [13], [18]), which may be affected by order reductions and become inefficient in the presence of a singularity, see for example [14].

Furthermore, we decided to control the global error instead of monitoring the local error because of the unsmoothness of the latter near the singular point and order reductions it suffers from, cf. [11], [6]. The implemented error estimate was proposed in its classical version by Stetter [17] and is based on an idea due to Zadunaisky [20], originally formulated for solutions obtained by Runge-Kutta schemes. (The same idea is also the basis for the acceleration technique known as *Iterated Defect Correction*, cf. [9], [15].) The classical error estimate works satisfactorily in the mesh points but fails to be asymptotically correct for finer grids including collocation points. Modifications proposed in [5] remedy this problem and provide an asymptotically correct error estimate for the full grid, enhancing the efficiency of the estimate and the grid flexibility.

The mesh selection strategy is based on equidistribution of the global error. A detailed description of the procedure is given in [4].

1.2 Contents of the paper

The paper is organized as follows: In §2 we describe the underlying numerical method, namely collocation, together with major details of the implementation. Moreover, the estimation procedure for the global error of the collocation scheme, and the solution of the nonlinear system resulting from the discretization process are discussed. Numerical tests illustrating the performance of the Newton solver are given in §3. §4 contains the description of the routines involved and a comprehensive specification of the control parameters. Finally, in §5 we compare the performance of `sbvp` with that of `bvp4c` and COLNEW by means of a large number of test examples.

Chapter 2

Solution of the problem

2.1 The method of collocation

2.1.1 General setting

We choose a suitable *mesh*

$$a = \tau_1 < \tau_2 < \ldots < \tau_{N+1} = b$$

and try to find *polynomial vectors*

$$\mathbf{y}_k(t) = \begin{pmatrix} y_k^1(t) \\ \vdots \\ y_k^d(t) \end{pmatrix}, \quad (1 \le k \le N),$$

of degree equal to or less than p such that they satisfy the differential equation

$$\mathbf{y}_k' = \mathbf{f}(t, \mathbf{y}_k)$$

on p distinct *collocation points* t_k^i in the interval[1] (τ_k, τ_{k+1}). Moreover, the boundary conditions

$$\mathbf{R}(\mathbf{y}_1(a), \mathbf{y}_N(b)) = \mathbf{0}$$

as well as the continuity conditions

$$\mathbf{y}_k(\tau_{k+1}) = \mathbf{y}_{k+1}(\tau_{k+1}), \quad (1 \le k \le N-1),$$

are required to hold.

The continuous function

$$\mathbf{y}(t) := \sum_{k=1}^{N} \mathbf{y}_k(t) \cdot \chi_{[\tau_k, \tau_{k+1}]}(t)$$

then serves as the approximation of the analytical solution $\mathbf{y}^*(t)$.

[1] The error estimate used in the mesh adaptation process (discussed in §2.2) is well-defined only if collocation points do not coincide with mesh points. If there is no need for an error estimate, the collocation points may be taken from $(\tau_k, \tau_{k+1}]$ for singular and from $[\tau_k, \tau_{k+1}]$ for regular boundary value problems.

In order to formulate the collocation problem in such a way that it can be directly translated into a solution algorithm, we have to introduce some additional notation.

First of all, we express the unknown polynomials $\mathbf{y}_k(t)$ in terms of $p+1$ basis polynomials $\varphi_{kj}(t)$,

$$\mathbf{y}_k(t) = \sum_{j=1}^{p+1} \mathbf{a}_{kj} \cdot \varphi_{kj}(t)\,.$$

Denoting[2]

$$\begin{aligned}
\varphi_{kj}^{i} &:= \varphi_{kj}(t_k^i), & \mathbf{y}_k^{i} &:= \mathbf{y}_k(t_k^i),\\
\varphi_{kj}^{-} &:= \varphi_{kj}(\tau_k), & \mathbf{y}_k^{-} &:= \mathbf{y}_k(\tau_k),\\
\varphi_{kj}^{+} &:= \varphi_{kj}(\tau_{k+1}), & \mathbf{y}_k^{+} &:= \mathbf{y}_k(\tau_{k+1}),\\
\varphi_{kj}^{\prime i} &:= \varphi_{kj}'(t_k^i), & \mathbf{y}_k^{\prime i} &:= \mathbf{y}_k'(t_k^i),
\end{aligned}$$

we can write the collocation problem in the following form:

$$\mathbf{R}(\mathbf{y}_1^{-}, \mathbf{y}_N^{+}) = \mathbf{R}\Big(\sum_{j=1}^{p+1} \mathbf{a}_{1j}\,\varphi_{1j}^{-}, \sum_{j=1}^{p+1} \mathbf{a}_{Nj}\,\varphi_{Nj}^{+}\Big) = \mathbf{0}, \tag{2.1a}$$

$$\mathbf{y}_k^{\prime i} - \mathbf{f}(t_k^i, \mathbf{y}_k^i) = \sum_{j=1}^{p+1} \mathbf{a}_{kj}\,\varphi_{kj}^{\prime i} - \mathbf{f}\Big(t_k^i, \sum_{j=1}^{p+1} \mathbf{a}_{kj}\,\varphi_{kj}^{i}\Big) = \mathbf{0},\ (1 \le k \le N), \tag{2.1b}$$

$$\mathbf{y}_k^{+} - \mathbf{y}_{k+1}^{-} = \sum_{j=1}^{p+1} \mathbf{a}_{kj}\,\varphi_{kj}^{+} - \sum_{j=1}^{p+1} \mathbf{a}_{k+1,j}\,\varphi_{k+1,j}^{-} = \mathbf{0},\ (1 \le k \le N\text{–}1). \tag{2.1c}$$

The system (2.1) provides $d + Npd + (N-1)d = N(p+1)d$ equations for the $N(p+1)d$ unknown coefficients $(\mathbf{a}_{kj})$.

2.1.2 Basis simplifications

We call the collocation points (t_k^i) of the intervals $[\tau_k, \tau_{k+1}]$ *equivalent* if there exist p real numbers ρ^i $(0 < \rho^1 < \ldots < \rho^p < 1)$ such that

$$t_k^i = \tau_k + \underbrace{(\tau_{k+1} - \tau_k)}_{=h_k} \cdot \rho^i$$

holds for all k. Furthermore, we speak of *equivalent* basis polynomials $\varphi_{kj}(t)$ if there exist $p+1$ polynomials $\varphi_j(t)$ such that

$$\varphi_{kj}(t) = \varphi_j\Big(\frac{t - \tau_k}{h_k}\Big).$$

If both collocation points and basis polynomials of the intervals $[\tau_k, \tau_{k+1}]$ are equivalent, the equalities

$$\varphi_{k_1 j}^{i} = \varphi_{k_2 j}^{i}\,, \quad \varphi_{k_1 j}^{-} = \varphi_{k_2 j}^{-}\,, \quad \varphi_{k_1 j}^{+} = \varphi_{k_2 j}^{+}\,, \quad \frac{\varphi_{k_1 j}^{\prime i}}{h_{k_1}} = \frac{\varphi_{k_2 j}^{\prime i}}{h_{k_2}}$$

hold for all k_1, k_2 and thus make the index k in $\varphi_{kj}^{i}, \varphi_{kj}^{-}, \varphi_{kj}^{+}$ redundant. In the sequel, we will restrict our attention to this case and omit the superfluous index.

[2]Throughout this paper, the index k refers to intervals, j $(1 \le j \le p+1)$ refers to the basis polynomials and i $(1 \le i \le p)$, which is always superscripted, refers to the collocation points.

2.1.3 Lagrange and Runge-Kutta basis

The $p+1$ Lagrange polynomials $\varphi_j(t)$ for the p collocation points ρ^i on the interval $[0,1]$ are defined uniquely by requiring

$$\varphi_j(0) = \delta_{j1}\,,\ \varphi_j(\rho^i) = \delta_{j(i+1)}\,,$$

or expressed in the above notation

$$\begin{pmatrix} \varphi_1^- \\ \vdots \\ \varphi_{p+1}^- \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \begin{pmatrix} \varphi_1^1 & \cdots & \varphi_1^p \\ \vdots & \ddots & \vdots \\ \varphi_{p+1}^1 & \cdots & \varphi_{p+1}^p \end{pmatrix} = \begin{pmatrix} 0 & \cdots & 0 \\ 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}.$$

The values φ_j^+ and the $\varphi_{kj}'^i$ can be calculated easily from this. The Runge-Kutta basis for the same interval is defined by

$$\varphi_j(0) = \delta_{j1}\,,\ \varphi_j'(\rho^i) = \delta_{j(i+1)}\,,$$

or

$$\begin{pmatrix} \varphi_1^- \\ \vdots \\ \varphi_{p+1}^- \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \begin{pmatrix} \varphi_1'^1 & \cdots & \varphi_1'^p \\ \vdots & \ddots & \vdots \\ \varphi_{p+1}'^1 & \cdots & \varphi_{p+1}'^p \end{pmatrix} = \begin{pmatrix} 0 & \cdots & 0 \\ 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}.$$

Again, the calculation of the φ_j^+ , φ_j^i and $\varphi_{kj}'^i$ is straightforward.

Apart from the Lagrange and the Runge-Kutta basis, the Legendre basis and the monomial basis have been implemented in the solution routine. For these bases, both φ_j^i and $\varphi_{kj}'^i$ depend on the choice of the collocation points.

2.1.4 Linear indexing and Jacobian

The collocation equations (2.1) represent a nonlinear system of equations for the unknown coefficients $(\mathbf{a}_{kj})$, which we attempt to solve by damped Newton iteration (cf. §2.3).

In order to preserve the sparse structure of the Jacobian, we arrange the coefficients $(\mathbf{a}_{kj})$ in a vector $\mathbf{x} \in \mathbb{R}^{N(p+1)d}$ in the following way:

$$\mathbf{x} := (a_{11}^1, \cdots, a_{11}^d, a_{12}^1, \cdots, a_{12}^d, \cdots, a_{1,p+1}^d, a_{21}^1, \cdots, a_{N,p+1}^d)^T. \tag{2.2}$$

Moreover, we reorder the collocation equations (2.1) to obtain

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} \mathbf{R}(\mathbf{y}_1^-, \mathbf{y}_N^+) \\ \mathbf{y}_1'^1 - \mathbf{f}(t_1^1, \mathbf{y}_1^1) \\ \vdots \\ \mathbf{y}_1'^p - \mathbf{f}(t_1^p, \mathbf{y}_1^p) \\ \mathbf{y}_1^+ - \mathbf{y}_2^- \\ \mathbf{y}_2'^1 - \mathbf{f}(t_2^1, \mathbf{y}_2^1) \\ \vdots \\ \mathbf{y}_N'^p - \mathbf{f}(t_N^p, \mathbf{y}_N^p) \end{pmatrix} = \mathbf{0}. \tag{2.3}$$

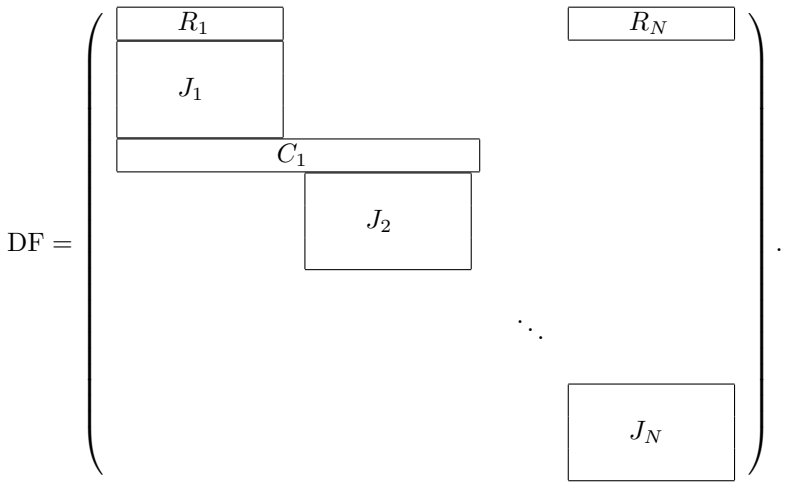

Figure 2.1: Block structure of the Jacobian DF

The solution of this nonlinear system of equations is denoted by $\mathbf{x}^*$. The block structure of the Jacobian $\mathrm{DF}(\mathbf{x})$ is shown in Figure 2.1, with R_1, R_N, J_k, C_k given by

$$
\begin{aligned}
R_1 &:= \left(\frac{\partial \mathbf{R}}{\partial \mathbf{z}_a} \cdot \varphi_1^- \ \cdots \ \frac{\partial \mathbf{R}}{\partial \mathbf{z}_a} \cdot \varphi_{p+1}^- \right), \\
R_N &:= \left(\frac{\partial \mathbf{R}}{\partial \mathbf{z}_b} \cdot \varphi_1^+ \ \cdots \ \frac{\partial \mathbf{R}}{\partial \mathbf{z}_b} \cdot \varphi_{p+1}^+ \right), \\
J_k &:= \begin{pmatrix} \varphi_{k1}'^1 \cdot \mathbf{1}_d - D\mathbf{f}(t_k^1, \mathbf{y}_k^1) \cdot \varphi_1^1 & \cdots & \varphi_{k,p+1}'^1 \cdot \mathbf{1}_d - D\mathbf{f}(t_k^1, \mathbf{y}_k^1) \cdot \varphi_{p+1}^1 \\ \vdots & & \vdots \\ \varphi_{k1}'^p \cdot \mathbf{1}_d - D\mathbf{f}(t_k^p, \mathbf{y}_k^p) \cdot \varphi_1^p & \cdots & \varphi_{k,p+1}'^p \cdot \mathbf{1}_d - D\mathbf{f}(t_k^p, \mathbf{y}_k^p) \cdot \varphi_{p+1}^p \end{pmatrix}, \\
C_k &:= \left(\varphi_1^+ \cdot \mathbf{1}_d \ \cdots \ \varphi_{p+1}^+ \cdot \mathbf{1}_d \ \ (-\varphi_1^- \cdot \mathbf{1}_d) \ \cdots \ (-\varphi_{p+1}^- \cdot \mathbf{1}_d) \right).
\end{aligned}
$$

Note that $\mathrm{DF}(\mathbf{x})$ can be decomposed into the sum of a constant matrix Φ' and the *elementwise* product of a constant matrix Φ and a matrix Ψ that depends on $\mathbf{x}$,

$$\mathrm{DF}(\mathbf{x}) = \Phi' - \Psi(\mathbf{x}) .* \Phi.$$

If the boundary value problem is linear, DF does not depend on $\mathbf{x}$ and the solution (in terms of polynomial coefficients in linear succession) is given by

$$\mathbf{x}^* = -\mathrm{DF}^{-1} \cdot \mathbf{F}(\mathbf{0}).$$

2.1.5 Quasilinearization

In this section, we show that the method of quasilinearization, which was proposed by Ascher et al. ([3]) for solving nonlinear collocation schemes, is equivalent to solving the nonlinear collocation equations by Newton's method as described above.

Given the boundary value problem (1.1) and an initial solution profile $\mathbf{y}(t)$ that is expressed in terms of the basis coefficients

$$\mathbf{x} = (a_{11}^1, \cdots, a_{11}^d, a_{12}^1, \cdots, a_{12}^d, \cdots, a_{1,p+1}^d, a_{21}^1, \cdots, a_{N,p+1}^d)^T,$$

the next approximation of the solution is obtained by solving the linear system

$$\mathrm{DF}(\mathbf{x}) \cdot \Delta\mathbf{x} = -\mathbf{F}(\mathbf{x})$$

and adding the *Newton increment* $\Delta\mathbf{x}$ to the initial approximation $\mathbf{x}$.

In the method of quasilinearization, the *differential equation* and the *boundary conditions* are linearized at the initial approximation $\mathbf{y}(t)$ to obtain the *linear* BVP

$$\mathbf{z}'(t) = \underbrace{\mathbf{f}(t,\mathbf{y}(t)) + \mathbf{f}'(t,\mathbf{y}(t))\cdot(\mathbf{z}(t)-\mathbf{y}(t))}_{=:\tilde{\mathbf{f}}(t,\mathbf{z}(t))}, \quad t \in (a,b),$$

$$\underbrace{\mathbf{R}(\mathbf{y}(a),\mathbf{y}(b)) + \frac{\partial\mathbf{R}}{\partial\mathbf{z}_a}(\mathbf{y}(a),\mathbf{y}(b))\cdot(\mathbf{z}(a)-\mathbf{y}(a)) + \frac{\partial\mathbf{R}}{\partial\mathbf{z}_b}(\mathbf{y}(a),\mathbf{y}(b))\cdot(\mathbf{z}(b)-\mathbf{y}(b))}_{=:\tilde{\mathbf{R}}(\mathbf{z}(a),\mathbf{z}(b))} = \mathbf{0}.$$

By definition, the solution of this linear BVP is the next solution approximation of the original, nonlinear BVP.

Clearly, the linearized boundary value problem can also be treated analogously to the nonlinear case (the solution is unique!) and the associated collocation equations solved by one step of Newton's method starting from the initial approximation $\mathbf{y}(t)$. Starting from the initial approximation $\mathbf{y}(t)$, the final solution is reached as the result of one Newton step.

The residual vector $\tilde{\mathbf{F}}$ and the Jacobian $\widetilde{\mathrm{DF}}$ of the linearized problem are obtained by substituting $\tilde{\mathbf{f}}$ for $\mathbf{f}$ and $\tilde{\mathbf{R}}$ for $\mathbf{R}$ in the definitions of $\mathbf{F}$ and DF. Adding the Newton increment $\widetilde{\Delta\mathbf{x}}$, i.e. the solution of the linear system

$$\widetilde{\mathrm{DF}} \cdot \widetilde{\Delta\mathbf{x}} = -\tilde{\mathbf{F}}$$

to the initial approximation $\mathbf{x}$ then yields the solution of the linearized problem.

