A First Course on Kinetics and Reaction Engineering

Example 38.2

Problem Purpose

This example illustrates some limitations to the use of the effectiveness factor and shows how to model an isothermal packed bed reactor by writing mole balances separately on each of the phases present in the reactor.

Problem Statement

The gas phase, heterogeneous catalytic reaction A → B takes place isothermally in a packed bed PFR with negligible pressure drop. The reaction rate per unit catalyst particle volume is second order in A, and at the operating temperature, the rate coefficient is equal to 1.72 x 10^{-2} L mol^{-1} s^{-1}. The void fraction within the packed bed is equal to 0.4. The feed to the reactor is 1 cm^3 s^{-1} of pure A, corresponding to a concentration of 1.16 mol L^{-1}. There are no temperature gradients anywhere in the reactor or in the catalyst particles, and there are no concentration gradients external to the spherical catalyst particles (diameter = 3 mm). The effective diffusion coefficient of A within the porous catalyst particles is equal to 2.7 x 10^{-7} cm^2 s^{-1}. The reactor diameter is 2.5 cm, and the conversion of A is 85%. First calculate the length of the reactor using the average of the effectiveness factors at the reactor inlet and outlet. Then calculate the reactor length using separate mole balances on the flowing gas phase and on the catalyst phase, and compare the results.

Problem Analysis

This is a reaction engineering problem involving a packed bed reactor with concentration gradients within the catalyst particles. We are told how to perform the calculations, mitigating most of the need to analyze the problem. It should be noted that an analytic expression for the effectiveness factor for a second order reaction in an isothermal, spherical catalyst particle is not available. Therefore, the effectiveness factor will have to be calculated numerically.

Problem Solution

The following quantities are specified in the problem statement: k = 1.72 x 10^{-2} L mol^{-1} s^{-1}, ε = 0.4, $\dot{V}$ = 1.0 cm^3 s^{-1} (since the total moles do not change upon reaction and the reactor is isothermal, the volumetric flow rate will be constant), $C_{A,in}$ = 1.16 mol L^{-1}, D_p = 3 mm, D_{eA} = 2.7 x 10^{-7} cm^2 s^{-1}, D_{PFR} = 2.5 cm and f_A = 0.85. We are asked to determine the reactor length, L.

We are told to first calculate the length using the average of the inlet and outlet effectiveness factors. If we are going to use an effectiveness factor, then the ideal PFR design equations can be used, except that the rate must be multiplied by the effectiveness factor to account for the concentration gradients within the catalyst particles. A PFR mole balance on A is given in equation (1) where the second order rate expression has been substituted for the rate, which is multiplied by the effectiveness factor to

account for the presence of concentration gradients. As was the case in Example 38.1, the (1 − ε) term in equation (1) is needed to convert the rate per volume of catalyst (which is what the given rate coefficient yields) to the rate per volume of reactor (which is used in the ideal PFR design equation). Specifically ε is the void volume per reactor volume, so (1 − ε) is the catalyst volume per reactor volume, and multiplying it by the rate per catalyst volume gives the rate per reactor volume. The inlet molar flow rate of A can be calculated from the given inlet volumetric flow rate and concentration, giving us the initial condition, equation (2), needed to solve equation (1). The final value of the molar flow rate of A is known from the specified conversion, as given in equation (3).

$$\frac{d\dot{n}_A}{dz} = \frac{-\pi D_{PFR}^2}{4}(1-\varepsilon)\eta\left(kC_A^2\right) \tag{1}$$

$$\dot{n}_A\left(z=0\right) = \dot{V}C_{A,in} \tag{2}$$

$$\dot{n}_A\left(z=L\right) = \dot{V}C_{A,in}\left(1-f_A\right) \tag{3}$$

Equation (1) can be solved numerically; to do so, in addition to the initial and final conditions above, code must be provided that evaluates the right-hand side of equation (1) given values for z and $\dot{n}_A$. Looking at the right-hand side of equation (1), the values of D_{PFR}, ε and k are known. The concentration of A can be calculated using equation (4), where $\dot{V}$ is also known. That leaves η as the only quantity that is needed in order to evaluate the right-hand side of equation (1). We are instructed to use the average of the effectiveness factors at the inlet and the outlet of the reactor in equation (1).

