MATLAB Programs

1. **Chapter No. Two**

Program 2.1

MATLAB m-file for the Bisection Method

function sol=bisect(fn,a,b,tol)
$fa = feval(fn, a); fb = feval(fn, b);$
if $fa*fb > 0$; fprintf('Endpoints have same sign') return
end
while abs $(b - a) > tol$
$c = (a + b)/2$; $fc = feval(fn, c)$;
if $fa * fc < 0; b = c$; else $a = c$; end
end; sol=$(a + b)/2$;

Program 2.2

MATLAB m-file for the Fixed-Point Method

function sol=fixpt(fn,x0,tol)
old= x0+1;
while abs(x0-old) > tol; old=x0;
$x0 = feval(fn, old)$; end; sol=$x0$;

Program 2.3

MATLAB m-file for the Newton's Method

function sol=newton(fn,dfn,x0,tol)
old = x0+1;
while abs $(x0 - old) > tol$; $old = x0$;
$x0 = old - feval(fn, old)/feval(dfn, old)$;
end; sol=x0;

Program 2.4

MATLAB m-file for the Secant Method

function sol=secant(fn,a,b,tol)
$x0 = a; x1 = b; fa = feval(fn, x0); fb = feval(fn, x1);$
while abs(x1-old)> tol
$new = x1 - fb * (x1 - x0)/(fb - fa);$
$x0 = x1; fa = fb; x1 == new; fb = feval(fn, new);$
end; sol=new;

Program 2.5

MATLAB m-file for first Modified Newton's Method

```
function sol=mnewton1(fn1,dfn1,x0,m,tol)
old = x0+1;
while abs (x0 - old) > tol; old = x0;
fa=feval(fn,old); fb=feval(dfn,old);
x0 = old - (m * fa)/fb;
end; sol=x0;
```

Program 2.6

MATLAB m-file for second Modified Newton's Method

```
function sol=mnewton2(fn1,dfn1,ddfn1,x0,tol)
old = x0+1;
while abs (x0 - old) > tol; old = x0;
fa=feval(fn,old); fb=feval(dfn,old); fc=feval(ddfn,old);
x0 = old - (fa * fb)/((fb). ^ 2 - (fa * fc));
end; sol=x0;
```

Program 2.7

MATLAB m-file for Newton's Method for a Nonlinear System

```
function sol=newton2(fn2,dfn2,x0,tol)
old=x0+1; while max(abs(x0-old))>tol; old=x0;
f = feval(fn2, old); f1 = f(1); f2 = f(2);
J=feval(dfn2,old);
f1x = J(1, 1); f1y = J(1, 2); f2x = J(2, 1); f2y = J(2, 2);
D = f1x * f2y - f1y * f2x;
h = (f2 * f1y - f1 * f2y)/D; k = (f1 * f2x - f2 * f1x)/D;
x0 = old + [h, k]; end; sol=x0;
```

2. **Chapter No. Three**

Program 3.1

MATLAB m-file for finding inverse of a matrix

```
function [Ainv]=INVMAT(A)
[n,n]=size(A); I=zeros(n,n);
for i=1:n; I(i,i)=1; end
m(1:n,1:n)=A; m(1 : n, n + 1 : 2 * n) = I;
for i=1:n; m(i, 1 : 2 * n) = m(i, 1 : 2 * n)/m(i, i);
for k=1:n; if i~ =k
m(k, 1 : 2*n) = m(k, 1 : 2*n)-m(k, i)*m(i, 1 : 2*n);
end; end; end
invrs = m(1 : n, n + 1 : 2 * n);
```

Program 3.2

MATLAB m-file for Gaussian Elimination Without Pivoting

```
function x=WP(B)
[n,t]=size(B); U=B;
for k=1:n-1; for i=k:n-1; m=U(i+1,k)/U(k,k);
for j=1:t; U(i+1,j)=U(i+1,j)-m*U(k,j);end;end end
i=n; x(i,1)=U(i,t)/U(i,i);
for i=n-1:-1:1; s=0;
for k=n:-1:i+1; s = s + U(i,k) * x(k,1); end
x(i,1)=(U(i,t)-s)/U(i,i); end; B; U; x; end
```

Program 3.3

MATLAB m-file for Gaussian Elimination by Partial Pivoting

```
function x=PP(B)
% B = input('input matrix in form[A : b]');
[n, t] = size(B); U = B;
for M = 1:n-1
mx(M) = abs(U(M, M)); r = M;
for i = M+1:n
if mx(M) < abs(U(i, M))
mx(M)=abs(U(i,M)); r = i; end; end
rw1(1,1:t)=U(r,1:t); rw2(1,1:t)=U(M,1:t);
U(M,1:t)=rw1 ; U(r,1:t)=rw2 ;
for k=M+1:n; m=U(k,M)/U(M,M);
for j=M:t; U(k,j)=U(k,j)-m*U(M,j); end;end
i=n; x(i)=U(i,t)/U(i,i);
for i=n-1:-1:1; s=0;
for k=n:-1:i+1; s = s + U(i,k) * x(k); end
x(i)=(U(i,t)-s)/U(i,i); end; B; U; x; end
```

