# A Collocation Code for Singular Boundary Value Problems in Ordinary Differential Equations *

Winfried Auzinger (w.auzinger@tuwien.ac.at), Günter Kneisl (eomer@gmx.at), Othmar Koch (othmar@fsmat.at) and Ewa Weinmüller (e.weinmueller@tuwien.ac.at)
*Vienna University of Technology, Austria*

**Abstract.** We present a MATLAB package for boundary value problems in ordinary differential equations. Our aim is the efficient numerical solution of systems of ODEs with a singularity of the first kind, but the solver can also be used for regular problems. The basic solution is computed using collocation methods and a new, efficient estimate of the global error is used for adaptive mesh selection. Here, we analyze some of the numerical aspects relevant for the implementation, describe measures to increase the efficiency of the code and compare its performance with the performance of established standard codes for boundary value problems.

**Keywords:** Collocation, singular boundary value problems

**AMS Subject Classification:** 65L05

## 1. Introduction

In this paper, we present a MATLAB code for the solution of the following class of nonlinear singular boundary value problems of the first order:

$$z'(t) = \underbrace{\frac{M(t)}{t} z(t) + f(t, z(t))}_{=:F(t,z(t))}, \quad t \in (0, 1], \tag{1a}$$

$$B_{a2} z(0) + B_{b2} z(1) = \beta_2, \tag{1b}$$

$$z \in C[0, 1], \tag{1c}$$

where $z$ and $f$ are vector-valued functions of dimension $n$, $M$ is a smooth $n \times n$ matrix, and $B_{a2},\ B_{b2} \in \mathbb{R}^{r \times n},\ \beta_2 \in \mathbb{R}^r,\ r \leq n$ are constant. Conditions (1b) may be nonlinear. In this text, however, we will restrict ourselves to linear boundary conditions, the case which is most relevant in applications.

The analytical properties of (1) have been discussed in full detail in [9]. It turns out that the requirement (1c) is equivalent to $n - r$

* This project was supported by the Austrian Research Fund (FWF) grant P-12507-MAT.

linearly independent conditions which together with (1b) are necessary for the problem to be well-posed. Therefore, we usually write (1) in its equivalent form (1a) plus the $n$ linearly independent conditions

$$B_a z(0) + B_b z(1) = \beta. \tag{1d}$$

The design of a reliable code taking into account the specific difficulties caused by the singularity is strongly motivated by applications from physics, chemistry, and mechanics (buckling of spherical shells), as well as research activities in related areas.

We chose collocation as the basic method for the numerical solution of (1). This decision was motivated by the advantageous convergence properties of this class of methods. As will be explained below, the global error of a collocation solution does not suffer from an order reduction. Such order reductions are typically observed for many other direct discretization methods or the well-known acceleration techniques, see for example [8], [16]. On the other hand, shooting methods can only be applied to a restricted class of singular problems, see [12]. For this subclass, however, an efficient numerical solution is possible for a suitable choice of the underlying solver for the associated initial value problems, see [11].

Our reason for choosing the *global* error estimate described in §1.2 instead of monitoring the local error is the non-smoothness of the latter near the singular point and the order reductions it suffers from. For an extensive discussion of this phenomenon and numerical experiments see for example [6]. Our a posteriori error estimate is a modification of ideas from [7], [14] and [17] and was first introduced in [5].

### 1.1. Collocation Methods

The basic method in the numerical solution algorithm is polynomial collocation with degree $\le m$. This means that on a mesh $\Delta = (t_0, \ldots, t_N)$ we approximate the analytical solution by a collocating function $p(t) := p_i(t),\ t \in [t_i, t_{i+1}],\ i = 0, \ldots, N-1$, where $p_i$ is a polynomial of degree $\le m$. The function $p(t)$ is uniquely specified by the conditions

$$p_i'(t_{i,j}) = F(t_{i,j}, p_i(t_{i,j})), \quad i = 0, \ldots, N-1,\ j = 1, \ldots, m, \tag{2a}$$

$$p_i(t_i) = p_{i-1}(t_i), \quad i = 1, \ldots, N-1, \tag{2b}$$

$$B_a p_0(0) + B_b p_{N-1}(1) = \beta. \tag{2c}$$

Here, the *collocation points* are given by

$$t_{i,j} := t_i + \rho_j (t_{i+1} - t_i), \quad 0 < \rho_1 < \cdots < \rho_m < 1.$$

