Department for Analysis and Scientific Computing

Vienna University of Technology, Austria

BVPSUITE1.1

A New MATLAB Solver for Singular Implicit Boundary Value Problems.

Georg Kitzhofer

Othmar Koch

Gernot Pulverer

Christa Simon

Ewa B. Weinmüller

Vienna, June 2010

BVPSUITE1.1 – A New MATLAB Solver for Singular/Regular Boundary Value Problems in ODEs

G. Kitzhofer, O. Koch, G. Pulverer, Ch. Simon, E. Weinmüller*

June 15, 2010

Abstract

Our aim is to provide an open domain MATLAB code `bvpsuite` for the efficient numerical solution of boundary value problems (BVPs) in ordinary differential equations (ODEs). Motivated by applications, we are especially interested in designing a code whose scope is appropriately wide, including fully implicit problems of mixed orders, parameter dependent problems, problems with unknown parameters, problems posed on semi-infinite intervals, eigenvalue problems (EVPs) and differential algebraic equations (DAEs) of index 1. Our main focus is on singular BVPs in which singularities in the differential operator arise. We first shortly recapitulate the analytical properties of singular systems and the convergence behavior of polynomial collocation used as a basic solver in the code for both singular and regular ODEs and DAEs. We also discuss the posteriori error estimate and the grid adaptation strategy implemented in our code. Finally, we describe the code structure, input/output parameters, and present the performance of the code. We provide numerical examples to help using our software package, especially the graphical user interface (GUI).

*Institute for Analysis and Scientific Computing, Vienna University of Technology, Wiedner Hauptstrasse 8–10/101, 1040 Wien, Austria, e.weinmueller@tuwien.ac.at, http://www.math.tuwien.ac.at/~ewa

Contents

1 Introduction

In the introduction, we concentrate on the theoretical results available for problems with singularities. Let us consider the numerical solution of singular BVPs of the form

$$z'(t) = \frac{M(t)}{t^\alpha} z(t) + f(t, z(t)), \quad t \in (0, 1], \tag{1a}$$

$$B_0 z(0) + B_1 z(1) = \beta, \tag{1b}$$

where $\alpha \geq 1$, z is an n-dimensional real function, M is a smooth $n \times n$ matrix and f is an n-dimensional smooth function on a suitable domain. B_0 and B_1 are constant matrices which are subject to certain restrictions for a well-posed problem. (1a) is said to feature a *singularity of the first kind* for $\alpha = 1$, while for $\alpha > 1$ the problem has a *singularity of the second kind*, also commonly referred to as *essential singularity*. The analytical properties of problem (1) have been discussed in [12], [14] with a special focus on the most general boundary conditions which guarantee well-posedness of the problem. When analyzing singular problems, we first note that their direction field is very unsmooth, especially close to the singular point. Consequently, depending on the spectrum of the matrix $M(0)$, we can encounter unbounded contributions to the solution manifold, such that $z \in C(0, 1]$. However, irrespective of the eigenvalues of $M(0)$, by posing proper homogeneous initial conditions, we can extend the above solution to $z \in C[0, 1]$. It also turns out that in such a case the condition $M(0)z(0) = 0$ must hold. For singular problems the solution's smoothness depends not only on the smoothness of the inhomogeneity f but also on the size of real parts of the eigenvalues of $M(0)$. To compute the numerical solution of (1) we use polynomial collocation. Our decision to use polynomial collocation was motivated by its advantageous convergence properties for (1), while in the presence of a singularity other high order methods show order reductions and become inefficient. In [6], [13], and [21] convergence results for collocation applied to problems with a singularity of the first kind, $\alpha = 1$, were shown. The usual high-order superconvergence at the mesh points does not hold in general for singular problems, however, the uniform superconvergence is preserved (up to logarithmic factors), see [21] for details.

Motivated by these observations, we have implemented the present MATLAB code. We stress, that for `bvpsuite` one singular point at either endpoint or two singular points at both endpoints of the interval of integration are admissible. The program can by applied directly to such singular BVPs and no pre-handling is necessary. Otherwise a reformulation of the problem may be necessary, cf. (25) and (26). For higher efficiency, we provide an estimate of the global error and adaptive mesh selection. Transformation of problems posed on semi-infinite intervals to a finite domain makes the solution of such problems also accessible to our methods. All these algorithmic components have been integrated into the code. While the first version of the code, `sbvp`, solves explicit first order ODEs [4], the present version, `bvpsuite1.1`, can be applied to arbitrary order problems also in implicit formulation. Consequently, differential algebraic equations [19] are also in the scope of the code. We have also implemented a module for EVPs in which we recast the EVP as an appropriately defined BVP. Moreover, a pathfollowing strategy extends the scope of the code to parameter dependent problems. For special problem classes, such as singularly perturbed models, systems of DAEs, parameter dependent problems and EVPs very good and efficient software already exists. We have not performed comparisons with those codes. However, we assess the properties

of `bvpsuite` when the code is applied to *singular BVPs* by comparing it with other, well-established software for BVPs in ODEs. Numerical simulation of relevant applications illustrates the scope and the performance of the implementation.

2 Basic Solver in the MATLAB Code `bvpsuite`

The code is designed to solve systems of differential equations of arbitrary mixed order including zero[1], subject to initial or boundary conditions,

$$F\big(t, p_1, \ldots, p_s, z_1(t), z_1'(t), \ldots, z_1^{(l_1)}(t), \ldots, z_n(t), z_n'(t), \ldots, z_n^{(l_n)}(t)\big) = 0, \tag{2a}$$

$$B\big(p_1, \ldots, p_s, z_1(c_1), \ldots, z_1^{(l_1-1)}(c_1), \ldots, z_n(c_1), \ldots, z_n^{(l_n-1)}(c_1), \ldots, \tag{2b}$$

$$z_1(c_q), \ldots, z_1^{(l_1-1)}(c_q), \ldots, z_n(c_q), \ldots, z_n^{(l_n-1)}(c_q)\big) = 0, \tag{2c}$$

where the solution $z(t) = (z_1(t), z_2(t), \ldots, z_n(t))^T$, and the parameters p_i, $i = 1, \ldots, s$, are unknown. In general, $t \in [a, b]$ or $t \in [a, \infty)$[2]. Moreover, $F : [a, b] \times \mathbb{R}^s \times \mathbb{R}^{l_1} \times \cdots \times \mathbb{R}^{l_n} \to \mathbb{R}^n$ and $B : \mathbb{R}^s \times \mathbb{R}^{ql_1} \times \cdots \times \mathbb{R}^{ql_n} \to \mathbb{R}^{l+s}$, where $l := \sum_{i=1}^n l_i$. Note that boundary conditions can be posed on any subset of distinct points $c_i \in [a, b]$, with $a \le c_1 < c_2 < \cdots < c_q \le b$. For the numerical treatment, we assume that the boundary value problem (2) is well-posed and has a locally unique solution z.

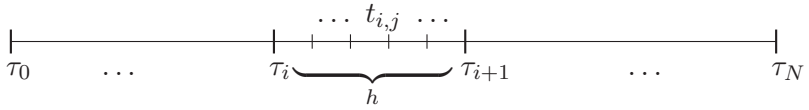

Figure 1: The computational grid

In order to find a numerical solution of (2) we consider meshes

$$\Delta := (\tau_0, \tau_1, \ldots, \tau_N), \tag{3}$$

with

$$h_i := \tau_{i+1} - \tau_i, \quad J_i := [\tau_i, \tau_{i+1}], \quad i = 0, \ldots, N-1, \quad \tau_0 = a,\ \tau_N = b. \tag{4}$$

partitioning the interval $[a, b]$. Every subinterval J_i contains m collocation points $t_{i,j}$, $j = 1, \ldots, m$ such that $t_{i,j} = \tau_i + \rho_j h$, $j = 1, \ldots, m$, with

$$0 < \rho_1 < \rho_2 \cdots < \rho_m < 1, \tag{5}$$

see Figure 1. To avoid a special treatment of the possible singular points $t = a$ and $t = b$, cf. [10], grids with $\rho_1 > 0$ and $\rho_m < 1$, e.g. Gaussian points, inner equidistant points, are used. Let $\mathbf{P}_m$ be the space of piecewise polynomial functions of degree $\le m$, which are globally continuous in $[a, b]$. In every subinterval J_i we make an ansatz $P_{i,k} \in \mathbf{P}_{m+l_k-1}$ for the k-th solution component z_k, $k = 1, \ldots, n$, of the problem (2). In order to compute the coefficients in the ansatz functions we require that (2) is satisfied exactly

[1] This means that differential-algebraic equations are also in the scope of the code.

[2] For the extension to unbounded domains, see Section 6.1.

at the collocation points. Moreover we require that the collocation polynomial $p(t) := P_i(t)$, $t \in J_i$, is a globally continuous function on $[a, b]$ with components in $C^{l_i-1}[a, b]$, $i = 1, \ldots, n$, and that the boundary conditions hold. All these conditions imply a nonlinear system of equations for the unknown coefficients in the ansatz function. For more details see [16].
For the representation of the collocation polynomial p we use the Runge-Kutta basis, see [2], and solve the resulting nonlinear system for the coefficients in this representation by a Newton iteration implemented in the subroutine 'solve_nonlinear_sys.m' from the MATLAB code sbvp, cf. [4], which is based on the 'fast frozen' Newton method.

3 Error Estimate for the Global Error of the Collocation

Our estimate for the global error of the collocation solution is a classical error estimate based on mesh halving. In this approach, we compute the collocation solution at N points on a grid Δ with step sizes h_i and denote this approximation by $p_\Delta(t)$. Subsequently, we choose a second mesh Δ_2 where in every interval J_i of Δ we insert two subintervals of length $h_i/2$. On this mesh, we compute the numerical solution based on the same collocation scheme to obtain the collocating function $p_{\Delta_2}(t)$. Using these two quantities, we define

$$\mathcal{E}(t) := \frac{2^m}{1 - 2^m}(p_{\Delta_2}(t) - p_\Delta(t)) \tag{6}$$

as an error estimate for the approximation $p_\Delta(t)$. Assume that the global error $\delta(t) := p_\Delta(t) - z(t)$ of the collocation solution can be expressed in terms of the principal error function $e(t)$,

$$\delta(t) = e(t)h_i^m + O(h_i^{m+1}), \quad t \in J_i, \tag{7}$$

where $e(t)$ is independent of Δ. Then obviously the quantity $\mathcal{E}(t)$ satisfies $\mathcal{E}(t) - \delta(t) = O(h^{m+1})$. This holds for problems with a singularity of the first kind and for regular problems. However, numerical results reported in [5] indicate that in case of an essential singularity (7) reads

$$\delta(t) = e(t)h_i^m + O(h_i^{m+\gamma}), \quad t \in J_i, \tag{8}$$

with $\gamma < 1$. Generally, estimates of the global error based on mesh halving work well for both problems with a singularity of the first kind and for essentially singular problems [5]. Since they are also applicable to higher-order problems and problems in implicit form (as for example DAEs) without the need for modifications, we have implemented this strategy in our code `bvpsuite`.

4 Adaptive Mesh Selection

The mesh selection strategy implemented in `bvpsuite` was proposed and investigated in [23]. Most modern mesh generation techniques in two-point BVPs generate a smooth function mapping a uniform auxiliary mesh to the desired nonuniform mesh. In [23] a new system of control algorithms for constructing a mesh density function $\phi(t)$ is described. The local mesh width $h_i = \tau_{i+1} - \tau_i$ is computed as $h_i = \epsilon_N/\varphi_{i+1/2}$, where $\epsilon_N = 1/N$ is the accuracy control parameter corresponding to $N - 1$ interior points, and the positive sequence $\Phi = \{\varphi_{i+1/2}\}_{i=0}^{N-1}$ is a discrete approximation to the continuous density function $\phi(t)$,

representing the mesh width variation. Using an error estimate, a feedback control law generates a new density from the previous one. Digital filters are employed to process the error estimate as well as the density.

For BVPs, an adaptive algorithm must determine the sequence $\Phi^{[\nu]}$ in terms of problem or solution properties. True adaptive approaches equidistribute some *monitor function*, a measure of the residual or error estimate, over the interval. As $\Phi^{[\nu]}$ will depend on the error estimates, which in turn depend on the distribution of the mesh points, the process of finding the density becomes iterative. In order to be able to generate the mesh density function, we decided to use the residual $r(t)$ to define the monitor function. The values of $r(t)$ are available from the substitution of the collocation solution $p(t)$ into the analytical problem (2). In the mesh adaptation routine implemented in the code, finding the optimal density function, is separated from finding the proper number of mesh points. We first try to provide a good density function Φ on a rather coarse mesh with a fixed number of points. For each density profile in the above iteration, we then estimate the number of mesh points necessary to reach the tolerance. When the decrease in this number is appropriately small, the density function is considered satisfactory. Using this density functions the problem is solved with the necessary number of mesh points to reach the tolerance.

5 Eigenvalue Problems for Ordinary Differential Equations

Eigenvalue problems for ODEs may have different formulations. We consider linear problems (9a), where the linear differential operator is denoted by L, subject to homogeneous boundary conditions (9b),

$$(Lz)(t) = \lambda z(t), \quad t \in [a, b], \tag{9a}$$

$$B_0 z(a) + B_1 z(b) = 0. \tag{9b}$$

Problems (9) can also be posed on semi-infinite intervals $[a, \infty)$. The aim is to determine values for the *eigenparameter* λ for which the BVP (9) has non-trivial solutions, so-called *eigenfunctions*.

In [3] it has been demonstrated how a code designed to solve BVPs in ODEs can be used to treat eigenvalue problems. The method can be applied in the case that for each eigenvalue the corresponding eigenspace is of dimension one. This requirement is not very restrictive, since it is most commonly satisfied in applications. To illustrate the approach proposed in [3] we consider a singular first order system of the form

$$z'(t) - \frac{M(t)}{t^\alpha} z(t) = \lambda z(t), \quad t \in (0, 1], \tag{10a}$$

$$B_0 z(0) + B_1 z(1) = 0. \tag{10b}$$

where the matrix $M(t)$ is assumed to be sufficiently smooth and $\alpha \geq 1$. In case that for a given eigenvalue the corresponding eigenspace is of dimension one, the normalization condition,

$$x(t) := \int_0^t |z(\tau)|^2 d\tau, \quad x(1) = 1,$$

ensures the uniqueness of the eigenfunction. We now augment problem (10) by the trivial ODE

$$\lambda'(t) = 0, \tag{11}$$

the additional equation derived from the normalization condition

$$x'(t) = |z(t)|^2, \tag{12}$$

and two new boundary conditions

$$x(0) = 0, \quad x(1) = 1, \tag{13}$$

which yields a 'standard' BVP with the unknowns $(\lambda, z(t), x(t))$. Here, $|z(t)|^2 := \sum_{k=1}^{n} |z_k(t)|^2$. For further theoretical results on the properties of the solution of (10)–(13), see [3]. After the described reformulation of the problem any suitable numerical method for singular BVPs in ODEs can be used. The problem is now treated as a standard BVP. As mentioned before the method applied here is polynomial collocation which has proven to be a robust technique for solving BVPs with singularities (cf. [2]). While in [3] theoretical results supporting the described solution approach are only provided for linear first order problems, extensive test runs (cf. [28]) show that this method provides reliable results also for more general situations. Using `bvpsuite` we can solve eigenvalue problems (9) of arbitrary order.

6 Pathfollowing and Problems on Semi-Infinite Intervals

In this section, we discuss two further features of `bvpsuite` which allow to cover additional types of difficulties important for a wide range of applications.

6.1 Pathfollowing Strategy

Our code realizes a pathfollowing strategy to follow solution branches in dependence of a known parameter. To describe the strategy in general terms, we consider (1) as a parameter-dependent operator equation

$$F(y; \lambda) = 0, \tag{14}$$

where $F : Y \times \mathbb{R} \to Z$, and Y, Z are Banach spaces (of possibly infinite dimension). Pathfollowing in this general setting has been discussed in detail in [31].

We are particularly interested in computing solution branches Γ with *turning points*. By definition, in a turning point the solution of (14) constitutes a local maximum (or minimum) of λ, and consequently is not locally unique as a function of the parameter λ. The situation is illustrated in Figure 2. There, we plot some functional of the solution against the parameter λ. The arrows indicate the turning points. Thus, in a turning point we cannot parametrize Γ as a function of λ. However, it is sufficient for our procedure that a tangent is uniquely determined at all points of Γ. This is guaranteed by realistic assumptions formulated for our problem in [20].

Now, we proceed by describing our pathfollowing strategy. As explained in [20], our assumptions on the problem ensure that at a point $(y_0, \lambda_0) \in \Gamma$, a tangent can be uniquely determined up to the sign. Additional criteria determine how to choose the direction. On the tangent just computed, a predictor (y_P, λ_P) is chosen for the computation of the next point on Γ, and finally a corrector equation is solved yielding (y_C, λ_C). One step of our procedure starting at (y_0, λ_0) is illustrated in Figure 2.

To demonstrate that our pathfollowing strategy indeed works for singular BVPs and generates meshes adapted to the solution profile, in [20] we considered an example from [11], describing the buckling of a

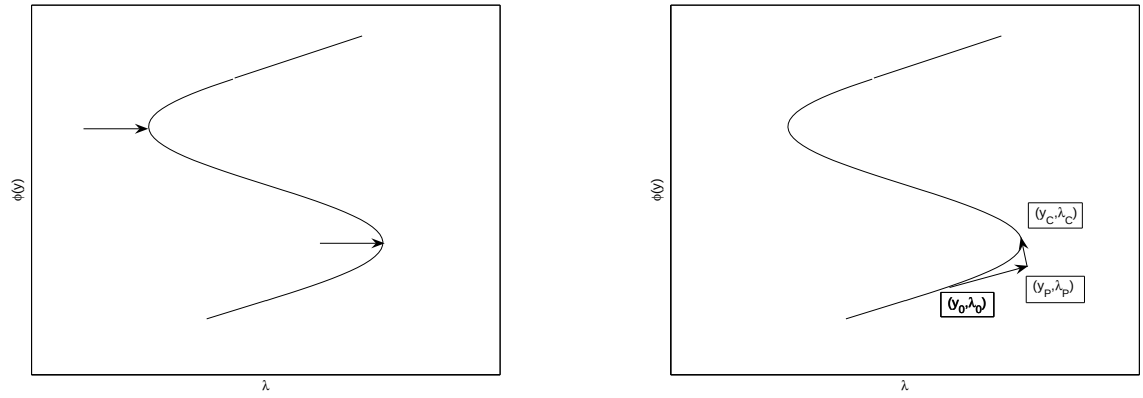

Figure 2: A solution branch with two turning points (left), one step of the pathfollowing procedure (right).

spherical shell. We followed the solution path Γ shown in Figure 3, starting at $\lambda = 0$. Figure 3 shows the maximum norm of the first solution component, $\|\beta\|_\infty$ along the path Γ. The crosses indicate points of Γ where the solution profiles of β and the second solution component Ψ are plotted in Figure 4, together with the meshes generated by our adaptive mesh selection procedure. A comparison with [11, Figure 10] shows that the solution is computed reliably and obviously the meshes are denser where the solution varies more rapidly.

