User Guide to BVPh 2.0

Yinlong Zhao and Shijun Liao*
School of Naval Architecture, Ocean and Civil Engineering
Shanghai Jiao Tong University, Shanghai 200240, China

Abstract

The Mathematica package `BVPh` is a free/open source software based on homotopy analysis method (HAM) for boundary value problems (BVPs). This tutorial shows how to use its newest version `BVPh 2.0` to solve and plot the solutions of boundary value problems (BVPs) for systems of ordinary differential equations. The input data for all the examples in this tutorial are in the appendix and available online (`http://numericaltank.sjtu.edu.cn/BVPh.htm`).

1 Introduction

The homotopy analysis method (HAM) has been proposed by Liao[1–7] 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 and equation-type of linear sub-problems. It is found[5,6] 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.

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[6] in May 2012, which is 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,[6] the `BVPh 1.0` is valid for lots of nonlinear BVPs and thus is a useful tool in practice.

*Corresponding author. E-mail address: sjliao@sjtu.edu.cn

However, the BVPh 1.0 can only deal with BVPs of single ordinary differential equation (ODE), it can not solve systems of ODEs. Now the newest version of BVPh, the BVPh 2.0, can deal with many systems of coupled ordinary differential equations (ODEs) defined in finite an/or semi-infinite 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. In this tutorial, we will show the usage of BVPh 2.0 to solve different kinds of systems of ODEs, including a system of coupled ODEs in finite interval, a system of coupled ODEs in semi-infinite interval, a system of coupled ODEs with algebraic property at infinity, a system of ODEs with an unknown parameter to be determined and a system of ODEs in different intervals.

This tutorial is organized as follows. In Section 2, we show how to load the package BVPh 2.0. In Section 3, an illustrative example is taken to show the use of BVPh 2.0 in detail. In Section 4, we take a glance at the modules, input, output and control parameters in the package. More examples are given in Section 5 to show the usage of the package. In Section 6, some conclutions are given. The reader is suggested to learn the illustrative example in Section 3 first. If one is puzzled with some parameter, one is encouraged to search the parameter in Section 4, and read its description there.

It should be pointed out that all the examples in this tutorial have been solved analytically (by the HAM) and/or numerically before. The package BVPh 2.0 gives agreeable results when the same physical parameters are given. However, this tutorial focus on how to use the package BVPh 2.0 to solve the problem, the validness of the approximations is shown i) always in the error curve of the approximations—the squared residual error usually decrease to as low as 10^{-10} and decreases at least 6 orders of magnitude; ii) sometimes in the value of the physical quantity of interest; iii) rarely in comparison with the numerical results in the figure.

The package BVPh 2.0 is developed on 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, or it will not work.

2 Installation

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). 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 source 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. However, if you break a long command intentionally in Mathematica, it will run as multi-line commands and may 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 `BVPh2_0.m`, the input data `Example.m` and the notebook file `runExample.nb`. The last two lines read in the files `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 source 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 the file `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 obtain the approximations by `BVPh 2.0`.

3 Illustrative example

Consider a system of ODEs[8]

$$f''' - (f')^2 + ff'' + 2\lambda g + \beta[2ff'f'' - f^2 f'''] = 0, \tag{1}$$

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

subject to

$$f'(0) = 1, f(0) = 0, g(0) = 0, \tag{3}$$

$$f'(\infty) = 0, g(\infty) = 0, \tag{4}$$

where λ is the rotation parameter, β is the 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. It has been solved by the HAM in Ref. 8.

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 (1) and (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 (3) and (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 of the boundary conditions 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 Section 5). 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 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, or it will take a lot of computation time. For this problem, the solution interval for each equation and the integral interval for squared residual 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) end point of the solution interval for the k-th equation f[k,z,{f,g},Lambda]. And zLintegral[k] (or zRintegral[k]) is the left (or right) end point of the integral interval to compute the squared residual for the k-th 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 end point of the integral interval to a finite value. That's 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 k-th equation. Note that i) η is the independent variable in the differential equations (1) and (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 k-th 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 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 squared residual is minimized without any constraint. Here we add no constraints to minimize the squared residual. 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. 1.

The accuracy of the kth-order approximation is measured by the 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. 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.