In this setting, a convergence order $O(h^m)$ can be guaranteed for regular problems with appropriately smooth data. Moreover, for special choices of the collocation nodes (for example the *Gaussian points*), (super-)convergence orders as high as $O(h^{2m})$ hold at the mesh points $t_i$, cf. e.g. [1]. For singular problems it was shown in [10] that this superconvergence breaks down in general. The highest attainable order in this case is[1] $O(h^{m+1})$. For a restricted class of singular problems it was shown that this convergence order indeed holds uniformly for $t \in [0, 1]$. Results from [15] and experimental experience indicate that the situation is similar for general nonlinear singular problems. For this reason we decided to use collocation at an even number of collocation points equidistantly spaced in the interior of each interval as the default for our code. For this choice, the convergence order is $O(h^m)$, which also enables us to provide an asymptotically correct estimate of the global error, see §1.2. However, collocation at Gaussian points is also implemented. Moreover, the user may choose the collocation points manually. Note that our error estimate works only if $\rho_m < 1$, and for singular problems collocation can be applied in a straightforward manner whenever $\rho_1 > 0$.

### 1.2. Global Error Estimation

In [5], a novel estimate of the global error, based on the defect correction idea, was introduced. The numerical solution $p(t)$ obtained by collocation with $\rho_m < 1$ is used to define a "neighboring problem" to (1). The original and the neighboring problem are solved using the backward Euler method at the points $t_{i,j},\ j = 1, \ldots, m$, and $t_{i,m+1} := t_{i+1},\ i = 0, \ldots, N-1$. This yields the grid vectors[2] $\xi_{i,j}$ and $\pi_{i,j}$ as the solutions of the respective schemes

$$\frac{\xi_{i,j} - \xi_{i,j-1}}{t_{i,j} - t_{i,j-1}} = F(t_{i,j}, \xi_{i,j}), \quad \frac{\pi_{i,j} - \pi_{i,j-1}}{t_{i,j} - t_{i,j-1}} = F(t_{i,j}, \pi_{i,j}) + \bar{d}_{i,j}, \tag{3}$$

where $\bar{d}_{i,j}$ is a defect term defined by

$$\bar{d}_{i,j} := \frac{p(t_{i,j}) - p(t_{i,j-1})}{t_{i,j} - t_{i,j-1}} - \sum_{k=1}^{m+1} \alpha_{j,k} F(t_{i,k}, p(t_{i,k})). \tag{4}$$

[1] For certain spectral properties of the matrix $M(0)$ this estimate may additionally be affected by logarithmic terms like $|\ln(h)|^\alpha$.

[2] Here and in Theorem 1, we assume that $j = 1, \ldots, m+1,\ i = 0, \ldots, N-1$.

Here, the coefficients $\alpha_{j,k}$ are chosen in such a way that the following quadrature rules have precision $m+1$:

$$\frac{1}{t_{i,j}-t_{i,j-1}}\int_{t_{i,j-1}}^{t_{i,j}}\varphi(\tau)\,d\tau \approx \sum_{k=1}^{m+1}\alpha_{j,k}\varphi(t_{i,k}).$$

For regular problems, $\pi_{i,j}-\xi_{i,j}$ is indeed an asymptotically correct estimate for the global error of the collocation solution $p(t_{i,j})$:

THEOREM 1. *Assume that the regular boundary value problem*

$$z'(t) = F(t,z(t)), \quad t\in[a,b], \tag{5a}$$
$$B_a z(a) + B_b z(b) = \beta, \tag{5b}$$

*has an isolated solution $z\in C^{m+2}[a,b]$. Then the following estimate holds in a neighborhood of $z$:*

$$\max_{i,j}\left|(p(t_{i,j})-z(t_{i,j}))-(\pi_{i,j}-\xi_{i,j})\right| = O(\mathbf{h}^{m+1}), \tag{6}$$

*where* $\mathbf{h} := \max_{i=0,\ldots,N-1}|t_{i+1}-t_i|$.

A proof of this theorem is given in [5].

