DIPLOMARBEIT

BVPsuite 2.0 – a new version of a collocation code for singular BVPs in ODEs, EVPs and DAEs

Ausgeführt am Institut für
Analysis and Scientific Computing
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr. Winfried Auzinger,
Univ.-Doz. Dipl.-Ing. Dr. Othmar Koch,
Ao.Univ.Prof. Dipl.-Ing. Dr. Ewa B. Weinmüller

von Stefan Wurm, BSc.

Liebenstraße 39/12
1120 Wien

December 6, 2016

Abstract

This thesis was devoted to the implementation of an updated version of the open domain MATLAB code `bvpsuite1.1`. The goal was to understand the structure of the old code, identify the drawbacks, and implement missing features. An important task was to implement the new version in a possibly modular form to improve the readability and accessibility of the code. All modules were finally collected in a new software package `bvpsuite2.0`. These modules were:
Solving BVPs in ODEs on finite and semi-infinite intervals;
Solving EVPs in ODEs on finite and semi-infinite intervals;
Solving Index-1 DAEs.
In all cases, the differential operators may include boundary singularities and the ODE system may be subject to multi-point boundary conditions. To enhance efficiency, all modules are equipped with an error estimate and a mesh adaptation strategy.
For every part of the code, a user manual is provided.

Zusammenfassung

Diese Arbeit war der Erstellung der neuen Version von Matlab Software `bvpsuite1.1` gewidmet. Das Ziel war es die Struktur des vorhandenen Codes zu verstehen, seine Schwächen zu lokalisieren und die fehlenden Komponenten zu implementieren. Wichtig war dabei das neue Programm in möglichts modularen Form bereitzustellen, um die Lesbarkeit und Benützerfreundlichkeit des Codes zu erhöhen. Alle Module wurden zum neuen Software Paket `bvpsuite2.0` zusammengefasst. Diese Module sind:
Lösung von RWPen gewöhnlicher Differentialgleichungen auf endlichen und halbunendlichen Intervallen.
Lösung von EVPen gewöhnlicher Differentialgleichungen auf endlichen und halbunendlichen Intervallen.
Lösung von Index-1 Algebro-Differentialgleichungen.
In allen Fällen können in den Differentialoperatoren Singularitäten auftreten und die Randbedingungen können *multi-point* Charakter haben. Um die Effizienz zu erhöhen wurden alle Module mit Fehlerschätzern und einer Strategie zur Gitteranpassung ausgestattet.
Für das ganze Programm wird auch ein Manual bereitgestellt.

Contents

Chapter 1

Introduction

The aim of this thesis was to design and implement a new version of the well-established MATLAB package `bvpsuite1.0` [27, 28]. This package has been developed at the Institut for Analysis and Scientific Computing, Vienna University of Technology, and can be used for the numerical solution of implicit boundary value problems (BVPs) in ordinary differential equations (ODEs) as well as eigenvalue problems (EVPs), and Index-1 Differential-Algebraic Equations (DAEs). The ODE system can have a general implicit form and be of mixed order[1] subject to multi-point boundary conditions.
The following problem will serve as an example,

$$F(t, z^{(4)}(t), z^{(3)}(t), z''(t), z'(t), z(t)) = 0, \quad 0 < t \leq 1, \tag{1.1}$$

$$b(z^{(3)}(0), z''(0), z'(0), z(0), z^{(3)}(1), z''(1), z'(1), z(1)) = 0. \tag{1.2}$$

Problem (1.1) may also include unknown parameters to be calculated together with the unknown solution z. In such a case, the problem has to be augmented by a correct number of additional boundary conditions. The underlying system of ODEs can be posed on a finite interval $[a, b]$ or on a semi-infinite interval $[a, \infty)$, where $a \geq 0$. For the latter case, an automatic transformation of the semi-infinite interval to a finite domain is provided, in the case that the boundary condition at infinity are of Dirichlet type. In the scope of `bvpsuite1.0` are also parameter-dependent problems, where for a given value of the parameter a related solution is to be found. Here, the pseudo arclength parametrization is used to move around inflection points in the solution/parameter path. However, the code is not prepared to handle bifurcation points in the solution/parameter path.

For many years, at the Institute for Analysis and Scientific computing, research was focused on the analysis and numerical solution of ODEs with time and space singularities. Such problems are frequently formulated in the following form: Find a

[1] The highest involved derivative may vary with the solution component and it can also be zero, which means that algebraic constrains which do not involve derivatives are also admitted.

solution $z \in C[0,1]$ such that

$$z'(t) = \frac{M(t)}{t^\alpha} z(t) + f(t, z(t)), \quad t \in (0,1], \tag{1.3}$$

$$B_0 z(0) + B_1 z(1) = \beta, \tag{1.4}$$

where $\alpha \geq 1$, the solution z is an n-dimensional real function, M is a given smooth $n \times n$ matrix and f is an n-dimensional smooth inhomogeneity. The $m \times n$, $m \leq n$, matrices B_0 and B_1 are constant and have to satisfy certain restrictions for the BVP (1.3)–(1.4) to be well-posed and have a locally unique solution. To satisfy $z \in C[0,1]$, boundary conditions (1.4) have to be augmented by additional $n - m$ requirements (in general homogeneous initial conditions) closing the system. The ODE system (1.3) is said to be singular with a *singularity of the first kind* for $\alpha = 1$, while for $\alpha > 1$, the singularity is called *singularity of the second kind* or *essential singularity*.

The analytical properties of problem (1.3)–(1.4), mainly the existence and uniqueness theory, can be found in [17, 20]. It turns out that the spectrum of the matrix $M(0)$ plays the crucial role in this context. To compute the numerical solution of (1.3)–(1.4), polynomial collocation has been proposed because in the presence of a singularity other high order finite difference methods may show order reductions [21]. The convergence analysis for collocation applied to problems with a singularity of the first kind, $\alpha = 1$, [8, 18, 29] indicates that for well-posed problems with sufficiently smooth solutions the classical stage order holds. However, the high-order superconvergence at the mesh points does not hold in general for singular problems; see [29] for details.

To make the computations more efficient, an adaptive mesh selection strategy based on an a posteriori estimate for the global error of the collocation solution has been implemented. Also a graphical user interface (GUI) has been provided to simplify the use of the code. The `bvpsuite1.0` software has already been successfully applied to a variety of problems, see for example [5, 11, 34, 24, 27], and was well-received in the community. However, experience showed that due to the complicated structure of the code, there was a need for a simpler modular structure of the package.

The thesis was carried out in cooperation with Markus Schöbinger [36]. The first goal was to understand the structure of the code, identify shortcomings, and implement missing features. An important task was to detach different modules to improve the accessibility. All modules were finally collected in a new software package `bvpsuite2.0`.

The following modules have been implemented:

- Solving BVPs in ODEs on finite intervals (including singularities, free parameters, and multi-point boundary conditions);

- Solving BVPs in ODEs on semi-finite intervals (including singularities, free parameters, and multi-point boundary conditions);
- Solving EVPs in ODEs on finite and semi-infinite intervals (including singularities and multi-point boundary conditions);
- Solving Index-1 DAEs (including singularities and multi-point boundary conditions);
- In a separate test module, we solve higher-index DAEs using the novel approach based on Least Squares Collocation [16].

All modules but the last are equipped with an error estimate and a mesh adaptation strategy. For every part of the code, we give step-by-step instructions for its use.

Chapter 2

Solving BVPs in ODEs on finite domains

In the following Sections 2.1 and 2.2, we use the notation and follow the presentation given in [24].

2.1 Problem setting

In this work, we deal with the following problem: Let $\mathcal{I} = [a, b] \subset \mathbb{R}$ be given, as well as a vector $\boldsymbol{c} \in \mathbb{R}^q$ with $c_i \in \mathcal{I}$ and $c_i \neq c_j$ for $i \neq j$.
Then, we try to find a vector function $\boldsymbol{z}(t) : \mathcal{I} \to \mathbb{R}^n$ and a vector of parameters $\boldsymbol{p} \in \mathbb{R}^s$, such that for all $t \in \mathcal{I}$ the ODE system

$$\boldsymbol{f}(t, \boldsymbol{p}, z_1(t), z_1'(t), \ldots, z_1^{(l_1)}(t), \ldots, z_i(t), \ldots, z_i^{(l_i)}(t), \ldots, z_n(t), \ldots, z_n^{(l_n)}(t)) = \mathbf{0}, \quad (2.1)$$

and the boundary conditions

$$\begin{aligned} &\boldsymbol{g}(\boldsymbol{p}, z_1(c_1), \ldots, z_1^{(l_1-1)}(c_1), \ldots, z_n(c_1), \ldots, z_n^{(l_n-1)}(c_1), \ldots, \\ &\quad \ldots, z_1(c_q), \ldots, z_1^{(l_1-1)}(c_q), \ldots, z_n(c_q), \ldots, z_n^{(l_n-1)}(c_q)) = \mathbf{0}, \end{aligned} \quad (2.2)$$

are satisfied. Here,

$$\boldsymbol{f} : \mathcal{I} \times \mathbb{R}^s \times \mathbb{R}^{\sum(l_i+1)} \to \mathbb{R}^n, \;\; \boldsymbol{g} : \mathbb{R}^s \times \mathbb{R}^{q\sum l_i} \to \mathbb{R}^{s+\sum l_i}$$

and we assume that $z_i \in C^{l_i-1}(\mathcal{I})$ and $z_i^{(l_i)}$ exist.

2.2 Collocation

To solve the BVP (2.1)–(2.2) numerically, polynomial collocation is used. We first discretize $\mathcal{I}$ by introducing a not necessarily equidistant mesh,

$$\Delta = \{a = \tau_0 < \tau_1 < \ldots < \tau_N = b, \quad h_i := \tau_i - \tau_{i-1}, \quad i = 1, \ldots, N\}.$$

Moreover, we choose a vector $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_j, \ldots, \rho_m)$, $\rho_j \in (0, 1)$ and $\rho_i \neq \rho_j$ for $i \neq j$, to generate the so-called (inner) collocation points located in the subinterval (τ_{i-1}, τ_i),

$$T_{ij} := \tau_{i-1} + \rho_j\, h_i, \quad i = 0, \ldots, N, \quad j = 1, \ldots, m.$$

The idea of collation is to approximate each component of $\boldsymbol{z}$ by a piecewise polynomial function. More precisely, for the component z_k on the subinterval (τ_{i-1}, τ_i), we make the ansatz

$$z_k\Big|_{(\tau_{i-1},\tau_i)} \approx P_{ik} \in \mathbb{P}_{m+l_k-1},$$

where P_{ik} is a polynomial of degree less than or equal to $m + l_k - 1$. The polynomials P_{ik} are represented using the so-called Runge-Kutta basis,

$$P_{ik}(t) = \sum_{j=1}^{l_k} Y_{ijk}\Phi_{ij}(t) + \sum_{j=1}^{m} Z_{ijk}\Psi_{ij}^{l_k}(t), \tag{2.3}$$

where for $t \in [\tau_{i-1}, \tau_i]$,

$$\Phi_{ij}(t) := \frac{(t-\tau_i)^{(j-1)}}{(j-1)!}$$

and

$$\Psi_{ij}^0(t) = \prod_{\substack{\nu=1\\ \nu\neq j}}^{m} \frac{t - T_{i\nu}}{T_{ij} - T_{i\nu}}, \quad \Psi_{ij}^k(t) = \int_{\tau_{i-1}}^{t} \Psi_{ij}^{k-1}(s)\,\mathrm{d}s, \quad k > 0.$$

These definitions result in the following convenient properties:

$$P_{ik}^{(d-1)}(\tau_i) = Y_{idk}, \quad d = 1, \dots, l_k, \quad P_{ik}^{(l_k)}(T_{ij}) = Z_{ijk}.$$

To calculate the unknown coefficients in (2.3), we require that the ODE system (2.1) is satisfied in all collocation points, the piecewise polynomial function is globally continuous together with its derivatives up to the order $l_k - 1$ (for z_k) and the boundary conditions (2.2) hold. This results in

$$\boldsymbol{f}(T_{ij}, \boldsymbol{p}, P_{i1}(T_{ij}), \dots, P_{i1}^{(l_1)}(T_{ij}), \dots, P_{in}(T_{ij}), \dots, P_{in}^{(l_n)}(T_{ij})) = \mathbf{0}, \tag{2.4}$$

$$\begin{aligned}\boldsymbol{g}(\boldsymbol{p}, P_{*1}(c_1), \dots, P_{*1}^{(l_1-1)}(c_1), \dots, P_{*n}(c_1), \dots, P_{*n}^{(l_n-1)}(c_1), \dots,\\ \dots, P_{*1}(c_q), \dots, P_{*1}^{(l_1-1)}(c_q), \dots, P_{*n}(c_q), \dots, P_{*n}^{(l_n-1)}(c_q)) = \mathbf{0},\end{aligned} \tag{2.5}$$

where $i = 1, \dots, N$, $j = 1, \dots, m$. The asterisks indicate that depending on the location of c_l the corresponding polynomial P_i is chosen in such a way that $c_l \in [\tau_{i-1}, \tau_i)$. Finally, the continuity conditions read:

$$P_{i-1,k}^{(d-1)}(\tau_i) = P_{i,k}^{(d-1)}(\tau_i) \leftrightarrow Y_{i-1,d,k} = Y_{i,d,k}, \quad d = 1, \dots, l_k, \tag{2.6}$$

where $i = 2, \dots, N$, $j = 1, \dots, m$. The system (2.4), (2.5), and (2.6) contains $Nmn + s + \sum l_i + (N-1)\sum l_i$ equations for the unknown coefficients, Y_{ijk} $(N\sum l_i$

unknowns) and Z_{ijk} (Nmn unknowns), as well as for $\boldsymbol{p}$ (s unknowns). Thus, the number of equations is equal to the number of unknowns and the discrete system is closed.

If the original problem was *linear*,

$$\text{Eq. (2.1)} \Leftrightarrow \underbrace{\frac{\partial \boldsymbol{f}(t,\boldsymbol{p},z_1(t),\dots,z_1^{(l_1)}(t),\dots,z_n(t),\dots,z_n^{(l_n)}(t))}{\partial(\boldsymbol{p},z_1(t),\dots,z_1^{(l_1)}(t),\dots,z_n(t),\dots,z_n^{(l_n)}(t))}}_{=:\boldsymbol{F}(t)\in\mathbb{R}^{n\times\left(s+\sum(l_i+1)\right)}} \begin{pmatrix} \boldsymbol{p} \\ z_1(t) \\ \vdots \\ z_1^{(l_1)}(t) \\ \vdots \\ z_n(t) \\ \vdots \\ z_n^{(l_n)}(t) \end{pmatrix} = \boldsymbol{r_f}(t) \in \mathbb{R}^n,$$

and

$$\text{Eq. (2.2)} \Leftrightarrow \underbrace{\frac{\partial \boldsymbol{g}(\boldsymbol{p},z_1(c_1),\dots,z_1^{(l_1-1)}(c_1),\dots,z_n^{(l_n-1)}(c_q))}{\partial(\boldsymbol{p},z_1(c_1),\dots,z_1^{(l_1-1)}(c_1),\dots,z_n^{(l_n-1)}(c_q))}}_{=:\boldsymbol{G}\in\mathbb{R}^{\left(s+\sum l_i\right)\times\left(s+q\sum l_i\right)}} \begin{pmatrix} \boldsymbol{p} \\ z_1(c_1) \\ \vdots \\ z_1^{(l_1-1)}(c_1) \\ \vdots \\ z_n^{(l_n-1)}(c_q) \end{pmatrix} = \boldsymbol{r_g} \in \mathbb{R}^{s+\sum l_i},$$

then this would also be the case for the system (2.4), (2.5), and (2.6). In the code the linear and the nonlinear case are treated within two different solver routines. If the user declares the problem as linear, the code uses a linear solver, which helps to save computation time.
For a *nonlinear* analytical problem, a nonlinear discrete system of equations has to be solved using a Newton procedure. For details see [36].

2.3 User's guide to linear BVPs on finite intervals

The use of `bvpsuite2.0` in the context of BVPs in ODEs posed on finite intervals will be illustrated by solving the following problem [28]:

Example 2.1
We consider the linear, singularly perturbed BVP, where $\varepsilon = 10^{-4}$,

$$\varepsilon z''(t) + z'(t) - (1+\varepsilon)z(t) = 0, \quad t \in [-1,1], \tag{2.7a}$$

$$z(-1) = 1 + \mathrm{e}^{-2}, \quad z(1) = 1 + \mathrm{e}^{\frac{-2(1+\varepsilon)}{\varepsilon}}, \tag{2.7b}$$

and proceed as indicated below.

1. Save the file `template_bvp.m` as a file with a new name (e. g. `singpert.m`).
2. Fill in the file with the specification of the following parameters:

n Number of solution components.

orders $\boldsymbol{l}$ Highest order of each component.

problem $\boldsymbol{f}$ Replace $z_i^{(j)}(t)$ by `z(i,j+1)`, $p(i)$ by `p(i)` and t by `t`, see (2.1).

jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}$ The Jacobian of $\boldsymbol{f}$ with respect to $z_i^{(j)}$; `ret(i,j,k)` corresponds to $\frac{\partial f_i}{\partial z_j^{(k)}}$. The same replacement rules as for "**problem**" apply.

interval The interval `[a,b]` on which the problem is solved.

linear `1` for a linear problem, `0` for a nonlinear problem.

parameters s The number of unknown parameters.

c Additional points in which boundary conditions are posed. Do not include a and b.

BV $\boldsymbol{g}$ Replace $z_i^{(j)}(a)$ by `za(i,j)`, $z_i^{(j)}(b)$ by `zb(i,j)`, $z_i^{(j)}(c_k)$ by `zc(i,j,k)` and $p(i)$ by `p(i)`. You can use only either `za(i,j)` and `zb(i,j)` **or** `zc(i,j,k)`, see (2.2).

dBV $\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}(\boldsymbol{z}(a),\boldsymbol{z}(b))}$ or $\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}(\boldsymbol{z}(c_1),\dots,\boldsymbol{z}(c_q))}$ The Jacobian of $\boldsymbol{g}$ with respect to $z_i^{(j)}(a)$ and $z_i^{(j)}(b)$, or $z_i^{(j)}(c_k)$;
`ret(1,i,j,k)` corresponds to $\frac{\mathrm{d}g_i}{\mathrm{d}z_j^{(k)}(a)}$ and `ret(2,i,j,k)` corresponds to $\frac{\mathrm{d}g_i}{\mathrm{d}z_j^{(k)}(b)}$, or `ret(cInd,i,j,k)` corresponds to $\frac{\mathrm{d}g_i}{\mathrm{d}z_j^{(k)}(c_{\mathrm{cInd}})}$.

dP $\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{p}}$ The Jacobian of $\boldsymbol{f}$ with respect to p_i; `ret(i,j)` corresponds to $\frac{\mathrm{d}f_i}{\mathrm{d}p_j}$.

dP_BV $\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\boldsymbol{p}}$ `ret(i,j)` corresponds to $\frac{\mathrm{d}g_i}{\mathrm{d}p_j}$.

initProfile The initial solution profile for the Newton solver.

EVP 1 for an eigenvalue problem, see Chapter 4, 0 else.

dLambda See Chapter 4.

In the context of Example 2.1 this means that we have to specify:

$$
\begin{aligned}
f &= \varepsilon z''(t) + z'(t) - (1+\varepsilon)z(t),\\
\frac{\partial f}{\partial z(t)} &= -(1+\varepsilon),\\
\frac{\partial f}{\partial z'(t)} &= 1,\\
\frac{\partial f}{\partial z''(t)} &= \varepsilon,\\
\boldsymbol{g} &= \begin{pmatrix} z(a) - 1 + \mathrm{e}^{-2} \\ z(b) - 1 + \mathrm{e}^{\frac{-2(1+\varepsilon)}{\varepsilon}} \end{pmatrix},\\
\frac{\partial g_1}{\partial z(a)} &= 1,\\
\frac{\partial g_1}{\partial z(b)} &= 0,\\
\frac{\partial g_2}{\partial z(a)} &= 0,\\
\frac{\partial g_2}{\partial z(b)} &= 1.
\end{aligned}
$$

The problem definition file (bvpfile) is enclosed below.

Listing 2.1: Problem file for Example 2.1

```
function [ret] = ↷
   ↪ template_bvp(request,z,za,zb,zc,t,p,lambda)
epsilon=10e-4;
switch request
    case 'n'
        ret=  1;
    case 'orders'
        ret=[ 2 ];
    case 'problem'
        ret=[epsilon*z(1,3)+z(1,2)-(1+epsilon)*z(1,1)
            ];
    case 'jacobian'
        %DON'T CHANGE THIS LINE:
        ret = ↷
           ↪ zeros(length(feval(mfilename,'problem', ↷
           ↪ zeros(feval(mfilename,'n'), ↷
           ↪ max(feval(mfilename,'orders'))+1), ↷
           ↪ [],[],[],0, ↷
```

```
              zeros(feval(mfilename,'parameters'),1))),
              feval(mfilename,'n'),
              max(feval(mfilename, 'orders'))+1);
            ret(1, 1, 1)= -(1+epsilon);
            ret(1, 1, 2)=1;
            ret(1, 1, 3)=epsilon;
    case 'interval'
        ret = [-1 ,1 ];
    case 'linear'
        ret= 1;
    case 'parameters'
        ret=0;
    case 'c'
        ret = [];
    case 'BV'
        ret=[za(1,1)-(1+exp(-2));
            zb(1,1)-(1+exp((-2*(1+epsilon))/epsilon))
            ];
    case 'dBV'
        %DON'T CHANGE THIS LINE:
        ret =
              zeros(max(length(feval(mfilename,'c')),
              2-length(feval(mfilename,'c'))),
              length(feval(mfilename,'problem',
              zeros(feval(mfilename,'n'),
              max(feval(mfilename,'orders'))+1),
              [],[],[],0,
              zeros(feval(mfilename,'parameters'),
              1))), feval(mfilename,'n'),
              max(feval(mfilename,'orders')));
            ret( 1,1 ,1 ,1 )= 1;
            ret( 2,2,1,1)=1;
    case 'dP'
        ret=[];
    case 'dP_BV'
        ret=[];
    case 'initProfile'
        ret.initialMesh = [ ];
        ret.initialValues = [ ];
    case 'EVP'
        ret = 0;
```

```
        case 'dLambda'
            ret = 0;
        otherwise
            ret = 0;

    end
```

3. Specify the numerical settings in `default_settings.m`. Alternatively, copy this file to create desired settings, e. g. for later use.

   The following settings are available:

   mesh τ, i. e. the mesh on which the problem is solved.

   collMethod Determines which collocation points are used. Currently, `'gauss'` (Gauss-Legendre points), `'lobatto'` (Lobatto points), `'uniform'` (uniformly distributed points) and `'user'` (user-specified points) are available.

   collPoints If for `collMethod` `'user'` is chosen, then here are the points on $[0, 1]$ specified. In the other cases, the number of points are specified (i. e. m).

   meshAdaptation `1`, if successive adaptation of the mesh should be done, until the tolerance (specified in `absTolMeshAdaptation` and `relTolMeshAdaptation`) is reached. This implies the use of error estimation, cf. Chapter 5. `0` else.

   errorEstimate `1`, if the error should be estimated, cf. Chapter 5. `0` else.

   absTolSolver and

   relTolSolver specify the tolerance levels used in the non-linear solver. Further informations can be found in [36].

   absTolMeshAdaptation and

   relTolMeshAdaptation specify the tolerance levels that should be reached in the mesh adaptation process.

   minInitialMesh specifies the minimal number of points in the initial profile. If the user declares a profile with fewer points, the missing points are inserted and the values are computed by interpolation.

   The further settings are expert settings for the non-linear solver, the meaning and remarks on usage can be found in [36].

   For our example we use:

Listing 2.2: The settings file for Example 2.1

```
function [ret] = default_settings(request)

switch request
    case 'mesh'
        ret=0:1/160:1;
    case 'collMethod'
        ret='gauss';
    case 'collPoints'
        ret=4;
    case 'meshAdaptation'
        ret=1;
    case 'errorEstimate'
        ret=0;
    case 'absTolSolver'
        ret=1e-12;
    case 'relTolSolver'
        ret=1e-12;
    case 'absTolMeshAdaptation'
        ret=1e-9;
    case 'relTolMeshAdaptation'
        ret=1e-9;
    case 'minInitialMesh'
        ret=50;

    case 'finemesh'
        ret = 0;

    case 'allowTRM'
        ret=1;
    case 'maxFunEvalsTRM'
        ret=90000000;
    case 'maxIterationsTRM'
        ret=90000000;
    case 'lambdaMin'
        ret = 0.001;
    case 'maxAdaptations'
        ret=18;
    case 'switchToFFNFactor'
        ret=0.5;
```

```
    case 'updateJacFactor'
        ret=0.5;
    case 'K'
        ret=200;

end

end
```

4. Call `bvpsuite2.0` by typing in the command window of MATLAB:

   ```
   [x,z,s]=bvpsuite2('singpert','default_settings');
   ```

   There are three return values:

   x Gives the mesh on which the solution was computed. (This mesh is different from the specified mesh if mesh adaptation is used.)

   z The values of the solution. `z(i,j)` corresponds to the approximation of $z_i(x_j)$.

   s A struct variable, which contains more information about the solution and can be used as an initial profile (as third argument of `bvpsuite2`) for further calls.

   For example, now we can plot the computed approximation by typing

   ```
   plot(x,y)
   ```

   and we obtain the solution visualized as seen in Fig. 2.1.

   In addition, we can visualize the estimated error by the command

   ```
   semilogy(s.x1tau,abs(s.errest);
   ```

   The obtained plot is shown in Fig. 2.2 and indicates the success of the mesh adaptation.

2.4 User's guide to non-linear problems

In this section we want to illustrate the procedure for non-linear problems by this example from economics [46]:

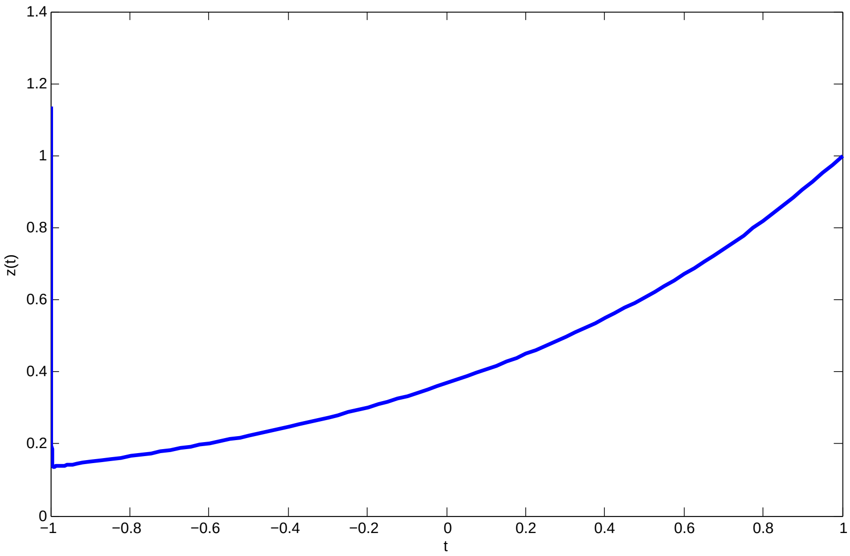


Figure 2.1: Example 2.1: Numerical solution.

Example 2.2 (Bayes Nash)
In auction theory, the first order condition any Bayesian Nash equilibrium needs to satisfy is

$$y'(t) = -\frac{v}{1-v}\frac{h(t)-y(t)}{k(t)-y(t)}, \quad t \in [a,b]\,,$$

with given $h(t)$ and $k(t)$ with $h(t) \geq k(t)$, $h(a) = k(a) = a$ and $h(b) = k(b) = b$. Furthermore, $v = \int_a^b y(s)\,\mathrm{d}s$ must hold.

