SBVP 1.0 - A MATLAB Solver for Singular Boundary Value Problems

Winfried Auzinger
Günter Kneisl
Othmar Koch
Ewa B. Weinmüller

ANUM Preprint No. 02/02

Technische
Universität
Wien

Vienna
University of
Technology

Institute for Applied Mathematics
and Numerical Analysis

Contact:

Winfried Auzinger
email: w.auzinger@tuwien.ac.at
URL: http://www.math.tuwien.ac.at/~winfried/

Günter Kneisl
email: eomer@gmx.at
URL: http://connect.to/eomer

Othmar Koch
email: othmar@fsmat.at
URL: http://fsmat.at/~othmar/

Ewa Weinmüller
email: e.weinmueller@tuwien.ac.at
URL: http://www.math.tuwien.ac.at/~ewa/

All:
Institut für Angewandte und Numerische Mathematik (E115)
Technische Universität Wien
Wiedner Hauptstraße 8–10
A-1040 Wien
Austria
URL: http://www.anum.tuwien.ac.at

The package SBVP 1.0 is freely available from
http://www.math.tuwien.ac.at/~ewa/

Für den Inhalt verantwortlich:
Dr. Winfried Auzinger, Dipl.-Ing. Günter Kneisl, Dr. Othmar Koch und Dr. Ewa Weinmüller, Wien.

Verlag:
Institut für Angewandte und Numerische Mathematik, Technische Universität Wien.

Contents

1 Introduction

1.1 Problem setting

The SBVP-package contains functions for solving boundary value problems for systems of nonlinear ODEs of the first order,

$$\begin{aligned} &\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \quad t \in (a, b), \\ &\mathbf{R}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}. \end{aligned}$$

The right-hand side of the differential equation may contain a singularity of the first kind, that is

$$\mathbf{f}(t, \mathbf{y}(t)) = \frac{1}{(t-a)} M(t) \cdot \mathbf{y}(t) + \mathbf{g}(t, \mathbf{y}(t)),$$

where M is a matrix which depends continuously on t and $\mathbf{g}$ is a smooth vector-valued function.

1.2 Solution approach

We decided to use collocation for the numerical solution of the underlying boundary value problems. A collocation solution is a piecewise polynomial function which satisfies the given ODE at a finite number of nodes (collocation points). This approach shows advantageous convergence properties compared to other direct higher order methods (see [7], [10]), which may be affected by order reductions and become inefficient in the presence of a singularity, see for example [8].

Furthermore, we decided to control the global error instead of monitoring the local error because of the unsmoothness of the latter near the singular point and order reductions it suffers from, cf. [5].

The mesh selection strategy is based on equidistribution of the global error. A detailed description of the error estimation and mesh selection algorithm is given in [4] and [3]. Further numerical aspects of the procedure are discussed in [1] and [2].

2 The package

2.1 Installation

Create a directory for the SBVP-package and unzip `SBVP.zip` into this directory. Then add it to the MATLAB search path. This is done by inserting the line

```
addpath('The full path of the directory you created');
```

into the MATLAB startup-file `startup.m`. You can locate this file by typing

```
>> which startup
```

at the MATLAB command line.
If no startup-file exists, you should create one in

```
...\Matlab\toolbox\local (Windows systems)
```

or

```
your Matlab directory, e.g. /home/username/matlab (Multiuser systems)
```

Now add search paths for the subdirectories `OutputFunctions` and `Demo`, too. When you call

```
>> startup
```

from within the MATLAB environment or restart[1] MATLAB, all SBVP functions will be available.

2.2 Files in this package

The solver package consists of the files

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

[1] On multiuser systems, you should start MATLAB from your MATLAB directory to ensure that the startup-file is executed.

2.3 Solver syntax

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

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

The input arguments are:

- `bvpfile` is the function handle or name of an m-file that defines the right-hand side of the differential equation, the boundary conditions and the associated Jacobians. `sbvp` automatically detects whether the BVP is linear. The `bvpfile` syntax is discussed in detail in §2.4.

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

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

  to obtain it. If `tau0` consists of the beginning and end of the integration interval only (`tau0 = [a b]`), `sbvp` automatically selects a suitable initial mesh on that integration interval.