Experimental evidence strongly supports the proposition that Theorem 1 also holds for singular problems of the form (1), see [5]. Only in the case where $M(0)$ has a multiple eigenvalue 0, the order of the estimate given in (6) may be affected by logarithmic terms.

Moreover, it was demonstrated in [5] that the above global error estimate enables us to construct an effective mesh selection algorithm that is robust with respect to the singularity. This algorithm was implemented in the code described here. For a detailed description of the procedure refer to [4], [5]. Examples of the meshes generated to equidistribute the error and efficiently satisfying prescribed tolerances are given in §4.

## 2. Numerical Aspects of the Implementation

We now discuss some numerical aspects that require special attention in the implementation and use of the collocation code. As an example we consider the so-called Emden Differential Equation, cf. [9],

$$z'(t) = \frac{1}{t}\begin{pmatrix}0 & 1\\ 0 & -1\end{pmatrix}z(t) - \begin{pmatrix}0\\ tz_1^5(t)\end{pmatrix}, \tag{7a}$$
$$\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}z(0) + \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}z(1) = \begin{pmatrix}\frac{\sqrt{3}}{2}\\ 0\end{pmatrix}. \tag{7b}$$

The exact solution of this problem is

$$z(t) = (z_1(t), z_2(t))^T = \left( \frac{1}{\sqrt{1+t^2/3}}, -\frac{t^2}{3\sqrt{(1+t^2/3)^3}} \right)^T .$$

All computations were carried out in MATLAB using IEEE double precision arithmetic, with machine accuracy $eps \approx 1.11 \cdot 10^{-16}$.

To solve the nonlinear equations (2) for the coefficients of the collocating function with respect to a basis of piecewise polynomials, we use the *damped Newton method*, cf. e.g. [1]. For the solution of the systems of linear equations occurring during the Newton iteration, the question of the conditioning of the Jacobians involved is an important issue. Though it does not change the course of the iteration substantially for most mesh sizes, it nevertheless affects the maximal accuracy that can be achieved. Two aspects of the conditioning will be discussed here: An undesirable growth of the condition number which is due to a bad scaling of the Jacobian, and the influence of the polynomial basis on the condition number.

The collocating function is written in terms of a basis of piecewise polynomials $\varphi_j$,

$$p_i(t) = \sum_{j=1}^{m+1} x_{i,j} \varphi_j \left( \frac{t - t_i}{t_{i+1} - t_i} \right), \quad t \in [t_i, t_{i+1}], \quad i = 0, \ldots, N-1, \tag{8}$$

where the coefficient vector $x = x_{i,j},\ j = 1, \ldots, m+1,\ i = 0, \ldots, N-1$, is to be determined. Our code allows us to choose the $\varphi_j$ as either the Lagrange, Runge-Kutta, Legendre or monomial basis on the interval $[0, 1]$, cf. [1]. The Legendre and monomial bases are not recommended in general, however, and will not be discussed here.

Substitution of (8) into (2) leads to a Newton iteration with a Jacobian of a well-known sparse structure, see [1]. However, bad scaling of this matrix results in a condition number that grows quadratically with $N$. If we rescale the system matrix with a preconditioner of the form[3]

$$A = \text{Blockdiag} \left( \frac{1}{h_1} I_n, I_{mn}, \frac{1}{h_1} I_n, I_{mn}, \frac{1}{h_2} I_n, \ldots, I_{mn} \right),$$

we observe the expected linear growth. Table I gives the condition numbers[4] of the Jacobians of the final Newton steps for problem (7) and the empirical growth rates for both the preconditioned ($\text{cond}_1$,

[3] $I_k$ denotes the $k$-dimensional identity matrix.

[4] These estimates were computed with the MATLAB function `condest`.

$\mathrm{ord}_1$) and unmodified ($\mathrm{cond}_2$, $\mathrm{ord}_2$) case. Here, equidistant collocation with $m = 4$, equidistant meshes, and the Runge-Kutta basis were used.

