A
Computational Methods

In this appendix we summarize many of the fundamental computational procedures required in reactor analysis and design. We illustrate these procedures using the numerical computing language Octave [5, 6] or, equivalently, MATLAB. Octave is freely available for a variety of hardware platforms and can be downloaded from `www.octave.org`. The MATLAB package is commercially available from The MathWorks, Inc., and is becoming a commonly available tool of industrial engineering practice. One is naturally free to use any numerical solution procedure that is convenient, but we have found that these two environments provide reliable and high-quality numerical solutions, and convenient user interfaces.

Many of the figures and tables in this text required numerical solutions of various types of chemical reactor problems. All computations required to produce every figure and table in this text were performed with Octave. The Octave files required to produce these figures and tables are freely available at these two websites:

```
www.nobhillpublishing.com
www.engineering.ucsb.edu/~jbraw/chemreacfun
```

A.1 Linear Algebra and Least Squares

In this section we briefly introduce Octave and MATLAB, and show how to enter reaction networks quickly and perform linear algebra operations of interest.

After starting an Octave session, type `doc` and select menu item `Introduction` at the Octave prompt or `demo` at the MATLAB prompt for information on how to enter matrices and perform matrix multiplication. Consider again the water gas shift reaction chemistry, for

example,

$$\begin{bmatrix} 0 & 1 & 0 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{H} \\ \mathrm{H_2} \\ \mathrm{OH} \\ \mathrm{H_2O} \\ \mathrm{CO} \\ \mathrm{CO_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Say we would like to find the number of linearly independent reactions, and then compute the species production rates from given reaction rates, $[r_1\ r_2\ r_3] = [1\ 2\ 3]$. A typical set of Octave or MATLAB commands to perform these operations would appear as follows

```
octave:1> stoi=[0 1 0 -1 -1 1; -1 1 1 -1 0 0; \
                1 0 -1 0 -1 1];
octave:2> rank(stoi)
ans = 2
octave:3> r=[1;2;3];
octave:4> R=stoi'*r
R =
   1
   3
  -1
  -3
  -4
   4
```

As we explored in Chapters 2 and 9, from measured $\boldsymbol{R}$, we can solve for $\boldsymbol{r}$ as a least-squares problem. Given the production rates and reaction rates are related by

$$\boldsymbol{R} = \boldsymbol{\nu}^T \boldsymbol{r} \tag{A.1}$$

then the least-squares solution is given by

$$\boldsymbol{r} = (\boldsymbol{\nu}\, \boldsymbol{\nu}^T)^{-1} \boldsymbol{\nu} \boldsymbol{R}$$

in which the superscript -1 indicates a matrix inverse. The command in Octave and MATLAB for the least-squares solution is

```
r = stoi' \ R
```

which can be recalled with the mnemonic of a "left division" of Equation A.1 by $\boldsymbol{\nu}^T$.

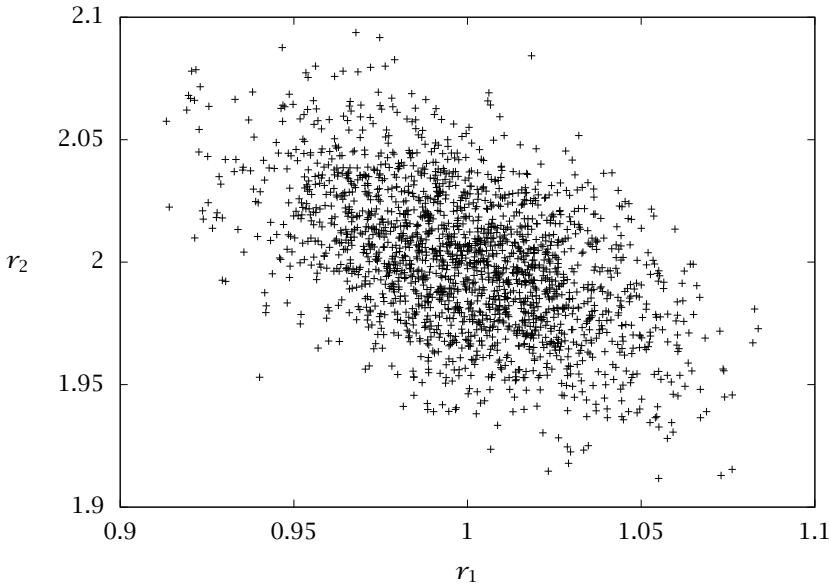

Figure A.1: Estimated reaction rates from 2000 production-rate measurements subject to measurement noise.

Example A.1: Estimating reaction rates

Consider the first and second water gas shift reaction as an independent set and let $[r_1\ r_2] = [1\ 2]$. Create 2000 production rate measurements by adding noise to $\boldsymbol{R} = \boldsymbol{\nu}^T \boldsymbol{r}$ and estimate the reaction rates from these data. Plot the distribution of estimated reaction rates.

Solution

The Octave commands to generate these results are

```
stoi = [0 1 0 -1 -1 1; -1 1 1 -1 0 0];
[nr, nspec] = size(stoi);
r = [1;2];
R = stoi'*r;
nmeas = 2000;
for i = 1:nmeas
  R_meas(:,i) = 0.05*randn(nspec, 1) + R;
end
r_est = stoi' \ R_meas;
```

```
plot(r_est(1,:), r_est(2,:), 'o')
```

Figure A.1 shows the estimates. We know from Chapter 9 that the distribution of estimates is a multivariate normal. We also know how to calculate the α-level confidence intervals. □

A.2 Nonlinear Algebraic Equations and Optimization

Determining equilibria for reacting systems with multiple phases and reactions can require significant computational effort. As we saw in Chapter 3, the phase and reaction equilibrium conditions generally lead to mathematical problems of two types: solving nonlinear algebraic equations and minimizing a nonlinear function subject to constraints. We explore computing solutions to these types of problems in Octave or MATLAB. In this section it is assumed that the required thermochemical data are available, but finding or measuring these data is often another significant challenge in computing equilibria for systems of industrial interest. See also Section 3.2.1 for a brief discussion of thermochemical databases.

A.2.1 Functions (`function`)

Octave and MATLAB provide many convenient built-in functions. Some that we have used in this text include the matrix exponential, `expm`; incomplete gamma function, `gammai`; pseudorandom number generators, `rand` and `randn`; and several others. Type `doc` and select the menu item `Function Index` for a complete list of Octave's built-in functions.

But often we need to define our own functions. To solve algebraic equations or optimization problems, for example, the user needs to provide a function that can be called by the solver to evaluate the nonlinear equations or the objective function. To define a function in Octave, $f(x) = x^2$ for example, the command structure is

```
function y = f(x)
  y = x*x;
endfunction
```

The first line states that the value of the variable `y` is returned by invoking the function named `f` with the input value `x`. The body of the function, then, is the calculation of the value of `y` to be returned. As the computational complexity of the function increases, we would like

to store our work so that it does not need to be reentered at the beginning of each subsequent Octave session. Storing, documenting and maintaining the calculations performed in solving complex industrial problems is one of the significant challenges facing practicing chemical engineers. Even for our purposes, we should be able to store and edit our computational problems. Octave commands and function definitions can be stored in text files, called m-files; the naming convention is `filename.m`. Editing these files with a text editor during an interactive Octave session provides a convenient means for debugging new calculations.