$$C_A = \frac{\dot{n}_A}{\dot{V}} \tag{4}$$

As noted in the problem analysis, it will be necessary to calculate the effectiveness factors numerically. As noted in the informational reading for this unit, a mole balance on A within the spherical, porous catalyst particle is given by equation (5) (the only differences from equation (38.11) in the informational reading are that the rate is second order and that here we are given the rate coefficient per catalyst volume, so the apparent catalyst density is not necessary - see Example 38.1). The boundary conditions, equations (6) and (7) are the same as those presented in the reading, but since there are no external concentration gradients, the surface concentration of A is equal to the bulk concentration. It is important to note that other than the bulk fluid concentration of A in boundary condition (6), the concentrations in these equations are all concentrations within the catalyst particle.

$$D_{eA}\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{dC_A}{dr}\right) = kC_A^2 \tag{5}$$

$$C_A\Big|_{r=R_{part}} = C_{A,bulk} \tag{6}$$

$$\left.\frac{dC_A}{dr}\right|_{r=0} = 0 \tag{7}$$

We know the values of the effective diffusion coefficient, rate coefficient and bulk fluid concentration at the inlet (1.16 mol L^{-1}) and outlet (15% of 1.16 mol L^{-1}, since the conversion is 85%). Therefore, equation (5) can be solved numerically for each of the two bulk fluid concentrations. Doing so will yield values for the concentration gradient at the surface of the catalyst particle. The flux in the radial direction (that is, leaving the particle) at the surface of the particle, N_A, can then be computed using equation (8). Examination of equation (8), knowing that the diffusion coefficient is positive and the concentration decreases as one moves toward the center of the particle (i. e. the derivative in equation (8) is positive), shows that N_A is negative, indicating that the flux is directed into the particle. Thus, the rate at which A enters the is simply $-N_A$ times the surface area of the particle. The rate at which A is consumed within the particle is $-r_A$ times the volume of the particle. At steady state, the rate at which A enters the particle must equal the rate at which A is consumed within the particle, equation (9). The effectiveness factor, by definition, is the actual rate at which A is consumed within the particle divided by the rate at which A would be consumed within the particle if there were no concentration gradients (i. e. if C_A = $C_{A,bulk}$ everywhere in the particle), equation (10).

$$N_A = D_{eA}\left(-\frac{dC_A}{dr}\right)\Bigg|_{r=R_{part}} \tag{8}$$

$$\frac{4\pi}{3}R_p^3\left(-r_A\right) = 4\pi R_p^2\left(-N_A\right) \quad \Rightarrow \quad r_A = \frac{3}{R_p}\left(N_A\right) \tag{9}$$

$$\eta = \frac{-r_A}{-r_{A,\text{ no gradient}}} = \frac{-\frac{3}{R_p}\left(N_A\right)}{kC_{A,bulk}^2} \tag{10}$$

Once the effectiveness factor has been calculated in this way at the inlet and outlet bulk concentrations and averaged, equation (1) can be solved numerically. Doing so, one finds that η = 0.059 at the inlet and η = 0.15 at the outlet giving an average value of η = 0.102. With that average value of the effectiveness factor, the solution of equation (1) shows that the required reactor length is L = 945 cm.

The problem specifies that we should also calculate the required reactor length by solving mole balances on A in the fluid phase and in the catalyst phase. No reaction takes place in the fluid phase due to the absence of catalyst, so there is no rate term. However, at any position in the reactor, some fluid phase A leaves the fluid phase and enters the catalyst phase. We have seen above that the flux of A into the catalyst, $-N_A$, is easily obtained from a mole balance on A in the catalyst phase. The rate at which A leaves the fluid phase, per unit volume of the reactor is then equal to that flux times the catalyst surface area per reactor volume, a_V. Thus, a mole balance on A in the fluid phase takes the form of equation (11).

