講座
パーソナルコンピュータで行うプラズマ解析

3．理想気体のモンテカルロシミュレーション

松　田　七　美　男
（東京電機大学）

Monte Carlo Simulation of Ideal Gas

MATSUDA Namio
Faculty of Engineering, Tokyo Denki University, Tokyo 101-8457, Japan
(Received 13 March 2000)

Abstract
Molecular gas transmission behavior in a cylindrical pipe is calculated using Monte Carlo methods. Visualization of the motion of molecular gas in a pipe and gas impinging on unit area is performed using an X Window System under a UNIX clone operating system of Linux.

Keywords :
molecluar gas transimission probability, conductance to gas flow, science visualization, Monte Carlo simulation, PC-UNIX, Linux

3.1　はじめに

モンテカルロ法で中性気体分子がパイプを通過する様子をシミュレートし，気体分子の通過確率 K を算出する方法の原理と実際の手順を述べる．内容に関して言えば，電離気体の流れの基礎として希薄中性気体分子の流れを位置づけることが可能かもしれない．本稿のもう一つの狙いは，一昔前のスーパーコンピュータ並みの演算性能を持つパーソナルコンピュータで，数値計算を楽しむ術を紹介することにある．UNIX 互換のフリーな OS，LinuxやFreeBSDが大学や研究機関で急速に広まっている[1]．多くはインターネットのサーバ用の安定な OS としてであるが，UNIX が提供する本来の素晴らしいプログラミング環境を GNU C や X Window System でのアニメーションの簡単な例とともに紹介する．

3.2　コンダクタンスの計算

3.2.1　コンダクタンスと通過確率

圧力 p_1，p_2 の 2 つの空間を導管で接続すると，気体分子に正味の流れが生じ，定常流量 Q は圧力差と比例関係

$$Q = C\Delta p = C(p_1 - p_2) \tag{1}$$

にある．この比例定数をその導管のコンダクタンスと呼び，真空装置を構成する部品の基本的な物理量となっている．これは気体分子からみれば，一方の口から導管に入って，他方の口から出ていく通過確率 K と捕らえることができる．開口自身が持つコンダクタンス（無限空間から開口部に入る確率）を C_0 とすると，導管全体のコンダクタンスは

$$C = KC_0 \tag{2}$$

と表され，流れの向きに依らない導管固有の定数となる．

特に分子流領域では，空間における分子同士の衝突による散乱を無視できるので，壁での散乱のみによって K が決定される．この場合，壁での気体分子の散乱がどのような角度分布を持つかが重要となる．

3.2.2　気体分子放出の余弦則

気体分子の散乱は多少の例外はあるものの，簡単な分布に従うことが知られている．すなわち壁面からの放出

author's e-mail : matuda@film.s.dendai.ac.jp

気体分子の角度分布は（入射時の履歴に無関係に）一般に余弦則に従う．これは，気体分子が壁表面にしばらく滞在し壁と熱的に十分馴染んだ後に放出されると考えれば自然な事柄である．すなわち，気体放出数 dn は

$$\mathrm{d}n \propto \cos\theta \mathrm{d}\omega \tag{3}$$

と表される．ところで球座標系では，$\mathrm{d}\omega = \sin\theta \mathrm{d}\theta \mathrm{d}\phi$ とするのが一般的であるから，シミュレーションする上で発生させる分布は

$$f(\theta) = \sin\theta\cos\theta \tag{4}$$

に比例したものでなければならない．この分布は逆変換法により発生させることができる．逆変換法とは，与えられた分布関数にしたがう乱数 x を以下の方法により求めるものである．

$$x = F^{-1}(u),\quad F(y) = \int_{\text{下限}}^{y} f(\tau)\mathrm{d}\tau \tag{5}$$

ここに u は区間(0,1)における一様乱数である．$f(\tau) = \sin\tau\cos\tau$ とおいて，

$$F(\theta) = \int_0^{\theta} \sin\tau\cos\tau\,\mathrm{d}\tau = \frac{1}{4}(1-\cos 2\theta) = \frac{1}{2}\sin^2\theta \tag{6}$$

