Program codes for DE and GA

The code for calculating the heat transfer area, and the header file, header.h- containing various constants defined for the pseudo random number generator, are common to both DE and GA.

header.h file:

```
 #include "stdio.h"
 /* #include "conio.h" */      /* conio.h not available on all platforms */
 #include "stdlib.h"
 #include "math.h"
 #include "memory.h"

 #define MAXPOP 500
 #define MAXDIM 35

 /*------Constants for rnd_uni()------------------------------------*/
 #define IM1 2147483563
 #define IM2 2147483399
 #define AM (1.0/IM1)
 #define IMM1 (IM1-1)
 #define IA1 40014
 #define IA2 40692
 #define IQ1 53668
 #define IQ2 52774
 #define IR1 12211
 #define IR2 3791
 #define NTAB 32
 #define NDIV (1+IMM1/NTAB)
 #define EPS 1.2e-7
 #define RNMX (1.0-EPS)
```

C.1. Code for GA

```
#include"header.h"
#include"area.c"


float dpt[1000],dps[1000];
int feval;
long  rnd_uni_init;
float rnd_uni(long *idum);    /* uniform pseudo random number generator */


float rnd_uni(long *idum)
{
  long j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  float temp;

  if (*idum <= 0)
  {
    if (-(*idum) < 1) *idum=1;
    else *idum = -(*idum);
    idum2=(*idum);
    for (j=NTAB+7;j>=0;j--)
    {
      k=(*idum)/IQ1;
      *idum=IA1*(*idum-k*IQ1)-k*IR1;
      if (*idum < 0) *idum += IM1;
      if (j < NTAB) iv[j] = *idum;
    }
    iy=iv[0];
  }
     k=(*idum)/IQ1;
  *idum=IA1*(*idum-k*IQ1)-k*IR1;
  if (*idum < 0) *idum += IM1;
  k=idum2/IQ2;
  idum2=IA2*(idum2-k*IQ2)-k*IR2;
  if (idum2 < 0) idum2 += IM2;
  j=iy/NDIV;
  iy=iv[j]-idum2;
  iv[j] = *idum;
  if (iy < 1) iy += IMM1;
  if ((temp=AM*iy) > RNMX) return RNMX;
  else return temp;

}/*------End of rnd_uni()--------------------------*/


main()

{

struct pop{
           short  s[7];

           } set1[120],set2[120],set3[120];

int   sno[120],scount[120],gen,maxcount,seed,N;
float f[120],area,F[120];
float prob[120],cprob[120],ran[120],pc,pm;
float sumf,r1;
int
h,k,i,j,l,mask,r,z,Do[120],pitch[120],tema[120],nop[120],len[120],bafs[120],bafc[120];

int   bDo,bpitch,btema,bnop,blen,bbafs,bbafc;
short temp1,temp2,flag1,flag2;

printf(" ENTER SEED :\n");
scanf(" %d",&seed);



for(N=36;N<=100;N=N+4)
{
for(pc=0.5;pc<=1.0;pc=pc+0.05)
{
for(pm=0.05;pm<=0.35;pm=pm+0.05)
{


/*-----Initialize random number generator-----------------------------*/

 rnd_uni_init = -(long)seed;  /* initialization of rnd_uni() */

for(i=0;i<N;i++)
{
Do[i]      =1+ rnd_uni(&rnd_uni_init)*12;
pitch[i]   =1+ rnd_uni(&rnd_uni_init)*2;
tema[i]    =1+ rnd_uni(&rnd_uni_init)*4;
nop[i]     =1+ rnd_uni(&rnd_uni_init)*5;
len[i]     =1+ rnd_uni(&rnd_uni_init)*8;
bafs[i]    =1+ rnd_uni(&rnd_uni_init)*6;
bafc[i]    =1+ rnd_uni(&rnd_uni_init)*7;

set1[i].s[0] =ceil(Do[i]*1023.0/12);
set1[i].s[1] =ceil(pitch[i]*1023.0/2);
set1[i].s[2] =ceil(tema[i]*1023.0/4);
```

```
set1[i].s[3] =ceil(nop[i]*1023.0/5);
set1[i].s[4] =ceil(len[i]*1023.0/8);
set1[i].s[5] =ceil(bafs[i]*1023.0/6);
set1[i].s[6] =ceil(bafc[i]*1023.0/7);


}

gen=0;

while(gen<100)
{

for(i=0;i<N;i++)
{
    for(z=0;z<7;z++)
    {
       if(set1[i].s[z]<0)
       set1[i].s[z]=-set1[i].s[z];
       if(set1[i].s[z]>=1023)
       set1[i].s[z]=1023;
      }

 if(set1[i].s[0]<86)
    set1[i].s[0]=86;
 if(set1[i].s[1]<512)
    set1[i].s[1]=512;
 if(set1[i].s[2]<256)
    set1[i].s[2]=256;
 if(set1[i].s[3]<205)
    set1[i].s[3]=205;
 if(set1[i].s[4]<128)
    set1[i].s[4]=128;
 if(set1[i].s[5]<171)
    set1[i].s[5]=171;
 if(set1[i].s[6]<147)
    set1[i].s[6]=147;

Do[i]      =set1[i].s[0]*12/1023.0;
pitch[i]   =set1[i].s[1]*2 /1023.0;
tema[i]    =set1[i].s[2]*4 /1023.0;
nop[i]     =set1[i].s[3]*5 /1023.0;
len[i]     =set1[i].s[4]*8 /1023.0;
bafs[i]    =set1[i].s[5]*6 /1023.0;
bafc[i]    =set1[i].s[6]*7 /1023.0;


f[i]=cal_area(Do[i],pitch[i],tema[i],nop[i],len[i],bafs[i],bafc[i]);

F[i]=1/(1+f[i]) ;

}

sumf=0;
for(i=0;i<N;i++)
sumf+=F[i];

for(i=0;i<N;i++)
prob[i]=F[i]/sumf;

cprob[0]=prob[0];
for(i=1;i<N;i++)
cprob[i]=cprob[i-1]+prob[i];

for(i=0;i<N;i++)
ran[i]=(float) rnd_uni(&rnd_uni_init);

/* roulette wheel selection*/

for(i=0;i<N;i++)
{
   for(j=0;j<N;j++)
   {
    if(ran[i]<cprob[j])
      {
        sno[i]=j;

        for(z=0;z<7;z++)
        set2[i].s[z]=set1[j].s[z];

