Chapter 9

HAM-Based Mathematica Package BVPh 2.0 for Nonlinear Boundary Value Problems

Yinlong Zhao* and Shijun Liao†

School of Naval Architecture, Ocean and Civil Engineering
Shanghai Jiao Tong University, Shanghai 200240, China
**zhaoyl@sjtu.edu.cn, †sjliao@sjtu.edu.cn*

We issue the new version `BVPh 2.0` of the Mathematica package `BVPh`, a free software based on the homotopy analysis method (HAM) for nonlinear boundary-value and eigenvalue problems. The aim of the package `BVPh` is to develop a kind of analytic tool for as many nonlinear boundary value problems (BVPs) as possible such that multiple solutions of highly nonlinear BVPs can be conveniently found out, and that the infinite interval and singularities of governing equations and/or boundary conditions at multi-points can be easily resolved. Unlike its previous versions, `BVPh 2.0` works for systems of coupled nonlinear ordinary differential equations. It is user-friendly and free available online (`http://numericaltank.sjtu.edu.cn/BVPh.htm`). Different from numerical packages (such as `BVP4c`), it is based on the idea "computing numerically with functions instead of numbers". Especially, unlike other packages, the convergence of results given by the `BVPh 2.0` is guaranteed by means of the so-called convergence-control parameter in the frame of the homotopy analysis method. In this chapter, we briefly describe how to install and use the `BVPh 2.0` with a simple user's guide. Five typical examples (governed by up to four coupled ODEs) are used to illustrate the validity of the `BVPh 2.0`, and the corresponding input data of these examples for the `BVPh 2.0` are free available online (`http://numericaltank.sjtu.edu.cn/BVPh.htm`). The `BVPh 2.0` indeed provides us with an easy-to-use tool to efficiently solve various types of coupled linear/nonlinear ordinary differential equations.

Contents

9.1. Introduction

Ordinary differential equations (ODEs) are widely used in mathematics, science and engineering. The so-called initial value problems (IVPs) specify some restrictions only at a single point. This kind of problems is often solved by means of numerical approach based on integration, such as the Runge–Kutta method. However, in many cases, a solution is described in a more complicated way. The so-called boundary value problems (BVPs)

may specify some restrictions at more than one point. Unlike IVPs, BVPs might have multiple solutions, or may satisfy some restrictions at infinity. Hence, generally speaking, BVPs are more difficult to solve than IVPs.

Thanks to the development of mathematical software such as Maple, Mathematica and MATLAB, some packages are developed to solve nonlinear BVPs, such as the `BVP4c`, the `Chebfun`, the `NOPH`, and the `BVPh`. The famous MATLAB package `BVP4c` [1, 2] implements a collocation method, instead of a shooting method. The collocation polynomial provides a C^1-continuous solution that is fourth-order accurate uniformly in $[a, b]$. Mesh selection and error control are based on the residual of the continuous solution. However, it is not easy for the `BVP4c` to resolve the singularity in governing equations and/or boundary conditions. Besides, the `BVP4c` regards an infinite interval as a kind of singularity and replaces it by a finite one: this leads to the additional inaccuracy and uncertainty of solutions.

The `Chebfun` [3, 4] is a collection of algorithms on the "chebfun" objects written in MATLAB. It aims to combine the feel of symbolics with the speed of numerics. The basis of the `Chebfun` is Chebyshev expansions, fast Fouries transform, baryentric interpolation and so on. The idea "computing numerically with functions instead of numbers" [4] behind the `Chebfun` makes it have the potential to handle unbounded domains and singularities in an easy way. Although linear differential equations can be solved in a single step by `Chebfun` without iteration, only a few examples for nonlinear differential equations are given [3, 4]. Actually, `Chebfun` uses Newton's iteration to solve nonlinear problems. However, it is well known that the convergence of Newton's iteration is strongly dependent upon initial guesses and thus is not guaranteed. Besides, `Chebfun` searches for multiple solutions of nonlinear differential equations by using different guess approximations. However, it is not very clear how to choose these different guess approximations.

Based on the homotopy in topology, the so-called homotopy analysis method (HAM) was proposed by Liao [8–14] to gain analytic approximations of highly nonlinear problems. The HAM has some advantages over other traditional analytic approximation methods. First, unlike perturbation techniques, the HAM is independent of small/large physical parameters, and thus is valid in more general cases. Besides, different from all

other analytic techniques, the HAM provides us a convenient way to guarantee the convergence of series solution. Furthermore, the HAM provides extremely large freedom to choose initial guess, equation-type and solution expression of linear sub-problems. It is found [13, 14] that lots of nonlinear BVPs in science, engineering and finance can be solved conveniently by means of the HAM, no matter whether the interval is finite or not.

Since the early HAM was proposed by Liao [9] in 1992, the HAM has been developed greatly in theory and widely applied to numerous nonlinear problems in lots of fields, with hundreds of publications. So, it is necessary to develop a HAM-based software to simplify the applications of the HAM. Based on the HAM, a Mathematica package `BVPh` 1.0— for nonlinear boundary value/eigenvalue problems with singularity and/or multipoint boundary conditions was issued by Liao [14] in May 2012, which is free available online (`http://numericaltank.sjtu.edu.cn/BVPh.htm`). Its aim is to develop a kind of analytic tool for as many nonlinear BVPs as possible such that multiple solutions of highly nonlinear BVPs can be conveniently found out, and that the infinite interval and singularities of governing equations and/or boundary conditions at multi-points can be easily resolved. As illustrated by Liao [14], the `BVPh 1.0` is valid for lots of nonlinear BVPs and thus is a useful tool in practice.

Currently, based on the HAM, the Maple package `NOPH` [5] for periodically oscillating systems of center and limit cycle types is developed, which delivers accurate approximations of frequency, mean of motion and amplitude of oscillation automatically. The `NOPH` combines Wu's elimination method and the homotopy analysis method (HAM). It is free available online (`http://numericaltank.sjtu.edu.cn/NOPH.htm`). Different from the `BVPh 1.0`, the `NOPH` is for periodic oscillations. This illustrates the general validity of the HAM for nonlinear problems once again.

It is a pity that the `BVPh 1.0` can only deal with BVPs of single ordinary differential equation (ODE), say, it can not solve systems of coupled ODEs. In this chapter we issue the new version `BVPh 2.0`, which works for many types of systems of coupled nonlinear ordinary differential equations (ODEs) in finite and/or semi-infinite intervals. Besides, new algorithms are used in some modules of `BVPh 2.0`. As a result, `BVPh 2.0` is much faster than `BVPh 1.0` in most cases. In this chapter, we illustrate how to use

`BVPh 2.0` to solve different kinds of systems of coupled nonlinear ODEs, including a system of two coupled ODEs in finite interval, a system of two coupled ODEs in semi-infinite interval, a system of two coupled ODEs with algebraic property at infinity, a system of three coupled ODEs with an unknown parameter to be determined, and a system of four coupled ODEs in different intervals.

This chapter is organized as follows. In § 9.2, we show how to install the package `BVPh 2.0`. In § 9.3, we take an example to illustrate how to use `BVPh 2.0` in detail. In § 9.4, we briefly describe some mathematical formulas based on which the `BVPh 2.0` is developed. In § 9.5, the iterative technique is illustrated for two typical kinds of base-functions in finite intervals. A simple user guide of `BVPh 2.0` is given in § 9.6. Some typical examples are given in § 9.7 to illustrate the potential of the `BVPh 2.0` and to show its validity. In § 9.8, some discussions are given. The reader is suggested to read § 9.3 at first for an illustrative example. If one is puzzled with some parameters, one is encouraged to search for them in § 9.6, and read the detailed description there.

In order to check the validity and correctness of the `BVPh 2.0`, we choose all of our typical examples from published articles, which were solved either analytically (by the HAM) or numerically before. As illustrated in this chapter, the package `BVPh 2.0` always gives results, which agree well with the published ones. The validity of the `BVPh 2.0` is checked for each example in the following way:

(1) the squared residual error usually decreases to as low as 10^{-10}, and decreases at least 6 orders of magnitude;
(2) the same physical quantities of interest are gained as the published ones;
(3) analytic solutions gained by `BVPh 2.0` agree well with the published ones.

Note that the package `BVPh 2.0` is developed with Mathematica 7.0. As some new features of Mathematica 7.0 are used, we strongly recommend you to use the `BVPh 2.0` in Mathematica 7.0 or higher version, since it might not work for some lower version of Mathematics.

9.2. Installation of the BVPh 2.0

The BVPh is a free/open-source software written in Mathematica for boundary value problems (BVPs). Its newest version BVPh 2.0 is available online (http://numericaltank.sjtu.edu.cn/BVPh.htm). The input data for this chapter's examples can also be found there. Since the commands of Mathematica are designed to be the same on different operating systems, the package written in Mathematica on Windows can be used in Mathematica on other operating systems.

The source file of the package BVPh 2.0 is BVPh2_0.m. The easiest way to load the package BVPh 2.0 to solve your problem is to put the file BVPh2_0.m and the input data for the problem, e.g., Example.m, in the same directory, then open a new notebook file and saved it as, e.g., runExample.nb, in the same directory and run the following codes.

```
(* Filename: runExample.nb *)
ClearAll["Global`*"];
SetDirectory[ToFileName[Extract["FileName" /.
NotebookInformation[EvaluationNotebook[]], {1},
FrontEnd`FileName]]];
<<BVPh2_0.m;
<<Example.m;
```

Note that the listings without an end-of-line semicolon is wrapped to fit the page width of this chapter. However, if you break the long command intentionally in Mathematica, it will not work as you expected. The above commands first clear all global variables, then set the current working directory to "the current directory". Here "the current directory" is where you put the source file BVPh2_0.m, the input data Example.m and the notebook file runExample.nb. The last two lines read in the package BVPh2_0.m and Example.m in the notebbook runExample.nb.

If you are familiar with Mathematica's file and directory operations, you can put the file BVPh2_0.m and the input data of the problem in different directories, then specifies the path where to get them. It is worth emphasizing that the pathname separator is "\\" under Windows, and "/" elsewhere. The sample codes to get the file BVPh2_0.m and the input data

`Example.m` in the notebook file `runExample.nb` are as follows on Windows.

```
(* Filename: runExample.nb *)
ClearAll["Global`*"];
<<"E:\\Package\\BVPh2_0.m";
<<"E:\\Project\\Example\\Example.m";
```

Here we assume the file `BVPh2_0.m` in the directory `E:\Package` and the input data `Example.m` in a different directory `E:\Project\Example`. The above sample codes may look like the following on Unix

```
(* Filename: runExample.nb *)
ClearAll["Global`*"];
<< "/home/user/Package/BVPh2_0.m";
<< "/home/user/Example/Example.m";
```

Here we assume `BVPh2_0.m` in the directory `/home/user/Package/` and the input data `Example.m` in a different directory `/home/user/Example`.

From now on, we will assume that the package `BVPh 2.0` has been successfully loaded so that the modules in the package are available. In the next section, we will use an illustrative example to show how to write the input data and how to get the approximations by `BVPh 2.0`.

9.3. Illustrative example

Consider a system of ODEs [6]

$$f''' - (f')^2 + ff'' + 2\lambda g + \beta[2ff'f'' - f^2 f'''] = 0, \tag{9.1}$$

$$g'' - f'g + fg' - 2\lambda f' + \beta[2ff'g' - f^2 g''] = 0, \tag{9.2}$$

subject to

$$f'(0) = 1, f(0) = 0, g(0) = 0, \tag{9.3}$$

$$f'(\infty) = 0, g(\infty) = 0, \tag{9.4}$$

where λ is rotation parameter, β is viscoelastic parameter, and the prime indicates the differentiation with respect to η. This system models two-dimensional flow of an upper convected Maxwell fluid in a rotating frame. Sajid [6] has solved it by the HAM.

To solve this problem by BVPh 2.0, we have to input the differential equations, boundary conditions, initial guesses and convergence-control parameters. The differential equations (9.1) and (9.2) can be coded as follows

```
TypeEQ = 1;
NumEQ = 2;
f[1,z_,{f_,g_},Lambda_]:=
    D[f,{z,3}]-D[f,z]^2+f*D[f,{z,2}]+2*la*g+
    beta*(2*f*D[f, z]*D[f,{z,2}]-f^2*D[f,{z,3}]);
f[2,z_,{f_,g_},Lambda_]:=
    D[g,{z,2}]-D[f,z]*g+f*D[g,z]-2*la*D[f,z]+
    beta*(2*f*D[f, z]*D[g, z]-f^2*D[g,{z, 2}]);
```

Here TypeEQ controls the type of governing equations: TypeEQ=1 corresponds to a system of ODEs without an unknown to be determined, TypeEQ=2 corresponds to a system of ODEs with an unknown, Lambda, to be determined. Since all the parameters in the problem will be given, we set TypeEQ to 1. Note that we use the delayed assignment SetDelayed(:=) in Mathematica to define these ODEs to avoid the evaluation when the assignment is made.

