A Fortran 77 parallel implementation of the Newton-Raphson-Kantorovich method for two-point boundary value problem: User Guide

Luis Acevedo-Arreguin[1], Pascale Garaud[1] & Jean-Didier Garaud[2],
[1] Applied Mathematics and Statistics, Baskin School of Engineering, UC Santa Cruz
[2] ONERA, Paris

March 2, 2010

1 Description of the package

`NRK_ParaSol` is a parallel implementation of the commonly-used Newton-Raphson-Kantorovich (NRK) algorithm ([2],[3]) originally developed by Gough & Moore, which solves systems of I first order, nonlinear, coupled, ordinary differential equations (ODEs) in the two-point boundary value problem expressed as

$$\sum_{j=1}^{I} M_{ij}(x,\mathbf{y})\frac{\mathrm{d}y_j}{\mathrm{d}x} = f_i(x,\mathbf{y}) \text{ , where } i = \{1,2,\dots I\} \text{ ,} \tag{1}$$

with boundary conditions

$$\begin{aligned} g_k(x_A,\mathbf{y}) &= 0 \text{ , where } k = \{1,2,\dots k_A\} \text{ ,} \\ g_k(x_B,\mathbf{y}) &= 0 \text{ , where } k = \{k_A+1,\dots I\} \text{ ,} \end{aligned} \tag{2}$$

for the vector of dependent variables $\mathbf{y} = \{y_1(x), y_2(x), \dots y_I(x)\}$ on a discretized interval $[x_A, x_B] = \{x_1, x_2, \dots x_N\}$. Here, $x_A = x_1$ and $x_B = x_N$, k_A is the number of conditions set at the boundary located at x_A, whereas $k_B = I - k_A$ is the number of conditions set at the boundary located at x_B. The code implementation requires that (1) the mesh be either monotonically increasing or monotonically decreasing, and (2) the number of boundary conditions at the first meshpoint be greater than or equal to that at the second meshpoint[1]: $k_A \geq k_B$. Note that this version of the algorithm is second-order accurate in the spatial discretization.

1.1 Installation

To install the package, untar the file `NRK_ParaSol.tar`. This creates a directory structure with seven directories: five of them containing the examples provided in this guide, a `/templates` directory, and the directory `/docs` with the documentation.

[1]Note that if $k_B > k_A$, the user simply needs to reverse the mesh.

Each example contains the software organized in two folders: `/src` and `/workdir`.

- The folder `/src` contains the subroutines organized in two sub-folders: the solver routines `/src_solver_routines`, which should not be modified, and the user routines `/src_user_routines`, which can be tailored at will.
- The folder `/workdir` contains the initialization `.h` file, and a sample `Makefile` and `PBS` file. This folder also contains all the output files organized in various directories (see below).

The directory `/templates` is organized in the same way as the example directories, but the subroutines in `/src_user_routines` should be completed by the user as explained in Section 1.3.

1.2 Directory `Workdir`

The directory `/workdir` is where the code is executed. It contains both input and output files and directories as described below. The `Makefile` provided is generic and should be modified to include the user's version of Fortran (`FORTRAN`). A `pp.pbs` file is provided if necessary (for users on the Pleiades supercomputer at Santa Cruz for example), though the code can be run directly using the standard `mpirun` command. Note that the user should verify that the number of processors used in the `mpirun` command matches that of the `init_simu.h` file.

1.2.1 Input files

The main input parameters are entered into the `init_simu.h` file:

- `ii`, the number of ordinary differential equations,
- `ka`, the number of conditions at the boundary x_A,
- `kb`, the number of conditions at the boundary x_B,
- `linearlhs`, an optimization flag which is set to 1 if the coefficients of the left-hand-side matrix M_{ij} are all independent of $\mathbf{y}$.
- `nn`, the number of meshpoints,
- `xa`, the coordinate of the first boundary,
- `xb`, the coordinate of the second boundary,
- `nprev`, an indicator for the use of previous results as initial guess,
- `niter`, the maximum number of iterations to be attempted, and
- `nproc`, the number of processors.

The file `init_simu.h` can be modified to include any user-defined parameter if needed (see Example 4). All other input files/data should be stored for clarity in the directory `/workdir/inputfiles`.

