A User's Guide for Translating Octave to MATLAB

James B. Rawlings and John W. Eaton
Department of Chemical and Biological Engineering
University of Wisconsin

John G. Ekedt
Department of Chemical Engineering
University of Texas

September 11, 2009

1 Solving Nonlinear Algebraic Equations

Code Translation

In this section we show how to translate Octave code for defining and solving a set of nonlinear algebraic equations into MATLAB code that solves the same problem. If you are interested in the code translation, start here. If you would like to first see the steady-state CSTR problem we are solving, go to the next section and return here.

Octave Code

```
------------s_shape.m------------
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
%
% multiplicity parameters
%
% units are kmol, min, kJ, K, m^3
%
%
k_m      = 0.001;
T_m      = 298;
E        = 8000;
c_Af     = 2;
C_p      = 4;
rho      = 1000;
C_ps     = rho*C_p;
T_f      = 298;
T_a      = T_f;
DeltaH_R = -3e5;
U        = 0;
```

```
function ret = st_st(x)
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
theta = x(1);
T     = x(2);
k         = k_m*exp(-E*(1/T - 1/T_m));
ret(1) = c_Af - (1+k*theta)*c_A;
ret(2) = U*theta*(T_a - T) + ...
  C_ps*(T_f - T) - k*theta*c_A*DeltaH_R;
endfunction
```

```
x0=[1; T_f];
nc_As = 200;
table = zeros(nc_As+1, 4);
table(1,:) = [0 T_f 0 0];
c_Av = linspace(0.995*c_Af,.005*c_Af,nc_As);
for i = 1: nc_As
  c_A = c_Av(i);
```

```
  [x, flag] = fsolve ('st_st', x0);
```

```
  theta = x(1);
  T     = x(2);
  conv     = (c_Af - c_A) / c_Af;
  table(i+1,:) = [theta, T, conv, flag];
  x0=x;
end

save s_shape_x.dat table

figure(1);
plot (table(:,1), table(:,3));

figure(2);
plot (table(:,1), table(:,2));
```

Matlab Code — First Alternative

```
------------s_shape.m------------
```

```
function dummy()
```

```
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
%
% multiplicity parameters
%
% units are kmol, min, kJ, K, m^3
%
%
k_m      = 0.001;
T_m      = 298;
E        = 8000;
c_Af     = 2;
C_p      = 4;
rho      = 1000;
C_ps     = rho*C_p;
T_f      = 298;
T_a      = T_f;
DeltaH_R = -3e5;
U        = 0;

x0=[1; T_f];
nc_As = 200;
table = zeros(nc_As+1, 4);
table(1,:) = [0 T_f 0 0];
c_Av = linspace(0.995*c_Af,.005*c_Af,nc_As);
for i = 1: nc_As
  c_A = c_Av(i);
```

```
  opt = optimset ('display', 'off', ...
        'largescale', 'off');
  [x, fval, flag] = fsolve (@st_st, x0, opt);
```

```
  theta = x(1);
  T     = x(2);
  conv     = (c_Af - c_A) / c_Af;
  table(i+1,:) = [theta, T, conv, flag];
  x0=x;
end
save s_shape_x.dat table

figure(1);
plot (table(:,1), table(:,3));

figure(2);
plot (table(:,1), table(:,2));
```

```
function ret = st_st(x)
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
theta = x(1);
T     = x(2);
k         = k_m*exp(-E*(1/T - 1/T_m));
ret(1) = c_Af - (1+k*theta)*c_A;
ret(2) = U*theta*(T_a - T) + ...
   C_ps*(T_f - T) - k*theta*c_A*DeltaH_R;
```

Comments.

1. The MATLAB file is declared to be a dummy function at the top. This trick allows us to define the function `st_st` inside the single file `s_shape.m`. See the next example for an alternative.

2. We move the definition of the function `st_st` to the end of the MATLAB file. Notice we strip off the `endfunction` line.

3. The call to Octave's algebraic equation solver `fsolve` is different than the same function in MATLAB.

   (a) We set two parameters in `optimset` before calling the MATLAB version of `fsolve`. We turn off the `'largescale'` feature in MATLAB because it does not work reliably. We turn off `'display'` to avoid excessive output to the screen.

   (b) Notice that there is an extra argument in the output from `fsolve` in the MATLAB version. The value of the algebraic equations at the solution is returned in `fval`. We pass in the extra argument `opt`, which contains the values of the two parameters set in `optimset`.

   (c) Notice that the quoted function name `'st_st'` has been changed to `@st_st` in the MATLAB version.

The following table highlights these changes.

Octave	MATLAB— I
(none)	`function dummy ()`
(none)	`opt = optimset ('display', 'off', ...` `'largescale', 'off')`
`[x, flag] = fsolve('st_st', x0)`	`[x, fval, flag] = fsolve(@st_st, x0, opt)`
`endfunction`	(none)

Octave Code

```
------------s_shape.m------------
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
%
% multiplicity parameters
%
% units are kmol, min, kJ, K, m^3
%
%
k_m      = 0.001;
T_m      = 298;
E        = 8000;
c_Af     = 2;
C_p      = 4;
rho      = 1000;
C_ps     = rho*C_p;
T_f      = 298;
T_a      = T_f;
DeltaH_R = -3e5;
U        = 0;
```

```
function ret = st_st(x)
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
theta = x(1);
T     = x(2);
k         = k_m*exp(-E*(1/T - 1/T_m));
ret(1) = c_Af - (1+k*theta)*c_A;
ret(2) = U*theta*(T_a - T) + ...
   C_ps*(T_f - T) - k*theta*c_A*DeltaH_R;
endfunction
```

```
x0=[1; T_f];
nc_As = 200;
table = zeros(nc_As+1, 4);
table(1,:) = [0 T_f 0 0];
c_Av = linspace(0.995*c_Af,.005*c_Af,nc_As);
for i = 1: nc_As
  c_A = c_Av(i);
```

```
  [x, flag] = fsolve ('st_st', x0);
```

```
theta = x(1);
  T     = x(2);
  conv     = (c_Af - c_A) / c_Af;
  table(i+1,:) = [theta, T, conv, flag];
  x0=x;
end

save s_shape_x.dat table

figure(1);
plot (table(:,1), table(:,3));

figure(2);
plot (table(:,1), table(:,2));
```