Table I. Condition number growth

| $N$ | $\mathrm{cond}_1$ | $\mathrm{ord}_1$ | $\mathrm{cond}_2$ | $\mathrm{ord}_2$ |
|---|---|---|---|---|
| 4 | 5.13e+02 | | 1.91e+03 | |
| 8 | 7.50e+02 | 0.54 | 5.54e+03 | 1.53 |
| 16 | 1.22e+03 | 0.70 | 1.80e+04 | 1.69 |
| 32 | 2.17e+03 | 0.82 | 6.37e+04 | 1.82 |
| 64 | 4.08e+03 | 0.90 | 2.38e+05 | 1.90 |
| 128 | 7.88e+03 | 0.95 | 9.22e+05 | 1.94 |
| 256 | 1.55e+04 | 0.97 | 3.62e+06 | 1.97 |
| 512 | 3.07e+04 | 0.98 | 1.43e+07 | 1.98 |
| 1024 | 6.11e+04 | 0.99 | 5.71e+07 | 1.99 |

Another important aspect for the conditioning of the Jacobian is the choice of the polynomial basis $\{\varphi_j\}$. In Table II the errors relative to the magnitude of the exact solution are given in multiples of the machine precision *eps* for the Runge-Kutta and the Lagrange bases. The best results are obtained for the Runge-Kutta basis which is our default choice in the code.

Table II. Attainable accuracy in multiples of *eps*

| $N$ | $m = 4$, RK | $m = 4$, La | $m = 6$, RK | $m = 6$, La | $m = 8$, RK | $m = 8$, La |
|---|---|---|---|---|---|---|
| 16 | 1.18e+08 | 1.18e+08 | 1.89e+04 | 1.88e+04 | 5.00e+00 | **3.08e+02** |
| 32 | 7.41e+06 | 7.41e+06 | 2.98e+02 | **1.37e+02** | 2.01e+00 | 6.13e+02 |
| 64 | 4.63e+05 | 4.63e+05 | 2.36e+00 | 4.36e+02 | **1.50e+00** | 1.21e+03 |
| 128 | 2.89e+04 | 2.87e+04 | 4.00e+00 | 8.79e+02 | 5.00e+00 | 2.42e+03 |
| 256 | 1.81e+03 | 1.31e+03 | **1.52e+00** | 1.75e+03 | 2.11e+00 | 4.84e+03 |
| 512 | 1.12e+02 | **8.95e+02** | 4.06e+00 | 3.50e+03 | 4.04e+00 | 9.67e+03 |
| 1024 | **8.00e+00** | 2.00e+03 | 4.51e+00 | 7.01e+03 | 3.00e+00 | 1.93e+04 |

## 3. Implementation Details

The package `sbvp 1.0` consists of the driver routine for the adaptive mesh selection, `sbvp.m`, the collocation solver, `sbvpcol.m`, the error estimation routine, `sbvperr.m`, the tool for setting solution options, `sbvpset.m`, and output routines. The code is freely available from `http://www.math.tuwien.ac.at/~ewa/`. A detailed description of the solver syntax, the specification of problem data, options for the solution of the boundary value problems and the solution of the associated nonlinear algebraic equations, and the usage of the output functions (which provide plots and phase portraits of the solution) is given in [2].

Various measures to increase the efficiency of the code have been taken in the implementation. The storage of the variables and their referencing are optimized with respect to MATLAB's memory representation of multidimensional arrays, in order to ensure that whenever possible connected memory areas are accessed. We also use preallocation of memory to speed up memory access. Moreover, we make use of MATLAB's capability to process vectorized operations efficiently. This feature can be exploited to its full extent when the problem specification is given in a vectorized format using the options `'fVectorized'` and `'JacVectorized'`, see the detailed description of the package in [2]. An example codelet given in [3] was shown to reduce the execution time by up to 70% as compared to loop versions performing the same task.

## 4. Comparisons

We now give a comparison of the performance of `sbvp` with the MATLAB 6.0 routine `bvp4c` and the Fortran 90 code COLNEW. We only present the results for a small set of singular test problems here. A larger set of examples is discussed in [3].

