Numerical analysis in MATLAB

Chitralekha Singh

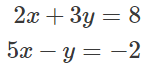
211CH061

In this report, we solved 3 real-time chemical engineering problems using numerical methods in MATLAB.

# Introduction

A system of linear equations can be represented in matrix form using a coefficient matrix, a variable matrix, and a constant matrix.

Consider the system,



The coefficient matrix can be formed by aligning the coefficients of the variables of each equation in a row. Make sure that each equation is written in standard form with the constant term on right.

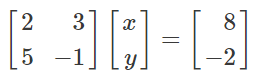
Then, the coefficient matrix for the above system is,



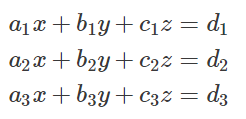
The variables we have are *x* and *y*. So we can write the variable matrix as .

On the right side of the equality we have the constant terms of the equations, 8 and −2. The two numbers in that order correspond to the first and second equations, and therefore take the places at the first and the second rows in the constant matrix. So, the matrix becomes,

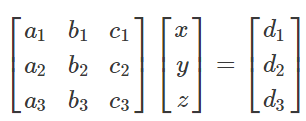
.

Now, the system can be represented as  .

In a similar way, for a system of three equations in three variables,



The matrix representation would be



We can generalize the result to *n* variables.

*1.Gaussian Elimination Method:*

Gaussian Elimination Method is one of the methods to solve system of linear equation (like AX=b). There are following steps involved in calculating solution of linear equation using Gaussian Elimination Method.

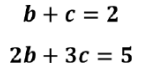
a. Create Augmented Matrix i.e. [A b]

b. Reduce that matrix to Row Echelon Form (This is called forward elimination)

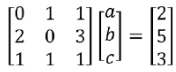
c. Forward Elimination will give us the value of nth unknown variable. Then using this value we can calculate remaining variables in reverse order. This is called backward substitution.

Let’s consider a system of Linear Equations ;

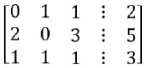




After putting these equation in matrix format we have ;

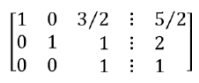


Step 1: Augmented Matrix i.e. [A b] ;



Step 2: Reduction of Matrix to a upper triangular matrix

after row elementary operation we have achieved a upper triangular matrix of above matrix which looks as;



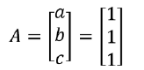
Step 3: Backward Substitution

It is clear from the last row that value of c=1. Remaining value can be calculated using this value of c.





Using above equation we have the final results:



*2.Gaussian Jordan method:*

Gaussian Jordan Method is modified form of Gaussian Elimination method. It is also used to solve the system of linear equations (AX=b). Procedure to implement Gaussian Jordan Method is almost same as Gaussian Jordan with few differences. Following are the steps involved in Gaussian Jordan Methods.

a. Create Augmented Matrix i.e. [A b]

b. Reduce that matrix to a Diagonal Matrix.

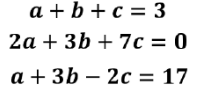
• The rows (if any) consisting entirely of zeros are grouped together at the bottom of the matrix.

• In each row that does not consist entirely of zeros, the leftmost nonzero element is a 1

• Each column that contains a leading 1 has zeros in all other entries

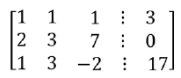
c. Stop process in step 2 if you obtain a row whose elements are all zeros except the last one on the right. In that case, the system is inconsistent and has no solutions. Otherwise, finish step 2 and read the solutions of the system from the final matrix.

Let us consider system of linear equations ;

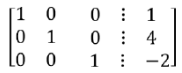


We will solve above system of Linear Equation using Gauss Jordan Method.

