# Shortley-Weller 近似による Poisson 方程式のシミュレーション

明治大学総合数理学部現象数理学科
船見　俊哉

2018 年 2 月 15 日

# 目 次

# 第1章　はじめに

## 1.1　Shortley-Weller 近似とは

Shortley-Weller 近似 (以下 S-W 近似と呼ぶ) は、G. H. Shortley と R. Weller が考えだした差分法の一種である [1]。あとで定義する「不等間隔格子点」を用いて行う差分法で、通常の差分法に比べていくらか複雑になっている。その一方、局所誤差が $O(h)$ であっても差分解の誤差は $O(h^2)$ で、通常の差分法と同じになるという利点がある [2][3]。

## 1.2　研究の狙い

この研究では、S-W 近似による Poisson 方程式の数値計算プログラムの作成を目標とした。先行研究では S-W 近似による熱方程式や波動方程式の陽解法プログラムは存在したが、陰解法プログラムは見つからなかった。先程述べたような利点がありながらも研究が進展しないのは、あくまでも想像だが、プログラムの作り方がわかりにくいせいかもしれない。ひとまずプログラムを作ることで、次の研究の一助になれば良いと考えている。

またプログラムが完成すれば、山本 [2] の主張の確認ができるだろう。さらに熱方程式や波動方程式の陰解法プログラムによる数値実験も行うことができると考えている。

# 第2章　S-W近似

## 2.1　言葉の定義

**定義 2.1**
2次元領域を縦横に平行に区切る線分を**格子軸**と呼び、縦の格子軸と横の格子軸の交点を**格子点**と呼ぶ。特に格子軸同士の幅が等間隔なものを**等間隔格子**と呼び、同様に**等間隔格子点**と呼ぶ。

## 2.2　差分格子

$R^2$ の有界領域 $\Omega$ 上で問題を解くことを考える。このとき、$\Omega$ の閉包を $\overline{\Omega}$ とし、$\Omega$ を含む長方形領域領域 $D$ を以下で定義する。

**定義 2.2**

$$D := (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max}) \supset \Omega$$

このようにして定義した $D$ を等間隔格子で分割する。$N_x, N_y$ をそれぞれ $x, y$ 方向の $D$ の分割数、$h_x, h_y$ をそれぞれ $x, y$ 方向の $D$ の分割幅とするとき、

$$h_x = \frac{x_{\max} - x_{\min}}{N_x},\ h_y = \frac{y_{\max} - y_{\min}}{N_y}$$

が成り立つ。さらに $x_i, y_j$ を

$$x_i := x_{\min} + i \times h_x \quad (0 \leq i \leq N_x, i \in \mathbb{N})$$
$$y_j := y_{\min} + j \times h_y \quad (0 \leq j \leq N_y, j \in \mathbb{N})$$

と定義すると、$(x_i, y_j)$ は等間隔格子点の座標であり、これを $(i, j)$ **の格子点**と呼ぶことにする。

## 2.3　不等間隔格子点

**定義 2.3**
$\Omega$ の境界 $\partial\Omega$ と格子軸との交点を**不等間隔格子点**という。

$\Omega$ 内の等間隔格子点 $P = (x_i, y_j)$ に対して、その左右上下の等間隔格子点を $P_W, P_E, P_N, P_S$ と置く (West, East, North, South のつもり)。このとき、$\partial\Omega$ 付近の点 $P$ に関して、$P_W, P_E, P_N, P_S$ のいずれかまたは複数が $\overline{\Omega}$ に含まれない点 $P$ が存在する。このような場合、図 2.1 のように $\partial\Omega$ 上に不等間隔格子点を取り、$P_W, P_E, P_N, P_S$ と置き換える。

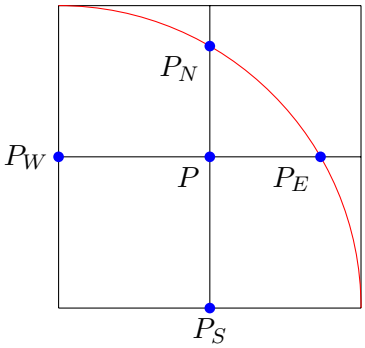


図 2.1: 不等間隔格子点の取り方

## 2.4 等間隔格子点の領域内外の判断

2.3 節で説明した手法を達成するためには、等間隔格子点が $\Omega$ の内部、外部、または境界上のいずれに属するかを判断する必要がある。特に $(i,j)$ の格子点に対して、次のような関数 $M$ を定義する。