Matlab Code — Second Alternative

```
------------s_shape.m------------
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
%
% multiplicity parameters
%
% units are kmol, min, kJ, K, m^3
%
%
k_m      = 0.001;
T_m      = 298;
E        = 8000;
c_Af     = 2;
C_p      = 4;
rho      = 1000;
C_ps     = rho*C_p;
T_f      = 298;
T_a      = T_f;
DeltaH_R = -3e5;
U        = 0;

x0=[1; T_f];
nc_As = 200;
table = zeros(nc_As+1, 4);
table(1,:) = [0 T_f 0 0];
c_Av = linspace(0.995*c_Af,.005*c_Af,nc_As);
for i = 1: nc_As
  c_A = c_Av(i);
```

```
  opt = optimset ('display', 'off', ...
    'largescale', 'off');
  [x, fval, flag] = fsolve (@st_st, x0, opt);
```

```
  theta = x(1);
  T     = x(2);
  conv     = (c_Af - c_A) / c_Af;
  table(i+1,:) = [theta, T, conv, flag];
  x0=x;
end

save s_shape_x.dat table

figure(1);
plot (table(:,1), table(:,3));

figure(2);
plot (table(:,1), table(:,2));
```

```
------------st_st.m------------
function ret = st_st(x)
global k_m T_m E c_Af c_A C_ps T_f ...
       DeltaH_R U T_a
theta = x(1);
T     = x(2);
k         = k_m*exp(-E*(1/T - 1/T_m));
ret(1) = c_Af - (1+k*theta)*c_A;
ret(2) = U*theta*(T_a - T) + ...
   C_ps*(T_f - T) - k*theta*c_A*DeltaH_R;
```

The following table highlights these changes.

Octave	MATLAB—II
(none)	`opt = optimset ('display', 'off', ... 'largescale', 'off')`
`[x, flag] = fsolve('st_st', x0)`	`[x, fval, flag] = fsolve(@st_st, x0, opt)`
`endfunction`	(none)
(none)	additional file `st_st.m` created

Comparison of the two alternatives.

1. The big advantage of MATLAB—I is that a single file contains all the code that solves the problem. The big disadvantage is that debugging the code at the command line does not work because the file `s_shape.m` is a function, and the variables defined in a function are not available at the MATLAB command line. One can work around this issue by copying the global command and pasting it into the MATLAB command window. Issuing this command at the command line makes the global variables available for debugging. Note one can add more variables to the global statement to gain access to more of the variables used in the function.

2. The big advantage of MATLAB—II is that debugging at the command line does work. The values of all variables in `s_shape.m` can be checked from the command line.

   The big disadvantage is that one has to create as many new files with distinct file names as there are internal functions like `st_st.m` defined in `s_shape.m`. The creation of many new file names in a large project can be a significant organizational nuisance. It can become difficult even to come up with distinct, meaningful file names in a large project.

Multiple Steady States in a Nonisothermal CSTR

The following example is taken from Section 6.3.1 of Rawlings and Ekerdt (2004). Consider an adiabatic, constant-volume CSTR with the following elementary reaction taking place in the liquid phase

$$\mathrm{A} \xrightarrow{k} \mathrm{B}$$

We wish to compute the steady-state reactor conversion and temperature. The data and parameters are listed in the following table.

Parameter	Value	Units
T_f	298	K
T_m	298	K
$\hat{C}_P$	4.0	kJ/kg K
c_{Af}	2.0	kmol/m^3
k_m	0.001	min^{-1}
E	8.0×10^3	K
ρ	10^3	kg/m^3
ΔH_R	-3.0×10^5	kJ/kmol
U^o	0	

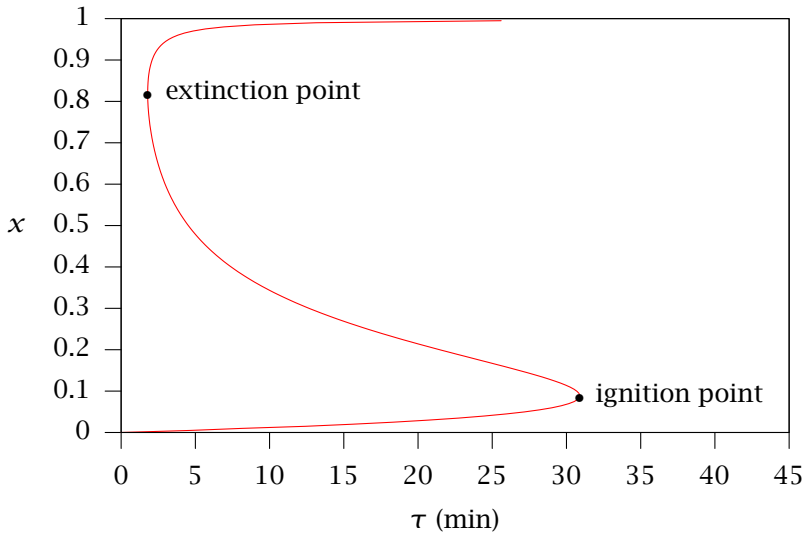

Figure 1: Steady-state conversion versus residence time; ignition and extinction points.

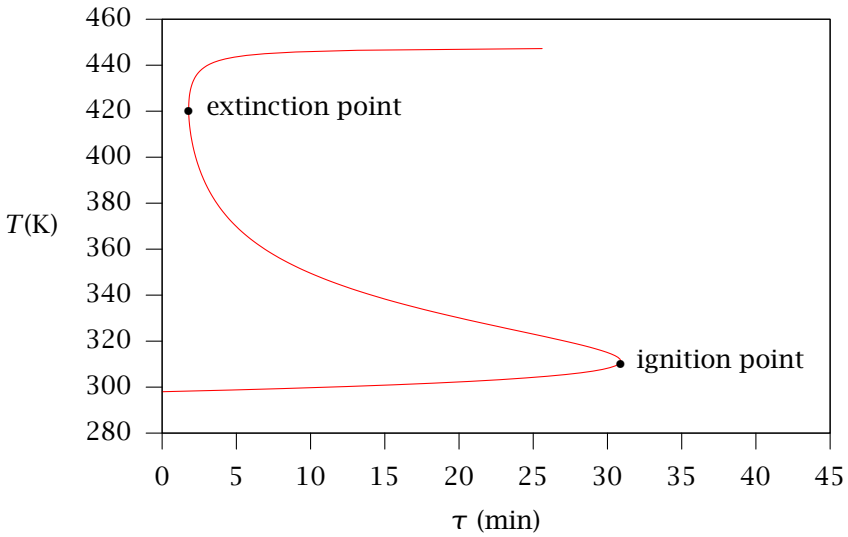

Figure 2: Steady-state temperature versus residence time; ignition and extinction points.

The material balance for component A is

$$\frac{d(c_A V_R)}{dt} = Q_f c_{Af} - Q c_A + R_A V_R$$

The production rate is given by

$$R_A = -k(T) c_A$$

For the steady-state reactor with constant-density, liquid-phase streams, the material balance simplifies to

$$\boxed{0 = c_{Af} - (1 + k\tau) c_A} \tag{1}$$

Equation 1 is one nonlinear algebraic equation in two unknowns: c_A and T. The temperature appears in the rate-constant function,

$$k(T) = k_m e^{-E(1/T - 1/T_m)}$$

A second equation is provided by the energy balance, and, in this example, we assume the heat capacity of the mixture is constant and independent of composition and temperature. With these assumptions, the steady-state energy balance reduces to

$$0 = -k c_A \Delta H_R V_R + Q_f \rho_f \hat{C}_P (T_f - T) + U^o A (T_a - T)$$

Dividing through by V_R and noting $U^o = 0$ for the adiabatic reactor gives

$$\boxed{0 = -k c_A \Delta H_R + \frac{C_{Ps}}{\tau}(T_f - T)} \tag{2}$$

in which $C_{Ps} = \rho_f \hat{C}_P$, a heat capacity per volume. The solution of Equations 1 and 2 for c_A and T provide the steady-state CSTR solution. The solution for this model given the parameters in the table is shown in Figures 1 and 2.

