= 1 = ラグランジュ補間公式による計算

ラグランジュ補間公式により離散データの

$$(0.5, 1.65)\,, (1.0, 2.72)\,, (2.0, 7.39)$$

の補間多項式を求め、補間点 $0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6$ における関数値を計算した結果およびプログラムソースを以下に示す。またグラフ 1 では、補間点を緑、関数値を赤で示した。

表 1: ラグランジュ補間公式による計算結果

x の値	計算結果
0.200000	1.412800
0.400000	1.537200
0.600000	1.796533
0.800000	2.190800
1.200000	3.384133
1.400000	4.183200
1.600000	5.117200

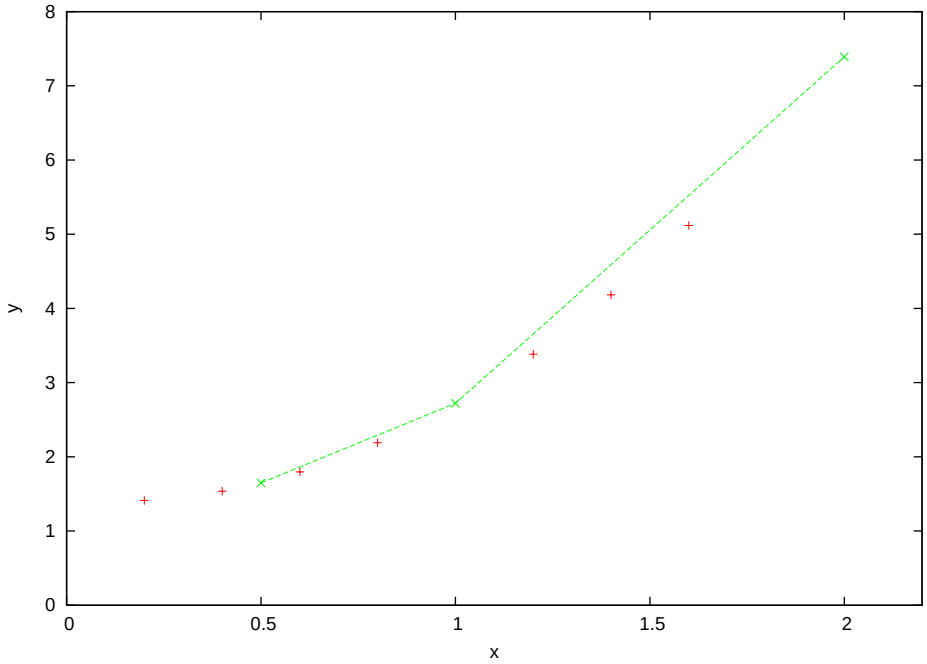

図 1: ラグランジュ補間による数値解析結果

```
/*      ラグランジュ補間公式による補間値の計算     */
#include <stdio.h>
#include <math.h>
#define N       2
```

```
#define N1     N+1

   double x[N1] = {0.5, 1.0, 2.0};
   double  hokan[N1] = {1.65, 2.72, 7.39 };
   double  xx[]={0.2,0.4,0.6,0.8,1.2,1.4,1.6,1.8};
   double lagrng(double xx);
main(void)
{
   int z;
   for(z=0;z<=6;z++)
   {   printf("%lf %lf\n",xx[z],lagrng(xx[z])); }
}
double lagrng(double xx)
{
   double p=0.0,pn,pd;
   int i,j;
   for(i=0; i<N1; i++)
   {
      pn=1.0; pd=1.0;
      for(j=0; j<N1; j++)
      {
          if( j!=i)
          {
            pn=pn*(xx-x[j]); pd = pd*(x[i]-x[j]);
          }
       }
       p=p+pn*hokan[i]/pd;
    }
    return(p);
}
```

