応用数値解析（演習編）

1. プログラム演習のアウトライン

1.1 はじめに

応用数値解析の講義編では有限要素法、境界要素法、差分法の概論について学んだ。今年度の応用数値解析・プログラム演習編では差分法を用いて工学上重要性の高い非圧縮性流体計算のプログラム作成を通し、理解を深めることを目的とする。

1.2 目標

機械工学の中の流体工学において非常に重要となる非圧縮性流体の物体（角柱など）回りの流れの 2 次元非定常計算を行い、物体後方にカルマン渦などが発生することを確認する。また、物体周りの圧力を積分することで物体に働く抗力やカルマン渦列の周期からストローハル数を計算することもできる。

規格化された 2 次元非圧縮性 Navier-Stokes 方程式は以下のように表わされる。

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{1-1}$$

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{\partial p}{\partial x}+\frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{1-2}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{\partial p}{\partial y}+\frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{1-3}$$

$\boldsymbol{u}=(u,v)$ は、流速ベクトルで、u と v はその x 方向および y 方向成分である。 p は圧力、Re はレイノルズ数（無次元数）であり、非圧縮性流体の流れの振る舞いはレイノルズ数によって大きく変わる。(1-1)式は連続方程式から導かれる非圧縮性の条件である。

非圧縮性流体の非定常計算を行うには、初期条件からスタートし、各時間ステップで(1-1)～(1-3)式を満たすように u， v， p を求めるということである。ここでは、SMAC 法を用いて非圧縮性流体方程式を解くことにする。

SMAC 法

n 時間ステップまで計算が終了し、n+1 ステップの計算を行うとき、(具体的な差分式は後述) SMAC 法は(1-2)および(1-3)式中の圧力を n ステップの圧力 p^n として n+1 ステップの u^{n+1} と v^{n+1} を計算する。しかし、この速度は連続の式(1-1)を満たさない。

$$\frac{\partial u^{n+1}}{\partial x}+\frac{\partial v^{n+1}}{\partial y}\neq 0 \tag{1-4}$$

そこで、圧力を p^n として求めた速度を u^* と v^* として保存しておくことにする。例えば時間微

分項に前進差分を適用した場合は、

$$u^* = u^n + \left[-u^n \frac{\partial u^n}{\partial x} - v^n \frac{\partial u^n}{\partial y} - \frac{\partial p^n}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u^n}{\partial x^2} + \frac{\partial^2 u^n}{\partial y^2} \right) \right] \Delta t \tag{1-5}$$

$$v^* = v^n + \left[-u^n \frac{\partial v^n}{\partial x} - v^n \frac{\partial v^n}{\partial y} - \frac{\partial p^n}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v^n}{\partial x^2} + \frac{\partial^2 v^n}{\partial y^2} \right) \right] \Delta t \tag{1-6}$$

で計算できる。

SMAC 法は n+1 ステップの p^{n+1}, u^{n+1}, v^{n+1} を $p^{n+1} = p^n + \delta p$, $u^{n+1} = u^* + \delta u$, $v^{n+1} = v^* + \delta v$ と仮定し、(1-1)式を満たすように δp, δu, δv を求める。つまり、

$$\delta u = -\frac{\partial \delta p}{\partial x} \Delta t \tag{1-7}$$

$$\delta v = -\frac{\partial \delta p}{\partial y} \Delta t \tag{1-8}$$

とし、$\frac{\partial}{\partial x}$(1-7) + $\frac{\partial}{\partial y}$(1-8) を計算すると、

$$\left(\frac{\partial u^{n+1}}{\partial x} + \frac{\partial v^{n+1}}{\partial y} \right) - \left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} \right) = -\left(\frac{\partial^2 \delta p}{\partial x^2} + \frac{\partial^2 \delta p}{\partial y^2} \right) \Delta t \tag{1-9}$$

である。n+1 ステップで(1-1)式が満足されるとすると、

$$\frac{\partial^2 \delta p}{\partial x^2} + \frac{\partial^2 \delta p}{\partial y^2} = \left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} \right) / \Delta t \tag{1-10}$$

となり、(1-10)式の右辺は既知であるので、(1-10)式は δp に関する Poisson 方程式である。もし、(1-10)式を満足するように δp を求めることができたならば、(1-7)と(1-8)式から

$$u^{n+1} = u^* - \frac{\partial \delta p}{\partial x} \Delta t \tag{1-11}$$

$$v^{n+1} = v^* - \frac{\partial \delta p}{\partial y} \Delta t \tag{1-12}$$

で求められる u^{n+1} と v^{n+1} は(1-1)式を満足することが分かる。

以上をまとめると、非圧縮性流体方程式を SMAC 法で計算するプロセスは、

① (1-5)式と(1-6)式に従って、速度 u^* と v^* を計算する。

② (1-10)式の右辺の速度 $\boldsymbol{u}^*(u^*, v^*)$ の divergence を計算する。

② Possion 方程式を解き、圧力の修正量 δp を求める。

③ 圧力の修正量 δp を使って、(1-11) 式と(1-12) 式に従って速度 u^* と v^* を修正し、u^{n+1} と v^{n+1} を求める。

2. 二次元拡散方程式の計算

計算領域$0 \le x \le L_x$， $0 \le y \le L_y$において、２次元拡散方程式

$$\frac{\partial f}{\partial t} = \kappa\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right) \tag{2-1}$$

を解く。ここで拡散係数κは正の定数（$\kappa = 1.0$）とする。初期条件は$0.3 \le x \le 0.7$, $0.3 \le y \le 0.7$の範囲内のみ$f(x, y) = 1.0$、その他の領域では$f(x, y) = 0.0$とする。時刻$0 \le t \le 0.02$まで計算し、途中、20ステップ毎にfのプロファイルをBMP形式に画像ファイルとして出力する。

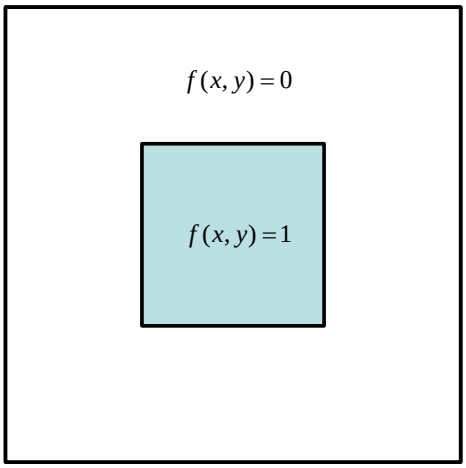


図 2-1　初期条件

x方向にN_x個の格子点、y方向にN_y個の格子点を用いる場合、格子間隔は$\Delta x = L_x/(N_x - 1)$， $\Delta y = L_y/(N_y - 1)$となる。2次元格子は以下のようになる。

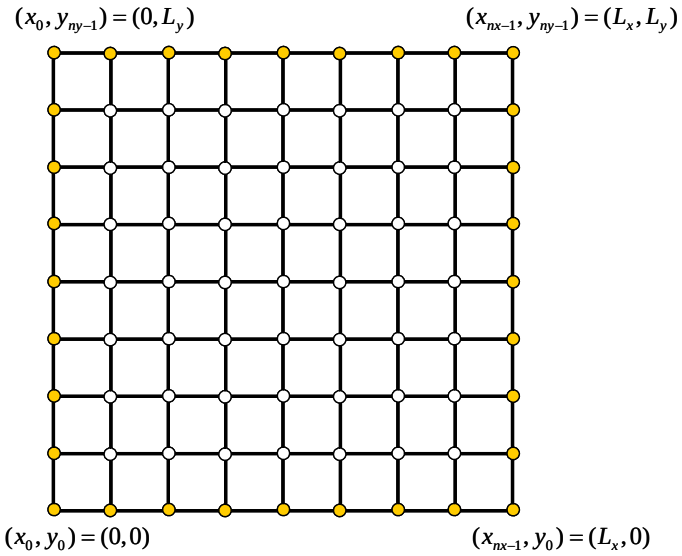


図 2-2　2 次元格子

黄色い格子点は境界条件を設定する格子点で、常に$f = 0$とする。

この 2 次元格子上で拡散方程式を計算するために、C 言語プログラムでは

```
Static  double  f[NY][NX],  fn[NY][NX];
```

と静的にメモリを確保している。f[NY][NX]が $f_{i,j}^{n}$,fn[NY][NX]が $f_{i,j}^{n+1}$ の内容を格納するための配列である。C 言語では配列要素の順番が逆になっているので、

$$f[j][i] \quad \rightarrow \quad f_{i,j}^{n} \tag{2-2}$$

であることに注意する必要がある。$\Delta x = L_x / 100$ および $\Delta y = L_y / 100$ としたければ、

```
#define  NX  (100+1)
#define  NY  (100+1)
```

としておくと良い。

時間ステップ間隔を Δt とし、時間微分項には前進差分、空間微分項には中心差分を適用すると、

$$\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} = \kappa \left(\frac{f_{i+1,j}^{n} - 2 f_{i,j}^{n} + f_{i-1,j}^{n}}{\Delta x^2} + \frac{f_{i,j+1}^{n} - 2 f_{i,j}^{n} + f_{i,j-1}^{n}}{\Delta y^2} \right) \tag{2-3}$$

となる。f^{n} は既知であるので、(2-3) 式は $f_{i,j}^{n+1}$ について容易に解くことができ、上図の白い格子点のみ計算するための C 言語プログラムは以下のようになる。

```
for(jy = 1; jy < ny - 1; jy++) {
    for(jx = 1; jx < nx - 1; jx++) {
        fn[jy][jx] = f[jy][jx] + kappa*dt*(
                (f[jy][jx+1] - 2.0*f[jy][jx] + f[jy][jx-1])/(dx*dx)
              + (f[jy+1][jx] - 2.0*f[jy][jx] + f[jy-1][jx])/(dy*dy)
        );
    }
}
```

時間ステップ幅 Δt は、Von Neumann の安定性解析から、

$$\Delta t = 0.2 * MIN(\Delta x, \Delta y)^2 / \kappa$$

と設定している。

以上をまとめると、拡散方程式を解くプログラム手順は、

（1） 初期条件の設定

（2） Δt の設定

（3） 境界内部の格子点で拡散方程式を計算　（$f_{i,j}^{n+1}$ の計算）

（4） 境界条件の設定

（5） 時間更新（全ての格子点上で $f_{i,j}^{n+1} \rightarrow f_{i,j}^{n}$）

$t > 0.02$ まで繰り返し

となる。

BMP ファイルを生成する関数

```
void   bmp_r8
// ==================================================================
(
   int      nx,            /* x 方向の格子点数                        */
   int      ny,            /* y 方向の格子点数                        */
   double   f[][nx],       /* 表示したい変数の配列                     */
   int      mul,           /* 倍率                                    */
   double   fmax,          /* 表示する f の値の最大値                  */
   double   fmin,          /* 表示する f の値の最小値                  */
   int      mesh,          /* メッシュを表示するか、しないか           */
   char     *filename,     /* 出力する BMP ファイル名                  */
   char     *palette       /* 配色を設定するパレットファイル           */
)
// ------------------------------------------------------------------
```

(1) 配列 f は f[ny][nx] で定義されてる必要がある。さもないと、配列要素が整合しなくなる。

(2) 倍率 mul = 1 ならば、縦 nx, 横 ny ピクセルの BMP ファイルが生成される。Mul は正の整数で、1 より大きい場合は縦 nx*mul, 横 ny*mul ピクセルに線形補間で拡大された BMP ファイルが生成される。

(3) fmax と fmin は表示する値の限度を表す。もし配列 f[][] 要素の値が -0.2 < f < 1.2 であり、fmax = 1.0, fmin = 0.0 とした場合、f > fmax の f は fmax に、f < fmin の f は fmin にクリッピングされる。BMP ファイルの各点は、fmin 〜 fmax を 256 階調で表した色が付けられる。