$\int_0^{\pi/2} f(\tau)\mathrm{d}\tau = 1/2$ で割って正規化した後，$u = F(\theta)$ と置いて，整理すれば

$$\sqrt{u} = \sin\theta \tag{7}$$

を得る．実は，式(7)の右辺を $\cos\theta$ と置いて乱数を発生させても全体として変化しないので，一般には

$$\theta = \arccos\sqrt{u} \text{ または } \cos\theta = \sqrt{u} \tag{8}$$

により，関数(4)に比例した θ の分布を発生させる．Fig. 1 に上述の方法に従って発生させた放出気体の角度分布を示す．余弦則に従っていることがわかる．

3.2.3　円形導管の分子流コンダクタンス

分子の通過確率は導管の幾何学的形状によって決定される．ここでは最も単純であり，かつ実用的な円形断面の導管を取り上げる．このような簡単な形状の導管に関しては，モンテカルロ法以外の方法で通過確率を求めることが可能であり，数表や近似式が得られている．話しが前後してしまうが，モンテカルロ法の位置づけと結果の妥当性を理解してもらうために，まずこれらの値との比較を行い，続いてモンテカルロ法の実際の計算手順を記述する．

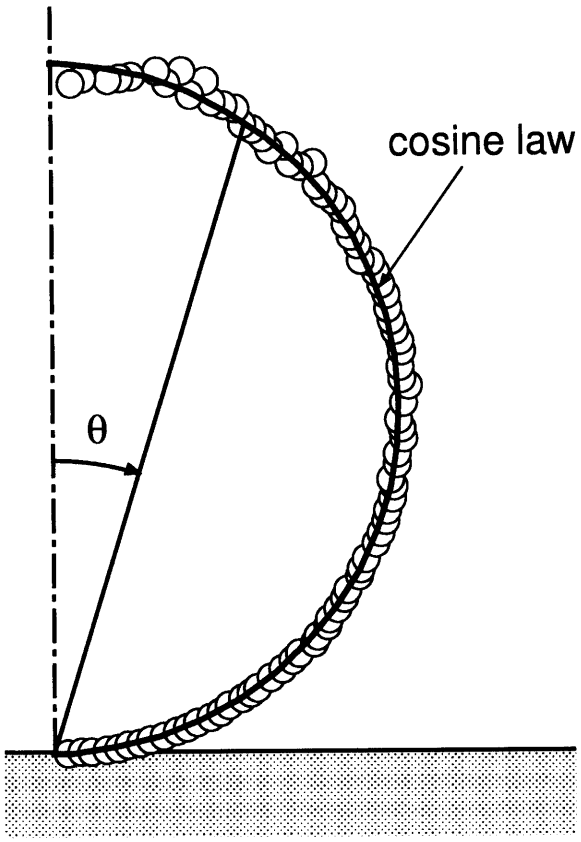

Fig. 1　Distribution of angle of emission calculated by the equation ; $\sqrt{u} = \cos\theta$.

A．近似式

円形断面導管の正確な通過確率 K は Clausing[2]自身の積分計算値（クラウジング係数とも呼ばれる）ではなく Cole[3]の再計算値が正しいとされ，表形式で離散的に与えられている．しかし，実際の装置の設計などにおいては，管の長さを L，直径を D として，任意の $x \equiv L/D$ に対する K を表す精度の良い近似式 $K(x)$ が必要となり，いくつかの近似式が考案されている．まずは，物理的な考察に基づく Dushman[4]の近似式は有名であるが，最大11%の誤差があり，あまり精度は高くない．

$$K_{\mathrm{D}} = \frac{1}{1+\frac{3x}{4}} \tag{9}$$

Clausing 係数に対する近似曲線を求める努力は，Kennard[5]，Henning[6]などよってなされ，色々な形の式が提案された．中でも Santeler[7]の近似式は，形が簡単なのにもかかわらず広い x で精度がよく，誤差は最大で0.6% 程度である．

$$K_S = \frac{1}{1+\frac{3x}{4}+\frac{1}{\frac{4}{x}+\frac{8}{7}}} \tag{10}$$

また，極めて複雑ではあるが，誤差の最大値がわずか0.13％という非常に精度の高い式がBerman[8]により与えられている．これは，円形断面導管のコンダクタンスを与える数値積分を級数展開して求めた理論的な近似式であるが，実用的とはいえない．

$$K_B = 1+x^2-x\sqrt{x^2+1}-\frac{2[(2-x^2)\sqrt{x^2+1}+(x^3-2)]^2}{9[x\sqrt{x^2+1}-\ln(x+\sqrt{x^2+1})]} \tag{11}$$

