An Introduction to Programming in MatLab with Applications in Chemical Engineering

R Rawatlal
School of Chemical Engineering
University of KwaZulu-Natal
May 2004

Programming with MatLab – Chemical Engineering Applications

University of Natal (Durban), School of Chemical Engineering, R Rawatlal, February 2004

Variables in MatLab

1. What is MatLab?

MatLab is probably the most popular simulation software used by engineers. The word 'MatLab' comes from Matrix Laboratory because MatLab is well suited to vector-matrix type calculations – it run fastest when the commands are in this form, rather than by the use of 'for' loops. [Example – find $\mathbf{c} = \mathbf{a} \times \mathbf{b} = \sum_j a_{ij} b_{ji}$ where **a** and **b** are matrices] As such, it is possible to write programs in MatLab that are very much closer to the equations of the process you are trying to simulate, than almost all other programming languages. [extend example above]

2. Command window vs. editor

In the MatLab command window, instructions are executed immediately. The user has full access to all the variables in the memory in the command window. [Extend example above] However, we usually wish to store all our commands in the form of a program, and this is accomplished with a MatLab file or 'm-file'.

An m-file contains MatLab-code instructions, and is of two types: script files and function files. A script file contains exactly the code that could be typed into the command window. The same results are generated. So script files may be regarded merely as collections of command window instructions that are collected in a file for the purpose of storage and convenience. [Example]

Functions

On the other hand, function files do not have access to the variables in the memory (unless special global variables are used… more about that later) and usually require arguments before they are executed. The variables used within a function cannot be accessed globally. The first line of the function file has the form: `[y1,y2] = function func_name(x1,x2)`
Example: temperature conversion

Vectors

… are similar to arrays; collections of numbers in a single variable name. The elements of the array may be accessed by using an index. For instance, we might assign the vector 'v' as follows: v = [10 20 30]. As an example, we note that the second element of vector v is 20. To access this value, we use the command v(2).

By default, MatLab creates row vectors; if you preferred that v be a column vector, then you would use the command w=[10; 20; 30] – the semi-colons tell MatLab to go to the next row. Alternatively, you could transpose the row vector as follows: w = [10 20 30]'

Matrices can just as easily be created:
The command: M = [10 20 30; 40 50 60]
…creates the following matrix:

$$M = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{bmatrix}$$

when testing programs, the 'rand(x,y)' command is very useful. The function 'rand' will generate a matrix of dimensions [x,y] and fill it with randomly generated numbers. For instance, the command
M = rand(2,3) generated:

```
M =
   0.9218    0.1763    0.9355
   0.7382    0.4057    0.9169
```

Vector Operations

The MatLab language has been created in such that mathematical vector matrix calculations can be performed easily and in a very natural way. For instance, consider the vector multiple of row vectors ***A*** and ***B***. Mathematically, we may write:

$$A.B^T = \sum_i A_i B_i \qquad (1)$$

To obtain such a result in a language like Visual Basic or C++, we would need the following commands:

```
AB = 0
For i=1:N
        AB = AB+ A(i)*B(i)
End;
```

In MatLab, however, this is accomplished much more simply, with the single command:

```
AB = A*B'
```

[Note that the ' mark transposes a vector or matrix.]

If ***A*** is a matrix and ***b*** is a vector, then the vector multiple may be stated mathematically as:

$$[A.b]_i = \sum_j A_{ij} b_j \tag{2}$$

To obtain such a result in a language like Visual Basic or C++, we would need the following commands:

```
For i = 1:M
        Ab(i) = 0
        For j=1:N
                Ab(i) = Ab(i)+ A(i,j)*b(j)
        End;
End;
```

In MatLab, however, this is accomplished much more simply, with the single command:

```
Ab = A*b
```

Five lines of code visual code vs. 1 line of MatLab!

Scalar Operations

Sometimes, we wish to perform operations on the individual elements of the array, rather than performing vector-matrix type calculations. For instance, given vector v = [10 20 30], we might wish to divide all the element by 10. This is handled in a natural way in MatLab:
w = v/10 … results in:
w = [1 2 3]
Now what if we wish to multiply each of the elements in v by each in w. We cannot use the command v*w, since that is the vector multiplication, which will multiply each element, and then add them all up. To multiply elements in the scalar sense, we must use a dot (.) before the operation sign: v.*w results in [10 40 90]. Similarly, v./w = [10 10 10].

It should be obvious that in a scalar operation, the two variables involved must have the same dimensions.

Plotting 2-D graphs

The command 'plot(x,y,c)' plots on the Cartesian plane vector y against vector x in colour c.

Example
Given components A and B, having molar masses [20, 40]x10^{-3}kg.mol^{-1} respectively, generate a graph that shows the variation of mole-fraction-A vs mass-fraction-B.

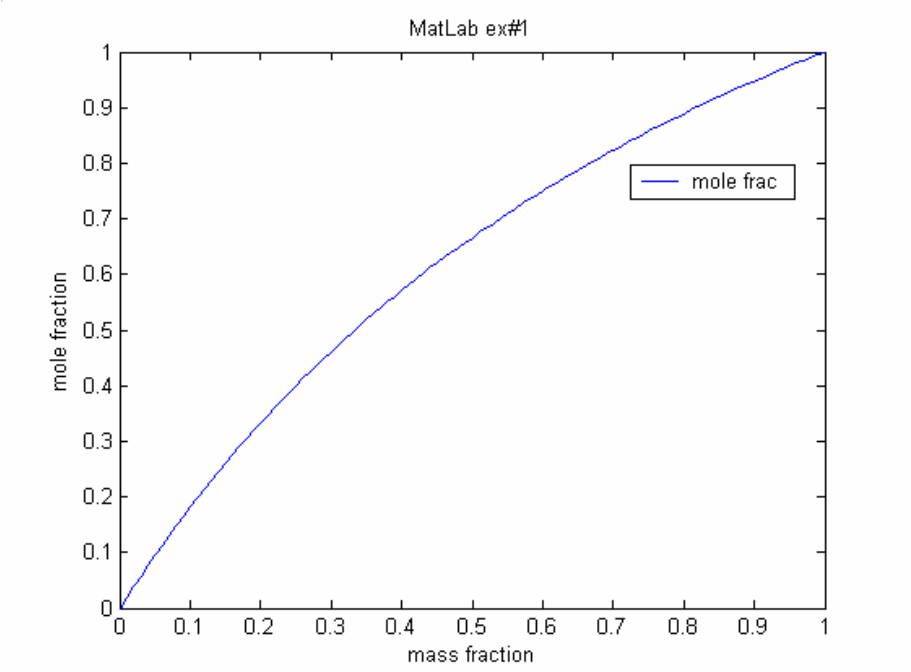
Soln: the relation between mole and mass fraction is given by:

