bvp4c

Solve two-point boundary value problems (BVPs) for ordinary differential equations

Syntax

```
sol = bvp4c(odefun,bcfun,solinit)
sol = bvp4c(odefun,bcfun,solinit,options)
sol = bvp4c(odefun,bcfun,solinit,options,p1,p2...)
```

Arguments

`odefun` A function that evaluates the differential equations $f(x, y)$. It can have the form

```
dydx = odefun(x,y)
dydx = odefun(x,y,p1,p2,...)
dydx = odefun(x,y,parameters)
dydx = odefun(x,y,parameters,p1,p2,...)
```

where `x` is a scalar corresponding to x, and `y` is a column vector corresponding to y. `parameters` is a vector of unknown parameters, and `p1,p2,...` are known parameters. The output `dydx` is a column vector.

`bcfun` A function that computes the residual in the boundary conditions $bc(y(a), y(b))$. It can have the form

```
res = bcfun(ya,yb)
res = bcfun(ya,yb,p1,p2,...)
res = bcfun(ya,yb,parameters)
res = bcfun(ya,yb,parameters,p1,p2,...)
```

where `ya` and `yb` are column vectors corresponding to $y(a)$ and $y(b)$. `parameters` is a vector of unknown parameters, and `p1,p2,...` are known parameters. The output `res` is a column vector.

`solinit` A structure with fields:

`x`	Ordered nodes of the initial mesh. Boundary conditions are imposed at a = `solinit.x(1)` and b = `solinit.x(end)`.
`y`	Initial guess for the solution such that `solinit.y(:,i)` is a guess for the solution at the node `solinit.x(i)`.
`parameters`	Optional. A vector that provides an initial guess for unknown parameters.

The structure can have any name, but the fields must be named `x`, `y`, and `parameters`. You can form `solinit` with the helper function `bvpinit`. See `bvpinit` for details.

`options` Optional integration argument. A structure you create using the `bvpset` function. See `bvpset` for details.

`p1,p2...` Optional. Known parameters that the solver passes to `odefun`, `bcfun`, and all the functions specified in `options`.

Description

`sol = bvp4c(odefun,bcfun,solinit)` integrates a system of ordinary differential equations of the form

$$y' = f(x, y)$$

on the interval [a,b] subject to general two-point boundary conditions

$$bc(y(a), y(b)) = 0$$

The `bvp4c` solver can also find unknown parameters p for problems of the form

$$y' = f(x, y, p)$$
$$bc(y(a), y(b), p) = 0$$

where p corresponds to `parameters`. You provide `bvp4c` an initial guess for any unknown parameters in `solinit.parameters`. The `bvp4c` solver returns the final values of these unknown parameters in `sol.parameters`.

`bvp4c` produces a solution that is continuous on [a,b] and has a continuous first derivative there. Use the function `deval` and the output `sol` of `bvp4c` to evaluate the solution at specific points `xint` in the interval [a,b].

```
sxint = deval(sol,xint)
```

The structure `sol` returned by `bvp4c` has the following fields:

`sol.x`	Mesh selected by `bvp4c`
`sol.y`	Approximation to $y(x)$ at the mesh points of `sol.x`
`sol.yp`	Approximation to $y'(x)$ at the mesh points of `sol.x`
`sol.parameters`	Values returned by `bvp4c` for the unknown parameters, if any
`sol.solver`	'bvp4c'

The structure `sol` can have any name, and `bvp4c` creates the fields `x`, `y`, `yp`, `parameters`, and `solver`.

`sol = bvp4c(odefun,bcfun,solinit,options)` solves as above with default integration properties replaced by the values in `options`, a structure created with the `bvpset` function. See `bvpset` for details.

`sol = bvp4c(odefun,bcfun,solinit,options,p1,p2...)` passes constant *known* parameters, `p1`, `p2`, ..., to `odefun`, `bcfun`, and all the functions the user specifies in `options`. Use `options = []` as a placeholder if no options are set.

