A First Course on Kinetics and Reaction Engineering
Supplemental Unit S6. Solving Boundary Value Differential Equations

Defining the Problem

This supplemental unit describes how to numerically solve a set of coupled, first order boundary value ordinary differential equations (ODEs) of the form shown in equation (1) or using matrix notation as in equation (2). It also shows how second order boundary value ODEs can be re-written as sets of first order boundary value ODEs with the proper form. The values of some of the dependent variables must be known at $x = a$, and the values of the other dependent variables must be known at $x = b$. A general form for the boundary conditions is given in equation (3), or in matrix form as equation (4).

$$\begin{aligned} &\frac{dy_1}{dx} = f_1\left(x,y_1,\cdots,y_n\right); \qquad a \le x \le b \\ &\vdots \\ &\frac{dy_n}{dx} = f_n\left(x,y_1,\cdots,y_n\right); \qquad a \le x \le b \end{aligned} \tag{1}$$

$$\frac{d\underline{y}}{dx} = \underline{f}\left(x,\underline{y}\right); \qquad a \le x \le b \tag{2}$$

$$\begin{aligned} &g_1\left(y_1(a),\cdots,y_n(a),y_1(b),\cdots,y_n(b)\right) = 0 \\ &\vdots \\ &g_n\left(y_1(a),\cdots,y_n(a),y_1(b),\cdots,y_n(b)\right) = 0 \end{aligned} \tag{3}$$

$$\underline{g}\left(\underline{y}(a),\underline{y}(b)\right) = 0 \tag{4}$$

This supplemental unit also describes how to solve boundary-value ODEs that can be written in the form of equation (5), or in the matrix form of equation (6), where the coefficients, S_{ij}, are constants and at least one of the S_{ij} is non-zero. The boundary equations are still given by equation (3) or, equivalently, equation (4), but with a equal to zero. Equations (5) or (6) have a singularity at the boundary where $x = 0$, so $\underline{\underline{S}}$ is called the singularity matrix.

$$\begin{aligned} &\frac{dy_1}{dx} = \frac{S_{11}y_1}{x} + \cdots + \frac{S_{1n}y_n}{x} + f_1\left(x,y_1,\cdots,y_n\right); \qquad 0 \le x \le b \\ &\vdots \\ &\frac{dy_n}{dx} = \frac{S_{n1}y_1}{x} + \cdots + \frac{S_{nn}y_n}{x} + f_n\left(x,y_1,\cdots,y_n\right); \qquad 0 \le x \le b \end{aligned} \tag{5}$$

$$\frac{d\underline{y}}{dx} = \frac{\underline{\underline{S}}\underline{y}}{x} + \underline{f}\left(x,\underline{y}\right); \qquad 0 \le x \le b \tag{6}$$

Overview of the Numerical Method

The approach to solving boundary value ODEs that will be presented here requires that all ODEs in the set are first order ordinary differential equations. This is not a significant limitation, however, because a higher order ODEs can be converted into a set of first order ODEs. In kinetics and reaction engineering, one rarely encounters an ODE greater that second order, so I'll only describe conversion of a second order ODE into a set of two first order ODEs here, but the method applies equally to higher order ODEs. As illustrated in Examples 1 and 2 of this supplemental unit, the following steps are involved:

- Define a new variable that is equal to the first derivative of the original dependent variable
    - This definition becomes one of the two first order ODEs in the set that replaces the second order ODE
- In the original second order ODE and in the associated boundary conditions
    - Replace any occurrence of the <u>second</u> derivative of the <u>original</u> dependent variable with the <u>first</u> derivative of the <u>new variable</u>
        - This results in a first order ODE that becomes the other first order ODE in the set that replaces the original second order ODE
    - Replace any occurrence of the <u>first derivative</u> of the <u>original</u> dependent variable with the <u>new variable</u>

The numerical solution process for a set of boundary value ODEs begins by dividing the range of the independent variable into a number of smaller, contiguous intervals. This is sometimes referred to as mesh generation, and the original endpoints of the range, together with the endpoints of the newly created intervals are called the mesh points. The intervals do not need to be equally sized, but in the present discussion they will be. The number of intervals will be one less than the number of mesh points.