= 2 = ニュートン法による非線形方程式の計算

ニュートン法を用いて、非線型方程式

$$f(x) = 2x^2 - 1$$

を求めた。真の解は、$\pm 1/\sqrt{2} = 0.7071\ldots$ であるが、初期値を 1.4 にして 5 回繰り返すと、0.7071 が得られた。プログラムソースを以下に示す。

```
/* ニュートン法を用いた数値解析 */
#include <stdio.h>
#include <math.h>
#define EPSILON 0.1E-5

    double function(double *f,double *df,double x);
    double newton(double *x1,int *no,int max);
main(void){
    int    num=0,max=0;
    double x1;

      printf("\n    初期値を入力\n");
      scanf("%lf",&x1);
      printf("    何回繰り返すか？  =  ");
      scanf("%d",&max);
      newton(&x1,&num,max);
      printf("\n    繰り返し回数: %d",num);
      printf("\n    方程式の解:  x= %9.5lf\n",x1);
      return(0);
}
double newton(double *x1,int *no,int max)
{
double f,df,x;
do{
    if(*no>=max)    break;
      x=*x1;
      function(&f,&df,x);
      *x1=x-f/df;
      (*no)++;
   }while(fabs(x-*x1)>EPSILON);
}
double function(double *f,double *df,double x)
{
    *f=2.0*x*x-1.0;
    *df=4.0*x;
}
```

= 3 = 複合台形公式による数値積分

複合台形公式を用いて、方程式

$$f(x) = x^3 + 2x^2 + 3x + 4$$

を、$0 \leq x \leq 3$ において積分した。表 2 には、分割数を変えた結果を示した。このプログラムソースを以下に示す。

表 2: 複合台形公式による数値積分

反復回数	10	20	30	40	50	100	1000	真の解
計算結果	64.042500	63.823125	63.782500	63.768281	63.761700	63.752925	63.750029	63.75

```
/*     複合台形公式による数値積分 */

#include <stdio.h>
#include <math.h>
#define TRUE   1
#define FALSE  2

main(void)
{
   double function(double x);
   double daikei(double a,double b,int n);
      double a=0.0;
      double b=3.0;
   int n[]={10,20,30,40,50,100,1000};
   int z;
   /* printf("積分範囲の始端と終端を入力 \n");
   scanf("%lf %lf",&a,&b);
   printf("区間の分割数を入力 \n");
   scanf("%d",&n);
   */
   for(z=0;z<=6;z++){
   printf("%d %f\n",n[z],daikei(a,b,n[z]));
   }
}
double function(double x)
{
    double w;
    w=x*x*x +2.0*x*x +3.0*x +4.0;
    return(w);
```

```
}

double daikei(double a,double b,int n)
{
    int j;
    double h,s;

    h=(b-a)/n;
    s=function(a);
    for(j=1; j<=n-1; j++) s=s+2.0*function(a+j*h);
    s=s+function(b);
    return(s*h/2.0);
}
```

= 4 = 常微分方程式の数値解法

常微分方程式 $y' = x + y$ を初期値 $x = 0, y = 0$ から $x = 2$ までについてオイラー、ホイン、ルンゲクッタ法を用いて解析した。結果を表 3 に示す。

表 3: 常微分方程式法の計算結果

x	Y(euler)	Y(heun)	Y(runge)	Y(真の解)
0.000	0.0000	0.0000	0.0000	0.0000
0.400	0.0000	0.0800	0.0784	0.0918
0.800	0.1600	0.3904	0.3920	0.4255
1.200	0.5440	1.0418	1.0566	1.1201
1.600	1.2416	2.1979	2.2447	2.3530
2.000	2.3782	4.1008	4.2136	4.3891