$$x_A = \frac{X_A / MM_A}{X_A / MM_A + (1 - X_A) / MM_B} \tag{3}$$

Solution:

```
X = 0:0.01:1;
MMa = 20e-3; MMb = 40e-3;
x = (X/MMa)./(X/MMa + (1-X)/MMb)
plot(X,x)
xlabel('mass fraction');ylabel('mole fraction')
legend('mole frac'); title('MatLab ex#1')
```

… which generates:



Ex: generate an enthalpy vs T graph from 0K to 1000K for data given in class. Func form.

Programming with MatLab – Chemical Engineering Applications

University of Natal (Durban), School of Chemical Engineering, R Rawatlal, February 2004

Generating combinations of variables; Visualizing results

We extend our knowledge of the scalar and vector operations available in MatLab in this section.

In engineering applications, we often wish to generate results for a widely varying set of conditions. For instance, if we have a reaction rate given in the form:

$$r_A = kC_A(1-C_A)^2 \text{ where } k = k_0 \exp\left(-\frac{E}{RT}\right) \tag{4}$$

… then it is obvious that the rate depends strongly on the temperature as well as the concentration of component-A, and it would help our understanding if we could observe how the rate changed for changes in both variables.

MatLab has various functions that are well suited to this task. In solving the problem given above, we expect to use two commands to generate the function r_A:

```
k = k0*exp(-E/R/T)
rA = k*CA/(1-CA)^2
```

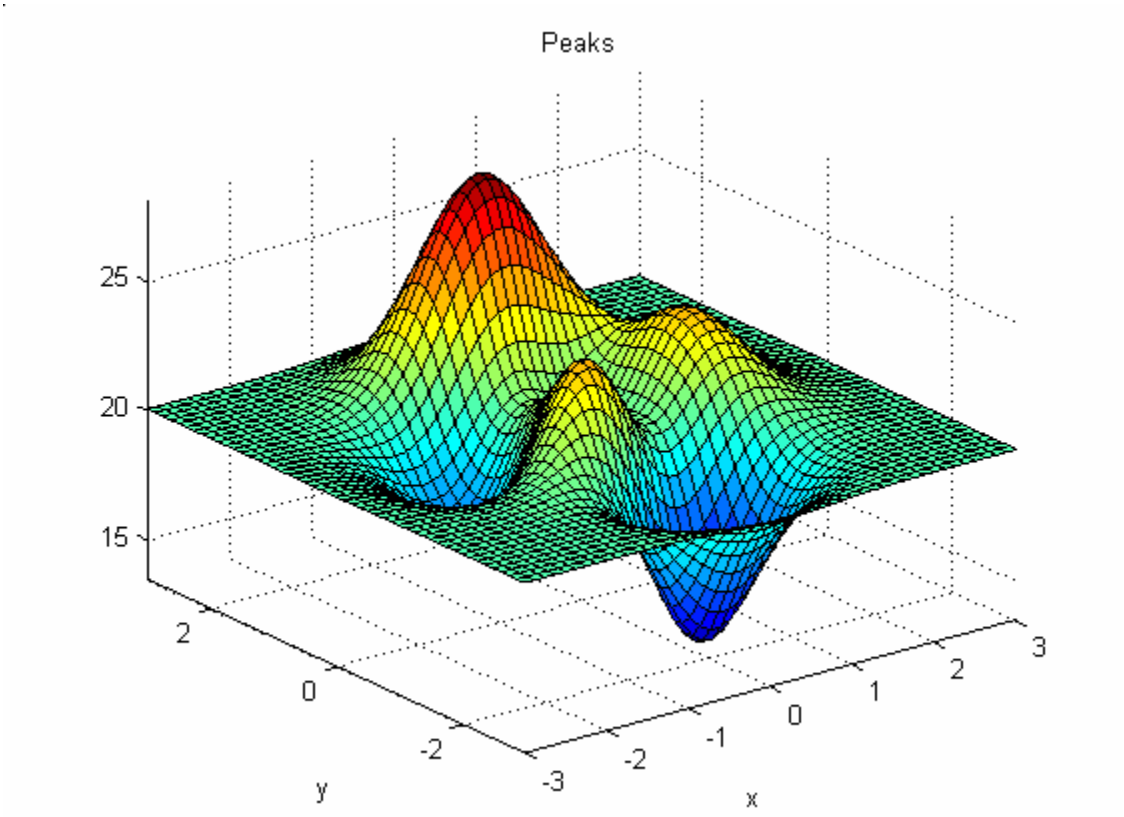
How can we generate rA for simultaneous variations in both T and CA?
A possible answer: Generate a matrix of rA values, where T varies along the row and CA varies on the column.

The ‘linspace’ command is useful in generating vectors of linearly spaced data.
x = linspace(x0,xf,n) creates a vector x that has x0 as the 1st element and xf as the last, with there being n linearly spaced elements. Therefore, x = linspace(0,1,5) creates:
x = [0 0.2500 0.5000 0.7500 1.0000]
…so by default, linspace creates row vectors. To create our temperature column vector, we then use: Ts = linspace(200,400,30)’ [we are choosing temperature to vary from 200 to 400K through 30 linearly spaced points] Similarly, we use CAs = linspace(0,1,32). [Using 32 [points just to make it different from 30.]