2 Solving Differential Equations

Code Translation

In this section we show how to translate Octave code for defining and solving a set of ordinary differential equations (ODEs) into MATLAB code that solves the same problem. If you are interested in the code translation, start here. If you would like to first see the chemical system that the differential equations describe, go to the next section and return here.

Octave Code

```
------------ester.m------------
global  tau k caf cbf qaq qbq

tau = 10; k = 0.1;
caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha = 1;
qaq = alpha/(1.+alpha);
qbq = 1./(1.+alpha);

function rhs = cstr(x,t)
  global  tau k caf cbf qaq qbq
  ca = x(1);   cb = x(2);
  rhs = [ (qaq*caf - ca)/tau - k*ca*cb;
          (qbq*cbf - cb)/tau - k*ca*cb;
          k*ca*cb;
          qaq/tau*caf;
          qbq/tau*cbf; ];
endfunction

tfin = 5*tau; nts  = 100;
tpts = linspace(0, tfin, nts)';

x0 = [ca0;cb0;0;ca0;cb0];

x = lsode('cstr', x0, tpts);

xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);

figure(1);
plot(tpts, [x(:,1),x(:,2)])

figure(2);
plot(tpts, [xa, xb])

table = [tpts, x, xa, xb];
save ester.dat table
```

Matlab Code — First Alternative

```
------------ester.m------------
function dummy()

global  tau k caf cbf qaq qbq

tau = 10; k = 0.1;
caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha = 1;
qaq = alpha/(1.+alpha);
qbq = 1./(1.+alpha);

tfin = 5*tau; nts  = 100;
tpts = linspace(0, tfin, nts)';

x0 = [ca0;cb0;0;ca0;cb0];

[tpts, x] = ode15s(@cstr, tpts, x0);

xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);

figure(1);
plot(tpts, [x(:,1),x(:,2)])

figure(2);
plot(tpts, [xa, xb])

table = [tpts, x, xa, xb];
save ester.dat table

function rhs = cstr(t, x)
  global  tau k caf cbf qaq qbq
  ca = x(1);   cb = x(2);
  rhs = [ (qaq*caf - ca)/tau - k*ca*cb;
          (qbq*cbf - cb)/tau - k*ca*cb;
          k*ca*cb;
          qaq/tau*caf;
          qbq/tau*cbf; ];
```

Comments.

1. The MATLAB file is declared to be a dummy function at the top. This trick allows us to define the function `cstr` inside the single file `ester.m`. See the next example for an alternative.

2. We move the definition of the function `cstr` to the end of the MATLAB file

   (a) Notice that the order of the arguments state (`x`) and time (`t`) have been switched in the input argument to the function `cstr`

   (b) Notice we strip off the `endfunction` line.

3. The call to Octave's ODE solver `lsode` is changed to a call to MATLAB's ODE solver `ode15s`. Other ODE solvers are available but chemical kinetics problems are often *stiff* differential equations so `ode15s` is a good, general-purpose choice.

   (a) Notice that the order of the initial condition (`x0`) and output times (`tpts`) has been switched in the input argument to the ODE solver call.

   (b) Notice that there is an extra argument in the output from the ODE solver in the MATLAB version. The times at which output is provided (`tpts`) is returned.

(c) Notice that the quoted function name `'cstr'` has been changed to `@cstr` in the MATLAB version.

The following table highlights these changes.

Octave	MATLAB— I
(none)	`function dummy ()`
`x = lsode('cstr', x0, tpts)`	`[tpts, x] = ode15s(@cstr, tpts, x0)`
`function rhs = cstr(x, t)`	`function rhs = cstr(t, x)`
`endfunction`	(none)

Octave Code

```
------------ester.m------------
global  tau k caf cbf qaq qbq

tau = 10; k = 0.1;
caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha = 1;
qaq = alpha/(1.+alpha);
qbq = 1./(1.+alpha);
```

```
function rhs = cstr(x,t)
  global  tau k caf cbf qaq qbq
  ca = x(1);   cb = x(2);
  rhs = [ (qaq*caf - ca)/tau - k*ca*cb;
          (qbq*cbf - cb)/tau - k*ca*cb;
          k*ca*cb;
          qaq/tau*caf;
          qbq/tau*cbf; ];
endfunction
```

```
tfin = 5*tau; nts  = 100;
tpts = linspace(0, tfin, nts)';

x0 = [ca0;cb0;0;ca0;cb0];
```

```
x = lsode('cstr', x0, tpts);
```

```
xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);

figure(1);
plot(tpts, [x(:,1),x(:,2)])

figure(2);
plot(tpts, [xa, xb])

table = [tpts, x, xa, xb];
save ester.dat table
```

Matlab Code — Second Alternative

```
------------ester.m------------
global  tau k caf cbf qaq qbq

tau = 10; k = 0.1;
caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha = 1;
qaq = alpha/(1.+alpha);
qbq = 1./(1.+alpha);

tfin = 5*tau; nts  = 100;
tpts = linspace(0, tfin, nts)';

x0 = [ca0;cb0;0;ca0;cb0];
```

```
[tpts, x] = ode15s(@cstr, tpts, x0);
```

```
xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);

figure(1);
plot(tpts, [x(:,1),x(:,2)])

figure(2);
plot(tpts, [xa, xb])

table = [tpts, x, xa, xb];
save ester.dat table
```

```
------------cstr.m------------
function rhs = cstr(t,x)
  global  tau k caf cbf qaq qbq
  ca = x(1);   cb = x(2);
  rhs = [ (qaq*caf - ca)/tau - k*ca*cb;
          (qbq*cbf - cb)/tau - k*ca*cb;
          k*ca*cb;
          qaq/tau*caf;
          qbq/tau*cbf; ];
```

The following table highlights these changes.

Octave	MATLAB— II
`x = lsode('cstr', x0, tpts)`	`[tpts, x] = ode15s(@cstr, tpts, x0)`
`function rhs = cstr(x, t)`	`function rhs = cstr(t, x)`
`endfunction`	(none)
(none)	additional file `cstr.m` created

Comparison of the two alternatives.