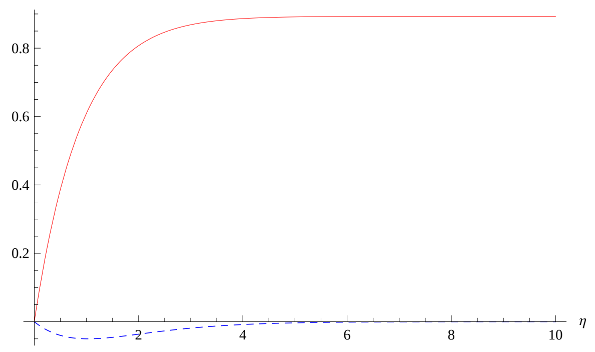

Fig. 1: The curve of $f(z)$ (solid) and $g(z)$ (dashed) for the illustrative example when $\beta = 1/5$, $\lambda = 1/10$.

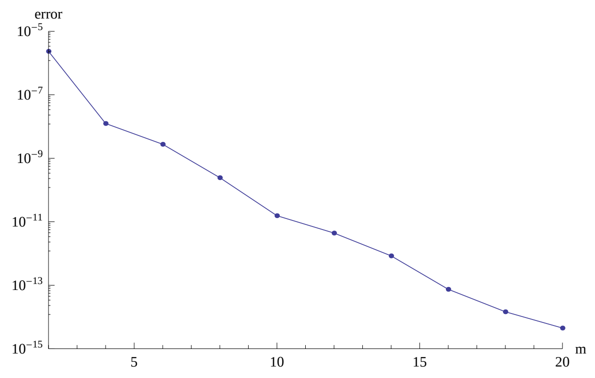

Fig. 2: Total error vs. order of approximation for the illustrative example when $\beta = 1/5$, $\lambda = 1/10$.

4 A glance at the BVPh 2.0

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

4.1 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`) to be determined or may not have an unknown parameter (when `TypeEQ=1`) to be determined . It is the basic module. For example, `BVPh[1,10]` gives the 1st to 10th-order homotopy-approximations. Thereafter, `BVPh[11,15]` further gives the 11th 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 an 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 squared residual of the system is less than a critical value `ErrReq`, whose default is 10^{-20}.

`GetErr` The module `GetErr[k_]` gives the squared residual 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 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.

4.2 Control parameters

`TypeEQ` A control parameter for the type of governing equations: `TypeEQ=1` corresponds 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 polynomial, or by the hybrid-base approximation. 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, `TypeBase=2` corresponds to the even Fourier approximation, 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 squared residual of governing equations to stop the computation. The default is 10^{-20}.

`NgetErr` A positive integer used in the module `BVPh`. The squared residual 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.

4.3 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 an 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 $u_{i,0}(z)$.
`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 end-point 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 end-point of the interval of the solution corresponding to the ith governing equation.
`zLintegral[i]`	The left end-point of the integral interval to compute the averaged squared residual of the ith governing equation. When the left end-point 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 end-point of the integral interval to compute the averaged squared residual of the ith governing equation. When the right end-point 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]`.

4.4 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-iteratioin 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 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 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 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 for each governing equation given by the kth-iteration homotopy-approximation (obtained by the iteration module `iter`).

4.5 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 vaiable 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.

5 More examples

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

5.1 Example 1: a system of ODEs in finite interval

Consider a system of coupled ODEs[9]

$$(1+K)f'''' - ReMf'' + 2Reff''' - Kg'' = 0, \tag{5}$$

$$(1+\frac{K}{2})g'' - ReK[2g - f''] + Re[2fg' - f'g] = 0, \tag{6}$$

$$\tag{7}$$

subject to

$$f(0) = 0,\ f(1) = 0,\ f'(1) = 1,\ f''(0) = 0, \tag{8}$$

$$g(1) = 0,\ g(0) = 0, \tag{9}$$

where K is the ratio of viscosities, Re is the Reynolds number and M is the Hartman number. Hayat[9] has solved this problem by the HAM.

Here we solve this problem by `BVPh 2.0`. Since there are two ODEs in system (5)–(6) 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 eight 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 in finite intervals, we do not have to specify the integral interval to compute the squared residual.

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[9] chose the convergence-control parameters `c0[1]=c0[2]=-0.7` through $\hbar$-curve. Here we minimize the 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 squared residual error 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. 3. This figure agrees with Hayat's[9] Fig. 9 and Fig. 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.[9] The total error of the system for every two order of approximations is plotted in Fig. 4 by the command

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

We can see from it that the error decreases beautifully.