Analysis shows $h(t) \geq y(t) \geq k(t)$, therefore we can use the boundary condition $y(a) = a$ or $y(b) = b$.

To formulate this problem in our setting, we introduce $u(t) := \int_a^t y(s)\,\mathrm{d}s$. Then it holds, $u'(t) = y(t)$, $u(a) = 0$ and $u(b) = v$. We will set v as an unknown parameter.

Summarizing, we set

$$\boldsymbol{f} := \begin{pmatrix} z_1'(t) + \frac{p}{1-p}\frac{h(t)-z_1(t)}{k(t)-z_1(t)} \\ z_2'(t) - z_1(t) \end{pmatrix}$$

$$\boldsymbol{g} := \begin{pmatrix} z_1(a) - a \\ z_2(a) \\ z_2(b) - p \end{pmatrix}$$

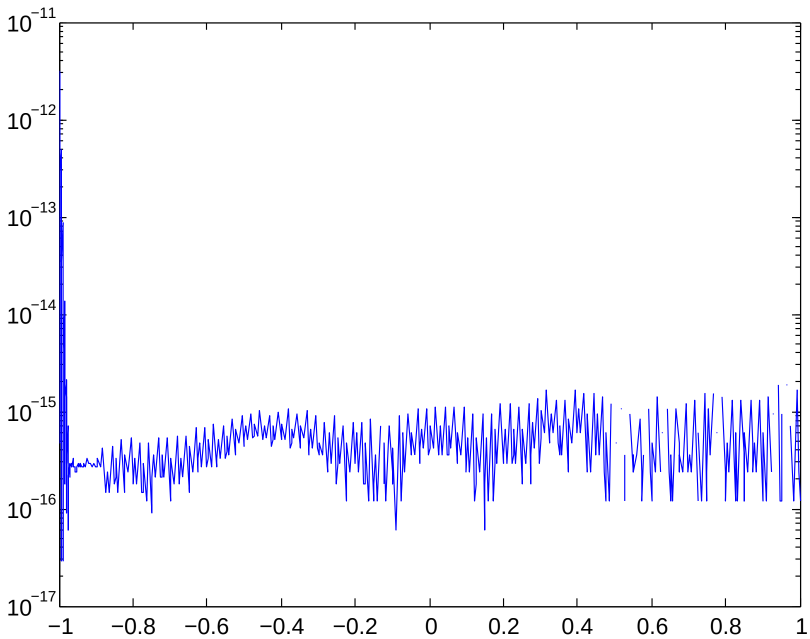


Figure 2.2: Example 2.1: Estimated error of the numerical solution.

and compute from that

$$
\begin{aligned}
\frac{\partial f_1}{\partial z_1} &= \frac{p}{1-p}\frac{h(t)-k(t)}{(k(t)-z_1(t))^2},\\
\frac{\partial f_1}{\partial z_1'} &= 1,\\
\frac{\partial f_2}{\partial z_1} &= -1,\\
\frac{\partial f_2}{\partial z_2'} &= 1,\\
\frac{\partial g_1}{\partial z_1(a)} &= 1,\\
\frac{\partial g_2}{\partial z_2(a)} &= 1,\\
\frac{\partial g_3}{\partial z_2(b)} &= 1,\\
\frac{\partial f_1}{\partial p} &= (1-p)^{-2},\\
\frac{\partial g_3}{\partial p} &= -1.
\end{aligned}
$$

For $a = 0$, $b = 1$, $h(t) := \sqrt{t}$ and $k(t) := t$ this leads to

```
function [ret] = template_bvp(request,z,za,zb,zc,t,p,lambda)
h=@(t) sqrt(t);
k=@(t) t;
switch request
    case 'n'
        ret=  2;
    case 'orders'
        ret=[ 1 1 ];
    case 'problem'
        ret=[z(1,2) + p(1)/(1-p(1)) * (h(t)-z(1,1))/(k(t)-z(1,1));
            z(2,2)-z(1,1)];
    case 'jacobian'
        %DON'T CHANGE THIS LINE:
        ret = zeros( ( feval( mfilename, 'problem' , zeros( feval( mfilename, 'n'), max( feval( mfilename, 'orders'))+1),[],[],[],0,zeros( feval( mfilename, 'parameters'),1))),feval( mfilename,'n'),max( feval( mfilename, 'orders'))+1);
         ret(1, 1, 1)= p(1)/(1-p(1)) * (h(t)-k(t))/(k(t)-z(1,1))^2;
         ret(1, 1, 2)=1;
         ret(2, 1, 1)=-1;
         ret(2, 2, 2)=1;
    case 'interval'
        ret = [0 , 1];
    case 'linear'
        ret= 0;
    case 'parameters'
        ret=1;
    case 'c'
        ret = [];
    case 'BV'
        ret=[za(1,1)-0;
            za(2,1);
            zb(2,1)-p(1)];
    case 'dBV'
        %DON'T CHANGE THIS LINE:
```

```
        ret = zeros( max( length( feval( mfilename, 'c')), 2-length( feval( mfilename ,'c'))), length( feval( mfilename, 'problem', zeros( feval( mfilename,'n'), max( feval( mfilename,'orders'))+1), [],[],[],0, zeros( feval( mfilename,'parameters'),1))), feval( mfilename,'n'), max( feval( mfilename,'orders')));
         ret( 1,1 ,1 ,1 )= 1;
         ret(1,2,2,1)=1;
         ret(2,3,2,1)=1;
    case 'dP'
        ret=[1/(1-p(1))^2;0];
    case 'dP_BV'
        ret=[0;0;-1];
    case 'initProfile'
        ret.initialMesh = 0:1/100:1;
        ret.initialValues = [sqrt(ret.initialMesh); 2/3*(ret.initialMesh).^(3/2)];
        ret.parameters=2/3;
    case 'EVP'
        ret = 0;
    case 'dLambda'
        ret = 0;
    otherwise
        ret = 0;

end
```

We take $h(t)$ as initial solution. By invoking `bvpsuite2.0` we obtain the solution as seen in Fig. 2.3.

The discretization of the initial guess is near enough to the discretization of the exact solution, as we can see, that the implemented solver for non-linear problems converges. As expected the solution lies between $h(t)$ and $k(t)$.

Summarizing, we can state that the implemented methods are a powerful tool to solve linear as well as non-linear BVPs, which may be even given in an implicit form.

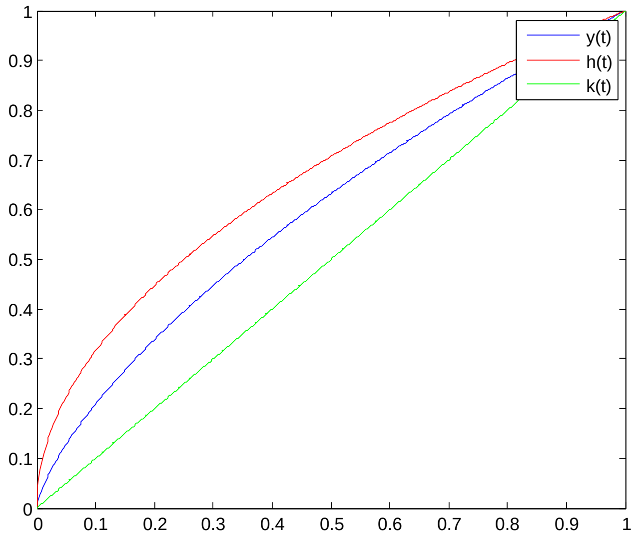


Figure 2.3: Example 2.2: Approximation of the equilibrium solution.

Chapter 3

Problems on $[a, \infty)$

In this chapter we investigate how to solve a boundary value problem

$$\boldsymbol{f}(t, \boldsymbol{p}, z_1(t), z_1'(t), \ldots, z_1^{(l_1)}(t), z_2(t), \ldots, z_i(t), \ldots, z_i^{(l_i)}(t), \ldots, z_n(t), \ldots, z_n^{(l_n)}(t)) = 0 \tag{3.1a}$$

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{p}, z_1(c_1), \ldots, z_1^{(l_1-1)}(c_1), \ldots, z_n(c_1), \ldots, z_n^{(l_n-1)}(c_1), \ldots, \\ \ldots, z_1(c_q), \ldots, z_1^{(l_1-1)}(c_q), \ldots, z_n(c_q), \ldots, z_n^{(l_n-1)}(c_q)) = 0, \end{aligned} \tag{3.1b}$$

which is not posed on a finite interval as in Section 2.1, but on a semi-infinite interval $\mathcal{I} = [a, \infty)$. This raises the issue that $\mathcal{I}$ cannot be discretized directly by a finite grid and therefore it is not possible to apply collocation directly.

This leads to the idea to consider a sufficiently smooth one-to-one mapping $\xi : \mathcal{I} \to \tilde{\mathcal{I}}$, where $\tilde{\mathcal{I}} \subset [\alpha, \beta]$ and $\alpha, \beta \in \mathbb{R}$, i.e. $\tilde{\mathcal{I}}$ is a finite, half-open interval. In the following, we introduce the transformed function $\tilde{z}(\tau), \tau \in \tilde{\mathcal{I}}$, which satisfies $\tilde{z}(\xi(t)) = z(t)$. Using that, (3.1) can be reformulated to a system

$$\tilde{\boldsymbol{f}}(\tau, \boldsymbol{p}, \tilde{z}_1(\tau), \ldots, \tilde{z}_1^{(l_1)}(\tau), \tilde{z}_2(\tau), \ldots, \tilde{z}_i(\tau), \ldots, \tilde{z}_i^{(l_i)}(\tau), \ldots, \tilde{z}_n(\tau), \ldots, \tilde{z}_n^{(l_n)}(\tau)) = 0 \tag{3.2a}$$

$$\begin{aligned} \tilde{\boldsymbol{g}}(\boldsymbol{p}, \tilde{z}_1(\xi(c_1)), \ldots, \tilde{z}_1^{(l_1-1)}(\xi(c_1)), \ldots, z_n(\xi(c_1)), \ldots, \tilde{z}_n^{(l_n-1)}(\xi(c_1)), \ldots, \\ \ldots, \tilde{z}_1(\xi(c_q)), \ldots, \tilde{z}_1^{(l_1-1)}(\xi(c_q)), \ldots, \tilde{z}_n(\xi(c_q)), \ldots, \tilde{z}_n^{(l_n-1)}(\xi(c_q))) = 0. \end{aligned} \tag{3.2b}$$

This problem posed on $\tilde{\mathcal{I}}$ can now be solved by the usual collocation method. However, this raises the following issues:

- What choice for ξ is appropriate? - In `bvpsuite2.0` a standard method is already implemented (cf. Sec. 3.1), but it is also possible to use a user-specified transformation. (cf. Sec. 3.2).
- How can $\tilde{\boldsymbol{f}}, \tilde{\boldsymbol{g}}$ be evaluated, depending on the respective choice of ξ?

These questions will be answered in the following subsections.

3.1 Standard transformation

`bvpsuite2.0` provides the standard transformation $\xi(t) = \frac{a}{t}$. This leads for $a \neq 0$ to $\tilde{\mathcal{I}} = \xi(\mathcal{I}) = (0,1]$. For $a = 0$ a workaround is used, which will be described later.

To obtain the corresponding $\tilde{\boldsymbol{f}}$ and $\tilde{\boldsymbol{g}}$, it is obvious that the derivatives of z have to be expressed only by derivatives of $\tilde{z}$:

$$\begin{aligned} z(t) &= \tilde{z}\left(\frac{a}{t}\right), \\ z'(t) &= -\frac{a}{t^2}\,\tilde{z}'\left(\frac{a}{t}\right), \\ z''(t) &= \frac{2a}{t^3}\,\tilde{z}'\left(\frac{a}{t}\right) + \left(\frac{a}{t^2}\right)^2 \tilde{z}''\left(\frac{a}{t}\right), \\ &\dots \end{aligned}$$

Further derivatives can be computed by using the following rule:

Lemma 3.1
It holds for $z \in C^n(\mathcal{I})$, $n \in \mathbb{N} \setminus \{0\}$:

$$z^{(n)}(t) = \sum_{i=1}^{n} \alpha_{in}\,\frac{a^i}{t^{n+i}}\,\tilde{z}^{(i)}\left(\frac{a}{t}\right), \quad t \in \mathcal{I}$$

and

$$z^{(n)}\left(\frac{a}{\tau}\right) = \sum_{i=1}^{n} \alpha_{in}\,\frac{\tau^{n+i}}{a^n}\,\tilde{z}^{(i)}\left(\tau\right), \quad \tau \in (0,1],$$

where

$$\begin{aligned} \alpha_{1,n} &= (-1)^n\, n!, \\ \alpha_{i,n} &= -\alpha_{i,n-1}\,(n+i-1) - \alpha_{i-1,n-1}, \quad i = 2, \dots, n-1, \\ \alpha_{n,n} &= (-1)^n. \end{aligned}$$

Proof. Proof by induction: For $n = 1$ the statement is already proven ($\alpha_{1,1} = -1$).

For the induction step we use the chain and product rule to obtain:

$$\begin{aligned} z^{(n+1)}(t) &= -\sum_{i=1}^{n} \alpha_{in}\,(n+i)\,\frac{a^i}{t^{n+i+1}}\,\tilde{z}^{(i)}\left(\frac{a}{t}\right) + \alpha_{in}\,\frac{a^{i+1}}{t^{n+i+2}}\,\tilde{z}^{(i+1)}\left(\frac{a}{t}\right) \\ &= -\alpha_{1n}\,(n+1)\,\frac{a}{t^{n+2}}\,\tilde{z}'\left(\frac{a}{t}\right) \\ &\quad -\sum_{i=1}^{n} \alpha_{in}\,(n+i)\,\frac{a^i}{t^{n+i+1}}\,\tilde{z}^{(i)}\left(\frac{a}{t}\right) + \alpha_{i-1,n}\,\frac{a^i}{t^{n+i+1}}\,\tilde{z}^{(i)}\left(\frac{a}{t}\right) \\ &\quad - \alpha_{nn}\,\frac{a^{n+1}}{t^{2n+2}}\,\tilde{z}^{(n+1)}\left(\frac{a}{t}\right) \end{aligned}$$

$$= \underbrace{-\alpha_{1n}\,(n+1)}_{\alpha_{1,n+1}}\,\frac{a}{t^{n+2}}\,\tilde{z}'\left(\frac{a}{t}\right)$$
$$+\sum_{i=1}^{n}\underbrace{\left(-\alpha_{in}\,(n+i)\,-\alpha_{i-1,n}\right)}_{\alpha_{i,n+1}}\,\frac{a^i}{t^{n+1+i}}\,\tilde{z}^{(i)}\left(\frac{a}{t}\right)$$
$$\underbrace{-\alpha_{nn}}_{\alpha_{n+1,n+1}}\,\frac{a^{n+1}}{t^{2n+2}}\,\tilde{z}^{(n+1)}\left(\frac{a}{t}\right),$$

which shows that the statement is also true for $n+1$. □

As we can see, each derivative of $z(t)$ can be expressed as a linear combination of derivatives of $\tilde{z}(t)$. If we collect all derivatives of all components of $z(t)$ and $\tilde{z}(t)$ in matrices,

$$\boldsymbol{Z}(t) = \begin{pmatrix} z_1(t) & z_1'(t) & \dots & z_1^{(m)}(t) \\ \vdots & \vdots & \ddots & \vdots \\ z_k(t) & z_k'(t) & \dots & z_k^{(m)}(t) \\ \vdots & \vdots & \ddots & \vdots \\ z_n(t) & z_n'(t) & \dots & z_n^{(m)}(t) \end{pmatrix} \in \mathbb{R}^{n\times(m+1)},$$

where $m = \max l_i$, then it follows from Lemma 3.1 that

$$\boldsymbol{Z}(t) = \tilde{\boldsymbol{Z}}\left(\frac{a}{t}\right)\boldsymbol{A}(t),$$

with

$$A_{i+1,j+1}(t) = \alpha_{ij}\,\frac{a^i}{t^{i+j}}.$$

Furthermore, the entries of $\boldsymbol{A}(t)$ can be computed by the following recurrence relation:

$$A_{11}(t) = 1, \quad A_{i1}(t) = A_{1i}(t) = 0,\ i > 1,$$
$$A_{i+1,j+1}(t) = -\frac{A_{ij}(t)\,a}{t^2} - \frac{A_{i+1,j}(t)(i+j+1)}{t},\ i > 1, j > 1.$$

This results in the following scheme:

$$\boldsymbol{A}(t) = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & -\frac{a}{t^2} & 2\frac{a}{t^3} & -6\frac{a}{t^4} & \dots & (-1)^m m!\frac{a}{t^{m+1}} \\ 0 & 0 & \frac{a^2}{t^4} & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & (-1)^m\frac{a^m}{t^{2m}} \end{pmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}.$$

Using this tool, it is possible to express $\boldsymbol{f}$ and $\tilde{\boldsymbol{g}}$ in terms of $\tau = \frac{a}{t}$ and $\tilde{\boldsymbol{Z}}$:

$$\boldsymbol{f}(t, \boldsymbol{p}, \boldsymbol{Z}(t)) = \boldsymbol{f}\left(\frac{a}{\tau}, \boldsymbol{p}, \tilde{\boldsymbol{Z}}(\tau)\, \boldsymbol{A}\left(\frac{a}{\tau}\right)\right) =: \tilde{\boldsymbol{f}}(\tau, \boldsymbol{p}, \tilde{\boldsymbol{Z}}(\tau)),$$

$$\begin{aligned}\boldsymbol{g}(\boldsymbol{p}, \boldsymbol{Z}(c_1), \ldots, \boldsymbol{Z}(c_q)) &= \boldsymbol{g}(\boldsymbol{p}, \tilde{\boldsymbol{Z}}\left(\frac{a}{c_1}\right) \boldsymbol{A}(c_1), \ldots, \tilde{\boldsymbol{Z}}\left(\frac{a}{c_q}\right) \boldsymbol{A}(c_q)) \\ &=: \tilde{\boldsymbol{g}}(\boldsymbol{p}, \tilde{\boldsymbol{Z}}(\tilde{c}_1), \ldots, \tilde{\boldsymbol{Z}}(\tilde{c}_q)), \\ \tilde{c}_i &:= \frac{a}{c_i}.\end{aligned}$$

From these equations, we compute the required transformed Jacobians:

$$\begin{aligned}\left.\frac{\partial \tilde{\boldsymbol{f}}(\tau,\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tau))}{\partial \tilde{\boldsymbol{Z}}(\tau)}\right|_{(\tau,\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tau))} &= \left.\frac{\partial \boldsymbol{f}\left(\frac{a}{\tau},\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tau)\boldsymbol{A}\left(\frac{a}{\tau}\right)\right)}{\partial \tilde{\boldsymbol{Z}}(\tau)}\right|_{(\tau,\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tau))} \\ &= \left.\frac{\partial \boldsymbol{f}(t,\boldsymbol{p},\boldsymbol{Z}(t))}{\partial \boldsymbol{Z}(t)}\right|_{\left(\frac{a}{\tau},\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tau)\boldsymbol{A}\left(\frac{a}{\tau}\right)\right)} \boldsymbol{A}\left(\frac{a}{\tau}\right), \\ \left.\frac{\partial \tilde{\boldsymbol{g}}(\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tilde{c}_1),\ldots,\tilde{\boldsymbol{Z}}(\tilde{c}_q))}{\partial \tilde{\boldsymbol{Z}}(\tilde{c}_i)}\right|_{(\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tilde{c}_1),\ldots,\tilde{\boldsymbol{Z}}(\tilde{c}_q))} &= \left.\frac{\partial \boldsymbol{g}(\boldsymbol{p},\boldsymbol{Z}(c_1),\ldots,\boldsymbol{Z}(c_q))}{\partial \boldsymbol{Z}(c_i)}\right|_{\left(\boldsymbol{p},\tilde{\boldsymbol{Z}}(\tilde{c}_1)\boldsymbol{A}\left(\frac{a}{\tilde{c}_1}\right),\ldots\right)} \boldsymbol{A}\left(\frac{a}{\tilde{c}_i}\right).\end{aligned}$$

3.1.1 Special case: $a = 0$

For $a = 0$, the transformation would be $\xi(t) = 0$, which is obviously not a one-to-one mapping. Instead the interval $\mathcal{I} = [0, \infty)$ is split up into $\mathcal{I}_1 = [0, 1)$ and $\mathcal{I}_2 = [1, \infty)$ and Eq. (3.1) is solved separately on these intervals with respective solutions $\boldsymbol{z}^{[1]}(t)$ and $\boldsymbol{z}^{[2]}(t)$. Since we want

$$\boldsymbol{z}(t) = \begin{cases} \boldsymbol{z}^{[1]}(t) & t \in \mathcal{I}_1, \\ \boldsymbol{z}^{[2]}(t) & t \in \mathcal{I}_2 \end{cases}$$

to be a sufficiently smooth solution on the whole interval $\mathcal{I}$, we stipulate

$$\lim_{s \to 1} \left.\frac{\mathrm{d}^k z_i^{[1]}(t)}{\mathrm{d}t^k}\right|_{t=s} = \left.\frac{\mathrm{d}^k z_i^{[2]}(t)}{\mathrm{d}t^k}\right|_{t=1}, \quad i = 1 \ldots n, k = 0 \ldots l_i - 1.$$

For $\boldsymbol{z}^{[1]}(t)$ on $[0, 1)$ the collocation can be applied directly. For $\boldsymbol{z}^{[2]}(t)$ the transformation for $a = 1$, $\xi(t) = \frac{1}{t}$, is applied and $\tilde{\boldsymbol{z}}^{[2]}(t)$ is solved on $\xi(\mathcal{I}_2) = (0, 1]$.

Putting this together, Eq. (3.1) can be reformulated to

$$\boldsymbol{f}(t,\boldsymbol{p},z_1^{[1]}(t),z_1^{[1]'}(t),\dots,z_1^{[1](l_1)}(t),\dots,z_n^{[1]}(t),\dots,z_n^{[1](l_n)}(t))=0,\quad t\in(0,1) \tag{3.3a}$$

$$\tilde{\boldsymbol{f}}(t,\boldsymbol{p},\tilde{z}_1^{[2]}(t),\tilde{z}_1^{[2]'}(t),\dots,\tilde{z}_1^{[2](l_1)}(t),\dots,\tilde{z}_n^{[2]}(t),\dots,\tilde{z}_n^{[2](l_n)}(t))=0,\quad t\in(0,1) \tag{3.3b}$$

$$\begin{aligned}\boldsymbol{g}(\boldsymbol{p},z_1(c_1),\dots,z_1^{(l_1-1)}(c_1),\dots,z_n(c_1),\dots,z_n^{(l_n-1)}(c_1),\dots,\\ \dots,z_1(c_q),\dots,z_1^{(l_1-1)}(c_q),\dots,z_n(c_q),\dots,z_n^{(l_n-1)}(c_q))=0,\end{aligned} \tag{3.3c}$$

$$\forall i=1\dots n,k=0\dots l_i-1:\quad z_i^{[1](k)}(1)-\tilde{z}_i^{[2](k)}(1)=0. \tag{3.3d}$$

Now Eq. (3.3a) and Eq. (3.3b) can be combined, as well as Eq. (3.3c) and Eq. (3.3d). This leads to the following BVP on $(0,1)$:

$$\begin{aligned}\bar{\boldsymbol{f}}(t,\boldsymbol{p},z_1(t),z_1'(t),\dots,z_1^{(l_1)}(t),\dots,z_n(t),\dots,z_n^{(l_n)}(t))=0\\ \bar{\boldsymbol{g}}(\boldsymbol{p},z_1(c_1),\dots,z_1^{(l_1-1)}(c_1),\dots,z_n(c_1),\dots,z_n^{(l_n-1)}(c_1),\dots,\\ \dots,z_1(c_q),\dots,z_1^{(l_1-1)}(c_q),\dots,z_n(c_q),\dots,z_n^{(l_n-1)}(c_q),\\ \dots,z_1(1),\dots,z_1^{(l_1-1)}(1),\dots,z_n(1),\dots,z_n^{(l_n-1)}(1))=0.\end{aligned}$$

Note that the number of equations has been doubled compared to Eq. (3.1).

3.2 User-defined transformation

`bvpsuite2.0` offers users the possibility to define a custom transformation $\xi(t)$. Since we have observed that the transformed solution is highly dependent on the transformation, a method was developed which provides the same result as Lemma 3.1 for a user-defined $\xi(t)$. All that is needed from the user is $\xi(t)$ and all its derivatives (at least up to order $\max_i l_i$) and the same for the inverse. Using Lemma A.1, the same steps as used for the standard transformation can be repeated.

3.3 Remarks on implementation and usage

The standard and the user-defined transformation were implemented separately. The code uses the file ‘feval_problem.m’ for every access to the problem data (i.e. the file defining the BVP). By default, this is the standard transformation. In the `bvpsuite2.0` directory there is a file named ‘feval_problem_2.m’ provided, which can be renamed to ‘feval_problem.m’, so that the user-defined transformation is used instead.

The definition of the user-defined transformation is provided in ‘trafo.m’.

3.4 User's guide to problems on semi-infinite domains

The procedure to define a problem on an interval $[a, \infty)$ is almost exactly the same as the one for finite intervals, which was described in Section 2.3. The difference is that the definition of the interval in the problem definition-file should read:

```
case 'interval'
    ret = [a,Inf];
```

This invokes the transformation process, which was described in Section 3.1 and Section 3.2.

Example 3.2
We consider the singular BVP

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

with $z'(0) = 0$ and $z(\infty) = 0.1$.

Then the corresponding definition file looks as follows:

```
function [ret] = ex_semiinf_nonlin_2(request,z,za,zb,zc,t,p,lambda)
switch request
    case 'n'
        ret= 1;
    case 'orders'
        ret=[ 2 ];
    case 'problem'
        ret=[z(1,3) + 2/t*z(1,2) - (4*(z(1,1)+1) * z(1,1) * (z(1,1)-0.1))];
    case 'jacobian'
        %DON'T CHANGE THIS LINE:
        ret = zeros( length( feval(mfilename,'problem', zeros(feval( mfilename,'n'), max(feval( mfilename,'orders'))+1), [],[],[], 0,zeros(feval( mfilename,'parameters'),1))), feval( mfilename,'n'), max( feval(mfilename,'orders') )+1);
         ret(1, 1, 1)= -4* z(1,1) * (z(1,1)-1/10) - (4*z(1,1)+4) * (z(1,1)-1/10) - (4*z(1,1)+4)*z(1,1);
         ret(1, 1, 2)= 2/t;
```

```
         ret(1, 1, 3)=1;
    case 'interval'
        ret = [0,Inf];
    case 'linear'
        ret= 0;
    case 'parameters'
        ret=0;
    case 'c'
        ret = [];
    case 'BV'
        ret=[za(1,2)-(0);zb(1,1)-(0.1)];
    case 'dBV'
        %DON'T CHANGE THIS LINE:
        ret = zeros( max( length( feval(mfilename,'c')), ↷
           ↪ 2-length(feval(mfilename,'c'))), length( ↷
           ↪ feval( mfilename, 'problem', zeros( ↷
           ↪ feval(mfilename,'n'), ↷
           ↪ max(feval(mfilename,'orders'))+1), ↷
           ↪ [],[],[],0, zeros( feval( ↷
           ↪ mfilename,'parameters'),1))), feval( ↷
           ↪ mfilename,'n'), ↷
           ↪ max(feval(mfilename,'orders')));
         ret( 1,1 ,1 ,2 )= 1;
         ret(2,2,1,1) = 1;
    case 'dP'
        ret=[];
    case 'dP_BV'
        ret=[];
    case 'initProfile'
        ret.initialMesh = [ 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];
        ret.initialValues = [-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];
        ret.parameters=0;
    case 'EVP'
        ret=0;
end
```

We use the same settings as in Listing 2.2 and call `bvpsuite2.0` with the command:

`[x,z,s]=bvpsuite2('ex_semiinf_nonlin_2','default_settings');`

Note that we have already defined an initial profile in the definition file. Therefore passing it as a third argument in the function call is not necessary.