Program 3.4

MATLAB m-file for the Gauss-Jordan Method

```
function sol=GaussJ(Ab)
[m,n]=size(Ab);
for i=1:m
Ab(i,:) = Ab(i,:)/Ab(i,i);
for j=1:m
if j == i; continue; end
Ab(j,:) = Ab(j,:) - Ab(j,i) * Ab(i,:);
end; end; sol=Ab;
```

Program 3.5

MATLAB m-file for the LU decomposition Method

function $A = lu - gauss(A)$

% LU factorization using Gauss Elimination (without pivoting)

[n,n] = size(A); for i=1:n-1; pivot = A(i,i);

for k=i+1:n; A(k,i)=A(k,i)/pivot;

for j=i+1:n; A(k,j)= A(k,j) - A(k,i)*A(i,j); end; end; end

Program 3.6

MATLAB m-file for using the Doolittle's Method

function sol = Doll(A,b)

[n,n]=size(A); u=A;l=zeros(n,n);

for i=1:n-1; if abs(u(i,i))$>$ 0

for i1=i+1:n; m(i1,i)=u(i1,i)/u(i,i);

for j=1:n

$u(i1, j) = u(i1, j) - m(i1, i) * u(i, j)$;end;end;end;end

for i=1:n; l(i,1)=A(i,1)/u(1,1); end

for j=2:n; for i=2:n; s=0;

for k=1:j-1; $s = s + l(i, k) * u(k, j)$; end

l(i,j)=(A(i,j)-s)/u(j,j); end; end y(1)=b(1)/l(1,1);

for k=2:n; sum=b(k);

for i=1:k-1; $sum = sum - l(k, i) * y(i)$; end

y(k)=sum/l(k,k); end

x(n)=y(n)/u(n,n);

for k=n-1:-1:1; sum=y(k);

for i=k+1:n; $sum = sum - u(k, i) * x(i)$; end

x(k)=sum/u(k,k); end; l; u; y; x

Program 3.7

MATLAB m-file for the Crout's Method

```
function sol = Crout(A, b)
[n,n]=size(A); u=zeros(n,n); l=u;
for i=1:n; u(i,i)=1; end
l(1,1)=A(1,1);
for i=2:n
u(1,i)=A(1,i)/l(1,1); l(i,1)=A(i,1); end
for i=2:n; for j=2:n; s=0;
if i <= j; K=i-1;
else; K=j-1; end
for k=1:K; s = s + l(i,k) * u(k,j); end
if j > i; u(i,j)=(A(i,j)-s)/l(i,i); else
l(i,j)=A(i,j)-s; end;end;end
y(1)=b(1)/l(1,1);
for k=2:n; sum=b(k);
for i=1:k-1; sum = sum - l(k,i) * y(i); end
y(k)=sum/l(k,k); end
x(n)=y(n)/u(n,n);
for k=n-1:-1:1; sum=y(k);
for i=k+1:n; sum = sum - u(k,i) * x(i); end
x(k)=sum/u(k,k); end; l; u; y; x;
```

Program 3.8

MATLAB m-file for the Jacobi Iterative Method

```
function x=JacobiM(Ab,x,acc)
[n,t]=size(Ab); b=Ab(1:n,t); R=1; k=1;
d(1,1:n+1)=[0 x]; while R > acc
for i=1:n
sum=0;
for j=1:n; if j ~ =i
sum = sum + Ab(i,j) * d(k,j + 1); end;
x(1,i) = (1/Ab(i,i)) * (b(i,1) - sum); end;end
k=k+1; d(k,1:n+1)=[k-1 x];
R=max(abs((d(k,2:n+1)-d(k-1,2:n+1))));
if k > 10 & R > 100
('Jacobi Method is diverges')
break; end; end; x=d;
```

Program 3.9

MATLAB m-file for the Gauss-Seidel Iterative Method

```
function x=GaussSM(Ab,x,acc)
[n,t]=size(Ab); b=Ab(1:n,t);R=1; k=1;
d(1,1:n+1)=[0  x]; k=k+1; while R > acc
for i=1:n; sum=0; for j=1:n
if j <= i - 1; sum = sum + Ab(i, j) * d(k, j + 1);
elseif j >= i + 1
sum = sum + Ab(i, j) * d(k - 1, j + 1); end; end
x(1, i) = (1/Ab(i, i)) * (b(i, 1) - sum);
d(k,1)=k-1; d(k,i+1)=x(1,i); end
R=max(abs((d(k,2:n+1)-d(k-1,2:n+1))));
k=k+1; if R > 100 & k > 10; ('Gauss-Seidel method is Diverges')
break ;end;end;x=d;
```