A.2.2 Nonlinear Algebraic Equations (`fsolve`)

We consider again the problem in Example 3.5. The equilibrium condition requires the solution of two equations in two unknowns,

$$PK_1^\circ \mathcal{Y}_I \mathcal{Y}_B - \mathcal{Y}_{P_1} = 0$$
$$PK_2^\circ \mathcal{Y}_I \mathcal{Y}_B - \mathcal{Y}_{P_2} = 0$$

in which $\mathcal{Y}_I, \mathcal{Y}_B, \mathcal{Y}_{P_1}, \mathcal{Y}_{P_2}$ are defined in terms of the two reaction extents in Equations 3.66. An Octave function defining these two equations is

```
function residual = dgdx(x)
 K1 = 108; K2 = 284; P  = 2.5;
 yI0 = 0.5; yB0 = 0.5; yP10 = 0; yP20 = 0;
 d   = 1 - x(1) - x(2);
 yI  = (yI0 - x(1) - x(2)) / d;
 yB  = (yB0 - x(1) - x(2)) / d;
 yP1 = (yP10 + x(1)) / d;
 yP2 = (yP20 + x(2)) / d;
 residual(1) = P*K1*yI*yB - yP1;
 residual(2) = P*K2*yI*yB - yP2;
endfunction
```

Notice that `x` is a vector containing the two reaction extents provided to the function named `dgdx` when it is invoked, and `residual` is a vector containing the values of the two equilibrium conditions to be returned by the function. We seek values of `x` such that `residual` is zero. With this function defined, we can use the built-in nonlinear equation solver to find its solution, by entering

```
[x,info] = fsolve(@dgdx, x0)
```

The `fsolve` function requires the name of the function, following the `@` sign, and an initial guess for the unknown extents, provided in the variable `x0`, and returns the solution in `x` and a flag `info` indicating if the calculation was successful. It is highly recommended to examine these information flags after all calculations. Reactor equilibrium problems can be numerically challenging, and even the best software can run into problems. After the function `dgdx` is defined, the following is a typical session to compute the solution given in Example 3.5

```
octave:1> x0=[0.2;0.2];
octave:2> [x,info] = fsolve(@dgdx,x0)
x =
  0.13317
  0.35087
info = 1
```

The value of `info = 1` indicates that the solution has converged.

A.2.3 Nonlinear Optimization (`fmincon`)

The other main approach to finding the reaction equilibrium is to minimize the appropriate energy function, in this case the Gibbs energy. This optimization-based formulation of the problem, as shown in Example 3.3, can be more informative than the algebraic approach discussed above. In Chapter 3, we derived the following Gibbs energy function for an ideal-gas mixture

$$\begin{aligned}\tilde{G} &= -\sum_i \varepsilon'_i \ln K^\circ_i + \left(1 + \sum_i \bar{\nu}_i \varepsilon'_i\right) \ln P \\ &\quad + \sum_j \left(y_{j0} + \sum_i \nu_{ij} \varepsilon'_i\right) \ln \left[\frac{y_{j0} + \sum_i \nu_{ij} \varepsilon'_i}{1 + \sum_i \bar{\nu}_i \varepsilon'_i}\right] \end{aligned} \tag{A.2}$$

The minimization of this function of ε'_i then determines the two equilibrium extents. Figure A.2 shows the lines of constant Gibbs energy determined by Equation A.2 as a function of the two reaction extents. We see immediately that the minimum is unique. Notice that $\tilde{G}$ is not defined for all reaction extents, and the following constraints are required to ensure nonnegative concentrations

$$0 \le \varepsilon'_1 \qquad 0 \le \varepsilon'_2 \qquad \varepsilon'_1 + \varepsilon'_2 \le 0.5$$

We proceed to translate this version of the equilibrium condition into a computational procedure. First we define a function that evaluates $\tilde{G}$ given the two reaction extents.

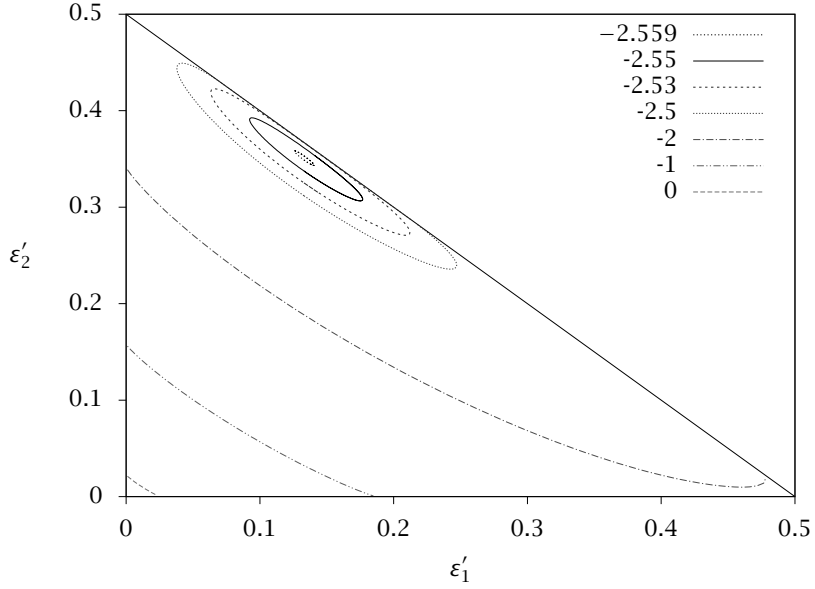

Figure A.2: Gibbs energy contours for the pentane reactions as a function of the two reaction extents.

```
function retval=gibbs(x)
 dg1= -3.72e3; dg2= -4.49e3; T=400; R=1.987; P=2.5;
 K1 = exp(-deltag1/(R*T)); K2 = exp(-deltag2/(R*T));
 yI0 = 0.5; yB0 = 0.5; yP10 = 0; yP20 = 0;
 d   = 1 - x(1) - x(2);
 yI  = (yI0 - x(1) - x(2)) / d;
 yB  = (yB0 - x(1) - x(2)) / d;
 yP1 = (yP10 + x(1)) / d;
 yP2 = (yP20 + x(2)) / d;
 retval = - (x(1)*log(K1)+x(2)*log(K2)) + ...
    (1-x(1)-x(2))*log(P) + yI*d*log(yI) + ...
    yB*d*log(yB) + yP1*d*log(yP1) + yP2*d*log(yP2);
endfunction
```

To solve the optimization we use the Octave or MATLAB function `fmincon`

```
[x,obj,info]=fmincon(@gibbs,x0, A, b, Aeq, beq, lb,ub)
```

in which `x0` is the initial guess as before and the new variables are used to define the constraints (if any). The variable `lb` provides lower

bounds for the extents and `ub` provides upper bounds for the extents. These are required to prevent the optimizer from "guessing" reaction extent values for which the Gibbs energy function is not defined. For example, consider what the `gibbs` function returns given negative reaction extents. Finally, we specify the sum of extents is less than 0.5 with the `A` and `b` arguments, which define linear inequality constraints of the form

$$Ax \leq b$$

For our problem, $\varepsilon_1' + \varepsilon_2' \leq 0.5$, so $A = [\ 1 \quad 1\ ]$ and $b = 0.5$. In problems requiring equality constraints, we use the arguments `Aeq` and `beq`, which define linear equality constraints of the form

$$A_{eq}x = b_{eq}$$

Since we do not have equality constraints in this problem, we use `[]` to indicate that these arguments are *empty*. Therefore, a typical session to solve for equilibrium composition by minimizing Gibbs energy is

```
octave:1> A=[1 1]; b=0.4999
octave:2> lb=[0;0]; ub=[0.5;0.5];
octave:3> x0=[0.2;0.2];
octave:4> [x,obj,info]=fmincon(@gibbs,x0,A,b,[],[],lb,ub)
x =

   0.13317
   0.35087

obj = -2.5593
info =  2
```

In this case, the value of `info = 2` indicates that the solution has converged, and the results are in good agreement with those computed using the algebraic approach, and the Gibbs energy contours depicted in Figure A.2.

