```
//
// Non-linear fitting program for CRIEPI Boron meter
//
// ver.1.0, 2004,3,4
// Hirokazu Kobayashi, CRIEPI, email: koba@criepi.denken.or.jp
//
//

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

char *malloc();
void free();
void lubksb();
void ludcmp();
float *vector();
float **matrix();
int *ivector();
void free_vector();
void free_ivector();
void free_matrix();
void nrerror();
float **inversion();
float **transform();
float **matrixdot();

int main (int argc, const char * argv[])
{
    int x_size=10;
    int max_halvings=10;
    int max_iter=50;
    float presse=-1.0;
    float tolerance=0.0001;
    int i,n=0,niter=0;
    int *indx;
    float sse;
    float *beta,*dbeta,*x,*y,**dy;
    float **z,**w,**t1,*col,**t,**t2,**t3;

    float  *linear_regression();


// main routine

    x=vector(1,x_size);
    y=vector(1,x_size);
    t1=matrix(1,3,1,3);
    dy=matrix(1,x_size,1,1);
    beta=vector(1,3);
    dbeta=vector(1,3);
    z=matrix(1,x_size,1,3);
    w=matrix(1,3,1,x_size);
    t2=matrix(1,3,1,x_size);
    t3=matrix(1,3,1,1);
    t=matrix(1,3,1,3);
    col=vector(1,3);
    indx=ivector(1,3);

//initial guess
    beta[1]=0.8;beta[2]=0.1;beta[3]=0.1;
//observed x,y data where x is time step
    for(i=1;i<=x_size;i++)x[i]=(float)i;
    y[1]=0.26;y[2]=0.31;y[3]=0.35;y[4]=0.40;y[5]=0.45;
```

```
   y[6]=0.49;y[7]=0.53;y[8]=0.56;y[9]=0.60;y[10]=0.62;

printf("Iteration      beta¥n");
for(;;)
{
//start of loop
   for(i=1;i<=x_size;i++)
      dy[i][1]=y[i]-(beta[1]*(1.0-exp(-beta[2]*x[i]))+beta[3]);
   sse=0.0;
   for(i=1;i<=x_size;i++)
      sse+=dy[i][1]*dy[i][1];
//long if block
   if(presse<0||sse<presse)
      {//error did not increase
      if(1.-sse/presse<=tolerance)
         {//error stabilized
         printf("Convergence obtained ¥n");
         printf("Final parameters beta= %5f %5f %5f¥n",beta[1],beta[2],beta[3]);
         printf("So, the bolon density is %5f¥n",beta[1]+beta[3]);
         return 0;
        }
      else
         {//error decreased
         presse=sse;
         dbeta=linear_regression(beta,x_size,dy);
         beta[1]+=dbeta[1];beta[2]+=dbeta[2];beta[3]+=dbeta[3];
        n=0;
         niter++;
         printf("    %5d ,beta= %5f %5f %5f¥n",niter,beta[1],beta[2],beta[3]);
         if(niter>=max_iter)
           {
            printf(" reached to ineration maximum¥n");
            return 1;
           }
        }
      }
   else
      {//error increased
      for(i=1;i<=3;i++)
         {dbeta[i]/=2.;
         beta[i]-=dbeta[i];}
      n++;
      if(n>max_halvings)
        {
         printf("Convergence not obtained after %5d halvings¥n",max_halvings);
         printf("Final parameters beta= %5f %5f %5f¥n",beta[1],beta[2],beta[3]);
         return 1;}
      else
         printf("Halving.... %5d ,beta= %5f %5f %5f¥n",n,beta[1],beta[2],beta[3]);
      }//end of if block
}//end of infinity loop

   return 0;
// end of main routine
/* output of this program should be,

Iteration      beta
       1 ,beta= 0.819051 0.068191 0.201943
       2 ,beta= 0.893859 0.064249 0.200478
       3 ,beta= 0.894444 0.064593 0.200398
       4 ,beta= 0.894576 0.064579 0.200404
Convergence obtained
Final parameters beta= 0.894576 0.064579 0.200404
```

```
So, the bolon density is 1.094980

*/

    float *linear_regression(beta,x_size,dy)
/*
    giving following parameters,
    x={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    beta={0.8, 0.1, 0.1}
    dy= {0.0838699, 0.0649846, 0.0426546, 0.036256, 0.0352245, 0.0290493, 0.0272682, ¥
    0.0194632, 0.0252557, 0.0143036}
    the function will return
    dbeta={0.0190519, -0.0318093, 0.101943}
*/
/*   test dy
    dy[1][1]=0.0838699;dy[2][1]=0.0649846;dy[3][1]=0.0426546;dy[4][1]=0.036256;dy[5][1]=0.0352245;
    dy[6][1]=0.0290493;dy[7][1]=0.0272682;dy[8][1]=0.0194632;dy[9][1]=0.0252557;dy[10][1]=0.0143036;
*/
    float *beta,**dy;
    int x_size;
  {
    for(i=1;i<=x_size;i++){
        z[i][1]=1.-exp(-beta[2]*x[i]);
        z[i][2]=beta[1]*exp(-beta[2]*x[i])*x[i];
        z[i][3]=1.0;
        }
        w=transform(z,x_size,3,w);
        //printf("pass 2¥n");
        t=matrixdot(w,3,x_size,3,z,t);
        //printf("pass 2¥n");
        // printf("%8f %8f %8f¥n",t[1][1],t[1][2],t[1][3]);
        t1=inversion(t,3,col,indx,t1);
        // printf("pass 3¥n");
        t2=matrixdot(t1,3,3,x_size,w,t2);
        //printf("pass 4¥n");
        t3=matrixdot(t2,3,x_size,1,dy,t3);
        //  printf("pass 5¥n");
        dbeta[1]=t3[1][1];
        dbeta[2]=t3[2][1];
        dbeta[3]=t3[3][1];
        return dbeta;
    } // end of function
} // end of main

float **inversion(a,n,col,indx,y)
// usage;
// square matrix a of n-row & collumn becomes inverted to
// square matrix y of n-row & collumn
// where vector col & indx each of n-length vector are used internaly.