        break;
        }
   }
}


for(i=0;i<N;i++)
scount[i]=0;

for(i=0;i<N;i++)
{
for(j=0;j<N;j++)
{
 if(i==sno[j])
  scount[i]++;
}
}

/* now find the best solution so far*/

maxcount=scount[0];

bDo      =set2[0].s[0]*12/1023.0;
bpitch   =set2[0].s[1]*2 /1023.0;
btema    =set2[0].s[2]*4 /1023.0;
bnop     =set2[0].s[3]*5 /1023.0;
blen     =set2[0].s[4]*8 /1023.0;
bbafs    =set2[0].s[5]*6 /1023.0;
bbafc    =set2[0].s[6]*7 /1023.0;


for(i=1;i<N;i++)
{
if(scount[i]>maxcount)
  {  maxcount=scount[i];
bDo      =set2[i].s[0]*12/1023.0;
bpitch   =set2[i].s[1]*2 /1023.0;
btema    =set2[i].s[2]*4 /1023.0;
bnop     =set2[i].s[3]*5 /1023.0;
blen     =set2[i].s[4]*8 /1023.0;
bbafs    =set2[i].s[5]*6 /1023.0;
bbafc    =set2[i].s[6]*7 /1023.0;
   }
}
area=cal_area(bDo,bpitch,btema,bnop,blen,bbafs,bbafc);

if (area<34.5)
{
printf("\n best solution so far for generation %d \n",gen);
printf("\n maxcount=%d  area=%f dpt=%f  dps=%f\n",maxcount,area,dpt[feval],dps[feval]);

printf("%d%d%d%d%d%d%d",bDo,bpitch,btema,bnop,blen,bbafs,bbafc);

printf(" N=%d  pc=%f   pm=%f \n",N,pc,pm);
break;
}

/*crossover*/

for(i=0;i<(N/2);i+=2)
{
  for(z=0;z<7;z++)
 {

r1=(float) rnd_uni(&rnd_uni_init);

if(r1<=pc)

 {
h=1+ rnd_uni(&rnd_uni_init)*10;
k=16-h;

temp1=set2[i].s[z]>>k;
temp1=temp1<<k;
temp2=set2[i+1].s[z]<<h;
temp2=temp2>>h;
set1[i].s[z]=temp1|temp2;

temp1=set2[i+1].s[z]>>k;
temp1=temp1<<k;
temp2=set2[i].s[z]<<h;
temp2=temp2>>h;
set1[i+1].s[z]=temp1|temp2;

  }
else

{
set1[i].s[z]=set2[i].s[z];
set1[i+1].s[z]=set2[i+1].s[z];
}

}
}

/*mutation*/

for(i=0;i<N;i++)
{
for(z=0;z<7;z++)
{
for(j=1;j<=10;j++)
{
r=(float) rnd_uni(&rnd_uni_init);
if(r<=pm)
{
k=16-j;l=j-1;
temp1=set1[i].s[z]>>k;
temp1=temp1<<k;
temp1=temp1<<l;
temp1=temp1>>l;
mask=1;
mask<<k;
temp2=~mask;
if((temp1&mask)==0)
set3[i].s[z]=set1[i].s[z]|mask;
else
set3[i].s[z]=set1[i].s[z]&temp2;
}
set1[i].s[z]=set3[i].s[z];

}
}
}
feval=0;
/*end of all operators*/

gen++;
}

/*
}
}
```

```
printf("\n------------------------------------------------------\n");
}
*/

}
```

C.2. Code for DE

The code of DE requires the following input file:

de.dat file:

```
1
15
5
7
70
1
0
0.7
1.0
10

choice of method
maximum no. of iterations
Output refresh cycle
No. of parameters D
No. of parents NP
Upper bound of parameter values
Lower bound of parameters values
Constant F
Crossing Over factor CR
seed for pseudo random number generator
```

The code of DE follows:

```
# include "header.h"
# include "area.c"

long  rnd_uni_init, nfeval;
float c[MAXPOP][MAXDIM], d[MAXPOP][MAXDIM];
float (*pold)[MAXPOP][MAXDIM],(*pnew)[MAXPOP][MAXDIM],(*pswap)[MAXPOP][MAXDIM];

void  assignd(int D, float a[], float b[]);
float rnd_uni(long *idum);    /* uniform pseudo random number generator */
float evaluate(int D, float tmp[]); /* obj. funct. */


void  assignd(int D, float a[], float b[])
{
   int j;
   for (j=0; j<D; j++)
   {
      a[j] = b[j];
   }
}

float rnd_uni(long *idum)
{
  long j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  float temp;

  if (*idum <= 0)
  {
    if (-(*idum) < 1) *idum=1;
    else *idum = -(*idum);
    idum2=(*idum);
    for (j=NTAB+7;j>=0;j--)
    {
      k=(*idum)/IQ1;
      *idum=IA1*(*idum-k*IQ1)-k*IR1;
      if (*idum < 0) *idum += IM1;
      if (j < NTAB) iv[j] = *idum;
    }
    iy=iv[0];
  }
  k=(*idum)/IQ1;
  *idum=IA1*(*idum-k*IQ1)-k*IR1;
  if (*idum < 0) *idum += IM1;
  k=idum2/IQ2;
  idum2=IA2*(idum2-k*IQ2)-k*IR2;
  if (idum2 < 0) idum2 += IM2;
  j=iy/NDIV;
  iy=iv[j]-idum2;
  iv[j] = *idum;
  if (iy < 1) iy += IMM1;
  if ((temp=AM*iy) > RNMX) return RNMX;
  else return temp;

}/*------End of rnd_uni()--------------------------*/



main(int argc, char *argv[])

{
   char  chr;             /* y/n choice variable                */
   char  *strat[15];      /* strategy-indicator                 */

   int   i, j, L, n;      /* counting variables                 */
   int   r1, r2, r3, r4;  /* placeholders for random indexes    */
   int   r5;              /* placeholders for random indexes    */
   int   D;               /* Dimension of parameter vector      */
   int   NP;              /* number of population members       */
   int   imin;            /* index to member with lowest energy */