Optimization is a powerful tool for solving many types of engineering modeling and design problems. We also rely heavily on optimization tools in Chapter 9 on parameter estimation.

A.3 Differential Equations

The real workhorse for simulating chemical reactors is the differential equation solver. A high-quality solver is an indispensable tool of the

reactor designer because chemical reactions can take place at widely different rates and time scales, presenting difficult numerical challenges. In the following we discuss several types of differential equations, including *ordinary* differential equations (ODEs), *implicit* differential equations, and *differential-algebraic* equations (DAEs), and the solvers that have been developed for each type.

A.3.1 Ordinary Differential Equations (lsode)

We start with ordinary differential equations (ODEs). The ODEs required to describe transient batch reactors and CSTRs, and steady-state PFR reactors are of the form

$$\frac{dx}{dt} = f(x)$$
$$x(0) = x_0$$

in which $x(t)$ is a vector of species concentrations to be found, t is time in the batch reactor and CSTR, but volume or length in the steady-state PFR, and f is the function defining the material balances for the species. If the differential equations are nonlinear (f is not a linear function of x) and one must keep track of several species simultaneously, then analytical solution is not possible and a numerical procedure is required. Type

```
doc
```

and select the menu item

```
Differential Equations
```

for an overview of Octave's ODE solver, `lsode` [7]. Type

```
help ode15s
```

for an overview of MATLAB's ODE solvers. We recommend always using the so-called "stiff" solvers for chemical reactor modeling. The time scales for different reactions may be widely separated, leading to stiff ODEs.

The first task is to formulate a function which defines f. If we examine the benzene pyrolysis example, Example 4.5, a suitable function is

```
function xdot = benz_pyrol_rhs(ext,volume)
 global k1 K1 k2 K2 R T P NBf
 NB = NBf - 2*ext(1) - ext(2);
 ND = ext(1) - ext(2);
 NH = ext(1) + ext(2);
 NT = ext(2);
 Q = NBf*R*T/P;
 cB = NB/Q;  cD = ND/Q;  cT = NT/Q;  cH = NH/Q;
 xdot(1) = k1*(cB*cB - cD*cH/K1);
 xdot(2) = k2*(cB*cD - cT*cH/K2);
endfunction
```

in which the meanings of the symbol names should be apparent from the example.

We then need to specify at which t values, volumes in this case, the solution should be reported, the initial condition (in this case, feed condition), x_0, and call the ODE solver. A typical Octave session would be

```
NBf = 60e3; R  = 0.08205; T  = 1033; P  = 1;
k1 = 7e5; k2 = 4e5; K1 = 0.31; K2 = 0.48;
x0=[0;0];
vout = linspace(0,1600,50)';
x = lsode(@benz_pyrol_rhs, x0, vout);
conv = ( 2*x(:,1) + x(:,2) ) /NBf;
yB   = ( NBf - 2*x(:,1) - x(:,2) ) /NBf;
yD   = ( x(:,1) - x(:,2) ) /NBf;
yT   = ( x(:,2) ) /NBf;
yH   = ( x(:,1) + x(:,2) ) /NBf;
```

in which the function `linspace` computes 50 linearly spaced points between 0 and 1600 L. Finally, any quantities such as conversions or yields, which are computable from x, are calculated for plotting. The results of this calculation are shown in Figures 4.20 and 4.21.

A.3.2 Octave-MATLAB Compatibility

Maintaining a high level of compatibility between a free software project like Octave and a commercial product like MATLAB can be a challenge. We illustrate one of the issues with the ODEs just discussed. The general form for time-varying ODEs is

$$\frac{dx}{dt} = f(x,t)$$

in which f may depend on time t as well as state x. Octave's interface to `lsode` defines this function as follows

```
function xdot = benz_pyrol_rhs(ext, volume)
```

In most problems in science and engineering, however, the function f is time invariant. For these time-invariant ODEs, we can simply drop the second argument and write more compactly

```
function xdot = benz_pyrol_rhs(ext)
```

In MATLAB on the other hand, the function defining the right-hand side for `ode15s` has to be of the form

```
function xdot = benz_pyrol_mat(volume, ext)
```

Notice that the order of the time argument, `volume`, and state argument, `ext`, have been reversed compared to the Octave convention. Even though the function seldom requires the argument `volume`, it must be present in the argument list in MATLAB.

We can enhance compatibility by defining an interface (or wrapper) in Octave with the name `ode15s`. The octave interface `ode15s` allows the user to follow the MATLAB convention for the right-hand side function, and calls `lsode` to solve the differential equations. The advantage is that the *same* user's code works correctly inside Octave and MATLAB. But the user should be aware that the solver `lsode`, which is a variable order predictor/corrector method, is being invoked when calling `ode15s` in Octave while the solver `ode15s` in MATLAB is a different method [8]. The choice of solution method should be transparent to the user as long as both methods are solving the problem accurately. But when the user encounters numerical difficulties on challenging problems, which is expected no matter what numerical tools one is using, knowing what is actually underneath the hood becomes important in diagnosing and fixing the problem. If we wish to solve the ODEs by calling the wrapper instead of `lsode` directly, then we change the line

```
x = lsode(@benz_pyrol_rhs, x0, vout);
```

to

```
[vsolver, x] = ode15s(@benz_pyrol_mat, vout, x0, opts);
```

Notice that `ode15s` returns the times (volumes) at which the solution was found `vsolver` in the first argument and the solution `x` as the second argument in the output. Comparing `lsode` and `ode15s` input argument lists, we also see that the initial condition `x0` and times (volumes)

Problem type	Wrapper name	Octave solver
Optimization	`fmincon`	`sqp`
ODEs	`ode15s`	`lsode`
Implicit DEs	`ode15i`	`daspk`
Implict DEs with root finding	`ode15i`	`dasrt`

Table A.1: Octave compatibility wrapper functions.

at which output is desired `vout` are also switched. The code posted at `www.engineering.ucsb.edu/~jbraw/chemreacfun` for Figures 4.20 or 4.21 show the complete solution for Example 4.5 using `ode15s`.

Table A.1 provides a list of some Octave compatibility functions that are available. Notice that the nonlinear optimizer `fmincon` already discussed is a wrapper of the successive quadratic programming method `sqp` provided in Octave by John Eaton.

A.3.3 Differential-Algebraic Equations (daspk)

Some reactor models require a more general structure than the ODE, $d\boldsymbol{x}/dt = \boldsymbol{f}(\boldsymbol{x}, t)$. The nonconstant density, nonconstant volume semibatch and CSTR reactors in Chapter 4 are more conveniently expressed as differential-algebraic equations (DAEs). To address these models, consider the more general form of ODEs

$$\mathbf{0} = \boldsymbol{f}(d\boldsymbol{x}/dt, \boldsymbol{x}, t) \tag{A.3}$$

These are known as *implicit* ODEs because one may not be able to solve this model for $d\boldsymbol{x}/dt$ explicitly. Differential algebraic equations (DAEs) are of the form

$$\begin{aligned} \frac{d\boldsymbol{x}}{dt} &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \\ \mathbf{0} &= \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

in which the subset of state variables $\boldsymbol{y}$ are known as the algebraic states because their time derivatives do not appear in the model. Both DAEs and ODEs are special cases of the implicit ODEs in Equation A.3. Brenan, Campbell and Petzold [2] provide further reading on existence and uniqueness of solutions to these models, which are considerably

more complex issues than in the case of simple ODEs. Initial conditions are required for $d\boldsymbol{x}/dt$ as well as $\boldsymbol{x}$ in this model,

$$\frac{d\boldsymbol{x}}{dt}(t) = \dot{\boldsymbol{x}}_0 \qquad \boldsymbol{x}(t) = \boldsymbol{x}_0, \qquad \text{at } t = 0$$

Brown, Hindmarsh and Petzold [3, 4] have provided a numerical package, `daspk`, to compute solutions to implicit differential, and differential-algebraic equations. The main difference between using `daspk` and `lsode` is the form of the user-supplied function defining the model. A second difference is that the user must supply $\dot{\boldsymbol{x}}_0$ as well as $\boldsymbol{x}_0$.