How do we convert these vectors to matrices? Ans: The ‘repmat’ command. The command repmat(x,[xz1 xz2]) takes vector or matrix ‘x’ and replicates it in the 1st dimension (the row) by amount xz1 and in the 2nd dimension (the column) by xz2. For instance, the command y = repmat([10 20 30],[3 2]) generates:

10	20	30	10	20	30
10	20	30	10	20	30
10	20	30	10	20	30

To recap what we are doing: We wish to consider how the reaction rate varies with both temperature as well as concentration. What we could do is generate a matrix of reaction rate values with temperature varying along the rows of the matrix and concentration varying along the y-axis. Since three variables are involved, with equation (4) being a relationship between the 3 varbs, we expect a surface in 3-dimensions to result.



The surface shown here is arbitrary; we wish to generate the surface corresponding to equation (4). First, we define the numbers that wish to put in; let's say that we are interested in temperatures in the range 400 to 500 K and concentrations 0 to 0.95 mol.m^{-3}. Assume that k_0 and E are 5 and 2x10^4 respectively. As such the following commands will generate the vectors for T and C:

k0 = 5; E = 2e4;	[store these values for later use]
CZ = 30; C = linspace(0,1,CZ)';	[generates a column vector of length 30]
TZ = 35; T = linspace(400,500,TZ);	[generates a row vector of length 35]

Now we wish to all possible values of *r* for the values of *T* and *C* that we are interested in. To do this, we could fix a value of T, allow C to vary, and see how *r* changes. Then, we consider another value of T, allow C to vary in another way and once more see how r varies. Let's look at the simple case of three concentration values of [0.2 0.5 and 0.8] and four temperature values of [420 440 460 and 480]. To generate all possible values of r, we could fix C=0.2 and let T vary as [420 440 460 and 480]. By doing this, we generate four values of r (using eq. (4)). Next we consider fixing C at 0.5 and again letting T vary as [420 440 460 and 480]. And finally, we use C = 0.8 and letting T vary as [420 440 460 and 480].

How to do this in C++? Answer:

```
C = [0.2 0.5 0.8];
T = [420 440 460 480];
For i = 1:3
    For j = 1:4
        r(i,j) = k0*exp(-E/R/T(j))*C(i)*(1-C(i))^2;
    End;
End;
```

Here, five lines of code are needed. Can we handle the problem more simply by the use of matrices? Let C be:

0.2 0.2 0.2 0.2
0.5 0.5 0.5 0.5
0.8 0.8 0.8 0.8
… and T be:
420 440 460 480
420 440 460 480
420 440 460 480

Then if we were to take the product C.T, we would have a number that results from C varying along the row and T varying along the column, while generating all possible values of that product in the ranges specified. To accomplish this for the reaction rate, we use:

```
CM = repmat(C,[1 TZ]);
TM = repmat(T,[CZ 1]);
kM = k0*exp(-E./R./TM);
r = kM.*CM.*(1-CM).^2;
```

… which generates all possible values of the variable r with respect to C and T.

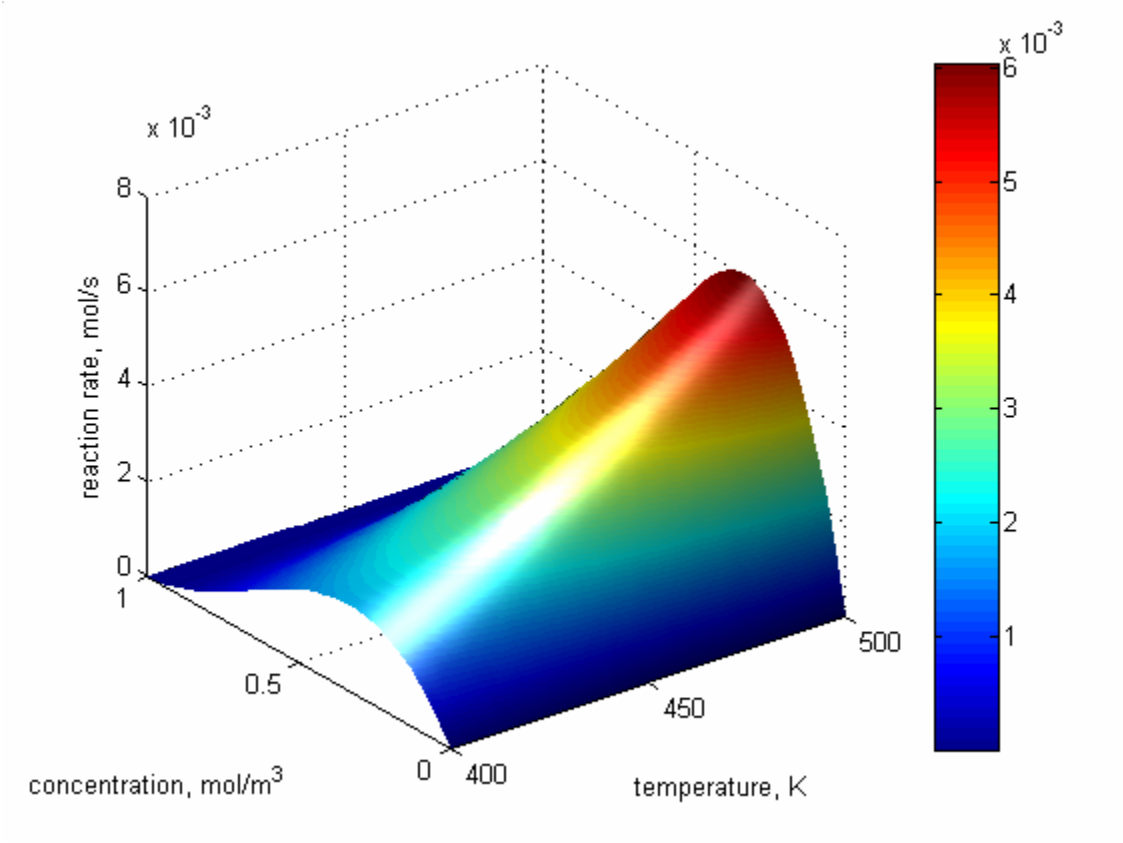
[explain why the vectors in the repmat function are as they are]

The ‘surf” command is used to plot 3-D surfaces. The command surf(x,y,z) generates a surface with matrices x, y and z on the x, y, and z axes respectively.

