Simulating and Fitting Data for Chemical Reactions

by
Namir Clement Shammas

Contents

1/Introduction

This study looks at simulating data of chemical reactions and performing nonlinear curve fitting on that data. Simulating the data for chemical reactions is the first step in the studied calculations. It replaces having a real (and very expensive) chemistry laboratory where one studies chemical reactions by measuring the concentration of chemicals, at different times, during the progress of a chemical reaction. The second step is to perform optimization-based curve fitting for nonlinear models that fit the data collected for the chemical reaction. This study looks at general types of chemical reactions.

2/Simulating the Data for Chemical Reactions

The first step is to simulate the measured concentrations of chemical reactants (and sometimes products) involved in chemical reactions. The better scheme of this simulation assumes that the concentrations of chemical are measured at time intervals that are **not necessarily equal**. This unequal time intervals mimics real life situations where a lab chemist measures concentrations of chemicals at her or his own reasonable discretion.

There are two general schemes used to generate simulated concentrations of chemicals in chemical reactions.

2.1/Using Mathematical Models

The first, and better choice, is to use integrated models that offer a direct way to calculate the concentration of a chemical given:

1. The initial concentration of one or more chemicals involved in the chemical reactions.
2. The values of one or more reaction rate constants.

The method starts with an array of time values for which the concentrations are measured. This array mimics a timetable used by a lab chemist to measure concentrations. Using the integrated equation(s) of the chemical reaction rate equation(s), the method calculates the exact values for the concentrations for the given timetable. In addition, the method generates another set of concentration values that mimics actual values that include some errors in the measurements. Since we are simulating such errors, we use a maximum percent of random errors that occur in the measurements. These errors are calculated using uniformly distributed random errors. You can alter the source code and use normally distributed random numbers. Thus, the calculations generate two arrays of concentration values—one theoretical and one observed (that includes random errors).

2.2/Using Numerical Methods for Integrating Ordinary Differential Equations

The available models for integrating differential equations for chemical reactions represent a small fraction of all possible reactions and also the conditions involving the presence or absence of chemicals involved in chemical reactions. Often, the integrated models make simplifications (such as the initial absence of one or more chemicals involved in the chemical reactions) in order to perform the analytical integration of the chemical reaction's differential equation.

The general approach for calculating the concentration of chemicals uses a good numerical method that solves one or more ordinary differential equations (one per chemical reaction). You have more flexibility is choosing the initial conditions for the differential equations of the chemical reactions. You also have a choice of good numerical method for solving one or more ODE. In this study I use the Runge-Kutta-Fehlberg method.

An important rule to observe in using numerical methods for solving ODEs in the value of the increment h used by the numerical method. You need to make sure that h is at least 1/10 (or better yet 1/100) the smallest least significant digit. Let me explain with an example. If you take measurements at time units (minutes, seconds, milliseconds, and so on) of 1.0, 2.1, 3.4, 5.6, and 7.75 then the smallest digit is the 0.05 in the value 7.75. The increment in time is therefore 0.01 and h is 0.001 or, better yet, 0.0001. If the time measurements were at 1.0, 2.2, 3.4, 6, and 7.7, the smallest digit is 0.2 in value 2.2. Thus, the normalized time value is 0.1 and h is 0.01 or 0.001. Also consider time readings of 20, 40, 50, 70 and 90, then the smallest normalized time value is 10 in any two-digit value listed. Thus, the value of h is 1 or 0.1. If we include the value 15 in the last list, then the smallest digit is 1 found in the value 15, and h is 0.1 or 0.001. The scheme for calculating the smallest normalized increment and h allow you to find values for the concentrations that match the values in the array of timed readings. The Appendix, located near the end of this document, shows the listing of the MATLAB function *findMinDigits* which returns the smallest digit found in an array of floating-point numbers.

3/Forming the Optimized Function

Once you obtain the simulated concentration values, you start the optimization stage. This stage needs arrays that define the lower and upper bounds of the variables (concentrations and reaction rate constants) involved in the optimization process. This process of course needs a function to optimize. The general form for the optimized function is:

```
Function sumErrorsSquared = rf5k(x)
…
```

```
Global variables defined here
A0 = x(1)
…
Kr1 = x(n)
Kr2 = x(n+1)
…
End

Function rate=fx(list of parameters)
…
rate = expression for the chemical reaction rate
end
```

The function *rkf5* is a special version of the Runge-Kutta-Fehlberg method (or any other good numerical ODE solver you choose). It takes one argument—*x* the array of optimized variables. The function returns the sum of errors squared. The optimization algorithm has the task of minimizing the sum of errors squared by fine-tuning the values in array *x*. The function *rkf5* uses global variables to pass the arrays for the measured time values and concentrations. The values of the elements in array *x* are mapped onto local variables (for the concentrations and reaction rate kinetics) to make the equations a bit clearer to follow in the code. The rest of the calculations in function *rkf5* use a loop to iterate over the range of measured time readings. The function uses a local helper function, *fx*, to calculate the reaction rate(s). It is important to point out that function *fx* passes various values for the concentrations and reaction rate constants, but never time values! This omission simplifies the numerical integration steps in function *rkf5*. Normally this function calculates intermediate values using coefficients *k1* through *k6* using x and y values (x is time and y is a concentration or a concentration fraction). Since function *fx* does not require the values of time, neither do the coefficients *k1* through *k6*. Thus, we implement the equations to calculate these coefficients using the concentration values and any other values, such as initial reactant concentrations and reaction rate constants. You can say that the procedure uses customized numerical ODE solvers to calculate coefficients *k1* through *k6*. This customization varies with the particulars of each chemical reaction!

As for the optimization method, I recommend an evolutionary-based optimization algorithm. There is a vast number of such methods. I have chosen to work with the very popular particle swarm optimization (PSO) which MATLAB makes available in the Optimization Toolbox. You tell this optimization method which function you want to optimize, the arrays that define the lower and upper limits for the optimized variables, the size of the particle swarm population, and the maximum number of iterations. Higher values of the last two parameters generally yield better results, albeit at the cost of more CPU effort and longer computational time.

I would like to remind you that the evolutionary optimization method I am using (and most of them do) relies on generating many random numbers. These random numbers will cause the optimization method to generate different results each time you execute the calculations. Thus, when you run the various scripts for the different chemical reactions, you will see slightly different output.

☛

Depending on the number of optimization variables, the chemical reaction differential equation(s), and the errors in the simulated observed variables, you may want to run the calculations several times and then average the values of the optimization variables. These averages should diverge to the true values.

4/Getting on with the Program

Now that the preliminaries are behind us, you may ask, “What’s next?” I will start with a very simple example to show proof of concept. The simple example is a first order chemical reaction. Chemists usually use calculators or PC math software, like Excel and MATLAB, to perform straightforward linearized regression on the set of observed concentration and time readings. These calculations are deterministic and do not involve the explicit use of an optimization function that utilizes random numbers. However, to show you how the method I am studying works, we will start with such a simple example.

☛

Keep in mind that most, if not all, nonlinear curve fitting (for chemical reactions) that uses optimization algorithms encounters many local minima. These local minima cause the values of the optimized variables to fluctuate above and below their theoretical values.

5/First Order Chemical Reaction

Consider the following first order chemical reaction:

$$A \rightarrow B$$

With the chemical reaction rate of:

$$dA/dt = -k_1 A \qquad (5.1)$$

Where t is time, A is the concentration of reactant A, and k_1 is the reaction rate constant. The analytical solution of the first rate chemical reaction is:

$$\ln(A) = \ln(A_0) - k_1 t \qquad (5.2)$$

Or,

$$A = A_0 * \exp(-k_1 t) \qquad (5.3)$$

Equation 5.2 says that a semi-log plot of A vs t gives an intercept of $\ln(A_0)$ and a slope of minus k_1. You can use the intercept to compare the value of the initial concentration with measure value. Measuring the initial value of reactant A is not required but helps to get an idea about errors. Of course, the regression analysis of equation 5.2 also yields the coefficient of determination to examine how much error was involved in the measurements of the concentrations.

☛

The MATLAB files mentioned in this section are found in the folder \Chemical Reaction Modeling with Optimized ODEs\First Order

The MATLAB function *react* calculates the theoretical and error-deviating values for reactant A, given the values for A_0, k_1, the array of time, and the maximum percent error.

```
function [Aobs,Ath,tArr] = react(A0,k,tArr,percErr)
%REACT calculates the array of concentrations for
% A = B, with rA = -k1*A
  n = length(tArr);
  A = zeros(1,n);
  for i=1:n
    Ath(i) = A0*exp(-k*tArr(i));
    Aobs(i) = Ath(i) * (1+percErr/100*(2*rand-1));
  end
end
```

The function react returns the theoretical values of A, the simulated *observed* values of A (given the maximum percent error), and the time array. The function uses uniformly distributed random numbers to simulate random errors. Notice that the values for array *Ath* are based on equation 5.3. The statement that calculates the values in array *Aobs* uses the MATLAB function *rand* to generate uniformly distributed random numbers:

```
Aobs(i) = Ath(i) * (1+percErr/100*(2*rand-1));
```

You can alter the above code to replace function *rand* with *randn* to use normally distributed random numbers:

```
Aobs(i) = randn(Ath(i), Ath(i) * percErr/100);
```

The above statement generates values for array *Aobs* that have a mean of *Ath(i)* and a standard deviation of *Ath(i) * percErr/100*.

Working in parallel to function *react* is function *rkf5* which yields the calculated values of the concentration of reactant A and the sum of errors squared. These errors are calculated by comparing the observed and calculated values of A.

```
function sumSqrErr = rkf5(x)
% rkf5 implements Runge-Kutta-Fehlberg
% rkf5 implements Runge-Kutta-Fehlberg
% A = B, with rA = -k1*A
  global tData
  global yData
  global incr;
  A0 = x(1);
  k = x(2);
  iData = 1;
  nData = length(tData);
  h = incr/10;
  nSteps = fix((tData(nData)-tData(1))/h + 0.5);
  sumSqrErr = 0;
  t = tData(1);
  y = A0;
  for iter=1:nSteps
    if t+h > tData(iData)
      sumSqrErr = sumSqrErr + (y - yData(iData))^2;
      iData = iData + 1;
    end
    k1 = h*fx(y,k);
    k2 = h*fx(y+k1/4,k);
    k3 = h*fx(y+(3*k1+9*k2)/32,k);
    k4 = h*fx(y+(1932*k1-7200*k2+7296*k3)/2197,k);
    k5 = h*fx(y+439/216*k1-8*k2+3680/513*k3-845/4104*k4,k);
    k6 = h*fx(y - 8/27*k1 + 2*k2 -3544/2565*k3 +1859/4104*k4 -
11/40*k5,k);
    y = y + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5
+ 2/55*k6;
    t = t + h;
  end
end

function y = fx(A,k)
  y = -k*A;
end
```

Function *rkf5* has one parameter—the array *x* which contains the optimization variables. The function also accesses the global values for the array of time, *tData*, the array of observed concentrations *yData*, and the minimum time increment, *incr*. The function copies the values of elements *x(1)* and *x(2)* into local variables *A0* (the initial concentration of reactant A) and *k* (the reaction rate constant). The function calculates the integration step increment *h* as *incr/10*. Dividing *incr* by values one or more orders higher than 10 will significantly increase the calculations time. The function uses a for loop to obtain the calculated values for the concentration of A. The if statement that appears as the first loop statement detects when to compare the value of the calculate concentration of A with its observed counterpart. This comparison calculates the sum of errors squared. The subsequent statements that calculate coefficients *k1* through *k6* call local function *fx* and passes two arguments—one for the value of the concertation of A (stored in variable *y*) and the reaction rate coefficient k. The function *fx* calculates the reaction rate using equation 5.1. Notice that the function *fx* has no parameter for time since it is not required in the calculations of the reaction rate.

The script file *go.m* is the main function that initializes the theorical and observed data and then performs nonlinear curve fitting by using optimization.

```
% Reaction A = B
% using rA = -k*A
clc
close
clear all
global tData
global yData
global incr
fprintf('A ==> B\n');
fprintf('A0 > 0\n');
fprintf('Reaction is first order\n');
fprintf('Wait please ...\n');
A0 = 100;
k = 0.05;
tArr = [0 1.2 2.2 3.3 4.5 5.3 6.1 6.9 7.2 7.9 8.1 8.8 9.3 10];
incr = findMinDigit(tArr);
[Aobs,Ath,tArr] = react(A0,k,tArr,10);
tData = tArr;
yData = Aobs;

fcn = @rkf5;
nvars = 2;
lb = [A0 k]/1.05;
ub = 1.05^2*lb;
[bestX, bestFx] = particleswarm(fcn,nvars,lb,ub);
fprintf('\nBest A0=%f, k=%f\n', bestX(1), bestX(2));
fprintf('Best fx = %e\n', bestFx);
% [bestX, bestFx] = scout([50 .01], [200 .1], [10 .01], [.1
.001], 10000, 100, 50, false, true)
n = length(tArr);
Acalc = zeros(n,1);
for i=1:n
  Acalc(i) = bestX(1)*exp(-bestX(2)*tArr(i));
end

plot(tArr,Ath,tArr,Aobs,'r',tArr,Acalc,'g');
grid;

for i=1:n
  err1 = (Acalc(i) - Ath(i))/Ath(i)*100;
  err2 = (Acalc(i) - Aobs(i))/Aobs(i)*100;
  fprintf('%f %f %f %f %f\n', Ath(i), Aobs(i), Acalc(i), err1,
err2);
end
fprintf('\n\n');
fprintf('R^2 = %f for comparing theoretical and calculated
values\n', rsqr(Ath,Acalc));
fprintf('R^2 = %f for comparing observed and calculated
values\n', rsqr(Aobs,Acalc));
```

The *go* script performs the following tasks:

- Declares the global arrays *tData* and *yData* to store time and concentration values, respectively. The script also declares the global variable *incr*.

- Assigns the values of the initial concentration of reactant A and the reaction rate constant to variables *A0* and *k*, respectively.
- Assigns the time values to array *tArr*.
- Calculates the minimum digit in the time array by calling function *findMinDigit* and passing it the argument *tArr*. The script stores the result of this function call in the global variable *incr*.
- Calls function *react* to obtain the values of the theoretical and simulated observed values of the concentration of reactant A. The function call stores these values in arrays *Ath* and *Aobs*, respectively. The arguments for the function call are *A0*, *k*, *tsArr*, and 10 (the maximum percentage of error used in calculating the observed concentration values).
- Copies the values of arrays *Aobs* and *tArr* into the global arrays *tData* and *yData*, respectively.
- Stores the handle of function *rkf*5 in the variable *fcn*.
- Assigns the number of optimization variables, 2, to variable *nvars*.
- Assigns values to the arrays *lb* and *ub* that store the lower and upper bounds of the optimization variables, respectively.
- Calls the MATLAB function *particleswarm* to perform particle swarm optimization. The arguments for this function call are *fcn*, *nvars*, *lb*, and *ub*. The function *particleswarm* uses the default particle population and maximum number of iterations. The function call stores the values of the best optimization variables in array *bestX*. The call also stores the value of the best optimized function in variable *bestFx*.
- Displays the values of the optimization variables and bets optimized function value stored in array *bestX* and variable *bestFx*, respectively.
- Calculates the estimated concentration values (stored in array *Acalc*) using a for loop. The loop uses the values of array *bestX* to calculate the values of array *Acalc*.
- Plots the values in arrays *Ath*, *Aobs*, and *Acalc*. This plot allows you to visually compare between the theoretical, observed, and calculated concentration values of reactant A.
- Calculates and displays the array of errors between the theoretical and calculated concentration values, and also between the observed and calculated concentration values.