K_D, K_Sおよびモンテカルロ法によるシミュレーション結果を，Fig. 2に示す．K_Dのずれはわかるが，K_Sやモンテカルロ法による値のずれは，この図では判別できない．すなわち，モンテカルロ法による計算値は，精度の高い近似式と同等の品質であることがわかる．解析式を異なる断面形状について求めることは極めて困難であるのに対し，モンテカルロ法では，断面形状の違いをそのまま取り込んで，一定の手順で計算を進めることが可能であるから，実用的に優れた方法であるといえる．

B．モンテカルロ法の計算手順

分子流領域における気体分子の運動では，容器壁での散乱のみを考慮するので，軌跡のシミュレートは，以下のように非常に明解で単純である（Fig. 3参照）．

(1)一方の開口部Aから気体分子1個を余弦則に従って入射させる．これは，気体分子の運動が等方一様ならば，ある面を通過する分子の角度分布は余弦則となるからである．

(2)気体分子が管内空間を直進して壁と衝突する位置を求める．この位置から再び気体を放出させる．その際，放出気体分子の角度分布は，壁の接平面に対して余弦則に従わせる．入射開口部Aあるいは反対側の開口部Bから気体分子が出るまで，この計算を続ける．

(3)開口部Bから気体分子が出た場合に，通過数を1増加させる．

試行回数をN，通過分子数をnとすれば，通過確率Kは明らかに，

$$K \equiv \frac{n}{N} \tag{12}$$

と定義される．さて，具体的に計算すべき量は，気体分子の直進運動の方向余弦，壁との衝突位置，開口部との交差位置などである．壁の位置$\vec{R}_s=(x_s, y_s, z_s)$から，方向余弦$(\alpha, \beta, \gamma)$の向きに気体が放出され直進して$\vec{R}_f=(x_f, y_f, z_f)$で再び壁に衝突したとすれば

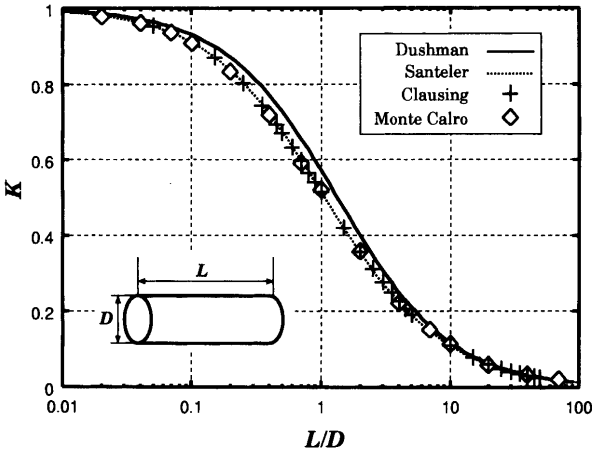

Fig. 2　Molecular transmission probability K for a cylindrical pipe evaluated by different authors.

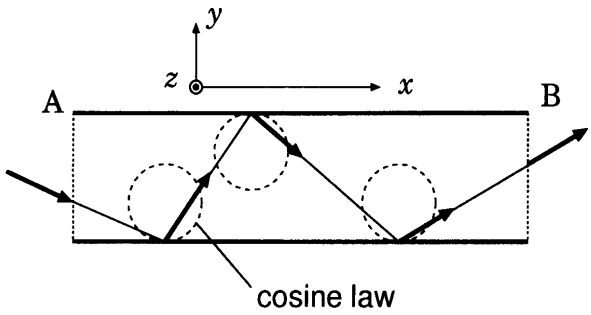

Fig. 3　Schematics of Monte Calro calculation of gas molecular transmission through cylindrical pipe.