[Q: why must they be matrices?]

The commands to plot the rate are then given by:
surf(TM,CM,r)
xlabel('temperature, K')
ylabel('concentration, mol.m^3')

which generates:



Exercise: Consider an ideal gas in a closed vessel. The vessel is constructed such that we can vary the temperature and pressure of the gas while keeping the vessel closed [no material exchange]. How does the concentration of the gas vary if we are interested in pressure varying from 1 to 2 atm and temperature 400 to 500? Produce a 3-D graph depicting this.

Programming with MatLab – Chemical Engineering Applications

University of Natal (Durban), School of Chemical Engineering, R Rawatlal, February 2004

Data Regression and Symbolic Variables

Introduction

So far we have learned how to generate and store data in vector and matrix variables. When using this data to solve larger problems, we must become familiar with techniques of data manipulation. For instance, data may be available as a set of numerical values, whereas we might wish to have an equation to represent that data. On the other hand, if we need to interpolate the available data to specific regions of interest, the technique by which this interpolation is performed can affect the final results, and so we must examine the methods available.

Fitting Equations to Data

Let's talk about this in the context of an experiment. Say we performed an experiment that relates the temperature of a metal to its thermal conductivity. We have taken readings of conductivity for the temperatures T = [200, 220, 250, 275, 310, 315, 325, 350] – the data is not evenly spaced. The resulting conductivity is given by: k = [0.15, 0.17, 0.19, 0.22, 0.26, 0.29, 0.32, 0.37]

In MatLab, it is very easy to fit data of this sort to polynomial functions. The single command p1=polyfit(T,k,n) instructs MatLab to take the two vectors T and k as the x and y values respectively and to fit a polynomial of order n such that p1 is a vector of the resulting polynomial parameters. For example, the command p1 = polyfit (T, k, 4) tells MatLab to find a vector p1 such that $y = p1(1)*x^3 + p1(2)*x^2 + p1(3)*x + p1(4)$. In other words, fit a cubic equation to the data. Figure 1 shown on the next page gives us an idea of how well the equation fits the data. It's not a bad fit, but we suspect we can do much better. By using p1 = polyfit (T, k, 7), we increase the polynomial order to 7, and at first glance, we get a much better data fit, as shown in Figure 2.

Be extremely careful!!! The first two figure are misleading because we have only looked at the data points for which the experiment was conducted. If we use the polynomials to generate the values of conductivity at intermediate temperatures as well, we find that the 7^{th} order polynomial oscillates badly, and would actually give a very poor estimate of conductivity for temperatures other then the those near the points where it was fitted. Actually, it turns out that the humble 4^{th} order polynomial was the better choice. This is not always the case; you must choose the order of the polynomial by balancing overall accuracy against fitting accuracy.