   int   refresh;         /* refresh rate of screen output      */
   int   strategy;        /* choice parameter for screen output */
   int   gen, genmax, seed;
   int   dia,pitch,tema,nop,len,bafs,bafc;   /* for output only */
   float trial_cost;      /* buffer variable                    */
   float inibound_h;      /* upper parameter bound              */
   float inibound_l;      /* lower parameter bound              */
   float tmp[MAXDIM], best[MAXDIM], bestit[MAXDIM]; /* members  */
   float cost[MAXPOP];    /* obj. funct. values                 */
   float cvar;            /* computes the cost variance         */
   float cmean;           /* mean cost                          */
   float F,CR;            /* control variables of DE            */
   float cmin;            /* help variables                     */

   FILE  *fpin_ptr;
   FILE  *fpout_ptr;



/*------Initializations----------------------------*/

 if (argc != 3)                                 /* number of arguments */
 {
    printf("\nUsage : de <input-file> <output-file>\n");
    exit(1);
 }


 fpout_ptr = fopen(argv[2],"r");          /* Open Output file for writing */

 if ( fpout_ptr != NULL )
 {
    printf("\nOutput file %s does already exist, \ntype y if you ",argv[2]);
    printf("want to overwrite it, \nanything else if you want to exit.\n");
    chr = (char)getchar();
    if ((chr != 'y') && (chr != 'Y'))
    {
      exit(1);
    }
 }
 fclose(fpout_ptr);

 strat[1]  = "DE/best/1/exp          ";
 strat[2]  = "DE/rand/1/exp          ";
 strat[3]  = "DE/rand-to-best/1/exp  ";
 strat[4]  = "DE/best/2/exp          ";
 strat[5]  = "DE/rand/2/exp          ";
 strat[6]  = "DE/best/1/bin          ";
 strat[7]  = "DE/rand/1/bin          ";
 strat[8]  = "DE/rand-to-best/1/bin  ";
 strat[9]  = "DE/best/2/bin          ";
 strat[10]  = "DE/rand/2/bin          ";


/*-----Read input data-------------------------------------------------*/

 fpin_ptr   = fopen(argv[1],"r");

 if (fpin_ptr == NULL)
 {
    printf("\nCannot open input file\n");
    exit(1);                                 /* input file is necessary */
 }

 fscanf(fpin_ptr,"%d",&strategy);     /*---choice of strategy------------------*/
 fscanf(fpin_ptr,"%d",&genmax);       /*---maximum number of generations-------*/
 fscanf(fpin_ptr,"%d",&refresh);      /*---output refresh cycle----------------*/
 fscanf(fpin_ptr,"%d",&D);            /*---number of parameters----------------*/
 fscanf(fpin_ptr,"%d",&NP);           /*---population size.--------------------*/
 fscanf(fpin_ptr,"%f",&inibound_h);   /*---upper parameter bound for init------*/
 fscanf(fpin_ptr,"%f",&inibound_l);   /*---lower parameter bound for init------*/
 fscanf(fpin_ptr,"%f",&F);            /*---weight factor-----------------------*/
 fscanf(fpin_ptr,"%f",&CR);           /*---crossing over factor----------------*/
 fscanf(fpin_ptr,"%d",&seed);         /*---random seed-------------------------*/


 fclose(fpin_ptr);

/*-----Checking input variables for proper range-----------------------------*/

  if (D > MAXDIM)
  {
     printf("\nError! D=%d > MAXDIM=%d\n",D,MAXDIM);
     exit(1);
  }
  if (D <= 0)
  {
     printf("\nError! D=%d, should be > 0\n",D);
     exit(1);
  }
  if (NP > MAXPOP)
  {
     printf("\nError! NP=%d > MAXPOP=%d\n",NP,MAXPOP);
     exit(1);
  }
  if (NP <= 0)
  {
     printf("\nError! NP=%d, should be > 0\n",NP);
     exit(1);
  }
  if ((CR < 0) || (CR > 1.0))
  {
     printf("\nError! CR=%f, should be ex [0,1]\n",CR);
     exit(1);
  }
  if (seed <= 0)
  {
     printf("\nError! seed=%d, should be > 0\n",seed);
     exit(1);
  }
  if (refresh <= 0)
  {
     printf("\nError! refresh=%d, should be > 0\n",refresh);
     exit(1);
  }
  if (genmax <= 0)
  {
     printf("\nError! genmax=%d, should be > 0\n",genmax);
     exit(1);
  }
  if ((strategy < 0) || (strategy > 10))
  {
 printf("\nError! strategy=%d, should be ex {1,2,3,4,5,6,7,8,9,10}\n",strategy);
     exit(1);
  }
  if (inibound_h < inibound_l)
  {
     printf("\nError! inibound_h=%f < inibound_l=%f\n",inibound_h, inibound_l);
     exit(1);
  }


/*-----Open output file-----------------------------------------------*/

   fpout_ptr   = fopen(argv[2],"w");

   if (fpout_ptr == NULL)
   {
      printf("\nCannot open output file\n");