$$\frac{x_f-x_s}{\alpha}=\frac{y_f-y_s}{\beta}=\frac{z_f-z_s}{\gamma} \tag{13}$$

が成立する．壁の形状が一般に関数

$$G(x, y, z)=0 \tag{14}$$

で表現されるとき，$\vec{R}_s$が与えられれば，(13)と(14)を連立して$\vec{R}_f$を求めることができる．もちろん，壁の形状が複雑な場合には連立方程式はかなり面倒であるが，円形断面では結局二次方程式を解くだけの問題となる．気体分子の直進運動の方向余弦は，任意の接平面からの放出を直接考えると厄介である．しかし(0, 1, 0)面での余弦則に従う分布を求め，これを適当に回転することで簡単に求めることができる．List 1にC言語でコーディングした，円形断面単管の通過確率を求めるプログラムのソースを示す．簡単な構造なので，コメントを見れば理解で

List 1　cyl.c

```
/*
    円形導管の通過確率（＝クラウジングファクター）を求める。
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>

int Pass_Number = 0;
int Back_Number = 0;
int Trial_Number = 0;

double L, D=1.0, L2D;      /* 管の寸法  L:長さ D：直径=1 固定 L2D: L と D の比 */
double xs,ys,zs,xf,yf,zf;  /* 気体分子の衝突位置 */
double al,bt,gm,ds;        /* 方向余弦 (al,bt,gm) */
double phi;                /* 軸周りの方位角 */

void nextpos()             /* 放出位置 xs,ys,zs と衝突位置の x 座標 xf から */
{                          /* 方向余弦を用いて，次の衝突位置 yf,zf を求める  */
  yf = bt*(xf-xs)/al + ys;
  zf = gm*(xf-xs)/al + zs;
}

double root(void)          /* 壁との衝突位置の x 座標 xf を求める */
{                          /* 壁からの放出の場合 */
  double btygmz=bt*ys+gm*zs, bt2gm2=bt*bt+gm*gm;
  return xs-2.0*al*btygmz/bt2gm2;
}

double rootp(double r)     /* 壁との衝突位置の x 座標 xf を求める */
{                          /* 入射開口部からの場合 */
  double btygmz=bt*ys+gm*zs, bt2gm2=bt*bt+gm*gm;
  return al*(-btygmz
         + sqrt(btygmz*btygmz - bt2gm2*(ys*ys+zs*zs-r*r)))/bt2gm2;
}

void direction1(void)      /* 入射開口部における x 軸まわりの余弦則の発生 */
 {
  al = sqrt(drand48());
  ds = sqrt(1.0 - al*al);
  phi = 2.0*M_PI*drand48();
  bt = ds*cos(phi);
  gm = ds*sin(phi);
}

void direction2(void)       /* 壁における y 軸まわりの余弦則の発生 */
{
  bt = sqrt(drand48());
  ds = sqrt(1.0 - bt*bt);
  phi = 2.0*M_PI*drand48();
  al = ds*cos(phi);
  gm = ds*sin(phi);
}

void flight()
{
  double theta, bt0, gm0, rr;

  xs = 0.0;                     /* 入射開口部では xs = 0  */
  rr = D/2.0*sqrt(drand48());   /* 円形断面で一様であるための半径の発生 */
  phi = 2.0*M_PI*drand48();
  ys = rr*cos(phi); zs = rr*sin(phi);
  direction1();                 /* 入射開口部における余弦則 */
  xf = rootp(D/2.0);            /* 初めての衝突位置の xf    */
  nextpos();                    /* xf より yf,zf を求める   */
  xs = xf;
  ys = yf;
  zs = zf;
                                /* 管の通過判定ルーチン */
  if ( xf < L ) {               /* xf < L ならば実在の壁に衝突 */
    while (1) {
      direction2();             /* y 軸周りに余弦則を発生 */
      theta = atan2(ys,zs);     /* それを位置に合わせて x 軸周りに */
      bt0 = bt;                 /* theta 回転させる */
      gm0 = gm;
      bt = -sin(theta)*bt0+cos(theta)*gm0;
      gm = -cos(theta)*bt0-sin(theta)*gm0;
      xf = root(D/2.0);         /* 次の衝突位置の xf 決定（壁から壁）*/
      nextpos();
      xs = xf;
      ys = yf;
      zs = zf;

      if (xf <= 0.0) {          /* xf <= 0 は入射開口部に逆戻り */
          Back_Number++;
          break;
      } else if ( L <= xf ) {   /* L <= xf は管を通過した */
          Pass_Number++;
          break;
      }
    }
  } else {                      /* 入射開口部から壁に衝突せず直接通過 */
    Pass_Number++;
  }
}

main (int argc, char **argv)
{
  int trial=0;
  double theta, x, x2, sx, KB;
  time_t *now;

  srand48(time(now));           /* 秒単位の時間により疑似乱数列を初期化 */
  L = atof(argv[1]);            /* 管の長さ，試行回数を引数から設定 */
  Trial_Number = atoi(argv[2]);
  L2D= L/D;

  printf("L/D = %f\t", L2D);    /* Berman の値の表示 */
  x = L2D; x2 = pow(x,2); sx = sqrt(1.0+x2);
  KB = 1.0+x2-x*sx-2*pow((2-x2)*sx+(x2*x-2),2)/(9*(x*sx-log(x+sx)));
  printf("%f by Berman's eqn.\t", KB);
  fflush(stdout);

  while ( trial < Trial_Number ){
    flight(L2D);
    trial++;
  }
                                /* 計算結果の表示 */
  printf("K = %f\n",(double)Pass_Number/Trial_Number);
}
```