1.2.2 Output files

The output files are organized in the following folders:

- `/guessd`: The NRK solver saves the initial guess in this folder.
- `/tempor`: The NRK solver saves the solution at each iteration in this folder.
- `/result`: The NRK solver saves the results in this folder once the accuracy criterion is satisfied.
- `/diagnosticfiles`: Solution errors are written in the file `ea.dat` included in this folder. The file `ea.dat` reports both the average error and the maximum error for each element of the computed vector $\mathbf{y}$.

1.3 User-defined subroutines

The directory `/src/src_user_routines` contains all user-modified routines. The driver routine is the `main.f`. Subroutines describing the ODEs and boundary conditions of the problem to be solved are `lhs.f`, `rhs.f`, and `bc.f`, respectively. In addition, the user should modify `mesh.f` where the mesh is created, `guess.f` where a guess is generated and `printresult.f` where results are printed to files.

- `main.f`: Driver routine.
- `rhs.f`: The user inputs the right hand side of equation (1), represented by the vector $\mathbf{f}$, into the array `f(i)`. The non-zero elements of the Jacobian, $\partial f_i/\partial y_j$, are input into the array `fd`(i,j). See examples for detail.
- `lhs.f`: The user inputs the non-zero elements of the matrix $\mathbf{M}$ corresponding to the left-hand-side of equation (1) through the function `am`(i,j). Likewise, the non-zero elements of the left-hand-side Jacobian $\partial M_{ij}/\partial y_k$ are input through the function `amd`(i,j,k). See examples for detail.
- `bc.f`: The user inputs the boundary conditions, defined by the vector $\mathbf{g}$, into the array `g(i)`. The user also inputs the non-zero elements of the Jacobian $\partial g_i/\partial y_j$ into the array `gs`(i,j). Finally, a permutation vector $\mathbf{v}$ is also defined in this subroutine to renumber the dependent variables to prevent formation of singular matrices. See Example 2b for detail.
- `mesh.f`: The user defines an array of meshpoints. A default file creating a linearly-spaced mesh is provided.
- `guess.f`: The user provides a trial solution either by writing a mathematical function, or by reading external files. A default file creating a constant initial guess is provided.
- `printresult.f`: The user specifies in this subroutine how the vector solution $\mathbf{y}$ is printed to files. A default file for printing is provided.

Some examples follow illustrating how to apply `NRK_ParaSol` under different circumstances.

2 Examples

2.1 Example 1

Let us consider first the following differential equation

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + \frac{\mathrm{d}y}{\mathrm{d}x} - 2y = e^x \tag{3}$$

on the interval $(0, 1)$, under the following boundary conditions

$$\begin{aligned} y(0) &= 1 \; , \\ y(1) &= 0 \; . \end{aligned} \tag{4}$$

This second-order equation can be rewritten as two first-order equations, hence $I = 2$. We define the parameters for this case in the file `init_simu.h` as follows:

```
c Initialization file for the specific simulations

c Initialization of parameters specific to the system of ODEs to solve
      integer ii,ka,kb
      parameter(ii = 2)  ! Number of equations
      parameter(ka = 1)  ! Number of boundary conditions at first meshpoint
      parameter(kb = 1)  ! Number of boundary conditions at second meshpoint
      integer linearlhs
      parameter(linearlhs=1) !If the lhs is linear then 1, otherwise 0

c Initialization of parameters specific to the mesh used
      integer nn
      parameter(nn = 1000)     !number of meshpoints
      double precision xa,xb  ! First and last meshpoint
      parameter(xa=0.d0)
      parameter(xb=1.d0)

c Initialization of relaxation parameters
      integer nprev
      parameter(nprev = 0)      !use previous guess (1) or not (0)
      double precision ucy,acy
      parameter(ucy = 1.d0)     ! convergence speed (must be le 1.0)
      parameter(acy = 1.d-16)   ! accuracy required
      integer niter
      parameter(niter = 10)     ! number of iterations to try.

c Initialization of quantities specific to the parallel implementaion
      integer nproc,nbppmax
c Number of processors :
      parameter(nproc=4)
c Maximum number of blocks per processor
      parameter(nbppmax = (nn+1)/nproc +1 )
```

Note that this input file also specifies the number of processors to be 4.

