Process model formulation and solution, 3E4

Tutorial 4

Kevin Dunn, dunnkg@mcmaster.ca **October 2010**

Tutorial objectives

- Derive linear models for actual systems.
- Solve these models by hand, and with computer software.

Solutions prepared by Ali Sahlodin.

Question 1 [1]

In the second tutorial you derived the set of linear equations for the reaction:

$$\mathsf{P_2I_4} + n\,\mathsf{P_4} + p\,\mathsf{H_2O} \longrightarrow 4\,\mathsf{PH_4I} + q\,\mathsf{H_3PO_4}$$

where n, p and q denote the stoichiometric coefficients for the $\mathsf{P_4}$, $\mathsf{H_2O}$ and $\mathsf{H_3PO_4}$ species respectively.

1. Solve the linear equations using Gauss elimination, by hand.
2. Verify your solution using the `mldivide` (MATLAB) or `solve` (Python) function. How long does it take to find the solution? Contrast this with the approach used in tutorial 2.

Solution

Recall from the second tutorial that we have the following balance equations;

balance for O : $p - 4q = 0$,

balance for P : $4n - q = 2$,

balance for H : $2p - 3q = 16$.

These equations can be written in the form of $Ax = b$ where A is the matrix of coefficients, and b is a column vector containing the right-hand side constants. Also $x = [n \quad p \quad q]^T$ is the vector of unknowns. So, we have:

$$\begin{bmatrix} 0 & 1 & -4 \\ 4 & 0 & -1 \\ 0 & 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 16 \end{bmatrix}$$

Now we solve this system using the Gauss elimination (R_i denotes the $i-th$ row of the matrix). Note that your solution procedure (not the final results) may be different if you have arranged the equations and unknowns differently so you have a different A and b than the above.

Forward Elimination

As a first step, $A_{2,1}$ must be eliminated using $A_{1,1}$ in the first row. However, $A_{1,1} = 0$ by which we cannot divide. So, we do a partial pivoting by switching the first and second rows in the above matrix. That is:

$$\begin{bmatrix} 4 & 0 & -1 \\ 0 & 1 & -4 \\ 0 & 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 16 \end{bmatrix}$$

Interestingly, $A_{2,1}$ is already zero with the new rearrangement. This is a lucky case since nothing needs to be done to the second row of the matrix. Now we eliminate $A_{3,2}$ using $A_{2,2}$. To do so, multiply R_2 by -2, and add the result to R_3:

$R_3 \leftarrow R_3 - 2R_2$

which results in

$$\begin{bmatrix} 4 & 0 & -1 \\ 0 & 1 & -4 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 16 \end{bmatrix}$$

The forward elimination is complete

Backward substitution

From R_3: $5x_3 = 16 \rightarrow x_3 = q = 3.2$. Substituting x_3 into R_2: $x_2 - 4 \times 3.2 = 0 \rightarrow x_2 = p = 12.8$. Finally, substituting x_2 and x_3 into R_1: $4x_1 - 3.2 = 2 \rightarrow x_1 = n = 1.3$.

Solution by software

MATLAB code

```
%Recall from Tutorial 2:
%   4*n - q - 2=0;     % from the P-balance
%   2*p - 3*q - 16=0; % from the H-balance
%   p - 4*q=0;         % from the O-balance
%Now rewrite the above eqautions in the form Ax=b

A=[4  0   -1;0  2   -3; 0  1   -4]; %mtrix of coefficients

b=[2;16;0]  %must be a column vector

tic
x=mldivide(A,b);   %or A\b
toc                %Elapsed time is 0.000024 seconds

fprintf('\n n=%d\n p=%d\n q=%d\n',x(1),x(2),x(3))
%Solution
%  n=1.300000e+00
%  p=1.280000e+01
%  q=3.200000e+00
% Time take: 0.000024 seconds
```