**定義 2.4**

$$M(i,j) = \begin{cases} 1 & ((x_i, y_j) \in \Omega) \\ 0 & ((x_i, y_j) \in \partial\Omega) \\ -1 & ((x_i, y_j) \in D \backslash \overline{\Omega}) \end{cases}$$

## 2.5 不等間隔格子点の座標と格子点間の距離

不等間隔格子点における未知関数の値は境界条件を使うことにする。境界条件を $g(x,y)$ とすると、不等間隔格子点の座標が必要になる。

**定義 2.5**

- $E(x_i, y_j)$ は、$(E(x_i, y_j), y_j)$ が $(x_i, y_j)$ の右側にある最も近い不等間隔格子点の $x$ 座標である。
- $W(x_i, y_j)$ は、$(W(x_i, y_j), y_j)$ が $(x_i, y_j)$ の左側にある最も近い不等間隔格子点の $x$ 座標である。
- $N(x_i, y_j)$ は、$(x_i, N(x_i, y_j))$ が $(x_i, y_j)$ の上側にある最も近い不等間隔格子点の $y$ 座標である。
- $S(x_i, y_j)$ は、$(x_i, S(x_i, y_j))$ が $(x_i, y_j)$ の下側にある最も近い不等間隔格子点の $y$ 座標である。

以上を用いて、$P = (x_i, y_j)$ と周りの 4 点との距離 $h_E, h_W, h_N, h_S$ を求める。

$$h_E = \begin{cases} h_x & (P_E \in \overline{\Omega}) \\ |E(x_i, y_j) - x_i| & (\text{else}) \end{cases}, \quad h_W = \begin{cases} h_x & (P_W \in \overline{\Omega}) \\ |W(x_i, y_j) - x_i| & (\text{else}) \end{cases}$$

$$h_N = \begin{cases} h_y & (P_N \in \overline{\Omega}) \\ |N(x_i, y_j) - y_j| & (\text{else}) \end{cases}, \quad h_S = \begin{cases} h_y & (P_S \in \overline{\Omega}) \\ |S(x_i, y_j) - y_j| & (\text{else}) \end{cases}$$

## 2.6 Laplacian の S-W 近似

例として Laplacian： $\Delta u(P) = u_{xx}(P) + u_{yy}(P)$ に対して S-W 近似を適用すると、以下のようになる。

$$\Delta u(P) \simeq \frac{2}{h_E + h_W}\left(\frac{u(P_E) - u(P)}{h_E} - \frac{u(P) - u(P_W)}{h_W}\right) + \frac{2}{h_N + h_S}\left(\frac{u(P_N) - u(P)}{h_N} - \frac{u(P) - u(P_S)}{h_S}\right)$$

# 第3章　円盤領域における Poisson 方程式の S-W 近似

## 3.1　領域と方程式

$$\Omega = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 < 1\}$$

とし、

$$\begin{cases} -\Delta u(x, y) = f(x, y) & ((x, y) \in \Omega) \\ u(x, y) = g(x, y) & ((x, y) \in \partial\Omega) \end{cases} \tag{3.1}$$

を考える。

2.4 節で定義した $M$ は、この問題では次のようになる。

$$M(i, j) = \begin{cases} 1 & (x_i^2 + y_j^2 < 1) \\ 0 & (x_i^2 + y_j^2 = 1) \\ -1 & (x_i^2 + y_j^2 > 1) \end{cases}$$

数学的にはこれで良いが、丸め誤差のため、数値計算において浮動小数点数の完全な一致を判定するプログラムは書くことができない。そこで $\partial\Omega$ に $\varepsilon$ 程度の幅を持たせて、$\partial\Omega$ 上ではないが極めて近い等間隔格子点は $\partial\Omega$ 上にあるという判断をする。例えば、図 3.1 では、点 $P$ は、半径 1 の円周 $= \partial\Omega$(赤線) 上にはない。しかし、半径 $1 \pm \varepsilon$ の円周=青線の間に存在しているため、このようなときは $\partial\Omega$ 上にあるという判断をする。double 型の丸め誤差は $10^{-14}$ ほどであるため、特に $\varepsilon = 10^{-13}$ とし、先程の $M$ を次のように書き換える。

$$M(i, j) = \begin{cases} 1 & (1 - (x_i^2 + y_j^2) > \varepsilon) \\ 0 & (|1 - (x_i^2 + y_j^2)| \leq \varepsilon) \\ -1 & (1 - (x_i^2 + y_j^2) < -\varepsilon) \end{cases}$$