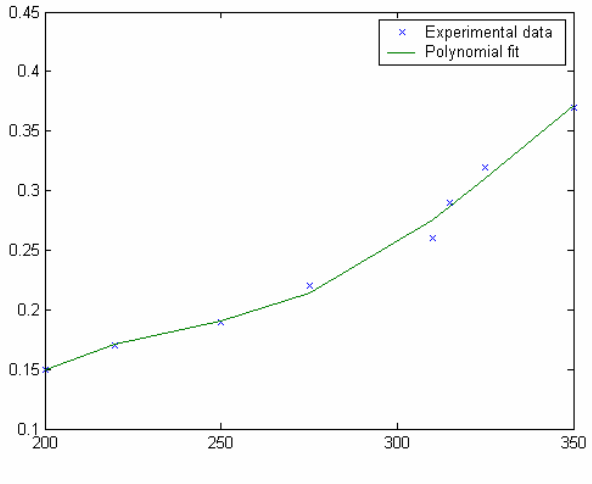


Figure 1

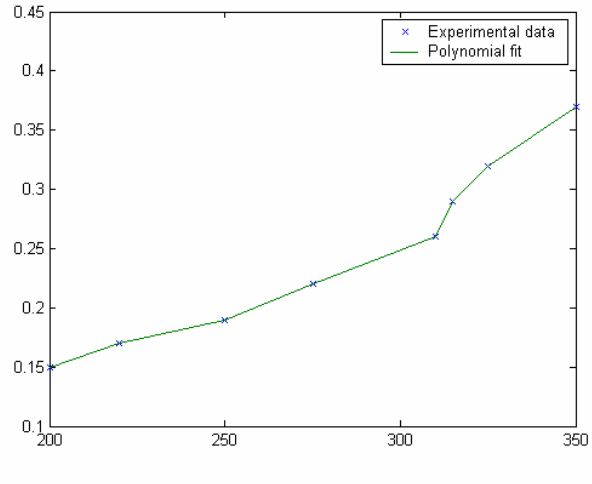


Figure 2

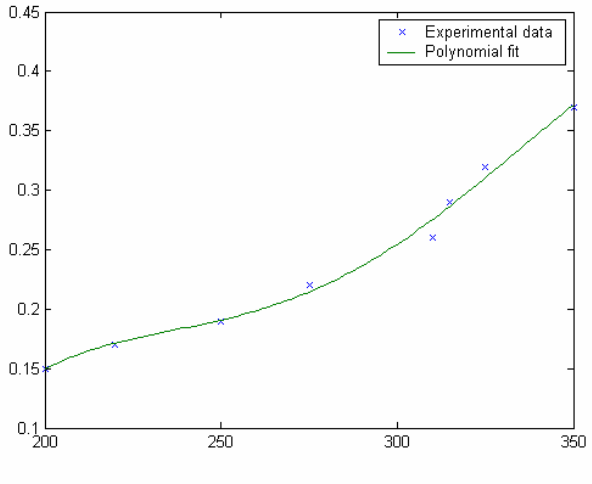


Figure 3

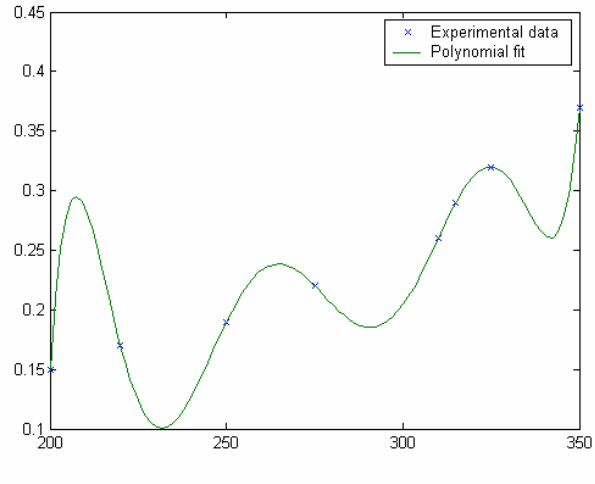


Figure 4

The topic of fitting data to equations falls under the general field of data regression, where we consider fitting data to any mathematical 'model'. [empirical vs fundamental models vs hybrid models] For this course, we will consider just polynomial functions as our 'models'.

Exercise:

Taking the data for the enthalpy vs. temperature graph generated in exercise #1 (see lesson #1), fit a polynomial that estimates the data to a reasonable degree of accuracy.

Interpolation

The command "yi = interp1(x,y,xi)" performs a 1-dimensional interpolation when given vectors x and y; it finds the value of y corresponding to and x value of xi. Let's take the data given in the example discussed earlier. Suppose we want to take the conductivity values at the eight data points and use these to interpolate for a finer range of temperature values than the eight data points used in the experiment.

The function interp1 can accept a vector of interpolation values for the variable xi. As such, we may chose a very fine set of temperatures, say T2 = linspace(200, 350, 1000) [that is, use 1000 data points between the temperature values 200 and 250K]. The command to generate the required data points is then given by: k2 = interp1(T,k,T2). The data is visualized by the command "plot(T,k,'x',T2,k2)", which generates Figure 5. We note that we don't need to fit any functions to the data points; interp1 without any special commands just finds the straight line interpolation between the two closest points of interest.

This is not always sufficient; sometimes, we would like to 'smoothen' the data so as to have nice smooth derivatives at the data points. [In Figure 5, the derivatives are not continuous] Fortunately, we may use cubic spline interpolation [revision of CS concept needed?] by specifying the method in the interp1 command as: k2 = interp1(T,k,T2,'splines'), which results in the much smoother graph of Figure 6, shown below:

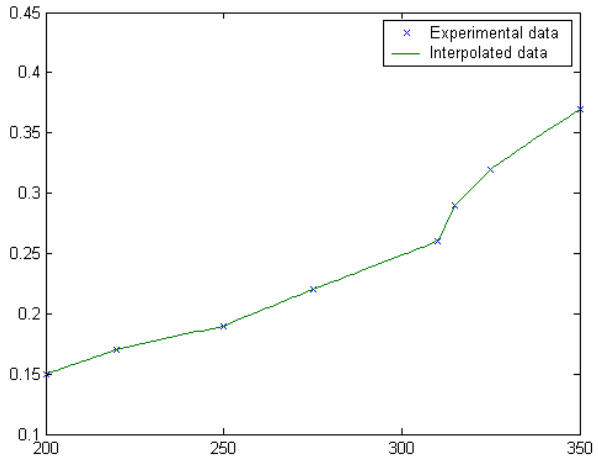


Figure 5

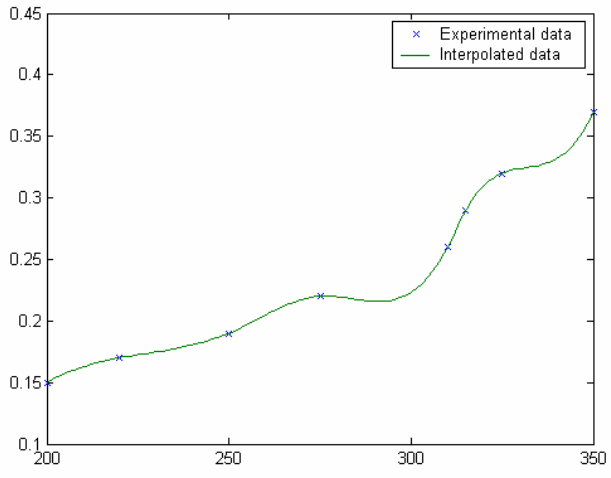


Figure 6

Programming with MatLab – Chemical Engineering Applications

University of Natal (Durban), School of Chemical Engineering, R Rawatlal, February 2004

Symbolic Manipulation, Numerical Differentiation and Integration

---

Symbolic Variables

An exciting addition to the MatLab engine is the symbolic toolbox. This module enables the user to perform symbolic manipulations such as the typical algebraic, but also differentiation an, where a solution exists, integration as well.

