Scilab で楽しむ確率論-第 1 稿

千代延大造

関西学院大学理工学部数理科学科

2009 年 3 月 6 日

Contents

0.1 はじめに

確率論は「偶然 (ランダム性)」が本質的な役割をはたす現象を定式化し，理解することを目標とする数学の一分野である．特に，偶然の積み重ねが引き起こす結果を見ることに大きな興味がある．コルモゴロフによって測度論を基礎におく定式化がされて以来，確率論は解析学の一分野として大きく発展してきたが，現在も統計物理学，数理生物学，情報理論，最適化問題や通信ネットワークなどの工学，その他さまざまな応用分野からの刺激を受けながら発展を続けている．

しかし，理論の発展とともに考察する対象が複雑になり，人間の頭脳だけで正しい結果を予想することは少しずつ困難になってきている．そこで力を発揮するのが計算機である．さまざまな偶然性をともなう現象を計算機上で実現し，モデルを可視化したり重要な量を数値計算をすることにより，正しい結果の予想の大きな助けになる．

もともと線形計算を手軽にかつ高速に行うために開発された数値計算ソフトウェアである Scilab は，このように計算機上でランダムな現象を実現する上で非常に**相性のいい**ソフトである．なぜなら，そのデータの基本構造は行列であるため，複数のランダムネスからなるサンプルを 1「行」で表現し，サンプルの集まりを「列」に並べることにより確率モデルの統計的性質を調べる事ができるからである．Scilab では行数や列数が大きくても行列計算をかなり高速に実行できるため，巨大なサンプル数のシミュレーションにおいても計算時間を比較的抑えることができる．またプログラムをすっきりと短いものにできる事も大きな利点である．

このノートは，初めて Scilab を使ってみようという人を対象に書いた．筆者自身，まだこのソフトの全貌を理解しているわけではない．と言うより，まだほとんど何も知らない．しかし，初級の知識だけでも十分におもしろい確率モデルを実現できる事だけは了解したつもりである．このソフトの良さは，インタープリタ方式である事と相俟って，とにかく**気軽に使える**事にある．Scilab を使いこなせるようになるには，何度でも機械 (パソコン) に問いかけてその返す答えを見て考えていく事である．このノートも，ひとつひとつの機能の説明

はほとんどしていない．このノートなり，他の資料なりを見て興味のあるものを打ち込んでみて欲しい．機械と応答を繰り返すうちにScilabを使いこなす事ができるようになるはずである．

0.2 Scilab最初の一歩

何はともあれ，Scilabを使ってみよう．まだ自分のパソコンに入ってない人はネット上からダウンロードする．”Scilab, download”で検索すればすぐに入手先サイトを見つける事ができる．現在WindowsおよびLinux版がVersion5.0.3である．Mac版は今のところVersion4.1.2のみ提供されているようである．

Scilabを開くとConsoleという画面が立ち上がり，次が現れる．

```
        -------------------------------------------
                        scilab-5.0.3

                    Consortium Scilab (DIGITEO)
                  Copyright (c) 1989-2008 (INRIA)
                  Copyright (c) 1989-2007 (ENPC)
         -------------------------------------------


Startup execution:
  loading initial environment
```

これで使える状態である．長いコードは’Application’から’Editor’を選んでEditorを使って書くが，しばらくはConsoleで機械と対話しながらScilabを覚えていこう．なお，’?’から’Scilab Help’を選べば各コマンドについて詳細な解説を読むことができる．検索を利用して’Help’を活用されたい．’Scilab Demonstrations’を選ぶといくつかの楽しいデモを見ることができる．個人的にはその中の’Simulation’の中の’Bike Simulation’などが印象的であった．

では，コードの書き方の紹介を始めよう．

0.2.1 行列としてのデータの表し方

最初に述べたとおり，Scilab のデータの基本構造は行列である．すなわち，行列で表されるデータを対象にしている．したがって，この行列で表されるデータの取り扱い方を覚える事が Scilab の基礎を理解する事と同値である．最も基礎的な事から始める．console を開いて以下を打ち込んでみてほしい．「// 」以下はコメントなので，打ち込む必要はない．1 行書くたびに機械が応答してくれるので，打ち込んだ事が何を意味しているかすぐに知ることができる．コメントがない所は，コメントがなくても機械が返す返事を見ればすぐにその意味を知ることができる所である．どうしてもわからなければ Help や [**1**]，[**2**] などにあたろう．

```
/// n × m 個の数字を n 個ずつならべ，n 個ごとにセミコロン（ ; ）を入れる
/// ことによって n 行 m 列の行列を作る事ができる．
/// 各行の成分の間にはスペースをあける，またはカンマ（,）を入れる．
/// 次の行に移る時には必ずセミコロン（ ; ）を入れる．
-->[5 4 3; 2 3 1]
 ans  =
    5.    4.    3.
    2.    3.    1.

-->A=[5, 4, 3; 2, 3, 1]
 A  =
    5.    4.    3.
    2.    3.    1.

-->[2;1;4]
 ans  =
    2.
    1.
    4.

/// 行列 A の転置を A’ によって表す．
```

```
-->A=[2,1,4]; B=A'
 B  =
    2.
    1.
    4.

/// 行列 A の n 行目 m 列目の成分を A(n,m) で取り出す事ができる.
/// また n 行目を A(n,:) によって, m 列目を A(:,m) によって取り出す事ができる.
-->A=[3,8; 2,5]
 A  =
    3.    8.
    2.    5.

-->A(1,2)
 ans  =
    8.

-->A(1,:)
 ans  =
    3.    8.

-->A(:,2)
 ans  =
    8.
    5.

// n 行 m 列の行列 A と n 行 m' 列の行列 B に対して A の右に B を並べた行列を
// [A, B] によって作る事ができる. また,
// n 行 m 列の行列 A と n' 行 m 列の行列 B に対して A の下に B を並べた行列を
// [A; B] によって作る事ができる.
```

```
-->[1:4, 2:5]
 ans  =
    1.    2.    3.    4.    2.    3.    4.    5.

-->A=[2,3; 4,5], B=[3,6; 4,2]
 A  =
    2.    3.
    4.    5.
 B  =
    3.    6.
    4.    2.

-->C=[A,B], D=[A;B]
 C  =
    2.    3.    3.    6.
    4.    5.    4.    2.
 D  =
    2.    3.
    4.    5.
    3.    6.
    4.    2.

-->E=[A(1,:);B(2,:)], F=[A(:,1), B(:,2)]
 E  =
    2.    3.
    4.    2.
 F  =
    2.    6.
    4.    2.
```

```
///  行列 A に対して A(k,:)=[] とすることにより，A は k 行目を取り去った新しい
/// 行列になる．列についても同様である．
-->A(1,:)=[]
 A  =
    4.    5.

-->A=[1,4; 2,5; 3,6], A([2,3],:)=[]
 A  =
    1.    4.
    2.    5.
    3.    6.
 A  =
    1.    4.
```