きると思うが，少し補足をする．プログラム起動時に，管の長さ L (L) と試行回数 N (Trial_Number) を引数に取る（105, 106行目）．初期入射位置の決定については，円形断面で一様であるから

$$\mathrm{d}n \propto \mathrm{d}S = r\mathrm{d}\phi\mathrm{d}r \tag{15}$$

に比例させる，すなわち動径位置 r に比例した分布を発生させる必要がある．逆変換法により，円の半径（動径の最大値）を R として

$$u = \frac{r^2}{R^2} \text{ あるいは } r = R\sqrt{u} \tag{16}$$

により r を求めればよいことがわかる（61行目）．管の長さと試行回数を引数に渡して，実行すると，

```
$ cyl 2.0 100000
L/D=2.000000  0.356575 by Berman's eqn. K=0.356030
```

のように，Berman の近似計算とモンテカルロ法による計算結果が表示される．

3.2.4　X11上の簡易アニメーション

計算物理の立場から言えば，通過確率を求められればそれで十分である．しかし，気体分子がどのように運動しているかをアニメーションすれば，現象自体の理解の

助けとなるかもしれない．ここでは，ごく簡単にPC-UNIXにおけるグラフィカルな表現実現の方法を紹介し，実例画面を示す．

PC-UNIXを含めてUNIXではグラフィカルな表示はMITのX Window Sysytem（以降Xと略記する）を用いることが事実上の標準である[9]．Xでは，図形や文字のディスプレイへの描画，マウスやキーボードからの入力の処理について標準プロトコルAPIが定められており，Xlibライブラリを用いればlook & feelを含めて，あらゆるプログラムが記述できるようになっている．しかし，Xlibで直接プログラムを記述すると一般に繁雑になってしまうことから，考えられる定型手続きを簡略に記述できるようにXt（X Toolkit）ライブラリや，look & feelを規定したXaw（X Athena Widget）ライブラリが標準で提供されている．Athena以外にも商用UNIXでポピュラーなWidgetセットMotifのフリーな互換ライブラリLesstif[10]なども開発されており，プログラマは自由にGUI構築環境を選択できる．しかし，逆に選択の幅があまりにも広いために，統一した操作感が得られないといったことが，一般ユーザの普及の妨げとなっていると問題視されるようになった．そこで，現在はGNOME[11]やKDE[12]といったGUI全般を規定した統合環境が開発され，急速に浸透しつつある．どの開発環境が主流となるかは予想できないが，Xlibまた場合によってはXtがその基礎を支えていることに変わりはない．ここでは，やや繁雑になるかもしれないが普遍的な価値のあるXlibを使った例題を紹介する．