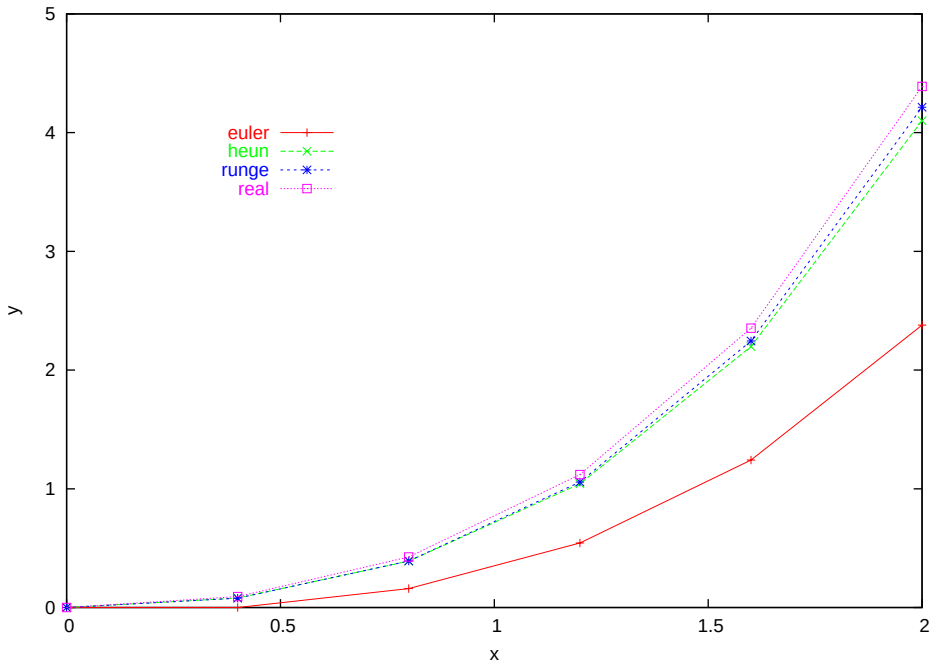

図 2: x の分割数 5 のときの結果

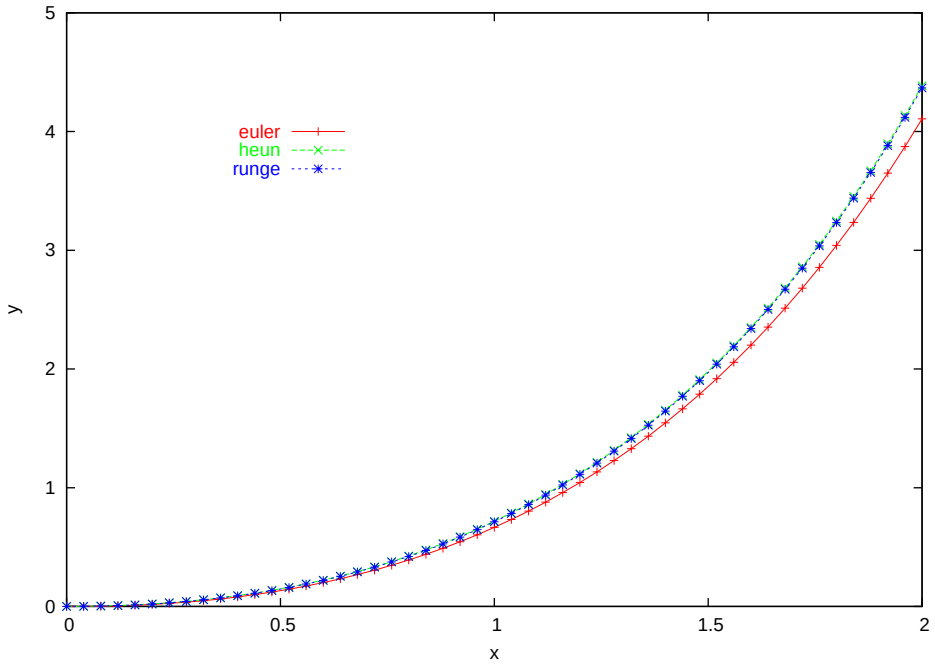

図 3: x の分割数 50 のときの結果

```
  /*常微分微分方程式の数値計算*/
#include <stdio.h>
#include <math.h>

double function(double x,double y);
double euler(double h,int n,double x0,double y1[]);
double heuns(double h,int n,double x0,double y2[]);
double runge(double h,int n,double x0,double y3[]);

int main(void)
{
   int n,i;
   double x0,yo;
   double y1[55],y2[55],y3[55];
   double h,xn;
   printf("\n x 座標の初期値 x0=");
   scanf("%lf",&x0);
   printf("y 座標の初期値 x0=");
   scanf("%lf",&yo);
   printf("\n x 座標の上限値 xn=");
   scanf("%lf",&xn);
   printf("\n 分割数は？n=");
   scanf("%d",&n);
   h=(xn-x0)/n;   y1[0]=y2[0]=y3[0]=yo;
```

```
   euler(h,n,x0,y1);
   heuns(h,n,x0,y2);
   runge(h,n,x0,y3);
   printf("\n    -E-          -H-           -R-\n");
   for(i=0;i<=n;i++)
   {
        printf("x=%6.3lf    %8.4lf",x0+h*i,y1[i]);