0.2.2 行列の演算

```
/// 対角行列は diag を用いる
-->diag([2,1,3])
 ans  =
    2.    0.    0.
    0.    1.    0.
    0.    0.    3.

/// 行列としての和，差，積，べき乗はそれぞれ A+B，A-B，A*B，A^n で表す．
-->A=[1, 2; 3, 4], B=[1, -1; 2, -1], C=A+B, D=A-B, E=A*B, F=A^2, G=F*A^{-1}
 A  =
    1.    2.
    3.    4.
 B  =
```

```
    1.  - 1.
    2.  - 1.
 C  =
    2.    1.
    5.    3.
 D  =
    0.    3.
    1.    5.
 E  =
    5.   - 3.
    11.  - 7.
F  =
    7.     10.
    15.    22.
 G  =
    1.    2.
    3.    4.

-->A=[1,2,3], B=[1,1,1]', C=A*B, D=B*A
 A  =
    1.    2.    3.
 B  =
    1.
    1.
    1.
 C  =
    6.
 D  =
    1.    2.    3.
    1.    2.    3.
```

```
    1.    2.    3.

/// 行または列ごとの定数倍も容易にできる.
-->A=[1, 2; 3, 4],B=diag([2,3]), C=A*B, D=B*A
 A  =
    1.    2.
    3.    4.
 B  =
    2.    0.
    0.    3.
 C  =
    2.    6.
    6.    12.
 D  =
    2.    4.
    9.    12.
```

Scilab の行列の演算には，上に掲げた通常の線形代数としての行列演算の他に,「成分ごとの」演算が準備されている．たし算や引き算では通常の演算と成分ごとの演算は同値であるが,積を考えると両者は当然異なる.

```
/// 行列 A,B に対し，A.*B，A./B によって各成分ごとの積，商をとってできる
/// 行列を表す．A.^n によって各成分ごとに n 乗した行列を表す．A.^B によって
/// A の各成分を B の各成分乗した行列を表す.
-->A=[1, 2; 3, 4], B=[1, -1; 2, -1], C=A.*B, D=A./B, E=A.^2, F=A.^B
 A  =
    1.    2.
    3.    4.
 B  =
    1.  - 1.
```

```
    2.  - 1.
 C  =
    1.  - 2.
    6.  - 4.
 D  =
    1.   - 2.
    1.5  - 4.
 E  =
    1.    4.
    9.    16.
 F  =
    1.    0.5
    9.    0.25
```

Scilab には行列の行列式やトレースなど線形代数学において重要な諸量を求めるコマンドが準備されている．ここでは，最も重要なもののみ掲げる．

```
/// 行列 A に対して det(A), trace(A), rank(A), inv(A), orth(A), sqrtm(A) は
/// それぞれ A の行列式，トレース，ランク，逆行列を表す．
-->A=[2,3; 3,7], B=det(A), C=trace(A), D=rank(A), E=inv(A)
 A  =
    2.    3.
    3.    7.
 B  =
    5.
 C  =
    9.
 D  =
    2.
 E  =
```

```
    1.4  - 0.6
  - 0.6    0.4
```

0.2.3 数列

次に Scilab における等差数列の作り方を示す．これは，グラフィックにおいても重要である．またすべての成分が 0 または 1 の行列の作り方とあわせて以下に示す．

```
/// a から b への公差 1 の等差数列を a:b によって表す..
-->1:5
 ans  =
    1.    2.    3.    4.    5.

-->1.5: 5.5
 ans  =
    1.5    2.5    3.5    4.5    5.5

/// a から b への公差 c の等差数列を a:c:b によって表す.
-->1: 0.5: 3
 ans  =
    1.    1.5    2.    2.5    3.

-->1:2:5
 ans  =
    1.    3.    5.

-->[1: 5; 3: 0.5: 5]
 ans  =
    1.    2.    3.    4.    5.
    3.    3.5    4.    4.5    5.
```

```
// zeros(n,m) はｎ行ｍ列の零行列，ones(n,m) はｎ行ｍ列の各成分が１の行列を表す．

-->zeros(2,5)
 ans  =
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.

-->ones(1,6)
 ans  =
    1.    1.    1.    1.    1.    1.

-->[ones(3,3), zeros(3,3)]
 ans  =
    1.    1.    1.    0.    0.    0.
    1.    1.    1.    0.    0.    0.
    1.    1.    1.    0.    0.    0.

-->[-ones(2,4); ones(1,4)]
 ans  =
  - 1.  - 1.  - 1.  - 1.
  - 1.  - 1.  - 1.  - 1.
    1.    1.    1.    1.

-->A=diag([2,3])*ones(2,5), B=ones(5,2)*diag([2,3])
 A  =
    2.    2.    2.    2.    2.
    3.    3.    3.    3.    3.
 B  =
    2.    3.
    2.    3.
```

```
    2.    3.
    2.    3.
    2.    3.

-->A=ones(1,5)./(1:5)
 A  =
    1.    0.5    0.3333333    0.25    0.2

-->A=ones(2,2)./[1,2; 3,4]
 A  =
    1.           0.5
    0.3333333    0.25
```

