A First Course on Kinetics and Reaction Engineering

Example S6.2

Problem Purpose

This example illustrates the use of the MATLAB template file SolvBVDifS.m to solve a second order boundary value ordinary differential equation with a singularity at the boundary where the independent variable equals zero.

Problem Statement

Consider the ODE and boundary conditions given in equations (1) through (3). This ODE might need to be solved in an analysis of the simultaneous reaction and diffusion within a porous, spherical catalyst pellet. In these equations, C_A represents the concentration of reactant A, r represents the radial distance from the center of the catalyst particle, k represents a rate coefficient, D_{ea} represents an effective diffusivity, R_p represents the radius of the catalyst particle and C_{As} represents the concentration of A at the outer surface of the catalyst particle. For present purposes k, D_{ea}, R_p and C_{As} will be assumed to be dimensionally consistent constants with the following values: D_{ea} = 3.0 x 10^{-7}; k = 0.02; C_{As} = 1.0 and R_p = 0.3. Once the solution has been found, the results should be used to calculate the steady state rate of reaction per particle volume according to equation (4).

$$\frac{d^2C_A}{dr^2}+\frac{2}{r}\frac{dC_A}{dr}-\frac{k}{D_{eA}}C_A=0 \qquad 0\le r\le R_p \tag{1}$$

$$\left.\frac{dC_A}{dr}\right|_{r=0}=0 \tag{2}$$

$$C_A\left(R_p\right)=C_{As} \tag{3}$$

$$rate=-\frac{3D_{eA}}{R_p}\left.\frac{dC_A}{dr}\right|_{r=R_p} \tag{4}$$

Problem Analysis

Examination reveals that equation (1) contains no partial derivatives; it is an ordinary differential equation. The boundary condition given in equation (2) applies at r = 0, while the boundary condition given in equation (3) applies at $r = R_p$. Thus, the boundary conditions do <u>not</u> all apply at the same boundary, and consequently this is a boundary value ODE problem. Further examination of the equations reveals that C_A is the dependent variable because it is the quantity that is differentiated. Similarly, r is the independent variable because the derivatives in the ODE are all taken with respect to it. In the second term of equation (1), r^{-1} appears, and this causes a singularity at the boundary where r = 0 (this term goes to infinity). Under these conditions, the MATLAB template file SolvBVDifS.m can be used to solve

the differential equation. Doing so will require modification of the template file in seven locations. It will also require that the second order ODE, equation (1), be converted to a set of two first order ODEs.

Problem Solution

The first thing we need to do is to re-write the second order ODE as a set of two, coupled first order ODEs. Here, as I do so, I will also change the variable names so they match those used in the template file. That is, I will define y_1 as being equal to C_A and x as being equal to r, as indicated in equations (4) and (5). Next I introduce a second dependent variable, y_2, and define it to equal the first derivative of C_A with respect to r, equation (6). Note that taking the derivative of both sides of equation (6) shows that the second derivative of C_A with respect to r is equal to the first derivative of y_2 with respect to x, equation (7).

$$y_1 \equiv C_A \tag{4}$$

$$x \equiv r \tag{5}$$

$$y_2 \equiv \frac{dC_A}{dr} = \frac{dy_1}{dx} \tag{6}$$

$$\frac{dy_2}{dx} = \frac{d}{dx}\left(\frac{dC_A}{dr}\right) = \frac{d}{dr}\left(\frac{dC_A}{dr}\right) = \frac{d^2C_A}{dr^2} \tag{7}$$

Below, I duplicated equation (6) as equation (8) because it is one of the two first order ODEs that will replace the second order ODE, equation (1). Next, I replaced the ***second*** derivative of C_A with respect to r in equation (1) with the ***first*** derivative of y_2 with respect to x, per equation (7) and I replaced the ***first*** derivative of C_A with respect to r in equation (1) with y_2, per equation (6). Doing so gives equation (9), the other first order ODE that, together with equation (8), forms a set of two first order ODEs that are equivalent to the original second order ODE, equation (1).

$$\frac{dy_1}{dx} = y_2 \tag{8}$$

$$\frac{dy_2}{dx} = -\frac{2y_2}{x} + \frac{k}{D_{eA}}y_1 \tag{9}$$

The second term in equation (9) introduces a singularity at the boundary where $x = 0$. Consequently, the equations should be written in the form given in equation (10), so that the MATLAB template file SolvBVDifS.m can be used to solve the ODEs. Comparing equations (8) and (9) to equation (10) shows that they are already in the proper form with f_1, f_2, S_{11}, S_{12}, S_{21} and S_{22} defined as in equations (11) and (12).