      exit(1);
   }


/*-----Initialize random number generator-----------------------------*/

 rnd_uni_init = -(long)seed;  /* initialization of rnd_uni() */
 nfeval       =  0;  /* reset number of function evaluations */



/*------Initialization------------------------------------------------*/
/*------Right now this part is kept fairly simple and just generates--*/
/*------random numbers in the range [-initfac, +initfac]. You might---*/
/*------want to extend the init part such that you can initialize-----*/
/*------each parameter separately.------------------------------------*/

   for (i=0; i<NP; i++)
   {
      for (j=0; j<D; j++) /* spread initial population members */
      {
 c[i][j] = inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h - inibound_l);
      }
      cost[i] = evaluate(D,c[i]); /* obj. funct. value */
    }
   cmin = cost[0];
   imin = 0;
   for (i=1; i<NP; i++)
   {
      if (cost[i]<cmin)
      {
        cmin = cost[i];
        imin = i;
      }
   }

   assignd(D,best,c[imin]);            /* save best member ever          */
   assignd(D,bestit,c[imin]);          /* save best member of generation */

   pold = &c; /* old population (generation G)   */
   pnew = &d; /* new population (generation G+1) */

/*=======================================================================*/
/*=========Iteration loop================================================*/
/*=======================================================================*/

   gen = 0;                          /* generation counter reset */
   while ((gen < genmax) /*&& (kbhit() == 0)*/) /* remove comments if conio.h */
   {                                            /* is accepted by compiler    */
      gen++;
      imin = 0;

      for (i=0; i<NP; i++)         /* Start of loop through ensemble  */
      {
        do                        /* Pick a random population member */
        {                         /* Endless loop for NP < 2 !!!     */
          r1 = (int)(rnd_uni(&rnd_uni_init)*NP);
        }while(r1==i);

        do                        /* Pick a random population member */
        {                         /* Endless loop for NP < 3 !!!     */
          r2 = (int)(rnd_uni(&rnd_uni_init)*NP);
        }while((r2==i) || (r2==r1));

        do                        /* Pick a random population member */
        {                         /* Endless loop for NP < 4 !!!     */
          r3 = (int)(rnd_uni(&rnd_uni_init)*NP);
        }while((r3==i) || (r3==r1) || (r3==r2));

        do                        /* Pick a random population member */
        {                         /* Endless loop for NP < 5 !!!     */
          r4 = (int)(rnd_uni(&rnd_uni_init)*NP);
        }while((r4==i) || (r4==r1) || (r4==r2) || (r4==r3));

        do                        /* Pick a random population member */
        {                         /* Endless loop for NP < 6 !!!     */
          r5 = (int)(rnd_uni(&rnd_uni_init)*NP);
        }while((r5==i) || (r5==r1) || (r5==r2) || (r5==r3) || (r5==r4));

/*=======There are some simple rules which are worth following:========*/
/*==1)  F is usually between 0.5 and 1 (in rare cases > 1)========*/
/*==2)  CR is between 0 and 1 with 0., 0.3, 0.7 and 1. being worth to be tried first=*/
/*==3)  To start off NP = 10*D is a reasonable choice. Increase NP if misconvergence=*/
/*           happens.              */
/*==4)  If you increase NP, F usually has to be decreased============================*/
/*==5)  When the DE/best... schemes fail DE/rand... usually works and vice versa=====*/


/*=======EXPONENTIAL CROSSOVER=======================================================*/

/*-------DE/best/1/exp---------------------------------------------------------------*/
/*-------Our oldest strategy but still not bad. However, we have found several-------*/
/*-------optimization problems where misconvergence occurs.--------------------------*/
          if (strategy == 1)              /* strategy DE0 (not in our paper) */
        {
          assignd(D,tmp,(*pold)[i]);
          n = (int)(rnd_uni(&rnd_uni_init)*D);
          L = 0;
```

```
           do
           {
             tmp[n] = bestit[n] + F*((*pold)[r2][n]-(*pold)[r3][n]);