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

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

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

- `bvpopt` is a parameter struct that determines the choice of collocation points, collocation basis, the tolerances for the zerofinder, etc. `bvpopt` should be generated by `sbvpset`, which is discussed in §2.5. If `bvpopt` is empty or not specified, `sbvp` evaluates

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

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

- `param1`, `param2` are parameters of the boundary value problem that are passed on to `bvpfile` (see the header of `BVPFilePattern` in §2.4 and `demofile2` at page 14).

The output arguments are:

- `tau` is the final mesh generated by `sbvp`.

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

2.4 The `bvpfile`

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

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

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

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

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

2.5 Solution options

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

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

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

Additionally,

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

alters an existing options structure `oldopt` and

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

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

Available options are

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

'**DegreeSelect**'	Selection mode of basis polynomial degree [{auto} | manual].
'**Degree**'	Highest degree of basis polynomials [integer {4}]. The collocation degree is set to the value of this property if the value of 'DegreeSelect' is 'manual'. Otherwise, the collocation degree is automatically determined from the tolerances.
'**Display**'	Displays information about the mesh adaptation process [0 | {1}].
'**ZfMethod**'	Defines the method for solving nonlinear problems [{Newton} | FSolve]. 'Newton' forces the use of damped Newton iteration while 'FSolve' uses the zerofinder provided by MATLAB.
'**ZfOpt**'	Parameter struct used by the zerofinding routines [struct]. This struct holds tolerances, termination conditions, etc. and should be generated by `optimset` (see §2.6).
'**OutputFcn**'	Function handle or name of output function (OF) [handle/string]. OFs are used to display intermediate results of the mesh adaptation process. At the beginning of this process, the solver calls `outputfcn(tcol0,ycol0,p,'init')` to initialize the OF. `tcol0` represents the initial collocation grid, `ycol0` is the initial approximation on `tcol0` and `p` is the collocation degree. After each mesh adaptation step with collocation grid `tcol` and approximation vector `ycol`, the solver calls `stop = outputfcn(tcol,ycol,p,'')`. If `stop` equals 1 or some termination condition of the mesh adaptation routine is met, the iteration is stopped and the solver calls `outputfcn(tcol,ycol,p,'done')`. Predefined OFs are `sbvpplot`, `sbvpphas2` and `sbvpphas3` (see §2.7).
'**OutputSel**'	Output selection indices [vector of integers]. This vector of indices specifies which components of the approximation vector `y` are passed to the OF. 'OutputSel' defaults to all components.
'**OutputTrace**'	Flag that forces stepwise proceeding of the output function [0 | {1}]. If 'OutputTrace' is set to 1, `sbvp` pauses after each mesh adaptation step and continues if a key is hit. If no output function is defined, 'OutputTrace' is ignored.

2.6 Zerofinder options

Options for the MATLAB zerofinder `fsolve`[2] can be set using the `optimset` command

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

Since it is possible to switch between `fsolve` and the Newton zerofinder (cf. 'ZfMethod' on page 7), options for the Newton zerofinder are set in the same way. Defining options for any zerofinder is thus achieved by

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

and `my_zfopt` is used to define the BVP options

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

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

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

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

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

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

2.7 Output functions

An output function (OF) is a MATLAB m-file that is used to monitor the mesh adaptation process. If an OF is specified, it will be passed the result of each adaptation step. The predefined OFs `sbvpplot`, `sbvpphas2` and `sbvpphas3` all display the solution approximation on the current mesh in order to illustrate the adaptation process graphically. `sbvpplot` simply draws the solution components versus t, `sbvpphas2` and `sbvpphas3` plot 2- and 3-dimensional phase portraits, respectively. Figure 1 shows three subsequent solution approximations of the boundary value problem defined in `demofile6` displayed by the OFs `sbvpplot` (left) and `sbvpphas2` (right).