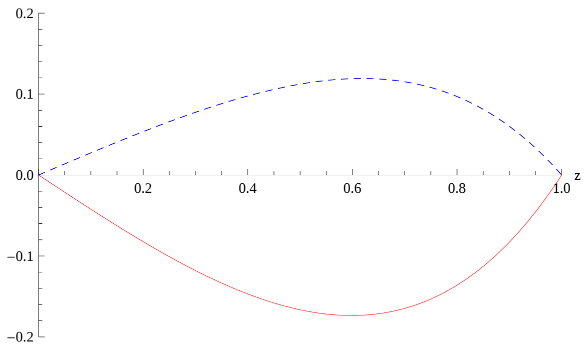

Fig. 3: The curve of $f(z)$ (solid), $g(z)$ (dashed) for Example 1.

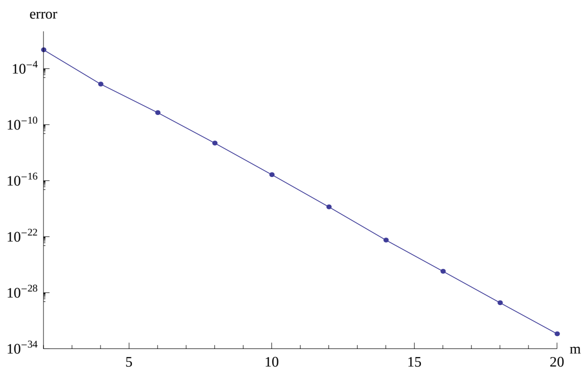

Fig. 4: Total error vs. order of approximation for Example 1.

5.2 Example 2: a system of coupled ODEs with algebraic property at infinity

Consider a set of two coupled nonlinear differential equations[5]

$$f'''(\eta) + \theta(\eta) - f'^2 = 0, \tag{10}$$

$$\theta''(\eta) = 3\sigma f'(\eta)\theta(\eta), \tag{11}$$

subject to

$$f(0) = f'(0) = 0,\ \theta(0) = 1,\ f'(+\infty) = \theta(+\infty) = 0, \tag{12}$$

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[5] employed the HAM to solve this system analytically. Now we use the `BVPh 2.0` to solve it.

Under the transformation

$$\xi = 1 + \lambda\eta,\ F(\xi) = f'(\eta),\ S(\xi) = \theta(\eta), \tag{13}$$

Eqs. (10) and (11) become

$$\lambda^2 F''(\xi) + S(\xi) - F^2(\xi) = 0, \tag{14}$$

$$\lambda^2 S''(\xi) = 3\sigma F(\xi)S(\xi), \tag{15}$$

subject to

$$F(1) = 0,\ S(1) = 1,\ F(+\infty) = S(+\infty) = 0. \tag{16}$$

Since there are two ODEs in system (14)–(15) 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 (16) 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 for computing 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 choosen 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 $\lambda = 1/3$. 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 approximations. 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 (14), (15) and (16). 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 Mathemcatica 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. 5. Here `trans={z->1+la*\[Eta]}` is the corresponding transformation, `\[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. 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.

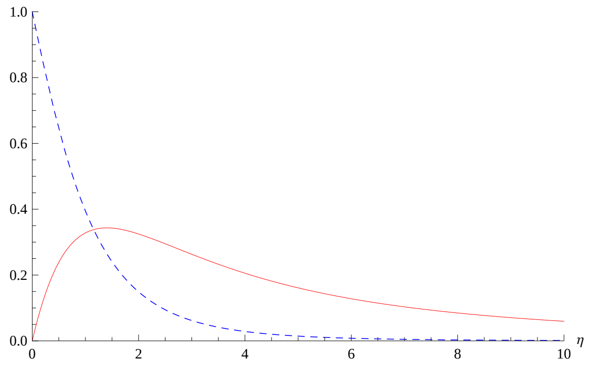

Fig. 5: The curve of $f'(\eta)$ (solid) and $\theta(\eta)$ (dashed) for Example 2.

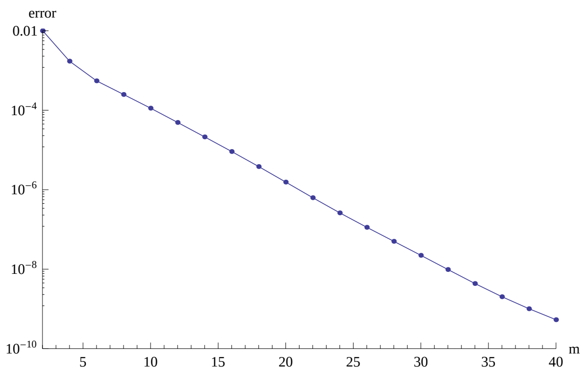

Fig. 6: Total error vs. order of approximation for Example 2.

