

T.C

MARMARA ÜNİVERSİTESİ

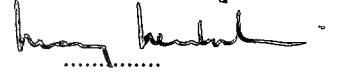
FEN BİLİMLERİ ENSTİTÜSÜ

LİE CEBİRLERİNDE C DİLİ UYGULAMALARI

SERPİL KARAGÖZ

69020

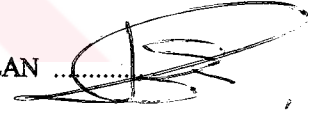
DANIŞMAN VE KOORDİNATÖR : PROF.DR.BARIŞ KENDİRLİ



JÜRİ ÜYESİ : YRD.DOÇ.DR.SİNAN KESKİN



JÜRİ ÜYESİ : YRD.DOÇ.DR.ŞEMSETTİN KILIÇARSLAN



MATEMATİK EĞİTİMİ

YÜKSEK LİSANS TEZİ

İSTANBUL 1997

ÖZET

Birinci Bölümde; Lie cebri kavramı açıklanmış ve Lie cebirlerine bir örnek verilmiştir.

İkinci Bölümde; Lie çarpımı açıklanmış ve Lie çarpımı yapan bir program hazırlanmıştır. İki örnekle program çalıştırılmış ve çıktılar program sonrasında verilmiştir.

Üçüncü Bölümde; Abelian Lie cebiri tanımlanmış , bir Lie cebirinin abelianlığını test eden program iki örnekle gösterilmiştir.

Dördüncü Bölümde; Lineer Lie cebirleri açıklanmış ve verilen bir Lie cebir elemanının hangi Lineer Lie cebirine ait olduğunu bulan programlar örneklerle verilmiştir.

Beşinci Bölümde; Yapı sabitleri anlatılmış verilen bir sayı kümesinin bir Lie cebiri oluşturup oluşturmadığını test eden bir program örneklerle verilmiştir.

Altıncı Bölümde; Lineer Lie cebirlerinin türevi açıklanmış ve verilen bir lineer operatörün türev operatörü olup olmadığını test eden bir program örneklerle birlikte verilmiştir.

Yedinci Bölümde; Bir Lie cebirinin elemanlarının adjointi tanımlanmıştır. Elemanların adjoint matrisini bulan program hazırlanmış ve örnek verilmiştir.

Sekizinci Bölümde; Killing Formunun ne olduğu ve önemi anlatılmıştır. Simplektik Lie cebirinin Killing Formunu bulan program hazırlanmıştır.

İÇİNDEKİLER

BÖLÜM	SAYFA
1-LİE CEBRİ KAVRAMI.....	1
Lie Cebirlerine Bir Örnek.....	1
Alt Lie Cebiri.....	2
2-LİE ÇARPIMI.....	2
Lie-pro.cpp.....	3
Flie-pro.cpp.....	5
Örnek 1.....	8
Örnek 2.....	9
3-ABELİAN LİE CEBİRİ.....	11
Abelian.cpp.....	12
Fabelian.cpp.....	15
Örnek 1.....	18
Örnek 2.....	20
4-LİNEER LİE CEBİRLERİ.....	24
Özel Lineer Cebir.....	25
Tek Boyutlu Ortogonal Cebir.....	25
Simplektik Lie Cebiri.....	26
Çift Boyutlu Ortogonal Cebir.....	27
Çift.cpp.....	28
Fçift.cpp.....	31
Örnek 1.....	35
Örnek 2.....	37
Tek.cpp.....	39
Ftek.cpp.....	44
Örnek 1.....	48
Örnek 2.....	51
Yeni.cpp.....	51
Fyeni.cpp.....	59
Örnek 1.....	66
Örnek 2.....	69
5-YAPI SABİTLERİ.....	72
Sabit.cpp.....	72
Fsabit.cpp.....	73
Örnek 1.....	75
Örnek 2.....	75
6-SEMİ SIMPLE LİE CEBİRİNİN TÜREVİ.....	75
Türev.cpp.....	75
Ftürev.cpp.....	80
Örnek 1.....	85

7-L NİN ELEMANLARININ ADJOİNTİ.....	85
Gaussjo.cpp.....	86
Ad.cpp.....	91
Fad.cpp.....	98
Örnek 1.....	104
8-KİLLİNG FORMU.....	107
Kmat.cpp.....	107
Fkmat.cpp.....	114
Örnek 1.....	122



LIE CEBRİ KAVRAMI

Lie cebirleri vektör uzaylarının lineer transformasyonları olarak hem değişme hem de birleşme özelliği olmayan bir operasyonla birlikte ortaya çıkmaktadırlar. Böyle bir sistemi şöyle tarif edebiliriz.

Tanım: F cismi üzerindeki L vektör uzayı ve

$$L \times L \rightarrow L$$

$(x,y) \rightarrow [x,y]$ x ve y nin komütatörü diye adlandırılan bu operasyon aşağıdaki aksiyomları sağlıyorsa bu vektör uzayına F cismi üzerinde bir Lie cebri denir.

L_1 : Komütatör operasyonu bilineerdir.

L_2 : $\forall x \in L$ için $[x,x]=0$ dir.

L_3 : $[x,[y,z]]+[y,[z,x]]+[z,[x,y]]=0$ ($x,y,z \in L$)

Üçüncü aksiyom Jacoby eşitliği diye adlandırılır.

Eğer $[x+y,x+y]$ üzerinde L_1 ve L_2 yi uygularsak

$[x+y,x+y]=0$ dir. (L_2 den dolayı)

$$[x,x+y]+[y,x+y]=[x,x]+[x,y]+[y,x]+[y,y]=0$$

$[x,x]=0$ ve $[y,y]=0$ (L_2 den dolayı)

öyleyse $[x,y]+[y,x]=0$ olmalıdır. Buradan

$$[x,y] = -[y,x] \dots \dots L_2'$$

Eğer char $F \neq 2$ ise L_2' L_2 yerine geçebilir.

Lie Cebirlerine Bir Örnek

$gl(n,F)$,F cismi üzerindeki $n \times n$ boyutlu matrislerin kümesi olsun. Komütatör operasyonumuz da

$[X,Y]=XY-YX$ (XY matris çarpımı) olsun. $gl(n,F)$ bir Lie cebridir. Şöyle ki

$$L_1: [a_1A_1+a_2A_2,B]=a_1[A_1,B]+a_2[A_2,B]$$

$$[A,b_1B_1+b_2B_2]=b_1[A,B_1]+b_2[A,B_2]$$

$$a_1,a_2,b_1,b_2 \in \mathbb{R} \quad A_1,A_2,A,B_1,B_2,B \in gl(n,F)$$

$$[a_1A_1+a_2A_2,B]=(a_1A_1+a_2A_2)B-B(a_1A_1+a_2A_2)$$

$$=a_1A_1B+a_2A_2B-Ba_1A_1-Ba_2A_2$$

$$=a_1A_1B-a_1BA_1+a_2A_2B-a_2BA_2$$

$$=a_1[A_1,B]+a_2[A_2,B]$$

$$[A,b_1B_1+b_2B_2]=A(b_1B_1+b_2B_2)-(b_1B_1+b_2B_2)A$$

$$=Ab_1B_1+Ab_2B_2-b_1B_1A-b_2B_2A$$

$$=b_1AB_1-b_1B_1A+b_2AB_2-b_2B_2A$$

$$=b_1[A,B_1]+b_2[A,B_2]$$

$[X, Y] = XY - YX$ operasyonu bilineerdi

$$L_2': [X, Y] = -[Y, X]$$

Sol taraftan $[X, Y] = XY - YX$

$$\text{Sağ taraftan } -[Y, X] = -(YX - XY)$$

$$= -YX + XY$$

$$= XY - YX$$

L_3 : Jacoby eşitliği

$$[X, [Y, Z]] + [Y, [Z, X]] + [Z, [X, Y]] = 0 \text{ dir.}$$

$$[X, [Y, Z]] + [Y, [Z, X]] + [Z, [X, Y]] = [X, YZ - ZY] + [Y, [ZX - XZ]] + [Z, XY - YX]$$

$$= X(YZ - ZY) - (YZ - ZY)X + Y(ZX - XZ) - (ZX - XZ)Y + Z(XY - YX) - (XY - YX)Z$$

$$= XYZ - XZY - YZX + ZYX + YZX - YXZ - ZXY + XZY + ZXY - ZYX -$$

$$XYZ + YXZ = 0$$

Görüldüğü gibi $[X, Y] = XY - YX$ operasyonu altında, $gl(n, F)$ bir Lie cebiridir.

Alt Lie Cebiri

Tanım: K, L Lie cebirinin bir alt uzayı olsun. Eğer $x, y \in K$ iken $[x, y] \in K$ oluyorsa K alt uzayı L nin bir alt Lie cebiri olur.

Yani tanımlanan operasyon alt uzay içinde kapalı kalıyorsa alt uzay bir alt Lie cebiri hâline gelir.

Açıktır ki L nin sıfırdan farklı her elemanının doğurduğu alt uzay bir boyutlu alt Lie cebri meydana getirmektedir.

Lie Çarpımı

$gl(n, F)$ F cismi üzerinde $n \times n$ boyutlu matrislerin kümesiydi. $gl(n, F)$ bir vektör uzayıdır. Bu vektör uzayının bazı da

$\{e_{ij} : 1 \leq i \leq n, 1 \leq j \leq n\}$ dir.

$$[e_{ij}, e_{kl}] = e_{ij} \cdot e_{kl} - e_{kl} \cdot e_{ij}$$

$$e_{ij} \cdot e_{kl} = e_{il} \text{ eğer } j=k$$

$$e_{ij} \cdot e_{kl} = 0 \text{ eğer } j \neq k$$

$$e_{ij} \cdot e_{kl} = \delta_{jk} \cdot e_{il}$$

$$\delta_{jk} = 1 \text{ eğer } j=k$$

$$\delta_{jk} = 0 \text{ eğer } j \neq k$$

$$[e_{ij}, e_{kl}] = e_{ij} \cdot e_{kl} - e_{kl} \cdot e_{ij}$$

$$= \delta_{jk} \cdot e_{il} - \delta_{li} \cdot e_{kj}$$

Bu yöntemle baz elemanları arasındaki Lie çarpımlarını kolayca hesaplayabiliriz.

F cismi üzerinde 2 boyutlu matrislerin kümesi olan $gl(2,F)$ ele

$$\text{ala lım } gl(2,F) = \left\{ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \mid a,b,c,d \in F \right\}$$

$$e_{11} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad e_{12} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad e_{21} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad e_{22} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$[e_{11}, e_{12}] = \delta_{11} \cdot e_{12} - \delta_{21} \cdot e_{11} = 1 \cdot e_{12} - 0 \cdot e_{11} = e_{12}$$

$$[e_{11}, e_{11}] = 0$$

$$[e_{11}, e_{21}] = \delta_{12} \cdot e_{11} - \delta_{11} \cdot e_{21} = 0 \cdot e_{11} - 1 \cdot e_{21} = -e_{21} \dots\dots\dots$$

Bu şekilde baz elemanları arasındaki bütün Lie çarpımlarını pratik olarak hesaplayabiliriz.

Verilen iki matrisin Lie çarpımlarını hesaplamak eğer boyut büyükse zor olmaktadır. Bu durumda bu hesabı yapacak bir program hazırlayalım.

Lie-pro.cpp

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y);
void subtr(float **z,float **x,float **y);
int n;
main()
{
float **a,**b,**m1,**m2,**s;
printf("This is LIE PRODUCT PROGRAMME\n");
printf("Enter dimension of your matrix\n");
scanf("%d",&n);
a=matrix(1,n,1,n);
b=matrix(1,n,1,n);
m1=matrix(1,n,1,n);
m2=matrix(1,n,1,n);
s=matrix(1,n,1,n);
printf("Enter your first matrix\n");
init_matrix(a,n);
printf("First matrix is\n");
display(a,n);
```

```

printf("Enter your second matrix\n");
init_matrix(b,n);
printf("Second matrix is\n");
display(b,n);
printf("Multiplication first and second matrix\n");
mult(m1,a,b);
display(m1,n);
printf("Multiplication second and first matrix\n");
mult(m2,b,a);
display(m2,n);
subt(s,m1,m2);
printf("LIE PRODUCT IS\n");
display(s,n);
return 0;
}
void init_matrix(float **x,int k)
{ float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{for(j=1;j<=k;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()")

```

```

m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void mult(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++){
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
for(k=1;k<=n;k++)
z[i][j]+=x[i][k]*y[k][j];
}
}
void subt(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++){
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=x[i][j]-y[i][j];
}
}

```

Programımızın çıktısını bir dosya içine yazılacak şekilde programımızı biraz değiştirelim.

Flie-pro.cpp

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);

```

```

void mult(float **z,float **x,float **y);
void subt(float **z,float **x,float **y);
int n;
FILE *fp;
main()
{
if((fp=fopen("Flie-pro","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
float **a,**b,**m1,**m2,**s;
fprintf(fp,"This is LIE PRODUCT PROGRAMME\n");
printf("Enter dimension of your matrix\n");
scanf("%d",&n);
a=matrix(1,n,1,n);
b=matrix(1,n,1,n);
m1=matrix(1,n,1,n);
m2=matrix(1,n,1,n);
s=matrix(1,n,1,n);
printf("Enter your first matrix\n");
init_matrix(a,n);
fprintf(fp,"First matrix is\n");
display(a,n);
printf("Enter your second matrix\n");
init_matrix(b,n);
fprintf(fp,"Second matrix is\n");
display(b,n);
fprintf(fp,"Multiplication first and second matrix\n");
mult(m1,a,b);
display(m1,n);
fprintf(fp,"Multiplication second and first martix\n");
mult(m2,b,a);
display(m2,n);
subt(s,m1,m2);
fprintf(fp,"LIE PRODUCT IS\n");
display(s,n);
fclose(fp);
return 0;
}
void init_matrix(float **x,int k)
{ float r;

```

```

int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"\r");
{for(j=1;j<=k;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void mult(float **z,float **x,float **y)
{

```

```

int i,j,k;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
for(k=1;k<=n;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void subt(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=x[i][j]-y[i][j];
}

```

Örnek 1:

3 boyutlu iki matrisi ele alalım ve Lie çarpımlarını hesaplayalım.

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 1 \\ -1 & 0 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 3 \\ 1 & -2 & 0 \\ 3 & 1 & 4 \end{bmatrix}$$

This is LIE PRODUCT PROGRAMME

First matrix is

1.000000 2.000000 0.000000

3.000000 4.000000 1.000000

-1.000000 0.000000 2.000000

Second matrix is

1.000000 0.000000 3.000000

1.000000 -2.000000 0.000000

3.000000 1.000000 4.000000

Multiplication first and second matrix

3.000000 -4.000000 3.000000

10.000000 -7.000000 13.000000

5.000000 2.000000 5.000000

Multiplication second and first matrix

-2.000000 2.000000 6.000000

-5.000000 -6.000000 -2.000000

2.000000 10.000000 9.000000

LIE PRODUCT IS

5.000000 -6.000000 -3.000000

15.000000 -1.000000 15.000000

3.000000 -8.000000 -4.000000

Örnek 2:

4x4 boyutlu iki matrisin Lie çarpımını yaptığımızda ise şu çıktıyı alıyoruz.

This is LIE PRODUCT PROGRAMME

First matrix is

1.000000 2.000000 6.000000 -8.000000
7.000000 5.000000 4.000000 -7.000000
5.000000 6.000000 7.000000 -66.000000
-4.000000 3.000000 2.000000 4.000000

Second matrix is

12.000000 4.000000 6.000000 8.000000
0.000000 9.000000 8.000000 7.000000
6.000000 5.000000 4.000000 5.000000
6.000000 78.000000 3.000000 4.000000

Multiplication first and second matrix

0.000000 -572.000000 22.000000 20.000000
66.000000 -453.000000 77.000000 83.000000
-294.000000 -5039.000000 -92.000000 -147.000000
-12.000000 333.000000 20.000000 15.000000

Multiplication second and first matrix

38.000000 104.000000 146.000000 -488.000000
 75.000000 114.000000 106.000000 -563.000000
 41.000000 76.000000 94.000000 -327.000000
 551.000000 432.000000 377.000000 -776.000000

LIE PRODUCT IS

-38.000000 -676.000000 -124.000000 508.000000
 -9.000000 -567.000000 -29.000000 646.000000
 -335.000000 -5115.000000 -186.000000 180.000000
 -563.000000 -99.000000 -357.000000 791.000000

Abelian Lie Cebiri

Tanım: F cismi üzerindeki her vektör uzayı $[x,y]=0 \quad \forall x,y \in V$ operasyonu bir Lie cebiridir. Böyle Lie cebirlerine abelian Lie cebiri denir.

Verilen bir Lie cebirinin abelian olup olmadığını anlamak için tanıma göre Lie cebirinin her elemanını birbiriyle çarpmak gerekmektedir.

Lie cebirinin her elemanı baz elemanlarının bir lineer kombinasyonudur. O halde bir Lie cebirinin abelian olup olmadığını anlamak için yalnızca baz elemanlarını birbiriyle çarpmak ve sonucu değerlendirmek yeterli olacaktır. Eğer bütün çarpımlar 0 oluyorsa Lie cebiri abeliandır.

Baz elemanlarını indeksleyip birbiriyle çarpımlarını yapan ve sonucun 0 olup olmadığını kontrol eden bir program, Lie cebirinin abelianlığını test edecektir.

Abelian.cpp

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y);
void subt(float **z,float **x,float **y);
int n,l dim;
main()
{
int i,j,k,w;
float **b[100],**l[100][100],**m1[100][100];
printf("Enter dimension of lie algebra\n");
scanf("%d",&l dim);
printf("Enter dimension of base elements as matrices\n");
scanf("%d",&n);
for(i=1;i<=l dim;i++)
{
b[i]=matrix(1,n,1,n);
}
for(i=1;i<=l dim;i++)
for(j=1;j<=l dim;j++)
{
{
l[i][j]=matrix(1,n,1,n);
m1[i][j]=matrix(1,n,1,n);
}
}
for(i=1;i<=l dim;i++)
{
printf("Enter the base element b[%d]\n",i);
init_matrix(b[i],n);
display(b[i],n);
}
for(i=1;i<=l dim;i++)
for(j=1;j<=l dim;j++)

```

```

{
{
mult(m1[i][j],b[i],b[j]);
}
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{printf("Lie product of l[%d][%d]\n",i,j);
{
subt(l[i][j],m1[i][j],m1[j][i]);
display(l[i][j],n);
}
printf("\n\n");
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
for(k=1;k<=ldim;k++)
for(w=1;w<=ldim;w++)
if(l[i][j][k][w]!=0.000000)
{
printf("It is not an ABELIAN LIE ALGEBRA\n");
return 0;
}
printf("It is an ABELIAN LIE ALGEBRA\n");
return 0;
}
void init_matrix(float **x,int k)
{ float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)

```

```

{for(j=1;j<=k;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void mult(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
for(k=1;k<=n;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void sub(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=0;
for(i=1;i<=n;i++)

```

```
for(j=1;j<=n;j++)
z[i][j]=x[i][j]-y[i][j]; }
```

Bu programın çıktısını dosyalayalım ve iki örnekle programı çalıştıralım.

Fabelian.cpp

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y);
void subt(float **z,float **x,float **y);
int n,ldim;
FILE *fp;
main()
{
if((fp=fopen("Fabelian","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
int i,j,k,w;
float **b[100],**l[100][100],**m1[100][100];
printf("Enter dimension of lie algebra\n");
scanf("%d",&ldim);
printf("Enter dimension of base elements as matrices\n");
scanf("%d",&n);
for(i=1;i<=ldim;i++)
{
b[i]=matrix(1,n,1,n);
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
l[i][j]=matrix(1,n,1,n);
m1[i][j]=matrix(1,n,1,n);
```

```

}
}
for(i=1;i<=ldim;i++)
{
printf("Enter the base element b[%d]\n",i);
init_matrix(b[i],n);
display(b[i],n);
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
mult(m1[i][j],b[i],b[j]);
}
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{fprintf(fp,"Lie product of l[%d][%d]\n",i,j);
{
subt(l[i][j],m1[i][j],m1[j][i]);
display(l[i][j],n);
}
printf("\n\n");
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
for(k=1;k<=ldim;k++)
for(w=1;w<=ldim;w++)
if(l[i][j][k][w]!=0.000000)
{
fprintf(fp,"It is not an ABELIAN LIE ALGEBRA\n");
return 0;
}
fprintf(fp,"It is an ABELIAN LIE ALGEBRA\n");
return 0;
}
void init_matrix(float **x,int k)
{ float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)

```

```

{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void mult(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
for(k=1;k<=n;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void sub(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=0;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
z[i][j]=x[i][j]-y[i][j];}

```

Örnek 1:

Bir Lie cebirinin baz elemanlarını girelim ve programımızı çalıştıralım.

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

1.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 1.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 1.000000

Lie product of l[1][1]

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of l[1][2]

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[1][3]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[2][1]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[2][2]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[2][3]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[3][1]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[3][2]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $\mathfrak{l}[3][3]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

It is an ABELIAN LIE ALGEBRA.

Örnek 2:

Bir Lie cebirinin baz elemanlarını girelim ve programımızı çalıştıralım.

1.000000 0.000000 0.000000

0.000000 -1.000000 0.000000

0.000000 0.000000 0.000000

1.000000 0.000000 0.000000

0.000000 1.000000 0.000000

1.000000 0.000000 0.000000

1.000000 0.000000 1.000000

1.000000 0.000000 1.000000

-1.000000 0.000000 1.000000

-1.000000 0.000000 0.000000

0.000000 1.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[1][1]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[1][2]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

-1.000000 0.000000 0.000000

Lie product of $l[1][3]$

0.000000 0.000000 1.000000

-2.000000 0.000000 -1.000000

1.000000 0.000000 0.000000

Lie product of $l[1][4]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[2][1]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

1.000000 0.000000 0.000000

Lie product of $l[2][2]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[2][3]$

-1.000000 0.000000 1.000000

-1.000000 0.000000 1.000000

1.000000 0.000000 1.000000

Lie product of $l[2][4]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

-1.000000 0.000000 0.000000

Lie product of $l[3][1]$

0.000000 0.000000 -1.000000

2.000000 0.000000 1.000000

-1.000000 0.000000 0.000000

Lie product of $l[3][2]$

1.000000 0.000000 -1.000000

1.000000 0.000000 -1.000000

-1.000000 0.000000 -1.000000

Lie product of $l[3][3]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of $l[3][4]$

0.000000 0.000000 1.000000

-2.000000 0.000000 -1.000000

1.000000 0.000000 0.000000

Lie product of $l[4][1]$

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Lie product of l[4][2]

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

1.000000 0.000000 0.000000

Lie product of l[4][3]

0.000000 0.000000 -1.000000

2.000000 0.000000 1.000000

-1.000000 0.000000 0.000000

Lie product of l[4][4]

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

It is not an ABELIAN LIE ALGEBRA

Lineer Lie Cebirleri

V, F , cismi üzerinde n boyutlu bir vektör uzayı olsun. $\text{End } V$ ile boyutu n^2 olan $V \rightarrow V$ ye bütün lineer transformasyonların kümesini gösterelim. $\text{End } V$ lineer operatör toplamı ve kompozisyonu altında bir halka oluşturur.

$$[x, y] = xy - yx \quad x, y \in \text{End } V$$

Bu operasyonla $\text{End } V$ F cismi üzerinde bir Lie cebiri haline gelir.

$\text{End } V$ yerine Genel Lie Cebirler anlamına gelen $\text{gl}(V)$ yi kullanacağız. Lie cebiri $\text{gl}(V)$ nin her alt Lie cebirine Lineer Lie Cebiri denir. Lineer Lie cebirleri klasik cebirler diye adlandırılan dört bölüme ayrılırlar. A_l, B_l, C_l ve D_l diye gösterilirler.

A_l: Özel Lineer Cebir

$\dim V = l+1$ ve $sl(l+1, F)$ izi 0 olan endomorfizmaların kümesi olsun.

$$\text{Tr}(xy) = \text{Tr}(yx) \text{ ve}$$

$\text{Tr}(x+y) = \text{Tr}(x) + \text{Tr}(y)$ olduğundan $sl(l+1, F)$ $gl(V)$ nin alt cebiridir. Hatta $sl(l+1, F)$ boyutu en fazla $(l+1)^2 - 1$ olduğundan $gl(V)$ nin öz alt cebiridir.

Standart baz = $\{e_{ij} \text{ eğer } i \neq j, h_i = e_{ii} - e_{i+1, i+1} \ 1 \leq i \leq l, 1 \leq j \leq l\}$ toplam olarak $l + (l+1)^2 - (l+1) = l(l+2)$ elemandan oluşmaktadır. Öyleyse $\dim A_l = l(l+2)$ dir.

$$A_l = \{T: T \in gl(l+1, F) \quad \text{Tr } T = 0\}$$

B_l Tek Boyutlu Ortogonal Cebir

$\dim V = 2l+1$ olsun. f de yozlaşmamış simetrik bilinear form olsun. Bu formun matrisi

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & I_l \\ 0 & I_l & 0 \end{bmatrix} \text{ dir.}$$

$f(x(v), w) = -f(v, x(w))$ eşitliğini sağlayan V nin bütün endomorfizmalarına tek boyutlu ortogonal cebir denir.

Böyle bir endomorfizmanın nasıl olması gerektiğinin analizini şu şekilde yapabiliriz.

$$x = \begin{bmatrix} a & b_1 & b_2 \\ c_1 & M & N \\ c_2 & P & Q \end{bmatrix} \Rightarrow x^t S = -Sx$$

- $a \in F$, b_1 $1 \times l$ boyutlu matris
- b_2 $1 \times l$ boyutlu matris
- c_1 $l \times 1$ boyutlu matris
- c_2 $l \times 1$ boyutlu matris
- M, N, P, Q $l \times l$ boyutlu matris

$$x^t S = \begin{bmatrix} \bar{a} & c_1^t & c_2^t \\ b_1^t & M^t & P^t \\ b_2^t & N^t & Q^t \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & I_l \\ 0 & I_l & 0 \end{bmatrix} = \begin{bmatrix} a & c_2^t & c_1^t \\ b_1^t & P^t & M^t \\ b_2^t & Q^t & N^t \end{bmatrix}$$

$$-Sx = - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & I_l \\ 0 & I_l & 0 \end{bmatrix} \begin{bmatrix} a & b_1 & b_2 \\ c_1 & M & N \\ c_2 & P & Q \end{bmatrix} = - \begin{bmatrix} a & b_1 & b_2 \\ c_2 & P & Q \\ c_1 & M & N \end{bmatrix}$$

$$\begin{bmatrix} a & c_2 & c_1 \\ b_1^t & P^t & M^t \\ b_2^t & Q^t & N^t \end{bmatrix} = - \begin{bmatrix} a & b_1 & b_2 \\ c_2 & P & Q \\ c_1 & M & N \end{bmatrix} \Leftrightarrow$$

$$P = -P, \quad N = -N, \quad c_2 = -b_1, \quad Q = -M, \quad a = 0, \quad c_1 = b_2$$

Sonuç olarak tek boyutlu bir ortogonal cebir aşağıdaki gibi ifade edilebilir.

$$Q(2l+1, F) = \begin{bmatrix} 0 & b_1 & b_2 \\ c_1 & M & N \\ c_2 & P & Q \end{bmatrix} : \begin{array}{l} c_2 = -b_1, P^t = -P, c_1^t = -b_2, \\ N = -N^t, Q = -M^t \\ a = 0 \end{array}$$

C_l: Simplektik Lie Cebiri

Dim V = 2l olsun.

$S = \begin{bmatrix} 0 & I_l \\ -I_l & 0 \end{bmatrix}$ matrisiyle V üzerinde yozlaşmamış skew-simetrik bir form tanımlarsak

$f(x(v), w) = -f(v, x(w))$ eşitliğini sağlayan endomorfizmalar simplektik Lie cebirini oluşturur. Yine böyle bir endomorfizmanın nasıl olması gerektiğinin analizini yapalım.

$$x = \begin{bmatrix} M & N \\ P & Q \end{bmatrix} \quad M, N, P, Q \in \text{gl}(l, F)$$

$$Sx = -x^t S$$

$$\begin{bmatrix} 0 & I_l \\ -I_l & 0 \end{bmatrix} \begin{bmatrix} M & N \\ P & Q \end{bmatrix} = - \begin{bmatrix} M & P \\ N & Q \end{bmatrix} \begin{bmatrix} 0 & I_l \\ -I_l & 0 \end{bmatrix}$$

$$\begin{bmatrix} P & Q \\ -M & -N \end{bmatrix} = - \begin{bmatrix} -P^t & M^t \\ -Q^t & N^t \end{bmatrix} \quad \Leftrightarrow \begin{matrix} P = P^t, Q = -M^t \\ N = N^t \end{matrix}$$

Baz = $\{e_{ii} - e_{l+1, l+1} \mid 1 \leq i \leq l\} \cup \{e_{ij} - e_{l+j, l+i} \mid 1 \leq i \neq j \leq l\}$ bazdaki eleman sayısı $l^2 - l$ dir. Yani $\dim \text{sp}(2l, F) = l^2 - l$ dir.

2l: Çift Boyutlu Ortogonal Cebir

$\dim V = 2l$ olsun. f yozlaşmamış simetrik bilineer form olsun.

$$S = \begin{bmatrix} 0 & I_l \\ I_l & 0 \end{bmatrix} \quad x = \begin{bmatrix} M & N \\ P & Q \end{bmatrix}$$

$$f(x(v), w) = -f(v, x(w))$$

Şimdi bu endomorfizmanın nasıl olması gerektiğinin analizini

yapalım.

$$x^t S = -Sx$$

$$\begin{bmatrix} M^t & P^t \\ N^t & Q^t \end{bmatrix} \begin{bmatrix} 0 & I_l \\ I_l & 0 \end{bmatrix} = - \begin{bmatrix} 0 & I_l \\ I_l & 0 \end{bmatrix} \begin{bmatrix} M & N \\ P & Q \end{bmatrix}$$

$$\begin{bmatrix} P^t & M^t \\ Q^t & N^t \end{bmatrix} = - \begin{bmatrix} P & Q \\ M & N \end{bmatrix} \quad \Leftrightarrow \begin{matrix} P^t = -P, Q^t = -M \\ N = -N^t \end{matrix}$$

Baz: $\{e_{l+i, j} \mid 1 \leq i \leq l, 1 \leq j \leq l, j > i\} \cup \{e_{i, l+j} \mid 1 \leq i \leq l, 1 \leq j \leq l, j > i\} \cup$

$\{e_{11} - e_{l+1, l+1}, e_{22} - e_{l+2, l+2}, \dots, e_{ll} - e_{2l, 2l}, e_{ij} - e_{j+l, l+i} \mid i \neq j\}$

$\dim Q(2l, F) = (l^2 - l)/2 + (l^2 - l)/2 + l^2 = l(2l - 1)$

Verilen bir matrisin hangi Lie cebirine ait olduğunu hesaplamak, bu hesabı elle yapmak, matrisin boyutu büyükse, zaman alır. Öncelikle boyutu 4 olan bir matrisin hangi lie cebirine ait olduğunu bulan bir program hazırlayalım.

Cift.cpp

```

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define DIM 4
#define l DIM/2
void init_matrix(float **x);
void display(float **x,int k);
void trans(float **x,float **y,int m,int n);
void ntrans(float **x,float **y,int m,int n);
void part_m(float **x,float **y,int k,int t);
float trace(float **x);
float **matrix(int nrl,int nrh,int ncl,int nch);
int compare(float **x,float **y,int m,int n);
int i,j;
main()
{
float sum;
float **ap,**mp,**np,**qp,**rp,**t1p,**t2p,**t3p,**t4p,**t5p;
ap=matrix(1,DIM,1,DIM);
mp=matrix(1,1,1,1);
np=matrix(1,1,1,1);
qp=matrix(1,1,1,1);
rp=matrix(1,1,1,1);
t1p=matrix(1,1,1,1);
t2p=matrix(1,1,1,1);
t3p=matrix(1,1,1,1);
t4p=matrix(1,1,1,1);
t5p=matrix(1,1,1,1);
init_matrix(ap);
display(ap,DIM);
sum=trace(ap);
while(sum!=0)
{
printf("YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
return 0;
}
printf(" YOUR MATRIX IS IN SPECIAL LINEAR ALGEBRA\n");
part_m(mp,ap,0,0);
printf("MATRIX M IS\n");
display(mp,l);

```

```

part_m(np,ap,0,l);
printf("MATRIX N IS\n");
display(np,l);
part_m(qp,ap,l,0);
printf("MATRIX Q IS\n");
display(qp,l);
part_m(rp,ap,l,l);
printf("MATRIX R IS\n");
display(rp,l);
ntrans(t1p,mp,l,l);
printf("NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t1p,l);
ntrans(t2p,np,l,l);
printf("NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t2p,l);
ntrans(t3p,qp,l,l);
printf("NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t3p,l);
trans(t4p,np,l,l);
printf("TRANSPOSE OF MATRIX N IS\n");
display(t4p,l);
trans(t5p,qp,l,l);
printf("TRANSPOSE OF MATRIX Q IS\n");
display(t5p,l);
while (compare(qp,t3p,l,l)==1 && compare(np,t2p,l,l)==1/
&& compare(rp,t1p,l,l)==1)
{printf("IT IS ALSO IN EVEN DIMENTIONAL ORTHOGONAL
ALGEBRA\n");
return 0;}
while (compare(np,t4p,l,l)==1 && compare(qp,t5p,l,l)==1/
&& compare(rp,t1p,l,l)==1)
{printf("IT IS ALSO IN SYMPLECTIC ALGEBRA\n");
return 0;}
return 0;
}
void init_matrix(float **x)
{ float r;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)

```

```

{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void ntrans(float **y,float **x,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
y[i][j]=-x[j][i];
}
int compare(float **x,float **y,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if (x[i][j]!=y[i][j])
return 0;
return 1;
}
void display(float **x,int k)
{
for(i=1;i<=k;i++)
{for(j=1;j<=k;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
void trans(float **y,float **x,int m,int n)
{for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{y[i][j]=x[j][i];
}
}
float trace(float **x)
{float sum=0;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)
{if(i==j) sum+=x[i][j];
}
printf("The trace is %f",sum);
return sum;
}

```

```

}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void part_m(float **x,float **y,int t,int k)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[t+i][k+j];
}

```

Programımızın çıktısını dosyalayalım ve programımızı yeniden yazalım.

Fçift.cpp

```

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define DIM 4
#define l DIM/2
void init_matrix(float **x);
void display(float **x,int k);
void trans(float **x,float **y,int m,int n);
void ntrans(float **x,float **y,int m,int n);
void part_m(float **x,float **y,int k,int t);
float trace(float **x);
float **matrix(int nrl,int nrh,int ncl,int nch);
int compare(float **x,float **y,int m,int n);
int i,j;

```

```

FILE *fp;
main()
{
if((fp=fopen("Cift","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
float sum;
float **ap,**mp,**np,**qp,**rp,**t1p,**t2p,**t3p,**t4p,**t5p;
ap=matrix(1,DIM,1,DIM);
mp=matrix(1,1,1,1);
np=matrix(1,1,1,1);
qp=matrix(1,1,1,1);
rp=matrix(1,1,1,1);
t1p=matrix(1,1,1,1);
t2p=matrix(1,1,1,1);
t3p=matrix(1,1,1,1);
t4p=matrix(1,1,1,1);
t5p=matrix(1,1,1,1);
init_matrix(ap);
display(ap,DIM);
sum=trace(ap);
while(sum!=0)
{
fprintf(fp,"\r");
fprintf(fp,"YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
return 0;
}
fprintf(fp,"\r");
fprintf(fp," YOUR MATRIX IS IN SPECIAL LINEAR ALGEBRA\n");
part_m(mp,ap,0,0);
fprintf(fp,"MATRIX M IS\n");
display(mp,1);
part_m(np,ap,0,1);
fprintf(fp,"MATRIX N IS\n");
display(np,1);
part_m(qp,ap,1,0);
fprintf(fp,"MATRIX Q IS\n");
display(qp,1);
part_m(rp,ap,1,1);
fprintf(fp,"MATRIX R IS\n");

```

```

display(rp,l);
ntrans(t1p,mp,l,l);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t1p,l);
ntrans(t2p,np,l,l);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t2p,l);
ntrans(t3p,qp,l,l);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t3p,l);
trans(t4p,np,l,l);
fprintf(fp,"TRANSPOSE OF MATRIX N IS\n");
display(t4p,l);
trans(t5p,qp,l,l);
fprintf(fp,"TRANSPOSE OF MATRIX Q IS\n");
display(t5p,l);
while (compare(qp,t3p,l,l)==1 && compare(np,t2p,l,l)==1 /
&& compare(rp,t1p,l,l)==1)
{
fprintf(fp,"\r");
fprintf(fp,"IT IS ALSO IN EVEN DIMENTIONAL ORTHOGONAL
ALGEBRA\n");
return 0;}
while (compare(np,t4p,l,l)==1 && compare(qp,t5p,l,l)==1/
&& compare(rp,t1p,l,l)==1)
{
fprintf(fp,"\r");
fprintf(fp,"IT IS ALSO IN SYMPLECTIC ALGEBRA\n");
return 0;}
return 0;
}
void init_matrix(float **x)
{ float r;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void ntrans(float **y,float **x,int m,int n)
{

```

```

for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
y[i][j]=-x[j][i];
}
int compare(float **x,float **y,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if (x[i][j]!=y[i][j])
return 0;
return 1;
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"\r");
{for(j=1;j<=k;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
void trans(float **y,float **x,int m,int n)
{for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{y[i][j]=x[j][i];
}
}
float trace(float **x)
{float sum=0;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)
{if(i==j) sum+=x[i][j];
}
printf("The trace is %f",sum);
return sum;
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{

```

```

int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void part_m(float **x,float **y,int t,int k)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[t+i][k+j];
}

```

Örnek1:

4x4 boyutlu bir matrisle programımızı çalıştıralım.

```

1.000000 2.000000 0.000000 3.000000
-1.000000 3.000000 3.000000 0.000000
1.000000 2.000000 -1.000000 1.000000
2.000000 1.000000 -2.000000 -3.000000

```

YOUR MATRIX IS IN SPECIAL LINEAR ALGEBRA

MATRIX M IS

```

1.000000 2.000000
-1.000000 3.000000

```

MATRIX N IS

0.000000 3.000000

3.000000 0.000000

MATRIX Q IS

1.000000 2.000000

2.000000 1.000000

MATRIX R IS

-1.000000 1.000000

-2.000000 -3.000000

NEGATIVE TRANSPOSE OF MATRIX M IS

-1.000000 1.000000

-2.000000 -3.000000

NEGATIVE TRANSPOSE OF MATRIX N IS

-0.000000 -3.000000

-3.000000 -0.000000

NEGATIVE TRANSPOSE OF MATRIX Q IS

-1.000000 -2.000000

-2.000000 -1.000000

TRANSPOSE OF MATRIX N IS

0.000000 3.000000

3.000000 0.000000

TRANSPOSE OF MATRIX Q IS

1.000000 2.000000

2.000000 1.000000

IT IS ALSO IN SYMPLECTIC ALGEBRA

Örnek 2:

Boyutu 4x4 olan başka bir matrisi ele alalım.

5.000000 7.000000 6.000000 3.000000

2.000000 -5.000000 3.000000 2.000000

0.000000 9.000000 0.000000 2.000000

4.000000 6.000000 5.000000 0.000000

YOUR MATRIX IS IN SPECIAL LINEAR ALGEBRA

MATRIX M IS

5.000000 7.000000

2.000000 -5.000000

MATRIX N IS

6.000000 3.000000

3.000000 2.000000

MATRIX Q IS

0.000000 9.000000

4.000000 6.000000

MATRIX R IS

0.000000 2.000000

5.000000 0.000000

NEGATIVE TRANSPOSE OF MATRIX M IS

-5.000000 -2.000000

-7.000000 5.000000

NEGATIVE TRANSPOSE OF MATRIX N IS

-6.000000 -3.000000

-3.000000 -2.000000

NEGATIVE TRANSPOSE OF MATRIX Q IS

-0.000000 -4.000000

-9.000000 -6.000000

TRANSPOSE OF MATRIX N IS

6.000000 3.000000

3.000000 2.000000

TRANSPOSE OF MATRIX Q IS

0.000000 4.000000

9.000000 6.000000

Şimdi boyutu tek sayı olan matrisler üzerinde duralım. Bu matrislerin hangi Lie cebirine ait olduğunu hesaplayan bir program hazırlayalım.

Tek.cpp

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define DIM 5
#define l (DIM-1)/2
void init_matrix(float **x);
void display(float **x,int k,int t);
void trans(float **x,float **y,int m,int n);
void ntrans(float **x,float **y,int m,int n);
void part1(float **x,float **y,int k,int t);
```

```

void part2(float **x,float **y,int k,int t);
void part_m2(float **x,float **y,int k,int t,int z,int u);
float trace(float **x);
float **matrix(int nrl,int nrh,int ncl,int nch);
int compare(float **x,float **y,int m,int n);
int i,j;
main()
{
float sum;
float **ap,**b1p,**b2p,**c1p,**c2p,**mp,**np,**qp,**rp,**t1p,**t2p;
float **t3p,**t4p,**t5p;
ap=matrix(1,DIM,1,DIM);
mp=matrix(1,1,1,1);
np=matrix(1,1,1,1);
qp=matrix(1,1,1,1);
rp=matrix(1,1,1,1);
b1p=matrix(1,1,1,1);
b2p=matrix(1,1,1,1);
c1p=matrix(1,1,1,1);
c2p=matrix(1,1,1,1);
t1p=matrix(1,1,1,1);
t2p=matrix(1,1,1,1);
t3p=matrix(1,1,1,1);
t4p=matrix(1,1,1,1);
t5p=matrix(1,1,1,1);
printf("FIRST COEFFICIENT MUST BE=0\n");
init_matrix(ap);
display(ap,DIM,DIM);
sum=trace(ap);
while(sum!=0)
{
printf("YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
return 0;
}
part1(b1p,ap,1,0);
printf("MATRIX B1 IS\n");
display(b1p,1,1);
part1(b2p,ap,1,1);
printf("MATRIX B2 IS\n");
display(b2p,1,1);
part2(c1p,ap,1,0);

```

```

printf("MATRIX C1 IS\n");
display(c1p,l,1);
part2(c2p,ap,l,1);
printf("MATRIX C2 IS\n");
display(c2p,l,1);
part_m2(mp,ap,1,0,1,0);
printf("MATRIX M IS\n");
display(mp,l,1);
part_m2(np,ap,1,0,1,1);
printf("MATRIX N IS\n");
display(np,l,1);
part_m2(qp,ap,1,1,1,0);
printf("MATRIX Q IS\n");
display(qp,l,1);
part_m2(rp,ap,1,1,1,1);
printf("MATRIX R IS\n");
display(rp,l,1);
ntrans(t1p,c1p,l,1);
printf("NEGATIVE TRANSPOSE OF MATRIX C1 IS\n");
display(t1p,l,1);
ntrans(t2p,c2p,l,1);
printf("NEGATIVE TRANSPOSE OF MATRIX C2 IS\n");
display(t2p,l,1);
ntrans(t3p,qp,l,1);
printf("NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t3p,l,1);
ntrans(t4p,np,l,1);
printf("NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t4p,l,1);
ntrans(t5p,mp,l,1);
printf("NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t5p,l,1);
while (compare(b2p,t1p,l,1)==1 && compare(b1p,t2p,l,1)==1/
&&compare(qp,t3p,l,1)==1&& compare(np,t4p,l,1)==1/
&& compare(rp,t5p,l,1)==1)
{printf("IT IS IN ODD DIMENTIONAL ORTHOGONAL
ALGEBRA\n");
return 0;}
printf("NOT IN ODD DIMENTIONAL ORTHOGONAL ALGEBRA\n");
return 0;
}

```

```

void init_matrix(float **x)
{ float r;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void ntrans(float **y,float **x,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
y[i][j]=-x[j][i];
}
int compare(float **x,float **y,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if (x[i][j]!=y[i][j])
return 0;
return 1;
}
void display(float **x,int k,int t)
{
for(i=1;i<=k;i++)
{for(j=1;j<=t;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
void trans(float **y,float **x,int m,int n)
{for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{y[i][j]=x[j][i];
}
}
float trace(float **x)
{float sum=0;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)

```

```

    {if(i==j) sum+=x[i][j];
    }
    printf("The trace is %f",sum);
    return sum;
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
    int i;
    float **m;
    /*allocate pointers to rows*/
    m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
    if(!m) printf("allocation failure 1 in matrix()");
    m-=nrl;
    /*allocate rows and set pointers to them*/
    for(i=nrl;i<=nrh;i++){
        m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
        if(!m[i]) printf("allocation failure 2 in matrix()");
        m[i]-=ncl;
    }
    /*return pointer to array of pointers to rows*/
    return m;
}
void part_m2(float **x,float **y,int t,int k,int z,int u)
{
    for(i=1;i<=l;i++)
        for(j=1;j<=l;j++)
            x[i][j]=y[i+t+k][j+z+u];
}
void part1(float **x,float **y,int k,int t)
{
    i=1;
    for(j=1;j<=l;j++)
        x[i][j]=y[i][j+k+t];
}
void part2(float **x,float **y,int k,int t)
{
    j=1;
    for(i=1;i<=l;i++)
        x[i][j]=y[i+k+t][j];
}

```

Programımızın çıktısını dosyalayacak şekilde programı biraz değiştirelim.

Ftek.cpp

```

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define DIM 5
#define l (DIM-1)/2
void init_matrix(float **x);
void display(float **x,int k,int t);
void trans(float **x,float **y,int m,int n);
void ntrans(float **x,float **y,int m,int n);
void part1(float **x,float **y,int k,int t);
void part2(float **x,float **y,int k,int t);
void part_m2(float **x,float **y,int k,int t,int z,int u);
float trace(float **x);
float **matrix(int nrl,int nrh,int ncl,int nch);
int compare(float **x,float **y,int m,int n);
int i,j;
FILE *fp;
main()
{
if((fp=fopen("Ftek","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
float sum;
float **ap,**b1p,**b2p,**c1p,**c2p,**mp,**np,**qp,**rp,**t1p,**t2p;
float **t3p,**t4p,**t5p;
ap=matrix(1,DIM,1,DIM);
mp=matrix(1,1,1,1);
np=matrix(1,1,1,1);
qp=matrix(1,1,1,1);
rp=matrix(1,1,1,1);
b1p=matrix(1,1,1,1);
b2p=matrix(1,1,1,1);
c1p=matrix(1,1,1,1);
c2p=matrix(1,1,1,1);
t1p=matrix(1,1,1,1);

```

```

t2p=matrix(1,1,1,1);
t3p=matrix(1,1,1,1);
t4p=matrix(1,1,1,1);
t5p=matrix(1,1,1,1);
printf("FIRST COEFFICIENT MUST BE=0\n");
init_matrix(ap);
display(ap,DIM,DIM);
sum=trace(ap);
while(sum!=0)
{
fprintf(fp,"YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
return 0;
}
part1(b1p,ap,1,0);
fprintf(fp,"MATRIX B1 IS\n");
display(b1p,1,1);
part1(b2p,ap,1,1);
fprintf(fp,"MATRIX B2 IS\n");
display(b2p,1,1);
part2(c1p,ap,1,0);
fprintf(fp,"MATRIX C1 IS\n");
display(c1p,1,1);
part2(c2p,ap,1,1);
fprintf(fp,"MATRIX C2 IS\n");
display(c2p,1,1);
part_m2(mp,ap,1,0,1,0);
fprintf(fp,"MATRIX M IS\n");
display(mp,1,1);
part_m2(np,ap,1,0,1,1);
fprintf(fp,"MATRIX N IS\n");
display(np,1,1);
part_m2(qp,ap,1,1,1,0);
fprintf(fp,"MATRIX Q IS\n");
display(qp,1,1);
part_m2(rp,ap,1,1,1,1);
fprintf(fp,"MATRIX R IS\n");
display(rp,1,1);
ntrans(t1p,c1p,1,1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX C1 IS\n");
display(t1p,1,1);
ntrans(t2p,c2p,1,1);

```

```

fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX C2 IS\n");
display(t2p,1,1);
ntrans(t3p,qp,1,1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t3p,1,1);
ntrans(t4p,np,1,1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t4p,1,1);
ntrans(t5p,mp,1,1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t5p,1,1);
while (compare(b2p,t1p,1,1)==1 && compare(b1p,t2p,1,1)==1/
&&compare(qp,t3p,1,1)==1&& compare(np,t4p,1,1)==1/
&& compare(rp,t5p,1,1)==1)
{fprintf(fp,"IT IS IN ODD DIMENTIONAL ORTHOGONAL
ALGEBRA\n");
return 0;}
fprintf(fp,"NOT IN ODD DIMENTIONAL ORTHOGONAL
ALGEBRA\n");
return 0;
}
void init_matrix(float **x)
{ float r;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)
{printf("Enter the coefficent x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void ntrans(float **y,float **x,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
y[i][j]=-x[j][i];
}
int compare(float **x,float **y,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if (x[i][j]!=y[i][j])
return 0;
}

```

```

return 1;
}
void display(float **x,int k,int t)
{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"r");
{for(j=1;j<=t;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
void trans(float **y,float **x,int m,int n)
{for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{y[i][j]=x[j][i];
}
}
float trace(float **x)
{float sum=0;
for(i=1;i<=DIM;i++)
for(j=1;j<=DIM;j++)
{if(i==j) sum+=x[i][j];
}
printf("The trace is %f",sum);
return sum;
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
}
}

```

```

m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void part_m2(float **x,float **y,int t,int k,int z,int u)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[i+t+k][j+z+u];
}
void part1(float **x,float **y,int k,int t)
{i=1;
for(j=1;j<=l;j++)
x[i][j]=y[i][j+k+t];
}
void part2(float **x,float **y,int k,int t)
{j=1;
for(i=1;i<=l;i++)
x[i][j]=y[i+k+t][j];
}

```

Örnek 1:

Programımız yalnızca 5x5 boyutlu matris için geçerlidir.Boyutu 5x5 olan bir matrisi ele alıp programı çalıştırsak şu çıktıyı alırız.

```

0.000000 1.000000 2.000000 3.000000 4.000000
-3.000000 0.000000 1.000000 0.000000 4.000000
-4.000000 1.000000 0.000000 4.000000 0.000000
-1.000000 0.000000 3.000000 0.000000 2.000000
-2.000000 3.000000 0.000000 2.000000 0.000000

MATRIX B1 IS

1.000000 2.000000

```

MATRIX B2 IS

3.000000 4.000000

MATRIX C1 IS

-3.000000

-4.000000

MATRIX C2 IS

-1.000000

-2.000000

MATRIX M IS

0.000000 1.000000

1.000000 0.000000

MATRIX N IS

0.000000 4.000000

4.000000 0.000000

MATRIX Q IS

0.000000 3.000000

3.000000 0.000000

MATRIX R IS

0.000000 2.000000

2.000000 0.000000

NEGATIVE TRANSPOSE OF MATRIX C1 IS

3.000000 0.000000

NEGATIVE TRANSPOSE OF MATRIX C2 IS

1.000000 0.000000

NEGATIVE TRANSPOSE OF MATRIX Q IS

-0.000000 -3.000000

-3.000000 -0.000000

NEGATIVE TRANSPOSE OF MATRIX N IS

-0.000000 -4.000000

-4.000000 -0.000000

NEGATIVE TRANSPOSE OF MATRIX M IS

-0.000000 -1.000000

-1.000000 -0.000000

NOT IN ODD DIMENTIONAL ORTHOGONAL ALGEBRA

Örnek 2:

Başka bir matrisi ele alalım.

0.000000 9.000000 8.000000 7.000000 6.000000

5.000000 8.000000 90.000000 9.000000 8.000000

7.000000 6.000000 5.000000 4.000000 7.000000

9.000000 9.000000 9.000000 9.000000 9.000000

9.000000 9.000000 9.000000 9.000000 9.000000

YOUR MATRIX IS NOT IN ANY ALGEBRAS

Tek.cpp ve Çift.cpp programlarını birleştirelim. Boyutu tek sayı veya çift sayı olan matrislerin hangi Lie cebrine ait olduğunu bulan bir program hazırlayalım. Bu yeni program her boyuttaki matris için hesap yapabilmektedir.

Yeni.cpp

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
void init_matrix(float **x,int k);
void display(float **x,int k,int t);
void trans(float **x,float **y,int m,int n);
void ntrans(float **x,float **y,int m,int n);
void part1(float **x,float **y,int k,int t,int l);
void part2(float **x,float **y,int k,int t,int l);
void part_m1(float **x,float **y,int k,int t,int l);
void part_m2(float **x,float **y,int k,int t,int z,int u,int l);
```

```

float trace(float **x,int k);
void even_case(void);
void odd_case(void);
float **matrix(int nrl,int nrh,int ncl,int nch);
int compare(float **x,float **y,int m,int n);
int i,j;
main()
{
printf("CLASSICAL ALGEBRAS made by Serpil KARAGÖZ\n");
printf("This is a program to find that whether a given matrix belongs\n");
printf("to the one of the following algebras or not\n");
printf("An SPECIAL LINEAR ALGEBRA\n");
printf("Bn ODD DIMENSIONAL ORTHAGONAL LIE ALGEBRA\n");
printf("Cn SYMPLECTIC ALGEBRA\n");
printf("Dn EVEN DIMENSIONAL ORTHAGONAL LIE ALGEBRA\n");
printf("1.EVEN DIMENSIONAL MATRIX\n");
printf("2.ODD DIMENSIONAL MATRIX\n");
printf("ENTER YOUR COHICE 1 OR 2\n");
char ch;
ch=getchar();
switch(ch){
case '1':
even_case();
break;
case '2':
odd_case();
break;
default:
printf("NO OPTION SELEÇTED/n");
}
return 0;
}
void even_case(void)
{
int DIM1;
int l1;
float sum;
float **alp,**mlp,**nlp,**qlp,**rlp,**t1p,**t2p,**t3p,**t4p,**t5p;
printf("ENTER DIMENSION OF YOUR MATRIX\n");
scanf("%d",&DIM1);
if(DIM1%2!=0)

```

```

printf("YOUR DIMENSION IS NOT EVEN\n");
else{
l1=(DIM1/2);
alp=matrix(1,DIM1,1,DIM1);
mlp=matrix(1,l1,1,l1);
nlp=matrix(1,l1,1,l1);
qlp=matrix(1,l1,1,l1);
rlp=matrix(1,l1,1,l1);
t1p=matrix(1,l1,1,l1);
t2p=matrix(1,l1,1,l1);
t3p=matrix(1,l1,1,l1);
t4p=matrix(1,l1,1,l1);
t5p=matrix(1,l1,1,l1);
printf("PLEASE ENTER YOUR MATRIX\n");
init_matrix(alp,DIM1);
display(alp,DIM1,DIM1);
sum=trace(alp,DIM1);
if(sum!=0)
{
printf("YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
printf("ENTER A MATRIX WHOUSE TRACE IS 0\n");
}
else printf(" YOUR MATRIX IS IN SPECIAL LINEAR ALGEBRA\n");
part_m1(mlp,alp,0,0,l1);
printf("MATRIX M IS\n");
display(mlp,l1,l1);
part_m1(nlp,alp,0,l1,l1);
printf("MATRIX N IS\n");
display(nlp,l1,l1);
part_m1(qlp,alp,l1,0,l1);
printf("MATRIX Q IS\n");
display(qlp,l1,l1);
part_m1(rlp,alp,l1,l1,l1);
printf("MATRIX R IS\n");
display(rlp,l1,l1);
ntrans(t1p,mlp,l1,l1);
printf("NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t1p,l1,l1);
ntrans(t2p,nlp,l1,l1);
printf("NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t2p,l1,l1);

```

```

ntrans(t3p,q1p,l1,l1);
printf("NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t3p,l1,l1);
trans(t4p,n1p,l1,l1);
printf("TRANSPOSE OF MATRIX N IS\n");
display(t4p,l1,l1);
trans(t5p,q1p,l1,l1);
printf("TRANSPOSE OF MATRIX Q IS\n");
display(t5p,l1,l1);
if (compare(q1p,t3p,l1,l1)==1 && compare(n1p,t2p,l1,l1)==1
&& compare(r1p,t1p,l1,l1)==1)
{printf("Q=negative Q transpose\n");
printf("N=negative N transpose\n");
printf("R=negative M transpose SO\n");
printf("IT IS ALSO IN EVEN DIMENSIONAL ORTHOGONAL
ALGEBRA\n");
}
else if (compare(n1p,t4p,l1,l1)==1 && compare(q1p,t5p,l1,l1)==1
&& compare(r1p,t1p,l1,l1)==1)
{printf("N=N transpose \n");
printf("Q=Q transpose \n");
printf("R=negative M transpose SO\n");
printf("IT IS ALSO IN SYMPLECTIC ALGEBRA\n");
}
else printf("BUT IT IS NOT IN SYMPLECTIC ALGEBRA\n");
}
}
void odd_case(void)
{
int DIM2;
int l2;
float sum;
float **a2p,**b1p,**b2p,**c1p,**c2p,**m2p,**n2p,**q2p,**r2p;
float **t6p,**t7p,**t8p,**t9p,**t10p;
printf("ENTER DIMENSION OF YOUR MATRIX\n");
scanf("%d",&DIM2);
if(DIM2%2==0)
printf("YOUR DIMENSION IS NOT ODD\n");
else{
l2=((DIM2-1)/2);
a2p=matrix(1,DIM2,1,DIM2);

```

```

m2p=matrix(1,l2,1,l2);
n2p=matrix(1,l2,1,l2);
q2p=matrix(1,l2,1,l2);
r2p=matrix(1,l2,1,l2);
b1p=matrix(1,1,1,l2);
b2p=matrix(1,1,1,l2);
c1p=matrix(1,l2,1,1);
c2p=matrix(1,l2,1,1);
t6p=matrix(1,l2,1,1);
t7p=matrix(1,l2,1,1);
t8p=matrix(1,l2,1,l2);
t9p=matrix(1,l2,1,l2);
t10p=matrix(1,l2,1,l2);
printf("PLEASE ENTER YOUR MATRIX\n");
printf("FIRST COEFFICIENT MUST BE=0\n");
init_matrix(a2p,DIM2);
display(a2p,DIM2,DIM2);
sum=trace(a2p,DIM2);
if(sum!=0)
{
printf("YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
printf("ENTER A MATRIX WHOSE TRACE 0\n");
}
part1(b1p,a2p,1,0,l2);
printf(" MATRIX B1 IS\n");
display(b1p,1,l2);
part1(b2p,a2p,l2,1,l2);
printf("MATRIX B2 IS\n");
display(b2p,1,l2);
part2(c1p,a2p,1,0,l2);
printf("MATRIX C1 IS\n");
display(c1p,l2,1);
part2(c2p,a2p,l2,1,l2);
printf("MATRIX C2 IS\n");
display(c2p,l2,1);
part_m2(m2p,a2p,1,0,1,0,l2);
printf("MATRIX M IS\n");
display(m2p,l2,l2);
part_m2(n2p,a2p,1,0,l2,1,l2);
printf("MATRIX N IS\n");
display(n2p,l2,l2);

```

```

part_m2(q2p,a2p,l2,1,1,0,l2);
printf("MATRIX Q IS\n");
display(q2p,l2,l2);
part_m2(r2p,a2p,l2,1,l2,1,l2);
printf("MATRIX R IS\n");
display(r2p,l2,l2);
ntrans(t6p,c1p,1,l2);
printf("NEGATIVE TRANSPOSE OF MATRIX C1 IS\n");
display(t6p,1,l2);
ntrans(t7p,c2p,1,l2);
printf("NEGATIVE TRANSPOSE OF MATRIX C2 IS\n");
display(t7p,1,l2);
ntrans(t8p,q2p,l2,l2);
printf("NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t8p,l2,l2);
ntrans(t9p,n2p,l2,l2);
printf("NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t9p,l2,l2);
ntrans(t10p,m2p,l2,l2);
printf("NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t10p,l2,l2);
if (compare(b2p,t6p,1,l2)==1 && compare(b1p,t7p,1,l2)==1 &&/
    compare(q2p,t8p,l2,l2)==1/
    && compare(n2p,t9p,l2,l2)==1 && compare(r2p,t10p,l2,l2)==1)
{
printf("B2=negative C1 transpose \n");
printf("B1=negative C2 transpose \n");
printf("Q=negative Q transpose \n");
printf("N=negative N transpose \n");
printf("R=negative M transpose \n");
printf("IT IS ALSO IN ODD DIMENSIONAL ORTHOGONAL
ALGEBRA\n");
}
else
{
printf("IT IS NOT IN ODD DIMENSIONAL ORTHOGONAL
ALGEBRA\n");
}
}
}
void init_matrix(float **x,int k)

```

```

{ float r;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void ntrans(float **y,float **x,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
y[i][j]=-1*x[j][i];
}
int compare(float **x,float **y,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if (x[i][j]!=y[i][j])
return 0;
return 1;
}
void display(float **x,int k,int t)
{
for(i=1;i<=k;i++)
{for(j=1;j<=t;j++)
{printf("%ft",x[i][j]);
}
printf("\n");
}
}
void trans(float **y,float **x,int m,int n)
{for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{y[i][j]=x[j][i];
}
}
float trace(float **x,int k)
{float sum=0;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{if(i==j) sum+=x[i][j];
}
}

```

```

}
printf("The trace is %f",sum);
return sum;
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void part_m1(float **x,float **y,int t,int k,int l)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[t+i][k+j];
}
void part_m2(float **x,float **y,int t,int k,int z,int u,int l)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[i+t+k][j+z+u];
}
void part1(float **x,float **y,int k,int t,int l)
{i=1;
for(j=1;j<=l;j++)
x[i][j]=y[i][j+k+t];
}
void part2(float **x,float **y,int k,int t,int l)
{j=1;
for(i=1;i<=l;i++)

```

```
x[i][j]=y[i+k+t][j];
}
```

Yine programımızın çıktısını dosyalayalım ve programı çalıştıralım.

Fyeni.cpp

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
void init_matrix(float **x,int k);
void display(float **x,int k,int t);
void trans(float **x,float **y,int m,int n);
void ntrans(float **x,float **y,int m,int n);
void part1(float **x,float **y,int k,int t,int l);
void part2(float **x,float **y,int k,int t,int l);
void part_m1(float **x,float **y,int k,int t,int l);
void part_m2(float **x,float **y,int k,int t,int z,int u,int l);
float trace(float **x,int k);
void even_case(void);
void odd_case(void);
float **matrix(int nrl,int nrh,int ncl,int nch);
int compare(float **x,float **y,int m,int n);
int i,j;
FILE *fp;
main()
{
if((fp=fopen("Yeni","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
printf("CLASSICAL ALGEBRAS made by Serpil KARAGÖZ\n");
printf("This is a program to find that whether a given matrix belongs\n");
printf("to the one of the following algebras or not\n");
printf("An SPECIAL LINEAR ALGEBRA\n");
printf("Bn ODD DIMENSIONAL ORTHAGONAL LIE ALGEBRA\n");
printf("Cn SYMPLECTIC ALGEBRA\n");
printf("Dn EVEN DIMENSIONAL ORTHAGONAL LIE ALGEBRA\n");
printf("1.EVEN DIMENSIONAL MATRIX\n");
printf("2.ODD DIMENSIONAL MATRIX\n");
printf("ENTER YOUR COHICE 1 OR 2\n");
char ch;
ch=getchar();
```

```

switch(ch){
case '1':
even_case();
break;
case '2':
odd_case();
break;
default:
printf("NO OPTION SELECTED\n");
}
return 0;
}
void even_case(void)
{
int DIM1;
int l1;
float sum;
float **a1p,**m1p,**n1p,**q1p,**r1p,**t1p,**t2p,**t3p,**t4p,**t5p;
printf("ENTER DIMENSION OF YOUR MATRIX\n");
scanf("%d",&DIM1);
if(DIM1%2!=0)
printf("YOUR DIMENSION IS NOT EVEN\n");
else{
l1=(DIM1/2);
a1p=matrix(1,DIM1,1,DIM1);
m1p=matrix(1,l1,1,l1);
n1p=matrix(1,l1,1,l1);
q1p=matrix(1,l1,1,l1);
r1p=matrix(1,l1,1,l1);
t1p=matrix(1,l1,1,l1);
t2p=matrix(1,l1,1,l1);
t3p=matrix(1,l1,1,l1);
t4p=matrix(1,l1,1,l1);
t5p=matrix(1,l1,1,l1);
printf("PLEASE ENTER YOUR MATRIX\n");
init_matrix(a1p,DIM1);
display(a1p,DIM1,DIM1);
sum=trace(a1p,DIM1);
if(sum!=0)
{
fprintf(fp,"YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
}
}
}

```

```

printf("ENTER A MATRIX WHOUSE TRACE IS 0\n");
}
else fprintf(fp," YOUR MATRIX IS IN SPECIAL LINEAR
ALGEBRA\n");
part_m1(m1p,alp,0,0,l1);
fprintf(fp,"MATRIX M IS\n");
display(m1p,l1,l1);
part_m1(n1p,alp,0,l1,l1);
fprintf(fp,"MATRIX N IS\n");
display(n1p,l1,l1);
part_m1(q1p,alp,l1,0,l1);
fprintf(fp,"MATRIX Q IS\n");
display(q1p,l1,l1);
part_m1(r1p,alp,l1,l1,l1);
fprintf(fp,"MATRIX R IS\n");
display(r1p,l1,l1);
ntrans(t1p,m1p,l1,l1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t1p,l1,l1);
ntrans(t2p,n1p,l1,l1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t2p,l1,l1);
ntrans(t3p,q1p,l1,l1);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t3p,l1,l1);
trans(t4p,n1p,l1,l1);
fprintf(fp,"TRANSPOSE OF MATRIX N IS\n");
display(t4p,l1,l1);
trans(t5p,q1p,l1,l1);
fprintf(fp,"TRANSPOSE OF MATRIX Q IS\n");
display(t5p,l1,l1);
if (compare(q1p,t3p,l1,l1)==1 && compare(n1p,t2p,l1,l1)==1
&& compare(r1p,t1p,l1,l1)==1)
{ fprintf(fp,"Q=negative Q transpose\n");
fprintf(fp,"N=negative N transpose\n");
fprintf(fp,"R=negative M transpose SO\n");
fprintf(fp,"IT IS ALSO IN EVEN DIMEMSIONAL ORTHOGONAL
ALGEBRA\n");
}
else if (compare(n1p,t4p,l1,l1)==1 && compare(q1p,t5p,l1,l1)==1/
&& compare(r1p,t1p,l1,l1)==1)

```

```

{fprintf(fp,"N=N transpose \n");
fprintf(fp,"Q=Q transpose \n");
fprintf(fp,"R=negative M transpose SO\n");
fprintf(fp,"IT IS ALSO IN SYMPLECTIC ALGEBRA\n");
}
else fprintf(fp,"BUT IT IS NOT IN SYMPLECTIC ALGEBRA\n");
}
}
void odd_case(void)
{
int DIM2;
int l2;
float sum;
float **a2p,**b1p,**b2p,**c1p,**c2p,**m2p,**n2p,**q2p,**r2p;
float **t6p,**t7p,**t8p,**t9p,**t10p;
printf("ENTER DIMENSION OF YOUR MATRIX\n");
scanf("%d",&DIM2);
if(DIM2%2==0)
printf("YOUR DIMENSION IS NOT ODD\n");
else{
l2=((DIM2-1)/2);
a2p=matrix(1,DIM2,1,DIM2);
m2p=matrix(1,l2,1,l2);
n2p=matrix(1,l2,1,l2);
q2p=matrix(1,l2,1,l2);
r2p=matrix(1,l2,1,l2);
b1p=matrix(1,1,1,l2);
b2p=matrix(1,1,1,l2);
c1p=matrix(1,l2,1,1);
c2p=matrix(1,l2,1,1);
t6p=matrix(1,l2,1,1);
t7p=matrix(1,l2,1,1);
t8p=matrix(1,l2,1,l2);
t9p=matrix(1,l2,1,l2);
t10p=matrix(1,l2,1,l2);
printf("PLEASE ENTER YOUR MATRIX\n");
printf("FIRST COEFFICIENT MUST BE=0\n");
init_matrix(a2p,DIM2);
display(a2p,DIM2,DIM2);
sum=trace(a2p,DIM2);
if(sum!=0)

```

```

{
fprintf(fp,"YOUR MATRIX IS NOT IN ANY ALGEBRAS\n");
printf("ENTER A MATRIX WHOUSE TRACE 0\n");
}
part1(b1p,a2p,1,0,l2);
fprintf(fp," MATRIX B1 IS\n");
display(b1p,1,l2);
part1(b2p,a2p,l2,1,l2);
fprintf(fp,"MATRIX B2 IS\n");
display(b2p,1,l2);
part2(c1p,a2p,1,0,l2);
fprintf(fp,"MATRIX C1 IS\n");
display(c1p,l2,1);
part2(c2p,a2p,l2,1,l2);
fprintf(fp,"MATRIX C2 IS\n");
display(c2p,l2,1);
part_m2(m2p,a2p,1,0,1,0,l2);
fprintf(fp,"MATRIX M IS\n");
display(m2p,l2,l2);
part_m2(n2p,a2p,1,0,l2,1,l2);
fprintf(fp,"MATRIX N IS\n");
display(n2p,l2,l2);
part_m2(q2p,a2p,l2,1,1,0,l2);
fprintf(fp,"MATRIX Q IS\n");
display(q2p,l2,l2);
part_m2(r2p,a2p,l2,1,l2,1,l2);
fprintf(fp,"MATRIX R IS\n");
display(r2p,l2,l2);
ntrans(t6p,c1p,1,l2);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX C1 IS\n");
display(t6p,1,l2);
ntrans(t7p,c2p,1,l2);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX C2 IS\n");
display(t7p,1,l2);
ntrans(t8p,q2p,l2,l2);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX Q IS\n");
display(t8p,l2,l2);
ntrans(t9p,n2p,l2,l2);
fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX N IS\n");
display(t9p,l2,l2);
ntrans(t10p,m2p,l2,l2);

```

```

fprintf(fp,"NEGATIVE TRANSPOSE OF MATRIX M IS\n");
display(t10p,l2,l2);
if (compare(b2p,t6p,1,l2)==1 && compare(b1p,t7p,1,l2)==1 &&/
    compare(q2p,t8p,l2,l2)==1/
    && compare(n2p,t9p,l2,l2)==1 && compare(r2p,t10p,l2,l2)==1)
{
fprintf(fp,"B2=negative C1 transpose \n");
fprintf(fp,"B1=negative C2 transpose \n");
fprintf(fp,"Q=negative Q transpose \n");
fprintf(fp,"N=negative N transpose \n");
fprintf(fp,"R=negative M transpose \n");
fprintf(fp,"IT IS ALSO IN ODD DIMENSIONAL ORTHOGONAL
ALGEBRA\n");
}
else
{
fprintf(fp,"IT IS NOT IN ODD DIMENSIONAL ORTHOGONAL
ALGEBRA\n");
}
}
}
}
void init_matrix(float **x,int k)
{ float r;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void ntrans(float **y,float **x,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
y[i][j]=-1*x[j][i];
}
int compare(float **x,float **y,int m,int n)
{
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
if (x[i][j]!=y[i][j])
return 0;
}

```

```

return 1;
}
void display(float **x,int k,int t)
{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"\r");
{for(j=1;j<=t;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
void trans(float **y,float **x,int m,int n)
{for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{y[i][j]=x[j][i];
}
}
float trace(float **x,int k)
{float sum=0;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{if(i==j) sum+=x[i][j];
}
printf("The trace is %f",sum);
return sum;
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
}
}

```

```

m[i]--=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void part_m1(float **x,float **y,int t,int k,int l)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[t+i][k+j];
}
void part_m2(float **x,float **y,int t,int k,int z,int u,int l)
{
for(i=1;i<=l;i++)
for(j=1;j<=l;j++)
x[i][j]=y[i+t+k][j+z+u];
}
void part1(float **x,float **y,int k,int t,int l)
{i=1;
for(j=1;j<=l;j++)
x[i][j]=y[i][j+k+t];
}
void part2(float **x,float **y,int k,int t,int l)
{j=1;
for(i=1;i<=l;i++)
x[i][j]=y[i+k+t][j];}

```

Örnek1:

Boyutu 5x5 olan bir matrisi ele alalım.

```

0.000000 -2.000000 -1.000000 -3.000000 -4.000000
3.000000 -1.000000 0.000000 0.000000 0.000000
4.000000 -2.000000 1.000000 0.000000 0.000000
2.000000 0.000000 0.000000 1.000000 2.000000
1.000000 0.000000 0.000000 0.000000 -1.000000

```

MATRIX B1 IS

-2.000000 -1.000000

MATRIX B2 IS

-3.000000 -4.000000

MATRIX C1 IS

3.000000

4.000000

MATRIX C2 IS

2.000000

1.000000

MATRIX M IS

-1.000000 0.000000

-2.000000 1.000000

MATRIX N IS

0.000000 0.000000

0.000000 0.000000

MATRIX Q IS

0.000000 0.000000

0.000000 0.000000

MATRIX R IS

1.000000 2.000000

0.000000 -1.000000

NEGATIVE TRANSPOSE OF MATRIX C1 IS

-3.000000 -4.000000

NEGATIVE TRANSPOSE OF MATRIX C2 IS

-2.000000 -1.000000

NEGATIVE TRANSPOSE OF MATRIX Q IS

-0.000000 -0.000000

-0.000000 -0.000000

NEGATIVE TRANSPOSE OF MATRIX N IS

-0.000000 -0.000000

-0.000000 -0.000000

NEGATIVE TRANSPOSE OF MATRIX M IS

1.000000 2.000000

-0.000000 -1.000000

B2=negative C1 transpose
 B1=negative C2 transpose
 Q=negative Q transpose
 N=negative N transpose
 R=negative M transpose

IT IS ALSO IN ODD DIMENSIONAL ORTHOGONAL ALGEBRA

Örnek 2:

6x6 boyutlu bir matrisi programa girelim.

```

1.000000 3.000000 2.000000 0.000000 0.000000 0.000000
-1.000000 4.000000 5.000000 0.000000 0.000000 0.000000
2.000000 3.000000 4.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 -1.000000 1.000000 -2.000000
0.000000 0.000000 0.000000 -3.000000 -4.000000 -3.000000
0.000000 0.000000 0.000000 -2.000000 -5.000000 -4.000000
  
```

YOUR MATRIX IS IN SPECIAL LINEAR ALGEBRA

MATRIX M IS

```

1.000000 3.000000 2.000000
-1.000000 4.000000 5.000000
2.000000 3.000000 4.000000
  
```

MATRIX N IS

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

MATRIX Q IS

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

MATRIX R IS

-1.000000 1.000000 -2.000000

-3.000000 -4.000000 -3.000000

-2.000000 -5.000000 -4.000000

NEGATIVE TRANSPOSE OF MATRIX M IS

-1.000000 1.000000 -2.000000

-3.000000 -4.000000 -3.000000

-2.000000 -5.000000 -4.000000

NEGATIVE TRANSPOSE OF MATRIX N IS

-0.000000 -0.000000 -0.000000

-0.000000 -0.000000 -0.000000

-0.000000 -0.000000 -0.000000

NEGATIVE TRANSPOSE OF MATRIX Q IS

-0.000000 -0.000000 -0.000000

-0.000000 -0.000000 -0.000000

-0.000000 -0.000000 -0.000000

TRANSPOSE OF MATRIX N IS

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

TRANSPOSE OF MATRIX Q IS

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

Q=negative Q transpose

N=negative N transpose

R=negative M transpose SO

IT IS ALSO IN EVEN DIMENSIONAL ORTHOGONAL ALGEBRA

Yapı Sabitleri

L sonlu boyutlu bir Lie cebiri olsun ve bazı da $\{e_1, e_2, e_3, \dots, e_n\}$ olsun.

$$\left[\sum_{i=1}^n a_i e_i, \sum_{j=1}^n b_j e_j \right] = \sum_{i=1}^n \sum_{j=1}^n a_i b_j [e_i, e_j]$$

$$[e_i, e_j] = \sum_{k=1}^n c_{ij}^k e_k$$

$\{c_{ij}^k : 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n\}$ kümesinin elemanlarına yapı sabitleri denir.

Teorem: Yapı sabitleri kümesinin elemanları aşağıdaki şartları sağlar.

- 1) $c_{ii}^k = 0 \quad \forall 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n$
- 2) $c_{ij}^k = -c_{ji}^k \quad \forall 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n$
- 3) $\sum (c_{ij}^k c_{rk}^l + c_{jr}^k c_{ik}^l + c_{ri}^k c_{jk}^l) = 0$

Verilen sabitlerin yapı sabitleri olup olmadığını bu teoremden yararlanarak bulabiliriz. Bu üç şartın verilen sabitlerce sağlandığını test edersek elimizdeki sabitlerin yapı sabitleri olup olmadığını ve sonuç olarak da bir Lie cebirini oluşturup oluşturmadığını anlayabiliriz. Aşağıdaki program verilen sabitlerin yapı sabitleri olup olmadığını test ediyor.

```
Sabit.cpp #include<stdio.h>
#include<math.h>
#include<stdlib.h>
int i,j,k,l,m,N;
float c[10][10][10],top;
main()
{
printf("Enter the dimension of the lie algebra\n");
scanf("%d",&N);
for(i=1;i<=N;i++)
for(j=1;j<=N;j++)
for(k=1;k<=N;k++)
{
printf("Enter your constants c[%d][%d][%d]\n",i,j,k);
scanf("%f",&c[i][j][k]);
}
}
```

```

for(i=1;i<=N;i++)
for(k=1;k<=N;k++)
{
if(c[i][i][k]!=0.000000)
{printf("It is not the set of structure constants\n");
return 0;
}
}
for(i=1;i<=N;i++)
for(j=1;j<=N;j++)
for(k=1;k<=N;k++)
{
if(c[i][j][k]+c[j][i][k]!=0.000000)
{printf("It is not the set of structure constants\n");
return 0;
}
}
for(l=1;l<=N;l++)
for(m=1;m<=N;m++)
for(i=1;i<=N;i++)
for(j=1;j<=N;j++)
{ top=0.000000;
for(k=1;k<=N;k++)
{ top+=c[i][j][k]*c[k][l][m]+c[j][l][k]*c[k][i][m]+c[l][i][k]*c[k][j][m];
}
if( top!=0.000000)
{
printf("It is not the set of structure constants\n");
return 0;
}
}
printf("It is the set of structure constants\n");
return 0;
}

```

Programımızın çıktısını dosyalayalım ve programı tekrar yazalım.

Fsabit.cpp

```

#include<stdio.h>
#include<math.h>

```

```

#include<stdlib.h>
int i,j,k,l,m,N;
float c[10][10][10],top;
FILE *fp;
main()
{
if((fp=fopen("Sabit","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
printf("Enter the dimension of the lie algebra\n");
scanf("%d",&N);
for(i=1;i<=N;i++)
for(j=1;j<=N;j++)
for(k=1;k<=N;k++)
{
printf("Enter your constants c[%d][%d][%d]\n",i,j,k);
scanf("%f",&c[i][j][k]);
}
for(i=1;i<=N;i++)
for(k=1;k<=N;k++)
{
if(c[i][i][k]!=0.000000)
{
fprintf(fp,"It is not the set of structure constants\n");
return 0;
}
}
for(i=1;i<=N;i++)
for(j=1;j<=N;j++)
for(k=1;k<=N;k++)
{
if(c[i][j][k]+c[j][i][k]!=0.000000)
{fprintf(fp,"It is not the set of structure constants\n");
return 0;
}
}
for(l=1;l<=N;l++)
for(m=1;m<=N;m++)
for(i=1;i<=N;i++)
for(j=1;j<=N;j++)

```

```

{ top=0.000000;
for(k=1;k<=N;k++)
{ top+=c[i][j][k]*c[k][l][m]+c[j][l][k]*c[k][i][m]+c[l][i][k]*c[k][j][m];
}
if( top!=0.000000)
{
fprintf(fp,"It is not the set of structure constants\n");
return 0;
}
}
fprintf(fp,"It is the set of structure constants\n");
return 0;
}

```

Örnek 1:

Aşağıdaki sabitlerin üç boyutlu bir Lie cebir oluşturup oluşturmadığını test edersek

$$S = \{0,0,0,0,2,0,0,0,-2,0-2,0,0,0,0,1,0,0,0,0,2,-1,0,0,0,0,0\}$$

Şu çıktıyı alırsınız.

It is the set of structure constants

Örnek 2:

$$S = \{1,2,4,0,8,9,0,7,6,1,1,-1,1,4,2,2,0,0,0,0,3,3,3,4,5,6,7\}$$

Programa bu sabitleri girersek şu çıktıyı alıyoruz

It is not the set of structure constant

Semisimple Lie Cebirinin Türevi

Teorem :Semisimple Lie cebirinin her bir türevine içtürev(INNER) dır.

Lemma:L bir Lie cebiri olsun.(Der L = Türevlerin kümesi) Der L, End(L) nin bir Lie cebiridir.

Tanım: Lineer operatör $\delta:L \rightarrow L$ bir türevdir.Eğer

$$\delta([x,y]) = [\delta(x),y] + [x,\delta(y)] \text{ ise}$$

Bu tanımdan yola çıkarak verilen bir lineer operatörün türev (derivasyon) olup olmadığını test eden bir program yapabiliriz.

Türev.cpp

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y,int w);

```

```

void add(float **z,float **x,float **y);
void subt(float **z,float **x,float **y);
void donusum1(float **X,float **x);
void donusum2(float **y,float **Y);
void display1(float **y);
void mult1(float **z,float **x,float **y);
int dim,ldim,i,j;
main()
{
float **lp,**a,**A,**b,**B,**l1,**l2,**s1,**SOL,**m1,**m2,**s2;
float **m3,**m4,**s3,**sag,**S1,**A1,**a1,**B1,**b1,**SAG;
printf("This is a test of whether a given linear operator is a derivation or
not\n");
printf("Enter dimension of the lie algebra\n");
scanf("%d",&ldim);
printf("Enter the dimension of the elements of the lie algebra\n");
scanf("%d",&dim);
printf("Enter the operator\n");
a1=matrix(1,dim,1,dim);
b1=matrix(1,dim,1,dim);
A1=matrix(1,ldim,1,1);
B1=matrix(1,ldim,1,1);
S1=matrix(1,ldim,1,1);
lp=matrix(1,ldim,1,ldim);
a=matrix(1,dim,1,dim);
b=matrix(1,dim,1,dim);
l1=matrix(1,dim,1,dim);
l2=matrix(1,dim,1,dim);
s1=matrix(1,dim,1,dim);
SOL=matrix(1,dim,1,1);
m1=matrix(1,dim,1,dim);
m2=matrix(1,dim,1,dim);
s2=matrix(1,dim,1,dim);
m3=matrix(1,dim,1,dim);
m4=matrix(1,dim,1,dim);
s3=matrix(1,dim,1,dim);
sag=matrix(1,dim,1,dim);
A=matrix(1,ldim,1,1);
B=matrix(1,ldim,1,1);
SAG=matrix(1,ldim,1,1);
init_matrix(lp,ldim);

```

```

display(lp,ldim);
printf("Enter two elements of the Lie Algebra\n");
init_matrix(a,dim);
display(a,dim);
donusum1(A,a);
init_matrix(b,dim);
display(b,dim);
donusum1(B,b);
mult(l1,a,b,dim);
mult(l2,b,a,dim);
subt(s1,l1,l2);
donusum1(S1,s1);
mult1(SOL,lp,S1);
display1(SOL);
mult1(A1,lp,A);
donusum2(a1,A1);
mult(m1,a1,b,dim);
mult(m2,b,a1,dim);
subt(s2,m1,m2);
mult1(B1,lp,B);
donusum2(b1,B1);
mult(m3,a,b1,dim);
mult(m4,b1,a,dim);
subt(s3,m3,m4);
add(sag,s2,s3);
donusum1(SAG,sag);
for(i=1;i<=ldim;i++)
{
if(SOL[i][1]!=SAG[i][1])
{
printf("The linear operator is not a derivation\n");
return 0;
}
}
printf("The linear operator is a derivation\n");
return 0;
}
void init_matrix(float **x,int k)
{ float r;
int i,j;
for(i=1;i<=k;i++)

```

```

for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{for(j=1;j<=k;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void mult(float **z,float **x,float **y,int w)
{
int i,j,k;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
z[i][j]=0.000000;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
for(k=1;k<=w;k++)

```

```

z[i][j]+=x[i][k]*y[k][j];
}
void mult1(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
z[i][j]=0.000000;
for(i=1;i<=ldim;i++)
for(k=1;k<=ldim;k++)
z[i][1]+=x[i][k]*y[k][1];
}
void add(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0.000000;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]+y[i][j];
}
void subt(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0.000000;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]-y[i][j];
}
void donusum1(float **X,float **x)
{int r,s;
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{X[(r-1)*dim+s][1]=x[r][s];}
for(s=1;s<=dim;s++)
{X[(dim-1)*dim+s][1]=x[r][s];}
printf("\n");
display1(X);
}

```

```

}
void donusum2(float **y,float **Y)
{
int r,s,k;
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{y[r][s]=Y[(r-1)*dim+s][1];}
for(s=1;s<=dim;s++)
{y[r][s]=Y[(dim-1)*dim+s][1];}
printf("\n");
y[dim][dim]=0.000000;
for(k=1;k<=dim-1;k++)
{y[dim][dim]+=y[k][k];}
}
void display1(float **y)
{
int i;
for(i=1;i<=ldim;i++)
printf("%f\n",y[i][1]);
}

```

Programın çıktısını dosyalayalım. Yeni programımız aşağıdaki gibi olacaktır.

Ftürev.cpp

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y,int w);
void add(float **z,float **x,float **y);
void subt(float **z,float **x,float **y);
void donusum1(float **X,float **x);
void donusum2(float **y,float **Y);
void display1(float **y);
void mult1(float **z,float **x,float **y);
int dim,ldim,i,j;
FILE *fp;
main()
{
if((fp=fopen("Türev","wb"))==NULL){

```

```

printf("can not open file\n");
exit(1);
}
float **lp,**a,**A,**b,**B,**l1,**l2,**s1,**SOL,**m1,**m2,**s2;
float **m3,**m4,**s3,**sag,**S1,**A1,**a1,**B1,**b1,**SAG;
printf("This is a test of whether a given linear operator is a derivation or
not\n");
printf("Enter dimension of the lie algebra\n");
scanf("%d",&ldim);
printf("Enter the dimension of the elements of the lie algebra\n");
scanf("%d",&dim);
printf("Enter the operator\n");
a1=matrix(1,dim,1,dim);
b1=matrix(1,dim,1,dim);
A1=matrix(1,ldim,1,1);
B1=matrix(1,ldim,1,1);
S1=matrix(1,ldim,1,1);
lp=matrix(1,ldim,1,ldim);
a=matrix(1,dim,1,dim);
b=matrix(1,dim,1,dim);
l1=matrix(1,dim,1,dim);
l2=matrix(1,dim,1,dim);
s1=matrix(1,dim,1,dim);
SOL=matrix(1,dim,1,1);
m1=matrix(1,dim,1,dim);
m2=matrix(1,dim,1,dim);
s2=matrix(1,dim,1,dim);
m3=matrix(1,dim,1,dim);
m4=matrix(1,dim,1,dim);
s3=matrix(1,dim,1,dim);
sag=matrix(1,dim,1,dim);
A=matrix(1,ldim,1,1);
B=matrix(1,ldim,1,1);
SAG=matrix(1,ldim,1,1);
init_matrix(lp,ldim);
display(lp,ldim);
printf("Enter two elements of the Lie Algebra\n");
init_matrix(a,dim);
display(a,dim);
donusum1(A,a);
init_matrix(b,dim);

```

```

display(b,dim);
donusum1(B,b);
mult(l1,a,b,dim);
mult(l2,b,a,dim);
subt(s1,l1,l2);
donusum1(S1,s1);
mult1(SOL,lp,S1);
display1(SOL);
mult1(A1,lp,A);
donusum2(a1,A1);
mult(m1,a1,b,dim);
mult(m2,b,a1,dim);
subt(s2,m1,m2);
mult1(B1,lp,B);
donusum2(b1,B1);
mult(m3,a,b1,dim);
mult(m4,b1,a,dim);
subt(s3,m3,m4);
add(sag,s2,s3);
donusum1(SAG,sag);
for(i=1;i<=ldim;i++)
{
if(SOL[i][1]!=SAG[i][1])
{
fprintf(fp,"The linear operator is not a derivation\n");
return 0;
}
}
fprintf(fp,"The linear operator is a derivation\n");
return 0;
}
void init_matrix(float **x,int k)
{ float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)

```

```

{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"\r");
{for(j=1;j<=k;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");

m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void mult(float **z,float **x,float **y,int w)
{
int i,j,k;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
z[i][j]=0.000000;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
for(k=1;k<=w;k++)
z[i][j]+=x[i][k]*y[k][j];
}

```

```

void mult1(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
z[i][j]=0.000000;
for(i=1;i<=ldim;i++)
for(k=1;k<=ldim;k++)
z[i][1]+=x[i][k]*y[k][1];
}
void add(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0.000000;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]+y[i][j];
}
void sub(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0.000000;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]-y[i][j];
}
void donusum1(float **X,float **x)
{int r,s;
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{X[(r-1)*dim+s][1]=x[r][s];}
for(s=1;s<=dim;s++)
{X[(dim-1)*dim+s][1]=x[r][s];}
printf("\n");
display1(X);
}
void donusum2(float **y,float **Y)

```

```

{
int r,s,k;
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{y[r][s]=Y[(r-1)*dim+s][1];}
for(s=1;s<=dim;s++)
{y[r][s]=Y[(dim-1)*dim+s][1];}
printf("\n");
y[dim][dim]=0.000000;
for(k=1;k<=dim-1;k++)
{y[dim][dim]+=y[k][k];}
}
void display1(float **y)
{
int i;
for(i=1;i<=ldim;i++)
printf("%f\n",y[i][1]);
}

```

Örnek1:

0.000000 -1.000000 0.000000

0.000000 -4.000000 0.000000

2.000000 0.000000 4.000000

-2.000000 0.000000

1.000000 2.000000

0.000000 0.000000

1.000000 0.000000

The linear operator is not a derivation

L nin Elemanlarının Adjointi

F Cebiri derken basit olarak F üzerinde bir vektör uzayı ve bilinear bir operasyonu kastediyoruz. Derivasyonu daha önce

tanımlamıştık. Eğer $x \in L$, $y \rightarrow [x, y]$ L nin bir endomorfismasıysa, L Lie cebiri bir F cebiri olduğundan belirli derivasyonlar doğal olarak ortaya çıkar. Biz bu derivasyonu adx ile göstereceğiz. Aslında $adx \in \text{Der}L$.

$$L \rightarrow \text{Der}L$$

$x \rightarrow adx$ bağıntısı L nin adjoint temsili diye adlandırılır.

$ad: L \rightarrow \text{gl}(L)$ bir monomorfizmadır. Yani her basit Lie cebiri, daha önce üzerinde durduğumuz, bir lineer Lie cebirine izomorfiktir.

L , Lie cebirinin her bir elemanının adjointini hesaplamak verilen Lie cebirinin bazına göre değişik hesaplama gerektirmektedir.

Özel lineer Lie cebirini ele alalım.

$$A \in \text{sl}(n, F)$$

$$\beta = \{b[1], b[2], \dots, b[\text{ldim}]\}$$

$$[b[i], b[j]] = \sum_{k=1}^{\text{ldim}} B_{ij}^k b[k]$$

$$A = \sum_{i=1}^{\text{ldim}} c_i b[i]$$

$$[A, b[j]] = \left[\sum_{i=1}^{\text{ldim}} c_i b[i], b[j] \right]$$

$$= \sum_{i=1}^{\text{ldim}} c_i [b[i], b[j]]$$

$$= \sum_{k=1}^{\text{ldim}} \left(\sum_{i=1}^{\text{ldim}} c_i B_{ij}^k \right) b[k]$$

$$T_{ij} = \sum_{i=1}^{\text{ldim}} c_i B_{ij}^k$$

1.yol: B_{ij}^k ler elle hesaplanıp bilgisayara girilir.

2.yol: Gaussjordan metoduyla B_{ij}^k ler hesaplanır.

Gaussjordan metoduyla deklemler sistemi çözen bir programa ihtiyacımız olacaktır.

Gaussjo.cpp

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SWAP(a,b){float temp=(a);(a)=(b);(b)=temp;}
#define dim 3
#define num 1
```

```

void gaussjo(float **a,int n,float **b,int m);
void free_ivector(int *v,int n1,int nh);
int *ivector(int n1,int nh);
void nerror(char error_text[]);
void displayl(float **y);
void initl(float **b);
void init(float **A);
void display(float **A);
float **matrix(int nrl,int nrh,int ncl,int nch);
main()
{
float **p,**q;
p=matrix(1,dim,1,dim);
q=matrix(1,dim,1,num);
init(p);
printf("\n\n");
initl(q);
display(p);
printf("\n\n");
displayl(q);
printf("\n\n");
gaussjo(p,dim,q,num);
display(p);
printf("\n\n");
displayl(q);
return 0;
}
void init(float **A)
{
float r;
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);A[i][j]=r;
printf("\n");}
}
void initl(float **B)
{
int i,j;
float r;

```

```

for(j=1;j<=num;j++)
{
for(i=1;i<=dim;i++)
{ printf("Enter the coefficient B[%d][%d]\n",i,j);
scanf("%f",&r);B[i][j]=r;}
printf("\n");
}
}
void display(float **A)
{
int i,j;
for(i=1;i<=dim;i++)
{for(j=1;j<=dim;j++)
{printf("%ft",A[i][j]);
}
}
printf("\n");
}
}
void display1(float **y)
{
int i,j;
for(i=1;i<=dim;i++)
{for(j=1;j<=num;j++)
{printf("%ft",y[i][j]);
}
}
printf("\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
}

```

```

}
/*return pointer to array of pointers to rows*/
return m;
}
void free_ivector(int *v,int nl,int nh)
{
free((char*)(v+nl));
}
void gaussjo(float **a,int n,float **b,int m)
{int *indxc,*indxr,*ipiv;
int i,icol,irow,j,k,l,ll;
float big,dum,pivinv;
indxc=ivector(1,n);
indxr=ivector(1,n);
ipiv=ivector(1,n);
for(j=1;j<=n;j++) ipiv[j]=0;
for(i=1;i<=n;i++)
{
big=0.0;
for(j=1;j<=n;j++)
if(ipiv[j]!=1)
for(k=1;k<=n;k++)
{
if(ipiv[k]==0)
{
if(fabs(a[j][k])>=big)
{
big=fabs(a[j][k]);
irow=j;
icol=k;
}
}
}
else if(ipiv[k]>1) nrerror("GAUSSJ:Singular matrix-1");
}
++(ipiv[icol]);
/*indxc[i]=the column of ith pivot element */
/*indxr[i]=the row in which that pivot element was originally located*/
if(irow!=icol)
{
for(l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
for(l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
}
}
}

```

```

}
/*divide the pivot row by the pivot element located at irow and icol*/
indxr[i]=irow;
indxc[i]=icol;
if(a[icol][icol]==0.0) nrerror("GAUSSJ:Singular matrix-2");
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
for(l=1;l<=n;l++)
a[icol][l]*=pivinv;
for(l=1;l<=m;l++) b[icol][l]*=pivinv;
for(ll=1;ll<=n;ll++)
if(ll!=icol)
{
dum=a[ll][icol];
a[ll][icol]=0.0;
for(l=1;l<=n;l++) a[ll][l]-=a[icol][l]*dum;
for(l=1;l<=m;l++)
b[ll][l]-=b[icol][l]*dum;
}
}
for(l=n;l>=1;l--)
{
if(indxr[l]!=indxc[l])
for(k=1;k<=n;k++)
SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
free_ivector(ipiv,1,n);
free_ivector(indxr,1,n);
free_ivector(indxc,1,n);
}
void nrerror(char error_text[])
{fprintf(stderr,"Numerical Recipes run-time error....\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"....now exiting to system...\n");
exit(0);
}
int *ivector(int nl,int nh)
/*allocation an int vector with range [nl.....nh]*/
{int *v;
v=(int*)malloc((unsigned)(nh-nl+1)*sizeof(int));
if(!v)nrerror("allocation failure in vector()");
}

```

```
return v-nl;
}
```

Bu programı kullanabileceğimiz bir denklem sistemi kuralım.

$$\sum_{i=1}^{ldim} b[i][k][l]c_i = A[k][l]$$

$$[A, b[j]][k][l] = \sum_{i=1}^{ldim} T_{ij} b[i][k][l]$$

1.denklem k=1 l=1

2.denklem k=1 l=2

:

:

:

n.denklem k=1 l=n

n+1 .denklem k=2 l=1

:

:

:

n+n.denklem k=2 l=n

:

:

:

Bilinmeyen sayısı n^2-1

Denklem sayısı n^2-1

Şimdi bu denklem sistemini çözerek verilen bir matrisin adjointini hesaplayan programı yazalım.

Ad.cpp

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SWAP(a,b){float temp=(a);(a)=(b);(b)=temp;}
void gaussjo(float **a,int n,float **b,int m);
void free_ivector(int *v,int n1,int nh);
int *ivector(int n1,int nh);
void nrerror(char error_text[]);
void displayl(float **y);
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
```

```

void mult(float **z,float **x,float **y);
void subtr(float **z,float **x,float **y);
int dim,ldim;
main()
{
int i,j,r,s;
float
**b[100]**a,**sol,**sag,**son[100]**ad,**l[100][100]**m1[100][100],
**p,**q[100];
printf("Enter dimension of the lie algebra\n");
scanf("%d",&ldim);
printf("Enter dimension of base elements as matrices\n");
scanf("%d",&dim);
ldim=dim*dim-1;
a=matrix(1,dim,1,dim);
sol=matrix(1,dim,1,dim);
sag=matrix(1,dim,1,dim);
ad=matrix(1,ldim,1,ldim);
p=matrix(1,ldim,1,ldim);
for(j=1;j<=ldim;j++)
{
q[j]=matrix(1,ldim,1,1);
}
for(i=1;i<=ldim;i++)
{
b[i]=matrix(1,dim,1,dim);
son[i]=matrix(1,dim,1,dim);
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
l[i][j]=matrix(1,dim,1,dim);
m1[i][j]=matrix(1,dim,1,dim);
}
}
for(i=1;i<=ldim;i++)
{
printf("Enter the base element b[%d]\n",i);
init_matrix(b[i],dim);
display(b[i],dim);
}
}

```

```

}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
mult(m1[i][j],b[i],b[j]);
}
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{printf("Lie product of l[%d][%d]\n",i,j);
{
subt(l[i][j],m1[i][j],m1[j][i]);
display(l[i][j],dim);
}
}
printf("\n\n");
}
printf("Finding the adjoint of a given element of the Lie algebra\n ");
printf("Enter the element of the Lie Algebra as a matrix\n");
init_matrix(a,dim);
display(a,dim);
for(j=1;j<=ldim;j++)
{
mult(sol,a,b[j]);
mult(sag,b[j],a);
subt(son[j],sol,sag);
display(son[j],dim);
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{q[j][(r-1)*dim+s][1]=son[j][r][s];}
for(s=1;s<=dim-1;s++)
{q[j][(dim-1)*dim+s][1]=son[j][r][s];}
printf("\n");
display1(q[j]);
}
printf("Enter the coordinates of base elements except (n,n)th
coordinate\n");
init_matrix(p,ldim);
for(j=1;j<=ldim;j++)
{
gaussjo(p,ldim,q[j],1);

```

```

printf("The inverse of the coefficient matrix\n");
display(p,ldim);
printf("\n\n");
display1(q[j]);
for(i=1;i<=ldim;i++)
{
ad[i][j]=q[j][i][1];
}
}
printf("The adjoint of the given matrix A\n");
display(ad,ldim);
return 0;
}
void init_matrix(float **x,int k)
{
float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{for(j=1;j<=k;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;

```

```

/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void free_ivector(int *v,int nl,int nh)
{
free((char*)(v+nl));
}
void gaussjo(float **a,int n,float **b,int m)
{int *indxc,*indxr,*ipiv;
int i,icol,irow,j,k,l,ll;
float big,dum,pivinv;
indxc=ivector(1,n);
indxr=ivector(1,n);
ipiv=ivector(1,n);
for(j=1;j<=n;j++) ipiv[j]=0;
for(i=1;i<=n;i++)
{
big=0.0;
for(j=1;j<=n;j++)
if(ipiv[j]!=1)
for(k=1;k<=n;k++)
{
if(ipiv[k]==0)
{
if(fabs(a[j][k])>=big)
{
big=fabs(a[j][k]);
irow=j;
icol=k;
}
}
}
else if(ipiv[k]>1) nerror("GAUSSJ:Singular matrix-1");
}
++(ipiv[icol]);
/*indxc[i]=the column of ith pivot element */

```

```

/*indxr[i]=the row in which that pivot element was originally located*/
if(irow!=icol)
{
for(l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
for(l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
}
/*divide the pivot row by the pivot element located at irow and icol*/
indxr[i]=irow;
indxc[i]=icol;
if(a[icol][icol]==0.0) nrerror("GAUSSJ:Singular matrix-2");
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
for(l=1;l<=n;l++)
a[icol][l]*=pivinv;
for(l=1;l<=m;l++) b[icol][l]*=pivinv;
for(ll=1;ll<=n;ll++)
if(ll!=icol)
{
dum=a[ll][icol];
a[ll][icol]=0.0;
for(l=1;l<=n;l++) a[ll][l]-=a[icol][l]*dum;
for(l=1;l<=m;l++)
b[ll][l]-=b[icol][l]*dum;
}
}
for(l=n;l>=1;l--)
{
if(indxr[l]!=indxc[l])
for(k=1;k<=n;k++)
SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
free_ivector(ipiv,1,n);
free_ivector(indxr,1,n);
free_ivector(indxc,1,n);
}
void nrerror(char error_text[])
{fprintf(stderr,"Numerical Recipes run-time error...\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"...now exiting to system...\n");
exit(0);
}

```

```

int *ivector(int nl,int nh)
/*allocation an int vector with range [nl.....nh]*/
{int *v;
v=(int*)malloc((unsigned)(nh-nl+1)*sizeof(int));
if(!v)nrerror("allocation failure in vector()");
return v-nl;
}
void mult(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
for(k=1;k<=dim;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void subt(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]-y[i][j];
}
void display1(float **y)
{
int i,j;
for(i=1;i<=ldim;i++)
{
for(j=1;j<=1;j++)
printf("%f\t",y[i][j]);
printf("\n");
}
}
}

```

Programın çıktısını dosyalım ve programı yeniden yazalım.

Fad.cpp

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SWAP(a,b){float temp=(a);(a)=(b);(b)=temp;}
void gaussjo(float **a,int n,float **b,int m);
void free_ivector(int *v,int n1,int nh);
int *ivector(int n1,int nh);
void perror(char error_text[]);
void display1(float **y);
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y);
void subt(float **z,float **x,float **y);
int dim,ldim;
FILE *fp;
main()
{
int i,j,r,s;
float**b[100]**a,**sol,**sag,**son[100]**ad,**l[100][100]**m1[100][
100]**p,**q[100];
if((fp=fopen("ad","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
printf("Enter dimension of the lie algebra\n");
scanf("%d",&ldim);
printf("Enter dimension of base elements as matrices\n");
scanf("%d",&dim);
ldim=dim*dim-1;
a=matrix(1,dim,1,dim);
sol=matrix(1,dim,1,dim);
sag=matrix(1,dim,1,dim);
ad=matrix(1,ldim,1,ldim);
p=matrix(1,ldim,1,ldim);
for(j=1;j<=ldim;j++)
{
q[j]=matrix(1,ldim,1,1);
}
for(i=1;i<=ldim;i++)

```

```

{
b[i]=matrix(1,dim,1,dim);
son[i]=matrix(1,dim,1,dim);
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
l[i][j]=matrix(1,dim,1,dim);
m1[i][j]=matrix(1,dim,1,dim);
}
}
for(i=1;i<=ldim;i++)
{
printf("Enter the base element b[%d]\n",i);
init_matrix(b[i],dim);
display(b[i],dim);
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
mult(m1[i][j],b[i],b[j]);
}
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{fprintf(fp,"Lie product of l[%d][%d]\n",i,j);
{
subt(l[i][j],m1[i][j],m1[j][i]);
display(l[i][j],dim);
}
printf("\n\n");
}
fprintf(fp,"finding the adjoint of a given element of the Lie algebra\n ");
fprintf(fp,"Enter the element of the Lie Algebra as a matrix\n");
init_matrix(a,dim);
display(a,dim);
for(j=1;j<=ldim;j++)
{
mult(sol,a,b[j]);
}
}

```

```

mult(sag,b[j],a);
subt(son[j] ,sol,sag);
display(son[j],dim);
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{q[j][(r-1)*dim+s][1]=son[j][r][s];}
  for(s=1;s<=dim-1;s++)
  {q[j][(dim-1)*dim+s][1]=son[j][r][s];}
  printf("\n");
  display1(q[j]);
}
  printf("Enter the coordinates of base elements except (n,n)th
coordinate\n");
  init_matrix(p,l dim);
for(j=1;j<=l dim;j++)
{
gaussjo(p,l dim,q[j],1);
printf("The inverse of the coefficient matrix\n");
display(p,l dim);
printf("\n\n");
display1(q[j]);
for(i=1;i<=l dim;i++)
{
ad[i][j]=q[j][i][1];
}
}
fprintf(fp,"The adjoint of the given matrix A\n");
display(ad,l dim);
return 0;
}
void init_matrix(float **x,int k)
{
float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)

```

```

{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"r");
{for(j=1;j<=k;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void free_ivector(int *v,int nl,int nh)
{
free((char*)(v+nl));
}
void gaussjo(float **a,int n,float **b,int m)
{int *indxc,*indxr,*ipiv;
int i,icol,irow,j,k,l,ll;
float big,dum,pivinv;
indxc=ivector(1,n);
indxr=ivector(1,n);
ipiv=ivector(1,n);
for(j=1;j<=n;j++) ipiv[j]=0;
for(i=1;i<=n;i++)

```

```

{
big=0.0;
for(j=1;j<=n;j++)
if(ipiv[j]!=1)
for(k=1;k<=n;k++)
{
if(ipiv[k]==0)
{
if(fabs(a[j][k])>=big)
{
big=fabs(a[j][k]);
irow=j;
icol=k;
}
}
else if(ipiv[k]>1) nerror("GAUSSJ:Singular matrix-1");
}
++(ipiv[icol]);
/*indx[i]=the column of ith pivot element */
/*indxr[i]=the row in which that pivot element was originally located*/
if(irow!=icol)
{
for(l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
for(l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
}
/*divide the pivot row by the pivot element located at irow and icol*/
indxr[i]=irow;
indx[i]=icol;
if(a[icol][icol]==0.0) nerror("GAUSSJ:Singular matrix-2");
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
for(l=1;l<=n;l++)
a[icol][l]*=pivinv;
for(l=1;l<=m;l++) b[icol][l]*=pivinv;
for(ll=1;ll<=n;ll++)
if(ll!=icol)
{
dum=a[ll][icol];
a[ll][icol]=0.0;
for(l=1;l<=n;l++) a[ll][l]-=a[icol][l]*dum;
for(l=1;l<=m;l++)

```

```

b[l1][l1]-=b[icol][l1]*dum;
}
}
for(l=n;l>=1;l--)
{
if(indxr[l]!=indxc[l])
for(k=1;k<=n;k++)
SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
free_ivector(ipiv,1,n);
free_ivector(indxr,1,n);
free_ivector(indxc,1,n);
}
void nrerror(char error_text[])
{ fprintf(stderr,"Numerical Recipes run-time error....\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"....now exiting to system...\n");
  exit(0);
}
int *ivector(int nl,int nh)
/*allocation an int vector with range [nl.....nh]*/
{int *v;
v=(int*)malloc((unsigned)(nh-nl+1)*sizeof(int));
if(!v)nrerror("allocation failure in vector()");
return v-nl;
}
void mult(float **z,float **x,float **y)
{
int i,j,k;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
for(k=1;k<=dim;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void subtr(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)

```

```

for(j=1;j<=dim;j++)
z[i][j]=0;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]-y[i][j];
}
void display1(float **y)
{
int i,j;
for(i=1;i<=ldim;i++)
{
for(j=1;j<=1;j++)
fprintf(fp,"%f\t",y[i][j]);
fprintf(fp,"\n");
}
}

```

Örnek 1:

```

1.000000 0.000000
0.000000 -1.000000

```

```

0.000000 1.000000
0.000000 0.000000

```

```

0.000000 0.000000
1.000000 0.000000
Lie product of l[1][1]

```

```

0.000000 0.000000
0.000000 0.000000
Lie product of l[1][2]

```

```

0.000000 2.000000
0.000000 0.000000
Lie product of l[1][3]

```

```

0.000000 0.000000
-2.000000 0.000000

```

Lie product of $l[2][1]$

0.000000 -2.000000

0.000000 0.000000

Lie product of $l[2][2]$

0.000000 0.000000

0.000000 0.000000

Lie product of $l[2][3]$

1.000000 0.000000

0.000000 -1.000000

Lie product of $l[3][1]$

0.000000 0.000000

2.000000 0.000000

Lie product of $l[3][2]$

-1.000000 0.000000

0.000000 1.000000

Lie product of $l[3][3]$

0.000000 0.000000

0.000000 0.000000

finding the adjoint of a given element of the Lie algebra

Enter the element of the Lie Algebra as a matrix

-2.000000 0.000000

1.000000 2.000000

0.000000 0.000000

2.000000 0.000000

0.000000

0.000000

2.000000

-1.000000 -4.000000
0.000000 1.000000
-1.000000
-4.000000
0.000000

0.000000 0.000000
4.000000 0.000000
0.000000
0.000000
4.000000

1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
0.000000
0.000000
2.000000

1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
-1.000000
-4.000000
0.000000

1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
0.000000
0.000000
4.000000

0.000000
4.000000

The adjoint of the given matrix A

0.000000 -1.000000 0.000000
0.000000 -4.000000 0.000000
2.000000 0.000000 4.000000

Killing Formu

L bir Lie cebiri olsun. $x, y \in L$ ise $K(x, y) = \text{Tr}(\text{adx ady})$ L üzerindeki bir simetrik bilineer form olarak tarif edilir ve Killing Formu olarak adlandırılır.

Bu formun hesaplanması ile bir Lie cebirinin semi simple olup olmadığı test edilebilir.

Teorem: L bir Lie cebiri olsun. Eğer L semi simple ise Killing formu yozlaşmamıştır. Bunun terside doğrudur.

Biliyoruz ki $K(x, y)$ ancak ve ancak $\det(K(e_i, e_j)) \neq 0$ ise yozlaşmamıştır. ($\{e_1, e_2, e_3, \dots, e_n\}$ L Lie cebirinin bazıdır.)

Şimdi bu matrisi hesaplayacak bir program hazırlayalım.

Kmat.cpp

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SWAP(a,b){float temp=(a);(a)=(b);(b)=temp;}
void gaussjo(float **a,int n,float **b,int m);
void free_ivector(int *v,int n1,int nh);
int *ivector(int n1,int nh);
void nrerror(char error_text[]);
void display1(float **y);
void init_matrix(float **x,int k);
void display(float **x,int k);
float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y,int w);
void subt(float **z,float **x,float **y);
int dim,ldim;
float trace(float **x,int k);
main()
{
```

```

int i,j,r,s,k,c;
float **b[100],**son[100][100],**admul[100][100]\
,**ad[100],**m1[100][100],**p,**killing,**q[100][100];
printf("Enter dimension of base elements as matrices\n");
scanf("%d",&dim);
ldim=dim*dim-1;
for(i=1;i<=ldim;i++)
{
ad[i]=matrix(1,ldim,1,ldim);
}
p=matrix(1,ldim,1,ldim);
killing=matrix(1,ldim,1,ldim);
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
{
{
q[k][j]=matrix(1,ldim,1,1);
}
}
for(i=1;i<=ldim;i++)
for(k=1;k<=ldim;k++)
{
{
b[i]=matrix(1,dim,1,dim);
son[i][k]=matrix(1,dim,1,dim);
m1[i][k]=matrix(1,dim,1,dim);
}
}
for(i=1;i<=ldim;i++)
for(k=1;k<=ldim;k++)
{
{
admul[i][k]=matrix(1,ldim,1,ldim);
}
}
for(i=1;i<=ldim;i++)
{
printf("Enter the base element b[%d]\n",i);
init_matrix(b[i],dim);
display(b[i],dim);
}
}

```

```

for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
mult(m1[i][j],b[i],b[j],dim);
}
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{printf("Lie product : son[%d][%d]\n",i,j);
{
subt(son[i][j],m1[i][j],m1[j][i]);
display(son[i][j],dim);
}
printf("\n\n");
}
printf("Finding the adjoint of basis elements\
of the Lie algebra\n ");
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
{
{
/*mult(sol,b[k],b[j],dim);
mult(sag,b[j],b[k],dim);
subt(son[k][j],sol,sag);
display(son[k][j],dim);*/
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)
{q[k][j][(r-1)*dim+s][1]=son[k][j][r][s];}
for(s=1;s<=dim-1;s++)
{q[k][j][(dim-1)*dim+s][1]=son[k][j][r][s];}
printf("\n");
display1(q[k][j]);
}
}
printf("Enter the coordinates of base elements\
except (n,n)th coordinate\n");
init_matrix(p,ldim);
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
{

```

```

{
gaussjo(p,ldim,q[k][j],1);
printf("The inverse of the coefficient matrix\n");
display(p,ldim);
printf("\n\n");
display1(q[k][j]);
}
}
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
for(i=1;i<=ldim;i++)
{
{
ad[k][i][j]=q[k][j][i][1];
}
}
}
for(k=1;k<=ldim;k++)
{
printf("The adjoint of the base element b[%d]\n",k);
display(ad[k],ldim);
printf("\n\n");
}
for(k=1;k<=ldim;k++)
for(c=1;c<=ldim;c++)
{
mult(admul[k][c],ad[k],ad[c],ldim);
killing[k][c]=trace(admul[k][c],ldim);
}
display(killing,ldim);
return 0;
}
void init_matrix(float **x,int k)
{
float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}

```

```

}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{for(j=1;j<=k;j++)
{printf("%f\t",x[i][j]);
}
printf("\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
/*return pointer to array of pointers to rows*/
return m;
}
void free_ivector(int *v,int nl,int nh)
{
free((char*)(v+nl));
}
void gaussjo(float **a,int n,float **b,int m)
{int *indxc,*indxr,*ipiv;
int i,icol,irow,j,k,l,ll;
float big,dum,pivinv;
indxc=ivector(1,n);
indxr=ivector(1,n);
ipiv=ivector(1,n);
for(j=1;j<=n;j++) ipiv[j]=0;

```

```

for(i=1;i<=n;i++)
{
big=0.0;
for(j=1;j<=n;j++)
if(ipiv[j]!=1)
for(k=1;k<=n;k++)
{
if(ipiv[k]==0)
{
if(fabs(a[j][k])>=big)
{
big=fabs(a[j][k]);
irow=j;
icol=k;
}
}
else if(ipiv[k]>1) nerror("GAUSSJ:Singular matrix-1");
}
++(ipiv[icol]);
/*indxc[i]=the column of ith pivot element */
/*indxr[i]=the row in which that pivot element\
was originally located*/
if(irow!=icol)
{
for(l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
for(l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
}
/*divide the pivot row by the pivot element\
located at irow and icol*/
indxr[i]=irow;
indxc[i]=icol;
if(a[icol][icol]==0.0) nerror("GAUSSJ:Singular matrix-2");
pivin=1.0/a[icol][icol];
a[icol][icol]=1.0;
for(l=1;l<=n;l++)
a[icol][l]*=pivin;
for(l=1;l<=m;l++) b[icol][l]*=pivin;
for(ll=1;ll<=n;ll++)
if(ll!=icol)
{
dum=a[ll][icol];

```

```

a[l1][icol]=0.0;
for(l=1;l<=n;l++) a[l1][l]-=a[icol][l]*dum;
for(l=1;l<=m;l++)
b[l1][l]-=b[icol][l]*dum;
}
}
for(l=n;l>=1;l--)
{
if(indxr[l]!=indxc[l])
for(k=1;k<=n;k++)
SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
free_ivector(ipiv,1,n);
free_ivector(indxr,1,n);
free_ivector(indxc,1,n);
}
void nrerror(char error_text[])
{fprintf(stderr,"Numerical Recipes run-time error...\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,".....now exiting to system...\n");
exit(0);
}
int *ivector(int nl,int nh)
/*allocation an int vector with range [nl.....nh]*/
{int *v;
v=(int*)malloc((unsigned)(nh-nl+1)*sizeof(int));
if(!v)nrerror("allocation failure in vector()");
return v-nl;
}
void mult(float **z,float **x,float **y,int w)
{
int i,j,k;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
z[i][j]=0;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
for(k=1;k<=w;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void subtr(float **z,float **x,float **y)

```

```

{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]-y[i][j];
}
void display1(float **y)
{
int i,j;
for(i=1;i<=ldim;i++)
{
for(j=1;j<=l;j++)
printf("%ft",y[i][j]);
printf("\n");
}
}
float trace(float **x,int k)
{int i;
float sum=0;
for(i=1;i<=k;i++)
{ sum+=x[i][i];}
return sum;
}

```

Programımızın çıktısını dosyalayacak şekilde programımızı yeniden yazalım.

Fkmat.cpp

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SWAP(a,b){float temp=(a);(a)=(b);(b)=temp;}
void gaussjo(float **a,int n,float **b,int m);
void free_ivector(int *v,int n1,int nh);
int *ivector(int n1,int nh);
void perror(char error_text[]);
void display1(float **y);
void init_matrix(float **x,int k);
void display(float **x,int k);

```

```

float **matrix(int nrl,int nrh,int ncl,int nch);
void mult(float **z,float **x,float **y,int w);
void subt(float **z,float **x,float **y);
int dim,ldim;
float trace(float **x,int k);
FILE *fp;
main()
{
int i,j,r,s,k,c;
float
**b[100],**son[100][100],**admul[100][100],**ad[100],**m1[100][100],
**p,**killing,**q[100][100];
if((fp=fopen("kmat","wb"))==NULL){
printf("can not open file\n");
exit(1);
}
printf("Enter dimension of base elements as matrices\n");
scanf("%d",&dim);
ldim=dim*dim-1;
for(i=1;i<=ldim;i++)
{
ad[i]=matrix(1,ldim,1,ldim);
}
p=matrix(1,ldim,1,ldim);
killing=matrix(1,ldim,1,ldim);
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
{
{
q[k][j]=matrix(1,ldim,1,1);
}
}
for(i=1;i<=ldim;i++)
for(k=1;k<=ldim;k++)
{
{
b[i]=matrix(1,dim,1,dim);
son[i][k]=matrix(1,dim,1,dim);
m1[i][k]=matrix(1,dim,1,dim);
}
}
}
}

```

```

for(i=1;i<=ldim;i++)
for(k=1;k<=ldim;k++)
{
{
admul[i][k]=matrix(1,ldim,1,ldim);
}
}
for(i=1;i<=ldim;i++)
{
printf("Enter the base element b[%d]\n",i);
init_matrix(b[i],dim);
display(b[i],dim);
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{
{
mult(m1[i][j],b[i],b[j],dim);
}
}
for(i=1;i<=ldim;i++)
for(j=1;j<=ldim;j++)
{printf("Lie product : son[%d][%d]\n",i,j);
{
subt(son[i][j],m1[i][j],m1[j][i]);
display(son[i][j],dim);
}
printf("\n\n");
}
printf("Finding the adjoint of basis elements\
of the Lie algebra\n ");
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
{
{
/*mult(sol,b[k],b[j],dim);
mult(sag,b[j],b[k],dim);
subt(son[k][j],sol,sag);
display(son[k][j],dim);*/
for(r=1;r<=dim-1;r++)
for(s=1;s<=dim;s++)

```

```

{q[k][j][(r-1)*dim+s][1]=son[k][j][r][s];}
  for(s=1;s<=dim-1;s++)
    {q[k][j][(dim-1)*dim+s][1]=son[k][j][r][s];}
  printf("\n");
  display1(q[k][j]);
}
}
printf("Enter the coordinates of base elements\
except (n,n)th coordinate\n");
init_matrix(p,ldim);
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
{
{
gaussjo(p,ldim,q[k][j],1);
fprintf(fp,"The inverse of the coefficient matrix\n");
display(p,ldim);
printf("\n\n");
display1(q[k][j]);
}
}
for(k=1;k<=ldim;k++)
for(j=1;j<=ldim;j++)
for(i=1;i<=ldim;i++)
{
{
ad[k][i][j]=q[k][j][i][1];
}
}
}
for(k=1;k<=ldim;k++)
{
fprintf(fp,"The adjoint of the base element b[%d]\n",k);
display(ad[k],ldim);
printf("\n\n");
}
for(k=1;k<=ldim;k++)
for(c=1;c<=ldim;c++)
{
mult(admul[k][c],ad[k],ad[c],ldim);

```

```

killing[k][c]=trace(admul[k][c],ldim);
}
fprintf(fp,"Killing Form is\n");
display(killing,ldim);
return 0;
}
void init_matrix(float **x,int k)
{
float r;
int i,j;
for(i=1;i<=k;i++)
for(j=1;j<=k;j++)
{printf("Enter the coefficient x[%d][%d]\n",i,j);
scanf("%f",&r);x[i][j]=r;
}
}
void display(float **x,int k)
{
int i,j;
for(i=1;i<=k;i++)
{
fprintf(fp,"\n");
for(j=1;j<=k;j++)
fprintf(fp,"%f ",x[i][j]);
}
fprintf(fp,"\n");
}
}
float **matrix(int nrl,int nrh,int ncl,int nch)
{
int i;
float **m;
/*allocate pointers to rows*/
m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
if(!m) printf("allocation failure 1 in matrix()");
m-=nrl;
/*allocate rows and set pointers to them*/
for(i=nrl;i<=nrh;i++){
m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
if(!m[i]) printf("allocation failure 2 in matrix()");
m[i]-=ncl;
}
}

```

```

}
/*return pointer to array of pointers to rows*/
return m;
}
void free_ivector(int *v,int nl,int nh)
{
free((char*)(v+nl));
}
void gaussjo(float **a,int n,float **b,int m)
{int *indxc,*indxr,*ipiv;
int i,icol,irow,j,k,l,ll;
float big,dum,pivinv;
indxc=ivector(1,n);
indxr=ivector(1,n);
ipiv=ivector(1,n);
for(j=1;j<=n;j++) ipiv[j]=0;
for(i=1;i<=n;i++)
{
big=0.0;
for(j=1;j<=n;j++)
if(ipiv[j]!=1)
for(k=1;k<=n;k++)
{
if(ipiv[k]==0)
{
if(fabs(a[j][k])>=big)
{
big=fabs(a[j][k]);
irow=j;
icol=k;
}
}
else if(ipiv[k]>1) nrerror("GAUSSJ:Singular matrix-1");
}
++(ipiv[icol]);
/*indxc[i]=the column of ith pivot element */
/*indxr[i]=the row in which that pivot element\
was originally located*/
if(irow!=icol)
{
for(l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
}
}
}
}

```

```

for(l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
}
/*divide the pivot row by the pivot element\
located at irow and icol*/
indxr[i]=irow;
indxc[i]=icol;
if(a[icol][icol]==0.0) nrerror("GAUSSJ:Singular matrix-2");
pivin=1.0/a[icol][icol];
a[icol][icol]=1.0;
for(l=1;l<=n;l++)
a[icol][l]*=pivin;
for(l=1;l<=m;l++) b[icol][l]*=pivin;
for(ll=1;ll<=n;ll++)
if(ll!=icol)
{
dum=a[ll][icol];
a[ll][icol]=0.0;
for(l=1;l<=n;l++) a[ll][l]-=a[icol][l]*dum;
for(l=1;l<=m;l++)
b[ll][l]-=b[icol][l]*dum;
}
}
for(l=n;l>=1;l--)
{
if(indxr[l]!=indxc[l])
for(k=1;k<=n;k++)
SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
free_ivector(ipiv,1,n);
free_ivector(indxr,1,n);
free_ivector(indxc,1,n);
}
void nrerror(char error_text[])
{fprintf(stderr,"Numerical Recipes run-time error....\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"....now exiting to system...\n");
exit(0);
}
int *ivector(int nl,int nh)
/*allocation an int vector with range [nl.....nh]*/
{int *v;

```

```

v=(int*)malloc((unsigned)(nh-nl+1)*sizeof(int));
if(!v)nerror("allocation failure in vector()");
return v-nl;
}
void mult(float **z,float **x,float **y,int w)
{
int i,j,k;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
z[i][j]=0;
for(i=1;i<=w;i++)
for(j=1;j<=w;j++)
for(k=1;k<=w;k++)
z[i][j]+=x[i][k]*y[k][j];
}
void subtr(float **z,float **x,float **y)
{
int i,j;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=0;
for(i=1;i<=dim;i++)
for(j=1;j<=dim;j++)
z[i][j]=x[i][j]-y[i][j];
}
void display1(float **y)
{
int i,j;
for(i=1;i<=ldim;i++)
{
for(j=1;j<=1;j++)
printf("%f\t",y[i][j]);
printf("\n");
}
}
float trace(float **x,int k)
{int i;
float sum=0;
for(i=1;i<=k;i++)
{ sum+=x[i][i];}
return sum;
}

```

}

Örnek 1:

0.000000 1.000000
0.000000 0.000000

1.000000 0.000000
0.000000 -1.000000

0.000000 0.000000
1.000000 0.000000

0.000000 0.000000
0.000000 0.000000
0.000000 -2.000000

0.000000 0.000000
0.000000 0.000000
0.000000 -1.000000

0.000000 2.000000
0.000000 0.000000
0.000000 0.000000

0.000000 0.000000
0.000000 0.000000
-2.000000 0.000000

0.000000 0.000000
0.000000 1.000000
0.000000 0.000000

2.000000 0.000000

0.000000 0.000000
0.000000 0.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000

1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000

1.000000 0.000000 0.000000

0.000000 0.000000 1.000000

The inverse of the coefficient matrix

0.000000 1.000000 0.000000

1.000000 0.000000 0.000000

0.000000 0.000000 1.000000

The adjoint of the base element b[1]

0.000000 -2.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

The adjoint of the base element b[2]

2.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 -2.000000

The adjoint of the base element b[3]

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 2.000000 0.000000

Killing Form is

0.000000 0.000000 0.000000

0.000000 8.000000 0.000000

0.000000 0.000000 0.000000

KAYNAKLAR

1-Introduction To Lie Algebras and Representation Theory
James E.Humphreys

2-Numerical Recipes in C

William H.Press
Brian P. Flannery
Soul A.Teukolsky
William T.Vetterling

3-C The Complete Reference

Herbert Schildt