行列 A の各行ごと，あるいは各列ごとの和をとる操作もよく用いる．また，また，A の各行各列ごとに左（上）から順次可算して新しい行列を作るコマンドも用意されている．

```
/// 行列 A の行または列の成分ごとのたし算を sum(A,'c'), sum(A,'r') によって表す
/// また，成分すべての和を sum(A) によって表す．
-->A=[2,3; 4,7], B=sum(A,'c'), C=sum(A,'r'), D=sum(A)
 A  =
    2.    3.
    4.    7.
 B  =
    5.
    11.
 C  =
    6.    10.
 D  =
    16.
/// 行列 A の行ごと列ごとに順次可算して作る新しい行列を
```

```
/// cumsum(A,'c'), cumsum(A,'r') によって表す.
-->A=(1:4), B=cumsum(A)
 A  =
    1.    2.    3.    4.
 B  =
    1.    3.    6.    10.

-->A=[2,3; 4,7], B=cumsum(A,'c'), C=cumsum(A,'r')
 A  =
    2.    3.
    4.    7.
 B  =
    2.    5.
    4.    11.
 C  =
    2.    3.
    6.    10.
```

0.2.4 条件を判別する

行列の成分の中から，必要な条件を満たすものだけを探し出すための Scilab のコマンドを紹介しよう.

```
/// 2 個の成分 T(True) および F(False) からなる行列を真偽行列という.
/// 行列 A に対して，(A のある条件) は，条件の真偽行列を表す.
/// 例えば (A>=0.5) とすると，0.5 以上である成分に対し
/// T(True), そうでない成分に対し F(False) を返す.
/// 真偽行列 B に対し~B は真偽を逆転させた行列を表す.
/// また，(A>=0.5)*1 とすると，T に対し 1，F に対し 0 を返す.
```

```
/// 各成分が [0,1] 上の一様乱数である rand(m,n) に対して試してみよう.

-->A=rand(2,5)
 A  =
    0.8782165    0.5608486    0.7263507    0.5442573    0.2312237
    0.0683740    0.6623569    0.1985144    0.2320748    0.2164633

-->B=(A>=0.4), C=~B, D=B*1
 B  =
  T T T T F
  F T F F F
 C  =
  F F F F T
  T F T T T
 D  =
    1.    1.    1.    1.    0.
    0.    1.    0.    0.    0.

/// 2個の真偽行列 A, B に対して (A&B) は「A かつ B」, (A|B) は「A または B」
/// を表す真偽行列である. すなわち, A, B がともに T である成分に対してのみ
/// (A|B) の成分は T であり, A または B のどちらかが T である成分に対して
/// (A&B) の成分は T である.

-->A=(rand(2,5)>=0.5), B=(rand(2,5)<=0.5)
 A  =
  T F F T T
  F T F T T
 B  =
  F T F T T
  T F T F T
```

```
-->C=(A|B), D=C*1
 C  =
  T T F T T
  T T T T T
 D  =
    1.    1.    0.    1.    1.
    1.    1.    1.    1.    1.

-->E=(A&B), F=E*1
 E  =
  F F F T T
  F F F F T
 F  =
    0.    0.    0.    1.    1.
    0.    0.    0.    0.    1.

/// An application ///
A=rand(4,2)
B=find(sum(A,'c')>1)
A(B,:)

/// Boolean 2 ///
A=[1 1 0 0 1]
B=[0 1 0 1 0]
bool2s(A|B)           // is 1 if either A or B is one, element-wise.
bool2s(A&B)           // is 1 if both A and B are one,  element-wise.
bool2s(~A)            // =1-A.


// Boolean 3 //
```

```
A=[ 2 4; -1 3]
B=[3 1; -1 6]
(A==B)
(A~=B)
(A>B)
(A>=B)

///条件を満たす成分だけを取り出して新しい行列を作る.
A=rand(1,4)
A=A(A>0.2)

/// 条件を満たさないものを除外して新しい行列を作る.
A=rand(1,4)
A(A<0.2)=[];A



/// Max and VarMax of numbers.///
A=rand(2,3)
[m,n]=max(A,'c')
// n gives row-wise which one  is the maximam and
// m gives the maximum values.

// How to use gsort.
A=[4 1 2 3  5 6;3 4  5  6 6 6]
B=gsort(A,'g','d')
// sort the elements of the array A in a decreasing order.
B=gsort(A,'g','i')
// sort the elements of the array A in the increasing order.
B=gsort(A,'lc','i')
// sorts the first row and the second rows follows accordingly.
```

```
//// How to use 'gsort' and 'mean'  ////
x=(rand(10,5)<(1/2)*ones(10,5));          /// 10 × 5 の{0,1}-行列.
SX=grand(10,1,'nor', 0,1)                 /// 10 × 1 の正規分布の確率変数.
A=[SX,x]                                  /// 1 列目 SX, 2~11 行目 x.
B=gsort(A, 'lr', 'i')
/// 1 列目の 10 の数字を大きい順に並べ 変える. 1 列目の並べ替えに
/// 従って, もとの各行を保つように並び替える.
```