Python code

```
# Recall from Tutorial 2:
#   4*n     -    q - 2  = 0   from the P-balance
#       2*p - 3*q - 16 = 0   from the H-balance
#         p - 4*q      = 0   from the O-balance

import numpy as np
```

```
from numpy.linalg import *
import time as time

A = np.array([[4, 0, -1], [0, 2, -3], [0, 1, -4]]);
b = np.array([2, 16, 0])

start = time.time()
x = solve(A, b);
ttaken = time.time()-start

print('Solution is: %s; calculated in %f seconds' % (x, ttaken))
# Solution is: [  1.3  12.8   3.2]; calculated in 0.000436 seconds
```

The time required to solve the above system using built-in $Ax = b$ solvers is much less than that required by the brute-force approach in Tutorial 2.

Question 2 [1.5]

When it comes down to the basics, any recipe can be reduced to 4 components: fat, carbohydrates, protein and moisture (usually water). What differentiates each recipe though is the list of ingredients which are combined, and the ratio of fat:carbohydrate:protein:water. Of course how the ingredients are mixed and the reactions that take place during cooking are also important, but that affects mainly taste and texture, not nutritional value.

Let's consider a biscuit recipe for this question. One can use butter for the fat component, use brown sugar and chocolate for the carbohydrate component (chocolate also contributes to the fat component), flour adds to the carbohydrate component also and somewhat to the protein component, while eggs, milk or water make up the rest of the recipe for protein and moisture.

We would like to blend raw materials where the ratio of fat:carbohydrate:protein:water of the uncooked ingredients is 20:50:10:20. We have the following ingredients available, broken down on a mass percentage basis.

Component	Butter	Lard	Flour	Sugar	Whole milk	Egg	Chocolate
Fat	85	81	0	0	4	10	14
Carbohydrate	0	1	86	98	6	1	75
Protein	1	4	10	0	4	35	6
Moisture	14	14	4	2	86	54	5

1. Write down the 4 mass balance equations, one for each component, in the form $Ax = b$, where x is vector that represents the mass of each ingredient to be used, a vector with 7 rows and one column.
2. What is the technical (mathematical) reason why we cannot solve this system of equations using Gauss elimination? Translate your mathematical answer to English, explaining it in terms of the recipe.
3. If we are constrained to using only butter, flour, whole milk and eggs: solve, using LU decomposition on the computer, for the amounts of each to be used, in grams, to obtain 100 grams of uncooked ingredients, in the required ratio.

If you prefer, you can interpret this question in the context of rubber manufacturing, where various oils, polypropylene and existing rubber materials are blended to create a new rubber with desired physical properties.

Solution

1. For a 100 units of the recipe, we should have 20 units of fat, 50 units of carbohydrate, 10 units of protein, and 20 units of moisture. Given the percentage of these components in each of the ingredients, the four mass balance equations can be written as:

Fat	$85x_1 + 81x_2 + 0x_3 + 0x_4 + 4x_5 + 10x_6 + 14x_7 = 20$
Carbohydrate	$0x_1 + 1x_2 + 86x_3 + 98x_4 + 6x_5 + 1x_6 + 75x_7 = 50$
Protein	$1x_1 + 4x_2 + 10x_3 + 0x_4 + 4x_5 + 35x_6 + 6x_7 = 10$
Moisture	$14x_1 + 14x_2 + 4x_3 + 2x_4 + 86x_5 + 54x_6 + 5x_7 = 20$

The above equations can be written in form $Ax = b$ as

$$\begin{bmatrix} 85 & 81 & 0 & 0 & 4 & 10 & 14 \\ 0 & 1 & 86 & 98 & 6 & 1 & 75 \\ 1 & 4 & 10 & 0 & 4 & 35 & 6 \\ 14 & 14 & 4 & 2 & 86 & 54 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 20 \\ 50 \\ 10 \\ 20 \end{bmatrix}$$

where x_i represent the mass of the 7 ingredients from butter to chocolate, respectively.