- Calculates the coefficient determination between the theoretical and calculated concentration values, and also between the observed and calculated concentration values.

Here is a sample session with the script in file *go.m*:

```
A ==> B
A0 > 0
Reaction is first order
Wait please ...

Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=102.990562, k=0.050538
Best fx = 3.561682e+02
100.000000 106.294474 102.990562 2.990562 -3.108263
94.176453 101.819662 96.930226 2.924056 -4.802055
89.583414 82.900255 92.153264 2.868668 11.161618
84.789370 91.799346 87.170066 2.807775 -5.042825
79.851622 81.965442 82.040665 2.741388 0.091774
76.720595 70.545207 78.789867 2.697154 11.687060
73.712337 70.446855 75.667880 2.652938 7.411297
70.822035 71.486084 72.669600 2.608742 1.655588
69.767633 76.151466 71.576131 2.592173 -6.008204
67.368004 73.631726 69.088262 2.553523 -6.170526
66.697681 62.130398 68.393459 2.542483 10.080509
64.403642 70.465220 66.016214 2.503852 -6.313761
62.813511 68.556763 64.368941 2.476267 -6.108547
60.653066 60.475664 62.131582 2.437661 2.738157


R^2 = 0.968644 for comparing theoretical and calculated values
R^2 = 0.855072 for comparing observed and calculated values
```

The calculated values for A0 and k are 102.990562 and 0.050538, respectively. They are close to the assigned values of A0 = 100 and k = 0.05, respectively. Figure 5.1 shows the plot generated by the script. The blue line shows the theoretical values of the concentration. The green line shows the calculated values of the concentration. Both of these curves are smooth. The shift between these two curves is due to the influence of the errors in the observed concentration values. These two curves are contrasted by the zig zagging red line that represents the observed values of the concentration.

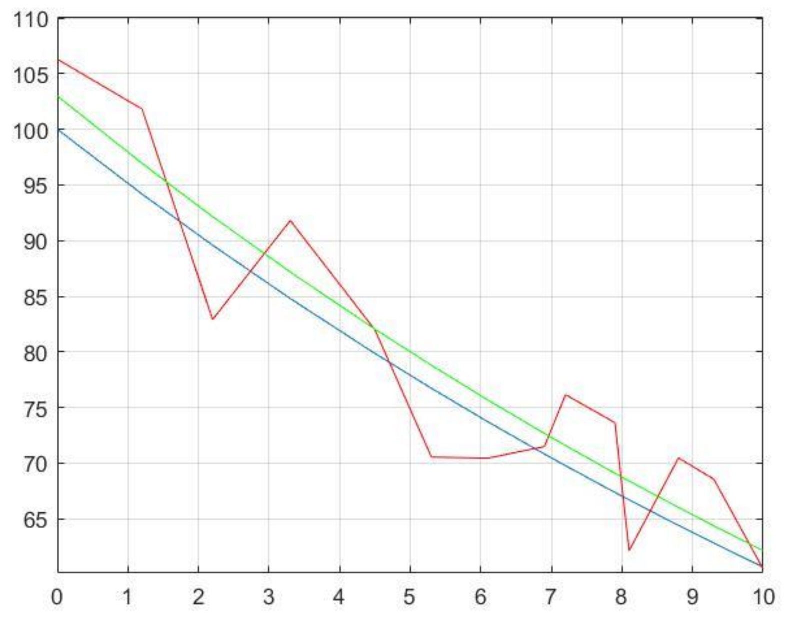

Figure 5.1. The plot generated for the simple first order reaction A→B.

The results generated by the *go.m* script for the simple first order reaction shows that the method of simulating concentration data and performing nonlinear fitting using the optimization of ODE solvers works well.

Armed with the above positive conclusion, we venture into more advanced chemical reactions.

6/First Order Chained Chemical Reaction

Consider the following first order chained chemical reaction:

$$A \rightarrow B \rightarrow C$$

With the chemical reaction rates of:

$$dA/dt = -k_1A \tag{6.1}$$
$$dB/dt = k_1A - k_2B \tag{6.2}$$
$$dC/dt = k_2B \tag{6.3}$$

Where t is time, A is the concentration of reactant A, and k_1 is the first reaction rate constant, B is the concentration of intermediate reactant B, and k_2 is the second reaction rate constant. C is the concentration of the final product. The analytical solution of the chained chemical reaction (assuming that the initial concentrations for reactants B and C are zero) is:

$$A = A_0 * \exp(-k_1 * t) \qquad (6.4)$$
$$B = A_0 * k_1/(k_2 - k_1)*(\exp(-k_1 * t) - \exp(-k_2 * t)) \qquad (6.5)$$
$$C = A0 - A - B \qquad (6.6)$$

Equation 6.4 can be solved easily using a linearized regression model that applies logarithmic transformations to both left and right sides of the equation. Equation 6.5 is a nonlinear model and is typically solved using optimized regression with the optimized variables A_0, k_1, and k_2.

☛	FYI
	If $B_0 > 0$, equation 6.5 becomes: $B = A_0 * k_1/(k_2 - k_1)*(\exp(-k_1 * t) - \exp(-k_2 * t)) + B_0*\exp(-k_2 * t)$ The above equation has started to appear in recently published books that deal with reaction kinetics. Equation 6.5 has previously been the dominantly popular analytical equation for the chain reaction. And equation 6.6 becomes: $C = A_0 + B_0 - A - B$ You can find MATLAB files that handle the above case for the chained chemical reaction where A_0 and B_0 are both positive in the folder \Chemical Reaction Modeling with Optimized ODEs\Chain Reaction First Order 2 Reactions - Ver 2.

☛

The MATLAB files mentioned in this section are found in the folder \Chemical Reaction Modeling with Optimized ODEs\Chain Reaction First Order 2 Reactions - Ver 1

Armed with the above equations, I present a new version of function *react* that returns the matrices *ConcTh* and *ConcObs*. The matrix *ConcTh* has the theoretical values for chemicals A, B, and C, stored in columns 1, 2, and 3, respectively. The matrix *ConcObs* has the simulated observed values for chemicals A, B, and C, stored in columns 1, 2, and 3, respectively. The parameters for the function react are *A0*, *k1*, *k2*, *tArr*, and *percErr* that pass the initial concentration of chemical A, the first reaction rate constant, the second reaction rate constant, the array of time values, and the maximum percent error, respectively. Here is the listing for function *react*:

```
function [ConcObs,ConcTh,tArr] = react(A0,k1,k2,tArr,percErr)
%REACT calculates teh array of concenrations for
% A --> B --> C  where B0 = C0 = 0
  n = length(tArr);
  ConcTh = zeros(n,3);
  ConcObs = zeros(n,3);
  rB = k1/(k2-k1);
  for i=1:n
    t = tArr(i);
    ConcTh(i,1) = A0*exp(-k1*t);
    ConcTh(i,2) = A0*rB*(exp(-k1*t) - exp(-k2*t));
    ConcTh(i,3) = A0 - ConcTh(i,1) - ConcTh(i,2);
    for j=1:3
      ConcObs(i,j) = ConcTh(i,j) * (1+percErr/100*(2*rand-1));
    end
  end
end
```

The above source code shows how the theoretical values for the concentrations of chemicals A, B, and C are calculated using equations 6.4, 6.5, and 6.6, respectively.

Here is the listing for the version of function *rkf5* that represents the optimized function that is based on a custom version of the Runge-Kutta-Fehlberg method:

```
function sumSqrErr = rkf5(x)
% rkf5 implements Runge-Kutta-Fehlberg
% A --> B --> C  where B0 = C0 = 0
  global tData
  global cData
  global incr;
  A0 = x(1);
  kr1 = x(2);
  kr2 = x(3);
  iData = 1;
  nData = length(tData);
  h = incr/10;
  nSteps = fix((tData(nData)-tData(1))/h + 0.5);
  sumSqrErr = 0;
  t = tData(1);
  y = [A0 0];
  for iter=1:nSteps
    if t+h > tData(iData)
      for j=1:2
        sumSqrErr = sumSqrErr + (y(j) - cData(iData,j))^2;
      end
      iData = iData + 1;
    end
    k1 = h*fx(y,kr1,kr2);
    k2 = h*fx(y+k1/4,kr1,kr2);
    k3 = h*fx(y+(3*k1+9*k2)/32,kr1,kr2);
    k4 = h*fx(y+(1932*k1-7200*k2+7296*k3)/2197,kr1,kr2);
    k5 = h*fx(y+439/216*k1-8*k2+3680/513*k3-
845/4104*k4,kr1,kr2);
    k6 = h*fx(y - 8/27*k1 + 2*k2 -3544/2565*k3 +1859/4104*k4 -
11/40*k5,kr1,kr2);
    y = y + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5
+ 2/55*k6;
    t = t + h;
  end
end

function y = fx(x,k1,k2)
  y(1) = -k1*x(1);
  y(2) = k1*x(1)-k2*x(2);
end
```

The source code of the above version of *rkf5* is similar to the first version of rkf5. The new version is a bit more elaborate since it is handling a more elaborate set of chained chemical reactions. The parameter of function *rkf5* is array *x* that stores values for *A0*, *k1*, and *k2*. The function copies the elements of array *x* into the local variables *A0*, *kr1*, and *kr2*, respectively. I inserted the letter r after the letter k to make the variables different from the other local variables *k1*, and *k2* that are used in the numerical solution of the ODEs. The function accesses the observed concentration values using the global matrix *cData*. The function also accesses the time values using the global array *tData*. Notice that the local function *fx* returns an array of reaction rates calculated based on equations 6.1 and 6.2. Thus, the new version of function *rkf5* solves a system of two ODEs.

The second version of script *go.m* deals with the case of the chained reaction. Here is the source code for the script in that MATLAB file:

```
% Chain rection:
%
```

```
%   A --> B -- > C
%
%   where [A0] > 0, [B0] = 0, and [C0] = 0
%
clc
close all
clear
global tData
global cData % a matrix
global incr
fprintf('Reaction: A --> B -- > C\n');
fprintf('where [A0] > 0, [B0] = 0, and [C0] = 0\n');
fprintf('All reactions are first order\n');
fprintf('Please wait ...\n');
A0 = 100;
k1 = 0.5;
k2 = 0.1;
tArr = [0 1.2 2.2 3.3 4.5 5.3 6.1 6.9 7.2 7.9 8.1 8.8 9.3 10];
incr = findMinDigit(tArr);
[ConcObs,ConcTh,tArr] = react(A0,k1,k2,tArr,10);
tData = tArr;
cData = ConcObs;

fcn = @rkf5;
nvars = 3;
lb = [A0 k1 k2]/1.1;
ub = 1.1^2*lb;
options   =   optimoptions('particleswarm',   'SwarmSize',   500,
'MaxIterations', 5000);
[bestX, bestFx] = particleswarm(fcn,nvars,lb,ub,options);
fprintf('\nBest  A0=%f,  k1=%f  k2=%f\n',  bestX(1),  bestX(2),
bestX(3));
fprintf('Best fx = %e\n', bestFx);
n = length(tArr);
ConcCalc = zeros(n,3);
A00 = bestX(1);
kk1 = bestX(2);
kk2 = bestX(3);
rB = kk1/(kk2-kk1);
for i=1:n
  t = tArr(i);
  ConcCalc(i,1) = A00*exp(-kk1*t);
  ConcCalc(i,2) = A00*rB*(exp(-kk1*t) - exp(-kk2*t));
  ConcCalc(i,3) = A00 - ConcCalc(i,1) - ConcCalc(i,2);
```

```
end

for j=1:3
  figure(j)

plot(tArr,ConcTh(:,j),tArr,ConcObs(:,j),'r',tArr,ConcCalc(:,j),'
g');
  if j == 1
    title('Concentration of A');
  elseif j == 2
    title('Concentration of B');
  else
    title('Concentration of C');
  end
  xlabel('Time');
  ylabel('Concentration');
  grid;
end

figure(4)
plot(tArr,ConcTh(:,1),tArr,ConcTh(:,2),'r',tArr,ConcTh(:,3),'g')
;
title('Theoretical concentrations of A, B, and C');
xlabel('Time');
ylabel('Concentration');
grid;

figure(5)
plot(tArr,ConcObs(:,1),tArr,ConcObs(:,2),'r',tArr,ConcObs(:,3),'
g');
title('Observed concentrations of A, B, and C');
xlabel('Time');
ylabel('Concentration');
grid;

figure(6)
plot(tArr,ConcCalc(:,1),tArr,ConcCalc(:,2),'r',tArr,ConcCalc(:,3
),'g');
title('Calculated concentrations of A, B, and C');
xlabel('Time');
ylabel('Concentration');
grid;

for j =1:3
```

```
  if j == 1
    fprintf('For concentration of A\n');
  elseif j == 2
    fprintf('For concentration of B\n');
  else
    fprintf('For concentration of C\n');
  end
  for i=1:n
    if ConcTh(i,j)~=0
      err1 = (ConcCalc(i,j) - ConcTh(i,j))/ConcTh(i,j)*100;
      err2 = (ConcCalc(i,j) - ConcObs(i,j))/ConcObs(i,j)*100;
      fprintf('%f  %f  %f  %f  %f\n',  ConcTh(i,j),  ConcObs(i,j), 
ConcCalc(i,j), err1, err2);
    end
  end
  fprintf('\n');
  fprintf('R^2  =  %f  for  comparing  theoretical  and  calculated 
values\n', rsqr(ConcTh(:,j),ConcCalc(:,j)));
  fprintf('R^2  =  %f  for  comparing  observed  and  calculated 
values\n\n', rsqr(ConcObs(:,j),ConcCalc(:,j)));
end
```

The *go* script performs the following tasks:

- Declares the global array *tData* and global matrix *yData* to store time and concentration values, respectively. The script also declares the global variable *incr*.
- Assigns the values of the initial concentration of reactant A and the reaction rate constants to variables *A0* and *k1*, and *k2*, respectively.
- Assigns the time values to array *tArr*.
- Calculates the minimum digit in the time array by calling function *findMinDigit* and passing it the argument *tArr*. The script stores the result of this function call in the global variable *incr*.
- Calls function *react* to obtain the values of the theoretical and simulated observed values of the concentration of chemicals A, B, and C. The function call stores these values in matrices *ConcTh* and *ConcObs*, respectively. The arguments for the function call are *A0*, *k1*, *k2*, *tsArr*, and 0 (the percentage of error used in calculating the observed concentration values).
- Copies the values of matrix *Concobs* and array *tArr* into the global variables *cData* and *tData* respectively.
- Stores the handle of function *rkf5* in variable *fcn*.