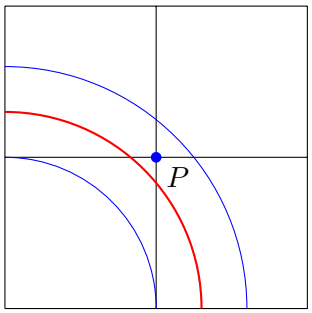


図 3.1: 境界線の幅

また、2.5 節で定義した $E, W, N, S$ は、円盤領域において次のようになる。

$$E(x_i, y_j) = \sqrt{1 - y_j^2},\ W(x_i, y_j) = -\sqrt{1 - y_j^2},$$
$$N(x_i, y_j) = \sqrt{1 - x_i^2},\ S(x_i, y_j) = -\sqrt{1 - x_i^2}$$

## 3.2 差分方程式

Poisson 方程式の S-W 近似による差分化は次のようになる。

$$\begin{cases} -\left(\frac{2}{h_E+h_W}\left(\frac{u(P_E)-u(P)}{h_E}-\frac{u(P)-u(P_W)}{h_W}\right)+\frac{2}{h_N+h_S}\left(\frac{u(P_N)-u(P)}{h_N}-\frac{u(P)-u(P_S)}{h_S}\right)\right)=f(P) & (P\in\Omega) \\ u(P)=g(P) & (P\in\partial\Omega) \end{cases} \tag{3.2}$$

このとき、$u(P)$ $(\forall P\in\Omega)$ が未知数となる。

続いて、この差分方程式を領域内および境界上の全ての等間隔格子点 $P$ について並べた連立方程式を構成する。そのような連立方程式を行列の書き方に沿って $AU=F$ とするが、ここで $A$ は係数行列、$U$ は未知ベクトル、$F$ は既知ベクトルであることを断っておく。

## 3.3 未知ベクトル $U$ の構成法

まずは、領域内部および境界上の等間隔格子点に 0 番目から番号をつけていく。例えば、$N_x=N_y=4$ のときは図 3.2 のように番号をつけていく。すなわち最も下の格子軸のうち左にある等間隔格子点から順番に番号をつけ、その格子軸上の領域内部または境界上の等間隔格子点が無くなったら次の格子軸に移るということを繰り返して行う。

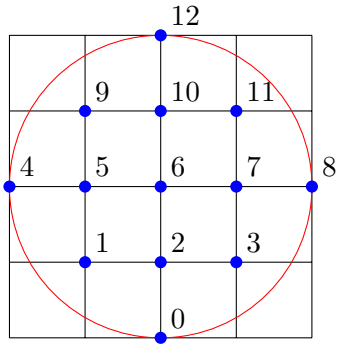


図 3.2: 番号のつけ方

後で番号 $k$ からその点の $(i,j)$ を求める必要があるため、次の定義を行う。

**定義 3.1**
$k$ 番目の点を $P_k$ と表すことにする。$P_k$ が $(i,j)$ の格子点であるとき、

$$p(k)=i,\ q(k)=j,\ V(i,j)=k$$

となる $p(k),q(k),V(i,j)$ を定義する。

このようにして数え上げた等間隔格子点の個数を $n$ とする。ここで $u(P_k)=u_k$ と表すことにする $(0\le k\le n-1)$。

3.2 節でも述べたように $u_k$ は未知数であるから、$U={}^t(u_0,u_1,\cdots,u_{n-1})$ として未知ベクトル $U$ を構成する。

## 3.4 係数行列 $A$ の構成法

係数行列 $A\in M(n,\mathbb{R})$ は次のようにして構成する。

$u_k$ の係数、すなわち対角成分 $A_{kk}$ は、

$$A_{kk} = \begin{cases} \dfrac{2(h_E h_W + h_N h_S)}{h_E h_W h_N h_S} & (P_k \in \Omega) \\ 1 & (P_k \in \partial\Omega) \end{cases}$$