(4) mesh は、mesh = 1 ならばメッシュをオーバーラップして描く。ただし、mul > 3 の場合のみ。

(5) filename は、BMP ファイルの名前

(6) palette は、(3)で説明した 256 階調に色を割り当てるためのファイル。(こちらで準備)

3. 二次元移流方程式の計算①（一次精度風上差分）

計算領域$0 \le x \le L_x$，$0 \le y \le L_y$において、２次元移流方程式

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = 0 \tag{3-1}$$

を解く。u, v 移流速度である。初期条件は拡散方程式を解いたときと同じように、$0.3 \le x \le 0.7$, $0.3 \le y \le 0.7$の範囲内のみ$f(x,y) = 1.0$、その他の領域では$f(x,y) = 0.0$とする。時刻$0 \le t \le 1.0$まで計算し、途中、20 ステップ毎にfのプロファイルを BMP 形式に画像ファイルとして出力する。ただし、今回は x 方向、y 方向ともに周期的境界条件とする。

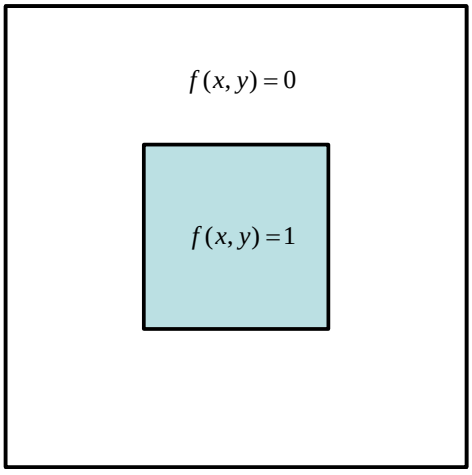


図 3-1　初期条件

2 次元格子上の(i, j)点での移流方程式の計算は、時間ステップ間隔をΔtとし、時間微分項には前進差分、空間微分項には 1 次精度風上差分を適用すると、

$$\begin{aligned}
&\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} + u\frac{f_{i,j}^{n} - f_{i-1,j}^{n}}{\Delta x} + v\frac{f_{i,j}^{n} - f_{i,j-1}^{n}}{\Delta y} = 0 \quad (u \ge 0,\ v \ge 0)\\
&\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} + u\frac{f_{i+1,j}^{n} - f_{i,j}^{n}}{\Delta x} + v\frac{f_{i,j}^{n} - f_{i,j-1}^{n}}{\Delta y} = 0 \quad (u < 0,\ v \ge 0)\\
&\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} + u\frac{f_{i,j}^{n} - f_{i-1,j}^{n}}{\Delta x} + v\frac{f_{i,j+1}^{n} - f_{i,j}^{n}}{\Delta y} = 0 \quad (u \ge 0,\ v < 0)\\
&\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} + u\frac{f_{i+1,j}^{n} - f_{i,j}^{n}}{\Delta x} + v\frac{f_{i,j+1}^{n} - f_{i,j}^{n}}{\Delta y} = 0 \quad (u < 0,\ v < 0)
\end{aligned} \tag{3-2}$$

となる。

時間ステップ幅Δtは、CFL 条件から、

$$\Delta t = 0.2 * MIN(\Delta x / u, \Delta y / v)$$

とする。

周期的境界条件は、

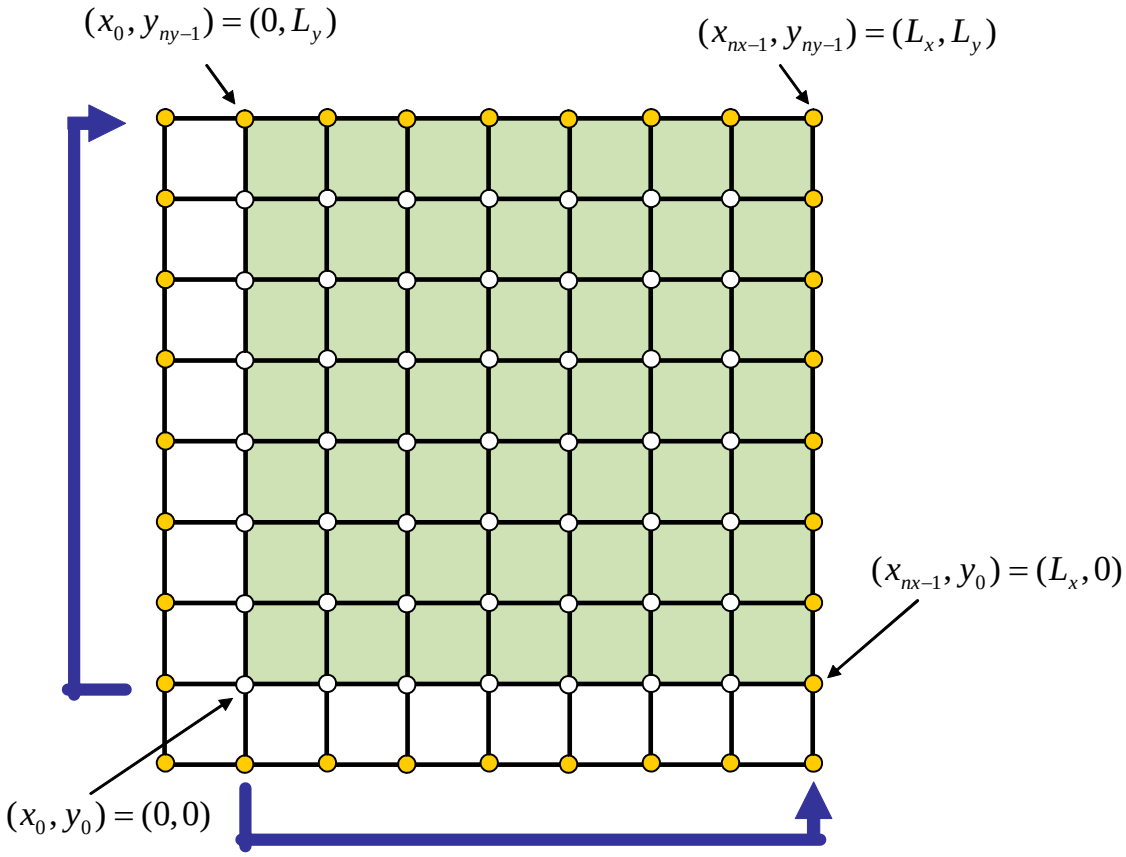


図 3-2　周期的境界条件

となり、 格子間隔は格子間隔は$\Delta x = L_x/(N_x - 2)$, $\Delta y = L_y/(N_y - 2)$となる。$\Delta x = L_x/100$および$\Delta y = L_y/100$としたければ、

```
#define  NX  (100+2)
#define  NY  (100+2)
```

としておくと良い。

周期境界および$u = 1,\ v = 1$とすることにより、時刻$t = 0$からスタートし、時刻$t = 1.0$には、プロファイルは元に戻るはずである。

初期状態を$f(x, y) = \sin(2\pi x)\sin(2\pi y)$と設定することにより、$t = 1.0$の計算結果を初期状態と比較することで誤差を以下のように評価することができる。

$$ERROR = \frac{1}{(N_x - 2)(N_y - 2)} \sum_{i=1}^{N_x-2} \sum_{j=1}^{N_y-2} \left| f_{i,j} - \sin(2\pi x_i)\sin(2\pi y_j) \right| \tag{3-3}$$

格子間隔を$\Delta x = L_x/200$および$\Delta y = L_y/200$などと変えることにより、誤差の変化を調べることができる。

演習課題 1:

２次元拡散方程式を計算するプログラムを修正し、２次元移流方程式を解くプログラムを作成する。

(1) #define NX と #define NY を修正する。
(2) Δx, Δy を修正する。
(3) Δt を CFL 条件に従って決める。
(4) diffusion2d() 関数の代わりに、移流方程式と解く関数 advection2d()を作成する。
(5) 境界条件を設定する関数 boundary()を周期境界用に修正する。
(6) 計算終了の時刻を修正する。
(7) 動画で動き確認する。$t = 1.0$ の結果を確認する。
(8) palette ファイルを Rainbow.pal から Seismic.pal に変更して計算を実行し、動画を確認して見る。
(9) 初期状態を $f(x, y) = \sin(2\pi x)\sin(2\pi y)$ と設定する。
(10) 誤差を(3)式で求める関数 error_f()を作成する。
(11) NX および NY を変更し、誤差の変化を調べる。

移流方程式と解く関数としては、

```
void   advection2d
// ====================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
   double   f[][nx],       /* dependent variable                    */
   double   fn[][nx],      /* time-integrated dependent variable    */
   double   u,             /* advection velocity in the x-direction */
   double   v,             /* advection velocity in the y-direction */
   double   dt,            /* time step interval                    */
   double   dx,            /* grid spacing in the x-direction       */
   double   dy             /* grid spacing in the y-direction       */
)
// --------------------------------------------------------------------
```

初期状態を $f(x, y) = \sin(2\pi x)\sin(2\pi y)$ と設定する関数の１例を以下に示す。

```
void   initial
// ====================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
   double   dx,            /* grid spacing in the x-direction       */
   double   dy,            /* grid spacing in the y-direction       */
   double   f[][nx]        /* dependent variable f                  */
)
// --------------------------------------------------------------------
```

```
{
    int     jx,   jy;
    double  x,    y;

    for(jy=0 ; jy < ny; jy++) {
        for(jx=0 ; jx < nx; jx++) {
            x = dx*(double)(jx - 1);
            y = dy*(double)(jy - 1);
            f[jy][jx] = sin(2.0*M_PI*x)*sin(2.0*M_PI*y);
        }
    }
}
```

ここで、M_PI は数学のπである。M_PI や数学関数 sin(x) を使うためには、#include <math.h> を#include <stdio.h>の次に追加するとともに、gcc でコンパイル・リンクする際に、-lm を加える必要がある。

(3-3)式の誤差を計算するプログラムの一例は、

```
double   error_f
// ==================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
   double   dx,            /* grid spacing in the x-direction       */
   double   dy,            /* grid spacing in the y-direction       */
   double   f[][nx]        /* dependent variable f                  */
)
// ------------------------------------------------------------------
{
    int     jx,   jy;
    double  x,    y,  err = 0.0;

    for(jy=1 ; jy < ny-1; jy++) {
        for(jx=1 ; jx < nx-1; jx++) {
            x = dx*(double)(jx - 1);
            y = dy*(double)(jy - 1);
            err += fabs(f[jy][jx] - sin(2.0*M_PI*x)*sin(2.0*M_PI*y));
        }
    }
    return err/(double)((nx - 2)*(ny - 2));
}
```

である。

4. 二次元移流方程式の計算②（Cubic セミ・ラグランジアン法）

計算領域$0 \le x \le L_x$， $0 \le y \le L_y$において、２次元移流方程式

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = 0 \tag{4-1}$$

を解く。u, vは移流速度である。初期条件および境界条件等は全て前の課題と同じである。ここでは、時間・空間 3 次精度の Cubic セミ・ラグランジアン法を用いる。(4-1)式を以下のように 2 つの方程式に分割して計算を進める Fractional Step 法を導入することで、前に学んだ 1 次元の Cubic セミ・ラグランジアン法を用いることができる。

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \tag{4-2}$$

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial y} = 0 \tag{4-3}$$

例えば、$u \ge 0$の場合、(4-2)式を点$(i,\ j)$上の$f_{i,j}^{n+1}$を Cubic セミ・ラグランジアン法で解くには、$f_{i-2,j}^{n}$，$f_{i-1,j}^{n}$，$f_{i,j}^{n}$，$f_{i+1,j}^{n}$の 4 点を通る 3 次多項式の補間関数を

$$F_{i,j}(x) = a(x - x_{i,j})^3 + b(x - x_{i,j})^2 + c(x - x_{i,j}) + f_{i,j}^{n} \tag{4-4}$$

$$a = \frac{f_{i+1,j}^{n} - 3f_{i,j}^{n} + 3f_{i-1,j}^{n} - f_{i-2,j}^{n}}{6\Delta x^3},\quad b = \frac{f_{i+1,j}^{n} - 2f_{i,j}^{n} + f_{i-1,j}^{n}}{2\Delta x^2},\quad c = \frac{2f_{i+1,j}^{n} + 3f_{i,j}^{n} - 6f_{i-1,j}^{n} + f_{i-2,j}^{n}}{6\Delta x} \tag{4-5}$$