The comparison of all three codes is not easy. COLNEW and `bvp4c` were designed to solve regular problems, while implementing `sbvp` we mainly focussed on singular problems. However, all three codes can deal with both. The basic concepts of COLNEW and `sbvp` are similar, while the philosophy of `bvp4c` is different. In this context, there are two points we would like to mention. While COLNEW and `sbvp` are based on collocation methods whose orders may vary, in a way such that they are also appropriate for high accuracies, the collocation method in `bvp4c` has a fixed order 4 and is mainly suitable for moderate accuracies. Therefore, it is clear in advance that this solver will be less competitive for stringent tolerances. Moreover, while COLNEW and `sbvp` require the analytical specification of the Jacobian, finite differences are used

in `bvp4c` by default to approximate the Jacobian. Many users will definitely find this very useful and comfortable, and we are considering an incorporation of this feature into a future version of our code. However, this way of specifying the Jacobian is more expensive in terms of function evaluations. Although for reasons mentioned here, the comparison of all three codes is difficult, we felt that we should not neglect the available MATLAB code `bvp4c`, especially as in its description [13] the option of using it to treat singular problems is explicitly mentioned. Below, we illustrate that for singular problems with a smooth solution, `sbvp` is competitive especially in the linear case.

First, we consider the singular boundary value problem

$$z'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 1+\alpha^2 t^2 & 0 \end{pmatrix} z(t) + \begin{pmatrix} 0 \\ ct^{k-1}\mathrm{e}^{-\alpha t}(k^2 - 1 - \alpha t(1+2k)) \end{pmatrix}, \quad (9a)$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} z(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} z(1) = \begin{pmatrix} 0 \\ c\mathrm{e}^{-\alpha} \end{pmatrix}, \quad (9b)$$

where $\alpha = 80$, $k = 16$ and $c = \left(\frac{\alpha}{k}\right)^k \mathrm{e}^k$. 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

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

A plot of this solution is given in Figure 1.

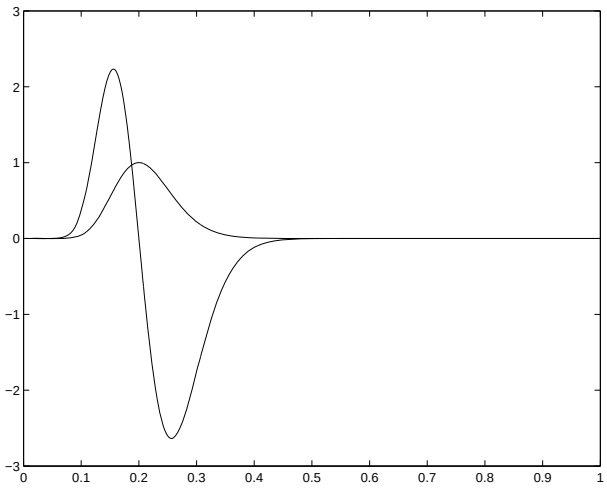


*Figure 1.* Solution of (9)

In Table III we show the number of mesh points ($N$) and the number of function evaluations of the right-hand side of (9a) (fcount) that the

different solvers required to reach a relative (rTOL) and an absolute (aTOL) tolerance of $10^{-5}$. To demonstrate that the computed results are indeed meaningful, we also record the maximum norms of the global error estimate (esterr $= \max_{i,j} |\pi_{i,j} - \xi_{i,j}|$) and the true global error (maxerr $= \max_{i,j} |p(t_{i,j}) - z(t_{i,j})|$). We tested 7 different methods for this task:

- `bvp4c`, which is based on collocation at three Lobatto points (see [13]). This is a method of order 4 for regular problems. Note that, unfortunately, `bvp4c` does not provide an error estimate to the user.
- COLNEW: The basic method here is collocation at Gaussian points. We chose the polynomial degrees $m = 4$ (CW-4) and $m = 6$ (CW-6), which results in (superconvergent) methods of orders 8 and 12 (for regular problems), respectively.
- `sbvp` is used with equidistant (`sbvp4` and `sbvp6`) and Gaussian (`sbvp4g` and `sbvp6g`) collocation points and polynomial degrees 4 and 6.

Table III. Comparisons for (9), aTOL=rTOL=$10^{-5}$

| | `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 |

The results given in Table III show that the performance of `sbvp` is comparable with that of COLNEW for this linear problem. For $m = 6$, which `sbvp` chooses as a default for these tolerances, the number of function evaluations is noticeably smaller. In this case, collocation at Gaussian points shows a slightly more advantageous behavior as compared to equidistant nodes. This is not the case, however, for $m = 4$. This is probably due to our error control mechanism, which is based on the *stage order* $m$, while for this example, the solution at the mesh points has the (super-)convergence order $2m$, see Table IV. This high order cannot be expected in general, however. Moreover, the uniform convergence order which is crucial for our procedure is only $m+1$. `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[5], and moreover makes it impossible to specify a Jacobian with a singularity at $t = 0$. However, this inefficiency is compensated to some extent by the evaluations of the analytic Jacobian for COLNEW and `sbvp`, which is not taken into consideration 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. Again, we note that `bvp4c` does not provide an error estimate, while the other two codes do.