Since we are dealing with a non-linear problem, some iterations of the non-linear solver are performed. Afterwards, we can plot the numerical solution,

`plot(x,z);`

The result can be seen in Fig. 3.1.

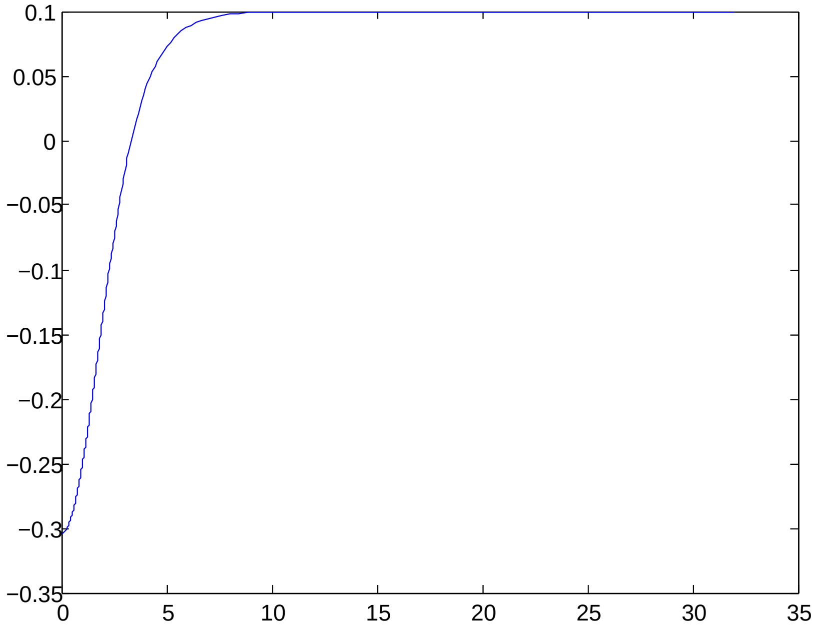


Figure 3.1: Example 3.2: The numerical solution.

Chapter 4

Treatment of Eigenvalue Problems (EVPs)

In this chapter eigenvalue problems of the form

$$G(t, z_1(t), z_1'(t), \dots, z_1^{(l_1)}(t), z_2(t), \dots, z_2^{(l_2)}(t), \dots, z_n(t), \dots, z_n^{(l_n)}(t)) = \lambda\, z(t) \tag{4.1}$$

$$\begin{aligned} B(&z_1(a), z_1'(a), \dots, z_1^{(l_1-1)}(a), z_2(a), \dots, z_2^{(l_2-1)}(a), \dots, z_n(a), \dots, z_n^{(l_n-1)}(a), \\ &z_1(b), z_1'(b), \dots, z_1^{(l_1-1)}(b), z_2(b), \dots, z_2^{(l_2-1)}(b), \dots, z_n(b), \dots, z_n^{(l_n-1)}(b)) = 0 \end{aligned} \tag{4.2}$$

are considered, where $z_i \in C^{l_i}([a, b])$ and $\lambda \in \mathbb{R}$ are unknown. Here, $G : [a, b] \times \mathbb{R}^{l_1+1} \times \dots \times \mathbb{R}^{l_n+1} \to \mathbb{R}^n$ should be linear in the $z_i^{(j)}$-components. Define $l := \sum_{i=1}^{n} l_i$, then $B : \mathbb{R}^{l_1} \times \dots \times \mathbb{R}^{l_n} \times \mathbb{R}^{l_1} \times \dots \times \mathbb{R}^{l_n} \to \mathbb{R}^l$. To uniquely define the eigenfunction $z(t) := (z_1(t), \dots, z_n(t))^T$ (for a fixed eigenvalue λ), $\|z\|^2_{L^2([a,b])} = 1$ is imposed, i.e. only normalized eigenfunctions are of interest.

4.1 Numerical realization

The first question to be answered is how the eigenvalue should be treated in the collocation scheme. In the previous version of `bvpsuite1.1` a new solution component $z_{n+1} = \lambda$ was introduced. Each symbolic occurrence of λ was replaced by z_{n+1} and additionally the differential equation $z_{n+1}' = 0$ (as λ is constant) was posed.

In the current `bvpsuite2.0` a different approach is selected and the eigenvalue λ is treated as an unknown parameter. The main advantage is that no auxiliary component has to be discretized and there is no enlargement of the equation system. This leads to a smaller system which has to be solved.

Secondly, the normalization condition is discretized in the following way. The auxiliary component

$$z_{n+1}(t) = \|z\|^2_{L^2([a,t])} = \int_a^t \sum_{i=1}^{n} z_i^2(s)\, \mathrm{d}s$$

is introduced. Then differentiation of z_{n+1} results in a new (non-linear!) differential equation,

$$z'_{n+1}(t) = \sum_{i=1}^{n} z_i^2(t). \tag{4.3}$$

Introducing a new parameter and a new component requires stating two additional boundary conditions, which we obtain from the definition of z_{n+1} and the normalization conditions,

$$z_{n+1}(a) = \int_a^a \sum_{i=1}^{n} z_i^2(s)\,\mathrm{d}s = 0 \tag{4.4}$$

and

$$z_{n+1}(b) = \int_a^b \sum_{i=1}^{n} z_i^2(s)\,\mathrm{d}s = 1. \tag{4.5}$$

Summarizing, the EVP is rewritten as a non-linear BVP (with an unknown parameter), which can be solved by the usual collocation approach. Note that in `bvpsuite1.1` one could experience problems when using Symbolic Toolbox of MATLAB. In `bvpsuite2.0` an implementation that does not use symbolic operations was sought, which is why the newly implemented MATLAB function `feval_problem` (cf. Section 3.3) is used again.

4.1.1 Extending `feval_problem` to EVPs posed on finite intervals

The task to manipulate the output of the problem file of the underlying EVP, such that it corresponds to the equivalent BVP, was fulfilled as follows. The manipulation of each output parameter that has to be changed is explained one by one.

n Because the auxiliary component is introduced, `n` is increased by 1.

problem The values of $F = G - \lambda z$ are copied and extended by the additional equation (4.3).

jacobian First, the values of the Jacobian of $F = G - \lambda z$ are copied. Then the derivatives of (4.3) are computed, i. e. $\frac{\partial F_{n+1}}{\partial z_i} = -2z_i$ and $\frac{\partial F_{n+1}}{\partial z_{n+1}} = 1$.

linear Because the non-linear equation (4.3) has been added, the system has to be solved as a non-linear system.

parameters Since the eigenvalue is seen as an unknown parameter, this value has to be increased by 1.

BV The values of B are copied and extended by the conditions (4.4) and (4.5).

dBV The values of $\mathrm{d}B$ are copied and extended by the derivatives of (4.4) and (4.5), i.e. $\frac{\partial B_{l+1}}{\partial z_{n+1}(a)} = \frac{\partial B_{l+2}}{\partial z_{n+1}(b)} = 1$.

dP The original values of `dP` and `dLambda` are combined properly.

dP_BV Since λ is not used in the extended boundary conditions, only the original values have to be copied.

4.2 User's guide to solve EVPs on finite intervals

Basically, the problem file of an EVP has the same structure as the one for a BVP. To activate the transformation described in the Section 4.1, the flag `EVP` has to be set to `1`. Then one can use the variable `lambda` to describe the problem. Note that additionally the output parameter `dLambda` has to be set, which describes the derivative of F with respect to the eigenvalue.

Example 4.1 (Bessel)
Consider the EVP

$$-z''(t) + \frac{c}{t^2}\, z(t) = \lambda\, z(t), \quad t \in (0, \pi)\,,\ c > 0 \tag{4.6}$$

$$z(0) = z(\pi) = 0. \tag{4.7}$$

This can be described by a `bvpsuite2.0` problem file as follows:

```
function [ret] = template_bvp(request,z,za,zb,zc,t,p,lambda)
c=3;
switch request
    case 'n'
        ret=  1;
    case 'orders'
        ret=[  2];
    case 'problem'
        ret=[ -z(1,3)+c/t^2*z(1,1)-lambda*z(1,1) ];
    case 'jacobian'
        %DON'T CHANGE THIS LINE:
        ret = zeros(length(feval(mfilename,'problem', zeros(feval(mfilename,'n'), max(feval(mfilename,'orders'))+1), [],[],[],0, zeros(feval(mfilename,'parameters'),1),0)), feval(mfilename,'n'), max(feval(mfilename,'orders'))+1);
```

```
         ret(1, 1, 1)=-lambda+c/t^2;
         ret(1,1,3)=-1;
    case 'interval'
        ret = [0 ,pi ];
    case 'linear'
        ret= 1;
    case 'parameters'
        ret=0;
    case 'c'
        ret = [];
    case 'BV'
        ret=[za(1,1);
            zb(1,1)];
    case 'dBV'
        %DON'T CHANGE THIS LINE:
        ret = zeros(max(length(feval(mfilename,'c')), ↷
           ↪ 2-length(feval(mfilename,'c'))), ↷
           ↪ length(feval(mfilename,'problem', ↷
           ↪ zeros(feval(mfilename,'n'), ↷
           ↪ max(feval(mfilename,'orders'))+1), ↷
           ↪ [],[],[],0, ↷
           ↪ zeros(feval(mfilename,'parameters'),1),0)), ↷
           ↪ feval(mfilename,'n'), ↷
           ↪ max(feval(mfilename,'orders')));

         ret( 1,1 ,1 ,1 )= 1;
         ret(2,2,1,1)=1;
    case 'dP'
        ret=[];
    case 'dP_BV'
        ret=[];
    case 'initProfile'
        ret.initialMesh = linspace(0,1,50);
        ret.initialValues = ret.initialMesh.* ↷
           ↪ (1-ret.initialMesh).* (1/2-ret.initialMesh);
        %ret.initialValues = ↷
           ↪ ret.initialValues/norm(ret.initialValues,2);
        ret.parameters = [ ];
        ret.lambda = -40;
    case 'EVP'
```

```
            ret = 1;
        case 'dLambda'
            ret = [ -z(1,1)];
        otherwise
            ret = 0;

end
```

Note that the parameter c is defined in the first line, so it can be modified easily.

Since the transformed problem is non-linear, an appropriate initial guess for the desired pair of eigenfunction and eigenvalue has to be chosen. However, there is an indefinite number of pairs of eigenfunctions and eigenvalues and finding the right one could be hard without further information. Therefore the MATLAB function `computeEVPStart` has been developed, which will be described in the next section.

4.2.1 `computeEVPStart`: a Matrix method

The idea behind this approach to find approximations of solutions of an EVP is to discretize the equations as in the standard collocation code, but solve the resulting system as a matrix-eigenvalue problem. In particular, the left-hand sides and right hand sides are discretized separately by matrices $A \in \mathbb{R}^{N \times N}$ and $M \in \mathbb{R}^{N \times N}$. Then the matrix EVP $A\zeta = \lambda M \zeta$ is solved for the discrete eigenvectors ζ_i and corresponding eigenvalues λ_i (e.g. by the built-in MATLAB-function `eig`, which uses the generalized Schur decomposition). Therefore, exactly N approximations can be computed in this way. ζ_i can be interpreted as the coefficients of a polynomial, approximating the i-th eigenfunction, whereas λ_i is the approximation for the i-th eigenvalue.

For the implementation, a trick had to be used. First, the discretization depends only on the derivatives of the equations - not on the equations themselves - since the EVP is linear. Then $\frac{\mathrm{d}F_j}{\mathrm{d}z_k}$ gives the coefficient of z_k in the j-th equation. So each coefficient of the right hand side of the equation has to contain the factor λ. By setting λ to 0, only the coefficients of the left-hand side remain, while $\lambda = 1$ gives the coefficients of both sides. Now the two parts can be separated easily. Note, that by this trick more complex right hand sides could also be handled. The boundary conditions and conditions of continuity are handled in a similar manner.

Since the approximation quality of the algebraic eigenvectors decreases with increasing indices, it is appropriate to compute more eigenvalues and eigenvectors than necessary. In our code five times as much as necessary are computed.

The call of `computeEVPStart` is similar to `bvpsuite1.1`. The first and second parameters are the names of the problem and the settings file. The third parameter denotes the number of desired initial guesses. The function returns a cell array of structure arrays as first output parameter. One of these structure arrays contains the following values:

- **initialMesh** The mesh on which the initial guess is given.
- **initialValues** The values of the initial guess on the mesh.
- **lambda** The guess of λ.

In practice, one of these structure arrays is taken as third input parameter of `bvpsuite2.0`, as shown in the following example.

Example 4.2 (Bessel, continued)
To find initial guesses for the first N eigenvalues, the module computeEVPStart can be used. The module computeEVPStart requires three arguments: the name of the problem-defining file, the file that contains the algorithm-defining parameters and the minimal number of desired eigenpairs.

```
[initProfiles,initialEVs] = ↷
  ↪ computeEVPStart('evp_bessel','default_settings',N);
```

Note: computeEVPStart may compute more than N eigenpairs. Therfore only the first N are kept.

```
initProfiles = initProfiles(1:N);
initialEVs = initialEVs(1:N);
```

For the result, a vector of length N is preallocated and cell arrays for colors and styles for plotting the eigenfunctions are defined.

```
EVs = zeros(N,1);

colors={'b','g','r','c','m','y','k'};
styles={'-',':','-.','--'};
```

Next, we iterate over all initial guesses and try to obtain a good approximation for each one.

The package `bvpsuite2.0` requires the problem-defining and algorithm-defining files as the first two arguments. The third (optional) argument is the initial profile. The first two output arguments are the mesh and corresponding values of the eigenfunction. The third one is a structure containing in particular the eigenvalue.

```
for i=1:N
    fprintf('\r\nSolve the EVP for the eigenvalue ↷
      ↪ #%d:\n',i);
```

```
    [x1,valx1,sol] = bvpsuite2('evp_bessel', 'default_settings', initProfiles{i});

    EVs(i) = sol.lambda;
    legends{i} = sprintf('\\lambda_{%d} = %4.1f',i,sol.lambda);
    plot(x1, valx1, strcat(colors{1+mod(i,7)},styles{ceil(i/7)}));
    hold on
end
hold off
legend(legends);
```

During the computation, information about the mesh adaption is displayed:

```
Solve the EVP for the eigenvalue #1:
Density update N=118

Solve the EVP for the eigenvalue #2:
Density update N=118

Solve the EVP for the eigenvalue #3:
Density update N=118

Solve the EVP for the eigenvalue #4:
Density update N=118
Density update N=118

Solve the EVP for the eigenvalue #5:
Density update N=118
Density update N=118

Solve the EVP for the eigenvalue #6:
Density update N=118
Density update N=118

Solve the EVP for the eigenvalue #7:
Density update N=118
Density update N=118
```

Finally, we display the resulting eigenvalues and compare them with the initial guesses.

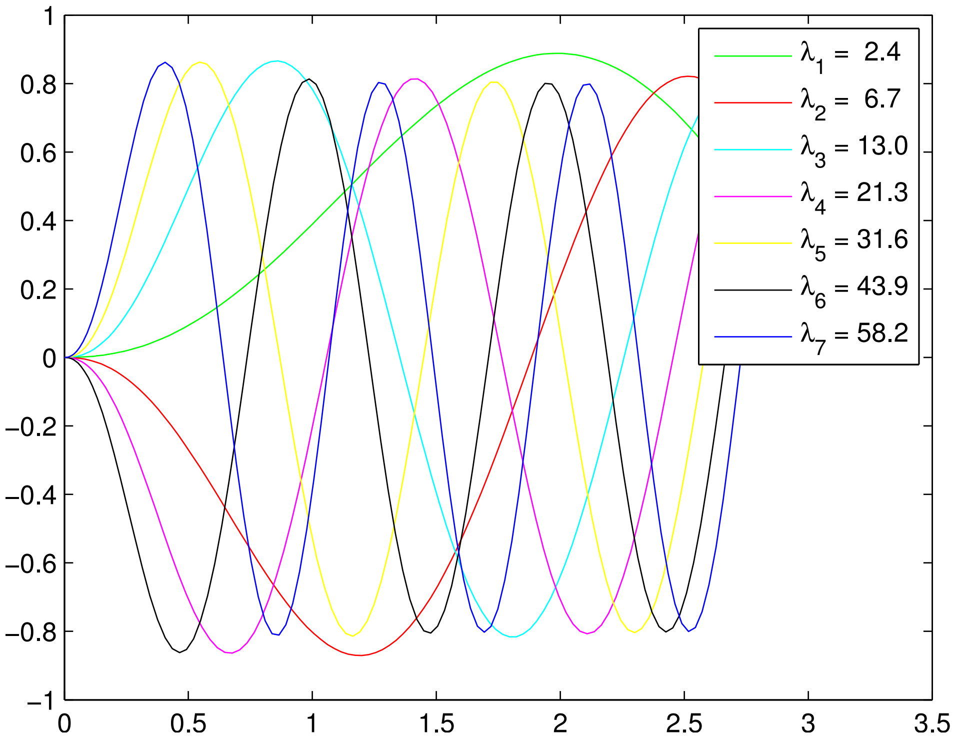


Figure 4.1: Example 4.2: Approximations of the first seven eigenfunctions.

```
fprintf('\r\nCompare the initial guesses with the ↷
   ↪ computed eigenvalues:\n');
fprintf('Initial guess  |  Computed value  |  Difference ↷
   ↪ \n');
fprintf('%14.6f | %16.6f | %2.4e ↷
   ↪ \n',[initialEVs,EVs,abs(initialEVs-EVs)]');
```

```
Compare the initial guesses with the computed eigenvalues:
Initial guess  |  Computed value  |  Difference
      2.417106 |         2.417106 | 5.9977e-08
      6.723654 |         6.723653 | 6.4377e-07
     13.027504 |        13.027501 | 3.5746e-06
     21.330745 |        21.330728 | 1.7108e-05
     31.633815 |        31.633736 | 7.8923e-05
     43.936988 |        43.936647 | 3.4136e-04
     58.240947 |        58.239508 | 1.4392e-03
```

Remark: This example shows that the quality of the initial guesses decreases as the index increases.

4.3 EVPs on $[a, \infty)$

By combining the methods for BVPs posed on $[a, \infty)$ and EVPs on finite intervals, it is possible to easily solve EVPs on $[a, \infty)$. First, the EVP is transformed to an EVP on $(0, 1)$. Afterwards the transformed EVP is treated as a nonlinear BVP described in Section 4.1.

Chapter 5

Error estimation and mesh adaptation

The algorithms of error estimation and mesh adaptation in `bvpsuite2.0` were almost completely unchanged from `bvpsuite1.1`. Only the data structure had to be adapted. For the theoretical background we refer to [24].

The error estimation and mesh adaptation are invoked by setting the respective properties in the settings file as shown in Example 2.1. The following combinations of parameter values are possible:

		errorEstimate	
		0	1
meshAdaptation	0	solving on a fixed mesh	only error estimation
	1	error estimation and mesh adaptation	

Chapter 6

Differential algebraic equations (DAEs)

`bvpsuite2.0` also supports solving DAEs, which should for reasons of simplicity here have the form

$$\boldsymbol{f}(t, \boldsymbol{z}(t), \boldsymbol{z}'(t)) = 0, \quad t \in \mathcal{I} = [a, b]$$

with boundary conditions

$$\boldsymbol{g}(\boldsymbol{z}(a), \boldsymbol{z}(b)) = 0.$$

A DAE is then characterized by the singularity of $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'(t)}$, i. e. rank $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'(t)} < n$. Thus, it is not possible to extract an ODE system by using the implicit function theorem.

This leads to the notion of the *differentiation index*, which gives information on the distance between the DAE and an ODE, which has the same set of solutions (after adding consistent boundary conditions). The index counts how often $\boldsymbol{f}$ has to be differentiated to extract an ODE, i. e. ODEs are Index-0 DAEs.

Example 6.1 (Differentiation)
Let $q(t) \in C^1([a, b])$. Then

$$\boldsymbol{f}(t, \boldsymbol{z}(t), \boldsymbol{z}'(t)) := \begin{pmatrix} z_1(t) - z_2'(t) \\ z_2(t) - q(t) \end{pmatrix} = \boldsymbol{0}$$

has the solution

$$\boldsymbol{z}(t) = \begin{pmatrix} q'(t) \\ q(t) \end{pmatrix}.$$

(Note: no boundary conditions are required!)

It holds:

$$\text{rank}\,\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'(t)} = \text{rank}\begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix} = 1 < 2.$$

However, by considering $f_1' = z_1'(t) - z_2''(t)$, $f_2'(t) = z_2'(t) - q'(t)$ and $f_2''(t) = z_2''(t) - q''(t)$, the DAE can be rewritten as

$$\bar{\boldsymbol{f}}(t, \boldsymbol{z}(t), \boldsymbol{z}'(t)) := \begin{pmatrix} z_1'(t) - q''(t) \\ z_2' - q'(t) \end{pmatrix} = \boldsymbol{0},$$

which is obviously an ODE.

By adding the consistent initial condition

$$\boldsymbol{z}(0) = \begin{pmatrix} q'(0) \\ q(0) \end{pmatrix},$$

we receive a properly stated initial value problem.

We have seen that differentiating $\boldsymbol{f}$ twice was necessary to obtain an ODE. Thus, the differentiating $q(t)$ can be seen as an Index-2-DAE.

6.1 Solving Index-1-DAEs

The numerical treatment of Index-1-DAEs is well investigated and collocation is an appropriate method for it. We will illustrate the procedure using the following example.

Example 6.2 (Hydrodynamic model of a semiconductor)
The following equations describe a hydrodynamic model for semiconductors in the isentropic case,

$$\begin{aligned} \varphi'(x) &= \rho(x)E(x) - \alpha J, \\ E'(x) &= \rho(x) - 1, \\ \varphi(x) &= \frac{J^2}{\rho(x)} + \rho(x), \end{aligned}$$

with

$$x \in [0, b], \quad \rho(0) = \rho(b) = \bar{\rho}.$$

These equations form a nonlinear Index-1-DAE system, for which no closed solution is available. Therefore we have to consider how to approximate the solution numerically.

The flow described by the solution can be divided into three categories, which also have a physical meaning.

- *subsonic* flow for $\rho > J$,
- *transonic* flow, if $\bar{\rho} > J$ and $\rho < J$ on a subinterval (a_ρ, b_ρ),
- *supersonic* flow for $\rho < J$.

The case of the *subsonic* flow can be treated easily with `bvpsuite2.0`.

We set $J = \frac{1}{2}, \alpha = 0, \bar{\rho} = 3, b = 10.3$. As initial guess constants are chosen, which satisfy the equations, but not the boundary conditions: $\varphi = 1.25, E = 0, \rho = 1$.

The corresponding bvpfile reads:

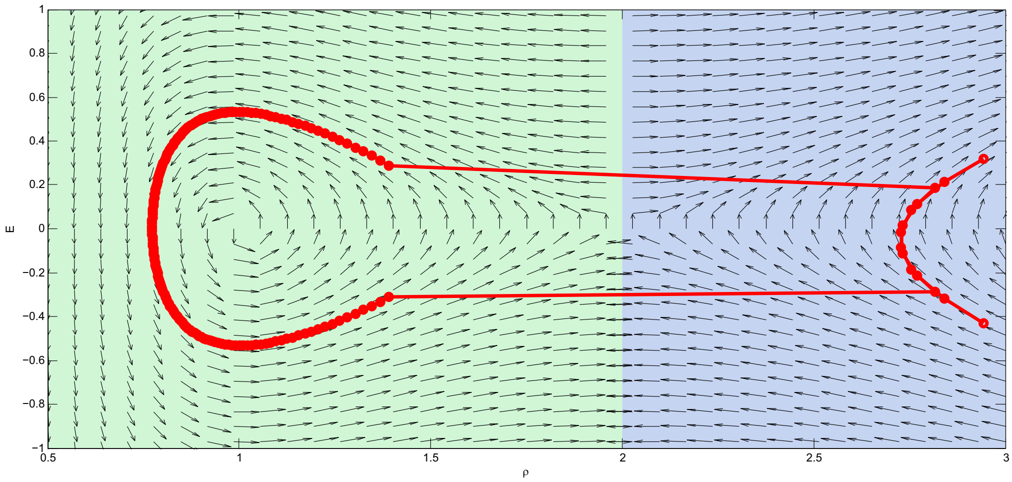


Figure 6.1: Example 6.2: Phase portrait: green - supersonic region, blue - subsonic region, red - transsonic flow.

```
function [ret] = semiconductor(request,z,za,zb,zc,t,p,lambda)
alpha=0
J=1/2;
rhobar=3;
b=10.3;
switch request
    case 'n'
        ret=  3;
    case 'orders'
        ret=[ 1 1 0 ];
    case 'problem'
        ret=[ z(1,2)-z(2,1)*z(3,1)+alpha*J;
               z(2,2)-z(3,1)+1;
               z(1,1)-J^2/z(3,1)-z(3,1)
            ];
    case 'jacobian'
        %DON'T CHANGE THIS LINE:
        ret = zeros(length(feval(mfilename, 'problem', zeros(feval(mfilename,'n'), max(feval(mfilename,'orders'))+1), [],[],[],0,
```

```
                zeros(feval(mfilename,'parameters'),1))),
                feval(mfilename,'n'),
                max(feval(mfilename,'orders')) + 1);
            ret(1, 1, 2)= 1;
             ret(1, 2, 1)= -z(3,1);
             ret(1, 3, 1)= -z(2,1);
             ret(2, 2, 2)= 1;
             ret(2, 3, 1)= -1;
             ret(3, 1, 1)= 1;
             ret(3, 3, 1)= J^2/z(3,1)^2-1;
        case 'dJ'
             ret=[alpha;0;-2*J/z(3,1)];
        case 'interval'
            ret = [0 , b];
        case 'linear'
            ret= 0;
        case 'parameters'
            ret=0;
        case 'c'
            ret = [];
        case 'BV'
%             ret=[za(3,1)-rhobar;
%                   zb(3,1)-rhobar];
             ret=[za(1,1)-J^2/rhobar-rhobar;
                  zb(1,1)-J^2/rhobar-rhobar];
        case 'dBV'
            %DON'T CHANGE THIS LINE:
            ret = zeros(max(length(feval(mfilename,'c')),
                2-length(feval(mfilename,'c'))),
                length(feval(mfilename,'problem',
                zeros(feval(mfilename,'n'),
                max(feval(mfilename,'orders'))+1),
                [],[],[],0,
                zeros(feval(mfilename,'parameters'),
                1))),feval(mfilename,'n'),
                max(feval(mfilename,'orders')));
             ret( 1,1 ,1 ,1 )= 1;
             ret( 2,2 ,1 ,1 )= 1;
        case 'dJ_BV'
            ret=[-2*J/rhobar;-2*J/rhobar];
        case 'dP'
```

```
        ret=[0;0;0];
    case 'dP_BV'
        ret=[0;0];
    case 'initProfile'
        ret.initialMesh = linspace(0,b,50);
        ret.initialValues = [
            rhobar*(1+J^2)*ones(1,length(ret.initialMesh));
            0*ones(1,length(ret.initialMesh));
            rhobar*ones(1,length(ret.initialMesh))];
    case 'EVP'
        ret = 0;
    case 'dLambda'
        ret = [];
    case 'J0'
        ret=alpha0;
    case 'J_target'
        ret=0;
    otherwise
        ret = 0;
end
```

In this case, the Fast-Frozen-Newton method converges and gives after some adaptations of the mesh the following solution, see Fig. 6.2.