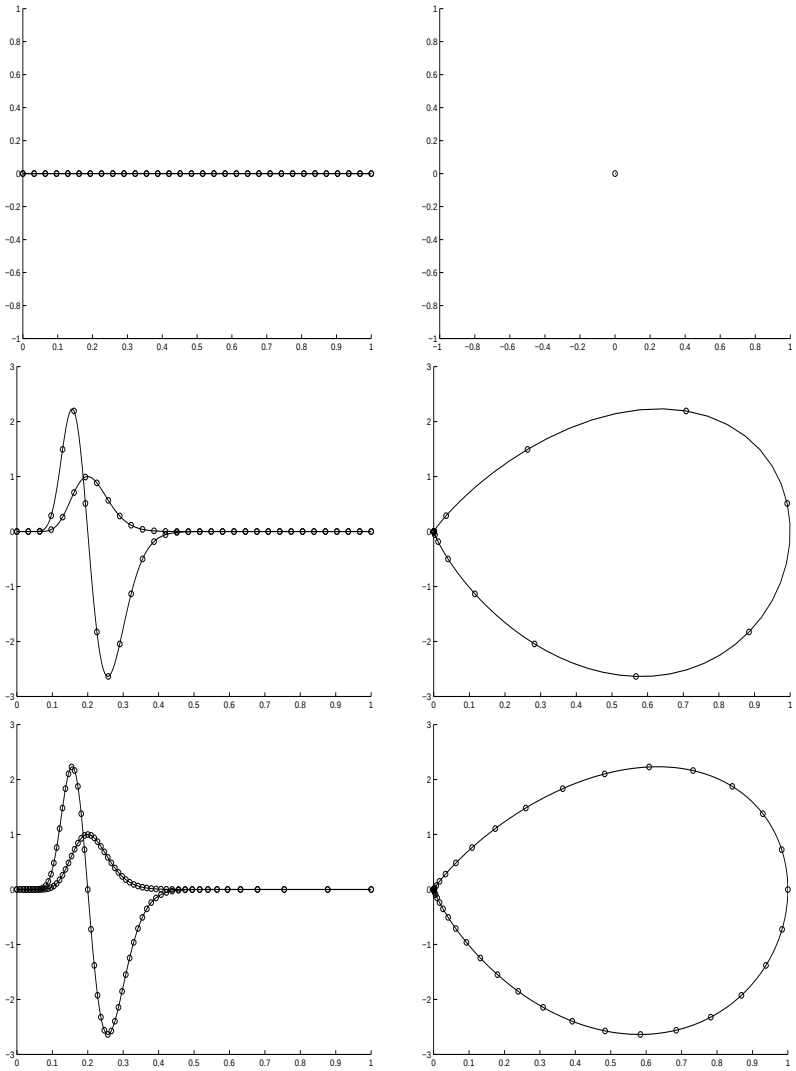

Figure 1: Initial guess and two subsequent solution approximations displayed by `sbvpplot` (left) and `sbvpphas2` (right)

2.8 Examples

We solve the 2-dimensional nonlinear BVP

$$\begin{aligned} x' &= y, \\ y' &= -\sin(x), \end{aligned}$$

$$\begin{aligned} x(0) &= 1, \\ y(0) + y(2) &= 0, \end{aligned}$$

that is defined in `demofile5`. The simplest `bvpfile` for this task is given by

```
function out=my_bvpfile(flag,t,y,ya,yb)
switch flag
case 'f'       % right-hand side of the differential equation
   out = [y(2) ; -sin(y(1))];
case 'df/dy'   % Jacobian of f
   out = [0 1; -cos(y(1)) 0];
case 'R'       % boundary conditions
   out=[ya(1)-1 ; ya(2) + yb(2)];
case 'dR/dya'  % Jacobian of the boundary conditions w.r.t. ya
   out=[1 0;0 1];
case 'dR/dyb'  % Jacobian of the boundary conditions w.r.t. yb
   out=[0 0;0 1];
case 'tau'     % initial mesh (integration interval only)
   out=[0 2];
otherwise
   error('unknown flag');
end
```

We will now illustrate how this specification can be modified step by step in order to improve the solution process. To accelerate the computations, we first vectorize the **f**- and **f**′-evaluation by replacing the corresponding entries by

```
case 'f'       % right-hand side of the differential equation (vectorized)
   out=[y(2,:) ; -sin(y(1,:))];

case 'df/dy'   % Jacobian of f (vectorized)
   out=zeros(2,2,length(t));  % allocate memory
   for i=1:length(t)
      out(:,:,i)=[0 1; -cos(y(1,i)) 0];
   end
```

We inform `sbvp` about the now vectorized `bvpfile` by adding the case

```
case 'bvpopt'  % solution options
   out = sbvpset('fVectorized',1,'JacVectorized',1);
```