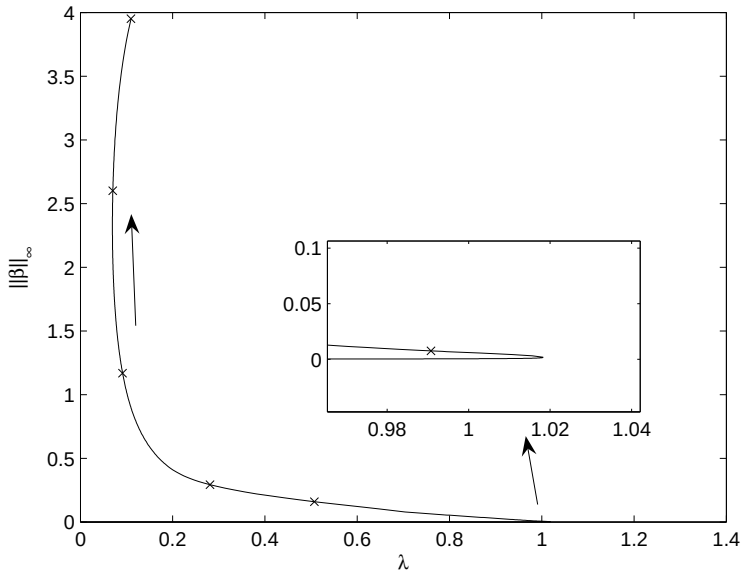

Figure 3: Values of $\|\beta\|_\infty$ along a solution branch.

6.2 Boundary Value Problems Posed on a Semi-Infinite Interval

Problems posed on semi-infinite intervals, $t \in [a, \infty)$, $a \geq 0$, are also in scope of `bvpsuite`. We always try to work on a finite domain, in order to exploit our efficient and robust mesh selection strategy. As a

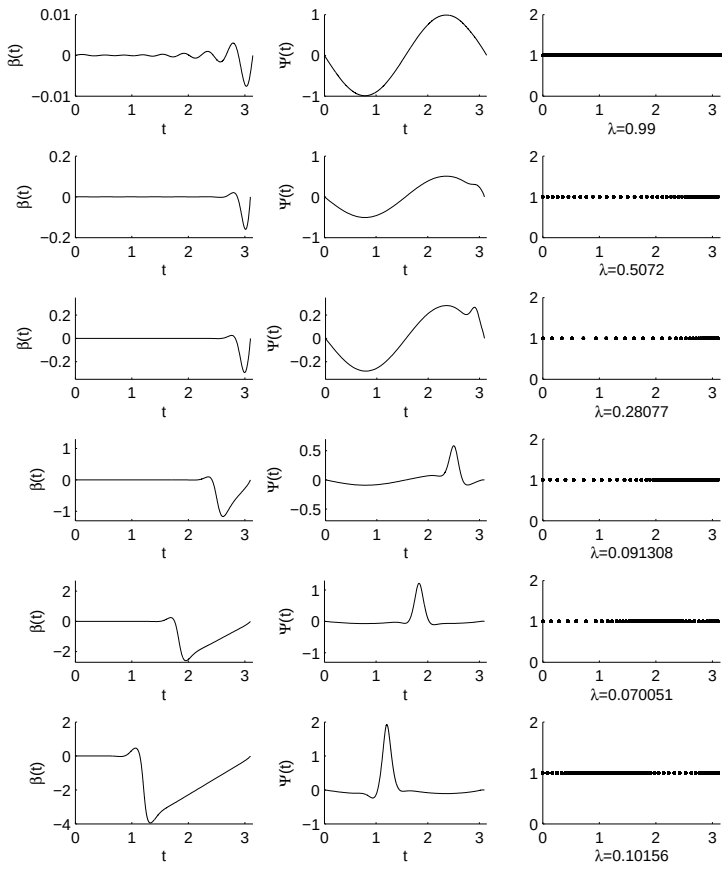

Figure 4: Solution profiles and automatically selected meshes at the points marked in Figure 3 along the solution branch.

possible special case, let us consider the differential equation

$$x'(\tau) = \tau^{\beta} f(\tau, x(\tau)), \quad \tau \in [a, \infty), \quad \beta \in \mathbb{R}.$$

For $a > 0$, we use the transformation $t := \frac{a}{\tau},\ z(t) := x\left(\frac{1}{\tau}\right)$ to obtain

$$z'(t) = -\frac{1}{t^{\beta+2}} f(1/t, z(t)), \quad t \in (0, 1].$$

For $a = 0$, the interval $[0, \infty)$ is split into $[0, 1]$ and $[1, \infty)$ and then the above transformation is applied to the semi-infinite interval $[1, \infty)$. Moreover, additional matching conditions are imposed to ensure the smoothness of the solution at the splitting point $t = 1$. Note that for $\beta > -1$ it follows $\beta + 2 > 1$ and therefore, an essential singularity arises in general, which however can be handled by the collocation method, error estimation procedure and adaptive mesh refinement. Using this approach we avoid mesh grading on long intervals and introducing additional perturbations due to the fact that boundary conditions at infinity are posed at some point $L < \infty$, L large. The above strategy has been successfully applied in a variety of relevant applications, cf. [8], [9], and [17].

The automatic interval reduction is now implemented in `bvpsuite`. The transformation can be applied to equations of arbitrarily high order. However, the boundary conditions at infinity have to be of Dirichlet

type when the transformation is carried out automatically. Otherwise, a manual transformation to a finite interval is recommended before running `bvpsuite`. In this case, further analysis of the solution and its derivatives near infinity is often necessary, cf. [28].

7 The Package

The current version of the package is `bvpsuite1.1` (Release June 2010).

7.1 Installation

Create an empty folder and copy the files of `bvpsuite1.1.zip` into it. Execute MATLAB as a user with rw-permission on that folder. The code has been tested for the MATLAB versions 7.1–7.2 (R2006a), R2007b (under Windows), R2008b (under Windows) R2009b (under Unix). The code should also work for later versions of MATLAB.

7.2 Files in this Package

The package `bvpsuite` contains the following m-files

- bvpsuite.m – main routine to start the GUI.
- equations.m – contains the most important parts of the code, e.g. setting up the nonlinear system of equations for the Newton solver.
- solve_nonlinear_sys.m – contains the Newton solver.
- run.m – manages routine calls.
- errorestimate.m – provides error estimates.
- meshadaptation.m – runs the automatic grid control.
- initialmesh.m – provides the initial data for the Newton solver.
- pathfollowing.m – realizes the pathfollowing routine.
- settings.m – opens a window to set parameters.
- sbvpset.m – sets the options for the Newton solver.
- EVPmodule.m – carries out the reformulation of an EVP to a BVP.
- trafomodule.m – automatically transforms a problem posed on a semi-infinite interval $[a, \infty)$, $a \geq 0$ to a finite domain $[0, 1]$.
- backtransf.m – back-transforms the solution to the interval $[a, L] \subset [a, \infty)$, L large.
- plot_results.m – provides graphical solution output.
- plotrange.m – defines settings for a solution plot on a subinterval $[a, L] \subset [a, \infty)$, L large.
- err.m – contains error messages.

7.3 Test Examples

In order to demonstrate the basic features of the GUI, we consider the following examples. Our main focus is on singular problems 7.2, 7.5, and 7.6.

Example 7.1 *We solve a singularly perturbed BVP on the interval* $-1 \leq t \leq 1$,

$$\varepsilon z''(t) + z'(t) - (1+\varepsilon)z(t) = 0, \tag{15a}$$

$$z(-1) = 1 + e^{-2}, \quad z(1) = 1 + e^{\frac{-2(1+\varepsilon)}{\varepsilon}}, \tag{15b}$$

where $\varepsilon = 10^{-4}$.

Example 7.2 *We solve a system of two singular differential equations posed on the interval* $0 < t \leq 1$,

$$z_1''(t) + \frac{3}{t}z_1'(t) = -\mu^2 z_2(t) - 2\gamma + z_1(t)z_2(t), \tag{16a}$$

$$z_2''(t) + \frac{3}{t}z_2'(t) = \mu^2 z_1(t) - \frac{1}{2}z_1^2(t), \tag{16b}$$

subject to the boundary conditions

$$z_1'(0) = 0, \quad z_2'(0) = 0, \quad z_1(1) = 0, \quad z_2'(1) + (1-\nu)z_2(1) = 0,$$

where $\nu = \frac{1}{3}$, $\mu^2 = 81$ *and* $\gamma = 1000$.

Example 7.3 *We demonstrate how to run the pathfollowing strategy by means of the following parameter dependent second order problem,*

$$z''(t) = \lambda z(t) \exp\left(\frac{8(1-z(t))}{1+\frac{2}{5}(1-z(t))}\right) \tag{17}$$

with the boundary conditions

$$z'(0) = 0, \quad z(1) = 1,$$

where $0 \leq t \leq 2$. *Here,* λ *is not an eigenvalue but a parameter. Moreover, the second boundary condition is posed within the interval of integration.*

Example 7.4 *Here, we treat an important special case of a mixed order system, a linear index-1 system of DAEs,*

$$t\begin{pmatrix} z_1'(t) \\ z_2'(t) \end{pmatrix} + B_{11}\begin{pmatrix} z_1(t) \\ z_2(t) \end{pmatrix} + B_{12}\begin{pmatrix} z_3(t) \\ z_4(t) \end{pmatrix} = \begin{pmatrix} g_1(t) \\ g_2(t) \end{pmatrix}, \tag{18a}$$

$$B_{21}\begin{pmatrix} z_1(t) \\ z_2(t) \end{pmatrix} + B_{22}\begin{pmatrix} z_3(t) \\ z_4(t) \end{pmatrix} = \begin{pmatrix} g_3(t) \\ g_4(t) \end{pmatrix}, \tag{18b}$$

posed on the interval $0 < t \leq 1$, *where* $B_{11}, B_{12}, B_{21}, B_{22} \in \mathbb{R}^{2\times 2}$, $g_1(t), g_2(t) \in C[0,1]$ *and the matrix* B_{22} *is nonsingular. The matrices are given as*

$$B_{11} = \begin{pmatrix} -11 & -18 \\ 12 & 19 \end{pmatrix}, \quad B_{12} = \begin{pmatrix} 3 & -1 \\ -2 & 1 \end{pmatrix}, \quad B_{21} = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}, \quad B_{22} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{5} \end{pmatrix},$$

and the right-hand side reads

$$\begin{pmatrix} g_1(t) \\ g_2(t) \end{pmatrix} = \begin{pmatrix} te^{mt}(m\sin(t) + cos(t)) - 12e^{mt}\sin(t) - 15\cos(mt) + 13.5 \\ -mt\sin(mt) + 13e^{mt}\sin(t) + 17\cos(mt) - 16 \end{pmatrix},$$
$$\begin{pmatrix} g_3(t) \\ g_4(t) \end{pmatrix} = \begin{pmatrix} e^{mt}\sin(t) + 2\cos(mt) - 2.5 \\ 2.2e^{mt}\sin(t) + 3\cos(mt) - 3 \end{pmatrix}.$$

The system (18) is subject to the following initial conditions,

$$2z_1(0) + 3z_2(0) = 0, \quad z_1(0) + z_2(0) = 0.$$

Examples 7.2 – 7.4 have been discussed in detail in [16].

Example 7.5 *In order to demonstrate how to use the GUI for the solution of an eigenvalue problem we consider a model describing the hydrogen atom which has been discussed in* [28]. *The problem is posed on the semi-infinite interval and reads,*

$$-z''(t) - \left(\frac{l(l+1)}{t^2} + \frac{\gamma}{t}\right) z(t) = \lambda z(t), \quad t \in (0, \infty), \tag{19a}$$
$$z(0) = 0, \quad z(\infty) = 0. \tag{19b}$$

For the test run we set $l = 0$ and $\gamma = 2$. The code embeds (19) into a BVP and automatically transforms it to a finite domain.

Example 7.6 *Finally, we consider the singular BVP posed on a semi-infinite interval which originates from hydrodynamics, cf.* [17], [18],

$$z''(t) + \frac{2}{t}z(t) = 4(z(t) + 1)z(t)(z(t) - 0.1), \quad t \in (0, \infty), \tag{20a}$$
$$z'(0) = 0, \quad z(\infty) = 0.1. \tag{20b}$$

7.4 Graphical User Interface – Tutorial

Getting Started

To run `bvpsuite`, start MATLAB, change to the folder where `bvpsuite` is installed, and type `bvpsuite` in the command line.

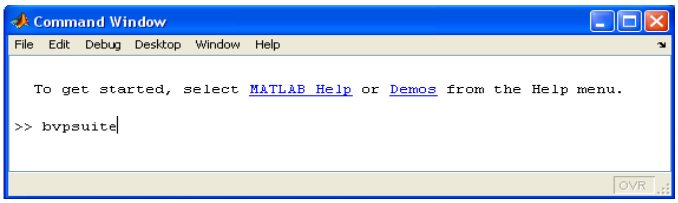

Figure 5: Start `bvpsuite`.

The GUI window shown in Figure 6 opens. All fields in this window will be described in detail in Section 7.5. Additionally, the question mark buttons can be pressed for detailed explanations of the input fields.

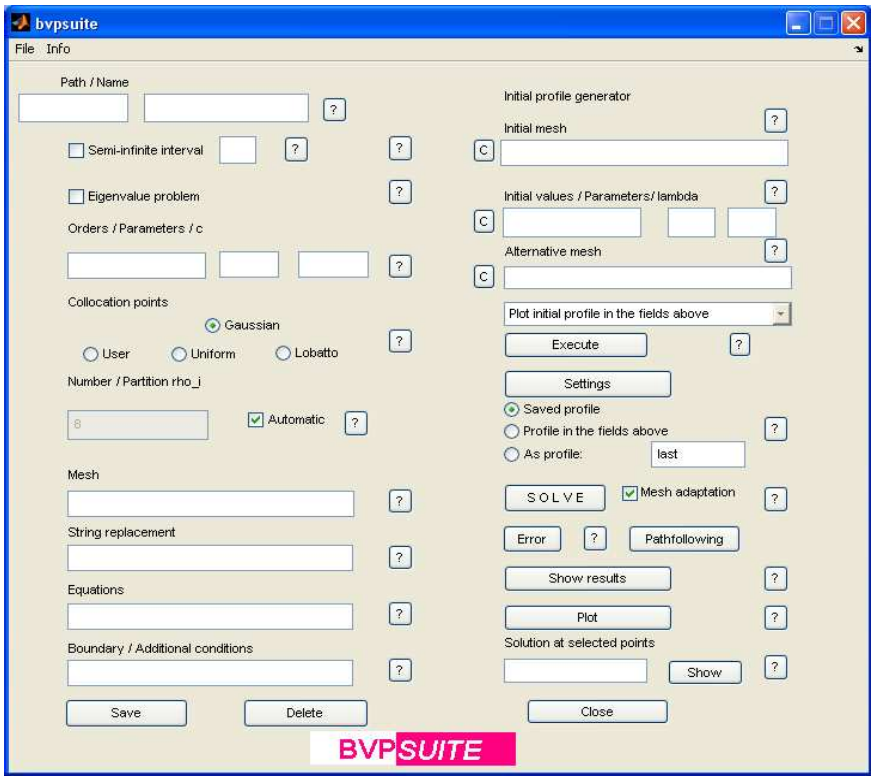

Figure 6: GUI window.

Example 7.1 This test example demonstrates how to find a numerical solution of problem (15) using the mesh adaptation procedure.

- Start choosing a path, e.g. `C:\MyFiles\Matlab\bvpsuite_files\examples_paper\` and the name of the file to be created, e.g. `ex1.m`.
- Define the orders of the solution components. Since there is only one component of second order, `[2]` has to be typed into the field 'Orders'. In the fields 'Parameters' and 'c' type `0` and `[]`. They can also stay empty.
- Leave the default settings for 'Collocation Points' – `Gaussian`, and 'Number / Partition rho_i' – `8` unchanged. Since 'Automatic' is ticked, the number of collocation points is set automatically, depending on the absolute tolerance parameter, see below and Example 7.2.
- Define 'Mesh' using a MATLAB row vector, for example of the form `a:h:b`, here `-1:0.02:1`, where `a` and `b` are the left and the right endpoint of the interval of integration, respectively, and `h` is the stepsize. In our case, the problem is solved on the interval $[a, b] = [-1, 1]$, and the initial mesh consisting of 101 points, i.e. 100 subintervals of length $h = 0.02$.
- In 'String replacement' type `eps=1e-4`. This defines the constant ε. Only strings that represent valid MATLAB variable names (see MATLAB reference docs) should be used for string replacement, with the following being reserved for direct use in the ODEs: zi $(z1, z2, \ldots)$, $zidj$ $(z1d3, z4d2 \ldots)$, $lambda$, pi $(p1, p2, \ldots)$, a, b, t. See further remarks in the manual.
- In 'Equations', the differential equation has to be specified as follows. Use only `zi` for the solution components, `eps*z1''+z1'-(1+eps)*z1=0`.

- Finally, specify the boundary conditions using `a` for the left and `b` for the right endpoint of the interval of integration, `z1(a)=1+exp(-2);z1(b)=1+exp(-2*(1+eps)/eps)`.

- The respective GUI is shown in Figure 7. Press 'Save' to store the data in `ex1.m`. Note that 'Mesh adaptation' is ticked as a default computational mode and therefore parameter values stored in 'Settings', in this case the default values, were used, cf. Figure 9.

- Press 'Solve'. When the computation is finished, an information window will appear. Accept it by pressing 'OK'.

- In the MATLAB window the output shown in Figure 8 can be found. The last line of this output indicates that the numerical solution has been found on the initial mesh with 101 points. Also, a few lines above, the output shows 99 points which is the number of the interior mesh points.