- Assigns the number of optimization variables, 3, to variable *nvars*.
- Assigns values to the arrays *lb* and *ub* that store the lower and upper bounds of the optimization variables, respectively. I used special expressions that estimate the ranges to be about plus and minus 5% of the actual values of the optimization variables.
- Sets the options for function *particleswarm* to use 500 particles and a maximum number of iterations of 5000. The script stores these options in the variable *options*.
- Calls the MATLAB function *particleswarm* to perform particle swarm optimization. The arguments for this function call are *fcn*, *nvars*, *lb*, *ub*, *options*. The function call stores the best optimization variables in array *bestX*. The call also stores the best optimized function in variable *bestFx*.
- Displays the values of the optimization variables and bets optimized function value stored in array *bestX* and variable *bestFx*.
- Calculates the estimated concentration values (stored in matrix *ConcCalc*) using a for loop. The loop uses the values of array *bestX* to calculate the values of matrix *ConcCalc*.
- Performs several plots for the various chemicals. Some of these plots compares the concentrations of the chemicals A, B, and C. Other plots are for the individual chemicals.
- Calculates and displays the array of errors between the theoretical and calculated concentration values, and also between the observed and calculated concentration values.
- Calculates the coefficient of determination between the theoretical and calculated concentration values, and also between the observed and calculated concentration values.

Here is a sample session with the script *go*:

```
Reaction: A --> B -- > C
where [A0] > 0, [B0] = 0, and [C0] = 0
All reactions are first order
Please wait ...

Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=99.968911, k1=0.501053 k2=0.099937
Best fx = 9.395978e-02
```

```
For concentration of A
100.000000 100.000000 99.968911 -0.031089 -0.031089
54.881164 54.881164 54.794842 -0.157288 -0.157288
33.287108 33.287108 33.199786 -0.262332 -0.262332
19.204991 19.204991 19.132443 -0.377753 -0.377753
10.539922 10.539922 10.486852 -0.503514 -0.503514
7.065121 7.065121 7.023630 -0.587267 -0.587267
4.735892 4.735892 4.704117 -0.670949 -0.670949
3.174564 3.174564 3.150610 -0.754561 -0.754561
2.732372 2.732372 2.710899 -0.785897 -0.785897
1.925470 1.925470 1.908931 -0.858976 -0.858976
1.742237 1.742237 1.726908 -0.879846 -0.879846
1.227734 1.227734 1.216035 -0.952857 -0.952857
0.956160 0.956160 0.946551 -1.004974 -1.004974
0.673795 0.673795 0.666532 -1.077892 -1.077892

R^2 = 0.999997 for comparing theoretical and calculated values
R^2 = 0.999997 for comparing observed and calculated values

For concentration of B
42.263600 42.263600 42.316545 0.125272 0.125272
58.705964 58.705964 58.757754 0.088218 0.088218
65.859228 65.859228 65.895719 0.055407 0.055407
66.528616 66.528616 66.547402 0.028238 0.028238
64.744220 64.744220 64.753642 0.014553 0.014553
61.998993 61.998993 62.001450 0.003962 0.003962
58.728804 58.728804 58.726502 -0.003919 -0.003919
57.428567 57.428567 57.424975 -0.006253 -0.006253
54.323762 54.323762 54.318042 -0.010528 -0.010528
53.429461 53.429461 53.423332 -0.011472 -0.011472
50.313196 50.313196 50.306197 -0.013912 -0.013912
48.124014 48.124014 48.116836 -0.014914 -0.014914
45.142687 45.142687 45.135725 -0.015423 -0.015423

R^2 = 0.999998 for comparing theoretical and calculated values
R^2 = 0.999998 for comparing observed and calculated values

For concentration of C
2.855236 2.855236 2.857524 0.080132 0.080132
8.006927 8.006927 8.011372 0.055506 0.055506
14.935781 14.935781 14.940748 0.033258 0.033258
22.931462 22.931462 22.934656 0.013930 0.013930
28.190659 28.190659 28.191639 0.003476 0.003476
33.265114 33.265114 33.263344 -0.005322 -0.005322
38.096632 38.096632 38.091799 -0.012687 -0.012687
39.839061 39.839061 39.833037 -0.015122 -0.015122
43.750768 43.750768 43.741938 -0.020184 -0.020184
```

```
44.828301 44.828301 44.818670 -0.021483 -0.021483
48.459070 48.459070 48.446679 -0.025570 -0.025570
50.919826 50.919826 50.905524 -0.028089 -0.028089
54.183519 54.183519 54.166654 -0.031124 -0.031124

R^2 = 1.000000 for comparing theoretical and calculated values
R^2 = 1.000000 for comparing observed and calculated values
```

The results are A0 = 99.968911, k1 = 0.501053, and k2 = 0.099937. These values are very close to the assigned values of A0 = 100, k1 = 0.5, and k2 = 0.1. This is not a big surprise since the simulated observed concentrations were calculated using 0% error and the trust region (i.e. the lower and upper ranges for the optimization variables) is minus and plus 5% of the assigned values. The output tables show that the first three columns of the various chemicals in close agreement, Again, this is expected in an ideal problem.

Figures 6.1, 6.2, and 6.3 show the theoretical, observed, and calculated concentrations for all three chemicals. These figures show the red line representing chemical B rise to a maximum in the middle of the observed reaction time.

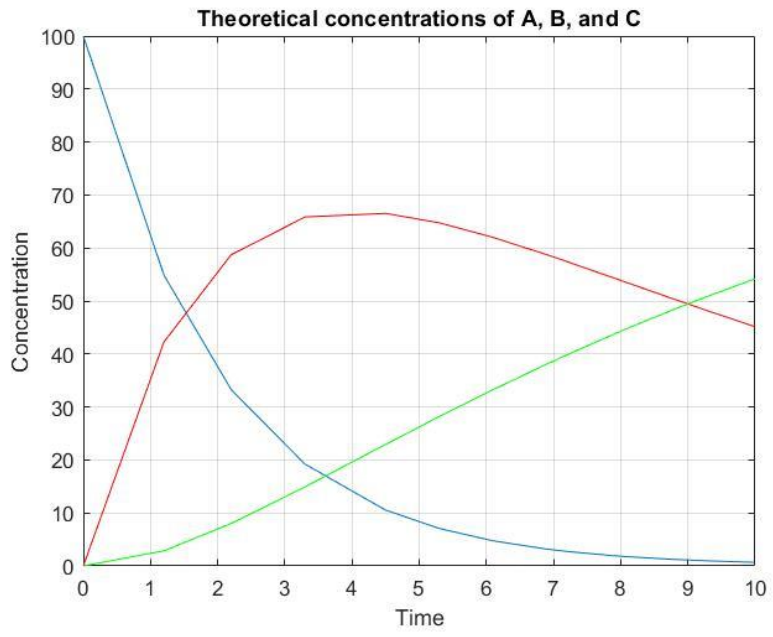

Figure 6.1. The theoretical concentrations of chemicals A, B, and C.

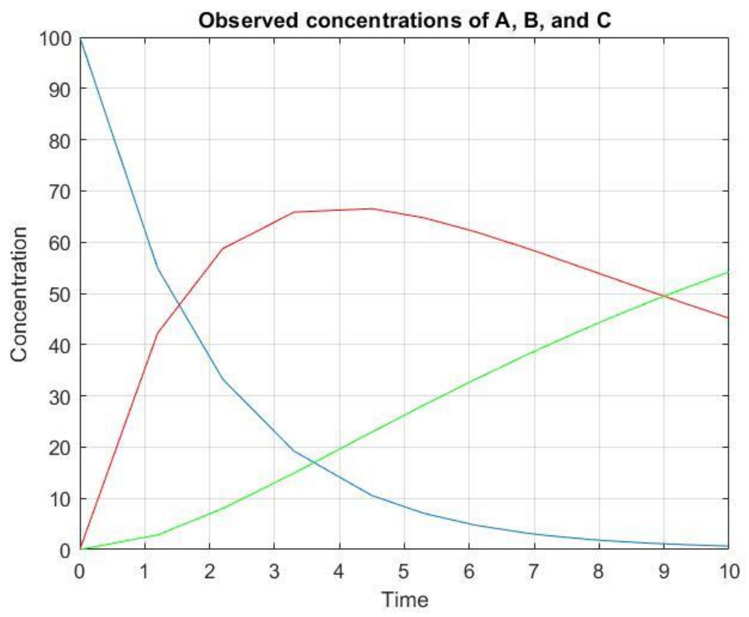

Figure 6.2. The observed concentrations of chemicals A, B, and C.

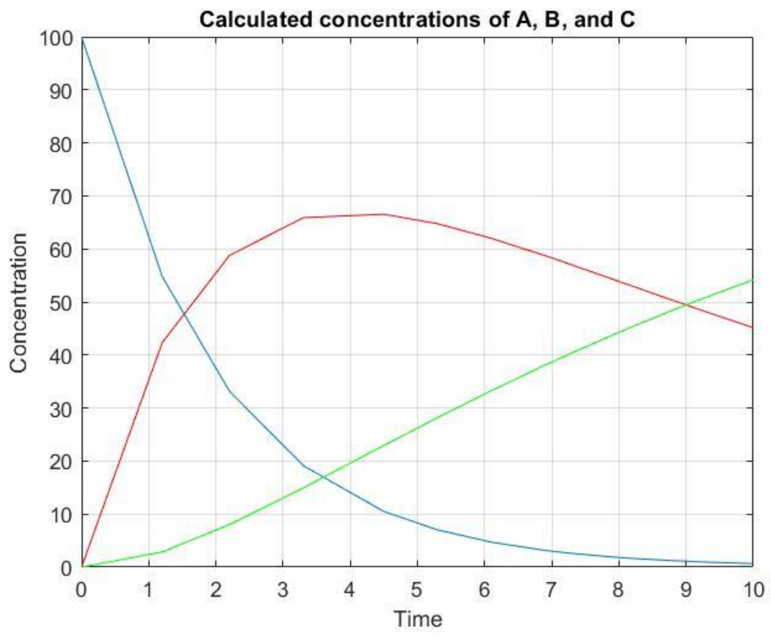

Figure 6.3. The calculated concentrations of chemicals A, B, and C.

Here is the output with a maximum of 10% random errors used in calculating the simulated observed concentration values.

```
Reaction: A --> B -- > C
where [A0] > 0, [B0] = 0, and [C0] = 0
All reactions are first order
Please wait ...
```

```
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=99.695925, k1=0.521301 k2=0.099264
Best fx = 1.886424e+02
For concentration of A
100.000000 97.191541 99.695925 -0.304075 2.576751
54.881164 49.749460 53.333461 -2.820098 7.204100
33.287108 34.920132 31.666622 -4.868210 -9.317005
19.204991 20.935836 17.846946 -7.071314 -14.754083
10.539922 10.647029 9.547425 -9.416552 -10.327803
7.065121 7.072374 6.291698 -10.947070 -11.038388
4.735892 4.684190 4.146192 -12.451728 -11.485399
3.174564 3.462640 2.732316 -13.930962 -21.091533
2.732372 2.729843 2.336746 -14.479209 -14.399982
1.925470 1.991316 1.622307 -15.744908 -18.530928
1.742237 1.902961 1.461684 -16.103084 -23.189006
1.227734 1.234943 1.014787 -17.344750 -17.827268
0.956160 0.993462 0.781944 -18.220387 -21.290958
0.673795 0.675124 0.542872 -19.430718 -19.589378

R^2 = 0.999096 for comparing theoretical and calculated values
R^2 = 0.995639 for comparing observed and calculated values

For concentration of B
42.263600 42.442163 43.438382 2.779654 2.347237
58.705964 59.661405 59.871340 1.985107 0.351877
65.859228 72.352095 66.702284 1.280088 -7.808772
66.528616 71.885129 66.987954 0.690437 -6.812501
64.744220 62.404527 64.995296 0.387797 4.151572
61.998993 64.655093 62.090408 0.147446 -3.966718
58.728804 53.049534 58.705880 -0.039035 10.662384
57.428567 53.426611 57.373024 -0.096716 7.386606
54.323762 57.987089 54.210546 -0.208410 -6.512732
53.429461 52.136034 53.303938 -0.234933 2.240110
50.313196 52.858353 50.156685 -0.311074 -5.111146
48.124014 46.007028 47.954987 -0.351231 4.234047
45.142687 43.914989 44.966441 -0.390419 2.394291

R^2 = 0.998948 for comparing theoretical and calculated values
R^2 = 0.964233 for comparing observed and calculated values

For concentration of C
2.855236 2.651952 2.924082 2.411212 10.261520
8.006927 7.678708 8.157963 1.886311 6.241352
14.935781 15.827193 15.146695 1.412139 -4.299547
```

```
22.931462 20.677345 23.160546 0.998995 12.009282
28.190659 27.586918 28.408932 0.774273 2.979724
33.265114 30.958485 33.459325 0.583827 8.078044
38.096632 41.269052 38.257729 0.422864 -7.296806
39.839061 37.625054 39.986154 0.369219 6.275340
43.750768 45.725945 43.863073 0.256692 -4.073994
44.828301 47.700689 44.930304 0.227540 -5.807852
48.459070 46.322729 48.524453 0.134926 4.753011
50.919826 49.621599 50.958994 0.076920 2.695186
54.183519 48.773154 54.186612 0.005709 11.099259

R^2 = 0.999934 for comparing theoretical and calculated values
R^2 = 0.981958 for comparing observed and calculated values
```

The above output shows the results as A0 = 99.695925, k1 = 0.521301 k2 = 0.099264. They are close to the theoretical values of A0 = 100, k1 = 0.5, and k2 = 0.1. These results deviate a bit more than the ones in the first output on account of the 10% maximum random error used in obtaining the simulated observed concentrations. You can also notice that the set of coefficient of determination values for the second output is less than their counterparts in the first output.

Figures 6.4, 6.5, and 6.6 show the theoretical, observed, and calculated concentrations for all three chemicals. These figures show the red line representing chemical B rise to a maximum in the middle of the observed reaction time. Figure 6.5 has more zigzags in the curves due to the random errors used in calculating the simulated observed concentrations.

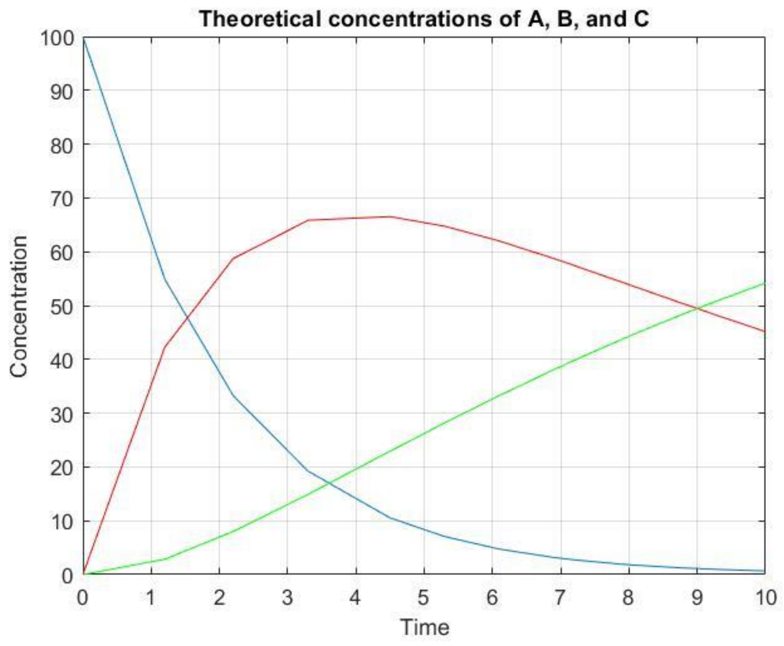

Figure 6.4. The theoretical concentrations of chemicals A, B, and C.