The 40th-order approximations of $f''(0)$ and $g''(0)$ are 0.693268 and -0.769-879, 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 approximations 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}}
```

The comparison is shown in Fig. 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 efficent, 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.

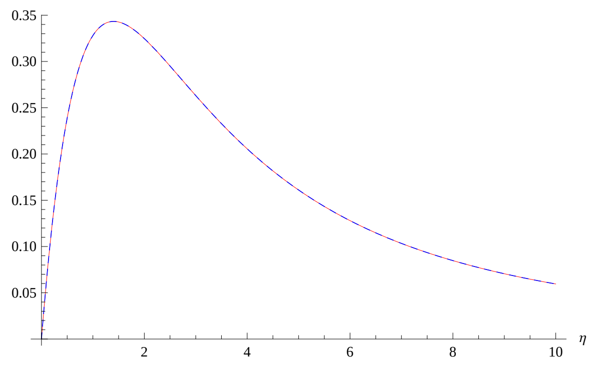

Fig. 7: The curve of 40th-order approximation of $f'(\eta)$ (solid) and $[20, 20]$ homotopy-Padé approximations of $f'(\eta)$ (dashed) for Example 2.

5.3 Example 3: a system of ODEs with an unknown parameter

Consider a system of ODEs[10]

$$U'' + (GrPr)\theta - N_r\phi + \sigma = 0, \tag{17}$$

$$\theta'' + N_b\theta'\phi' + N_t(\theta')^2 + N_b + N_t - U = 0, \tag{18}$$

$$\phi'' + \frac{N_t}{N_b}\theta'' - L_eU = 0, \tag{19}$$

subject to

$$U(-1) = U(1) = 0,\ \theta(-1) = \theta(1) = 0,\ \phi(-1) = \phi(1) = 0, \tag{20}$$

with an additional condition

$$\int_0^1 UdY = RePr, \tag{21}$$

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[10] solved this problem by the HAM.

Here we solve this problem by `BVPh 2.0`. Since there are three ODEs in system (17)–(19) 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 (21), 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 in finite intervals, we do not have to specify the integral interval to compute the 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 squared residual error of the 3th-order approximations to get the optimal values by the module `GetOptiVar` as follows

```
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. 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 squared residual error is `ErrTotal[20]`. `Lambda[19]` is about 18.272555944, which is the same with Xu's result.[10] The 20th-order approximations are plotted in Fig. 8. The total error of the system for every two order of approximations are plotted in Fig. 9.

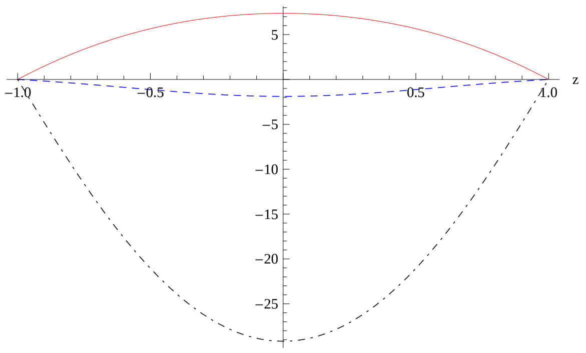

Fig. 8: The curve of U (solid), θ (dashed) and $\phi(z)$ (dot dashed) for Example 3.

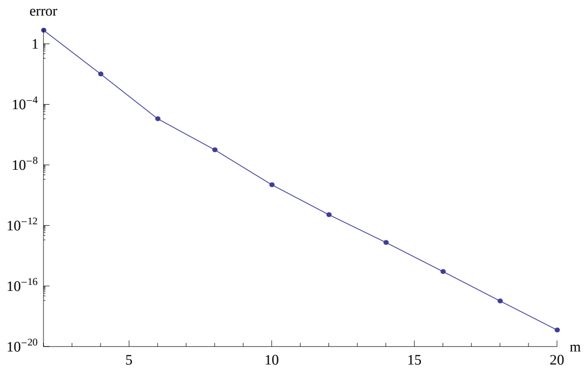

Fig. 9: Total error vs. order of approximation for Example 3.