$u < 0$の場合、$f_{i-1,j}^{n}$，$f_{i,j}^{n}$，$f_{i+1,j}^{n}$，$f_{i+2,j}^{n}$の 4 点を通る 3 次多項式の補間関数を

$$F_{i,j}(x) = a(x - x_{i,j})^3 + b(x - x_{i,j})^2 + c(x - x_{i,j}) + f_{i,j}^{n} \tag{4-6}$$

$$a = \frac{f_{i+2,j}^{n} - 3f_{i+1,j}^{n} + 3f_{i,j}^{n} - f_{i-1,j}^{n}}{6\Delta x^3},\quad b = \frac{f_{i+1,j}^{n} - 2f_{i,j}^{n} + f_{i-1,j}^{n}}{2\Delta x^2},\quad c = \frac{-f_{i+2,j}^{n} + 6f_{i+1,j}^{n} - 3f_{i,j}^{n} - 2f_{i-1,j}^{n}}{6\Delta x} \tag{4-7}$$

とすることができる。ここで、x方向の全ての格子間隔はΔxであるとしている。$n+1$ステップの値は

$$f_{i,j}^{n+1} = F_{i,j}^{n}(x_j - u\Delta t) = a(-u\Delta t)^3 + b(-u\Delta t)^2 + c(-u\Delta t) + f_{i,j}^{n} \tag{4-8}$$

で求めることができる。(4-5)式と(4-7)式から、点$(i,\ j)$で$f_{i,j}^{n+1}$を求めるために、x方向に最も遠い情報として 2 点離れた$f_{i+2,j}^{n}$と$f_{i-2,j}^{n}$を用いている。 同じように(4-3)式を解くと、y方向に最も遠い情報としては 2 点離れた$f_{i,j-2}^{n}$と$f_{i,j+2}^{n}$を参照する。従って、正味の計算領域の周囲に 2 点の幅の境界条件設定用の格子を配置する必要があり、格子間隔は$\Delta x = L_x/(N_x - 4)$，$\Delta y = L_y/(N_y - 4)$となる。$\Delta x = L_x/100$および$\Delta y = L_y/100$としたければ、

```
#define  NX  (100+4)
```

```
#define  NY  (100+4)
```

としておく必要がある。格子点配置と周期境界条件は、以下のようになる。計算点は白丸の格子点で黄色い格子点は境界条件を設定する必要がある。

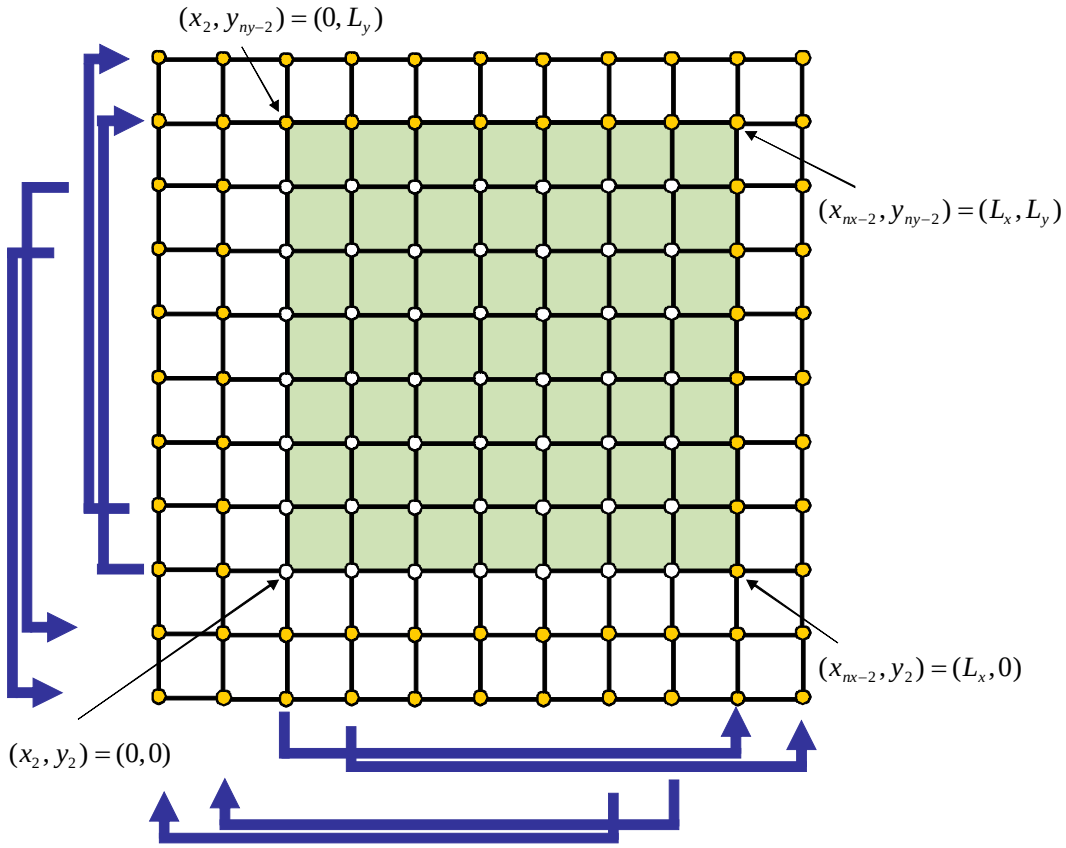


図 4-1　周期的境界条件

演習課題 2:

Cubic セミ・ラグランジアン法を用いて、(4-2)式の移流方程式を解くプログラムを作成する。

(1) #define NX と #define NY を修正する。

(2) Δx, Δy を修正する。

(3) Δt を CFL 条件に従って決める。

(4) x 方向の移流方程式と解く関数 x_advection2d()を作成する。

(5) 移流速度を $u = 1$, $v = 0$ または $u = -1$, $v = 0$ と設定する。

(6) 境界条件を設定する関数 boundary()を周期境界用に修正する。

(7) 計算終了の時刻を修正する。

(8) 動画で動き確認する。$t = 1.0$ の結果を確認する。

(9) 初期状態を $f(x, y) = \sin(2\pi x)\sin(2\pi y)$ と設定する。

(10) 誤差を一次風上差分のときにならって、関数 error_f()を作成する。

(11) NX および NY を変更し、誤差の変化を調べる。nx = 100+4, ny = 100+4 のときは、誤差は 4.947×10^{-5} となる。

(12) 初期状態を矩形状のプロファイルに変更し、$t=1.0$ の結果と初期状態を比較する。

(4-2)式の x 方向の移流方程式を解く関数としては、

```
void   x_advection2d
// ====================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
   double   f[][nx],       /* dependent variable                    */
   double   fn[][nx],      /* time-integrated dependent variable    */
   double   u,             /* advection velocity in the x-direction */
   double   dt,            /* time step interval                    */
   double   dx,            /* grid spacing in the x-direction       */
)
// --------------------------------------------------------------------
```

のような関数仕様とすることができる。また、main( )関数の時間発展の部分は、

```
do {
      x_advection2d(nx,ny,f,fn,1.0,dt,dx);
      boundary(nx,ny,fn);
      update(nx,ny,f,fn);

      time += dt;
} while(icnt++ < 9999 && time + 0.5*dt < 1.0);
```

のようになる。画面表示や BMP ファイル作成の部分などは、省略してある。

演習課題 3:

Cubic セミ・ラグランジアン法を用いて、(4-3)式の移流方程式を解くプログラムを作成する。

(1) y 方向の移流方程式と解く関数 y_advection2d()を作成する。

(2) 移流速度を $v=1$ または, $v=-1$ と設定する。

(3) 動画で動き確認する。$t=1.0$ の結果を確認する。

(4) 初期状態を $f(x,y)=\sin(2\pi x)\sin(2\pi y)$ と設定する。

(5) NX および NY を変更し、誤差の変化を error_f()で調べる。nx = 100+4, ny = 100+4 のときは、誤差は 4.947×10^{-5} となる。

(6) 初期状態を矩形上のプロファイルに変更し、$t=1.0$ の結果と初期状態を比較する。

(4-3)式の y 方向の移流方程式を解く関数としては、

```
void   y_advection2d
// ====================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
```

```
    double   f[][nx],       /* dependent variable                    */
    double   fn[][nx],      /* time-integrated dependent variable    */
    double   u,             /* advection velocity in the x-direction */
    double   dt,            /* time step interval                    */
    double   dy             /* grid spacing in the y-direction       */
)
// ------------------------------------------------------------------
```

のような関数仕様とすることができる。また、main( )関数の時間発展の部分は、

```
do {
      y_advection2d(nx,ny,f,fn,1.0,dt,dx);
      boundary(nx,ny,fn);
      update(nx,ny,f,fn);

      time += dt;
} while(icnt++ < 9999 && time + 0.5*dt < 1.0);
```

のようになる。

演習課題 4:

Cubic セミ・ラグランジアン法をベースに、Fractional Step 法を用いて(4-1)式の移流方程式を解くプログラムを作成する。

(1) Fractional Step 法により x 方向の移流方程式を解いて時間更新した後に、y 方向の移流方程式と解いて時間更新する。

```
do {
        x_advection2d(nx,ny,f,fn,1.0,dt,dx);
        boundary(nx,ny,fn);
        update(nx,ny,f,fn);

        y_advection2d(nx,ny,f,fn,1.0,dt,dy);
        boundary(nx,ny,fn);
        update(nx,ny,f,fn);

        time += dt;
} while(icnt++ < 9999 && time + 0.5*dt < 1.0);
```

(2) 移流速度を $u=1$, $v=1$ と設定する。

(3) 動画で動き確認する。$t=1.0$ の結果を確認する。

(4) 初期状態を $f(x,y)=\sin(2\pi x)\sin(2\pi y)$ と設定する。

(5) NX および NY を変更し、誤差の変化を error_f()で調べる。nx = 100+4, ny = 100+4 のときは、誤差は 9.894×10^{-5} となる。

(6) 初期状態を矩形状のプロファイルに変更し、$t=1.0$ の結果と初期状態を比較する。一次風上差分法を用いた場合と、プロファイルの形を比較する。

5. 二次元移流拡散方程式の計算

２次元移流拡散方程式

$$\frac{\partial f}{\partial t}+u\frac{\partial f}{\partial x}+v\frac{\partial f}{\partial y}=\kappa\left(\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}\right) \tag{5-1}$$