Recall that the solution to an ODE is an expression for the variation of its dependent variable with respect to the independent variable. The next step in the numerical solution of a set of boundary value ODEs is to choose, for each of the intervals created by mesh generation, a mathematically convenient function to serve as an approximation to the true solution within that interval. The functional form of the approximating function is usually chosen to be the same in each interval, but this is not required. The MATLAB function `bvp4c` uses a cubic polynomial, equation (7), to approximate the true solution in each of the intervals created by mesh generation. That is, in equation (7), y_i represents an approximation to the true value of the dependent variable, y, within the range of x corresponding to interval i. The constant coefficients a_i, b_i, c_i and d_i in equation (7) are initially unknown. Each interval created by mesh generation will have its own, unique set of values of a_i, b_i, c_i and d_i for the first dependent variable, its own, unique set of values of a_i, b_i, c_i and d_i for the second dependent variable, and so on. Clearly, if all these values of a_i, b_i, c_i and d_i in each interval and for each dependent variable are known, the the approximate value of y can be computed anywhere within range of x. Therefore, the numerical solution of a set of boundary value ODEs boils down to finding the values of all the coefficients that appear in the approximating polynomials in all the intervals that span the range of x.

$$y_i = a_i x^3 + b_i x^2 + c_i x + d_i \tag{7}$$

Note that the approximating functions do not have to be cubic polynomials; they could be any order polynomial (straight line, quadratic, etc.) or they could be some other non-polynomial expression. Note also that different forms of approximating functions could be used in different intervals. Much of the variation in methods for solving boundary value ODEs numerically arises from the functions that are selected as the approximating functions. Nonetheless, if a mesh is chosen that divides the range of x into N intervals, solving the ODEs becomes equivalent to finding the N values of a_i, b_i, c_i and d_i for interval for each dependent variable in the set of ODEs. For convenience, let's consider the solution of a single ODE to show how these unknown coefficients are determined. In that case, there will be $4N$ unknown coefficients, and so we need $4N$ equations containing those coefficients that can be solved to find their values.

Figure 1 may be helpful in discussing the origin of these $4N$ equations. Figure 1 shows the range between x = 0 and x = x_f divided into 7 intervals. The true solution is represented by the thick gray curve in the figure. Within each interval, the true solution is approximated using a polynomial; the polynomials are represented by dashed lines. Each different colored dashed line is a different polynomial.

We want the solution we obtain to be continuous across the whole range of x. Therefore, at each mesh point where two approximating polynomial functions meet, we must require that the values of y predicted by the two polynomials meeting at that point be equal. Referring to Figure 1, notice that the green dashed polynomial and the yellow dashed polynomial meet at x = x_2. In order for the approximating solution to be continuous, the value of y at x = x_2 given by the green dashed polynomial must equal the value of y at x = x_2 given by the yellow polynomial. If equation (7) gives the functional form of the approximating function in each interval, this leads to equation (8) where a_2, b_2, c_2 and d_2 are the coefficients for the green dashed polynomial (in the second interval) and a_3, b_3, c_3 and d_3 are the coefficients for the yellow dashed polynomial (in the third interval). Notice, that since we defined the mesh, we know the value of x_2, and therefore the only unknown quantities in equation (8) are the eight coefficients from the approximating functions. The same must be true for the red and green dashed polynomials at x = x_1, for the yellow and orange dashed polynomials at x = x_3, and so on for each of the internal mesh points. Thus, if there are N intervals, this will lead