0.2.5 ループ

Scilab の良さを生かすには行列の形のデータをそのまま演算する事である. したがって, ループ文を用いて各成分ごとに演算するという事はできるだけ避けたい. しかし, それはループ文は不要だという事ではない. 行列のデータをある条件を満たすまで繰り返し演算させるといった事がしばしば必要になる. ループ文の作り方を学ぼう.

まず for 文から始める. ループに入ると 1 行ごとの機械の応答はない. コードそのものも長くなってきますので, エディタを開いてそれに書いていこう. "Editor" というタブを押すと画面が開く. 1 つのコードを書くたびに名前をつけて保存し, "Execute" から "Load into Scilab" を選択するとコードを実行して console 画面に結果が表示される.

```
/// 行列を行ごと入れ変える.
A=rand(3,4)
X=[2 1 3]
B=zeros(3,4);
/// B という新しい行列を以下定義する.
/// 最初にサイズを指定しておくと良い.
for i=1:3
  B(i,:)=A(X(i),:);
end
```

```
B
```

次に

```
R=(rand(2,2)>0.5)
```

と 1 行で書けるものをループを用いて書いてみる.

```
/// Create a random 0-1-valued matrix.
R=grand(2,2,"unf",0,1)
for i=1:2;
  for j=1:2 ;
    if R(i,j)>0.5 then R(i,j)=1;
    else R(i,j)=0;
    end
  end
end
R
```

次に，while 文の使い方として，次の 2 つの例を挙げておく．最初はフィボナッチ数列を作る.

```
// Fibonacci sequence,
  a=1; b=1;
while a+b<100
  c=a+b
  a=b;
  b=c;
end
```

次に有名なユークリッドの互除法のコードを紹介しよう．ここでは，while，if，else，break などの記号が出てくる．詳しくは Help やその他の資料を参照せよ．ユークリッドの互除法のコードを何も見ないで書けるようになれば，ループ文の作り方も免許皆伝と言える．

```
   //// GCD, Euclid の互除法  ////
a=1000
b=210
while b>0
  c=a-b*int(a/b);
  if c==0,
    a=b ;
    break,
  else
    a=b, b=c;
  end;
end
a
```

0.2.6 関数

例えば

$$y = (x-1)(x+1)(x+3)$$

や

$$y = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

など 1 行の式で定義される関数を Scilab で定義するには deff を用いるのが便利である．最初の式を $mypol(x)$ という名前で定義したければ

```
deff('y=mypol(x)', 'y=((x-1).*(x+1)).(x+3)');
```

とする．成分ごとのかけ算を2回用いて定義している事に注意せよ．例えば$[-4,4]$上0.1ずつの幅でならべたベクトル(-4:　0.1 4)の各成分に対して$x-1, x+1, x+3$を求めてそれを各成分ごとにかける事によって定義する．

```
x=-4:0.1:4;
mypol(x)
```

で-4から4まで0.1飛びの値xに対して$mypol(x)$の値を返す．

ちなみに，Scilabには**数式処理的な機能もある**．この多項式は行列A=diag([-3,-1,1])の特性方程式であるから

```
A=diag([-3,-1,1])
y=poly(A,'x')
```

と書くと

```
 y  =

              2   3
 - 3 - x + 3x + x
```

と返してくる．そこで

```
deff('y=mypol(x)', 'y=-3-x+3*x^2+x^3');
```

としても良い．

同様，後者の関数を定義しそれを描画したければ

```
deff( 'y=Gauss(x)', 'y=1/sqrt(2*%pi)*exp(-(x^2)/2)' );
clf();
x=-4:0.01:4;
plot2d(x,Gauss(x));
```

とする.

ループ文など複数行を用いて関数を定義したい時には function を使う. 使い方の例を挙げておこう.

```
/// Factorial, Calculate k!.//
k=1;
function[x]=fact(k)
  x=1;
  for j=1:k,
    x=x*j;
  end
endfunction
```

によって階乗 $k!$ が定義された. $k = 10$ までの値は

```
for k=1:10;
fact(k)
end
```

によって知る事ができる. もちろん, 1 行の式でかける関数も

```
function y=f(x)
  y=sin(x)
endfunction
```

のように定義できる.

0.2.7 描画

Scilab は簡単に使えるすぐれた描画機能を持っている. ここでは, 再び 1 行ごとに応答を確かめながら, グラフを描く事が簡単にできる事を確かめられたい. まず 2 次元のグラフ, すなわち実数上の関数 $f(x)$ に対して $y = f(x)$ のグラフを描いてみる.

```
clf()
x=[-2*%pi :0.1: 2*%pi]';
plot2d(x,sin(x))

clf()
plot2d(x,sin(x),1,frameflag= 6)

clf()
plot2d(x,2*sin(x),12)

clf()
plot2d(x-4,sin(x),1,leg="sin(x)")
```

あるいは

```
function y=f(x)
```

```
  y=sin(x)
endfunction
clf()
plot2d(x,f(x))
```

で $y(x) = \sin(x)$ のグラフを得る事ができる．他のソフト同様，描画の細かい指定をしていく事もできる．詳しくは，Help で plot2d の項目を参照せよ．