である。$P_E, P_W, P_N, P_S$ の番号を $k_E, k_W, k_N, k_S$ とすると、

$$A_{kk_E} = \begin{cases} \dfrac{-2}{h_E(h_E + h_W)} & (P_E \in \overline{\Omega}) \\ 0 & (\text{else}) \end{cases}, A_{kk_W} = \begin{cases} \dfrac{-2}{h_W(h_E + h_W)} & (P_W \in \overline{\Omega}) \\ 0 & (\text{else}) \end{cases}$$

$$A_{kk_N} = \begin{cases} \dfrac{-2}{h_N(h_N + h_S)} & (P_N \in \overline{\Omega}) \\ 0 & (\text{else}) \end{cases}, A_{kk_S} = \begin{cases} \dfrac{-2}{h_S(h_N + h_S)} & (P_S \in \overline{\Omega}) \\ 0 & (\text{else}) \end{cases}$$

それ以外の $A$ の成分は全て 0 である。

## 3.5 ベクトル $F$ の構成法

天下り的だが、まず $n$ 次元縦ベクトル $F' = {}^t(F'_0, F'_1, \cdots, F'_{n-1})$ を以下のようにして定義する。

$$F'_k = \begin{cases} f(P_k) & (P_k \in \Omega) \\ g(P_k) & (P_k \in \partial\Omega) \end{cases}$$

S-W 近似では領域外の等間隔格子点の代わりに境界上の不等間隔格子点を使うが、そのときの未知関数の値は境界条件を使う。そのため実は既知関数である。そこで境界条件を使うようなときは、不等間隔格子点における未知関数の値 = 境界条件の値とその係数を、右辺に移項してから計算を行う。

そのために次のような $n$ 次元縦ベクトル $F_E$(右補正ベクトルと呼ぶことにする) を定義する。

$$F_{Ek} = \begin{cases} \dfrac{2}{h_E(h_E + h_W)} & (M(p(k)+1, q(k)) = -1) \\ 0 & (\text{else}) \end{cases}$$

また左、上、下補正ベクトル $F_W, F_N, F_S$ を同様に定義する。

$$F_{Wk} = \begin{cases} \dfrac{2}{h_W(h_E + h_W)} & (M(p(k)-1, q(k)) = -1) \\ 0 & (\text{else}) \end{cases}$$

$$F_{Nk} = \begin{cases} \dfrac{2}{h_N(h_N + h_S)} & (M(p(k), q(k)+1) = -1) \\ 0 & (\text{else}) \end{cases}$$

$$F_{Sk} = \begin{cases} \dfrac{2}{h_S(h_N + h_S)} & (M(p(k), q(k)-1) = -1) \\ 0 & (\text{else}) \end{cases}$$