を解く。計算領域、初期値プロファイル、周期境界条件等は全て前の課題のときと同じである。

2 次元格子上の $(i,\ j)$ 点での移流方程式の計算は、時間ステップ間隔を Δt とし、時間微分項には前進差分、移流項の空間微分には 1 次精度風上差分、拡散項の 2 階微分には中心差分を適用する。

$$\frac{f_{i,j}^{n+1}-f_{i,j}^{n}}{\Delta t}+u\frac{f_{i,j}^{n}-f_{i-1,j}^{n}}{\Delta x}+v\frac{f_{i,j}^{n}-f_{i,j-1}^{n}}{\Delta y}=\kappa\left(\frac{f_{i+1,j}^{n}-2f_{i,j}^{n}+f_{i-1,j}^{n}}{\Delta x^2}+\frac{f_{i,j+1}^{n}-2f_{i,j}^{n}+f_{i,j-1}^{n}}{\Delta y^2}\right) \quad (u\geq 0,\ v\geq 0)$$

$$\frac{f_{i,j}^{n+1}-f_{i,j}^{n}}{\Delta t}+u\frac{f_{i+1,j}^{n}-f_{i,j}^{n}}{\Delta x}+v\frac{f_{i,j}^{n}-f_{i,j-1}^{n}}{\Delta y}=\kappa\left(\frac{f_{i+1,j}^{n}-2f_{i,j}^{n}+f_{i-1,j}^{n}}{\Delta x^2}+\frac{f_{i,j+1}^{n}-2f_{i,j}^{n}+f_{i,j-1}^{n}}{\Delta y^2}\right) \quad (u<0,\ v\geq 0)$$

$$\frac{f_{i,j}^{n+1}-f_{i,j}^{n}}{\Delta t}+u\frac{f_{i,j}^{n}-f_{i-1,j}^{n}}{\Delta x}+v\frac{f_{i,j+1}^{n}-f_{i,j}^{n}}{\Delta y}=\kappa\left(\frac{f_{i+1,j}^{n}-2f_{i,j}^{n}+f_{i-1,j}^{n}}{\Delta x^2}+\frac{f_{i,j+1}^{n}-2f_{i,j}^{n}+f_{i,j-1}^{n}}{\Delta y^2}\right) \quad (u\geq 0,\ v<0)$$

$$\frac{f_{i,j}^{n+1}-f_{i,j}^{n}}{\Delta t}+u\frac{f_{i+1,j}^{n}-f_{i,j}^{n}}{\Delta x}+v\frac{f_{i,j+1}^{n}-f_{i,j}^{n}}{\Delta y}=\kappa\left(\frac{f_{i+1,j}^{n}-2f_{i,j}^{n}+f_{i-1,j}^{n}}{\Delta x^2}+\frac{f_{i,j+1}^{n}-2f_{i,j}^{n}+f_{i,j-1}^{n}}{\Delta y^2}\right) \quad (u<0,\ v<0)$$

時間ステップ幅 Δt は、CFL 条件の

$$\Delta t\leq 0.2*MIN(\Delta x/u,\Delta y/v)$$

と、拡散方程式の数値安定性の条件

$$\Delta t\leq 0.1*MIN(\Delta x^2,\Delta y^2)/\kappa$$

を同時に Δt 満足（小さい方）するとする。

$0.01 \le \kappa \le 0.1$ の間で κ を適当に変化させて $t = 1.0$ のプロファイルを初期状態と比較し、拡散項の影響を確認せよ。
次に、(5-1)式 の時間積分に対して Fractional Step 法を用い、

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \tag{5-2}$$

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial y} = 0 \tag{5-3}$$

$$\frac{\partial f}{\partial t} = \kappa\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right) \tag{5-4}$$

に分割し、(5-2)式および(5-3)式に対しては、Cubic セミ・ラグランジアン手法で計算し、(5-4)式については、2 次元拡散方程式の際に使用した関数 diffusion2d( ) を用いることができる。ただし、配列のサイズが変更されているので for ループの範囲に注意する必要がある。

Fractional Step 法により x 方向の移流方程式を解いて時間更新した後に、y 方向の移流方程式を解き、さらに拡散方程式を解いて時間更新する。

```
do {
        x_advection2d(nx,ny,f,fn,1.0,dt,dx);
        boundary(nx,ny,fn);
        update(nx,ny,f,fn);

        y_advection2d(nx,ny,f,fn,1.0,dt,dy);
        boundary(nx,ny,fn);
        update(nx,ny,f,fn);

        diffusion2d(nx,ny,f,fn,kappa,dt,dy);
        boundary(nx,ny,fn);
        update(nx,ny,f,fn);

        time += dt;
} while(icnt++ < 9999 && time + 0.5*dt < 1.0);
```

演習課題 5:

$0.01 \le \kappa \le 0.1$ の間で κ を適当に変化させて $t = 1.0$ のプロファイルを初期状態と比較し、拡散項の影響を確認せよ。また、移流項に一次風上差分を用いた場合との比較を行え。

6. 二次元 Burgers 方程式の計算

流速$\boldsymbol{u}=(u,\ v)$に対する 2 次元 Burgers 方程式

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{6-1}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{6-2}$$

を計算領域$0\le x\le 1.0$，$0\le y\le 1.0$において周期境界条件で解く。粘性係数からくるレイノルズ数は 200 とする。初期条件は、非圧縮性流体の条件$\nabla\cdot\boldsymbol{u}=0$、つまり

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{6-3}$$

を満足するように設定する。$u_x(x,y)=\sin(k_x x)\sin(k_y y)$、$v_y(x,y)=-\sin(k_x x)\sin(k_y y)$とすれば、

$$\begin{aligned} u(x,y)&=-\frac{1}{k_x}\cos(k_x x)\sin(k_y y)\\ v(x,y)&=\frac{1}{k_y}\sin(k_x x)\cos(k_y y)\end{aligned} \tag{6-4}$$

(6-4)式 の$u,\ v$は(6-3)式を満足することが分る。さらに、(6-4)式は周期的にもなっている。

6.1 配列の宣言

移流項に対して 1 次風上差分を用い、2 次元 Burgers 方程式を解く。2 次元移流方程式の計算と同じ計算格子を用いると、従属変数に対して以下のように配列を宣言する。

```
#define  NX  (100+1)
#define  NY  (100+1)
#define  Re  200.0

int  main(int argc, char *argv[])
{
     int     nx = NX,     ny = NY,     icnt = 0;
     static  double    u[NY][NX],  un[NY][NX],  v[NY][NX],  vn[NY][NX],
                       rot[NY][NX],  div[NY][NX];


}
```

計算結果の渦度表示や$\nabla\cdot\boldsymbol{u}$の表示のために、rot[NY][NX] と div[NY][NX] も宣言しておく。

6.2 初期速度場の設定

初期値を設定するプログラムの一例は、

```
void   initial
// ==================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
   double   dx,            /* grid spacing in the x-direction       */
   double   dy,            /* grid spacing in the y-direction       */
   double   u[][nx],       /* velocity in the x-direction           */
   double   v[][nx],       /* velocity in the y-direction           */
)
// ------------------------------------------------------------------
{
     int     jx,   jy;
     double  x,    y,   kx = 2.0*M_PI,   ky = 2.0*M_PI;

     for(jy=0 ; jy < ny; jy++) {
         for(jx=0 ; jx < nx; jx++) {
             x = dx*(double)(jx - 1);
             y = dy*(double)(jy - 1);
             u[jy][jx] = - cos(kx*x)*sin(ky*y)/kx;
             v[jy][jx] = sin(kx*x)*cos(ky*y)/ky;

             x += 0.3;  y += 0.7;
             u[jy][jx] += - 0.6*cos(2.0*kx*x)*sin(2.0*ky*y)/kx;
             v[jy][jx] += 0.6*sin(2.0*kx*x)*cos(2.0*ky*y)/ky;
         }
     }
```

振幅・位相をずらした$k_x = k_y = 4\pi$の波も加えてある。

6.3　Burgers 方程式の計算

(6-1) および(6-2)式は、移流拡散方程式を解く際に作成した関数 advection_diffusion2d() を使うことができる。ただし、double u,　double v として一定速度を与えていた引数の部分を配列 double u[][nx],　double v[][nx] として関数内部も変更する。

```
void   advection_diffusion2d
// ==================================================================
 (
   int      nx,            /* x-dimension size                      */
   int      ny,            /* y-dimension size                      */
   double   f[][nx],       /* dependent variable                    */
   double   fn[][nx],      /* time-integrated dependent variable    */
   double   u[][nx],       /* advection velocity in the x-direction */
   double   v[][nx],       /* advection velocity in the y-direction */
   double   Re,            /* diffusion coefficient                 */
   double   dt,            /* time step interval                    */
   double   dx,            /* grid spacing in the x-direction       */
   double   dy             /* grid spacing in the y-direction       */
)
```

```
// ----------------------------------------------------------------------
```

6.4 時間発展 loop の作成

main 関数の時間発展ループは以下のようになる。

```
do {  fprintf(stderr,"time(%4d)=%7.5f¥n",icnt,time + dt);

      advection_diffusion2d(nx,ny,u,un,u,v,Re,dt,dx,dy);
      advection_diffusion2d(nx,ny,v,vn,u,v,Re,dt,dx,dy);
      boundary(nx,ny,un);      boundary(nx,ny,vn);
      update(nx,ny,u,un);      update(nx,ny,v,vn);

      time += dt;

      if(icnt % nout == 1) {
         sprintf(filename,"f%03d.bmp",icnt/nout);
         rotation(nx,ny,dx,dy,u,v,rot);    boundary(nx,ny,rot);
         bmp_r8(nx,ny,rot,4,3.0,-3.0,MESH_OFF,filename,
                "Rainbow.pal");
   }

} while(icnt++ < 9999 && time < 2.0);
```

となる。u, v のそれぞれに対して境界条件 boundary( ) および時間更新 update( ) が使われていることに注意する必要がある。これらは既に移流方程式、移流拡散方程式で作成したものである。

6.5 渦度の計算

計算結果に対して、流れの渦度ωで表示すると分り易い。渦度は、$\omega = \nabla \times \boldsymbol{u} = \dfrac{\partial v}{\partial x} - \dfrac{\partial u}{\partial y}$

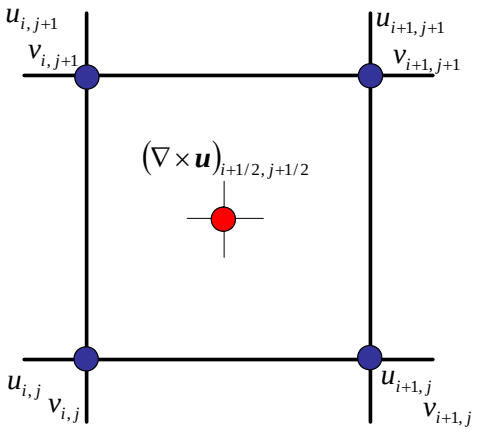


図 6-1　渦度の計算に必要な格子点

であり、渦度ωを速度 u, v の定義点の中点$(i+1/2, j+1/2)$で定義すると、

$$\omega_{i+1/2,j+1/2} = \frac{1}{2}\left(\frac{v_{i+1,j+1} - v_{i,j+1}}{\Delta x} + \frac{v_{i+1,j} - v_{i,j}}{\Delta x}\right) - \frac{1}{2}\left(\frac{u_{i+1,j+1} - u_{i+1,j}}{\Delta y} + \frac{u_{i,j+1} - u_{i,j}}{\Delta y}\right) \tag{6-5}$$

し、配列の index は整数でなければならないので、

$$(\nabla \cdot \boldsymbol{u})_{i+1.2,j+1/2} = div[j][i] \tag{6-6}$$

として使う。

渦度の計算精度は 2 次精度になる。もし、u, v と同じ格子点上で定義した場合は、中心差分を用いなければならず、

$$\omega_{i,j} = \frac{v_{i+1,j} - v_{i-1,j}}{2\Delta x} - \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} \tag{6-7}$$

渦度ωの値を格納する配列を rot[NY][NX]とすると、

```
void   rotation
// ==================================================================
 (
   int      nx,            /* x-dimension size                    */
   int      ny,            /* y-dimension size                    */
   double   dx,            /* grid spacing in the x-direction     */
   double   dy,            /* grid spacing in the y-direction     */
   double   u[][nx],       /* velocity in the x-direction         */
   double   v[][nx],       /* velocity in the y-direction         */
   double   rot[][nx]      /* vortex                              */
)
// ------------------------------------------------------------------
{
     int     i,    j;

     for(j=0 ; j < ny; j++) {
         for(i=0 ; i < nx; i++) {
             rot[j][i] = 0.5*(v[j+1][i+1] - v[j+1][i]
                              + v[j][i+1] - v[j][i])/dx
                       - 0.5*(u[j+1][i+1] - u[j][i+1]
                              + u[j+1][i] - u[j][i])/dy;
         }
     }
}
```