        printf("   %8.4lf    %8.4lf\n",y2[i],y3[i]);
   }
   return(0);
}

double function(double x,double y)
{
       return(x+y);
}

double euler(double h,int n,double x0,double y1[])
{

     int   i;

     for(i=0; i<=n; i++)
       {
        y1[i+1]=y1[i]+h*function(x0+h*i,y1[i]);
       }
}

double heuns(double h,int n,double x0,double y2[])
{
     int i;
     double k1,k2;

     for(i=0;  i<=n; i++){
     k1=h*function(x0+h*i,y2[i]);
     k2=h*function(x0+h*(i+1),y2[i]+k1);
     y2[i+1]=y2[i]+(k1+k2)/2.0;
     }
}

double runge(double h,int n,double x0,double y3[])
{
    int   i;
    double   xi,k1,k2,k3,k4;
```

```
    for(i=0; i<=n; i++)
      {
         xi=x0+h*i;
         k1=h*function(xi,y3[i]);
         k2=h*function(xi+h/2.0,y3[i]+k1/2.0);
         k3=h*function(xi+h/2.0,y3[i]+k2/2.0);
         k4=h*function(xi+h/2.0,y3[i]+k3);
         y3[i+1]=y3[i]+(k1+2.0*k2+2.0*k3+k4)/6.0;
      }
}
```