2.1.1 Case 1a

By defining

$$\begin{aligned} y_1 &= y \ , \\ y_2 &= \frac{\mathrm{d}y}{\mathrm{d}x} \ , \end{aligned} \tag{5}$$

we can rewrite equation (3) in the following way

$$\frac{\mathrm{d}}{\mathrm{d}x}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ 2y_1 - y_2 + e^x \end{pmatrix} \ , \tag{6}$$

from which we can obtain the expressions for the RHS functions:

$$\begin{aligned} f_1 &= y_2 \ , \\ f_2 &= 2y_1 - y_2 + e^x \ . \end{aligned} \tag{7}$$

The Jacobian matrix is

$$\begin{pmatrix} \frac{\partial f_1}{\partial y_1} = 0 & \frac{\partial f_1}{\partial y_2} = 1 \\ \frac{\partial f_2}{\partial y_1} = 2 & \frac{\partial f_2}{\partial y_2} = -1 \end{pmatrix} \ ,$$

for which only the non-zero terms need to be entered. Hence, the core part of the subroutine `rhs.f` is written as

```
f(1) = y(2)
fd(1,2) = 1.d0

f(2) = 2.d0*y(1)-y(2)+dexp(x)
fd(2,1) = 2.d0
fd(2,2) = -1.d0
```

The corresponding boundary conditions can be expressed within the subroutine `bc.f` as

$$\begin{aligned} g_1 &= y(x_A) - 1 = y_1(x_A) - 1 \ , \\ g_2 &= y(x_B) - 0 = y_1(x_B) - 0 \ . \end{aligned} \tag{8}$$

We define $y_1(x_A)$ as `ya(1)` and $y_2(x_A)$ as `ya(2)`. Similarly, $y_1(x_B)$ is `yb(1)` and $y_2(x_B)$ is `yb(2)`. Hence, these functions along with their corresponding derivatives are coded in the subroutine `bc.f` as

```
g(1) = ya(1)-1.d0
gs(1,1) = 1.d0

g(2) = yb(1)
gs(2,1) = 1.d0
```

In a similar fashion, the subroutine `lhs.f` contains the expressions for M_{ij} and $\partial M_{ij}/\partial y_k$. By contrast with `rhs.f` and `bc.f` these quantities are returned through function calls, M_{ij} in `am` and $\partial M_{ij}/\partial y_k$ in `amd`. This is done to ease the memory requirement for very large systems of ODEs. In this example, the M_{ij} matrix is unity. This can be input as:

```
am=0.d0
if(i.eq.j) am=1.d0
```

in the `am` function. The `amd` function needs not to be entered if M_{ij} is independent of $\mathbf{y}$ (i.e. if `linearlhs = 1`).

All the corresponding subroutines adapted for this example are included in the directory `/example1a/src/src_user_routines`.

2.1.2 Case 1b

Alternatively, we can express equation (3) as

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \frac{\mathrm{d}}{\mathrm{d}x} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ 2y_1 + e^x \end{pmatrix} , \tag{9}$$

from which we can obtain the expressions for the subroutines `rhs.f`, `bc.f`, and `lhs.f` to be slightly different. In the case of `lhs.f`, for example, the nonzero elements of the matrix $\mathbf{M}$ are now input as

```
am=0.d0
if(i.eq.j) am=1.d0
if(i.eq.2 .and. j.eq.1) am=1.d0
```

The user may compare the subroutines `rhs.f`, `bc.f`, and `lhs.f` in the directory
`/example1b/src/src_user_routines`
with the corresponding subroutines at
`/example1a/src/src_user_routines`.

2.1.3 Result comparison

When the code for this example is executed, the exact solution is written along with the numerical one in the file `Y001.dat` both in the directory
`/example1a/workdir/result`
and
`/example1b/workdir/result`.