A．点を動かすアニメーションの例

アプリケーションごとに窓を設けて表示を行うWindow Systemは，今やGUIの標準手法となっている．一つの窓をスクリーンに開けるためには，定型的な初期化の手続きが必要である．List 2に窓を一つ開けて，その中で点を動かすプログラムソースを示す．コンパイル時には，

```
$ gcc -o animdemo animdemo.c -lX 11 -lm
```

のようにXのライブラリをリンクする必要がある．

誌面も少ないので全体の構成とXに特徴的な事柄のみ説明する．まず，ウィンドウには親子関係があり，全部の親はルートウィンドウと呼ばれる．この上にさらに子どものウィンドウを開いてゆく，この例ではルートの名前はrootで(15, 27行)，その上にwinというウィンドウを開いている(15, 35行)．ウィンドウを開くためにはいくつかのサーバ情報が必要であるが，定型手続きを記せばよい(26～28行)．これらのウィンドウは画面に表示させることも(42行，XMapWindow)，取り消す(XUnMapWindow)ことも可能である．一方，画像は作成できるが画面には表示されないPixmapと呼ばれる部品もある．実は，アニメーションではウィンドウを更新して表示するタイミングが重要であるが，Xサーバが元々データをバファリングするために同期を取ることが一般に難しい．それでも，なるべくスムーズに描画を行わせるために，一旦Pixmapに画面を作成しておき，いっきにウィンドウへと複写した（54行）直後バッファをフラッシュする(55行)といった手法が取られる．また Xでは，クライアントとサーバがネットワーク越しに描画情報を伝達し合うので，色や線種や塗りつぶしのモードを描画要求時に指定したのでは効率が悪いとされている．そこであらかじめ，グラフィックコンテキストという属性（GC）をまとめて指定しておき(37～40行)，描画時にその属性値を使って描く仕組みとなっている．最後に，Xサーバは，マウスやキーボードや画面のオーバーラップなどの情報をイベント（event）と称して管理している．このイベントの通知に応じてクライアント（アプリケーション）側がアクションを起こすように全体が構成される．イベントを使うには構造体の宣言(22行目)，種類と監視をする部品の選択(43行目)，タイプの判別とその処理(60, 61)を記述する．この例では，ウィンドウ内にカーソルを置きいずれかのキーを押下すると，点を動かす無限ルーチンから脱出し，終了する．

非常に大雑把ではあるが，この点の動きを衝突位置や方向余弦から計算して，ある一定の間隔で表示させれば気体分子運動のアニメーションとなることはすぐに想像がつくことと思う．いくつかプログラムを書いてみれば，手続き自身は面倒ではあるが難しくないことがわかるはずである．ただし，アニメーションの画面1枚1枚の切替がこちらの思うタイミングで行われないことなどが最後の問題として残り，汎用のシステムの限界と観念しなければならない場合がある．

B．円形断面管内の圧力分布

円形断面導管内を気体分子が通過する様子のアニメーションの実行例をFig. 4に示す．軸方向に輪切りにしてその区間ごとの壁への衝突頻度も表示させている．両端が開いている一様管(a)内の圧力分布は0次近似で，軸方向に沿って一次関数となることはよく知られている事柄であり，この図でもそのように現れている．他端が閉じている管(b)の側面に穴などの気体放出源がある場合には，穴から開口端に向かっては一次関数，閉端側では一