However, since the equations

$$\begin{aligned}
\tilde{\mathbf{f}}(t,\mathbf{y}(t)) &= \mathbf{f}(t,\mathbf{y}(t)),\\
\tilde{\mathbf{f}}'(t,\mathbf{y}(t)) &= \mathbf{f}'(t,\mathbf{y}(t)),\\
\tilde{\mathbf{R}}(\mathbf{y}(a),\mathbf{y}(b)) &= \mathbf{R}(\mathbf{y}(a),\mathbf{y}(b)),\\
\frac{\partial\tilde{\mathbf{R}}}{\partial\mathbf{z}_a}(\mathbf{y}(a),\mathbf{y}(b)) &= \frac{\partial\mathbf{R}}{\partial\mathbf{z}_a}(\mathbf{y}(a),\mathbf{y}(b)),\\
\frac{\partial\tilde{\mathbf{R}}}{\partial\mathbf{z}_b}(\mathbf{y}(a),\mathbf{y}(b)) &= \frac{\partial\mathbf{R}}{\partial\mathbf{z}_b}(\mathbf{y}(a),\mathbf{y}(b))
\end{aligned}$$

hold for all t and therefore

$$\mathbf{F}(\mathbf{x}) = \tilde{\mathbf{F}}(\mathbf{x}), \qquad \mathrm{DF}(\mathbf{x}) = \widetilde{\mathrm{DF}}(\mathbf{x}),$$

the new approximation of the solution is the same for both quasilinearization and the solution of the nonlinear collocation equations using Newton's method. Consequently, both methods are equivalent.

2.2 The error estimate

The method for the estimation of the global error described in this section is based on an idea due to Zadunaisky (cf. [20], [8]). In this original version, the error estimate for

the high-order method is obtained by applying this method twice: to the underlying analytical problem first, and to a properly defined 'neighboring problem' next. In [17] and [10] this procedure was modified in order to reduce the amount of computational work: instead of applying the high-order method twice, it is applied only to solve the original problem. Additionally, a computationally cheap low-order method is used to solve the original and the neighboring problem.

2.2.1 The neighboring problem

Let $\mathbf{y}(t)$ be the piecewise polynomial solution approximation obtained by solving the original boundary value problem

$$\mathbf{z}'(t) = \mathbf{f}(t, \mathbf{z}(t)), \quad t \in (a, b), \tag{2.4a}$$

$$\mathbf{R}(\mathbf{z}(a), \mathbf{z}(b)) = \mathbf{0} \tag{2.4b}$$

by collocation. Define a neighboring problem as

$$\mathbf{z}'(t) = \mathbf{f}(t, \mathbf{z}(t)) + \mathbf{d}(t), \quad t \in (a, b), \tag{2.5a}$$

$$\mathbf{R}(\mathbf{z}(a), \mathbf{z}(b)) = \mathbf{0} \tag{2.5b}$$

with the defect

$$\mathbf{d}(t) := \mathbf{y}'(t) - \mathbf{f}(t, \mathbf{y}(t)).$$

By construction, $\mathbf{y}(t)$ is the exact solution of (2.5).

In the original version proposed by Zadunaisky, [20], (2.5) is solved by the same numerical method as for (2.4). Thus, some quantities which are necessary for the solution process have been computed earlier and can be reused[3].

However, Stetter ([17]) and Frank ([10]) suggest a computationally cheaper method: Both (2.4) and (2.5) are solved using a computationally cheap auxiliary method in the points $(\tau_1, \ldots, \tau_{N+1})$. Our choice for this auxiliary method is the backward Euler scheme. It was shown in [10] that the difference between the two numerical solutions computed in this way constitutes an asymptotically correct estimate of the global error for the high order solution of (2.4).

If we apply the same idea on the *collocation grid*

$$\mathbf{s} = (s_1, \ldots, s_{N(p+1)+1}) := (\tau_1, t_1^1, \ldots, t_1^p, \tau_2, t_2^1, \ldots, \tau_{N+1}),$$

however, the estimate is no longer asymptotically correct. Yet, if we replace $\mathbf{d}(t_k^i)$ by[4]

$$\bar{\mathbf{d}}_k^i := \frac{\mathbf{y}(t_k^i) - \mathbf{y}(t_k^{i-1})}{\delta_k^i} - \sum_{l=1}^{p+1} \alpha_{il}\, \mathbf{f}(t_k^i, \mathbf{y}(t_k^i)), \quad (1 \le i \le p+1) \tag{2.6}$$

we obtain an error estimate which is asymptotically correct on grids with an even number of equidistantly distributed collocation points. A proof of this fact is given in [5].

The error estimate modified in this way has been implemented in the solution routine.

[3] If $\mathbf{f}$ is linear, (2.4) and (2.5) lead to the same Jacobian DF, the LU-decomposition of which has already been calculated. If $\mathbf{f}$ is nonlinear, at least Φ and Φ' can be reused.

[4] For convenience, we denote

$$\begin{aligned} t_k^0 &:= \tau_k, \\ t_k^{p+1} &:= \tau_{k+1}, \\ \delta_k^i &:= t_k^i - t_k^{i-1} \ (1 \le i \le p+1). \end{aligned}$$

The α_{il} in (2.6) are certain quadrature coefficients.

2.2.2 The backward Euler method

The backward Euler discretization of the boundary value problem (2.4) on the collocation grid $\mathbf{s}$ yields the relations

$$\mathbf{R}(\mathbf{u}_1, \mathbf{u}_{M+1}) = 0,$$

$$\frac{\mathbf{u}_{k+1} - \mathbf{u}_k}{\delta_k} = \mathbf{f}(s_{k+1}, \mathbf{u}_{k+1}), \quad (1 \le k \le M),$$

where $\delta_k := s_{k+1} - s_k$ and $M := N(p+1)$.
If we arrange the $\mathbf{u}_k$ in linear succession,

$$\mathbf{u} = (\mathbf{u}_1^T, \ldots, \mathbf{u}_{M+1}^T)^T = (u_1^1, \ldots, u_1^d, u_2^1, \ldots, u_{M+1}^d)^T,$$

we obtain a nonlinear system of equations defined by

$$\mathbf{G}(\mathbf{u}) = \begin{pmatrix} \mathbf{R}(\mathbf{u}_1, \mathbf{u}_{M+1}) \\ \frac{\mathbf{u}_2 - \mathbf{u}_1}{\delta_1} - \mathbf{f}(s_2, \mathbf{u}_2) \\ \vdots \\ \frac{\mathbf{u}_{M+1} - \mathbf{u}_M}{\delta_M} - \mathbf{f}(s_{M+1}, \mathbf{u}_{M+1}) \end{pmatrix} = \mathbf{0}.$$

If we denote the derivative of $\mathbf{f}$ w.r.t. the second argument by $D\mathbf{f}_k := D\mathbf{f}(s_k, \mathbf{u}_k)$, then the Jacobian DG reads

$$\mathrm{DG}(\mathbf{u}) = \begin{pmatrix} \frac{\partial \mathbf{R}}{\partial z_a}(\mathbf{u}_1, \mathbf{u}_{M+1}) & & & & \frac{\partial \mathbf{R}}{\partial z_b}(\mathbf{u}_1, \mathbf{u}_{M+1}) \\ -\frac{1}{\delta_1}\mathbf{1}_d & \frac{1}{\delta_1}\mathbf{1}_d - D\mathbf{f}_2 & & & \\ & -\frac{1}{\delta_2}\mathbf{1}_d & \frac{1}{\delta_2}\mathbf{1}_d - D\mathbf{f}_3 & & \\ & & \ddots & & \\ & & & & \frac{1}{\delta_M}\mathbf{1}_d - D\mathbf{f}_{M+1} \end{pmatrix}.$$

$\mathrm{DG}(\mathbf{u})$ can be decomposed into the sum of a constant matrix I that contains only $\mathbf{1}_d$ terms and a matrix J that consists of the $D\mathbf{f}$- and boundary terms,

$$\mathrm{DG} = I - J(\mathbf{u}).$$

Similarly as for the collocation equations (2.1), $\mathbf{G}(\mathbf{u}) = \mathbf{0}$ is solved by damped Newton iteration.

2.3 Solution of nonlinear equations

2.3.1 Damped Newton iteration

Here, we briefly recall the method of damped Newton iteration, see for example [7]. Given an approximation $\mathbf{x}$ of the solution of the nonlinear system of equations

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}\,,$$

the *Newton increment* $\Delta\mathbf{x}$ is obtained by solving the linear system

$$\mathrm{DF}(\mathbf{x}) \cdot \Delta\mathbf{x} = -\mathbf{F}(\mathbf{x})\,,$$

and the next approximation $\mathbf{x}_\lambda$ is defined by

$$\mathbf{x}_\lambda := \mathbf{x} + \lambda \cdot \Delta\mathbf{x}, \quad 0 < \lambda \leq 1.$$

With the definition

$$\Delta\mathbf{x}_\lambda := -\mathrm{DF}^{-1}(\mathbf{x}) \cdot \mathbf{F}(\mathbf{x}_\lambda),$$

the new approximation $\mathbf{x}_\lambda$ is accepted if the *monotonicity condition*[5]

$$\|\Delta\mathbf{x}_\lambda\| \leq \left(1 - \frac{\lambda}{2}\right) \cdot \|\Delta\mathbf{x}\| \tag{2.7}$$

is satisfied. If not, λ is halved until (2.7) holds or some termination condition is met. Once a λ has been accepted, the next iteration step starts with $\lambda_{\text{new}} = \min(2\lambda, 1)$. In the first step of the damped Newton iteration, we start with $\lambda = 1$. This corresponds to the classical Newton iteration.

2.3.2 Termination conditions

The modulus of the n-th Newton residual $\mathbf{F}(\mathbf{x}^{(n)})$ cannot be considered a measure of the accuracy of the solution approximation $\mathbf{x}^{(n)}$, since the function $\mathbf{F}$ is scale dependent. In fact, if A is a regular matrix, $\mathbf{F}$ could be replaced by $\hat{\mathbf{F}} = A \cdot \mathbf{F}$ without altering the iteration,

$$\begin{aligned} \mathbf{x}^{(1)} = \hat{\mathbf{x}}^{(1)} \Rightarrow \quad \mathbf{x}^{(2)} &= \mathbf{x}^{(1)} - \mathrm{DF}^{-1} \cdot \mathbf{F}(\mathbf{x}^{(1)}) \\ &= \mathbf{x}^{(1)} - \mathrm{DF}^{-1} \cdot A^{-1} \cdot A \cdot \mathbf{F}(\mathbf{x}^{(1)}) \\ &= \hat{\mathbf{x}}^{(1)} - (\hat{\mathrm{DF}})^{-1} \cdot \hat{\mathbf{F}}(\mathbf{x}^{(1)}) = \hat{\mathbf{x}}^{(2)}. \end{aligned}$$

The accuracy of $\mathbf{x}^{(n)}$ is much better reflected in the value of

$$\frac{\|\Delta\mathbf{x}^{(n)}\|}{\|\mathbf{x}^{(n)}\|},$$

as

$$m = -\left[\log_{10}\left(\frac{\|\Delta\mathbf{x}^{(n)}\|}{\|\mathbf{x}^{(n)}\|}\right)\right]$$

suggests that the first m digits of $\mathbf{x}^{(n)}$ coincide with the fixed point of the iteration.

The Newton iteration is thus terminated if one of the following conditions is satisfied:

1. $\|\mathbf{x}^{(n)}\| \neq 0$, $\|\mathbf{F}(\mathbf{x}^{(n)})\| < \mathbf{TolFun}$ *or* $\|\Delta\mathbf{x}^{(n)}\|/\|\mathbf{x}^{(n)}\| < \mathbf{TolX}$.
2. $\|\mathbf{x}^{(n)}\| = 0$, $\|\mathbf{F}(\mathbf{x}^{(n)})\| < \mathbf{TolFun}$ *or* $\|\Delta\mathbf{x}^{(n)}\| < \mathbf{TolX}$.
3. The maximum number of function evaluations is exceeded.
4. The maximum number of iterations is exceeded.

TolFun and **TolX** are user-defined tolerances, the default values of which have been chosen as

$$\begin{aligned} \mathbf{TolFun} &= 0, \\ \mathbf{TolX} &= 10^{-12}. \end{aligned}$$

[5]Throughout this paper, $\|\cdot\|$ denotes the maximum norm $\|\cdot\|_\infty$.

2.4 Bases and round-off errors

In this section, we examine the influence of round-off errors on the accuracy of the solution approximation $\mathbf{y}(t)$ obtained by collocation. From the coefficients $(\mathbf{a}_{kj})$, the values $\mathbf{y}(\tau_k)$ are calculated as

$$\mathbf{y}(\tau_k) = \begin{pmatrix} \mathbf{a}_{k1} & \cdots & \mathbf{a}_{k,p+1} \end{pmatrix} \begin{pmatrix} \varphi_1^- \\ \vdots \\ \varphi_{p+1}^- \end{pmatrix}, \ (1 \le k \le N),$$

$$\mathbf{y}(\tau_{N+1}) = \begin{pmatrix} \mathbf{a}_{N1} & \cdots & \mathbf{a}_{N,p+1} \end{pmatrix} \begin{pmatrix} \varphi_1^+ \\ \vdots \\ \varphi_{p+1}^+ \end{pmatrix}.$$

Thus if either the Lagrange basis, the Runge-Kutta basis or the monomial basis is chosen for collocation, the values of $\mathbf{y}(\tau_k) = \mathbf{a}_{k1}$ $(1 \le k \le N)$ show the same level of accuracy as the $(\mathbf{a}_{kj})$. It is therefore sufficient to investigate the accuracy of the $(\mathbf{a}_{kj})$, or alternatively, the accuracy of $\mathbf{x}$.

In the linear case, the numerical solution $\mathbf{x}$ is obtained from

$$\mathbf{x} = -\mathrm{DF}^{-1} \cdot \mathbf{F}(\mathbf{0}).$$

The only non-vanishing components of $\mathbf{F}(\mathbf{0})$ correspond to the inhomogeneities of the differential equation and the boundary conditions. If the evaluation of these inhomogeneities is implemented carefully, the relative error of $\mathbf{F}(\mathbf{0})$ will not exceed a small multiple of the machine accuracy **eps**. The same is true for the evaluation of DF, if no cancellation effects occur. The standard conditioning estimates for linear systems then yield

$$\frac{\|\mathbf{x} - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} \le \kappa(\mathrm{DF}) \cdot c \cdot \mathbf{eps},$$

where c is a moderate constant. In practice, this estimate may turn out to be quite pessimistic.

In the nonlinear case, $\mathbf{x}$ is obtained by Newton iteration starting from an initial approximation $\mathbf{x}^{(0)}$. Since any subsequent approximation $\mathbf{x}^{(n)}$ can be interpreted as an exact initial iterate, we have to analyze the round-off errors that occur in the generation of the subsequent approximation, if we want to explain why the step from $\mathbf{x}^{(n)} \to \mathbf{x}^{(n+1)}$ does not improve accuracy, once a certain level has been reached.
To improve the readability of the following sections, we try to avoid too many indices. Therefore we will restrict ourselves to the discussion of scalar problems.

2.4.1 Evaluation of F

To evaluate the discretization residual $\mathbf{F}$, the quantities $y_k^i,\ y_k'^i,\ y_k^+,\ y_k^-$ have to be computed from $\mathbf{x}^{(n)}$ by matrix multiplication, e.g.

$$\begin{pmatrix} y_k^1 & \cdots & y_k^p \end{pmatrix} = \begin{pmatrix} a_{k1} & \cdots & a_{k,p+1} \end{pmatrix} \begin{pmatrix} \varphi_1^1 & \cdots & \varphi_1^p \\ \vdots & \ddots & \vdots \\ \varphi_{p+1}^1 & \cdots & \varphi_{p+1}^p \end{pmatrix}.$$

Marking the computational values by a tilde, we estimate

$$\begin{aligned}
|\widetilde{y}_k^i - y_k^i| &= \left|\widetilde{\sum_{j=1}^{p+1} a_{kj}\varphi_j^i} - \sum_{j=1}^{p+1} a_{kj}\varphi_j^i\right| \\
&\leq (p+1)\frac{\mathbf{eps}}{2}\max_j |a_{kj}\varphi_j^i| \qquad (2.8)\\
&\leq (p+1)\frac{\mathbf{eps}}{2}\max_{i,j} |\varphi_j^i| \cdot \max_{k,j} |a_{kj}| \\
&= (p+1)\frac{\mathbf{eps}}{2}\max_{i,j} |\varphi_j^i| \cdot \|\mathbf{x}\| =: \varepsilon_y \\
|\widetilde{y}_k^- - y_k^-| &\leq \varepsilon_y \\
|\widetilde{y}_k^+ - y_k^+| &\leq \varepsilon_y \\
|\widetilde{y}_k'^i - y_k'^i| &\leq (p+1)\frac{\mathbf{eps}}{2h_{\min}}\max_{i,j} |\varphi_j'^i| \cdot \|\mathbf{x}\| =: \varepsilon_{y'}
\end{aligned}$$

and *assume* that

$$\begin{aligned}
|\widetilde{f}_k^i - f_k^i| &\leq |Df(t_k^i, y_k^i)| \cdot \varepsilon_y + |f_k^i| \cdot \mathbf{eps} \\
&\leq \max_{k,i} |Df(t_k^i, y_k^i)| \cdot \varepsilon_y + \max_{k,i} |f_k^i| \cdot \mathbf{eps} =: \varepsilon_f \\
|\widetilde{R} - R| &\leq \left(\left|\frac{\partial R}{\partial z_a}\right| + \left|\frac{\partial R}{\partial z_b}\right|\right) \cdot \varepsilon_y + |R| \cdot \mathbf{eps} =: \varepsilon_R
\end{aligned}$$

which yields

$$\|\widetilde{\mathbf{F}} - \mathbf{F}\| \leq \max(\varepsilon_R,\, 2\varepsilon_y,\, \varepsilon_{y'} + \varepsilon_f)\,.$$

2.4.2 Simplifications for Lagrange and Runge-Kutta basis

If the Lagrange basis is used as the collocation basis, no error occurs in the evaluation of y_k^- and y_k^i, as

$$\left(y_k^- y_k^1 \;\cdots\; y_k^p\right) = \left(a_{k1} \;\cdots\; a_{k,p+1}\right).$$

We thus obtain

$$\|\widetilde{\mathbf{F}} - \mathbf{F}\| \leq \max\left(\left|\frac{\partial R}{\partial z_b}\right| \cdot \varepsilon_y + |R| \cdot \mathbf{eps},\, \varepsilon_y,\, \varepsilon_{y'} + \max_{k,i} |f_k^i| \cdot \mathbf{eps}\right),$$

and can avoid the *error propagation* in f.
If the Runge-Kutta basis is used, the $y_{kj}'^i$ and y_j^- are exact. This yields

$$\|\widetilde{\mathbf{F}} - \mathbf{F}\| \leq \max(\left|\frac{\partial R}{\partial z_b}\right| \cdot \varepsilon_y + |R| \cdot \mathbf{eps},\, \varepsilon_y,\, \varepsilon_f)\,.$$

2.4.3 Evaluation of DF

The most critical part in the evaluation of the Jacobian $\mathrm{DF}(\mathbf{x})$ is the error propagation in $Df(t_k^i, \widetilde{y}_k^i)$ that occurs for all bases except the Lagrange basis. However, the entries of DF that contain Df terms read

$$\underbrace{\varphi_{kj}'^i}_{O(1/h_k)} - Df(t_k^i, y_k^i) \cdot \underbrace{\varphi_j^i}_{O(1)},$$

and these entries will usually not approach zero as $y(t)$ approaches $y^*(t)$. Their relative error should thus be negligible.