As an example, we can solve the previous ODEs using `daspk`. First we modify the right-hand side function `benz_pyrol_rhs` of the ODEs to return the *residual* given in Equation A.3. We call the new function `benz_pyrol`

```
function res = benz_pyrol(ext, extdot, volume)
 global k1 K1 k2 K2 R T P NBf
 NB = NBf - 2*ext(1) - ext(2);
 ND = ext(1) - ext(2);
 NH = ext(1) + ext(2);
 NT = ext(2);
 Q = NBf*R*T/P;
 cB = NB/Q;  cD = ND/Q;  cT = NT/Q;  cH = NH/Q;
 res(1) = extdot(1) - k1*(cB*cB - cD*cH/K1);
 res(2) = extdot(2) - k2*(cB*cD - cT*cH/K2);
endfunction
```

Notice that when this function returns zero for `res`, the variables `extdot` and `ext` satisfy the differential equation. With this function defined, we then solve the model by calling `daspk`

```
x0=[0;0]; xdot0 = [0;0];
vout = linspace(0,1600,50)';
[x, xdot] = daspk(@benz_pyrol, x0, xdot0, vout);
```

Notice that both the state `x` and state time (volume) derivative `xdot` are returned at the specified times (volumes) `vout`.

For compatibility with MATLAB's implicit ODE solver, `ode15i`, Octave provides a wrapper function with this name. The differences are that arguments are in a different order in the function defining the residual

```
function resid = benz_pyrol_mat(volume, x, xdot)
```

and in the call to `ode15i`

```
[vstop,x] = ode15i (@benz_pyrol_mat, vout, x0, xdot0);
```

A.3.4 Automatic Stopping Times (dasrt)

We often need to stop ODE solvers when certain conditions are met. Two examples are when we have reached a certain conversion, and when we have created a new phase of matter and need to change the ODEs governing the system. The program `dasrt` enables us to stop when specified conditions are met exactly. The following code illustrates how to find the PFR volume at which 50% conversion of benzene is achieved. The user provides one new function, `stop`, which the ODE solver checks while marching forward until the function reaches the value zero. The PFR volume is found to be $V_R = 403.25$ L, in good agreement with Figure 4.20.

```
function val = stop(ext, volume)
  global NBf xBstop
  NB = NBf - 2*ext(1) - ext(2);
  xB = 1 - NB/NBf;
  val = xBstop - xB;
endfunction

octave:1> x0 = [0;0]; xdot0 = [0;0];
octave:2> vout = linspace(0,1600,50)'; xBstop = 0.5;
octave:3> [x, xdot, vstop] = dasrt(@benz_pyrol, ...
                                   @stop, x0, xdot0, vout);
octave:4> vstop(end)
ans = 403.25
```

For compatibility with MATLAB, Octave provides an option flag to `ode15i` that signals that a stopping condition has been specified in the model. To use `ode15i` to solve the benzene pyrolysis problem, replace the call to `dasrt`

```
[x,xdot,vstop] = dasrt(@benz_pyrol,@stop,x0,xdot0,vout);
```

with

```
opts = odeset ('Events', @stop_mat);
[vstop,x] = ode15i(@benz_pyrol_mat,vout,x0,xdot0,opts);
```

The `opts` flag `'Events'` tells `ode15i` to look for stopping conditions as zeros of function `stop_mat`. Note that the function `stop_mat` requires the arguments in the following order

```
function val = stop(volume, ext)
```

which is the reverse required by `dasrt` in function `stop` defined above.

A.4 Parametric Sensitivities of Differential Equations (paresto)

As discussed in Chapter 9, it is often valuable to know not just the solution to a set of ODEs, but how the solution changes with the model parameters. Consider the following vector of ODEs with model parameters $\boldsymbol{\theta}$.

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})$$
$$\boldsymbol{x}(0) = \boldsymbol{g}(\boldsymbol{x}_0;\boldsymbol{\theta})$$

The sensitivities, $\boldsymbol{S}$, are defined as the change in the solution with respect to the model parameters

$$S_{ij} = \left(\frac{\partial x_i}{\partial \theta_j}\right)_{\theta_{k\neq j}}$$
$$\boldsymbol{S} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\theta}^T}$$

We can differentiate the S_{ij} to obtain

$$\frac{dS_{ij}}{dt} = \frac{d}{dt}\left[\frac{\partial x_i}{\partial \theta_j}\right] = \frac{\partial}{\partial \theta_j}\frac{dx_i}{dt} = \frac{\partial}{\partial \theta_j} f_i(\boldsymbol{x};\boldsymbol{\theta})$$

Using the chain rule to perform the final differentiation gives

$$\left(\frac{\partial f_i}{\partial \theta_j}\right)_{\theta_{k\neq j}} = \sum_k \left(\frac{\partial f_i}{\partial x_k}\right)_{x_{l\neq k},\theta_j} \left(\frac{\partial x_k}{\partial \theta_j}\right)_{\theta_{k\neq j}} + \left(\frac{\partial f_i}{\partial \theta_j}\right)_{x_l,\theta_{k\neq j}} \tag{A.4}$$

Substituting this back into Equation A.4 and writing the sum as a matrix multiply yields

$$\frac{d\boldsymbol{S}}{dt} = \boldsymbol{f}_{\boldsymbol{x}}\boldsymbol{S} + \boldsymbol{f}_{\boldsymbol{\theta}}$$

in which the subscript denotes differentiation with respect to that variable. Notice the sensitivity differential equation is linear in the unknown $\boldsymbol{S}$. The initial conditions for this matrix differential equation is easily derived from the definition of $\boldsymbol{S}$

$$S_{ij}(0) = \frac{\partial x_i(0)}{\partial \theta_j} = \frac{\partial g_i}{\partial \theta_j}$$

$$\frac{d\boldsymbol{S}}{dt} = \boldsymbol{f}_{\boldsymbol{x}}\boldsymbol{S} + \boldsymbol{f}_{\boldsymbol{\theta}}$$
$$\boldsymbol{S}(0) = \boldsymbol{g}_{\boldsymbol{\theta}}$$

Notice that if none of the initial conditions are parameters for which one is computing sensitivities, then $\boldsymbol{S}(0) = \mathbf{0}$.

Example A.2: Two simple sensitivity calculations

Consider the differential equation describing a first-order, isothermal, irreversible reaction in a constant-volume batch reactor

$$\begin{aligned} \frac{dc_A}{dt} &= -kc_A \\ c_A(0) &= c_{A0} \end{aligned} \tag{A.5}$$

1. First write out the solution to the differential equation.
2. Next consider the rate constant to be the parameter in the problem, $\theta = k$. Take the partial derivative of the solution directly to produce the sensitivity to this parameter, $S = \partial c_A / \partial k$.
3. Next write the differential equation describing dS/dt. What is $S(0)$? Solve this differential equation and compare to your previous result. Plot $c_A(t), S(t)$ versus t.
4. Repeat steps 2 and 3 using the initial concentration as the parameter in the problem, $\theta = c_{A0}$.