1. **Chapter No. Four**

Program 4.1

MATLAB m-file for the Lagrange Interpolation Method

```
function fi=lint(x,y,x0)
dxi=x0-x; m=length(x); L=zeros(size(y));
L(1) = prod(dxi(2 : m))/prod(x(1) - x(2 : m));
L(m) = prod(dxi(1 : m - 1))/prod(x(m) - x(1 : m - 1));
for j=2:m-1
num = prod(dxi(1 : j - 1)) * prod(dxi(j + 1 : m));
dem = prod(x(j) - x(1 : j - 1)) * prod(x(j) - x(j + 1 : m));
L(j)=num/dem; end; fi = sum(y. * L);
```

Program 4.2

MATLAB m-file for the Divided Differences

```
function D=divdiff(x,y)
% Construct divided difference table
m = length(x); D = zeros(m, m); D(:, 1) = y(:);
for j=2:m; for i=j:m
D(i, j) = (D(i, j-1)-D(i-1, j-1))/(x(i)-x(i-j+1));
end; end
```

Program 4.3

MATLAB m-file for Newton's Interpolation Method

function Y=Ndivf(x,y,x0)

$m = length(x); D = zeros(m, m); D(:, 1) = y(:);$

for j=2:m; for i=j:m;

$D(i, j) = (D(i, j-1)-D(i-1, j-1))/(x(i)-x(i-j+1)); end; end;$

$Y = D(m, m) * ones(size(x0));$

for $i = m - 1 : -1 : 1;$

$Y = D(i, i) + (x0 - x(i)) * Y$; end

Program 4.4

MATLAB m-file for the Linear Spline

function LS=LSpline(X,Y,x)

n=length(X);

for i=n-1:-1:1

$D = x - X(i);$

if $(D >= 0)$; break; end; end

$D = x - X(i)$; if $(D < 0); i = 0;$

$D = x - X(1)$; end

$M = (Y(i + 1) - Y(i))/(X(i + 1) - X(i));$

$LS = Y(i) + M * D$; end

1. **Chapter No. Five**

Program 5.1

MATLAB m-file for the Composite Trapezoidal Rule

function TN=TrapezoidR(fn,a,b,n); h=(b-a)/n;

s=(feval(fn,a)+feval(fn,b))/2;

for k=1:n-1

$x = a + h * k$

s=s+feval(fn,x); end

$TN = s * h;$

Program 5.2

MATLAB m-file for the Composite Simpson's Rule

function SN=SimpsonR(fn,a,b,n);

h=(b-a)/n;

s=feval(fn,a)+feval(fn,b);

for k=1:2:n-1

$s = s + 4 * feval(fn, a + k * h)$; end

for k=2:2:n-2

$s = s + 2 * feval(fn, a + k * h)$; end

$SN = (s * h)/3;$

Program 5.3

MATLAB m-file for computing Error term of the Composite Simpson's Rule

```
function k=ErrorSR(fn,a,b,M,eps)
% M is a bound for the fourth derivative of fn on [a,b]
% eps is the required accuracy
L = abs(b-a); n = ceil(L*sqrt(sqrt(L*M/180/eps)));
if mod(n,2) == 1; n = n + 1; end
k=SimpsonR(fn,a,b,n);
```

1. **Chapter No. Six**

Program 6.1

MATLAB m-file for Euler's Method

```
function sol=Euler1(fun1,a,b,y0,n)
h=(b-a)/n; x=a+(0:n)*h; y(1)=y0;
for k = 1 : n
y(k + 1) =
y(k) + h * feval(fun1, x(k), y(k));
end; sol=[x',y'];
```

Program 6.2

MATLAB m-file for Taylor's Method of order 2

```
function sol=tayl1(fun1,dfun1,a,b,y0,n)
h=(b-a)/n; x = a + (0 : n) * h; y(1)=y0;
for k=1:n
y(k + 1) = y(k) + h * feval(fun1, x(k), y(k))
        + (h.^ 2*feval(dfun1, x(k), y(k)))/2;
end; sol = [x', y'];
```

Program 6.3

MATLAB m-file for the Modified Euler's Method

```
function sol=mod1(fun1,a,b,y0,n)
h = (b - a)/n; x = a + (0 : n) * h; y(1) = y0;
for k = 1 : n
k1 = feval(fun1, x(k), y(k));
k2 = feval(fun1, x(k) + h, y(k) + h * k1);
y(k + 1) = y(k) + h * (k1 + k2)/2;
end; sol = [x', y'];
```