A First Course on Kinetics and Reaction Engineering
Example S5.1

Problem Purpose

This example illustrates the use of the MATLAB template file SolvBVDif.m to solve a second order boundary value ordinary differential equation.

Problem Statement

Solve the boundary value ODE in equation (1) with the boundary conditions specified in equations (2) and (3) to find the value of C_A at $z = L$, and make a plot of C_A versus z. Further assume that D, u_s, k, K, C_A^0 and L are known constants with the following values (in dimensionally consistent units): D = 8.0 x 10^{-6}; u_s = 0.01; k = 0.012; K = 1.0; C_A^0 = 1.0 and L = 1.25. As equation (1) indicates, the ODE applies over the range of z from 0 to L.

$$-D\frac{d^2C_A}{dz^2}+u_s\frac{dC_A}{dz}=-\left(\left(k+\frac{k}{K}\right)C_A-\frac{k}{K}C_A^0\right)\qquad 0\le z\le L \tag{1}$$

$$u_sC_A\left(0\right)-D\left.\frac{dC_A}{dz}\right|_{z=0}=u_sC_A^0 \tag{2}$$

$$\left.\frac{dC_A}{dz}\right|_{z=L}=0 \tag{3}$$

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 z = 0, while the boundary condition given in equation (3) applies at z = L. Thus, the boundary conditions do <u>not</u> all apply at the same boundary, and consequently this is a boundary value ODE problem. Finally, z^{-1} does not appear in the equation, so there is no singularity at the boundary where z = 0. Under these conditions, the MATLAB template file, SolvBVDif.m can be used to solve the ODE. In order to do so, the second order ODE will need to be converted into a set of two first order ODEs and the template file will have to be modified in six places.

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. Equation (1) includes the second derivative of C_A with respect to z, so it is a second order ODE and will need to be converted into a set of two 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 z, 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 z, equation (6). Note that taking the derivative of both sides of equation (6) shows that the second derivative of C_A with respect to z is equal to the first derivative of y_2 with respect to x, equation (7).

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

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

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

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

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 z 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 z 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). Similarly, the boundary conditions are converted by replacing the *first* derivative of C_A with respect to z with y_2, leading to equations (10) and (11). Equations (8) through (11) 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).

$$\frac{dy_1}{dx} = f_1\left(x,y_1,y_2\right) = y_2 \tag{8}$$

$$\frac{dy_2}{dx} = f_2\left(x,y_1,y_2\right) = \frac{1}{D}\left(\left(k+\frac{k}{K}\right)y_1 + u_s y_2 - \frac{k}{K}C_A^0\right) \tag{9}$$

$$u_s y_1(0) - D y_2(0) - u_s C_A^0 = g_1\left(y_1(0),y_2(0),y_1(L),y_2(L)\right) = 0 \tag{10}$$

$$y_2(L) = g_2\left(y_1(0),y_2(0),y_1(L),y_2(L)\right) = 0 \tag{11}$$

Examination of equations (8) and (9) shows that x^{-1} does not appear in the equations, so there still is no singularity at the boundary where x = 0, so SolvBVDif.m, still can be used to solve them. Here I will do so by following the step-by-step instructions for using SolvBVDif.m that are provided with this unit. A copy of SolvBVDif.m was saved in the working directory as S6_Example_1.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 = 8.0 x 10^{-6}; u_s = 0.01; k =

0.012; K = 1.0; C_A^0 = 1.0 and L = 1.25, so each one is declared and assigned the appropriate value. All of these changes can be seen in Listing 1.

```
% Modified version of the MATLAB template file SolvBVDif.m that was used to
% solve Example 1 of Supplemental Unit S6 of "A First Course on Kinetics
% and Reaction Engineering."
%
function result = S6_Example_1
    % Known quantities and constants (in consistent units)
    D = 8.0E-6;
    us = 0.01;
    k = 0.012;
    K = 1.0;
    CA0 = 1.0;
    L = 1.25;
```

Listing 1. Modified version of SolvBVDif.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 (8) and (9) and setting the corresponding rows of the column vector dydx equal to their values. This modification appears within the internal function named bvodes. The resulting, modified version of bvodes is shown in Listing 2.

```
    % Function that evaluates the derivatives
    function dydx = bvodes(x,y)
        dydx = [
            y(2)
            (1/D)*((k+k/K)*y(1)+us*y(2)-k*CA0/K)
        ];
    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 (10) and (11) 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 y at $x = 0$ are available in the column vector y_at_start and the values of y at $x = L$ 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 = [
            us*y_at_start(1)-D*y_at_start(2)-us*CA0
            y_at_end(2)
        ];

    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 = L, so I entered those values on the first two lines. I chose to use 40 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 = 0.0;
    x_range_high = L;
    n_mesh_points = 40;
```