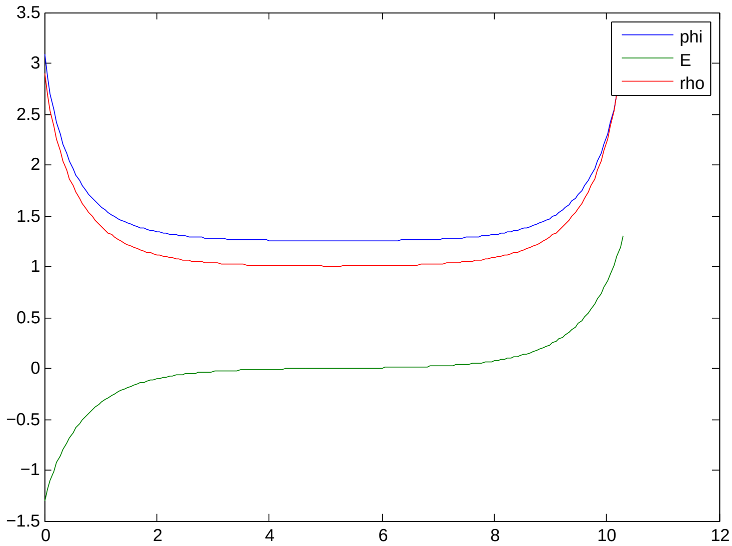


Figure 6.2: Example 6.2: The solution for the subsonic case.

For the transonic and supersonic cases, regularization and the path-following method had to be used to achieve convergence of the non-linear solver.

In conclusion, we can see that `bvpsuite2.0` can handle Index-1-DAEs exactly the same way as ODEs.

6.2 Higher-index DAEs

While we have seen in Section 6.1 that DAEs with Index 1 can be treated like ODE systems, the situation for DAEs with a higher index is much more difficult. These problems are considered as ill-posed (by definition of Hadamard), since the solution's behavior is not continuously dependent on the (initial) data.

This behavior can be seen by continuing Example 6.1.

Example 6.3 (Example 6.1 cont.)
We are using `bvpsuite2.0` to solve Example 6.1 for $q(t) := \sin(t)$ on $[0, 4\pi]$ with different (uniformly distributed) grids. The initial condition $z_2(0) = 0$ has to be set in order to close the system of equations. The computations were carried out with four Gaussian collocation points.

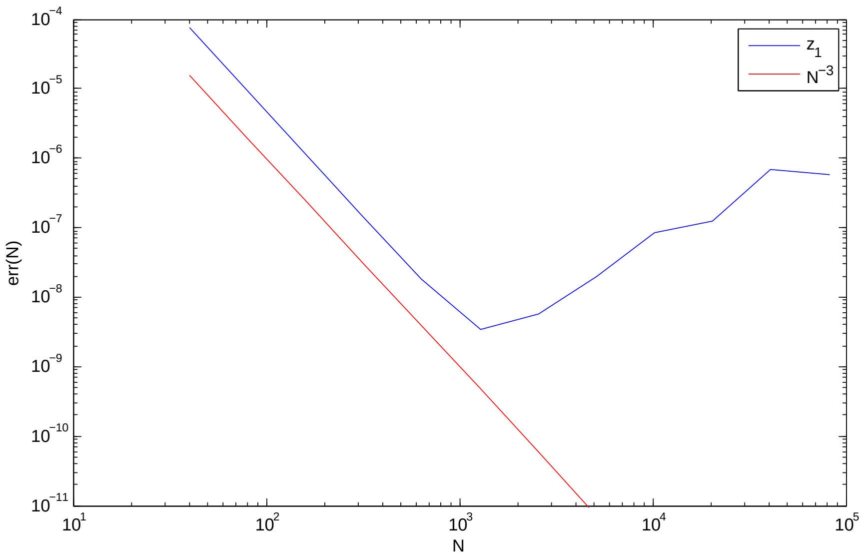


Figure 6.3: Example 6.3: Double-logarithmic plot of $\max_t |z_1(t) - \cos(t)|$ in dependence of the grid size N.

It is evident from Fig. 6.3 and Table 6.1 that convergence is observed up to a critical grid size of 2560 points. For finer grids, the error increases, which shows the instability of collocation for higher-index DAEs.

N	error	order
40	7.416e-05	
80	9.238e-06	3.005
160	1.153e-06	3.001
320	1.442e-07	2.999
640	1.807e-08	2.996
1280	3.492e-09	2.372
2560	5.783e-09	-0.727
5120	1.990e-08	-1.783
10240	8.581e-08	-2.108
20480	1.263e-07	-0.558
40960	6.896e-07	-2.447
81920	5.766e-07	0.258

Table 6.1: Example 6.3: Error $|z_1(t) - \cos(t)|$ and the corresponding convergence orders.

In addition, in [45] collocation was tested with further higher-index examples - the breakdown of convergence was evident.

6.2.1 Over-determined collocation

Since we are dealing with ill-posed problems, the idea of regularization seems rather natural. A popular regularization technique for ill-posed problems is over-determination. In this section we investigate how this can be applied in the context of collocation for higher-index DAEs.

To implement over-determination, we proceed as described in Section 2.2, with the difference that Eq. (2.4) is evaluated not only at the collocation points, but also at additional points. This leads to a system of equations where the number of equations is greater than the number of unknowns.

A disadvantage of over-determination is the increased size of the number of equations and the necessity to use a least-square solver, which increases computation time.

This idea was implemented for linear DAEs and tested for some examples, for which good results could be achieved. First, we show the improvement of results for Example 6.3.

Note: This functionality is currently not included in `bvpsuite2.0` *since for the moment it is usable for experimental purposes only. The current version of the code can be found in Appendix B.*

Example 6.4 (Example 6.3 cont.)
As additional points we choose the midpoints of the intervals formed by collocation (Gauß points) and grid points (cf. Fig. 6.4).

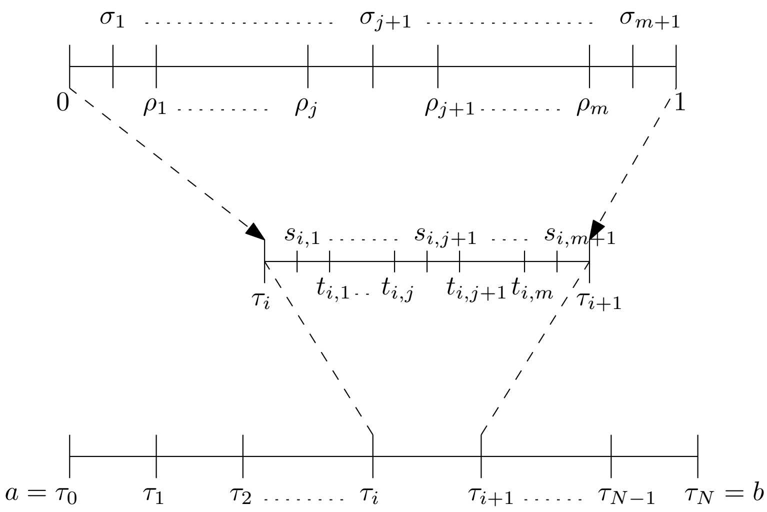


Figure 6.4: Scheme of additional points for over-determined collocation.

Using the same method as in Example 6.3 we obtain results shown in Table 6.2.

N	error	order
40	1.116e-05	0
80	7.088e-07	3.977
160	4.447e-08	3.994
320	2.782e-09	3.998
640	1.739e-10	3.999
1280	1.111e-11	3.967
2560	1.860e-11	-0.742
5120	8.966e-11	-2.268

Table 6.2: Example 6.4: Error $|z_1(t) - \cos(t)|$ and the corresponding convergence orders.

We observe that this procedure results in a better error level, and even the order of convergence increases by one.

Example 6.5 (Index-3 DAE)
We consider the Index-3 system

$$\begin{aligned} &x_2'(t) + x_1(t) = q_1(t), \\ &t\eta x_2'(t) + x_3'(t) + (\eta+1)x_2(t) = q_2(t), \\ &t\eta x_2(t) + x_3(t) = q_3(t), \end{aligned} \tag{6.1}$$

with

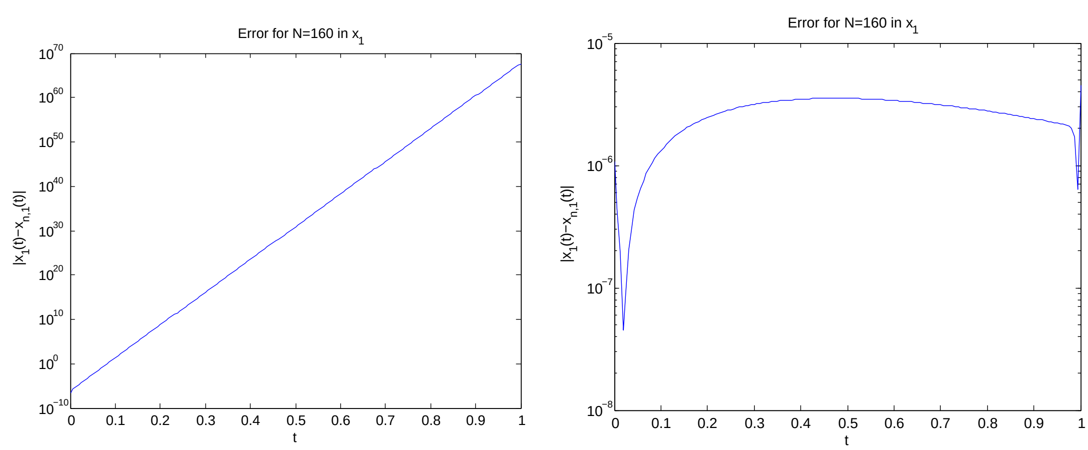
$$
\begin{aligned}
q_1(t) &= \Big[-2\mathrm{e}^{-2t}\sin(t) + \mathrm{e}^{-2t}\cos(t)\Big] + \mathrm{e}^{-t}\sin(t) \\
&= \mathrm{e}^{-2t}\left[-2\sin(t) + \cos(t)\right] + \mathrm{e}^{-t}\sin(t), \\
q_2(t) &= t\eta\Big[-2\mathrm{e}^{-2t}\sin(t) + \mathrm{e}^{-2t}\cos(t)\Big] + \Big[-\mathrm{e}^{-t}\cos(t) - \mathrm{e}^{-t}\sin(t)\Big] + \\
&\quad + (\eta+1)\Big[\mathrm{e}^{-2t}\sin(t)\Big] \\
&= \mathrm{e}^{-2t}\left[-2t\eta\sin(t) + t\eta\cos(t) + (\eta+1)\sin(t)\right] - \mathrm{e}^{-t}\left[\cos(t) + \sin(t)\right], \\
q_3(t) &= \mathrm{e}^{-2t}t\eta\sin(t) + \mathrm{e}^{-t}\cos(t).
\end{aligned}
$$

No boundary conditions are required to determine the solution

$$
\begin{aligned}
x_1 &= \mathrm{e}^{-t}\sin(t), \\
x_2 &= \mathrm{e}^{-2t}\sin(t), \\
x_3 &= \mathrm{e}^{-t}\cos(t).
\end{aligned}
$$

Therefore we do not impose boundary conditions in the over-determined variant. However for the non-over-determined method we choose the consistent initial conditions $x_2(0) = 0$, $x_3(0) = 1$.

For both variants, the normal and over-determined collocation was tested with $\eta = -5$, $N = 160$ (number of uniformly distributed gridpoints), and three uniformly distributed collocation points. We compare the errors in x_1, since this component is the most critical one.



(a) Classical collocation **(b)** Over-determined collocation

Figure 6.5: Example 6.5: Comparsion of errors in x_1 for classical and over-determined collocation.

We observe that using the method of over-determination leads to a much better result, where the error is bounded in contrast to the obvious divergence of the standard variant.

We now turn to another example.

Example 6.6 (Index-2 DAE)
Consider

$$\begin{aligned}
&x_1'(t) + \lambda x_1(t) - x_2(t) - x_3(t) = g_1(t), \\
&x_2'(t) + (\eta t(1-\eta t) - \eta)x_1(t) + \lambda x_2(t) - \eta t x_3(t) = g_2(t), \\
&(1-\eta t)x_1(t) + x_2(t) = g_3(t),
\end{aligned} \tag{6.2}$$

with

$$\begin{aligned}
g_1(t) &= (\lambda - 1)\mathrm{e}^{-t}\sin(t) - \mathrm{e}^{-2t}\sin(t), \\
g_2(t) &= \mathrm{e}^{-2\,t}\cos(t) - 2\,\mathrm{e}^{-2\,t}\sin(t) - \mathrm{e}^{-t}\sin(t)\,\left(\eta + \eta\,t\,\left(\eta\,t - 1\right)\right) + \\
&\quad + \lambda\,\mathrm{e}^{-2\,t}\sin(t) - \eta\,t\,\mathrm{e}^{-t}\cos(t)\,, \\
g_3(t) &= \mathrm{e}^{-2\,t}\sin(t) - \mathrm{e}^{-t}\sin(t)\,\left(\eta\,t - 1\right).
\end{aligned}$$

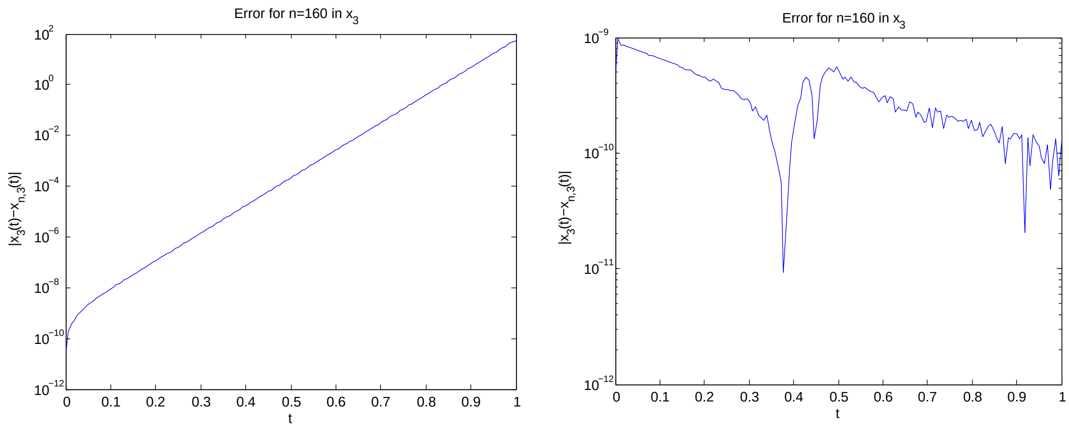
For the non-over-determined variant, we again use boundary conditions to close the system of equations,

$$\begin{aligned}
x_1(0) &= 0, \\
x_2(0) &= 0.
\end{aligned}$$

Then the exact solution reads:

$$\begin{aligned}
x_1 &= \mathrm{e}^{-t}\sin(t), \\
x_2 &= \mathrm{e}^{-2t}\sin(t), \\
x_3 &= \mathrm{e}^{-t}\cos(t).
\end{aligned}$$

Here we compare the error in the third component, since now this is the most critical one.



(a) Classical collocation **(b)** Over-determined collocation

Figure 6.6: Example 6.6: Comparsion of errors in x_3 for classical and over-determined collocation.

These figures show again the superiority of the over-determined method and the breakdown of classical collocation.

Further experiments and results

In the sequel, we provide the detailed results for the standard method and over-determined collocation.

Example 6.5 (Index-3)	2 uniform collocation points	$\eta = -2$
x_1 grid size	error	order
20	1.46e-02	0.0
40	7.04e-03	1.1
80	3.46e-03	1.0
160	1.72e-03	1.0
320	8.54e-04	1.0
640	4.26e-04	1.0
x_2 grid size	error	order
20	1.40e-04	0.0
40	3.32e-05	2.1
80	8.10e-06	2.0
160	2.00e-06	2.0
320	4.97e-07	2.0
640	1.24e-07	2.0
x_3 grid size	error	order
20	1.75e-04	0.0
40	4.13e-05	2.1
80	1.01e-05	2.0
160	2.48e-06	2.0
320	6.16e-07	2.0
640	1.54e-07	2.0

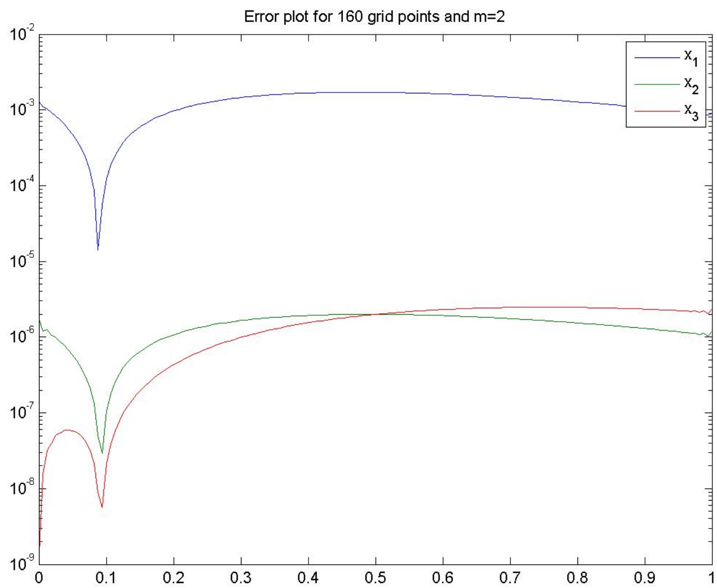


Table 6.3: Example 6.5 (Index-3): Results obtained with two collocation points for the standard over-determined collocation.

Example 6.5 (Index-3)	3 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	4.01e-04	0.0
40	9.20e-05	2.1
80	2.20e-05	2.1
160	5.38e-06	2.0
320	1.33e-06	2.0
640	3.30e-07	2.0

x_2 grid size	error	order
20	3.18e-06	0.0
40	3.54e-07	3.2
80	4.17e-08	3.1
160	5.06e-09	3.0
320	6.23e-10	3.0
640	7.71e-11	3.0

x_3 grid size	error	order
20	6.35e-06	0.0
40	7.07e-07	3.2
80	8.33e-08	3.1
160	1.01e-08	3.0
320	1.25e-09	3.0
640	1.54e-10	3.0

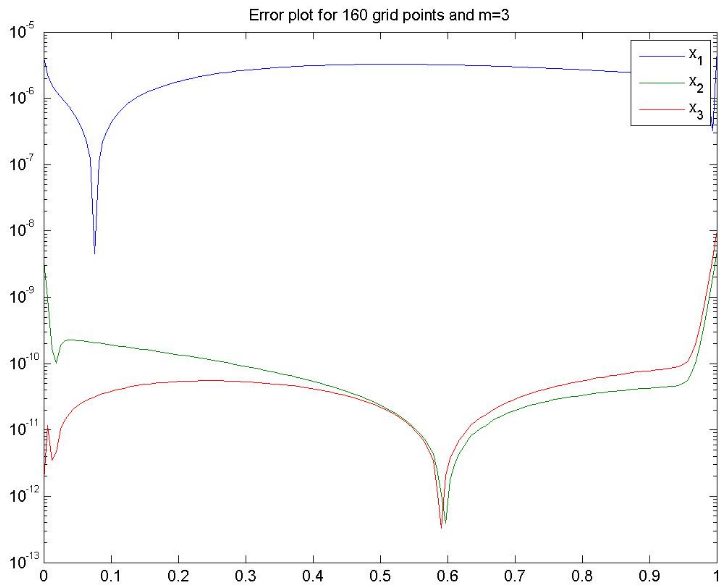


Table 6.4: Example 6.5 (Index-3): Results obtained with three collocation points for the standard over-determined collocation.

Example 6.5 (Index-3)	4 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	3.58e-06	0.0
40	4.14e-07	3.1
80	4.99e-08	3.1
160	7.16e-09	2.8
320	1.12e-08	-0.6
640	6.14e-08	-2.5

x_2 grid size	error	order
20	1.19e-08	0.0
40	6.69e-10	4.1
80	4.01e-11	4.1
160	3.00e-12	3.7
320	3.83e-12	-0.4
640	9.35e-12	-1.3

x_3 grid size	error	order
20	2.31e-08	0.0
40	1.32e-09	4.1
80	7.84e-11	4.1
160	4.87e-12	4.0
320	2.90e-12	0.7
640	7.31e-12	-1.3

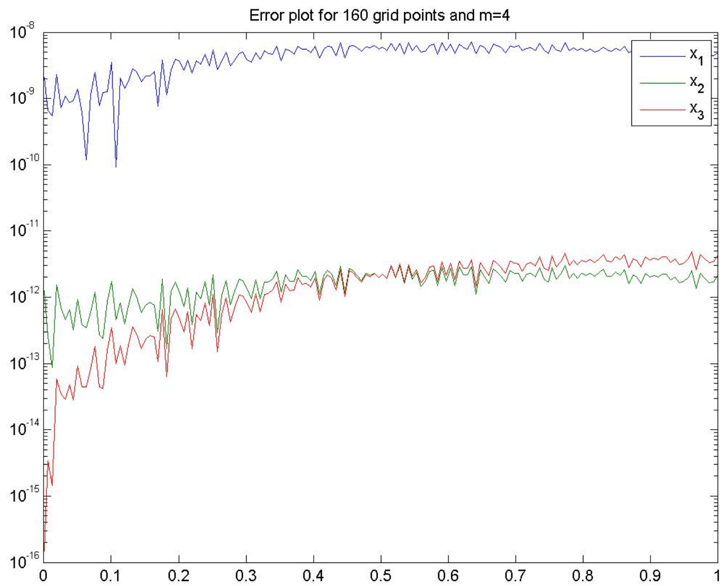


Table 6.5: Example 6.5 (Index-3): Results obtained with four collocation points for the standard over-determined collocation.

Example 6.5 (Index-3)	6 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	2.11e-10	0.0
40	8.41e-10	-2.0
80	2.68e-09	-1.7
160	1.72e-08	-2.7
320	1.02e-07	-2.6
640	3.72e-07	-1.9

x_2 grid size	error	order
20	4.79e-13	0.0
40	8.88e-13	-0.9
80	1.34e-12	-0.6
160	4.41e-12	-1.7
320	1.35e-11	-1.6
640	2.14e-11	-0.7

x_3 grid size	error	order
20	7.24e-13	0.0
40	5.33e-13	0.4
80	1.66e-12	-1.6
160	2.59e-12	-0.6
320	8.05e-12	-1.6
640	1.56e-11	-1.0

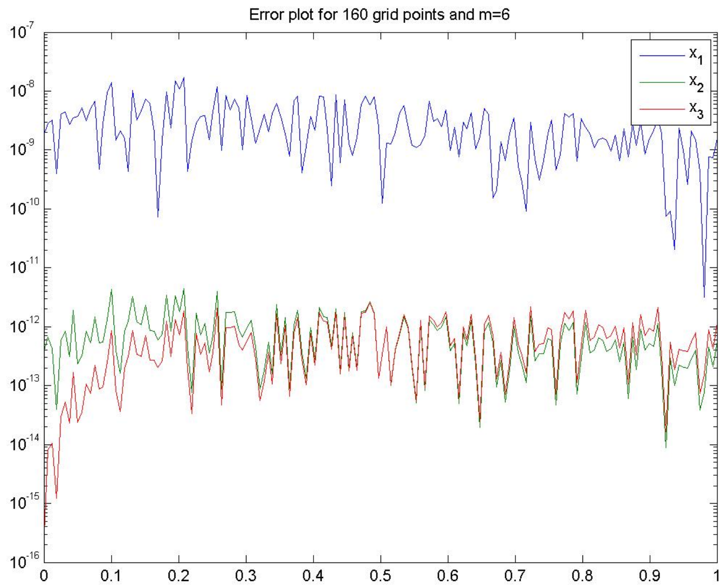


Table 6.6: Example 6.5 (Index-3): Results obtained with six collocation points for the standard over-determined collocation.

Example 6.6 (Index-2)	2 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	7.37e-03	0.0
40	2.41e-03	1.6
80	7.41e-04	1.7
160	2.01e-04	1.9
320	5.12e-05	2.0
640	1.28e-05	2.0

x_2 grid size	error	order
20	1.92e-01	0.0
40	6.26e-02	1.6
80	1.93e-02	1.7
160	5.22e-03	1.9
320	1.33e-03	2.0
640	3.34e-04	2.0

x_3 grid size	error	order
20	1.95e-01	0.0
40	6.36e-02	1.6
80	1.96e-02	1.7
160	5.35e-03	1.9
320	1.39e-03	1.9
640	6.13e-04	1.2

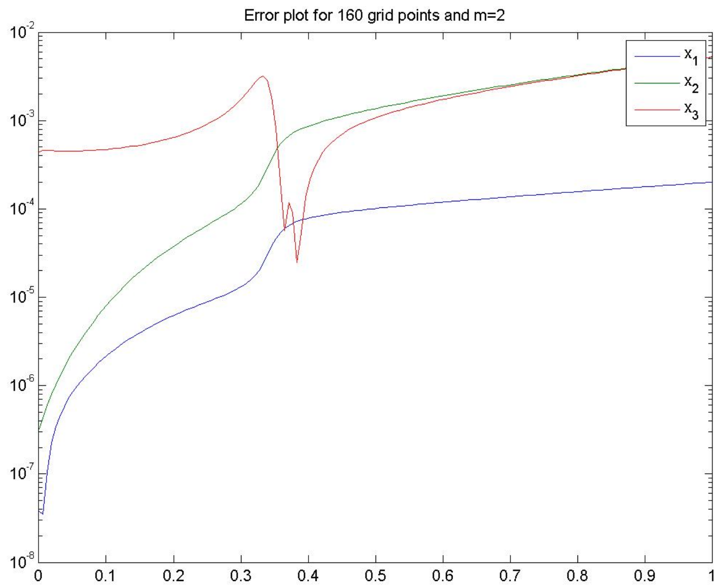


Table 6.7: Example 6.6 (Index-2): Results obtained with two collocation points for the standard over-determined collocation.

Example 6.6 (Index-2)	3 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.59e-06	0.0
40	9.22e-07	2.0
80	2.22e-07	2.1
160	5.44e-08	2.0
320	1.35e-08	2.0
640	3.36e-09	2.0

x_2 grid size	error	order
20	9.38e-05	0.0
40	2.40e-05	2.0
80	5.77e-06	2.1
160	1.41e-06	2.0
320	3.50e-07	2.0
640	8.72e-08	2.0

x_3 grid size	error	order
20	1.89e-04	0.0
40	5.92e-05	1.7
80	1.49e-05	2.0
160	2.90e-06	2.4
320	5.80e-07	2.3
640	1.26e-07	2.2

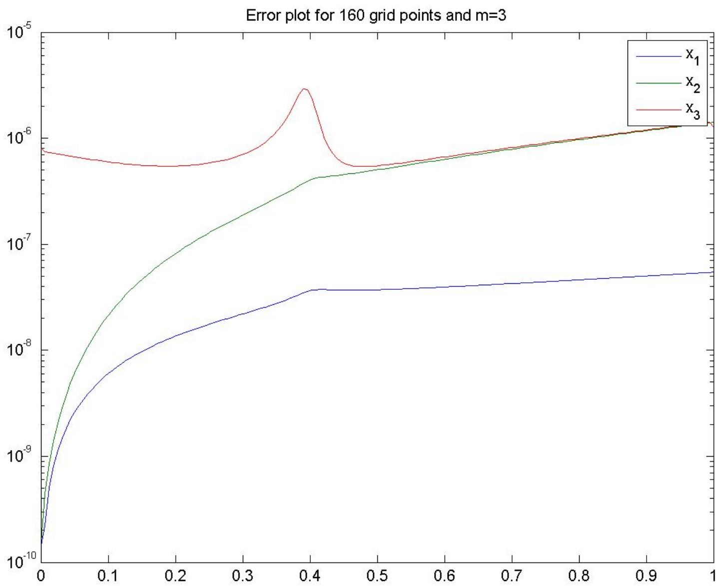


Table 6.8: Example 6.6 (Index-2): Results obtained with three collocation points for the standard over-determined collocation.