2. The above system cannot be solved for x because the number of unknowns is **not** equal to the number of equations. In other words, the degrees of freedom is not zero, and we must have either more equations or less unknowns to be able to solve this system. In terms of the recipe, there are infinite combinations of the ingredients that yield the desired component ratio. So, there is no unique recipe given only the above information.

3. If we limit ourselves to the use of only butter, flour, whole milk and eggs, we imply that the mass of the other three ingredients is zero in our recipe. So, we have eliminated three unknowns by fixing them to zero. Now, we have four unknowns in four equations. This system is now solvable. The new system will only have the coefficients and unknowns corresponding to butter, flour, whole milk and eggs as:

$$\begin{bmatrix} 85 & 0 & 4 & 10 \\ 0 & 86 & 6 & 1 \\ 1 & 10 & 4 & 35 \\ 14 & 4 & 86 & 54 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 20 \\ 50 \\ 10 \\ 20 \end{bmatrix}$$

Recall from class that $A = LU$, but with a permutation or pivot matrix, it becomes $PA = LU$, or alternatively: $A = P^{-1}LU$

$Ax = b \rightarrow P^{-1}LUx = b$. Let us $y = Ux$.

Then solve $(P^{-1}L)y = b$ for y. Finally, solve $Ux = y$ for x. It can be implemented as follows:

MATLAB code

```
A = [85, 81,  0,  0,  4, 10, 14; ...
      0,   1, 86, 98,  6,  1, 75; ...
      1,   4, 10,  0,  4, 35,  6; ...
      14, 14,  4,  2, 86, 54,  5]

A = A(:,[1, 3, 5, 6]);   % choose columns corresponding to
                         % the four unknowns

b = [20, 50, 10, 20]';   % vector of ratios (Ax=b)

[L,U,P] = lu(A);         % LU decomposition of A=inv(P)*L*U
                         % (P is the permutation matrix)

y = mldivide(inv(P)*L,b);    % Solve for y=Ux

x = mldivide(U,y)*100    % Solve for x, and multiply by 100
                         % to scale  the results for 100 grams
                         % of the uncooked ingredients.
```

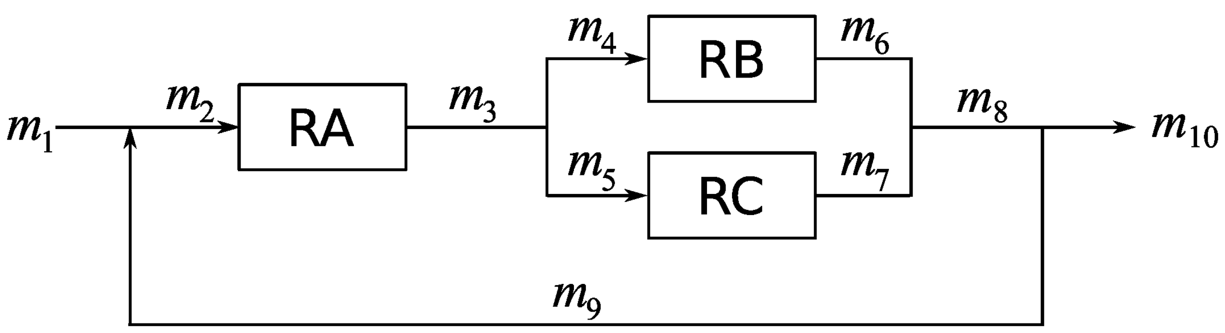
```
fprintf('\nbutter=%f\n flour=%f\n whole milk=%f\n eggs=%f\n',...
    x(1),x(2),x(3),x(4))

% butter=21.813184
% flour=57.284641
% whole milk=10.523732
% eggs=10.378442

Total_mass = sum(x)      % Total mass = 100
```