The boundary conditions (9.3) and (9.4) are defined in a semi-infinite interval, from 0 to $+\infty$. They are coded as

```
NumBC = 5;
BC[1,z_,{f_,g_}]:=(D[f, z]-1)/.z->0;
BC[2,z_,{f_,g_}]:=f/.z->0;
BC[3,z_,{f_,g_}]:=g/.z->0;
BC[4,z_,{f_,g_}]:=D[f,z]/.z->infinity;
BC[5,z_,{f_,g_}]:=g/.z->infinity;
```

Here NumBC is the number of boundary conditions of the problem. For this problem, we have 5 boundary conditions, so NumBC is set to 5. The symbol infinity is introduced in our package to denote ∞. When an expression contains infinity, the limit of the expression is computed as z approaches ∞. The delayed assignment (:=) is also used to avoid the evaluation when the assignment is made — the same reason as defining the differential equations.

For a multi-layer problem, the differential equations in the system are not necessarily in the same interval (see Example 4 in § 9.7). Hence, we have to give each equation its solution interval. To measure the accuracy of the approximate solutions, we have to compute the squared residual error over the corresponding solution interval. In practice, when the differential equation is defined in a semi-infinite interval, we simply truncate the infinite interval to a finite interval to compute the squared residual error, or it will take a lot of computation time. For this problem, the solution interval for each equation and the integral interval for the squared residual error are defined as

```
zL[1] = 0;
zR[1] = infinity;
zL[2] = 0;
zR[2] = infinity;
zRintegral[1] = 10;
zRintegral[2] = 10;
```

Here `zL[k]` (or `zR[k]`) is the left (or right) endpoint of the solution interval for the kth equation `f[k,z,{f,g},Lambda]`. And `zLintegral[k]` (or `zRintegral[k]`) is the left (or right) endpoint of the integral interval to compute the squared residual error for the kth equation. If the value of `zL[k]` (or `zR[k]`) is a finite number, `zLintegral[k]` (or `zRintegral[k]`) is set to the same value automatically. However, if any of them contains the symbol `infinity`, we have to set the corresponding endpoint of the integral interval to a finite value. That is why we write explicitly `zRintegral[1]=10` and `zRintegral[2]=10`. For this problem, the squared residual is integrated over the range [0, 10] for both equations.

The auxiliary linear operators for this problem are chosen as $\mathcal{L}_1 = \frac{\partial^3}{\partial \eta^3} - \frac{\partial}{\partial \eta}$, $\mathcal{L}_2 = \frac{\partial^2}{\partial \eta^2} - 1$, which are coded as

```
L[1,u_] := D[u,{z,3}]-D[u,z];
L[2,u_] := D[u,{z,2}]-u;
```

Here `L[k,u]` is the auxiliary linear operator corresponding to the kth equation. Note that i) η is the independent variable in the differential equations

(9.1) and (9.2), while z is the universal independent variable in the package `BVPh 2.0`; ii) the delayed assignment `SetDelayed(:=)` is used to define the operator; iii) u is a formal parameter.

For this problem, the initial guesses are $f_0 = 1 - e^{-z}$ and $g_0 = ze^{-z}$. They are coded as

```
U[1,0] = 1-Exp[-z];
U[2,0] = alpha*z*Exp[-z];
```

Here `alpha` is an introduced convergence-control parameter that will be determined later. `U[k,0]` is the initial guess of the kth equation. Note that `U[k,0]` and `u[k,0]` are usually the same in the package `BVPh 2.0`.

We want to solve this problem when the physical parameters $\beta = 1/5$ and $\lambda = 1/10$. These two parameters are coded as

```
beta = 1/5;
la = 1/10;
```

So far, we have defined all the input of this problem properly, except the convergence-control parameter `c0[k]` and `alpha`. Usually, the optimal values of the convergence-control parameters are obtained by minimizing the averaged squared residual error. For this problem, we get the approximate optimal values of `c0[1]`, `c0[2]` and `alpha` by minimizing the squared residual error of the 3rd-order approximation as

```
GetOptiVar[3, {}, {c0[1],c0[2],alpha}];
```

The first parameter of `GetOptiVar` denotes which order approximation is used. Here 3 means the 3rd-order approximation is used. The second parameter denotes a list of constraints used in the optimization. When the second parameter of `GetOptiVar` is an empty list, it means the averaged squared residual is minimized without any constraint. Here we add no constraints to minimize the averaged squared residual error. The third parameter is a list of the variables to be optimized. Here we want to optimize `c0[1]`, `c0[2]` and `alpha`. After some computation, it gives the optimized convergence-control parameters `c0[1]=-1.26906`, `c0[2]=-1.19418`

and `alpha=-0.0657063`.

Now we can use

```
BVPh[1,10]
```

to get the 10th-order approximation. If we are not satisfied with the accuracy of the 10th-order approximation, we can use `BVPh[11,20]`, instead of `BVPh[1,20]`, to get 20th-order approximation or higher order approximation.

The kth-order approximation of the ith differential equation is stored in `U[i,k]`. We can use

```
Plot[{U[1,20], U[2,20]}, {z,0,10},
AxesLabel->{"\[Eta]", ""},
PlotStyle->{{Thin, Red},{Dashed, Blue}}]
```

to plot the 20th-order approximate solution, which is shown in Fig. 9.1.

The accuracy of the kth-order approximation is measured by the averaged squared residual. We can use

```
ListLogPlot[Table[{2*i,ErrTotal[2*i]},{i,1,10}],
Joined->True,Mesh->All,
PlotRange->{{2,20},{10^(-15),10^(-5)}},
AxesLabel->{"m", "error"}];
```

to plot the curve of the total error versus the order of approximation, which is shown in Fig. 9.2. Note that `ErrTotal[k]` stores the total error of the system when the kth-order approximation is used, while `Err[k]` is a list that stores the error for each ODE in the system when the kth-order approximation is used.

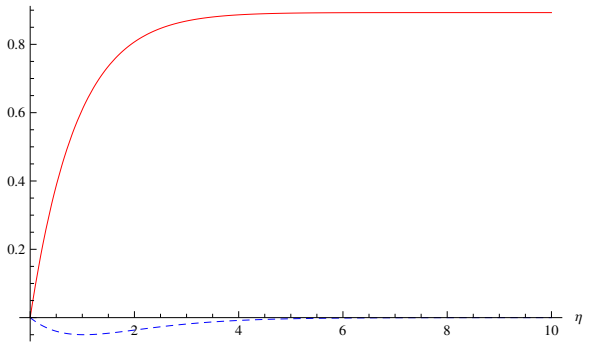

Fig. 9.1. The curve of $f(z)$ (solid) and $g(z)$ (dashed) for the illustrative example when $\beta = 1/5$, $\lambda = 1/10$.

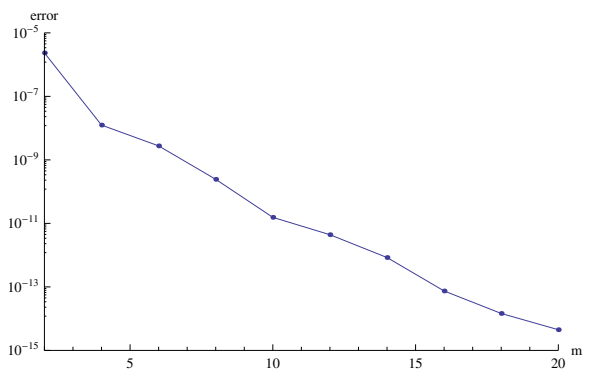

Fig. 9.2. Total error vs. order of approximation for the illustrative example when $\beta = 1/5$, $\lambda = 1/10$.

9.4. Brief mathematical formulas

The `BVPh 2.0` is based on the HAM. It is an extension of `BVPh 1.0` to systems of ODEs. Here the mathematical formulas are briefly described.

9.4.1. *Boundary value problems in a finite interval*

Consider a system of `NumEQ` ordinary differential equations (ODEs),

$$\mathcal{F}_i[z, u_1, u_2, \cdots] = 0,\ z \in [\mathtt{zL_i}, \mathtt{zR_i}],\ 1 \leqslant i \leqslant \mathtt{NumEQ}, \tag{9.5}$$

subject to the `NumBC` linear boundary conditions

$$\mathcal{B}_k[z, u_1, u_2, \cdots] = \gamma_k,\ 1 \leqslant k \leqslant \texttt{NumBC}, \tag{9.6}$$

where $\mathcal{F}_i$ is a nonlinear differential operator, $\mathcal{B}_k$ is a linear differential operator, z is the independent variable, $u_i(z)$ is a smooth function, `NumEQ` is a positive integer, γ_k, $\texttt{zL}_\texttt{i}$ and $\texttt{zR}_\texttt{i}$ are constants, respectively. The boundary conditions may be defined at multipoints including the two endpoints. Assume that at least one solution exists, and that all solutions are smooth.

Let $q \in [0, 1]$ denote an embedding parameter, $u_{i,0}(z)$ an initial guess of the solution $u_i(z)$, respectively. In the frame of the HAM, we construct such a continuous deformation $\phi_i(z; q)$ that, as q increases from 0 to 1, $\phi_i(z; q)$ varies continuously from the initial guess $u_{i,0}(z)$ to the true solution $u_i(z)$ of (9.5) and (9.6). Such kind of continuous deformations are governed by the so-called zeroth-order deformation equations

$$\begin{aligned}(1-q)\mathcal{L}_i[\phi_i(z;q) - u_{i,0}(z)] = qc_{0,i}H_i(z)\mathcal{F}_i[\phi_1(z;q), \phi_2(z;q), \cdots],\\ z \in [\texttt{zL}_\texttt{i}, \texttt{zR}_\texttt{i}],\ q \in [0,1],\ 1 \leqslant i \leqslant \texttt{NumEQ}\end{aligned} \tag{9.7}$$

where $\mathcal{L}_i$ is an auxiliary linear operator, $c_{0,i}$ is the so-called convergence-control parameter, $H_i(z)$ is an auxiliary function, corresponding to the ith governing equation in (9.5), respectively.

Note that the HAM provides us extremely large freedom to choose the auxiliary linear operator $\mathcal{L}_i$, the convergence-control parameter $c_{0,i}$ and the auxiliary function $H_i(z)$, $1 \leqslant i \leqslant \texttt{NumEQ}$. Assume that all of them are properly chosen so that the homotopy-Maclaurin series

$$\phi_i(z; q) = u_{i,0}(z) + \sum_{k=1}^{+\infty} u_{i,k}(x) q^k \tag{9.8}$$

absolutely converges at $q = 1$, where

$$u_{i,m} = \mathcal{D}_m[\phi(z; q)] = \frac{1}{m!} \frac{\partial^m \phi(z; q)}{\partial q^m}\bigg|_{q=0}. \tag{9.9}$$

Here, $\mathcal{D}_m$ is called the mth-order homotopy-derivative operator [8]. Then, we have the so-called homotopy-series solution

$$u_i(z) = u_{i,0} + \sum_{m=1}^{+\infty} u_{i,m}(z), \tag{9.10}$$

where $u_{i,m}$ is governed by the so-called mth-order deformation equation

$$\mathcal{L}_i[u_{i,m}(z) - \chi_m u_{i,m-1}(z)] = c_{0,i} H_i(z) \delta_{i,m-1}(z),\ z \in [\texttt{zL}_\texttt{i}, \texttt{zR}_\texttt{i}], \quad (9.11)$$

subject to the `NumBC` homogeneous linear boundary conditions

$$\mathcal{B}_k[z, u_1, u_2, \cdots] = 0,\ 1 \leqslant k \leqslant \texttt{NumBC}, \quad (9.12)$$

where

$$\chi_m = \begin{cases} 0, & m \leq 1, \\ 1, & m > 1, \end{cases} \quad (9.13)$$

and

$$\begin{aligned} \delta_{i,m-1}(z) &= \mathcal{D}_{m-1}\{\mathcal{F}_i[\phi_1(z;q), \phi_2(z;q), \cdots]\} \\ &= \mathcal{D}_{m-1}\left\{\mathcal{F}_i\left[\sum_{i=0}^{m-1} u_{1,i}(z) q^i, \sum_{i=0}^{m-1} u_{2,i}(x) q^i, \cdots\right]\right\} \end{aligned} \quad (9.14)$$

can be easily obtained by means of the theorems proved in Ref. 8.

The mth-order approximation of $u_i(z)$ is given by

$$u_i(x) \approx U_{i,m}(x) = \sum_{k=0}^{m} u_{i,k}(z). \quad (9.15)$$

To measure the accuracy of the mth-order approximations $U_{i,m}$, $1 \leqslant i \leqslant$ `NumEQ`, the averaged squared residual error for the system (9.5) is defined as

$$E_m = \sum_{i=1}^{\texttt{NumEQ}} \frac{\int_{\texttt{zLintegral[i]}}^{\texttt{zRintegral[i]}} |\mathcal{F}_i[U_{1,m}, U_{2,m}, \cdots]|^2 \, dz}{\texttt{zRintegral[i]} - \texttt{zLintegral[i]}}. \quad (9.16)$$

Here `zLintegral[i]` and `zRintegral[i]` are two endpoints of the integral interval over which the squared residual of the ith governing equation is integrated. Theoretically speaking, the smaller E_m, the more accurate the mth-order approximation $U_{i,m}(z)$. Given the initial guess $u_{i,0}(z)$, the auxiliary linear operator $\mathcal{L}_i$ and the auxiliary function $H_i(z)$, the squared residual error E_m is dependent on the convergence-control parameters $c_{0,i}$, $i = 1, 2, \cdots,$ `NumEQ`. Hence, the optimal values of $c_{0,i}$ can be determined by the minimum of E_m at some proper order m.