以上の 4 つの補正ベクトルと $F'$ を用いて

$$F = F' + F_E + F_W + F_N + F_S$$

と定義する。

## 3.6 シミュレーション結果

以上のようにして構成した連立方程式 $AU = F$ を $U$ について解き、その結果を可視化した。$f$ と $g$ は

$$f(x, y) = -4(x^2 + y^2 - 1)e^{-x^2-y^2}$$
$$g(x, y) = e^{-1}$$

としている。

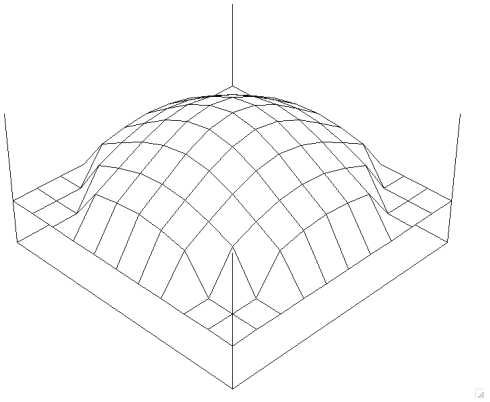

図 3.3: $N_x = N_y = 10$

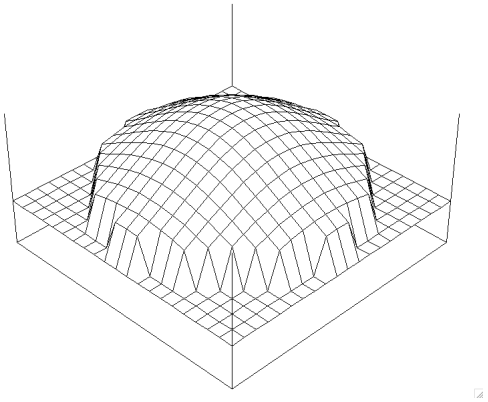

図 3.4: $N_x = N_y = 20$

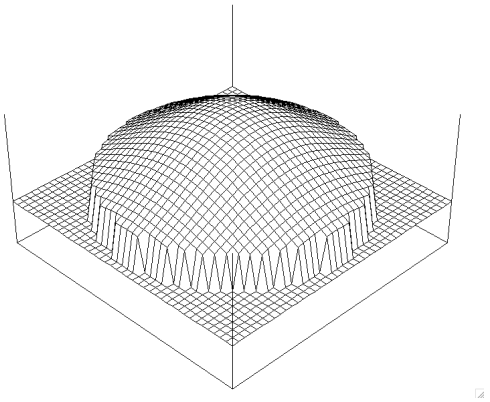

図 3.5: $N_x = N_y = 40$

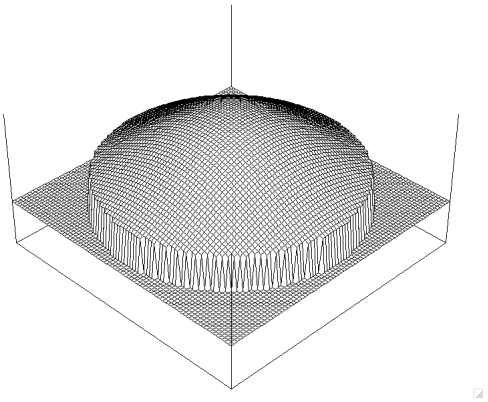

図 3.6: $N_x = N_y = 80$

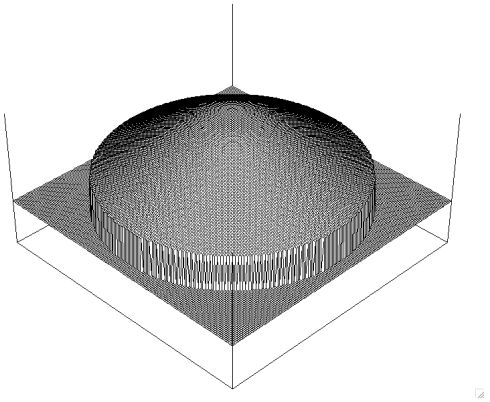

図 3.7: $N_x = N_y = 160$

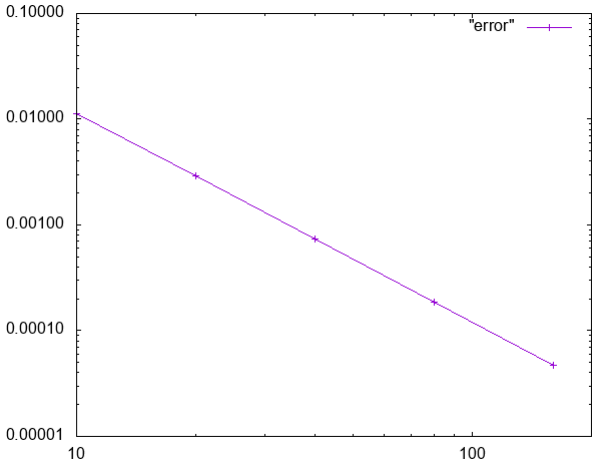


図 3.8: 誤差の推移

次に厳密解 $u$ と数値解 $U$ の誤差を調べた。ここでいう誤差とは、各等間隔格子点 $P$ における $|u(P) - U(P)|$ の最大値のことを指す。先程の $f$ と $g$ に対して厳密解 $u$ は

$$u(x, y) = e^{-x^2-y^2}$$

となる。

図 3.8 が分割数を増やしたときの誤差の推移である。横軸が分割数、縦軸が誤差 $\max_{P \in \overline{\Omega}} \{u(P) - U(P)\}$ である。

