A First Course on Kinetics and Reaction Engineering
Supplemental Unit S2. Solving Non-Differential Equations

Defining the Problem

This supplemental unit describes how to solve a set of non-differential equations numerically. For the purpose of the present discussion, the equations will be written in the form given in equation (1), or in the vector form given in equation (2). The number of equations, or equivalently, the number of functions in f must be equal to the number of unknown variables in $\underline{z}$. The functions in f may not contain derivatives or integrals. The goal in solving the equations is to find the values of the unknown variables, $\underline{z}$, that satisfy the equations (1) or (2).

$$\begin{aligned} 0 &= f_1\left(z_1, z_2, \cdots, z_n\right) \\ 0 &= f_2\left(z_1, z_2, \cdots, z_n\right) \\ &\vdots \\ 0 &= f_n\left(z_1, z_2, \cdots, z_n\right) \end{aligned} \tag{1}$$

$$0 = \underline{f}\left(\underline{z}\right) \tag{2}$$

Information and Data Required for Numerical Solution

To numerically solve a system of non-differential equations like those in equation (1) or (2), you will need to use mathematics software that provides one or more non-linear equation solvers. Irrespective of the particular brand of software you use, you typically will need to provide two things as input to that software:

- Code that evaluates the functions, f_1 through f_n, given values of the unknown variables, z_1 through z_n
- A guess for the solution, that is, for the values of z_1 through z_n that cause all of the functions, f_1 through f_n, to equal zero.

At the minimum, once the software has solved the set of equations, you will be given a set of values for the variables, z_1 through z_n, that cause all of the functions, f_1 through f_n, to equal zero (to within some very small tolerance). That is, it will give you a solution to the equations. You should note that sets of equations like equations (1) or (2) can have multiple solutions. Most numerical methods will only find one of those solutions. The guess that you provide as input determines which solution the method finds. Hence, if the solution returned to you by the software does not make physical sense, you should change the guess that you provide as input and solve the equations again. Similarly, if you are looking for multiple solutions, then you should solve the equations multiple times providing different guesses as input each time.

Overview of the Numerical Method

The most common approach to numerically solving equations like equations (1) involves repeatedly approximating them as linear equations. This approach is favored because it is quite easy to solve linear equations, and it is equally easy to automate the process for doing so in a computer code. As noted above, you are required to provide a guess, call it $\underline{z}_0$, for the solution when you solve equations like equations (1) or (2). This guess is used to approximate the functions, $\underline{f}$, as linear equations. The linear equations are then solved for values of $\underline{z}$. The resulting values of $\underline{z}$ are then used as a new guess, $\underline{z}_0$, and the whole process is repeated. This iteration continues until one of several termination or convergence criteria is satisfied.

The linear approximations of the equations being solved are generated by expanding each function as a Taylor series and then dropping the second- and higher-order terms, resulting in the linear form shown in equation (3).

$$f_1(z_1,z_2,\cdots,z_n) \simeq f_1(\underline{z}_0) + \left.\frac{\partial f_1}{\partial z_1}\right|_{\underline{z}_0} \left(z_1 - (z_1)_0\right) + \left.\frac{\partial f_1}{\partial z_2}\right|_{\underline{z}_0} \left(z_2 - (z_2)_0\right) + \cdots + \left.\frac{\partial f_1}{\partial z_n}\right|_{\underline{z}_0} \left(z_n - (z_n)_0\right) \tag{3}$$

The Taylor series expansions include the partial derivative of the functions, $\underline{f}$, with respect to each of the unknown variables, $\underline{z}$, as can be seen in equation (3). If these had to be supplied as analytic functions in order to do the Taylor series expansions, the process would be very tedious, and in addition it could not be generalized easily. Each time a new set of equations needed to be solved, it would be necessary to take all the partial derivatives for that particular set of equations. For this reason, the derivatives appearing in equation (3) are usually approximated numerically using finite differences, as indicated in equation (4) where δz_j represents a small change in z_j. Using approximate partial derivatives is a reasonable thing to do since equation (3) is just an approximation to the actual function, f_1. In this way, analytic partial derivatives are not needed in order to approximate the equations (1) as linear equations (3).

$$\left.\frac{\partial f_i}{\partial z_j}\right|_{\underline{z}_0} \simeq \frac{f_i\left((z_1)_0,\cdots,\left((z_j)_0+\delta z_j\right),\cdots,(z_n)_0\right) - f_i\left((z_1)_0,\cdots,(z_j)_0,\cdots,(z_n)_0\right)}{\delta z_j} \tag{4}$$