A simple example: differentiate the function $f(x) = 3x^2$.
Matlab code: syms x; diff('3*x^2')
[the "syms " command declares x to be a symbolic variable; "diff" differentiates a function]

Differentiate the more complex function: $f(x) = 3\ln\left(\frac{1+x^5}{x^2}\right)$

We use the commands: diff('(x+1)/log((x-5)^3)'), which yields a result
1/log((x-5)^3)-3*(x+1)/log((x-5)^3)^2/(x-5)

The result is a little more complex, and it requires some skill in transferring this single line to an understandable equation form. However, MatLab has some additional functions to aid us. "Pretty" will help to represent the expression by in a prettier form – in an expression closer to the understandable equation; hence, the command "pretty(diff('(x+1)/log((x-5)^3)'))" generates

```
        1                x + 1
------------ - 3 ----------------------
          3                3 2
log((x - 5) )    log((x - 5) )  (x - 5)
```

… which we can easily understand to mean:

$$\frac{df(x)}{dx} = \frac{1}{\ln(x-5)^3} - 3\frac{x+1}{\left(\ln(x-5)^3\right)^2 (x-5)}$$

Finally, the expression may also be further simplified by the use of the 'simple' command. MatLab attempts, by various expansions and factorizations, to further simplify your equation.

Exercise: Use the following eqn. relating saturation pressure as it varies with temperature [Antoine eqn in integ form] to generate a 15 element vector. Find a suitable polynomial expression for it. Show graphically it fits well.

Introduction to Numerical Differentiation and Integration

By using the commands discussed so far, we are able to able to perform arithmetic operations on large sets of data. However, when solving the equations commonly encountered in chemical engineering, we must also be able to perform differentiation and integration as well. In this section, we consider how these operations may be efficiently performed in MatLab.

Referencing Data Elements

When given a vector x = [2 4 6 8 10 12 14 16 18 20], it is possible to determine the value of any element of in vector by specifying the index. For instance, to extract the 3rd element, we use x(3), which yields a result of 6. What if we wish to extract an entire section of the vector? Ans: we can use a vector of indices as our references. For instance, we may desire all elements from the 4th to the 8th positions. The command x(4:8) extracts these elements. Alternatively, we may use another variable: i = 4:8; x(i)

The 'end' command is also quite useful in determining the last element in an array. For instance the command x(end) will pick out the last number (20) in the vector x above. To pick out all elements from the 7th to the end, say, we use x(10:end) which then returns 14 16 18 20.

We shall use this principle when determining derivatives and integrals in MatLab.

Numerical Differentiation

Assume that we have two variables, x and y, where it is known that y is a function of x. We don't know the exact functional relationship between y and x, but we have vectors of the values of y and x. We are interested in determining the derivative of y with respect to x by the use of these vectors.

We note first that the derivative may be approximated as follows:

$$\frac{dy}{dx} \simeq \frac{\Delta y}{\Delta x} = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \quad \text{or} \quad \frac{dy}{dx} \simeq \frac{\Delta y}{\Delta x} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{5}$$

In terms of the vectors, the subscript i may be considered as the index into the vector. Let us say that we have a vector of size N (N elements in the vector). To obtain the vector y_{i-1}, what values can i assume? Ans: only values greater than 1 and less than or equal to N. Otherwise, when i = 1, $y_{i-1} = y_0$, which does not exist – only y_1 onwards exists. Therefore, i must start from 2, and our vector i is then given by i = 2:N. On the other hand, if we wish to generate y_{i+1}, what

must i look like? In this case, i = 1 is valid, since $y_{i+1} = y_2 = y(2)$, which exists. However, i=N does not exist, since $y_{i+1} = y_{N+1} = y(N+1)$ does not exist – there are only N elements in y.

Therefore, to generate the derivative using 'left-hand' differences, we use the following commands:

```
i=2:N;
dydx = (y(i) – y(i-1)) ./ (x(i) – x(i-1))
```

Or, using right-hand differences, we write:

```
i=1:N-1;
dydx = (y(i+1) – y(i)) ./ (x(i+1) – x(i));
```

There are more accurate techniques of evaluating derivatives (see other courses). We wish here merely to illustrate the principles by which derivatives may be determined. The derivative may also be obtained analytically by the use of symbolic variables [see previous sections]. Class ex: compare the results of numerical and analytical integration in MatLab.

Integration

Let's say have a known function f(x) and we wish to determine it's integral: $F = \int_a^b f(x)dx$. A common approximation to this integration can be represented by:

$$F = \int_a^b f(x)dx \simeq \sum_{i=1}^{N-1} \overline{f}_i \Delta x_i \tag{6}$$

…which is a form of the Reimann Integral. The value $\overline{f}_i$ is the mean value of the function f(x) over the i^{th} interval, and Δx_i is the change in x over the i^{th} interval. Since we claim that f(x) is known, we may assume that we have two vectors, f and x, of length N. To generate $\overline{f}_i$, we shall assume that the average value of f over the interval is sufficiently close to the mean [more sophisticated methods discussed in other courses], hence:

```
i = 1:N-1;
fav = 0.5*(f(i) + f(i+1));
```

The size of the interval is given by:

```
delx = x(i+1)-x(i);
```

The integral may then be determined by the 'sum' or 'cumsum' commands. The command sum(v) will return the sum over the elements in vector v. For instance, v = [10 20 30], so sum(v) = 10+20+30 = 60.

On the other hand, cumsum(v) generates the cumulative sum – it returns a vector of the sum of all the elements obtained *up to the current element*. For instance,

cumsum(v) = [[10] [10+20] [10+20+30]] = [10 30 60].

Therefore, eq. (6) may be determined:
sum(fav.*delx) or cumsum(fav.*delx).

We use ‘sum’ if we are only interested in the final result, and ‘cumsum’ if we wish to observe how the integral changes with x.