## 3.7 まとめ

図 3.8 からわかるように、山本 [2] の主張を確認することができた。

また今回作成したプログラムをたたき台にして、熱方程式や波動方程式など様々な偏微分方程式に対して陰解法による数値計算プログラムが作成できると考えている。

# 付 録A　使用したプログラムのソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <glsc.h>


/*楕円板の半径*/
#define rx (1.0)
#define ry (1.0)
/*円の中心*/
#define a (0)
#define b (0)
/*分割領域*/
#define xmax (rx+0.1)
#define xmin (-rx-0.1)
#define ymax (ry+0.1)
#define ymin (-ry-0.1)
/*マーカーの大きさ*/
#define marker_size (1)
/*正方形の分割数*/
int Nx,Ny;
/*正方形の分割幅*/
#define hx ((xmax-xmin)/Nx)
#define hy ((ymax-ymin)/Ny)
/*丸め誤差*/
#define er (10e-13)

int select;

/*2 乗*/
double d(double x)
{
return x*x;
}

/*境界上なら 1, それ以外 0*/
int onBoundary(double x,double y)
{
```

```
if(fabs(1-(d(x)+d(y)))<=er)
{
return 1;
}else
{
return 0;
}
}

/*領域内なら 1, それ以外 0*/
int inDomain(double x,double y)
{
if(1-(d(x)+d(y))>er)
{
return 1;
}else
{
return 0;
}
}

double f(double x,double y)
{
if(select==1)
{
return 4;
}else if(select==2)
{
return -4*(d(x)+d(y)-1)*exp(-d(x)-d(y));
}else if(select==3)
{
return -2*(d(x)+d(y))*sin(2*x*y);
}else
{
return -4*(d(x)+d(y)-1)*exp(1-d(x)-d(y));
}
}

double g(double x,double y)
{
if(select==1)
{
return 0;
}else if(select==2)
{
```

```
return exp(-1);
}else if(select==3)
{
return sin(x*y)*cos(x*y);
}else
{
return 1;
}
}

/*厳密解 u*/
double analysis_u(double x,double y)
{
if(inDomain(x,y)==1)
{
if(select==1)
{
return 1-(d(x)+d(y));
}else if(select==2)
{
return exp(-d(x)-d(y));
}else if(select==3)
{
return sin(x*y)*cos(x*y);
}else
{
return exp(1-d(x)-d(y));
}
}else if(onBoundary(x,y)==1)
{
if(select==1)
{
return 1-(d(x)+d(y));
}else if(select==2)
{
return exp(-1);
}else if(select==3)
{
return sin(x*y)*cos(x*y);
}else
{
return exp(1-d(x)-d(y));
}
}else
{
```

```
return 0;
}
}

/*最大値*/
void maximum(double *X,int *P,int *Q,int n)
{
int k;
double err;
double xi,yj;

err=fabs(X[0]-analysis_u(xmin+P[0]*hx,ymin+Q[0]*hy));

double x1,y1;
x1=0;
y1=0;
for(k=0;k<n;k++)
{
xi=xmin+P[k]*hx;
yj=ymin+Q[k]*hy;

if(err<fabs(X[k]-analysis_u(xi,yj)))
{
err=fabs(X[k]-analysis_u(xi,yj));
x1=xi;
y1=yj;
}
}

printf("最大誤差=%e\n",err);
printf("(%f,%f)\n",x1,y1);
}

/*LU 分解*/
void lu(int n,double **a1,double **l1,double **u1)
{
int i,j,k;
double q;

for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
u1[i][j]=a1[i][j];
l1[i][j]=0.0;
```

```
}
l1[i][i]=1.0;
}

for(k=0;k<n-1;k++)
{
for(i=k+1;i<n;i++)
{
q=u1[i][k]/u1[k][k];
for(j=k;j<n;j++)
{
u1[i][j]=u1[i][j]-q*u1[k][j];
}
l1[i][k]=q;
}
}
}

/*係数行列の形の連立方程式を解く*/
void solve(int n,double **l1,double **u1,double *B)
{
int i,j,k;
/*Ly=b を解く*/
double s;
s=0;
B[0]=B[0]/l1[0][0];
for(i=1;i<n;i++)
{
s=0;
for(j=0;j<i;j++)
{
s+=l1[i][j]*B[j];
}
B[i]=(B[i]-s)/l1[i][i];
}

/*Ux=b を解く*/
B[n-1]=B[n-1]/u1[n-1][n-1];
for(i=n-2;i>=0;i--)
{
s=0;
for(j=i+1;j<n;j++)
{
s+=u1[i][j]*B[j];
}
```