Thus, equations (1) are solved by guessing a solution, $\underline{z}_0$, using that guess and equation (4) to generate the linear approximations (3), solving the approximate linear equations for $\underline{z}$, and then starting another iteration of this process using the resulting $\underline{z}$ as the new guess, $\underline{z}_0$. These iterations may be repeated many times, but there need to be criteria for when it should stop. Usually there are several criteria and as soon as any one of them is satisfied, the iterations are stopped. Common termination/ convergence criteria include the following:

- A specified number of iterations have occurred
- The functions in equation (1) have been evaluated a specified number of times
- The solutions in successive iterations are getting worse instead of better
- The values of the functions are changing by less than some specified amount
- The values of the unknowns are changing by less than some specified amount

The "specified" quantities in these criteria can usually be set when you start the process, but if you don't set them, default values are typically used.

Note that in some of the situations above, the process is stopped before it finds a solution, either because it is taking too long or because it is having difficulty finding a solution. In these cases, *the results being returned do not represent a solution*, and you need to restart the process using a different guess or with different settings for the termination/convergence criteria. Put differently, the numerical approach described here may terminate before it has converged to a solution. Good software will warn you when this happens, but it is not a bad idea to check the result to see whether it does, indeed, satisfy the original equations (1).

If the method fails to converge, but the returned results are reasonable (i. e. not approaching infinity, etc.) you can try restarting the solution using the results as a guess. This may or may not work. Sometimes the method fails to converge because the guess you provided is too far from the true solution. In these cases finding a solution will require a better initial guess. There is one additional thing you can do that may help in obtaining a converged solution. If any of the functions, f, contain fractions where the unknowns, z, appear in the denominator, multiply through the entire equation and eliminate the denominator, if this is possible. If it is possible to eliminate the denominator in this way, then the resulting function won't "blow up" if the denominator gets close to zero, and this can sometimes improve convergence of the numerical methods.

The approach that has been outlined here has one disadvantage: it will only find one solution to the equations. If the equations being solved are non-linear, then it is quite possible that the number of solutions is greater than one, just as a quadratic equation has two solutions. As a consequence, you should always check the solution that results from numerically solving a set of non-linear equations. If it does not make physical sense in the context of the problem you are solving, you should attempt to find a different solution. Similarly, if the physical situation is one where multiple solutions are possible, then again, you should attempt to find a different solution. The best way to find additional solutions is to repeat the solution process using a different initial guess.

MATLAB Template Files

MATLAB contains a built in function named `fsolve` that can be used to solve sets of non-differential equations. (It will solve linear and non-linear equations.) It takes very few lines of code to use the built-in function `fsolve` to solve a set of algebraic equations. Nonetheless, to spare you the tedium of doing so each time you need to solve a set of algebraic equations, and to allow you to focus more on kinetics and reaction engineering and less on computer programming, a MATLAB template file is provided for you to use. The MATLAB template file is named SolvNonDif.m, and it requires four modifications each

time it is used. A set of step-by-step instructions for using SolvNumDif.m are included with this supplemental unit.

The required modifications involve defining all constants that are needed to solve the problem, entering the code to evaluate the functions, $\underline{f}$, given values for the variables, $\underline{z}$, entering a guess for the values of the unknowns, $\underline{z}_0$, and entering any code needed to calculate additional quantities using the results from solving the equations.

The template file is set up to be convenient in situations where solving the set of algebraic equations is the heart of the problem you are addressing. As such, the fourth required modification listed above involves simple processing of the results to calculate other quantities. For example, you might solve CSTR design equations to find the outlet molar flow rate of a reactant. Then, knowing that outlet molar flow rate, you might want to calculate the fractional conversion of the reactant. The template file is set up so that you do this within the template file. Such a set up is convenient, for example, when solving homework problems; the entire problem solution can be contained within a single modified version of the template file.

The template file does return the resulting values it finds for $\underline{z}$ in the form of a column vector. Thus, an alternative way to use the template file is to skip the fourth required modification within the template file. Then, when the file has completed its execution and returns the solution, you can calculate any additional quantities at the MATLAB command prompt. The choice between adding the code to calculate additional quantities within the template file or calculating those quantities at the MATLAB command prompt is yours to make. In most cases, the examples provided in “A First Course on Kinetics and Reaction Engineering” calculate all quantities within the modified template file.