2.2 Example 2

NRK can also be used to find solutions to eigenvalue problems. Let us consider now the following eigenvalue differential equation

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + \omega^2 y = 0 \tag{10}$$

on the interval $(0, 1)$, under the following boundary conditions

$$\begin{aligned} y(0) &= 0 , \\ y(1) &= 0 , \\ \frac{\mathrm{d}y}{\mathrm{d}x}(0) &= 1 . \end{aligned} \tag{11}$$

2.2.1 Case 2a

By defining

$$\begin{aligned} y_1 &= y \ , \\ y_2 &= \frac{\mathrm{d}y}{\mathrm{d}x} \ , \\ y_3 &= \omega \ , \end{aligned} \tag{12}$$

we can rewrite equation (10) in the following way

$$\frac{\mathrm{d}}{\mathrm{d}x}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_2 \\ -y_3^2 y_1 \\ 0 \end{pmatrix} \tag{13}$$

from which we can obtain the expressions for the subroutine `rhs.f`:

$$\begin{aligned} f_1 &= y_2 \ , \\ f_2 &= -y_3^2 y_1 \ , \\ f_3 &= 0 \ . \end{aligned} \tag{14}$$

Hence, the core part of the subroutine `rhs.f` is written as

```
f(1) = y(2)
fd(1,2) = 1.d0

f(2) = -y(3)**2*y(1)
fd(2,1) = -y(3)**2
fd(2,3) = -2*y(3)*y(1)

f(3) = 0.d0
```

Likewise, the corresponding boundary conditions can be expressed within the subroutine `bc.f` as

$$\begin{aligned} g_1 &= y(x_A) - 0 = y_1(x_A) = y_A(1) \ , \\ g_2 &= \frac{\mathrm{d}y}{\mathrm{d}x}(x_A) - 1 = y_2(x_A) - 1 = y_A(2) - 1 \ , \\ g_3 &= y(x_B) - 0 = y_1(x_B) = y_B(1) \ . \end{aligned} \tag{15}$$

These functions along with their corresponding derivatives are coded in the subroutine `bc.f` as

```
g(1) = ya(1)
gs(1,1) = 1.d0

g(2) = ya(2) - 1.d0
gs(2,2) = 1.d0

g(3) = yb(1)
gs(3,1) = 1.d0
```

Finally, the subroutine `lhs.f` contains the expressions for M_{ij} and $\partial M_{ij}/\partial y_j$, which reduce to `am=1` for the elements on the diagonal of the matrix $\mathbf{M}$. As in Example 1, `amd` needs not to be entered if the problem in `lhs.f` is linear.

Note that since this is an eigenvalue problem, we expect a number of solutions. Typically, different solutions are found starting from different initial guesses. The user can modify the subroutine `guess.f` to find solutions corresponding to different eigenvalues. We change the default values in `guess.f` to solve case 2b to obtain a specific eigenvalue solution. For example, if y_3 is set to 7 for all $x(i)$ in `guess.f`, then we get the eigenvalue y_3 closest to 7 and its corresponding solution y_1.

2.2.2 Case 2b

This example illustrates the use of the permutation vector. By defining

$$\begin{aligned} y_1 &= \omega \;, \\ y_2 &= y \;, \\ y_3 &= \frac{\mathrm{d}y}{\mathrm{d}x} \;, \end{aligned} \tag{16}$$

we can rewrite equation (10) in the following way

$$\frac{\mathrm{d}}{\mathrm{d}x}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ y_3 \\ -y_1^2 y_2 \end{pmatrix} \tag{17}$$

from which we can obtain the expressions for the subroutine `rhs.f`:

$$\begin{aligned} f_1 &= 0 \;, \\ f_2 &= y_3 \;, \\ f_3 &= -y_1^2 y_2 \;. \end{aligned} \tag{18}$$

Hence, the core part of the subroutine `rhs.f` is finally written as

```
f(2) = y(3)
fd(2,3) = 1.d0

f(3) = -y(1)**2*y(2)
fd(3,2) = -y(1)**2
fd(3,1) = -2*y(1)*y(2)

f(1) = 0.d0
```

Similarly the boundary conditions are now

```
g(1) = ya(2)
gs(1,2) = 1.d0

g(2) = ya(3) - 1.d0
gs(2,3) = 1.d0
```

```
      g(3) = yb(2)
      gs(3,2) = 1.d0
```

Note that in this case, the Jacobian matrix `gs` becomes

$$\texttt{gs}(I,I) \quad = \quad \left(\begin{array}{cc|c} \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{0} & \mathbf{0} & 1 \\ \hline 0 & 1 & 0 \end{array}\right) \tag{19}$$