```
B[i]=(B[i]-s)/u1[i][i];
}
}

/*東側の境界*/
double E(double y)
{
return a+(rx/ry)*sqrt(d(ry)-d(y-b));
}

/*西側の境界*/
double W(double y)
{
return a-(rx/ry)*sqrt(d(ry)-d(y-b));
}

/*北側の境界*/
double N(double x)
{
return b+(ry/rx)*sqrt(d(rx)-d(x-a));
}

/*南側の境界*/
double S(double x)
{
return b-(ry/rx)*sqrt(d(rx)-d(x-a));
}

int main()
{
printf(" 1. 4\n 2. -4(x^2+y^2-1)exp(-x^2-y^2)\n 3. -2(x^2+y^2)sin(2xy)\n 4. その他
\n");
scanf("%d",&select);
printf("Nx,Ny="); scanf("%d",&Nx);
Ny=Nx;

/*カウンタ*/
int i,j,k;
/*(i,j) 番目の座標*/
double xi,yj;
/*領域内の点の番号*/
int **V;
int *V2;
V=(int**)malloc(sizeof(int*)*(Nx+1));
V2=malloc(sizeof(int)*(Nx+1)*(Ny+1));
```

```
for(i=0;i<=Nx;i++)
{
V[i]=V2+i*(Ny+1);
}
/*領域内外、境界線上の判定*/
int **M;
int *M2;
M=(int**)malloc(sizeof(int*)*(Nx+1));
M2=malloc(sizeof(int)*(Nx+1)*(Ny+1));
for(i=0;i<=Nx;i++)
{
M[i]=M2+i*(Ny+1);
}
/*番号*/
int num;
num=0;
for(j=0;j<=Ny;j++)
{
yj=ymin+j*hy;
for(i=0;i<=Nx;i++)
{
xi=xmin+i*hx;
if(inDomain(xi,yj)==1) //領域内部
{
V[i][j]=num;
num+=1;
M[i][j]=1;
}else if(onBoundary(xi,yj)==1) //境界上
{
V[i][j]=num;
num+=1;
M[i][j]=0;
}else//領域外部
{
V[i][j]=-1;
M[i][j]=-1;
}
}
}
/*行列の大きさ*/
int n;
n=num;
/*点の番号から (i,j) を逆引きする*/
int *p;
int *q;
```

```
p=(int*)malloc(sizeof(int*)*(n));
q=(int*)malloc(sizeof(int*)*(n));
for(j=0;j<=Ny;j++)
{
for(i=0;i<=Nx;i++)
{
if(M[i][j]>=0)
{
p[V[i][j]]=i;
q[V[i][j]]=j;
}
}
}
/*解を格納する配列*/
double **u;
double *u2;
u=(double**)malloc(sizeof(double*)*(Nx+1));
u2=malloc(sizeof(double)*(Nx+1)*(Ny+1));
for(i=0;i<=Nx;i++)
{
u[i]=u2+i*(Ny+1);
}
for(i=0;i<=Nx;i++)
{
for(j=0;j<=Ny;j++)
{
u[i][j]=0;
}
}

printf("hx=%f\thy=%f\n",hx,hy);
printf("n=%d\n",n);


/*係数行列 A*/
double **A;
double *A2;
A=(double**)malloc(sizeof(double*)*n);
A2=malloc(sizeof(double)*n*n);
for(i=0;i<n;i++)
{
A[i]=A2+i*n;
}
for(i=0;i<n;i++)
{
```

```
for(j=0;j<n;j++)
{
A[i][j]=0;
}
}
/*隣接点との距離を he,hw,hn,hs に格納する*/
double he,hw,hn,hs;


/*右辺のベクトル*/
double *F;
F=(double*)malloc(sizeof(double*)*n);
for(i=0;i<n;i++)
{
F[i]=0;
}
/*F に数値を入れる*/
for(k=0;k<n;k++)
{
xi=xmin+p[k]*hx;
yj=ymin+q[k]*hy;
if(M[p[k]][q[k]]==1)
{
F[k]=f(xi,yj);
}else if(M[p[k]][q[k]]==0)
{
F[k]=g(xi,yj);
}else
{
F[k]=0;
}
}


double *FE;
FE=(double*)malloc(sizeof(double*)*n);
double *FW;
FW=(double*)malloc(sizeof(double*)*n);
double *FN;
FN=(double*)malloc(sizeof(double*)*n);
double *FS;
FS=(double*)malloc(sizeof(double*)*n);
for(i=0;i<n;i++)
{
FE[i]=0;
```