= 5 = ロトカ-ボルテラ式の数値解析

ロトカボルテラの式

$$\begin{cases} x' = x(-2+y) \\ y' = y(3-4x) \end{cases}$$

初期条件 $(x_0 = 4, y_0 = 2)$ で $t : 0 \sim 200$ についてルンゲクッタ法を用いて計算した。図 4 に結果を示す。

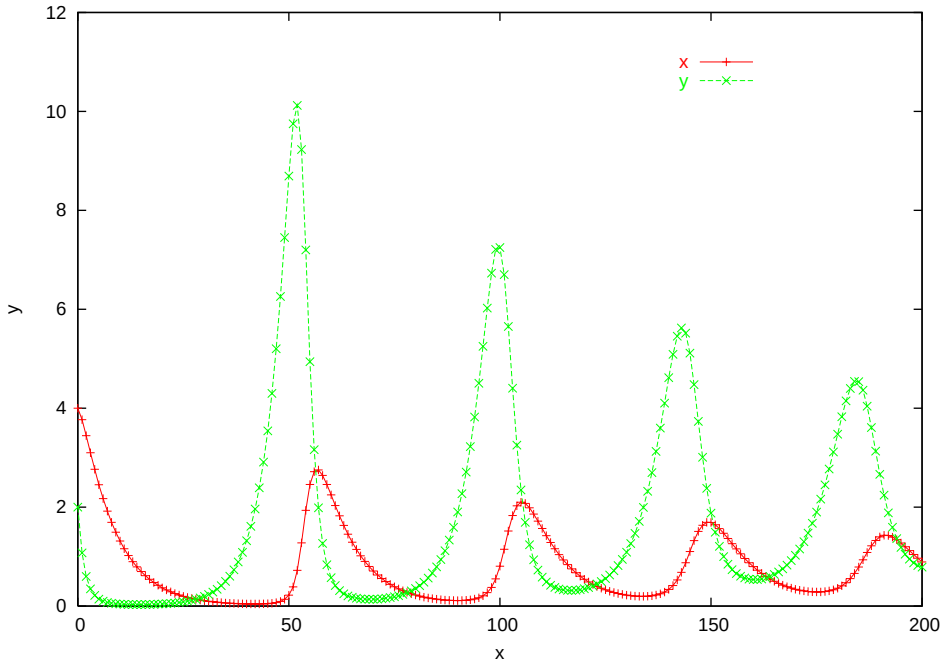

図 4: ロトカボルテラ式の数値解析結果

```
/* ロトカボルテラ式の数値計算*/

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

  double function1(double x1,double y1);
  double function2(double x2,double y2);
  double rotoka(double h,int n,double t,double x4[],double y4[]);

int main(void)
{

  int i,n;
  double x0,yo,to;
  double x4[110],y4[110];
  double h,tn;
```

```
  printf("\n x の初期値は ");
  scanf("%lf",&x0);
  printf("\n y の初期値は ");
  scanf("%lf",&yo);
  printf("\n t の初期値は ");
  scanf("%lf",&to);
  printf("\n t の境界値は ");
  scanf("%lf",&tn);
  printf("\n 分割数は ");
  scanf("%d",&n);

  h=(tn-to)/n;
  x4[0]=x0;
  y4[0]=yo;

 rotoka(h,n,to,x4,y4);
  printf("\n      t       x      y\n");
   for(i=0;i<=n; i++)
     {
          printf("%6.3lf %8.4lf %8.4lf\n",to+h*i,x4[i],y4[i]);
     }return(0);
}

double function1(double x1,double y1)
{
   return(x1*(-0.2+0.1*y1));
}
double function2(double x2,double y2)
{
   return(y2*(0.3-0.4*x2));
}
double rotoka(double h,int n,double to,double x4[],double y4[])
{
    int i;
    double ti,k1,k2,k3,k4,m1,m2,m3,m4;

    for(i=0; i<=n; i++)
      {
          ti=to+h*i;
          k1=h*function1(x4[i],y4[i]);
          m1=h*function2(x4[i],y4[i]);
          k2=h*function1(x4[i]+k1/2,y4[i]+m1/2);
          m2=h*function2(x4[i]+k1/2,y4[i]+m1/2);
          k3=h*function1(x4[i]+k2/2,y4[i]+m2/2);
```

```
            m3=h*function2(x4[i]+k2/2,y4[i]+m2/2);
            k4=h*function1(x4[i]+k3,y4[i]+m3);
            m4=h*function2(x4[i]+k3,y4[i]+m3);

            y4[i+1]=y4[i]+(m1+m2+m3+m4)/6.0;
            x4[i+1]=x4[i]+(k1+k2+k3+k4)/6.0;
        }
}
```

= 6 = 熱伝導方程式の数値解析

放物型の偏微分方程式

$$\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2} \qquad (c^2 = 1/12)$$

の解を、初期条件 $u(x,0) = x(1-x), \quad (0 \le x \le 1)$, 境界条件 $u(0,t) = 0, u(1,t) = 0$ のもとで差分法により求めた。結果を表 5 に示す。

表 4: 熱伝導方程式の計算結果

縦 t 横 x	0.0	0.25	0.50	0.75	1.0
0.0000	0.000000	0.187500	0.250000	0.187500	0.000000
0.3750	0.000000	0.125000	0.187500	0.125000	0.000000
0.7500	0.000000	0.93750	0.125000	0.093750	0.000000
1.1250	0.000000	0.062500	0.0.093750	0.062500	0.000000
1.5000	0.000000	0.046875	0.062500	0.046875	0.000000
1.8750	0.000000	0.031250	0.046875	0.031250	0.000000

