A First Course on Kinetics and Reaction Engineering

Example 33.1

Problem Purpose

This problem will help you determine whether you have mastered the learning objectives for this unit. It illustrates the analysis of a tubular reactor using the axial dispersion model. It also supports the claim made in the informational reading that axial dispersion has little effect beyond a distance of ca. 50 particle diameters into the reactor.

Problem Statement

Consider an isothermal tubular packed bed reactor with axial dispersion where the Danckwerts boundary conditions apply. The irreversible reaction A → B takes place with a forward rate coefficient equal to 1.2 × 10^{-2} s^{-1}. The feed contains only A at a concentration of 1M. The axial Peclet number, based on the packing diameter and the superficial velocity, is equal to 2 and the inlet superficial velocity is 0.01 m/s. The reactor contains a fixed bed of 0.004 m diameter catalyst particles. Compare a plot of the conversion versus length to that for a plug flow reactor for reactor lengths from zero to fifty times the particle diameter.

Problem Analysis

This problem is straightforward; we are told the reactor is tubular with axial dispersion and to use the Danckwerts boundary conditions. We simply need to set up the mole balance equations for the axial dispersion model and solve them. The results will then be plotted. We will also need to solve the PFR design equations to make the requested comparison.

Problem Solution

The following quantities are given in the problem statement: k = 1.2 × 10^{-2} s^{-1}, $C_{A,feed}$ = 1 mol L^{-1}, Pe_{ax} = 2, u_s = 0.01 m s^{-1}, d_p = 0.004 m. The reactor is isothermal, we will ignore pressure drop, and the reaction does not change the total number of moles. As a consequent, the total molar flow rate, the volumetric flow rate and the superficial velocity are all constant.

Axial Dispersion Model The general form of the axial dispersion model mole balance is given in equation (1). In the present problem, there is only one reaction taking place, so the summation reduces to a single term. In addition, the superficial velocity is a constant and can be taken outside of the derivative. We normally begin by writing a mole balance for each species present in the system. Here, after writing the mole balance for A, equation (2), we can see that after substitution of the rate expression, the resulting differential equation contains only one dependent variable, namely the concentration of A. As a consequence, equation (2) can be solved independently of any other equations. In addition, solving equation (2) will allow us to calculate the conversion along the length of the reactor and make the requested plot. Therefore, we will not write the mole balance on B, since we do not need it in order to answer the question.

$$-D_{ax}\frac{d^2C_i}{dz^2}+\frac{d}{dz}\left(u_sC_i\right)=\sum_{\substack{j=all\\reactions}}\nu_{i,j}r_j \tag{1}$$

$$-D_{ax}\frac{d^2C_A}{dz^2}+u_s\frac{dC_A}{dz}=-r=-kC_A \tag{2}$$

Since we are using the Danckwerts boundary conditions, equation (2) is a boundary value ordinary differential equation (ODE). We will solve this equation numerically. No matter what software is used, solving equation (2) numerically will require us to provide code that is given values for the dependent variable and its derivatives as a function of the independent variable and that uses those values to evaluate the ODEs and the boundary conditions. In the present case, the three terms needed in order to evaluate the ODE, other than the dependent variable and its derivative, are the axial dispersion coefficient, the superficial velocity and the rate coefficient. The latter two are given in the problem statement. The axial dispersion coefficient can be calculated from the quantities given in the problem statement and listed above by rearranging the definition of the axial Peclet number, equation (3).

$$Pe_{ax}=\frac{u_sd_p}{D_{ax}}\qquad\Rightarrow\qquad D_{ax}=\frac{u_sd_p}{Pe_{ax}} \tag{3}$$

The general forms of the Danckwerts boundary conditions are given in equations (4) and (5). Writing them for reactant A leads to equations (6) and (7).

$$\text{at } z=0;\ u_sC_i\left(z=0\right)-D_{ax}\left.\frac{dC_i}{dz}\right|_{z=0}=u_sC_{i,feed} \tag{4}$$

$$\text{at } z=L;\ \left.\frac{dC_i}{dz}\right|_{z=L}=0 \tag{5}$$

$$u_sC_A\left(0\right)-D_{ax}\left.\frac{dC_A}{dz}\right|_{z=0}-u_sC_{A,feed}=0 \tag{6}$$

$$\left.\frac{dC_A}{dz}\right|_{z=L}=0 \tag{7}$$

All of the quantities appearing in the boundary conditions are known except for the reactor length. Here, we will calculate the reactor length using equation (8). Upon doing so, equation (2) can be solved to find the concentration of A along the length of the reactor, $C_A(z)$. Knowing that, and recognizing that the volumetric flow rate is constant, the fractional conversion along the length of the reactor, $f_A(z)$, can be calculated using equation (9) and plotted.