             n = (n+1)%D;
             L++;
           }while((rnd_uni(&rnd_uni_init) < CR) && (L < D));
         }

/*-------DE/rand/1/exp-------------------------------------------------------------*/
/*-------This is one of my favourite strategies. It works especially well when the----*/
/*-------"bestit[]"-schemes experience misconvergence. Try e.g. F=0.7 and CR=0.5------*/
/*-------as a first guess.----------------------------------------------------------*/

         else if (strategy == 2)          /* strategy DE1 in the techreport */
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           L = 0;
           do
           {
             tmp[n] = (*pold)[r1][n] + F*((*pold)[r2][n]-(*pold)[r3][n]);
             n = (n+1)%D;
             L++;
           }while((rnd_uni(&rnd_uni_init) < CR) && (L < D));
         }

/*-------DE/rand-to-best/1/exp-----------------------------------------------------*/
/*-------This strategy seems to be one of the best strategies. Try F=0.85 and CR=1.--*/
/*-------If you get misconvergence try to increase NP. If this doesn't help you------*/
/*-------should play around with all three control variables.------------------------*/

        else if (strategy == 3)        /* similiar to DE2 but generally better */
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           L = 0;
           do
           {
             tmp[n] = tmp[n] + F*(bestit[n] - tmp[n]) + F*((*pold)[r1][n]-(*pold)[r2][n]);
             n = (n+1)%D;
             L++;
           }while((rnd_uni(&rnd_uni_init) < CR) && (L < D));
         }

/*-------DE/best/2/exp is another powerful strategy worth trying---------------------*/

        else if (strategy == 4)
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           L = 0;
           do
           {
             tmp[n] = bestit[n] +
                      ((*pold)[r1][n]+(*pold)[r2][n]-(*pold)[r3][n]-(*pold)[r4][n])*F;
             n = (n+1)%D;
             L++;
           }while((rnd_uni(&rnd_uni_init) < CR) && (L < D));
         }

/*-------DE/rand/2/exp seems to be a robust optimizer for many functions-------------*/

        else if (strategy == 5)
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           L = 0;
           do
           {
             tmp[n] = (*pold)[r5][n] +
                      ((*pold)[r1][n]+(*pold)[r2][n]-(*pold)[r3][n]-(*pold)[r4][n])*F;
             n = (n+1)%D;
             L++;
           }while((rnd_uni(&rnd_uni_init) < CR) && (L < D));
         }


/*=======Essentially same strategies but BINOMIAL CROSSOVER=======================*/

/*-------DE/best/1/bin------------------------------------------------------------*/

        else if (strategy == 6)
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           for (L=0; L<D; L++) /* perform D binomial trials */
           {
             if ((rnd_uni(&rnd_uni_init) < CR) || L == (D-1)) /* change at least one
parameter */
             {
               tmp[n] = bestit[n] + F*((*pold)[r2][n]-(*pold)[r3][n]);
             }
             n = (n+1)%D;
           }
         }

/*-------DE/rand/1/bin------------------------------------------------------------*/

        else if (strategy == 7)
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           for (L=0; L<D; L++) /* perform D binomial trials */
           {
             if ((rnd_uni(&rnd_uni_init) < CR) || L == (D-1)) /* change at least one
parameter */
             {
               tmp[n] = (*pold)[r1][n] + F*((*pold)[r2][n]-(*pold)[r3][n]);
             }
             n = (n+1)%D;
           }
         }

/*-------DE/rand-to-best/1/bin----------------------------------------------------*/

        else if (strategy == 8)
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           for (L=0; L<D; L++) /* perform D binomial trials */
           {
             if ((rnd_uni(&rnd_uni_init) < CR) || L == (D-1)) /* change at least one
parameter */
             {
               tmp[n] = tmp[n] + F*(bestit[n] - tmp[n]) + F*((*pold)[r1][n]-
(*pold)[r2][n]);
             }
             n = (n+1)%D;
           }
         }

/*-------DE/best/2/bin-----------------------------------------------------------*/

        else if (strategy == 9)