Note that i) for boundary value problems without an unknown to be determined, we should set the control parameter `TypeEQ = 1` for the `BVPh` 2.0; ii) the intervals $[\texttt{zL}_\texttt{i}, \texttt{zR}_\texttt{i}]$ are not necessarily the same, so the package can solve multi-layer problems without any modification.

9.4.2. *Eigenvalue-like problems in a finite interval*

Consider a system of `NumEQ` ordinary differential equations (ODEs)

$$\mathcal{F}_i[z, u_1, u_2, \cdots, \lambda] = 0,\ z \in [\mathtt{zL_i}, \mathtt{zR_i}],\ 1 \leqslant i \leqslant \mathtt{NumEQ}, \tag{9.17}$$

subject to the `NumBC` linear boundary conditions,

$$\mathcal{B}_k[z, u_1, u_2, \cdots] = \gamma_k,\ 1 \leqslant k \leqslant \mathtt{NumBC}. \tag{9.18}$$

Note that there is an unknown parameter λ in the system (9.17), and the boundary conditions (9.18) include an additional boundary condition determining the unknown λ. This additional boundary condition is the zeroth boundary condition in `BVPh 1.0`. However, it is treated the same as the other boundary conditions in `BVPh 2.0`, that is, any one of the boundary conditions in (9.18) can be the additional boundary condition provided that $\mathtt{NumEQ} - 1$ is the order of the system (9.17).

All related formulas are the same as those given in § 9.4.1, except that a deformation $\Lambda(q)$ is also constructed such that $\Lambda(q)$ varies continuously from the initial guess λ_0 to λ as q increases from 0 to 1.

Note that, when the BVPh 2.0 is used to solve eigenvalue-like problems in a finite interval, we should set `TypeEQ=2`.

9.4.3. *Problems in a semi-infinite interval*

When the right endpoint $\mathtt{zR}_i$ of the solution interval in (9.5) is $+\infty$, the governing equation is defined in a semi-infinite interval $[\mathtt{zL}_i, +\infty]$. The `BVPh` can solve this kind of problem without truncating the domain. This feature is quite different from `BVP4c`, which regards a semi-infinite interval as a kind of singularity and replaces it by a finite one.

All related formulas are the same as those given in § 9.4.1, except that the finite intervals $z \in [\mathtt{zL_i}, \mathtt{zR_i}]$ are replaced by the semi-infinite ones $z \in [\mathtt{zL_i}, \infty]$. However, completely different initial guesses $u_{i,0}(z)$ and auxiliary linear operators $\mathcal{L}_i$ are used for this kind of problems, because their solutions are expressed by completely different base functions.

Note that i) when the `BVPh 2.0` is used to solve problems in a semi-infinite interval, the symbol `infinity` is used to denote ∞ in input data; ii) the boundary conditions at `infinity` are considered as a limiting process;

iii) to get the error of the approximations, we have to provide finite intervals over which the squared residuals are integrated, as the numerical integration method is used in `BVPh 2.0`; iv) the integral interval for squared residual error can be set through `zLintegral[i]` and `zRintegral[i]`. For example, `zLintegral[i]=0` and `zRintegral[i]=10` means the squared residual of the ith governing equation is integrated over $[0, 10]$.

9.5. Approximation and iteration of solutions

Although the high-order deformation equations (9.11) are always linear, they are still not easy to solve in general, because the right-hand side term $\delta_{i,m-1}$ may be rather complicated. To solve this problem, we can approximate $\delta_{i,m-1}$ in the form

$$\delta_{i,m-1} \approx \sum_{k=0}^{N_t} b_k e_k(z)$$

where e_k denotes the base-function, N_t denotes the number of truncated terms, the coefficient b_k is uniquely determined by $\delta_{i,m-1}(z)$ and the base-functions $e_k(z)$.

To further accelerate the rate of convergence, we can employ the iteration approach by using the Mth-order approximation as a new initial guess $u^*_{i,0}(z)$, i.e.,

$$u_{i,0}(z) + \sum_{m=1}^{M} u_{i,m}(z) \to u^*_{i,0}(z).$$

The above expression provides us the so-called Mth-order iteration formula. In this way, the convergence of the homotopy-series can be greatly accelerated.

The iteration approach of the `BVPh 2.0` is currently possible only when the base functions are of polynomials, trigonometric functions and hybrid-base functions. We consider here the approximation of a smooth function $f(z)$ in these three different kinds of base functions. The approximation of a function in a semi-infinite interval is proposed in Ref. 20 in the frame of HAM.

9.5.1. *Polynomials*

It is well known that a smooth function $f(z)$ in the finite interval $z \in [-1, 1]$ can be well approximated by the Chebyshev polynomials

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{N_t} a_k T_k(z), \tag{9.19}$$

where $T_k(z)$ is the kth Chebyshev polynomial of the first kind, N_t denotes the number of Chebyshev polynomials, and

$$\begin{aligned} a_k &= \frac{2}{N_t+1} \sum_{i=1}^{N_t+1} f(x_i) T_k(x_i), \\ &= \frac{2}{N_t+1} \sum_{i=1}^{N_t+1} f\left[\cos\left\{\frac{\pi(i-\frac{1}{2})}{N_t+1}\right\}\right] \cos\left\{\frac{\pi k(i-\frac{1}{2})}{N_t+1}\right\}. \end{aligned}$$

When Chebyshev polynomial is used to approximate the boundary value/eigenvalue problems in a finite interval $z \in [a, b]$ by means of the `BVPh 2.0`, we should set `TypeL = 1`, `ApproxQ = 1`.

9.5.2. *Trigonometric functions*

It is well known that the Fourier series

$$\frac{a_0}{2} + \sum_{n=1}^{+\infty} \left(a_n \cos\frac{n\pi z}{a} + b_n \sin\frac{n\pi z}{a} \right)$$

of a continuous function $f(z)$ in a finite interval $z \in (-a, a)$ converges to $f(z)$ in the interval $z \in (-a, a)$, where

$$a_n = \frac{1}{a}\int_{-a}^{a} f(t)\cos\frac{n\pi t}{a}dt,\ b_n = \frac{1}{a}\int_{-a}^{a} f(t)\sin\frac{n\pi t}{a}dt.$$

For a continuous function $f(z)$ in $[0, a]$, we can define $f(z) = f(-z)$ in $z \in [-a, 0)$ and its Fourier series reads

$$f(z) = \frac{a_0}{2} + \sum_{n=1}^{+\infty} a_n \cos\frac{n\pi z}{a}. \tag{9.20}$$

Alternatively, we can define $f(z) = -f(-z)$ in $z \in [-a, 0)$ and its Fourier series reads

$$f(z) \sim \sum_{n=1}^{+\infty} b_n \sin\frac{n\pi z}{a}. \tag{9.21}$$

Note that in (9.21), we write "$\sim$" instead of "$=$", because it holds only if $f(0) = f(a) = 0$ is assumed.

In practice, we have the following approximations

$$f(z) \approx \frac{a_0}{2} + \sum_{n=1}^{N_t} a_n \cos \frac{n\pi z}{a} \tag{9.22}$$

or

$$f(z) \approx \sum_{n=1}^{N_t} b_n \sin \frac{n\pi z}{a}, \tag{9.23}$$

where N_t denotes the number of truncated terms.

When the above-mentioned trigonometric functions are used to solve boundary value/eigenvalue problems in a finite interval $z \in [0, a]$ by means of the `BVPh 2.0`, we should set `TypeL = 2`, `ApproxQ = 1` and `HYBRID = 0` with `TypeBase = 1` for the odd expression (9.23) and `TypeBase = 2` for the even expression (9.22), respectively.

9.5.3. *Hybrid-base functions*

Note that the first-order derivative of the even Fourier series (9.20) equals to zero at the two endpoints $z = 0$ and $z = a$, but the original function $f(z)$ may have arbitrary values of $f'(0)$ and $f'(a)$. So, in case of $f'(0) \neq 0$ and/or $f'(a) \neq 0$, one had to use many terms of the even Fourier series (9.20) so as to obtain an accurate approximation near the two endpoints. To overcome this disadvantage, we first express $f(z)$ by such a combination

$$f(z) \approx Y(z) + w(z), \tag{9.24}$$

where

$$Y(z) = \left(f'(0)z - \frac{f'(0) + f'(a)}{2a} z^2 \right) \cos \frac{\pi z}{a} \tag{9.25}$$

and then approximate $w(z) = f(z) - Y(z)$ by the even Fourier series

$$w(z) \approx \frac{\bar{a}_0}{2} + \sum_{n=1}^{N_t} \bar{a}_n \cos \frac{n\pi z}{a} \tag{9.26}$$

with the Fourier coefficient

$$\bar{a}_n = \frac{2}{a} \int_0^a [f(t) - Y(t)] \cos \frac{n\pi t}{a} dt.$$

Here N_t is the number of truncated terms. Note that $Y(z)$ in (9.25) satisfies

$$Y'(0) = f'(0),\ Y'(a) = f'(a)$$

so that $w'(0) = w'(a) = 0$. Therefore, we often need a few terms of the even Fourier series to accurately approximate $w(z)$. Note also that both the trigonometric and polynomial functions are used in (9.24) to approximate $f(z)$. It is found that, by means of such kind of approximations based on hybrid-base functions, one often needs much less terms to approximate a given smooth function $f(z)$ in $[0, a]$ than the traditional Fourier series.

Alternatively, for a continuous function $f(z)$ in $[0, a]$, we can use

$$Y(z) = f(0) + \frac{f(a) - f(0)}{a} z \tag{9.27}$$

or

$$Y(z) = \frac{f(0) + f(a)}{2} + \frac{f(0) - f(a)}{2} \cos \frac{\pi z}{a}, \tag{9.28}$$

and approximate $w(z)$ by the odd Fourier series

$$w(z) \approx \sum_{n=1}^{N_t} \bar{b}_n \sin \frac{n\pi z}{a}, \tag{9.29}$$

where

$$\bar{b}_n = \frac{2}{a} \int_0^a [f(t) - Y(t)] \sin \frac{n\pi t}{a} dt.$$

Note that $Y(z)$ in (9.27) and (9.28) satisfies

$$Y(0) = f(0),\ Y(a) = f(a)$$

so that $w(0) = w(a) = 0$.

It is suggested to use the hybrid-base approximation (9.24) with (9.25) for an even function $f(z)$, and (9.24) with (9.27) or (9.28) for an odd function $f(z)$, respectively. If $f(z)$ is neither an odd nor even function, both of them work.

When the above-mentioned hybrid-base approximation is used, we have even larger freedom to choose the initial guess $u_0(z)$. For example, for a 2nd-order boundary value/eigenvalue problem in a finite interval $z \in [0, a]$, we may choose such an initial guess in the form

$$u_0(z) = B_0 + B_1 \cos \frac{\kappa\pi z}{a} + B_2 \sin \frac{\kappa\pi z}{a} \tag{9.30}$$

or

$$u_0(z) = (B_0 + B_1 z + B_2 z^2) \cos \frac{\kappa \pi z}{a}, \tag{9.31}$$

where B_0, B_1 and B_2 are determined by linear boundary conditions, κ is a positive integer.

When the above-mentioned trigonometric functions are used to solve boundary value/eigenvalue problems in a finite interval $z \in [0, a]$ by means of the BVPh 2.0, we should set `TypeL = 2`, `ApproxQ = 1` and `HYBRID = 1` with `TypeBase = 1` for the odd expression (9.29) and `TypeBase = 2` for the even expression (9.26), respectively.

9.6. A simple user guide of the BVPh 2.0

In this section, we will take a glance at the Mathematica package `BVPh 2.0`.

9.6.1. *Key modules*

`BVPh` The module `BVPh[k_,m_]` gives the kth to mth-order homotopy approximations of a system of ordinary differential equations (ODEs) subject to some boundary conditions. The system may have an unknown parameter (when `TypeEQ = 2`) or may not have an unknown parameter (when `TypeEQ = 1`). It is the basic module. For example, `BVPh[1,10]` gives the 1st-order to 10th-order homotopy-approximations. Thereafter, `BVPh[11,15]` further gives the 11th-order to 15th-order homotopy-approximations. For problems with an unknown parameter, the initial guess of the unknown parameter is determined by an algebraic equation. Thus, if there are more than one initial guesses of the unknown parameter, it is required to choose one among them by inputting an integer, such as 1 or 2, corresponding to the 1st or the 2nd initial guess of the unknown parameter, respectively.

`iter` The module `iter[k_,m_,r_]` provides us homotopy approximations of the kth to mth iteration by means of the rth-order iteration formula. It calls the basic module `BVPh`. For example, `iter[1,10,3]` gives homotopy-approximations of the 1st to 10th iteration by the 3rd-order iteration formula. Furthermore, `iter[11,20,3]` gives the homotopy-approximations of the 11th to 20th iterations. The iteration stops when

the averaged squared residual error of the system is less than a critical value `ErrReq`, whose default is 10^{-20}.

`GetErr` The module `GetErr[k_]` gives the averaged squared residual error of the governing equation at the kth-order homotopy-approximation gained by the module `BVPh`. Note that, `errer[i,k]` provides the residual of the ith governing equation at the kth-order homotopy-approximation gained by `BVPh`, and `ErrTotal[k]` gives the total averaged squared residual error of the system at the kth-order homotopy-aprroximation gained by `BVPh`.