2.4.4 The Newton increment

After evaluation of $\mathbf{F}(\mathbf{x}^{(n)})$ and $\mathrm{DF}(\mathbf{x}^{(n)})$, the solution of the linear system

$$\widetilde{\mathrm{DF}(\mathbf{x}^{(n)})} \cdot \widetilde{\Delta\mathbf{x}^{(n)}} = -\widetilde{\mathbf{F}(\mathbf{x}^{(n)})}$$

is computed. Two kinds of errors affect the accuracy of the Newton increment, round-off errors occurring in the LU-decomposition and error propagation of the perturbed residual $\widetilde{\mathbf{F}}$. As we cannot influence the round-off errors occurring in the LU-decomposition[6], and as it is reasonable to assume that they do not depend on the collocation basis, we only consider error propagation here. With (2.8), we estimate

$$\frac{\|\widetilde{\Delta\mathbf{x}^{(n)}} - \Delta\mathbf{x}^{(n)}\|}{\|\Delta\mathbf{x}^{(n)}\|} \leq \kappa(\mathrm{DF}) \frac{\|\widetilde{\mathbf{F}} - \mathbf{F}\|}{\|\mathbf{F}\|} \leq \kappa(\mathrm{DF}) \frac{\max(\varepsilon_R, 2\varepsilon_y, \varepsilon_{y'} + \varepsilon_f)}{\|\mathbf{F}\|}$$

for any collocation basis.

The relative error of the Newton increment depends significantly on the condition number $\kappa(\mathrm{DF})$. Typical condition number quotients with respect to the condition number $\kappa(\mathrm{DF_{RK}})$ (Runge-Kutta basis) are shown in Figure 2.2. We observed that the quotients do not depend on the number of intervals N.

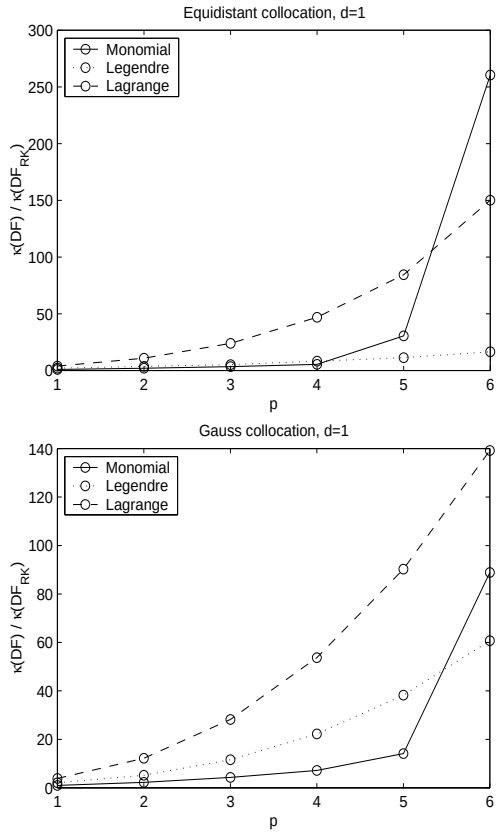

Figure 2.2: Typical condition number quotients for Gauss and equidistant collocation observed for the Emden BVP (3.1)

[6]In MATLAB 6, the algorithm is that of the LAPACK routine `dgetrf`.

2.4.5 Conclusions

Since a Newton step

$$\mathbf{x}^{(n)} \rightarrow \mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \Delta\mathbf{x}^{(n)}$$

will improve the quality of the solution approximation as long as

$$\frac{\|\Delta\mathbf{x}^{(n)}\|}{\|\mathbf{x}^{(n)}\|} \gg \mathbf{eps}$$

and

$$\frac{\|\widetilde{\Delta\mathbf{x}^{(n)}} - \Delta\mathbf{x}^{(n)}\|}{\|\Delta\mathbf{x}^{(n)}\|} \ll 1\,,$$

the conditions that have to be satisfied in order to obtain a reasonable new approximation $\mathbf{x}^{(n+1)}$ from $\mathbf{x}^{(n)}$ are given by

$$\kappa(\mathrm{DF}(\mathbf{x}^{(n)})) \cdot \frac{\max(\varepsilon_R, 2\varepsilon_y, \varepsilon_{y'} + \varepsilon_f)}{\|\mathbf{F}\|} \ll 1, \tag{2.9}$$

$$\frac{\|\Delta\mathbf{x}^{(n)}\|}{\|\mathbf{x}^{(n)}\|} \gg \mathbf{eps}.$$

For the Lagrange and Runge-Kutta basis, (2.9) is replaced by

$$\kappa(\mathrm{DF}(\mathbf{x}^{(n)})) \cdot \frac{\max\left(\left|\frac{\partial R}{\partial z_b}\right| \cdot \varepsilon_y + |R| \cdot \mathbf{eps}, \varepsilon_y, \varepsilon_{y'} + \max_{k,i} |f_k^i| \cdot \mathbf{eps}\right)}{\|\mathbf{F}\|} \ \ll \ 1$$

and

$$\kappa(\mathrm{DF}(\mathbf{x}^{(n)})) \cdot \frac{\max(\left|\frac{\partial R}{\partial z_b}\right| \cdot \varepsilon_y + |R| \cdot \mathbf{eps}, \varepsilon_y, \varepsilon_f)}{\|\mathbf{F}\|} \ \ll \ 1\,,$$

respectively.
Hence if we deal with a problem of moderate $|Df|$, or a linear problem, where no y_k^i evaluation is necessary, the Runge-Kutta basis is usually the best choice. For problems with large Jacobians $Df(t_k^i, y_k^i)$ *at solution values* $y^*(t_k^i)$ *which significantly differ from zero*, the Lagrange basis might be more favorable.

Chapter 3

Numerical tests

3.1 The Newton iteration

Given an initial approximation $\mathbf{x}^{(0)}$ of the solution vector $\mathbf{x}^*$ of the nonlinear system

$$\mathbf{F}(\mathbf{x}) = \mathbf{0},$$

subsequent approximations are computed as

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \underbrace{\mathrm{DF}^{-1}(\mathbf{x}^{(n)}) \cdot (-\mathbf{F}(\mathbf{x}^{(n)}))}_{=\Delta\mathbf{x}^{(n)}}.$$

If the problem is smooth and $\mathbf{x}^{(0)}$ is sufficiently close to $\mathbf{x}^\star$, the order of convergence of the Newton iteration is 2, that is

$$\|\mathbf{x}^{(n+1)} - \mathbf{x}^*\| \leq C \cdot \|\mathbf{x}^{(n)} - \mathbf{x}^*\|^2 .$$

In order to determine the empirical order of convergence of the Newton iteration in our implementation, we compute the values[1]

$$\widetilde{\gamma}^{(n)} = \log\left(\frac{\|\Delta\mathbf{x}^{(n+1)}\|}{\|\Delta\mathbf{x}^{(n)}\|}\right) \Big/ \log\left(\frac{\|\Delta\mathbf{x}^{(n)}\|}{\|\Delta\mathbf{x}^{(n-1)}\|}\right).$$

Table 3.1 shows typical numerical values of a Newton iteration observed for the *Emden boundary value problem*[2],

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}\mathbf{z}(t) - \begin{pmatrix} 0 \\ tz_1^5(t) \end{pmatrix}, \tag{3.1a}$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} \frac{\sqrt{3}}{2} \\ 0 \end{pmatrix}, \tag{3.1b}$$

with exact solution

$$\mathbf{z} = (1/\sqrt{1 + t^2/3}, -t^2/\sqrt{9(1 + t^2/3)^3})^T.$$

A plot of this solution is given in Figure 3.1.

[1] We can expect meaningful results whenever the damping factor λ equals 1, that is whenever the iteration reduces to the classical Newton's method.

[2] Throughout this section, the Emden BVP serves as the test problem. We used the Runge-Kutta basis, equidistant collocation points and collocation degree $p = 4$. For Table 3.1, we chose the stepsizes $h_k \equiv 1/100$ and the initial approximations $x_0(t) \equiv 3/2,\ y_0(t) \equiv 0$ for the first and second solution components, respectively.

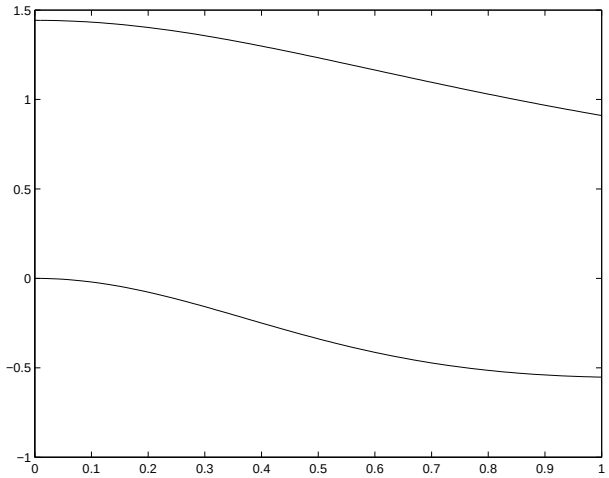

Figure 3.1: Solution of (3.1)

n	Fcount	$\|\mathbf{F}(\mathbf{x}^{(n)})\|$	$\|\Delta\mathbf{x}^{(n)}\|$	$\|\ \mathbf{x}^{(n)}\|$	$\widetilde{\gamma}^{(n)}$	$\lambda^{(n)}$
1	2	7.58e+000	1.88e+000	1.50e+000		0.500
2	6	6.55e+000	9.35e-001	2.23e+000	-0.19	0.125
3	7	5.75e+000	1.07e+000	2.24e+000	-3.02	0.250
4	8	4.38e+000	7.17e-001	2.12e+000	1.70	0.500
5	9	2.31e+000	3.64e-001	2.02e+000	2.51	1.000
6	10	3.94e-001	6.65e-002	1.81e+000	1.01	1.000
7	11	4.01e-002	1.20e-002	1.74e+000	2.30	1.000
8	12	9.40e-004	2.38e-004	1.73e+000	1.92	1.000
9	13	4.23e-007	1.25e-007	1.73e+000	2.01	1.000
10	14	1.07e-013	3.24e-014	1.73e+000		1.000

Table 3.1: Convergence of the damped Newton iteration, $\kappa(\mathrm{DF}(\mathbf{x}^{(10)})) = 7.87 \cdot 10^5$

Note the damping in the first 4 steps, which results in an increase of Fcount, the number of evaluations of $\mathbf{F}$ during the iteration, by more than one in the first two steps.

3.2 The collocation solver

In this section, we examine the influence of subsequent mesh refinement on the condition number of the Jacobian $\mathrm{DF}(\mathbf{x}^{(n_0)})$ and the solution approximation

$$\mathbf{y}(t) = \sum_{k=1}^{N} \Big(\sum_{j=1}^{p+1} \mathbf{a}_{kj}\, \varphi_{kj}(t) \Big) \cdot \chi_{[\tau_k, \tau_{k+1}]}(t).$$

3.2.1 The conditioning of DF

Numerical experiments strongly support the hypothesis that the condition number $\kappa(\mathrm{DF}(\mathbf{x}^{(n_0)}))$ evaluated at the final solution approximation shows the following asymp-

n	Fcount	$\|\mathbf{F}(\mathbf{x}^{(n)})\|$	$\|\Delta\mathbf{x}^{(n)}\|$	$\|\ \mathbf{x}^{(n)}\|$	$\widetilde{\gamma}^{(n)}$	$\lambda^{(n)}$
1	2	6.34e+001	1.88e+000	1.50e+000		0.500
2	6	3.17e+001	9.35e-001	2.23e+000	-0.19	0.125
3	7	2.77e+001	1.07e+000	2.24e+000	-3.02	0.250
4	8	2.08e+001	7.17e-001	2.12e+000	1.70	0.500
5	9	1.04e+001	3.64e-001	2.02e+000	2.51	1.000
6	10	3.94e-001	6.65e-002	1.81e+000	1.01	1.000
7	11	4.01e-002	1.20e-002	1.74e+000	2.30	1.000
8	12	9.40e-004	2.38e-004	1.73e+000	1.92	1.000
9	13	4.23e-007	1.25e-007	1.73e+000	2.01	1.000
10	14	1.07e-013	3.25e-014	1.73e+000		1.000

Table 3.2: Preconditioned Newton iteration, $\kappa(\mathrm{DF}(\mathbf{x}^{(10)})) = 8.62 \cdot 10^3$

totic behavior:

$$\kappa(\mathrm{DF}(\mathbf{x}^{(n_0)})) = O(\frac{1}{h_{\min}} \cdot N), \quad \text{for } h_{\min} \to 0,\ N \to \infty.$$

On a fixed interval $[a, b]$, the condition number therefore responds to subsequent mesh refinement by growing *quadratically*.
A closer look at the Jacobian reveals that a bad scaling is responsible for this behavior, as the non-zero elements of the rows corresponding to the boundary and continuity conditions are $O(1)$, while the non-zero elements of the rows corresponding to the collocation equations are $O(1/h_k)$. Rescaling the nonlinear system $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ to

$$\widetilde{\mathbf{F}}(\mathbf{x}) := A \cdot \mathbf{F}(\mathbf{x}) = \mathbf{0}$$

with A defined as

$$A = \begin{pmatrix} \frac{1}{h_1}\mathbf{1}_d & & & & & & \\ & \mathbf{1}_{pd} & & & & & \\ & & \frac{1}{h_1}\mathbf{1}_d & & & & \\ & & & \mathbf{1}_{pd} & & & \\ & & & & \frac{1}{h_2}\mathbf{1}_d & & \\ & & & & & \ddots & \\ & & & & & & \mathbf{1}_{pd} \end{pmatrix}$$

results in a Jacobian $\widetilde{\mathrm{DF}}(\mathbf{x})$, the condition number of which is virtually *independent* of the stepsizes h_k and grows only *linearly* with N (Figure 3.2).

As has been discussed in §2.3.2, this rescaling does *not* change the course of the Newton iteration as long as round-off error effects in the solution of the linear system

$$\mathrm{DF}(\mathbf{x}^{(n)}) \cdot \Delta\mathbf{x}^{(n)} = \mathbf{F}(\mathbf{x}^{(n)})$$

can be neglected. To illustrate this, we solved the problem (3.1) again using the preconditioner A (Table 3.2). The value of $h_k \equiv h_{\min} = 1/100$ is reflected in the quotient of the respective condition numbers.

Note that the estimates in §2.4.5 are significantly improved in the case where the preconditioner is used.

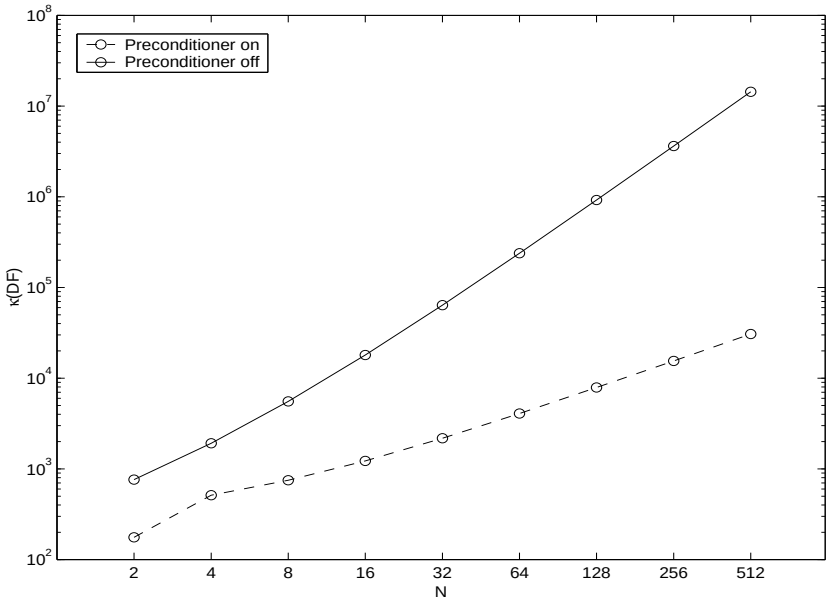

Figure 3.2: Condition numbers of the Jacobians observed for the Emden BVP (3.1)

3.2.2 Orders of convergence

For all tested *regular* and most of the tested *singular* boundary value problems[3], we observe orders of convergence as shown in Table 3.3. For some of the tested singular

Equidistant		Gaussian	
p	order	p	order
1	2	1	2
2	2	2	4
3	4	3	6
4	4	4	8
5	6	5	10
6	6	6	12

Table 3.3: Orders of convergence for equidistant and Gaussian collocation points

problems[4], however, the order of convergence for Gauss collocation is reduced to $p+1$ in general (breakdown of superconvergence due to the singularity, see [13], [18] and the discussion in §5).
Figure 3.3 shows typical results for the error $\|y(t) - y^*(t)\|$ occurring in the solution of the simple *regular* BVP

$$\begin{aligned} &y'(t) = y(t)\,, \quad t \in (0,2), \\ &y(0) + y(2) = 1 + e^2 \end{aligned}$$

on the meshes that subdivide $[0,2]$ into $2, 4, 8, \ldots, 64$ congruent intervals.