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           for (L=0; L<D; L++) /* perform D binomial trials */
           {
             if ((rnd_uni(&rnd_uni_init) < CR) || L == (D-1)) /* change at least one
parameter */
             {
               tmp[n] = bestit[n] +
                      ((*pold)[r1][n]+(*pold)[r2][n]-(*pold)[r3][n]-(*pold)[r4][n])*F;
             }
             n = (n+1)%D;
            }
         }

/*-------DE/rand/2/bin-----------------------------------------------------------*/

        else
         {
           assignd(D,tmp,(*pold)[i]);
           n = (int)(rnd_uni(&rnd_uni_init)*D);
           for (L=0; L<D; L++) /* perform D binomial trials */
            {
             if ((rnd_uni(&rnd_uni_init) < CR) || L == (D-1)) /* change at least one
parameter */
             {
               tmp[n] = (*pold)[r5][n] +
                      ((*pold)[r1][n]+(*pold)[r2][n]-(*pold)[r3][n]-(*pold)[r4][n])*F;
             }
             n = (n+1)%D;
            }
         }


/* for discrete function optimization check that parameter values do not cross
   the specified boundary limits  */

for (L=0; L<D; L++)
{ if (tmp[L]<inibound_l)
     tmp[L] = inibound_l;
  if (tmp[L]>inibound_h)
     tmp[L] = inibound_h;
}

/*=======Trial mutation now in tmp[]. Test how good this choice really was.=======*/

         trial_cost = evaluate(D,tmp);         /* Evaluate new vector in tmp[] */

         if (trial_cost <= cost[i])    /* improved objective function value ? */
         {
            cost[i]=trial_cost;
            assignd(D,(*pnew)[i],tmp);
            if (trial_cost<cmin)               /* Was this a new minimum? */
            {                                 /* if so...*/
               cmin=trial_cost;               /* reset cmin to new low...*/
               imin=i;
               assignd(D,best,tmp);
            }
         }
         else
         {
            assignd(D,(*pnew)[i],(*pold)[i]); /* replace target with old value */
         }

      }                          /* End mutation loop through pop. */

      assignd(D,bestit,best);  /* Save best population member of current iteration */

      /* swap population arrays. New generation becomes old one */

      pswap = pold;
      pold  = pnew;
      pnew  = pswap;

/*----Compute the energy variance (just for monitoring purposes)-----------*/

      cmean = 0.;                              /* compute the mean value first */
      for (j=0; j<NP; j++)
      {
         cmean += cost[j];
      }
      cmean = cmean/NP;

      cvar = 0.;                             /* now the variance  */
      for (j=0; j<NP; j++)
      {
         cvar += (cost[j] - cmean)*(cost[j] - cmean);
      }
      cvar = cvar/(NP-1);

        dia   = ceil( best[0] * 12);
        pitch = ceil( best[1] *  2);
        tema  = ceil( best[2] *  4);
        nop   = ceil( best[3] *  5);
        len   = ceil( best[4] *  8);
        bafs  = ceil( best[5] *  6);
        bafc  = ceil( best[6] *  7);

/*----Output part----------------------------------------------------------*/

      if (gen%refresh==1)    display after every refresh generations
      {
      printf("\n\n\n Best-so-far cost funct. value=%-15.10g\n",cmin);

 printf("\n dia   = %d ",  dia);
 printf("\n pitch = %d ",pitch);
 printf("\n tema  = %d ", tema);
 printf("\n nop   = %d ",  nop);
 printf("\n len   = %d ",  len);
 printf("\n bafs  = %d ", bafs);
 printf("\n bafc  = %d ", bafc);

       printf("\n\n Generation=%d  NFEs=%ld   Strategy: %s
",gen,nfeval,strat[strategy]);
       printf("\n NP=%d    F=%-4.2g    CR=%-4.2g   cost-variance=%-10.5g\n",
               NP,F,CR,cvar);
      }

      fprintf(fpout_ptr,"%d  %ld   %-15.10g\n",gen,nfeval,cmin);

if (cmin<34.5)
break;


   }
/*=======================================================================*/
/*=========End of iteration loop========================================*/
/*=======================================================================*/

/*-------Final output in file-------------------------------------------*/


   fprintf(fpout_ptr,"\n\n\n Best-so-far obj. funct. value = %-15.10g\n",cmin);

fprintf(fpout_ptr,"\n dia   = %d ",  dia);
fprintf(fpout_ptr,"\n pitch = %d ",pitch);
fprintf(fpout_ptr,"\n tema  = %d ", tema);