`hp` The module `hp[f_,m_,n_]` gives the `[m,n]` homotopy-padé approximation of a list of the homotopy-approximations `f`, where `f[[i+1]]` denotes the ith-order homotopy-approximation of the same governing equation.

`GetBC` The module `GetBC[i_,k_]` gives the ith boundary condition of the kth-order deformation equation.

9.6.2. *Control parameters*

`TypeEQ` A control parameter for the type of governing equations: `TypeEQ = 1` corresponding to a nonlinear problem without an unknown parameter, `TypeEQ = 2` corresponds to a nonlinear problem with an unknown parameter (called eigenvalue problem or eigenvalue-like problem), respectively.

`TypeL` A control parameter for the type of auxiliary linear operator: `TypeL = 1` corresponds to polynomial approximation, and `TypeL = 2` corresponds to a trigonometric approximation or a hybrid-base approximation, respectively.

`ApproxQ` A control parameter for approximation of solutions. When `ApproxQ = 1`, the right-hand side term of all higher-order deformation equations are approximated by Chebyshev polynomials (9.19), trigonometric functions in § 9.5.2, or by the hybrid-base functions in § 9.5.3. When `ApproxQ = 0`, there is no such kind of approximation. When `TypeL = 2`, `ApproxQ = 1` is valid only for problems in a finite interval $z \in [0, a]$, where $a > 0$ is a constant.

`HYBRID` A control parameter for the hybrid-base functions. When `HYBRID = 1`, hybrid-base functions are employed in approximation.

When `HYBRID = 0`, trigonometric functions without polynomials are employed in approximation. This parameter is usually used in conjunction with `TypeBase`, and is valid only when `TypeL = 2` and `ApproxQ = 1` for problems in a finite interval $z \in [0, a]$.

`TypeBase` A control parameter for the type of Fourier series approximation: `TypeBase = 1` corresponds to the odd Fourier approximation (9.23) or (9.29), `TypeBase = 2` corresponds to the even Fourier approximation (9.22) or (9.26), respectively. This parameter is usually used in conjunction with `HYBRID`, and is valid only when `TypeL = 2` and `ApproxQ = 1` for problems in a finite interval $z \in [0, a]$.

`Ntruncated` A control parameter to determine the number of truncated terms used to approximate the right-hand side of higher-order deformation equations. The larger `Ntruncated`, the better the approximations, but the more CPU time. It is valid only when `ApproxQ = 1`. The default is 10.

`NtermMax` A positive integer used in the module `truncated`, which ignores all polynomial terms whose order is higher than `NtermMax`. The default is 90.

`ErrReq` A critical value of the averaged squared residual error of governing equations to stop the computation. The default is 10^{-20}.

`NgetErr` A positive integer used in the module `BVPh`. The averaged squared residual error of governing equations is calculated when the order of approximation is divisible by `NgetErr`. The default is 2.

`Nintegral` Number of discrete points with equal space, which are used to numerically calculate the integral of a function. It is used in the module `int`. The default is 50.

`ComplexQ` A control parameter for complex variables. `ComplexQ = 1` corresponds to the existence of complex variables in governing equations and/or boundary conditions. `ComplexQ = 0` corresponds to the nonexistence of such kind of complex variables. The default is 0.

`FLOAT` A control parameter for floating-point computation. When `FLOAT = 1`, floating-point numbers are employed in computation. When `FLOAT = 0`, rational numbers are employed in computation. The default is 1.

9.6.3. *Input*

`NumEQ` The number of governing equations.

`f[i_,z_,{u_,⋯},lambda_]` The ith governing equation with or without an unknown parameter, corresponding to $\mathcal{F}_i[z, u, \cdots]$ or $\mathcal{F}_i[z, u, \cdots, \lambda]$ in either a finite interval $z \in [a, b]$ or a semi-infinite interval $z \in [b, +\infty)$, where a and b are bounded constants. Note that the formal parameter `lambda` denotes the unknown parameter λ to be determined, or the eigenvalue λ for eigenvalue problems, but has no meaning at all for problems without an unknown parameter λ, or non-eigenvalue problems.

`NumBC` The number of boundary conditions.

`BC[k_,z_,{u_,⋯}]` The kth boundary condition corresponding to $\mathcal{B}_k[z, u, \cdots]$, where $1 \le k \le$ `NumBC`. Note that the symbol `infinity` denotes ∞ in boundary conditions.

`U[i,0]` The initial guess $U_{i,0}(z)$, i.e., $u_{i,0}(z)$.

`c0[i]` The convergence-control parameter $c_{0,i}$, corresponding to the ith governing equation.

`H[i_,z_]` The auxiliary function corresponding to the ith governing equation. The default is `H[i_,z_]:=1`.

`L[i_,f_]` The auxiliary linear operator corresponding to the ith governing equation.

`zL[i]` The left endpoint of the interval of the solution corresponding to the ith governing equation. Note that intervals of the solutions are not necessarily the same, especially for multi-layer flow problem.

`zR[i]` The right endpoint of the interval of the solution corresponding to the ith governing equation.

`zLintegral[i]` The left endpoint of the integral interval to compute the averaged squared residual error of the ith governing equation. When the left endpoint of the solution interval for the ith governing equation is a finite number, `zLintegral[i]` is automatically set to `zL[i]`. Otherwise, the user has to specify the value of `zLintegral[i]`.

`zRintegral[i]` The right endpoint of the integral interval to compute the averaged squared residual error of the ith governing equation. When the right endpoint of the solution interval for the ith governing equation is a finite number, `zRintegral[i]` is automatically set to `zR[i]`. Otherwise, the user has to specify the value of `zRintegral[i]`.

9.6.4. *Output*

`U[i,k]` The kth-order homotopy-approximation of the solution to the ith governing equation given by the basic module `BVPh`.

`V[i,k]` The kth-iteration homotopy-approximation of the solution to the ith governing equation given by the iteration module `iter`.

`Lambda[k]` The kth-order homotopy-approximation of the eigenvalue λ or the unknown parameter λ given by the basic module `BVPh`.

`LAMBDA[k]` The kth-iteration homotopy-approximation of the eigenvalue λ or the unknown parameter λ given by the iteration module `iter`.

`error[i,k]` The residual of the ith governing equation given by the kth-order homotopy-approximation (obtained by the basic module `BVPh`).

`Err[k]` A list of the averaged squared residual error of each governing equation given by the kth-order homotopy-approximation (obtained by the basic module `BVPh`).

`ErrTotal[k]` The total of the averaged squared residual error for each governing equation given by the kth-order homotopy-approximation (obtained by the basic module `BVPh`).

`ERR[k]` A list of the averaged squared residual error of each governing equation given by the kth-iteration homotopy-approximation (obtained by the iteration module `iter`).

`ERRTotal[k]` The total of the averaged squared residual error for each governing equation given by the kth-iteration homotopy-approximation (obtained by the iteration module `iter`).

9.6.5. *Global variables*

All control parameters and output variables mentioned above are global. Besides these, the following variables and parameters are also global.

`z` The independent variable z.

`u[i,k]` The solution to the kth-order deformation equation of the ith governing equation.

`lambda[k]` A constant variable, corresponding to λ_k.

`delta[i,k]` A function dependent upon z, corresponding to the right-hand side term $\delta_{i,k}(z)$ in the high-order deformation equation.

`L[i,w]` The ith auxiliary linear operator $\mathcal{L}_i$ applied to w.

`Linv[i,f]` The inverse operator of $\mathcal{L}_i$, corresponding to $\mathcal{L}_i^{-1}$, applied to f.

`sNum` A positive integer to determine which initial guess λ_0 is chosen when there are multiple solutions of λ_0.

9.7. Examples

More examples are given in this section to show the usage the package `BVPh 2.0`.

9.7.1. *Example 1: A system of ODEs in finite interval*

Consider a system of coupled ODEs [7]

$$(1+K)f'''' - ReMf'' + 2Reff''' - Kg'' = 0, \tag{9.32}$$

$$\left(1+\frac{K}{2}\right)g'' - ReK[2g - f''] + Re[2fg' - f'g] = 0, \tag{9.33}$$

subject to

$$f(0) = 0,\ f(1) = 0,\ f'(1) = 1,\ f''(0) = 0, \tag{9.34}$$

$$g(1) = 0,\ g(0) = 0, \tag{9.35}$$

where K is the ratio of viscosities, Re is the Reynolds number and M is the Hartman number. Hayat [7] has solved this problem by the HAM.

Here we solve this problem by `BVPh 2.0`. Since there are two ODEs in system (9.32)–(9.33) without an unknown to be determined, we have `NumEQ = 2` and `TypeEQ=1`. The system is input as

```
TypeEQ = 1;
NumEQ = 2;
f[1,z_,{f_,g_},Lambda_]:=(1+K)*D[f,{z,4}]
    -Rey*M*D[f,{z,2}]+2*Rey*f*D[f,{z,3}]-K*D[g,{z,2}];
f[2,z_,{f_,g_},Lambda_]:=(1+K/2)*D[g,{z,2}]
    -Rey*K*(2*g-D[f,{z,2}])+Rey*(2*f*D[g,z]-D[f,z]*g);
```

The six boundary conditions are defined as

```
NumBC = 6;
BC[1,z_,{f_,g_}]:=f/.z->0;
BC[2,z_,{f_,g_}]:=f/.z->1;
BC[3,z_,{f_,g_}]:=(D[f, z]-1)/.z->1;
BC[4,z_,{f_,g_}]:=D[f,{z,2}]/.z->0;
BC[5,z_,{f_,g_}]:=g/.z->1;
BC[6,z_,{f_,g_}]:=g/.z->0;
```

Now let us input the solution intervals

```
zL[1]=0;    zR[1]=1;
zL[2]=0;    zR[2]=1;
```

Since all the solution intervals are finite, we do not have to specify the integral interval to compute the averaged squared residual error.

The initial guesses are chosen as $f_0 = (z^3 - z)/2$ and $g_0 = 0$. They are input as

```
U[1,0] = (z^3-z)/2;
U[2,0] = 0;
```

The auxiliary linear operators are chosen as $\mathcal{L}_1 = \frac{\partial^4}{\partial z^4}$ and $\mathcal{L}_2 = \frac{\partial^2}{\partial z^2}$. They are defined as

```
L[1,u_]:=D[u,{z,4}];
L[2,u_]:=D[u,{z,2}];
```

Note that we use the delayed assignment `SetDelayed(:=)` to define these linear operators.

Without loss of generality, let us consider the case when $Re = M = 2$ and $K = 1/2$. These physical parameters are input as

```
Rey = M = 2;
K = 1/2;
```

At this time, we have input all the data for this problem, except the convergence-control parameters `c0[k]`. Hayat [7] chose the convergence-

control parameters `c0[1]=c0[2]=-0.7` through $\hbar$-curve. Here we minimize the averaged squared residual error of the 4th-order approximations to get optimal values for `c0[k]`

```
GetOptiVar[4,{},{c0[1],c0[2]}];
```

The convergence-control parameters `c0[1]` and `c0[2]` are found to be about -0.5825 and -0.721452, respectively.

Then we call the main module `BVPh` to get the 20th-order approximations

```
BVPh[1, 20];
```

The 20th-order approximations are stored in `U[i,20],i=1,2`, while the corresponding averaged squared residual error of the system is `ErrTotal[20]`. We can use

```
Plot[{U[1,20],U[2,20]},{z,0,1},AxesLabel->{"z",""},
PlotStyle->{{Thin, Red}, {Dashed, Blue}},
PlotRange->{{0, 1}, {-0.2, 0.2}}]
```

to plot the 20th-order approximations, which is shown in Fig. 9.3. This figure agrees with Hayat's Figs. 9 and 12 when $M = 2$, $Re = 2$ and $K = 0.5$. The 20th-order approximations give the values of $f''(1) = 3.61076396287$ and $g'(1) = -0.738463496789$, which are the same with Hayat's result [7]. The total error `ErrTotal[k]` of the system for every two order of approximations is plotted by the command

```
ListLogPlot[Table[{2 i, ErrTotal[2*i]}, {i, 1, 10}],
Joined -> True, Mesh -> All,
PlotRange -> {{2, 20}, {10^(-34), 1}},
AxesLabel -> {"m", "error"}]
```

in Fig. 9.4. We can see from it that the error decreases beautifully.

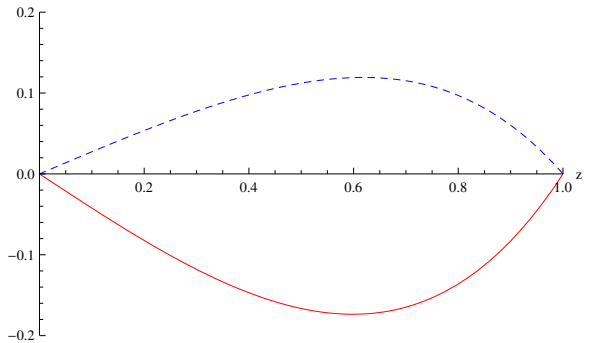

Fig. 9.3. The curve of $f(z)$ (solid), $g(z)$ (dashed) for Example 1.