1. The big advantage of MATLAB—I is that a single file contains all the code that solves the problem. The big disadvantage is that debugging the code at the command line does not work because the file `ester.m` is a function, and the variables defined in a function are not available at the MATLAB command line. One can work around this issue by copying the global command and pasting it into the MATLAB command window. Issuing this command at the command line makes the global variables available for debugging. Note one can add more variables to the global statement to gain access to more of the variables used in the function.

2. The big advantage of MATLAB—II is that debugging at the command line does work. The values of all variables in `ester.m` can be checked from the command line.

   The big disadvantage is that one has to create as many new files with distinct file names as there are internal functions like `cstr.m` defined in `ester.m`. The creation of many new file names in a large project can be a significant organizational nuisance. It can become difficult even to come up with distinct, meaningful file names in a large project.

Esterification Reaction

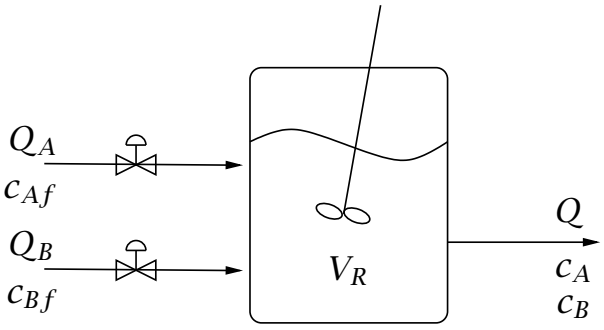

Figure 3: Organic acid and base streams fed into a CSTR.

This example is taken from Exercise 4.24 on the textbook website `www.che.wisc.edu/~jbraw/chemreacfun`. Consider the liquid-phase organic esterification reaction taking place in a CSTR depicted in Figure 3. Two streams, an acid stream containing no base, and a base stream containing no acid, are fed into the CSTR. The esterification reaction and its rate are given by

$$\mathrm{A} + \mathrm{B} \longrightarrow \text{ ester} + \mathrm{H_2O} \qquad r = kc_Ac_B$$

in which A is the organic acid and B is the organic base. The acid and base are dissolved in an organic solvent and the acid and base feed streams have feed concentrations c_{Af} and c_{Bf}, respectively. The density of the fluid is independent of concentration over the concentration range of interest here. The reactor's volume is constant during the entire operation. Because of the constant density and reactor volume, the flowrates are related by

$$Q = Q_A + Q_B$$

The material balances for A and B are

$$\frac{dc_A}{dt} = \frac{1}{\tau}\left[\frac{Q_A}{Q}c_{Af} - c_A\right] - kc_Ac_B \tag{3}$$

$$\frac{dc_B}{dt} = \frac{1}{\tau}\left[\frac{Q_B}{Q}c_{Bf} - c_B\right] - kc_Ac_B \tag{4}$$

We assign the first two elements of the state vector to be the reactor's A and B concentrations

$$x_1 = c_A \qquad x_2 = c_B$$

Next we extend the definition of A and B conversion to handle the transient case. A sensible definition is

$$x_A = \frac{\text{total moles A consumed by reaction}}{\text{total moles A fed into system}}$$

$$x_A = \frac{\int_0^t kc_A(t)c_B(t)V_R(t)dt}{\int_0^t Q_A(t)c_{Af}(t)dt}$$

So we require differential equations to track the amount of A (and B) consumed by reaction and the amount of A and B fed into the system. Therefore define new elements of the state vector, x_3, x_4, and x_5 with evolution equations

$$\frac{dx_3}{dt} = kc_Ac_B \qquad \frac{dx_4}{dt} = \frac{Q_A}{Q\tau}c_{Af} \qquad \frac{dx_5}{dt} = \frac{Q_B}{Q\tau}c_{Bf}$$

The appropriate initial conditions are

$$x_3(0) = 0 \qquad x_4(0) = c_{A0} \qquad x_5(0) = c_{B0}$$

Then we can compute the conversions of A and B from

$$x_A = \frac{x_3}{x_4} \qquad x_B = \frac{x_3}{x_5}$$

Figure 4 shows the full transient solution starting with the reactor at the feed stream concentrations of A and B. Figure 5 shows the A and B conversion versus time. These figures were computed by running Octave on the file `ester.m`.

3 Boundary-Value Problems with Shooting Method

Code Translation

In this section we show how to translate Octave code for using a shooting method for solving a boundary-value problem (BVP) into MATLAB code that solves the same problem. If you are interested in the code translation, start here. If you would like to first see the chemical system that the differential equations describe, go to the next section and return here.

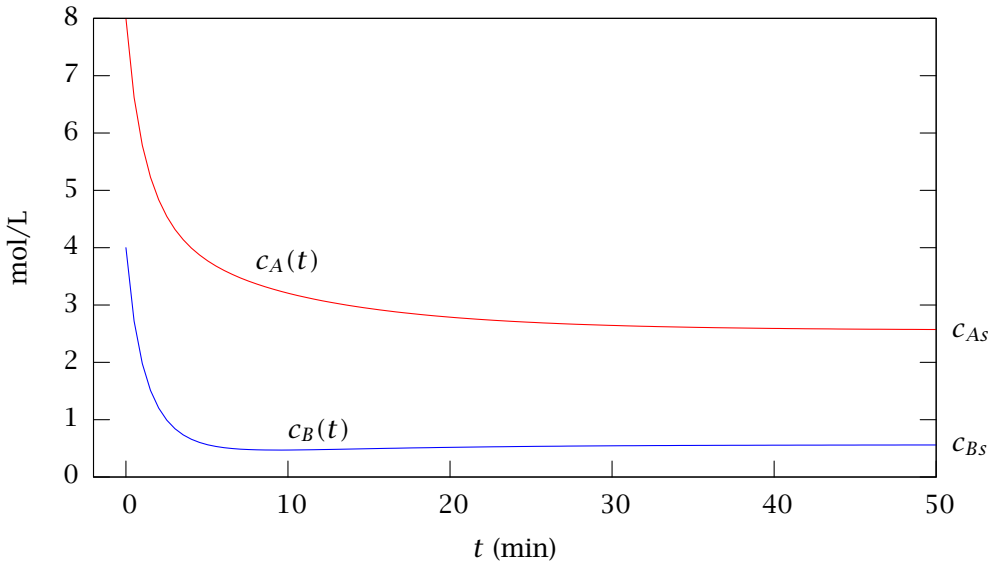

Figure 4: Transient A and B concentrations versus time.