fprintf(fpout_ptr,"\n nop   = %d ",  nop);
fprintf(fpout_ptr,"\n len   = %d ",  len);
fprintf(fpout_ptr,"\n bafs  = %d ", bafs);
fprintf(fpout_ptr,"\n bafc  = %d ", bafc);


fprintf(fpout_ptr,"\n\n Generation=%d  NFEs=%ld   Strategy: %s
",gen,nfeval,strat[strategy]);
   fprintf(fpout_ptr,"\n NP=%d    F=%-4.2g    CR=%-4.2g    cost-variance=%-10.5g\n",
           NP,F,CR,cvar);

   fclose(fpout_ptr);
   return(0);
}

/*-----------End of main()-----------------------------------------*/
```

C.3. Code for calculating heat transfer area

```
#define PI 3.1416

float evaluate(int D,float tmp[])

{

 int    i,j,Np,flag,var[7];
 extern long nfeval;
 float  h[30],tw,od,di,L,abso,dps,dpt;
 float  T1,T2,T,t1,t2,t,dT1,dT2,dTlg,dTm;
 float  Q,mk,mc,Ft,area,At,pt,k1,n1;
 float  Db,bc,Ds,Uo[300],denc,denk,Cpc,Kc,visc,Nt,Ntp,Ac,Acp,ut;
 float  Ret,Prt,ht,jht,viscw,vicor,jhs,As,lb,Lb,Cpk,Kk,visk;
 float  Res,us,Prs,hoc,Fn,Hb,lc;
 float  Ncv,Fw,Bb,Rad,Rw,Nw,Fb,Ab;
 float  Fl,Atb,ang,Asb,Al,a,beta,hs;
 float  temp,Kw,jfs,jft,Fdb,betad,Fdl,dpi,dpc;
 float  Nwv,Ra,Aw,uw,uz,dpw,dpe,Nb;

nfeval++;

var[0] = ceil( tmp[0]*12 );
var[1] = ceil( tmp[1]*2  );
var[2] = ceil( tmp[2]*4  );
var[3] = ceil( tmp[3]*5  );
var[4] = ceil( tmp[4]*8  );
var[5] = ceil( tmp[5]*6  );
var[6] = ceil( tmp[6]*7  );


for (i=0; i<D ; i++)
{
if ( var[i] ==0 )
     var[i] = 1;
}

T1=200;  T2=90;  T=145;
t1=40;   t2=78;  t=59;

Q=1509400;
mk=5.55;   mc=19.44;
denc=820; Cpc=2050; Kc=0.134; visc=0.0032;
denk=730; Cpk=2470; Kk=0.132; visk=0.00043;
jht=0.004; jhs=0.008; jfs=0.05;
Kw=45;

j=0;

Uo[j]=300;


switch(var[0]){

         case 1: od=0.25; di=0.206;
                break;
         case 2: od=0.375; di=0.319;
                break;
         case 3: od=0.5;   di=0.43;
                break;
         case 4: od=0.625; di=0.481;
                break;
```

```
          case 5: od=0.75;  di=0.606;
                 break;
          case 6: od=0.875; di=0.685;
                 break;
          case 7: od=1.0;  di=0.834;
                 break;
          case 8: od=1.25;  di=1.06;
                 break;
          case 9: od=1.5;   di=1.282;
                 break;
          case 10: od=1.75;  di=1.532;
                 break;
          case 11: od=2.0;  di=1.76;
                  break;
          case 12:  od=2.5; di=2.204;
                  break;


        }

switch(var[3]){
            case 1: Np=1;
                   break;
            case 2: Np=2;
                   break;
            case 3: Np=4;
                   break;
            case 4: Np=6;
                   break;
            case 5: Np=8;
                   break;


         }

switch(var[4]){

           case 1: L=6;
                  break;
           case 2: L=8;
                  break;
           case 3: L=10;
                  break;
           case 4: L=12;
                  break;
           case 5: L=16;
                  break;
           case 6: L=20;
                  break;
           case 7: L=22;
                  break;
           case 8: L=24;
                   break;

         }

switch(var[5]){

           case 1: lb=0.2;
                  break;
           case 2: lb=0.25;
                  break;
           case 3: lb=0.3;
                  break;
           case 4: lb=0.35;
                  break;
           case 5: lb=0.4;
                  break;
           case 6: lb=0.45;
                  break;
          }

switch(var[6]){
            case 1: lc=0.15;
                   break;
            case 2: lc=0.2;
                   break;
            case 3: lc=0.25;
                   break;
            case 4: lc=0.3;
                   break;
            case 5: lc=0.35;
                   break;
            case 6: lc=0.4;
                   break;
            case 7: lc=0.45;
                   break;
          }


switch(var[1]){
           case 1: switch(Np){

                        case 1:k1=0.319;n1=2.142;
                                break;
                        case 2:k1=0.249;n1=2.207;
                                break;
                        case 4:k1=0.175;n1=2.285;
                                break;
                        case 6:k1=0.0743;n1=2.499;
                                break;