Listing 4. Code resulting from the fourth required modification of SolvBVDif.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 = L. (It is possible to provide a guess that varies over the range from x = 0 to x = L. If you want or need to do this, consult the MATLAB documentation.) It was noted previously that the first dependent variable corresponds to the concentration of A. Since this is just a guess, I simply guessed that the average concentration of A is equal to C_A^0. The second dependent variable corresponds to the derivative of C_A with respect to z. If the A was completely converted over the range of x, then on average, the derivative would equal the negative of the concentration divided by the distance, L, and that's what I used as a guess. These guesses are entered 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 = [
        CA0
        -CA0/L
    ];
```

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

The final required modification involves performing any calculations that use the results from solving the ODEs. In this problem we are asked for the value of C_A (or y_1) at z (or x) equal to L. As the comments in the template file indicate, 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 the end of the reactor where $x = L$. This is necessary because MATLAB may change the number of mesh points from what I originally set up. MATLAB does this to ensure that the result is accurate. (See the MATLAB documentation if you wish to change the 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. Here I simply print out the value of y_1, that is the concentration of A, at the last mesh point where $x = L$. (It will print out when the function is executed because I didn't put a semicolon at the end of the line.) Finally, I entered code to generate a very simple plot of the concentration of A as a function of x. Consult the MATLAB documentation to learn how to make the plot more attractive, add a legend, etc. The resulting code is shown in Listing 6.

```
    % Calculate the value of CA (y(1)) at z (x) = L
    lastPoint = length(result.x);
    CAf = result.y(1,lastPoint)
    % Plot the concentration as a function of axial position
    plot(result.x,result.y(1,:))
    xlabel('Axial Position')
    ylabel('Concentration of A')
```

Listing 6. Code resulting from the sixth and final required modification of SolvBVDif.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 7 at the MATLAB command prompt. Examining the output, also shown in Listing 7, you can see that MATLAB increased the number of mesh points. This is apparent because I specified 40 mesh points, but the `x`, `y` and `yp` vectors have 48 elements. As the listing shows, the 40th mesh point corresponds to x = 1.2402, not 1.25. The 48th mesh point corresponds to x = 1.25 = L. The code I wrote (sixth required modification) anticipated that this might happen, and as you can see, the value that was reported for `CAf` is indeed equal to C_A (y_1) for mesh point 48, not mesh point 40. In addition to the output seen in Listing 7, a plot of C_A vs. z (y_1 vs x) was also generated. It is shown in Figure 1.

```
>> result = S6_Example_1

CAf =

    0.5250


result =

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

>> result.x(48)

ans =

    1.2500

>> result.x(40)

ans =

    1.2402

>> result.y(1,48)

ans =

    0.5250
```

Listing 7. Results of executing the modified template file, S6_Example_1.m and additional MATLAB command line entries.

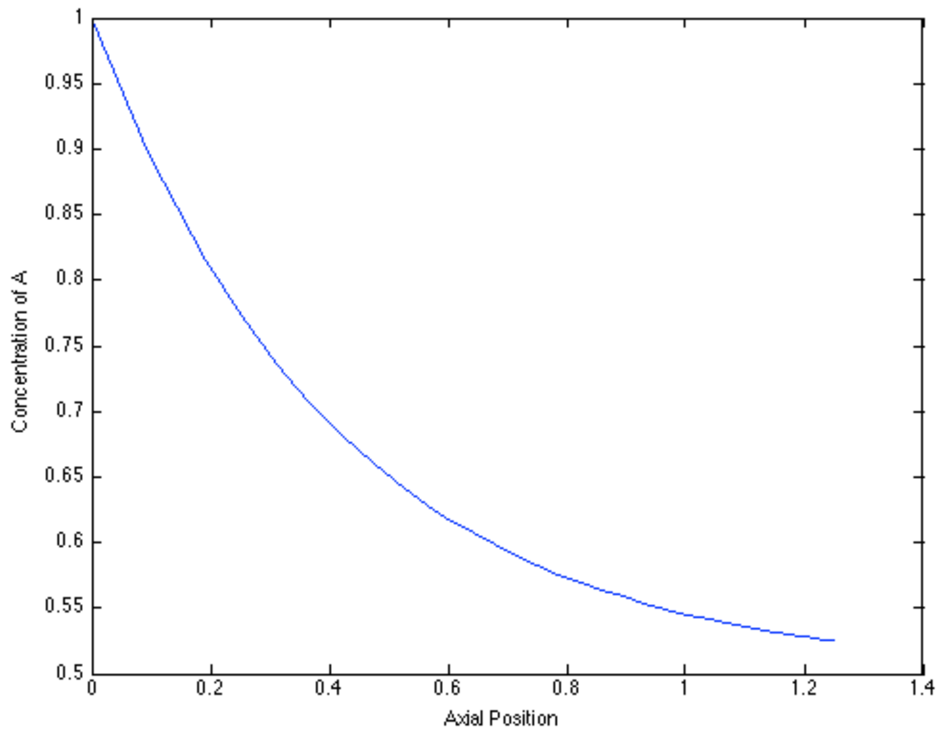

Figure 1. Plot of C_A (y_1) versus z (x).