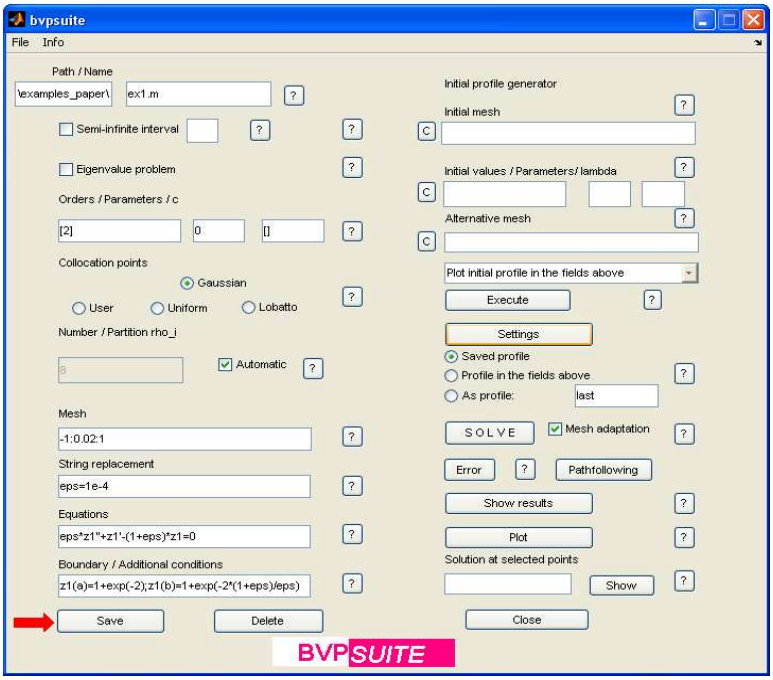

Figure 7: Example 7.1: The input data.

```
Solve problem by linear solver ...
Initialize error estimate ...
Solve problem by linear solver ...
Tolerance Factor: 5327.1199
N suggested: 516
N used:      99
---------------
Iteration 1
Updating density
Solve problem by linear solver ...
Initialize error estimate ...
Solve problem by linear solver ...
Tolerance Factor: 8925.1767
N suggested: 369
N used:      99
---------------
Iteration 2
Updating density
Solve problem by linear solver ...
Initialize error estimate ...
Solve problem by linear solver ...
Tolerance Factor: 0.083071
Tolerance on initial mesh satisfied
>>
```

Figure 8: Example 7.1: MATLAB output of the mesh adaptation.

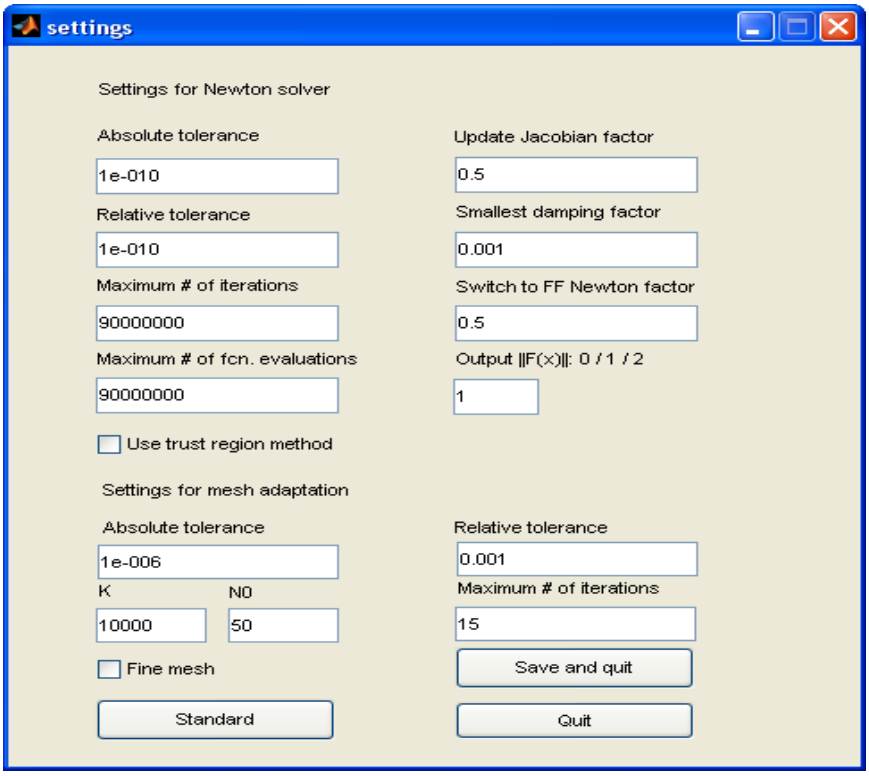

Figure 9: Example 7.1: Standard settings.

- Switch back to the GUI and press 'Error'. Close window 'Figure 2'. The window 'Figure 1' shows the estimate of the absolute error of the numerical solution found on the mesh with 101 points using the mesh adaptation procedure, see Figure 10. The maximal absolute error is approximately $1.5 \cdot 10^{-5}$.

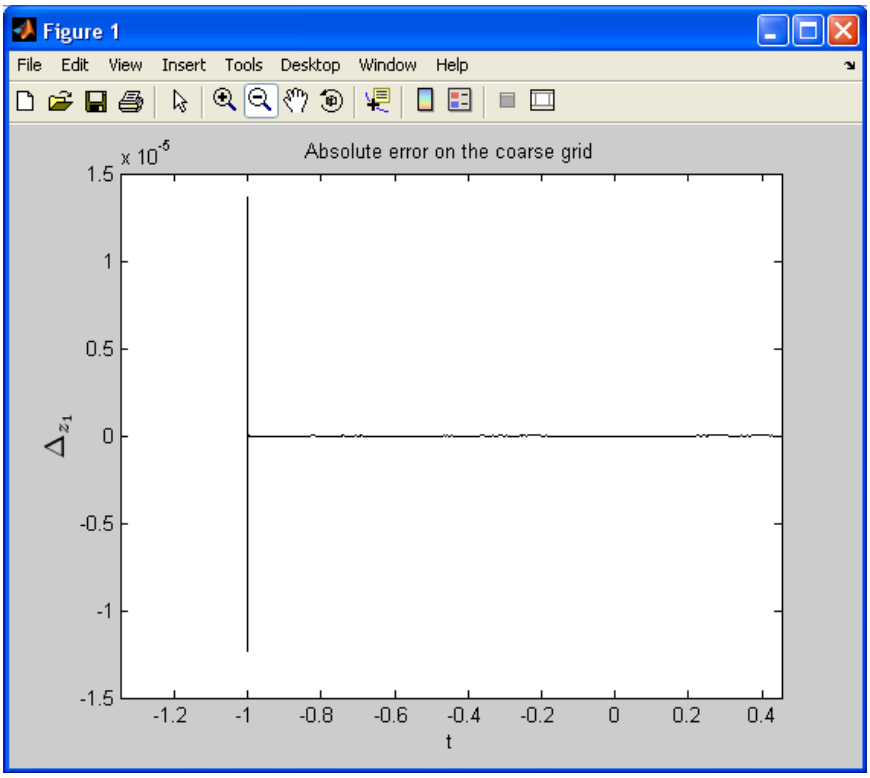

Figure 10: Example 7.1: Plot of the error estimate.

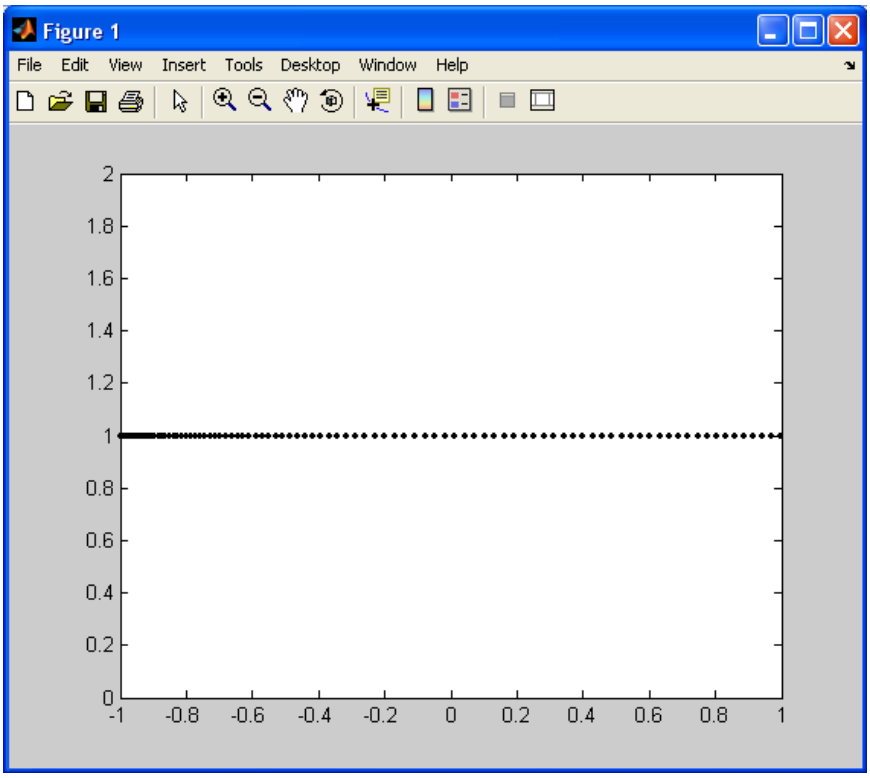

Figure 11: Example 7.1: Final adapted mesh.

- To plot the final grid, export the values of the numerical solution to the workspace by clicking 'Show results' and typing `plot(x1,1,'k.')` into the command line.
- To show the advantage of the mesh adaptation strategy, we compare the above run to the one on an equidistant mesh with the same number of mesh points. Remove the check mark for 'Mesh adaptation' by clicking on it and press 'Solve'. Now the problem is solved on the mesh specified in the field 'Mesh', i.e. with 101 equidistantly spaced mesh points.
- After the calculation was successfully completed, click 'OK' in the 'SUCCESS' window and change to the MATLAB command line. The numerical simulation produced the output shown in Figure 12.

```
Solve problem by linear solver ...
>>
```

Figure 12: Example 7.1: MATLAB output of the computation on the equidistant mesh.

- Close all windows with MATLAB figures and change back to the GUI. Press 'Error' and close window 'Figure 2'. The window 'Figure 1' shows the error on an equidistant mesh with 101 points, cf. Figure 13. Compare Figures 10 and 13. The errors related to the adapted and to the equidistant mesh with the same number of mesh points differ by 6 (!) orders of magnitude.

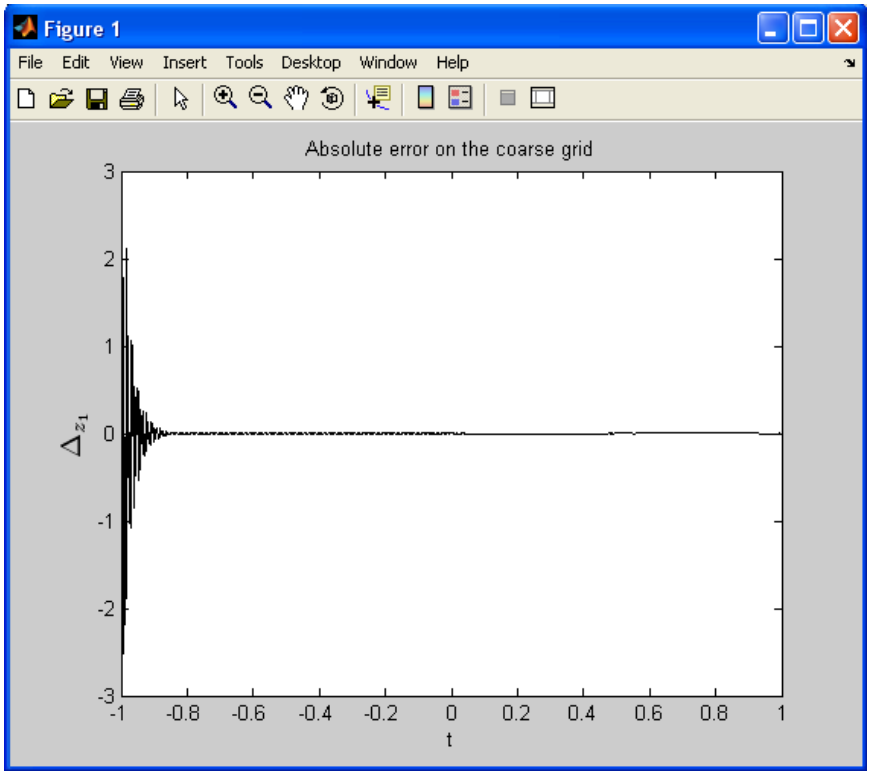

Figure 13: Example 7.1: Plot of the error estimate from the run using the equidistant mesh.

Remark: Choose among other options to display your results. 'Show results' imports the values of the solution on the grid, consisting of mesh points and collocation points, to the MATLAB workspace. (Internally, the mesh Δ, cf. (3), is denoted by `x1`, the collocation points by `tau`. Alternatively you can choose 'Selected Points' using a MATLAB row vector to display the solution at points of your choice. All results are stored in the file 'last.mat'. Using 'Plot' a graphical overview of all solution components on the grid is provided, cf. Figure 14.

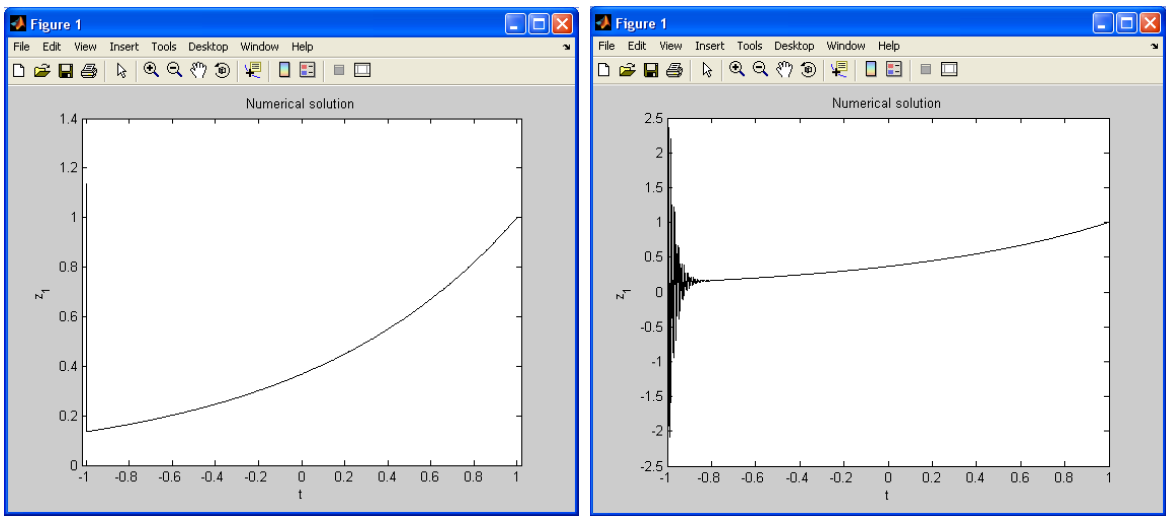

Figure 14: Example 7.1: Plots of the numerical solutions obtained with mesh adaptation (left) and on the equidistant mesh with the same number of points (right).

Example 7.2 This example demonstrates how to find a numerical solution of a problem (16), involving a system of two singular differential equations. This time we do not use the default values provided in 'Settings' but change them according to our needs.

- Start choosing a path, e.g. `C:\MyFiles\Matlab\bvpsuite_files\examples_paper\` and the name of the file, e.g. `ex2.m`.

- Press 'Settings' to choose the parameters. Type `0.0001` in the fields 'Absolute tolerance' and 'Relative tolerance', and `0` in the field 'N0'. This means that the program is run with the smallest number of grid points in the initial mesh, see below. Note that you have to change the value 'N0' back to the default value 50 to run another example under the standard settings. Press 'Save and quit' to exit the settings window, see Figure 15.

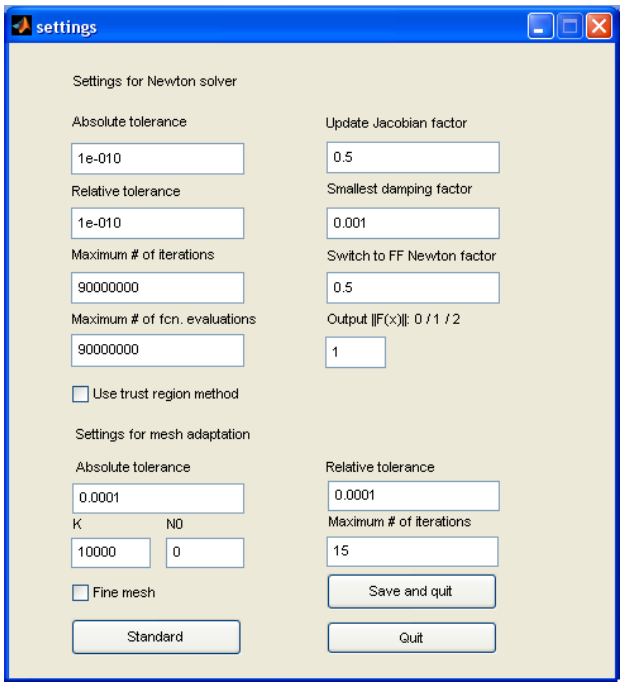

Figure 15: Example 7.2: Settings for the mesh adaptation.

- Define the orders of the solution components. Since both are of second order, `[2 2]` has to be typed into the field 'Orders'. In the fields 'Parameters' and 'c' type `0` and `[]`. They can also stay empty.

- For the collocation points take the default choice `Gaussian`. In the field 'Number' type `2`.

- Define the mesh in the field 'Mesh' using a MATLAB row vector, here `0:0.5:1`. This means that the number of points in the uniform initial mesh use for the calculations is $\max\{N, N0, 8\} = 8$, where $N = 3$ was specified by the user.

- In 'Equations', the differential equations have to be specified as follows, separated by semicolons. Use zi for the solution components,
  `z1''+3/t*z1'=-musquare*z2-2*gamma+z1*z2;`
  `z2''+3/t*z2'=musquare*z1-(1/2)*z1^2.`
  You can use any ASCII editor to type the equations and export them by 'copy and paste'.

- The given constants 'musquare' and 'gamma' are specified separately in 'String replacement': `musquare=81;gamma=1000;`.

- To define the boundary conditions type
  `z1'(a)=0;z2'(a)=0;z1(b)=0;z2'(b)+(2/3)*z2(b)=0`,
  where `a` and `b` are the endpoints of the interval.

- Make sure that the box 'Mesh adaptation' is ticked.

- Finally, press 'Save' to save the input data, cf. Figure 16, and 'SOLVE' to start the computations.