で求めることができる。

6.7 速度の divergence の計算

一方、初期状態で$\nabla\cdot\boldsymbol{u}=0$となるように速度を設定しているが、Burgers 方程式の時間発展を計算して行くと次第に満たされなくなる。それを確認するために、速度の発散$\nabla\cdot\boldsymbol{u}$を表示させる必要がある。渦度と同じように速度 u, v から 1/2 格子ずらした中点に定義すると、

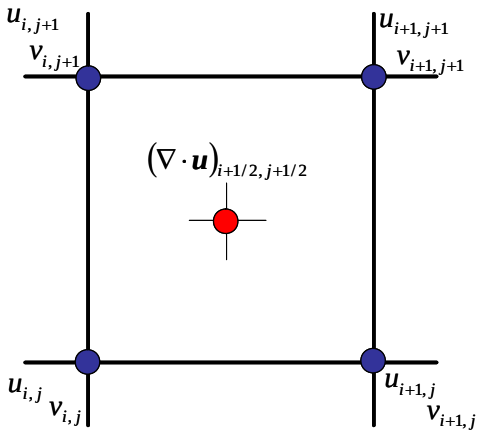


図 6-2　速度の divergence の計算に必要な格子点

$$
\begin{aligned}
(\nabla\cdot\boldsymbol{u})_{i+1/2,j+1/2} &= \frac{1}{2}\left(\frac{u_{i+1,j+1}-u_{i,j+1}}{\Delta x}+\frac{u_{i+1,j}-u_{i,j}}{\Delta x}\right) \\
&\quad -\frac{1}{2}\left(\frac{v_{i+1,j+1}-v_{i+1,j}}{\Delta y}+\frac{v_{i,j+1}-v_{i,j}}{\Delta y}\right)
\end{aligned}
\tag{6-8}
$$

により 2 次精度で速度の divergence を計算することができる。

$\nabla\cdot\boldsymbol{u}$ の値を格納する配列を div[NY][NX]とすると、

```
void   divergence
// ==================================================================
 (
   int      nx,            /* x-dimension size                    */
   int      ny,            /* y-dimension size                    */
   double   dx,            /* grid spacing in the x-direction     */
   double   dy,            /* grid spacing in the y-direction     */
   double   u[][nx],       /* velocity in the x-direction         */
   double   v[][nx],       /* velocity in the y-direction         */
   double   div[][nx]      /* vortex                              */
)
// ------------------------------------------------------------------
{
     int     i,    j;

     for(j=0 ; j < ny; j++) {
         for(i=0 ; i < nx; i++) {
             div[j][i] = 0.5*(u[j+1][i+1] - u[j+1][i]
                            + u[j][i+1] - u[j][i])/dx
```

```
                    - 0.5*(v[j+1][i+1] - v[j][i+1]
                        + v[j+1][i] - v[j][i])/dy;
            }
        }
    }
```

6.8 速度の divergence の計算

2 次元 Burgers 方程式を $t = 2.0$ まで解いて、渦度 ω および速度の発散 $\nabla \cdot \boldsymbol{u}$ の時間・空間変化を 連番 BMP ファイルで確認せよ。

6.9 Cubic セミ・ラグランジアン法による Burgers 方程式の計算

Fractional Step 法を用い、

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \tag{6-9}$$

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial y} = 0 \tag{6-10}$$

$$\frac{\partial f}{\partial t} = \kappa\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right) \tag{6-11}$$

に分割し時間積分させる。(6-9)式および(6-10)式に対しては、Cubic セミ・ラグランジアン法で計算し、(6-11) 式については、2 次元拡散方程式の際に使用した関数 diffusion2d() を用いることができる。ただし、配列のサイズが変更されているので for ループの範囲に注意する必要がある。

Fractional Step 法により x 方向の移流方程式を解いて時間更新した後に、y 方向の移流方程式を解き、さらに拡散方程式を解いて時間更新する。

```
do {
    x_advection2d(nx,ny,u,un,u,dt,dx);
    x_advection2d(nx,ny,v,vn,u,dt,dx);
    boundary(nx,ny,un); boundary(nx,ny,vn);
    update(nx,ny,u,un); update(nx,ny,v,vn);

    y_advection2d(nx,ny,u,un,v,dt,dy);
    y_advection2d(nx,ny,v,vn,v,dt,dy);
    boundary(nx,ny,un); boundary(nx,ny,vn);
    update(nx,ny,u,un); update(nx,ny,v,vn);

    diffusion2d(nx,ny,u,un,1.0/Re,dt,dx,dy);
```

```
        diffusion2d(nx,ny,v,vn,1.0/Re,dt,dx,dy);
        boundary(nx,ny,un); boundary(nx,ny,vn);
        update(nx,ny,u,un); update(nx,ny,v,vn);

        time += dt;
    } while(icnt++ < 9999 && time + 0.5*dt < 1.0);
```

演習課題 6:

Cubic セミ・ラグランジュ法を用いて 2 次元 Burgers 方程式を $t = 2.0$ まで解いて、渦度 ω および速度の発散 $\nabla \cdot \boldsymbol{u}$ の時間・空間変化を 連番 BMP ファイルで確認せよ。また、移流計算を一次風上差分で計算した結果と比較せよ。

7. Poisson 方程式の解法

MAC 法や SMAC 法で非圧縮性流体方程式を計算する過程で、非圧縮性の条件$\nabla \cdot \boldsymbol{u} = 0$を満足しなくなってしまった流れ場$\boldsymbol{u} = (u, v)$を修正するための圧力を求めるために、 Poisson 方程式を解く必要がある。

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = \left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} \right) / \Delta t \tag{7-1}$$

$\boldsymbol{u}^* = (u^*, v^*)$は Burgers 方程式で時間発展した直後の速度である。

ここでは、与えられた右辺項sに対して、次の Poisson 方程式を解くプログラムを作成する。

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = s \tag{7-2}$$

２回微分項に対して、2 次精度の中心差分法を用いると、

$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} = s_{i,j} \tag{7-3}$$

となる。

(7-3)式を反復法の 1 つである SOR 法を用いて解く。初期値$p_{i,j}^{(0)}$からスタートし、n 回の反復計算で$p_{i,j}^{(n)}$まで計算できたとすると、$p_{i,j}^{(n+1)}$は

$$p_{i,j}^{(n+1)} = (1-\omega) p_{i,j}^{(n)} + \omega \left(\frac{p_{i+1,j}^{(n)} + p_{i-1,j}^{(n+1)}}{\Delta x^2} + \frac{p_{i,j+1}^{(n)} + p_{i,j-1}^{(n+1)}}{\Delta y^2} - s_{i,j} \right) \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} \tag{7-4}$$

で計算できる。ここでωは加速係数（または緩和係数）であり、Poisson 方程式に対しては$\omega = 1.8$付近の値で最も早く反復計算が収束することが知られている。

配列の i と j の index を for ループを

```
for(j = 1; j < ny-1; j++) {
    for(i = 1; i < nx-1; i++) {
        (4)式の計算
    }
}
```

とすることで、$p_{i,j}^{(n+1)}$を計算する際には$p_{i-1,j}^{(n+1)}$および$p_{i,j-1}^{(n+1)}$は計算されているため、$p_{i,j}^{(n+1)}$と$p_{i,j}^{(n)}$の 2 つの配列を用意する必要はない。(7-4)式が解けたかどうかの判定は、全ての格子点で$\left| p_{i,j}^{(n+1)} - p_{i,j}^{(n)} \right| < \varepsilon$であれば、反復計算は収束したと判定される。ここで$\varepsilon$は小さい数で例えば$\varepsilon = 10^{-8}$などとする。（問題に依存する。）

```
double   sor
// ==========================================================================
 (
   int      nx,            /* grid number in the x-direction        */
   int      ny,            /* grid number in the y-direction        */
   double   dx,            /* grid spacing in the x-direction       */
   double   dy,            /* grid spacing in the y-direction       */
   double   f[][nx],       /* dependent variable                    */
   double   s[][nx],       /* source term                           */
   double   omega          /* relaxation parameter                  */
)
// --------------------------------------------------------------------------
{
     int      i,    j;
     double   fn,   err = 0.0;

     for(j = 1; j < ny-1; j++) {
         for(i = 1; i < nx-1; i++) {
             fn = (  (f[j][i+1] + f[j][i-1])/(dx*dx)
                   + (f[j+1][i] + f[j-1][i])/(dy*dy) - s[j][i] )
                *0.5*dx*dx*dy*dy/(dx*dx + dy*dy);

             err = MAX2(fabs(fn - f[j][i]), err);

             f[j][i] = (1.0 - omega)*f[j][i] + omega*fn;
         }
     }

     return err;
}
```

関数 sor( ) は、(7-4)式を計算領域全体に対して１回の反復計算を行い、収束誤差を戻り値として返す。Poisson 方程式の反復計算を行う関数 poisson2d( ) の中では、

```
while(icnt++ < imax) {
        if(eps > (err = sor(nx,ny,dx,dy,f,s,omega))) return icnt;
}
return icnt;
```

が計算され、反復回数の最大値として imax = 99999 が設定され、icnt はループ・カウンターである。

Poisson 方程式の計算の検証として、$s(x,y) = -(k_x^2 + k_y^2)\sin(k_x x)\sin(k_y y)$ とすれば(7-2)式の解析解は $p(x,y) = \sin(k_x x)\sin(k_y y)$ となる。$k_x = k_y = 2\pi$ と取れば、計算領域 $0 \le x \le 1$, $0 \le y \le 1$ に対して周囲の境界条件は $p = 0$ と設定でき、初期も計算領域の全ての点で $p = 0$ と設定している。サンプル・コードでは、Poisson 方程式を SOR 法で解き、解析解との誤差を表示している。

演習課題:7

（１） サンプル・コードをコンパイル・実行し、格子点数が NX = 64+1, NY=64+1 のときの計算精度が 3.356×10^{-4} であることを確認し、格子点数が NX = 128+1, NY=128+1 および格子点数が NX = 256+1, NY=256+1 のときの計算精度を確認せよ。

（２） サンプル・コードは $\Delta x \neq \Delta y$ の場合も計算できるようになっているが、設定では $\Delta x = \Delta y$ になっているので、そのような場合は(7-4)式は大幅に簡単になる。関数 sor() についても、$\Delta x = \Delta y$ を仮定した場合の計算に修正し、結果が変わらないことと計算速度が向上することを time ./run で確認せよ。

（３） Poisson 方程式に対する差分式(3)の計算誤差を以下の残差

$$R = \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} - s_{i,j} \tag{7-5}$$

として求めることができる。この残差式を使って、Poisson 方程式の反復計算の過程で残差の最大値がどのように減少するかを反復計算の 50 回ごとに出力して確認せよ。一例として、残差の絶対値の最大値を返す関数の中は、

```
int      i,    j;
double   res,  rmax = 0.0;

for(j = 1; j < ny-1; j++) {
    for(i = 1; i < nx-1; i++) {
        res = (f[j][i+1] - 2.0*f[j][i] + f[j][i-1])/(dx*dx)
            + (f[j+1][i] - 2.0*f[j][i] + f[j-1][i])/(dy*dy)
            - s[j][i];

        rmax = MAX2( fabs(res), rmax);
    }
}

return rmax;
```

とすることができる。

8. 非圧縮性流体の計算

規格化された 2 次元非圧縮性 Navier-Stokes 方程式は以下のように表わされる。

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{8-1}$$

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{\partial p}{\partial x}+\frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{8-2}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{\partial p}{\partial y}+\frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{8-3}$$

$\boldsymbol{u}=(u,v)$ は、流速ベクトルで、u と v はその x 方向および y 方向成分である。 p は圧力、Re はレイノルズ数（無次元数）であり、 (8-1)式は連続方程式から導かれる非圧縮性の条件である。