```
FW[i]=0;
FN[i]=0;
FS[i]=0;
}


/*係数行列を作る*/
for(k=0;k<n;k++)
{
i=p[k];
j=q[k];

xi=xmin+i*hx;
yj=ymin+j*hy;

he=hx;
hw=hx;
hn=hy;
hs=hy;

if(M[i][j]>0) //P が内部
{
if(M[i+1][j]<0) //PE が外部
{
he=E(yj)-xi;
FE[k]=2*g(E(yj),yj)/(he*(he+hw));
}
if(M[i-1][j]<0) //PW が外部
{
hw=xi-W(yj);
FW[k]=2*g(W(yj),yj)/(hw*(he+hw));
}
if(M[i][j+1]<0) //PN が外部
{
hn=N(xi)-yj;
FN[k]=2*g(xi,N(xi))/(hn*(hn+hs));
}
if(M[i][j-1]<0) //PS が外部
{
hs=yj-S(xi);
FS[k]=2*g(xi,S(xi))/(hs*(hn+hs));
}


A[k][k]=2*(he*hw+hn*hs)/(he*hw*hn*hs);
```

```
if(M[i+1][j]>=0) //PE が内部
{
A[k][V[i+1][j]]=-2/(he*(he+hw));
}
if(M[i-1][j]>=0) //PW が内部
{
A[k][V[i-1][j]]=-2/(hw*(he+hw));
}
if(M[i][j+1]>=0) //PN が内部
{
A[k][V[i][j+1]]=-2/(hn*(hn+hs));
}
if(M[i][j-1]>=0) //PS が内部
{
A[k][V[i][j-1]]=-2/(hs*(hn+hs));
}
}else if(M[i][j]==0) //P が境界上
{
A[k][k]=1;
}
}

for(k=0;k<n;k++)
{
F[k]=F[k]+FE[k]+FW[k]+FN[k]+FS[k];
}

/*LU 分解の L*/
double **L;
double *L2;
L=(double**)malloc(sizeof(double*)*n);
L2=malloc(sizeof(double)*n*n);
for(i=0;i<n;i++)
{
L[i]=L2+i*n;
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
L[i][j]=0;
}
}
/*LU 分解の U*/
```

```
double **U;
double *U2;
U=(double**)malloc(sizeof(double*)*n);
U2=malloc(sizeof(double)*n*n);
for(i=0;i<n;i++)
{
U[i]=U2+i*n;
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
U[i][j]=0;
}
}
/*A を LU 分解する*/
lu(n,A,L,U);
/*方程式を解く*/
solve(n,L,U,F);
/*解を代入*/
for(k=0;k<n;k++)
{
u[p[k]][q[k]]=F[k];
}
/*誤差の最大値を求める*/
maximum(F,p,q,n);

g_init("Graph",200,180);
g_device(G_DISP);

g_def_scale(0,xmin-0.2,xmax+0.2,ymin-0.2,ymax+0.2,15.0,15.0,150.0,150.0);

g_cls();
g_sel_scale(0);

g_marker_size(marker_size);
g_marker_type(-1);

g_hidden(100.0,100.0,50.0,-0.5,1.0,500.0,
45.0,35.0,10.0,10.0,180.0,180.0,
u[0],Nx+1,Ny+1,1,0,1,1);
g_sleep(G_STOP);
g_term();
}
```

# 参考文献


[1] G. H. Shortley, R. Weller : The Numerical Solution of Laplace's Equation, *Mendenhall Laboratory of Physics, Ohio State University, Columbus, Ohio*, 1938

[2] 山本　哲郎：数値解析入門 (増訂板)、サイエンス社 (2003) 216-217

[3] Lisl Weynans : Super-convergence in maximum norm of the gradient for the Shortley- Weller method. [Research Report] RR-8757, INRIA Bordeaux; INRIA. 2017, pp.14. ¡hal- 01176994v2¿

[4] 久保田　祥史：S-W 近似によって様々な領域の熱方程式を解く 2007
`http://nalab.mind.meiji.ac.jp/~mk/labo/report/open/2007-kubota.pdf`

[5] 濱　勇樹：S-W 近似による楕円領域での波動方程式のシミュレーション 2011
`http://nalab.mind.meiji.ac.jp/~mk/labo/report/pdf/2011-hama.pdf`