```
/*=================================================================*
 * Estimation of Heat Transfer Parameters in a Trickle Bed Reactor *
 * Using Differential Evolution and Orthogonal Collocation         *
 *                                                                 *
 * Authors: B. V. Babu & K. K. N. Sastry                           *
 *=================================================================*/


/* Standard Library Inclusion */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>
#include <time.h>
#include <ctype.h>
#include <string.h>
#include "d:\students\sastry\deoc\complex.h" /* Complex Arithmetic for polynomial root solving */

#define COLLOCPTS   6   /* Collocation Points */
#define XACC            1.0e-10
#define LOWERLIMIT      0.0
#define UPPERLIMIT      1.0
#define INTERVALS       100
#define TINY        1.0e-20
#define EPSS        1.0e-7
#define EPS     2.0e-5
#define MR      8
#define MT      10
#define MAXIT       80
#define MAXM        (2*(COLLOCPTS+2))
#define YES     1
#define NO      0
#define MAXSTP      10000
#define TINY1       1.0e-30
#define SAFETY      0.9
#define PGROW       -0.2
#define PSHRNK      -0.25
#define ERRCON      1.89e-4
#define MAXPOP      20
#define MAXDIM      2

/*------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 EPS1 1.2e-7
#define RNMX (1.0-EPS1)

static float maxarg1, maxarg2;
#define FMAX(a, b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ? (maxarg1) : (maxarg2))


/*-------------------------Globals-------------------------------*/

long  rnd_uni_init;
float c[MAXPOP][MAXDIM], d[MAXPOP][MAXDIM];
float (*pold)[MAXPOP][MAXDIM];
float (*pnew)[MAXPOP][MAXDIM];
float (*pswap)[MAXPOP][MAXDIM];
float A[COLLOCPTS+1][COLLOCPTS+1], B[COLLOCPTS+1][COLLOCPTS+1];
int kmax = 0, kount;
float *xp, **yp, dxsav;
float PecletNo, BiotNo;

/*---------Function declarations-------------------------------*/
```

```
void  assignd(int D, float a[], float b[]);
float rnd_uni(long *idum);
float evaluate(int D, float tmp[], long *nfeval);

void CollocMatrix     (void);
void LUDecompose      (float[COLLOCPTS+1][COLLOCPTS+1], int, int[], float*);
void LUBackSubstitute (float[COLLOCPTS+1][COLLOCPTS+1], int, int[], float[]);
void ZRoots           (fcomplex[], int, fcomplex[], int);
void ODEInt           (float[], int, float, float, float, float, float, int*, int*);
void nerror       (char[]);

float evaluate(int D, float tmp[], long *nfeval)
{
    int nok, nbad, j;
    float Tcal[COLLOCPTS], result = 0.0;
    float T0 = 400.0, Tw = 200.0;
    float Texp[COLLOCPTS] = {40.315, 35.646, 28.779, 21.542, 15.436, 11.320};

    (*nfeval)++;
    PecletNo = tmp[0];
    BiotNo = tmp[1]*10;
    for(j = 0; j < COLLOCPTS; j++)
    Tcal[j] = 200.0;
    ODEInt(Tcal, COLLOCPTS, LOWERLIMIT, UPPERLIMIT, 0.001, 0.0001, 0.0, &nok, &nbad);
    for(j = 0; j < COLLOCPTS; j++)
    result += pow(((Texp[j]-Tcal[j])/(T0-Tw)), 2.0);
    return result;
}

void dervis(float x, float y[], float dydx[])
{
    int j, k;

    for(j = 0; j < COLLOCPTS; j++)
    {
        dydx[j] = 0.0;
        for(k = 0; k < COLLOCPTS; k++)
            dydx[j] += PecletNo*(B[j][k] - (B[j][COLLOCPTS]*A[COLLOCPTS][k]/(BiotNo + A[COLLOCPTS
][COLLOCPTS])))*y[k];
    }
}

void CollocMatrix(void)
{
    int j, k, l, Index[COLLOCPTS + 1];
    float Q[COLLOCPTS+1][COLLOCPTS+1], Root[COLLOCPTS+1];
    float QInverse[COLLOCPTS+1][COLLOCPTS+1], d, Column[COLLOCPTS+1];

    void JacobiRoot(float[]);