As the boundary value problem is nonlinear, we need a zerofinding routine to compute the numerical solution. The options for the zerofinding routine can be adjusted as follows. We extend the previous `bvpfile` by adding

```
case 'bvpopt'  % solution options
   my_zfopt = optimset('Display','iter','TolX',1e-13);
        out = sbvpset('fVectorized',1,'JacVectorized',1,'ZfOpt',my_zfopt);
```

Moreover, we can provide an initial approximation of the solution, e.g.

```
case 'y0'      % initial approximation
   out=ones(2,length(t));
```

Note that the variable `t` contains the initial mesh chosen by `sbvp`.
Combining all these modifications, we obtain the `bvpfile`

```
function out=my_bvpfile(flag,t,y,ya,yb)
switch flag
case 'f'       % right-hand side of the differential equation (vectorized)
   out=[y(2,:) ; -sin(y(1,:))];
case 'df/dy'   % Jacobian of f (vectorized)
   out=zeros(2,2,length(t));  % allocate memory
   for i=1:length(t)
      out(:,:,i)=[0 1; -cos(y(1,i)) 0];
   end
case 'R'       % boundary conditions
   out=[ya(1)-1 ; ya(2) + yb(2)];
case 'dR/dya'  % Jacobian of the boundary conditions w.r.t. ya
   out=[1 0;0 1];
case 'dR/dyb'  % Jacobian of the boundary conditions w.r.t. yb
   out=[0 0;0 1];
case 'tau'     % initial mesh (integration interval only)
   out=[0 2];
case 'y0'      % initial approximation
   out=ones(2,length(t));
case 'bvpopt'  % solution options
   my_zfopt = optimset('Display','iter','TolX',1e-13);
        out = sbvpset('fVectorized',1,'JacVectorized',1,'ZfOpt',my_zfopt);
otherwise
   error('unknown flag');
end
```

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

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

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

```
>> tau0 = linspace(0,2,51);            % define an initial mesh
>>
>> [tau,y] = sbvp(@my_bvpfile,tau0);   % sbvp gets y0 and bvpopt
                                       % from my_bvpfile
```

In order to force `sbvp` to use a collocation degree equal to 6, yet keep all the options defined in the bvpfile, proceed in this way:

```
>>oldopt = my_bvpfile('bvpopt');       % get old option struct
>>newopt = sbvpset(oldopt,'DegreeSelect','manual','Degree',6);
>>
>>[tau,y] = sbvp(@my_bvpfile,'','',newopt);   % sbvp obtains tau0 and y0
                                               % from my_bvpfile
```