Example 6.6 (Index-2)	4 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.17e-08	0.0
40	3.02e-09	3.4
80	2.74e-10	3.5
160	2.02e-11	3.8
320	1.57e-12	3.7
640	5.75e-13	1.5

x_2 grid size	error	order
20	8.25e-07	0.0
40	7.85e-08	3.4
80	7.13e-09	3.5
160	5.24e-10	3.8
320	3.96e-11	3.7
640	1.45e-11	1.5

x_3 grid size	error	order
20	7.42e-07	0.0
40	1.38e-07	2.4
80	1.77e-08	3.0
160	1.64e-09	3.4
320	1.39e-10	3.6
640	3.68e-10	-1.4

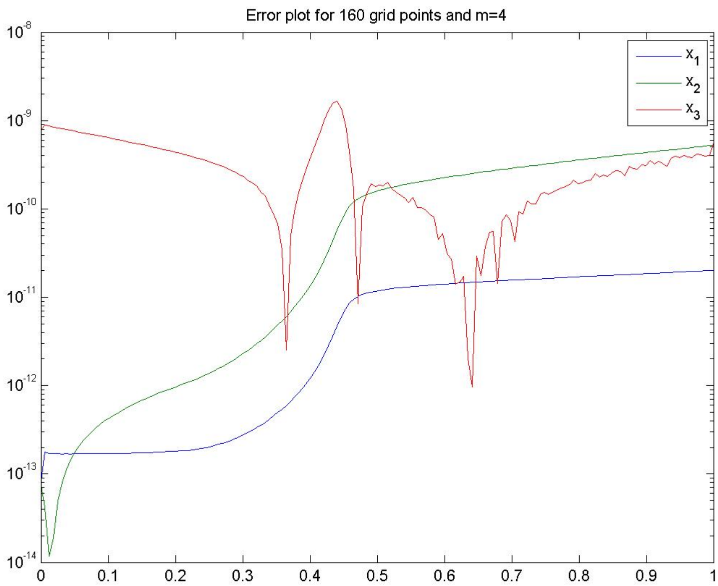


Table 6.9: Example 6.6 (Index-2): Results obtained with four collocation points for the standard over-determined collocation.

Example 6.6 (Index-2)	6 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	6.13e-13	0.0
40	4.87e-13	0.3
80	1.99e-13	1.3
160	2.78e-13	-0.5
320	6.90e-13	-1.3
640	9.63e-13	-0.5

x_2 grid size	error	order
20	1.02e-11	0.0
40	1.25e-11	-0.3
80	5.10e-12	1.3
160	6.82e-12	-0.4
320	1.75e-11	-1.4
640	2.34e-11	-0.4

x_3 grid size	error	order
20	3.62e-11	0.0
40	5.12e-11	-0.5
80	2.02e-10	-2.0
160	1.25e-10	0.7
320	4.52e-10	-1.9
640	1.29e-09	-1.5

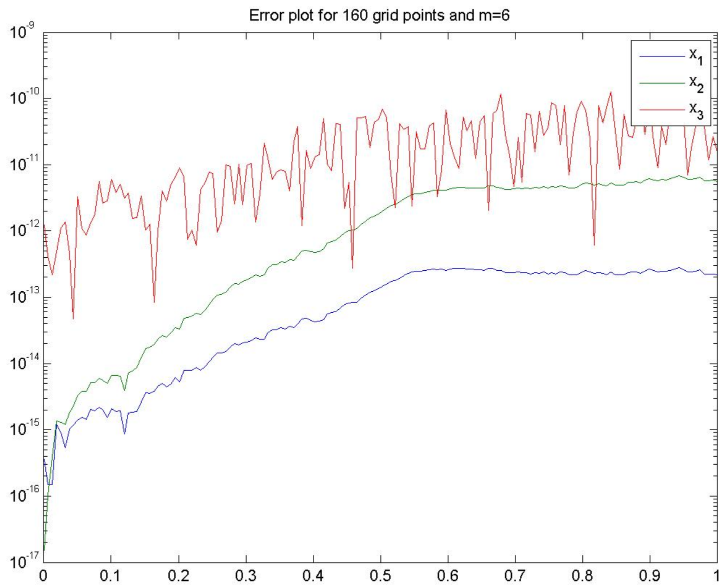


Table 6.10: Example 6.6 (Index-2): Results obtained with six collocation points for the standard over-determined collocation.

In order to obtain more insight into the method's behavior, further experiments in different settings were made. Modifications include:

- Changing the number of additional collocation points.
- Using different weights for collocation and initial and continuity conditions.
- Using different norms in which the residual is minimized.

Changing the number of additional collocation points. Instead of dividing every interval spanned by the collocation points, only one additional point is sufficient to reach over-determination. Therefore experiments were done with only one point placed in the center of a grid interval (for an even number of collocation points) or one point placed in the center of two inner-most collocation points.

It turns out that the number of additional points is not crucial; as soon as over-determination is reached, convergence is achieved. This can be verified by the following experimental results:

Example 6.5 (Index-3)	2 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	1.45e-02	0.0
40	7.01e-03	1.0
80	3.45e-03	1.0
160	1.71e-03	1.0
320	8.53e-04	1.0
640	4.26e-04	1.0

x_2 grid size	error	order
20	1.36e-04	0.0
40	3.23e-05	2.1
80	8.20e-06	2.0
160	2.06e-06	2.0
320	5.17e-07	2.0
640	1.29e-07	2.0

x_3 grid size	error	order
20	1.96e-04	0.0
40	4.62e-05	2.1
80	1.12e-05	2.0
160	2.77e-06	2.0
320	6.87e-07	2.0
640	1.71e-07	2.0

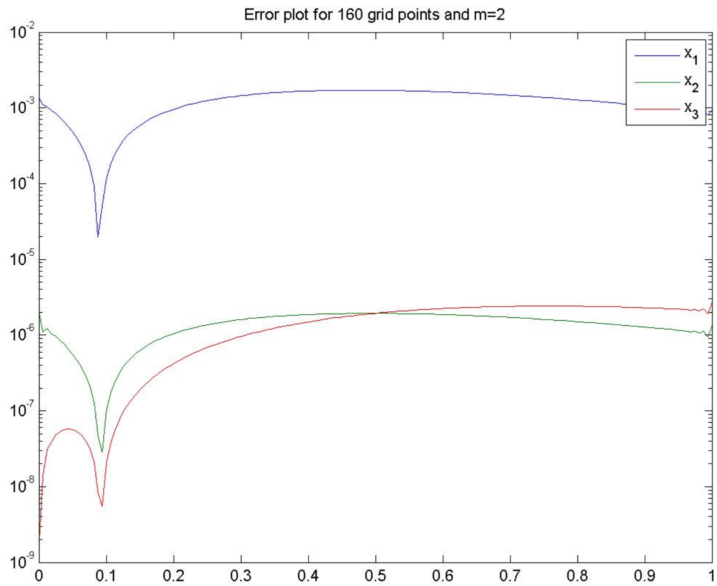


Table 6.11: Example 6.5 (Index-3): Results obtained with two collocation points and only one additional point.

Example 6.5 (Index-3)	3 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	4.60e-04	0.0
40	1.04e-04	2.1
80	2.46e-05	2.1
160	6.01e-06	2.0
320	1.48e-06	2.0
640	3.69e-07	2.0

x_2 grid size	error	order
20	3.94e-06	0.0
40	4.30e-07	3.2
80	5.03e-08	3.1
160	6.08e-09	3.0
320	7.47e-10	3.0
640	9.29e-11	3.0

x_3 grid size	error	order
20	7.86e-06	0.0
40	8.59e-07	3.2
80	1.00e-07	3.1
160	1.22e-08	3.0
320	1.49e-09	3.0
640	1.86e-10	3.0

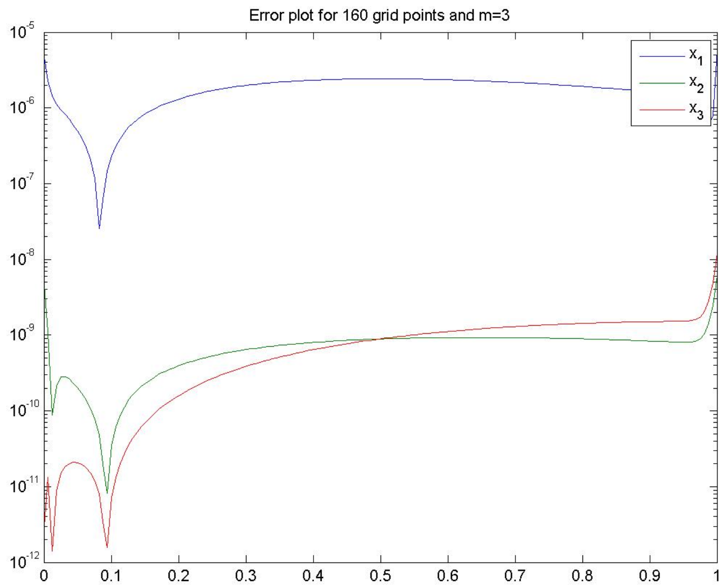


Table 6.12: Example 6.5 (Index-3): Results obtained with three collocation points and only one additional point.

Example 6.5 (Index-3)	4 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	4.04e-06	0.0
40	4.62e-07	3.1
80	5.54e-08	3.1
160	7.75e-09	2.8
320	1.48e-08	-0.9
640	5.87e-08	-2.0

x_2 grid size	error	order
20	1.89e-08	0.0
40	1.06e-09	4.2
80	6.30e-11	4.1
160	3.83e-12	4.0
320	5.70e-12	-0.6
640	1.12e-11	-1.0

x_3 grid size	error	order
20	3.77e-08	0.0
40	2.12e-09	4.2
80	1.26e-10	4.1
160	7.67e-12	4.0
320	3.50e-12	1.1
640	1.02e-11	-1.5

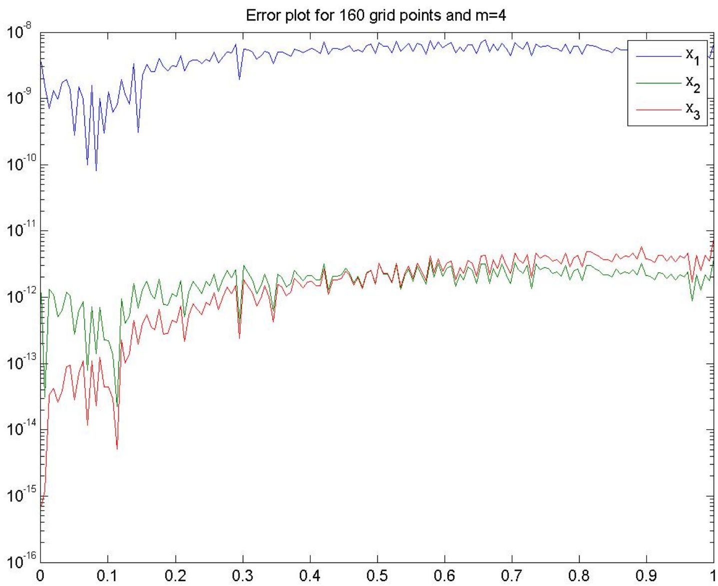


Table 6.13: Example 6.5 (Index-3): Results obtained with four collocation points and only one additional point.

Example 6.5 (Index-3)	6 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	2.71e-10	0.0
40	6.70e-10	-1.3
80	9.72e-09	-3.9
160	1.93e-08	-1.0
320	7.42e-08	-1.9
640	3.38e-07	-2.2

x_2 grid size	error	order
20	6.63e-13	0.0
40	8.36e-13	-0.3
80	7.41e-12	-3.1
160	4.37e-12	0.8
320	9.74e-12	-1.2
640	2.64e-11	-1.4

x_3 grid size	error	order
20	1.17e-12	0.0
40	9.44e-13	0.3
80	3.04e-12	-1.7
160	7.02e-12	-1.2
320	9.07e-12	-0.4
640	2.06e-11	-1.2

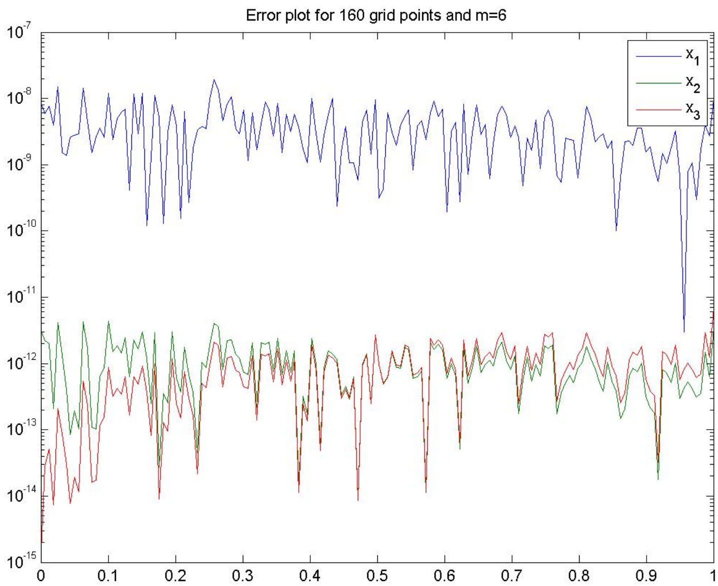


Table 6.14: Example 6.5 (Index-3): Results obtained with six collocation points and only one additional point.

Example 6.6 (Index-2)	2 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	4.51e-03	0.0
40	1.72e-03	1.4
80	5.35e-04	1.7
160	1.44e-04	1.9
320	3.66e-05	2.0
640	9.18e-06	2.0

x_2 grid size	error	order
20	1.17e-01	0.0
40	4.47e-02	1.4
80	1.39e-02	1.7
160	3.74e-03	1.9
320	9.52e-04	2.0
640	2.39e-04	2.0

x_3 grid size	error	order
20	1.19e-01	0.0
40	4.54e-02	1.4
80	1.42e-02	1.7
160	3.86e-03	1.9
320	1.29e-03	1.6
640	5.98e-04	1.1

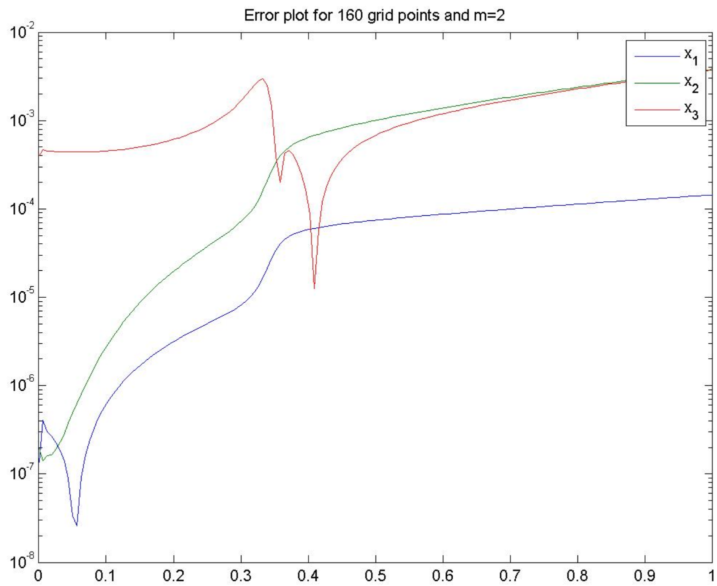


Table 6.15: Example 6.6 (Index-2): Results obtained with two collocation points and only one additional point.

Example 6.6 (Index-2)	3 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	8.79e-06	0.0
40	1.50e-06	2.5
80	3.02e-07	2.3
160	6.75e-08	2.2
320	1.60e-08	2.1
640	3.89e-09	2.0

x_2 grid size	error	order
20	2.29e-04	0.0
40	3.91e-05	2.5
80	7.85e-06	2.3
160	1.76e-06	2.2
320	4.16e-07	2.1
640	1.01e-07	2.0

x_3 grid size	error	order
20	2.36e-04	0.0
40	4.64e-05	2.3
80	1.02e-05	2.2
160	2.31e-06	2.1
320	5.37e-07	2.1
640	1.29e-07	2.1

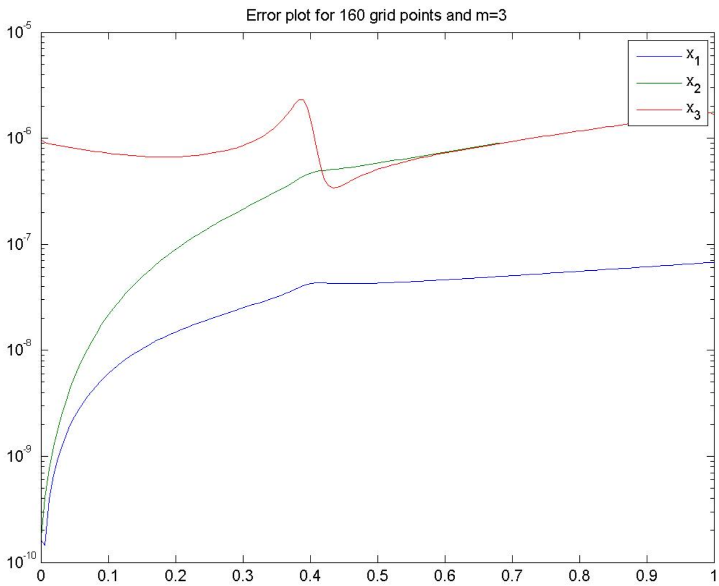


Table 6.16: Example 6.6 (Index-2): Results obtained with three collocation points and only one additional point.

Example 6.6 (Index-2)	4 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	1.60e-08	0.0
40	1.46e-09	3.5
80	1.09e-10	3.7
160	7.15e-12	3.9
320	6.29e-13	3.5
640	6.43e-13	-0.0

x_2 grid size	error	order
20	4.08e-07	0.0
40	3.79e-08	3.4
80	2.83e-09	3.7
160	1.86e-10	3.9
320	1.62e-11	3.5
640	1.67e-11	-0.0

x_3 grid size	error	order
20	5.19e-07	0.0
40	7.08e-08	2.9
80	7.99e-09	3.1
160	1.02e-09	3.0
320	1.75e-10	2.5
640	6.17e-10	-1.8

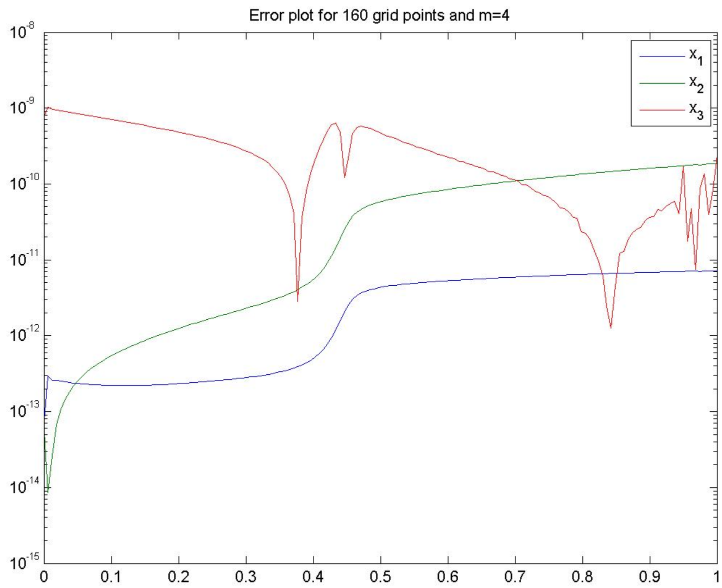


Table 6.17: Example 6.6 (Index-2): Results obtained with four collocation points and only one additional point.

Example 6.6 (Index-2)	6 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.54e-13	0.0
40	3.66e-13	-0.0
80	3.31e-13	0.1
160	1.46e-13	1.2
320	4.15e-13	-1.5
640	5.00e-13	-0.3

x_2 grid size	error	order
20	8.22e-12	0.0
40	8.99e-12	-0.1
80	8.60e-12	0.1
160	3.48e-12	1.3
320	1.08e-11	-1.6
640	1.28e-11	-0.2

x_3 grid size	error	order
20	5.42e-11	0.0
40	9.67e-11	-0.8
80	1.07e-10	-0.2
160	4.34e-10	-2.0
320	6.66e-10	-0.6
640	2.26e-09	-1.8

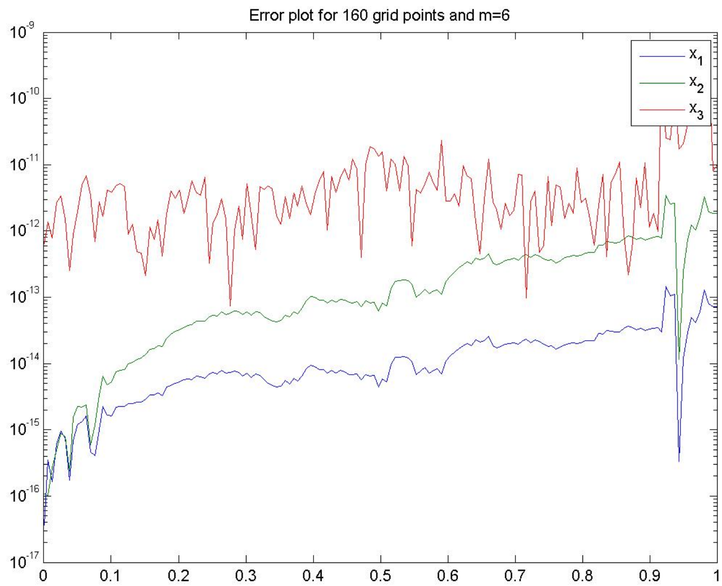


Table 6.18: Example 6.6 (Index-2): Results obtained with six collocation points and only one additional point.

Using different weights for collocation vs. initial and continuity conditions. Analyzing the solutions obtained by the standard over-determined collocation shows that the resulting approximations are not continuous. This happens because the stated conditions need not be satisfied exactly, since over-determination allows a residual greater than 0. In order to cope with this fact, one possibility is to weight the respective conditions by multiplying with a factor greater than one.

While the norm of the approximation error is not improved by this technique, it can be verified that the obtained solution is smoother. We observe that for higher η and a higher number of collocation points difficulties occur. One reason for this is the higher influence of round-off errors.

Example 6.5 (Index-3)	2 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	1.46e-02	0.0
40	7.05e-03	1.1
80	3.46e-03	1.0
160	1.72e-03	1.0
320	8.54e-04	1.0
640	4.26e-04	1.0

x_2 grid size	error	order
20	1.26e-04	0.0
40	2.99e-05	2.1
80	7.28e-06	2.0
160	1.83e-06	2.0
320	4.60e-07	2.0
640	1.15e-07	2.0

x_3 grid size	error	order
20	1.82e-04	0.0
40	4.29e-05	2.1
80	1.04e-05	2.0
160	2.57e-06	2.0
320	6.38e-07	2.0
640	1.59e-07	2.0

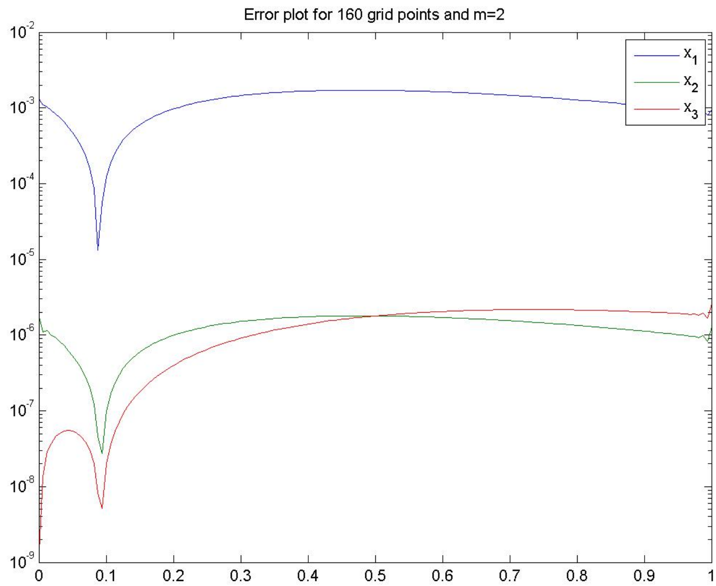


Table 6.19: Example 6.5 (Index-3): Results obtained with two collocation points and continuity conditions weighted with $w = 10$.

Example 6.5 (Index-3)	3 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	3.61e-04	0.0
40	8.30e-05	2.1
80	1.99e-05	2.1
160	4.87e-06	2.0
320	1.21e-06	2.0
640	3.01e-07	2.0

x_2 grid size	error	order
20	2.81e-06	0.0
40	3.13e-07	3.2
80	3.70e-08	3.1
160	4.50e-09	3.0
320	5.55e-10	3.0
640	6.89e-11	3.0

x_3 grid size	error	order
20	5.61e-06	0.0
40	6.26e-07	3.2
80	7.40e-08	3.1
160	9.00e-09	3.0
320	1.11e-09	3.0
640	1.38e-10	3.0

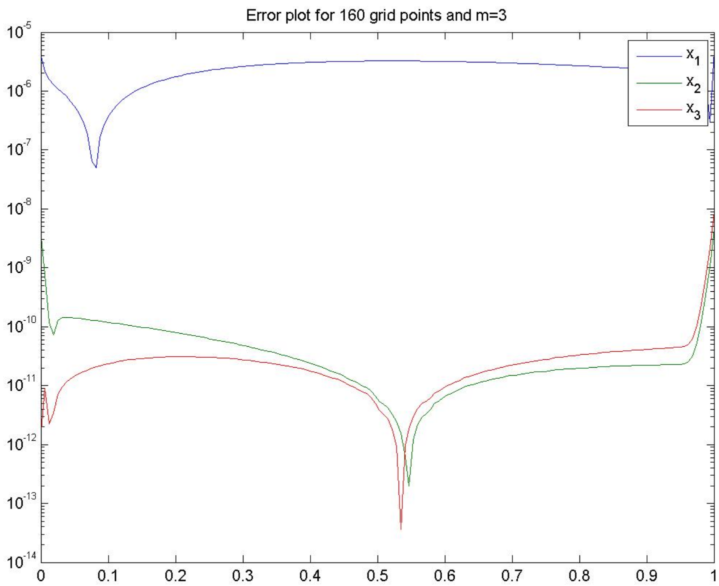


Table 6.20: Example 6.5 (Index-3): Results obtained with three collocation points and continuity conditions weighted with $w = 10$.

Example 6.5 (Index-3)	4 uniform collocation points	$\eta = -2$
x_1 grid size	error	order
20	3.37e-06	0.0
40	3.89e-07	3.1
80	4.68e-08	3.1
160	7.40e-09	2.7
320	1.39e-08	-0.9
640	6.46e-08	-2.2
x_2 grid size	error	order
20	1.29e-08	0.0
40	7.30e-10	4.1
80	4.35e-11	4.1
160	3.28e-12	3.7
320	4.76e-12	-0.5
640	8.77e-12	-0.9
x_3 grid size	error	order
20	2.57e-08	0.0
40	1.46e-09	4.1
80	8.69e-11	4.1
160	6.57e-12	3.7
320	3.73e-12	0.8
640	7.07e-12	-0.9

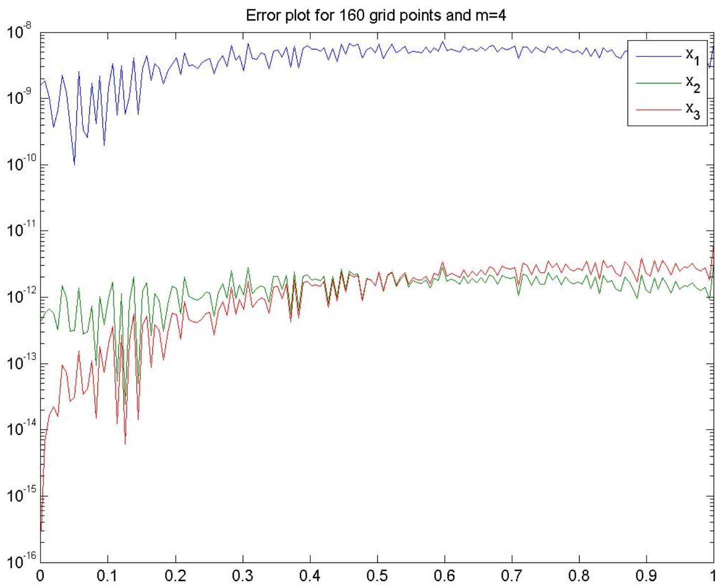


Table 6.21: Example 6.5 (Index-3): Results obtained with four collocation points and continuity conditions weighted with $w = 10$.