The catalyst area per reactor volume is simply the catalyst area per catalyst volume times the catalyst volume per total reactor volume, (1 − ε, as explained above), equation (12).

$$\frac{d\dot{n}_A}{dV_{PFR}} = \frac{d\dot{n}_A}{\left(\pi D_{PFR}^2 / 4\right)dz} = -A_V\left(-N_A\right) \quad \Rightarrow \quad \frac{d\dot{n}_A}{dz} = \frac{\pi D_{PFR}^2 A_V}{4}\left(N_A\right) \tag{11}$$

$$A_V = \left(\frac{A_{particle}}{V_{particle}}\right)\left(\frac{V_{particle}}{V_{PFR}}\right) = \left(\frac{4\pi R_p^2}{\frac{4\pi R_p^3}{3}}\right)\left(1-\varepsilon\right) = \frac{3}{R_p}\left(1-\varepsilon\right) \tag{12}$$

Equation (11) can be solved numerically. To do so, one must provide an initial condition, a final condition and code that evaluates the right hand side of equation (11), given values for z and $\dot{n}_A$. The initial condition and final condition are the same as before, equations (2) and (3). The reactor diameter is known and A_V can be calculated using equation (12), but the code to evaluate the right hand side of equation (11) will need to calculate N_A. In order to do so, that code will, in turn, need to solve equation (5), the mole balance on A in the catalyst phase. The numerical solution of equation (5) has already been described; the only difference here is that it is being solved within the code we write to evaluate the right hand side of equation (11).

Upon writing the code, as just described, and solving equation (11) numerically, one finds that the required length of the reactor is L = 936 cm. Thus, using the average of the inlet and outlet effectiveness factors in the ideal PFR design equation did not introduce much error (945 vs. 936 cm). In addition, it over-predicted the required reactor length, so if those calculations had been used, the reactor would have exceeded the specified conversion by a small amount, causing no problems.

The effectiveness factor will also be affected by temperature gradients, or even by a temperature change along the length of the reactor (e. g. due to adiabatic operation) in the absence of gradients between the fluid and the catalyst or within the catalyst. In such cases, using an average effectiveness factor in the PFR design equations could lead to a larger error, and simultaneously solving the fluid phase and catalyst phase design equations may be preferred.

Calculation Details Using MATLAB

There are several ways one could structure the code for solving this problem. If you examine the solution given above, you see that in both cases, the primary equation to be solved is an initial value ODE, either the ideal PFR mole balance incorporating an effectiveness factor or a mole balance on the fluid phase. In both cases, it is necessary to solve the catalyst phase mole balance and use the result to calculate either the flux of reactant at the catalyst surface or the efficiency factor. To reduce the replication of code, I opted to use four MATLAB functions. The first, Example_38_2, is where the known quantities given in the problem statement are entered, after which the second, Example_38_2a, is called to calculate the reactor length by solving the ideal PFR mole balance incorporating an effectiveness factor,

and the third, Example_38_2b, is called to calculate the reactor length by solving a mole balance on the fluid phase. Each of the latter two function call the fourth function, Example_38_2_cat_bal, which calculates the reactant flux in the radial direction at the surface of the catalyst particles and the effectiveness factor by solving a mole balance on the catalyst phase.

One issue that arises when you use several functions like this, is that each of the functions typically requires the values of several of the constant quantities given in the problem statement. One approach is to enter all these quantities in one of the functions, and then copy and paste the exact same code into each of the others. This is not a good programming practice. If one quantity changes or is typed incorrectly, it has to be corrected in four different place. For this reason, I created a structure array named `given` in the main function file. This structure array contains each of the given quantities. For example, the given rate coefficient, k, is stored as `given.k`. The structure array can then be passed to each of the other functions as an argument, giving those functions access to all the given quantities. Then, if a value needs to be changed, only one change is required, and it is ensured that all of the functions are using the same set of values for the given quantities.