Additional examples of `bvpfile`s included as demo-files in the SBVP-package can be found on the following pages.

```
function out=demofile1(flag,t,y,ya,yb)
%  DEMOFILE1  demonstrates the usage of SBVP
%
%  We try to solve the linear BVP
%    y'=y , y(0)+y(2)=1+exp(2)
%
%  Solution syntax:
%    [tau,y] = sbvp(@demofile1);
%    plot(tau,y,'.-');               % plot solution

switch flag
case 'f'       % right-hand side of the differential equation
   out=y;
case 'df/dy'   % Jacobian of f
   out=1;
case 'R'       % boundary condition
   out=ya+yb-(1+exp(2));
case 'dR/dya'  % Jacobian of the boundary condition w.r.t. ya
   out=1;
case 'dR/dyb'  % Jacobian of the boundary condition w.r.t. yb
   out=1;
case 'tau'     % initial mesh (= beginning and end of the integration interval)
   out=[0 2];
otherwise
   error('unknown flag');
end
```

```
function out=demofile2(flag,t,y,ya,yb,lambda)
%  DEMOFILE2  demonstrates the usage of SBVP
%
%  We try to solve the linear BVP
%    y'=lambda * y , y(0)+y(2)=4
%  using a parameter lambda, that is passed as last argument.
%
%  Solution syntax:
%    my_lambda = 3;
%    [tau,y] = sbvp(@demofile2,'','','',my_lambda);
%                           % determine mesh such that tolerances are satisfied
%                           % starting with the initial mesh defined in DEMOFILE2

switch flag
case 'f'       % right-hand side of the differential equation
   out=lambda * y;
case 'df/dy'   % Jacobian of f
   out=lambda;
case 'R'       % boundary condition
   out=ya+yb-4;
case 'dR/dya'  % Jacobian of the boundary condition w.r.t. ya
   out=1;
case 'dR/dyb'  % Jacobian of the boundary condition w.r.t. yb
   out=1;
case 'tau'     % (initial) mesh
   out=linspace(0,2,51);
otherwise
   error('unknown flag');
end
```

```
function out=demofile3(flag,t,y,ya,yb)
%  DEMOFILE3  demonstrates the usage of SBVP
%
%  We try to solve the linear BVP
%    y'=y , y(0)+y(2)=1+exp(2)
%
%  using the Runge-Kutta basis of degree 6 on Gaussian
%  collocation points. Note the BVPOPT-case.
%
%  Solution syntax:
%    [tau,y] = sbvp(@demofile3);

switch flag
case 'f'       % right-hand side of the differential equation
   out=y;
case 'df/dy'   % Jacobian of f
   out=1;
case 'R'       % boundary condition
   out=ya+yb-(1+exp(2));
case 'dR/dya'  % Jacobian of the boundary condition w.r.t. ya
   out=1;
case 'dR/dyb'  % Jacobian of the boundary condition w.r.t. yb
   out=1;
case 'tau'     % mesh
   out=[0 2];
case 'bvpopt'  % solution options
   out=sbvpset('ColPts','Gauss','Basis','RungeKutta','DegreeSelect','Manual','Degree',6);
otherwise
   error('unknown flag');
end
```

```
function out=demofile4(flag,t,y,ya,yb)
%  DEMOFILE4  demonstrates the usage of SBVP
%
%  We try to solve the 2-dimensional linear BVP
%    x' = y
%    y' =-x,
%
%  x(0)     = 1
%  y(0)+y(2)= 0
%
%  using vectorized f-evaluation.
%
%  Solution syntax:
%    [tau,y] = sbvp(@demofile4);

switch flag
case 'f'
%  out=[y(2) ; -y(1)]    would be the non-vectorized version
   out=[y(2,:) ; -y(1,:)];
case 'df/dy'   % Jacobian of f
   out=[0 1;-1 0];
case 'R'       % boundary conditions
   out=[ya(1)-1 ; ya(2) + yb(2)];
case 'dR/dya'  % Jacobian of the boundary conditions w.r.t. ya
   out=[1 0;0 1];
case 'dR/dyb'  % Jacobian of the boundary conditions w.r.t. yb
   out=[0 0;0 1];
case 'tau'     % mesh
   out=[0 2];
case 'bvpopt'  % solution options
   out=sbvpset('fvectorized',1);
otherwise
   error('unknown flag');
end
```

```
function out=demofile5(flag,t,y,ya,yb)
%  DEMOFILE5 demonstrates the usage of SBVP
%
%  We try to solve the 2-dimensional nonlinear BVP
%    x' = y
%    y' =-sin(x),
%
%  x(0)     = 1
%  y(0)+y(2)= 0
%
%  using vectorized f- and f'-evaluation.
%  We check the user-supplied Jacobians using the
%  'CheckJac' option, and we display information
%  about the zerofinding process using the 'Display'
%  option. Furthermore we define tolerances for
%  the zerofinder and an upper limit to the number
%  of function evaluations (discretization residual
%  function F(x)). Note that we also provide an
%  initial approximation for the solution.
%
%  Solution syntax:
%    [tau,y] = sbvp(@demofile5);

switch flag
case 'f'       % right-hand side of the differential equation (vectorized)
   out=[y(2,:) ; -sin(y(1,:))];
case 'df/dy'   % Jacobian of f (vectorized)
   out=zeros(2,2,length(t)); % allocate memory
   for i=1:length(t)
      out(:,:,i)=[0 1; -cos(y(1,i)) 0];  % out(:,:,i) = df/dy(ti,yi)
   end
case 'R'       % boundary conditions
   out=[ya(1)-1 ; ya(2) + yb(2)];
case 'dR/dya'  % Jacobian of the boundary conditions w.r.t. ya
   out=[1 0;0 1];
case 'dR/dyb'  % Jacobian of the boundary conditions w.r.t. yb
   out=[0 0;0 1];
case 'tau'     % mesh
   out=[0 2];
case 'y0'      % initial approximation
   out=ones(2,length(t));
case 'bvpopt'  % solution options
   my_zfopt = optimset('Display','iter','TolX',1e-13,'MaxFunEvals',10);
   out=sbvpset('fVectorized',1,'JacVectorized',1,'CheckJac',1,'ZfOpt',my_zfopt);
otherwise
   error('unknown flag');
end
```