Examples

Example 1. Boundary value problems can have multiple solutions and one purpose of the initial guess is to indicate which solution you want. The second order differential equation

$$y'' + |y| = 0$$

has exactly two solutions that satisfy the boundary conditions

$$y(0) = 0$$
$$y(4) = -2$$

Prior to solving this problem with `bvp4c`, you must write the differential equation as a system of two first order ODEs

$$y_1' = y_2$$
$$y_2' = -|y_1|$$

Here $y_1 = y$ and $y_2 = y'$. This system has the required form

$$y' = f(x, y)$$
$$bc(y(a), y(b)) = 0$$

The function f and the boundary conditions bc are coded in MATLAB as functions `twoode` and `twobc`.

```
function dydx = twoode(x,y)
  dydx = [ y(2)
           -abs(y(1))];

function res = twobc(ya,yb)
  res = [ ya(1)
          yb(1) + 2];
```

Form a guess structure consisting of an initial mesh of five equally spaced points in [0,4] and a guess of constant values $y_1(x) \equiv 1$ and $y_2(x) \equiv 0$ with the command

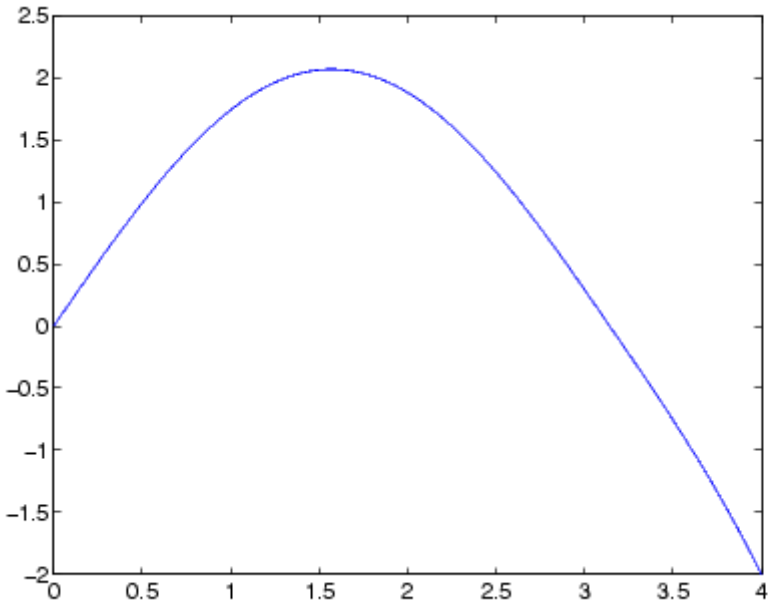
```
solinit = bvpinit(linspace(0,4,5),[1 0]);
```

Now solve the problem with

```
sol = bvp4c(@twoode,@twobc,solinit);
```

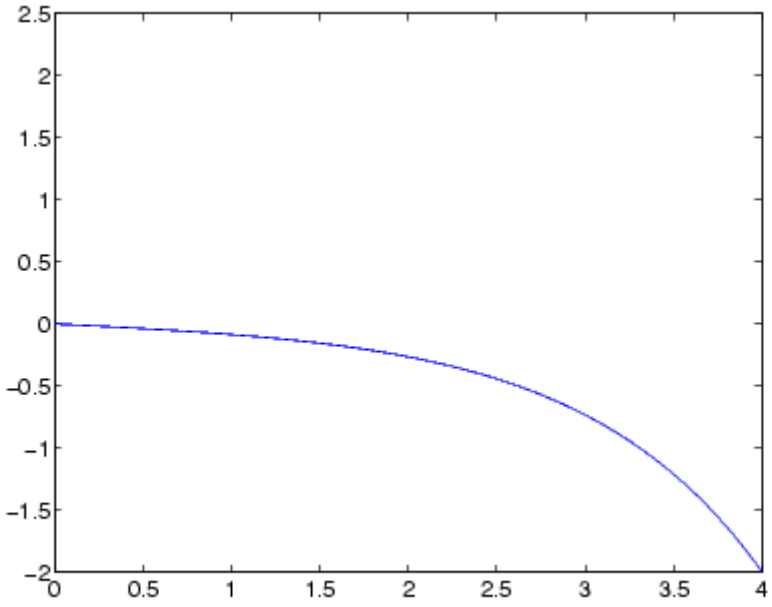
Evaluate the numerical solution at 100 equally spaced points and plot $y(x)$ with

```
x = linspace(0,4);
y = deval(sol,x);
plot(x,y(1,:));
```



You can obtain the other solution of this problem with the initial guess

```
solinit = bvpinit(linspace(0,4,5),[-1 0]);
```



