# BVP4C
# Automatic Solution of 1D Boundary Value Problems (IBVP's)

---

**BVP4C** is a directory of MATLAB programs which illustrate how to use the MATLAB command **bvp4c()**, which can solve boundary value problems (BVP's) in one spatial dimension.

## Example of a 1D Boundary Value Problem:

A simple two point boundary value problem involves a second degree differential equation

```
        y"(x) = f(x,y)
```

which is to hold over some interval

```
        a < x < b
```

with boundary conditions

```
        y(a) = ya
        y(b) = yb
```

## Defining an Initial Guess for the Solution

The solution to your boundary value problem begins by specifying an initial guess for the solution. If the problem is linear, or only mildly nonlinear, then a simple guess may be sufficient. For difficult problems, it may be necessary to exert some effort to preparing a good initial guess. The guess function is a MATLAB structure, which is typically defined by a call to **bvpinit()** as follows:

```
        solinit = bvpinit ( xinit, yinit, parameters )
```

where

- *xinit* is a vector of x values which begin with the left endpoint and conclude with the right endpoint.
- *yinit* is a vector of M values. The initial guess for the solution and its first M-1 derivatives will be constant functions with the values yinit(1) through yinit(M).

- *yinit* can instead be a function handle, in which case it must be written in such a way that it accepts a scalar value x, and returns a vector containing the M components of the initial guess at x.
- *parameters* is a vector of initial guesses for unknown parameters. This argument is optional, and is usually omitted. It is useful, however, when solving eigenvalue problems.

## Computing a Solution Estimate

Once the solution guess has been defined, the simplest call to **bvp4c()** has the form

```
        sol = bvp4c ( odefun, bcfun, solinit )
```

where

- *odefun* is the handle for a function that evaluates the differential equation.
- *bcfun* is the handle for a function that evaluates the boundary conditions.
- *solinit* is a structure containing the initial guess for the solution, as returned by **bvpinit()**.

The simplest call to the user-written function **odefun()** has the form

```
        dydx = odefun ( x, y )
```

where

- *x* is a scalar value where the ODE is to be evaluated.
- *y* contains the M components of the solution estimate at x.
- *dydx* is a column M-vector containing the right hand side of the first-order differential equations.

The simplest call to the user-written function **bcfun()** has the form

```
        res = bcfun ( ya, yb )
```

where

- *ya*, *yb* are M component vectors of the solution estimates at the left and right endpoints.
- *res* is a column M-vector containing the residual of the boundary conditions.

## Evaluating the Computed Solution

The **bvp4c()** function returns as output the quantity *sol*, which contains information that can be used to evaluate the solution components at any point in the domain. To do this, however, you must invoke the **deval()** function. For instance, if the solution has been computed over the interval [0,4], and we wish to evaluate the solution y(x) at 101 evenly spaced points within that interval, then the sequence of commands might be:

```
solinit = bvpinit ( xinit, yinit );
sol = bvp4c ( odefun, bcfun, solinit );
x = linspace ( 0.0, 4.0, 101 );
y = deval ( sol, x );
plot ( x, y(1,:) );
```

Note that **deval** evaluates all M components of the solution. In the common case of a second order BVP, y(1,:) would contain the solution, and y(2,:) the derivative of the solution, at each point x.

## Licensing:

## Languages:

**BVP4C** is available in a MATLAB version.

## Related Data and Programs:

FD1D_BVP, a MATLAB program which applies the finite difference method (FDM) to a two point boundary value problem (BVP) in one spatial dimension.

FEM1D, a MATLAB program which applies the finite element method (FEM) to a 1D linear two point boundary value problem (BVP).

FEM1D_BVP_LINEAR, a MATLAB program which applies the finite element method (FEM), with piecewise linear elements, to a two point boundary value problem (BVP) in one spatial dimension, and compares the computed and exact solutions with the L2 and seminorm errors.

FEM1D_SPECTRAL_NUMERIC, a MATLAB program which applies the spectral finite element method (FEM) to solve the two point boundary value problem (BVP_ u'' = - pi^2 sin(x) over [-1,+1] with zero boundary conditions, using as basis elements the functions x^n*(x-1)*(x+1), and carrying out the integration numerically, using MATLAB's quad() function, by Miro Stoyanov.

FEM1D_SPECTRAL_SYMBOLIC, a MATLAB program which applies the spectral finite element method (FEM) to solve the two point boundary value problem (BVP) u" = - pi^2 sin(x) over [-1,+1] with zero boundary conditions, using as basis elements the functions x^n*(x-1)*(x+1), and carrying out the integration using MATLAB's symbolic toolbox, by Miro Stoyanov.

PDEPE, MATLAB programs which illustrate how MATLAB's pdepe() function can be used to solve initial boundary value problems (IBVP's) in one spatial dimension.

STRING_SIMULATION, a MATLAB program which simulates the behavior of a vibrating string by solving the corresponding initial boundary value problem (IBVP), creating files that can be displayed by gnuplot.

## Reference:

- http://www.mathworks.com/help/matlab/bvp4c.html , the MathWorks help page for BVP4C.
- Lawrence Shampine, Jacek Kierzenka, Mark Reichelt,
  Solving boundary value problems for ordinary differential equations in MATLAB with bvp4c.
- Jacek Kierzenka, Lawrence Shampine,,
  A BVP Solver Based on Residual Control and the Matlab PSE,
  ACM Transactions on Mathematical Software,
  Volume 27, Number 3, September 2001, pages 299-316.

## Examples and Tests:

SAMPLE 1 sets up a solution to the problem y" + abs(y) = 0, y(0) = 0, y(4) = -2.

- sample1.m, defines the problem, calls bvp4c() to solve it, and plots the results.
- sample1.png, a plot of the solution Y(X) and derivative Y'(X).

EXAMPLE 1 sets up a solution to a system of five first order ODE's. This is a sample problem for the MUSN program.

- example1.m, defines the problem, calls bvp4c() to solve it, and plots the results.
- example1.png, a plot of the solution Y(X).

EXAMPLE 2 sets up a solution to a y"+3py/(p+x^2)^2=0, for which an analytic solution is known.

- example2.m, defines the problem, calls bvp4c() to solve it, and plots the results.

- example2.png, a plot of the solution Y(X).

EXAMPLE 3 sets up a solution to Mathieu's equation, an eigenvalue problem y" + (lambda-2*q*cos(2x)y)=0, y'(0) = 0, y'(pi) = 0, y(0) = 1, with q = 5, and lambda an unknown eigenvalue which we estimate to be 15. The special functional form is used to specify the initial guess for the solution.

- example3.m, defines the problem, calls bvp4c() to solve it, and plots the results.
- example3.png, a plot of the solution Y(X).

EXAMPLE 4 sets up problem modeling the propagation of nerve impulses, in which the solution is expected to be periodic, with the period unknown.

- example4.m, defines the problem, calls bvp4c() to solve it, and plots the results.
- example4.png, a plot of the solution Y(X) versus the initial guess.

BRATU sets up the Bratu equation, which includes a parameter lambda. Depending on the value of lambda, the equation may have 2, 1 or 0 solutions.

- bratu.m, defines the problem, calls bvp4c() to solve it, and plots the results.
- bratu_0.450000.png, a plot of the solution for lambda = 0.45.
- bratu_1.000000.png, a plot of the solutions for lambda = 1.0.
- bratu_3.500000.png, a plot of the solution for lambda = 3.5.

You can go up one level to the MATLAB source codes.

---

*Last revised on 07 September 2013.*