$$\frac{dy_i}{dx} = \sum_{\text{all } j} S_{ij}\frac{y_j}{x} + f_i\left(x, \underline{y}\right) \tag{10}$$

$$S_{1,1} = 0; \quad S_{1,2} = 0; \quad f_1\left(x, \underline{y}\right) = y_2 \tag{11}$$

$$S_{2,1} = 0; \quad S_{2,2} = -2; \quad f_2\left(x, \underline{y}\right) = \frac{k}{D_{eA}} y_1 \tag{12}$$

The boundary conditions must also be converted by replacing the ***first*** derivative of C_A with respect to r with y_2, leading to equations (13) and (14). While transforming the boundary equations, I also wrote them in the form used in the step-by-step instructions for use of SolvBVDifS.m: $\underline{g}\left(\underline{y}(0), \underline{y}\left(R_p\right)\right) = 0$.

Equations (8), (9), (13) and (14) constitute a set of coupled first order ODEs and their boundary conditions that are entirely equivalent to the original second order ODE and its boundary conditions, equations (1) through (3).

$$\left.\frac{dC_A}{dr}\right|_{r=0} = y_2(0) = g_1\left(y_1(0), y_2(0), y_1\left(R_p\right), y_2\left(R_p\right)\right) = 0 \tag{13}$$

$$y_1\left(R_p\right) - C_{As} = g_2\left(y_1(0), y_2(0), y_1\left(R_p\right), y_2\left(R_p\right)\right) = 0 \tag{14}$$

Continuing with the step-by-step instructions for using SolvBVDifS.m, a copy of SolvBVDifS.m was saved in the working directory as S6_Example_2.m. The function declaration statement was changed so that the function name was the same as the filename without the ".m" filetype. At the same time, the long initial comment was changed to a shorter comment indicating the purpose of the modified template file. The first required modification involves entering each constant that appears in the problem in consistent units. This problem involves several constants: D_{ea} = 3.0 x 10^{-7}; k = 0.02; C_{As} = 1.0 and R_p = 0.3, so each one is entered. All of these changes can be seen in Listing 1.

```
% Modified version of the MATLAB template file SolvBVDifS.m used to solve
% Example 2 of Supplemental Unit S6 of "A First Course on Kinetics and
% Reaction Engineering."
%
function result = S6_Example_2
    % Known quantities and constants (in consistent units)
    Dea = 3.0E-7;
    k = 0.02;
    Rp = 0.3;
    CAs = 1.0;
```

Listing 1. Modified version of SolvBVDifS.m after renaming and making the first required modification.

The second required modification involves entering the code to evaluate the functions f_1 and f_2 in equations (11) and (12) and setting the corresponding rows of the column vector dydx equal to their values. This modification appears within the internal function named bvodes. It is important to note that

the function, f_2, does not include the term $-\frac{2y_2}{x}$. That term will be added to f_2 automatically by MATLAB as long as we enter the elements of the singularity matrix, $\underline{\underline{S}}$, correctly. This will be done in a subsequent required modification of SolvBVDifS.m. The resulting, modified version of `bvodes` is shown in Listing 2.

```
% Function that evaluates the column vector f in dydx = S*y/x + f(x,y)
function dydx = bvodes(x,y)
    dydx = [
        y(2)
        k/Dea*y(1)
    ];
end % of internal function bvodes
```

Listing 2. Internal function bvodes after making the second required modification.

The third required modification involves entering the code to evaluate the functions g_1 and g_2 in equations (13) and (14) and setting the corresponding rows of the column vector `res` equal to their values. This modification appears within the internal function named `bvs`. The comment at the start of that internal function notes that the values of $\underline{y}$ at $x = 0$ are available in the column vector `y_at_start` and the values of $\underline{y}$ at $x = R_p$ are available in the column vector `y_at_end`. The resulting, modified version of `bvs` is shown in Listing 3.

```
% Function that calculates the errors at the boundaries
function res = bvs(y_at_start,y_at_end)
    % y_at_start is a column vector containing y values at the starting
    % x boundary and y_at_end is a column vector containing y values at
    % the ending x boundary.
    res = [
        y_at_start(2)
        y_at_end(1) - CAs
    ];
end % of internal function bvs
```

Listing 3. Internal function bvs after making the third required modification.