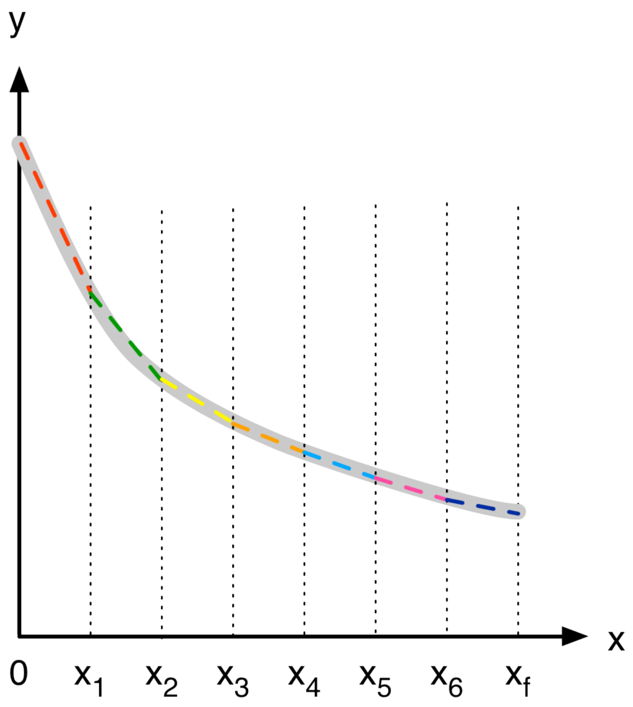

Figure 1. The thick gray line represents the true solution; the colored, dashed lines represent approximating functions.

to N-1 equations of the form of equation (8) since there are N-1 internal mesh points where two approximating functions meet.

$$a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2 = a_3x_2^3 + b_3x_2^2 + c_3x_2 + d_3 \tag{8}$$

Remember that the ODEs will all be first order ODEs, and as a consequence, there will be one boundary condition for each dependent variable. In terms of Figure 1, this means that the value of y will be specified either at x = 0 or at x = x_f. If the boundary condition is specified at x = 0, then the red-dashed polynomial must give the specified value of $y(0)$, which leads to equation (9) where $y(0)$ is known. If the boundary condition is specified at x = x_f, then the purple dashed polynomial must give the specified value of $y(x_f)$, which leads to equation (10). Thus, the boundary condition for y gives one more equation, either equation (9) or equation (10) depending upon whether the boundary condition for y was specified at x = 0 or at x = x_f. In total we now have N equations containing the unknown coefficients, but we need $4N$ equations.

$$y(0) = a_1(0)^3 + b_1(0)^2 + c_1(0) + d_1 \quad \Rightarrow \quad y(0) = d_1 \tag{9}$$

$$y(x_f) = a_7x_f^3 + b_7x_f^2 + c_7x_f + d_7 \tag{10}$$

To generate the $3N$ additional equations we need, we first note that within interval i, the derivative of y with respect to x can be approximated by taking the derivative of equation (7), leading to equation (11). We know that within interval i, the derivative should satisfy the ODE we are trying to solve, equation (12). We need to remember that equation (11) gives only an approximation to the true value of the derivative. However, since equation (12) contains 3 undetermined coefficients, we are free to require equation (11) to exactly satisfy equation (12) at any 3 points within interval i. This will give us 3 additional equations that apply to this interval. We call these three points *collocation points*. It is convenient to choose the two endpoints and the midpoint of the interval as the collocation points. Since the values of x corresponding to the mesh points are known, the values of x corresponding to the collocation points are also known.

$$\left(\frac{dy}{dx}\right)_i = 3a_ix^2 + 2b_ix + c_i \tag{11}$$

$$\frac{dy}{dx} = f(y,x) \tag{12}$$

If we substitute equation (11) into equation (12) in place of the derivative and we substitute equation (7) into equation (12) in place of y and then evaluate the resulting equation at one of the collocation points, we obtain equation (13) where x_{cp} denotes a collocation point and where the only unknown quantities are the coefficients from the approximating function. Note that there are three collocation points in each interval, giving us three equations for each interval, and since there are N intervals, this gives us the additional $3N$ equations that are needed in order to determine the values of the undetermined coefficients in the approximating functions.

$$3a_i x_{cp}^2 + 2b_i x_{cp} + c_i = f\left(\left(a_i x_{cp}^3 + b_i x_{cp}^2 + c_i x_{cp} + d_i\right), x_{cp}\right) \tag{13}$$

A few additional points should be noted. First, the function f will not necessarily be linear in y, and therefore, equation (13) will not necessarily be linear in the undetermined coefficients. In general, where we are solving a <u>*set*</u> of ODEs, there will be a different function, f, for each dependent variable, and each of those functions could include <u>*all*</u> of the dependent variables in the set being solved. That is, in place of equation (12) there will be a set of equations of the form given in equation (14) where N_{ODEs} is equal to the number of ODEs in the set (and therefore the number of dependent variables) and j ranges from 1 to N_{ODEs}. Once again, the functions, f_j, need not be linear in the y's.