Solution

1. Equation A.5 is our standard, first-order linear equation, whose solution is
$$c_A = c_{A0} e^{-kt}$$
2. Taking the partial derivative of $c_A(t)$ with respect to $\theta = k$ produces directly
$$S_1 = -t c_{A0} e^{-kt} \tag{A.6}$$
Evaluating the various partial derivatives gives
$$f_x = -k \qquad f_\theta = -c_A \qquad g_\theta = 0$$
so the sensitivity differential equation is
$$\begin{aligned} \frac{dS_1}{dt} &= -kS_1 - c_A \\ S_1(0) &= 0 \end{aligned}$$

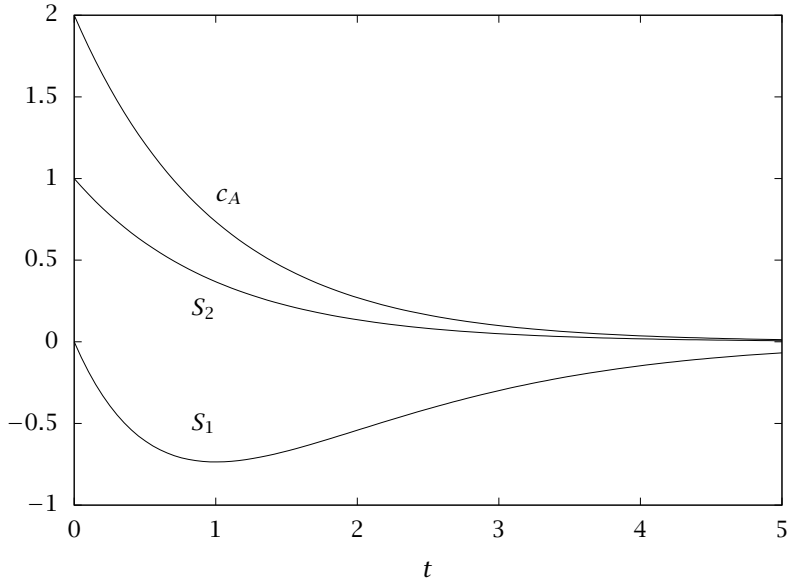

Figure A.3: Solution to first-order differential equation $dc_A/dt = -kc_A$, and sensitivities $S_1 = \partial c_A/\partial k$ and $S_2 = \partial c_A/\partial c_{A0}$.

This equation is linear and can be solved with the method used in the series reactions in Chapter 4, or with Laplace transforms as in Exercise 4.6. The result is

$$S_1(t) = -tc_{A0}e^{-kt}$$

which agrees with Equation A.6. The functions $c_A(t), S_1(t)$ are shown in Figure A.3.

3. Taking the partial derivative of $c_A(t)$ with respect to $\theta = c_{A0}$ produces directly

$$S_2(t) = e^{-kt} \tag{A.7}$$

Evaluating the various partial derivatives for this parameter choice gives

$$f_x = -k \qquad f_\theta = 0 \qquad g_\theta = 1$$

so the sensitivity differential equation is

$$\frac{dS_2}{dt} = -kS_2$$
$$S_2(0) = 1$$

This equation is our standard, first-order linear equation whose solution is

$$S_2(t) = e^{-kt}$$

which agrees with Equation A.7. This sensitivity also is plotted in Figure A.3. Notice that only when the initial condition is the parameter is the sensitivity initially nonzero.

□

The following Octave commands show how to use `paresto` to solve Example A.2.

```
ca0 = 2; k   = 1; tfinal = 5;
nts    = 100; tout = linspace(0,tfinal,nts)';

model = struct;
model.transcription = 'shooting';
model.x = {'ca'};
model.p = {'k'};
model.d = {'dummy'};
model.tout = tout;
model.ode = @(t, y, p) {-p.k*y.ca};
% dummy objective function; just finding sensitivities
model.lsq = @(t, y, p) {y.ca};
pe = paresto(model);

%% initialize state and parameters
x0 = ca0; thetaac = k;

y_ac = pe.simulate(0, x0, thetaac);

theta0.k = k;
theta0.ca = ca0;
lb.k = theta0.k; lb.ca = theta0.ca;
ub.k = theta0.k; ub.ca = theta0.ca;

[est, y, p] = pe.optimize(y_ac, theta0, lb, ub);

plot (tout, [y.ca', est.dca_dk, est.dca_dca0]);
```

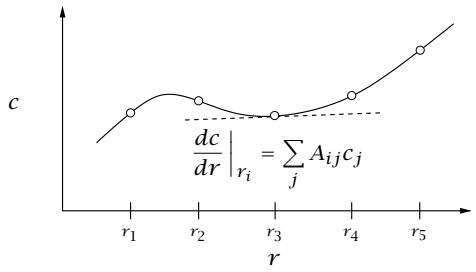

Figure A.4: Function $c(r)$ and its values at five collocation points. Derivatives and integrals of the polynomial interpolant are linear combinations of the function values at the points.

A.5 Boundary-Value Problems and Collocation (col1oc)

In the collocation method, we approximate a function by passing a polynomial through values of the function at selected points. The selected points are known as collocation points. The locations of the collocation points have a large impact on how well the method works. Evenly spaced points, which seems a natural first choice, turns out to have mediocre properties. Choosing the points as zeros of a member of a family of orthogonal polynomials turns out to have much better properties. This choice is referred to as orthogonal collocation.

Figure A.4 shows values of a function $c(r)$ at five collocation points. The function is approximated by passing a polynomial through these points. To solve differential equations and boundary-value problems (BVP), we first compute the required derivatives of the polynomial approximation. We then find the values of the function such that the differential equation is satisfied at the collocation points. If we increase the number of collocation points, n_c, we require the differential equation to be satisfied at more locations, and we obtain a more accurate solution.

The derivatives and integrals of the polynomial interpolant can be

computed as linear combinations of the values at the collocation points

$$\left.\frac{dc}{dr}\right|_{r_i} = \sum_{j=1}^{n_c} A_{ij} c_j$$

$$\left.\frac{d^2c}{dr^2}\right|_{r_i} = \sum_{j=1}^{n_c} B_{ij} c_j$$

$$\int_0^1 f(r)dr = \sum_{j=1}^{n_c} Q_j f(r_j)$$

To obtain the locations of the collocation points and derivatives and integral weighting matrices and vectors, we use the function `colloc`, based on the methods described by Villadsen and Michelsen [9].

```
[R A B Q] = colloc(npts-2, 'left', 'right');
```

The strings `'left'` and `'right'` specify that we would like to have collocation points at the endpoints of the interval in addition to the zeros of the orthogonal polynomial. We solve for the concentration profile for the reaction-diffusion problem in a catalyst pellet to illustrate the collocation method.

Example A.3: Single-pellet profile

Consider the isothermal, first-order reaction-diffusion problem in a spherical pellet

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{dc}{dr}\right) - \Phi^2 c = 0 \tag{A.8}$$

$$c = 1, \qquad r = 3$$

$$\frac{dc}{dr} = 0, \qquad r = 0$$

The effectiveness factor is given by

$$\eta = \left.\frac{1}{\Phi^2}\frac{dc}{dr}\right|_{r=3}$$

1. Compute the concentration profile for a first-order reaction in a spherical pellet. Solve the problem for $\phi = 10$.

