Computer Programs for MATH-505

Chapter 2 Systems of Linear Equations

Program 2.1

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 2.2

MATLAB m-file for the Simple Gaussian Elimination Method

```
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 2.3

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 2.4

MATLAB m-file for the Gaussian Elimination by Total Pivoting

```
function x=TP(B)
% B = input('input matrix in form[A/b]');
[n,m]=size(B);U=B; w=zeros(n,n);
for i=1:n; N(i)=i; end
for M = 1:n-1; r=M; c=M;
for i = M:n; for j = M:n
if max(M) < abs(U(i, j)); max(M)=abs(U(i,j));
r = i; c = j; end; end; end
rw1(1,1:m)=U(r,1:m); rw2(1,1:m)=U(M,1:m);
U(M,1:m)=rw1;U(r,1:m)=rw2 ; cl1(1:n,1)= U(1:n,c);
cl2(1 : n, 1) = U(1 : n, M); U(1 : n, M) = cl1(1 : n, 1);
U(1 : n, c) = cl2(1 : n, 1); p = N(M); N(M) = N(c);
N(c) = p; w(M, 1 : n) = N;
for k = M + 1 : n; e = U(k, M)/U(M, M);
for j = M : m; U(k, j) = U(k, j) − e ∗ U(M, j); end; end
i = n; x(i, 1) = U(i, m)/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, m) − s)/U(i, i); end
for i=1:n; X(N(i), 1) = x(i, 1); end; B; U; X; end
```

Program 2.5
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 2.6

MATLAB m-file for the LU decomposition Method
function $A = lu - gauss(A)$
% LU factorization 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 2.7

MATLAB m-file for using the Doolittle's Method
function sol = Doolittle(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 2.8

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 2.9

MATLAB m-file for the Cholesky Method

```
function sol = Cholesky(A, b)
[n,n]=size(A); l=zeros(n,n); u=l;
l(1, 1) = (A(1, 1)) \ 0.5; u(1,1)=l(1,1);
for i=2:n; u(1,i)=A(1,i)/l(1,1);
l(i,1)=A(i,1)/u(1,1); 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);
elseif i == j
l(i, j) = (A(i, j) - s) \ 0.5; u(i,j)=l(i,j);
else; l(i,j)=(A(i,j)-s)/u(j,j); 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 2.10
MATLAB m-file for LU Decomposition for Tridiagonal System

```
function sol=TRiDLU(Tb)
[m,n]=size(Tb); L=eye(m); U=zeros(m);
U(1,1)=Tb(1,1);
for i=2:m
U(i-1,i) = Tb(i-1,i);
L(i,i-1) = Tb(i,i-1)/U(i-1,i-1);
U(i,i) - L(i,i-1) * Tb(i-1,i); end
disp('The lower-triangular matrix') L;
disp('The upper-triangular matrix') U;
y = inv(L) * Tb(:,n);
x = inv(U) * y;
```

Chapter 3 Conditioning of Linear Systems

Program 3.1
MATLAB m-file for finding Residual Vector

```
function r=RES(A,b,x0)
[n,n]=size(A);
for i=1:n; R(i) = b(i);
for j=1:n
R(i)=R(i)-A(i,j)*x0(j);end
RES(i)=R(i); end
r=RES'
```

Chapter 4 Iterative Methods for Linear Systems

Program 4.1

MATLAB m-file for the Jacobi Iterative Method

```
function x=JacobiM(Ab,x,acc)   % Ab = [A—b]
[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 4.2

MATLAB m-file for the Gauss-Seidel Iterative Method

```
function x=GaussSM(Ab,x,acc)    % Ab = [A—b]
[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;
```

Program 4.3

MATLAB m-file for the SOR Iterative Method

```
function sol=SORM(Ab,x,w,acc)    % Ab = [A—b]
[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 - w) * d(k - 1, i + 1) + (w/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))));
if R > 100 & k > 10; break; end
k=k+1; end; x=d;
```

Program 4.4

MATLAB m-file for the Conjugate Gradient Method