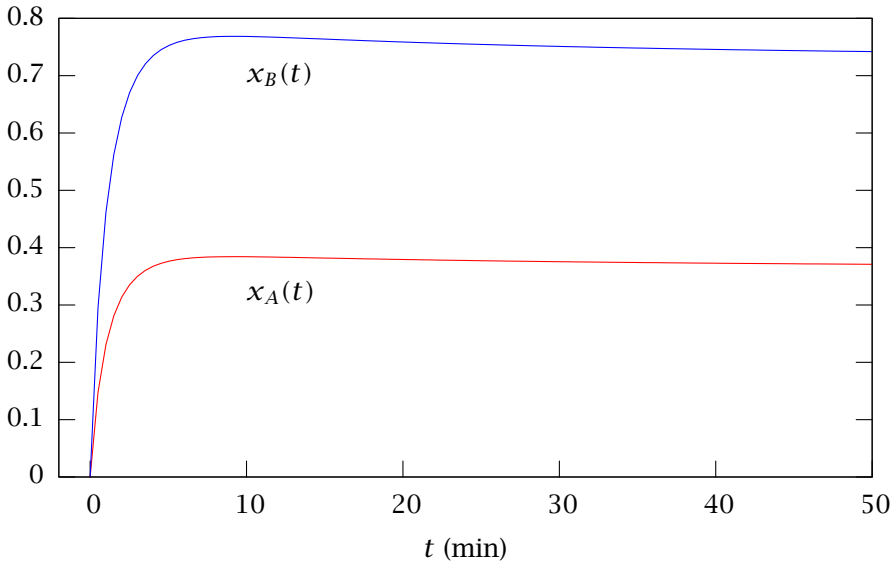

Figure 5: Conversion of A and B versus time.

Octave Code

```
---------ammonia_profile.m-----------
global K0 Tstd delH0 P R Rg Nn0 Nh0 Nnh0  ...
       A k0 E delH0 K0 Tstd deltaH ...
       UA Ta0 len  xnh0 xh0 xn0 Q0 tau rxrate
P  = 300;  R  = 82.05e-6;  v0 = 0.16;
A  = 1; UA = 1/2;
delG0 = -4250; delH0 = -12000;
Tstd  = 298; Rg = 1.987;
K0 = exp(-delG0/(Rg*Tstd));
deltaH = - 2.342; tau    = 1500;
Ta0 = 323; k0 = 7.794e11; E = 20000;
len = linspace(0,12,250)';
%% Initial flows
Q0   = v0*A; xn0  = 0.985*0.25;
xh0  = 0.985*0.75; xnh0 = 0.015;
```

```
function xdot=ivpjbr(x, t)
  global P R Rg Nn0 Nh0 Nnh0 A k0 E ...
  delH0 Tstd K0 deltaH UA Q0 tau rxrate
  Nnh = x(1);
  T   = x(2);
  Ta  = x(3);
  RT  = R*T;
  Nn=Nn0 - 1/2*(Nnh - Nnh0);
  Nh=Nh0 - 3/2*(Nnh - Nnh0);
  Nt = Nn + Nh + Nnh;
  pn = (Nn/Nt)*P;
  ph = (Nh/Nt)*P;
  pnh = (Nnh/Nt)*P;
  qadd = (Ta - T)*UA;
  k    = k0*exp(-E/T);
  K = K0*exp(-delH0/Rg*(1/T - 1/Tstd));
  rxrate = k/RT*(K^2*pn*ph^(1.5)/pnh ...
           - pnh/ph^(1.5));
  xdot   = [A * 2* rxrate; ...
            qadd - rxrate*deltaH; ...
            qadd];
endfunction
```

```
function error = feed_cond (T0)
  global Ta0 len Nnh0 Nn0 Nh0 P R ...
         xnh0 xh0 xn0 Q0
  Nnh0 = xnh0*Q0*P/(R*T0);
  Nh0 = xh0*Q0*P/(R*T0);
  Nn0 = xn0*Q0*P/(R*T0);
  x0 = [Nnh0;T0;T0];
  x  = lsode('ivpjbr',x0,len);
  Taend = x((length(x)),3);
  error = Ta0 - Taend;
endfunction
```

```
%% Solve for the upper steady state
%% using T0guess=800 K;
Tg = 800;
[T0, info] = fsolve('feed_cond',Tg);
x0=[Nnh0; T0; T0];
x=lsode('ivpjbr',x0,len);
NT = Nn0 + Nh0 + 2*Nnh0 - x(:,1);
xnh = x(:,1) ./ NT;
table1 = [len x xnh];
save ammonia_profile.dat, table1;

figure(1);
plot(table1(:,1),table1(:,2));
figure(2);
plot(table1(:,1),table1(:,4));
```

Matlab Code — Second Alternative

```
---------ammonia_profile.m-----------
global K0 Tstd delH0 P R Rg Nn0 Nh0 Nnh0  ...
       A k0 E delH0 K0 Tstd deltaH ...
       UA Ta0 len  xnh0 xh0 xn0 Q0 tau rxrate
P  = 300;  R  = 82.05e-6;  v0 = 0.16;
A  = 1; UA = 1/2;
delG0 = -4250; delH0 = -12000;
Tstd  = 298; Rg = 1.987;
K0 = exp(-delG0/(Rg*Tstd));
deltaH = - 2.342; tau    = 1500;
Ta0 = 323; k0 = 7.794e11; E = 20000;
len = linspace(0,12,250)';
%% Initial flows
Q0   = v0*A; xn0  = 0.985*0.25;
xh0  = 0.985*0.75; xnh0 = 0.015;
%% Solve for the upper steady state
%% using T0guess=800 K;
Tg = 800;
```

```
opt = optimset ('display', 'off', ...
      'largescale', 'off');
[T0, fval, info] = fsolve(@feed_cond, ...
                   Tg, opt);
```

```
x0=[Nnh0; T0; T0];
[len, x] = ode15s(@ivpjbr, len, x0);
NT = Nn0 + Nh0 + 2*Nnh0 - x(:,1);
xnh = x(:,1) ./ NT;
table1 = [len x xnh];
save ammonia_profile.dat, table1;
figure(1);
plot(table1(:,1),table1(:,2));
figure(2);
plot(table1(:,1),table1(:,4));
```

```
---------ivpjbr.m-----------
function xdot=ivpjbr(t, x)
  global P R Rg Nn0 Nh0 Nnh0 A k0 E ...
  delH0 Tstd K0 deltaH UA Q0 tau rxrate
  Nnh = x(1);
  T   = x(2);
  Ta  = x(3);
  RT  = R*T;
  Nn=Nn0 - 1/2*(Nnh - Nnh0);
  Nh=Nh0 - 3/2*(Nnh - Nnh0);
  Nt = Nn + Nh + Nnh;
  pn = (Nn/Nt)*P;
  ph = (Nh/Nt)*P;
  pnh = (Nnh/Nt)*P;
  qadd = (Ta - T)*UA;
  k    = k0*exp(-E/T);
  K = K0*exp(-delH0/Rg*(1/T - 1/Tstd));
  rxrate = k/RT*(K^2*pn*ph^(1.5)/pnh ...
           - pnh/ph^(1.5));
  xdot   = [A * 2* rxrate; ...
            qadd - rxrate*deltaH; ...
            qadd];
```

```
---------feed_cond.m-----------
function error = feed_cond (T0)
  global Ta0 len Nnh0 Nn0 Nh0 P R ...
         xnh0 xh0 xn0 Q0
  Nnh0 = xnh0*Q0*P/(R*T0);
  Nh0 = xh0*Q0*P/(R*T0);
  Nn0 = xn0*Q0*P/(R*T0);
  x0 = [Nnh0;T0;T0];
  [len, x]  = ode15s(@ivpjbr, len, x0);
  Taend = x((length(x)),3);
  error = Ta0 - Taend;
```

Comments.