The fourth required modification is where the range of the independent variable is set and the number of mesh points are chosen. All you need to do here is remove the comments at the end of the three lines defining the variables and replace them with the corresponding value of the low end of the range of x, the high end of the range of x, and the number of mesh points you want to use. In this problem the range is from x = 0 to x = R_p, so I entered those values on the first two lines. I chose to use 20 mesh points, but MATLAB may increase the number of mesh points if it needs to do so in order to achieve sufficient accuracy. The resulting code is shown in Listing 4.

```
% Set up the initial mesh
x_range_low = 0;
x_range_high = Rp;
n_mesh_points = 20;
```

Listing 4. Code resulting from the fourth required modification of SolvBVDifS.m.

The template file is set up so that a single value needs to be guessed for each dependent variable. Essentially we just need to guess the average value of each dependent variable over the range from x = 0 to x = R_p. (It is possible to provide a guess that varies over the range from x = 0 to x = R_p. If you want or need to do this, consult the MATLAB documentation.) As a guess for the average value of C_A (which equals y_1), I'm just going to use the value at the external surface of the particle, C_{As}. As a guess for the average value of y_2, I'll use the external concentration divided by the particle radius. (Remember, y_2 represents the derivative of C_A with respect to r; if the concentration went to zero at the particle center, this guess would equal the average value of y_2.) The guesses are entered as rows in the column vector named `yinit`. The first row of `yinit` contains the guess for y_1 and the second row contains the guess for y_2. The resulting code is shown in Listing 5.

```
% Guesses
yinit = [
    CAs
    CAs/Rp
];
```

Listing 5. Code resulting from the fifth required modification of SolvBVDifS.m.

The sixth required modification involves entering the singularity matrix. Specifically, the S_{ij} elements of the singularity matrix, equations (11) and (12), are entered row by row with the individual elements separated by commas. Listing 6 shows the results of doing so.

```
% Provide the matrix, S, that MATLAB uses to handle singularities
S = [
    0,0
    0,-2
];
```

Listing 6. Code resulting from the sixth required modification of SolvBVDifS.m.

The final required modification involves performing any calculations that use the results from solving the ODEs. In the present problem, we want to use the results of solving the ODEs to calculate the rate according to equation (4). As the comments in the code state, the variable `result.x` is a vector containing the x values of the mesh points. I first get the index of the last mesh point, which corresponds to $x = r$ = R_p. This is necessary because MATLAB may have changed the number of mesh points to some

number different than the twenty mesh points I originally specified. MATLAB will do this, when necessary, to ensure sufficient accuracy of the results. The variable `result.y` is a matrix where the columns correspond to the dependent variables (the first column is y_1 and the second column is y_2); the rows contain the values of those variables at each of the mesh points. Since y_2 is equal to $\frac{dC_A}{dr}$, the last element of the second column of `result.y` will equal $\frac{dC_A}{dr}$ at $r = R_p$. I can then use that value to calculate the rate using equation (4), as requested. The resulting code is shown in Listing 7.

```
    % Calculate the steady state rate:
    lastPoint = length(result.x);
    rate = (-3*Dea/Rp)*result.y(2,lastPoint)
```

Listing 7. Code resulting from the seventh and final required modification of SolvBVDifS.m.

That completes the required modifications, so once the file is saved to make the changes permanent, it can be executed by typing the first line shown in Listing 8 at the MATLAB command prompt. Examining the output, also shown in Listing 8, you can see that MATLAB increased the number of mesh points. This is apparent because I specified 20 mesh points, but the x, y and y*p* vectors have 48 elements. The code I wrote (seventh required modification) anticipated that this might happen, so the rate reported in the listing should be correct. (Note: in case you were worried, the rate is negative because A is being consumed by the reaction.)

```
>> result = S6_Example_2

rate =

  -7.7460e-04


result =

    solver: 'bvp4c'
         x: [1x48 double]
         y: [2x48 double]
        yp: [2x48 double]
     stats: [1x1 struct]
```

Listing 8. Results of executing the modified template file, S6_Example_2.m.