Example 6.5 (Index-3)	6 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	1.84e-10	0.0
40	7.87e-10	-2.1
80	3.35e-09	-2.1
160	1.86e-08	-2.5
320	5.68e-08	-1.6
640	3.17e-07	-2.5

x_2 grid size	error	order
20	3.84e-13	0.0
40	8.34e-13	-1.1
80	1.42e-12	-0.8
160	4.35e-12	-1.6
320	6.89e-12	-0.7
640	1.85e-11	-1.4

x_3 grid size	error	order
20	6.84e-13	0.0
40	4.73e-13	0.5
80	1.59e-12	-1.8
160	3.60e-12	-1.2
320	7.11e-12	-1.0
640	1.59e-11	-1.2

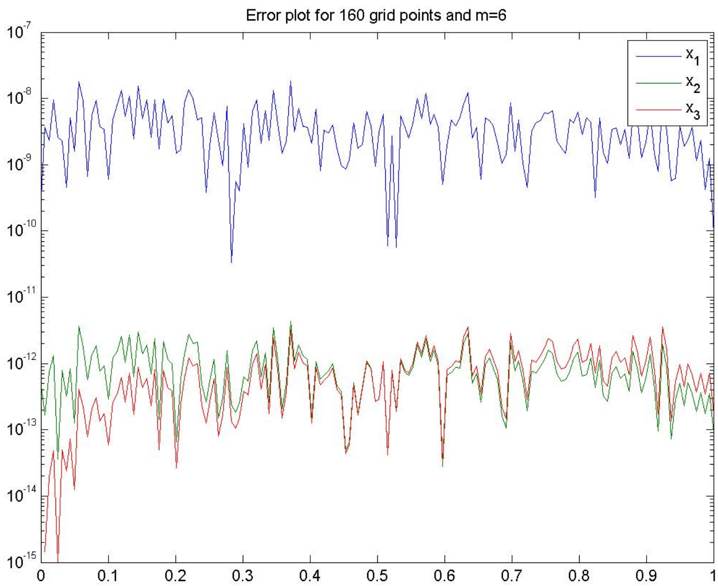


Table 6.22: Example 6.5 (Index-3): Results obtained with six collocation points and continuity conditions weighted with $w = 10$.

Example 6.6 (Index-2)	2 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.22e-01	0.0
40	3.22e-01	0.0
80	3.22e-01	-0.0
160	3.22e-01	0.0
320	3.22e-01	0.0
640	3.22e-01	-0.0

x_2 grid size	error	order
20	1.77e-01	0.0
40	1.77e-01	-0.0
80	1.77e-01	0.0
160	1.77e-01	-0.0
320	1.77e-01	-0.0
640	1.77e-01	0.0

x_3 grid size	error	order
20	1.00e+00	0.0
40	1.00e+00	0.0
80	1.00e+00	0.0
160	1.00e+00	0.0
320	1.00e+00	0.0
640	1.00e+00	0.0

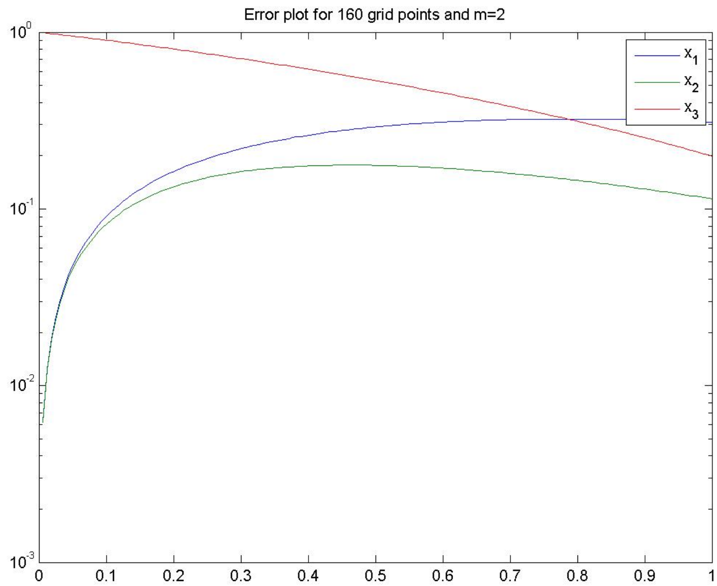


Table 6.23: Example 6.6 (Index-2): Results obtained with two collocation points and continuity conditions weighted with $w = 10$.

Example 6.6 (Index-2)	3 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.22e-01	0.0
40	3.22e-01	0.0
80	3.22e-01	-0.0
160	3.22e-01	0.0
320	3.22e-01	0.0
640	3.22e-01	-0.0

x_2 grid size	error	order
20	1.77e-01	0.0
40	1.77e-01	-0.0
80	1.77e-01	0.0
160	1.77e-01	-0.0
320	1.77e-01	-0.0
640	1.77e-01	0.0

x_3 grid size	error	order
20	1.00e+00	0.0
40	1.00e+00	0.0
80	1.00e+00	0.0
160	1.00e+00	0.0
320	1.00e+00	0.0
640	1.00e+00	0.0

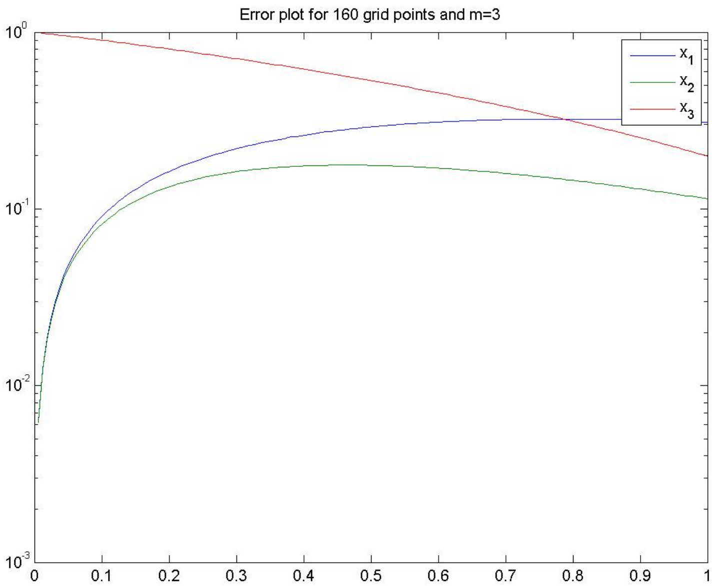


Table 6.24: Example 6.6 (Index-2): Results obtained with three collocation points and continuity conditions weighted with $w = 10$.

Example 6.6 (Index-2)	4 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.22e-01	0.0
40	3.22e-01	0.0
80	3.22e-01	-0.0
160	3.22e-01	0.0
320	3.22e-01	0.0
640	3.22e-01	-0.0

x_2 grid size	error	order
20	1.77e-01	0.0
40	1.77e-01	-0.0
80	1.77e-01	0.0
160	1.77e-01	-0.0
320	1.77e-01	-0.0
640	1.77e-01	0.0

x_3 grid size	error	order
20	1.00e+00	0.0
40	1.00e+00	0.0
80	1.00e+00	0.0
160	1.00e+00	0.0
320	1.00e+00	0.0
640	1.00e+00	0.0

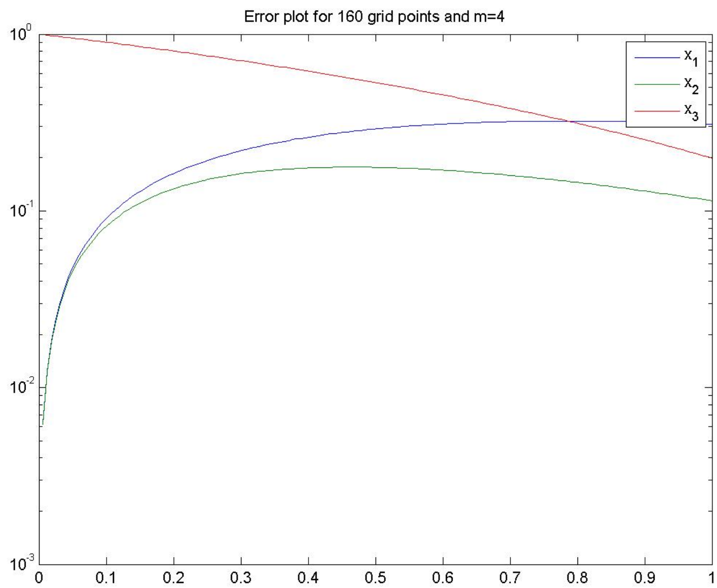


Table 6.25: Example 6.6 (Index-2): Results obtained with four collocation points and continuity conditions weighted with $w = 10$.

Example 6.6 (Index-2)	6 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	3.22e-01	0.0
40	3.22e-01	0.0
80	3.22e-01	-0.0
160	3.22e-01	0.0
320	3.22e-01	0.0
640	3.22e-01	-0.0

x_2 grid size	error	order
20	1.77e-01	0.0
40	1.77e-01	-0.0
80	1.77e-01	0.0
160	1.77e-01	-0.0
320	1.77e-01	-0.0
640	1.77e-01	0.0

x_3 grid size	error	order
20	1.00e+00	0.0
40	1.00e+00	0.0
80	1.00e+00	0.0
160	1.00e+00	0.0
320	1.00e+00	0.0
640	1.00e+00	0.0

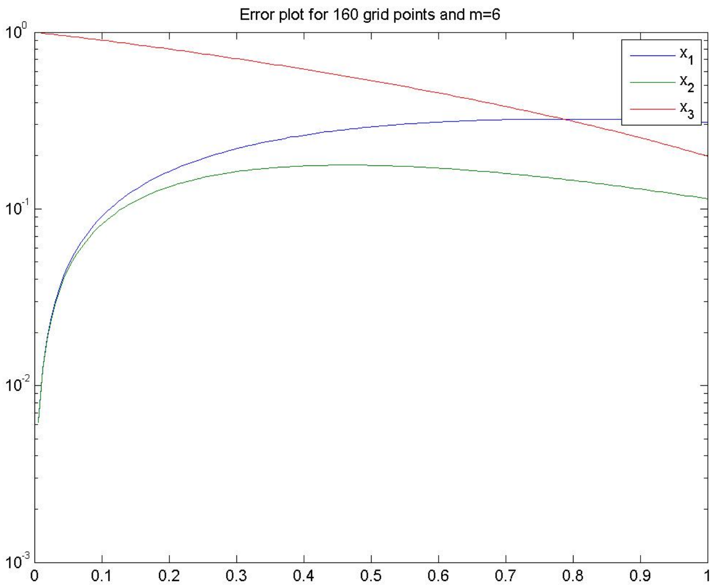


Table 6.26: Example 6.6 (Index-2): Results obtained with six collocation points and continuouity conditions weighted with $w = 10$.

Using different norms in which the residual is minimized. In the previous variants the residuum was minimized in the 2-norm. However, it is more natural to minimize in a modification of the L^2-norm. Detailed explanations on this topic can be found in [16]. In the following we present some experimental results.

Example 6.5 (Index-3)	2 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	1.46e-02	0.0
40	7.04e-03	1.0
80	3.46e-03	1.0
160	1.71e-03	1.0
320	8.54e-04	1.0
640	4.26e-04	1.0

x_2 grid size	error	order
20	1.26e-04	0.0
40	2.98e-05	2.1
80	7.27e-06	2.0
160	1.79e-06	2.0
320	4.46e-07	2.0
640	1.11e-07	2.0

x_3 grid size	error	order
20	1.70e-04	0.0
40	4.01e-05	2.1
80	9.75e-06	2.0
160	2.40e-06	2.0
320	5.97e-07	2.0
640	1.49e-07	2.0

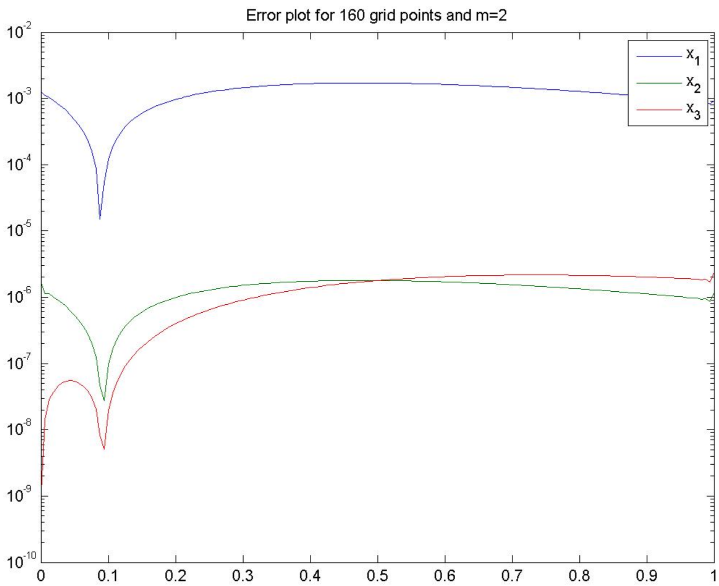


Table 6.27: Example 6.5 (Index-3): Results obtained with two collocation points and least square in L^2-sense.

Example 6.5 (Index-3)	3 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	3.26e-04	0.0
40	7.52e-05	2.1
80	1.81e-05	2.1
160	4.43e-06	2.0
320	1.10e-06	2.0
640	6.16e-07	0.8

x_2 grid size	error	order
20	2.42e-06	0.0
40	2.72e-07	3.2
80	3.22e-08	3.1
160	3.91e-09	3.0
320	4.84e-10	3.0
640	1.51e-10	1.7

x_3 grid size	error	order
20	4.84e-06	0.0
40	5.43e-07	3.2
80	6.43e-08	3.1
160	7.82e-09	3.0
320	9.68e-10	3.0
640	1.50e-10	2.7

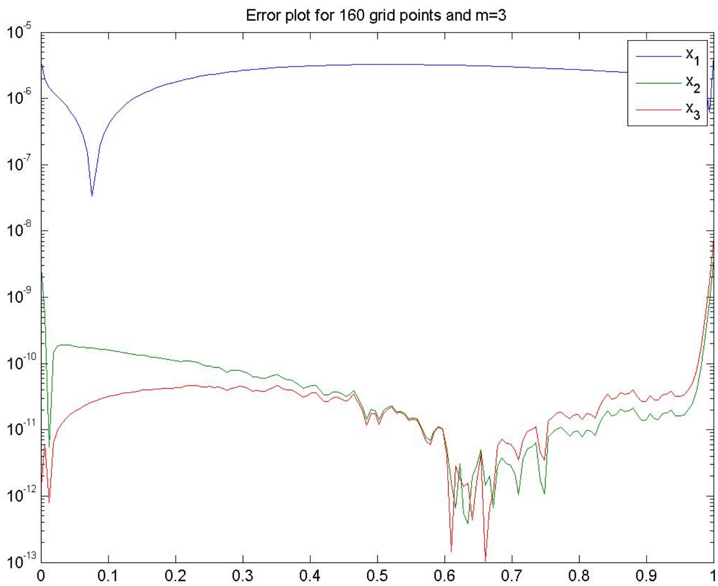


Table 6.28: Example 6.5 (Index-3): Results obtained with three collocation points and least square in L^2-sense.

Example 6.5 (Index-3)	4 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	3.46e-06	0.0
40	3.99e-07	3.1
80	5.12e-08	3.0
160	5.38e-08	-0.1
320	4.19e-07	-3.0
640	1.80e+00	-22.0

x_2 grid size	error	order
20	1.18e-08	0.0
40	6.69e-10	4.1
80	4.03e-11	4.1
160	2.76e-11	0.5
320	1.15e-10	-2.1
640	3.08e-04	-21.4

x_3 grid size	error	order
20	2.35e-08	0.0
40	1.34e-09	4.1
80	8.06e-11	4.1
160	2.85e-11	1.5
320	6.97e-11	-1.3
640	1.08e-04	-20.6

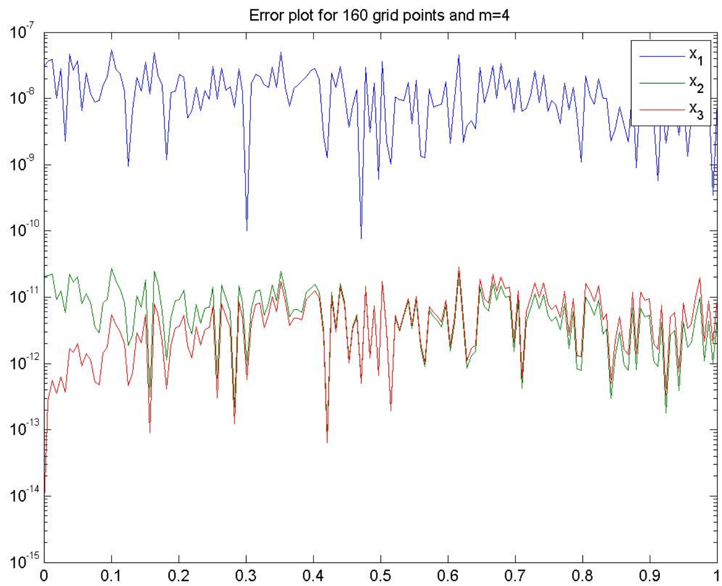


Table 6.29: Example 6.5 (Index-3): Results obtained with four collocation points and least square in L^2-sense.

Example 6.5 (Index-3)	6 uniform collocation points	$\eta = -2$

x_1 grid size	error	order
20	3.51e-10	0.0
40	4.08e-09	-3.5
80	1.58e-08	-2.0
160	1.59e-07	-3.3
320	1.65e+00	-23.3
640	8.78e+00	-2.4

x_2 grid size	error	order
20	9.96e-13	0.0
40	4.34e-12	-2.1
80	9.02e-12	-1.1
160	3.80e-11	-2.1
320	2.68e-04	-22.7
640	9.57e-04	-1.8

x_3 grid size	error	order
20	1.08e-12	0.0
40	4.12e-12	-1.9
80	8.81e-12	-1.1
160	3.52e-11	-2.0
320	9.44e-05	-21.4
640	2.66e-04	-1.5

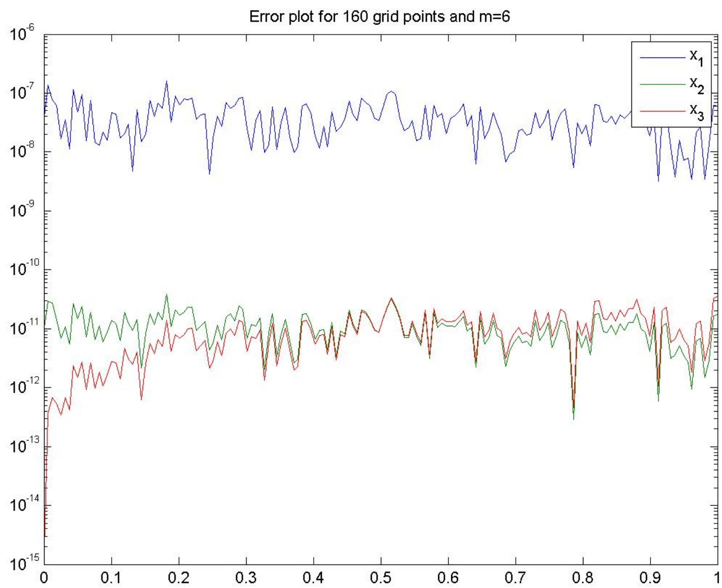


Table 6.30: Example 6.5 (Index-3): Results obtained with six collocation points and least square in L^2-sense.

Example 6.6 (Index-2)	2 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	6.84e-04	0.0
40	5.04e-05	3.8
80	8.43e-06	2.6
160	1.79e-06	2.2
320	4.28e-07	2.1
640	1.06e-07	2.0

x_2 grid size	error	order
20	1.77e-02	0.0
40	1.30e-03	3.8
80	1.30e-04	3.3
160	1.77e-05	2.9
320	3.50e-06	2.3
640	8.62e-07	2.0

x_3 grid size	error	order
20	2.01e-02	0.0
40	6.19e-03	1.7
80	3.23e-03	0.9
160	1.63e-03	1.0
320	8.16e-04	1.0
640	4.07e-04	1.0

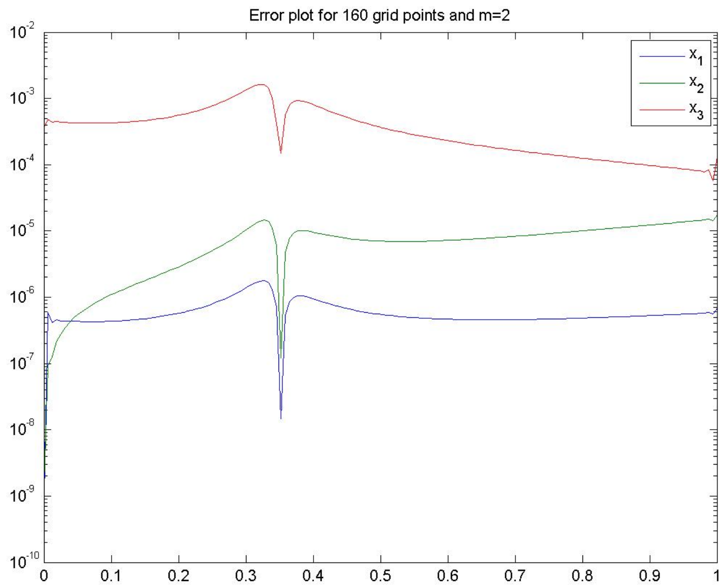


Table 6.31: Example 6.6 (Index-2): Results obtained with two collocation points and least square in L^2-sense.

Example 6.6 (Index-2)	3 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	2.49e-06	0.0
40	1.47e-07	4.1
80	1.33e-08	3.5
160	9.82e-10	3.8
320	5.84e-11	4.1
640	2.73e-12	4.4

x_2 grid size	error	order
20	6.39e-05	0.0
40	3.79e-06	4.1
80	2.53e-07	3.9
160	1.61e-08	4.0
320	9.85e-10	4.0
640	5.08e-11	4.3

x_3 grid size	error	order
20	1.64e-04	0.0
40	4.13e-05	2.0
80	9.61e-06	2.1
160	1.74e-06	2.5
320	3.14e-07	2.5
640	6.35e-08	2.3

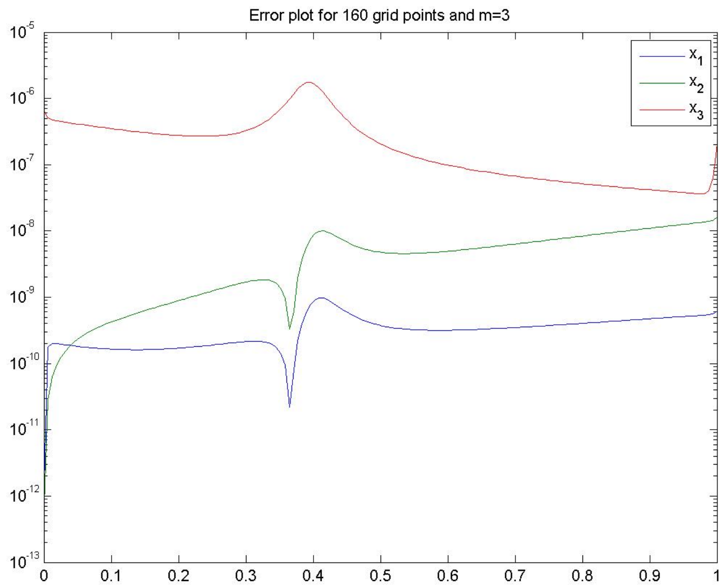


Table 6.32: Example 6.6 (Index-2): Results obtained with three collocation points and least square in L^2-sense.

Example 6.6 (Index-2)	4 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	2.13e-09	0.0
40	1.27e-10	4.1
80	6.32e-12	4.3
160	3.51e-13	4.2
320	2.08e-13	0.8
640	6.57e-13	-1.7

x_2 grid size	error	order
20	5.09e-08	0.0
40	2.30e-09	4.5
80	1.00e-10	4.5
160	3.06e-12	5.0
320	4.93e-12	-0.7
640	1.31e-11	-1.4

x_3 grid size	error	order
20	4.67e-07	0.0
40	6.91e-08	2.8
80	7.73e-09	3.2
160	9.79e-10	3.0
320	3.48e-10	1.5
640	8.25e-10	-1.2

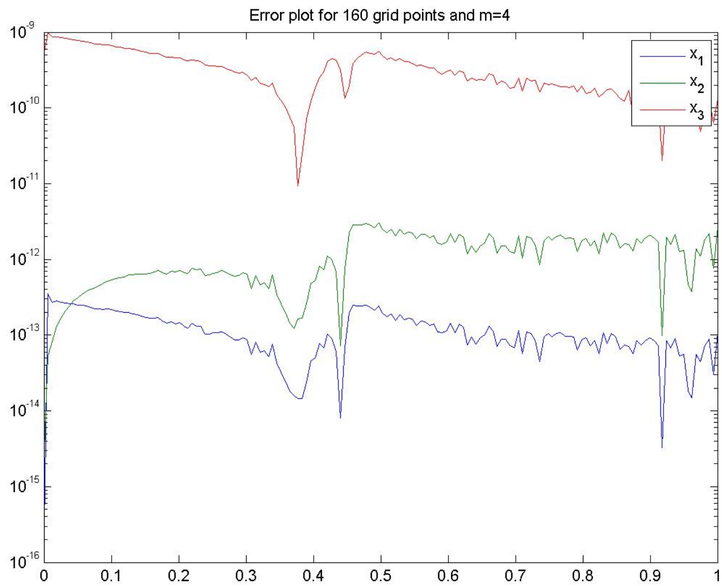


Table 6.33: Example 6.6 (Index-2): Results obtained with four collocation points and least square in L^2-sense.

Example 6.6 (Index-2)	6 uniform collocation points	$\eta = -25$, $\lambda = -1$

x_1 grid size	error	order
20	9.70e-14	0.0
40	1.29e-13	-0.4
80	1.87e-13	-0.5
160	1.40e-13	0.4
320	2.83e-13	-1.0
640	6.58e-13	-1.2

x_2 grid size	error	order
20	2.34e-12	0.0
40	3.00e-12	-0.4
80	4.81e-12	-0.7
160	3.31e-12	0.5
320	5.67e-12	-0.8
640	1.28e-11	-1.2

x_3 grid size	error	order
20	3.89e-11	0.0
40	1.10e-10	-1.5
80	2.64e-10	-1.3
160	5.83e-10	-1.1
320	9.03e-10	-0.6
640	3.14e-09	-1.8

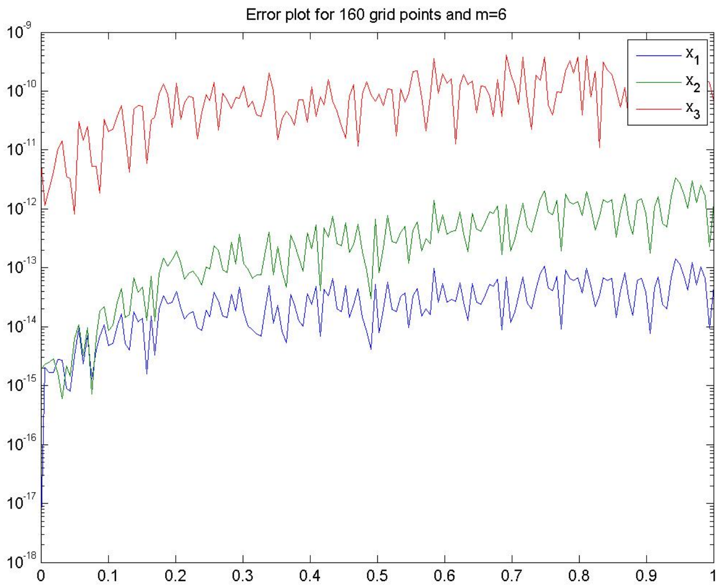


Table 6.34: Example 6.6 (Index-2): Results obtained with six collocation points and least square in L^2-sense.

