Pimpri Chinchwad Education Trust's

Pimpri Chinchwad College of Engineering

LAUC MATLAB COMBINE A.Y. 2024-25 SEM I

Date: 09/12/2024

UNIT I : MATRICES

MATLAB PROGRAM 1 : Matrix to Linear system of Equation

```
% 1) Matrix to Linear system of Equation
% Take matrix A from the user
A=input('Enter your matrix A of order m * n:')


% Get the number of rows and columns of matrix A
[m, n]=size(A) % assign values to m & n


% Prompt user to enter Matrix B
B=input('Enter your matrix B of order m *1:') % Take matrix B from the user


% Check if the number of rows in A matches the number of rows in B
if size(B, 1) ~= m
error('Error: The number of rows in Matrix B must match the number of rows in
Matrix A.')
end


% Initialize symbolic variables
vars= sym('x', [1, n]) % Creates symbolic variables x1, x2, ..., xn


X=vars.' % Transpose symbolic variable array to a column vector


% Define the system of linear equations
equations = A * X == B


disp('The system of equations is set up successfully.')
```

MATLAB PROGRAM 2 : System of Linear Equation to Matrix

```
% 2) Equation to Matrix
% Prompt user for the number of variables
num_vars = input('Enter the number of variables: ')


% Initialize symbolic variables
vars=sym('x', [1, num_vars])
```

```
% Prompt user for the number of equations
num_eqns = input('Enter the number of equations: ')


% Inform the user about the symbolic variables
fprintf('Kindly enter equations using the following variables: ');
disp(vars);


equations = [];  % Initialize an empty array to hold equations


for k = 1:num_eqns
% Ask for the equation in string format
eqStr = input(sprintf('Enter equation %d (e.g., x1 + 2*x2 + x3 = 5): ',
k),"s");
    equations = [equations; str2sym(eqStr)];  % Convert string to symbolic
equation
end


% Convert the equations to matrix form [A|B]
[A, B] = equationsToMatrix(equations, vars)
```

MATLAB PROGRAM 3 : Unique, Infinite & No solution of Linear System

```
% 3) Nature of the solution
A=input('Enter your matrix A of order m*n :')
B=input('Enter your column matrix B of order m*1 :')
% construct augmented matrix
A_B=[A B] % augmented matrix
r1=rank(A_B)
r2=rank(A)
[m, n]=size(A)
if r1==r2 & r1==n
disp('Given system is consistent and has unique solution')
elseif r1==r2 & r1<n
disp('Given system is consistent and has infinite solution')
else
disp('Given system is inconsistent and has no solution')
end
```

MATLAB PROGRAM 4 : Unique, Infinite & No solution of Linear System

```
% 4) Linear System Solution
symsx y z
equations=[1*x+2*y+3*z==3, 3*x+6*y+9*z==9, 7*x+8*y+10*z==34]
sol=solve(equations, [x y z], 'ReturnConditions', true)
x=sol.x
y=sol.y
z=sol.z
```

MATLAB PROGRAM 5 : Linearly Dependent & Independent Vectors

```
% 5) Linear Dependent &Independent Vectors
num_vect=input('Enter number of vectors: ')


% Initialize an empty matrix to hold vectors
vect_mat=[];


for k=1:num_vect
    vector=input(sprintf('Enter your %d vector as row vector: ', k))
vect_mat=[vect_mat; vector]; % Appending row-wise
end


A=vect_mat'


% Check the rank of the matrix
if rank(A) == num_vect
disp('Vectors are linearly independent (LI)');
else
disp('Vectors are linearly dependent (LD)');
end
```

MATLAB PROGRAM 6 : Eigen Values & Eigen Vectors

```
% 6) Eigen Values & Eigen Vectors
A = input('Enter the square Matrix A: ')
[m, n]=size(A)


% Check whether matrix A is square or not
```

```
if m ~= n
error('Your matrix is not a square matrix')
end


[V, D]=eig(A) % V eigen vector matrix & D is diagonal maytrix in which e.v. on the diagonal


for k=1:m
fprintf('The %d eigen value is %s', k, D(k,k))
fprintf('The corresponding eigen vector is: \n')
disp(V(:, k))
end
```

UNIT II : DIFFERENTIAL CALCULUS

MATLAB PROGRAM 7 : Limit / Indeterminate Forms

```
% 7) Limit/Indeterminate Forms
% syntax
syms x % Symbolic Variable
 limit(cos(x), x, pi/2) % input angle consider in radian
 limit(sind(x), x, 90)  % Input angle consider in degree

% Interactive Programming
 g=input("Enter the function g(x): ");
 a=input("Enter the value 'a' for which limit x tends to 'a': ");
 h=limit(g, x, a);
fprintf("The limit of the function g(x)=%s at x tends to %d is %s", g, a, h)
```

MATLAB PROGRAM 8 : Taylor's and Maclaurin's Theorem

```
% 8) Taylor's and Maclaurin's theorem
%syntax
taylor(exp(x), x, a=2) % If a=0 the it gives Maclaurin's expansion
taylor(exp(x), x, 0) % Maclaurin's series


% By deafulttaylor() expand series up to degree 5
taylor(exp(x), x, 0, Order=10) % Expand function up to degree 9.
% If Order=n then we get expansion up to degree n-1
%sympref("PolynomialDisplayStyle","ascend") % expand series in ascending
powers
%sympref("PolynomialDisplayStyle","descend") % expand series in descending
powers
%sympref("PolynomialDisplayStyle","default") % expand series in deafult
setting
% Expansion of function of more than one variable
syms x y z
taylor(sin(y)+cos(z), y, 0) % expand in powers of y
taylor(sin(y)+cos(z), z, 0) % expand in powers of y
taylor(sin(y)+cos(z), [y, z], 0) % expand in both powers of x & y

% Interactive Programming
 f=input("Enter the function f(x): ")
 a=input("Expansion at point a: ") % expansion in powers of (x-a)
 b=input("Order of the expansion is b: ")
 g=taylor(f, x, a, Order=b);
fprintf("The expansion of f(x)=%s at x=%i is \n %s", f, a, g)
```