    JacobiRoot(Root);
    for(k = 0; k < COLLOCPTS; k++)
        for(j = 0; j < (COLLOCPTS+1); j++)
            Q[k][j] = pow(Root[k], (float)(2*j));

    Root[COLLOCPTS] = 1.0;
    for(j = 0; j < (COLLOCPTS+1); j++)
        Q[COLLOCPTS][j] = 1;
    LUDecompose(Q, COLLOCPTS+1, Index, &d);
    for(j = 0; j < (COLLOCPTS+1); j++)
    {
        for(k  = 0; k < (COLLOCPTS+1); k++)
            Column[k] = 0.0;
        Column[j] = 1.0;
        LUBackSubstitute(Q, COLLOCPTS+1, Index, Column);
        for(k = 0; k < COLLOCPTS+1; k++)
            QInverse[k][j] = Column[k];
    }
    for(k = 0; k < COLLOCPTS+1; k++)
    {
        for(j = 0; j < COLLOCPTS+1; j++)
        {
            A[k][j] = B[k][j] = 0.0;
            for(l = 0; l < (COLLOCPTS+1); l++)
            {
                A[k][j] += (2*l)*pow(Root[k],(float)(2*l-1))*QInverse[l][j];
```

```
                B[k][j] += (4*l*l)*pow(Root[k],(float)(2*(l-1)))*QInverse[l][j];
            }
        }
    }
}

void JacobiRoot(float Root[])
{
    fcomplex AllRoots[2*COLLOCPTS+1];
    float RealCoeff[2*COLLOCPTS+1] = {1.0, 0.0, -48.0, 0.0, 540.0, 0.0, -2400.0, 0.0, 4950.0, 0.0
, -4752.0, 0.0, 1716.0};
    int j, k = 0;
    fcomplex Coefficients[2*COLLOCPTS+1];

    for(j = 0; j < (2*COLLOCPTS+1); j++)
        Coefficients[j] = Complex(RealCoeff[j], 0.0);
    ZRoots(Coefficients, (2*COLLOCPTS), AllRoots, NO);
    for(j = 1; j < (2*COLLOCPTS+1); j++)
    {
        if(fabs(AllRoots[j].i) <= 2*EPS*fabs(AllRoots[j].r))
        {
            if((AllRoots[j].r > 0.0)&&(AllRoots[j].r < 1.0))
            {
                Root[k] = AllRoots[j].r;
                k++;
            }
        }
    }
}

void ZRoots(fcomplex a[], int m, fcomplex roots[], int polish)
{
    void Laguerre(fcomplex[], int, fcomplex*, int*);
    int i, its, j, jj;
    fcomplex x, b, c, ad[MAXM];

    for(j = 0; j <= m; j++)
        ad[j]  = a[j];
    for(j = m; j >= 1; j--)
    {
        x = Complex(0.0, 0.0);
        Laguerre(ad, j, &x, &its);
        if(fabs(x.i) <= 2.0*EPS*fabs(x.r))
            x.i = 0.0;
        roots[j] = x;
        b = ad[j];
        for(jj = j-1; jj >= 0; jj--)
        {
            c = ad[jj];
            ad[jj] = b;
            b = Cadd(Cmul(x,b),c);
        }
    }
    if(polish)
        for(j = 1; j <= m; j++)
            Laguerre(a, m, &roots[j], &its);
    for(j = 2; j <= m; j++)
    {
        x = roots[j];
        for(i = j-1; i >= 1; i--)
        {
            if(roots[i].r <= x.r)
                break;
            roots[i+1] = roots[i];
        }
        roots[i+1] = x;
    }
}

void Laguerre(fcomplex a[], int m, fcomplex *x, int *its)
{
    int iter, j;
    float abx, abp, abm, err;
    fcomplex dx, x1, b, d, f, g, h, sq, gp, gm, g2;
    static float frac[MR+1] = {0.0, 0.5, 0.25, 0.75, 0.13, 0.38, 0.62, 0.88, 1.0};
```

```
    for(iter = 1; iter <= MAXIT; iter++)
    {
        *its = iter;
        b = a[m];
        err = Cabs(b);
        d = f = Complex(0.0, 0.0);
        abx = Cabs(*x);
        for(j = m-1; j >= 0; j--)
        {
            f = Cadd(Cmul(*x,f),d);
            d = Cadd(Cmul(*x,d),b);
            b = Cadd(Cmul(*x,b),a[j]);
            err = Cabs(b) + abx*err;
        }
        err *= EPSS;
        if(Cabs(b) <= err) return;
        g = Cdiv(d,b);
        g2 = Cmul(g,g);
        h = Csub(g2, RCmul(2.0, Cdiv(f,b)));
        sq = Csqrt(RCmul((float) (m-1),Csub(RCmul((float) m, h),g2)));
        gp = Cadd(g,sq);
        gm = Csub(g,sq);