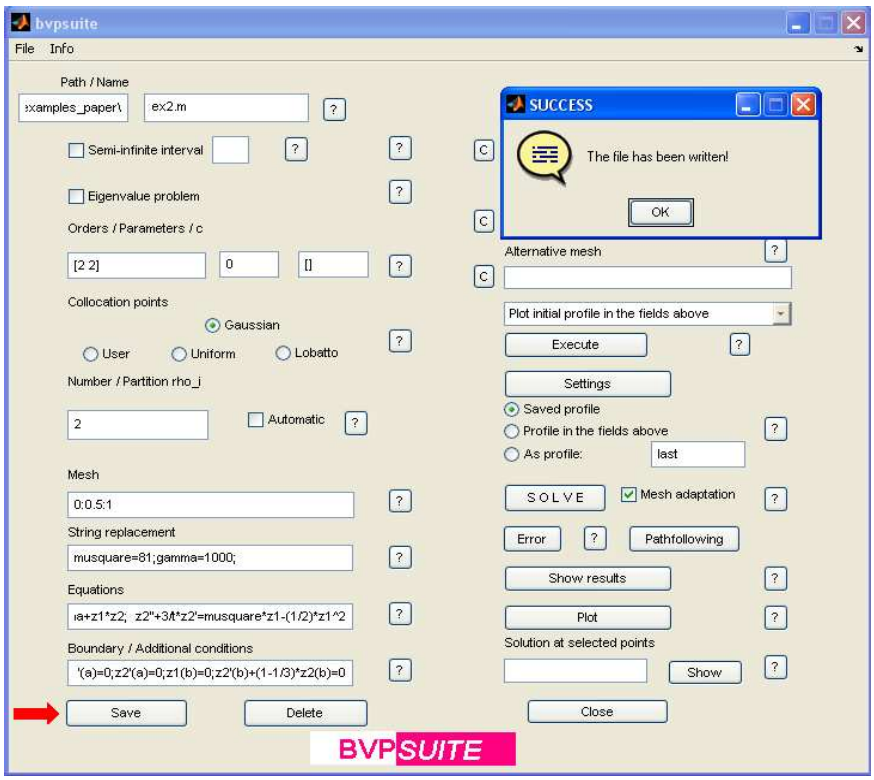

Figure 16: Example 7.2: The input data.

- To plot the results on the final grid export the values to the workspace by clicking 'Show results'. Press 'Plot' for the graph of the numerical solution. Type `plot(x1,1,'.','color','black')` in the command line to plot the mesh.

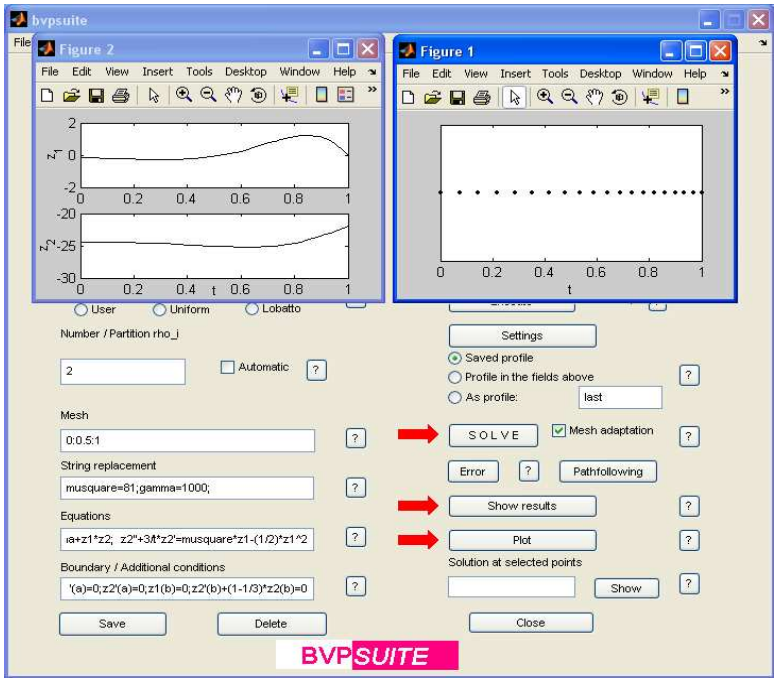

Figure 17: Example 7.2: Results for the run with mesh adaptation.

Remark: The code allows to specify an alternative mesh and the last solution as an initial profile. Input any mesh in 'Alternative mesh' and choose 'As profile: `last`'. After pressing 'SOLVE' you see that the Newton procedure needs less iterations than before to find the solution. You could use this realization also for stricter tolerance requirements.
Another option would be to use the 'Execute' button. After choosing the option 'Use last solution as initial profile' and pressing 'Execute', the initial profile fields will be filled in. The advantage is that now the user can modify the inputs. Use the 'Plot initial profile in the fields above' option of 'Execute' to see a graphical presentation of your inputs. An initial profile may be set manually, for details refer to Section 7.5. In this case, check 'Profile in the fields above' before pressing 'SOLVE'.

Example 7.3 This test example demonstrates how to solve a parameter dependent problem using the pathfollowing strategy implemented in `bvpsuite` and how to input boundary conditions posed at interior points of the interval.

- Since the differential equation is of second order, type `[2]` in the field 'Orders'.
- The parameter λ shall be denoted by `p1` in the specifications. The number of parameters, here 1, has to be defined in the field 'Parameters', `1`.
- The boundary conditions for this example are posed within the interval $[0, 2]$. Therefore their respective positions, denoted by 'c', have to be specified as a MATLAB row vector `[0 1]`.
  Instead of using `a` and `b` to describe the positions of the boundary conditions use `c1` and `c2`, because c_2 is not identical to b.
- Use 5 Gaussian points and choose the initial mesh `0:0.1:2`.
- The equation has to be typed as in Example 7.2 with λ denoted by `p1`.

- Since the parameter λ is a new variable, we need three boundary conditions for the first run (sum of orders plus number of parameters). In this case start with $\lambda = 0$, and in 'Boundary/Additional conditions' type `z1'(c1)=0;z1(c2)=1;p1=0`.
- Save the inputs (see Figure 18) and solve the problem by pressing the buttons 'Save'. Make sure that the box 'Mesh adaptation' is ticked and press 'SOLVE'.
- Create an empty directory to store all the information that the pathfollowing routine provides.
- Press the button 'Pathfollowing', and the pathfollowing window will open, cf. Figure 19. Use the last solution to start the pathfollowing routine.
- Press the lower 'Find' button and choose the directory you have created to save the path information.
- For this run, 110 steps are made along the solution/parameter path, each with stepsize 0.02. Start from the first point in the path, which means 'Initial index' `1`.
- To plot $z(0)$ against λ type `[1;0]` in the 'Pathdata matrix'. The data for the graph will be read from the pathdata matrix. For the detailed explanation of the pathdata matrix cf. Section 7.5.
- Press 'SOLVE'. When the computation has been finished use the button 'Show path' to show the graph.

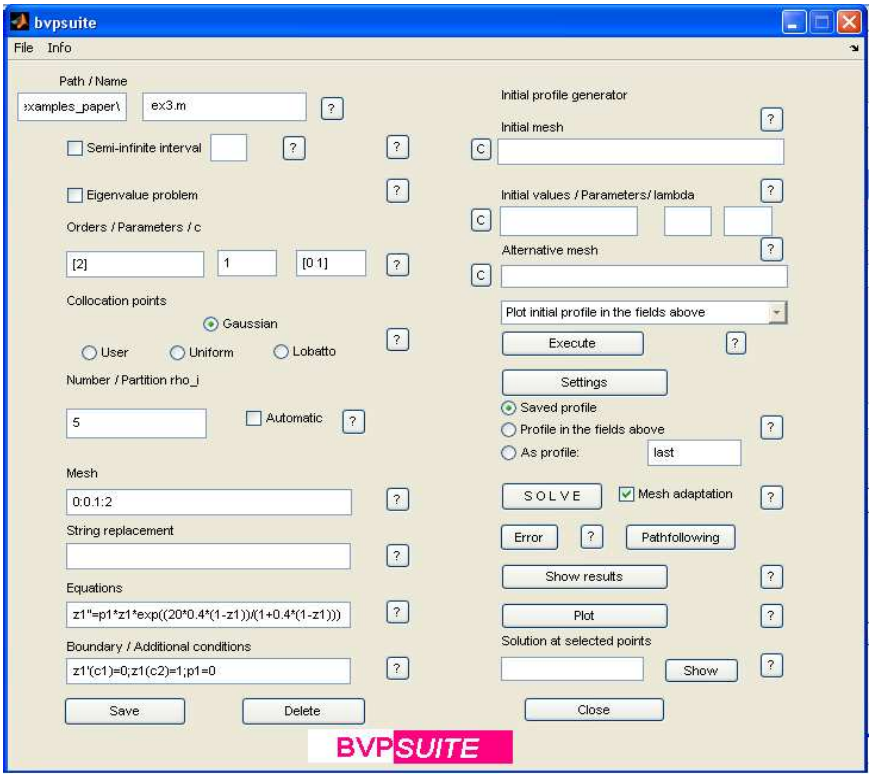

Figure 18: Example 7.3: Input data.

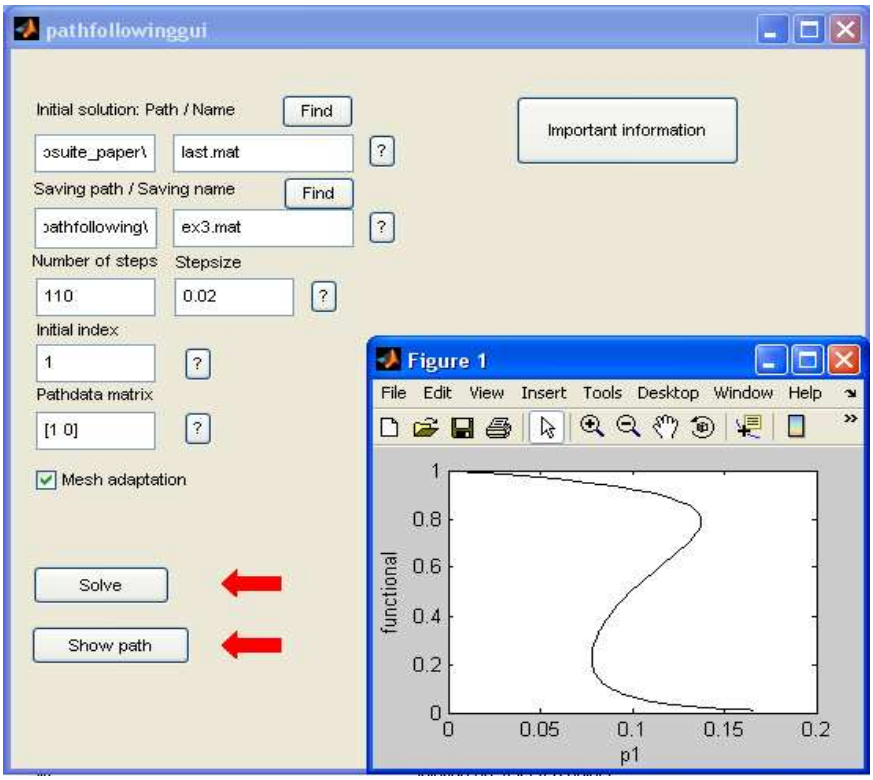

Figure 19: Example 7.3: GUI for the pathfollowing.

- To plot the solution of the i-th step load the mat-file with number i into the MATLAB workspace and type `plot(x1tau,valx1tau)`. For the corresponding value of λ load the file ex3.mat to the workspace and type `coeff(end)`. The evolution of the value of λ is saved in the variable `parametervalue`.

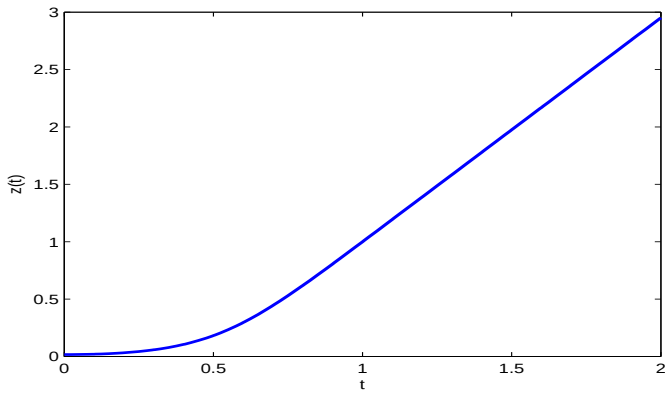

Figure 20: Example 7.3: Solution on the interval $[0, 2]$, $\lambda = 0.1495341$.

Example 7.4 This simulation deals with the solution of a system of DAEs.

- The solution of a system of DAEs does not require any special options, as DAEs are a special case of mixed order systems. In 'Orders' type `[1 1 0 0]`.
  Zeros indicate that the derivatives of z_3 and z_4 do not occur in the system of DAEs.

- Choose 5 collocation points, and then press 'Save'. Make sure that the box 'Mesh adaptation' is ticked and press 'SOLVE'. The input and results of the simulation are displayed in Figure 21.

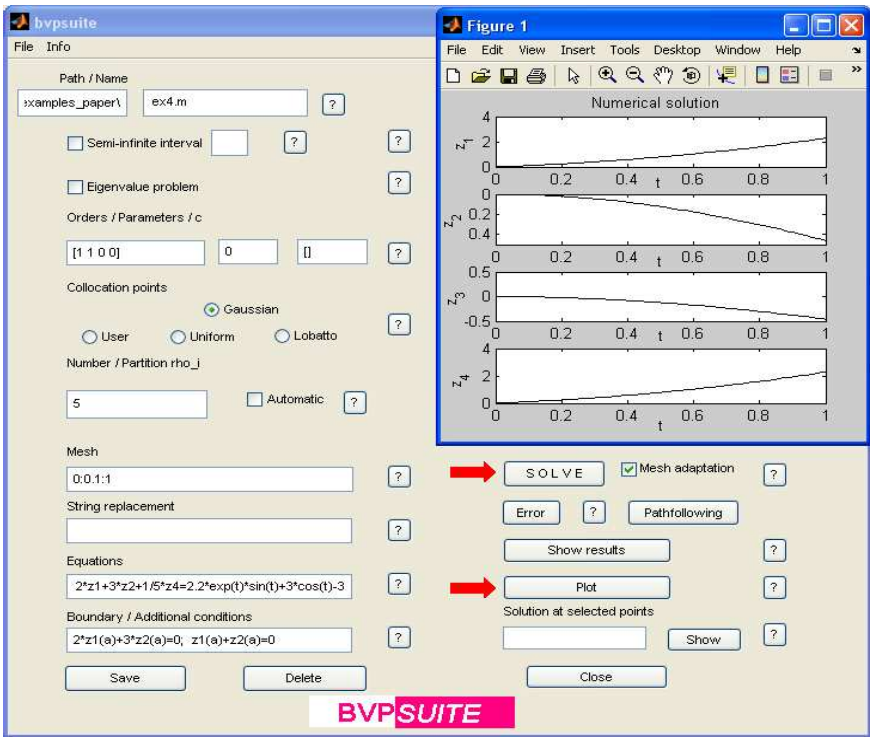

Figure 21: Example 7.4: Input data and results.

Example 7.5 The present example shows how the GUI can be used to solve an EVP posed on $[0, \infty)$. The inputs are shown in Figure 22. For problems posed on semi-infinite intervals the code performs an automatic transformation to a finite interval. Consequently, `bvpsuite` solves the problem on the finite domain first and provides the user with a back-transformation of the numerical solution to a truncated interval of arbitrary length.

- Tick the check box 'Semi-infinite interval'.
- Insert the left endpoint of the semi-infinite interval, which is `0`, in the edit field next to it.
- Tick the check box 'Eigenvalue problem'.
- Since the given differential equation is of order 2, insert the row vector `[2]` in the field 'Orders'.
- Choose 5 Gaussian 'Collocation points'.
- In the field 'Mesh' insert `20`, which corresponds to an equidistant initial mesh with 20 mesh points.
- Write the differential equation (19a) in the field 'Equations' and denote the eigenvalue λ by `lambda` `z1'' - 2/t*z1 = lambda*z1`. The solution has to be denoted by `z1`.
- For the 'Boundary conditions' write `z1(a)=0, z1(b)=0`. Note that you must not use the endpoints `0` and ∞ in the boundary conditions but write instead `a` and `b` for the endpoints of the interval.

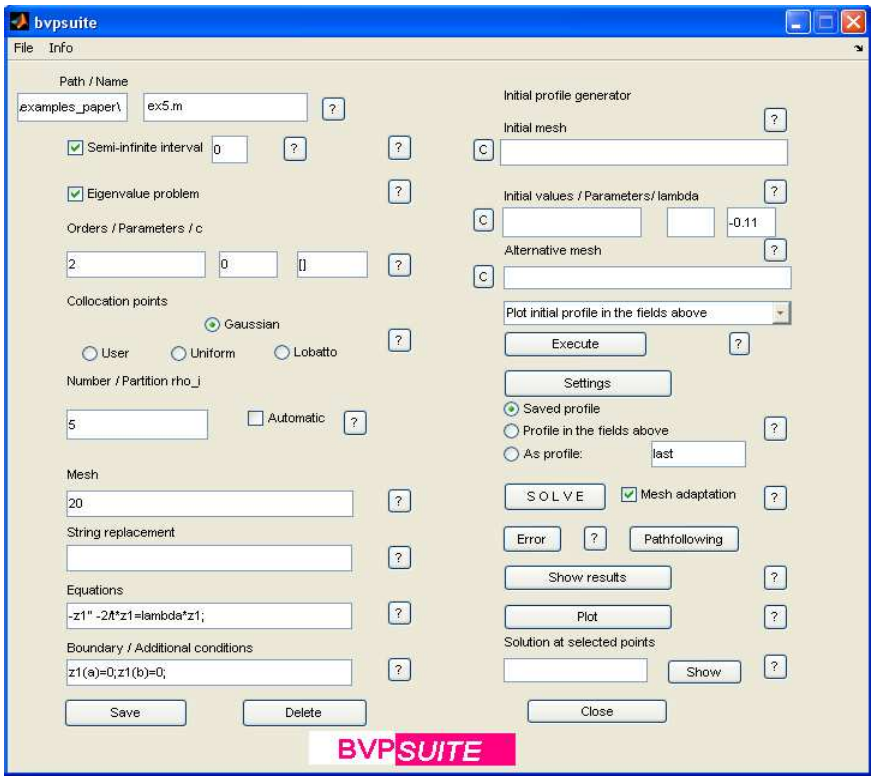

Figure 22: Example 7.5: Input data.

- No starting guess for the eigenfunction is provided, therefore leave the fields 'Initial mesh' and 'Initial values' empty.

- Insert the guess `-0.11` for the eigenvalue in the field called 'lambda'.