We then see that the submatrix associated with the boundary conditions at x_A, namely

$$\texttt{gs}(k_A,k_A) \quad = \quad \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{20}$$

is singular. The matrix `gs(`k_A,k_A`)` must be non-singular for this parallel algorithm to work. Hence, we create a permutation of the columns of `gs(`I,I`)` (which is equivalent to renumbering the dependent variables) to make the new `gs(`k_A,k_A`)` non-singular:

$$\begin{pmatrix} \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{0} & \mathbf{0} & 1 \\ 0 & 1 & 0 \end{pmatrix} \xrightarrow{v^T=[2,3,1]} \begin{pmatrix} \mathbf{1} & 0 & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} \\ 1 & 0 & 0 \end{pmatrix} , \tag{21}$$

where `v(I)` is the permutation vector, whose subroutine is in the last part of the file `bc.f`:

```
      subroutine pervector(v)

c *******************************************************************
c Subroutine where the permutation vector for the boundary condition
c functions is input
c *******************************************************************

      implicit none
      include 'init_simu.h'

      integer v
      dimension v(ii)

      integer i

      v(1) = 2
      v(2) = 3
      v(3) = 1

      return
      end
```

Finally, the subroutine `lhs.f` contains the expressions for M_{ij} and $\partial M_{ij}/\partial y_j$, which reduce to `am=1` for the elements on the diagonal of the matrix $\mathbf{M}$.

2.3 Example 3

Let us consider now the Chebishev differential equation ([4]):

$$(1-x^2)\frac{\mathrm{d}^2y}{\mathrm{d}x^2} - x\frac{\mathrm{d}y}{\mathrm{d}x} + \omega^2 y = 0 \tag{22}$$

under the following boundary conditions

$$\begin{aligned} y(0) &= 0 \ , \\ y(1) &= 1 \ , \\ \frac{\mathrm{d}y}{\mathrm{d}x}(0) &= -3 \ . \end{aligned} \tag{23}$$

By defining

$$\begin{aligned} y_1 &= y \ , \\ y_2 &= \frac{\mathrm{d}y}{\mathrm{d}x} \ , \\ y_3 &= \omega \ , \end{aligned} \tag{24}$$

we can rewrite equation (22) in the following way

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & (1-x^2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \frac{\mathrm{d}}{\mathrm{d}x} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_2 \\ xy_2 - y_3^2 y_1 \\ 0 \end{pmatrix} \ , \tag{25}$$

from which we can obtain the expressions for the subroutine `rhs.f`:

$$\begin{aligned} f_1 &= y_2 \ , \\ f_2 &= xy_2 - y_3^2 y_1 \ , \\ f_3 &= 0 \ . \end{aligned} \tag{26}$$

The core part of the subroutine `rhs.f` is

```
f(1) = y(2)
fd(1,2) = 1.d0

f(2) = x*y(2)-y(3)**2.d0*y(1)
fd(2,1) = -y(3)**2.d0
fd(2,2) = x
fd(2,3) = -2.d0*y(3)*y(1)

f(3) = 0.d0
```

Likewise, the corresponding boundary conditions can be expressed within the subroutine `bc.f` as

$$\begin{aligned} g_1 &= y(x_A) - 0 = y_1(x_A) = y_A(1) \ , \\ g_2 &= \frac{\mathrm{d}y}{\mathrm{d}x}(x_A) + 3 = y_2(x_A) + 3 = y_A(2) + 3 \ , \\ g_3 &= y(x_B) - 1 = y_1(x_B) - 1 = y_B(1) - 1 \ . \end{aligned} \tag{27}$$

These functions along with their corresponding derivatives are coded in the subroutine `bc.f` as

```
       g(1) = ya(1)
       gs(1,1) = 1.d0

       g(2) = ya(2) + 3.d0
       gs(2,2) = 1.d0

       g(3) = yb(1) - 1.d0
       gs(3,1) = 1.d0
```

As in previous examples, the subroutine `lhs.f` contains the expressions for M_{ij} and $\partial M_{ij}/\partial y_j$. This example requires setting $M_{2,2} = (1 - x^2)$. This is done as:

```
      am=0.d0

      if(i.eq.j) am=1.d0
      if(i.eq.2 .and. j.eq.2) am=1.d0-x**2.d0
```

The exact solution is written along with the numerical one in the file `Y001.dat` in the directory `/example3/workdir/result`. The file `Y003.dat` in the same directory shows the eigenvalue numerically computed, and the eigenvalue which corresponds to the exact solution provided in the file `Y001.dat`.