```
function x=CONJG(A,b,x0,acc,maxI)
x = x0; r = b - A * x0; v = r;
alpha=r' * r; iter=0;flag=0;
normb=norm(b); if normb < eps; normb=1; end
while (norm(r)/normb > acc)
u = A * v; t = alpha/(u' * v); x = x + t * v;
r = r - t * u; beta = r' * r;
v = r + beta/alpha * v; alpha = beta;
iter = iter+1; if (iter == maxI); flag= 1;
break; end; end
```

Chapter 5 The Eigenvalue problems

Program 5.1

MATLAB m-file for finding trace of matrix

```
function [trc]=trac(A)
n=max(size(A)); trc=0;
for i=1:n; for k=1:n
if i==k tracc=A(i,k); trc=trc+tracc; else
trc=trc;
end; end; end
```

Program 5.2

MATLAB m-file for using Cayley-Hamilton Theorem

function [c,Ainv]= chim(A)

n=max(size(A));

for i=1:n; for j=1:n

I(i,j)=0; I(i,i)=1; end; end

AA=A; AAA=A; c=[1];

for k=1:n; traceA=0;

for g=1:n % Loop to find the trace of matrix A.

traceA=traceA+A(g,g); end

$cc = -1/k * traceA$; % To find coefficients of the polynomial.

$c = [c, cc]; if k < n;$

for i=1:n; for j=1:n; $b(i,j) = A(i,j) + cc * I(i,j)$; end; end

for i=1:n; for j=1:n; s=0;

for m=1:n; $ss = AA(i,m) * b(m,j)$; s=s+ss; end;

A(i,j)=s; end; end; end; end

for i=1:n; for j=1:n

$su1(i,j) = c(n) * I(i,j) + c(n-1) * AA(i,j)$; su2(i,j)=0; end; end

if $n > 2$

for z=2:n-1; for i=1:n; for j=1:n; s=0;

for m=1:n; $ss = AAA(i,m) * AA(m,j); s = s + ss$; end

am(i,j)=s; end; end; AAA=am;

for i=1:n; for j=1:n

$su2(i,j) = su2(i,j) + c(n-z) * AAA(i,j)$; end; end;end;end

for i=1:n; for j=1:n

su(i,j)=su1(i,j)+su2(i,j); $Ainv(i,j) = -1/c(n+1) * su(i,j)$;

end;end

Program 5.3

MATLAB m-file for using Sourian-Frame theorem

```
function [c,Ainv]=sourian(A)
[n,n]=size(A);
for i=1:n; for j=1:n; b0(i,j)=0;b0(i,i)=1;end;end
AA=A; c[1]; for k = 1:n;
traceA=0; for i=1:n; traceA=traceA+A(i,i);end;
cc = -1/k * tracA; c = [c, cc]; if k < n;
for i=1:n; for j=1:n
b(i, j) = A(i, j) + cc * b0(i, j);end;end
for i=1:n; for j=1:n; s = 0; for m =1:n;
ss = AA(i, m) * b(m, j); s = s + ss; end;
A(i, j) = s; end; end; end; end;
for i=1:n; for j=1:n;
ai(i,j)=-1/c(n+1)*b(i,j); end; end
disp('Coefficients of the polynomial') c
disp('Inverse of the matrix A=ai') ai
```

Program 5.4

MATLAB m-file for using Bocher's Theorem

```
function c=BOCH(A)
[n,n]=size(A);
for i=1:n; for j=1:n; I(i,i)=1; end; end
A(n-1)=-trace(A);
for i=2:n; s=0; p=1;
for k=i-1:-1:1
s = s + A(n - k) * trace(A^ p); p = p + 1; end
if i=n; A(n - i) = (-1/i) * (s + trace(A^ i)); else
Ao = (-1/i) * (s + trace(A^ i)); end; end
coeff=[Ao  A]; if ao=0; s=A ^ (n-1);
for i=1:n-2
s = s + a(n - i) * A^ (n - (i + 1)); end
s = s + A(1) * I; Ainv = -s/Ao; else; end
```

Chapter 6 Numerical Computation of Eigenvalues

Program 6.1
MATLAB m-file for Power method

```
function sol=POWERM1(A,X,maxI)
[n,n]=size(A);
for k = 1:maxI;
X=A*X;
m=abs(X(1,1));
for i = 1:n;
if abs(m) < abs(X(i,1));
m=X(i,1);
end; end
k;m;X=X/m;
end
```

Program 6.2
MATLAB m-file for Inverse Power method