Python code

```
import numpy as np
from numpy.linalg import *
from scipy.linalg import *

A = np.array([[85, 81,  0,  0,  4, 10, 14],
              [ 0,  1, 86, 98,  6,  1, 75],
              [ 1,  4, 10,  0,  4, 35,  6],
              [14, 14,  4,  2, 86, 54,  5]])

# Selected certain columns of A
A = A[:,[0,2,4,5]]
b = np.array([20, 50, 10, 20])

# LU decomposition
LU, P = lu_factor(A)
x = lu_solve((LU, P), b) * 100
print(' Butter=%g\n Flour=%g\n Whole milk=%g\n Eggs=%g\n' %\
        (x[0], x[1], x[2], x[3]))

# Butter=21.8132
# Flour=57.2846
# Whole milk=10.5237
# Eggs=10.3784

# Verify with the "solve" command:
x_solve = linalg.solve(A, b) * 100
print(x_solve)                          # same as "x"
```

Question 3 [1.5]

In class we derived the molar balance for a species in a similar flowsheet. Use the slightly modified flowsheet below



and let α be the proportion of m_3 that is sent to reactor RB, and the rest, $(1-\alpha)m_3$, is sent to reactor RC. Similarly, let β represent the proportion of m_8 that leaves the system, and let $(1-\beta)m_8$ be the amount sent to the recycle stream. Also let X_A be the conversion of the species in reactor RA, and similarly for X_B and X_C.

1. Write a MATLAB or Python function that will return a matrix A given the five inputs, α, β, X_A, X_B, and X_C. This matrix A represents the coefficients in the molar balances on the flowsheet, and these balances can be solved using $Ax = b$.
2. Given that m_1 = 10 mol/second, and that the molar conversion of the inlet stream to reactor A is 5%, i.e. $X_A = 0.05$, $X_B = 0.60$ and $X_C = 0.85$. What are the molar flow rates, m_6, m_7, m_9, and m_{10} when:
   - $\alpha = 0.2, \beta = 0.7$
   - $\alpha = 0.2, \beta = 0.4$

   Are your answers reasonable?

   You should use either MATLAB or Python's built-in functions to solve the system of equations.

Solution

1. The feed stream, m_1, is the molar flow rate of the species of interest. A molar balance can be written around each reactor, as well as split and each junction point:

   $m_1 =$ some given value mol/s

   $m_2 = m_1 + m_9$

   $m_3 = (1 - X_A)m_2$

   $m_4 = \alpha m_3$

   $m_5 = (1 - \alpha)m_3$

   $m_6 = (1 - X_B)m_4$

   $m_7 = (1 - X_C)m_5$

   $m_8 = m_6 + m_7$

   $m_9 = (1 - \beta)m_8$

   $m_{10} = \beta m_8$

Rearranging the above equations to have all unknowns on one side, and all known values on the other side gives:

$m_1 =$ some given value

$-m_1 + m_2 - m_9 = 0$

$-(1 - X_A)m_2 + m_3 = 0$

$-\alpha m_3 + m_4 = 0$

$-(1 - \alpha)m_3 + m_5 = 0$

$-(1 - X_B)m_4 + m_6 = 0$

$-(1 - X_C)m_5 + m_7 = 0$

$-m_6 - m_7 + m_8 = 0$

$-(1 - \beta)m_8 + m_9 = 0$

$-\beta m_8 + m_{10} = 0$

If we have the values of the parameters α, β, X_A, X_B, and X_C, then we can write a function that accepts these input arguments and whose output is the matrix A. This function, and the code that calls it later on can be implemented as shown below. With the given parameter values, the molar flow rates can be solved easily. First we have to evaluate the matrix A by calling the function `matrix_A`, and providing the values of the parameters. Then, the system is solved for m_i.