Appendix A

Auxiliary results

A.1 Derivatives of a transformed function

Consider the functions $f(t), t \in \mathcal{I} \subset \mathbb{R}$ and $\tilde{f}(\tau), \tau \in \tilde{\mathcal{I}} \subset \mathbb{R}$ as well as a sufficiently smooth one-to-one mapping $\xi : \mathcal{I} \to \tilde{\mathcal{I}}$, with $f(t) = \tilde{f}(\xi(t))$.

Lemma A.1
It holds for $n \in \mathbb{N}$:

$$f^{(n)}(t) = \sum_{j=1}^{\nu_n} c_j^{(n)} \cdot \left(\prod_{k=1}^{n+1} \left(\xi^{(k-1)}(t) \right)^{A_{j,k}^{(n)}} \right) \cdot \tilde{f}^{\left(b_j^{(n)}\right)}(\xi(t)),$$

where

$$\boldsymbol{A^{(n)}} \in \mathbb{N}^{\nu_n \times (n+1)}, \boldsymbol{b^{(n)}} \in \mathbb{N}^{\nu_n}, \boldsymbol{c^{(n)}} \in \mathbb{N}^{\nu_n}, \nu_n \in \mathbb{N}$$

and

$$b_j^{(n)} \leq n, \ \forall j = 1, \dots, \nu_n.$$

$\boldsymbol{A^{(n)}}, \boldsymbol{b^{(n)}}, \boldsymbol{c^{(n)}}$ can be computed recursively from $\boldsymbol{A^{(n-1)}}, \boldsymbol{b^{(n-1)}}, \boldsymbol{c^{(n-1)}}$.

Proof. Proof by induction: For $n = 0$ the statement obviously holds true with $\boldsymbol{A^{(0)}} = \boldsymbol{b^{(0)}} = (0), \boldsymbol{c^{(0)}} = (1)$. For demonstration purposes we explain the case $n = 1$, too: $\boldsymbol{A^{(1)}} = \begin{pmatrix} 0 & 1 \end{pmatrix}, \boldsymbol{b^{(1)}} = \boldsymbol{c^{(1)}} = (1)$, which corresponds to the common chain rule: $f'(t) = \xi'(t)\tilde{f}'(\xi(t))$

For the inductive step, we consider the statement as true and get

$$f^{(n+1)}(t) = \left(f^{(n)}(t) \right)' = \sum_{j=1}^{\nu_n} c_j^{(n)} \cdot \left(\left(\prod_{k=1}^{n+1} \left(\xi^{(k-1)}(t) \right)^{A_{j,k}^{(n)}} \right) \cdot \tilde{f}^{\left(b_j^{(n)}\right)}(\xi(t)) \right)'.$$

By using the product rule and chain rule, we derive

$$\left(\left(\prod_{k=1}^{n+1}\left(\xi^{(k-1)}(t)\right)^{A^{(n)}_{j,k}}\right)\cdot \tilde{f}^{\left(b^{(n)}_j\right)}(\xi(t))\right)' =$$

$$= \left(\sum_{\ell=1}^{n+1}\left(\xi^{(\ell-1)}(t)\right)^{A^{(n)}_{j,\ell}-1}\cdot A^{(n)}_{j,l}\cdot \xi^{(\ell)}(t)\cdot \prod_{k\neq\ell}\left(\xi^{(k-1)}(t)\right)^{A^{(n)}_{j,k}}\right)\cdot \tilde{f}^{\left(b^{(n)}_j\right)}(\xi(t))+$$
$$+\left(\prod_{k=1}^{n+1}\left(\xi^{(k-1)}(t)\right)^{A^{(n)}_{j,k}}\right)\cdot \xi'(t)\cdot \tilde{f}^{\left(b^{(n)}_j+1\right)}(\xi(t)).$$

This shows, that with

$$\nu_{n+1} = \nu_n(n+2),$$

$$\boldsymbol{A^{(n+1)}} = \left(\begin{array}{c|c} \multicolumn{2}{c}{\bar{\boldsymbol{A}}^{(n+1)}} \\ \hline \boldsymbol{A^{(n)}} + \begin{pmatrix} 0 & 1 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \\ 0 & 1 & 0 & \dots \end{pmatrix} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \end{array}\right) \in \mathbb{N}^{\nu_{n+1}\times(n+2)}, \quad \bar{\boldsymbol{A}}^{(n+1)} \in \mathbb{N}^{\nu_n(n+1)\times(n+2)},$$

where

$$\bar{A}^{(n+1)}_{\bar{j},k} = \begin{cases} A^{(n)}_{j,k}-1 & \lceil\frac{\bar{j}}{\nu_n}\rceil = k \wedge A^{(n)}_{j,k}\neq 0 \\ A^{(n)}_{j,k}+1 & \lceil\frac{\bar{j}}{\nu_n}\rceil = k-1 \wedge A^{(n)}_{j,k}\neq 0 \\ A^{(n)}_{j,k} & \text{otherwise} \end{cases}, \quad \bar{j}\equiv j \mod \nu_n \text{ for } k\leq n+1,$$

$$\bar{A}^{(n+1)}_{\bar{j},n+2} = \begin{cases} 1 & \lceil\frac{\bar{j}}{\nu_n}\rceil = n+1 \\ 0 & \text{otherwise} \end{cases},$$

$$\boldsymbol{b}^{(n+1)} = \begin{pmatrix} \boldsymbol{b}^{(n)} \\ \vdots \\ \boldsymbol{b}^{(n)} \\ \boldsymbol{b}^{(n)} + \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \end{pmatrix} \text{ and } \boldsymbol{c}^{(n+1)} = \begin{pmatrix} c_1^{(n)} \cdot A_{1,1}^{(n)} \\ c_2^{(n)} \cdot A_{2,1}^{(n)} \\ \vdots \\ c_{\nu_n}^{(n)} \cdot A_{\nu_n,1}^{(n)} \\ c_1^{(n)} \cdot A_{1,2}^{(n)} \\ \vdots \\ c_{\nu_n}^{(n)} \cdot A_{\nu_n,2}^{(n)} \\ \vdots \\ c_1^{(n)} \cdot A_{1,n+1}^{(n)} \\ \vdots \\ c_{\nu_n}^{(n)} \cdot A_{\nu_n,n+1}^{(n)} \\ \boldsymbol{c}^{(n)} \end{pmatrix},$$

the statement holds true for $f^{(n+1)}$. The values of $\boldsymbol{b}^{(n+1)}$ can only be greater by 1 than the values of $\boldsymbol{b}^{(n)}$, which shows $b_j^{(n+1)} \leq n+1$. □

Remark A.2
There also exists a closed form of $f^{(n)}$, the so-called "Faà di Bruno's formula", where, however, a sum over all partitions of $\{1, \dots, n\}$ of Bell polynomials are required. Since we will need *all* derivatives up to n anyway, the recursive computation is more convenient here.

A possible implementation of this recursion is the following, which is also the implementation in `bvpsuite2.0`:

```
function [ der ] = higherchainrule( n )
%CHAINRULE computes the chainrule up to the n-th derivative
der = cell(n+1,1);

der{1}.A=0; %is the 0-th derivate, i.e. f(tau(t))
der{1}.b=0;
der{1}.c=1;

for j=2:n+1 %compute the (j-1)-th derivative
    [M,N]=size(der{j-1}.A);
    [K,L]=find(der{j-1}.A);
    A = zeros(length(K),j);
    b = zeros(length(K),1);
    c = zeros(length(K),1);
```

```
    %first the tau-derivatives from the chain rule
    for ii = 1:length(K)
        A(ii,:)=[der{j-1}.A(K(ii),:),0]; %copy the row to the next A-matrix
        A(ii,L(ii)) = A(ii,L(ii)) - 1;
        A(ii,L(ii)+1) = A(ii,L(ii)+1) + 1;
        b(ii)=der{j-1}.b(K(ii)); %the tau-derivatives corresponds to the same f-derivative
        c(ii)=der{j-1}.c(K(ii)) * der{j-1}.A(K(ii),L(ii)); %the new coefficient is the old coefficient times the power of the tau-derivative
    end
    %now the f-derivatives (note: the inner derivative gives one tau')
    c = [c;der{j-1}.c];
    b = [b;der{j-1}.b+1];
    A = [A;[der{j-1}.A,zeros(M,1)] + [zeros(M,1),ones(M,1),zeros(M,N-1)]];

    %Relaxation
    [U,IA,IU]=unique([A,b],'rows');
    der{j}.c = zeros(length(IA),1);
    for k = 1:length(IU)
       der{j}.c(IU(k))=der{j}.c(IU(k))+c(k);
    end
    der{j}.A=U(:,1:end-1);
    der{j}.b=U(:,end);
end
end
```

In this code snippet, at the end of each recursion step, a "relaxation" is done: this means that equal rows in $(\boldsymbol{A}^{(n)}, \boldsymbol{b}^{(n)})$, which would correspond to the same summand, can be reduced to one row, where the corresponding $\boldsymbol{c}^{(n)}$ coefficients have to be summed up.