        abp = Cabs(gp);
        abm = Cabs(gm);
        if(abp < abm)
            gp = gm;
        dx = ((FMAX(abp, abm) > 0.0 ? Cdiv(Complex((float) m, 0.0), gp) : RCmul(exp(log(1+abx)),
Complex(cos((float)iter), sin((float)iter)))));
        x1 = Csub(*x, dx);
        if(x -> r == x1.r && x -> i == x1.i) return;
        if(iter%MT)
            *x = x1;
        else
            *x = Csub(*x, RCmul(frac[iter/MT], dx));
    }
    nerror("too many iterations in Laguerre()");
    return;
}

void LUDecompose(float a[COLLOCPTS+1][COLLOCPTS+1], int n, int indx[], float *d)
{
    int l, imax, j, k;
    float big, dum, sum, temp;
    float vv[5];

    *d = 1.0;
    for(l = 0; l < n; l++)
    {
        big = 0.0;
        for(j = 0; j < n; j++)
            if((temp = fabs(a[l][j])) > big)
                big = temp;
        if(big == 0.0)
            nerror("Singular matrix in LUDecompose");
        vv[l] = 1.0/big;
    }
    for(j = 0; j < n; j++)
    {
        for(l = 0; l < j; l++)
        {
            sum = a[l][j];
            for(k = 0; k < l; k++)
                sum -= a[l][k]*a[k][j];
            a[l][j] = sum;
        }
        big = 0.0;
        for(l = j; l < n; l++)
        {
            sum = a[l][j];
            for(k = 0; k < j; k++)
                sum -= a[l][k]*a[k][j];
            a[l][j] = sum;
            if((dum = vv[l]*fabs(sum)) >= big)
            {
                big = dum;
                imax = l;
```

```
            }
        }
        if(j != imax)
        {
            for(k = 0; k < n; k++)
            {
                dum = a[imax][k];
                a[imax][k] = a[j][k];
                a[j][k] = dum;
            }
            *d = -(*d);
            vv[imax] = vv[j];
        }
        indx[j] = imax;
        if(a[j][j] == 0.0)
            a[j][j] = TINY;
        if(j != (n-1))
        {
            dum = 1.0/(a[j][j]);
            for(l = j+1; l < n; l++)
                a[l][j] *= dum;
        }
    }
}

void LUBackSubstitute(float a[COLLOCPTS+1][COLLOCPTS+1], int n, int indx[], float b[])
{
    int l, ii = -1, ip, j;
    float sum;

    for(l = 0; l < n; l++)
    {
        ip = indx[l];
        sum = b[ip];
        b[ip] = b[l];
        if(ii >= 0)
            for(j = ii; j <=  l-1; j++)
                sum -= a[l][j]*b[j];
        else if(sum)
            ii = l;
        b[l] = sum;
    }
    for(l = n-1; l >= 0; l--)
    {
        sum = b[l];
        for(j = l+1; j < n; j++)
            sum -= a[l][j]*b[j];
        b[l] = sum/a[l][l];
    }
}

void ODEInt(float ystart[], int nvar, float x1, float x2, float eps, float h1, float hmin, int *n
ok, int *nbad)
{
    int nstp, j;
    float xsav, x, hnext, hdid, h;
    float yscal[5], y[5], dydx[5];

    void rkqs(float[], float[], int, float*, float, float, float[], float*, float*);

    x = x1;
    h = (x2 > x1) ? fabs(h1) : -fabs(h1);
    *nok = (*nbad) = kount = 0;
    for(j = 0; j < nvar; j++)
        y[j] = ystart[j];
    if(kmax > 0)
        xsav = x - dxsav*2.0;
    for(nstp = 1; nstp <= MAXSTP; nstp++)
    {
        dervis(x, y, dydx);
        for(j = 0; j < nvar; j++)
            yscal[j] = fabs(y[j])+fabs(dydx[j]*h)+TINY1;
        if(kmax > 0 && kount < kmax-1 && fabs(x-xsav) > fabs(dxsav))
        {
            xp[++kount] = x;
            for(j = 0; j < nvar; j++)
```

```
                xsav = x;
        }
        if((x+h-x2)*(x+h-x1) > 0.0)
            h = x2 - x;
        rkqs(y, dydx, nvar, &x, h, eps, yscal, &hdid, &hnext);