MATLAB code

```
% NOTE: the code below must appear in a file called "matrix_A.m"

function A = matrix_A(XA, XB, XC, alpha, beta)

%   Calculates and returns the A matrix that represents the
%   reactor network.

%   A has 10 rows (equations) and 10 columns (for unknowns m1,
%   m2 ... m10) for the given equations

    A =[1,       0,          0,       0,       0,   0,   0,         0,   0,   0
        1,      -1,          0,       0,       0,   0,   0,         0,   1,   0
        0,  (1-XA),         -1,       0,       0,   0,   0,         0,   0,   0
        0,       0,      alpha,      -1,       0,   0,   0,         0,   0,   0
        0,       0,  (1-alpha),       0,      -1,   0,   0,         0,   0,   0
        0,       0,          0,  (1-XB),       0,  -1,   0,         0,   0,   0
        0,       0,          0,       0,  (1-XC),   0,  -1,         0,   0,   0
        0,       0,          0,       0,       0,   1,   1,        -1,   0,   0
        0,       0,          0,       0,       0,   0,   0, (1-beta),  -1,   0
        0,       0,          0,       0,       0,   0,   0,     beta,   0,  -1] ;

end

% NOTE: the rest of this code must appear in another m-file, for
%       example, "tut4_q3.m"

alpha = 0.2;
beta = 0.7;
XA = 0.05;
XB = 0.60;
XC = 0.85;

A= matrix_A(XA,XB,XC,alpha,beta);      % Call the function
b = [10, 0,0, 0, 0, 0, 0, 0, 0, 0]'    % Transpose to get
                                       % a column vector

% Solve the linear system:
m = A\b

% Results:
% m =  [10.0000, 10.6045, 10.0742, 2.0148, 8.0594, 0.8059,
%        1.2089, 2.0148,    0.6045, 1.4104]
```

Python code

```
# NOTE: with Python, all this code can go in a single file

import numpy as np
from numpy.linalg import *

def matrix_A(alpha, beta, XA, XB, XC):
    """
    Calculates and returns the A matrix that represents the reactor network.

    A has 10 rows (equations) and 10 columns (unknowns m1, m2 ... m10) for
    the given equations.
    """

    A = np.array([[1,       0,          0,       0,       0,   0,   0,         0,   0,   0],
                  [1,      -1,          0,       0,       0,   0,   0,         0,   1,   0],
                  [0,  (1-XA),         -1,       0,       0,   0,   0,         0,   0,   0],
                  [0,       0,      alpha,      -1,       0,   0,   0,         0,   0,   0],
                  [0,       0,  (1-alpha),       0,      -1,   0,   0,         0,   0,   0],
                  [0,       0,          0,  (1-XB),       0,  -1,   0,         0,   0,   0],
                  [0,       0,          0,       0,  (1-XC),   0,  -1,         0,   0,   0],
                  [0,       0,          0,       0,       0,   1,   1,        -1,   0,   0],
                  [0,       0,          0,       0,       0,   0,   0,  (1-beta),  -1,   0],
                  [0,       0,          0,       0,       0,   0,   0,      beta,   0,  -1]])
    return A

# Python knows that the above function ends at this point, because we
# remove the 4-space indent from here onwards.

alpha = 0.2;
beta = 0.7;
XA = 0.05;
XB = 0.60;
XC = 0.85;

b = np.array([10, 0,0, 0, 0, 0, 0, 0, 0, 0])
A = matrix_A(alpha, beta, XA, XB, XC)
# Solve the linear system
x = solve(A, b)
print(x)
# [ 10.          10.60445387  10.07423118   2.01484624   8.05938494
#    0.80593849   1.20890774   2.01484624   0.60445387   1.41039236]
```

The solution for the two scenarios above differ only in terms of the β value.

- $\beta = 0.7$ is $x = [10.0, 10.6, 10.1, 2.01, 8.06, 0.806, 1.21, 2.01, 0.604, 1.41]$
- $\beta = 0.4$ is $x = [10.0, 11.3, 10.7, 2.14, 8.58, 0.858, 1.29, 2.14, 1.29, 0.859]$

It shows that as β decreases (i.e. we recycle more of our stream m_8) that the amount of the species leaving the overall reactor decreases from 1.41 to 0.859. This is expected, because we know that as we increase the recycle rate, that a greater amount of unconverted material will be reacted. So even this relatively simplistic model makes engineering sense.

Use the above code and see what happens when you let $\beta = 0.0$ or $\beta = 1.0$: do the results still make sense?