- Press the 'Settings' button to change the relative tolerance. Set the 'Relative tolerance' (cf. Figure 23) to 10^{-6} and 'N0' to `0`. This means that the number of points in the initial uniform mesh for the calculations is $\max\{N, N0, 8\} = 20$. Press 'Save and quit'.

- Before pressing the 'Save' button, the fields 'Path' and 'Name' have to be filled in to specify the location and the name for the automatically generated bvpfile. Press 'Save'.

- Make sure that the box 'Mesh adaptation' is ticked.

- Finally, press the 'SOLVE' button to start the computation. After the computation has finished several options to display the results are available.

- Press 'Plot' to obtain a plot of the solution. Since the problem is posed on a semi-infinite interval specify the plot range, see Figure 24.
  Alternatively, press 'Show results' to import the values of the solution on predefined points to the MATLAB workspace. The variable 'lambda' contains the computed eigenvalue and 'eigenfunction' is the numerical approximation for the eigenfunction.
  Moreover, selected points in form of a MATLAB row vector may be typed into the field under 'Solution at selected points'. Then, after pressing 'Show', the code provides the corresponding values of the numerical solution. All results are stored in 'last.mat'.

The mesh adaptation algorithm provided a final mesh with 39 points, cf. Figure 25. Note that the algorithm successfully placed most of the mesh points near the origin in order to efficiently satisfy the tolerance requirements. This corresponds to the solution behavior in this region very well. Figure 25 shows the approximation of the eigenfunction on the interval $[0, 1]$ in the uppermost graph. The central graph indicates the solution behavior on the interval $[1, \infty)$ transformed to $[0, 1]$. The approximation of the eigenfunction on the interval $[0, 80]$ and the back-transformed mesh are shown in Figure 26.

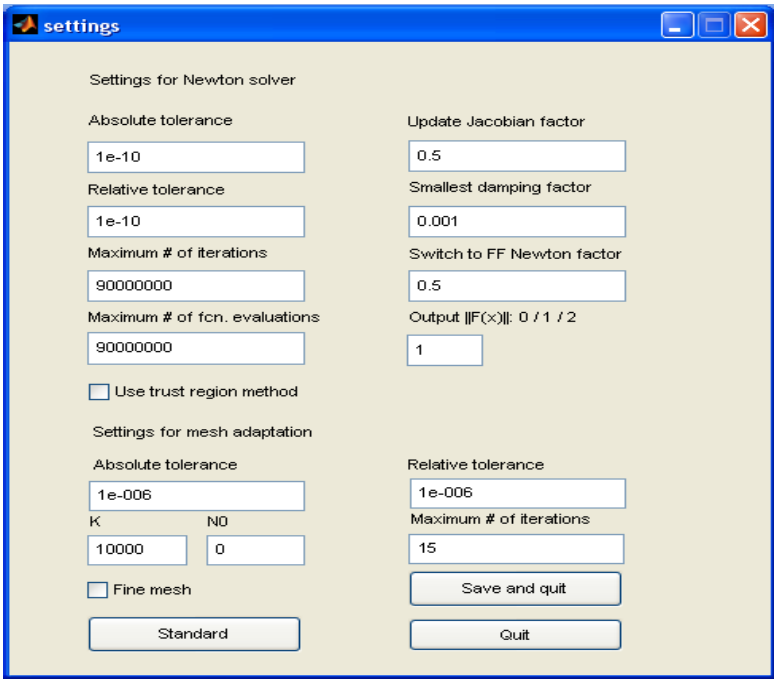

Figure 23: Example 7.5: Settings for the mesh adaptation.

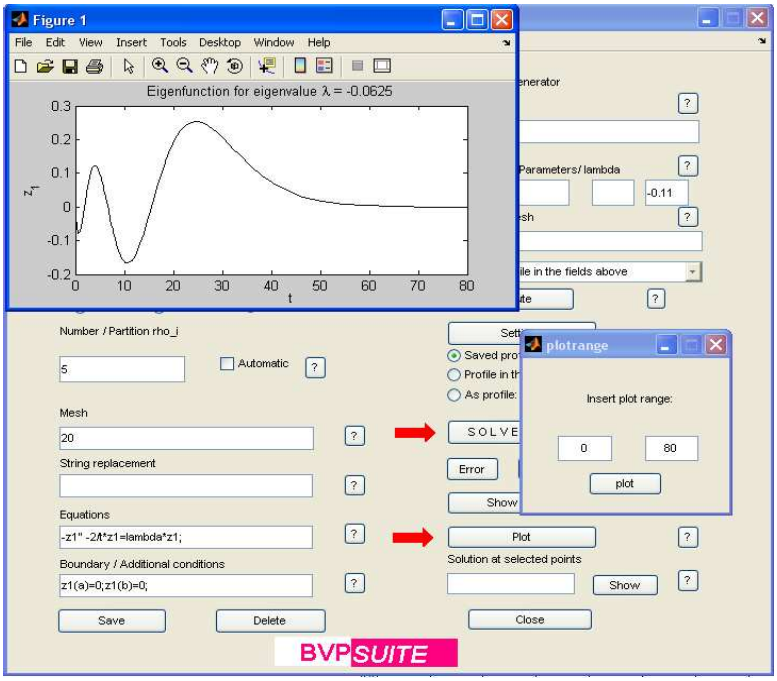

Figure 24: Example 7.5: Plot of the solution.

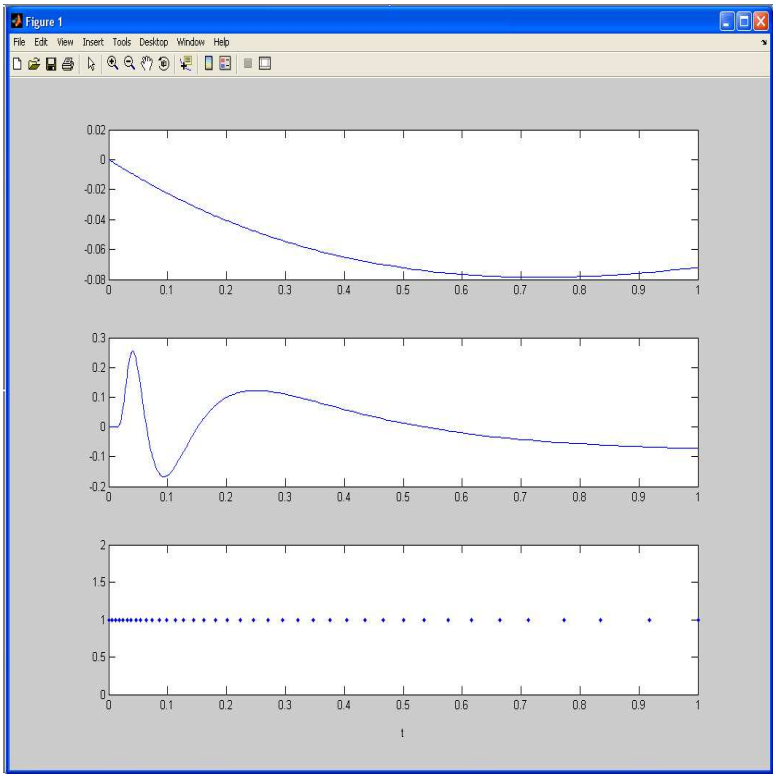

Figure 25: Example 7.5: Approximations of the solution z on $[0,1]$ (uppermost graph) and z on $[1,\infty)$ transformed to $[0,1]$ (central graph). The computations were carried out with 5 Gaussian points and $tol_a = tol_r = 10^{-6}$. The automatically selected mesh contains 39 points (lower graph).

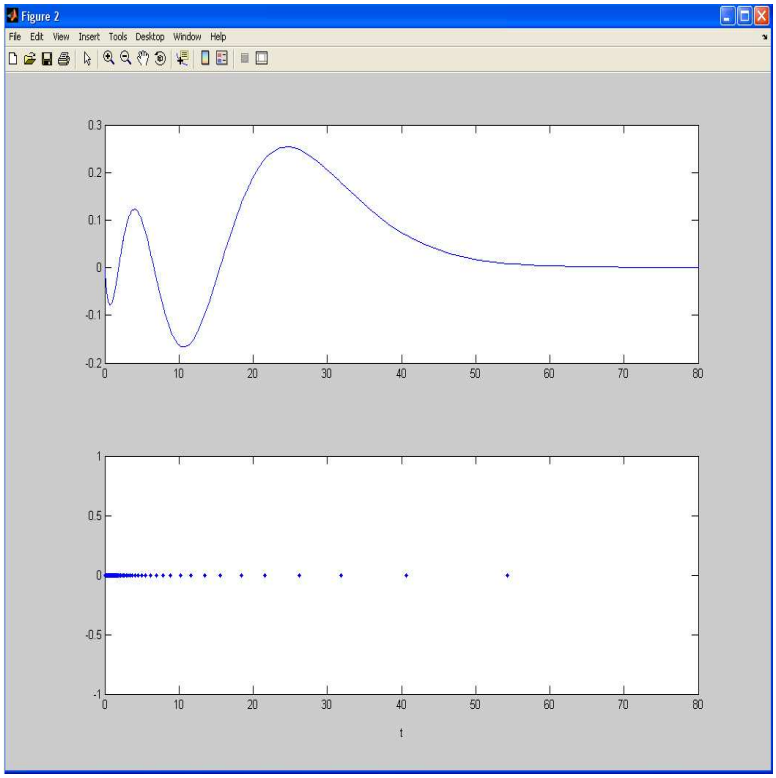

Figure 26: Example 7.5: Approximation of the eigenfunction and the back-transformed automatically selected mesh on a truncated interval $[0,80]$.

Example 7.6 Here, the BVP is posed on a semi-infinite interval. As already mentioned in Example 7.5 the code offers a module that automatically transforms the problem to a finite domain for the numerical

treatment. It is important to know that at infinity only Dirichlet boundary conditions are admissible for a successful automatic transformation, see Figure 27 for input.

- Tick the check box 'Semi-infinite interval'.
- Insert the left endpoint of the semi-infinite interval, which is `0`, in the edit field next to it.
- Since the given differential equation is of order 2, insert the row vector `[2]` in the field 'Orders'.
- In this example the 'Collocation points' are chosen to be Gaussian.
- For the 'Number' of collocation points choose `3`.
- In the field 'Mesh' insert `50` for the number of equidistantly spaced mesh points in the interval $[0, 1]$.
- Write the differential equation (20a), `z1''+2/t*z1' = 4*(z1+1)*z1*(z1-0.1)` in the field 'Equations'. The solution has to be denoted by `z1`, no other variable name is admissible.
- For the 'Boundary conditions' write `z1'(a)=0, z1(b)=0.1`.

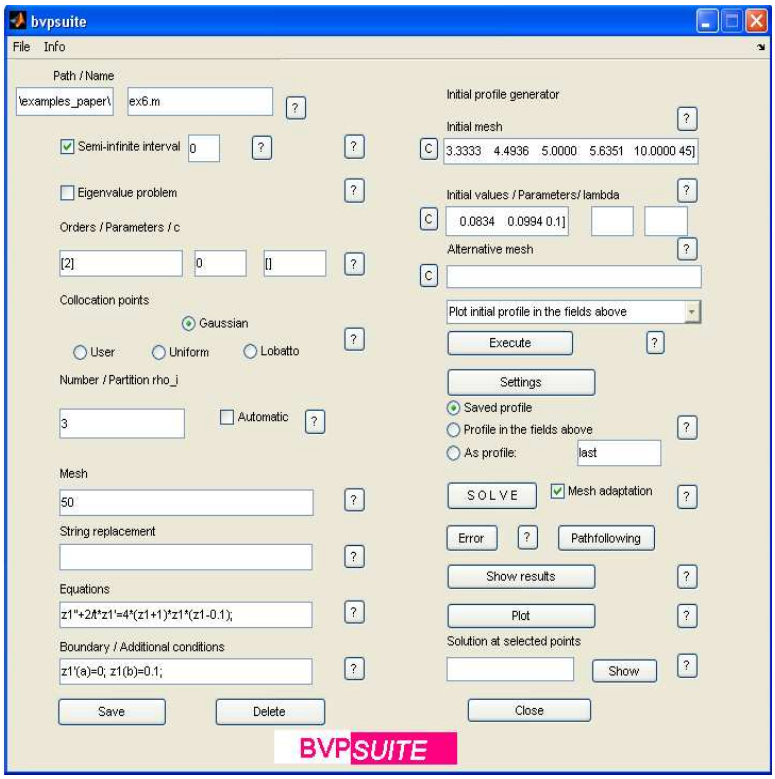

Figure 27: Example 7.6: Input data.

- For the 'Initial mesh' insert,

```
[ 0.0225    0.1000    0.1775    0.2000    0.2225    0.3000    0.3775
  0.4000    0.4225    0.5000    0.5775    0.6000    0.6225    0.7000
  0.7775    0.8000    0.8225    0.9000    0.9775    1.0000    1.0231
  1.1111    1.2157    1.2500    1.2862    1.4286    1.6063    1.6667
  1.7317    2.0000    2.3666    2.5000    2.6493    3.3333    4.4936
  5.0000    5.6351   10.0000    45].
```

- The corresponding 'Initial values' read,

```
[-0.3042   -0.3037   -0.3024   -0.3020   -0.3014   -0.2991   -0.2962
 -0.2952   -0.2942   -0.2902   -0.2857   -0.2842   -0.2828   -0.2773
 -0.2713   -0.2694   -0.2675   -0.2607   -0.2535   -0.2513   -0.2490
 -0.2400   -0.2286   -0.2248   -0.2207   -0.2041   -0.1825   -0.1750
 -0.1669   -0.1336   -0.0899   -0.0751   -0.0592    0.0007    0.0593
  0.0729    0.0834    0.0994    0.1].
```

- Before pressing the 'Save' button, fill in the fields 'Path' and 'Name' in order to specify the location and the name for the automatically generated bvpfile. Press 'Save'. Here, the default parameter values in 'Settings' are used and the mesh adaptation is utilized.

- Make sure that the box 'Mesh adaptation' is ticked.

- Press the 'SOLVE' button to start the computations, see Figure 28. Again there are several options to display the results. 'Show results' imports the numerical solution to the MATLAB workspace. The code automatically applies a back-transformation and provides the user with a solution approximation on a truncated interval of required length.

- Press 'Show results' in order to load the solution values into the workspace, cf. Figure 28. The solution values are stored in 'sol_infinite', 'tau_infinite' comprises the corresponding grid points, cf. Figure 29. Additionally, type 'load last.mat' in MATLAB's workspace to see the computed solution on the finite interval.

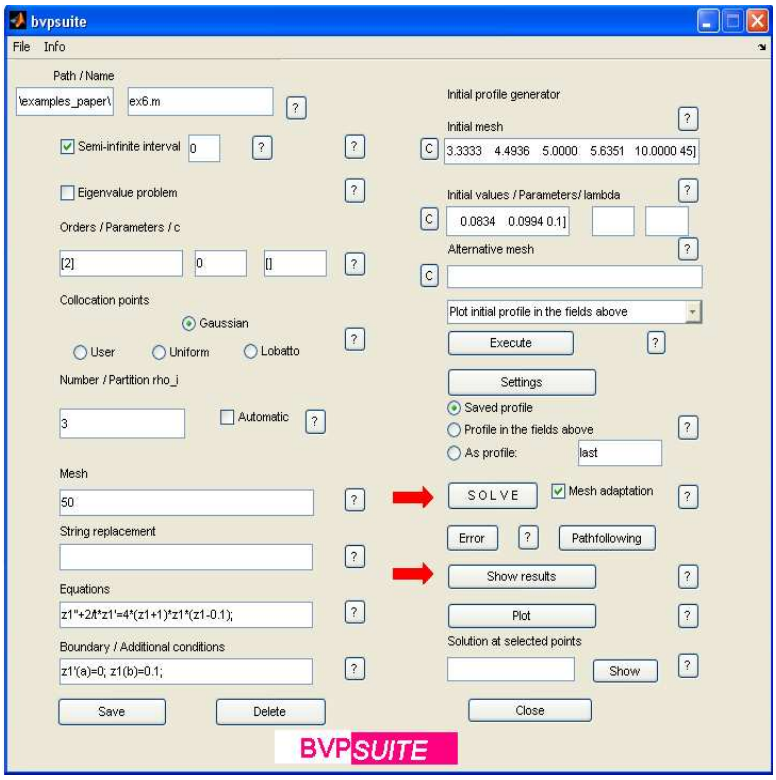

Figure 28: Example 7.6: 'Solve' and 'Show results'.

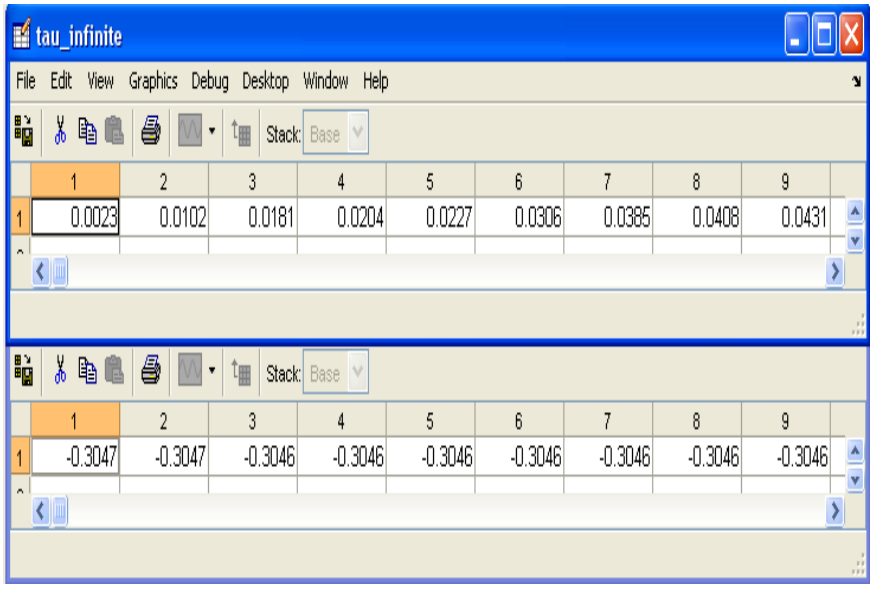

Figure 29: Example 7.6: Results.

7.5 Graphical User Interface – Controls

In this section all options for the GUI fields (some of which were already used in the examples) are now described in detail.