非圧縮性流体の非定常計算を行うために、ここでは、SMAC 法を用い、初期条件からスタートし、(8-1)～(8-3)式を満たすように u， v， p の時間ステップを進める。

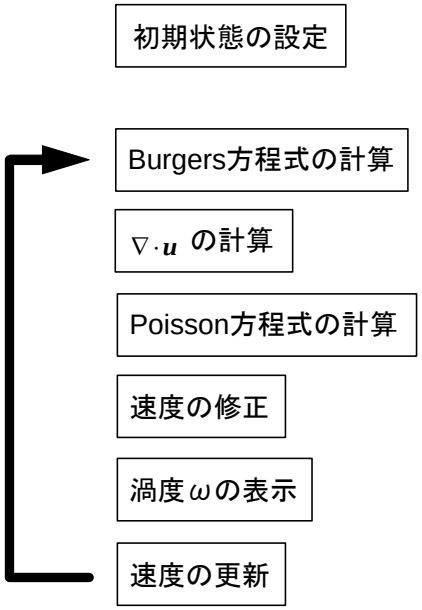


図 8-1 非圧縮性流体計算のフロー

8.1 速度・圧力の変数配置

8.1.1 コロケート格子

非圧縮性流体計算で現れる速度 u，v と圧力 p を格子上のどこで定義するかによって計算精度や安定性が変わり、主に 3 種類の変数配置が使われる。全て同じ格子点上で定義する方法はコロケート格子配置と呼ばれ、計算精度は高いが速度と圧力のカップリングが悪く、計算が不安定になり易い。Poisson 方程式の右辺項となる $\nabla\cdot\boldsymbol{u}$ は圧力の定義される点で求められなかれ

ばならないので、

$$\left(\nabla\cdot\boldsymbol{u}\right)_{i,j}=\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}+\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y} \tag{8-4}$$

となり、速度の参照点が $i-1$ から $i+1$ および $j-1$ から $j+1$ の 3 格子点にまたがっていることがわかる。一方、Poisson 方程式で求めた圧力で速度を修正する際も、$\boldsymbol{u}^{n+1}=\boldsymbol{u}^{*}-\nabla p\,\Delta t$ の離散化式は、

$$\begin{aligned} u_{i,j}^{n+1}&=u_{i,j}^{*}-\frac{p_{i+1,j}-p_{i-1,j}}{2\Delta x}\Delta t\\ v_{i,j}^{n+1}&=v_{i,j}^{*}-\frac{p_{i,j+1}-p_{i,j-1}}{2\Delta y}\Delta t \end{aligned} \tag{8-5}$$

となり、やはり圧力勾配の参照点が x, y の各方向に 3 点にまたがってしまう。

8.1.2 Arakawa-C 型スタッガード格子

速度と圧力を以下のように交互に定義する配置方法を（Arakawa-C 型）スタッガード格子と呼び、標準的なスタッガード格子配置である。

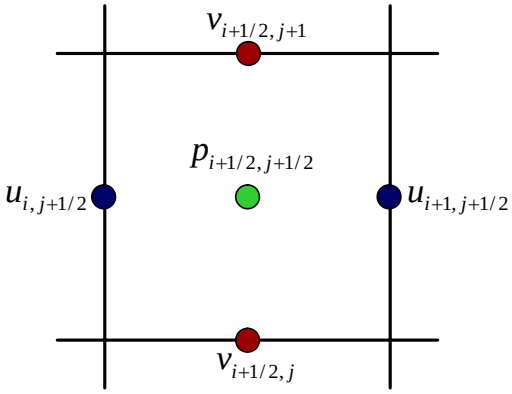


図 8-2　Arakawa-C 型スタッガード格子配置

Poisson 方程式の右辺項となる $\nabla\cdot\boldsymbol{u}$ は圧力の定義される $(i+1/2,\ j+1/2)$ 点で求められ、

$$\left(\nabla\cdot\boldsymbol{u}\right)_{i+1/2,j+1/2}=\frac{u_{i+1,j+1/2}-u_{i,j+1/2}}{\Delta x}+\frac{v_{i+1/2,j+1}-v_{i+1/2,j}}{\Delta y} \tag{8-6}$$

のように x, y の各方向に最小の 2 点の参照となっている。Poisson 方程式で求めた圧力で速度を修正する際も、

$$\begin{aligned} u_{i,j+1/2}^{n+1}&=u_{i,j+1/2}^{*}-\frac{p_{i+1/2,j+1/2}-p_{i-1/2,j+1/2}}{\Delta x}\Delta t\\ v_{i+1/2,j}^{n+1}&=v_{i+1/2,j}^{*}-\frac{p_{i+1/2,j+1/2}-p_{i+1/2,j-1/2}}{\Delta y}\Delta t \end{aligned} \tag{8-7}$$

圧力勾配の参照が 2 点の差となっていて間に格子点が存在しないため、カップリングが良く、安定な計算できる。さらに空間差分精度はΔx^2およびΔy^2になっている。

8.1.3 Arakawa-B 型スタッガード格子

速度と圧力を以下のように定義する方法を（Arakawa-B 型）スタッガード格子と呼ぶ。

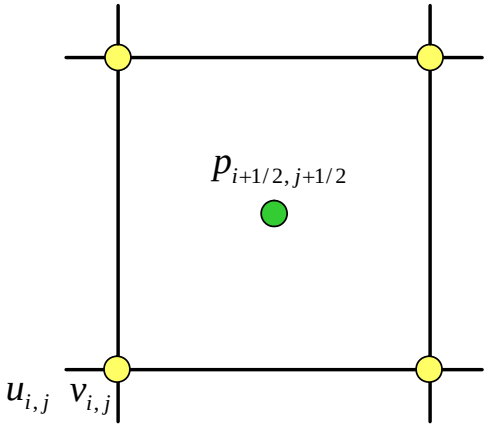


図 8-3　Arakawa-B 型スタッガード格子配置

圧力の定義点での$\nabla \cdot \boldsymbol{u}$は、

$$
\begin{aligned}
(\nabla \cdot \boldsymbol{u})_{i+1/2,j+1/2} &= \frac{1}{2}\left(\frac{u_{i+1,j+1}-u_{i,j+1}}{\Delta x}+\frac{u_{i+1,j}-u_{i,j}}{\Delta x}\right) \\
&+\frac{1}{2}\left(\frac{v_{i+1,j+1}-v_{i+1,j}}{\Delta y}+\frac{v_{i,j+1}-v_{i,j}}{\Delta y}\right)
\end{aligned}
\tag{8-8}
$$

となり一見参照点が広がっているように見えるが、x, y の各方向の 1 格子以内に収まっている。Poisson 方程式で求めた圧力で速度を修正は

$$
\begin{aligned}
u_{i,j}^{n+1} &= u_{i,j}^{*}-\frac{1}{2}\left(\frac{p_{i+1/2,j+1/2}-p_{i-1/2,j+1/2}}{\Delta x}+\frac{p_{i+1/2,j-1/2}-p_{i-1/2,j-1/2}}{\Delta x}\right)\Delta t \\
v_{i,j}^{n+1} &= v_{i,j}^{*}-\frac{1}{2}\left(\frac{p_{i+1/2,j+1/2}-p_{i+1/2,j-1/2}}{\Delta y}+\frac{p_{i-1/2,j+1/2}-p_{i-1/2,j-1/2}}{\Delta y}\right)\Delta t
\end{aligned}
\tag{8-9}
$$

となり、やはり圧力点の参照は x, y の各方向の 1 格子以内に収まっている。

ここでは、速度に対する物体境界条件の設定が容易な**Arakawa-B 型スタッガード格子**を用いることにする。以前作成した Burgers 方程式を計算するための関数をそのまま使うことができる点もメリットがある。速度成分$u_{i,j}$および$v_{i,j}$はそのまま配列$u[j][i]$と$v[j][i]$に対応させて計算すれば良いが、圧力の定義点は 1/2 格子ずれていて index が整数ではないので、$p_{i+1/2,j+1/2} \rightarrow p[j][i]$ と対応付けることにする。

8.2 物体境界条件の設定

物体表面で流体の速度は物体の速度と一致していなければならない。この速度境界条件に対する物理的解釈は他書に譲るとして、流れの中に静止した物体がある場合に物体表面での速度はゼロである。これを no slip 条件という。この $u=0$, $v=0$ の条件が物体表面で常に成り立つことを(8-2)式, (8-3)式 に代入すると、レイノルズ数が大きい場合には物体表面の法線方向に対して圧力勾配がゼロという条件が導き出される。

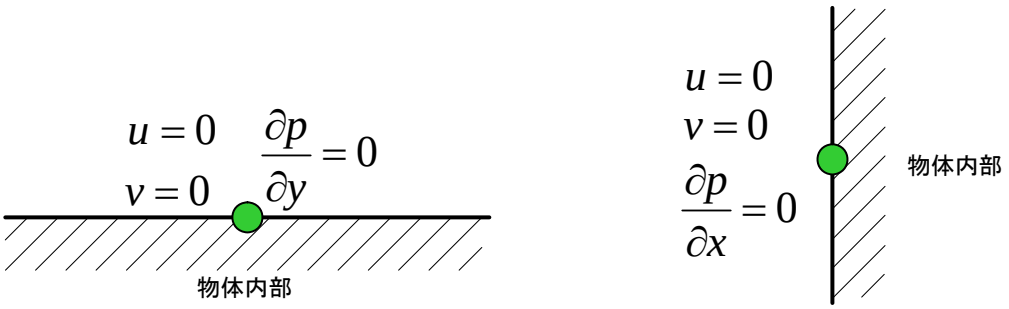


図 8-4　物体表面の速度、圧力の境界条件

実用的には Cartesian 格子（直交格子）を用いず、格子の形状を物体の表面形状に合わせて座標変換する境界適合座標系が用いられることが多いが、ここでは矩形状の物体などのように物体表面が x 方向または y 方向に一致しているような物体のみを扱うことにする。

物体内部は流体ではないので計算する必要がなく、計算領域の中で物体内部の格子、物体表面上の格子、流体内部の格子を識別するための配列 iw[NY][NX] を導入する。図 8-5 に計算領域に矩形状の物体が存在するとき、格子点での識別（物体フラグ）の仕方と境界条件を示す。○印は速度 u, v が定義される位置で、□は圧力が定義される位置である。物体表面を速度の定義点に一致させると、速度の境界条件の設定が非常に容易になる。白い○は Navier-Stokes 方程式で流体計算を行う格子で iw=0、赤い●は物体表面で iw=1、●は物体内部を表し iw = 2 などと設定する。●上では $u=0, v=0$ である（例えば図 8-5 の 1 番格子）。Cubic セミ・ラグランジアン法を移流計算に用いる場合には、物体内部の点を参照する可能性があり、そのために●点でも $u=0, v=0$ としておく。白い□印は圧力を Poisson 方程式で計算する格子であり、緑の■は物体内部であり圧力を計算しない。ただし、圧力を図 8-5 の 2 番格子で計算する際には右隣りの 4 番格子を参照するが、この格子は物体内部にある。2 番格子と 4 番格子の中間に物体表面があり、そこで x 方向の圧力勾配がゼロであるので、

$$\frac{\partial p}{\partial x}=\frac{p_{i+1/2,j+1/2}-p_{i-1/2,j+1/2}}{\Delta x}=0 \tag{8-10}$$

から 4 番格子の圧力に 2 番格子の圧力の値をコピーすればよいことが分かる。同じように、3 番格子を計算する際に 4 番格子が参照される。3 番と 4 番格子の間に物体境界があり、そこで y 方向の圧力勾配がゼロであるため、3 番格子を計算する直前に 4 番格子に 3 番格子の圧力値をコピーすれば良い。