MATLAB PROGRAM 9 : Successive Differentiation

```
% 9) Successive Differentiation
% syntax
symsx y % Symbolic variable
diff(x^3-2*x, x, 2)  % differentiate function x^3-2*x 2 times w.r.t. x
diff(x^2*y-3*x*y^3, x, 1) % First Order partial derivative w.r.t. x
diff(x^2*y-3*x*y^3, y, 2) % Second Order partial derivative w.r.t. y


% Interactive Programming
f=input("Enter the function f(x)=")
n=input("Enter order of the derivative n=")

for i=1:n
f_d=diff(f, x, i);
fprintf("The %i derivative of the function f(x) is %s \n", i, f_d);
end
```

MATLAB PROGRAM 10 : Successive Differentiation Using Partial Fraction

```
% 10) Successive Differentiation Using Partial Fraction
syms x
f=input('Enter your function f(x): ')


% Perform partial fraction decomposition
f_pf=partfrac(f)


fprintf('The partial fraction of the function f(x) is %s', f_pf)


n=input('Enter the required order of derivative n: ')


for i=1:n
f_d=diff(f_pf, x, i)
fprintf('The %i derivative of the function f(x) is %s', i, f_d)
end
```

UNIT III: ORDINARY DIFFERENTIAL EQUATION

MATLAB PROGRAM 11 : Exact &Non Exact Differential Equation

```
% 11) Exact &Non Exact Differential Equation


symsx y C


% Define M(x, y) and N(x, y)
M = input('Enter M(x, y): ')
N = input('Enter N(x, y): ')


% Check for exactness
dM_dy = diff(M, y)
dN_dx = diff(N, x)


if dM_dy~=dN_dx
disp('Given differential equation is non excat')
    IF=input('Enter integrating factor mu(x, y): ')
    M = M * IF
    N = N * IF
M_intx = int(M, x)  % Integrate M w.r.t. x
N_inty = int(N - diff(M_intx, y), y);  % Integrate remaining terms w.r.t. y
    GS = M_intx + N_inty;
fprintf('The general solution(GS) is %s ==C', GS)
else
disp('Given differential equation is exact')
M_intx = int(M, x);  % Integrate M w.r.t. x
N_inty = int(N - diff(M_intx, y), y);  % Integrate remaining terms w.r.t. y
    GS = M_intx + N_inty;
fprintf('The general solution(GS) is %s ==C', GS)
end
```

OR

```
% 11) Exact & Non Exact Differential Equation
syms x y c

disp("Given functions are")
M=input("Enter the function M(x,y)")
N=input("Enter the function N(x,y)")
```

```
disp("First order partial derivatives are")
My=diff(M,y)
Nx=diff(N,x)

if My==Nx
    disp("Since, My=Nx, given D.E. is Exact")
    terms = sym(children(expand(N)));
    terms_free_from_x = sum(terms(~has(terms, x)));
    sol=int(M,x)+int(terms_free_from_x,y)==c
else
    disp("Since, My=!Nx, given D.E. is Non-Exact")
    IF=input("Enter Integrating factor to convert it in to exact")
    M1=expand(M*IF);
    N1=expand(N*IF);
    terms = sym(children(N1));
    terms_free_from_x = sum(terms(~has(terms, x)));
    sol=int(M1,x)+int(terms_free_from_x,y)==c
end
```

MATLAB PROGRAM 12: Heat Flow

```
% 12) Heat Flow
syms T(x) x


x1=input('Enter the inner radius of the pipe (x1): ')
x2=input('Enter the outer radius of the pipe (x2): ')
T1=input('Enter the inner temperature of the pipe (T1): ')
T2=input('Enter the inner temperature of the pipe (T2): ')
k=input('Enter coefficient of thermal conductivity: ')


% find q heat flux/loss first
q= -2*pi*k*(T2-T1)/(log(x2/x1))


% Display the heat flux
fprintf('Heat flux q = %s', q);


a=input('Enter distance normal to the surface area(a): ')
Temp=(T2 - T1) / log(x2 / x1) * log(a / x1) + T1
```

UNIT IV : LDE

MATLAB PROGRAM 13 : Solving higher order LDE

```
% 13) Solving Homogeneous LDE

syms x y(x)
ode= diff(y,x,2)+5*diff(y,x)+6*y==0
sol=dsolve(ode);
expand(sol)

% Interactive Programming
syms x y(x)
ode=input("Enter Differential equation")
sol=dsolve(ode);
expand(sol)


% 14) Solving Non-homogeneous LDE

syms x y(x)
ode= diff(y,x,2)+3*diff(y,x)+2*y==exp(exp(x))
sol=dsolve(ode);
expand(sol)


% Interactive Programming
syms x y(x)
ode=input("Enter Differential equation")
sol=dsolve(ode);
expand(sol)



% 15) Application on Electric Circuit
```

Step I: Generate differential equation from given electric circuit

% $\frac{d^2q}{dt^2} + 3\frac{dq}{dt} + 2q = \sin(t)$

Step II: solve using previous code of higher order LDE;

```
syms t q(t)
ode= diff(q,t,2)+3*diff(q,t)+2*q==sin(t)
sol=dsolve(ode);
expand(sol)
```