$$L=50d_p \tag{8}$$

$$f_A(z) = \frac{C_{A,feed} - C_A(z)}{C_{A,feed}} \tag{9}$$

Plug Flow Reactor It was shown in the informational reading that equation (10) is an alternative form of the general plug flow reactor mole balance design equation. Noting, as above, that the superficial velocity is constant and can be taken outside of the derivative and that there is only one reaction, the plug flow reactor mole balance for A takes the form of equation (11) after substitution of the rate expression. As was the case with the axial dispersion model, we can solve this equation numerically and answer the questions asked without writing a mole balance on B.

$$\frac{d}{dz}\left(u_s C_i\right) = \sum_{\substack{j=all \\ reactions}} \nu_{i,j} r_j \tag{10}$$

$$\frac{dC_A}{dz} = f_1\left(z, C_A\right) = \frac{-kC_A}{u_s} \tag{11}$$

Equation (11) is an initial value ordinary differential equation that can be solved numerically. In order to do so, no matter what software is used, it is necessary to provide initial values for the independent and dependent variables, the final value of either the independent or dependent variable, and code that is given values for the independent and dependent variables and that uses those values to evaluate the function f_1 in equation (11). Here the initial values will be provided at the reactor inlet where z = 0 and C_A = $C_{A,feed}$. We will solve equation (11) using a range of final values from z = 0 to z = L. For each final value, equation (11) can be solved to obtain the corresponding value of C_A, and then equation (9) can be used to calculate the corresponding value of f_A. The results can then be used to generate the requested plot.

The code needed to evaluate function f_1 in equation (11) will be given values for C_A and z. It is then trivial to evaluate f_1, since the rate coefficient and superficial velocity are known constants. With that, equation (11) can be solved numerically.

The resulting plots for the axial dispersion and plug flow reactor models are shown in Figure 1. If you look closely at very low z values, you will see that the two models predict slightly different conversions, but by the time the length of the reactor is fifty times the diameter of the bed packing, the two models have become essentially equal. This is consistent with the statement in the informational reading that adding the axial dispersion term has very little effect (compared to a PFR) except for very short reactors.