List 2　animdemo.c

```
#include <stdio.h>
#include <unistd.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#define BORDER 2
#define WIDTH  320
#define HEIGHT 240
#define DX 3
#define DY 2

main (int argc, char **argv)
{
  Display *dpy;
  Window root, win;
  int screen, depth;
  GC gc, gcr;
  Pixmap pix;
  Colormap cmap;
  int x=0, y=0, dx=DX, dy=DY, Wait;
  XColor black, white, green3, exact;
  XEvent event;


  Wait = atoi(argv[1]);
  dpy = XOpenDisplay("");
  root = DefaultRootWindow(dpy);
  screen = DefaultScreen(dpy);
  depth = DefaultDepth(dpy, screen);
  cmap = DefaultColormap(dpy, screen);
  XAllocNamedColor(dpy, cmap, "green3", &green3, &exact);
  white.pixel = WhitePixel(dpy, screen);
  black.pixel = BlackPixel(dpy, screen);

  win = XCreateSimpleWindow(dpy, root,
        100, 100, WIDTH, HEIGHT, BORDER, black.pixel, white.pixel);
  gc = XCreateGC(dpy, win, 0, NULL);
  gcr = XCreateGC(dpy, win, 0, NULL);
  XSetForeground(dpy, gc, green3.pixel);
  XSetForeground(dpy, gcr, white.pixel);
  pix = XCreatePixmap(dpy, root, WIDTH, HEIGHT, depth);
  XMapWindow(dpy, win);
  XSelectInput(dpy, win, KeyPressMask);

  while (1) {
    if ( x < DX || x >= WIDTH -DX ){
      x = abs(x - DX); dx *= -1;
    } else x += dx;
    if ( y < DY || y >= HEIGHT - DY ) {
      y = abs(y - DY) ; dy *= -1;
    } else y += dy;

    XFillRectangle(dpy, pix, gc, x, y, 6, 6);
    XCopyArea(dpy, pix, win, gc, 0, 0, WIDTH, HEIGHT, 0, 0);
    XFlush(dpy);
    usleep(Wait);
    XFillRectangle(dpy, pix, gcr, 0, 0, WIDTH, HEIGHT);
    XCopyArea(dpy, pix, win, gc, 0, 0, WIDTH, HEIGHT, 0, 0);

    XNextEvent(dpy, &event);
    if (event.type == KeyPress) break;
  }
  XCloseDisplay(dpy);
}
```

定圧力となると0次近似計算されるが，シミュレーションではわずかに異なる結果を示している．詳しくは述べないが教科書に載っている0次近似においては，気体分子の速度分布は管内で常に一様等方であることを仮定して計算を行っている．しかし実際には管を通過する際に軸方向に方向が揃ってくることが知られており[13,14]，近似誤差の原因となっている．モンテカルロ法ではこのような現象を自然に取り込むことになるので，0次近似の計算予測とは異なる結果が一目瞭然となる場合がある．

基本的な問題は，速度分布が一様等方でない場合には，圧力pと衝突頻度Γの間の比例係数が変化するの

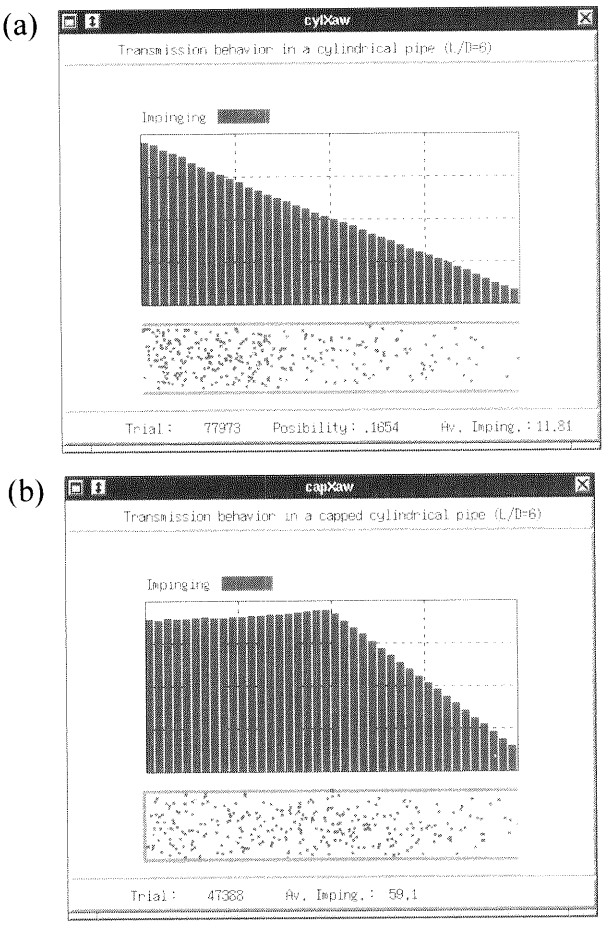

Fig. 4　Distribution of gas impinging on unit area in a pipe ; (a) both sides open (b) one side closed.