5.4 Example 4: a system of ODEs in different intervals

Consider a two-phase flow[11]

(i) Region 1

$$\frac{d^2u_1}{dy^2} + \frac{Gr}{Re}\sin(\phi)\theta_1 = P, \tag{22}$$

$$\frac{d^2\theta_1}{dy^2} + PrEc\left(\frac{du_1}{dy}\right)^2 = 0, \tag{23}$$

(ii) Region 2

$$\frac{d^2u_2}{dy^2} + \frac{Gr}{Re}\frac{nbh^2}{\lambda}\sin(\phi)\theta_2 - \frac{M^2h^2}{\lambda}u_2 = \frac{h^2}{\lambda}P, \tag{24}$$

$$\frac{d^2\theta_2}{dy^2} + EcPr\frac{\lambda}{\lambda_T}\left(\frac{du_2}{dy}\right)^2 + EcPr\frac{h^2}{\lambda_T}M^2u_2^2 = 0, \tag{25}$$

subject to

$$u_1(1) = 1,\ \theta_1(1) = 1, \tag{26}$$

$$u_1(0) = u_2(0),\ \theta_1(0) = \theta_2(0), \tag{27}$$

$$u_1'(0) = \frac{\lambda}{h}u_2'(0),\ \theta_1'(0) = \frac{\lambda_T}{h}\theta_2'(0), \tag{28}$$

$$u_2(-1) = 0,\ \theta_2(-1) = 0, \tag{29}$$

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[11] investigate this model analytically by regular perturbation method and numerically by finite difference technique.

The BVPh 2.0 can solve this problem (22)–(29) 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 (26)–(29) 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 in finite intervals, we do not have to specify the integral interval to compute the 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 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 a empty list, which means that we give no constraints 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 squared residual error is `ErrTotal[30]`. The 30th-order approximations are plotted in Fig. 10. The value of $\theta(y)$ agrees with Umavathi's result[11] (black dots), as shown in Fig. 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[11] Table 3.

The total error of the system for every two order of approximations is plotted in Fig. 11.

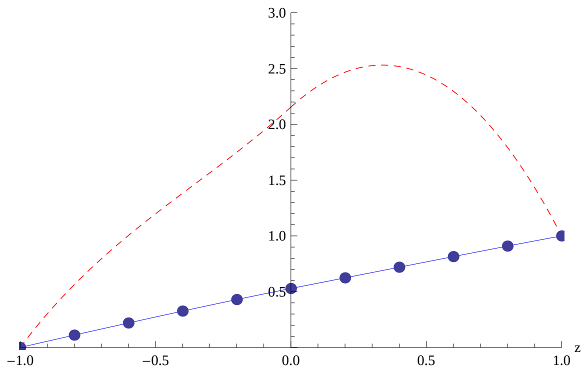

Fig. 10: The curve of $u(y)$ (solid) and $\theta(y)$ (dashed) for Example 4. The black dots are the values for $\theta(y)$ obtained by Umavathi.[11]

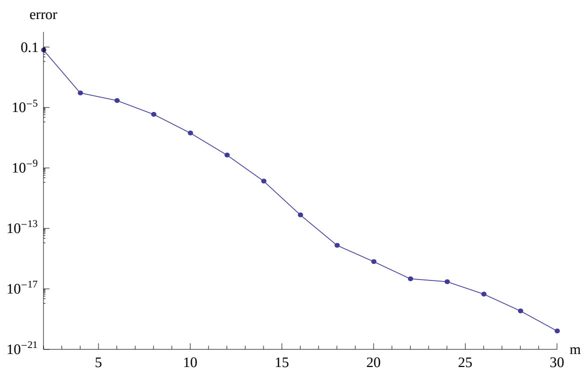

Fig. 11: Total error vs. order of approximation for Example 4.

5.5 Example 5: iterative solutions of the Gelfand equation

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

Consider the Gelfand equation[12–14]

$$u'' + (K-1)\frac{u'}{z} + \lambda e^u = 0,\ u'(0) = u(1) = 0, \tag{30}$$

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[6] , an additional boundary condition

$$u(0) = A \tag{31}$$

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 (31), 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 squared residual error.