1. The call to Octave's ODE solver `lsode` is changed to a call to MATLAB's ODE solver `ode15s`.
   (a) Notice that the order of the initial condition (`x0`) and output times (`tpts`) has been switched in the input argument to the ODE solver call.
   (b) Notice that there is an extra argument in the output from the ODE solver in the MATLAB version. The lengths at which output is provided (`len`) is returned.
   (c) Notice that the quoted function name `'ivpjbr'` has been changed to `@ivpjbr` in the MATLAB version.
2. The call to Octave's algebraic equation solver `fsolve` is different than the same function in MATLAB.
   (a) We set two parameters in `optimset` before calling the MATLAB version of `fsolve`. We turn off the `'largescale'` feature in MATLAB because it does not work reliably. We turn off `'display'` to avoid excessive output to the screen.
   (b) Notice that there is an extra argument in the output from `fsolve` in the MATLAB version. The value of the algebraic equations at the solution is returned in `fval`. We pass in the extra argument `opt`, which contains the values of the two parameters set in `optimset`.
   (c) Notice that the quoted function name `'feed_cond'` has been changed to `@feed_cond` in the MATLAB version.

The following table highlights these changes.

Octave	MATLAB— II
`x = lsode('ivpjbr', x0, len)`	`[len, x] = ode15s(@ivpjbr, len, x0)`
`function xdot = ivpjbr(x, t)`	`function xdot = ivpjbr(t, x)`
`endfunction`	(none)
(none)	`opt = optimset ('display', 'off', ...` `'largescale', 'off')`
`[x, flag] = fsolve('feed_cond', x0)`	`[x, fval, flag] = fsolve(@feed_cond, ...` `x0, opt)`
(none)	additional file `ivpjbr.m` created
(none)	additional file `feed_cond.m` created

Ammonia Synthesis

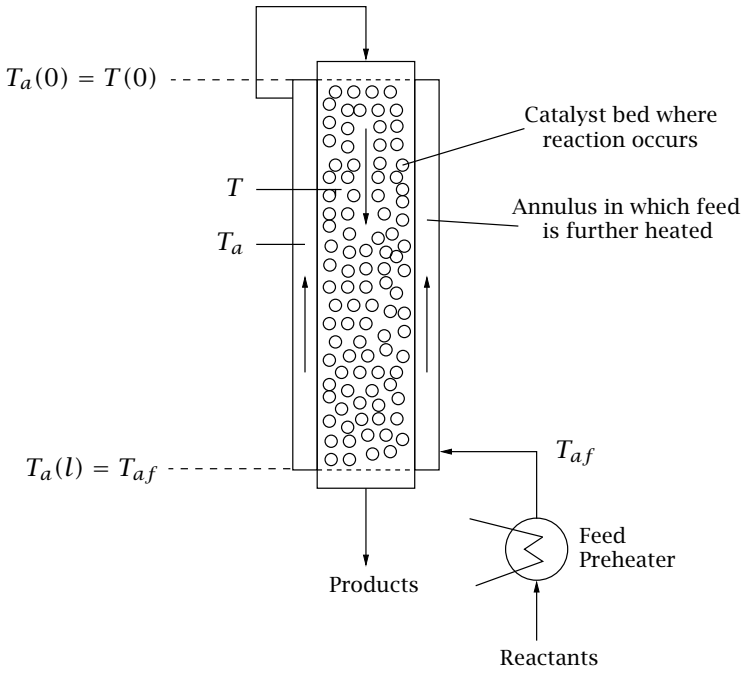

Figure 6: Autothermal plug-flow reactor; the heat released by the exothermic reaction is used to preheat the feed.

The following example is taken from Example 6.7 in Rawlings and Ekerdt (2004). Calculate the steady-state conversion for the synthesis of ammonia using the autothermal process shown in Figure 6 (van Heerden, 1953). A rate expression for the reaction

$$\mathrm{N_2} + 3\mathrm{H_2} \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} 2\mathrm{NH_3}$$

over an iron catalyst at 300 atm pressure is suggested by Temkin and Pyzhev (1940)

$$r = k_{-1}/RT \left[K^2(T) \frac{P_N P_H^{3/2}}{P_A} - \frac{P_A}{P_H^{3/2}} \right]$$

in which P_N, P_H, P_A are the partial pressures (divided by 1.0 atm) of nitrogen, hydrogen and ammonia, respectively, and K is the equilibrium constant for the reaction forming one mole of ammonia. For illustration, we assume the thermochemical properties are constant and the gases form an ideal-gas mixture.

The mass and energy balances provide the following differential equations for the molar flow of ammonia, N_A; temperature of the reactor, T; and temperature of

Parameter	Value	Units
P	300	atm
Q_f	0.16	m^3/s
x_{Af}	0.015	
x_{Nf}	0.985(1/4)	
x_{Hf}	0.985(3/4)	
A_c	1	m^2
l	12	m
T_{af}	323	K
$\gamma = \frac{2\pi RU^o}{Q\rho\hat{C}_P}$	0.5	1/m
$\beta = \frac{\Delta H_R A_c}{Q\rho\hat{C}_P}$	-2.342	m^2 s K/mol
ΔG°	-4.25×10^3	cal/mol
ΔH°	-1.2×10^4	cal/mol
k_{-10}	7.794×10^{11}	atm/s
E_{-1}/R	2×10^4	K

Table 1: Parameter values for Example 6.7; heat of reaction and mixture heat capacity assumed constant.

the heat exchanger, T_a;

$$\begin{aligned} \frac{dN_A}{dz} &= 2A_c r & N_A(0) &= N_{Af} \\ \frac{dT}{dz} &= -\beta r + \gamma(T_a - T) & T(0) &= T_a(0) \\ \frac{dT_a}{dz} &= \gamma(T_a - T) & T_a(l) &= T_{af} \end{aligned} \tag{5}$$

in which

$$\beta = \frac{\Delta H_R A_c}{Q\rho\hat{C}_P} \qquad \gamma = \frac{2\pi RU^o}{Q\rho\hat{C}_P}$$

Equation 5 is a boundary-value problem, rather than an initial-value problem, because T_a is specified at the exit of the reactor. A simple solution strategy is to guess the reactor inlet temperature, solve the model to the exit of the reactor, and then compare the computed feed preheat temperature to the specified value, T_{af}. This strategy is known as a shooting method. We guess the missing values required to produce an initial-value problem. We solve the initial-value problem, and then iterate on the guessed values until we match the specified boundary conditions.