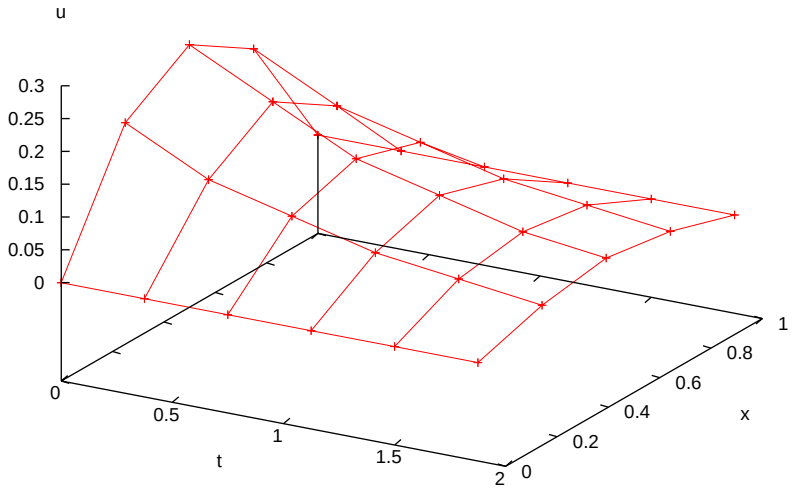

図 5: 放物型偏微分方程式の数値解析結果

```
/*  放物型偏微分方程式（熱伝導方程式）の解 */

#include <stdio.h>
#include <math.h>
#define NMAX   11
#define NNMAX  21
```

```
#define TRUE    1
#define FALSE   0

   double  intcon(double h,int n);
   double  solute(int n,double fa,double fb,int nn,double lamda);
   double  f[NMAX],u[NMAX][NNMAX];
main()
{
   int n,nn,i,j;
   double c2=1.0/12.0;
   double a,b,fa,fb,h,lamda,k;
   printf("\n\n x 座標の始端と終端を入力 \n");
   scanf("%lf %lf",&a,&b);
   printf("始端と終端での境界条件を入力\n");
   scanf("%lf %lf",&fa,&fb);
   printf("x 座標の分割数は？(<%2d)\n",NMAX-1);
   scanf("%d",&n);
   printf("t 座標方向のステップ数は (<%2d)\n",NNMAX-1);
   scanf("%d",&nn);
   printf("ラムダの値を入力 \n");
   scanf("%lf",&lamda);

    h=(b-a)/n;
    k=lamda*h*h/c2;
   intcon(h,n);
   solute(n,fa,fb,nn,lamda);

   for(j=0; j<=nn; j++)
     {
          printf("%8.4f : ",j*k);
          for(i=0; i<=n; i++)
             printf("%10.6f",u[i][j]);
             printf("\n");
      }
}
double intcon(double h,int n)
{
   int i;
   for(i=0; i<=n; i++)
      {

          f[i]=i*h*(1.0-i*h);
      }
}
```

```
double solute(int n,double fa,double fb,int nn,double lamda)
{
   int i,j;
   for(i=0; i<=n; i++)
     {  u[i][0]=f[i];
     }
   for(j=0; j<nn; j++)
     {
       u[0][j+1]=fa; u[n][j+1]=fb;
       for(i=1; i<n; i++)
        u[i][j+1]=lamda*u[i-1][j]+(1.0-2.0*lamda)*u[i][j]+lamda*u[i+1][j];
     }
}
```

= 7 = 楕円型方程式の数値解析

楕円型の偏微分方程式 (ディリクレ問題)

$$\Delta u = 0 \qquad \left(\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)$$

の解を、境界条件 $(x, y); 0 \leq x \leq 1, 0 \leq y \leq 1$ の下辺 $y = 0$ で x^2、上辺 $y = 1$ で x、左端 $= 0$ と右端 $= 1$ ではそれぞれ一定値をとるものとした。表 6 に結果を示す。

表 5: 楕円方程式の計算結果

縦 x 横 y	0.0	0.2	0.4	0.6	0.8	1.0
0.0	0.000000	0.040000	0.160000	0.360000	0.640000	1.000000
0.2	0.000000	0.070611	0.204239	0.414050	0.688324	1.000000
0.4	0.000000	0.092276	0.242472	0.457160	0.719286	1.000000
0.6	0.000000	0.124097	0.292183	0.505044	0.748010	1.000000
0.8	0.000000	0.163348	0.348676	0.555357	0.775842	1.000000
1.0	0.000000	0.200000	0.400000	0.600000	0.800000	1.000000