        if(hdid == h)
            ++(*nok);
        else
            ++(*nbad);
        if((x - x2)*(x2 - x1) >= 0.0)
        {
            for(j = 0; j < nvar; j++)
                ystart[j] = y[j];
            if(kmax)
            {
                xp[++kount] = x;
                for(j = 0; j < nvar; j++)
                    yp[j][kount] = y[j];
            }
            return;
        }
        if(fabs(hnext) <= hmin)
            nerror("Step size too small in ODEInt()");
        h = hnext;
    }
    nerror("Too many steps in routine ODEInt()");
}

void rkqs(float y[], float dydx[], int n, float *x, float htry, float eps, float yscal[], float *
hdid, float *hnext)
{
    void rkck(float[], float[], int, float, float, float[], float[]);
    int j;
    float errmax, h, xnew, yerr[5], ytemp[5];

     h = htry;
     for(;;)
     {
        rkck(y, dydx, n, *x, h, ytemp, yerr);
        errmax = 0.0;
        for(j = 0; j < n; j++)
            errmax = FMAX(errmax, fabs(yerr[j]/yscal[j]));
        errmax /= eps;
        if(errmax > 1.0)
        {
            h = SAFETY*h*pow(errmax,PSHRNK);
            if(h < 0.1*h)
                h *= 0.1;
            xnew = (*x)+h;
            if(xnew == *x)
                nerror("Stepsize underflow in rkqs");
            continue;
        }
        else
        {
            if(errmax > ERRCON)
                *hnext = SAFETY*h*pow(errmax,PGROW);
            else
                *hnext = 5.0*h;
            *x += (*hdid = h);
            for(j = 0; j < n; j++)
                y[j] = ytemp[j];
            break;
        }
     }
}

void rkck(float y[], float dydx[], int n, float x, float h, float yout[], float yerr[])
{
    int j;
    static float a2 = 0.2, a3 = 0.3, a4 = 0.6, a5 = 1.0, a6 = 0.875;
    static float b21 = 0.2, b31 = 3.0/40.0, b32 = 9.0/40.0, b41 = 0.3;
    static float b42 = -0.9, b43 = 1.2, b51 = -11.0/54.0, b52 = 2.5;
    static float b53 = -70.0/27.0, b54 = 35.0/27.0, b61 = 1631.0/55296.0;
    static float b62 = 175.0/512.0, b63 = 575.0/13824.0, b64 = 44275.0/110592.0;
    static float b65 = 253.0/4096.0, c1 = 37.0/378.0;
```

```
    static float c3 = 250.0/621.0, c4 = 125.0/594.0, c6 = 512.0/1771.0;
    static float dc5 = 277.0/14336.0;
    float dc1 = c1 - 2825.0/27648.0, dc3 = c3 - 18575.0/48384.0;
    float dc4 = c4 - 13525.0/55296.0, dc6 = c6 - 0.25;
    float ak2[5], ak3[5], ak4[5], ak5[5], ak6[5], ytemp[5];

    for(j = 0; j < n; j++)
        ytemp[j] = y[j] + b21*h*dydx[j];
    dervis(x+a2*h, ytemp, ak2);
    for(j = 0; j < n; j++)
        ytemp[j] = y[j] + h*(b31*dydx[j]+b32*ak2[j]);
    dervis(x+a3*h, ytemp, ak3);
    for(j = 0; j < n; j++)
        ytemp[j] = y[j] + h*(b41*dydx[j]+b42*ak2[j]+b43*ak3[j]);
    dervis(x+a4*h, ytemp, ak4);
    for(j = 0; j < n; j++)
        ytemp[j] = y[j] + h*(b51*dydx[j]+b52*ak2[j]+b53*ak3[j]+b54*ak4[j]);
    dervis(x+a5*h, ytemp, ak5);
    for(j = 0; j < n; j++)
        ytemp[j] = y[j] + h*(b61*dydx[j]+b62*ak2[j]+b63*ak3[j]+b64*ak4[j]+b65*ak5[j]);
    dervis(x+a6*h, ytemp, ak6);
    for(j = 0; j < n; j++)
        yout[j] = y[j] + h*(c1*dydx[j]+c3*ak3[j]+c4*ak4[j]+c6*ak6[j]);
    for(j = 0; j < n; j++)
        yerr[j] = h*(dc1*dydx[j]+dc3*ak3[j]+dc4*ak4[j]+dc5*ak5[j]+dc6*ak6[j]);
}

void nerror(char error_text[])
{
    fprintf(stderr, "Orthogonal Collocation Run-Time Error...\n");
    fprintf(stderr, "%s\n", error_text);
    fprintf(stderr, "... Now Exiting to system...\n");