                        case 8:k1=0.0365;n1=2.675;
                                break;
                        }
                break;

           case 2: switch(Np){

                        case 1:k1=0.215;n1=2.207;
                                break;
                        case 2:k1=0.156;n1=2.291;
                                break;
                        case 4:k1=0.158;n1=2.263;
                                break;
                        case 6:k1=0.0402;n1=2.617;
                                break;
                        case 8:k1=0.0331;n1=2.647;
                                break;
                        }
                break;
        }



dT1=T1-t2;
dT2=T2-t1;

dTlg = (dT1-dT2)/( log(dT1/dT2) );

if(Np==1)
  Ft=1.0;
else
  Ft=0.88;

dTm= dTlg*Ft;

od=od*0.0254;

di=di*0.0254;

Ac=PI*di*di/4;
```

```
pt=1.25*od;

L=L*12*0.0254;

At=PI*od*L;

/* iteration for assumed U */

do{


area=Q/(Uo[j]*dTm) ;

Nt=area/At;

Db = od * ( pow ( (Nt/k1),(1/n1) ) ) ;

switch(var[2]){
        case 1:bc=10*Db+8;
               break;
        case 2:bc=38;
               break;
        case 3:bc=(-2.232*Db*Db+31.12*Db+43.9);
               break;
        case 4:bc=(2.232*Db*Db+3.875*Db+88.1);
               break;
        }
if(bc<0)
bc=-bc;
Ds=Db+0.001*bc;

Lb=lb*Ds;

/* Tube side h.t.c */

Ntp=Nt/Np;

Acp=Ntp*Ac;

ut=mc/(denc*Acp);

Ret=denc*ut*di/visc;

Prt=visc*Cpc/Kc;

i=0;

ht=jht*Ret*( pow(Prt,0.333) )*Kc/di ; /* without viscosity correction */

h[i]=ht;

do
  {

     tw=(Uo[j]*(T-t)/h[i])+t;               /* finding wall temp.*/

 viscw=4.155*pow(10,-7)*tw*tw-0.00009903*tw+0.007596;

    i++;

h[i]=ht*pow((visc/viscw),0.14);          /* with viscosity correction*/

abso=h[i]-h[i-1];

if(abso<0)
abso=-abso;

  } while(abso>1 );

ht=h[i];
```

```
vicor=pow((visc/viscw),0.14);


/* shellside h.t.c by BELL'S method */

As=(pt-od)*Ds*Lb/pt;

Res=(mk*od)/(As*visk);

us=mk/(As*denk);

Prs=visk*Cpk/Kk;

/* ideal tube bank crossflow coeff.*/

hoc= jhs*Res*pow(Prs,0.333)*Kk/od;

/* tube row corr. factor*/

Hb=( Ds/2-Ds*(0.5-lc) );

if(var[1]==1)
Ncv=(Db-2*Hb)/0.87*pt;
else
Ncv=(Db-2*Hb)/pt;

if(Ncv>=35)
Ncv=35;

if(Res<=2000)
  Fn=1.0;
else
  Fn=4.44*pow(10,-6)*pow(Ncv,3)-0.0003857*pow(Ncv,2)+0.0121*Ncv+0.9086;


/* window corr. factor */

Bb=Hb/Db;

if(Bb>=0.4)
Bb=0.4;

Rad=-23.26*pow(Bb,3)+20.44*pow(Bb,2)-4.508*Bb+0.3976;
Rw=2*Rad;

if(Rw>=0.8)
Rw=0.8;

Nw=Nt*Rad;

Fw=4.068*pow(Rw,3)-5.521*pow(Rw,2)+1.338*Rw+1.061;

/* bypass corr. factor */

Ab=Lb*(Ds-Db);

if(Res<100)
 Fb=exp(-1.5*Ab/As);
else
 Fb=exp(-1.35*Ab/As);

/*leakage corr. factor*/

Atb=1.257*pow(10,-3)*od*(Nt-Nw);

ang=2* acos(1-2*lc);

Asb=2.4*pow(10,-3)*Ds*(2*PI-ang);

Al=Atb+Asb;
```

```
a=Al/As;

if(a>=0.6)
a=0.6;

beta=-15.68*pow(a,4)+20.4*pow(a,3)-8.813*a*a+1.913*a+0.02091;

Fl=1-(beta*(Atb+2*Asb)/Al );

if(Fl<=0)
Fl=0.1;

hs=hoc*Fn*Fw*Fb*Fl;

temp=(1/hs)+od*log(od/di) /(2*Kw) +(od/di) * (1/ht+0.00035)+0.0002;

j++;

Uo[j]= 1/temp;

area=Q/(Uo[j]*dTm);

abso=(Uo[j]-Uo[j-1]);

if(abso<0)
abso=-abso;

  } while (abso>1);


/* shellside pressure drop */

/* crossflow zone pressure drop */

if(Res<100)
Fdb=exp(-5*Ab/As);
else
Fdb=exp(-4*Ab/As);

betad=-1.126*a*a + 1.578*a + 0.08491;

Fdl=1-( betad*(Atb+2*Asb)/Al );
if(Fdl<=0)
Fdl=0.1;

dpi=8*jfs*Ncv*denk*us*us/2;

dpc=dpi*Fdb*Fdl;

/* winodw zone pressure drop */

if(var[1]==1)
Nwv=Hb/(0.87*pt);
else
Nwv=Hb/pt;

Ra=-23.26*pow(lc,3)+20.44*lc*lc-4.508*lc+0.3976;

Aw=(PI*Ds*Ds* Ra/4) - (Nw*PI*od*od/4);

flag=0;

if(Aw<0)
{  Aw=-Aw;
   flag=1;
}

uw=mk/(Aw*denk);

uz=pow( (uw*us),0.5 );
```

```
dpw=Fdl*(2+0.6*Nwv)*denk*uz*uz/2;

/*end zone pressure drop */

dpe= dpi*(Nwv+Ncv)*Fdb/Ncv;

/* total shell side presuure drop */

Nb=L/Lb-1;

dps=( 2*dpe+(Nb-1)*dpc+Nb*dpw)/1000;


/* Tubeside pressure drop */

if(Ret<1000)
 jft = 0.01;
else
 jft = -4.335*pow(10,-18)*pow(Ret,3)+4.836*pow(10,-12)*Ret*Ret-5.083*pow(10,-7)
       *Ret+0.009503;
if(jft<0)
jft=-jft;
 if(jft>0.009)
    jft=0.005;
dpt=( Np*( (8*jft*L)/(di*vicor) + 2.5 ) * denc*ut*ut/2 )/1000;


/* assign high value for area if pressure drops exceed */


if( (dpt>100) || (dps>100) || (flag==1) )

         {    area= pow(10,7);
             return(area);

        }
else
     return(area);


}
```