With that background, the main MATLAB function for solving this problem is trivially simple, as shown in Listing 1. It stores all of the given quantities (in consistent units) in the structure array and then sequentially calls the two functions that calculate the reactor length. The structure array is passed to each function as an argument. The file, Example_38_2.m accompanies this solution.

```
function Example_38_2
%Example_38_2 Code used in the solution of Example 38.2 of AFCoKaRE

    % Known quantities and constants entered in a structure array
    given = struct;
    given.k = 17.2; % cm^3/mol/s
    given.eps = 0.4;
    given.VFR = 1.; % cm^3/s
    given.CA0 = 1.16E-3; % mol/cm^3
    given.Rp = 0.3/2; % cm
    given.DA = 2.7E-7; % cm^2/s
    given.fA = 0.85;
    given.D = 2.5; % cm

    % Call Example_38_2a to solve using the average effectiveness factor
    L_avg_eta = Example_38_2a(given)

    % Call Example_38_2b to simultaneously solve the fluid phase and
    % catalyst phase mole balances
    L_phase_balances = Example_38_2b(given)

end % of Example_38_2
```

Listing 1. Full listing of the MATLAB function Example_38_2.

Next let's consider the function Example_38_2_cat_bal which calculates the reactant flux in the radial direction at the surface of the catalyst particles and the effectiveness factor by solving the mole balance on the catalyst phase. The catalyst phase mole balance, equation (5) is a second-order ordinary differential equation. Examination of the boundary conditions given in equations (6) and (7) shows that one boundary condition applies at r = 0 and the other applies at r = R_p, making this a boundary-value ODE problem. Supplemental Unit S6 describes how to solve boundary-value ODEs using MATLAB, and it provides two template files that can be modified to do so. The use of either template file requires that higher-order ODEs first be re-written as sets of first order ODEs. To do so, the chain rule for differentiation is first applied to equation (5), resulting, after re-arrangement, in equation (13). Two new dependent variables, y_1 and y_2, are then defined as in equations (14) and (15). Taking the derivative of equation (14) and substituting equation (15) into the result gives the first of the two required first order ODEs, equation (16). Substitution of equations (14) and (15) into equation (13) gives the second of the two required first order ODEs, equation (17).

$$\frac{d^2C_A}{dr^2} = -\frac{2}{r}\frac{dC_A}{dr} + \frac{kC_A^2}{D_{eA}} \tag{13}$$

$$y_1 = C_A \tag{14}$$

$$y_2 = \frac{dC_A}{dr} \tag{15}$$

$$\frac{dy_1}{dr} = y_2 \tag{16}$$

$$\frac{dy_2}{dr} = -2\frac{y_2}{r} + \frac{ky_1^2}{D_{eA}} \tag{17}$$

The boundary equations must also be re-written in terms of the new dependent variables. Substituting equations (14) and (15) into equations (6) and (7) gives the required boundary conditions in equations (18) and (19).

$$y_1\left(R_p\right) = C_{A,bulk} \tag{18}$$

$$y_2\left(0\right) = 0 \tag{19}$$

Examining equation (17) shows that there is a singularity at r = 0, so the template file SolvBVDifS.m from Supplemental Unit S6 can be modified and used to solve the catalyst phase mole balance numerically. A copy of SolvBVDifS.m was saved as Example_38_2_cat_bal.m for this purpose. Recall that we want Example_38_2_cat_bal to solve the catalyst phase mole balance and use the result to calculate the reactant flux in the radial direction at the surface of the catalyst particles and the effectiveness factor. To do this, the value of the concentration of A at the surface will be needed in addition to the quantities given in the problem statement. Therefore function definition was changed so that the structure array

containing the quantities given in the problem statement and the reactant concentration at the catalyst surface are passed to the function as arguments and so that it returns the desired flux and effectiveness factor. At the same time, the long initial comment was deleted, and a new comment describing the purpose of the function was added at the top of the function. The results of these modifications are shown in Listing 2.

```
function [flux, eta] = Example_38_2_cat_bal(given,CAs)
%Example_38__cat_bal Solve the catalyst phase mole balance for Example 38.2
%of AFCoKaRE and return the rate

    % Structure array given contains known quantities and constants
```

Listing 2. Modification of the function declaration in the template file SolvBVDiffS.m.