Solving Integrals Analytically

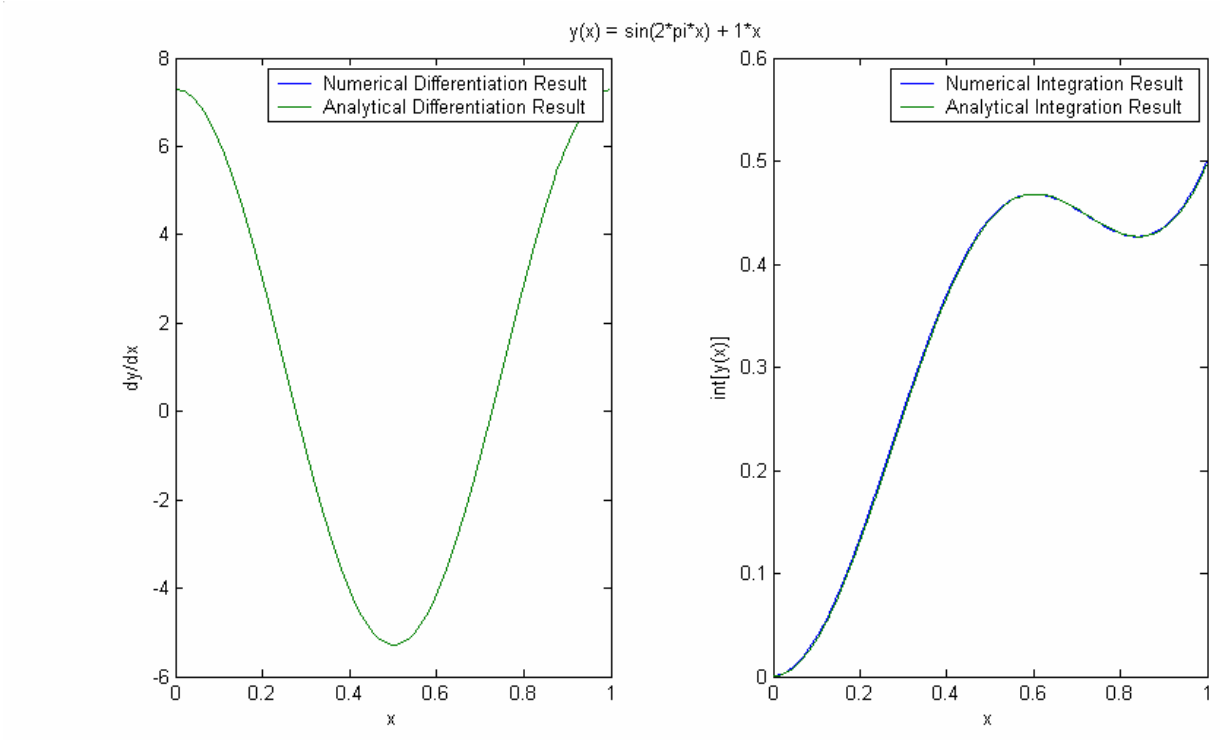
Integration may also be performed by use of the MatLab symbolic toolbox. The command dsolve(‘DF=f(x)’,’I(0)=0’,’x’) instructs MatLab to solve the equation

$$\frac{dF}{dx} = f(x) \qquad \text{y(0)=0} \tag{7}$$

…by integration. How does this help us to solve eq. (6)? If we differentiate eq. (6) wrt. x, we obtain exactly eq. (7), which means that the dsolve command above is solving exactly the same problem.

Solve the integral discussed in the class exercise numerically and analytically, and compare the results.

Final solution



Solving Differential Equations

The integration problems dealt with in the previous section contained known functions as integrands. However, when solving differential equations, we often have to integrate functions whose values are not known. In fact, the solving of differential equations are the most common type of problem encountered in many fields of chemical engineering.

For example, consider the reaction 2A →B in a vessel whose temperature is changing with time. For a second order reaction, the reaction rate is given by

$$r_A = -kC_A^2 \text{ where } k = k_0 \exp\left(-\frac{E}{RT}\right) \tag{8}$$

When write a mole balance on the reactor, we obtain:

$$\frac{dC_A}{dt} = -kC_A^2 \text{ where } k(t) = k_0 \exp\left(-\frac{E}{RT(t)}\right) \tag{9}$$

In this case, since temperature is changing with time, and since the concentration C_A is not known initially, we cannot simply integrate as follows:

$$C_A(t) = \int \frac{dC_A}{dt} dt = \int \left[-kC_A^2\right] dt = \int \left[-k_0 \exp\left(-\frac{E}{RT(t)}\right) C_A^2\right] dt \tag{10}$$

…since the C_A is unknown.

Equation (9) is an ordinary differential equation (ODE) which may be solved using MatLab's suite of ODE solvers. How does the ODE solver work? Consider a problem having the general form:

$$\frac{dy}{dt} = f(t,y) \text{ with } y(t_0) = y_0 \tag{11}$$

The equation $y(t_0) = y_0$ is referred to as the initial condition. It means that at a certain value of t [namely, t_0], we know that value of y [namely, y_0]. We wish to obtain the values of y at all other time values [besides just t_0]. In other words, we want the function y(t).

To get the solution, the MatLab solver starts at $t = t_0$ and evaluates f(t,y). Since y is known at t_0, f is known, hence dy/dt is known. The solver then attempts to determine the value of y at some time slightly greater than t_0 in the following way:

$$y(t_0 + \Delta t) \simeq y(t_0) + \frac{dy}{dt}\Delta t = y_0 + f(t_0, y_0)\Delta t \tag{12}$$

Obviously, the greater the time step, the greater the inaccuracy. MatLab optimizes the time step (Δt); it uses a small enough step such that accuracy is preserved, but it is also large enough that the problem is solved quickly. (The smaller the step, the long it takes to solve the problem over a range [0 t].)

After this first 'step', we then have $y(t_0 + \Delta t)$, which may be used to calculate the next value of y… and so on, until we solve the problem over the required integration limits.

Continuing with the heated batch reactor example, our equations are:

$$\frac{dC_A}{dt} = -kC_A^2 \text{ where } k(t) = k_0 \exp\left(-\frac{E}{RT(t)}\right) \tag{13}$$

and $C_A(0) = 1\text{mol.m}_{-3}$ [initial condition]

We want to know how the concentration varies over the time 0 to 10s, given that the temperature varies according to the equation: $T(t) = -3t^2 + 35t + 300$ [K]. We need to write two m-files: a main programs that calls the ODE solver, and a sub-program that returns that right hand side of the ODE.