Field	Description
Path / Name	Choose the path and input the name of the m-file (input: *.m), which you would like to edit or last.log if you would like to see your last input. Use 'File → Open example' to find your file.
Semi-infinite interval	Tick this check box in case that your problem is posed on a semi-infinite interval $[a, \infty)$, $a \geq 0$ and type the left endpoint of the interval in the edit field to the right. A transformation of the problem is carried out in order to transform it to a finite interval. As a result, the code computes a solution on this finite domain. A graphical back-transformation allocates the solution on a truncated interval $[a, L]$. The right point L has to be specified by the user, cf. 'Plot' .
Eigenvalue problem	Tick this check box to declare that your example constitutes an eigenvalue problem.

Orders / Parameters / c	Choose the orders of the solution components in the form of a MATLAB row vector. (Be careful to choose the correct order!) E.g.: [3 4 2], if you have 3 equations and therefore 3 solution components with the highest derivative(z_1)=3, derivative(z_2)=4 and derivative(z_3)=2, respectively[3]. Specify the number of unknown parameters, denote them by `p1,p2,...`, in the equations. When solving a BVP posed on the interval $[a, b]$ with two-point boundary conditions specified at a and b, leave the field 'c' empty. Note that `bvpsuite` is also capable of solving systems where the boundary conditions are posed in the interior of the *finite* interval $[a, b]$. If you solve such a problem, specify a MATLAB row vector with the positions of the interior points. In the field 'Boundary / Additional conditions' specify `z2(c1)=...,z1(c3)=...,` instead of `z2(a)=...`. The variables a and b for the endpoints of the interval may no longer be used!
Collocation points	Choose the predefined collocation points, 'Gaussian', 'Lobatto' or 'Uniform' (equidistant), or choose 'User' to specify your own collocation points. In the latter case uncheck the check box 'Automatic'.
Number / Partition rho_i	The number of predefined collocation points (Gaussian, Lobatto, or Uniform) has to be specified in 'Number'. When the button 'User' is ticked, the user can specify the number and his own distribution of collocation points. When the number of collocation points is m, then one has to describe their distribution in the field 'Partition rho_i'. There, m values $0 \leq \rho_i \leq 1$, $i = 1, \ldots, m$ in the form of a MATLAB row vector have to be specified. They describe the positions of the collocation points in the interval $[0, 1]$.
Automatic	When the GUI window opens this check box is ticked. In such a case, the code automatically determines the number of collocation points to be used for the computations depending on the prescribed 'Absolute tolerance' in 'Settings'.

[3]Note that 'Orders' are not always identical to the orders of the differential equations.

Mesh	Specify the position of the mesh points, e.g. `[0 1.7 2.5 3]`. Note that `a` and `b`, the left and right endpoints of the interval have to be in the list. In this case `a=0` and `b=3`. You can also specify the mesh in the form `a:h:b`, where `h` is the step-size. In case that your problem is posed on a semi-infinite interval only insert the number of mesh points, e.g. `50`. This defines an equidistant mesh on the finite domain. Later, a graphical back-transformation of the solution is carried out by the code.
String replacement	You can define string replacements for parameters frequently occurring in the equations and boundary conditions. Syntax: `string_to_be_replaced_1 = replacing_string_1; string_to_be_replaced_2 = replacing_string_2; ...` E.g.: `m = 5*z1''-8;j=3;`. Be careful with string replacements: The strings `t,z,z1,z2,...,z1d2,z3d4,..., p1,p2,...,c1,c2,...,lambda`, and `a`, `b` must not be replaced. See also Example 7.1.
Equations	Specify the differential equations using variables $z_1(t), ..., z_n(t), z_1'(t), ..., z_n'(t), z_1''(t), ..., z_n''(t)$, written in the following form: `z1,...,zn,z1',...,zn',z1'',...,zn''`. Higher derivatives (≥ 3), e.g. $z^{(3)}$, have to be written as `z1d3`. The equations can be specified in the form `left side 1=right side 1; left side 2=right side 2`. E.g.: `z1''=z2;z2''=-z1`. If you want to solve an eigenvalue problem always denote the eigenvalue by `lambda`.
Boundary / Additional conditions	Specify the boundary and additional conditions separated by semicolons. The number of necessary conditions is equal to the sum of orders and parameters. Write $z_i(a) =:$ `zi(a)`, $z_i'(a) =:$ `zi'(a)`, $z_i(b) =:$ `zi(b)`, $z_i'(b) =:$ `zi'(b)`. For derivatives $k \geq 3$ type `zidk(a)`. E.g.: `z1(a) = 3;z2(a) = 0;z1(b) = 0;z3d4(b)=7...` In case of a finite interval $[a, b]$, specify the boundary conditions at the interior points in the following way: `z2(c1)=...;z1(c3)=...` Attention: Here, `a` and `b` must not be used!
Initial mesh	Specify the mesh points for the initial solution values in the form of a MATLAB row vector, e.g. `[3 4.5 7]`.

Initial values	Choose the values of the initial profile in the form of a MATLAB matrix. The rows in this matrix contain the guess for the solution values at the mesh points, e.g. for 3 mesh points and two solution components, `[1 4 2;3 6 8]`. If you leave the field 'Initial values' empty, the initial profile will be set to the constant function equal to 1. In this case 'Initial mesh' also has to be left empty. Additionally, for unknown parameters you can also specify their initial values in the corresponding field. For the solution of EVPs you can provide a guess for the eigenvalue ('lambda') and for the eigenfunction ('Initial values'). Alternatively, you can insert a value for 'lambda' only. Then the initial profile for the eigenfunction is set to the constant function equal to 1.
Alternative mesh	This mesh replaces the 'Mesh' on the left hand side of the GUI window if the option 'Profile in the fields above' or the option 'As profile' is ticked.
Execute	Executes the selected action in the drop down menu. 'Plot initial profile above' shows a graphical output of the profile that was defined in the fields above. 'Plot saved initial profile' shows the initial profile which was saved together with the file. 'Use last solution as initial profile' replaces the values above by the solution computed in the last run.
Settings	Opens the window for special settings. For details see 'Description of Settings'.
Radio button menu	'Saved Profile': Solve the problem using the saved initial profile. 'Profile in the fields above': Use the initial profile displayed in the fields above for the next run. 'As profile': Use solution values stored in 'last' (or in another *.mat file from the current directory) on the alternative mesh as initial guess for the next run. Attention: Do not change the number of collocation points when using the third option!
Solve / Mesh adaptation	Starts the computations. Mesh adaptation is the default computational mode and therefore the check box 'Mesh adaptation' is checked when the GUI window opens. The mesh adaptation algorithm is based on the residual and global error control.

Error	Plots a graphical output of the global discretization error and copies the values to the MATLAB workspace. The output variables are 'x1tau', 'x1tau2', and 'valerror', 'valerror2'. There are two error plots. In the `Figure 1`, the values in 'x1tau' and 'valerror' are associated with the coarse mesh and show the error estimate of the absolute error of the solution computed on the coarse mesh. In the `Figure 2`, the values 'x1tau2' and 'valerror2' are associated with the fine mesh (with twice as many points) and show the estimate of the absolute error of the solution computed on the fine mesh. These two meshes are used during the error estimation procedure. In case of a problem posed on the semi-infinite interval the output variables are 'tau_infinite', 'error_infinite', and 'tau_infinite_2', 'error_infinite_2'.
Pathfollowing	Opens the GUI window for pathfollowing. See 'Description of Pathfollowing' for details.
Show results	The results of the last calculation (stored in 'last.mat') are exported to the MATLAB workspace (overwriting existing variables with the same names). 'coeff': The coefficients with respect to the Runge-Kutta basis for the collocation polynomials and numerical values of the unknown parameters. 'parameter': Numerical values of the unknown parameters. 'x1': The mesh points. 'x1tau': The mesh points and the collocation points. 'valx1tau': The values of the numerical solution in the mesh and collocation points. 'parameters': Numerical approximation for the parameters.
Show results	In the context of problems posed on semi-infinite intervals, 'tau_infinite' is 'x1tau' transformed to a truncated interval. For an EVP posed on a semi-infinite domain the 'eigenfunction' is an approximation of the eigenfunction on 'tau_infinite', or otherwise, on 'x1tau'. For a BVP posed on a semi-infinite interval 'sol_infinite' is an approximation of its solution on 'tau_infinite'.
Plot	Plots the results stored in 'last.mat' on 'x1tau' or 'tau_infinite'.
Solution at selected points	Input for a MATLAB row vector containing additional, user selected points, for the values of the numerical approximation at those points.

Description of Settings

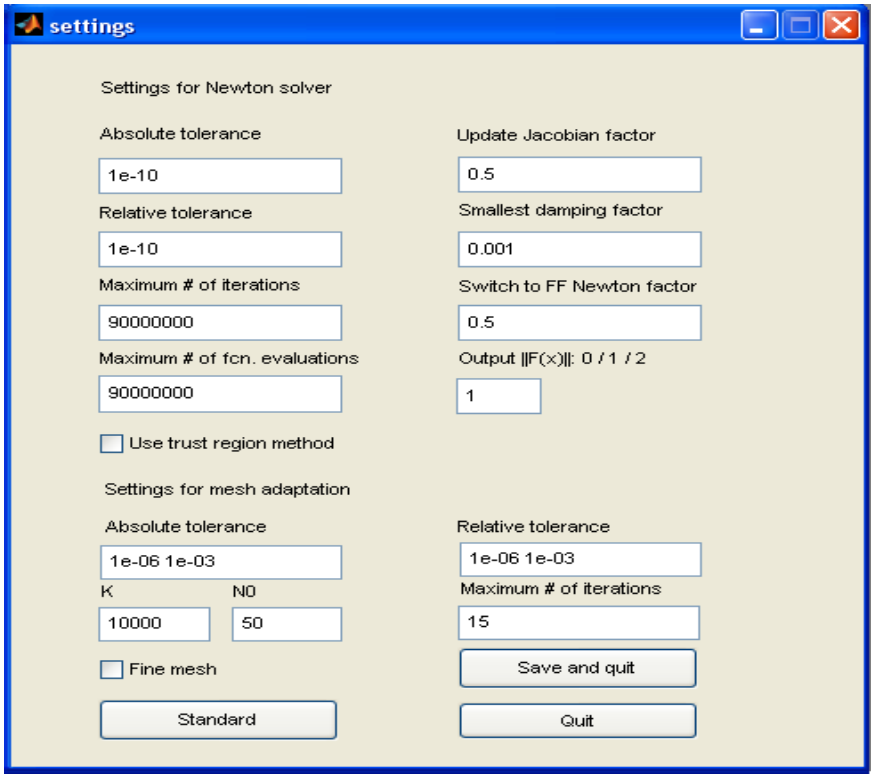

Figure 30: Settings for the Newton solver and the mesh adaptation.

Settings for the Newton solver	Set absolute and relative tolerance of the Newton solver. Decide if the norm of the residual, $\|\|F(x)\|\|$, of the nonlinear algebraic system $F(x) = 0$, for the current approximation shall be displayed: 0 ... nothing will be displayed, 1 ... display only on MATLAB prompt, 2 ... display on MATLAB prompt and in GUI. All other values of the settings window refer to the internal process of the Newton solver and we recommend not to change those values.

Settings for mesh adaptation	Choose the absolute and relative tolerance for the mesh adaptation algorithm. For different solution components different absolute and relative tolerance requirements can be specified, e.g. `1e-6 1e-3` in the field 'Absolute tolerance' means that for the first solution component we require the absolute tolerance 10^{-6} and for the second solution component 10^{-3} to be satisfied. For the default values of the tolerance parameters, equal for all solution components, see Figure 9. Use relative tolerance `0` if you want to control only absolute errors and vice versa. Choose the maximal number of iterations for the mesh adaptation algorithm, default value `15`. K is the maximal allowed ratio between the largest and the smallest stepsize in the mesh, default value `10000`. $N0$ is the total number of mesh points in the initial mesh, default `50`. For the first run an equidistant mesh with the number of points equal to $\max\{N, N0, 8\}$ is taken, where N is the number specified by the user via 'Mesh'. If $N0$ has been changed manually, it has to be set back to `50` by hand for the next run. Choosing the option 'Fine mesh' allows to slightly change the realization of the mesh adaptation algorithm. In this case tolerances will also be controlled on the fine mesh when they are not satisfied on the coarse mesh. According to our error estimation the fine mesh has twice as many mesh points as the coarse mesh.

Description of Pathfollowing, see Figure 19

General information	This routine solves a parameter dependent problem, i.e. the solver provides a solution for each value of the parameter, following a path in the solution/parameter space.
General information	Therefore, the routine should be called from an already calculated solution/parameter pair $(sol_0, p1_0)$, being a starting point in the path. In each following step the routine modifies both, the solution and the parameter. The value of the parameter $p1_0$ has to be specified at the very end of the field 'Boundary / Additional conditions', for example in the form `p1=7`.
Initial solution: Path / Name	Specify the *.mat file of your initial solution/parameter pair.

Saving path / Saving name	Specify the folder where the data, solutions and the corresponding values of parameters should be stored. Provide a name for a file containing a solution/parameter pair, for example solpar.mat. You will then find numbered solpari.mat files containing the name you have chosen. The solpar.mat file (without a number i) contains your path data specified below, see 'Pathdata matrix'.
Number of steps / Stepsize	In order to follow the path in the solution/parameter space, specify the number of steps and the constant stepsize for each step. It may be necessary, when going around turning points, to manually change the stepsize (decrease it). When the run prematurely terminates, because the stepsize was too large, you can return to any successfully computed previous point in the solution/parameter path.
Initial index	Input here the number of the solution/parameter pair you want to go back to and rerun. All following data points will be replaced by your new results.
Pathdata matrix	Input an $n \times 2$ matrix, with n being the number of paths you want to show, to save significant data from your pathfollowing strategy. To graphically show the results, input a MATLAB matrix in the field 'Pathdata matrix'. Specify the number of the solution component in the first column (0 represents the parameters). In the second column specify at which point the solution component has to be evaluated (it is not necessary to choose exactly a mesh point, it can be any point in the interval of integration). For example: `[3,0.4;0,2;1,0]` saves 3 paths, the first one shows $z_3(0.4)$ against the pathfollowing parameter $p1$, the second one shows another parameter denoted by $p2$ against $p1$, and the third $z_1(0)$ against $p1$.
Pathdata matrix	If you wish to express the maximum norm of a solution component against $p1$, use a column vector `[2;1]`. This vector means that the maximum norm, `1`, of the second solution component, `2`, will be plotted against $p1$.

7.6 MATLAB Command Line Syntax

In this section the command line calling syntax of the package is described in detail. We suggest to use the GUI instead of the command line, if you are just beginning to work with `bvpsuite`. Only advanced users with the need for extensive testing will benefit from the command line functions.

Overview and Function Calls

Two m-files constitute the basis for the command line functionality of `bvpsuite`. The file `bvpsuite_cl` is the main routine which manages which module of `bvpsuite` is used,. The file `options_cl` initializes input parameters which may be adjusted by the user. Make sure that your current bvpfile and the file `bvpsuite_cl.m` are in the same directory. To use `bvpsuite` via the command line, simply type

```
[outlog,optionsstruct]=bvpsuite_cl(bvpfile,optionsstruct)
```

where `bvpfile` is the name of the file containing the problem to be solved and `optionsstruct` is a MATLAB struct with input parameters for the call of the requested routine. The struct `outlog` is a MATLAB struct with the output data.
To set the default values of the input parameters in `optionsstruct`, type

```
options=options_cl('routine name')
```

The following table shows which settings for `'routine name'` are possible:

Field	Description
'run'	Starts the routine `run.m`. This routine calculates a numerical solution of the given problem.
'initialmesh'	Starts the routine `initialmesh.m`. This routine provides the coefficients with respect to the Runge-Kutta basis for the initial profile on the initial mesh. The discrete values of the initial guess provided by the user are first interpolated by a cubic spline[4]. Then the coefficients of this representation are recalculated to a Runge-Kutta basis. In the context of semi-infinite intervals, linear interpolation is used.
'meshadaptation'	Starts the routine `meshadaptation.m`. This routine calculates the numerical solution of the given problem up to the prescribed tolerance. To find such a solution, mesh adaptation is used.
'errorestimate'	Starts the routine `errorestimate.m`. This routine calculates the estimate for the global error of the collocation scheme using the mesh-halving technique.
'equations'	Starts the routine `equations.m`. The description of its output can be found in options for `'equations'` below.

Input/output arguments for the above routines are now specified.

[4]When the values at the interval endpoints are not specified by the user, extrapolation is automatically carried out.

Options for `'run'`

One of the most important routines of the `bvpsuite` package is `run.m`. It manages the solution process. The entries of the `optionsstruct` with mode `'run'` are given in the following table. The default settings are shown in []-brackets.

Input parameters	Description
plot	Possible settings: [0], 1. If set to 1, the solution will be plotted in an extra window.
x1	The initial mesh on which the problem has to be solved. If no mesh is provided, the default mesh will be read from the bvpfile. For the semi-infinite interval the number of equidistant points for the initial mesh.
start	Coefficients with respect to the Runge-Kutta basis for the initial profile on the initial mesh, cf. 'initialmesh'. If no initial profile is provided, the default values for the coefficients are zero.
bvpopt	The options for the Newton solver. If not provided, the options will be read from the bvpfile.
visible	Possible settings: [0], 1. If set to 1, information about the size of the residual in all iteration steps of the Newton procedure will be provided on the command line.
plotrange	Left and right endpoint of the plot interval in case of a problem posed on a semi-infinite interval.

To set one of the options above, type

```
optionsstruct.name_of_option=desired_setting
```

After calling `bvpsuite_cl`, the output of the routine `run.m` is saved into the MATLAB struct `outlog`. It provides the following information.

Output data	Description
coeff	The coefficients of the collocation polynomial (numerical solution) with respect to the Runge-Kutta basis.
x1	The mesh on which the numerical solution was computed.
valx1	The values of the solution at the mesh points of x1.
x1tau	The grid on which the solution was computed.
valx1tau	The values of the solution on the grid x1tau.
tau_infinite	The grid x1tau transformed to a truncated interval.

sol_infinite	The problem is posed on a semi-infinite interval. The solution is first computed on the grid x1tau of a finite domain and then back-transformed to a large truncated interval. Its values are stored in sol_infinite.
lambda	The numerical approximation for an eigenvalue.
eigenfunction	The numerical approximation for an eigenfunction on the grid tau_infinite.

It is important to note that the above output data can be used to start the numerical solution of another (related) problem or even for the same problem. It makes sense to use `x1` and `coeff` from `outlog` as initial parameters in `optionsstruct` for another call of the same problem, but with stricter tolerance requirement.

Options for `'initialmesh'`

In `initialmesh` the following assignment is realized. Given an initial solution profile on a starting mesh, the routine provides the coefficients for the representation of this initial profile with respect to the Runge-Kutta basis for an arbitrary mesh.