/*  matrix
     {{1, 10, 3},
     {11, 12, 31},
     {3, 31, 10}}
    may be inverted to
    {{8.76041666, "0.0729166, -2.8541666},
     {0.17708, -0.010416, -0.020833},
     {-3.17708333, 0.0104166, 1.0208333}}
*/
// test of inversion
/*
    int n=3,*indx;
    a=matrix(1,n,1,n);
    y=matrix(1,n,1,n);
```

```
    col=vector(1,n);
    indx=ivector(1,n);

    a[1][1]=1.;a[1][2]=10.;a[1][3]=3.;
    a[2][1]=11.;a[2][2]=12.;a[2][3]=31.;
    a[3][1]=3.;a[3][2]=31.;a[3][3]=10.;

    y=inversion(a,n,d,col,indx);
    printf("%8f %8f %8f¥n",y[1][1],y[1][2],y[1][3]);
    printf("%8f %8f %8f¥n",y[2][1],y[2][2],y[2][3]);
    printf("%8f %8f %8f¥n",y[3][1],y[3][2],y[3][3]);


    free_matrix(a,1,n,1,n);
    free_matrix(y,1,n,1,n);
*/
float **a,**y,*col;
int n, *indx;
{
        int i,j;
        float d;
        ludcmp(a,n,indx,&d);
        for(j=1;j<=n;j++){
            for(i=1;i<=n;i++) col[i]=0.0;
            col[j]=1.0;
            lubksb(a,n,indx,col);
            for(i=1;i<=n;i++) y[i][j]=col[i];
           }
        return y;
}

float **matrixdot(a,n,l,m,b,y)
// usage;
// matrix a of n-row, l-collumn
// dot
// matrix b of l-row, m-collum
// makes matrix y of n-row, m-collumn.

/* test of matrix dot
    a=matrix(1,3,1,2);
    b=matrix(1,2,1,3);
    y=matrix(1,3,1,3);
    a[1][1]=1.;a[1][2]=10.;
    a[2][1]=11.;a[2][2]=12.;
    a[3][1]=3.;a[3][2]=31.;

    b[1][1]=2.;b[1][2]=3.;b[1][3]=4.;
    b[2][1]=5.;b[2][2]=6.;b[2][3]=7.;

    y=matrixdot(a,3,2,3,b,y);
    printf("¥n");
    printf("%8f %8f %8f ¥n",y[1][1],y[1][2],y[1][3]);
    printf("%8f %8f %8f ¥n",y[2][1],y[2][2],y[2][3]);
    printf("%8f %8f %8f ¥n",y[3][1],y[3][2],y[3][3]);
will produce
2.000000 63.000000 74.000000
82.000000 105.000000 128.000000
161.000000 195.000000 229.000000
*/

float **a,**b,**y;
int n,l,m;
{
```

```
    int i,j,k;
    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++){
        y[i][j]=0.0;
            for(k=1;k<=l;k++)
            y[i][j]+=a[i][k]*b[k][j];
        }
    return y;
}

float **transform(a,n,m,y)
float **a,**y;
int n,m;
{
    int i,j;

    for(j=1;j<=n;j++)
        for(i=1;i<=m;i++){
            y[i][j]=a[j][i];
        }
    return y;
}

void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...¥n");
        fprintf(stderr,"%s¥n",error_text);
        fprintf(stderr,"...now exiting to system...¥n");
        exit(1);
}


float *vector(nl,nh)
int nl,nh;
{
        float *v;

        v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}


float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
```

```
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
        free((char*) (v+nl));
}


void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}


#define TINY 1.0e-20;

void ludcmp(a,n,indx,d)
int n,*indx;
float **a,*d;
{
        int i,imax,j,k;
        float big,dum,sum,temp;
        float *vv,*vector();
        void nrerror(),free_vector();

        vv=vector(1,n);
        *d=1.0;
        for (i=1;i<=n;i++) {
                big=0.0;
                for (j=1;j<=n;j++)
                        if ((temp=fabs(a[i][j])) > big) big=temp;
                if (big == 0.0) nrerror("Singular matrix in routine LUDCMP");
                vv[i]=1.0/big;
        }
        for (j=1;j<=n;j++) {
                for (i=1;i<j;i++) {
                        sum=a[i][j];
                        for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
                        a[i][j]=sum;
                }
```

```
		big=0.0;
		for (i=j;i<=n;i++) {
			sum=a[i][j];
			for (k=1;k<j;k++)
				sum -= a[i][k]*a[k][j];
			a[i][j]=sum;
			if ( (dum=vv[i]*fabs(sum)) >= big) {
				big=dum;
				imax=i;
			}
		}
		if (j != imax) {
			for (k=1;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) {
			dum=1.0/(a[j][j]);
			for (i=j+1;i<=n;i++) a[i][j] *= dum;
		}
	}
	free_vector(vv,1,n);
}

#undef TINY


void lubksb(a,n,indx,b)
float **a,b[];
int n,*indx;
{
	int i,ii=0,ip,j;
	float sum;

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