In the main program, 'ex8main.m', we write:

```
[t,Ca] = ode45('dCadt',[0 10],1);
```

…which calls ode45 and informs it that it can get the RHS of the ODE by looking in file 'dCadt.m', to integrate over the range 0 to 10, and to use an initial value of 1. In the file 'dCadt.m', we write:

```
function dCadt = ddt(t,Ca)
k0 = 1e3; E = 2e4; R = 8.314;
T = -3*t.^2 + 35*t + 300;
k = k0*exp(-E/R/T);
dCadt = -k*Ca^2;
```

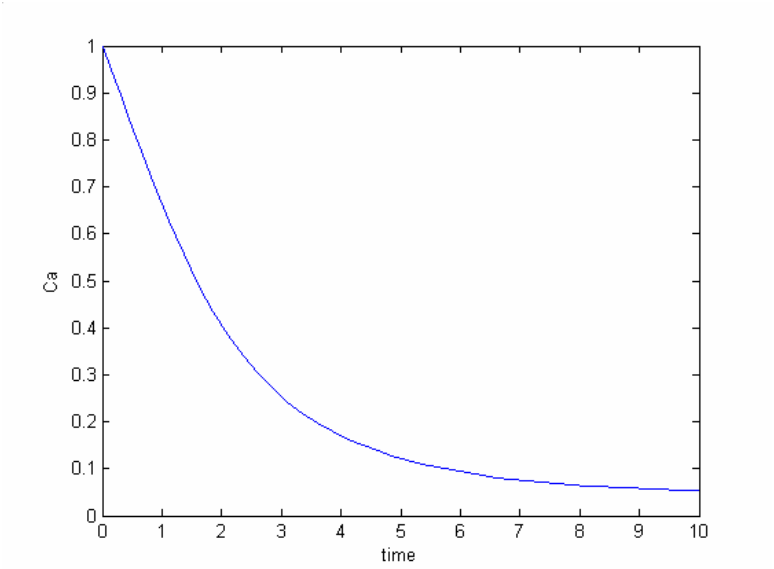
…which is all the information needed to generate the RHS of the ODE. On running the main program we obtain the following result shown in the figure.

Multi-variable Integration

What if there is more than one variable that depends on time? Then we could integrate them each with two calls to the solver. However, what if these variables were interdependent? For example, if the reaction scheme is more complex, how can we obtain the final results?

Example: Consider the following reaction scheme:

$$2A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \xrightarrow{k_3} C$$



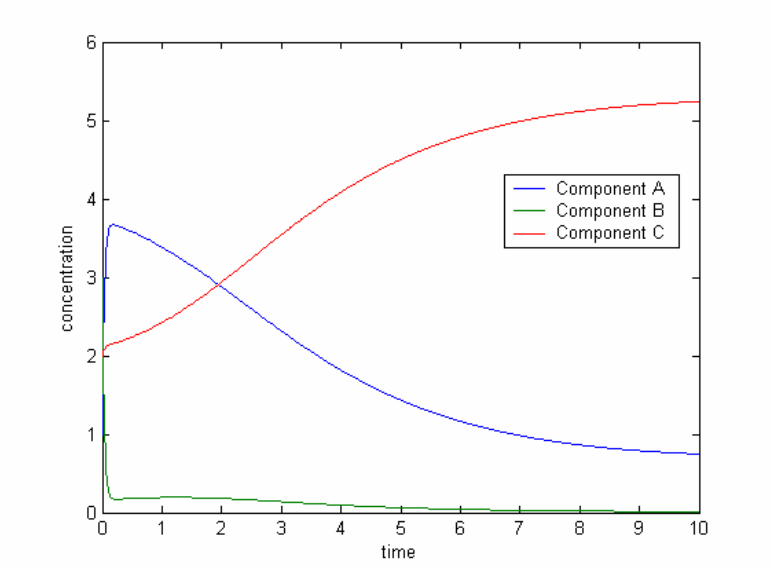
In this case, the system is much more complex – there are three concentrations that are interrelated. Identify the three reactions involved and confirm that they results in the following mathematical structure:

$$\frac{dC_A}{dt} = -k_1 C_A^2 + k_2 C_B \tag{14}$$

$$\frac{dC_B}{dt} = k_1 C_A^2 - k_2 C_B - k_3 C_B$$

$$\frac{dC_C}{dt} = k_3 C_B$$

These equations cannot be decoupled, and they must be solved simultaneously. The programs will be discussed in detail in class. A simulation result is presented below:



Coupling of material and energy balances in a reactor will be discussed in class.

Partial Differential Equations

The MatLab PDE solver can solve PDEs of the type:

$$C\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t}=x^{-m}\frac{\partial}{\partial x}\left(x^{m}f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right)+s\left(x,t,u,\frac{\partial u}{\partial x}\right) \tag{15}$$

…with an initial condition of the type

$$u(x,t_0)=u_0(x) \tag{16}$$

… and boundary conditions of the type

$$p(x,t,u)+q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right)=0 \tag{17}$$

Example: Reaction-Diffusion in a spherical catalyst particle

For this system, the PDE of interest is given by

$$\frac{\partial u}{\partial t}=\frac{D}{R}\frac{1}{x^2}\frac{\partial}{\partial x}\left(x^2\frac{\partial u}{\partial x}\right)-ku \tag{18}$$

… where u is the concentration, x is the radial position in the particle, R is the particle radius and D is the diffusion rate through the particle. The initial condition is u(x) = 1000, and the boundary conditions are:

$$\frac{du}{dx}=\frac{kR}{D}\left(u_s-u\right) \text{ at the surface of the particle } (x=1) \tag{19}$$

$$\frac{du}{dx}=0 \text{ at the centre of the particle } (x=0)$$

Hence, $m=2, C=1, f=\frac{\partial u}{\partial x}, s=-ku, p(x=1)=\frac{kR}{D}\left(u_s-u\right), q(x=1)=1, p(x=0)=0, q(x=0)=1$, which generates [program discussed in class]: