objective-function(void) I

```
double x1,x2,x3;
int i;
for(i = 1; i <= POP-SIZE; i++)
{
x1 = CHROMOSOME[i][1];
x2 = CHROMOSOME[i][2];
x3 = CHROMOSOME[i][3];
OBJECTIVE[i][1] = sqrt(x1)+sqrt(x2)+sqrt(x3);
}
```

objective-function(void) II

```
for(i=1;i<=POP-SIZE;i++)
OBJECTIVE[i][0]= OBJECTIVE[i][1];
```

constraint-check(double x[])

```
double a;
int n;
for(n=1;n<=N;n++) if(x[n]<0) return 0;
a = x[1]*x[1]+2*x[2]*x[2]+3*x[3]*x[3];
if(a>1) return 0;
return 1;
```

initialization(void)

```
double x[N+1];
int i,j;
for(i=1; i<=POP-SIZE; i++)
{
mark:
for(j=1; j<=N; j++) x[j]=myu(0,1);
if(constraint-check(x)==0) goto mark;
for(j=1; j<=N; j++) CHROMOSOME[i][j]=x[j];
}
```

main() I

```
int i, j;
double a;
q[0]=0.05; a=0.05;
for(i=1; i<=POP-SIZE; i++) {a=a*0.95; q[i]=q[i-1]+a;}
initialization();
evaluation(0);
for(i=1; i<=GEN; i++)
{
selection();
```

main() II

```
crossover();
mutation();
evaluation(i);
printf("\n Generation NO.% d \n", i);
printf("x=(");
for(j=1; j<=N; j++)
{
if(j<N) printf("%3.4f,",CHROMOSOME[0][j]);
else printf("%3.4f",CHROMOSOME[0][j]);
}
```

main() III

```
if(M==1) printf(")\n f=%3.4f \n", OBJECTIVE[0][1]);
else
{
printf(")\n f=(");
for(j=1; j<=M; j++)
{
if(j<M) printf("%3.4f,", OBJECTIVE[0][j]);
else printf("%3.4f', OBJECTIVE[0][j]);
}
```

main() IV

```
printf(") Aggregating Value=%3.4f \n",
OBJECTIVE[0][0]);
}
}
printf("\n");
return 1;
```

evaluation(int gen) I

```
double a;
int i, j, k, label;
objective-function();
if(gen==0)
{
for(k=0; k<=M; k++)
OBJECTIVE[0][k]=OBJECTIVE[1][k];
for(j = 1; j <= N; j++)
CHROMOSOME[0][j]=CHROMOSOME[1][j];
```

evaluation(int gen) II

```
}
for(i=0; i<POP-SIZE; i++)
{
label=0; a=OBJECTIVE[i][0];
for(j=i+1; j<=POP-SIZE; j++)
if((TYPE*a)<(TYPE*OBJECTIVE[j][0]))
{
a=OBJECTIVE[j][0];
label=j;
}
```

evaluation(int gen) III

```
if(label!=0)
{
for(k=0; k<=M; k++)
{
a=OBJECTIVE[i][k];
OBJECTIVE[i][k]=OBJECTIVE[label][k];
OBJECTIVE[label][k]=a;
}
for(j=1; j<=N; j++)
{
```

evaluation(int gen) IV

```
a=CHROMOSOME[i][j];
CHROMOSOME[i][j]=CHROMOSOME[label][j];
CHROMOSOME[label][j]=a;
}
}
}
```

selection() I

```
double r, temp[POP-SIZE+1][N+1];
int i, j, k;
for(i=1; i<=POP-SIZE; i++)
{
r=myu(0, q[POP-SIZE]);
for(j=0; j<=POP-SIZE; j++)
{
if(r<=q[j])
{
```

selection() II

```
for(k=1; k<=N; k++) temp[i][k]=CHROMOSOME[j][k];
break;
}
}
}
for(i=1; i<=POP-SIZE; i++)
for(k=1; k<=N; k++)
CHROMOSOME[i][k]=temp[i][k];
```

crossover() I

```
int i, j, jj, k, pop;
double r, x[N+1], y[N+1];
pop=POP-SIZE/2;
for(i=1; i<=pop; i++)
{
if(myu(0,1)>P-CROSSOVER) continue;
j=(int)myu(1,POP-SIZE);
jj=(int)myu(1,POP-SIZE);
r=myu(0,1);
```

crossover() II

```
for(k=1; k<=N; k++)
{
x[k]=r*CHROMOSOME[j][k]+(1-
r)*CHROMOSOME[jj][k];
y[k]=r*CHROMOSOME[jj][k]+(1-
r)*CHROMOSOME[j][k];
}
if(constraint-check(x)==1)
for(k=1; k<=N; k++) CHROMOSOME[j][k]=x[k];
if(constraint-check(y)==1)
```

crossover() III

```
for(k=1; k<=N; k++) CHROMOSOME[jj][k]=y[k];
}
```

mutation(void) I

```
int i, j, k;
double x[N+1], y[N+1], infty, direction[N+1];
double INFTY=10, precision=0.0001;
for(i=1; i<=POP-SIZE; i++)
{
if(myu(0,1)>P-MUTATION) continue;
for(k=1; k<=N; k++) x[k] = CHROMOSOME[i][k];
for(k=1; k<=N; k++)
if(myu(0,1)<0.5) direction[k]=myu(-1,1);
```

mutation(void) II

```
else direction[k]=0;
infty=myu(0,INFTY);
while(infty>precision)
{
for(j=1; j<=N; j++) y[j]=x[j]+infty*direction[j];
if(constraint-check(y)==1)
{
for(k=1; k<=N; k++) CHROMOSOME[i][k]=y[k];
break;
}
```

mutation(void) III

```
infty=myu(0,infty);
}
}
```