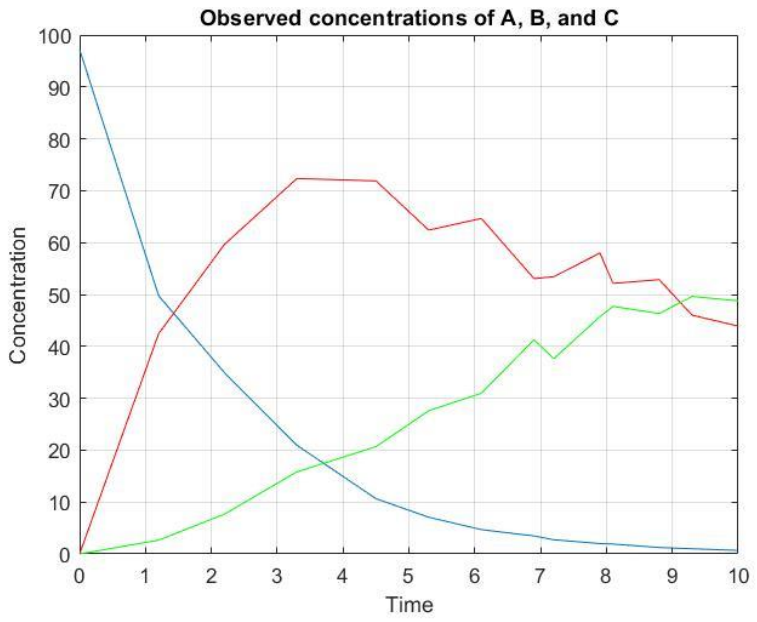

Figure 6.5. The observed concentrations of chemicals A, B, and C.

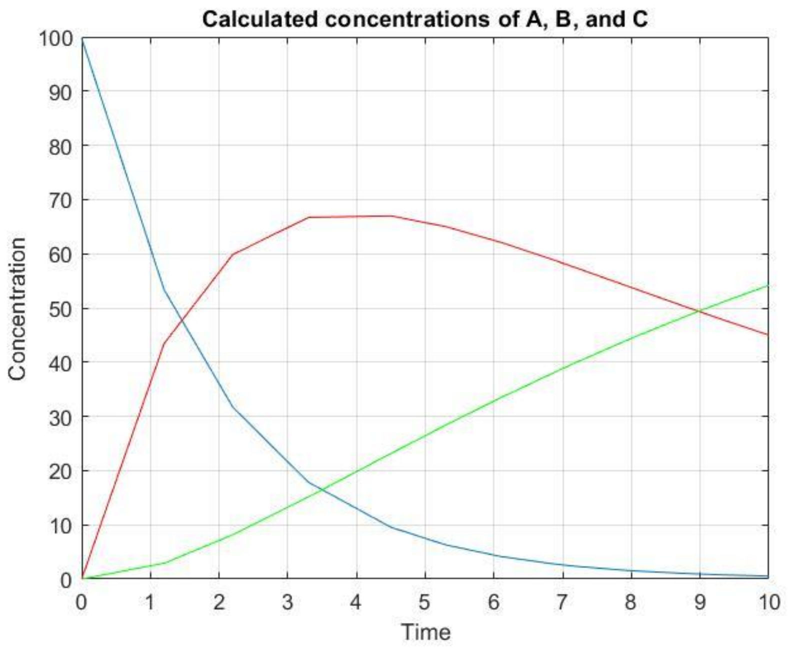

Figure 6.6. The calculated concentrations of chemicals A, B, and C.

Here is a third output set for the chain reaction calculations using 10% maximum random error and plus and minus 10% for the upper and lower ranges, respectively.

```
Reaction: A --> B -- > C
where [A0] > 0, [B0] = 0, and [C0] = 0
All reactions are first order
Please wait ...

Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=99.676403, k1=0.497518 k2=0.098598
Best fx = 1.293750e+02
For concentration of A
100.000000 103.019324 99.676403 -0.323597 -3.244946
54.881164 50.365868 54.866729 -0.026302 8.936331
33.287108 35.576133 33.361046 0.222122 -6.226328
19.204991 18.793794 19.300267 0.496101 2.694897
10.539922 11.009150 10.623804 0.795841 -3.500241
7.065121 7.443516 7.135502 0.996165 -4.138019
4.735892 4.833154 4.792576 1.196886 -0.839572
3.174564 3.486609 3.218944 1.398007 -7.676927
2.732372 2.843656 2.772635 1.473530 -2.497552
1.925470 1.928535 1.957240 1.649970 1.488404
1.742237 1.609845 1.771863 1.700437 10.064184
```

```
1.227734 1.237920 1.250782 1.877271 1.038978
0.956160 1.004255 0.975319 2.003770 -2.881340
0.673795 0.686695 0.688491 2.181131 0.261576

R^2 = 0.999987 for comparing theoretical and calculated values
R^2 = 0.996564 for comparing observed and calculated values

For concentration of B
42.263600 40.231564 42.013365 -0.592081 4.428864
58.705964 61.958367 58.464789 -0.410819 -5.638590
65.859228 60.045716 65.715203 -0.218687 9.441949
66.528616 68.381529 66.517319 -0.016980 -2.726188
64.744220 62.752033 64.817665 0.113439 3.291736
61.998993 62.467556 62.148429 0.241029 -0.510869
58.728804 57.410698 58.943840 0.366151 2.670482
57.428567 56.748283 57.665464 0.412508 1.616228
54.323762 54.771993 54.606037 0.519617 -0.302994
53.429461 57.637032 53.723308 0.549970 -6.790294
50.313196 53.532814 50.642967 0.655435 -5.398273
48.124014 45.575314 48.475372 0.730111 6.363222
45.142687 43.325327 45.519120 0.833875 5.063534

R^2 = 0.999778 for comparing theoretical and calculated values
R^2 = 0.974260 for comparing observed and calculated values

For concentration of C
2.855236 3.009205 2.796308 -2.063852 -7.074847
8.006927 7.556217 7.850568 -1.952807 3.895474
14.935781 15.176698 14.660933 -1.840197 -3.398402
22.931462 23.670604 22.535280 -1.727678 -4.796344
28.190659 30.489497 27.723237 -1.658076 -9.072829
33.265114 33.099958 32.735399 -1.592407 -1.101390
38.096632 40.965723 37.513618 -1.530356 -8.426812
39.839061 36.343679 39.238304 -1.507960 7.964590
43.750768 44.106593 43.113126 -1.457443 -2.252424
44.828301 47.462124 44.181232 -1.443439 -6.912653
48.459070 46.964872 47.782654 -1.395850 1.741262
50.919826 53.779690 50.225711 -1.363153 -6.608404
54.183519 59.490145 53.468792 -1.319086 -10.121598

R^2 = 0.999139 for comparing theoretical and calculated values
R^2 = 0.980844 for comparing observed and calculated values
```

The above output shows the results as A0 = 99.676403, k1 = 0.497518, and k2 = 0.098598. They are still close to the theoretical values of A0 = 100, k1 = 0.5, and k2

= 0.1. These results deviate a bit more than the ones in the two outputs on account of:

1. The 10% maximum random error used in obtaining the simulated observed concentrations.
2. The plus and minus 10% deviation from the theoretical values used in calculating the upper and lower ranges for the optimization variables.

You can also notice that the coefficient of determination values for the third output is slightly less than their counterparts in the first two outputs.

7/Reversible Chemical Reaction: Two Chemicals with First-Second Order Reacions

This section deals with a reversible chemical reaction between compounds A and R:

$$A \leftrightarrows R \tag{7.1}$$

The differential equation describing the chemical reaction is:

$$dA/dt = -k_1A + k_2R^2 \tag{7.2}$$

The author Ancheyta[1] has established tables 4.4 and 4.5 **in his book** to provide analytical solutions for several reversible chemical reactions for one or two reactants and one or two products. Ancheyta's Table 4.4 (**in his book**) shows a set of four reversible reactions where the products are initially absent. By contrast, Table 4.5 (**in his book**) shows a set of four reversible reactions where the products are initially present. I will focus on the case of reversible reactions involving two chemicals. The beauty of Ancheyta's work is that he uses the same set of equations for all eight reversible reactions. The only difference between these equations is how some basic parameters that Ancheyta calls a, b, and c are calculated. The equations that Ancheyta uses are based on x_A which is the fraction of ***remaining*** reactant A and is equal to A/A_0. Thus, x_A is initially 1. As the reversible reaction proceeds, x_A reaches an equilibrium value (greater than 0 and less than 1). Mots books that cover chemical reaction kinetics use x as the fraction of reactant converted at time t. That version of x_A is equal to $1 - A/A_0$.

☛

Table 4.4 and Table 4.5 refer to tables in the book *Chemical Reaction Kinetics-Concepts, Methods and Case Studies*, 2017, by Jorge Ancheyta.

John Wiley & Sons Ltd. The information in these tables is most valuable in calculating analytical values for the concentrations of chemicals involved in various reversible chemical reactions.

The reaction rates for all eight reactions presented by Ancheyta is (I am simplifying x_A by dropping off the subscript A and just using x):

$dx/dt = k_1*(a*x^2 + b*x + c)$ (7.3)
$dA/dt = A_0 * dx/dt$

For the reaction modeled by equation 7.2, the values for coefficients a, b, and c are calculated using the following set of equations:

$K = k_1/k_2$ (7.4)
$M_{RA} = R_0/A_0$
$a = A_0 / K$
$b = –(1 + 2* M_{RA}* A_0 / K)$
$c = 1 – M_{RA}^2 *A_0 / K$

Ancheyta presents the following sets of three equations used for all eight reversible reactions he covers in Tables 4.4 and Table 4.5.

For $b^2 > 4*a*c$ and with $D_1 = \sqrt{}(b^2 – 4*a*c)$ (7.5)
$k_1 = \ln[(2*a*x+B_m)(2*a*x+B_p)*B_p/B_m]/(D_1*t)$
$B_p = b + D_1$
$B_m = b - D_1$

For $b^2< 4*a*c$ and with $D_2 = \sqrt{}(4*a*c – b^2)$
$k_1 = 2/(D_2*t)*[\arctan((2*a*x+b)/D_2) – \arctan(b/D_2)]$

For $b^2= 4*a*c$
$k_1 = [3*a*x/(b*(2*a*x+b))]/t$

While working with the equations in 7.5 may seem a bit intimidating, I was able to rewrite and code these equations to be in the form x = f(t,…).

☛

The MATLAB files mentioned in this section are found in the folder \Chemical Reaction Modeling with Optimized ODEs\Second-Second Order reversible - Tables 4.4 & 4.5\Table 4.4 Reactions

Here is the source code for the MATLAB function *react1* that generates the theoretical and observed values of x:

```
function [Xobs,Xth,tArr] = react1(A0,R0,k1,k2,tArr,percErr)
%REACT calculates the array of concenration FRACTIONS for
% A <==> R, with [A0] > 0 and [R0] > 0
  n = length(tArr);
  Xth = zeros(1,n);
  Xobs = Xth;
  Keq = k1/k2;
  Mra = R0/A0;
  a = -A0 / Keq;
  b = -1 -2*R0/Keq;
  c = 1 - Mra^2*A0/Keq;
  if b^2 > 4*a*c
    D1 = sqrt(b^2 - 4*a*c);
    K = k1*D1;
    Bp = b+D1;
    Bm = b-D1;
    B1 = Bm/Bp;
    Cm = Bm/2/a;
    Cp = Bp/2/a;
    for i=1:n
      t = tArr(i);
      x = (Cp*B1 - Cm*exp(-K*t))/(exp(-K*t) - B1);
      Xth(i) = x;
      Xobs(i) = x * (1 + percErr/100*(2*rand-1));
    end
  elseif b^2 < 4*a*c
    D2 = sqrt(4*a*c - b^2);
    K = k2*D2/2;
    for i=1:n
      t = tArr(i);
      x = (D2*tan(K*t + atan(b/D2)) - b)/(2*a);
      Xth(i) = x;
      Xobs(i) = x * (1 + percErr/100*(2*rand-1));
    end

  else
   for i=1:n
      t = tArr(i);
      x = ((k1*t * b)*b/4/a) / (1 - (k1*t * b)/2);
      Xth(i) = x;
      Xobs(i) = x * (1 + percErr/100*(2*rand-1));
    end
  end
end
```

The parameters of function *react1* are *A0*, *R0*, *k1*, *k2*, *tArr*, and *percErr*. The above code is based on equations 7.4 and 7.5. The function returns the arrays *Xth* and *Xobs* that contain the theoretical and observed values of the x (the fraction of chemical A still remaining).

The listing for function *rkf5_1* is:

```
function sumSqrErr = rkf5_1(x)
% rkf5_1 implements Runge-Kutta-Fehlberg
%   for A <--> R with dR/dt = k1*A - k2*R
  global tData
  global xData
  global incr
  A0 = x(1);
  R0 = x(2);
  kr1 = x(3);
  kr2 = x(4);
  iData = 1;
  nData = length(tData);
  h = incr/10;
  nSteps = fix((tData(nData)-tData(1))/h + 0.5);
  sumSqrErr = 0;
  t = tData(1);
  y = 0;
  Keq = kr1/kr2;
  Mra = R0/A0;
  a = -A0 / Keq;
  b = -1 -2*R0/Keq;
  c = 1 - Mra^2*A0/Keq;
  for iter=1:nSteps
    if t+h > tData(iData)
      sumSqrErr = sumSqrErr + (y - xData(iData))^2;
      iData = iData + 1;
    end
    k1 = h*fx(y,kr1,a,b,c);
    k2 = h*fx(y+k1/4,kr1,a,b,c);
    k3 = h*fx(y+(3*k1+9*k2)/32,kr1,a,b,c);
    k4 = h*fx(y+(1932*k1-7200*k2+7296*k3)/2197,kr1,a,b,c);
    k5 = h*fx(y+439/216*k1-8*k2+3680/513*k3-
845/4104*k4,kr1,a,b,c);
```

```
    k6 = h*fx(y - 8/27*k1 + 2*k2 -3544/2565*k3 +1859/4104*k4 -
11/40*k5,kr1,a,b,c);
    y = y + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5
+ 2/55*k6;
    t = t + h;
  end
end

function r = fx(y,k1,a,b,c)
  r = k1*(a*y^2 + b*y + c);
end
```

The above function has a single array-type parameter *x*. The function copies the elements of array *x* into the local variables *A0*, *R0*, *kr1*, and *kr2*. The function then uses these values to calculate coefficients a, b, c. The local function *fx* uses these coefficients to calculate dx/dt, the normalized reaction rate. The parameters of function *fx* are *y*, *k1*, *a*, *b*, and *c*. The code in function *fx* simply implements equation 7.3.