[3]This full superconvergence order is observed for the problems (3.1), (5.1), (5.2), (5.3), (5.4), (5.5), (5.6), (5.10), and examples (5.12) and (5.13) discussed in §5.

[4]More precisely, for problems (5.7), (5.8) and (5.9) we observe a convergence order of $p+1$ for p even and of $p+2$ for p odd; the convergence order for example (5.11) is reduced to $p+1/2$, which is probably due to the presence of an eigenvalue $1/2$ of the matrix $M(0)$.

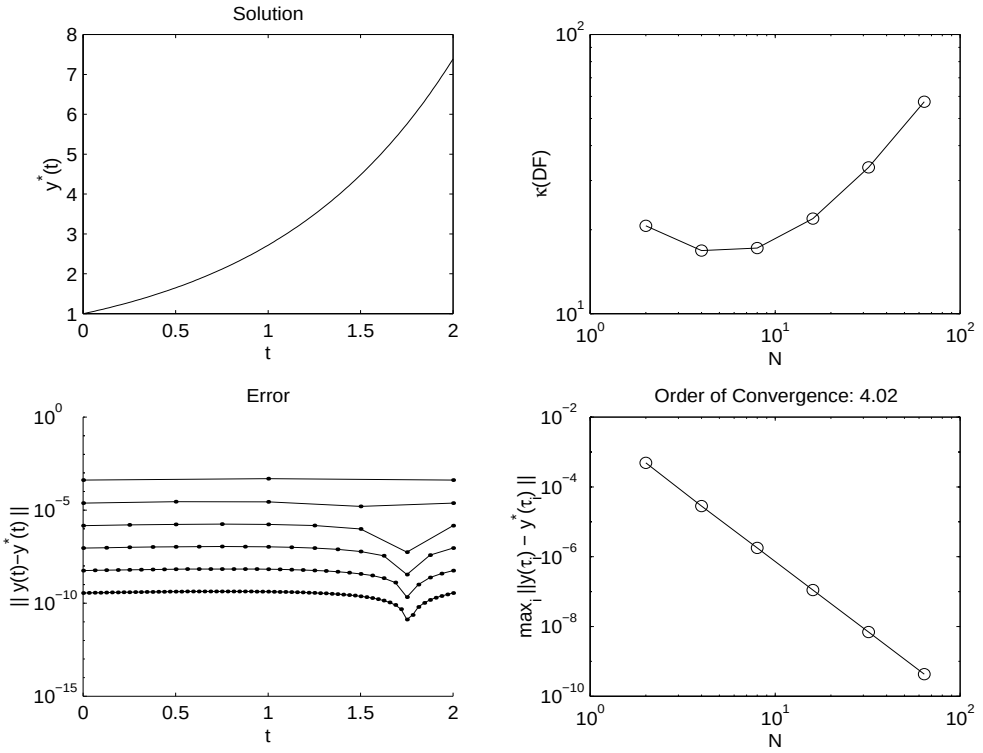

Figure 3.3: Convergence of the sequence of solution approximations for p=4 and equidistant collocation points

p	$\max_k(\|\mathbf{e}(\tau_k)\|)/\max_k(\|\mathbf{y}(\tau_k)\|)$	N_{opt}	$\|\Delta\mathbf{x}^{(n_0)}\|/\|\mathbf{x}^{(n_0)}\|$	$\kappa(\mathbf{x}^{(n_0)})$
4	6.03 **eps**	1024	4.02e-016	6.11e+004
6	2.21 **eps**	64	2.80e-016	6.71e+003
8	3.02 **eps**	32	3.41e-016	1.39e+004

Table 3.4: Attainable accuracy for the Runge-Kutta basis

p	$\max_k(\|\mathbf{e}(\tau_k)\|)/\max_k(\|\mathbf{y}(\tau_k)\|)$	N_{opt}	$\|\Delta\mathbf{x}^{(n_0)}\|/\|\mathbf{x}^{(n_0)}\|$	$\kappa(\mathbf{x}^{(n_0)})$
4	941 **eps**	512	1.36e-014	3.00e+005
6	237 **eps**	32	2.04e-014	7.49e+004
8	300 **eps**	16	2.64e-014	1.64e+005

Table 3.5: Attainable accuracy for the Lagrange basis

3.2.3 Bases and round-off errors

In this section, we investigate the round-off error effects discussed in §2.4 in dependence of the collocation basis.
According to the conclusions in that section, we expect the Runge-Kutta basis to be best suited for the Emden BVP (3.1), and the numerical tests confirm this fact. Tables 3.4 and 3.5 show the respective results, where $\mathbf{e}(\cdot)$ denotes the global error. Note that the accuracy of $\mathbf{x}$ carries over directly to the accuracy of $\mathbf{y}(t)$, and that for the Lagrange basis, the smallest possible error is attained for smaller N than for the Runge-Kutta basis – a direct consequence of the respective condition numbers.

3.3 The error estimate

3.3.1 The conditioning of DG

Similarly as in §3.2.1, a rescaling of the backward Euler system (cf.§2.2.2)

$$\mathbf{G}(\mathbf{u}) = \mathbf{0}$$

to

$$\tilde{\mathbf{G}}(\mathbf{u}) := B \cdot \mathbf{G}(\mathbf{u}) = \mathbf{0}$$

significantly improves the conditioning of the Jacobian, if B is chosen as

$$B = \begin{pmatrix} \frac{1}{\delta_1}\mathbf{1}_d & & & \\ & \mathbf{1}_d & & \\ & & \ddots & \\ & & & \mathbf{1}_d \end{pmatrix}.$$

In the rescaled version, the condition number of the Jacobian responds to subsequent mesh refinement by growing linearly with the number of intervals. Although the conditioning of the auxiliary scheme is not of such prime importance as for the collocation equations, we implemented the preconditioner B as a safety measure. However, we refrain from a detailed analysis of this aspect.

3.3.2 Order of convergence

Denoting the exact global error on the collocation grid by $\mathbf{e}_k := \mathbf{e}(s_k)$ and the error estimate by $\boldsymbol{\varepsilon}_k$, we observed

$$\max_k |\boldsymbol{\varepsilon}_k - \mathbf{e}_k| = O(h_{\max}^{p+1})$$

for all problems tested. For equidistantly spaced collocation points and even collocation degree p, where

$$\max_k |\mathbf{e}_k| = O(h_{\max}^{p})$$

holds, the error estimate is thus asymptotically correct. This is illustrated in Figure 3.4.

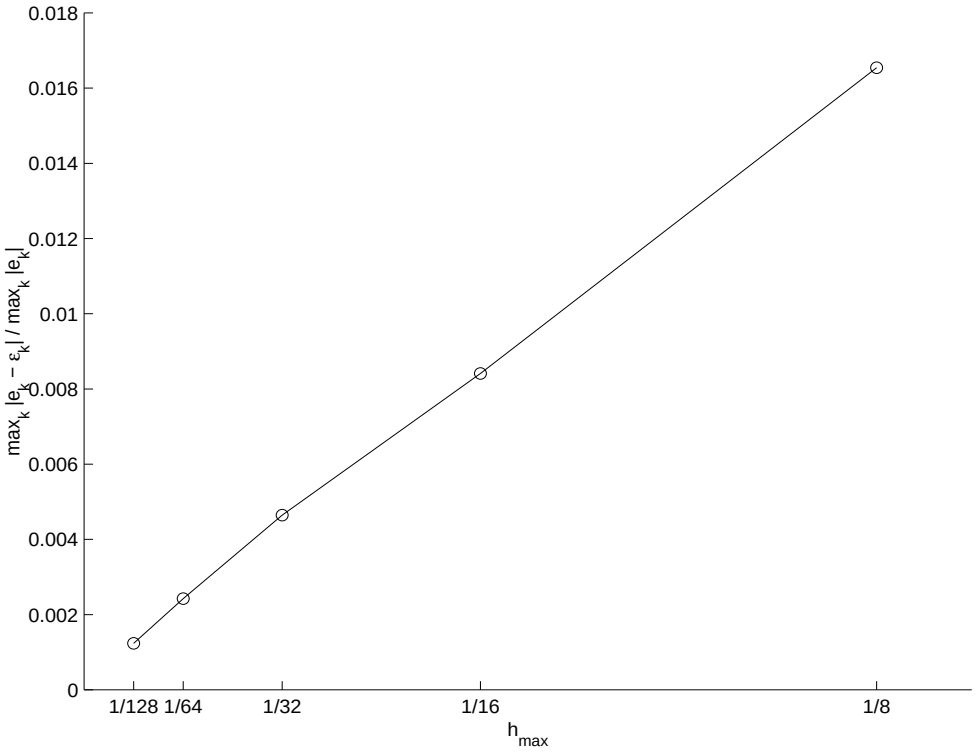

Figure 3.4: Quality of the error estimate observed for the Emden BVP (3.1)

Chapter 4

The solution routines

4.1 Description

4.1.1 Modules

The solver package consists of the files

`sbvp.m`	driver routine (adaptive mesh selection)
`sbvpcol.m`	collocation solver
`sbvperr.m`	error estimation routine
`sbvpset.m`	tool for setting solution options
`sbvpplot.m`	output routine
`sbvpphas2.m`	output routine
`sbvpphas3.m`	output routine
	`bvpfile` that demonstrates how to ...
`demofile1.m`	... define BVPs
`demofile2.m`	... use parameters
`demofile3.m`	... set options
`demofile4.m`	... define multidimensional problems
`demofile5.m`	... set zerofinder options
`demofile6.m`	... use output functions

4.1.2 Solver syntax

From the MATLAB command line or any MATLAB program, `sbvp` is called by

```
[tau,y,tcol,ycol,err_norm] = ...
      sbvp(bvpfile[,tau0,y0,bvpopt,param1,param2,...])
```

The input arguments are:

- `bvpfile` is the function handle or name of an m-file that defines the right-hand side of the differential equation, the boundary conditions and the associated Jacobians. `sbvp` automatically detects whether the BVP is linear. The `bvpfile` syntax is discussed in detail in §4.1.3.

- `tau0` is a strictly monotonous sequence that defines the integration interval and the initial mesh. `tau0` should be a row vector. If `tau0` is empty or not specified, `sbvp` evaluates

  ```
  bvpfile('tau')
  ```

  to obtain it. If `tau0` consists of the beginning and end of the integration interval only (`tau0 = [a b]`), `sbvp` automatically selects a suitable initial mesh on that integration interval. Based on considerations in [4], we choose the number $N+1$ of points in an equidistant initial mesh according to the formula

  $$N+1 = \left\lfloor \frac{1}{\sqrt[p]{\mathbf{AbsTol}}} \right\rfloor,$$

  where **AbsTol** denotes the absolute error tolerance, see below.

- `y0` is an initial approximation of the solution and should be a $d \times (N+1)$-matrix, where d is the dimension of the system to be solved and $N+1$ is the number of mesh points in the initial mesh `tau0`. If `y0` is empty or not specified, `sbvp` evaluates

  ```
  bvpfile('y0',tau0)
  ```

  to obtain it and uses the default initial approximation (identically zero) if that fails. For linear problems, no initial approximation is necessary.

- `bvpopt` is a parameter struct that determines the choice of collocation points, collocation basis, the tolerances for the zerofinder, etc. `bvpopt` should be generated by `sbvpset`, which is discussed in §4.1.4. If `bvpopt` is empty or not specified, `sbvp` evaluates

  ```
  bvpfile('bvpopt')
  ```

  to obtain it and uses the default options if that fails.

- `param1`, `param2` are parameters of the boundary value problem that are passed on to `bvpfile` (see the header of `BVPFilePattern` in §4.1.3).

The output arguments are:

- `tau` is the final mesh generated by `sbvp`.
- `y` is the final solution approximation on the mesh `tau`.
- `tcol` is the final collocation grid. The collocation grid contains mesh points and collocation points and is thus finer than `tau`.
- `ycol` is the final solution approximation on the collocation grid `tcol`.
- `err_norm` is the norm of the last error estimate given on `tcol`.

4.1.3 The `bvpfile`

A `bvpfile` is a MATLAB m-file that defines the boundary value problem to be solved. The generic `bvpfile` reads

```
function out=BVPFilePattern(flag,t,y,ya,yb,param1,param2,...)

switch flag
case 'f'       % right-hand side of the differential equation
```

```
  out=<insert f(t,y) here>;
case 'df/dy'   % Jacobian of f
  out=<insert df/dy(t,y) here>;
case 'R'       % boundary conditions
  out=<R(ya,yb)>;
case 'dR/dya'  % Jacobian of the boundary conditions w.r.t. ya
  out=<dR/dya(ya,yb)>;
case 'dR/dyb'  % Jacobian of the boundary conditions w.r.t. yb
  out=<dR/dyb(ya,yb)>;
case 'tau'    % (optional)
  out=<initial mesh>;
case 'y0'      % (optional)
  out=<initial approximation y0(t)>
case 'bvpopt'  % (optional)
  out=<solution options for the BVP>
otherwise
  error('unknown flag');
end
```

To specify a boundary value problem, one should simply provide explicit expressions for the terms enclosed by `<...>`.

If the underlying boundary value problem is linear, the `bvpfile` can (but need not) be coded such that it returns the inhomogeneities of the differential equation and the boundary conditions only. Note that in order to keep the syntax consistent with `R(ya,yb)=dR/dya*ya+dR/dyb*yb-beta=0`, `-beta` has to be specified in the latter case.

As an example, choose the Emden BVP (3.1). The corresponding `bvpfile` is given by

```
function out = my_bvpfile(flag,t,y,ya,yb)
switch flag
case 'f'       % right hand side of the diff. eq.
   out = [y(2)/t ; -y(2)/t - t*y(1)^5];
case 'df/dy'   % Jacobian of f
   out = [0  1/t ; -t*5*y(1)^4  -1/t];
case 'R'       % Boundary condition.
   out = [yb(1)-sqrt(3)/2 ; ya(2)];
case 'dR/dya'  % Jacobian of the BC
   out = [0 0;0 1];
case 'dR/dyb'  % Jacobian of the BC
   out = [1 0;0 0];
case 'tau'    % mesh
   out = [0 1];
otherwise
   error('unknown flag');
end
```

From the MATLAB command line, the problem is then solved by

```
>> [tau,y] = sbvp(@my_bvpfile);
```

Optionally, we can add the cases

```
case 'y0'     % initial approximation
   out=[ones(1,size(t)) ; zeros(1,size(t))];
case 'bvpopt' % solution options
   out=sbvpset('Basis','Lagrange');
```

to `my_bvpfile` to define an initial approximation and, in this example, enforce the use of the Lagrange basis for collocation.

However, even if `tau0`, `y0` and `bvpopt` are specified in the `bvpfile`, any of these quantities may be passed as a command line argument. Command line arguments override the corresponding definitions in the `bvpfile`. Arguments that should still be obtained from the `bvpfile` can be omitted or indicated by `''`, respectively.

Calling syntax 1:

```
>> tau0 = linspace(0,1,51);            % define an initial mesh
>>
>> [tau,y] = sbvp(@my_bvpfile,tau0);   % sbvp determines y0 and bvpopt
                                       % from my_bvpfile
```

Calling syntax 2:

```
>>bvpopt = sbvpset('Basis','Lagrange','DegreeSelect','manual','Degree',6);
>>
>>[tau,y] = sbvp(@my_bvpfile,'','',bvpopt);   % sbvp determines tau0 and y0
                                               % from my_bvpfile
```

By default, the user is informed about the mesh adaptation process, e.g.

```
Determine initial mesh ... N = 31
Solve BVP ... 0.11 seconds
Estimate error ...  0.06 seconds
Equidistribute error ... N = 56
Control hmax to hmin ratio ... N = 57
Solve BVP ... 0.11 seconds
Estimate error ...  0.06 seconds

Tolerances satisfied
```

This feature can be turned off, however.

4.1.4 Solution options

All options that concern the solution process should be collected in the options structure `bvpopt` using the command

```
bvpopt = sbvpset([property,value,property,value,...]);
```

Any unspecified properties have default values. It is sufficient to type only the leading characters that uniquely identify the property. Case is ignored for property names. `bvpopt` is then passed to `sbvp` as an argument.

Additionally,

```
bvpopt = sbvpset(oldopt[,property,value,property,value,...]);
```

alters an existing options structure `oldopt` and

```
bvpopt = sbvpset(oldopt,newopt);
```

combines an existing options structure `oldopt` with a new options structure `newopt`. Any new properties overwrite corresponding old properties. `sbvpset` with no input arguments displays all property names and their possible values.

Available options are

'**RelTol**'	Relative error tolerance [nonnegative scalar {1e-3}]. The estimated error $\boldsymbol{\varepsilon}(t)$ on the final mesh satisfies $$\max_t(\|\boldsymbol{\varepsilon}(t)\|/(\mathbf{AbsTol} + \|\mathbf{y}(t)\| \cdot \mathbf{RelTol})) < 1.$$
'**AbsTol**'	Absolute error tolerance [positive scalar {1e-6}].
'**MaxMeshPts**'	Maximum number of mesh points [integer {10000}].
'**IntMaxMinRatio**'	Upper limit to the quotient of the length of the longest and shortest subinterval in the final mesh [positive scalar {10}].
'**CheckJac**'	Compare user supplied Jacobians to finite-difference approximations [{0} | 1].
'**fVectorized**'	Vectorized **f**-evaluation [{0} | 1]. Set this property if the `bvpfile` is coded such that `bvpfile('f',[t1 t2 ...],[y1 y2 ...])` returns `[f(t1,y1) f(t2,y2) ...]`.
'**JacVectorized**'	Vectorized Jacobian-evaluation [{0} | 1]. Set this property if the `bvpfile` is coded such that `bvpfile('df/dy',[t1 t2 ...],[y1 y2 ...])` returns `[f'(t1,y1) f'(t2,y2) ...]`, or alternatively a three-dimensional array `J` where `J(:,:,i)` contains `f'(ti,yi)`.
'**Basis**'	Basis of collocation polynomials [{RungeKutta} | Lagrange | Legendre | Monomial].
'**ColPts**'	Distribution of collocation points [{equidistant} | gauss | numerical vector containing $0 < \rho_1 < \cdots < \rho_p < 1$)].