Scilab による 3 次元のグラフの描画は，手で描く事が 2 次元の時より困難であるため，より有用である．平面上の関数 $f(x, y)$ に対して $z = f(x, y)$ のグラフを描いてみる．

```
clf()
deff('z=f(x,y)','z=(x-1).*exp(x.*y)' );
x=-1:0.1:1;
y=0:0.1:2;
subplot(1,2,1)
fplot3d1(x,y,f,alpha=60,theta=60);
subplot(1,2,2)
fcontour(x,y,f)
```

コイン投げのような反復試行の結果をグラフで表示したい時，ヒストグラムを用いる．これを描く機能を説明してこの章を終わる．

```
histplot([0:0.01:1],rand(1,10000));
```

$[0, 1]$ 上の一様分布に従う確率変数を 10000 回発生させて，それらが 0.01 刻みの間隔に入る回数を数えて棒グラフにしたものを得る．回数を増やしていくとグラフの高さは一定に近づいて行く事を見てとる事ができる．これは**大数の法則**の反映である．

0.3　コイン投げ—-極限定理のシミュレーション

前に述べた描画の機能を用いて，コイン投げに代表される独立同分布の確率変数列に対する大数の法則や中心極限定理を視覚的に実感する事ができる．コイン投げについてプログラムしてみよう．

```
Y=bool2s(rand(1,1000)<0.5);
///  coin-tossing 1000 times.
hatS=cumsum(Y)./(1:n);
plot(hatS)
```

rand は $[0,1]$ 上の一様分布に従う乱数を 1000 個発生させたものなので，Y は確率 $1/2$ で 0 または 1 の値を取る独立な確率変数列 $\{X_k, k=1,\ldots,n\}$ である．したがって，hatS は

$$\hat{S}_n = \frac{1}{n}\sum_{k=1}^{n} X_k,$$

すなわち n 回コインを投げて表がでる「**頻度**」を $n=1$ から $n=1000$ まで並べたものである．何度繰り返しても n が大きくなるとほぼ hatS は 0.5 である事を見る事ができる．これに伴う $\hat{S}_n$ の「**揺らぎ**」の法則が**中心極限定理**である．それによると，

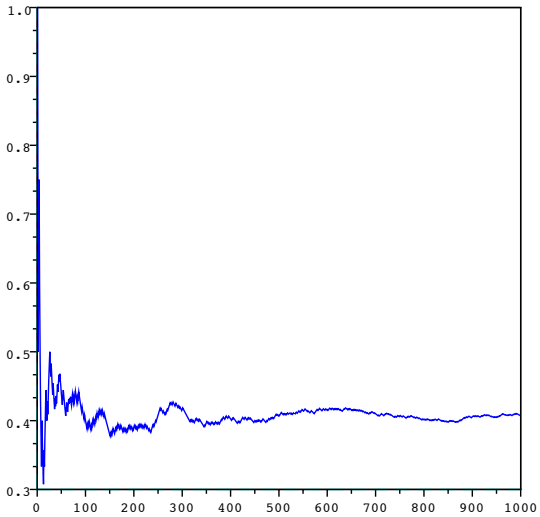
$$Z_n = \frac{n(\hat{S}_n - p)}{\sqrt{np(1-p)}} \to N(0,1)$$

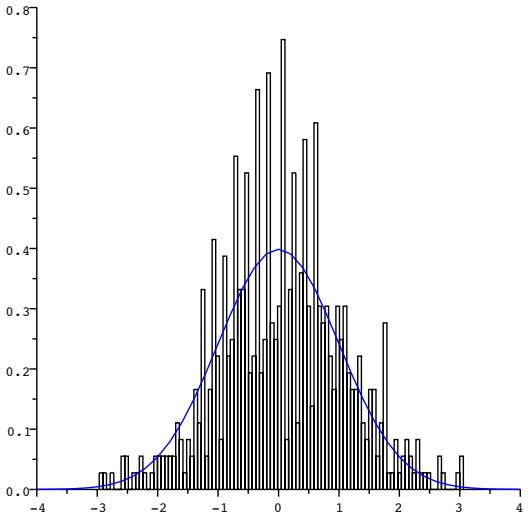
である．この事をヒストグラムを用いて視覚的に見る事ができる．今度は n 回のコイン投げを行うたびに上の Z_n を求める，それを N 回繰り返したものが実直線上を等分した間隔に入る回数を数える事により，Z_n の確率分布を近似的に求めるのである．

```
n=1000;  //number of coin tossing
N=1200;  //number of simulation
X=rand(N,n);
Y=bool2s(X<p);
m=n*p;
```

```
sigma=sqrt(n*p*(1-p));
Z=(sigma)^(-1)*(sum(Y,"c")-m);
histplot(100,Z)
```

n 回のコイン投げを N 回行うので，あらかじめ乱数を $N \times n$ 行列の形で発生させ，各行ごとに Z_n を求める．それら N 個の値のヒストグラムが最後に得たものである．次が上の 2 個のコードの実行例である．行列演算や描画についての Scilab の便利な機能が生かされている事を実感してもらえると思う．





0.4 Monte Carlo 法

大数の法則という言葉が出てきたので，この法則を応用した数値計算の技法である Monte Carlo 法について，ごく簡単なアイデアを Scilab のコードの実例とともに紹介しよう．大数の強法則より，分布 μ にしたがう独立同分布な確率変数列 $\{X_k\}_{k=1,2,\ldots}$ および適当な関数 f に対して確率 1 で