Table IV. Convergence order of collocation at Gaussian points, $m = 4$, for (9)

| $N$ | err | ord |
|---|---|---|
| 32 | 5.91e−06 | |
| 64 | 3.50e−08 | 7.39 |
| 128 | 1.51e−10 | 7.85 |
| 256 | 6.11e−13 | 7.95 |

Another interesting aspect is the maximal accuracy that could be achieved by the three programs for (9). For tolerances $10^{-14}$ (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 rTOL=aTOL=$10^{-11}$, yielding

[5] In general, $z'(0) = (I - M(0))^{-1}(M'(0)z(0) + f(0, z(0)))$ holds if the real parts of the eigenvalues of $M(0)$ are greater than 1. For many problems from applications we can conclude from this relation that $z'(0) = 0$. In [13] a similar approach is adopted for a second order problem which is then transformed to the first order form.

a solution on a mesh containing 5533 points with a true error of $1.25 \cdot 10^{-12}$.

To conclude the discussion of (9), we give a graphical representation of the meshes and the exact and estimated errors (denoted by $\star$ and $\bullet$, respectively) generated by `bvp4c` (Figures 2 and 3 — no error estimate can be given in Figure 3), CW-4 (Figures 4 and 5) and `sbvp4` (Figures 6 and 7) for rTOL=aTOL=$10^{-5}$.

*Figure 2.* Mesh generated by `bvp4c` for (9), aTOL=rTOL=$10^{-5}$

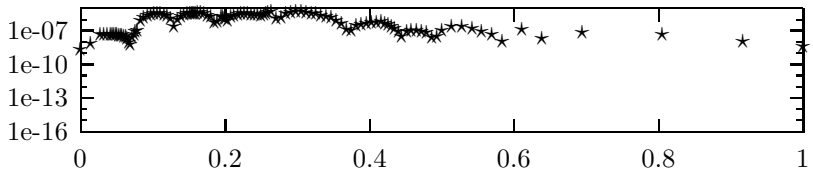


*Figure 3.* Exact global error for `bvp4c` applied to (9)

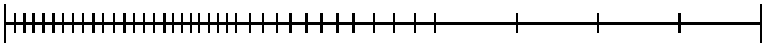

*Figure 4.* Mesh generated by CW-4 for (9), aTOL=rTOL=$10^{-5}$

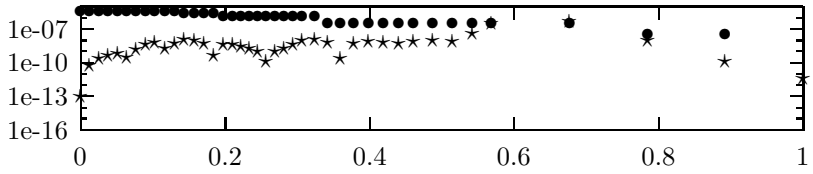


*Figure 5.* Exact ($\star$) and estimated ($\bullet$) global error for CW-4 applied to (9)

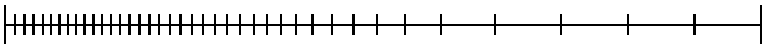

*Figure 6.* Mesh generated by `sbvp4` for (9), aTOL=rTOL=$10^{-5}$

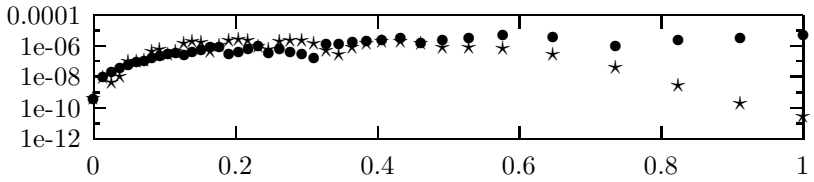


*Figure 7.* Exact ($\star$) and estimated ($\bullet$) global error for `sbvp4` applied to (9)

The next example we discuss is

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

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} z(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} z(1) = \begin{pmatrix} 0 \\ \sin(k^2) \end{pmatrix}, \quad (10b)$$

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

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

and its plot is given in Figure 8.