Figures 7 and 8 show the results for the parameter values listed in Table 1, which are based on those used by van Heerden (1953). The feed consists of 1.5% ammonia and 98.5% stoichiometric mixture of nitrogen and hydrogen. Three steady-state solutions exist for these parameter values. The profile in the reactor for the upper steady states is shown in Figures 7 and 8. It is important to operate at the upper steady state so that a reasonably large production of ammonia is achieved.

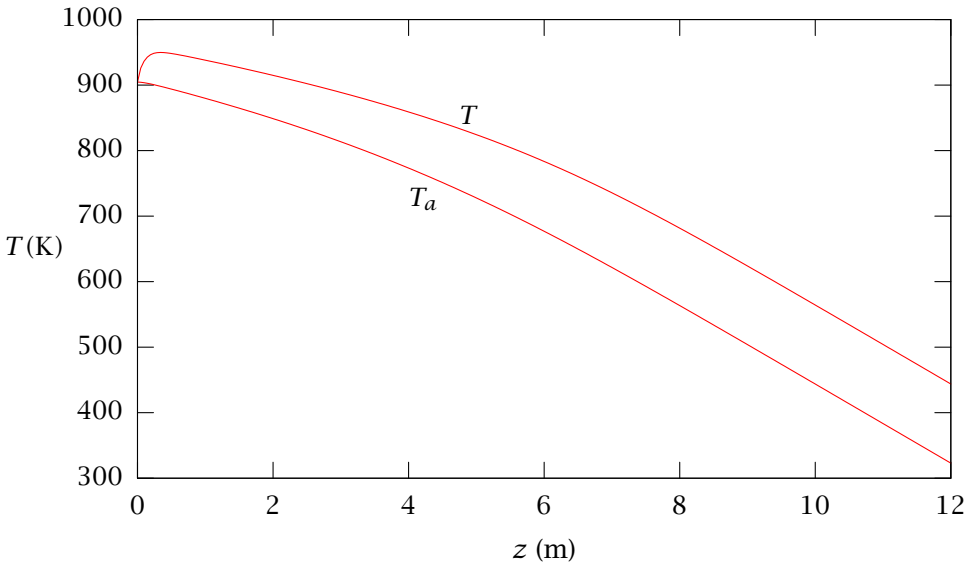

Figure 7: Reactor and coolant temperature profiles versus reactor length; upper steady state.

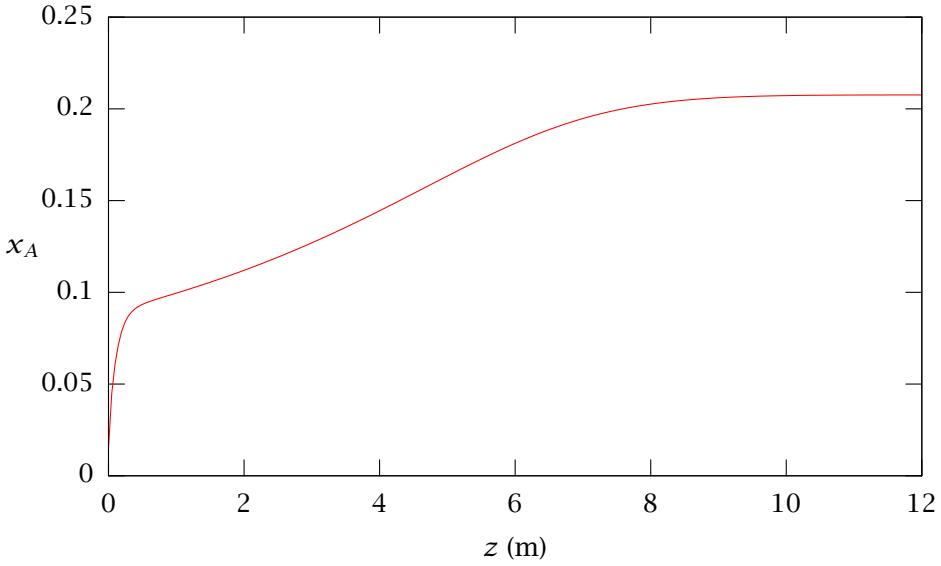

Figure 8: Ammonia mole fraction versus reactor length; upper steady state.

4 Parameter Estimation for Differential-Equation Models

Code Translation

In this section we show how to translate Octave code for estimating the parameters appearing in a set of ODEs into MATLAB code that solves the same problem. If you are interested in the code translation, start here. If you would like to first see the differential-equation model and the parameter-estimation problem under discussion, go to the next section and return here.

Octave Code

```
------------nthorder.m------------
global ca ymeas tout theta k n

kac   = 0.5;
ca0ac = 2;
nac   = 2.5;
thetaac = [kac; ca0ac; nac];
tfinal = 5;
nts    = 100;
tout = linspace(0,tfinal,nts)';
```

```
function xdot = massbal(x, t)
  global k n
  ca      = x;
  if (ca <= 0.0)
    xdot = 0;
  else
    xdot = -k*ca^n;
  endif
endfunction
```

```
%% solve model to create measurement data
x0 = ca0ac;
k  = kac;
n  = nac;
theta = thetaac;
```

```
caac = lsode('massbal', x0, tout);
```

```
add measurement noise
measvar = 1e-2;
measstddev = sqrt(measvar);
%% set seed for 'reproducible' random numbers
randn('seed',0);
noise = measstddev*randn(length(tout),1);
ymeas = caac + noise;
```

```
function phi = model(theta)
  global ca ymeas tout k n
  k   = theta(1);
  ca0 = theta(2);
  n   = theta(3);
  x0  = ca0;
  %% solve model
    [x] = lsode('massbal', x0, tout);
calculate fit to data
  ca = x;
  resid = ymeas - ca;
  phi = resid'*resid;
endfunction
```

```
%% optimize parameter fit
small  = 1e-3;
lb     = [small; small; small];
large  = 5;
ub     = [large;large;large];
theta0 = [1; 1; 1];
```

```
[theta, obj, info] = npsol ...
   (theta0, 'model', lb, ub);
```

```
phi = model(theta);
caest = ca;
table  = [tout ymeas caest caac];
save nthorder.dat table;
plot (table(:,1), table(:,2), ...
      '+', table(:,1), table(:,3:4));
```