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} f(X_k) = \int f(x)\mu(dx)$$

が成立する．確率 1 で成立するのだから，十分大きな n に対して $f(X_k)$ 達の算術平均 $\hat{S}_n = \frac{1}{n}\sum_{k=1}^{n} f(X_k)$ は必ず積分 $\int f(x)\mu(dx)$ に近いであろう，よって逆にこの積分の近似値としてこの算術平均をとろう，という考え方である．積分

$$m = \int_0^1 e^{-x^2/2} dx$$

をこの考え方により近似的に求めてみよう．コード

```
/// Crude Monte Carlo.
deff('y=f(x)', 'y=exp(-x^2)')
n=1000;
mean(f(rand(1,n)))
```

を 1 回試してみると，

```
 ans  =

    0.7532813
```

を得た．これはサンプルをとるたびに定まる値であるから，1 回の試行で得た値だけを見ても心もとない思いをする．n を無限大にすればサンプル値は必ず真の値に近づく，と大数の法則は言うが，現実のサンプリングは有限の n に対して行うからである．しかし，中心極限定理を応用することにより，95 パーセントの確率でサンプル値は真の値に十分近い，という形の結論を得る事ができる．より正確には，

$$P\left(\left[\hat{S}_n - \sqrt{\frac{\hat{S}_n(1-\hat{S}_n)}{n}}z, \hat{S}_n + \sqrt{\frac{\hat{S}_n(1-\hat{S}_n)}{n}}z\right]\right) = 0.95$$

が成立する．(式に現われる区間を**信頼区間**という．z は 0.95 に応じて定まる値で，約 1.96 である．) これは統計学における**推測理論**の重要な帰結である．

前にやったように `rand(1,n)` を `rand(N,n)` として得た N 個の結果を `histplot` で視覚的に分類してみるとよい．N を大きくするとヒストグラムのばらつきが小さくなっている事を視覚的に確かめる事ができよう．

もう一つ，もっと有名な例を挙げよう．Monte Carlo 法をもちいて**円周率** π を近似的に求めてみよう．$[0,1]\times[0,1]$ 内の一様分布に従う i.i.d. が領域 $\{(x,y), x^2+y^2\leq 1\}$ に入る頻度が $\pi/4$ を近似するので

```
//Monte-Carlo calculation of pi
n=100000;
x=rand(2,n);
p=sum(bool2s(x(1,:)^2+x(2,:)^2<=1.0 ))/n*4
```

によって π の近似値を得る事ができる．これも，N 回行った結果をヒストグラム化する事を試みられたい．

先ほど見たとおり，大数の法則よりコイン投げの回数を大きくなると表が出る頻度は確率 1 で 0.5 に近づく．したがって，表が出る頻度が 0.65 以上である確率 $P(\hat{S}_n \geq 0.65)$ は非常に小さい値である．その事を事象 $\{\hat{S}_n \geq 0.65\}$ は **Rare Event** であるという. いずれにせよ，この事象の確率は 2 項分布 $B(n,0.5)$ から計算できる．2 項分布 $B(n,0.5)$ は Scilab では `binomial(0.5,n)` によって $n+1$ 個の成分のベクトルとして与えられるのでこの確率は

```
A=binomial(0.5,n);
p=0.65;
B=[zeros(1,int(p*n)) ones(1,n+1-int(p*n))];
P=A*B'
```

によって計算することができ，Scilab は

```
--> P  =

    0.0017588
```

と答えてくれる．この確率を Monte Carlo 法から求めてみよう．Scilab のコードは今までと同様簡単に書く事ができる．grand(1,N,'bin',n,0.5) により $B(n, 0.5)$ に従う確率変数を N 個発生させる事ができるので MC による確率の推定量およびその分散は

```
 N=10000; // the number of sampling.
Crudemean=mean(bool2s(grand(1,N,'bin',n,0.5)/n>=p))
CrudeVar=variance(bool2s(grand(1,N,'bin',n,0.5)/n>=p))
```

で計算できる．しかし，非常に大きなサンプル数による推定にもかかわらず，1 回試みた結果は

```
Crudemean  =

    0.0026
```

で，あまり正確な答えを出してくれるとは言いがたい．真の値との比は 1 からかなりかけ離れている．実は，Rare Event を Monte Carlo 法で推定するためには，相当に大きなサンプル数を必要とすることを中心極限定理から確かめる事ができる．それほど大きくない数のサンプルから Rare Event の確率を正確に推定するためのさまざまな技法が研究されている．代表的なものが**重点サンプリング法** (**Importance Sampling method**) と呼ばれるものである．それを用いた Scilab の活用法を述べるのはまた別の機会にしたい．

0.5　確率変数・確率過程の生成

先ほどの例で現れた grand を用いると正規分布やポアソン分布など重要な確率分布に従う確率変数を機械に生成する事ができる．しかし，$[0, 1]$ 上の一様分布に従う確率変数 (達) からこれらのさまざまな分布に従う確率変数を生成させることは，確率変数・確率分布を学ぶ上からも，また Scilab のコードを書く技術を修得する上でも非常に有益である．以後，$[0, 1]$ 上の一様分布，あるいはそれにに従う確率変数の集まりを $U(0, 1)$ と書く事にする．

0.5.1 離散確率変数の生成法

例えば

$$P(X=1)=P(X=2)=P(X=3)=0.2,\quad P(X=4)=0.4$$

であるような X は，$U(0,1)$ に従う Y を用いて

$$X=\begin{cases}1, & 0\le Y\le 0.2\\ 2, & 0.2\le Y\le 0.4\\ 3, & 0.4\le Y\le 0.6\\ 4, & 0.6\le Y\end{cases} \tag{1}$$