'**DegreeSelect**'	Selection mode of basis polynomial degree [{auto} | manual].
'**Degree**'	Highest degree of basis polynomials [integer {4}]. The collocation degree is set to the value of this property if the value of 'DegreeSelect' is 'manual'. Otherwise, the collocation degree is automatically determined from the tolerance parameter **AbsTol**. This choice is based on test results given in [4].
'**Display**'	Displays information about the mesh adaptation process [0 | {1}].
'**ZfMethod**'	Defines the method for solving nonlinear problems [{Newton} | FSolve]. 'Newton' forces the use of damped Newton iteration while 'FSolve' uses the zerofinder provided by MATLAB.
'**ZfOpt**'	Parameter struct used by the zerofinding routines [struct]. This struct holds tolerances, termination conditions, etc. and should be generated by `optimset` (see §4.1.5).
'**OutputFcn**'	Function handle or name of output function (OF) [handle/string]. OFs are used to display intermediate results of the mesh adaptation process. At the beginning of this process, the solver calls `outputfcn(tcol0,ycol0,p,'init')` to initialize the OF. `tcol0` represents the initial collocation grid, `ycol0` is the initial approximation on `tcol0` and `p` is the collocation degree. After each mesh adaptation step with collocation grid `tcol` and approximation vector `ycol`, the solver calls `stop = outputfcn(tcol,ycol,p,'')`. If `stop` equals 1 or some termination condition of the mesh adaptation routine is met, the iteration is stopped and the solver calls `outputfcn(tcol,ycol,p,'done')`. Predefined OFs are `sbvpplot`, `sbvpphas2` and `sbvpphas3` (see §4.1.6).
'**OutputSel**'	Output selection indices [vector of integers]. This vector of indices specifies which components of the approximation vector `y` are passed to the OF. 'OutputSel' defaults to all components.
'**OutputTrace**'	Flag that forces stepwise proceeding of the output function [0 | {1}]. If 'OutputTrace' is set to 1, `sbvp` pauses after each mesh adaptation step and continues if a key is hit. If no output function is defined, 'OutputTrace' is ignored.

4.1.5 Zerofinder options

Options for the MATLAB zerofinder `fsolve`[1] can be set using the `optimset` command

```
my_fsolveopt = optimset([property,value,property,value,...]);
```

Since it is possible to switch between `fsolve` and the Newton zerofinder (cf. 'ZfMethod' on page 32), options for the Newton zerofinder are set in the same way. Defining options for any zerofinder is thus achieved by

```
my_zfopt = optimset([property,value,property,value,...]);
```

and `my_zfopt` is used to define the BVP options

```
my_bvpopt = sbvpset('ZfOpt',my_zfopt,...);
```

Available options for the Newton zerofinder *and* `fsolve` are

'**Display**'	Level of display [{final} | iter | off]. If 'Display' is set to 'final', the zerofinder reports how the iteration has been terminated. 'iter' will display such quantities as the number of function evaluations, the value of the residual $\mathbf{F}(\mathbf{x})$ and the stepsize λ of each iteration step. 'off' will display no output at all.
'**MaxIter**'	Maximum number of iterations allowed [integer {20}].
'**MaxFunEvals**'	Maximum number of function evaluations allowed [integer {50}].
'**TolX**'	Termination tolerance on $\mathbf{x}$ [positive scalar {1e-12}].
'**TolFun**'	Termination tolerance on the function value $\mathbf{F}(\mathbf{x})$ [nonnegative scalar {0}].

The following options affect `fsolve` only and are ignored by the Newton zerofinder. They are discussed briefly in the MATLAB HelpDesk and in detail in the User Guide to the Optimization Toolbox.

'**DerivativeCheck**'	'**Jacobian**'	'**LineSearchType**'
'**Diagnostics**'	'**JacobianPattern**'	'**MaxPCGIter**'
'**DiffMaxChange**'	'**LargeScale**'	'**TolPCG**'
'**DiffMinChange**'	'**LevenbergMarquardt**'	'**TypicalX**'

The use of `fsolve` may however result in a significantly longer runtime of the program. In the same situation as in Table 3.1, where our solution routine required 10 steps and 14 function evaluations of the system of nonlinear equations for the coefficients of the collocation solution, the same computation using `fsolve` took 29 iterations and 30 function evaluations.

4.1.6 Output functions

An output function (OF) is a MATLAB m-file that is used to monitor the mesh adaptation process. If an OF is specified, it will be passed the result of each adaptation

[1] `fsolve` is included in the Optimization Toolbox.

step. The predefined OFs `sbvpplot`, `sbvpphas2` and `sbvpphas3` all display the solution approximation on the current mesh in order to illustrate the adaptation process graphically. `sbvpplot` simply draws the solution components versus t, `sbvpphas2` and `sbvpphas3` plot 2- and 3-dimensional phase portraits, respectively. Figure 4.1 shows three subsequent solution approximations of (5.1) displayed by the OFs `sbvpplot` (left) and `sbvpphas2` (right).

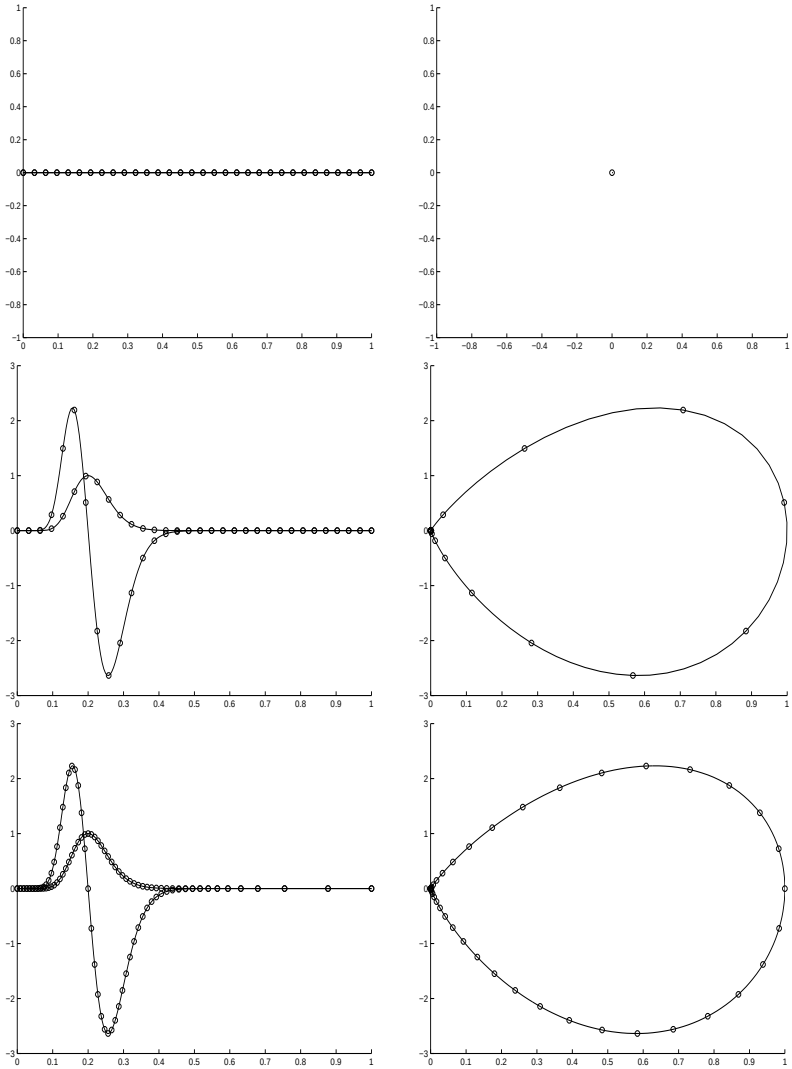

Figure 4.1: Initial guess and two subsequent solution approximations displayed by `sbvpplot` (left) and `sbvpphas2` (right)

4.1.7 Log-structs and test routines

In order to be able to monitor the BVP solution process conveniently, the routine `sbvpcol`, which computes the collocation solution on a given mesh `tau`, has been extented by a log-procedure. `sbvpcol`, which uses the same input syntax as `sbvp`, returns the mesh and the collocation grid, as well as the solution approximation $\mathbf{y}(t)$

on them. If `sbvpcol` is called with 7 output arguments[2]

```
[y_tau,tau,y_s,s,f,fprime,log] = sbvpcol(...),
```

it will return a log-struct `log` that contains information about the solution process. `log` comprises the fields

'**bvpfile**'	Name of the `bvpfile`
'**bvpopt**'	Options that have been used in the solution process
'**tau**'	The mesh
'**parameters**'	Parameters of the `bvpfile`
'**condestDF**'	Condition number of DF evaluated at the final solution approximation

for both linear and nonlinear problems and the fields

'**y0**'	Initial approximation
'**norm_F**'	Vector of the norms $\|\mathbf{F}(\mathbf{x}^{(n)})\|$
'**norm_delta_x**'	Vector of the norms $\|\Delta\mathbf{x}^{(n)}\|$
'**lambda**'	Vector of the values $\lambda^{(n)}$ used in the iteration process
'**norm_x**'	Vector of the norms $\|\mathbf{x}^{(n)}\|$, where $\mathbf{x}^{(n)}$ is the solution approximation of the n-th Newton step
'**itcount**'	Number of Newton steps
'**Fcount**'	Number of function evaluations (evaluations of $\mathbf{F}$)

for nonlinear problems only, at least if the Newton zerofinder has been chosen. If `fsolve` is used to solve nonlinear equations, `log` contains the logstruct returned by `fsolve`.

4.2 Developing the solver

4.2.1 Efficient referencing

In order to optimize the run-time of a program such as `sbvp.m`, MATLAB's internal array handling has to be taken into consideration.
There is little to say about one-dimensional arrays. No matter if they represent row or column vectors, the elements of a vector are always stored in linear succession in the

[2]The output arguments `f` and `fprime` are intended for internal use of `sbvp` only. They serve to reduce the number of evaluations of $\mathbf{f}(t,\mathbf{y})$ and $\mathbf{f}'(t,\mathbf{y})$ in the error estimation routine.

computer memory:

array	memory
$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$	$a_1 \dots a_n$
$(a_1, \dots, a_n)$	$a_1 \dots a_n$

Two-dimensional arrays are more interesting, as they are stored *columnwise*,

array	memory
$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$	$a_{11}\ a_{21}\ \cdots\ a_{(n-1)n}\ a_{nn}$

and hence referencing rows and columns differs significantly. Extracting a column of a matrix and assigning it to another variable means copying *a connected memory area*, whereas for referencing a row, the appropriate elements have to be singled out first. Referencing columns is thus faster than referencing rows.
Three-dimensional arrays `A(1:k,1:l,1:m)` are represented in memory as

$$a_{111}\ \cdots\ a_{k11}\ a_{121}\ \cdots\ a_{kl1}\ a_{112}\ \cdots\ a_{kl2}\ \cdots\ a_{klm},$$

therefore referencing subarrays (matrices)

```
A(:,:,j)
```

is fastest.

4.2.2 Vectorization

Vectorization is another technique of reducing the run-time of a MATLAB program. The idea is to use as few loops as possible. So if there exists a MATLAB command that performs the task of a loop, it should be used instead. As an example, we discuss a vectorized way to assemble a matrix

$$B = \big(\underbrace{A\ A\ \dots\ A}_{m \text{ times}}\big)$$

from a given $n \times n$ matrix A. The source code of the straightforward loop version reads

```
B=zeros(m*n,n); % preallocation of memory
for k=1:m
    B(:,(k-1)*m+1:k*m) = A;
end
```

but this can easily be improved. First arrange the elements of A in linear succession,

```
a = A(:)
```

then use

```
b = a * ones(1,m)
```

straightforward loop	100 %
lo-hi loop	83 %
idx loop	61 %
vectorized	31 %

Table 4.1: Relative run-times

to build

$$b = \begin{pmatrix} a_{11} & \cdots & a_{11} \\ a_{21} & \cdots & a_{21} \\ \vdots & & \vdots \\ a_{nn} & \cdots & a_{nn} \end{pmatrix}.$$

The internal, linear memory representations of `b` and `B` already coincide, only the shape is different. Reshaping `b` yields the desired result,

```
B = reshape(b,n,n*m)
```

Combining all these steps, we obtain

```
B = reshape(A(:)*ones(1,m), n, n*m)
```

4.2.3 Indexing

In cases where no vectorized alternatives are available, the efficiency can still be enhanced by avoiding repeated calculations of indices which are used several times. For the above example, such a loop alternative could read

```
idx = reshape(1:n*m,n,m);  % assemble indices
B=zeros(m*n,n);            % preallocation of memory
for k=1:m
    B(:,idx(:,m)) = A;
end
```

This method of indexing is quite fast, but it is also quite memory consuming. In fact, the index vector `idx` needs the same amount of memory as `B`. A less expensive alternative would be

```
lo = 1:n:n*m;
hi = n:n:n*m;     % hi = lo + (n-1)
B=zeros(m*n,n);   % preallocation of memory
for k=1:m
    B(:,lo(m):hi(m)) = A;
end
```

Table 4.1 shows the relative execution times of all the discussed alternatives compared to the straightforward loop implementation for $n = 5$, $m = 10$.

4.2.4 Efficient storage of `sbvpcol` variables

The MATLAB representations of the quantities

$$\mathbf{a}_{kj},\ \varphi_j^i,\ \varphi_j^+,\ \varphi_j^-,\ \varphi_{kj}'^i,$$

are chosen such that no row references and only few loops are necessary in the evaluation of the residual $\mathbf{F}(\mathbf{x})$ and the Jacobian $\mathrm{DF}(\mathbf{x})$.

The coefficient vector $\mathbf{a}_{kj}$

The $(\mathbf{a}_{kj})$ form a three-dimensional array and are thus stored in this way:

$$\texttt{a(l,j,k)} = a_{kj}^l.$$

Thus, the linear order of the values in the memory coincides with the linear order of the coefficients (2.2) that was used to define the Jacobian. Hence, conversions between the MATLAB representations of $\mathbf{x}$ and $(\mathbf{a}_{kj})$ are achieved by a simple `memcopy` command,

$$\texttt{x=a(:), a(:)=x}.$$

Furthermore,

$$\texttt{a(:,:,k)} = \begin{pmatrix} \mathbf{a}_{k1} & \cdots & \mathbf{a}_{k,p+1} \end{pmatrix},$$

i.e. the coefficients that belong to one interval $[\tau_k, \tau_{k+1}]$, and therefore appear in conjunction often, can be referenced quickly.

The φ-terms

Once the constant parts Φ, Φ' of the Jacobian $\mathrm{DF}(\mathbf{x})$ have been assembled, the φ_j^i are used in the computation of

$$\mathbf{y}_k^i = \sum_{j=1}^{p+1} \mathbf{a}_{kj} \cdot \varphi_j^i$$

only. Representing the φ_j^i as

$$\texttt{phi(i,j)} = \varphi_j^i \quad \Rightarrow \quad \texttt{phi} = \begin{pmatrix} \varphi_1^1 & \cdots & \varphi_1^p \\ \vdots & \ddots & \vdots \\ \varphi_{p+1}^1 & \cdots & \varphi_{p+1}^p \end{pmatrix}$$

allows a very efficient calculation of the $\mathbf{y}_k^i$ as a simple matrix product,

$$\begin{pmatrix} \mathbf{y}_k^1 & \cdots & \mathbf{y}_k^p \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{k1} & \cdots & \mathbf{a}_{k,p+1} \end{pmatrix} \begin{pmatrix} \varphi_1^1 & \cdots & \varphi_1^p \\ \vdots & \ddots & \vdots \\ \varphi_{p+1}^1 & \cdots & \varphi_{p+1}^p \end{pmatrix},$$

or in MATLAB notation

$$\texttt{y = a(:,:,k) * phi}.$$

Following the same argumentation, the MATLAB representations of $\varphi_j^+, \varphi_j^-, \varphi_{kj}'^i$ become

$$\texttt{phi\_plus} = \begin{pmatrix} \varphi_1^+ \\ \vdots \\ \varphi_{p+1}^+ \end{pmatrix}, \quad \texttt{phi\_minus} = \begin{pmatrix} \varphi_1^- \\ \vdots \\ \varphi_{p+1}^- \end{pmatrix},$$

$$\texttt{phi\_prime(i,j,k)} = \varphi_{kj}'^i.$$

Chapter 5

Comparisons

To conclude this presentation, we give a comparison of the performance of `sbvp` with the standard MATLAB 6.0 routine `bvp4c` and the Fortran 90 code COLNEW. For a description of these solvers, see [16] and [2], respectively. It turns out that the packages were obviously written with a different scope and for different platforms, and therefore the conclusion is not to favor any particular implementation, but to learn the advantages and drawbacks of each code. Here, we illustrate that for singular problems with a smooth solution, `sbvp` is competitive especially in the linear case, and we point out possible improvements for future versions of `sbvp`.

First, we consider the singular boundary value problem

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 1+\alpha^2 t^2 & 0 \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ ct^{k-1}\mathbf{e}^{-\alpha t}(k^2-1-\alpha t(1+2k)) \end{pmatrix}, \tag{5.1a}$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} 0 \\ c\mathbf{e}^{-\alpha} \end{pmatrix}, \tag{5.1b}$$

where $\alpha = 80$, $k = 16$ and $c = \left(\frac{\alpha}{k}\right)^k \mathbf{e}^k$. Note that this problem was used with a different choice of parameters to illustrate the performance of our mesh selection procedure in [5]. The exact solution is

$$\mathbf{z}(t) = (ct^k\mathbf{e}^{-\alpha t}, ct^k\mathbf{e}^{-\alpha t}(k-\alpha t))^T.$$

A plot of this solution is given in Figure 5.1.