```
function sol=INVERSEPM1(A,X,maxI)
[n,n]=size(A);
B=inv(A);
for k = 1:maxI;
X=B*X;
m=abs(X(1,1));
for i = 1:n;
if abs(m) < abs(X(i,1));
m=X(i,1);
end; end
k;m;X=X/m;
end
```

Program 6.3

MATLAB m-file for Shifted Inverse Power method

```
function sol=ShiftedIPM1(A,X,mu,maxI)
[n,n]=size(A);
ID=zeros(n,n);
for i = 1:n;
for j = 1:n;
if i==j;ID(i,j)=1;
end; end; end;
B=inv(A-mu*ID);
for k = 1:maxI;
X=B*X;
m=abs(X(1,1));
for i = 1:n;
if abs(m) < abs(X(i,1));
m=X(i,1);
end; end
eigenvalue=1/m+mu;
k;m;X=X/m;eigenvalue;
end
```

Program 6.4

MATLAB m-file for using Deflation Method

```
function [Lamda,X]=DEFLATED(A,Lamda,XA)
[n,n]=size(a); Q=eye(n);
Q(:,1) = XA(:,1); B = inv(Q)*A*Q; c=B(2:n,2:n);
[xv, ev] = eig(c,'nobalance');
for i=1:n-1
b = -(B(1,2:n)*xv(:,i))/(Lamda - ev(i,i));
Xb(:,i) = [b  xv(:,i)']'; XA(:,i+1) = Q*Xb(:,i); end
Lamda=[Lamda;diag(ev)]; Lamda; XA; end
```

Program 6.5

MATLAB m-file for the Jacobi method
function sol=JOBM(A)
[n,n]=size(A); QQ=[ ];
for $u = 1 : .5 * n * (n - 1)$; for i=1:n; for j=1:n
if $(j > i)$; aa(i,j)=A(i,j); else; aa(i,j)=0;
end; end; end
aa=abs(aa); mm=max(aa); m=max(mm);
[i,j]=find(aa==m); i=i(1); j=j(1);
$t = .5 * atan(2 * A(i, j)/(A(i, i) - A(j, j)))$; c=cos(t); s=sin(t);
for ii=1:n; for jj=1:n; Q(ii,jj)=0.0;
if (ii==jj); Q(ii,jj)=1.0; end; end; end
Q(i,i)=c; Q(i,j)=-s; Q(j,i)=s; Q(j,j)=c;
for i1=1:n; for j1=1:n;
QT(i1,j1)=Q(j1,i1); end; end
for i2=1:n; for j2=1:n; s=0;
for $k = 1 : n; ss = QT(i2, k) * A(k, j2)$;
s=s+ss; end; QTA(i2,j2)=s; end; end
for i3=1:n; for j3=1:n; s=0;
for k=1:n; $ss = QTA(i3, k) * Q(k, j3)$; s=s+ss; end
A(i3,j3)=s; end; end; QQ=[QQ,Q]; end; D=A
y=[]; for k=1:n; yy=A(k,k); y=[y;yy]; end; eigvals=y
x=Q; if $(n > 2)$ % Compute eigenvectors
x(1:n,1:n)=QQ(1:n,1:n);
for $c = n + 1 : n : n * n$; xx(1:n,1:n)=QQ(1:n,c:n+c-1);
for i=1:n; for j=1:n; s=0;
for k=1:n; $ss = x(i, k) * xx(k, j)$; s=s+ss; end
x1(i,j)=s; end; end; x=x1; end; end

Program 6.6

MATLAB m-file for the Sturm Sequence method
function sol=SturmS(A)
[n,n] = size(A);
ff(1,:)=[1 0 0 0 0]; ff(2,:)=[A(1,1) -1 0 0 0];
for i=3:n+1; h=[A(i-1,i-1) -1];
$ff(i, 1) = h(1) * ff(i - 1, 1) - A(i - 1, i - 2)\hat{}\ 2 * ff(i - 2, 1)$;
for z=2:n+1
$ff(i, z) = h(1) * ff(i - 1, z) + h(2) * ff(i - 1, z - 1) - A(i - 1, i - 2)\hat{}\ 2 * ff(i - 2, z)$; end; end
for i=1:n+1; y(i)=ff(n+1,n+2-i);end;
eigval=roots(y)

Program 6.7

MATLAB m-file for the Given's method

function sol=Given(A)

$[n, n] = size(A)$; for i=1:n; for j=i+2:n

$t = -atan(A(i, j)/A(i, i + 1))$;

$c = cos(t); s = sin(t)$;

for ii=1:n; for jj=1:n;

$Q(ii, jj) = 0.0$; if $(ii == jj)$ $Q(ii, jj) = 1.0$; end; end;end

$Q(i+1, i+1) = c; Q(i+1, j) = s; Q(j, i+1) = -s; Q(j, j) = c$;

for i1=1:n; for j1=1:n;

$QT(i1, j1) = Q(j1, i1)$; end; end

for i2=1:n; for j2=1:n; s=0;

for k=1:n; $ss = QT(i2, k) * A(k, j2)$;

$s = s + ss$; end; $QTA(i2, j2) = s$; end; end

for j3=1:n; s=0; for k=1:n

$ss = QTA(i3, k) * Q(k, j3); s = s + ss$; end

$A(i3, j3) = s$; end; end; end; end

disp('Tridiagonal matrix of A is:') T=A:

Program 6.8

MATLAB m-file for the Householder method

function sol=HHHM(A)

$[n, n] = size(A); Q = eye(n)$; for k=1:n-2

$alfa = sign(A(k + 1, k)) * sqrt(A((k + 1) : n, k)' * A((k + 1) : n, k))$;

$w = zeros(n, 1)$;

$w(k+1, 1) = A(k+1, k)+alfa; w((k+2) : n, 1) = A((k+2) : n, k)$;

$P = eye(n)-2*w*w'/(w'*w); Q = Q*P; A = P*A*P; end$

T=A % this is the tridiagonal matrix

% using Sturm sequence method

ff(1,:)=[1 0 0 0 0]; ff(2,:)=[A(1,1) -1 0 0 0];

for i=3:n+1

$h = [A(i-1, i-1) - 1]; ff(i, 1) = h(1) * ff(i-1, 1) - A(i - 1, i-2)$ ^ $2*ff(i-2, 1)$;

for z=2:n+1

$ff(i, z) = h(1) * ff(i-1, z) + h(2) * ff(i-1, z-1) - A(i - 1, i-2)$^ 2*$ff(i-2, z)$; end; end

for i=1:n+1; y(i)=ff(n+1,n+2-i); end; alfa; u; Q; eigval=roots(y)

Program 6.9

MATLAB m-file for the QR method

function sol=QRM(A)

[n,n]=size(A); M=0; for i=1:n; for j=1:n

if $j < i$; M=M+1;end; end;end;

for i=1:n; I(i,i)=1;end

dd=1; while $dd > 0.0001$; Q=I; Qs=I; kk=1;

for i=2:n; for j=1:i-1

$t = -atan((A(i,j)/A(j,j)))$; $Q(j,j) = \cos(t)$;

$Q(j,i) = sin(t)$; $Q(i,j) = -sin(t)$; $Q(i,i) = \cos(t)$; Q;

$A = Q' * A$; $Qs(:,:,kk) = Q$; $kk = kk + 1$; $Q = I$; end; end;

$Q = Qs(:,:,M)$; $forc = M - 1 : -1 : 1$

$Q = Qs(:,:,c) * Q$; end; $R = A$; Q; $A = R * Q$; $k = 1$;

for i=1:n; for j=1:n

if $j < i$; $m(k) = A(i,j)$; $k = k + 1$; end; end; end;

m; $dd = max(abs(m))$; end; for i=1:n; eigvals(i)=A(i,i); end

Program 6.10

MATLAB m-file for the Upper Hessenberg Form

function sol=hes(A)

$n = length(A(1,:))$; for i = 1:n-1; $m = eye(n)$;

$[wj] = max(abs(A(i+1:n, i)))$;

$if\ j > i + 1$;

$t = m(i+1,:)$; $m(i+1,:) = m(j,:)$;

$m(j,:) = t$; $A = m * A * m'$; end;

$m = eye(n)$; $m(i+2 : n, i+1) = -A(i+2 : n, i)/(A(i+1, i))$;

$mi = m$; $mi(i+2:n, i+1) = -m(i+2:n, i+1)$;

$A = m * A * mi$; $mesh(abs(A))$; end

Chapter 7 Approximating Functions

Program 7.1
MATLAB m-file for Aitken's method