The file *go1.m* drives the calculations for the reversible equation studied in this section. Here is the listing for that script file:

```
% A <==> R, with [A0] > 0 and [R0] = 0
% (-rA) = k1*A - k2*R^2
% source of equations is
% "Chemical Reaction Kinetics=Concepts, Methods and Case
Studies"
% by Jorge Ancheyta
clc
close
clear all
global tData
global xData
global incr
fprintf('A <==> R at (-rA) = k1*A - k2*R^2\n');
fprintf('Please wait ...\n');
A0 = 1;
R0 = .0;
k1 = 0.1;
k2 = 0.05;
tArr = [0 1.2 2.2 3.3 4.5 5.3 6.1 6.9 7.2 7.9 8.1 8.8 9.3 10];
% tArr = [0 1 2 3 5 7 9 10 12];
incr = findMinDigit(tArr);
[Xobs,Xth,tArr] = react1(A0,R0,k1,k2,tArr,0);
tData = tArr;
```

```
xData = Xobs;
n = length(tArr);
Ath = A0 * (1-Xth);
Aobs = A0 * (1-Xobs);
Rth = (A0+R0) * Xth;
Robs = (A0+R0) * Xobs;
fcn = @rkf5_1;
nvars = 4;
lb = [A0/1.05 0 k1/1.05 k2/1.05];
ub = [A0*1.05 A0 k1*1.05 k2*1.05];
Vmax = (ub - lb)/10;
options = optimoptions('particleswarm', 'SwarmSize', 500,
'MaxIterations', 5000, ...
   'DisplayInterval', 100);
[bestX, bestFx] = particleswarm(fcn,nvars,lb,ub,options);
% % % tic;
% % % [bestX,bestFx] = pso(fcn,lb,ub,Vmax,500,5000);
% % % toc
fprintf('\nBest A0=%f, R0 =%f, k1=%f, k2=%f\n', bestX(1),
bestX(2), bestX(3), bestX(4));
fprintf('Best fx = %e\n', bestFx);
fprintf('Estimated xeq = %f\n', newton1(0.5,1e-
6,100,k1/k2,A0,R0/A0));
Acalc = zeros(n,1);
Rcalc = zeros(n,1);
A00 = bestX(1);
R00 = bestX(2);
kk1 = bestX(3);
kk2 = bestX(4);
Keq = kk1/kk2;
Mra = R00/A00;
a = -A00 / Keq;
b = -1 -2*R00/Keq;
c = 1 - Mra^2*A00/Keq;
if b^2 > 4*a*c
  D1 = sqrt(b^2 - 4*a*c);
  K = kk1*D1;
  Bp = b+D1;
  Bm = b-D1;
  B1 = Bm/Bp;
  Cm = Bm/2/a;
  Cp = Bp/2/a;
  for i=1:n
    t = tArr(i);
    x = (Cp*B1 - Cm*exp(-K*t))/(exp(-K*t) - B1);
    Acalc(i) = A00 * (1- x);
    Rcalc(i) = A00 + R00 - Acalc(i);
    Acalc(i) = Acalc(i); % correction
  end
elseif b^2 < 4*a*c
  D2 = sqrt(4*a*c - b^2);
  K = kk2*D2/2;
  for i=1:n
    t = tArr(i);
    x = (D2*tan(K*t + atan(b/D2)) - b)/(2*a);
    Acalc(i) = A00 * (1- x);
    Rcalc(i) = A00 + R00 - Acalc(i);
    Acalc(i) = Acalc(i); % correction
  end

else
 for i=1:n
    t = tArr(i);
    x = (kk1*t * b)/2 * x + (kk1*t * b)*b/4/a;
    Acalc(i) = A00 * (1- x);
    Rcalc(i) = A00 + R00 - Acalc(i);
    Acalc(i) = Acalc(i) + A0 - A00; % correction
  end
end


figure(1)
plot(tArr,Ath,tArr,Aobs,'r',tArr,Acalc,'g');
title('Concentration of A');
xlabel('Time');
ylabel('Concentration');
grid;

figure(2)
plot(tArr,Rth,tArr,Robs,'r',tArr,Rcalc,'g');
title('Concentration of R');
xlabel('Time');
ylabel('Concentration');
grid;

fprintf('For concentration os A\n');
for i=1:n
  if Ath(i)~=0
    err1 = (Acalc(i) - Ath(i))/Ath(i)*100;
    err2 = (Acalc(i) - Aobs(i))/Aobs(i)*100;
    fprintf('%f %f %f %f %f\n', Ath(i), Aobs(i), Acalc(i), err1,
err2);
  end
end
```

```
fprintf('\n');
fprintf('R^2 = %f for comparing theoretical and calculated
values\n', rsqr(Ath,Acalc));
fprintf('R^2 = %f for comparing observed and calculated
values\n', rsqr(Aobs,Acalc));

fprintf('For concentration os R\n');
for i=1:n
  if Rth(i)~=0
    err1 = (Rcalc(i) - Rth(i))/Rth(i)*100;
    err2 = (Rcalc(i) - Robs(i))/Robs(i)*100;
    fprintf('%f %f %f %f %f\n', Rth(i), Robs(i), Rcalc(i), err1,
err2);
  end
end
fprintf('\n');
fprintf('R^2 = %f for comparing theoretical and calculated
values\n', rsqr(Rth,Rcalc));
fprintf('R^2 = %f for comparing observed and calculated
values\n', rsqr(Robs,Rcalc));
```

The above listing is similar to previous *go.m* script files that I presented earlier. Please notice the following differences:

- The problem has four variables, A0, R0, k1, and k2. The initial value for R0 is 0.
- The call to function *react1* returns arrays *Xth* and *Xobs*. The script uses the values in these arrays to calculate values for arrays *Ath*, *Aobs*, *Rth*, and *Robs*.
- After calling function *particleswarm*, the script copies the values of array *bestX* into variables *A00*, *R00*, *kk1*, and *kk2*. The script uses these values to calculate the values for arrays *Acalc* and *Rcalc*.
- The values for parameters a, b, and c that appear in equation set 7.4 are calculated in functions *react1* and *rkf5*_1 and also in script *go1.m*. I usually write the MATLAB statements to calculate *a*, *b*, and *c* in function *react1* (and any other version of react) and then copy (and edit if needed) these statements in functions *rkf5_1* and script *go1* (and any versions of the function and script I happen to be working with).

Here is a sample session with script go1.m:

```
A <==> R at (-rA) = k1*A - k2*R^2
Please wait ...

Optimization ended: relative change in the objective value
```

```
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=0.952705, R0 =0.022754, k1=0.100377, k2=0.048442
Best fx = 5.251261e-07
Estimated xeq = 0.666667
For concentration of A
1.000000 1.000000 0.952705 -4.729500 -4.729500
0.887175 0.887175 0.844985 -4.755577 -4.755577
0.803934 0.803934 0.765667 -4.759970 -4.759970
0.723172 0.723172 0.688799 -4.753083 -4.753083
0.647092 0.647092 0.616426 -4.739076 -4.739076
0.602775 0.602775 0.574266 -4.729692 -4.729692
0.563143 0.563143 0.536549 -4.722569 -4.722569
0.527813 0.527813 0.502904 -4.719293 -4.719293
0.515595 0.515595 0.491263 -4.719315 -4.719315
0.489124 0.489124 0.466026 -4.722408 -4.722408
0.482059 0.482059 0.459286 -4.724124 -4.724124
0.458960 0.458960 0.437236 -4.733196 -4.733196
0.443919 0.443919 0.422865 -4.742664 -4.742664
0.424735 0.424735 0.404518 -4.760109 -4.760109

R^2 = 0.969315 for comparing theoretical and calculated values
R^2 = 0.969315 for comparing observed and calculated values
For concentration os R
0.112825 0.112825 0.130474 15.643062 15.643062
0.196066 0.196066 0.209792 7.000670 7.000670
0.276828 0.276828 0.286659 3.551635 3.551635
0.352908 0.352908 0.359033 1.735607 1.735607
0.397225 0.397225 0.401193 0.999018 0.999018
0.436857 0.436857 0.438910 0.470121 0.470121
0.472187 0.472187 0.472555 0.077924 0.077924
0.484405 0.484405 0.484196 -0.043045 -0.043045
0.510876 0.510876 0.509433 -0.282387 -0.282387
0.517941 0.517941 0.516173 -0.341355 -0.341355
0.541040 0.541040 0.538223 -0.520779 -0.520779
0.556081 0.556081 0.552594 -0.627159 -0.627159
0.575265 0.575265 0.570941 -0.751517 -0.751517

R^2 = 0.997065 for comparing theoretical and calculated values
R^2 = 0.997065 for comparing observed and calculated values
```

The output shows the results A0 = 0.952705, R0 = 0.022754, k1 = 0.100377, and k2 = 0.048442. These values are close to the theoretical values A0 = 1, R0 = 0, k1 = 0.1, and k2 = 0.05.

Figure 7.1 and 7.2 show the curves for the theoretical, observed, and calculated values for chemicals A and R, respectively. The curves are close since the script is using 0% errors.

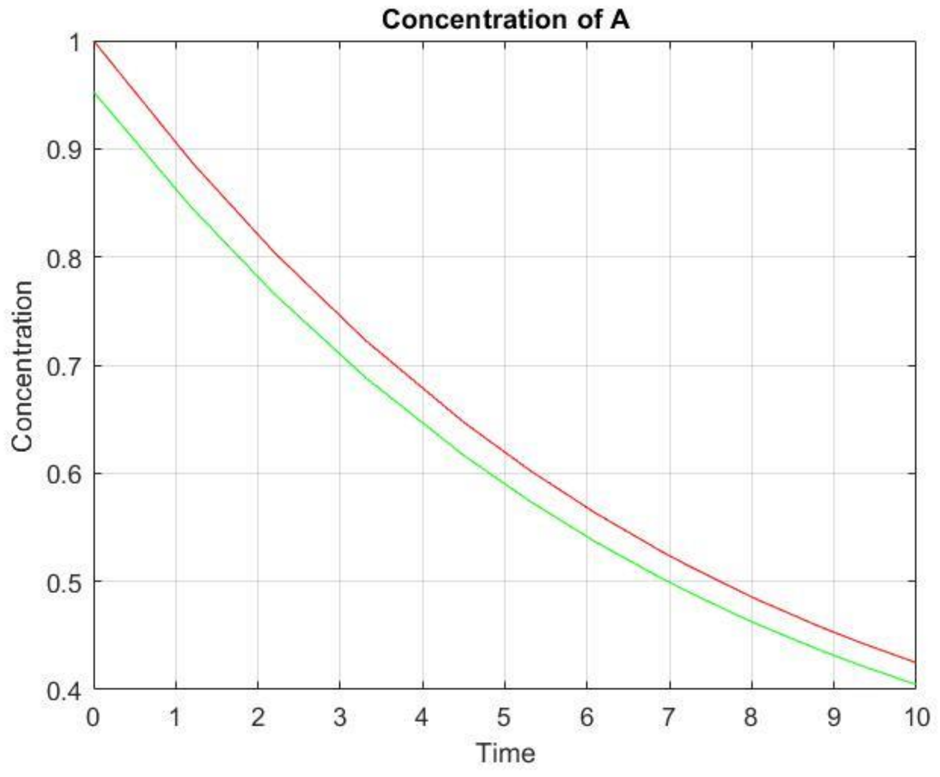

Figure 7.1. The theoretical, observed, and calculated concentrations of chemical A.

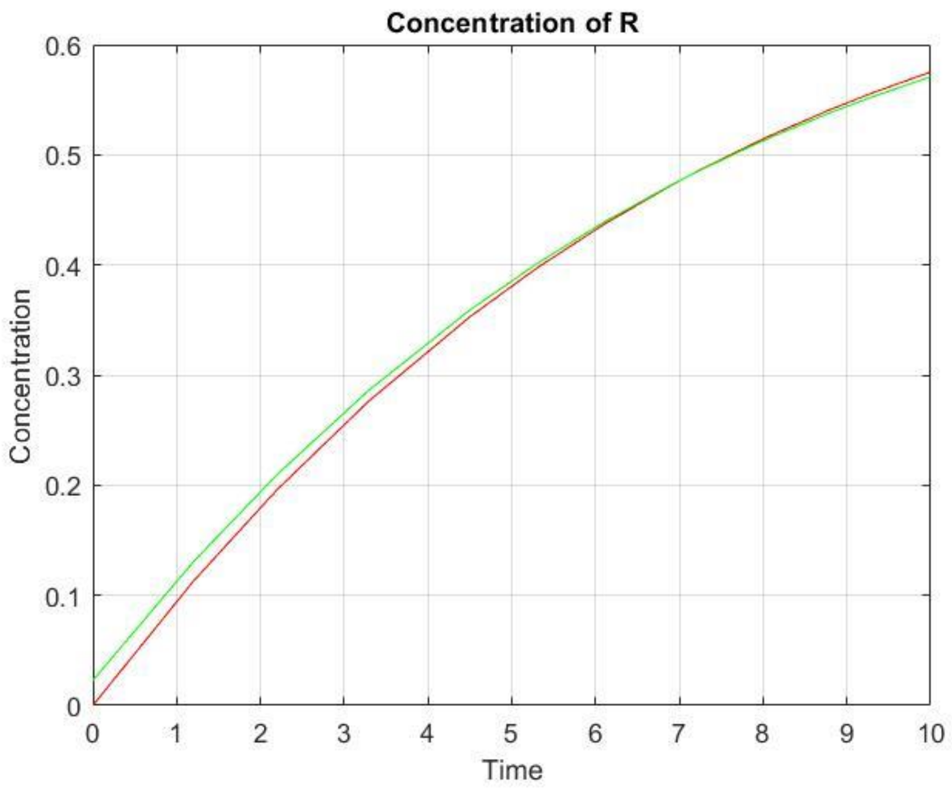

Figure 7.2. The theoretical, observed, and calculated concentrations of chemical R.

Adjusting the maximum random errors to 10% in the *go1.m* script file yields the following sample session:

```
A <==> R at (-rA) = k1*A - k2*R^2
Please wait ...
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=1.050000, R0 =0.158779, k1=0.105000, k2=0.052500
Best fx = 7.624325e-03
Estimated xeq = 0.666667
For concentration of A
1.000000 1.000000 1.050000 5.000000 5.000000
0.887175 0.894700 0.928692 4.679658 3.799219
0.803934 0.797280 0.841436 4.664799 5.538255
0.723172 0.701306 0.758876 4.937015 8.208919
0.647092 0.633141 0.683258 5.589026 7.915596
0.602775 0.570238 0.640302 6.225655 12.286805
0.563143 0.604822 0.602646 7.014689 -0.359711
0.527813 0.566094 0.569740 7.943656 0.644148
0.515595 0.538617 0.558518 8.324929 3.694971
```

```
0.489124 0.456429 0.534500 9.277005 17.104735
0.482059 0.528002 0.528162 9.563710 0.030238
0.458960 0.498811 0.507663 10.611805 1.774777
0.443919 0.473858 0.494512 11.397062 4.358769
0.424735 0.439967 0.477981 12.536277 8.640265

R^2 = 0.935470 for comparing theoretical and calculated values
R^2 = 0.933682 for comparing observed and calculated values
For concentration of R
0.112825 0.105300 0.280087 148.250073 165.991063
0.196066 0.202720 0.367344 87.357033 81.207731
0.276828 0.298694 0.449904 62.521328 50.623623
0.352908 0.366859 0.525521 48.911681 43.248927
0.397225 0.429762 0.568477 43.112312 32.277259
0.436857 0.395178 0.606133 38.748782 53.382232
0.472187 0.433906 0.639039 35.335930 47.275873
0.484405 0.461383 0.650261 34.239233 40.937242
0.510876 0.543571 0.674279 31.984932 24.046247
0.517941 0.471998 0.680618 31.408336 44.199284
0.541040 0.501189 0.701116 29.586604 39.890435
0.556081 0.526142 0.714267 28.446501 35.755550
0.575265 0.560033 0.730798 27.036843 30.491971

R^2 = 0.074024 for comparing theoretical and calculated values
R^2 = -0.145325 for comparing observed and calculated values
```

The output shows the results A0 = 1.050000, R0 = 0.158779, k1 = 0.105000, and k2 = 0.052500. These values are still close to the theoretical values A0 = 1, R0 = 0, k1 = 0.1, and k2 = 0.05.