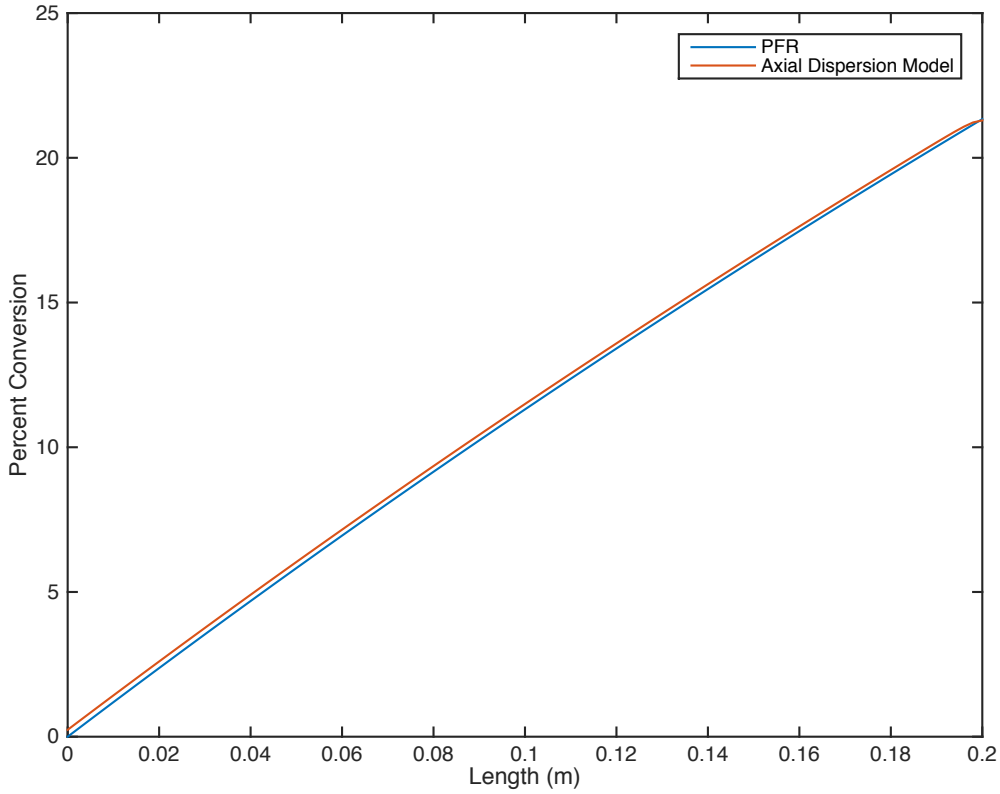

Figure 1. Conversion versus axial position in a PRF and a tubular reactor with axial dispersion.

Calculation Details Using MATLAB

Three MATLAB functions were used in the solution of this problem. One solves the PFR design equations and returns the conversion versus length, one solves the axial dispersion model and returns the conversion versus length and the third calls each of the first two and uses the results to make the plot shown in Figure 1. At this point in the course, the solution of initial value ODEs using MATLAB and the template files from Supplemental Unit S5 should be familiar. The third MATLAB function does no calculations. Therefore, those two MATLAB functions will not be described here, but they accompany this solution as Example_33_1.m and Example_33_1_prf.m.

The axial dispersion design equation is a mixed boundary ODE. The solution of mixed boundary ODEs is described in Supplemental Unit S6, which also provides template files for solving them. There are two template files provided. One is used if the ODE contains the reciprocal of the independent variable which causes a singularity at the point where that variable equals zero. Examining equation (2), it can be seen that the reciprocal of z is not present. Therefore the code presented here was generated by modifying the MATLAB template file SolvBVDif.m, a copy of which was saved as Example_33_1_ad.m and modified to solve this problem. The modified file accompanies this solution.

In order to use the template file, the second-order ODE of equation (2) must be reduced to a set of first order ODEs, and the boundary conditions must be re-written in terms of the new dependent variables. This is a straightforward process and is explained in Supplemental Unit S6. We begin by defining C_A' as in equation (12). With that definition, the second derivative of C_A with respect to z will equal the first derivative of C_A' with respect to z, as shown in equation (13). Substitution of equations (12) and (13) into equation (2) leads to equation (14).

$$\frac{dC_A}{dz} = C_A' \tag{12}$$

$$\frac{d^2C_A}{dz^2} = \frac{dC_A'}{dz} \tag{13}$$

$$-D_{ax}\frac{dC_A'}{dz} + u_s C_A' = -r \quad \Rightarrow \quad \frac{dC_A'}{dz} = \frac{r + u_s C_A'}{D_{ax}} \tag{14}$$

Thus, the single second order ODE, equation (2), is equivalent to the two first order ODEs, equations (12) and (14), re-written here as equations (15) and (16). Similarly substituting equation (12) into the original boundary conditions, equations (6) and (7), leads to the corresponding boundary conditions in terms of the new dependent variables, equations (17) and (18).

$$\frac{dC_A}{dz} = f_1\left(z, C_A, C_A'\right) = C_A' \tag{15}$$

$$\frac{dC_A'}{dz} = f_2\left(z, C_A, C_A'\right) = \frac{r + u_s C_A'}{D_{ax}} \tag{16}$$

$$\begin{aligned} &u_s C_A(0) - D_{ax}\left.\frac{dC_A}{dz}\right|_{z=0} - u_s C_{A,feed} = 0 \quad \Rightarrow \\ &f_3\left(C_A(0), C_A'(0), C_A(L), C_A'(L)\right) = u_s C_A(0) - D_{ax} C_A'(0) - u_s C_{A,feed} = 0 \end{aligned} \tag{17}$$

$$\left.\frac{dC_A}{dz}\right|_{z=L} = 0 \quad \Rightarrow \quad f_4\left(C_A(0), C_A'(0), C_A(L), C_A'(L)\right) = C_A'(L) = 0 \tag{18}$$

With the equations in this form, the modification of SolvBVDif.m can be undertaken. To begin, the initial comment was replaced and the function definition was changed to it returns a vector containing axial positions between 0 and L and a second vector containing the corresponding conversions. The known constants from the problem statement were then entered. All of these modifications can be seen in Listing 1.

```
% Modified version of the MATLAB template file SolvBVDif.m used to model
% the tubular reactor with axial dispersion in Example 33.1 of "A First
% Course on Kinetics and Reaction Engineering."
%
function [ax_length,ax_conv] = Example_33_1_ad
    % Known quantities and constants (in consistent units)
    k = 1.2E-2; % /s
    CAfeed = 1.0e-3; % mol/m^3
    Peax = 2.0;
    us = 0.01; % m/s
    d = 0.004; % m
    L = 50*d;
    Dax = us*d/Peax;
```

Listing 1. Modified initial comment and function statement followed by entry of known constants.