と定義することにより得られる．これはこの後述べる**逆関数法**の離散版である．1 個の $[0,1]$ 区間上の一様乱数からこの離散分布に従う確率変数を生成するためには，以下のようにすれば良い．

```
X=rand(1);
P=[0.2, 0.2, 0.2, 0.4];
sum=0;
Y=0;
for j=1:4
  sum=sum+P(j);
  Y=Y+1;
  if (X<sum)
   break;
  end
end
Y
```

行列の所で出てきた `[m,n]=max(A,'c')` の機能を使うと上のコードと同値のものを，Scilab のコマンド `[m,k]=max(A)` を用いて 1 行のコードで書く事もできる．Help によると

```
For A, a real vector or matrix, max(A) is the largest element A. [m,k]=max(A)
gives in addition the index of the maximum.
```

とある．すなわち，[m,k]=max(A) は最大値 m のみならず，最初に最大値をとる位置 (index)k を教えてくれるコマンドである．そこで

```
[m,X]=max(bool2s(rand(1)*ones(1,4)<cumsum(P)));
X
```

としてみよう．cumsum(P)=(0.2 ，0.4，0.6，1.)であるから，もし rand(1)=0.75 であれば，bool2s(rand(1)*ones(1,4)<cumsum(P))=(0. ，0. ，0. ，1.) である．4 番目で初めて全体の最大値 1 をとるので $X = 4$ である．言い換えれば，0.4 の確率で 4 の値をとる．これが上で定めたい離散確率変数であった．

しかも，この場合 X=rand(1,1) を X=rand(N,1) と変更し，それに応じて多少の修正を加えることにより，N 個の独立な X を得ることができる．Scilab の行列演算の理解に対する良い練習問題である．読者は試みられたい．

0.5.2 逆関数法による連続確率変数の生成

確率変数 X の分布関数 $F(x) = P(X \leq x)$ の逆関数 F^{-1} が計算できる場合には，$U(0,1)$ から分布関数 F を持つ確率変数を生成する事ができる．なぜなら，$U \in U(0,1)$ として $X = F^{-1}(U)$ とすると

$$P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x)$$

であるからである．

命題 1. F を以下の条件を満たす実数上の関数とする．

1. F は単調増大な連続関数．

2. $\lim_{x\to-\infty} F(x) = 0,\ \lim_{x\to\infty} F(x) = 1.$

この時，$U(0,1)$ に従う確率変数 U に対して $X = F^{-1}(U)$ とおくと，X は分布関数 F をもつ．すなわち，任意の x に対して $P(X \leq x) = F(x)$ である．

その最も基本的な例が指数分布 $Exp(\lambda)$ である．その分布関数は

$$F(x) = 1 - e^{-\lambda x}$$

であるから，$F^{-1}(x) = \dfrac{1}{\lambda}\log(1-x)$ である．したがって，命題 (1) より，`Y=-log(rand(1,n))/lambda;` から $Exp(\lambda)$ に従う確率変数が生成される．次のコードは，生成した n 個の変数のヒストグラムと，$Exp(\lambda)$ の密度関数のグラフを比較するものである．

コード 1. 指数分布に従う確率変数の生成

```
clf()
lambda=0.2;
X=0:0.2:30;
plot(X,lambda*exp(-lambda*X))
n=10000;
Y=-log(rand(1,n))/lambda;
histplot(X,Y)
```

結果は以下の通りである．n を大きくすればするほど，ヒストグラムと密度関数のグラフが近づいて行く事がわかるであろう．この現象も大数の法則の効果の表れである．

0.5.3 棄却法 (Acceptance-Rejection Method)

逆関数法は簡明な方法であるが，分布関数の逆関数があらかじめ明示されている場合にのみ適用できる方法であった．次に，より一般の密度関数 $f(x)$ を持つ独立な絶対連続確率変数の列を生成する方法を紹介する．話を 1 次元に限定するが，この節で紹介する方法は本質的一般次元において実行できる．

逆関数法の場合と同様，一様分布に従う独立確率変数列を生成できる事を前提に，それとは異なる分布 f に従う独立確率変数列 $\{X_k\}_{k=1,2,\cdots}$ を生成する．ただし，生成した列が f に従うものであるか否かは，大数の法則に基づき，

$$\lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n}1_{[s,t]}(X_k)=F(t)-F(s) \tag{2}$$

がすべての $a<b$ に対して成立するか否かで判断するものとする．

簡単のため,f は $\mathbb{R}$ 上連続かつ有界な台を持つものとする．すなわち，ある有界閉区間 $[a,b]\subset\mathbb{R}$ の外で $f(x)=0$ であるとする．このとき $M=\sup_{x\in\mathbb{R}}f(x)<\infty$ である．平面内の領域 $A=\{(x,y);0\le y\le f(x)\}$ は長方形 $R=[a,b]\times[0,M]$ 内の部分集合である．R 上に一様分布する独立確率変数列 $\{(X_k,Y_k)\}$, $k=1,\cdots$ を生成し，もし $(X_k,Y_k)\in A$ であるときはその X_k を受容 (accept) し，そうでなければ棄却 (reject) する．受容した X_k 達を改めて Z_j, $j=1,\cdots$ とする．これが独立同分布の確率変数列である事は明らかである．また，$a<s<t<b$ に対して $R_{s,t}=\{(x,y)\in R;s\le x\le t\}=\{(x,y);s\le x\le t,\,0\le y\le f(x)\}$ とおくと

$$P(Z_j\in[s,t])=P((X_k,Y_k)\in R_{s,t})=|R_{s,t}|=\int_s^t f(z)dz$$

である．したがって各 Z_j は密度関数を f とする確率分布に従う．以上をまとめよう．

命題 2. X_k, $k=1,\cdots$ を $U(a,b)$ に従う独立確率変数列，Y_k, $k=1,\cdots$ をそれとは独立な $U(0,M)$ に従う独立確率変数列であるとする．各 k ごとに，もし $Y_k\le f(X_k)$ ならばその X_k を受容する．受容された X_k たちを改めて Z_j, $j=1,\cdots$ とするとそれは密度関数 f を持つ確率分布にしたがう独立確率変数列である．