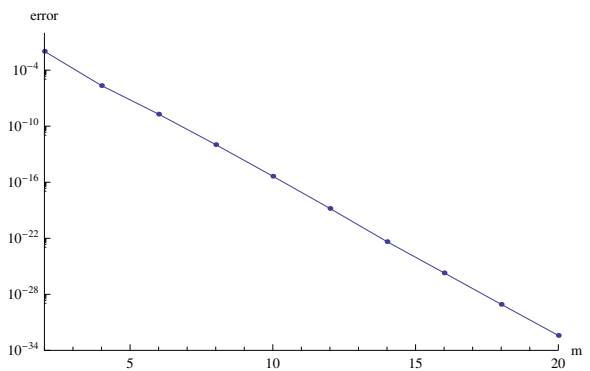

Fig. 9.4. Total error vs. order of approximation for Example 1.

9.7.2. *Example 2: A system of ODEs with algebraic property at infinity*

Consider a set of two coupled nonlinear differential equations [13]

$$f'''(\eta) + \theta(\eta) - f'^2 = 0, \tag{9.36}$$

$$\theta''(\eta) = 3\sigma f'(\eta)\theta(\eta), \tag{9.37}$$

subject to

$$f(0) = f'(0) = 0,\ \theta(0) = 1,\ f'(+\infty) = \theta(+\infty) = 0, \tag{9.38}$$

where the prime denotes differentiation with respect to the similarity variable η, σ is the Prandtl number, $f(\eta)$ and $\theta(\eta)$ relate to the velocity profile

and temperature distribution of the boundary layer, respectively. Liao [13] employed the HAM to solve this system analytically. Now we use the `BVPh 2.0` to solve it.

Under the transformation

$$\xi = 1 + \lambda\eta,\ F(\xi) = f'(\eta),\ S(\xi) = \theta(\eta), \tag{9.39}$$

Eqs. (9.36) and (9.37) become

$$\lambda^2 F''(\xi) + S(\xi) - F^2(\xi) = 0, \tag{9.40}$$

$$\lambda^2 S''(\xi) = 3\sigma F(\xi) S(\xi), \tag{9.41}$$

subject to

$$F(1) = 0,\ S(1) = 1,\ F(+\infty) = S(+\infty) = 0. \tag{9.42}$$

Since there are two ODEs in system (9.40)–(9.41) without an unknown to be determined, we have `NumEQ = 2` and `TypeEQ=1`. This new system is defined as

```
TypeEQ = 1;
NumEQ = 2;
f[1,z_,{F_,S_},Lambda_]:=la^2*D[F,{z,2}]+S-F^2;
f[2,z_,{F_,S_},Lambda_]:=la^2*D[S,{z, 2}]-3*sigma*F*S;
```

The four boundary conditions (9.42) are defined as

```
NumBC = 4;
BC[1,z_,{F_, S_}] := F /. z -> 1;
BC[2,z_,{F_, S_}] := (G - 1) /. z -> 1;
BC[3,z_,{F_, S_}] := F /. z -> infinity;
BC[4,z_,{F_, S_}] := G /. z -> infinity;
```

Now let us input the solution intervals and integral intervals to compute averaged squared residual error

```
zL[1] = 1;    zR[1] = infinity;
zL[2] = 1;    zR[2] = infinity;
zRintegral[1] = 10;
zRintegral[2] = 10;
```

The initial guesses are chosen as $F_0 = \gamma(\xi^{-2} - \xi^{-3})$, $S_0 = \xi^{-4}$, and they are input as

```
U[1, 0] = gamma*(z^(-2) - z^(-3));
U[2, 0] = z^(-4);
```

The auxiliary linear operators are $\mathcal{L}_F = \frac{\xi}{3}\frac{\partial^2}{\partial\xi^2} + \frac{\partial}{\partial\xi}$ and $\mathcal{L}_S = \frac{\xi}{5}\frac{\partial^2}{\partial\xi^2} + \frac{\partial}{\partial\xi}$, which are defined as

```
L[1, u_] := D[u, {z, 2}]*z/3 + D[u, z];
L[2, u_] := D[u, {z, 2}]*z/5 + D[u, z];
```

Without loss of generality, let us consider the case when $\sigma = 1$, $\gamma = 3$ and $\lambda = 1/3$. We use the same convergence-control parameters `c0[1]` = `c0[2]`$= -1/2$ as Liao [13]. These physical parameters and the control parameters `c0[k]` are defined as

```
sigma = 1;    gamma = 3;
la = 1/3;    c0[1] = -1/2;    c0[2] = -1/2;
```

Then we call the main module `BVPh`

```
BVPh[1, 20];
```

to get the 20th-order approximation. If we are not satisfied with the accuracy of the 20th-order approximation, we can use `BVPh[21,40]`, instead of `BVPh[1,40]`, to get 40th-order approximation or higher order approximation.

Note that `U[1,40]` and `U[2,40]` are the 40th-order approximations of the transformed system (9.40), (9.41) and (9.42). To plot the curve of the 40th-order approximations for the original problem, we first replace z with $1+\lambda\eta$ to obtain the 40th-order approximations for $f'(\eta)$ and $g(\eta)$, then plot the curve we want. This is done in Mathematica by the following command

```
trans = {z -> 1 + la*\[Eta]};
Plot[Evaluate[{U[1, 40], U[2, 40]} /. trans],
{\[Eta], 0, 10},PlotRange -> {{0, 10}, {0, 1}},
AxesLabel -> {"\[Eta]", ""},
PlotStyle -> {{Thin, Red}, {Dashed, Blue}}]
```

and the curve is shown in Fig. 9.5. Here `trans={z->1+la*\[Eta]}` is the corresponding transformation, and `\[Eta]` is the symbol η in Mathematica.

The total error `ErrTotal[k]` of the transformed system for every two order approximations is plotted in Fig. 9.6 by the following command

```
ListLogPlot[Table[{2 i, ErrTotal[2*i]}, {i, 1, 20}],
Joined -> True, Mesh -> All,
PlotRange -> {{2, 40}, {10^(-10), 0.01}},
AxesLabel -> {"m", "error"}]
```

Note that `ErrTotal[k]` not only measures the accuracy of the kth-order approximations for the transformed problem, but also measures the corresponding approximations for the original problem.

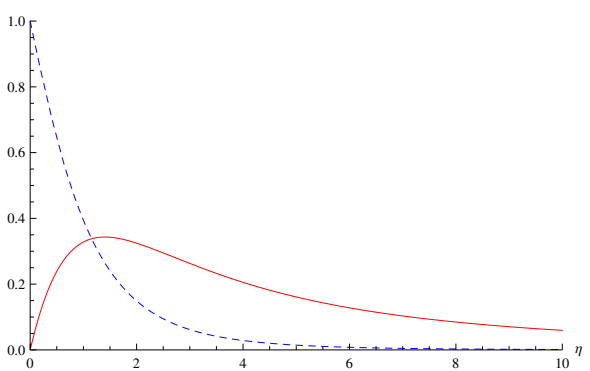

Fig. 9.5. The curve of $f'(\eta)$ (solid) and $\theta(\eta)$ (dashed) for Example 2.

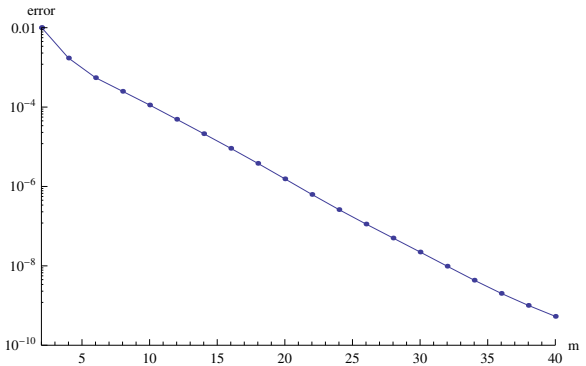

Fig. 9.6. Total error vs. order of approximation for Example 2.

The 40th-order approximations of $f''(0)$ and $g''(0)$ are 0.693268 and -0.769879, respectively. Kuiken's numerical result is $f''(0) \approx 0.693212$ and $g'(0) \approx -0.769861$. To get more accurate result, we have two choices. One is to call the module `BVPh` to get higher order approximation as before, the other is to apply the Padé approximation to the current approximations. The latter is done by calling the module `hp` as follows

```
hp[Table[D[(U[1,i]/.trans),\[Eta]]/.\[Eta]->0,
{i,0,40}],20,20]
hp[Table[D[(U[2,i]/.trans),\[Eta]]/.\[Eta]->0,
{i,0,40}],20,20]
```

which give 0.693212 and -0.769861, the $[20, 20]$ homotopy-Padé approximations of $f''(0)$ and $g'(0)$, respectively.

Note that we can compare the curve of $2n$th-order approximation and the $[n, n]$ homotopy-Padé approximation in a simple and efficient way. Here we compare `U[1,40]` and the $[20, 20]$ homotopy-Padé approximation of `U[1,i]`, $i = 0 \cdots 40$, in the Mathematica by the following command.

```
Plot[{U[1, 40]/.trans, hp[Table[U[1,i]/.trans,
{i, 0, 40}],20, 20]},{\[Eta],0,10},PlotRange->Full,
AxesLabel->{"\[Eta]", ""},
PlotStyle->{{Thin,Red},{Dashed,Blue}}
```

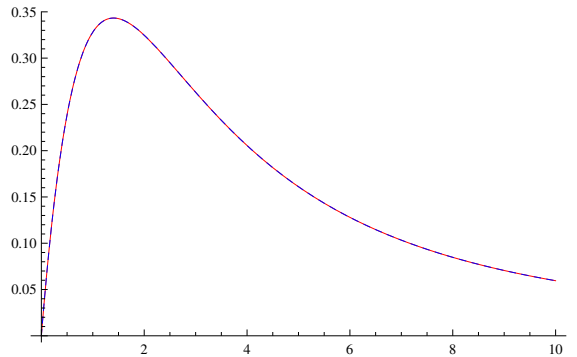

Fig. 9.7. The curve of 40th-order approximation of $f'(\eta)$ (solid) and $[20, 20]$ homotopy-Padé approximations of $f'(\eta)$ (dashed) for Example 2.

The comparison is shown in Fig. 9.7. From it we can see that the two are almost the same. This validate the convergence of the approximations to some extent. The above command is very efficient, because the `Plot` command in Mathematica first substitute the sample points into the expression and then applies the `hp` to a list of numerical values, rather than applies the `hp` to a list of expressions and then substitute the sample points into the resulting expression.

9.7.3. *Example 3: A system of ODEs with an unknown parameter*

Consider a system of ODEs [15]

$$U'' + (GrPr)\theta - N_r\phi + \sigma = 0, \tag{9.43}$$

$$\theta'' + N_b\theta'\phi' + N_t(\theta')^2 + N_b + N_t - U = 0, \tag{9.44}$$

$$\phi'' + \frac{N_t}{N_b}\theta'' - L_eU = 0, \tag{9.45}$$

subject to

$$U(-1) = U(1) = 0,\ \theta(-1) = \theta(1) = 0,\ \phi(-1) = \phi(1) = 0, \tag{9.46}$$

with an additional condition

$$\int_0^1 UdY = RePr, \tag{9.47}$$

where Gr is the Grashof number, Pr the Prandtl number, Nr the buoyancy ratio, σ the pressure parameter, Nb the Brownian motion parameter, Nt the thermophoresis parameter, Le the Lewis number, and Re the Reynolds number. All of the above parameters will be given for a special case except σ, which is to be determined from the system. Xu [15] solved this problem by the HAM.

Here we solve this problem by `BVPh 2.0`. Since there are three ODEs in system (9.43)–(9.45) with an unknown σ to be determined, we have `NumEQ = 3` and `TypeEQ=2`. The system is input as

```
TypeEQ = 2;    NumEQ = 3;
f[1,z_,{f_,g_,s_},sigma_] :=
    D[f,{z,2}]+Gr*Pr*g-Nr*s+sigma;
f[2,z_,{f_,g_,s_},sigma_] :=
    D[g,{z,2}]+Nb*D[g,z]*D[s,z]+Nt*(D[g,z])^2-f;
f[3,z_,{f_,g_,s_},sigma_] :=
    D[s,{z,2}]+Nt/Nb*D[f,{z,2}]-Le*f;
```

The seven boundary conditions, including the additional condition (9.47), are defined as

```
NumBC = 7;
BC[1,z_,{f_,g_,s_}] :=f/.z->-1;
BC[2,z_,{f_,g_,s_}] :=f/.z->1;
BC[3,z_,{f_,g_,s_}] :=g/.z->-1;
BC[4,z_,{f_,g_,s_}] :=g/.z->1;
BC[5,z_,{f_,g_,s_}] :=s/.z->-1;
BC[6,z_,{f_,g_,s_}] :=s/.z->1;
BC[7,z_,{f_,g_,s_}] :=Integrate[f,{z,0,1}]-Ra*Pr;
```

Now let us input the solution intervals

```
zL[1] = -1;    zR[1] = 1;
zL[2] = -1;    zR[2] = 1;
zL[3] = -1;    zR[3] = 1;
```

Since all the solution intervals are finite, we do not have to specify the

integral interval to compute the averaged squared residual error.