Example 2. This boundary value problem involves an unknown parameter. The task is to compute the fourth ($q = 5$) eigenvalue λ of Mathieu's equation

$$y'' + (\lambda - 2\,q\cos 2x)\,y = 0$$

Because the unknown parameter λ is present, this second order differential equation is subject to *three* boundary conditions

$$y'(0) = 0$$
$$y'(\pi) = 0$$
$$y(0) = 1$$

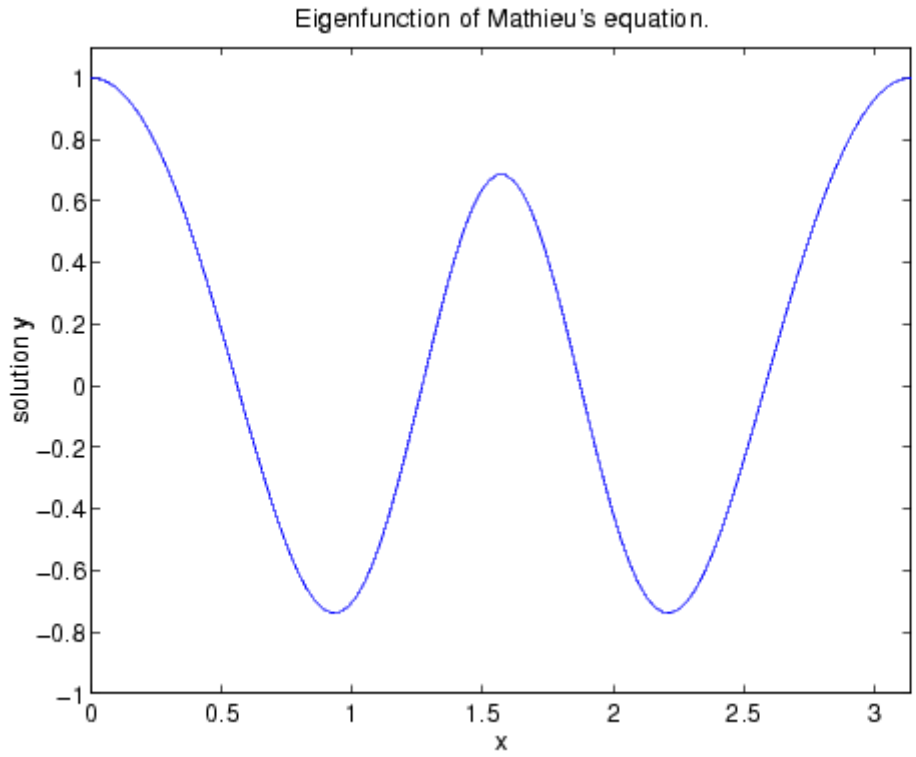
It is convenient to use subfunctions to place all the functions required by `bvp4c` in a single M-file.

```
function mat4bvp

lambda = 15;
solinit = bvpinit(linspace(0,pi,10),@mat4init,lambda);
sol = bvp4c(@mat4ode,@mat4bc,solinit);

fprintf('The fourth eigenvalue is approximately %7.3f.\n',...
        sol.parameters)

xint = linspace(0,pi);
Sxint = deval(sol,xint);
plot(xint,Sxint(1,:))
axis([0 pi -1 1.1])
title('Eigenfunction of Mathieu''s equation.')
xlabel('x')
ylabel('solution y')
% --------------------------------------------------------------
function dydx = mat4ode(x,y,lambda)
q = 5;
dydx = [  y(2)
         -(lambda - 2*q*cos(2*x))*y(1) ];
% --------------------------------------------------------------
function res = mat4bc(ya,yb,lambda)
res = [  ya(2)
         yb(2)
        ya(1)-1 ];
% --------------------------------------------------------------
function yinit = mat4init(x)
yinit = [  cos(4*x)
          -4*sin(4*x) ];
```

The differential equation (converted to a first order system) and the boundary conditions are coded as subfunctions `mat4ode` and `mat4bc`, respectively. Because unknown parameters are present, these functions must accept three input arguments, even though some of the arguments are not used.

The guess structure `solinit` is formed with `bvpinit`. An initial guess for the solution is supplied in the form of a function `mat4init`. We chose $y = \cos 4x$ because it satisfies the boundary conditions and has the correct qualitative behavior (the correct number of sign changes). In the call to `bvpinit`, the third argument (`lambda = 15`) provides an initial guess for the unknown parameter λ.

After the problem is solved with `bvp4c`, the field `sol.parameters` returns the value $\lambda = 17.097$, and the plot shows the eigenfunction associated with this eigenvalue.



Algorithms

`bvp4c` is a finite difference code that implements the three-stage Lobatto IIIa formula. This is a collocation formula and the collocation polynomial provides a C^1-continuous solution that is fourth order accurate uniformly in [a,b]. Mesh selection and error control are based on the residual of the continuous solution.

See Also

`@` (`function_handle`), `bvpget`, `bvpinit`, `bvpset`, `deval`

References

[1] Shampine, L.F., M.W. Reichelt, and J. Kierzenka, "Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c," available at `ftp://ftp.mathworks.com/pub/doc/papers/bvp/`.