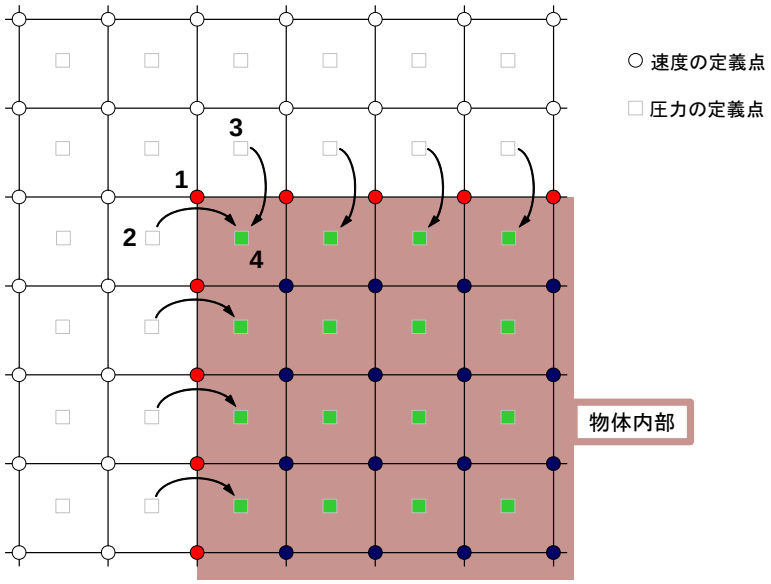


図 8-5　物体表面の速度、圧力の境界条件

8.3 計算領域の境界条件

物体まわりの非圧縮性流れの数値シミュレーションを行うには、物体を囲んだある程度広い範囲の領域の流れを計算する必要がある。この計算領域の境界に流入境界条件と流出境界条件を設定する必要がある。ここでは、$u=1$，$v=0$ 流れが左から流入し右から流出する設定とする。流入境界では $u=1$，$v=0$，$p=0$ を終始固定して与え続ける。**図 8-6** の計算領域の左端および上下端が青色で示され、流入（固定）境界条件であるとする。一方、右端は緑色で示した流出境界条件であり、ここでは最も単純な

$$\frac{\partial u}{\partial x}=0,\quad \frac{\partial v}{\partial x}=0,\quad \frac{\partial p}{\partial x}=0 \tag{8-11}$$

を与える。

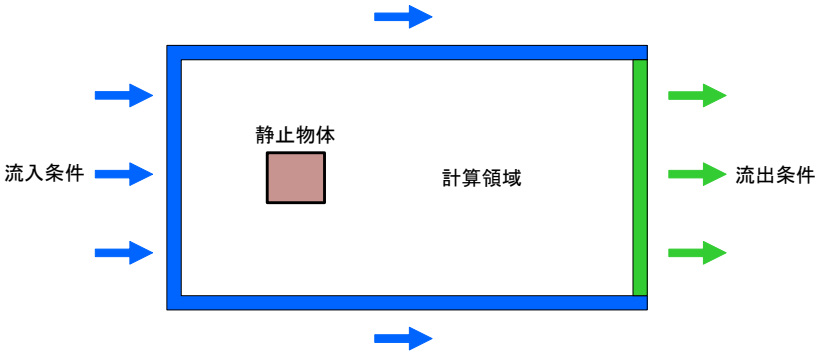


図 8-6　計算領域の流入・流出条件

8.4 格子配置と境界条件

移流計算が上下・左右に１点の参照だけであるので、境界条件として１格子与えれば良い。計算領域を$0 \le x \le L_x$， $0 \le y \le L_y$に設定し、格子間隔を$\Delta x = L_x / 100$および$\Delta y = L_y / 200$と設定した場合、速度を格納する配列は NX=100+1, NY=200+1 で static double u[NY][NX], v[NY][NX]; となる。**図 8-7** に計算領域と速度の境界条件を示す。

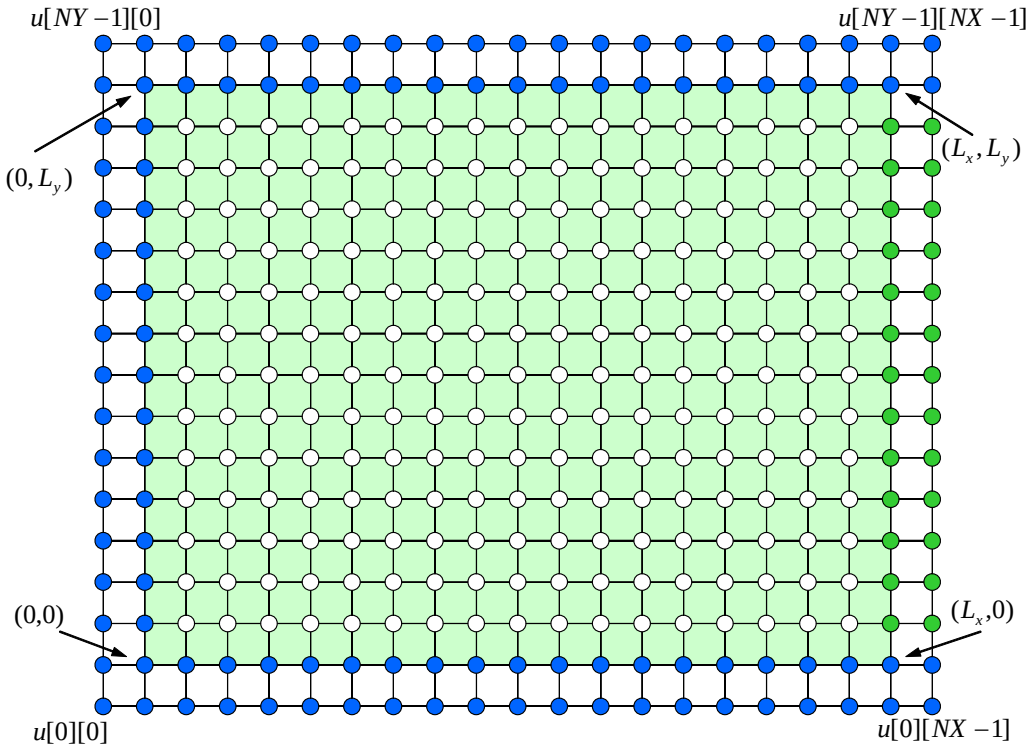


図 8-7　計算領域の速度の境界条件

速度 u, v ともに同じ点で定義され、全ての青い●点で流入（固定）境界条件$u = 1,\ v = 0$が課される。白い○点では Navier-Stokes 方程式で流体計算を行う。緑●点は流出境界条件として、左側の最近接の○点の値をコピーする。

```
for (j = 2; j < ny - 2; j++) {
   u[j][nx - 1] = u[j][nx - 3];   u[j][nx - 2] = u[j][nx - 3];
   v[j][nx - 1] = v[j][nx - 3];   v[j][nx - 2] = v[j][nx - 3];
}
```

計算領域の圧力境界条件を**図 8-8** に示す。灰色の■は全く使われない点であるが、u や v の配列の index と整合するために p[NY][NX]として同じサイズで確保されている。全ての青い■点には流入（固定）条件$p = 0$を課す。白い□では Poisson 方程式が解かれ、緑■点は流出境界条件

```
for (j = 1; j < ny - 2; j++)   p[j][nx - 2] = p[j][nx - 3];
```

を設定する。

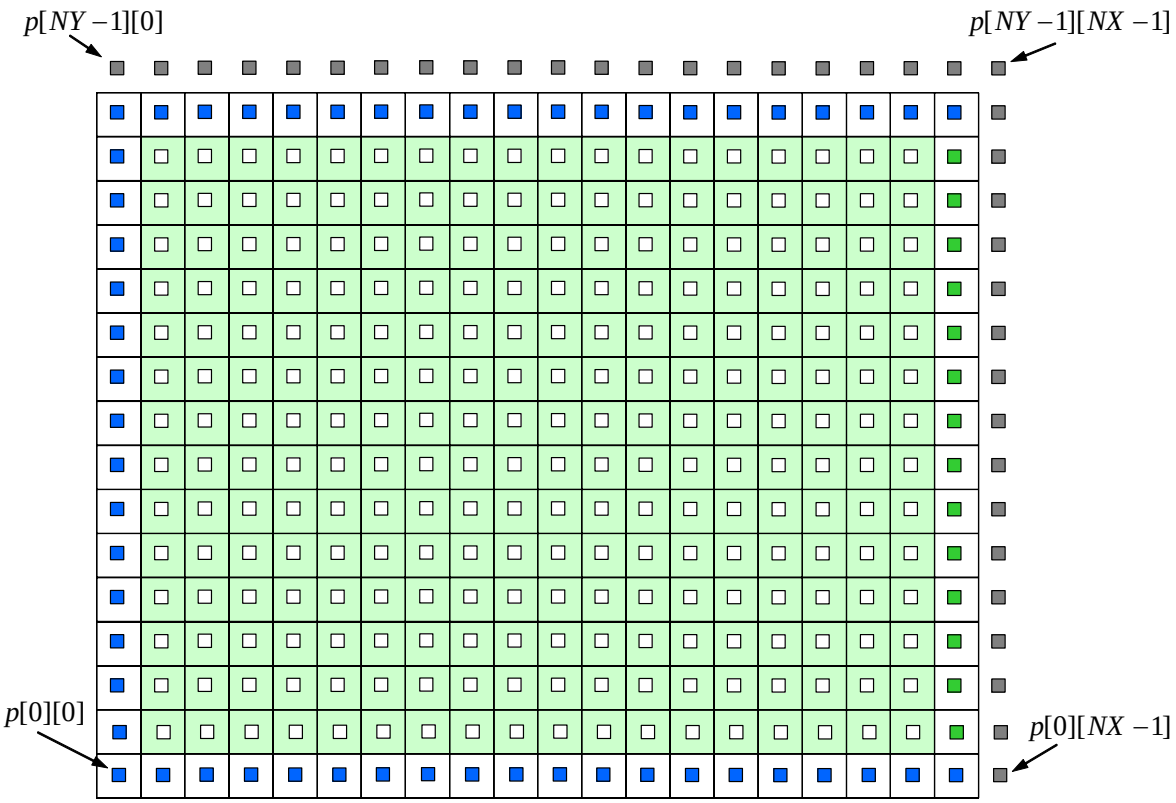
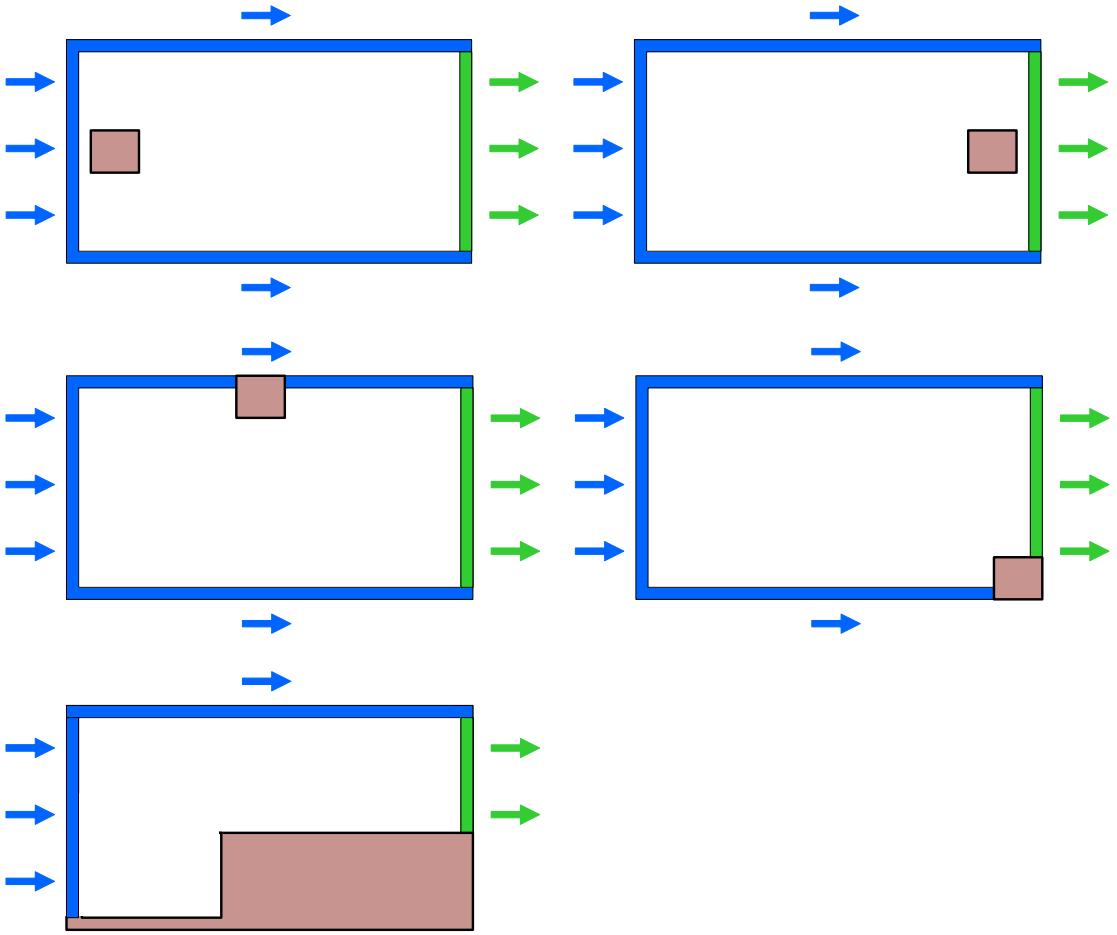