```
function P=Aitken1(x,y,x0)
n=size(x,1);
if n==1
n=size(x,2); end
for i=1:n
P(i,1)=y(i); end
for i=2:n
t=0;
for j=2:i
t=t+1;
P(i,j)=(P(i,j-1)*(x0-x(j-1))-P(t,t)*(x0-
x(i)))/(x(i)-x(j-1));
end; end
```

Program 7.2
MATLAB m-file for the Linear Least Squares Fit

```
function [a,b]=linefit(x,y)
n=length(x); S1 = sum(x); S2 = sum(y);
S3 = sum(x. * x); S4 = sum(x. * y);
a = (n * S4 - S1 * S2)/(n * S3 - (S1) ^ 2);
b = (S3 * S2 - S4 * S1)/(n * S3 - (S1) ^ 2);
for k = 1 : n
p1 = a + b * x(k);
Error(k) = abs(p1 - y(k)); end
Error = sum(Error. * Error);
```

Program 7.3
MATLAB m-file for the Polynomial Least-Square Fit

```
function C=polyfit(x,y,n)
m=length(x);  for i = 1 : 2 * n + 1
a(i) = sum(x.^ (i - 1));end
for i = 1 : n + 1 % coefficients of vector b
b(i) = sum(y. * x.^ (i - 1)); end
for i=1:n+1; for j=1:n+1
A(i, j) = a(j + i - 1); end;end
C = A \ b'; % Solving linear system
for k = 1 : m; S = C(1); for i = 2 : m + 1
S = S + C(i). * x(k).^ (i - 1); end;
p2(k) = S; Error(k) = abs(y(k) - p2(k)); end
Error = sum(Error. * Error);
```

Program 7.4

MATLAB m-file for the Nonlinear Least Square Fit

function [A,B]=exp1fit(x,y)% Least square fit $y = ax^b$

%Transform the data from (x,y) to (X,Y), $X = log(x), Y = log(y)$;

$n = length(x); X = log(x); Y = log(y);$

$S1 = sum(X); S2 = sum(Y); S3 = sum(X. * X); S4 = sum(W. * Z);$

$B = (n * S4 - S1 * S2)/(n * S3 - (S1)\ \hat{}\ 2);$

$A = (S3 * S2 - S4 * S1)/(n * S3 - (S1)\hat{}\ 2);$

$b = B; a = exp(A);$

for k=1:n

$y = a * X(k).\hat{}\ b; Error(k) = abs(y(k) - y);$ end

$Error = sum(Error. * Error);$

Program 7.5

MATLAB m-file for the Nonlinear Least-Square Fit

function [A,B]=exp2fit(x,y) % Least square fit $y = ae^{bx}$

% Transform the data from (x,y) to (x,Y), $Y = log(Y);$

$n = length(x); Y = log(y); S1 = sum(x); S2 = sum(Y);$

$S3 = sum(x. * x); S4 = sum(x. * Y);$

$B = (n * S4 - S1 * S2)/(n * S3 - (S1)\ \hat{}\ 2);$

$A = (S3 * S2 - S4 * S1)/(n * S3 - (S1)\hat{}\ 2);$

$b = B; a = exp(A)$

for k=1:n

$y = a * exp(b * x(k)); Error(k) = abs(y(k) - y);$ end

$Error = sum(Error. * Error);$

Program 7.6

MATLAB m-file for the Least Square Plane Fit

function Sol=planefit(x,y,z)

$n = length(x); S1 = sum(x); S2 = sum(y); S3 = sum(z);$

$S4 = sum(x. * x); S5 = sum(x. * y); S6 = sum(y. * y);$

$S7 = sum(z. * x); S8 = sum(z. * y);$

$A = [S4\ S5\ S1; S5\ S6\ S2; S1\ S2\ n]; B = [S7\ S8\ S3]'; C = A \backslash B;$

for k=1:n

$z = C(1) * x(k) + C(2) * y(k) + C(3);$

$Error(k) = abs(z(k) - z);$ end

$Error = sum(Error. * Error);$

Program 7.7

MATLAB m-file for the Overdetermined Linear System
function sol=overd(A,b)
$x = (A' * A) \setminus (A' * b)$; % Solve the normal equations
sol=x;