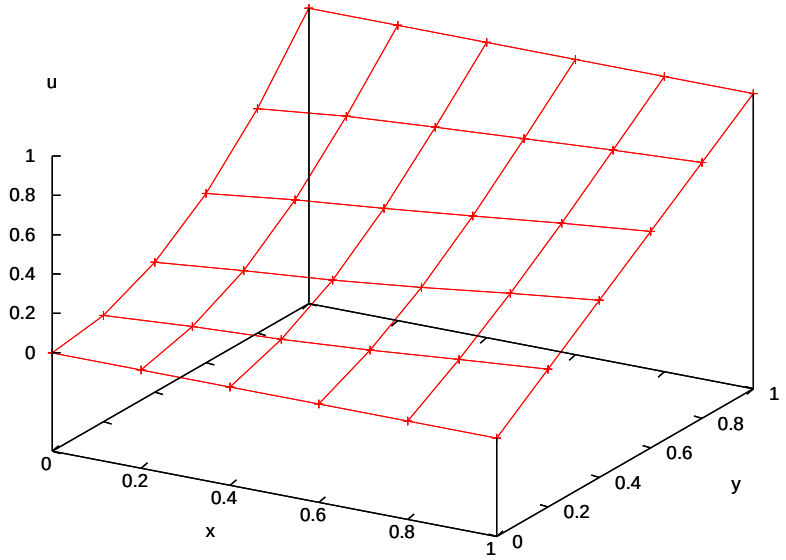

図 6: 楕円型偏微分方程式の数値解析結果

```
/*  楕円型偏微分方程式の解 */

#include <stdio.h>
#include <math.h>
#define NMAX   11
```

```
#define NNMAX  21
#define EPS    0.0005
#define TRUE    1
#define FALSE   0

   double  intcon(int nx, double h,double k);
   double  solute(int nx,int ny, double aa,double bb,double h,double k);
   double  f1[NMAX],f2[NMAX],u[NMAX][NNMAX];
main()
{
   int nx,ny,i,j;
   double aa,bb,h,k;
   printf("\n\n 左端, 右端の境界値 \n");
   scanf("%lf %lf",&aa,&bb);
   printf("x,y の分割数\n");
   scanf("%d %d",&nx,&ny);

    h=1.0/nx;
    k=1.0/ny;
   intcon(nx,h,k);
   solute(nx,ny,aa,bb,h,k);

   for(j=0; j<=ny; j++)
     {
          printf("%8.4f : ",j*k);
          for(i=0; i<=nx; i++){
             printf("%10.6f",u[i][j]);printf("     ");
           }  printf("\n");
      }
}
double intcon(int nx,double h,double k)
{
   int i;
   double x1,x2;
   for(i=0; i<=nx; i++)
      {
          x1=i*h;
          x2=i*k;
          f1[i]=x1*x1;
          f2[i]=x2;
      }
}

double solute(int nx,int ny, double aa,double bb,double h,double k)
```

```
{
   int i,j;
   double ubefore,uafter,uzettaiue,uzettaiue2,uzettaisita,utotal;
   uzettaisita=0.0;
   uzettaiue2=0.0;
   utotal=1.0;

   for(i=0;i<=nx; i++){
     for(j=0;j<=ny; j++){
       u[i][j]=0.0; } }
   while(utotal>EPS){
   for(i=0; i<=nx; i++)
     {  u[i][0]=f1[i];
        u[i][ny]=f2[i];
     }
   for(j=1; j<ny; j++)
     {
       u[0][j]=aa; u[nx][j]=bb;
       for(i=1; i<nx; i++){
        ubefore=u[i][j];
        u[i][j]=1.0/(2.0*(h*h+k*k))*(k*k*(u[i+1][j]+u[i-1][j])+h*h*(u[i][j+1]+u[i][j-1]));
        uafter=u[i][j];
        uzettaisita=uzettaisita+uafter;
if(ubefore<=uafter){
        uzettaiue=uafter-ubefore;
}else{
  uzettaiue=(uafter-ubefore)*(-1.0);}
        uzettaiue2=uzettaiue2+uzettaiue;
        }utotal=uzettaiue/uzettaisita;
     }}printf("%10.6f", utotal);
}
```