Input parameters	Description
x0	The mesh on which the values of the initial profile are provided.
valx0	The values of the profile on the mesh x0.
x1	The mesh that shall be used for the numerical computations. It is also allowed to use the same mesh `x0`.
par	Starting guesses for the unknown parameters.
lambda	Initial guess for the eigenvalue.

The output of the `initialmesh` routine is quite simple. It consists only of the mesh `x1` and the corresponding coefficients `coeff`.

Output data	Description
x1	The mesh on which the coefficients are given.
coeff	The coefficients corresponding to the initial profile on x1, cf. 'initialmesh'.

Options for `'meshadaptation'`

In `meshadaptation.m` we try to calculate a numerical solution satisfying the prescribed tolerance requirements. This is done by adjusting the mesh until the error estimate indicates that the desired accuracy has been reached. The final mesh and the numerical solution are then returned to the user. The options for the routine are listed below.

Input parameters	Description
aTOL	The prescribed absolute tolerance; [10^{-6}].
rTOL	The prescribed relative tolerance; [10^{-3}].
K	The maximal ratio between the largest and the smallest step-size in the mesh; [100].
plot	Possible settings: [0], 1. If set to 1, the solution will be plotted in an extra window.
x1	The initial mesh on which the problem has to be solved. If no mesh is provided, the default mesh will be read from the bvpfile.
start	The coefficients of the initial solution profile with respect to the Runge-Kutta basis, cf. 'initialmesh'. If no initial profile is provided, the default values for the coefficients are zero.
bvpopt	The options for the Newton solver. If not provided, the options will be read from the bvpfile.
visible	Possible settings: [0], 1. If set to 1, information about the size of the residual in all iteration steps of the Newton procedure will be provided on the command line.
maxiter	Maximal number of iterations in the mesh adaptation strategy (correcting the mesh density and the number of mesh points) before the routine terminates with an error message; [10].
finemesh	Possible settings: [0], 1. If set to 1, the realization of the mesh adaptation algorithm is slightly changed. In this case tolerances will also be controlled on the fine mesh. According to our error estimation strategy, the fine mesh has twice as many mesh points as the coarse mesh.

The results of the `meshadaptation.m` routine can be accessed using the `outlog` containing the following data.

Output data	Description
coeff	The coefficients of the collocation polynomial (numerical solution) with respect to the Runge-Kutta basis.
x1	The mesh on which the numerical solution was found.
valx1	The values of the solution in the mesh points of x1.
x1tau	The grid, mesh points and collocation points on which the solution was computed.
valx1tau	The values of the solution on the grid x1tau.
error1	The error estimate on the mesh x1.

fine	Boolean; Indicates if `finemesh` was used to satisfy the tolerance requirements.
tau_infinite	The grid x1tau transformed to a large interval.
sol_infinite	The problem is posed on a semi-infinite interval. The solution is first computed on the grid x1tau of a finite domain and then back-transformed to a large truncated interval. Its values are stored in sol_infinite.
lambda	The numerical approximation for an eigenvalue.
eigenfunction	The numerical approximation for an eigenfunction on the grid tau_infinite in case that the EVP is posed on a semi-infinite interval, otherwise on the grid x1tau.

Options for `'errorestimate'`

The routine `errorestimate.m` is used to estimate the global error of the collocation solution after it has been computed by `run.m`.

Input parameters	Description
coeff	The coefficients of the collocation polynomial (numerical solution) with respect to the Runge-Kutta basis.
plot	Possible settings: [0], 1. If set to 1, the error estimate will be plotted in an extra window.
bvpopt	The options for the Newton solver. If not provided, the options will be read from the bvpfile.
x1	The initial mesh on which the problem has to be solved. If no mesh is provided, the default mesh will be read from the bvpfile.
visible	Possible settings: [0], 1. If set to 1, information about the size of the residual in all iteration steps of the Newton procedure will be provided.

The output arguments from `errorestimate.m` provide the error estimate on the coarse mesh as well as on the fine mesh. The full list of the output data can be found below.

Output data	Description
x1tau	The coarse grid, mesh and collocation points.
valerror	The error of the solution on the coarse grid x1tau.
maxerrorvalx0	Maximum norm of valerror.
x1tau_2	The fine grid, mesh and collocation points.
coeff_2	The coefficients of the collocation polynomial with respect to the Runge-Kutta basis for the grid x1tau_2.

val_2x1tau	The values on the coarse grid of the numerical solution computed on the fine grid.
val_2x1_2	The values at the mesh points in the fine mesh of the numerical solution computed on the fine grid.
valerror2	The error of the solution computed on the fine grid given on the coarse grid.
tau_infinite	The grid x1tau transformed to a truncated interval. Applies for BVPs posed on semi-infinite intervals.
tau_infinite_2	The grid x1tau_2 transformed to a truncated interval. Applies for BVPs posed on semi-infinite intervals.
error_infinite	valerror transformed to the large, truncated interval. Applies for BVPs posed on semi-infinite intervals.
error_infinite_2	valerror2 transformed to the large, truncated interval. Applies for BVPs posed on semi-infinite intervals.
sol_infinite	The problem is posed on a semi-infinite interval. The solution is first computed on the grid x1tau of a finite domain and then back-transformed to a large truncated interval. Its values are stored in sol_infinite.
lambda	The numerical approximation for an eigenvalue.
eigenfunction	The numerical approximation for an eigenfunction. For EVPs posed on semi-infinite intervals, 'eigenfunction' is given on tau_infinite, otherwise, for EVPs posed on a finite domain, the 'eigenfunction' is given on x1tau.

Options for `'equations'`

The routine `equations.m` is used to set up the equations for the Newton Solver.

Input parameter	Description
module	Compulsory; Determines what you would like to do with this routine. 'F' is the nonlinear algebraic system to be solved by the Newton iteration. 'DF' is the Jacobian of 'F'.

module	'residual' is obtained by substituting the collocation polynomial into the system of ODEs. It is called by meshadaptation.m, see Section 4. On return the values of the residual are available. 'polynomial' transforms the polynomial coefficients with respect to the Runge-Kutta basis to the ones with respect to the monomial basis. 'valx1' calculates the solution values in the mesh points x1 (without collocation points). 'valx1tau' computes the solution values on the grid x1tau (mesh points and collocation points). 'basispolynomials0' is the output of the Runge-Kutta basis coefficients for an algebraic constraint, order 0. 'rho' on output specifies the position of the collocation points on the interval $[0, 1]$. 'value' returns values of the solution on a user-specified vector of points. 'diffvalue' is the same for the first derivative. 'tau' returns the positions of all collocation points.
coeff	Compulsory; The coefficients with respect to the Runge-Kutta basis for the collocation polynomials and numerical values of the unknown parameters.
x1	Compulsory; The mesh points.
outputpoints	Optional; The vector of arbitrary output points for the 'module' options 'outputpoints' and 'residual'.
linear	Optional, Boolean; Determines if the problem is solved by the linear or nonlinear solver.

Use of the Package `bvpsuite` via the Command Line

Here we indicate how the command line tool can be used. We calculate a solution for the bvpfile `ex1.m` and then provide the error estimate using only two command line functions. Note that the bvpfile and the command line routines have either to be in the same directory or a relative path has to be used instead.

- `options=options_cl('run');` % saves all necessary input parameters for run into options;
- `options.visible=1` % changes the setting for visible from default 0 to 1;
- `[sol,optsol]=bvpsuite_cl('ex1',options)` % calls the routine run for ex1.m with optionsstruct options, the output is saved in `sol`, the options are saved in optsol;
- `options=options_cl('errorestimate');` % saves all necessary input parameters for errorestimate into options;
- `options.x1=sol.x1;` % replaces the mesh stored in options.x1 by the mesh stored in sol.x1;

- `options.coeff=sol.coeff;` % replaces the solution coefficients stored in options.coeff by the coefficients stored in sol.coeff;
- `[err,opterr]=bvpsuite_cl('ex1',options)` % calls the routine errorestimate for ex1.m with the solution provided by run;

The code example above illustrates how simple it is to control the whole package `bvpsuite` using only two major functions. Note that the options for the solution process are saved in `optsol` and the options for the error estimation are saved in `opterr`. Therefore, they could be easily reused. It is also not necessary to use `options_cl`. Only the mode-option has to be set and the routine `bvpsuite_cl` will take care of the rest. We stress that this tool is of use for advanced and experienced users who know how to handle possible difficulties arising when not all input parameters are set in a correct way.

7.7 The bvpfile

This section describes the general form of the bvpfiles. Those files are generated automatically by the GUI. It is not recommended to write a bvpfile without using the GUI, but experienced users can modify the bvpfiles. In general, the inputs correspond to the inputs of the GUI.

Example 7.2

```
% Function call and internal input parameters for the code
function [ret]=ex1(nr,D,R,t,z,za,zb,D1,D2,p,Dpgl,Dppar,DpRgl,DpRpar,zc)
% Check the number of input parameters
if (nargin<15) zc=[]; if (nargin<14) DpRpar=[]; if (nargin<13) DpRgl=[];
if (nargin<12) Dppar=[]; if (nargin<11) Dpgl=[]; if (nargin<10) p=[];
if (nargin<9) D2=[]; if (nargin<8) D1=[]; if (nargin<7) zb=[]; if (nargin<6)
za=[]; if (nargin<5) z=[]; if (nargin<4) t=[]; if (nargin<3) R=[];
if (nargin<2) D=[];end;end;end;end;end;end;end;end;end;end;end;end;end;end;
ret=intern(nr,D,R,t,z,za,zb,D1,D2,p,Dpgl,Dppar,DpRgl,DpRpar,zc);
% Internal function call
function ret=intern(nr,D,R,t,z,za,zb,D1,D2,p,Dpgl,Dppar,DpRgl,DpRpar,zc)
switch nr
     case 'x0'
% Define length of solution vector of the system
         for j=1:(length(intern('x1'))-1)*(intern('n')*intern('m')
                  +sum(intern('ordnung')))+intern('parameter')
             u(j,1)=1;
         end
         ret=u;
% Define order of differential equations
     case 'ordnung'
```

```
        ret=[2  2];
% Define number of parameters
    case 'parameter'
        ret=0;
% Define positions for boundary conditions posed in the interior of the interval
    case 'c'
        ret=[];
% Define the number of equations
    case 'n'
        ret=2;
% Define whether the problem is defined on an infinite domain
    case 'Infinite'
        ret=false;
% Define whether the problem is an EVP
    case 'EVP'
        ret=false;
% Left endpoint of the semi-infinite interval, [a,∞), a ≥ 0
    case 'Endpoint'
        ret=[];
% Standard collocation points (Gaussian, Lobatto)
    case 'standard'
        ret=true;
% Vector of collocation points (in this case standard), 1 means Gaussian (2 means Lobatto), 5 specifies
% the number of collocation points
    case 'rho'
        ret=[1 5];
% Mesh vector without collocation points
    case 'x1'
        ret=0:0.1:1;
% Equations (z(1,3)=z1", z(1,2)=z1', z(1,1)=z1)
    case 'g'
        ret=[z(1,3)+3/t*z(1,2)-(-(81)*z(2,1)-2*(1000)+z(1,1)*z(2,1));
            z(2,3)+3/t*z(2,2)-((81)*z(1,1)-(1/2)*z(1,1)^2)];
% Derivatives of the above nonlinear equations with respect to ...
    case 'Dg'
        switch D
         case 1
% ... [z(1,1) z(2,1); z(1,1) z(2,1)]
                ret=[-z(2,1) 81-z(1,1) ;-81+z(1,1) 0 ;];
         case 2
```

```
% ... [z(1,2) z(2,2); z(1,2) z(2,2)]
                ret=[3/t 0 ;0 3/t ;];
          case 3
% ... [z(1,3) z(2,3); z(1,3) z(2,3)]
                ret=[1 0 ;0 1 ;];
          end
% Derivatives with respect to the parameters
     case 'Dpg'
          ret=[0;0;];
% Additional conditions
% za(component,derivative)
% zb(component,derivative)
% Boundary conditions
     case 'R'
          ret=[za(1,2)-(0);za(2,2)-(0);zb(1,1)-(0);
               zb(2,2)+(1-1/3)*zb(2,1)-(0)];
% Derivatives of the boundary conditions
     case 'DR'
          switch R
% First boundary condition with respect to ...
             case 1
                switch D
% [za(1,1),za(2,1)]
                     case 'a1'
                          ret=[0 0 ;];
% [za(1,2),za(2,2)]
                     case 'a2'
                          ret=[1 0 ;];
% [zb(1,1),zb(2,1)]
                     case 'b1'
                          ret=[0 0 ;];
% [zb(1,2),zb(2,2)]
                     case 'b2'
                          ret=[0 0 ;];
                 end
% Second boundary condition with respect to ...
             case 2
                switch D
                     case 'a1'
                          ret=[0 0 ;];
```

```
                    case 'a2'
% [za(1,2),za(2,2)]
                        ret=[0 1 ;];
                    case 'b1'
                        ret=[0 0 ;];
                    case 'b2'
                        ret=[0 0 ;];
                end
            case 3
                switch D
                    case 'a1'
                        ret=[0 0 ;];
                    case 'a2'
                        ret=[0 0 ;];
                    case 'b1'
                        ret=[1 0 ;];
                    case 'b2'
                        ret=[0 0 ;];
                end
            case 4
                switch D
                    case 'a1'
                        ret=[0 0 ;];
                    case 'a2'
                        ret=[0 0 ;];
                    case 'b1'
                        ret=[0 2/3 ;];
                    case 'b2'
                        ret=[0 1 ;];
                end
        end
% Derivatives with respect to the parameters
    case 'DpR'
        ret=[0;0;0;0;];
% Number of collocation points
    case 'm'
        if (intern('standard'))
            help=intern('rho');
            ret=help(2);
        else
```

```
            ret=length(intern('rho'));
        end
% Initialize linear solver
    case 'linear'
        ret=0;
 end
%Values read by bvpsuite GUI:
%Endpoint
%%#Endpoint
%startwert
%1%#startwert
%dim
%[2 2]%#dim
%parameter
%0%#parameter
%c
%[]%#c
%variablen
%musquare=81;gamma=1000;%#variablen
%standard
%true%#standard
%Infinite
%false%#Infinite
%EVP
%false%#EVP
%rho
%[1 5]%#rho
%x1
%0:0.1:1%#x1
%abstol
%1e-014%#abstol
%reltol
%1e-014%#reltol
%auto
%false%#auto
%g
%z1''+3/t*z1'=-musquare*z2-2*gamma+z1*z2;
z2''+3/t*z2'=musquare*z1-(1/2)*z1^2%#g
%r1
%z1'(a)=0;z2'(a)=0;z1(b)=0;z2'(b)+(1-1/3)*z2(b)=0%#r1
```

8 Code Performance

In this section, we comment on the performance of our code `bvpsuite` when compared to other available software for the numerical solution of BVPs in ODEs. Since our focus is on singular BVPs, we have chosen those codes which explicitly claim that singular problems are in their scope. Therefore, we take into considerations the standard MATLAB code `bvp4c` and its higher order versions `bvp5c`, `bvp6c`, cf. [15], [26], and two FORTRAN codes, `BVP_Solver` [27] and the COLNEW solver described in [1].

Our main intention while designing `bvpsuite` was to provide a MATLAB code which can cope with a wide range of applications and works dependably and efficiently for a large range of tolerances with an emphasis on high accuracy. Therefore, we have chosen the *fully implicit formulation* of the nonlinear system of equations and nonlinear boundary conditions, see Section 2. The order of the differential equations in the components of the system can be arbitrary and different for different components. Thus, there is no need to transform a higher order system to its first order form. The code can cope with free unknown parameters for which the appropriate number of additional boundary conditions need to be specified at the borders or within the interval of integration. In its scope are nonlinear singular BVPs with a singularity of the first or of the second kind[5]. Over the years, we have been able to give a good theoretical justification for all components of the code, also in the context of singular problems, see for instance the list of publications at `http://www.math.tuwien.ac.at/~ewa`.
The code can solve index-1 differential-algebraic equations, a coupled system of differential equations and algebraic constraints. Also, it is equipped with a pathfollowing strategy in case of known parameter values such that the turning points in the solution/parameter path do not constitute a difficulty. Recently, we have equipped the code with modules for the solution of eigenvalue problems of arbitrary order, see the references below. Moreover, for a problem posed on a semi-infinite interval $[0, \infty)$ the code automatically reduces the problem to the interval $[0, 1]$ and after numerical computations it provides the approximate solution on an arbitrary interval $[0, b]$, $b < \infty$.
The order of the collocation solver is chosen automatically in dependence of the tolerance specified by the user and varies between two and eight. We stress that since in our code Gaussian points (or equidistant interior collocation points) are used, we avoid the evaluation at the singular point and therefore also in the case of singular problems only *one numerical method on the whole interval* is used and no pre-handling is necessary. In other words, there is no distinction between the solution of singular or regular problems with `bvpsuite`. The error estimate and the grid adaptation routine have been described in Sections 3 and 4, respectively

The code `bvp4c`[6] [25] can solve *explicit nonlinear systems of order one* with nonlinear boundary conditions

[5] and clearly, problems with no singularity

[6] In the following, we refer to `bvp4c` only, even when all three variants of the code are addressed.

and unknown parameters. However, the singular problems have to show a special structure,

$$z'(t) = \frac{S}{t} z(t) + f(t, z(t)), \quad t \in (0, T], \tag{21}$$

with a constant matrix S. This means that only a singularity of the first kind in this particular form is in the scope of the code. The basic solution method is polynomial collocation at four, five or six Lobatto points, respectively. Within one routine the order of the method is fixed to four, five or six. The quantity to be estimated and controlled is the residual, and residual and error in case of `bvp5c` [15].

The `BVP_Solver` [27] covers the same class of problems, regular and singular, as `bvp4c`. The methods used here are implicit Runge-Kutta schemes (MIRKDC) of orders two, four, and six. The code controls the defect in the differential equations and boundary conditions and also provides an estimate for the global error using the extrapolation technique.

Finally, COLNEW [1] can solve *explicit* nonlinear systems of ODEs of varying order up to four. The basic solver is collocation at Gaussian points whose number ranges from one to seven. The code controls the global error estimated from the mesh halving principle which in case of Gaussian points is strongly related to the residual. In this code a pathfollowing strategy is also available and the code can cope with free parameters.