Step 1: Augmented Matrix [A b] ;



Step 2: We will reduce above matrix into a Diagonal matrix, The finalized Matrix is ;

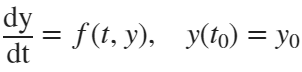


It is clear from above matrix form that a=1, b=4, c=-2



*3.Runge-Kutta method for ODE:*

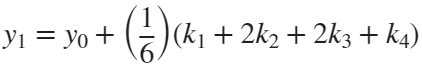
Let an initial value problem be specified as follows:



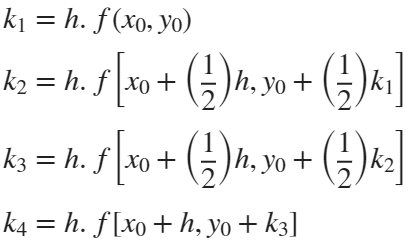
Here y is an unknown function (scalar or vector) of time t, which we would like to approximate; we are told that , the rate at which y changes, is a function of t and of y itself. At the initial time the corresponding y value is . The function f and the initial conditions  are given.

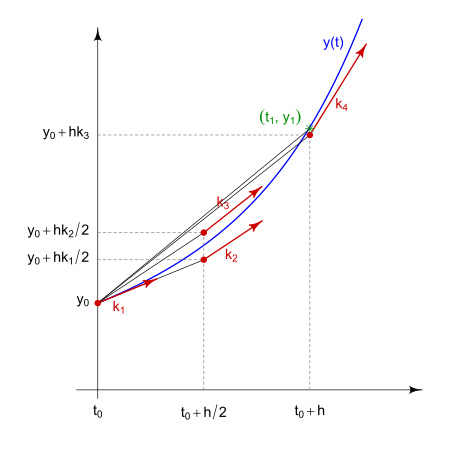
Now we pick a step-size *h* > 0 and define:

**4th Order Runge-Kutta method**:



Here,



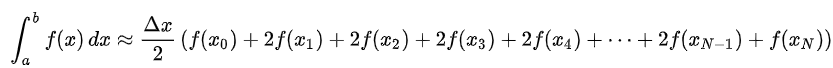


*4.Trapezoidal rule:*

Trapezoidal Rule is a technique for approximating the definate integral. It is based on approximating the area under a curve by dividing the interval into small trapezoids and summing their areas. The rule assumes that the function is approximately linear between adjacent points.

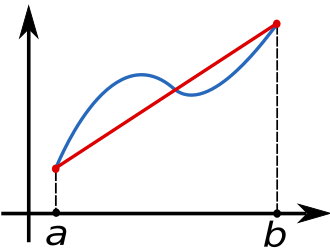
To apply the trapezoidal rule, the interval is divided into subintervals of equal width, and the function values at the endpoints of each subinterval are used to calculate the area of the trapezoid formed. The areas of all the trapezoids are then summed to estimate the integral.

Mathematically, the formula for the trapezoidal rule can be expressed as:



Where:

* *a* and *b* are the limits of integration.
* x is the width of each subinterval (x=(*b*−*a)/N)*�, where N is the number of subintervals).
*  are the evenly spaced points within the interval.



The accuracy of the trapezoidal rule improves as the number of subintervals increases, approaching the true integral value as the width of the subintervals approaches zero. While it's a relatively simple method, it may not be as accurate as more advanced techniques like Simpson's rule for functions with significant curvature.

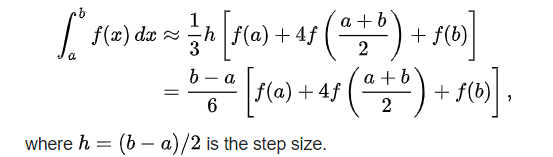
*5.Simpson's rules:*

Simpson's rules are numerical integration techniques used to approximate the definite integral of a function over an interval. They provide a more accurate estimation of the integral compared to simpler methods like the trapezoidal rule. Simpson's rules are based on approximating the function by piecewise quadratic or cubic polynomials.

a) Simpson's 1/3 Rule:

Simpson's 1/3 rule uses quadratic approximations to the function. The interval is divided into subintervals of equal width, and the function values at the endpoints and the midpoint of each subinterval are used to construct a quadratic polynomial. The areas under these polynomials are then combined using specific coefficients.