で，導管全体を通じての衝突頻度分布が圧力分布と相似であるとは見なせないという点にある．したがって，閉管の場合も衝突頻度が一定でないこと即圧力が一定でないとも結論できない．さらに言えば，速度分布が一様等方でない場合の圧力の定義とは何かという基本的問題を考え直さなければならないというところにまで発展する．このように根源的な問題までも顕在化させるモンテカルロ法のシミュレーション結果を解析式に反映させる努力は続いているが，現在のところ完成しているとは言いがたく，改めて理論の発展はモンテカルロシミュレーションに負うところ大であると言える．はなはだ頼りない解説であるが，現状を正直に述べた．

3.3　Linux について

Linux は Linus Torvalds がヘルシンキ大学に在籍していた1991年に，Minix を手本としてスクラッチから起こされた UNIX 互換(POSIX 準拠)の OS である．ハードウェア性能が著しく向上した現在のパーソナルコンピュータを活かす強力な OS として BSD とともに急速に普及している[15]．UNIX は，動作が安定であり，あらゆるプログラミング言語が手に入れられることが最大の利点であろう．例えば，C，C++，Fortran(f2c, g77)，Pascal(p2c, gpc)，Lisp などのコンパイル言語，AWK, Perl, Python, Ruby などのスクリプト言語，またネットワーク上での使用に最も期待の高まっている JAVA などを簡単にインストールして使うことが可能となっている．

加えて，PC-UNIX はまさに一昔前のスーパーコンピュータ並の演算性能を持つパーソナルな計算機をいつでも自由に使用できることも大きな魅力である．特に，高速ネットワークを介したクラスタシステムにより，現在のスーパーコンピュータ並の演算性能を1　10のコストで構成する試みは注目に値する[16]．

商用 UNIX はインストール済みのマシンを購入して使用するのが普通であるが，PC-UNIX は自分でインストールするのが常識である．普及初期には，このインストールでつまづく例も多くあったが，今はインストール自身も非常に簡便となってきている．例えば，雑誌の付録についてきた CD-ROM でブートして質問事項に3つ答えるだけで済むというものまで登場している．

参考文献

[1] 佐渡秀治編：bit, **31(9)**, 2 (1999).
[2] P. Clausing, Ann. Phys. **12**, 961 (1932) ; republished in J. Vac. Sci. Technol. **8**, 636 (1971).
[3] R.J. Cole, Prog. Astronaut. Aeronaut. **51**, 261 (1976).
[4] S. Dushman, *Scientific Foundations of Vacuum Techniques*, 2 nd ed. (Wiley, Nwe Yprk, 1949) Chap. 2.
[5] H. Kennard, *Kinetic Theory of Gases* (MacGraw-Hill, New York, 1983) p.306.
[6] H. Henning, Vacuum **28**, 151 (1978).
[7] D.J. Santeler, J. Vac. Sci. Technol. **A 4**, 338 (1986).
[8] A.S. Bermann, J. Appl. Phys. **8**, 15 (1957).
[9] 参考書を一つあげるとすれば，松田晃一：X ウィンドウ実践技術講座(ソフトリサーチ・センター, 1992)．
[10] Lesstif ウェブサイト(Hungary) : http : //www.lesstif.org
[11] 日本 GNOME ユーザー会 : http : //www.gnome.gr.jp
[12] 日本 KDE ユーザー会 : http : //www.kde.gr.jp/
[13] K. Nanbu, Vacuum **35**, 573 (1985).
[14] P. Krasuski, J. Vac. Sci. Technol. **A 5**, 2488 (1986).
[15] 日本 Linux 協会：http : //www.linux.or.jp/
[16] 新情報処理開発機構：http : //www.rwcp.or.jp/lab/pdslab/TFCC/