Figure 7.3 and 7.4 show the curves for the theoretical, observed, and calculated values for chemicals A and R, respectively. The curves show the effect of using 10% maximum random error on the red line which represents the observed values.

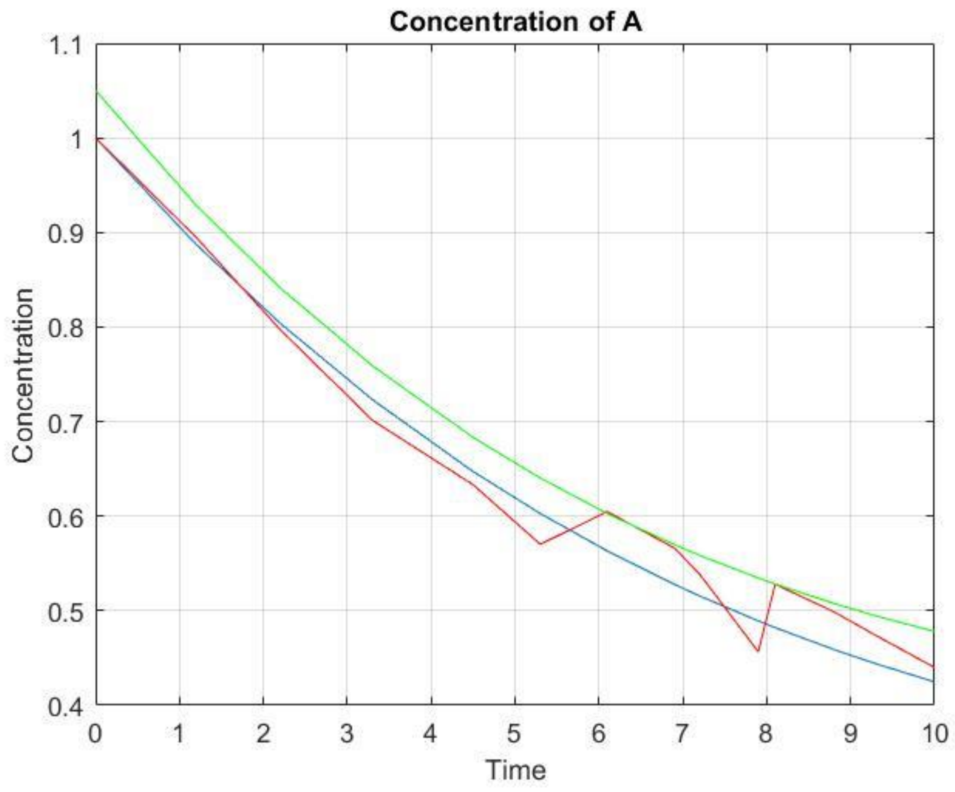

Figure 7.3. The theoretical, observed, and calculated concentrations of chemical A.

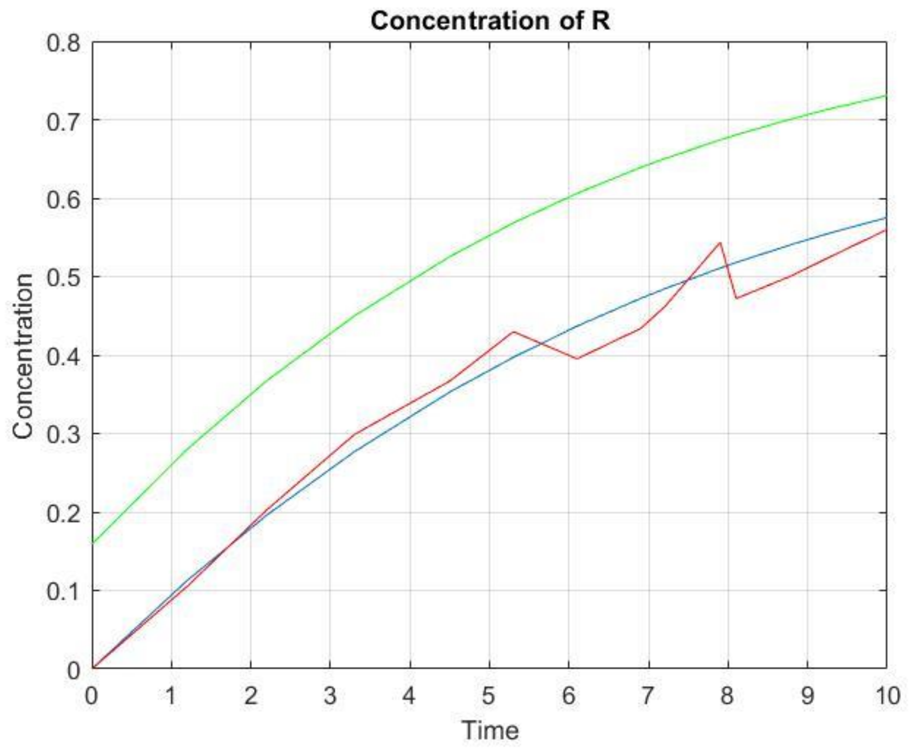

Figure 7.4. The theoretical, observed, and calculated concentrations of chemical R.

8/Reversible Reaction: Two Reactants and Two Products with Second-Second Order Reactions

This section deals with a reversible chemical reaction between compounds A, B, R and S:

$$A + B \leftrightarrows R + S \qquad (8.1)$$

The differential equation describing the chemical reaction is:

$$dA/dt = -k_1 * A * B + k_2 * R * S \qquad (8.2)$$

I again use Ancheyta[1] and he how he models reversible reactions. Again, the rate of change in x (the remaining fraction of compound A) is:

$$dx/dt = k_1 * (a * x^2 + b * x + c) \qquad (8.3)$$

For the reaction modeled by equation 13, the values for coefficients a, b, and c are calculated using the following equations (taken from Tale 4.5[1]):

$$\begin{aligned}
&K = k_1/k_2 \\
&M_{RA} = R_0/A_0 \qquad (8.4) \\
&M_{BA} = B_0/A_0 \\
&M_{SA} = S_0/A_0 \\
&M_{BS} = B_0/S_0 \\
&a = A_0 * (1 + 1/ K) \\
&b = -A_0 * (1 + M_{BA} + (M_{RA} + M_{BS}) / K)) \\
&c = A_0 * (M_{BA} - M_{RA} * M_{SA}/ K)
\end{aligned}$$

☛

The MATLAB files mentioned in this section are found in the folder \Chemical Reaction Modeling with Optimized ODEs\Second-Second Order reversible - Tables 4.4 & 4.5\Table 4.5 Reactions

The listing of *react1* (based on Table 4.5) is:

```
function [Xobs,Xth,tArr] =
react1(A0,B0,R0,S0,k1,k2,tArr,percErr)
%REACT calculates the array of concentration FRACTIONS for
% A + B <==> R + S, with all chemicals' conc > 0
% (-rA) = k1*A*B - k2*R*S
```

```
  n = length(tArr);
  Xth = zeros(1,n);
  Xobs = Xth;
  Keq = k1/k2;
  Mra = R0/A0;
  Msa = S0/A0;
  Mba = B0/A0;
  Mbs = B0/S0;
  a = A0 *(1 +  1/Keq);
  b = -A0*(1 + Mba + (Mra + Mbs)/Keq);
  c = A0*(Mba - Mra*Msa/Keq);
  if b^2 > 4*a*c
    D1 = sqrt(b^2 - 4*a*c);
    K = k1*D1;
    Bp = b+D1;
    Bm = b-D1;
    B1 = Bm/Bp;
    Cm = Bm/2/a;
    Cp = Bp/2/a;
    for i=1:n
      t = tArr(i);
      x = (Cp*B1 - Cm*exp(-K*t))/(exp(-K*t) - B1);
      Xth(i) = x;
      Xobs(i) = x * (1 + percErr/100*(2*rand-1));
    end
  elseif b^2 < 4*a*c
    D2 = sqrt(4*a*c - b^2);
    K = k2*D2/2;
    for i=1:n
      t = tArr(i);
      x = (D2*tan(K*t + atan(b/D2)) - b)/(2*a);
      Xth(i) = x;
      Xobs(i) = x * (1 + percErr/100*(2*rand-1));
    end

  else
   for i=1:n
      t = tArr(i);
      x = ((k1*t * b)*b/4/a) / (1 - (k1*t * b)/2);
      Xth(i) = x;
      Xobs(i) = x * (1 + percErr/100*(2*rand-1));
    end
  end
end
```

The function *react1* has six optimization variables that pass as parameters—*A0*, *B0*, *R0*, *S0*, *k1*, and *k2*. Other parameters include the usual *tArr* and *percErr*. The function react1 implements the equation set 8.4 to calculate the array of x values, *Xth* and *Xobs*.

The listing of *rkf5_1* is:

```
function sumSqrErr = rkf5_1(x)
% rkf5_1 implements Runge-Kutta-Fehlberg
% A + B <==> R + S, with all chemcials's conc > 0
% (-rA) = k1*A*B - k2*R*S
  global tData
  global xData
  global incr
  A0 = x(1);
  B0 = x(2);
  R0 = x(3);
  S0 = x(4);
  kr1 = x(5);
  kr2 = x(6);
  iData = 1;
  nData = length(tData);
  h = incr/10;
  nSteps = fix((tData(nData)-tData(1))/h + 0.5);
  sumSqrErr = 0;
  t = tData(1);
  y = 0;
  Keq = kr1/kr2;
  Mra = R0/A0;
  Msa = S0/A0;
  Mba = B0/A0;
  Mbs = B0/S0;
  a = A0 *(1 +  1/Keq);
  b = -A0*(1 + Mba + (Mra + Mbs)/Keq);
  c = A0*(Mba - Mra*Msa/Keq);
  for iter=1:nSteps
    if t+h > tData(iData)
      sumSqrErr = sumSqrErr + (y - xData(iData))^2;
      iData = iData + 1;
    end
    k1 = h*fx(y,kr1,a,b,c);
    k2 = h*fx(y+k1/4,kr1,a,b,c);
    k3 = h*fx(y+(3*k1+9*k2)/32,kr1,a,b,c);
    k4 = h*fx(y+(1932*k1-7200*k2+7296*k3)/2197,kr1,a,b,c);
    k5 = h*fx(y+439/216*k1-8*k2+3680/513*k3-
845/4104*k4,kr1,a,b,c);
    k6 = h*fx(y - 8/27*k1 + 2*k2 -3544/2565*k3 +1859/4104*k4 -
11/40*k5,kr1,a,b,c);
```

```
    y = y + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5
+ 2/55*k6;
    t = t + h;
  end
end

function r = fx(y,k1,a,b,c)
  r = k1*(a*y^2 + b*y + c);
end
```

The parameter of the above function is *x*—an array passing six optimization variables. The function copies the elements of array x into the local variables *A0*, *B0*, *R0*, *S0*, *kr1*, and *kr2*. Notice that the local function *fx* is coded just like the one in the last section.

The listing for the script file *go1.m*, for the reaction examined in this section, is:

```
% A + B <==> R + S, with all chemicals's conc > 0
% (-rA) = k1*A*B - k2*R*S
% source of equations is
% "Chemical Reaction Kinetics=Concepts, Methods and Case
Studies"
% by Jorge Ancheyta
clc
close
clear all
global tData
global xData
global incr
fprintf('A + B <==> R + S at (-rA) = k1*A*B - k2*R*S\n');
fprintf('Please wait ...\n');
A0 = 1;
B0 = 0.8;
R0 = 0.05;
S0 = 0.075;
k1 = 0.1;
k2 = 0.05;
tArr = [0 1.2 2.2 3.3 4.5 5.3 6.1 6.9 7.2 7.9 8.1 8.8 9.3 10];
% tArr = [0 1 2 3 5 7 9 10 12];
incr = findMinDigit(tArr);
[Xobs,Xth,tArr] = react1(A0,B0,R0,S0,k1,k2,tArr,0); %0 % error
tData = tArr;
xData = Xobs;
n = length(tArr);
Ath = A0 * (1-Xth);
Aobs = A0 * (1-Xobs);
```

```
Rth = (A0+R0) * Xth;
Robs = (A0+R0) * Xobs;
fcn = @rkf5_1;
nvars = 6;
lb = [A0 B0 R0 S0 k1 k2]/1.05;
ub = 1.05^2*lb;
Vmax = (ub - lb)/10;
options = optimoptions('particleswarm', 'SwarmSize', 500,
'MaxIterations', 5000, ...
   'DisplayInterval', 100);
[bestX, bestFx] = particleswarm(fcn,nvars,lb,ub,options);
fprintf('\nBest A0=%f, B0=%f\n', bestX(1), bestX(2));
fprintf('Best R0=%f, S0=%f\n', bestX(3), bestX(4));
fprintf('Best k1=%f, k2=%f\n', bestX(5), bestX(6));
fprintf('Best fx = %e\n', bestFx);
%fprintf('Estimated xeq = %f\n', newton1(0.5,1e-
6,100,k1/k2,A0,R0/A0));
Acalc = zeros(n,1);
Rcalc = zeros(n,1);
A00 = bestX(1);
B00 = bestX(2);
R00 = bestX(3);
S00 = bestX(4);
kk1 = bestX(5);
kk2 = bestX(6);
Keq = kk1/kk2;
Mba = B00/A00;
Mra = R00/A00;
Msa = S00/A00;
Mbs = B00/S00;
a = A00 *(1 +  1/Keq);
b = -A00*(1 + Mba + (Mra + Mbs)/Keq);
c = A00*(Mba - Mra*Msa/Keq);
if  b^2 > 4*a*c
  D1 = sqrt(b^2 - 4*a*c);
  K = kk1*D1;
  Bp = b+D1;
  Bm = b-D1;
  B1 = Bm/Bp;
  Cm = Bm/2/a;
  Cp = Bp/2/a;
  for i=1:n
    t = tArr(i);
    x = (Cp*B1 - Cm*exp(-K*t))/(exp(-K*t) - B1);
    Acalc(i) = A00 * (1- x);
    Rcalc(i) = A00 + R00 - Acalc(i);
    Acalc(i) = Acalc(i); % correction
  end
elseif  b^2 < 4*a*c
  D2 = sqrt(4*a*c - b^2);
  K = kk2*D2/2;
  for i=1:n
    t = tArr(i);
    x = (D2*tan(K*t + atan(b/D2)) - b)/(2*a);
    Acalc(i) = A00 * (1- x);
    Rcalc(i) = A00 + R00 - Acalc(i);
    Acalc(i) = Acalc(i); % correction
  end

else
 for i=1:n
    t = tArr(i);
    x = (kk1*t * b)/2 * x + (kk1*t * b)*b/4/a;
    Acalc(i) = A00 * (1- x);
    Rcalc(i) = A00 + R00 - Acalc(i);
    Acalc(i) = Acalc(i) + A0 - A00; % correction
  end
end

figure(1)
plot(tArr,Ath,tArr,Aobs,'r',tArr,Acalc,'g');
title('Concentration of A');
xlabel('Time');
ylabel('Concentration');
grid;

figure(2)
plot(tArr,Rth,tArr,Robs,'r',tArr,Rcalc,'g');
title('Concentration of R');
xlabel('Time');
ylabel('Concentration');
grid;

fprintf('For concentration of A\n');
for i=1:n
  if Ath(i)~=0
    err1 = (Acalc(i) - Ath(i))/Ath(i)*100;
    err2 = (Acalc(i) - Aobs(i))/Aobs(i)*100;
    fprintf('%f %f %f %f %f\n', Ath(i), Aobs(i), Acalc(i), err1,
err2);
  end
end
fprintf('\n');
fprintf('R^2 = %f for comparing theoretical and calculated
values\n', rsqr(Ath,Acalc));
fprintf('R^2 = %f for comparing observed and calculated
values\n', rsqr(Aobs,Acalc));

fprintf('For concentration os R\n');
for i=1:n
  if Rth(i)~=0
    err1 = (Rcalc(i) - Rth(i))/Rth(i)*100;
    err2 = (Rcalc(i) - Robs(i))/Robs(i)*100;
    fprintf('%f %f %f %f %f\n', Rth(i), Robs(i), Rcalc(i), err1,
err2);
  end
end
fprintf('\n');
fprintf('R^2 = %f for comparing theoretical and calculated
values\n', rsqr(Rth,Rcalc));
fprintf('R^2 = %f for comparing observed and calculated
values\n', rsqr(Robs,Rcalc));
```