The initial guess is chosen as $U_0 = \frac{A}{2}[1 + \cos(\pi z)]$, which is input 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 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 side term of all high-order deformation equations is approximated by the hybrid-base approximations. `TypeBase=2` means the even Fourier series 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 squared residual error of the 6th-order approximations. This is done in BVPh 2.0 by calling the function `GetOptiVar`

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

After some computation, we get the optimal value for the convergence-control parameter `c0[1]`$= -0.522418$. 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.[6] The kth iteration approximations of u and λ are stored in `V[1,k]`, and `LAMBDA[k]`, while the corresponding squared residual error is stored in `ERRTotal[k]`. The 6th iteration approximation is plotted in Fig. 12 by

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

The total error for each iteration is plotted in Fig. 13 by

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

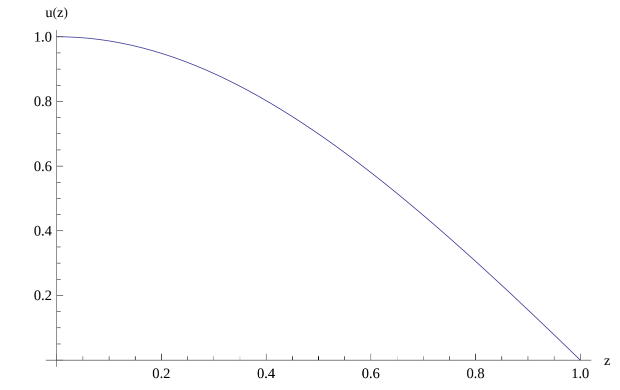

Fig. 12: The curve of the eigenfunction $u(z)$ corresponding to the eigenvalue $\lambda = 1.90921$ when $A = 1$ and $K = 2$ for Example 5.

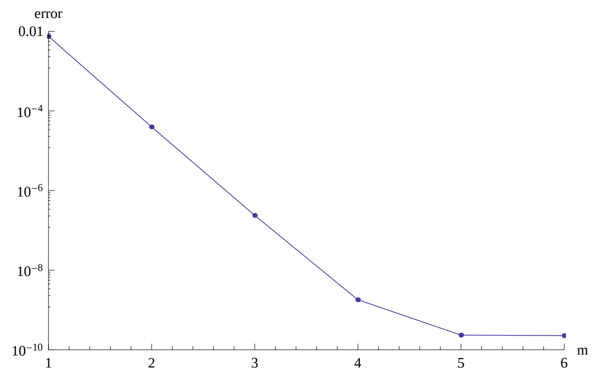

Fig. 13: Total error for each iteration vs. iteration times m for Example 5.

6 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 HAM.

However, the `BVPh 1.0` can only deal with problems of a single ODE. Its newest version `BVPh 2.0` can now deal with many systems of ODEs. This tutorial takes five examples to demonstrate the use of `BVPh 2.0`, including a system of coupled ODEs in finite interval, a system of coupled ODEs in semi-infinite interval, a system of coupled ODEs with algebraic property at infinity, a system of ODEs with an unknown parameter to be determined and a system of 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 `BVPh` is a free/open source software available at `http://numericaltank.sjtu.edu.cn/HAM.htm`

It is well-known that the iterative method can gain accurate approximations more efficiently by means of 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.[15] This method will be fulfilled in the future version of `BVPh`.

Appendix A. Codes for examples

Here are the input data for all the examples in this tutorial. 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  *)
<< Illustrative.m
```

A.2. Input data 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 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 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 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 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 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]
```

References

1. S. J. Liao, *Proposed homotopy analysis techniques for the solution of nonlinear problem.* Ph.D. thesis, Shanghai Jiao Tong University (1992).

2. 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).

3. 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).

4. S. J. Liao, Series solutions of unsteady boundary-layer flows over a stretching flat plate, *Stud. Appl. Math.*. **117**(3), 239–263 (2006).

5. S. J. Liao, *Beyond Perturbation—Introductioin to the Homotopy Analysis Method*. Chapman & Hall/CRC Press, Boca Raton (2003).

6. S. J. Liao, *Homotopy Analysis Method in Nonlinear differential equations.* Springer-Verlag Press, New York (2011).

7. S. J. Liao, Notes on the homotopy analysis method: Some definitions and theorems, *Commun. Nonlinear Sci. Numer. Simulat.*. **14**, 983–997 (2009).

8. 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).

9. 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).

10. 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).

11. 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).

12. J. P. Boyd, An analytical and numerical study of the two-dimensional Bratu equation, *J. Sci. Comput.*. **1**(2), 183–206 (1986).

13. J. Jacobsen and K. Schmitt, The Liouville-Bratu-Gelfand problem for radial operators, *J. Differ. Equations.* **184**, 283–298 (2002).

14. J. S. McGough, Numerical continuation and the Gelfand problem, *Appl. Math. Comput.*. **89**(1-3), 225–239 (1998).

15. 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.*. Online.