The next things in the template file are definitions of two internal functions named `bvodes` and `bvs`. The first of these functions, `bvodes`, must be modified so that given values for the dependent (y_1 and y_2) and independent (r) variables, it returns a vector containing the values of the right-hand sides of the ODEs, ***excluding the terms associated with the singularity***. In this case, everything needed in order to do that is available in the structure array, `given`. The resulting modified code is shown in Listing 3.

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

Listing 3. Internal function bvodes.

The internal function, `bvs`, is passed the values of the dependent variables at the two ends of the range of the independent variable. Hence, in the present case, `y_at_start` is a vector containing the values of y_1 and y_2 at r = 0 and `y_at_end` is a vector containing the values of y_1 and y_2 at $r = R_p$. This function must be modified so that it returns the corresponding error in each of the boundary conditions. In the present case, if the boundary condition in equation (18) is satisfied, $y_1(R_p) - C_{A,bulk}$ should equal zero. If that term does not equal zero, its value is the error in the first boundary condition. Similarly, if the boundary condition in equation (19) is satisfied, $y_2(0)$ should equal zero, and if it does not, its value is the error in the second boundary condition. Thus, the internal function, `bvs`, is modified to return these values as shown in Listing 4.

```
% Function that calculates the errors at the boundaries
function res = bvs(y_at_start,y_at_end)
    res = [
        y_at_start(2)
        y_at_end(1) - CAs
    ];

end % of internal function bvs
```

Listing 4. Internal function bvs.

The template file must next be modified to discretize the range of the independent variable by specifying its lower limit and its upper limit along with the number of elements to divide the range into. Here I arbitrarily chose to divide the range into 20 parts. The modified code is shown in Listing 5.

```
x_range_low = 0.;
x_range_high = given.Rp;
n_mesh_points = 20;

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

Listing 5. Modifications for discretization in the radial direction.

The next necessary modification involves providing a guess for each of the dependent variables. Even though the radial direction has been divided into 20 sections, we only need to provide a single guess for each of the dependent variables. I noted that if the concentration of A goes to zero at the center of the particle, then the average concentration of A would be one-half of the surface concentration and that the derivative of the concentration with respect to r would equal the surface concentration divided by the particle radius. I then used these values as the guesses, as shown in Listing 6.

```
% Guesses
yinit = [
    CAs/2.
    CAs/given.Rp
];

% The next line creates a structure containing the mesh and guesses
solinit=bvpinit(x,yinit);
```

Listing 6. Specification of guesses for the dependent variables.

Recall that the terms associated with the singularity were not included when the internal function, bvodes, evaluated the right hand side of the boundary value ODEs being solved. The reason is that the MATLAB solver used by the template file requires that the ODEs be written in the form shown in equation (20) and that the internal function, bvodes, returns only the values of the functions, f_i in those equations.

The last modification needed before the equations can be solved is to define the singularity matrix. Comparing equations (16) and (17) to equation (20) shows that they are already in the proper form with S_{11}, S_{12}, S_{21} and S_{22} defined as in equations (21) and (22). The corresponding modification of the template file is shown in Listing 7.

$$\frac{dy_i}{dx} = \sum_{\text{all } j} S_{ij} \frac{y_j}{x} + f_i\left(x, \underline{y}\right) \tag{20}$$

$$S_{1,1} = 0; \quad S_{1,2} = 0 \tag{21}$$

$$S_{2,1} = 0; \quad S_{2,2} = -2 \tag{22}$$