2.4 Example 4

Let us consider now the van der Pol's differential equation ([1]):

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} - \epsilon(1-y^2)\frac{\mathrm{d}y}{\mathrm{d}x} + \frac{1}{4}u_0^2 y = 0 \qquad (28)$$

under the following boundary conditions

$$\begin{aligned} y(0) &= 0\ , \\ \frac{\mathrm{d}y}{\mathrm{d}x}(0) &= u_0\ . \end{aligned} \qquad (29)$$

In this example, the parameters u_0 and ϵ are entered in the `init_simu.h` file as

```
c Initialization of model-specific parameters
      double precision epsil,u0
      parameter(epsil=1.d-6)
      parameter(u0=1.d0)
```

By defining

$$\begin{aligned} y_1 &= y\ , \\ y_2 &= \frac{\mathrm{d}y}{\mathrm{d}x}\ , \end{aligned} \qquad (30)$$

we can rewrite equation (28) in the following way

$$\begin{pmatrix} 1 & 0 \\ -\epsilon(1-y_1^2) & 1 \end{pmatrix} \frac{\mathrm{d}}{\mathrm{d}x} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ \frac{1}{4}u_0^2 y_1 \end{pmatrix} , \tag{31}$$

from which we can obtain the expressions for the subroutine `rhs.f`:

$$\begin{aligned} f_1 &= y_2 , \\ f_2 &= \frac{1}{4}u_0^2 y_1 . \end{aligned} \tag{32}$$

The core part of the subroutine `rhs.f` is

```
f(1) = y(2)
fd(1,2) = 1.d0

f(2) = 0.25d0*u0**2.d0*y(1)
fd(2,1) = 0.25d0*u0**2.d0
```

Likewise, the corresponding boundary conditions can be expressed within the subroutine `bc.f` as

$$\begin{aligned} g_1 &= y(x_A) - 0 = y_1(x_A) = y_A(1) , \\ g_2 &= \frac{\mathrm{d}y}{\mathrm{d}x}(x_A) - u_0 = y_2(x_A) - u_0 = y_A(2) - u_0 . \end{aligned} \tag{33}$$

These functions along with their corresponding derivatives are coded in the subroutine `bc.f` as

```
g(1) = ya(1)
gs(1,1) = 1.d0

g(2) = ya(2) - u0
gs(2,2) = 1.d0
```

The subroutine `lhs.f` contains the expressions for M_{ij} and $\partial M_{ij}/\partial y_j$. This example no longer has a linear left-hand-side. Indeed,

$$M_{2,1} = -\epsilon(1-y_1^2) , \tag{34}$$

$$\frac{\partial M_{2,1}}{\partial y_1} = 2\epsilon y_1 , \tag{35}$$

As a result, we must set `linearlhs=0` in the file `/workdir/init_simu.h`. The LHS is then coded as

```
double precision function am(i,j,x,y,in)
implicit none
include 'init_simu.h'
```

```
      integer i,j,in
      double precision x,y
      dimension y(*)

      am=0.d0

      if(i.eq.j) am=1.d0
      if(i.eq.2 .and. j.eq.1) am=-epsil*(1.d0-y(j)**2)

      return
      end

c ***************************************************************
      double precision function amd(i,j,l,x,y,in)
      implicit none
      include 'init_simu.h'

      integer i,j,l,in
      double precision x,y
      dimension y(*)

      amd=0.d0
      if(i.eq.2 .and. j.eq.1 .and. l.eq.1) amd=2.d0*epsil*y(j)

      return
      end
```

The first-order approximate solution (for $\epsilon \to 0$) is written along with the numerical one in the file `Y001.dat` in the directory `/example4/workdir/result`.

References

[1] Fogiel, M.
The Differential equations Problem Solver
Research & Education Association, 1996, pp. 1261-1263

[2] Garaud and Garaud
Dynamics of the solar tachocline II. The stratified case
MNRAS 391, 1239-1258 (2008)

[3] Press, Teukolsky, Vetterling, and Flannery
Numerical Recipes
Cambridge University Press, 2007, pp. 964-970

[4] Rivlin, T. J.
The Chebyshev Polynomials
John Wiley & Sons, 1974, pp. 4, 31

[5] Weinberger, H. F.
A First Course in Partial Differential equations
John Wiley & Sons, 1965, pp. 120, 415