Matlab Code — Second Alternative

```
------------nthorder.m------------
global ca ymeas tout theta k n

kac   = 0.5;
ca0ac = 2;
nac   = 2.5;
thetaac = [kac; ca0ac; nac];
tfinal = 5;
nts    = 100;
tout = linspace(0,tfinal,nts)';
%% solve model to create measurement data
x0 = ca0ac;
k  = kac;
n  = nac;
theta = thetaac;
```

```
[tdum, caac] = ode15s(@massbal, tout, x0);
```

```
add measurement noise
measvar = 1e-2;
measstddev = sqrt(measvar);
%% set seed for 'reproducible' random numbers
randn('seed',0);
noise = measstddev*randn(length(tout),1);
ymeas = caac + noise;

%% optimize parameter fit
small  = 1e-3;
lb     = [small; small; small];
large  = 5;
ub     = [large;large;large];

theta0 = [1; 1; 1];
```

```
[theta, obj, info] = fmincon (@model,...
  theta0, [], [], [], [], lb, ub);
```

```
phi = model(theta);
caest = ca;
table  = [tout ymeas caest caac];
save nthorder.dat table;
plot (table(:,1), table(:,2), ...
      '+', table(:,1), table(:,3:4));
```

```
------------massbal.m------------
function xdot = massbal(t, x)
  global k n
  ca      = x;
  if (ca <= 0.0)
    xdot = 0;
  else
    xdot = -k*ca^n;
  end
```

```
------------model.m------------
function phi = model(theta)
  global ca ymeas tout k n
  k   = theta(1);
  ca0 = theta(2);
  n   = theta(3);
  x0  = ca0;
  %% solve model
[tdum, x] = ode15s(@massbal, tout, x0);
calculate fit to data
  ca = x;
  resid = ymeas - ca;
  phi = resid'*resid;
```

Comments.

1. Octave's nonlinear optimizer `npsol` is changed to MATLAB's `fmincon`.

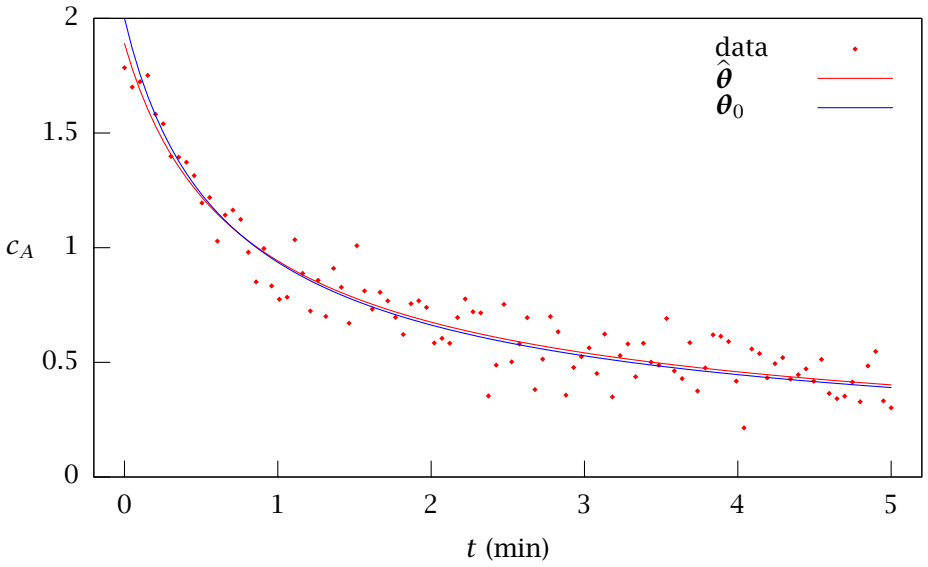

Figure 9: Experimental measurement and best parameter fit for nth-order kinetic model, $r = kc_A^n$.

2. Octave's ODE solver `lsode` is changed to MATLAB's `ode15s` as in previous sections.

3. In the MATLAB version II, we split out the two functions `massbal` and `model` and create .m files `massbal.m` and `model.m`.

The following table highlights these changes.

Octave	MATLAB— II

`[theta, obj, info] = npsol ...`
`  (theta0, 'model', lb, ub);`
	`[theta, obj, info] = fmincon ...`
	`  (@model,theta0, [], [], [], [], ...`
	`  lb, ub);`

`x = lsode('cstr', x0, tpts)`	`[tpts, x] = ode15s(@cstr, tpts, x0)`
`function xdot = massbal(x, t)`	`function xdot = massbal(t, x)`
`endfunction`	(none)
`endif`	`end`
(none)	additional file `massbal.m` created
(none)	additional file `model.m` created

Estimating Reaction Order and Rate Constant

To illustrate a parameter-estimation problem, we solve Example 9.4 in Rawlings and Ekerdt (2004). The problem is to find the rate constant and reaction order from isothermal concentration measurements in a batch reactor. The kinetics are

for the irreversible, nth order reaction

$$\mathrm{A} + \mathrm{B} \longrightarrow \text{ products}, \qquad r = kc_A^n$$

taking place in a liquid-phase batch reactor containing a large excess of reactant B. Given the measured concentration of component A shown in Figure 9, determine the best values of the model parameters.

The material balance for species A is

$$\frac{dc_A}{dt} = -kc_A^n$$
$$c_A(0) = c_{A0}$$

Given the data shown in Figure 9, it does not seem reasonable to assume we know c_{A0} any more accurately than the other measurements, so we include it as a parameter to be estimated. The model therefore contains three unknown parameters

$$\boldsymbol{\theta}^T = \begin{bmatrix} k & c_{A0} & n \end{bmatrix}$$

We can generate a reasonable initial parameter set by guessing values and solving the model until the model simulation is at least on the same scale as the measurements. We provide this as the starting point, and then solve the nonlinear optimization problem in

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta})$$

using the following least-squares objective

$$\Phi(\boldsymbol{\theta}) = \sum_i (\tilde{x}_i - x_i)^2$$

in which $\tilde{x}_i$ is the experimental measurement at time t_i, and x_i is the solution to the model at time t_i, $x_i = x(t_i; \boldsymbol{\theta})$. Note x_i is the only part of the objective function that depends on the model parameters. We minimize this objective function to obtain our parameter estimates.

$$\boldsymbol{\theta}_0 = \begin{bmatrix} k \\ c_{A0} \\ n \end{bmatrix} = \begin{bmatrix} 0.5 \\ 2.0 \\ 2.5 \end{bmatrix} \qquad \hat{\boldsymbol{\theta}} = \begin{bmatrix} 0.47 \\ 1.89 \\ 2.50 \end{bmatrix}$$

Both the parameters that generate the data, $\boldsymbol{\theta}_0$, and the parameter estimates, $\hat{\boldsymbol{\theta}}$, are given. We see that the data provided in Figure 9 allow accurate estimation of the reaction order and rate constant, as well as the initial A concentration.

References

James B. Rawlings and John G. Ekerdt. Chemical Reactor Analysis and Design Fundamentals. Nob Hill Publishing, Madison, WI, 2004. 609 pages, ISBN 0–615–11884–4.

M. Temkin and V. Pyzhev. Kinetics of ammonia synthesis on promoted iron catalysts. Acta Physicochimica U.R.S.S., 12(3):327–356, 1940.

C. van Heerden. Autothermic processes. Properties and reactor design. Ind. Eng. Chem., 45(6):1242–1247, 1953.