The initial guesses are chosen as $U_0 = \epsilon_1 - 3(-25+\epsilon_1)z^2/2 + 5(-15+2\epsilon_1)z^4/2$, $\theta_0 = \epsilon_2(1-z^2)$ and $\phi_0 = \epsilon_3(1-z^2)$, where ϵ_1, ϵ_2 and ϵ_3 are constants to be optimized. They are input as

```
U[1,0]=eps1-3/2*(-25+4eps1)z^2+5/2*(-15+2eps1)*z^4;
U[2,0]=eps2*(1-z^2);
U[3,0]=eps3*(1-z^2);
```

The auxiliary linear operators are chosen as $\mathcal{L}_1 = \mathcal{L}_2 = \mathcal{L}_3 = \frac{\partial^2}{\partial Y^2}$. They are defined as

```
L[1, u_] := D[u, {z, 2}];
L[2, u_] := D[u, {z, 2}];
L[3, u_] := D[u, {z, 2}];
```

Note that we use the delayed assignment `SetDelayed(:=)` to define these linear operators.

Without loss of generality, let us consider the case when $Nr = 3/20$, $Nt = Nb = 1/20$, $Le = 10$, $Gr = 5$, $Pr = 1$, and $Re = 5$. These physical parameters are input as

```
Nr = 3/20;    Nt = 1/20;
Nb = 1/20;    Le = 10;
Gr = 5;    Pr = 1;    Ra = 5;
```

At this time, we have input all the data for this problem, except the convergence-control parameters `c0[k]`, `eps1`, `eps2` and `eps3`. We minimize the averaged squared residual error of the 3th-order approximations to get optimal values for these parameters by the module `GetOptiVar` as follow

```
c0[1] = c0[2] = c0[3] = h;
GetOptiVar[3, {}, {eps1, eps2, eps3, h}];
```

Note that we put constraints `c0[1]=c0[2]=c0[3]` on `c0[1]`, `c0[2]` and `c0[3]` to simplify the computation. There is no constraint on `eps1`, `eps2` and `eps3`.

After some computation, we get optimal values for all the convergence-control parameters `c0[1]` = `c0[2]` = `c0[3]` ≈ -0.769452, `eps1` ≈ 7.56408, `eps2` ≈ -2.58887 and `eps3` ≈ -30.0044. Now we can use

```
BVPh[1,10]
```

to get the 10th-order approximation. If we are not satisfied with the accuracy of the 10th-order approximation, we can use `BVPh[11,20]` to get 20th-order approximation or higher order approximation.

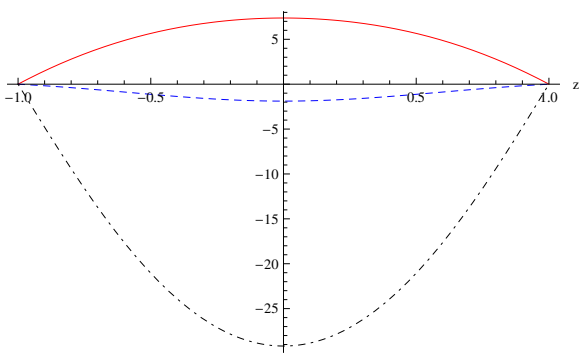

Fig. 9.8. The curve of U (solid), θ (dashed) and $\phi(z)$ (dot dashed) for Example 3.

The 20th-order approximations of U, θ and ϕ are stored in `U[1,20]`, `U[2,20]` and `U[3,20]`, the 20th-order approximation of σ is stored in `Lambda[19]`, while the corresponding averaged squared residual error of the system is stored in `ErrTotal[20]`. `Lambda[19]` is about 18.272555944, which is the same with Xu's result [15]. The 20th-order approximations are plotted in Fig. 9.8. The total error of the system for every two order of approximations is plotted in Fig. 9.9.

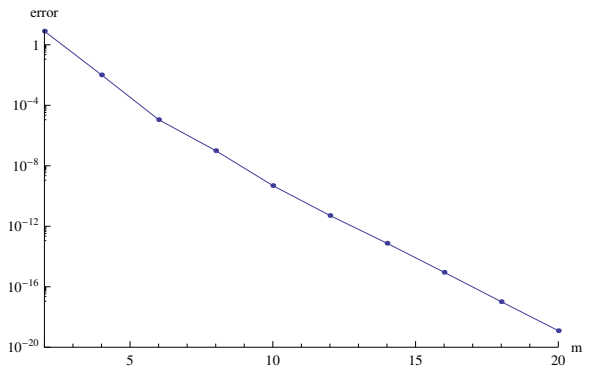

Fig. 9.9. Total error vs. order of approximation for Example 3.

9.7.4. *Example 4: A system of ODEs in different intervals*

Consider a two-phase flow [16]:

(i) Region 1

$$\frac{d^2u_1}{dy^2} + \frac{Gr}{Re}\sin(\phi)\theta_1 = P, \tag{9.48}$$

$$\frac{d^2\theta_1}{dy^2} + PrEc\left(\frac{du_1}{dy}\right)^2 = 0, \tag{9.49}$$

(ii) Region 2

$$\frac{d^2u_2}{dy^2} + \frac{Gr}{Re}\frac{nbh^2}{\lambda}\sin(\phi)\theta_2 - \frac{M^2h^2}{\lambda}u_2 = \frac{h^2}{\lambda}P, \tag{9.50}$$

$$\frac{d^2\theta_2}{dy^2} + EcPr\frac{\lambda}{\lambda_T}\left(\frac{du_2}{dy}\right)^2 + EcPr\frac{h^2}{\lambda_T}M^2u_2^2 = 0, \tag{9.51}$$

subject to

$$u_1(1) = 1,\ \theta_1(1) = 1, \tag{9.52}$$

$$u_1(0) = u_2(0),\ \theta_1(0) = \theta_2(0), \tag{9.53}$$

$$u_1'(0) = \frac{\lambda}{h}u_2'(0),\ \theta_1'(0) = \frac{\lambda_T}{h}\theta_2'(0), \tag{9.54}$$

$$u_2(-1) = 0,\ \theta_2(-1) = 0, \tag{9.55}$$

where Gr is the Grashof number, Ec is the Eckert number, Pr is the Prandtl number, Re is the Reynolds number, M is the Hartmann number and P

is the dimensionless pressure gradient. This model describes a two-fluid magnetohydrodynamic Poiseuille–Couette flow and heat transfer in an inclined channel. Umavathi [16] investigate this model analytically by regular perturbation method and numerically by finite difference technique. The `BVPh 2.0` can solve this problem (9.48)–(9.55) directly without difficulty. Since all the parameters in the system will be given, we have `NumEQ = 4` and `TypeEQ=1`. The system is input as

```
TypeEQ = 1;
NumEQ = 4;
f[1,z_,{u1_,s1_,u2_,s2_},Lambda_]:=
    D[u1, {z, 2}]+ Gr/Ra*Sin[phi]*s1 - P ;
f[2,z_,{u1_,s1_,u2_,s2_},Lambda_]:=
    D[s1, {z, 2}] + Pr*Ec*(D[s1, z])^2;
f[3,z_,{u1_,s1_,u2_,s2_},Lambda_]:=D[u2,{z, 2}]-h^2/lamb*P
    +Gr/Ra*Sin[phi]*n*b*h^2/lamb*s2-M^2*h^2/lamb*u2;
f[4,z_,{u1_,s1_,u2_,s2_},lambda_]:=D[s2,{z,2}]
    +Pr*Ec*lamb/lambT*D[u2,z]^2+Pr*Ec*h^2/lambT*M^2*u2^2;
```

The eight boundary conditions (9.52)–(9.55) are defined as

```
NumBC=8;
BC[1,z_,{u1_,s1_,u2_,s2_}]:=(u1-1)/.z->1;
BC[2,z_,{u1_,s1_,u2_,s2_}]:=(u1-u2)/.z->0;
BC[3,z_,{u1_,s1_,u2_,s2_}]:=u2/.z->-1;
BC[4,z_,{u1_,s1_,u2_,s2_}]:=(D[u1,z]-D[u2,z]*lamb/h)/.z->0;
BC[5,z_,{u1_,s1_,u2_,s2_}]:=(s1-1)/.z->1;
BC[6,z_,{u1_,s1_,u2_,s2_}]:=(s1-s2)/.z->0;
BC[7,z_,{u1_,s1_,u2_,s2_}]:=s2/.z->-1;
BC[8,z_,{u1_,s1_,u2_,s2_}]:=(D[s1,z]-D[s2,z]*lambT/h)/.z->0;
```

Now let us input the solution intervals

```
    zL[1] = 0;  zR[1] = 1; (* u1 *)
    zL[2] = 0;  zR[2] = 1; (* s1 *)
    zL[3] = -1; zR[3] = 0; (* u2 *)
    zL[4] = -1; zR[4] = 0; (* s2 *)
```

Note that the solution intervals are not the same. Since all the solution intervals are finite, we do not have to specify the integral interval to compute the averaged squared residual error.

The initial guesses are chosen as $u_{1,0} = \frac{\lambda}{h}(z - z^2) + 1$, $\theta_{1,0} = \frac{z\lambda_T}{h} + (1 - \frac{\lambda_T}{h})z^2$, $u_{2,0} = 1 + z$ and $\theta_{2,0} = z + z^2$. They are input as

```
U[1, 0] = (z - z^2)*lamb/h+1;            (* u1 *)
U[2, 0] = z*lambT/h +(1-lambT/h)*z^2;    (* s1 *)
U[3, 0] = 1 + z;                         (* u2 *)
U[4, 0] = z^2 + z;                       (* s2 *)
```

The auxiliary linear operators are chosen as $\mathcal{L}_1 = \mathcal{L}_2 = \mathcal{L}_3 = \mathcal{L}_4 = \frac{\partial^2}{\partial y^2}$. They are defined as

```
L[1,u_]:=D[u,{z,2}];
L[2,u_]:=D[u,{z,2}];
L[3,u_]:=D[u,{z,2}];
L[4,u_]:=D[u,{z,2}];
```

Note that we use the delayed assignment `SetDelayed(:=)` to define these linear operators, and z is the independent variable in the package.

Without loss of generality, let us consider the case when $Pr = 7/10$, $Ec = 1/100$, $P = -5$, $b = 1$, $n = 1$, $Re = 1$, $M = 2$, $Gr = 5$, $h = 1$, $\lambda = 1$, $\lambda_T = 1$, and $\phi = \pi/6$. These physical parameters are input as

```
P = -5;  b = 1;
n = 1;   Ra = 1;
M = 2;   Gr = 5;
lamb = 1;  lambT = 1;
h = 1;  phi = Pi/6;
Pr = 7/10; Ec = 1/100;
```

At this time, we have input all the data for this problem, except the convergence-control parameters `c0[k]`. We minimize the averaged squared residual error of the 4th-order approximations to obtain optimal values for `c0[k]` by the command

```
GetOptiVar[4,{},{c0[1],c0[2],c0[3],c0[4]}];
```

Note that the second parameter of `GetOptiVar` is an empty list, which means that we give no constraint on the convergence-control parameters `c0[k]`.

After some time, we obtain the optimal values for `c0[k]`, which reads `c0[1]` ≈ -0.898166, `c0[2]` ≈ -0.946828, `c0[3]` ≈ -0.780946 and `c0[4]` ≈ -1.12363. Then we call the main module `BVPh` to get the 30th-order approximations

```
BVPh[1, 30];
```

The 30th-order approximations for u_1, θ_1, u_2, θ_2 are stored in `U[1,30]`, `U[2,30]`, `U[3,30]` and `U[4,30]`, respectively, while the corresponding averaged squared residual error of the system is `ErrTotal[30]`. The 30th-order approximations are plotted in Fig. 9.10. The value of $\theta(y)$ agrees with Umavathi's result [16] (black dots), as shown in Fig. 9.10. The 30th-order approximation of $\theta(y)$ gives the heat transfer rate $Nu_+ = \theta_1'(1) = 0.8860625$ and $Nu_- = \theta_2'(1) = 1.122312$, which agrees with $Nu_+ = 0.88606$ and $Nu_- = 1.12230$ in Umavathi's [16] Table 3. The total error `ErrTotal[k]` of the system for every two order of approximations is plotted in Fig. 9.11.

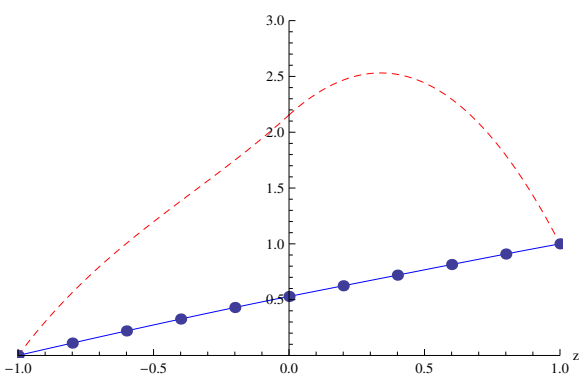

Fig. 9.10. The curve of $u(y)$ (solid) and $\theta(y)$ (dashed) for Example 4. The black dots are the values for $\theta(y)$ obtained by Umavathi [16].