The second modification occurs within an internal function named bvodes; this is where the ODEs are evaluated. This internal function is passed a value for the independent variable (as the scalar x) and corresponding values for the dependent variables (as the vector y). The modification involves using these values to evaluate functions f_1 and f_2 in equations (15) and (16) and returning them in a column vector named dydx. Doing so is quite straightforward, as can be seen in Listing 2. The dependent variables were first extracted from the vector y, giving them more meaningful names. The rate was then calculated. Finally, the functions f_1 and f_2, equations (15) and (16), were evaluated and returned as the elements of the vector, dydx.

```
    % Function that evaluates the derivatives
    function dydx = bvodes(x,y)
        CA = y(1);
        CA_prime = y(2);
        r = k*CA;
        dydx = [
            CA_prime
            (r + us*CA_prime)/Dax;
        ];
    end % of internal function bvodes
```

Listing 2. Modified version of the internal function bvodes.

The third modification occurs within an internal function named bvs; this is where the boundary conditions are evaluated. This internal function is passed the values of the dependent variables at z = 0 ($C_A(0)$ and $C_A'(0)$) in a vector named y_at_start and it is passed the values of $C_A(L)$ and $C_A'(L)$ in a vector named y_at_end. The modification involves using these values to evaluate functions f_3 and f_4 in equations (17) and (18) and returning them in a column vector named res. Doing so is quite

straightforward, as can be seen in Listing 3. The values of the dependent variables at z = 0 and z = L are first extracted from the vectors `y_at_start` and `y_at_end`, giving them more meaningful names, and then functions f_3 and f_4 are evaluated and stored as the elements of the vector `res`.

```
% Function that calculates the errors at the boundaries
function res = bvs(y_at_start,y_at_end)
    CA_inlet = y_at_start(1);
    CA_outlet = y_at_end(1);
    CA_prime_inlet = y_at_start(2);
    CA_prime_outlet = y_at_end(2);
    res = [
        us*CA_inlet - Dax*CA_prime_inlet - us*CAfeed;
        CA_prime_outlet;
    ];
end % of internal function bvs
```

Listing 3. Modified version of the internal function bvs.

The fourth modification involves creating an initial mesh of points within the reactor where the values of the dependent variables will be calculated. As can be seen in Listing 4. I chose to simply use 100 points between z = 0 and z = L. The code that solves the ODEs may change the number of mesh points, so after solving the equations the number of mesh points may not equal 100. The fifth modification involves providing a guess for the value of the dependent variables at those mesh points. You only provide a single guess for each dependent variable, and that same guess is used at all of the mesh points. As can be seen in Listing 4, I used the feed concentration of A as my guess for C_A and I used the rate at the feed concentration divided by the superficial velocity as my guess for C_A'. You may be wondering how I arrived at the latter guess. If you look at equation (11), you can see that my guess is simply the value of C_A' at the inlet to a PFR.

```
% Set up the initial mesh
x_range_low = 0.0;
x_range_high = L;
n_mesh_points = 100.;

% The next line creates an array for the independent variable
x = linspace(x_range_low,x_range_high,n_mesh_points);

% Guesses
yinit = [
    CAfeed
    k*CAfeed/us
];
```

Listing 4. Modification to set up the initial mesh and provide guesses.

The final modification occurs after the equations have been solved. In this case it involves using the result to calculate the conversion at each mesh point and then setting the return variables so they contain the z values of the mesh points and the conversions at the mesh points. These modifications can be seen in Listing 5.

```
    % Create a structure containing the mesh and guesses
    solinit=bvpinit(x,yinit);

    % Solve the odes
    result = bvp4c(@bvodes,@bvs,solinit);

    ax_length = result.x;
    ax_conv = (CAfeed - result.y(1,:))*100./CAfeed;

end % of file Example_33_1_ad.m
```

Listing 5. Modification to assign values to the return variables.

As already noted, this function is called by the function, Example_33_1, which uses the results (along with PFR results obtained by calling Example_33_1_pfr) to make the plot shown as Figure 1.