```
    S = [
        0,0
        0,-2
    ];
```

Listing 7. Specification of the singularity matrix.

At this point, the solver can be used to solve the boundary value ODEs. It returns a structure, as described in Supplemental Unit S6 and the MATLAB documentation. The member of that structure named y is a matrix where the first row contains the values of y_1 at each of the mesh points, the second row contains the values of y_2 at each of the mesh points, and so on. We wish to calculate the flux of A in the radial direction at the particle surface using equation (8). Since y_2 is the derivative of C_A with respect to r, the last element in the second row of the matrix y is the value of that derivative at r = R_p. The flux can thus be calculated by multiplication by the negative of the diffusion coefficient.

We also wish to calculate the effectiveness factor. This can be accomplished using equation (10). Having calculated the flux and noting that the concentration of A in the bulk fluid is the same as its concentration at the surface (no external gradients), we have everything needed to calculate the effectiveness factor. The final modification of the template file to calculate the flux and effectiveness factor is shown in Listing 8.

```
    % Solve the ODEs
    options = bvpset('SingularTerm',S);
    soln = bvp4c(@bvodes,@bvs,solinit,options);

    lastPoint = length(soln.x);
    flux = -soln.y(2,lastPoint)*given.DA; % solution, equation (8)
    eta = -3*flux/(given.Rp*given.k*CAs^2); % solution, equation (10)

end % of Example_38_2a
```

Listing 8. Calculation of the flux and effectiveness factor.

Now that we have a function that will calculate the flux of reactant at the particle surface and the effectiveness factor, we can write the code to solve the initial value ODE that corresponds to the ideal PFR mole balance modified to include an effectiveness factor. The ODE to be solved is given in equation (1), with the initial condition specified in equation (2) and the final value given in equation (3). Equation (3) shows that the final value of the dependent variable is known, so the ODE can be solved numerically by modifying the MATLAB template file SolvIVDifD.m from Supplemental Unit S5.

As noted above, we will pass the structure array containing the values of the quantities given in the problem statement to this function and we want it to return the length of the reactor. Therefore a copy of the template file SolvIVDifD.m was saved as Example_38_2a.m and modified for this problem. The long comment at the top of the template file was removed. The function declaration was changed to indicate the structure array as an argument and the reactor length as the return value. A short comment was added at the top of the file to give its purpose.

In addition to the quantities given in the problem statement, values for the effectiveness factor at the reactor inlet and outlet will be needed in order to solve the ODE in equation (1). Specifically the effectiveness factor at the reactor inlet and outlet will be averaged and used in equation (1). Since each of these are constants, I entered code to calculate them next. Having written a separate function to calculate the effectiveness factor, all that needs to be done is to calculate the surface concentration at the inlet and outlet, call Example_38_2_cat_bal to calculate the corresponding effectiveness factors, and take their average. All of the modifications mentioned so far are shown in Listing 9.

```
function L = Example_38_2a(given)
%Example_38_2a Solve AFCoKaRE Example 38.2 using an average effectiveness

    % Structure array given contains known quantities and constants

    % Average the effectiveness factors at the inlet and outlet
    CAin = given.CA0;
    [flux, eta_in] = Example_38_2_cat_bal(given,CAin);
    CAout = (1-given.fA)*CAin;
    [flux, eta_out] = Example_38_2_cat_bal(given,CAout);
    eta_avg = (eta_in + eta_out)/2.;
```

Listing 9. Function definition, initial comments and calculation of the average effectiveness factor.

The template file was next modified to provide the initial and final conditions. In this problem, the initial value of z is z = 0 and the initial value of $\dot{n}_A$ is calculated according to equation (2). The final condition, equation (3), is specified by making two modifications. First, a large final value is specified for the ***independent*** variable (`tf` in the MATLAB code) so that the known final value of the ***dependent*** variable, $\dot{n}_A(L)$, is reached first. Second, the internal function named stop is modified so that the variable `stop_when` will equal zero when the final condition is reached. Subtracting the molar flow rate of A at the reactor outlet from both sides of equation (3) yields equation (23). Clearly, the right-hand side of equation

(23) is equal to zero when the final condition is satisfied, so the variable, stop_when, is set equal to the right-hand side of equation (23). These modifications are shown in Listing 10.

$$0 = \dot{V}C_{A,in}\left(1 - f_A\right) - \dot{n}_A\left(z = L\right) \tag{23}$$