In Table 5.1 we show the number of mesh points (N) and the number of function evaluations of the right-hand side of (5.1a) (fcount) that the different solvers required to reach a relative (**RelTol**) and an absolute (**AbsTol**) tolerance of 10^{-5}. To demonstrate that the computed results are indeed meaningful, we also provide the values of the error estimate (esterr) and the true error (maxerr). We tested 7 different methods for this task:

- `bvp4c`, which is based on collocation at three Lobatto points (see [16]). This is a method of order 4 for regular problems. Note that `bvp4c` does not provide an error estimate to the user.
- COLNEW. The basic method here is collocation at Gaussian points. We chose the collocation degrees $p = 4$ (CW-4) and $p = 6$ (CW-6), which results in (superconvergent) methods of orders 8 and 12 (for regular problems), respectively.

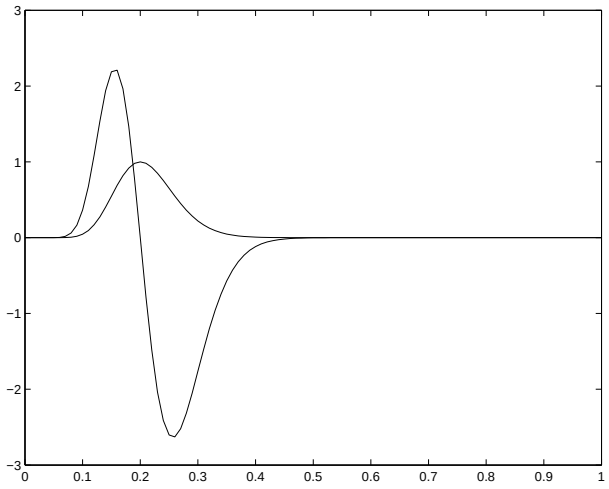

Figure 5.1: Solution of (5.1)

	bvp4c	CW-4	CW-6	sbvp4	sbvp4g	sbvp6	sbvp6g
N	116	41	21	40	40	20	14
fcount	3622	340	390	283	413	178	136
esterr		4.42e−05	1.26e−07	1.66e−05	6.52e−06	8.68e−06	4.25e−06
maxerr	5.77e−06	3.18e−06	7.34e−07	2.98e−06	5.12e−06	3.37e−06	4.30e−07

Table 5.1: Comparisons for (5.1), **AbsTol**=**RelTol**=10^{-5}

- `sbvp` is used with equidistant (`sbvp4` and `sbvp6`) and Gaussian (`sbvp4g` and `sbvp6g`) collocation points and polynomial degrees 4 and 6.

The results given in Table 5.1 show that the performance of `sbvp` is comparable with that of COLNEW for this linear problem. For $p = 6$, which `sbvp` chooses as a default for these tolerances, the number of function evaluations is noticeably smaller. In that case, collocation at Gaussian points shows a slightly more advantageous behavior as compared to equidistant nodes. This is not the case, however, for $p = 4$. This is probably caused by our error control mechanism, which is based on the *stage order* p, since for this example, the solution at the mesh points shows the (super-)convergence order $2p$, see Table 5.2. This cannot be expected in general however. Moreover, the uniform convergence order which is crucial for our procedure is only $p + 1$. `bvp4c` encounters some difficulties solving this problem, which is apparent from the number of mesh points required as well as from the count of function evaluations. The latter is even less favorable because `bvp4c` uses a finite difference approximation for the Jacobian which requires additional evaluations of the right-hand side. The reason for this is the choice of collocation points ($\rho_1 = 0$), which on the one hand forces the user to modify the right-hand side by computing $z'(0)$ using Taylor's method, see [16], and moreover makes it impossible to specify a Jacobian with a singularity at $t = 0$. However, this inefficiency is compensated to some degree by the evaluations of the analytic Jacobian carried out in COLNEW and `sbvp`, which have not been minored here. But even if we measure the total CPU-time used for the computations, the 1.06 seconds required by `sbvp4` show a clear advantage as compared to the 7.82 seconds it takes `bvp4c` to complete the task, see Table 5.10.

Another interesting aspect is the maximal accuracy that could be achieved by the three

N	err	ord
32	5.91e−06	
64	3.50e−08	7.39
128	1.51e−10	7.85
256	6.11e−10	7.95

Table 5.2: Convergence order of collocation at Gaussian points, $m = 4$, for (5.1)

programs for (5.1). For tolerances **AbsTol** = **RelTol** = 10^{-14} (and a restriction on the mesh size of $N \leq 10^4$), `sbvp6g` was able to compute a solution on a mesh with $N = 253$ and an estimated error of $6.47 \cdot 10^{-15}$. The true error of this solution was $4.88 \cdot 10^{-15}$. In the same situation, CW-6 yields a solution on 321 mesh points with an estimated error of $8.82 \cdot 10^{-16}$ and a true error of $3.15 \cdot 10^{-14}$. The strictest tolerance that could be satisfied using `bvp4c` was **AbsTol**=**RelTol**=10^{-11}, which yielded a solution on a mesh containing 5533 points with a true error of $1.25 \cdot 10^{-12}$.

To conclude the discussion of (5.1), we give a graphical representation of the meshes and exact and estimated errors (denoted by $\star$ and $\bullet$, respectively) generated by `bvp4c` (Figures 5.2 and 5.3 — no error estimate can be given in Figure 5.3), CW-4 (Figures 5.4 and 5.5) and `sbvp4` (Figures 5.6 and 5.7) for **AbsTol**=**RelTol**=10^{-5}.

Figure 5.2: Mesh generated by `bvp4c` for (5.1), **AbsTol**=**RelTol**=10^{-5}

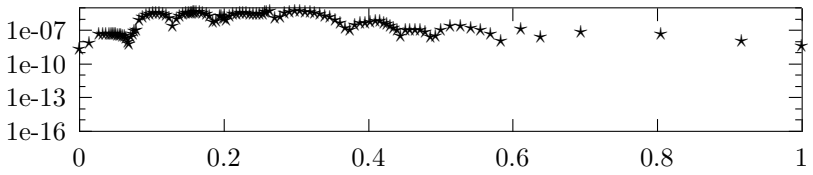

Figure 5.3: Exact global error for `bvp4c` applied to (5.1)

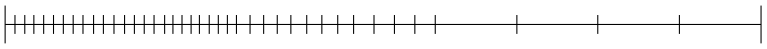

Figure 5.4: Mesh generated by CW-4 for (5.1), **AbsTol**=**RelTol**=10^{-5}

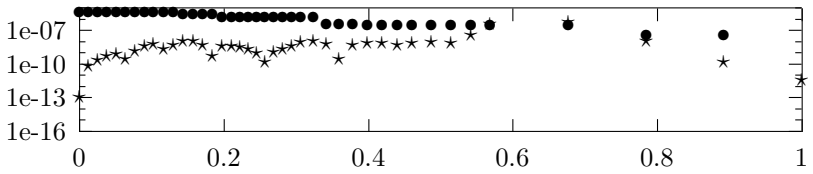

Figure 5.5: Exact ($\star$) and estimated ($\bullet$) global error for CW-4 applied to (5.1)

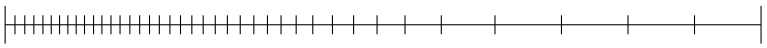

Figure 5.6: Mesh generated by `sbvp4` for (5.1), **AbsTol**=**RelTol**=10^{-5}

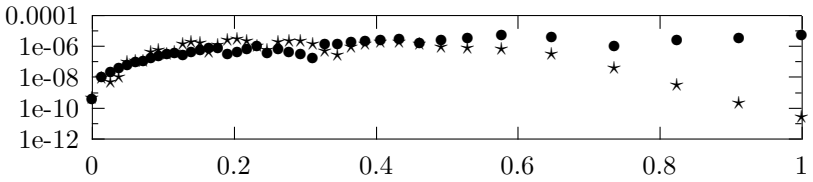

Figure 5.7: Exact ($\star$) and estimated ($\bullet$) global error for `sbvp4` applied to (5.1)

The next example we discuss is

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 2 & 6 \end{pmatrix}\mathbf{z}(t) - \begin{pmatrix} 0 \\ 4k^4t^5\sin(k^2t^2) + 10t\sin(k^2t^2) \end{pmatrix}, \tag{5.2a}$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} 0 \\ \sin(k^2) \end{pmatrix}, \tag{5.2b}$$

where $k = 5$. The exact solution of this problem is

$$\mathbf{z}(t) = (t^2\sin(k^2t^2), 2k^2t^4\cos(k^2t^2) + 2t^2\sin(k^2t^2))^T,$$

and its plot is given in Figure 5.8.

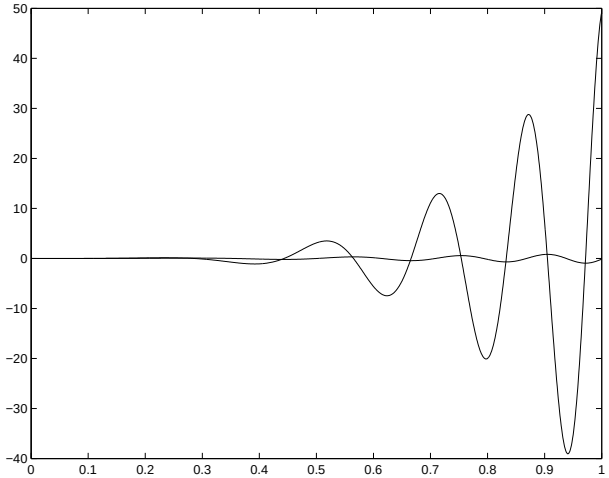

Figure 5.8: Solution of (5.2)

Table 5.3 shows the mesh size N and the function counts fcount for the methods discussed earlier and in addition for collocation of degree 8 in `sbvp`, which is the default value for the chosen tolerances 10^{-9}. Obviously, the fixed order of `bvp4c` is a disadvantage for such strict tolerances. Collocation at Gaussian points is advantageous for this example, since no order reduction is observed. For Gaussian points, `sbvp` is

	bvp4c	CW-4	CW-6	sbvp4	sbvp4g	sbvp6	sbvp6g	sbvp8	sbvp8g
N	2718	641	133	1324	541	154	109	55	37
fcount	102362	5140	1836	11533	3883	2005	1228	921	606

Table 5.3: Comparisons for (5.2), **AbsTol**=**RelTol**=10^{-9}

competitive with COLNEW, and the use of order 8 results in a very satisfactory performance. Moreover, all the error estimates are quite reliable for this example: maxerr $\approx$ esterr $\approx$ **AbsTol**=**RelTol** holds in every case.

Finally, we discuss in detail the results for a nonlinear example. This is a second order problem taken from [16]. In contrast to the cited paper, where the standard transformation $y(t) \to (y(t), y'(t))$ to a first order system of differential equations is used, we use Euler's transformation $y(t) \to (y(t), ty'(t)) =: \mathbf{z}(t)$ to obtain

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ t\varphi^2 z_1(t)\exp\left(\frac{\gamma\beta(1-z_1(t))}{1+\beta(1-z_1(t))}\right) \end{pmatrix}, \tag{5.3a}$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{5.3b}$$

For the parameter values $\varphi = 0.6,\ \gamma = 40,\ \beta = 0.2$, this problem has multiple solutions. If we start the computation at the initial solution profile $z(t) \equiv (1, 0)^T$, we obtain the solution shown in Figure 5.9.

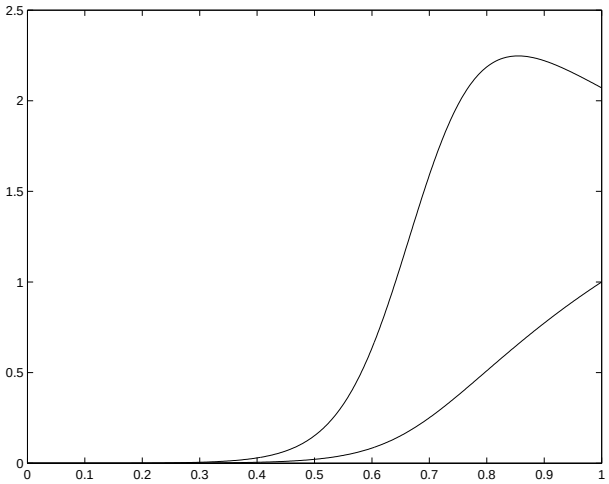

Figure 5.9: One solution of (5.3)

Table 5.4 gives the values of N and fcount for the same methods as discussed before and **AbsTol**=**RelTol**=10^{-7}. The meshes generated by COLNEW and `sbvp` are comparable in size. However, the associated values of fcount are not very favorable for `sbvp`, although using Gaussian points improves the performance noticeably (no order reduction is encountered in this case, either). Accordingly, a quasi-Newton iteration could be worthwhile trying instead of the damped Newton method described in §3.1. The situation changes in favor of `sbvp` for tolerances 10^{-9}, see Table 5.5. However, in this case `sbvp` slightly underestimates the global error.

We now give a comprehensive overview of the numerical experiments. The set of 14

	bvp4c	CW-4	CW-6	sbvp4	sbvp4g	sbvp6	sbvp6g
N	205	41	21	57	57	22	15
fcount	6856	1080	840	6051	3699	3741	1431

Table 5.4: Comparisons for (5.3), **AbsTol**=**RelTol**=10^{-7}

	bvp4c	CW-4	CW-6	sbvp4	sbvp4g	sbvp6	sbvp6g	sbvp8	sbvp8g
N	839	161	41	193	102	32	32	22	14
fcount	23546	3000	1320	20067	6669	5397	3351	5058	3084

Table 5.5: Comparisons for (5.3), **AbsTol**=**RelTol**=10^{-9}

model problems used for the test runs include the examples (3.1), (5.1), (5.2) and (5.3), and the following systems:

$$\text{Problem (5.1) with parameters } \alpha = 360 \text{ and } k = 324. \tag{5.4}$$

A plot of the solution of this problem is given in Figure 5.10.

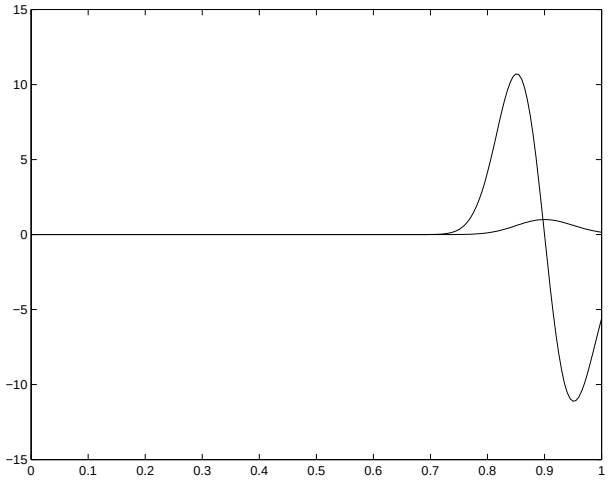

Figure 5.10: Solution of (5.4)

$$\text{Problem (5.1) with parameters } \alpha = 40 \text{ and } k = 36, \tag{5.5}$$

see Figure 5.11 for a plot of the solution.

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ -100t^2 & 2 \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ 1000t^2 + 10\cos(10t) - 10 \end{pmatrix}, \tag{5.6a}$$

$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} 10 - \sin(10) \\ 0 \end{pmatrix}, \tag{5.6b}$$

with exact solution

$$\mathbf{z}(t) = (10t - \sin(10t), 10t - 10t\cos(10t))^T.$$

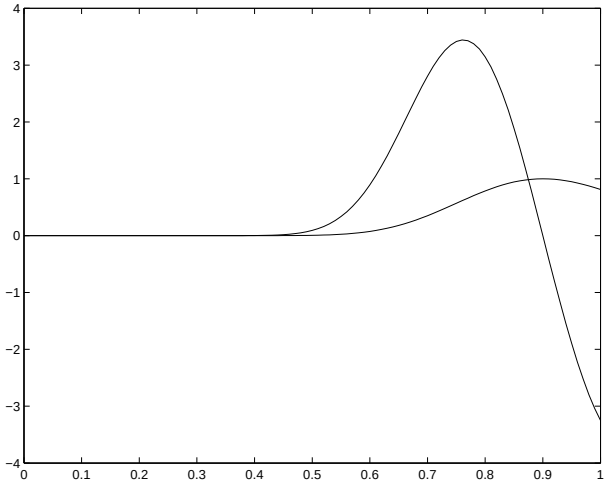

Figure 5.11: Solution of (5.5)

The plot of the solution is given in Figure 5.12.

For the problems specified above, the convergence order of collocation at Gaussian points is $2p$ at the meshpoints τ_k.

Now two examples follow, where collocation at Gaussian points indeed shows an order reduction:

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ -2p^2 & -3p \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ (6p^2+5p+1)t^p \end{pmatrix}, \tag{5.7a}$$

$$\mathbf{z}(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{5.7b}$$

The exact solution is

$$\mathbf{z}(t) = (t^{p+1}, (p+1)t^{p+1})^T.$$

These polynomial solution components are plotted in Figure 5.13.

It was shown in [19] that for a collocation method with p points, the convergence order for (5.7) is no higher than $p+1$ in general. This can be observed for Gaussian points and p even. For p odd, however, we observe a convergence order of $p+2$. The same applies to

$$z'(t) = -\frac{p}{t}z(t) + (2p+1)t^p, \tag{5.8a}$$

$$z(0) = 0, \tag{5.8b}$$

with exact solution

$$z(t) = t^{p+1},$$

see [13]. For both (5.7) and (5.8) we chose $p = 4$, which implies that for collocation of higher order, the numerical solution coincides with the exact (polynomial) solution up to round-off errors.