$$\frac{dy_j}{dx} = f_j\left(y_1, y_2, \cdots, y_j, \cdots, y_{N_{ODEs}}, x\right) \tag{14}$$

Hence, the numerical solution of a set of boundary value ODEs ultimately involves solving a set of non-linear algebraic equations to determine the values of the coefficients appearing in the polynomial or other functions used to approximate the true solution in each interval of the mesh. Because these equations are nonlinear, they too will be solved numerically. Again, there are very many variations on the general approach outlined here for solving sets of boundary value ODEs. Interested readers should take a course on numerical methods or consult a textbook devoted to the subject. The solution of the set of nonlinear algebraic equations to find the values of the coefficients will typically require an initial guess for the values of those coefficients, or an initial guess that can be used to generate a guess for those coefficients.

There is another issue that may be encountered when solving sets of boundary value ODEs numerically. If the ODEs being solved contain the reciprocal of the independent variable and if the range of the independent variable includes zero, then the equation will blow up at that point. This is a type of mathematical singularity. Generally, singularities can cause numerical methods to fail, but there is one singularity of this kind that is very commonly encountered in physics and engineering problems, including problems from kinetics and reaction engineering. The singularity occurs when one of the endpoints of the range of the independent variable corresponds to zero, that is, one of the boundaries is where the independent variable is equal to zero. MATLAB and some other mathematics programs are able to solve sets of boundary value ODEs that contain such a singularity as long as each of the ODEs can be written in the form shown in equation (5), or equivalently (6), with the coefficients, S_{ij}, all being constants. Taken together, these coefficients, S_{ij}, form a matrix which is sometimes called the singularity matrix. The details involved in solving equations with singularities like these will not be discussed here.

MATLAB Template Files

The built-in MATLAB function, bvp4c, can be used to solve set of boundary value ODEs using an approach like the one described here. When solving sets of equations such as in equation (1) or (2), bvp4c is called with three arguments. The first two arguments are the names (technically function handles) of two user-supplied functions. The first user supplied function must take a scalar, x, and a column vector, y, and return the values of the functions, $\underline{f}$, in equation (1) or (2) as a column vector. The second user-supplied function takes the values of $\underline{y}$ at each of the two boundaries and returns the error, or residual, in each of the boundary conditions as a column vector. The third argument to bvp4c is a structure. It contains the values of x corresponding to the mesh points and a guess for the value of each dependent variable, y_i. The latter guesses are a single value for each dependent variable; that same value is used at each mesh point. (It is possible to specify different guesses at each mesh point; see the MATLAB documentation). MATLAB provides another built-in function, bvpinit, that will create the structure to pass to bvp4c, given column vectors containing the mesh points and the guesses.

When solving sets of boundary value ODEs with singularities of the form shown in equations (5) and (6), an additional argument is passed to bvp4c. That argument is a structure that can be used to configure several different options (again, see the MATLAB documentation). In the present case, it is used to inform bvp4c that the ODEs are of the form shown in equations (5) and (6), and to pass the singularity matrix, $\underline{\underline{S}}$, to it. Again, MATLAB provides a built-in function, bvpset, that will create the structure to pass to bvp4c, given the matrix S.

Using the MATLAB ODE solver, bvp4c, is not particularly difficult, but it does require care to set all the arguments and function calls properly. This can be distracting if you are required to do it every time you solve a set of boundary value ODEs. Therefore, two MATLAB template files named SolvBVDif.m and SolvBVDifS.m are provided with this supplemental unit. The names are intended as mnemonic for **Solv**e **B**oundary **V**alue **Dif**ferential equations. SolvBVDif.m is used when solving equations like (1) and (2) that do not have a singularity while SolvBVDifS.m is used when solving equations like (5) and (6) that *do* have a singularity (and hence the final S in the name). These template files handle most of the programming details so that you can focus on the kinetics and reaction engineering aspects of the problems you solve, and not on computer programming.

The template files do require some modifications each time they are used. A set of step-by-step instructions for making these modifications for each of the template files are provided with this supplemental unit. Example S6.1 illustrates how to modify and use the SolvBVDif.m to solve a problem where the ODEs do not have a singularity, and Example S6.2 illustrates how to modify and use SolvBVDifS.m when there is a singularity of the form shown in equations (5) and (6).