2. Plot the concentration profile for $n_c = 5, 10, 30$ and 50. How many collocation points are required to reach accuracy in the concentration profile for this value of Φ.

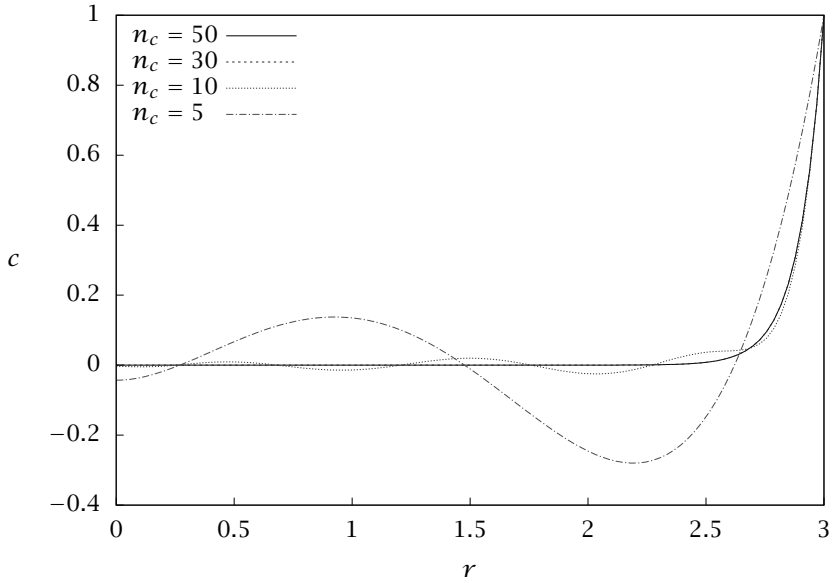

Figure A.5: Dimensionless concentration versus dimensionless radial position for different numbers of collocation points.

3. How many collocation points are required to achieve a relative error in the effectiveness factor of less than 10^{-5}?

Solution

First we perform the differentiation in Equation A.8 to obtain

$$\frac{d^2c}{dr^2} + \frac{2}{r}\frac{dc}{dr} - \Phi^2 c = 0$$

We define the following Octave function to evaluate this equation at the interior collocation points. At the two collocation endpoints, we satisfy the boundary conditions $dc/dr = 0$ at $r = 0$ and $c = 1$ at $r = 3$. The function is therefore

```
function retval = pellet(c)
  global Phi A B R
% differential equation at all points
  retval = B*c + 2*A*c./R - Phi^2*c;
```

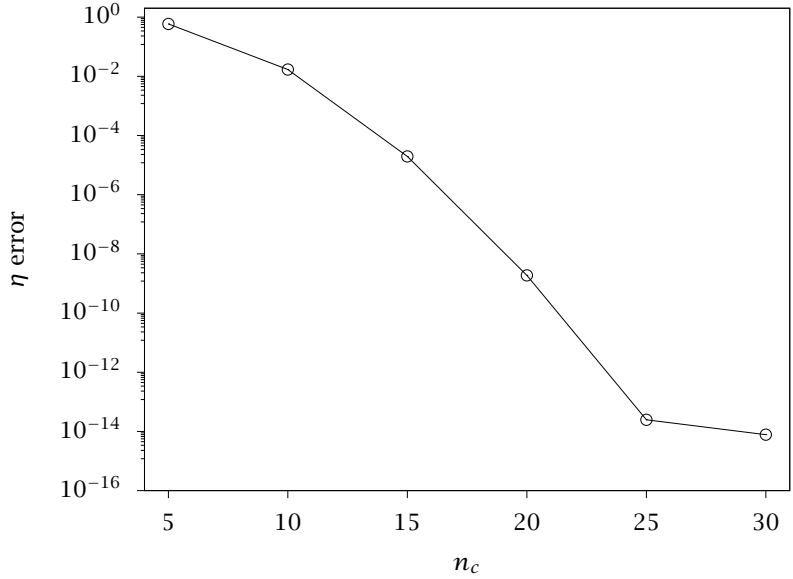

Figure A.6: Relative error in the effectiveness factor versus number of collocation points.

```
% write over first and last points with BCs
  retval(1)    = A(1,:)*c;
  retval(end) = 1 - c(end);
endfunction
```

Figure A.5 shows the concentration profiles for different numbers of collocation points. We require about $n_c = 30$ to obtain a converged concentration profile. Figure A.6 shows the relative error in the effectiveness factor versus number of collocation points. If one is only interested in the pellet reaction rate, about 16 collocation points are required to achieve a relative error of less than 10^{-6}. □

Example A.4: A simple fixed-bed reactor problem

Here we set up the collocation approach of this section for the single, second-order reaction

$$\mathrm{A} \longrightarrow \mathrm{B} \qquad r = kc_A^2$$

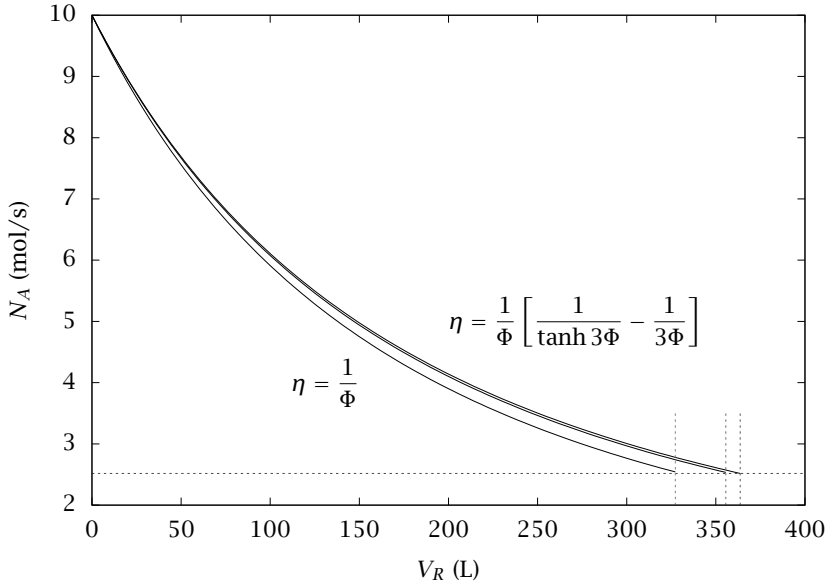

Figure A.7: Molar flow of A versus reactor volume for second-order, isothermal reaction in a fixed-bed reactor; two approximations and exact solution.

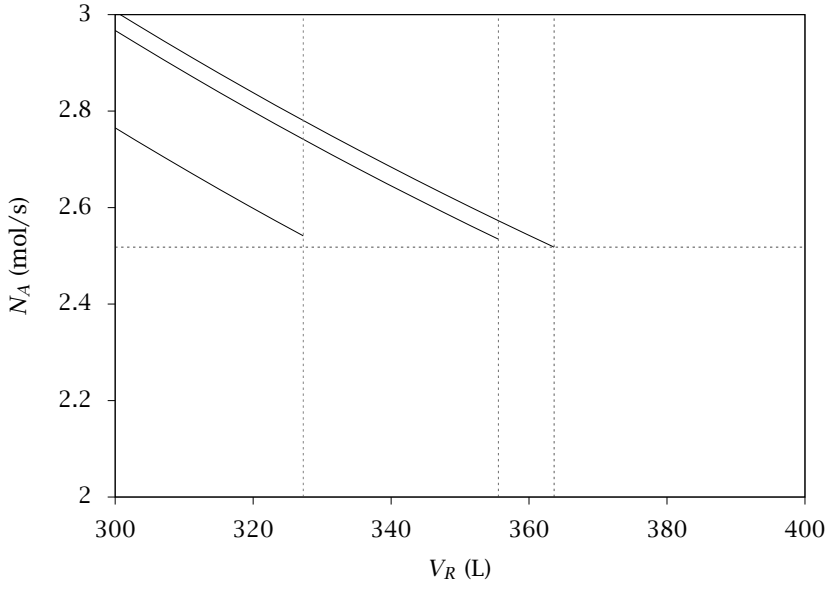

Figure A.8: Magnified view of Figure A.7.