The code for the above listing is an extended version of the one that appear in the last section. The main difference is the presence of data for chemicals B and S. Thus, the script in *go1.m* optimizes a function with six variables. Here is a sample session with the script in file go1.m:

```
A + B <==> R + S at (-rA) = k1*A*B - k2*R*S
Please wait ...
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=0.952381, B0=0.765337
Best R0=0.052380, S0=0.071429
Best k1=0.104966, k2=0.052500
Best fx = 6.488403e-08
For concentration of A
1.000000 1.000000 0.952381 -4.761905 -4.761905
0.935489 0.935489 0.890792 -4.777914 -4.777914
0.910784 0.910784 0.867283 -4.776152 -4.776152
0.897555 0.897555 0.854733 -4.770922 -4.770922
0.890964 0.890964 0.848501 -4.765920 -4.765920
0.888779 0.888779 0.846442 -4.763530 -4.763530
0.887514 0.887514 0.845252 -4.761823 -4.761823
0.886781 0.886781 0.844564 -4.760648 -4.760648
0.886594 0.886594 0.844389 -4.760314 -4.760314
0.886281 0.886281 0.844097 -4.759708 -4.759708
```

```
0.886216 0.886216 0.844036 -4.759572 -4.759572
0.886047 0.886047 0.843878 -4.759197 -4.759197
0.885967 0.885967 0.843804 -4.759004 -4.759004
0.885892 0.885892 0.843734 -4.758810 -4.758810

R^2 = -0.494633 for comparing theoretical and calculated values
R^2 = -0.494633 for comparing observed and calculated values
For concentration os R
0.067736 0.067736 0.113968 68.252587 68.252587
0.093677 0.093677 0.137477 46.756613 46.756613
0.107568 0.107568 0.150028 39.472809 39.472809
0.114488 0.114488 0.156259 36.485365 36.485365
0.116782 0.116782 0.158319 35.567899 35.567899
0.118110 0.118110 0.159509 35.050364 35.050364
0.118880 0.118880 0.160196 34.754404 34.754404
0.119077 0.119077 0.160372 34.679294 34.679294
0.119405 0.119405 0.160664 34.553901 34.553901
0.119473 0.119473 0.160725 34.527783 34.527783
0.119651 0.119651 0.160883 34.459903 34.459903
0.119735 0.119735 0.160957 34.427799 34.427799
0.119813 0.119813 0.161027 34.397741 34.397741

R^2 = -0.440060 for comparing theoretical and calculated values
R^2 = -0.440060 for comparing observed and calculated values
```

The results in the above output show A0 = 0.952381, B0 = 0.765337, R0 = 0.052380, S0 = 0.071429, k1 = 0.104966, and k2 = 0.052500. Compare these values with the theoretical ones of A0 = 1, B0 = 0.8, R0 = 0.05, S0 = 0.075, k1 = 0.1, and k2 = 0.05. The results are close enough. The tabulated values show some deviation between the calculated concentrations of the chemicals A and R and their theoretical (and also observed counterpart, calculated with 0% error).

Here is another sample output with observed concentrations calculated at 10% maximum random error:

```
A + B <==> R + S at (-rA) = k1*A*B - k2*R*S
Please wait ...
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=1.028834, B0=0.761905
Best R0=0.052314, S0=0.078750
Best k1=0.096000, k2=0.047619
Best fx = 4.116528e-04
```

```
For concentration of A
1.000000 1.000000 1.028834 2.883361 2.883361
0.935489 0.938118 0.966021 3.263793 2.974432
0.910784 0.907346 0.940105 3.219295 3.610402
0.897555 0.906866 0.925239 3.084399 2.025919
0.890964 0.894640 0.917251 2.950401 2.527323
0.888779 0.888418 0.914412 2.884086 2.925937
0.887514 0.898110 0.912678 2.835334 1.622013
0.886781 0.881662 0.911617 2.800776 3.397572
0.886594 0.875805 0.911336 2.790745 4.056932
0.886281 0.880867 0.910851 2.772229 3.403951
0.886216 0.890544 0.910746 2.767999 2.268495
0.886047 0.886942 0.910467 2.756117 2.652349
0.885967 0.881701 0.910330 2.749878 3.246973
0.885892 0.880606 0.910196 2.743462 3.360247

R^2 = 0.265590 for comparing theoretical and calculated values
R^2 = 0.274464 for comparing observed and calculated values
For concentration os R
0.067736 0.064976 0.115126 69.961921 77.181872
0.093677 0.097287 0.141043 50.563193 44.976416
0.107568 0.097790 0.155909 44.940373 59.431965
0.114488 0.110628 0.163897 43.156370 48.151848
0.116782 0.117161 0.166735 42.775104 42.312685
0.118110 0.106984 0.168470 42.637588 57.471882
0.118880 0.124255 0.169530 42.605897 36.437871
0.119077 0.130404 0.169811 42.606890 30.219226
0.119405 0.125090 0.170297 42.621401 36.139371
0.119473 0.114928 0.170401 42.627157 48.267445
0.119651 0.118711 0.170681 42.648666 43.778780
0.119735 0.124214 0.170818 42.663437 37.519311
0.119813 0.125364 0.170952 42.681538 36.364043

R^2 = -0.591027 for comparing theoretical and calculated values
R^2 = -0.563634 for comparing observed and calculated values
```

The above results show A0 = 1.028834, B0 = 0.761905, R0 = 0.052314, S0 = 0.078750, k1 = 0.096000, and k2 = 0.047619. These values are still reasonably close to the theoretical ones of A0 = 1, B0 = 0.8, R0 = 0.05, S0 = 0.075, k1 = 0.1, and k2 = 0.05.

9/Using ODE Solver to Generate Data for Chain Reaction

Section 6 presented the simple chain reaction that involved thee chemicals and two first-order chemical reactions:

$$A \rightarrow B \rightarrow C$$

With the chemical reaction rates of:

$dA/dt = -k_1A$ (9.1)
$dB/dt = k_1A - k_2B$ (9.2)
$dC/dt = k_2B$ (9.3)

Where t is time, A is the concentration of reactant A, and k_1 is the first reaction rate constant, B is the concentration of reactant B, and k_2 is the second reaction rate constant. This section assumes that we don't have the analytical solution for the above differential equations. The solution would be to use Runge-Kutta-Fehlberg to generate the data for the concentration of the chemicals A, B, and C.

☛

The MATLAB files mentioned in this section are found in the folder \Chemical Reaction Modeling with Optimized ODEs\Chain Reaction First Order 3 Reactions - Ver 1

Here is the listing for the special version of MATLAB function *react2*:

```
function [ConcObs,ConcTh,tArr] =
react2(A0,B0,C0,kr1,kr2,tArr,percErr)
%REACT calculates the array of concentrations for
% A --> B --> C  where B0 = C0 = 0
  global incr
  n = length(tArr);
  ConcTh = zeros(n,3);
  ConcObs = zeros(n,3);
  iData = 1;
  h = incr/100;
  nSteps = fix((tArr(n)-tArr(1))/h + 0.5);
  t = tArr(1);
  y = [A0 B0 C0];
  for iter=1:nSteps
    if t+h > tArr(iData)
      for j=1:3
        ConcTh(iData,j) = y(j);
      end
      iData = iData + 1;
    end
    k1 = h*fx(y,kr1,kr2);
    k2 = h*fx(y+k1/4,kr1,kr2);
    k3 = h*fx(y+(3*k1+9*k2)/32,kr1,kr2);
    k4 = h*fx(y+(1932*k1-7200*k2+7296*k3)/2197,kr1,kr2);
    k5 = h*fx(y+439/216*k1-8*k2+3680/513*k3-
845/4104*k4,kr1,kr2);
    k6 = h*fx(y - 8/27*k1 + 2*k2 -3544/2565*k3 +1859/4104*k4 -
11/40*k5,kr1,kr2);
    y = y + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5
+ 2/55*k6;
    t = t + h;
  end

  for j=1:3
    ConcTh(iData,j) = y(j);
  end

  for i=1:n
    for j=1:3
      ConcObs(i,j) = ConcTh(i,j) * (1+percErr/100*(2*rand-1));
    end
  end
end

function y = fx(x,k1,k2)
  y(1) = -k1*x(1);
  y(2) = k1*x(1)-k2*x(2);
  y(3) = k2*x(2);
end
```

The parameters of function *react2* are *A0*, *B0*, *C0*, *kr1*, *kr2*, *tArr*, and *percErr*. The first five parameters are related to the chemical reactions. Function *react2* is a special version of Runge-Kutta-Fehlberg that generates the concentrations of the chemicals A, B, and C. The function *react2* returns the matrices *ConcTh* and *ConcObs* that store the concentrations of chemicals A, B, and C in columns 1, 2, and 3, respectively, of these two matrices. Notice that function *react2* calculates the integration increment, *h*, as *incr/100* and not the usual *incr/10* that you see in other versions of function *react*. The reason is two-fold. The first, and main reason, is that we need to generate more accurate concentrations values. Second, since the driving script calls *react2* only once we can afford to have smaller values of h.

The counterpart of function *react2* is function *rkf5_2*, listed next:

```
function sumSqrErr = rkf5_2(x)
% rkf5 implements Runge-Kutta-Fehlberg
% A --> B --> C --> D where B0 = C0 = D0 = 0
  global tData
  global cData
  global incr;
  A0 = x(1);
  B0 = x(2);
  C0 = x(3);
  kr1 = x(4);
  kr2 = x(5);
  iData = 1;
  nData = length(tData);
  h = incr/10;
  nSteps = fix((tData(nData)-tData(1))/h + 0.5);
  sumSqrErr = 0;
  t = tData(1);
  y = [A0 B0 C0];
  for iter=1:nSteps
    if t+h > tData(iData)
      for j=1:3
        sumSqrErr = sumSqrErr + (y(j) - cData(iData,j))^2;
      end
      iData = iData + 1;
    end
    k1 = h*fx(y,kr1,kr2);
    k2 = h*fx(y+k1/4,kr1,kr2);
    k3 = h*fx(y+(3*k1+9*k2)/32,kr1,kr2);
    k4 = h*fx(y+(1932*k1-7200*k2+7296*k3)/2197,kr1,kr2);
    k5 = h*fx(y+439/216*k1-8*k2+3680/513*k3-
845/4104*k4,kr1,kr2);
    k6 = h*fx(y - 8/27*k1 + 2*k2 -3544/2565*k3 +1859/4104*k4 -
11/40*k5,kr1,kr2);
    y = y + 16/135*k1 + 6656/12825*k3 + 28561/56430*k4 - 9/50*k5
+ 2/55*k6;
    t = t + h;
  end
end

function y = fx(x,k1,k2)
  y(1) = -k1*x(1);
  y(2) = k1*x(1)-k2*x(2);
  y(3) = k2*x(2);
end
```

The script *go2.m* contains the test code:

```
% Chain rection:
%
%   A --> B -- > C --> D
%
%   where[A0] > 0, [B0] = 0, [C0] = 0, and [D0] = 0
```

```
%
clc
close all
clear
global tData
global cData % a matrix
global incr

fprintf('Reaction: A --> B -- > C --> D\n');
fprintf('where[A0] > 0, [B0] = 0, [C0] = 0, and [D0] = 0\n');
fprintf('All reactions are first order\n');
fprintf('Please wait ...\n');
A0 = 100;
k1 = 0.05;
k2 = 0.5;
k3 = 2;
tArr = [0 1.2 2.2 3.3 4.5 5.3 6.1 6.9 7.2 7.9 8.1 8.8 9.3 10];
% tArr = [0 1 2 3 5 7 9 10 12];
incr = findMinDigit(tArr);
[ConcObs,ConcTh,tArr] = react(A0,k1,k2,k3,tArr,10);
tData = tArr;
cData = ConcObs;

fcn = @rkf5;
nvars = 4;
lb = [50 .01 0.1 1];
ub = [200 0.1 1 5];
options = optimoptions('particleswarm', 'SwarmSize', 500,
'MaxIterations', 5000, ...
   'DisplayInterval', 100);
[bestX, bestFx] = particleswarm(fcn,nvars,lb,ub,options);
fprintf('\nBest A0=%f, k1=%f, k2=%f, k3=%f\n', bestX(1),
bestX(2), bestX(3), bestX(4));
fprintf('Best fx = %e\n', bestFx);
% [bestX, bestFx] = scout([50 .01], [200 .1], [10 .01], [.1
.001], 10000, 100, 50, false, true)
n = length(tArr);
ConcCalc = zeros(n,4);
A00 = bestX(1);
kk1 = bestX(2);
kk2 = bestX(3);
kk3 = bestX(4);
rB = kk1/(kk2-kk1);
c0 = kk1*kk2;
c1 = (kk2-kk1)*(kk3-kk1);
c2 = (kk1-kk2)*(kk3-kk2);
c3 = (kk1-kk3)*(kk2-kk3);
```

```
for i=1:n
  t = tArr(i);
  ConcCalc(i,1) = A00*exp(-kk1*t);
  ConcCalc(i,2) = A00*rB*(exp(-kk1*t) - exp(-kk2*t));
  ConcCalc(i,3) = A00*c0*(exp(-kk1*t)/c1 + exp(-kk2*t)/c2 +
exp(-kk3*t)/c3);
  ConcCalc(i,4) = A00 - ConcCalc(i,1) - ConcCalc(i,2) -
ConcCalc(i,3);
end

for j=1:4
  figure(j)

plot(tArr,ConcTh(:,j),tArr,ConcObs(:,j),'r',tArr,ConcCalc(:,j),'
g');
  if j == 1
    title('Concentration of A');
  elseif j == 2
    title('Concentration of B');
  elseif j == 3
    title('Concentration of C');
  else
    title('Concentration of D');
  end
  xlabel('Time');
  ylabel('Concentration');
  grid;
end

figure(5)
plot(tArr,ConcTh(:,1),tArr,ConcTh(:,2),'r',tArr,ConcTh(:,3),'g',
tArr,ConcTh(:,4),'y');
title('Theoretical concentrations of A, B, and C');
xlabel('Time');
ylabel('Concentration');
grid;

figure(6)
plot(tArr,ConcObs(:,1),tArr,ConcObs(:,2),'r',tArr,ConcObs(:,3),'
g',tArr,ConcObs(:,4),'y');
title('Observed concentrations of A, B, and C');
xlabel('Time');
ylabel('Concentration');
grid;

figure(7)
```

```
plot(tArr,ConcCalc(:,1),tArr,ConcCalc(:,2),'r',tArr,ConcCalc(:,3
),'g',tArr,ConcCalc(:,4),'y');
title('Calculated concentrations of A, B, and C');
xlabel('Time');
ylabel('Concentration');
grid;


for j =1:4
  if j == 1
    fprintf('For concentration of A\n');
  elseif j == 2
    fprintf('For concentration of B\n');
  elseif j == 3
    fprintf('For concentration of C\n');
  else
    fprintf('For concentration of D\n');
  end
  for i=1:n
    if ConcTh(i,j)~=0
      err1 = (ConcCalc(i,j) - ConcTh(i,j))/ConcTh(i,j)*100;
      err2 = (ConcCalc(i,j) - ConcObs(i,j))/ConcObs(i,j)*100;
      fprintf('%f %f %f %f %f\n', ConcTh(i,j), ConcObs(i,j),
ConcCalc(i,j), err1, err2);
    end
  end
  fprintf('\n');
  fprintf('R^2 = %f for comparing theoretical and calculated
values\n', rsqr(ConcTh(:,j),ConcCalc(:,j)));
  fprintf('R^2 = %f for comparing observed and calculated
values\n\n', rsqr(ConcObs(:,j),ConcCalc(:,j)));
end
```