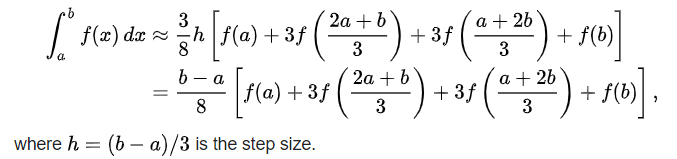
Mathematically, the formula for Simpson's 1/3 rule can be expressed as:



b) Simpson's 3/8 Rule:

Simpson's 3/8 rule uses cubic approximations to the function. Similar to the 1/3 rule, the interval is divided into subintervals, but this time the function values at the endpoints and two intermediate points of each subinterval are used to construct a cubic polynomial. The areas under these polynomials are combined using specific coefficients.

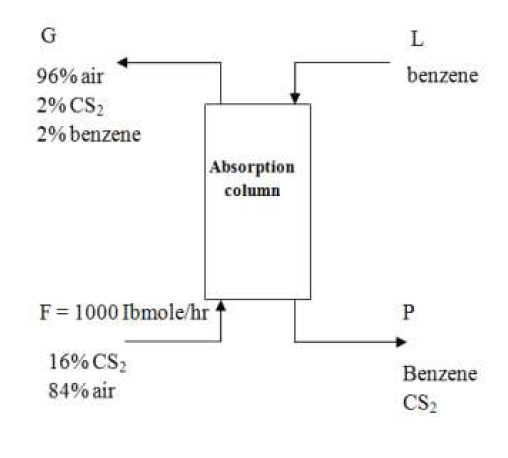
Mathematically, the formula for Simpson's 3/8 rule can be expressed as:



Let's start solving problems using these methods in MATLAB

***PROBLEM-1:***

**A gaseous mixture (F) consists of 16 mol% CS2 and 84 mol% air are fed to the absorption column at a rate of 1000 Ibmole/hr. Most of the CS2 input are absorbed by liquid benzene (L) which is fed to the top of the column. 1 % of benzene input are evaporated and out with the exit gas stream which consists of 96 mol% air, 2 mol% CS2 and 2 mol% benzene. The product liquid stream (P) consists of benzene and CS2. Calculate the mole flow rates of (G), (L) and (P) and the compositions.**



We will solve this problem using Gaussian Elimination and Gaussian Jordan Method.

A] Gaussian Elimination Method:

%Gaussian Elimination Method

% Coefficient matrix

A = [2, 0.01, -1; 2, -1, 0; 0, 1, -1];

% Right-hand side vector

b = [0; 0; 840];

% Augmented matrix [A | b]

aug = [A, b];

% Perform Gaussian elimination

n = size(aug, 1);

for row = 1:n

% Divide the row by the diagonal element

diagonal\_element = aug(row, row);

aug(row, :) = aug(row, :) / diagonal\_element;

% Eliminate elements below the diagonal

for i = row + 1:n

factor = aug(i, row);

aug(i, :) = aug(i, :) - factor \* aug(row, :);

end

end

% Back substitution

x = zeros(n, 1);

x(n) = aug(n, n + 1);

for row = n - 1:-1:1

x(row) = aug(row, n + 1);

for j = row + 1:n

x(row) = x(row) - aug(row, j) \* x(j);

end

end

% Display the results

FG = x(1);

FL = x(2);

FP = x(3);

Mole Flow Rates:

disp(['Flow rate of G: ', num2str(FG),' lbmole/hr']);

Flow rate of G: -42000 lbmole/hr

disp(['Flow rate of L: ', num2str(FL),' lbmole/hr']);

Flow rate of L: -84000 lbmole/hr

disp(['Flow rate of P: ', num2str(FP),' lbmole/hr']);

Flow rate of P: -84840 lbmole/hr

% Calculate compositions

xL = FL / (FG \* 2); % CS2 in liquid

yP = FP / (FP + 840 \* (1 - FG / 1000)); % Benzene in product

Compositions:

disp(['composition of CS2 in L: ' , num2str(xL)]);

composition of CS2 in L: 1

disp(['composition of Benzene in P: ' , num2str(yP)]);

composition of Benzene in P: 1.7414

B] Gaussian Jordan Method:

% Gaussian Jordan Method

% Coefficient matrix

A = [2, 0.01, -1; 2, -1, 0; 0, 1, -1];

% Right-hand side vector

b = [0; 0; 840];

% Augmented matrix [A | b]

aug= [A, b];

% Perform Gauss-Jordan elimination

n = size(aug, 1);

for row = 1:n

% Divide the row by the diagonal element

diagonal\_element = aug(row, row);

aug(row, :) = aug(row, :) / diagonal\_element;

% Eliminate elements above and below the diagonal

for i = 1:n

if i ~= row

factor = aug(i, row);

aug(i, :) = aug(i, :) - factor \* aug(row, :);

end

end

end

% Display the results

FG = aug(1, n + 1);

FL = aug(2, n + 1);

FP = aug(3, n + 1);

Mole Flow Rates:

disp(['Flow rate of G: ', num2str(FG),' lbmole/hr']);

Flow rate of G: -42000 lbmole/hr

disp(['Flow rate of L: ', num2str(FL),' lbmole/hr']);

Flow rate of L: -84000 lbmole/hr

disp(['Flow rate of P: ', num2str(FP),' lbmole/hr']);

Flow rate of P: -84840 lbmole/hr

% Calculate compositions

xL = FL / (FG \* 2); % CS2 in liquid

yP = FP / (FP + 840 \* (1 - FG / 1000)); % Benzene in product

Compositions:

disp(['composition of CS2 in L: ' , num2str(xL)]);

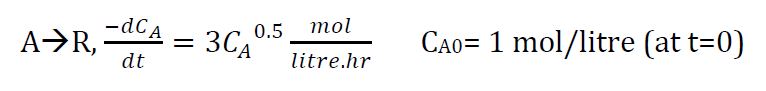
composition of CS2 in L: 1

disp(['composition of Benzene in P: ' , num2str(yP)]);

composition of Benzene in P: 1.7414

***PROBLEM-2:***

**Find the conversion after 1hour in a batch reactor for the reaction by using 4th order Runge Kutta Method.**



%Runge-kutta 4th order method

% Define the reaction rate equation

n = 0.5; % Reaction order

k = 3; % Rate constant

rate\_eq = @(t, CA) -k \* CA^n;

% Set initial conditions

CA0 = 1.0; % Initial concentration of A

% Define the time span

t\_span = [0, 1]; % Time span from 0 to 1 hour

% Use ode45 to solve the ODE

[t, CA] = ode45(rate\_eq, t\_span, CA0);

% Calculate the conversion after 1 hour

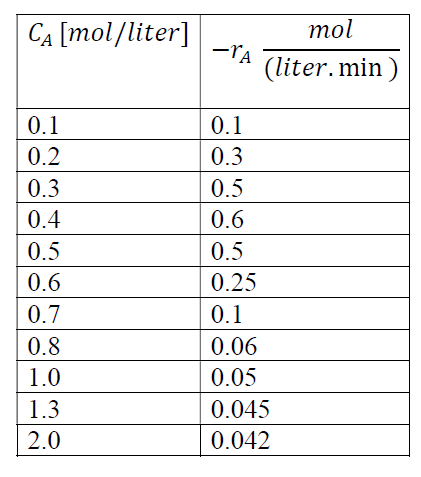
conversion = (CA0 - CA(end)) / CA0;

disp(['Conversion after 1 hour: ', num2str(conversion)]);

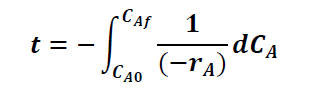
Conversion after 1 hour: 1.0017-2.5429e-06i

***PROBLEM-3:***

**We are planning to operate a batch reactor to convert A into R. This is a liquid reaction, the stoichiometry is 𝐴 → 𝑅, and the rate of reaction is given in the following Table. How long must we react each batch for the concentration to drop from** **= 1.3 mol/liter to** **= 0.3 mol/liter?**



The batch time can be determined from the following equation.Use trapezoidal and Simpson’s rules.