Solve the model exactly and compare the numerical solution to the solution obtained with the *approximate* Thiele modulus and effectiveness factor approaches shown in Figure 7.26.

Solution

The model for the fluid and particle phases can be written from Equations 7.67–7.77

$$\frac{dN_A}{dV} = R_A = -(1-\epsilon_B)\frac{S_p}{V_p}D_A \left.\frac{d\tilde{c}_A}{dr}\right|_{r=R}$$

$$\frac{d^2\tilde{c}_A}{dr^2} + \frac{2}{r}\frac{d\tilde{c}_A}{dr} + \frac{\tilde{R}_A}{D_A} = 0$$

$$\tilde{c}_A = c_A \qquad r = R$$

$$\frac{d\tilde{c}_A}{dr} = 0 \qquad r = 0$$

The collocation method produces algebraic equations for the pellet profile as for the previous example. We use a DAE solver to integrate the differential equation for N_A and these coupled algebraic equations.

The results for N_A versus reactor length using 25 collocation points for the pellet are shown in Figure A.7. Also shown are the simplified effectiveness factor calculations for this problem from Example 7.5. A magnified view is shown in Figure A.8. Notice the effectiveness factor approach gives a good approximation for the bed performance. It is not exact because the reaction is second order. □

A.6 Parameter Estimation with ODE Models (paresto)

We have developed the function `paresto.m` to estimate parameters in differential equation models [1]. We present a simple example to illustrate some of the features.

Example A.5: Estimating two rate constants in reaction A→ B→ C

Consider the irreversible series reactions

$$\mathrm{A} \xrightarrow{k_1} \mathrm{B} \xrightarrow{k_2} \mathrm{C} \tag{A.9}$$

Estimate the two rate constants k_1 and k_2 from the measurements shown in Figure A.9. We would also like to know how much confidence to place in these parameter estimates.

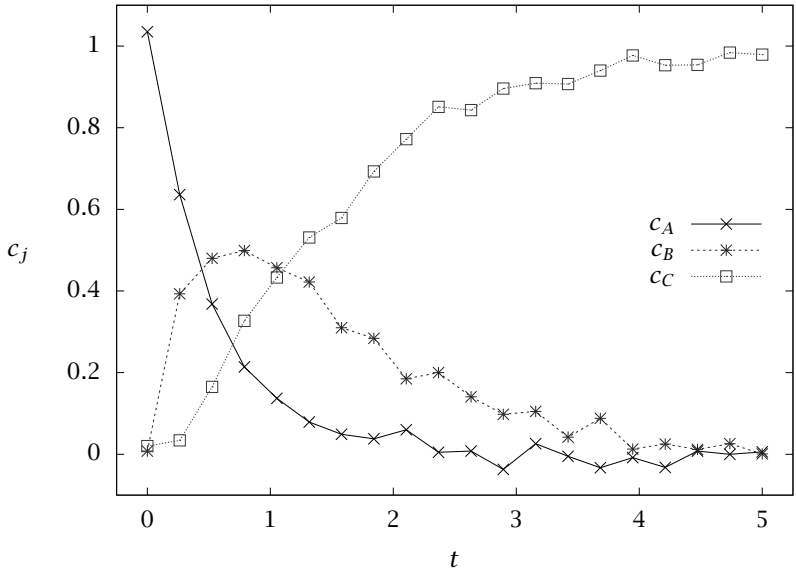

Figure A.9: Measurements of species concentrations in Reactions A.9 versus time.

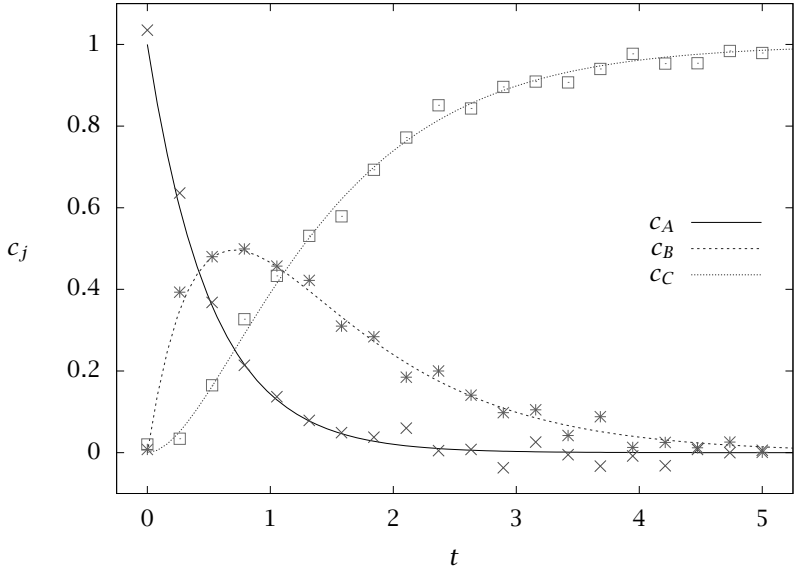

Figure A.10: Fit of model to measurements using estimated parameters.

Solution

For this problem we require only the following simple paresto options.

```
model.x = {'ca', 'cb', 'cc'};
model.p = {'k1', 'k2'};
model.d = {'m_ca', 'm_cb', 'm_cc'};

model.ode = @massbal_ode;
model.lsq = @(t, y, p) {y.ca-y.m_ca, y.cb-y.m_cb, y.cc-y.m_cc};

tfinal = 6;
nplot = 75;
tplot = linspace(0, tfinal, nplot)';

k1 = 2; k2 = 1; p_ac = [k1; k2];

ca0 = 1; cb0 = 0; cc0 = 0; x_ac = [ca0; cb0; cc0];

thetaic   = [0.5; 3; x_ac];
thetalb   = [1e-4; 1e-4; 0.9*x_ac];
thetaub   = [10; 10; 1.1*x_ac];
est_ind = 1:2;

%% load measurements from a file
tabledat = load ('ABC_data.dat');
tmeas = tabledat(:,1);
ymeas = tabledat(:,2:4);

[tout,~,ic] = unique([tmeas; tplot]);
% index of times at which measurement is made
meas_ind = ic(1:numel(tmeas));
model.tout = tout;
% interploate measurement onto new grid
y_noisy = interp1(tmeas, ymeas, tout, 'previous');
y_noisy(isnan(y_noisy)) = 0.;
% use index of measurement times in objective
model.lsq_ind = meas_ind';

% Create a paresto instance
pe = paresto(model);

%% estimate the parameters
est = pe.optimize(y_noisy', thetaic, thetalb, thetaub);
```

```
% Also calculate confidence intervals with 95 % confidence
theta_conf = pe.confidence(est, est_ind, 0.95);

disp('Estimated Parameters and Bounding Box')
[est.theta(est_ind)  theta_conf]

%plot the model fit to the noisy measurements
plot(model.tout, est.x, tmeas, ymeas', 'o');
```

We also require the function `massbal_ode` to define the right-hand side of the differential equations

```
function xdot = massbal_ode(t, x, p)
  r1 = p.k1*x.ca;
  r2 = p.k2*x.cb;
  xdot = {-r1, r1-r2, r2};
endfunction
```

Running the `ABC.m` file produces the following parameter estimates and confidence intervals.

$$\hat{\boldsymbol{\theta}} = \begin{bmatrix} 2.02 \\ 0.993 \end{bmatrix} \pm \begin{bmatrix} 0.236 \\ 0.0860 \end{bmatrix} \qquad \boldsymbol{\theta}_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

The estimates are close to the true values $\boldsymbol{\theta}_0$ used to generate the data. The confidence intervals are reasonably tight given the three species measurements with the noise level indicated in Figure A.9. The fit of the model using the estimated parameters is shown in Figure A.10.