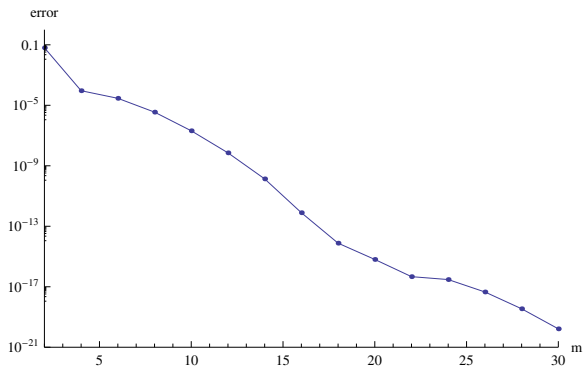

Fig. 9.11. Total error vs. order of approximation for Example 4.

9.7.5. *Example 5: Iterative solutions of the Gelfand equation*

When the problem is defined in a finite interval, the `BVPh 1.0` can solve it using an iterative approach. The `BVPh 2.0` has inherited this feature. However, there are some minor differences in the input.

Consider the Gelfand equation [17–19]

$$u'' + (K-1)\frac{u'}{z} + \lambda e^u = 0,\ u'(0) = u(1) = 0, \tag{9.56}$$

where the prime denotes the differentiation with respect to z, $K \geq 1$ is a constant, $u(z)$ and λ denote eigenfunction and eigenvalue, respectively. Following Liao [14] , an additional boundary condition

$$u(0) = A \tag{9.57}$$

is added to distinguish different eigenfunctions.

To solve this problem by `BVPh 2.0`, we have to input the differential equations, boundary conditions, and initial guesses. Since the problem is a single ODE with an unknown λ to be determined, we set `NumEQ = 1` and `TypeEQ=2`. The differential equation can be coded as follows

```
TypeEQ = 2;
NumEQ = 1;
f[1,z_,{u_},lambda_] :=
D[u,{z,2}] +(K-1)*D[u,z]/z+lambda*Exp[u];
```

The three boundary conditions, including the additional condition (9.57), are defined as

```
NumBC = 3;
BC[1, z_, {u_}] := (u-A)/. z -> 0;
BC[2, z_, {u_}] := D[u,z]/. z -> 0;
BC[3, z_, {u_}] := u /. z -> 1;
```

Now let us input the solution intervals

```
zL[1] = 0;    zR[1] = 1;
```

Since the solution interval is finite, we do not have to specify the integral interval to compute the averaged squared residual error.

The initial guess is chosen as $U_0 = \frac{A}{2}[1 + \cos(\pi z)]$, which is input into Mathematica as

```
U[1,0] = A/2*(1 + Cos[Pi*z]);
```

The auxiliary linear operator is chosen as $\mathcal{L} = \frac{\partial^2}{\partial z^2} + \left(\frac{\pi}{a}\right)^2$, which is defined in Mathematica as

```
L[1,f_] :=  D[f,{z,2}]+Pi^2*f;
```

Note that we use the delayed assignment `SetDelayed(:=)` to define the linear operator.

Without loss of generality, let us consider the case when $A = 1$ and $K = 2$. The physical parameters are input as

```
K = 2;    A = 1;
```

Because we want to approximate the right-hand sides using the hybrid-base function and use an iterative approach to get the approximations, the control parameters in `BVPh 2.0` are modified to

```
TypeL       =  2;
HYBRID      =  1; (* hybrid-base functions *)
TypeBase    =  2; (* even Fourier series   *)
ApproxQ     =  1;
Ntruncated  =  30;
```

Here `TypeL = 2`, `HYBRID = 1` and `ApproxQ = 1` together mean that the right-hand sides of all high-order deformation equations is approximated by the hybrid-base approximations. `TypeBase = 2` means the even expression (9.26) is used (`TypeBase = 1` also applies to this problem). `Ntruncated = 20` means $N_t = 30$.

At this time, we have input all the data for this problem, except the convergence-control parameter `c0[1]`. To get optimal `c0[1]`, we minimize the averaged squared residual error of the 6th-order approximations. This is done in `BVPh 2.0` by calling the module `GetOptiVar`

```
GetOptiVar[6, {}, {c0[1]}];
```

After some computation, we get the optimal value for the convergence-control parameter `c0[1]`$= -0.522418\cdots$. Now we can use the 3rd-order iteration HAM approach

```
iter[1,6,3]
```

to get the desired approximation. Here 6 means the iteration times. After about 40 seconds, the 6th iteration gives the eigenvalue 1.90921, which is the same with Liao's result [14]. The kth iteration approximations of u and λ are stored in `V[1,k]`, and `LAMBDA[k]`, while the corresponding averaged squared residual error is stored in `ERRTotal[k]`. The 6th iteration approximation is plotted in Fig. 9.12 by

```
Plot[V[1,6],{z,0,1},AxesLabel->{"z", "u(z)"}]
```

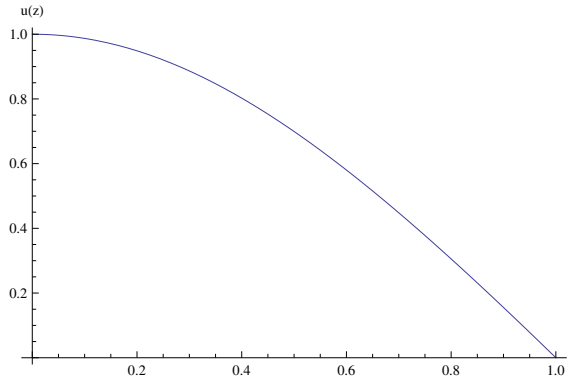

Fig. 9.12. The curve of the eigenfunction $u(z)$ corresponding to the eigenvalue $\lambda = 1.90921$ when $A = 1$ and $K = 2$ for Example 5.

The total error `ERRTotal[k]` of the problem for each iteration is plotted in Fig. 9.13 by the command

```
ListLogPlot[Table[{i, ERRTotal[i]}, {i, 1, 6}],
PlotRange->{{1,6},{10^-10,0.01}},Joined->True, Mesh->All,
AxesLabel->{"m","error"}]
```

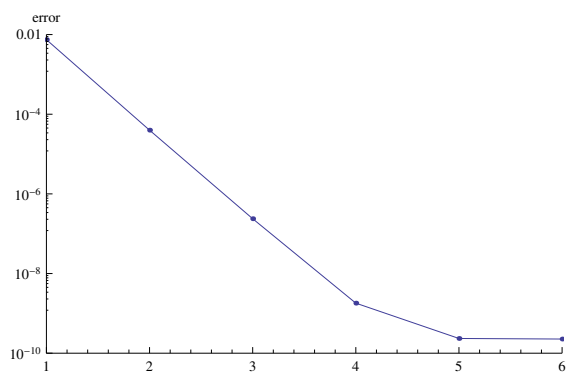

Fig. 9.13. Total error for each iteration vs. iteration times m for Example 5.

9.8. Conclusions

The homotopy analysis method (HAM) has been successfully applied to solve lots of nonlinear problems in science and engineering. Based on the HAM, the Mathematica package `BVPh 1.0` was issued by Liao in May, 2012. The aim of `BVPh` is to provide an analytic tool for as many nonlinear BVPs as possible in the frame work of the HAM.

However, `the BVPh 1.0` can only deal with problems of a single ODE. Unlike `BVPh 1.0`, the new version `BVPh 2.0` works for many types of systems of coupled nonlinear ODEs. In this chapter, we briefly describe how to install and use the `BVPh 2.0`. Five typical examples are employed to demonstrate the validity of `BVPh 2.0`, including a system of two coupled ODEs in finite interval, a system of two coupled ODEs in semi-infinite interval, a system of two coupled ODEs with algebraic property at infinity, a system of three coupled ODEs with an unknown parameter to be determined, and a system of four coupled ODEs in different intervals. Besides, new algorithms are used in some modules of `BVPh 2.0`. Hence, `BVPh 2.0` is much faster than `BVPh 1.0` in most cases. The package `BVPh 2.0` and all input data for these examples are free available online at `http://numericaltank.sjtu.edu.cn/BVPh.htm`.

It is well known that the iterative method can gain accurate approximations more efficiently by means of the HAM. The `BVPh 2.0` has also inherited the feature of `BVPh 1.0` to solve the problems in finite interval using the iterative method. For problems in semi-infinite interval, an iterative method for two typical kinds of based functions is proposed [20]. This iterative approach will be employed in the future version of `BVPh`.

Appendix A. Codes for examples

Here are the input data for all the examples in this chapter. Note that the listings without an end-of-line semicolon is wrapped to fit the page width. However, if you break the long command intentionally in Mathematica, it will run as multi-line commands and may not work as you expected.

A.1. Sample codes to run the illustrative example

```
(*           Filename: runIllustrative.nb           *)

(* 1. Clear all global variables                    *)
ClearAll["Global`*"];

(* 2. Read in the package BVPh 2.0                  *)
<< "E:\\Package\\BVPh2_0.m"

(* 3. Set the current working directory to          *)
(*    "the current directory"                       *)
SetDirectory[ToFileName[Extract["FileName" /.
NotebookInformation[EvaluationNotebook[]], {1},
FrontEnd`FileName]]];

(* 4. Read in your input data in current directory  *)
(* Note that the two files runIllustrative.nb and   *)
(* Illustrative.m are in the current directory      *)
<< Illustrative.m
```

A.2. Input data of BVPh 2.0 for the illustrative example

```
(*           Filename: Illustrative.m               *)
Print["The input file ",$InputFileName," is loaded !"];

(* Modify  control parameters in BVPh if necessary  *)

(*           Define the governing equation          *)
TypeEQ = 1;
NumEQ = 2;
f[1,z_,{f_,g_},Lambda_]:=
    D[f,{z,3}]-D[f,z]^2+f*D[f,{z,2}]+2*la*g+
    beta*(2*f*D[f, z]*D[f,{z,2}]-f^2*D[f,{z,3}]);
f[2,z_,{f_,g_},Lambda_]:=
```

```
    D[g,{z,2}]-D[f,z]*g+f*D[g,z]-2*la*D[f,z]+
    beta*(2*f*D[f, z]*D[g, z]-f^2*D[g,{z, 2}]);

(*              Define Boundary conditions          *)
NumBC = 5;
BC[1,z_,{f_,g_}]:=(D[f, z]-1)/.z->0;
BC[2,z_,{f_,g_}]:=f/.z->0;
BC[3,z_,{f_,g_}]:=g/.z->0;
BC[4,z_,{f_,g_}]:=D[f,z]/.z->infinity;
BC[5,z_,{f_,g_}]:=g/.z->infinity;

(* solution interval and integral interval for error *)
zL[1] = 0;
zR[1] = infinity;
zL[2] = 0;
zR[2] = infinity;
zRintegral[1] = 10;
zRintegral[2] = 10;

(*               Define initial guess               *)
U[1,0] = 1-Exp[-z];
U[2,0] = ahpha*z*Exp[-z];

(*        Define the auxiliary linear operator      *)
L[1,u_] := D[u,{z,3}]-D[u,z];
L[2,u_] := D[u,{z,2}]-u;

(*              Define physical parameters          *)
beta = 1/5;
la = 1/10;

(*                  Print input data                *)
PrintInput[{f[z],g[z]}];

(*                  Get optimal c0                  *)
```

```
GetOptiVar[3, {}, {c0[1],c0[2],alpha}];

(*          Gain 10th-order HAM approximation       *)
BVPh[1, 10];
```

A.3. Input data of BVPh 2.0 for Example 1

```
(*             Filename: Example1.m                 *)
Print["The input file ",$InputFileName," is loaded !"];

(*  Modify control parameters in BVPh if necessary  *)
ErrReq=10^-30;

(*           Define the governing equation          *)
TypeEQ = 1;
NumEQ = 2;
f[1,z_,{f_,g_},Lambda_]:=(1+K)*D[f,{z,4}]
   -Rey*M*D[f,{z,2}]+2*Rey*f*D[f,{z,3}]-K*D[g,{z,2}];
f[2,z_,{f_,g_},Lambda_]:=(1+K/2)*D[g,{z,2}]
   -Rey*K*(2*g-D[f,{z,2}])+Rey*(2*f*D[g,z]-D[f,z]*g);

(*              Define Boundary conditions          *)
NumBC = 6;
BC[1,z_,{f_,g_}]:=f/.z->0;
BC[2,z_,{f_,g_}]:=f/.z->1;
BC[3,z_,{f_,g_}]:=(D[f, z]-1)/.z->1;
BC[4,z_,{f_,g_}]:=D[f,{z,2}]/.z->0;
BC[5,z_,{f_,g_}]:=g/.z->1;
BC[6,z_,{f_,g_}]:=g/.z->0;

(* solution interval and integral interval for error *)
zL[1]=0;
zR[1]=1;
zL[2]=0;
zR[2]=1;
```

```
(*               Define initial guess               *)
U[1,0] = (z^3-z)/2;
U[2,0] = 0;

(*        Define the auxiliary linear operator      *)
L[1,u_]:=D[u,{z,4}];
L[2,u_]:=D[u,{z,2}];

(*              Define physical parameters          *)
Rey = M = 2;
K = 1/2;

(*                   Print input data               *)
PrintInput[{f[z],g[z]}];

(*                   Get optimal c0                 *)
GetOptiVar[4,{},{c0[1],c0[2]}];

(*          Gain 20th-order HAM approximation       *)
BVPh[1, 20];
```