The next two examples read:

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ -3tz_1^5(t) + tz_1^3(t) \end{pmatrix}, \tag{5.9a}$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}, \tag{5.9b}$$

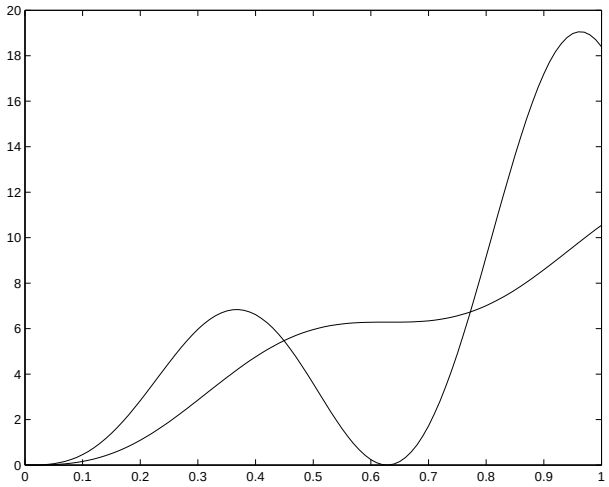

Figure 5.12: Solution of (5.6)

with exact solution

$$\mathbf{z}(t) = (1/\sqrt{1+t^2}, -t^2/\sqrt{(1+t^2)^3})^T,$$

see Figure 5.14 for a plot of this solution. Here, the convergence order of collocation at Gaussian points is equal to $2p$ at the meshpoints.

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ t\frac{-2(t^2+2)-8}{(t^2+2)^2}z_1^2(t) + \frac{8t^3}{(t^2+2)^2}z_1^3(t) \end{pmatrix}, \quad (5.10a)$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} \frac{1}{\ln(3)} \\ 0 \end{pmatrix}, \quad (5.10b)$$

with exact solution

$$\mathbf{z}(t) = (1/\ln(t^2+2), -2t^2/(\ln^2(t^2+2)(t^2+2)))^T,$$

which is shown in Figure 5.15.

The convergence behavior for this problem is analogous as for example (5.7).

The last three problems are given in [16]. However, the boundary conditions were modified in order to obtain a well-posed problem in the sense of [12].

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & \frac{1}{2} \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ \frac{z_1^3(t)}{2} \end{pmatrix}, \quad (5.11a)$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\mathbf{z}(16) = \begin{pmatrix} \frac{1}{6} \\ 0 \end{pmatrix}. \quad (5.11b)$$

Figure 5.16 gives a plot of the solution of example (5.11).

Note that the presence of the positive fractional eigenvalue $1/2$ for the matrix $M(0)$ seems to influence the convergence order of the applied collocation schemes. We observe a convergence order of at most $p + 1/2$ for any choice of collocation points.

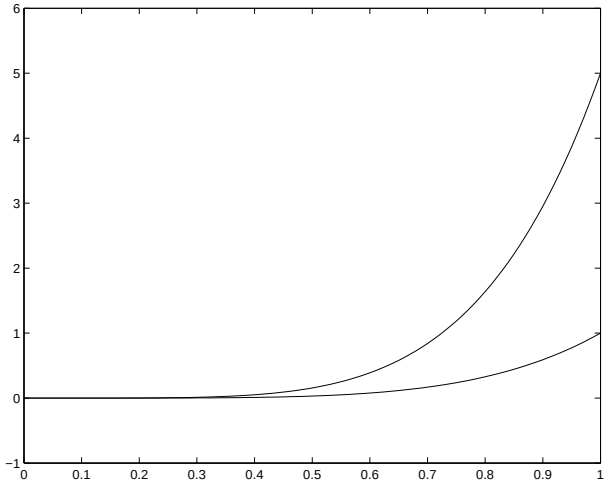

Figure 5.13: Solution of (5.7)

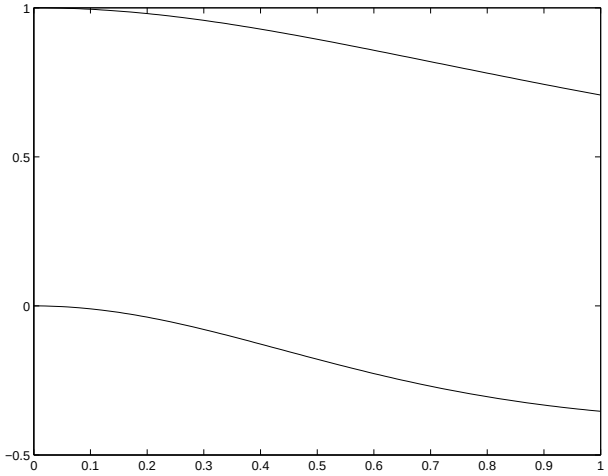

Figure 5.14: Solution of (5.11)

$$\mathbf{z}'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}\mathbf{z}(t) + \begin{pmatrix} 0 \\ \frac{tz_1(t)}{\varepsilon(z_1(t)+0.1)} \end{pmatrix}, \tag{5.12a}$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\mathbf{z}(0) + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\mathbf{z}(1) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \tag{5.12b}$$

where $\varepsilon = 10^{-1}$.

Figure 5.17 shows a plot of this solution.

$$\text{Problem (5.12) with } \varepsilon = 10^{-2}. \tag{5.13}$$

Figure 5.18 suggests that the latter solution will be probably more difficult to compute. The last two problems do not show order reductions for collocation at Gaussian points.

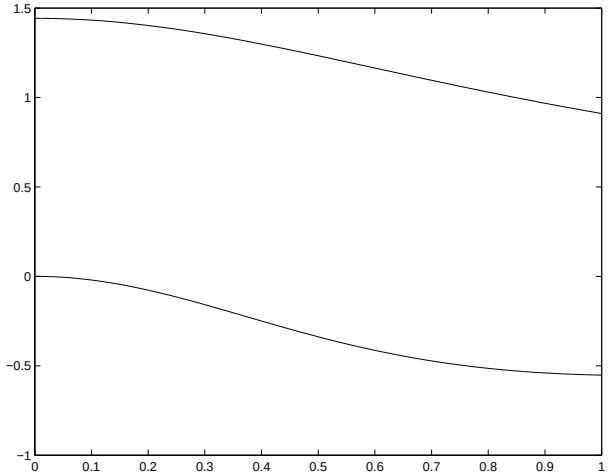

Figure 5.15: Solution of (5.11)

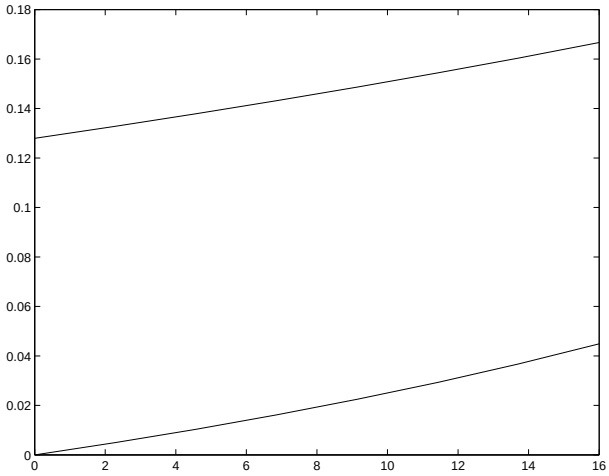

Figure 5.16: Solution of (5.11)

The results for the choice **AbsTol**=**RelTol**=10^{-5} are presented in Tables 5.6–5.10.

In the tables, horizontal lines separate the results for linear problems, nonlinear problems with a known solution, and nonlinear problems with a reference solution computed with very strict tolerance.

Some table entries are left blank to indicate that the corresponding computation was not successful. This means in general that the tolerances could not be satisfied.

The results for example (5.4) do not quite fit into the picture, as the behavior of the numerical solution process does not appear very regular. The reason seems to be the unsmoothness of the solution. In fact, the error estimate computed by `sbvp` seems to enter the asymptotic regime only for a step-size where the collocation solution is already very accurate. Therefore, we do not draw any conclusions from the results for this problem.

For many of the other problems (more precisely, examples (5.8), (5.7), (3.1), (5.9), (5.10), (5.3) and (5.11)), the tolerances are already satisfied on the (quite coarse)

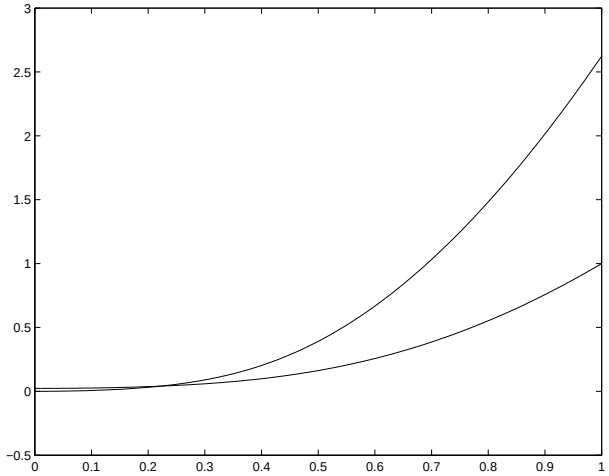

Figure 5.17: Solution of (5.12)

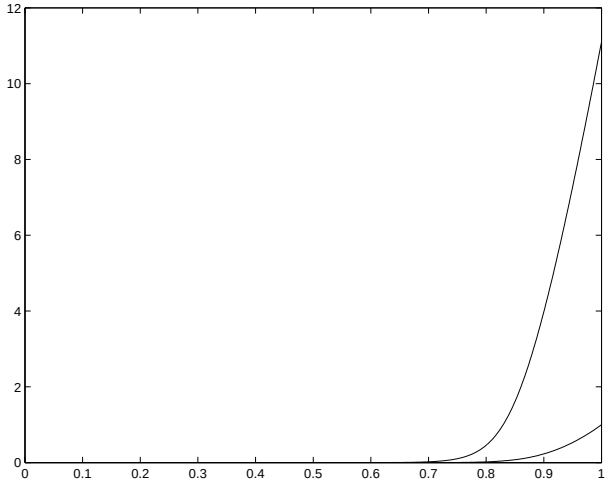

Figure 5.18: Solution of (5.13)

equidistant initial grid. This observation holds for both COLSYS and `sbvp`. In particular, the same initial grid yields the solution with the desired accuracy for both equidistant and Gaussian points. Thus the latter show no advantage, even for the majority of problems where no order reduction is observed. In most cases, the accuracy obtained at the initial grid is far better than the (modest) tolerances 10^{-5}.

Overall, we observe that the error estimates work quite reliably for all three codes (and all tolerances). It is interesting to note that `sbvp` slightly overestimates the error in the vast majority of test runs and is therefore very reliable, which seems to justify a small amount of extra computational work in some cases.

Although the CPU-time is not an easily accessible and reliable measure of the efficiency of the code (especially, the Fortran 90 code COLNEW and the MATLAB codes `bvp4c` and `sbvp` cannot be compared in that respect), we note that the required CPU-time and the count of function evaluations give a quite consistent impression of the performance

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	102			631	86	82	187
(5.1)	116	41	21	40	40	20	14
(5.2)	209	81	41	139	85	39	31
(5.5)	67	41	11	43	43	11	7
(5.6)	83	81	11	85	82	25	10
(5.7)	27	11	11	18	18	7	7
(5.8)		11	11	18	18	7	7
(3.1)	11	11	11	18	18	7	7
(5.9)	19	11	11	18	18	7	7
(5.10)	24	11	11	18	18	7	7
(5.3)	10	11	11	18	18	7	7
(5.11)	5	11	11	18	18	7	7
(5.12)	22	11	11	18	18	7	7
(5.13)		13	11	19	19	16	7

Table 5.6: Number N of points in the final mesh, TOL=10^{-5}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	4660			4205	530	1451	1780
(5.1)	3622	340	390	283	413	178	136
(5.2)	10353	660	450	1238	613	311	360
(5.5)	2597	300	90	518	518	115	87
(5.6)	3081	620	90	783	628	388	150
(5.7)	726	60	90	88	88	45	45
(5.8)		60	90	88	88	45	45
(3.1)	335	180	270	683	683	351	351
(5.9)	477	240	300	989	904	507	465
(5.10)	759	160	180	768	683	351	351
(5.3)	228	160	240	615	615	315	315
(5.11)	122	180	270	683	683	351	351
(5.12)	761	280	420	972	972	537	501
(5.13)		476	390	1795	1885	2313	1089

Table 5.7: Number fcount of function evaluations, TOL=10^{-5}

of `sbvp` and `bvp4c`, even in spite of the differences in the accounting mentioned earlier[1]

Tables 5.11–5.15 give the analogous results for **AbsTol**=**RelTol**=10^{-7}. For these stricter tolerances, it turns out that our choice of the initial mesh may be inferior to the strategies adopted by COLNEW and `bvp4c`, in the respect that too many mesh points are used. Since many "easy" problems are quite accurately solved on coarse grids, our strategy is unnecessarily expensive. However, the computational effort is still quite low.

Finally, in Tables 5.16–5.21 we provide the respective results for the choice of tolerances **AbsTol**=**RelTol**=10^{-9}. For these strict tolerances, we also include the results obtained by `sbvp` with the default chioce $p = 8$, with both equidistant (`sbvp8`) and Gaussian (`sbvp8g`) points.

Naturally, `bvp4c` cannot compete with the other codes for stricter tolerances, since the order of the method is fixed at 4. It turns out that the choice $p = 8$ is quite

[1] Examples (5.11) and (5.12) show that the measurement of the CPU-time may vary significantly.

Example	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)			2.26e−10	2.02e−05	7.59e−10	7.31e−13
(5.1)	4.42e−05	1.26e−07	1.66e−05	6.52e−06	8.68e−06	4.25e−06
(5.2)	1.12e−05	5.76e−06	5.76e−06	4.05e−06	4.71e−06	2.58e−05
(5.5)	4.61e−07	1.62e−06	3.76e−06	1.90e−06	6.82e−06	1.12e−05
(5.6)	1.17e−06	1.99e−05	5.79e−05	2.82e−05	1.33e−05	5.79e−05
(5.7)	4.05e−07	1.72e−10	1.46e−05	2.05e−06	1.71e−06	1.57e−07
(5.8)	5.89e−08	2.03e−11	6.54e−08	2.17e−09	1.17e−15	1.72e−16
(3.1)	2.92e−09	6.93e−13	2.07e−08	1.98e−10	1.50e−09	2.37e−11
(5.9)	3.61e−08	2.46e−10	4.92e−07	3.01e−09	1.12e−07	9.02e−10
(5.10)	1.29e−07	1.06e−10	4.61e−07	9.19e−09	9.93e−08	3.21e−09
(5.3)	1.71e−09	4.01e−13	1.15e−08	1.16e−10	8.29e−10	1.36e−11
(5.11)	4.14e−10	4.97e−12	5.21e−11	9.72e−13	5.73e−13	9.70e−15
(5.12)	4.05e−07	8.32e−09	1.98e−07	2.01e−08	1.21e−07	1.36e−08
(5.13)	5.87e−06	1.07e−05	8.32e−06	1.57e−06	1.30e−06	8.56e−06

Table 5.8: Estimated errors of the solutions, TOL=10^{-5}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	5.27e−05			1.23e−11	9.88e−06	1.24e−10	1.93e−12
(5.1)	5.77e−06	3.18e−06	7.34e−07	2.98e−06	5.12e−06	3.37e−06	4.30e−07
(5.2)	2.95e−05	8.34e−06	8.38e−06	1.90e−07	4.34e−08	3.29e−07	1.36e−07
(5.5)	7.19e−06	8.21e−07	2.04e−06	2.15e−07	1.92e−07	3.14e−06	2.45e−06
(5.6)	3.65e−04	1.69e−06	3.55e−05	3.22e−05	1.90e−05	1.07e−05	1.90e−04
(5.7)	3.44e−06	4.11e−07	3.25e−10	8.17e−07	2.84e−07	9.59e−07	2.13e−08
(5.8)		4.65e−08	2.96e−11	4.22e−09	3.55e−10	2.22e−16	5.42e−20
(3.1)	4.63e−07	3.60e−08	3.58e−08	1.66e−10	1.32e−11	5.32e−11	5.39e−13
(5.9)	9.03e−07	6.15e−08	5.13e−08	3.77e−08	7.02e−10	2.27e−08	5.28e−10
(5.10)	4.83e−07	1.20e−07	1.75e−08	1.30e−08	6.21e−10	9.91e−09	4.98e−11
(5.3)	5.95e−07	6.79e−14	5.27e−16	8.61e−11	4.09e−11	4.54e−11	4.23e−11
(5.11)	2.31e−06	5.07e−11	1.19e−15	9.84e−10	9.86e−10	1.54e−10	1.54e−10
(5.12)	9.41e−07	4.63e−07	7.11e−09	9.91e−09	1.89e−09	2.39e−08	1.94e−09
(5.13)		1.36e−05	2.43e−08	2.81e−06	4.67e−06	2.77e−06	2.19e−06

Table 5.9: Maximal errors of the solutions, TOL=10^{-5}

advantageous for `sbvp`.Moreover, for Gaussian points this results in a convergence order of 16. Note that such a high order is not implemented in COLNEW.

Finally, the error estimate in COLNEW may be quite unreliable, cf. examples (5.5) and (5.6) for all tolerances.