The above script is like the one in section 6 with the following noted difference. The above script calls function *react2* twice. The first call returns the theoretical and observed concentration values. The second call returns the calculated concentrations of the chemicals. The return list contains the matrix *ConcDummy*. You can replace that matrix name with the tilde character (~) to tell MATLAB to discard the returned value for the corresponding return parameter. The second call to function *react2* is necessary since we are not using the analytical equations for the solutions of the differential equations. The normal use for this method is for when the analytical solutions are not available.

Here is a sample session with the above script:

```
Reaction: A --> B -- > C --> D
where[A0] > 0, [B0] = 0, [C0] = 0, and [D0] = 0
All reactions are first order
Please wait ...
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than
OPTIONS.FunctionTolerance.

Best A0=98.164769, k1=0.525000, B0=21.000000, k2=0.100533, and
C0 =9.523810
Best fx = 1.602221e+02
For concentration of A
100.000000 97.572487 98.164769 -1.835231 0.607017
54.881164 51.293188 52.281751 -4.736439 1.927279
33.287108 31.619224 30.927550 -7.088504 -2.187509
19.204991 17.988198 17.359630 -9.608758 -3.494335
10.539922 11.561227 9.245596 -12.280223 -20.029279
7.068655 6.542167 6.077980 -14.015041 -7.095305
4.738261 5.083712 3.993517 -15.717658 -21.444851
3.176151 2.996903 2.623928 -17.386560 -12.445359
2.733739 2.728553 2.241561 -18.003843 -17.848006
1.926433 2.030953 1.552199 -19.426293 -23.572877
1.743109 1.801868 1.397482 -19.828156 -22.442590
1.228348 1.267552 0.967705 -21.218958 -23.655550
0.956638 0.966972 0.744288 -22.197592 -23.029003
0.673795 0.644774 0.515121 -23.549201 -20.108260

R^2 = 0.997788 for comparing theoretical and calculated values
R^2 = 0.998989 for comparing observed and calculated values

For concentration of B
20.000000 20.321844 21.000000 5.000000 3.337081
60.002009 59.349868 61.565267 2.605343 3.732779
74.756340 81.414332 75.903930 1.535107 -6.768345
80.237703 84.989672 80.734227 0.618817 -5.007014
79.281179 74.057905 79.154765 -0.159450 6.882263
76.520437 75.393722 76.080494 -0.574936 0.910913
72.870929 74.110641 72.198455 -0.922829 -2.580178
68.765614 65.759882 67.931453 -1.213049 3.302274
67.168962 64.023244 66.289753 -1.308952 3.540133
63.406035 60.165346 62.449351 -1.508822 3.796214
62.331984 61.780274 61.359395 -1.560337 -0.681250
58.614102 59.908642 57.603907 -1.723467 -3.847083
56.020211 56.425835 54.997598 -1.825437 -2.531177
52.500276 53.510854 51.475953 -1.951080 -3.802782

R^2 = 0.996041 for comparing theoretical and calculated values
```

```
R^2 = 0.965536 for comparing observed and calculated values

For concentration of C
10.000000 9.696440 9.523810 -4.761905 -1.780351
15.116828 14.876896 14.841560 -1.820934 -0.237522
21.956551 21.438653 21.857098 -0.452954 1.951828
30.557306 29.192957 30.594721 0.122441 4.801720
40.178899 41.381369 40.288217 0.272078 -2.641654
46.410908 47.155129 46.530105 0.256829 -1.325464
52.390810 52.564667 52.496606 0.201936 -0.129481
58.058235 58.216840 58.133197 0.129116 -0.143674
60.097299 60.080603 60.157265 0.099781 0.127598
64.667532 59.567473 64.687029 0.030149 8.594549
65.924907 64.916292 65.931701 0.010305 1.564181
70.157550 73.552919 70.116966 -0.057847 -4.671402
73.023150 74.060945 72.946693 -0.104703 -1.504507
76.825930 78.020466 76.697504 -0.167165 -1.695661

R^2 = 0.999940 for comparing theoretical and calculated values
R^2 = 0.993039 for comparing observed and calculated values
```

The results show A0 = 98.164769, k1 = 0.525000, B0 = 21.000000, k2 = 0.100533, and C0 = 9.523810. These values are close to the actual values of A0 = 100, B0 = 20, C0 = 10, k1 = 0.5, and k2 = 0.1. Moreover, the correlation coefficients for all the tabulated observed and calculated concentrations are close to 1. Thus, using ODE solvers to calculate the theoretical and observed concentrations of chemical works well.

Figures 9.1, 9.2, and 9.3 show the curves for the theoretical, observed, and calculated concentrations of chemicals A, B, and C, respectively.

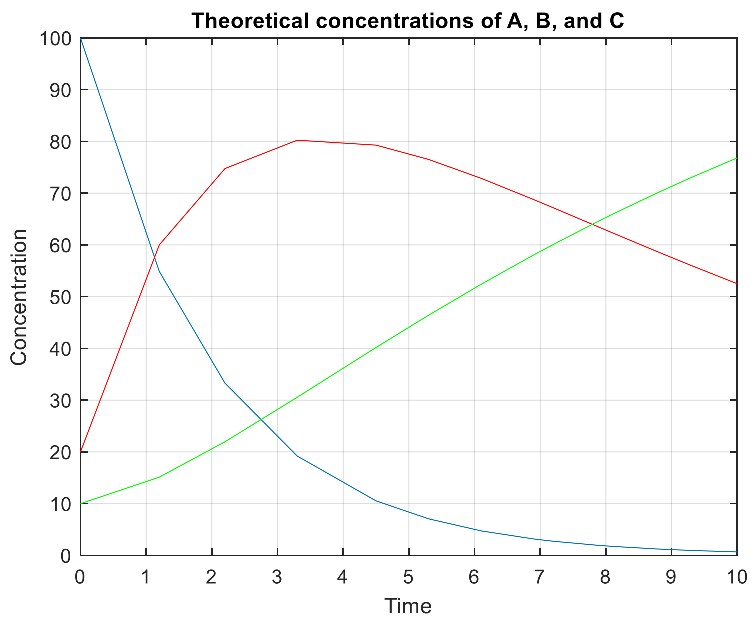

Figure 9.1. The theoretical concentrations of chemicals A, B, and C.

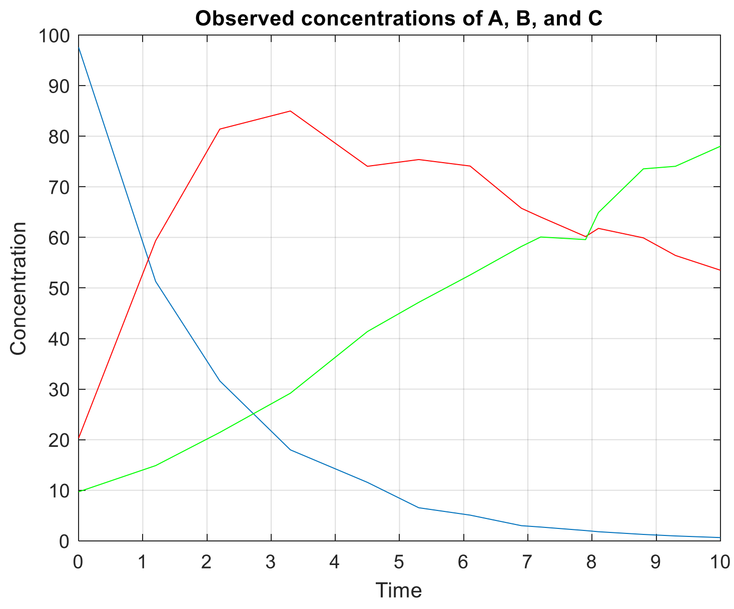

Figure 9.2. The observed concentrations of chemicals A, B, and C.

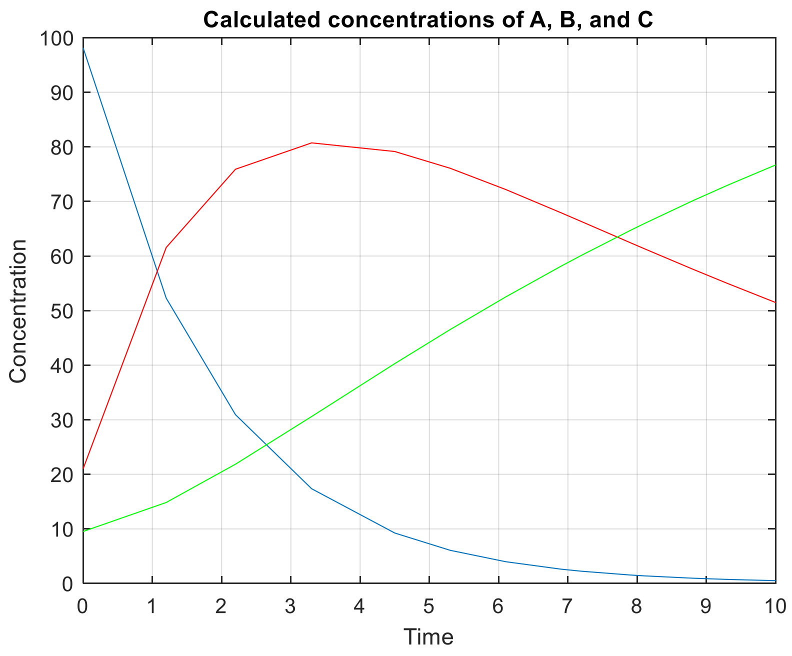

Figure 9.2. The calculated concentrations of chemicals A, B, and C.

10/Conclusion

The method described in this study can simulate the concentrations of chemicals involved in chemical reactions. Once such data is at hand, you can optimize special functions that are based on good ODE overs. Of course, the error contaminating data plays in important role in the quality of the results. Also affecting the quality of the results is the differential models for the chemical reaction, the number of optimization variables, the optimization algorithm used, and the parameters used to operate the optimization algorithm.

11/Appendix

This appendix has the listings of the MATLAB functions *findMinDigit* and *rsqr*. Here is the listing of function *findMinDigit*:

```
function d = findMinDigit(X)
%FINDMINDIGIT Summary of this function goes here
%   Detailed explanation goes here
  d = 1e+99;
  n = length(X);
  for i=1:n
    dd = mindigit(X(i));
    if dd < d
      d = dd;
    end
  end
end

function d = mindigit(x)
%MINDIGIT Summary of this function goes here
%   Detailed explanation goes here
  if x==0
    d=1;
    return;
  end
  s = num2str(x);
  n = length(s);
  if isempty(strfind(s,'.'))
    d = 1;
    i = n;
    while i>0 && strcmp(s(i),'0')
      i = i - 1;
      d = 10*d;
    end
  else
    i = strfind(s,'.');
    d = 1/10^(n-i);
  end
end
```

The function *findMinDigit* and its helper function *minDigit* find the minimum digit by converting reals into strings and attempting to locate the decimal character. If one is found, the function *minDigit* counts, n, the number of digits after the decimal and returns 10^{-n} as the answer. If the converted string has no decimal character, the function *minDigit* counts, m, the index of the first non-zero digit scanned from right to left. The function returns $10^{(m-1)}$ as the answer.

Here is the listing of function *rsqr* which calculates the coefficient of determination between two arrays of similar values:

```
function r = rsqr(y,f)
%RSQR Summary of this function goes here
%   Detailed explanation goes here
 n=length(y);
 ymean = mean(y);
 SSres = 0;
 SStot = 0;
 for i=1:n
   SSres = SSres + (y(i) - f(i))^2;
   SStot = SStot + (y(i) - ymean)^2;
 end
 r = 1 - SSres/SStot;
end
```

12/Files Included

When you download this document from my website you will also see a link to a zip file that contains many sets of MATLAB files. Each set resides in different folders. Some similar calculations (like those of Table 4,4 and Table 4.5 in [1]) are placed in the same folders

I highly recommend that you get Jorge Ancheyta's book, *Chemical Reaction Kinetics-Concepts, Methods and Case Studies.*

13/Book References

1. *Chemical Reaction Kinetics-Concepts, Methods and Case Studies*, 2017, by Jorge Ancheyta. John Wiley & Sons Ltd.
2. *Chemical Kinetics of Homogeneous Systems*, 1971, by Robert Schaal. D. Reidel Publishing Company.
3. *Introduction to Chemical Engineering Kinetics and Reactor Design*, 2nd Edition, 2014, by Charles G. Hill and Thatcher W. Root. John Wiley & Sons Ltd.

Document History

I developed this document using numbered sections to compartmentalize the reference numbers of equations and figures. Thus, I can number equations and listings using the format *sectionNumber***.***sequenceNumber* where the *sectionNumber* starts at the value of 1 for each separate section. This approach allows me to insert more equations and figure later, if need be, without having to renumber all subsequent equations and/or figures that appear after the newly inserted material in the document. Any renumbering needed occurs only in that section where new equations and figures are inserted.

Version	***Date***	***Comments***
1.0.0	November 25, 2019	Initial release.