A.4. Input data of BVPh 2.0 for Example 2

```
(*             Filename: Example2.m                 *)
Print["The input file ",$InputFileName," is loaded !"];

(*  Modify control parameters in BVPh if necessary  *)

(*          Define the governing equation          *)
TypeEQ = 1;
NumEQ = 2;
f[1,z_,{F_,S_},Lambda_]:=la^2*D[F,{z,2}]+S-F^2;
f[2,z_,{F_,S_},Lambda_]:=la^2*D[S,{z, 2}]-3*sigma*F*S;
```

```
(*            Define Boundary conditions            *)
NumBC = 4;
BC[1,z_,{F_, S_}] := F /. z -> 1;
BC[2,z_,{F_, S_}] := (G - 1) /. z -> 1;
BC[3,z_,{F_, S_}] := F /. z -> infinity;
BC[4,z_,{F_, S_}] := G /. z -> infinity;

(* solution interval and integral interval for error *)
zL[1] = 1;
zR[1] = infinity;
zL[2] = 1;
zR[2] = infinity;
zRintegral[1] = 10;
zRintegral[2] = 10;

(*             Define initial guess                 *)
U[1, 0] = gamma*(z^(-2) - z^(-3));
U[2, 0] = z^(-4);

(*      Defines the auxiliary linear operator       *)
L[1, u_] := D[u, {z, 2}]*z/3 + D[u, z];
L[2, u_] := D[u, {z, 2}]*z/5 + D[u, z];

(*      Define physical and control parameters      *)
sigma = 1;
gamma = 3;
la = 1/3;
c0[1] = -1/2;
c0[2] = -1/2;

(*                Print input data                  *)
PrintInput[{f[z], g[z]}];

(*       Gain 20th-order HAM approximation          *)
BVPh[1, 20];
```

A.5. Input data of BVPh 2.0 for Example 3

```
(*           Filename: Example3.m                    *)
Print["The input file ",$InputFileName," is loaded !"];

(* Modify   control parameters in BVPh if necessary *)

(*           Define the governing equation          *)
TypeEQ = 2;
NumEQ = 3;
f[1,z_,{f_,g_,s_},sigma_] :=
  D[f,{z,2}]+Gr*Pr*g-Nr*s+sigma;
f[2,z_,{f_,g_,s_},sigma_] :=
  D[g,{z,2}]+Nb*D[g,z]*D[s,z]+Nt*(D[g,z])^2-f;
f[3,z_,{f_,g_,s_},sigma_] :=
  D[s,{z,2}]+Nt/Nb*D[f,{z,2}]-Le*f;

(*            Define boundary conditions            *)
NumBC = 7;
BC[1,z_,{f_,g_,s_}] :=f/.z->-1;
BC[2,z_,{f_,g_,s_}] :=f/.z->1;
BC[3,z_,{f_,g_,s_}] :=g/.z->-1;
BC[4,z_,{f_,g_,s_}] :=g/.z->1;
BC[5,z_,{f_,g_,s_}] :=s/.z->-1;
BC[6,z_,{f_,g_,s_}] :=s/.z->1;
BC[7,z_,{f_,g_,s_}] :=Integrate[f,{z,0,1}]-Ra*Pr;

(*             Define solution interval             *)
zL[1] = -1;
zR[1] = 1;
zL[2] = -1;
zR[2] = 1;
zL[3] = -1;
zR[3] = 1;
```

```
(*       Defines the auxiliary linear operator      *)
L[1, u_] := D[u, {z, 2}];
L[2, u_] := D[u, {z, 2}];
L[3, u_] := D[u, {z, 2}];

(*             Define physical parameters           *)
Nr = 1/5;
Nt = 1/20;
Nb = 1/20;
Le = 10;
Gr = 5;
Pr = 1;
Ra = 5;

(*              Define initial guess                *)
U[1,0]=eps1-3/2*(-25+4eps1)z^2+5/2*(-15+2eps1)*z^4;
U[2,0]=eps2*(1 - z^2);
U[3,0]=eps3*(1 - z^2);

(*                Print input data                  *)
PrintInput[{f[z], g[z], s[z]}];

(*    Get optimal convergence-control parameters    *)
c0[1] = c0[2] = c0[3] = h;
GetOptiVar[3, {}, {eps1, eps2, eps3, h}];

(*         Gain 10th-order HAM approximation        *)
BVPh[1, 20];
```

A.6. Input data of BVPh 2.0 for Example 4

```
(*           Filename: Example4.m                   *)
Print["The input file ",$InputFileName," is loaded !"];

(*  Modify control parameters in BVPh if necessary  *)
```

```
(*        Define the governing equation             *)
TypeEQ = 1;
NumEQ = 4;
f[1,z_,{u1_,s1_,u2_,s2_},Lambda_]:=
D[u1, {z, 2}]+ Gr/Ra*Sin[phi]*s1 - P ;
f[2,z_,{u1_,s1_,u2_,s2_},Lambda_]:=
D[s1, {z, 2}] + Pr*Ec*(D[s1, z])^2;
f[3,z_,{u1_,s1_,u2_,s2_},Lambda_]:=D[u2,{z, 2}]
-h^2/lamb*P+Gr/Ra*Sin[phi]*n*b*h^2/lamb*s2
-M^2*h^2/lamb*u2;
f[4,z_,{u1_,s1_,u2_,s2_},lambda_]:=D[s2,{z,2}]
+Pr*Ec*lamb/lambT*D[u2,z]^2+Pr*Ec*h^2/lambT*M^2*u2^2;

(*           Define Boundary conditions             *)
    NumBC=8;
    BC[1,z_,{u1_,s1_,u2_,s2_}]:=(u1-1)/.z->1;
    BC[2,z_,{u1_,s1_,u2_,s2_}]:=(u1-u2)/.z->0;
    BC[3,z_,{u1_,s1_,u2_,s2_}]:=u2/.z->-1;
    BC[4,z_,{u1_,s1_,u2_,s2_}]:=
        (D[u1,z]-D[u2,z]/m/h)/.z->0;
    BC[5,z_,{u1_,s1_,u2_,s2_}]:=(s1-1)/.z->1;
    BC[6,z_,{u1_,s1_,u2_,s2_}]:=(s1-s2)/.z->0;
    BC[7,z_,{u1_,s1_,u2_,s2_}]:=s2/.z->-1;
    BC[8,z_,{u1_,s1_,u2_,s2_}]:=
        (D[s1,z]-D[s2,z]/K/h)/.z->0;

(*              Define solution interval            *)
    zL[1]=0; (* u1 *)
    zR[1]=1;
    zL[2]=0; (* s1 *)
    zR[2]=1;
    zL[3]=-1;(* u2 *)
    zR[3]=0;
    zL[4]=-1;(* s2 *)
```

```
    zR[4]=0;

(*                 Define initial guess              *)
    U[1, 0] = (z - z^2)*lamb/h+1;           (* u1 *)
    U[2, 0] = z*lambT/h +(1-lambT/h)*z^2;   (* s1 *)
    U[3, 0] = 1 + z;                        (* u2 *)
    U[4, 0] = z^2 + z;                      (* s2 *)

(*       Define the auxiliary linear operator       *)
    L[1,u_]:=D[u,{z,2}];
    L[2,u_]:=D[u,{z,2}];
    L[3,u_]:=D[u,{z,2}];
    L[4,u_]:=D[u,{z,2}];

(*           Define physical parameters             *)
    P = -5;  b = 1;
    n = 1;   Ra = 1;
    M = 2;   Gr = 5;
    lamb = 1;  lambT = 1;
    h = 1;  phi = Pi/6;
    Pr = 7/10; Ec = 1/100;

(*                  Print input data                *)
PrintInput[{u1[z], s1[z], u2[z], s2[z]}];

(*                  Get optimal c0                  *)
GetOptiVar[4,{},{c0[1],c0[2],c0[3],c0[4]}]

(*       Gain 10th-order HAM approximation          *)
BVPh[1,30];
```

A.7. Input data of BVPh 2.0 for Example 5

```
(*          Filename: Example5.m                    *)
Print["The input file ",$InputFileName," is loaded !"];
```

```
(* Modify  control parameters in BVPh if necessary  *)
TypeL       =  2;
HYBRID      =  1; (* hybrid-base functions *)
TypeBase    =  2; (* even Fourier series   *)
ApproxQ     =  1;
Ntruncated  =  20;

(*        Define the governing equation             *)
TypeEQ = 2;
NumEQ = 1;
f[1,z_,{u_},lambda_] :=
  D[u,{z,2}] +(K-1)*D[u,z]/z+lambda*Exp[u];

(*           Define Boundary conditions             *)
NumBC = 3;
BC[1, z_, {u_}] := (u-A)/. z -> 0;
BC[2, z_, {u_}] := D[u,z]/. z -> 0;
BC[3, z_, {u_}] := u /. z -> 1;

(*              Define solution interval            *)
zL[1] = 0;
zR[1] = 1;

(*                 Define initial guess             *)
U[1,0] =  A/2*(1 + Cos[Pi*z]);

(*        Define the auxiliary linear operator      *)
L[1,f_] :=  D[f,{z,2}]+Pi^2*f;

(*           Define physical parameters             *)
K  =  2;
A=1;

(*                Print input data                  *)
```

```
PrintInput[{u[z]}];

(*                 Get optimal c0                    *)
GetOptiVar[6,{},c0[1]];

(*                 Print input data                  *)
PrintInput[{u[z]}];

(*         Use 3rd-order iteration approach         *)
iter[1,6,3]
```

Acknowledgment

This work is partly supported by National Natural Science Foundation of China under Grant No. 11272209 and State Key Laboratory of Ocean Engineering under Grant No. GKZD010056.

References

[1] L. F. Shampine, I. Gladwell and S. Thompson, *Solving ODEs with MATLAB.* Cambridge University Press, Cambridge (2003).
[2] J. Kierzenka, L. F. Shampine, A BVP solver based on residual control and the Matlab PSE, *ACM TOMS.* **27**(3), 299–316 (2001).
[3] Z. Battles and L. N. Trefethen, An extension of Matlab to continuous functions and operators, *SIAM J. Sci. Comput.* **25**(5), 1743–1770 (2004).
[4] L. N. Trefethen, Computing numerically with functions instead of numbers, *Math. Comput. Sci.* **1**, 9–19 (2007).
[5] Y. P. Liu, S. J. Liao and Z. B. Li, Symbolic computation of strongly nonlinear periodic oscillations, *J. Symb. Comput.* **55**, 72–95 (2013).
[6] M. Sajid, Z. Iqbal, T. Hayat and S. Obaidat, Series solution for rotating flow of an upper convected Maxwell fluid over a strtching sheet, *Commun. Theor. Phys.* **56**(4), 740–744 (2011).
[7] T. Hayat, M. Nawa and A. A. Hendi, Heat transfer analysis on axisymmetric MHD flow of a micropolar fluid between the radially stretching sheets, *J. Mech.* **27**(4), 607–617 (2011).
[8] S. J. Liao, Notes on the homotopy analysis method: Some definitions and theorems, *Commun. Nonlinear Sci. Numer. Simulat.* **14**(4), 983–997 (2009).
[9] S. J. Liao, *Proposed homotopy analysis techniques for the solution of nonlinear problem.* Ph.D. thesis, Shanghai Jiao Tong University (1992).
[10] S. J. Liao, A uniformly valid analytic solution of two-dimensional viscous flow over a semi-infinite flat plate, *J. Fluid Mech.* **385**, 101–128 (1999).

[11] S. J. Liao, On the analytic solution of magnetohydrodynamic flows of non-newtonian fluids over a stretching sheet, *J. Fluid Mech.* **488**, 189–212 (2003).
[12] S. J. Liao, Series solutions of unsteady boundary-layer flows over a stretching flat plate, *Stud. Appl. Math.* **117**(3), 239–263 (2006).
[13] S. J. Liao, *Beyond Perturbation—Introductioin to the Homotopy Analysis Method*. Chapman & Hall/CRC Press, Boca Raton (2003).
[14] S. J. Liao, *Homotopy Analysis Method in Nonlinear Differential Equations*. Springer-Verlag Press, New York (2012).
[15] H. Xu, T. Fan and I. Pop, Analysis of mixed convection flow of a nanofluid in a vertical channel with the Buongiorno mathematical model, *Int. Commun. Heat Mass.* **44**, 15–22 (2013).
[16] J. C. Umavathi, I. C. Liu and J. Prathap Kumar. Magnetohydrodynamic Poiseuille-Couette flow and heat transfer in an inclined channel, *J. Mech.* **26**(4), 525–532 (2010).
[17] J. P. Boyd, An analytical and numerical study of the two-dimensional Bratu equation, *J. Sci. Comput.* **1**(2), 183–206 (1986).
[18] J. Jacobsen and K. Schmitt, The Liouville-Bratu-Gelfand problem for radial operators, *J. Differ. Equations.* **184**, 283–298 (2002).
[19] J. S. McGough, Numerical continuation and the Gelfand problem, *Appl. Math. Comput.* **89**(1-3), 225–239 (1998).
[20] Y. L. Zhao, Z. L. Lin and S. J. Liao, An iterative analytical approach for nonlinear boundary value problems in a semi-infinite domain, *Comput. Phys. Commun.* **184**(9): 2136–2144 (2013).