A] Trapezoidal Rule:

%Trapezoidal Rule

% Given rate data

CA\_data = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, 1.3, 2.0]; % mol/liter

rate\_data = [0.1, 0.3, 0.5, 0.6, 0.5, 0.25, 0.1, 0.06, 0.05, 0.045, 0.042]; % mol/(liter.min)

% Given concentrations

CA0 = 1.3; % mol/liter

CAf = 0.3; % mol/liter

% Function to interpolate rate at a given concentration

rate\_interpolated = interp1(CA\_data, rate\_data, CA0);

% Function to integrate

func = @(CA) 1 / (rate\_interpolated \* CA);

% Trapezoidal rule integration

n = 1000; % Number of intervals

h = (CA0 - CAf) / n; % Interval width

sum\_trapezoidal = 0;

for i = 1:n-1

CA = CAf + i \* h;

sum\_trapezoidal = sum\_trapezoidal + func(CA);

end

% Calculate batch time using trapezoidal rule

t\_trapezoidal = 1 / rate\_interpolated \* h \* (sum\_trapezoidal + (func(CAf) + func(CA0)) / 2);

disp(['Batch time (Trapezoidal Rule): ' , num2str(t\_trapezoidal), ' minutes']);

Batch time (Trapezoidal Rule): 724.1175 minutes

B] Simpson's Rules:

%Simpson's Rules

% Given rate data

CA\_data = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, 1.3, 2.0]; % mol/liter

rate\_data = [0.1, 0.3, 0.5, 0.6, 0.5, 0.25, 0.1, 0.06, 0.05, 0.045, 0.042]; % mol/(liter.min)

% Given concentrations

CA0 = 1.3; % mol/liter

CAf = 0.3; % mol/liter

% Function to interpolate rate at a given concentration

rate\_interpolated = interp1(CA\_data, rate\_data, CA0);

% Function to integrate

func = @(CA) 1 / (rate\_interpolated \* CA);

% Simpson's 1/3 rule integration

n = 1000; % Number of intervals

h = (CA0 - CAf) / n; % Interval width

sum\_simpson\_13 = func(CAf) + func(CA0);

for i = 1:n-1

CA = CAf + i \* h;

sum\_simpson\_13 = sum\_simpson\_13 + (mod(i, 2) + 1) \* 2 \* func(CA);

end

% Calculate batch time using Simpson's 1/3 rule

t\_simpson\_13 = 1 / rate\_interpolated \* h / 3 \* sum\_simpson\_13;

disp(['Batch time (Simpson''s 1/3 Rule): ' , num2str(t\_simpson\_13), ' minutes']);

Batch time (Simpson's 1/3 Rule): 724.1171 minutes

% Simpson's 3/8 rule integration

n = 1000; % Number of intervals

h = (CA0 - CAf) / n; % Interval width

sum\_simpson\_38 = func(CAf) + func(CA0);

for i = 1:n-1

CA = CAf + i \* h;

if mod(i, 3) == 0

sum\_simpson\_38 = sum\_simpson\_38 + 2 \* func(CA);

else

sum\_simpson\_38 = sum\_simpson\_38 + 3 \* func(CA);

end

end

% Calculate batch time using Simpson's 3/8 rule

t\_simpson\_38 = 1 / rate\_interpolated \* h \* 3 / 8 \* sum\_simpson\_38;

disp(['Batch time (Simpson''s 3/8 Rule): ' , num2str(t\_simpson\_38), ' minutes']);

Batch time (Simpson's 3/8 Rule): 724.0221 minutes

# SUMMARY:

We carried out different Numerical methods to solve different chemical engineering problems like a Process Calculation question, also solving a Reaction Kinetic problem by the Runge-Kutte method, and another Reaction Kinetics problem with Concentration and Rate data given.