    exit(1);
}


/*---------Function definitions--------------------------------------*/

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;

}


void main(int argc, char *argv[])
{
    char  chr;              /* y/n choice variable                */
    int   k, 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   gen, genmax, seed;
    long  nfeval;           /* number of function evaluations     */
    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);
 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",&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);

  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 (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);
   }
 rnd_uni_init = -(long)seed;
 nfeval       =  0;
   CollocMatrix();
   for (k=0; k<NP; k++)
   {
      for (j=0; j<D; j++)
     c[k][j] = inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h - inibound_l);
      cost[k] = evaluate(D,c[k],&nfeval);
   }
   cmin = cost[0];
   imin = 0;
   for (k=1; k<NP; k++)
   {
      if (cost[k]<cmin)
      {
     cmin = cost[k];
     imin = k;
      }
   }
   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) */
   gen = 0;                          /* generation counter reset */
   while (gen < genmax)
   {
      gen++;
      imin = 0;
      for (k=0; k<NP; k++)         /* Start of loop through ensemble  */
      {
     do { r1 = (int)(rnd_uni(&rnd_uni_init)*NP); }while(r1==k);
     do { r2 = (int)(rnd_uni(&rnd_uni_init)*NP); }while((r2==k) || (r2==r1));
```

```
    assignd(D,tmp,(*pold)[k]);
    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;
        }
     trial_cost = evaluate(D,tmp,&nfeval);  /* Evaluate new vector in tmp[] */
     if (trial_cost <= cost[k])   /* improved objective function value ? */
     {
        cost[k]=trial_cost;
        assignd(D,(*pnew)[k],tmp);
        if (trial_cost<cmin)          /* Was this a new minimum? */
        {                               /* if so...*/
           cmin=trial_cost;           /* reset cmin to new low...*/
           imin=k;
           assignd(D,best,tmp);
        }
     }
     else
        assignd(D,(*pnew)[k],(*pold)[k]); /* replace target with old value */
      }   /* End mutation loop through pop. */
      assignd(D,bestit,best);  /* Save best population member of current iteration */
      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);

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

       if (gen%refresh==1)   /* display after every refresh generations */
       { /* ABORT works only if conio.h is accepted by your compiler */
     printf("\n\n\n Best-so-far cost funct. value=%-15.10g\n",cmin);

     for (j=0;j<D;j++)
       printf("\n best[%d]=%-15.10g",j,best[j]);
     printf("\n\n Generation=%d  NFEs=%ld ",gen,nfeval);
     printf("\n NP=%d    F=%-4.2g    CR=%-4.2g   cost-variance=%-10.5g\n",
               NP,F,CR,cvar);
       }

       fprintf(fpout_ptr,"%ld   %-15.10g\n",nfeval,cmin);
    }
/*-------Final output in file-----------------------------------------*/
    best[1] *= 10;
    fprintf(fpout_ptr,"\n\n\n Best-so-far obj. funct. value = %-15.10g\n",cmin);

    for (j=0;j<D;j++)
      fprintf(fpout_ptr,"\n best[%d]=%-15.10g",j,best[j]);
    fprintf(fpout_ptr,"\n\n Generation=%d  NFEs=%ld   Strategy: %s    ",gen,nfeval);
    fprintf(fpout_ptr,"\n NP=%d    F=%-4.2g    CR=%-4.2g    cost-variance=%-10.5g\n",
            NP,F,CR,cvar);
    fclose(fpout_ptr);
}
```