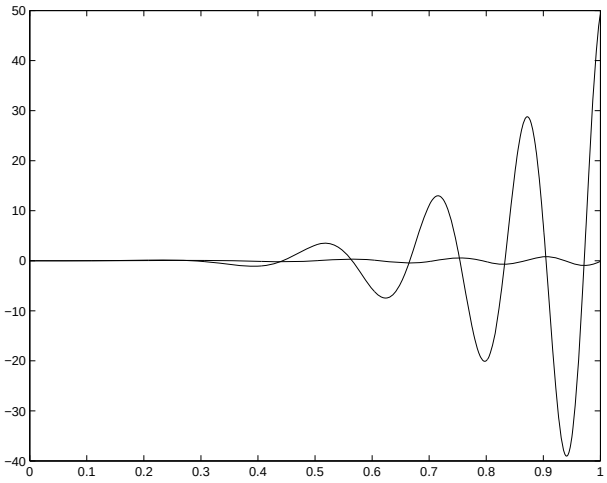


*Figure 8.* Solution of (10)

Table V shows the mesh size $N$ and the function count 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

Table V. Comparisons for (10), aTOL=rTOL=$10^{-9}$

| | 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 |

tolerances. Collocation at Gaussian points is advantageous for this example. Indeed, no order reduction is observed. If Gaussian points are considered, `sbvp` is competitive with COLNEW, and if the possibility to use the order 8 is exploited, the code works very efficiently. Moreover, all the error estimates are quite reliable for this example; maxerr $\approx$ esterr $\approx$ rTOL = aTOL holds in every case.

Finally, we give the results for a nonlinear example. This is a second order problem for $y(t)$ taken from [13], but in contrast to this paper, where the standard transformation $y(t) \to (y(t), y'(t))$ is used, we apply Euler's transformation $y(t) \to (y(t), ty'(t)) =: z(t)$ to obtain

$$z'(t) = \frac{1}{t}\begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix} z(t) + \begin{pmatrix} 0 \\ t\phi^2 z_1(t) \exp\left(\frac{\gamma\beta(1-z_1(t))}{1+\beta(1-z_1(t))}\right) \end{pmatrix}, \tag{11a}$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} z(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} z(1) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{11b}$$

For the parameter values $\phi = 0.6,\ \gamma = 40,\ \beta = 0.2$, 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 9.

Table VI gives the values of $N$ and fcount for aTOL=rTOL=$10^{-7}$. The meshes generated by COLNEW and `sbvp` are comparable; however, the values of fcount are not very favorable for `sbvp`, although using Gaussian points improves the performance noticeably (no order reduction is observed in this case, either). A Quasi-Newton iteration could certainly be worthwhile trying.

We have also run experiments at lower tolerances (aTOL = rTOL= $10^{-3}$ or $10^{-4}$). We do not include these results here since they perfectly fit into the picture. COLNEW and `sbvp` perform quite comparably, with certain deficiencies of our code resulting from the Newton procedure, whereas `bvp4c` usually requires more meshpoints to reach the tolerance. For very low tolerances or easy problems, none of the codes requires an adaptation of the initial mesh and no significant conclusions can be drawn from such tests, cf. [3].

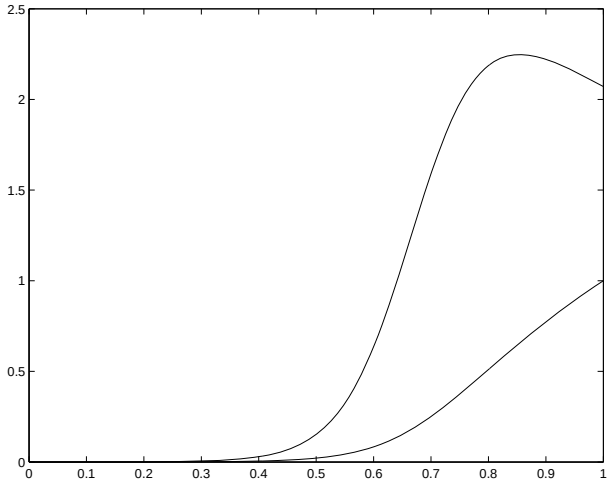


*Figure 9.* One solution of (11)