Example	bvp4c	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	9.98	9.77	1.40	3.49	4.22
(5.1)	7.82	1.06	1.32	0.91	0.80
(5.2)	22.72	2.75	1.62	1.11	1.22
(5.5)	5.60	1.61	1.58	0.79	0.71
(5.6)	6.68	2.06	1.73	1.42	0.87
(5.7)	1.68	0.62	0.58	0.54	0.53
(5.8)		0.55	0.52	0.51	0.48
(3.1)	0.84	1.25	1.24	0.95	0.92
(5.9)	1.16	1.81	1.71	1.26	1.52
(5.10)	1.80	1.66	1.45	1.12	1.07
(5.3)	0.57	1.33	1.29	0.95	0.99
(5.11)	0.35	1.24	1.25	0.94	0.95
(5.12)	1.68	1.53	1.51	1.12	1.11
(5.13)		2.50	2.54	3.35	1.94

Table 5.10: CPU-time for the computation of the solutions, TOL=10^{-5}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	317			169	139	71	71
(5.1)	413	81	25	151	94	32	29
(5.2)	285	195	41	376	343	97	47
(5.5)	264	81	21	88	57	22	15
(5.6)	309	161	41	487	321	43	49
(5.7)	107	21	11	57	57	15	15
(5.8)		11	11	57	57	15	15
(3.1)	43	11	11	57	57	15	15
(5.9)	65	11	11	57	57	15	15
(5.10)	80	21	11	57	57	15	15
(5.3)	30	11	11	57	57	15	15
(5.11)	9	11	11	57	57	15	15
(5.12)	93	21	11	57	57	15	15
(5.13)		41	21	62	63	18	16

Table 5.11: Number N of points in the final mesh, TOL=10^{-7}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	13271			1678	1428	605	605
(5.1)	11513	660	426	1528	1058	318	395
(5.2)	4996	2464	810	3408	2278	1508	584
(5.5)	6139	620	210	1008	283	346	101
(5.6)	11601	1260	450	5143	2283	591	773
(5.7)	2536	140	90	283	283	101	101
(5.8)		60	90	283	283	101	101
(3.1)	1001	180	270	2467	2467	899	899
(5.9)	1405	260	330	3251	2971	1081	1081
(5.10)	2297	320	180	2747	2467	899	899
(5.3)	922	160	240	2243	2243	815	815
(5.11)	2814	200	300	2467	2467	899	899
(5.12)	2424	680	480	2971	2971	1333	1249
(5.13)		2000	1050	5732	5769	2385	2253

Table 5.12: Number fcount of function evaluations, TOL=10^{-7}

Example	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)			5.16e−08	6.55e−08	4.51e−07	3.14e−08
(5.1)	1.31e−07	5.97e−08	3.22e−08	3.15e−08	1.13e−07	4.36e−08
(5.2)	7.64e−08	1.21e−07	1.19e−07	2.82e−09	1.52e−08	4.81e−07
(5.5)	1.21e−08	1.21e−08	2.70e−08	7.05e−08	9.78e−08	1.76e−07
(5.6)	3.71e−08	2.88e−09	3.51e−07	1.14e−07	4.29e−07	2.90e−08
(5.7)	1.26e−08	1.72e−10	1.08e−07	6.09e−09	1.07e−08	4.85e−10
(5.8)	5.89e−08	2.03e−11	5.27e−10	5.65e−12	2.95e−16	3.10e−16
(3.1)	2.92e−09	6.93e−13	1.75e−10	5.17e−13	9.37e−12	6.49e−14
(5.9)	3.60e−08	2.72e−11	4.07e−09	7.79e−12	6.45e−10	3.03e−12
(5.10)	4.16e−09	1.06e−10	3.94e−09	2.41e−11	5.98e−10	1.02e−11
(5.3)	1.71e−09	4.01e−13	9.74e−11	3.03e−13	5.14e−12	3.73e−14
(5.11)	4.42e−11	3.04e−15	4.34e−13	4.38e−15	3.40e−15	1.38e−16
(5.12)	1.52e−08	7.14e−10	1.68e−09	5.44e−11	8.95e−10	5.65e−11
(5.13)	2.18e−08	7.42e−09	6.66e−08	3.66e−09	8.62e−08	1.99e−08

Table 5.13: Estimated errors of the solutions, TOL=10^{-7}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	5.04e−07			1.41e−08	3.16e−08	2.65e−07	3.19e−09
(5.1)	4.30e−08	9.66e−08	1.76e−07	1.31e−09	2.76e−09	2.32e−08	8.14e−09
(5.2)	9.06e−06	3.45e−08	1.23e−07	1.40e−09	5.62e−11	3.41e−10	1.98e−09
(5.5)	5.96e−08	7.19e−07	7.15e−07	1.16e−09	8.26e−09	2.27e−08	1.27e−08
(5.6)	1.95e−06	1.03e−06	1.01e−06	5.16e−08	5.34e−08	2.54e−07	2.12e−08
(5.7)	2.91e−08	1.18e−08	3.25e−10	2.98e−09	5.19e−10	7.07e−10	2.77e−11
(5.8)		4.65e−08	2.96e−11	1.08e−11	9.16e−13	2.22e−16	1.11e−16
(3.1)	1.98e−09	3.60e−08	3.58e−08	4.83e−13	3.44e−14	1.15e−13	2.05e−15
(5.9)	1.02e−08	6.15e−08	5.13e−08	8.99e−11	1.19e−12	4.40e−11	7.24e−13
(5.10)	1.94e−09	1.78e−08	1.75e−08	2.98e−11	1.67e−12	1.98e−11	2.50e−13
(5.3)	4.02e−09	6.79e−14	5.27e−16	1.12e−10	1.12e−10	4.50e−11	4.50e−11
(5.11)	3.86e−08	5.07e−11	1.13e−15	2.68e−09	2.68e−09	7.71e−10	7.71e−10
(5.12)	2.77e−09	1.50e−08	1.61e−09	4.55e−09	4.55e−09	1.81e−09	1.82e−09
(5.13)		4.80e−09	1.22e−10	4.69e−06	4.37e−06	2.22e−06	3.30e−06

Table 5.14: Maximal errors of the solutions, TOL=10^{-7}

Example	bvp4c	sbvp4	sbvp4g	sbvp6	sbvp6g
(5.4)	27.48	3.78	3.28	1.66	1.63
(5.1)	24.49	3.49	2.54	1.17	1.33
(5.2)	11.47	7.14	4.91	3.39	1.61
(5.5)	13.04	2.46	0.96	1.24	0.68
(5.6)	23.85	11.55	5.22	1.72	2.09
(5.7)	5.55	1.00	0.97	0.64	0.65
(5.8)		0.76	0.73	0.60	0.56
(3.1)	2.61	3.09	3.11	1.54	1.58
(5.9)	3.07	4.07	3.86	1.98	1.85
(5.10)	5.14	3.93	3.50	1.80	1.74
(5.3)	2.09	3.33	3.35	1.57	1.66
(5.11)	0.55	3.09	3.10	1.52	1.56
(5.12)	5.07	3.57	3.60	1.88	1.91
(5.13)		6.15	6.22	3.15	3.02

Table 5.15: CPU-time for the computation of the solutions, TOL=10^{-7}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g	sbvp8	sbvp8g
(5.4)	386			481	247	127	88	66	66
(5.1)	1500	171	81	466	227	91	52	31	25
(5.2)	2718	641	133	1324	541	154	109	55	37
(5.5)	496	161	41	232	105	49	32	22	14
(5.6)	1146	321	81	1396	773	133	77	31	14
(5.7)	397	41	11	157	102	32	32	14	14
(5.8)		41	11	102	102	32	32	14	14
(3.1)	139	21	11	102	102	32	32	14	14
(5.9)	234	41	11	102	102	32	32	14	14
(5.10)	338	41	11	102	102	32	32	14	14
(5.3)	118	21	11	102	102	32	32	14	14
(5.11)	25	11	11	102	102	32	32	14	14
(5.12)	320			102	102	32	32	14	14
(5.13)		75	41	211	112	33	33	17	14

Table 5.16: Number N of points in the final mesh, TOL=10^{-9}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g	sbvp8	sbvp8g
(5.4)	8839			4508	2558	2082	1228	718	718
(5.1)	41274	2320	1170	4383	2203	1270	808	390	471
(5.2)	102362	5140	1836	11533	3883	2005	1228	921	606
(5.5)	9756	1260	450	2428	1028	773	220	426	120
(5.6)	43144	2540	930	9809	5333	2152	1018	570	120
(5.7)	9474	300	90	1808	508	220	220	120	120
(5.8)		300	90	508	508	220	220	120	120
(3.1)	3023	360	300	4952	4447	1987	1987	1212	1212
(5.9)	6784	740	330	6265	5760	2576	2576	1420	1420
(5.10)	7953	640	180	4952	4447	1987	1987	1095	1095
(5.3)	3082	340	270	4043	4043	1801	1801	991	1108
(5.11)	5819	200	300	4447	4447	1987	1987	1095	1095
(5.12)	7640			6265	5760	13479	2576	1628	1524
(5.13)		2396	1680	19469	10523	15008	4681	3041	2876

Table 5.17: Number fcount of function evaluations, TOL=10^{-9}

Example	colnew4	colnew6	sbvp4	sbvp4g	sbvp6	sbvp6g	sbvp8	sbvp8g
(5.4)			6.88e−10	9.89e−10	4.74e−11	1.59e−11	1.15e−09	3.18e−11
(5.1)	7.40e−10	1.41e−11	3.58e−10	3.44e−10	1.89e−10	4.33e−10	5.80e−10	3.93e−10
(5.2)	1.17e−08	4.09e−11	9.03e−10	3.26e−10	4.25e−10	4.84e−10	2.71e−10	7.36e−09
(5.5)	3.53e−10	1.21e−10	5.24e−10	7.52e−10	1.12e−10	7.67e−10	9.93e−11	8.26e−10
(5.6)	1.16e−09	2.35e−11	4.65e−09	9.91e−10	9.27e−10	2.90e−09	1.10e−09	1.05e−08
(5.7)	3.96e−10	1.72e−10	2.88e−10	3.28e−10	8.34e−11	2.02e−12	6.32e−13	1.60e−14
(5.8)	5.76e−11	2.03e−11	4.93e−11	2.97e−13	5.97e−16	4.75e−16	1.41e−15	4.06e−16
(3.1)	8.90e−11	6.98e−13	1.66e−11	2.69e−14	8.06e−14	1.77e−15	3.55e−15	3.33e−16
(5.9)	4.13e−11	2.72e−11	3.83e−10	4.08e−13	5.37e−12	1.16e−14	7.26e−13	2.21e−15
(5.10)	1.28e−10	1.06e−10	3.73e−10	1.26e−12	5.08e−12	4.11e−14	7.40e−13	8.34e−15
(5.3)	5.22e−11	4.02e−13	9.20e−11	1.60e−14	4.44e−14	9.99e−16	9.99e−16	4.44e−16
(5.11)	4.42e−11	3.04e−15	4.08e−14	3.05e−16	8.32e−17	1.38e−16	1.11e−16	1.11e−16
(5.12)			1.58e−10	2.86e−12	7.97e−12	2.22e−13	4.62e−12	1.11e−13
(5.13)	1.00e−09	3.92e−11	4.95e−10	2.05e−10	9.06e−10	9.05e−11	1.35e−09	3.28e−10

Table 5.18: Estimated errors of the solutions, TOL=10^{-9}

Example	bvp4c	colnew4	colnew6	sbvp4	sbvp4g
(5.4)	8.90e−08			4.90e−11	2.10e−10
(5.1)	2.25e−10	9.12e−10	1.12e−11	3.56e−12	2.03e−11
(5.2)	3.69e−10	3.10e−10	3.25e−11	3.14e−12	6.37e−12
(5.5)	1.13e−09	7.20e−07	7.20e−07	7.36e−12	3.82e−11
(5.6)	1.00e−08	1.01e−06	1.01e−06	2.43e−10	4.17e−10
(5.7)	1.30e−10	3.69e−10	3.25e−10	2.22e−11	2.52e−11
(5.8)		3.13e−11	2.96e−11	5.70e−13	4.80e−14
(3.1)	1.29e−11	3.58e−08	3.58e−08	2.58e−14	1.77e−15
(5.9)	1.35e−11	5.13e−08	5.13e−08	4.65e−12	5.56e−14
(5.10)	5.12e−12	1.74e−08	1.75e−08	1.52e−12	8.81e−14
(5.3)	1.30e−11	1.22e−15	1.55e−15	2.46e−11	2.46e−11
(5.11)	8.35e−10	5.07e−11	1.13e−15	2.60e−09	2.60e−09
(5.12)	1.69e−11			3.05e−09	3.05e−09
(5.13)		9.04e−11	2.13e−14	5.95e−06	5.85e−06

Table 5.19: Maximal errors of the solutions (1), TOL=10^{-9}

Example	sbvp6	sbvp6g	sbvp8	sbvp8g
(5.4)	2.09e−11	2.17e−12	7.66e−10	2.49e−12
(5.1)	1.10e−11	3.17e−11	1.23e−10	2.22e−11
(5.2)	3.76e−11	1.74e−12	1.10e−11	1.16e−11
(5.5)	2.60e−11	2.72e−11	1.52e−11	2.69e−11
(5.6)	1.93e−10	1.79e−09	6.68e−10	1.44e−08
(5.7)	6.37e−13	7.46e−14	4.48e−13	3.44e−15
(5.8)	3.33e−16	1.11e−16	8.88e−16	1.11e−16
(3.1)	3.67e−16	2.22e−16	5.55e−16	3.33e−16
(5.9)	1.48e−13	2.22e−15	5.24e−14	4.44e−16
(5.10)	6.15e−14	1.33e−15	2.53e−14	6.66e−16
(5.3)	6.01e−11	6.01e−11	4.86e−11	4.86e−11
(5.11)	1.36e−09	1.36e−09	4.85e−10	4.85e−10
(5.12)	2.60e−09	2.60e−09	2.58e−09	2.58e−09
(5.13)	4.26e−06	5.02e−06	4.15e−06	4.72e−06

Table 5.20: Maximal errors of the solutions (2), TOL=10^{-9}

Example	bvp4c	sbvp4	sbvp4g	sbvp6	sbvp6g	sbvp8	sbvp8g
(5.4)	18.01	10.04	5.55	4.72	2.94	2.01	1.93
(5.1)	144.49	9.71	4.85	3.07	2.13	1.39	1.55
(5.2)	322.82	31.08	8.48	4.38	2.82	2.38	1.81
(5.5)	21.5	5.32	2.46	2.08	0.82	1.52	0.68
(5.6)	82.51	28.08	13.23	4.96	2.56	1.78	0.68
(5.7)	28.02	4.00	1.41	0.91	0.87	0.73	0.80
(5.8)		1.02	1.01	0.73	0.70	0.60	0.68
(3.1)	6.84	5.57	5.23	2.75	2.73	2.00	2.89
(5.9)	13.43	7.28	6.93	3.51	3.49	2.52	2.39
(5.10)	16.42	6.34	6.13	3.26	3.01	2.16	2.19
(5.3)	6.64	5.67	5.63	2.90	2.95	1.89	2.01
(5.11)	1.20	5.21	5.26	2.69	2.71	1.83	1.82
(5.12)	15.57	6.91	6.66	3.45	3.40	2.29	3.38
(5.13)		20.32	11.03	5.71	5.50	4.06	5.72

Table 5.21: CPU-time for the computation of the solutions, TOL=10^{-9}

Bibliography

[1] U. Ascher, J. Christiansen, and R. Russell, *A collocation solver for mixed order systems of boundary values problems*, Math. Comp., 33 (1978), pp. 659–679.

[2] ——, *Collocation software for boundary value ODEs*, ACM Transactions on Mathematical Software, 7 (1981), pp. 209–222.

[3] U. Ascher, R. Mattheij, and R. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[4] W. Auzinger, O. Koch, W. Polster, and E. Weinmüller, *Ein Algorithmus zur Gittersteuerung bei Kollokationsverfahren für singuläre Randwertprobleme*, Techn. Rep. ANUM Preprint Nr. 21/01, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology, Austria, 2001. Available at `http://www.math.tuwien.ac.at/preprints.htm`.

[5] W. Auzinger, O. Koch, and E. Weinmüller, *Efficient collocation schemes for singular boundary value problems.* To appear in Numer. Algorithms. Also available as ANUM Preprint Nr. 5/01 at `http://www.math.tuwien.ac.at/preprints.htm`.

[6] W. Auzinger, P. Kofler, and E. Weinmüller, *Steuerungsmaßnahmen bei der numerischen Lösung singulärer Anfangswertaufgaben*, Techn. Rep. Nr. 124/98, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology, 1998. Available at `http://www.math.tuwien.ac.at/~ewa/abstrc90.html#steuer`.

[7] P. Deuflhard and A. Hohmann, *Numerical Ananlysis: A First Course in Scientific Computation*, vol. 1, de Gruyter, Berlin, 1995.

[8] R. Frank, *Schätzungen des globalen Diskretisierungsfehlers bei Runge-Kutta-Methoden*, ISNM, 27 (1975), pp. 45–70.

[9] ——, *The method of Iterated Defect Correction and its application to two-point boundary value problems, Part I*, Numer. Math., 25 (1976), pp. 409–419.

[10] R. Frank and C. Überhuber, *Iterated Defect Correction for differential equations, Part I: Theoretical results*, Computing, 20 (1978), pp. 207–228.

[11] M. Gräff and E. Weinmüller, *Schätzungen des lokalen Diskretisierungsfehlers bei singulären Anfangswertproblemen*, Techn. Rep. Nr. 66/86, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technolgy, Austria, 1986.

[12] F. d. Hoog and R. Weiss, *Difference methods for boundary value problems with a singularity of the first kind*, SIAM J. Numer. Anal., 13 (1976), pp. 775–813.

[13] ——, *Collocation methods for singular boundary value problems*, SIAM J. Numer. Anal., 15 (1978), pp. 198–217.

[14] ——, *The application of Runge-Kutta schemes to singular initial value problems*, Math. Comp., 44 (1985), pp. 93–103.

[15] O. Koch and E. Weinmüller, *Iterated Defect Correction for the solution of singular initial value problems*, SIAM J. Numer. Anal., 38 (2001), pp. 1784–1799.

[16] L. Shampine, J. Kierzenka, and M. Reichelt, *Solving Boundary Value Problems for Ordinary Differential Equations in* MATLAB *with* `bvp4c`, 2000. Available at `ftp://ftp.mathworks.com/pub/doc/papers/bvp/`.

[17] H. J. Stetter, *The defect correction principle and discretization methods*, Numer. Math., 29 (1978), pp. 425–443.

[18] E. Weinmüller, *Collocation for singular boundary value problems of second order*, SIAM J. Numer. Anal., 23 (1986), pp. 1062–1095.

[19] ——, *Stability of singular boundary value problems and their discretization by finite differences*, SIAM J. Numer. Anal., 26 (1989), pp. 180–213.

[20] P. Zadunaisky, *On the estimation of errors propagated in the numerical integration of ODEs*, Numer. Math., 27 (1976), pp. 21–39.