We compare the performance of the codes by solving the following boundary value problem:

$$z'(t) = \frac{1}{t} \begin{pmatrix} 0 & 1 \\ 2 & 6 \end{pmatrix} z(t) + \begin{pmatrix} 0 \\ 4k^2 t^5 \sin(k^2 t^2) + 10t \sin(k^2 t^2) \end{pmatrix}, \tag{22a}$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} z(0) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} z(1) = \begin{pmatrix} 0 \\ \sin(k^2) \end{pmatrix}, \tag{22b}$$

where the analytical solution is known,

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

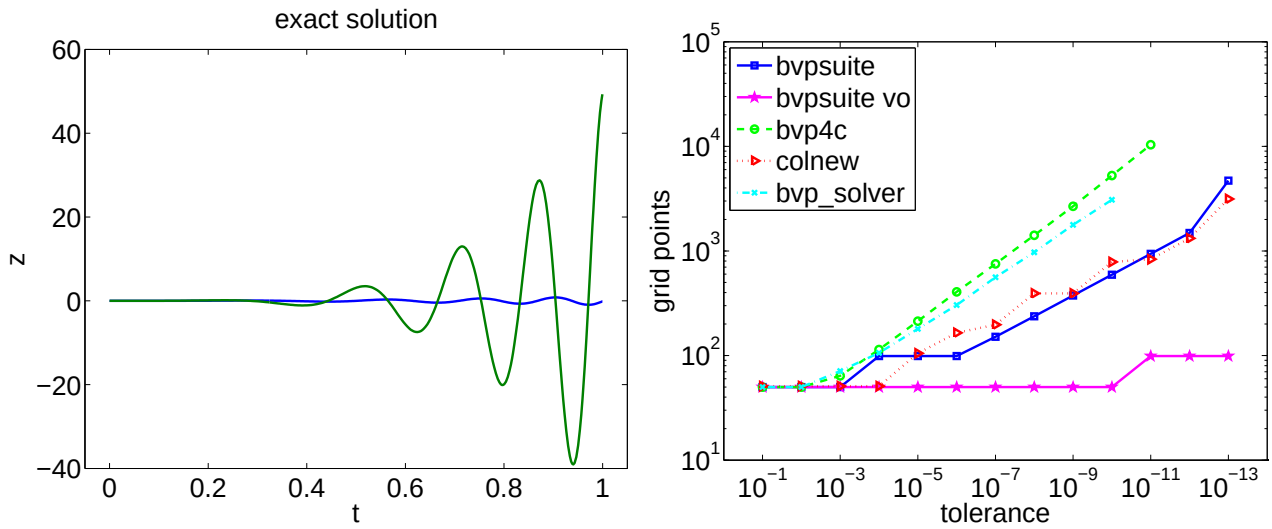

Figure 31: Problem (22), $k = 5$: The exact solution (left) and the number of grid points in the final mesh (right).

For the simulation the order of the methods implemented in the codes was set to four. Additionally, we also used the possibility of our code to adapt the order of the method to the prescribed tolerance, see

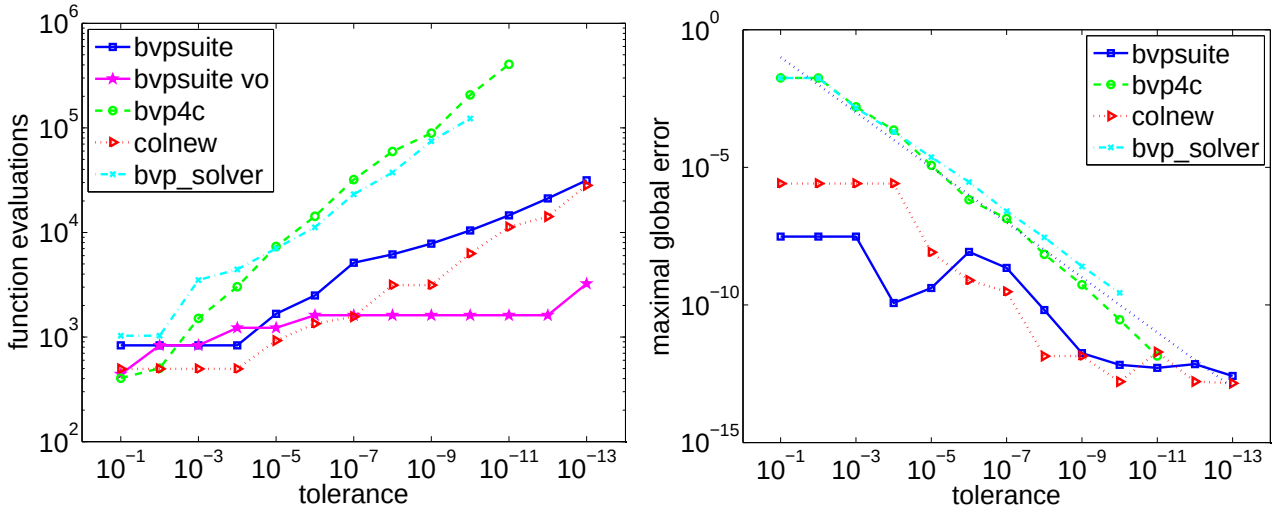

Figure 32: Problem (22), $k = 5$: The number of function evaluations (left) and the tolerance versus accuracy graph (right).

curve 'bvpsuite vo' in Figures 31 and 32. The results show that this approach provides the most efficient solution method. Therefore the flexibility of our code also constitutes a significant improvement of the performance. The model problem (22) discussed here, gives a typical picture observed in all tests. The results of the simulation can be found in Figures 31 and 32. When the order is fixed to four for comparison, the number of grid points required by `bvpsuite` and COLNEW is comparable and smaller throughout than for `bvp4c` and the `BVP_Solver`. Especially, for strict tolerances, the gap between `bvpsuite` and COLNEW and the other two codes is significant. The strictest tolerance succssesfully reached by the `BVP_Solver` was 10^{-10}, and by `bvp4c` 10^{-11}, while `bvpsuite` and COLNEW reached an accuracy of 10^{-13}. It is worth mentioning that for this example `bvp4c` and `BVP_Solver` work better at target than `bvpsuite` and COLNEW.

9 Applications

As already mentioned, eigenvalue problems, see [3], [28], and differential algebraic equations, cf. [7], [22], are within the scope of our code, but it can also be applied in case of non-standard singularities. In [24], we investigated the following singular equation which originates from the theory of shallow membrane caps,

$$(t^3u'(t))' + t^3\Big(\frac{1}{8u^2(t)} - \frac{a_0}{u(t)} - b_0t^{2\gamma-4}\Big) = 0, \quad t \in (0,1], \tag{23}$$

subject to boundary conditions

$$\lim_{t\to 0+} t^3u'(t) = 0, \quad u(1) = 0,$$

where a_0, b_0, and γ are given constants. Note that this problem has a more challenging structure than (1). After rewriting (23), we obtain the explicit version of the equation,

$$u''(t) + \frac{3}{t}\,u'(t) + \Big(\frac{1}{8u^2(t)} - \frac{a_0}{u(t)} - b_0t^{2\gamma-4}\Big) = 0, \quad u(1) = 0. \tag{24}$$

Here, a singularity of the first kind occurs at $t = 0$, but at the same time due to the boundary condition at $t = 1$ the problem has a so-called *phase singularity* at the other end of the interval. For such more

involved problems existence and uniqueness of solutions is shown by means of generalized lower and upper functions, involving limiting processes, cf. [24] and references therein. Our code `bvpsuite` could be used to approximate solutions[7] of the membrane problem. However, a theoretical justification for the collocation method in view of the problem structure is still an open question.

Another source of challenging problems with an interesting solution structure are reaction-diffusion equations, see [29], [30]. In [29], the simple looking, parameter dependent problem of the form

$$u''(t) = \frac{\lambda}{\sqrt{u(t)}}, \quad t \in (0,1], \quad u(0) = u(1) = 1, \tag{25}$$

where λ is a given parameter, turns out to have a very challenging structure. Depending on the value of λ there exist the so-called positive solutions, $u(t) > 0$ for all $t \in [0,1]$, pseudo dead core, and dead core solutions, such that $u(t) = 0$ for a certain point $t \in (0,1)$, or $u(t) = 0$ for a certain subinterval $t \in [\alpha, \beta], 0 < \alpha < \beta < 1$, respectively. In order to find the latter two solutions, we simulated the problem numerically using `bvpsuite`. Here, we utilized the fact that the above equation can be treated in its fully implicit form,

$$u''(t)\sqrt{u(t)}u(t) = \lambda u(t), \quad t \in (0,1], \quad u(0) = u(1) = 1. \tag{26}$$

Clearly, in the case that the analytical problem is especially involved, the numerical approach may sometimes constitute the only source of information about the solution structure. We faced this type of difficulty in [30]. Since the problem is again parameter dependent, we have applied the pathfollowing strategy implemented in `bvpsuite` to solve

$$((u'(t))^3)' + \frac{u'(t)}{t^2} = \lambda \left(\frac{1}{\sqrt{u(t)}} + (u'(t))^2 \right), \quad t \in (0,1), \tag{27a}$$

$$u'(0) = 0, \quad 0.1u(1) + u'(1) = 1. \tag{27b}$$

The results of this simulation are shown in Figure 35. We can see that for a certain range of λ the positive solution is unique, and for the other part of the path, we could find two different positive solutions, see Figures 33 and 34. According to Figure 35, we have moved around a turning point at $\lambda \approx 1.8442$.

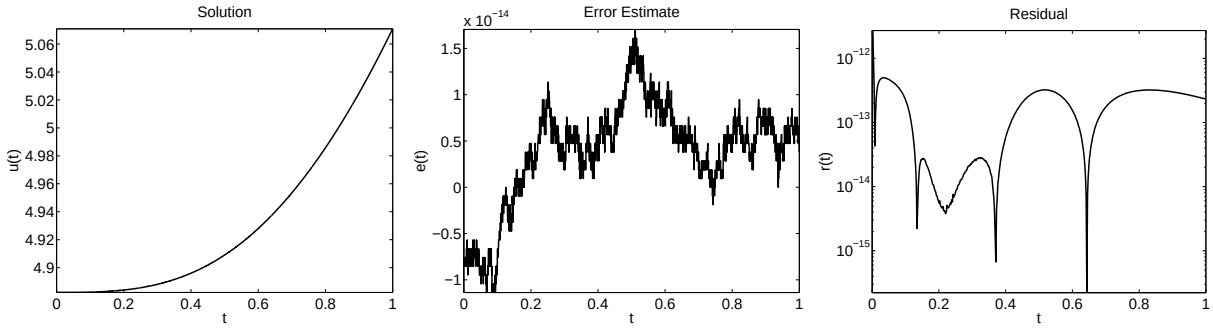

Figure 33: Problem (27): The numerical solution, the error estimate and the residual for $\lambda = 1.42604644036221$.

Finally, in the last step of the procedure, we obtained a solution which nearly reaches a pseudo dead core solution with the collocation solution $p(0) \approx u(0) \approx 0$.

[7] even though $u'(0)$ may become unbounded

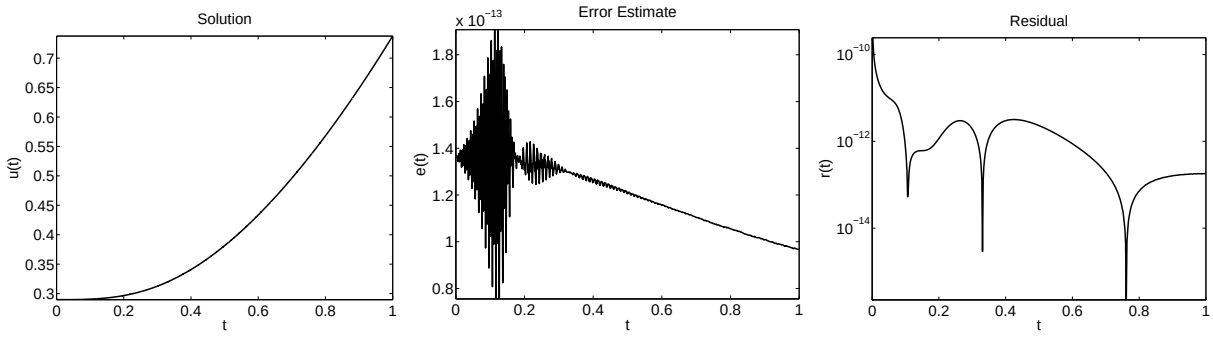

Figure 34: Problem (27): The numerical solution, the error estimate and the residual for $\lambda = 1.42139222684689$.

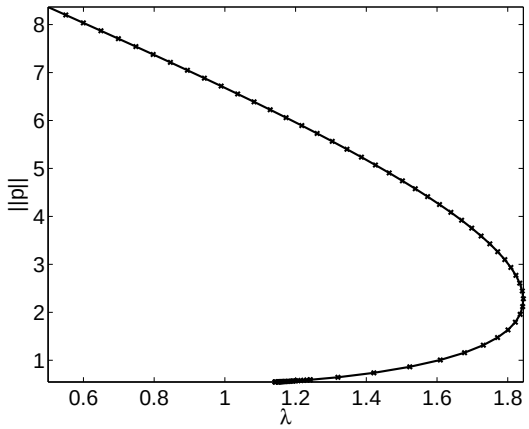

Figure 35: Graph of the $\|p\| - \lambda$ path obtained in 76 steps of the pathfollowing procedure, where $\|p\| = \max_{t\in[0,1]} |p(t)|$. The turning point has been found at $\lambda \approx 1.8442$.

References

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

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

[3] W. Auzinger, E. Karner, O. Koch, and E.B. Weinmüller. Collocation methods for the solution of eigenvalue problems for singular ordinary differential equations. *Opuscula Math.*, 26(2):229–241, 2006.

[4] W. Auzinger, G. Kneisl, O. Koch, and E.B. Weinmüller. A collocation code for boundary value problems in ordinary differential equations. *Numer. Algorithms*, 33:27–39, 2003.

[5] W. Auzinger, O. Koch, D. Praetorius, and E.B. Weinmüller. New a posteriori error estimates for singular boundary value problems. *Numer. Algorithms*, 40:79–100, 2005.

[6] W. Auzinger, O. Koch, and E.B. Weinmüller. Analysis of a new error estimate for collocation methods applied to singular boundary value problems. *SIAM J. Numer. Anal.*, 42(6):2366–2386, 2005.

[7] W. Auzinger, H. Lehner, and E.B. Weinmüller. Defect-based a posteriori error estimation for index-1 DAEs. In preparation.

[8] C. J. Budd, O. Koch, and E.B. Weinmüller. Computation of self-similar solution profiles for the nonlinear Schrödinger equation. *Computing*, 77:335–346, 2006.

[9] C. J. Budd, O. Koch, and E.B. Weinmüller. From nonlinear PDEs to singular ODEs. *Appl. Numer. Math.*, 56:413–422, 2006.

[10] J. Cash, G. Kitzhofer, O. Koch, G. Moore, and E.B. Weinmüller. Numerical solution of singular two-point boundary value problems. To appear in JNAIAM J. Numer. Anal. Indust. Appl. Math.

[11] M. Gräff, R. Scheidl, H. Troger, and E.B. Weinmüller. An investigation of the complete post-buckling behavior of axisymmetric spherical shells. *ZAMP*, 36:803–821, 1985.

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

[13] F.R. de Hoog and R. Weiss. Collocation methods for singular boundary value problems. *SIAM J. Numer. Anal.*, 15:198–217, 1978.

[14] F.R. de Hoog and R. Weiss. On the boundary value problem for systems of ordinary differential equations with a singularity of the second kind. *SIAM J. Math. Anal.*, 11:41–60, 1980.

[15] J. Kierzenka and L. Shampine. A BVP solver that controls residual and error. *JNAIAM J. Numer. Anal. Indust. Math.*, 3:27–41, 2008.

[16] G. Kitzhofer. *Numerical Treatment of Implicit Singular BVPs*. Ph.D. Thesis, Inst. for Anal. and Sci. Comput., Vienna Univ. of Technology, Austria. In preparation.

[17] G. Kitzhofer, O. Koch, P. Lima, and E.B. Weinmüller. Efficient numerical solution of the density profile equation in hydrodynamics. *J. Sci. Comput.*, 32:411–424, 2007.

[18] G. Kitzhofer, O. Koch, and E.B. Weinmüller. Collocation methods for the computation of bubble-type solutions of a singular boundary value problem in hydrodynamics. Techn. Rep. ANUM Preprint Nr. 14/04, Inst. for Anal. and Sci. Comput., Vienna Univ. of Technology, Austria, 2004. Available at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.

[19] G. Kitzhofer, O. Koch, G. Pulverer, Ch. Simon, and E.B. Weinmüller. The New Matlab Code `bvpsuite` for the Solution of Singular Implicit BVPs. To appear in JNAIAM, J. Numer. Anal. Indust. Appl. Math., 2010.

[20] G. Kitzhofer, O. Koch, and E.B. Weinmüller. Pathfollowing for essentially singular boundary value problems with application to the complex Ginzburg–Landau equation. *BIT Numerical Mathematics*, 49:141, 2009.

[21] O. Koch. Asymptotically correct error estimation for collocation methods applied to singular boundary value problems. *Numer. Math.*, 101:143–164, 2005.

[22] O. Koch, R. März, D. Praetorius, and E.B. Weinmüller. Collocation methods for index-1 DAEs with a singularity of the first kind. *Math. Comp.*, 79:281–304, 2010.

[23] G. Pulverer, G. Söderlind, and E.B. Weinmüller. Automatic grid control in adaptive BVP solvers. To appear in Numer. Algorithms.

[24] I. Rachůnková, O. Koch, G. Pulverer, and E.B. Weinmüller. On a singular boundary value problem arising in the theory of shallow membrane caps. *Math. Anal. and Appl.*, 332:523–541, 2007.

[25] L. Shampine and J. Kierzenka. A BVP solver based on residual control and the MATLAB PSE. *ACM Trans. Math. Software*, 27:299–315, 2001.

[26] 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/`.

[27] L. Shampine, P. Muir, and H. Xu. A user-friendly Fortran BVP solver. Available at `http://cs.smu.ca/~muir/BVP_SOLVER_Files/ShampineMuirXu2006.pdf`.

[28] Ch. Simon. *Numerical Solution of Singular Eigenvalue Value Problems for Systems of ODEs*. Master thesis, Vienna Univ. of Technology, Vienna, Austria, 2009.

[29] S. Staněk, G. Pulverer, and E.B. Weinmüller. Analysis and numerical solution of positive and dead core solutions of singular two-point boundary value problems. *Comput. Math. Appl.*, 56:1820–1837, 2008.

[30] S. Staněk, G. Pulverer, and E.B. Weinmüller. Analysis and numerical solution of positive and dead core solution of singular Sturm-Liouville problems. Submitted to Comput. Math. Appl., 2009.

[31] R. Winkler. Path-following for two-point boundary value problems. Techn. Rep. 78, Department of Mathematics, Humboldt-University Berlin, Germany, 1985.