Table VI. Comparisons for (11), aTOL=rTOL=$10^{-7}$

| | `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 |

## 5. Conclusions

We have presented the MATLAB package `sbvp` for singular boundary value problems. It turns out that `sbvp` is competitive for this problem class. Our next goal is to test the performance of `sbvp` applied to regular problems. Here, the efficient treatment of another difficulty – steep slopes in the solution profile – may be the key issue. Our tests for the regular case have not been extensive so far, since our main focus was on singular problems. On the basis of future experiments we shall be able to decide if further modifications of the code are necessary.

## References

1. Ascher, U., R. Mattheij, and R. Russell: 1988, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Englewood Cliffs, NJ: Prentice-Hall.
2. Auzinger, W., G. Kneisl, O. Koch, and E. Weinmüller: 2002a, 'SBVP 1.0 — A MATLAB Solver for Singular Boundary Value Problems'. Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Tech-

nology, Austria. Also available as ANUM Preprint Nr. 2/02 at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.
3. Auzinger, W., G. Kneisl, O. Koch, and E. Weinmüller: 2002b, 'A Solution Routine for Singular Boundary Value Problems'. Techn. Rep. ANUM Preprint Nr. 1/02, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology, Austria. Available at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.
4. Auzinger, W., O. Koch, W. Polster, and E. Weinmüller: 2001, '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. Available at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.
5. Auzinger, W., O. Koch, and E. Weinmüller: 2002c, 'Efficient Collocation Schemes for Singular Boundary Value Problems'. *Numer. Algorithms* **31**, 5–25.
6. Auzinger, W., P. Kofler, and E. Weinmüller: 1998, 'Steuerungsmaßnahmen bei der numerischen Lösung singulärer Anfangswertaufgaben'. Techn. Rep. Nr. 124/98, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology. Available at `http://www.math.tuwien.ac.at/~ewa/`.
7. Frank, R. and C. Überhuber: 1978, 'Iterated Defect Correction for Differential Equations, Part I: Theoretical Results'. *Computing* **20**, 207–228.
8. Frommlet, F. and E. Weinmüller: 2001, 'Asymptotic Error Expansions for Singular Boundary Value Problems'. *Math. Models Methods Appl. Sci.* **11**, 71–85.
9. Hoog, F. d. and R. Weiss: 1976, 'Difference Methods for Boundary Value Problems with a Singularity of the First Kind'. *SIAM J. Numer. Anal.* **13**, 775–813.
10. Hoog, F. d. and R. Weiss: 1978, 'Collocation Methods for Singular Boundary Value Problems'. *SIAM J. Numer. Anal.* **15**, 198–217.
11. Koch, O. and E. Weinmüller: 2001, 'Iterated Defect Correction for the Solution of Singular Initial Value Problems'. *SIAM J. Numer. Anal.* **38**(6), 1784–1799.
12. Koch, O. and E. Weinmüller: 2003, 'The Convergence of Shooting Methods for Singular Boundary Value Problems'. *Math. Comp.* **72**(241), 289–305.
13. Shampine, L., J. Kierzenka, and M. Reichelt: 2000, 'Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with `bvp4c`'. Available at `ftp://ftp.mathworks.com/pub/doc/papers/bvp/`.
14. Stetter, H. J.: 1978, 'The Defect Correction Principle and Discretization Methods'. *Numer. Math.* **29**, 425–443.
15. Weinmüller, E.: 1986a, 'Collocation for Singular Boundary Value Problems of Second Order'. *SIAM J. Numer. Anal.* **23**, 1062–1095.
16. Weinmüller, E.: 1986b, 'On the Numerical Solution of Singular Boundary Value Problems of Second Order by a Difference Method'. *Math. Comp.* **46**, 93–117.
17. Zadunaisky, P.: 1976, 'On the Estimation of Errors Propagated in the Numerical Integration of ODEs'. *Numer. Math.* **27**, 21–39.