□

A.7 Exercises

Exercise A.1: Water gas shift reaction and production rates

Consider the water gas shift reactions

$$\begin{aligned} H_2O + CO &\rightleftharpoons CO_2 + H_2 \\ H_2O + H &\rightleftharpoons H_2 + OH \\ OH + CO &\rightleftharpoons CO_2 + H \end{aligned}$$

Take the second and third reactions as an independent set and estimate the reaction rates from the same 2000 production-rate measurements used in Example A.1.

(a) What value of $[r_2\ r_3]$ produces the error-free production rate corresponding to $[r_1\ r_2] = [1\ 2]$

(b) Create 2000 production-rate data points with measurement noise as in Example A.1. Estimate $[r_2\ r_3]$ from these data and plot the result. Compare to Figure A.1. Explain the differences and similarities in the two figures.

Exercise A.2: Semi-batch polymerization reactor

Consider again Example 4.3, in which monomer is converted to polymer in a semi-batch reactor.

(a) Solve the problem with `dasrt` and compute the time t_1 at which the reactor fills. Compare your answer with the result $t_1 = 11.2$ min given in Chapter 4.

(b) Calculate the solution for both addition policies and compare your numerical result to the analytical result in Figures 4.15-4.18.

Exercise A.3: Sensitivity for second-order kinetics

Repeat the analysis in Example A.2 for the differential equation describing a *second-order*, isothermal, irreversible reaction in a constant-volume batch reactor

$$\begin{aligned} \frac{dc_A}{dt} &= -kc_A^2 \\ c_A(0) &= c_{A0} \end{aligned}$$

Note: instead of solving the sensitivity differential equation, show only that the sensitivity computed by taking the partial derivative of the solution to the differential equation, $c_A(t)$, *satisfies* the sensitivity differential equation and initial condition.

Exercise A.4: Multiple solutions to the Weisz-Hicks problem

Using orthogonal collocation, compute the solution $c(r)$ to the Weisz-Hicks problem, Equations 7.65–7.66, for the following parameter values: $\beta = 0.6$, $\gamma = 30$, $\Phi = 0.001$. Compare your result to the calculation presented in Figure A.11.

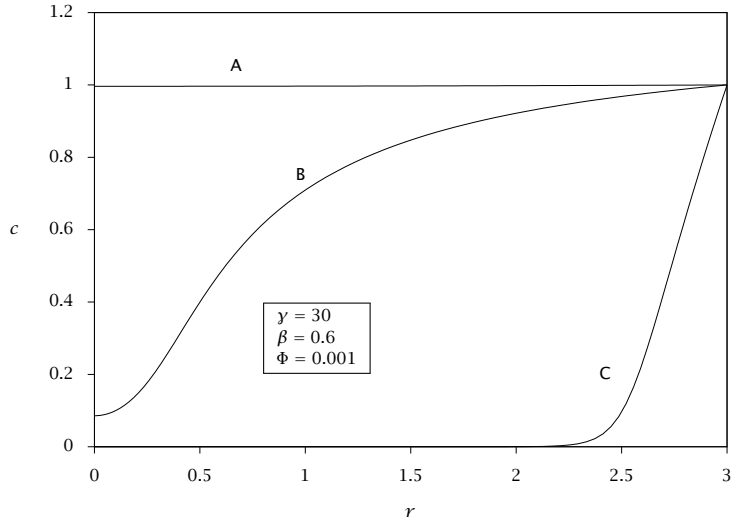

Figure A.11: Dimensionless concentration versus radius for the non-isothermal spherical pellet: lower (A), unstable middle (B), and upper (C) steady states.

Exercise A.5: Weisz-Hicks problem, increasing β

Using orthogonal collocation, resolve the Weisz-Hicks problem for the following parameter values: $\beta = 0.85$, $\gamma = 30$, $\Phi = 0.001$. Use initial guesses for the $c(r)$ profile based on the results in Figure A.11.

(a) How many solutions are you able to find? Given the results in Figure 7.19, how many solutions do you expect. Explain and justify your answer.

(b) Compare and contrast your $c(r)$ solutions to those in Figure A.11.

Exercise A.6: Comparing shooting and collocation

Solve the isothermal, first-order reaction-diffusion problem with an ODE solver, an algebraic equation solver, and the shooting method.

$$\frac{1}{r^q}\frac{d}{dr}\left(r^q\frac{dc}{dr}\right) - \frac{2}{n+1}\Phi^2 c^n = 0$$

$$c = 1, \qquad r = q+1$$

$$\frac{dc}{dr} = 0, \qquad r = 0$$

(a) Solve the problem with the shooting method for the first-order reaction in a spherical geometry, $n = 1$, $q = 2$, using $\Phi = 0.1, 1, 10, 50$. Compare $c(r)$ from the shooting method with the analytical solution.

(b) Solve the problem using the collocation method. Which method do you prefer and why?

(c) Consider the variable transformation

$$w = \ln(c)$$

Write the model and boundary conditions in the new variable, w. Solve the transformed model with the collocation method. Compare the collocation solution of the transformed model to the solution of the original model.

Exercise A.7: Transient solution of the dispersed PFR

Revisit the transient PFR example of Chapter 8, Example 8.1. Solve the problem for a range of dispersion numbers. Can you get a CSTR-like solution for large values of the dispersion number? Can you get a PFR-like solution for small values of the dispersion number? Do you have numerical accuracy problems at either of these limits?

Bibliography

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi—a software framework for nonlinear optimization and optimal control. *Math. Prog. Comp.*, 11(1):1–36, March 2019. https://doi.org/10.1007/s12532-018-0139-4.

[2] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* Elsevier Science Publishers, New York, 1989.

[3] P. Brown, A. Hindmarsh, and L. Petzold. Consistent initial condition calculation for differential-algebraic systems. *SIAM J. Sci. Comp.*, 19(5):1495–1512, 1998.

[4] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold. Using Krylov methods in the solution of large-scale differential-algebraic systems. *SIAM J. Sci. Comp.*, 15(6):1467–1488, November 1994.

[5] J. W. Eaton. Octave: Past, present and future. In K. Hornik and F. Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2001, Technische Universität Wien, Vienna, Austria*, 2001. http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/, ISSN 1609-395X.

[6] J. W. Eaton and J. B. Rawlings. Octave—a high level interactive language for numerical computations. *CACHE News*, (40):11–18, Spring 1995.

[7] A. C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. In R. S. Stepleman, editor, *Scientific Computing*, pages 55–64, Amsterdam, 1983. North-Holland.

[8] L. F. Shampine and M. W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comp.*, 18(1):1–22, January 1997.

[9] J. Villadsen and M. L. Michelsen. *Solution of Differential Equation Models by Polynomial Approximation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.