The next problem is the linear singular boundary value problem

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

where the parameters are $\mu = 1,\ \alpha = 80$ and $k = 16$, and $c = \left(\frac{\alpha}{k}\right)^k \mathrm{e}^k$.

```
function out=demofile6(flag,t,y,ya,yb)
%  DEMOFILE6  demonstrates the usage of SBVP
%
%  We try to solve a 2-dimensional linear singular BVP
%  at a very high accuracy level (small RelTol and AbsTol).
%
%  The approximation of the solution after each mesh adaptation
%  step is displayed by the output function SBVPLOT.
%  Try SBVPPHAS2 as well.
%
%  Solution syntax:
%    [tau,y] = sbvp(@demofile6);

% some parameters

mu=1;
p1=0.2;
p2=0.05;
k=(p1/p2)^2;     % k=16
alpha=p1/(p2^2); % alpha=80
c=((alpha/k)^k)*exp(k);
mu2 = mu^2;
alpha2 = alpha^2;

switch flag
case 'f'       % right-hand side of the diff. eq. (inhomogeneity only, vectorized)
   out=[zeros(1,length(t)) ; c*t.^(k-1).*exp(-alpha*t).*(k^2-mu2-alpha*t*(1+2*k))];
case 'df/dy'   % Jacobian of f (vectorized differently than in DEMOFILE5)
   out=zeros(2,length(t));  % allocate memory
   for i=1:length(t)
      out(:,[2*i-1 2*i]) = 1/t(i) * [0 1; mu2+alpha2*t(i)^2 0];
   end
case 'R'       % boundary conditions (negative inhomogeneity only)
   out=-[0; c*exp(-alpha)];
case 'dR/dya'  % Jacobian of the boundary conditions w.r.t. ya
   out=[0 1; 0 0];
case 'dR/dyb'  % Jacobian of the boundary conditions w.r.t. yb
   out=[0 0; 1 0];
case 'tau'     % mesh
   out=[0 1];
case 'bvpopt'  % solution options
   out=sbvpset('RelTol',1e-9,'AbsTol',1e-9,'JacVectorized',1,'fVectorized',1,...
               'IntMaxMinRatio',20,'OutputFcn',@sbvpplot);
otherwise
   error('unknown flag');
end
```

2.9 Hints for Troubleshooting

If you encounter troubles solving your own problems, here is a checklist you might want to follow.

- Check if the problem was formulated correctly, i.e. whether the `bvpfile` returns correct outputs.
- If you do not find any errors, solve the problem with 'CheckJac' set to 1 (cf. §2.5). The analytic derivatives you supplied will be compared with a finite difference approximation of these derivatives.
- Furthermore, if the problem is nonlinear, you might get a better understanding of what is going wrong if you set the zerofinder Display to 'iter' (cf. §2.6). This provides some information about the zerofinding process.
- Try to find a better initial approximation.
- Try the alternative zerofinder (cf. §2.5).
- Check if your problem is well-posed, i.e. if there exists a solution of the analytic problem, that is (locally) unique and depends *continuously* on the problem parameters. See [6] for a discussion of necessary and sufficient conditions for the well-posedness in the singular case.

References

[1] W. Auzinger, G. Kneisl, O. Koch, and E. Weinmüller, *A collocation code for boundary value problems in ordinary differential equations*. Submitted to Numer. Algorithms. Also available as ANUM Preprint Nr. 18/01 at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.

[2] ——, *A solution routine for singular boundary value problems*, Techn. Rep. ANUM Preprint Nr. 1/02, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology, Austria, 2002. Available at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.

[3] W. Auzinger, O. Koch, W. Polster, and E. Weinmüller, *Ein Algorithmus zur Gittersteuerung bei Kollokationsverfahren für singuläre Randwertprobleme*, Techn. Rep. ANUM Preprint Nr. 21/01, Inst. for Appl. Math. and Numer. Anal., Vienna Univ. of Technology, Austria, 2001. Available at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.

[4] W. Auzinger, O. Koch, and E. Weinmüller, *Efficient collocation schemes for singular boundary value problems*. To appear in Numer. Algorithms. Also available as ANUM Preprint Nr. 5/01 at `http://www.math.tuwien.ac.at/~inst115/preprints.htm`.

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

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

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

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

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

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