```
% Initial values
t0 = 0.0;
z0 = [
    CAin*given.VFR
]; % solution, equation 2

tf = 960.;
%tf = 96.6;
options = odeset('Events',@stop);
[t, zz, te, ze, ie] = ode45(@odeqns,[t0, tf],z0,options);

% Function that provides the integration stopping criterion
function [stop_when, isterminal, direction] = stop(t,z)
    isterminal = 1;
    direction = 0;

    stop_when = z(1) - (1-given.fA)*CAin*given.VFR;
end % of internal function stop
```

Listing 10. Modifications required to specify the initial and final conditions.

The internal function, odeqns, must be modified so that it returns the right-hand side of equation (1). All the quantities necessary for doing so are available, leading to the code shown in Listing 11. The only other necessary modification (not shown here) is to set the final value of the independent variable equal to the return value, L.

```
% Function that evaluates the ODEs
function dzdt = odeqns(t,z)
    rate = given.k*z(1)^2/given.VFR^2;
    if (z(1) < 0)
        rate = 0.0;
    end % if (z(1) < 0)
    dzdt = [
        -pi()*given.D^2/4*eta_avg*(1-given.eps)*rate;
    ]; % solution, equation 1
end % of internal function odeqns
```

Listing 11. Modification of internal function odeqns for the solution of the ideal PFR mole balance incorporating an effectiveness factor.

The function for solving the fluid phase and catalyst phase mole balances, Example_38_2b, is quite similar to that for solving the ideal PFR model incorporating an effectiveness factor, Example_38_2a. The only differences are, first, it is not necessary to calculate the average effectiveness factor at the beginning

of the function, and, second, the initial value ODE being solved is given by equation (11). The function can be created by making a copy of Example_38_2a.m and saving it as Example_38_2b.m. The function declaration must be changed accordingly, and the initial comment should be modified. The code for calculating the average effectiveness factor can then be deleted and replaced with a single line of code that calculates Av according to equation (12). The results are shown in Listing 12.

```
function L = Example_38_2b(given)
%Example_38_2b Solve AFCoKaRE Example 38.2 using phase balances

    % Structure array given contains known quantities and constants
    Av = 3./given.Rp*(1-given.eps); % solution, equation (12)
```

Listing 11. Function definition, initial comments and calculation of A_v.

The only other change is in the internal function, odeqns, which must be modified so it returns the right hand side of equation (11). Looking at that equation, it can be seen that all the necessary quantities are available except for the flux. Since a separate function was written to calculate the flux, all that is necessary is to call that function and then evaluate equation (11). This is shown in Listing 12.

```
    % Function that evaluates the gas phase mole balance ODE
    function dzdt = odeqns(t,z)
        CAs = z(1)/given.VFR;
        % Calculate the flux of A by solving the catalyst phase mole
        % balance on A
        [NA, eta] = Example_38_2_cat_bal(given,CAs);
        % Evaluate the gas phase mole balance on A
        dzdt = [
            pi()*given.D^2/4*Av*NA % solution, equation 11
        ];
    end % of internal function odeqns
```

Listing 12. Modification of internal function odeqns for the solution of the fluid and catalyst phase mole balances.

At this point, if all four files are located in the MATLAB search path, the problem can be solved by typing Example_38_2 at the MATLAB command prompt. Doing so produces the output shown in listing 13.

```
>> Example_38_2

L_avg_eta =
  944.5968

L_phase_balances =
  936.2184
```

Listing 13. Output from the execution of Example_38_2