図 8-8　計算領域の圧力境界条件

8.5 計算領域の中の物体の配置

流れの中に静止した物体があると、流れが塞き止められるため流れは大きく変わる。流れに対する計算領域は物体を含みさえすればどのように取ってもよいのではなく、流入や流出境界の近傍に物体があると、適切な境界条件を与えることはできない。物体によって乱された流れの影響が境界に到達しなことが理想であるが、亜音速（音速より遅い）流れでは影響が必ず境界にまで到達してしまう。境界条件はできるだけ物体から離れて設定することが望ましいが、計算領域が増えるため計算コストが増えるため妥協点を探すしか方法がない。**図 8-9** は計算領域中に物体を配置した例であり、良い例と悪い例を示す。青い領域は流入境界であり、緑色が流出境界である。物体は茶色で示してある。

【悪い物体配置の例】



【良い物体配置の例】

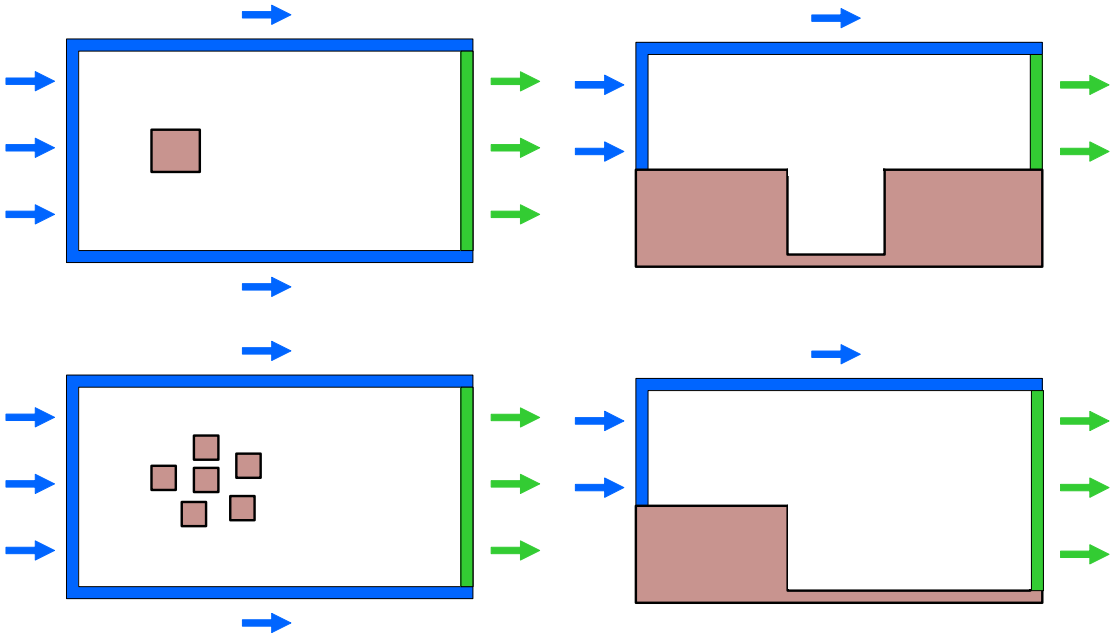


図 8-9　計算中の物体配置

8.6 Main( ) 関数の計算フロー

main( ) 関数では、最初に速度、圧力などの必要な配列を全て宣言する。iw[NY][NX]はその格子点が流体計算をすべき点か壁面上か壁内部（または物体）の点であるかを識別するための配列である。ip[NY][NX]は iw[NY][NX]に従って圧力の計算点を識別する。

```
set_wall(nx,ny,dx,dy,iw,ip);
```

で設定する。識別フラグに従って、速度の初期値を

```
initial(nx,ny,dx,dy,u,v,p,iw);
```

で設定する。

時間発展させる loop 内は、移流項を一次風上で解く場合は、

```
advection_diffusion2d(nx,ny,u,un,u,v,Re,dt,dx,dy,iw);
advection_diffusion2d(nx,ny,v,vn,u,v,Re,dt,dx,dy,iw);

boundary(nx,ny,un);          boundary(nx,ny,vn);
update(nx,ny,u,un,iw);       update(nx,ny,v,vn,iw);

div_udt(nx,ny,dt,dx,dy,u,v,s,iw);

it = poisson2d(nx,ny,dx,dy,p,s,1.8,1.0e-06,ip);

correction(nx,ny,dt,dx,dy,u,v,p,iw);
boundary(nx,ny,un);          boundary(nx,ny,vn);

time += dt;
```

プログラム 8-1　移流項を一次風上差分で計算する場合の非圧縮性流体計算

になる。これまでに作成した関数 advection_diffusion2d( ), boundary( ), update( ), poisson2d( ) が使われていて、違いは壁識別フラグ iw[][] と ip[][] が付け加えられているだけで、本質は全く変わっていない。div_udt( ) は、u, v から$\nabla\cdot\boldsymbol{u}$を求めた後に$\Delta t$で割り算している点が div_u( )からの変更点である。correction( ) は、速度が$\nabla\cdot\boldsymbol{u}=0$を満たすように(8-9)式に従って Poisson 方程式で求められた圧力を使って、

```
for(j=2 ; j < ny-2; j++) {
    for(i=2 ; i < nx-2; i++) {
        if(iw[j][i] > 0) continue;

        u[j][i] += - 0.5*(p[j][i]   - p[j][i-1]
                        + p[j-1][i] - p[j-1][i-1])/dx*dt;

        v[j][i] += - 0.5*(p[j][i]   - p[j-1][i]
                        + p[j][i-1] - p[j-1][i-1])/dy*dt;
    }
}
```

プログラム 8-2　非圧縮性条件を満たすように速度を修正する計算部分

と計算している。Boundary( ) では、これまでの周期的境界条件を変更し、計算領域の右側境界から流れが放出されるように境界条件が設定されている。

一方、Fractional Step 法を用い、移流項を Cubic セミ・ラグランジュ法で計算する場合は、

```
x_advection2d(nx,ny,u,un,u,dt,dx,iw);
x_advection2d(nx,ny,v,vn,u,dt,dx,iw);
boundary(nx,ny,un);          boundary(nx,ny,vn);
update(nx,ny,u,un,iw);       update(nx,ny,v,vn,iw);

y_advection2d(nx,ny,u,un,v,dt,dy,iw);
y_advection2d(nx,ny,v,vn,v,dt,dy,iw);
boundary(nx,ny,un);          boundary(nx,ny,vn);
update(nx,ny,u,un,iw);       update(nx,ny,v,vn,iw);

diffusion2d(nx,ny,u,un,1.0/Re,dt,dx,dy,iw);
diffusion2d(nx,ny,v,vn,1.0/Re,dt,dx,dy,iw);
boundary(nx,ny,un);          boundary(nx,ny,vn);
update(nx,ny,u,un,iw);       update(nx,ny,v,vn,iw);

div_udt(nx,ny,dt,dx,dy,u,v,s,iw);

it = poisson2d(nx,ny,dx,dy,p,s,1.8,1.0e-06,ip);

correction(nx,ny,dt,dx,dy,u,v,p,iw);
boundary(nx,ny,un);          boundary(nx,ny,vn);

time += dt;
```

プログラム 8-3　移流項を Cubic セミ・ラグランジアン法で計算する非圧縮性流体計算

となる。div_udt( ) 以降は一次風上差分法のプログラムと同じである。

初期設定は initial( ) の中で、u = 0.98, v = 0.02 の一様な速度が設定されている。僅かに v = 0.02 としたのは、物体後方にできる渦を非対称にしてカルマン渦を早く発生させるためである。

演習課題:8

（1） Poisson 方程式で求められた圧力を使って、速度が $\nabla \cdot \boldsymbol{u} = 0$ を満たすように修正する関数 correction( ) を**プログラム 8-2** に従って完成させよ。

（2） main( )関数のなかの時間発展 loop 内で移流項と拡散項を計算する部分を、一次風上差分法にするか、Cubic セミ・ラグランジアン法にするかに従って、プログラム 1 または**プログラム 8-3** のように完成させよ。

（3） 物体フラグを設定する set_wall( ) の中で、計算領域中に物体を配置せよ。例えば、矩形状の単一物体を置く場合は、

```
/****************** wall (object) setting ******************/
     for(j=0 ; j < ny; j++) {
         for(i=0 ; i < nx; i++) {
             iw[j][i] = 0;

             if(nx/5-5 < i && i < nx/5+5 &&
                ny/2-5  < j && j < ny/2+5)    iw[j][i] = 2;
         }
     }
/*************************************************************/
```

に設定する。各自、自由に物体を配置して良い。物体が思い通りに配置されたかは、渦を表示する際に確認することができる。物体は、**図 8-9** の説明に基づき適正に配置しなければならない。

（４） 非圧縮性流体計算を実行し、様々な流れが計算されることを確認せよ。単一の矩形物体（角柱）を配置した場合、後方にはカルマン渦が発生する。

（５） 渦の発生がレイノルズ数に従ってどのように変化するか調べよ。

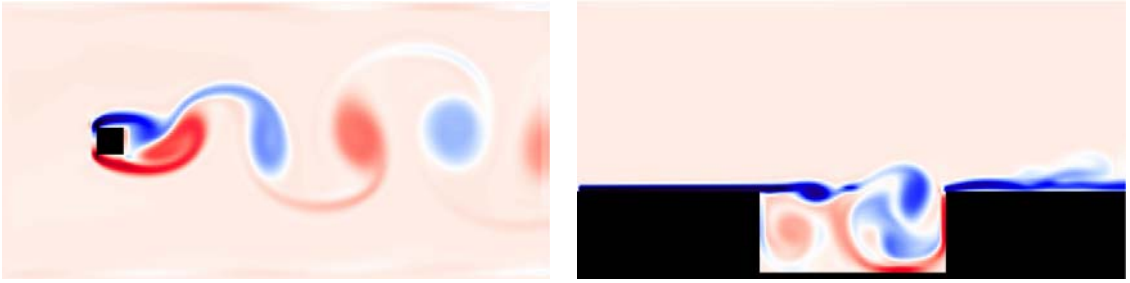


図 8-10 非圧縮性流体計算の例