Appendix B

Over-determined collocation MATLAB-code

```
function [z,err,resid,maxerror,tau,normerror,L2normerror] = solveDAEs(dae, par, tau, m, colltype, overdet, showPlot, bvpfilestring, teval, addTestCombo)
%solveDAEs solves several examples of DAEs
%Stefan Wurm 2014-

%% INPUT:
%
% # dae ... which example to solve
% # par ... specifies parameters used by the example
% # tau ... mesh on which the problem is solved (on reference interval
% [0,1])
% # m ... number of collocation points
% # colltype ... 'u' for uniform collocation points, 'g' for gauss collocation points
% # overdet ... should the system be overdetermined

%% OUTPUT:
%
% # z ... solution evaluated at mesh points
% # resid ... ||A*x-b||^2
% # maxerror ... max error for each component
% # normerror ... ||err||^2_H1D (component-wise)
% # L2normerror ... ||err||^2_L2 (component-wise)

weight=10;

%% Define Example DAE1 (Index 3-Problem) as IVP
if(dae==1)
eta=par(1);

Fkoeff=@(t)[[1,0,0;0,(eta+1),0;0,t*eta,1],[0,1,0;0,t*eta,1;0,0,0]];
f=@(t) [exp(-2*t)*(-2*sin(t)+cos(t))+exp(-t)*sin(t);
        exp(-2*t) * (-2*t*eta*sin(t)+t*eta*cos(t)+(eta+1)*sin(t)) - exp(-t)*(cos(t)+sin(t));
        exp(-2*t)*t*eta*sin(t)+exp(-t)*cos(t)
];

sol=@(t)[exp(-t).*sin(t); exp(-2*t).*sin(t); exp(-t).*cos(t)];
solD1=@(t)[-exp(-t).*sin(t)+exp(-t).*cos(t); -2*exp(-2*t).*sin(t)+exp(-2*t).*cos(t); -exp(-t).*cos(t)-exp(-t).*sin(t)];

BVa = []; %[0,1,0;0,0,1];
ga = []; %[0;1];

if(overdet==0)
    BVa = [0,1,0;0,0,1];
     ga = [0;1];
end

BVb = [];
gb = [];

%BVb = [0,eta,1];
%gb = ((eta)*exp(-2)*sin(1)+exp(-1)*cos(1));

N=3;
l=[0,1,1];
```

```
a=0;
b=1;
end

%% Define Example DAE2 (Index 2-Problem)
if(dae==2)
eta=par(1);
lambda=par(2);

Fkoeff=@(t) [[lambda,-1,-1; eta*t*(1-eta*t)-eta,lambda,-eta*t; (1-eta*t),1,0],[1,0,0;0,1,0;0,0,0]];
f=@(t) [(lambda-1)*exp(-t)*sin(t)-exp(-2*t)*sin(t);
      exp(-2*t)*cos(t) - 2*exp(-2*t)*sin(t) - exp(-t)*sin(t)*(eta+eta*t*(eta*t-1)) + lambda*exp(-2*t)*sin(t)-eta*t*exp(-t)*cos(t);
      exp(-2*t)*sin(t)-exp(-t)*sin(t)*(eta*t-1)
];

sol=@(t)[exp(-t).*sin(t);exp(-2*t).*sin(t);exp(-t).*cos(t)];
solD1=@(t)[-exp(-t).*sin(t)+exp(-t).*cos(t); -2*exp(-2*t).*sin(t)+exp(-2*t).*cos(t); -exp(-t).*cos(t)-exp(-t).*sin(t)];

%Type 1
BVa = [1,0,0];%;0,0,1];
ga = [0];%;1];
BVb = [1-eta,1,0];
gb = exp(-2)*sin(1)-exp(-1)*sin(1)*(eta-1);
% Type 2 (IV)
BVa = [0,1,0;1,0,0];
ga = [0;0];
BVb = [];
gb = [];


N=3;
l=[1,1,0];

a=0;
b=1;
end

%Temporary sovle dae1 with bvpsuite in case of overdet = 0
if(overdet == 0 && (dae==6) )
      global predefgrad
      if(colltype == 'u')
      predefgrad = [0,m];
      else
          predefgrad = [1,m];
      end
      opt=options_cl('run');
      opt.x1=tau;
      [bvpsol,solopt]=bvpsuite_cl(bvpfilestring,opt);
      z=zeros(N,length(tau));
      for l=1:length(tau)
          for d=1:N
              z(d,l)=bvpsol.valx1tau(d,l*(m+1)-m);
          end
      end
      resid=0;
else

%% Configure collocation parameters
tau=a+tau*(b-a);
switch colltype
      case 'u'
          rho=1/(m+1):1/(m+1):1-1/(m+1);
      case 'g'
          rho=gauss(m);
end
%% Define additional points (at reference interval [0,1]) at which conditions are set

%sigma=sigma(floor(length(sigma)/2):ceil(length(sigma)/2));
switch overdet
      case 0
          sigma=[];
      case 1
          sigma=1/2*[rho(1),[rho(2:end)+rho(1:end-1)],rho(end)+1];
      case 2
```

```
        sigma=sigma(floor(length(sigma)/2):ceil(length(sigma)/2));
    case 3
        sigma=1/2;
        if(~isempty(find(rho == sigma,1)))
            if(length(rho)>1)
                sigma=(sigma+rho(find(rho>sigma,1)))/2;
            else
                sigma=3/4;
            end
        end
end




%% Now the computation of solution starts

tic
%m=length(rho);
z=length(sigma);

h=tau(2:end)-tau(1:end-1);

dim=(sum(l)+m*N)*length(h);
dim1=(sum(l)+(m+z)*N)*length(h);%(0+(m+z)*N)*length(h);
%A=spalloc(dim,dim,dim*(m+sum(l)));
bVec=zeros(dim1,1);

t=tau(1:end-1)'*ones(1,length(rho))+h'*rho;
if(z>0)
    s=tau(1:end-1)'*ones(1,length(sigma))+h'*sigma;
else
    s=[];
end


%Compute coefficents of RK-basis-polynomials in monom-basis
PHI = zeros(max(l),max(l));
for i=1:max(l)
   tmp = phi(i);
   PHI(i,max(l)-length(tmp)+1:max(l))=tmp;
end

PSI = zeros(m,max(l)+1,m+max(l)+1);
for k=1:m
    for i=0:max(l)
        tmp=psi(rho,k,i);
        PSI(k,i+1,end-length(tmp)+1:end)=tmp;
    end
end

%Collocate space for non-zero-entries of A
row=1;
nze=length(ga)*N*max(l) + (length(h)-1)*(sum(l.^2/2+l/2)+N*max(l)*(m+1)) + length(h)*(m+z)*N*((sum(l.^2/2+l/2)+N*(max(l)+1)*m));
nze=nze*2;
a=zeros(nze,1);
rows=zeros(nze,1);
cols=zeros(nze,1);
if(overdet)
    L2=lagrangeInt([rho,sigma],1);
else
    L2=lagrangeInt(rho,1);
end
L=sqrtm(L2);
Wnze=length(h)*N*(m+z)^2+length(ga)+length(gb)+(length(h)-1)*sum(l);
Wrows=zeros(Wnze,1);
Wcols=zeros(Wnze,1);
w=zeros(Wnze,1);
ind=1;
Wind=1;
%Start collocation
for int=1:length(h)
   %Write initial conditions
    if(int==1)
```

```
        for bc=1:length(ga)
            for j=1:N
                for d=1:l(j)
                     a(ind)=BVa(bc,(d-1)*N+j);
                     rows(ind)=row;
                     cols(ind)=(sum(l)+m*N)*(int-1) + sum(l(1:j-1)) + m*(j-1)+d;
                     ind=ind+1;
                  %  A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j-1))+m*(j-1)+d) = BVa((d-1)*length(ga)+j,bc);
                     bVec(row)=ga(bc);
                end
            end
            Wrows(Wind)=row;
            Wcols(Wind)=row;
            w(Wind)=1;
            Wind=Wind+1;
            row = row+1;
        end
    else %Write continous conditions
         for j=1:N
             for d=1:l(j)
                  for k = d:l(j)
                       a(ind)=polyval(PHI(k-d+1,:),h(int-1));
                       rows(ind)=row;
                       cols(ind) = (sum(l)+m*N)*(int-2)+sum(l(1:j-1)) + m*(j-1)+k;
                       ind=ind+1;
                   %   A(row,(sum(l)+m*N)*(int-2)+sum(l(1:j-1))+m*(j-1)+k) = polyval(PHI(k-d+1,:),h(int-1));
                  end
                  for k=1:m
                       a(ind)=polyval(reshape(PSI(k,l(j)-d+2,:), m+max(l)+1,1), 1) * h(int-1)^(l(j)-d+1);
                       rows(ind)=row;
                       cols(ind)=(sum(l)+m*N)*(int-2)+sum(l(1:j))+m*(j-1)+k;
                       ind=ind+1;
                    %  A(row,(sum(l)+m*N)*(int-2)+sum(l(1:j))+m*(j-1)+k) = polyval(reshape(PSI(k,l(j)-d+2,:),m+max(l)+1,1),1)*h(int-1)^(l(j)-d+1);
                  end
                       a(ind)=-1;
                       rows(ind)=row;
                       cols(ind)=(sum(l)+m*N)*(int-1)+sum(l(1:j-1))+m*(j-1)+d;
                       ind=ind+1;
                 %A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j-1))+m*(j-1)+d)=-1;
                  Wrows(Wind)=row;
                  Wcols(Wind)=row;
                  w(Wind)=weight;
                  Wind=Wind+1;
                  row = row+1;
             end
         end
    end
    %Write endpoint conditions
    if(int==length(h))
        for bc=1:length(gb)
            for j=1:N
                for k=1:l(j)
                     for d=1:k
                          a(ind)=a(ind) + BVb(bc,(d-1)*N+j) * polyval(PHI(k-d+1,:),h(int));
                          rows(ind)=row;
                     cols(ind)=(sum(l)+m*N) * (int-1)+sum(l(1:j-1)) + m*(j-1)+d;
                     ind=ind+1;
                     end

                end
                for k1 = 1:m
                            for d=1:l(j)
                            a(ind)=a(ind) + BVb(bc,(d-1)*N+j) * polyval(reshape(PSI(k1,l(j)-d+2,:), m+max(l)+1,1),1) * h(int)^(l(j)-d+1);
                  %  A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j1))+m*(j1-1)+k1) = A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j1))+m*(j1-1)+k1)+Fval(j,(d-1)*N+j1) * polyval(reshape(PSI(k1,l(j1)-d+2,:), m+max(l)+1,1), (t(int,k)-tau(int))/h(int)) *h(int)^(l(j1)-d+1);
                            end
                       rows(ind)=row;
                       cols(ind)=(sum(l)+m*N) * (int-1) + sum(l(1:j)) + m*(j-1) + k1;
                       ind=ind+1;
                 end
```

```
            end
            bVec(row)=gb(bc);
            Wrows(Wind)=row;
            Wcols(Wind)=row;
            w(Wind)=1;
            Wind=Wind+1;
            row = row+1;
        end
    end

    % Write conditions at collocation points t_ij
    if(z>0)
        ip = [t(int,:),s(int,:)];
    else
        ip = t(int,:);
    end

    for k=1:m+z
        Fval = Fkoeff(ip(k));
        fval = f(ip(k));
        for j=1:N

            for j1 = 1:N
                for k1 = 1:l(j1)
                    for d=1:k1
                    a(ind)=a(ind) + Fval(j,(d-1)*N+j1) * polyval(PHI(k1-d+1,:),ip(k)-tau(int));
                %   A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j1-1))+m*(j1-1)+k1) = A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j1-1)) + m*(j1-1)+k1)+Fval(j,(d-1)*N+j1) * polyval(PHI(k1-d+1,:),t(int,k)-tau(int));
                    end
                    rows(ind)=row;
                    cols(ind)=(sum(l)+m*N) * (int-1) + sum(l(1:j1-1)) + m*(j1-1)+k1;
                    ind=ind+1;
                end
            end
             for j1 = 1:N
                for k1 = 1:m
                        for d=1:l(j1)+1
                        a(ind)=a(ind) + Fval(j,(d-1)*N+j1) * polyval(reshape(PSI(k1,l(j1)-d+2,:), m+max(l)+1,1), (ip(k)-tau(int))/h(int)) * h(int)^(l(j1)-d+1);
                 %  A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j1))+m*(j1-1)+k1) = A(row,(sum(l)+m*N)*(int-1)+sum(l(1:j1))+m*(j1-1)+k1)+Fval(j,(d-1)*N+j1) *polyval(reshape(PSI(k1,l(j1)-d+2,:),m+max(l)+1,1), (t(int,k)-tau(int))/h(int))*h(int)^(l(j1)-d+1);
                        end
                     rows(ind)=row;
                     cols(ind)=(sum(l)+m*N)* (int-1) + sum(l(1:j1)) + m*(j1-1) + k1;
                     ind=ind+1;
                end
             end
             bVec(row) = fval(j);
             Wrows(Wind:(Wind+(m+z)-1))=row*ones(m+z,1);
             startW=(sum(l)+(m+z)*N)*(int-1)+j;
             Wcols(Wind:(Wind+(m+z)-1)) = (startW:N:startW+(m+z)*N-1);
        %      w(Wind:(Wind+(m+z)-1))=sqrt(h(int))*L(k,:); %L2-Norm
             w(Wind:(Wind+(m+z)-1)) = ((row*ones(m+z,1)) == (startW:N:startW+(m+z)*N-1)');
%                  % nur gewichtet
             Wind=Wind+m+z;
             row = row+1;
        end
    end
    %W((row-(m+z)*N):(row-1),(row-(m+z)*N):(row-1))=sqrt(h(int))*L;
end
ind=ind-1;
dim1=max(rows(1:ind));
bVec=bVec(1:dim1);
A=sparse(rows(1:ind),cols(1:ind),a(1:ind),dim1,dim,nze);
%A=sparse(rows(1:ind),cols(1:ind),a(1:ind),length(rows),length(cols),nze);
% W=spdiags(w,0,dim1,dim1);
%W=speye(length(rows),length(rows));
% A1=A(find(w~=weight),:);
% b1=bVec(find(w~=weight),:);
% A2=A(find(w==weight),:);
% b2=bVec(find(w==weight),:);
```

```
%coeff=A\bVec; %Funktionierende Version
%coeff=[full(A);zeros(1,dim)]\[(bVec);0]; %Erzwinge LeastSquare ohne Gewichte
%coeff=lsqlin(A,bVec,A1,b1,A2,b2);
W=sparse(Wrows,Wcols,w,dim1,dim1,Wnze);
     %W=speye(dim1,dim1); % in 2-Norm ohne Gewicht

if(overdet)
     coeff=(W*A)\(W*bVec);
else
     %coeff=[full(A);zeros(1,dim)]\[(bVec);0]; %Erzwinge LeastSquare ohne Gewichte
     coeff=A\bVec;
end
toc

% S=svd(full(W*A));
% fprintf('W*A: %0.5g & %0.5g & %0.5g \\\\ \r\n',max(S),min(S),max(S)/min(S));
% S=svd(full(W));
% fprintf('W: %0.5g & %0.5g & %0.5g \\\\ \r\n',max(S),min(S),max(S)/min(S));
% S=svd(full(A));
% fprintf('A: %0.5g & %0.5g & %0.5g \\\\ \r\n',max(S),min(S),max(S)/min(S));


fprintf('Size of system: %d x %d \r\n', size(W*A));
fprintf('Number of intervals: %d \r\n', length(h));
fprintf('Number of coll points: %d \r\n', m);
fprintf('Number of additional points: %d \r\n', z);
resid = norm(W*(A*coeff-bVec),2)^2;
fprintf('Residuum ||A*z-b||^2: %0.5g \r\n',resid);
%Compute values at mesh points
z=zeros(N,length(tau));
normerror=zeros(N,1);
for i=1:length(h)
     for j=1:N
          if(l(j)>0)
               z(j,i) = coeff((sum(l)+m*N)*(i-1)+sum(l(1:j-1))+m*(j-1)+1);
          else
               for k=1:m
                    z(j,i) = z(j,i) + coeff((sum(l)+m*N)*(i-1) + sum(l(1:j)) + m*(j-1)+k) * polyval(reshape(PSI(k, l(j)+1, :), m+max(l)+1, 1), 0);
               end
          end
     end
end
for j=1:N
     for k=1:l(j)
          z(j,end) = z(j,end) + coeff((sum(l)+m*N) * (i-1)+sum(l(1:j-1)) + m*(j-1)+k) * polyval(PHI(k,:),h(end));
     end
     for k=1:m
          z(j,end) = z(j,end) + coeff((sum(l)+m*N)* (i-1) + sum(l(1:j)) + m*(j-1) + k) * polyval(reshape(PSI(k, l(j)+1, :), m+max(l)+1, 1), 1) * h(end)^(l(j));
     end
end

%Alternative computation of values at mesh points by evaluating the
%polynomials at the right endpoint
z2=zeros(N,length(tau));
for i=1:length(h)
     for j=1:N
          for k=1:l(j)
               z2(j,i+1) = z2(j,i+1) + coeff((sum(l)+m*N) * (i-1)+sum(l(1:j-1)) + m*(j-1)+k) * polyval(PHI(k,:),h(i));
          end
          for k=1:m
               z2(j,i+1) = z2(j,i+1) + coeff((sum(l)+m*N) * (i-1)+sum(l(1:j)) + m*(j-1)+k) * polyval(reshape(PSI(k,l(j)+1,:), m+max(l)+1,1), 1) * h(i)^(l(j));
          end
     end
end
%% End solver
   oldtau=tau;
if(nargin>8 && ~isempty(teval)) %Evaluate solution at arbitrary t-values
   z=zeros(N,length(teval));
   for tp = teval
      i=find(tp>tau,1,'last');
      for j=1:N
         for k=1:l(j)
              z(j,i+1) = z(j,i+1) + coeff((sum(l)+m*N) * (i-1) + sum(l(1:j-1)) + m*(j-1)+k)
```

```
                * polyval(PHI(k,:),tp-tau(i));
            end
            for k=1:m
                z(j,i+1) = z(j,i+1) + coeff((sum(l)+m*N) * (i-1) + sum(l(1:j)) + m*(j-1)+k) * polyval(reshape(PSI(k,l(j)+1,:), m+max(l)+1, 1),1) * (h(i))^(l(j));
            end
        end
    end
    tau=teval;
end
end
%% Compute error
err=abs(z-sol(tau));

%% Compute Integral-norm of error
for d=0:1 %Iterate to get the H1D-norm
    if(overdet)
        L2=lagrangeInt([0,rho,sigma,1],1);
        tmp=zeros(length(h),N,length(rho)+length(sigma)+2);
    else
        L2=lagrangeInt([0,rho,1],1);
        tmp=zeros(length(h),N,length(rho)+2);
    end
    for int=1:length(h)
        if(overdet)
            ip = [oldtau(int),t(int,:),s(int,:),oldtau(int+1)];
        else
            ip = [oldtau(int),t(int,:),oldtau(int+1)];
        end
        for j=1:N
            if(d<=l(j))
                for i=1:length(ip)
                    tp=ip(i);
                    if(d==0)
                        tmpex=sol(tp);
                    elseif(d==1)
                        tmpex=solD1(tp);
                    end
                    tmp(int,j,i)=-tmpex(j);
                    for k=d+1:l(j)
                        tmp(int,j,i) = tmp(int,j,i) + coeff((sum(l)+m*N) * (int-1)+sum(l(1:j-1)) + m*(j-1)+k) * polyval(PHI(k-d,:),tp-oldtau(int));
                    end
                    for k=1:m
                        if(l(j)+1-d>0)
                            tmp(int,j,i) = tmp(int,j,i) + coeff((sum(l)+m*N)*(int-1)+sum(l(1:j)) + m*(j-1)+k) * polyval(reshape(PSI(k,l(j)+1-d,:), m+max(l)+1,1), (tp-oldtau(int))/h(int)) * (h(int))^(l(j)-d);
                        else
                            tmp(int,j,i) = tmp(int,j,i) + coeff((sum(l)+m*N)*(int-1)+sum(l(1:j)) + m*(j-1)+k) * polyval(psi(rho,k,l(j)-d), (tp-oldtau(int))/h(int)) * (h(int))^(l(j)-d);
                        end
                    end
                end
                normerror(j) = normerror(j) + h(int) * reshape(tmp(int,j,:),1,length(ip)) * L2*reshape(tmp(int,j,:), length(ip), 1);
            end
        end
    end
    if(d==0)
        L2normerror = normerror; %save the L2-norm-error
    end
end
maxerror=zeros(N,1);
for j=1:N
    maxerror(j)=max(err(j,:));
    %maxerror(j)=max(max(tmp(:,j,:)));
    fprintf('Abs. Fehler z_%d: %0.5g \r\n',j,maxerror(j));
    fprintf('||Abs. Fehler z_%d||_L2: %0.5g \r\n',j,sqrt(L2normerror(j)));
    fprintf('||Abs. Fehler z_%d||_H1D: %0.5g \r\n',j,sqrt(normerror(j)));
end
if(nargin>=10)
    [addTestComboM,~]=size(addTestCombo);
    z(N+1:N+addTestComboM,:)=addTestCombo*z;
```

```
    err(N+1:N+addTestComboM,:)=abs(z(N+1:N+addTestComboM,:)-addTestCombo*sol(tau));
    for j=1:addTestComboM
        maxerror(N+j)=max(err(N+j,:));
        fprintf('Abs. Fehler z_%d (Linkomb %d): %0.5g \r\n',N+j, addTestComboM(j),
            ↪ maxerror(N+j));
    end
end

%%
%incont=abs(z(:,2:end-1)-z2(:,2:end-1));

%% Plot solutions and absolute errors

%spy(A)
if(showPlot>0)
  % close all
for j=1:N
   figure
   plot(tau,z(j,:));
end

for j=1:N
   figure
   semilogy(tau,err(j,:));
end

  % figure
  % semilogy(tau(2:end-1),incont);

end
%% Show specification of A
fprintf('Dimension of matrix: %d x %d \r\n',dim1,dim);
fprintf('Non-zero entries in matrix: %d \r\n',nnz(A));
end


function poly = phi(k)
 poly = [1/factorial(k-1),zeros(1,k-1)];
end

function poly = psi(rho,k,order)
poly = 1;
 for j=1:length(rho)
     if(j~=k)
        poly = conv(poly,[1/(rho(k)-rho(j)),-rho(j)/(rho(k)-rho(j))]);
     end
 end
 for j=1:order
     poly = polyint(poly);
 end
 for j=order:-1
     poly = polyder(poly);
 end
end

function L = lagrangeInt(tau,d)
m=length(tau);
M=zeros(m,m);
for i=1:m
   M(i,:) = psi(tau,i,0);
end
L=zeros(m,m);
for i=1:m
    for j=1:i
        tmp=polyint(conv(M(i,:),M(j,:)));
        L(1+(i-1)*d:i*d,1+(j-1)*d:j*d)=(polyval(tmp,1)-polyval(tmp,0))*eye(d);
        L(1+(j-1)*d:j*d,1+(i-1)*d:i*d)=L(1+(i-1)*d:i*d,1+(j-1)*d:j*d);
    end
end
end

function [rho]=gauss(k)

%psi(:,:,i) for n-th order psi(:,:,n-i) equals i-th derivative
%for example for 3-rd order:
%psi(:,:,1) ... second derivative
%psi(:,:,2) ... first derivative
%psi(:,:,3) ... no derivative
```

```
switch k
  case 1
 rho=[0.5];
 psi(1,:,1)=[0 1 0];
 psi(1,:,2)=[0.5 0 0];
     case 2
      %C: rho: ci in Gaussian collocation, zeros of Legendre Polynomials shiftet to [0,1]
 %C: transformation from [-1,1] to [0,1]: y= 1/2x+1/2
 rho=[0.21132486540518711774542560974 8  0.78867513459481288225457439025 2];
 psi(1,:,1)=[0 -0.86602540378443864676372317075 5  1.36602540378443864676372317076 0];
 psi(2,:,1)=[0  0.86602540378443864676372317075 5  -0.36602540378443864676372317075 4 0];
 %0.th derivative
 %firs coll.point
 psi(1,:,2)=[-0.28867513459481288225457439025 2  0.68301270189221932338186158537 8 0 0];
 %second coll.point
 psi(2,:,2)=[0.28867513459481288225457439025 2  -0.18301270189221932338186158537 7 0 0];
    case 3
 rho=[1/2-1/10*15^(1/2) 1/2 1/2+1/10*15^(1/2)];
 psi(1,:,1)=[0 10/9 -5/3-1/6*15^(1/2) 5/6+1/6*15^(1/2) 0];
 psi(2,:,1)=[0 -20/9 10/3 -2/3 0];
 psi(3,:,1)=[0 10/9 -5/3+1/6*15^(1/2) 5/6-1/6*15^(1/2) 0];
 psi(1,:,2)=[5/18 -5/9-1/18*15^(1/2) 5/12+1/12*15^(1/2) 0 0];
 psi(2,:,2)=[-5/9 10/9 -1/3 0 0];
 psi(3,:,2)=[5/18 -5/9+1/18*15^(1/2) 5/12-1/12*15^(1/2) 0 0];
    case 4

 rho=[1/2-1/70*(525+70*30^(1/2))^(1/2) 1/2-1/70*(525-70*30^(1/2))^(1/2) 1/2+1/70*(525-70*30^(1/2))^(1/2) 1/2+1/70*(525+70*30^(1/2))^(1/2)];
 psi(1,:,1)=[0 -245/24/(525+70*30^(1/2))^(1/2)*30^(1/2) 7/36*(105+(525+70*30^(1/2))^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        -7/24*(45+(525+70*30^(1/2))^(1/2)+30^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        1/120*(35+(525+70*30^(1/2))^(1/2))*(10+30^(1/2))*30^(1/2)/(525+70*30^(1/2))^(1/2) 0];
 psi(2,:,1)=[0 245/24/(525-70*30^(1/2))^(1/2)*30^(1/2) -7/36*(105+(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        -7/24*(-45+30^(1/2)-(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        1/120*(35+(525-70*30^(1/2))^(1/2))*(-10+30^(1/2))*30^(1/2)/(525-70*30^(1/2))^(1/2) 0];
 psi(3,:,1)=[0 -245/24/(525-70*30^(1/2))^(1/2)*30^(1/2) -7/36*(-105+(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        7/24*(-45+30^(1/2)+(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        1/120*(-35+(525-70*30^(1/2))^(1/2))*(-10+30^(1/2))*30^(1/2)/(525-70*30^(1/2))^(1/2) 0];
 psi(4,:,1)=[0 245/24/(525+70*30^(1/2))^(1/2)*30^(1/2) 7/36*(-105+(525+70*30^(1/2))^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        7/24*(45-(525+70*30^(1/2))^(1/2)+30^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        1/120*(-35+(525+70*30^(1/2))^(1/2))*(10+30^(1/2))*30^(1/2)/(525+70*30^(1/2))^(1/2) 0];
 psi(1,:,2)=[-49/24/(525+70*30^(1/2))^(1/2)*30^(1/2) 7/144*(105+(525+70*30^(1/2))^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        -7/72*(45+(525+70*30^(1/2))^(1/2)+30^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        1/240*(35+(525+70*30^(1/2))^(1/2))*(10+30^(1/2))*30^(1/2)/(525+70*30^(1/2))^(1/2) 0 0];
 psi(2,:,2)=[49/24/(525-70*30^(1/2))^(1/2)*30^(1/2) -7/144*(105+(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        7/72*(45-30^(1/2)+(525-70*30^(1/2))^(1/2))*30^(1/2)/(525-70*30^(1/2))^(1/2) ...
        1/240*(35+(525-70*30^(1/2))^(1/2))*(-10+30^(1/2))*30^(1/2)/(525-70*30^(1/2))^(1/2) 0 0];
 psi(3,:,2)=[-49/24/(525-70*30^(1/2))^(1/2)*30^(1/2) -7/144*(-105+(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        7/72*(-45+30^(1/2)+(525-70*30^(1/2))^(1/2))/(525-70*30^(1/2))^(1/2)*30^(1/2) ...
        1/240*(-35+(525-70*30^(1/2))^(1/2))*(-10+30^(1/2))*30^(1/2)/(525-70*30^(1/2))^(1/2) 0 0];
 psi(4,:,2)=[49/24/(525+70*30^(1/2))^(1/2)*30^(1/2) 7/144*(-105+(525+70*30^(1/2))^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        7/72*(45-(525+70*30^(1/2))^(1/2)+30^(1/2))/(525+70*30^(1/2))^(1/2)*30^(1/2) ...
        1/240*(-35+(525+70*30^(1/2))^(1/2))*(10+30^(1/2))*30^(1/2)/(525+70*30^(1/2))^(1/2) 0 0];

case 5

 rho=[1/2-1/42*(245+14*70^(1/2))^(1/2) 1/2-1/42*(245-14*70^(1/2))^(1/2) 1/2 1/2+1/42*(245-14*70^(1/2))^(1/2) 1/2+1/42*(245+14*70^(1/2))^(1/2)];
 psi(1,:,1)=[0 567/25/(35+2*70^(1/2))*70^(1/2) -27/40*(84+(245+14*70^(1/2))^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
        3/20*(343+9*(245+14*70^(1/2))^(1/2)+2*70^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
        -3/280 *(1911+77*(245+14*70^(1/2))^(1/2)+42*70^(1/2)
```

```
                ↪ +(245+14*70^(1/2))^(1/2)*70^(1/2))  /(35+2*70^(1/2))*70^(1/2) ...
            3/280*(21+(245+14*70^(1/2))^(1/2))*(14+70^(1/2))*70^(1/2)/(35+2*70^(1/2))  0];
psi(2,:,1)=[0  567/25/(-35+2*70^(1/2))*70^(1/2) ↷
    ↪ -27/40*(84+(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            -3/20*(-343+2*70^(1/2)-9*(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            3/280*(-1911+42*70^(1/2)-77*(245-14*70^(1/2))^(1/2) ↷
                ↪ +(245-14*70^(1/2))^(1/2)*70^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            -3/280*(21+(245-14*70^(1/2))^(1/2))*(-14+70^(1/2))*70^(1/2)/(-35+2*70^(1/2)) ↷
                ↪ 0];
psi(3,:,1)=[0  336/25  -168/5  1232/45  -112/15  8/15  0];
psi(4,:,1)=[0  567/25/(-35+2*70^(1/2))*70^(1/2) ↷
    ↪ 27/40*(-84+(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            -3/20*(-343+2*70^(1/2)+9*(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            -3/280*(1911-42*70^(1/2)-77*(245-14*70^(1/2))^(1/2) ↷
                ↪ +(245-14*70^(1/2))^(1/2)*70^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            3/280*(-21+(245-14*70^(1/2))^(1/2))*(-14+70^(1/2))*70^(1/2)/(-35+2*70^(1/2)) ↷
                ↪ 0];
psi(5,:,1)=[0  567/25/(35+2*70^(1/2))*70^(1/2) ↷
    ↪ 27/40*(-84+(245+14*70^(1/2))^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            -3/20*(-343+9*(245+14*70^(1/2))^(1/2)-2*70^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            3/280*(-1911+77*(245+14*70^(1/2))^(1/2)-42*70^(1/2) ↷
                ↪ +(245+14*70^(1/2))^(1/2)*70^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            -3/280*(-21+(245+14*70^(1/2))^(1/2))*(14+70^(1/2))*70^(1/2)/(35+2*70^(1/2))  0];
psi(1,:,2)=[189/50/(35+2*70^(1/2))*70^(1/2) ↷
    ↪ -27/200*(84+(245+14*70^(1/2))^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            3/80*(343+9*(245+14*70^(1/2))^(1/2)+2*70^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            -1/280*(1911+77*(245+14*70^(1/2))^(1/2)+42*70^(1/2) ↷
                ↪ +(245+14*70^(1/2))^(1/2)*70^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            3/560*(21+(245+14*70^(1/2))^(1/2))*(14+70^(1/2))*70^(1/2)/(35+2*70^(1/2))  0  0];
psi(2,:,2)=[189/50/(-35+2*70^(1/2))*70^(1/2) ↷
    ↪ -27/200*(84+(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            3/80*(343-2*70^(1/2)+9*(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            1/280*(-1911+42*70^(1/2)-77*(245-14*70^(1/2))^(1/2) ↷
                ↪ +(245-14*70^(1/2))^(1/2)*70^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
            -3/560*(21+(245-14*70^(1/2))^(1/2))*(-14+70^(1/2))*70^(1/2)/(-35+2*70^(1/2)) ↷
                ↪ 0  0];
psi(3,:,2)=[56/25  -168/25  308/45  -112/45  4/15  0  0];
psi(4,:,2)=[189/50/(-35+2*70^(1/2))*70^(1/2) ↷
    ↪ 27/200*(-84+(245-14*70^(1/2))^(1/2))/(-35+2*70^(1/2))*70^(1/2) ↷
    ↪ -3/80*(-343+2*70^(1/2)+ 9*(245-14*70^(1/2))^(1/2))/ (-35+2*70^(1/2))*70^(1/2) ...
           -1/280*(1911-42*70^(1/2)-77*(245-14*70^(1/2))^(1/2)+ ↷
               ↪ (245-14*70^(1/2))^(1/2)*70^(1/2))/(-35+2*70^(1/2))*70^(1/2) ...
           3/560*(-21+(245-14*70^(1/2))^(1/2))*(-14+70^(1/2))*70^(1/2)/(-35+2*70^(1/2))  0 ↷
               ↪ 0];
psi(5,:,2)=[189/50/(35+2*70^(1/2))*70^(1/2) ↷
    ↪ 27/200*(-84+(245+14*70^(1/2))^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            3/80*(343-9*(245+14*70^(1/2))^(1/2)+2*70^(1/2))*70^(1/2)/(35+2*70^(1/2)) ...
            1/280*(-1911+77*(245+14*70^(1/2))^(1/2)-42*70^(1/2)+ ↷
                ↪ (245+14*70^(1/2))^(1/2)*70^(1/2))/(35+2*70^(1/2))*70^(1/2) ...
            -3/560*(-21+(245+14*70^(1/2))^(1/2))*(14+70^(1/2))*70^(1/2)/(35+2*70^(1/2))  0 ↷
                ↪ 0];
      case 6

rho=[0.33765242898423986094e-1 0.16939530676686774317 0.38069040695840154568 ↷
    ↪ 0.61930959304159845432 0.83060469323313225683 0.96623475710157601391];
psi(1,:,1)=[0 -8.1412617290086756177 28.978672208695686824 -40.408362140839295291 ↷
    ↪ 27.785390555068868256 -9.6944498478780709320 1.5656732001510719330 0];
psi(2,:,1)=[0 24.533738740695531559 -83.334379226361945114 107.81105264377978693 ↷
    ↪ -64.863346973441684340 16.973778445028729194 -.94046284317634892902 0];
psi(3,:,1)=[0 -36.168340495472103373 113.68329746902200059 -130.85406519770904404 ↷
    ↪ 65.101388388983683720 -12.145253252968680079 .61693005543048870860 0];
psi(4,:,1)=[0 36.168340495472103373 -103.32674550381061965 104.96268528468059170 ↷
    ↪ -44.848707621074554040 7.6576120141334378913 -.37922770211461375460 0];
psi(5,:,1)=[0 -24.533738740695531559 63.868053217811244240 -59.145237622403034740 ↷
    ↪ 23.711846151968643422 -3.9123422341959200109 .19180001403866795482 0];
psi(6,:,1)=[0 8.1412617290086756177 -19.868898165356366883 17.633927032490995438 ↷
    ↪ -6.8865705015049570236 1.1206548758805039360 -.54712724329265912892e-1 0];
psi(1,:,2)=[-1.1630373898583822311 4.8297787014492811373 -8.0816724281678590582 ↷
    ↪ 6.9463476387672170640 -3.2314832826260236440 .78283660007553596650 0 0];
psi(2,:,2)=[3.5048198200993616513 -13.889063204393657519 21.562210528755957386 ↷
    ↪ -16.215836743360421085 5.6579261483429097313 -.47023142158817446451 0 0];
psi(3,:,2)=[-5.1669057850674433390 18.947216244837000098 -26.170813039541808808 ↷
    ↪ 16.275347097245920930 -4.0484177509895600263 .30846502771524435430 0 0];
psi(4,:,2)=[5.1669057850674433390 -17.221124250635103275 20.992537056936118340 ↷
    ↪ -11.212176905268638510 2.5525373380444792971 -.18961385105730687730 0 0];
psi(5,:,2)=[-3.5048198200993616513 10.644675536301874040 -11.829047524480606948 ↷
    ↪ 5.9279615379921608555 -1.3041140780653066703 .95900007019333977410e-1 0 0];
psi(6,:,2)=[1.1630373898583822311 -3.3114830275593944805 3.5267854064981990876 ↷
```

```
      ↪ −1.7216426253762392559 .37355162529350131200 −.27356362164632956446e−1 0 0];
  case 7
   rho=[.254460438286207377369051579761e−1, .129234407200302780068067613360, ↷
      ↪ .297077424311301416546696793962, .500000000000000000000000000000, ↷
      ↪ .702922575688698583453303206038, .870765592799697219931932386640, ↷
      ↪ .974553956171379262263094842024];
 case 8
   rho=[.198550717512318841582195657153e−1, .101666761293186630204223031762, ↷
      ↪ .237233795041835507091130475405, .408282678752175097530261928820, ↷
      ↪ .591717321247824902469738071180, .762766204958164492908869524595, ↷
      ↪ .898333238706813369795776968238, .980144928248768115841780434285];
 case 9
   rho=[.159198802461869550822118985482e−1, .819844463366821028502851059651e−1, ↷
      ↪ .193314283649704801345648980329, .337873288298095535480730992678, ↷
      ↪ .500000000000000000000000000000, .662126711701904464519269007322, ↷
      ↪ .806685716350295198654351019671, .918015553663317897149714894035, ↷
      ↪ .984080119753813044917788101452];
 case 10
   rho=[.130467357414141399610179939578e−1, .674683166555077446339516557883e−1, ↷
      ↪ .160295215850487796882836317443, .283302302935376404600367028417, ↷
      ↪ .425562830509184394557586999435, .574437169490815605442413000565, ↷
      ↪ .716697697064623595399632971583, .839704784149512203117163682557, ↷
      ↪ .932531683344492255366048344212, .986953264258585860038982006042];
 case 11
   rho=[.108856709269715035980309994386e−1, .564687001159523504624211153480e−1, ↷
      ↪ .134923997212975337953291873984, .240451935396594092037137165271, ↷
      ↪ .365228422023827513834234007300, .500000000000000000000000000000, ↷
      ↪ .634771577976172486165765992700, .759548064603405907962862834729, ↷
      ↪ .865076002787024662046708126016, .943531299884047649537578884652, ↷
      ↪ .989114329073028496401969000561];
 case 12
   rho=[.921968287664037465472545492536e−2, .479413718147625716607670669405e−1, ↷
      ↪ .115048662902847656481553083394, .206341022856691276351648790530, ↷
      ↪ .316084250500909903123654231678, .437383295744265542263779315268, ↷
      ↪ .562616704255734457736220684732, .683915749499090096876345768322, ↷
      ↪ .793658977143308723648351209470, .884951337097152343518446916606, ↷
      ↪ .952058628185237428339232933060, .990780317123359625345274545075];
 case 13
   rho=[.790847264070592526358527559645e−2, .412008003885110173967260817496e−1, ↷
      ↪ .992109546333450436028967552086e−1, .178825330279829889678007696502, ↷
      ↪ .275753624481776573561043573936, .384770842022432602967235939451, ↷
      ↪ .500000000000000000000000000000, .615229157977567397032764060549, ↷
      ↪ .724246375518223426438956426064, .821174669720170110321992303498, ↷
      ↪ .900789045366654956397103244791, .958799199611488982603273918250, ↷
      ↪ .992091527359294074736414724404];
 case 14
   rho=[.685809565159383057920136664797e−2, .357825581682132413318044303111e−1, ↷
      ↪ .863993424651175034051026286748e−1, .156353547594157264925990098490, ↷
      ↪ .242375681820922954017354640724, .340443815536055119782164087916, ↷
      ↪ .445972525646328168966877674890, .554027474353671831033122325110, ↷
      ↪ .659556184463944880217835912084, .757624318179077045982645359276, ↷
      ↪ .843646452405842735074009901510, .913600657534882496594897371325, ↷
      ↪ .964217441831786758668195569689, .993141904348406169420798633352];
 case 15
   rho=[.600374098975728575521714070669e−2, .313633037996470478461205261449e−1, ↷
      ↪ .758967082947863918996758396129e−1, .137791134319914976291906972693, ↷
      ↪ .214513913695730576231386631373, .302924326461218315051396314509, ↷
      ↪ .399402953001282738849685848303, .500000000000000000000000000000, ↷
      ↪ .600597046998717261150314151697, .697075673538781684948603685491, ↷
      ↪ .785486086304269423768613368627, .862208865680085023708093027307, ↷
      ↪ .924103291705213608100324160387, .968636696200352952153879473855, ↷
      ↪ .993996259010242714244782859293];    end
end
```

List of Figures

List of Tables

Bibliography

[1] U. Ascher and U. Bader. A new basis implementation for a mixed order boundary value ODE solver. *SIAM J. Scient. Stat. Comput.*, 8:483–500, 1987.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] W. Auzinger, O. Koch, and E. Weinmüller. Collocation methods for boundary value problems with an essential singularity. In I. Lirkov, S. Margenov, J. Wasniewski, and P. Yalamov, editors, *Large-Scale Scientific Computing*, volume 2907 of *Lecture Notes in Computer Science*, pages 347–354. Springer Verlag, 2004.

[8] W. Auzinger, O. Koch, and E. 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.

[9] W. Auzinger, H. Lehner, and E.B. Weinmüller. Defect-based a posteriori error estimation for index-1 DAEs. *BIT*, 2011

[10] Ch. Buchner and W. Schneider. Explosive crystallization in thin amorphous layers on heat conducting substrates. Poster-Presentation, 14th International Heat Transfer Conference, Washington D.C., USA, 08. - 13. August 2010.

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

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

[13] J. Cash, G. Kitzhofer, O. Koch, G. Moore, and E.B. Weinmüller. Numerical solution of singular two-point BVPs. *JNAIAM J. Numer. Anal. Indust. Appl. Math.*, 4:129–149, 2009.

[14] 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.

[15] N. Hale and D. Moore. A Sixth-Order Extension to the MATLAB `bvp4c` of J. Kierzenka and L. Shampine. Techn. Rept. No. NA-08/04, Oxford University Computing Laboratory, Oxford, United Kingdom, 2008.
Available at http://web.comlab.ox.ac.uk//files/720/NA-08-04.pdf.

[16] M. Hanke, R. März, C. Tischendorf, E.B. Weinmüller, and S. Wurm. Least-Squares Collocation for Higher Index Differential-Algebraic Equations., submitted to JCAM.

[17] 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.

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

[19] F.R. de Hoog and R. Weiss. The numerical solution of boundary value problems with an essential singularity. *SIAM J. Numer. Anal.*, 16:637–669, 1979.

[20] 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.

[21] F.R. de Hoog and R. Weiss. The application of Runge-Kutta schemes to singular initial value problems. *Math. Comp.*, 44:93–103, 1985.

[22] H. Keller. Approximation methods for nonlinear problems with application to two-point boundary value problems. *Math. Comp.*, 29:464–474, 1975.

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

[24] G. Kitzhofer. *Numerical Treatment of Implicit Singular BVPs.* Ph.D. Thesis, Inst. for Anal. and Sci. Comput., Vienna Univ. of Technology, Austria, 2013.

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

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

[27] G. Kitzhofer, G. Pulverer, O. Koch, Ch. Simon, and E. Weinmüller. The New MATLAB Code `bvpsuite` for the Solution of Singular Implicit BVPs. *JNAIAM J. Numer. Anal. Indust. Appl. Math.*, 5:113–134, 2010.
The version `bvpsuite1.1` is available at http://www.math.tuwien.ac.at/∼ewa.

[28] G. Kitzhofer, G. Pulverer, O. Koch, Ch. Simon, and E. Weinmüller. Manual: *BVPSUITE – A New MATLAB Code for Singular Implicit Boundary Value Problems*, 2009.
Available at http://www.math.tuwien.ac.at/∼ewa.

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

[30] 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.*, 79:129–149, 2009.

[31] A. Köppl. *Anwendung von Ratengleichungen auf anisotherme Kristallisation von Kunststoffen.* Ph. D. Thesis, Vienna Univ. of Technology, Austria, 1990.

[32] A. Köppl, J. Berger, and W. Schneider. Ausbreitungsgeschwindigkeit und Struktur von Kristallisationswellen. In *Proceedings of GAMM*, Stuttgard, Germany, 1987.

[33] G. Pulverer, G. Söderlind, and E.B. Weinmüller. Automatic grid control in adaptive BVP solvers. *Numer. Algorithms*, 2011

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

[35] R. D. Russell, and J. Christiansen. Adaptive Mesh Selection Strategies for Solving Boundary Value Problems. *SIAM J. Numer. Anal.*, 15:59–80, 1978.

[36] M. Schöbinger. A new version of a collocation code fro singukare BVPs: Nonlinear solver and its application to m- Laplacians. Master Thesis, Inst. for Anal. and Sci. Comput., Vienna Univ. of Technology, Austria, 2015.

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

[38] 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/`.

[39] L. Shampine, P. Muir, and H. Xu. A User-Friendly Fortran BVP Solver, *JNAIAM J. Numer. Anal. Indust. Appl. Math.*, 1:201–217, 2006.

[40] Ch. Simon. *Numerical Solution of Singular Eigenvalue Value Problems for Systems of ODEs with a Focus on Problems Posed on Semi-Infinite Intervals.* Master's thesis, Vienna Univ. of Technology, Vienna, Austria, 2009.

[41] 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.

[42] S. Staněk, G. Pulverer, and E.B. Weinmüller. Analysis and numerical solution of positive and dead core solution of singular Sturm-Liouville problems. *Adv. Difference Equ.*, 2010

[43] H. J. Stetter. Analysis of Discretization Methods for Ordinary Differential Equations. Springer-Verlag, Berlin-Heidelberg-New York, 1973.

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

[45] S. Wurm. Numerische Lösung von Algebro-Differentialgleichungen höheren Indizes mittels Kollokationscode bvpsuite. Bachelor's thesis, Vienna Univ. of Technology, Vienna, Austria, 2013.

[46] B. Kasberger. A Multi-Unit Auction with Identity-Dependent Externalities. Personal correspondence. 2016.