このアイデアに基づくアルゴリズムは以下の通りである．

アルゴリズム 1. 1. $X\sim U(a,b)$ および $Y\sim U(0,c)$ を生成する．

2. もし $Y\le f(X)$ ならば $Z=X$ とする．

3. Z を掃き出す．

このアルゴリズムに従って，半円分布 (密度関数が $f(x) = \frac{2}{\pi}\sqrt{1-x^2}$ により与えられる $[-1,1]$ 上の分布）に従う独立確率変数列を生成してみよう．Scilab では，ある条件を満たす数列を取り出して新しい数列を作る事は極めて容易であった．それを利用して以下のコードを得る．

コード 2. 棄却法による独立確率変数列の生成：

```
deff('y=f(x)', 'y=2*sqrt(1-x^2)/%pi');

n=100000;
X=2*rand(1,n)-1;
Y=2*rand(1,n)/%pi;
Z=X(Y<=f(X));

xx=-1:0.01:1;
plot(xx, f(xx));
histplot(xx, Z);
```

以上が棄却法の基本的な考え方と実行例である．逆関数法とは異なり，棄却法では 1000 個の一様乱数を生成しても同数の乱数を作り出す事はできない．すなわち，この方法では「効率」が問題になる．1000 個の一様乱数に対してできるだけ多くの乱数を生成したいのである．棄却する個数を減らすための工夫を考えよう．今まで述べたやりかたでは密度関数のグラフ $y = f(x)$ を含むような長方形をとった．長方形をとる理由は，その上の一様分布を簡単に生成できるからである．しかしその結果，前の記号で $\frac{|R\backslash A|}{|R|}$ の割合で棄却される．この値を小さくできれば棄却される割合を減らす事ができる．もしも，例えばある定数 $C > 0$ が存在して $f(x) \leq Ce^{-x}$ がすべての $x \in \mathbb{R}$ で成り立つならば，長方形のかわりに $R = \{(x,y); 0 \leq y \leq Ce^{-x}\}$ ととると $A \subset R$ である．もし R をこのように取る事によって R を長方形にとる場合よりも $\frac{|R\backslash A|}{|R|}$ が小さくなるならば，それだけ棄却される割合が減り，より効率よく乱数を生成できる．このアイデアを命題としてまとめておこう．

命題 3. ある確率密度関数 f に対して正定数 C を $f(x) \leq Ce^{-x}, \forall x \in \mathbb{R}$ を満たすものとする．$X_k,\ k = 1, \cdots$ を $Exp(1)$ に従う独立確率変数列，$Y_k,\ k = 1, \cdots$ をそれとは独立な $U(0, Ce^{-X_k})$ に従う独立確率変数列であるとする．各 k ごとに，もし $Y_k \leq f(X_k)$ ならばその X_k を受容する．受容された X_k たちを改めて $Z_j,\ j = 1, \cdots$ とするとそれは密度関数 f を持つ確率分布にしたがう独立確率変数列である．

証明：作り方より $(X_k, Y_k) \in R$ である．(X_k, Y_k) の joint density $g(x, y)$, Y_k の X_k についての条件付き密度関数を $g(y|x)$, X_k の密度関数を $g(x)$ とすると

$$g(x, y) = g(y|x)g(x) = \frac{1}{Cg(x)}g(x) = \frac{1}{C},$$

すなわち (X_k, Y_k) は R 上一様に分布する．以後の議論は命題 (2) と全く同様である．

このアイデアをアルゴリズムとして提示しておこう．$Y \sim U(0, 1)$ ならば $Ce^{-X}Y \sim U(0, Ce^{-X})$ である事に注意されたい．

アルゴリズム 2. 1. $X \sim Exp(1)$ を生成する．

2. $Y \sim U(0, 1)$ を生成する．

3. もし $Y \leq \dfrac{f(X)}{Ce^{-X}}$ ならば $Z = X$ とする．

4. Z を掃き出す．

このアルゴリズムを利用して標準正規分布 $N(0, 1)$ に従う確率変数列を効率良く生成する事ができる．まず，密度関数

$$f(x) = \sqrt{\frac{2}{\pi}}e^{-\frac{x^2}{2}}1_{[0,\infty)}$$

に従う非負値確率変数 X を生成する事を考えよう．もしそれが実現できたら確率 $\frac{1}{2}$ で X，確率 $\frac{1}{2}$ で $-X$ とする事により $N(0, 1)$ に従う確率変数を生成できる．この $f(x)$ に対して

$$f(x) \leq \sqrt{\frac{2e}{\pi}}e^{-x}$$

である事がわかる．従って，$C = \sqrt{\frac{2e}{\pi}}$ であるから，アルゴリズム (2) の不等式 $Y \leq \frac{f(X)}{Ce^{-X}}$ は

$$Y \leq e^{-\frac{(X-1)^2}{2}}$$

となる．$-\log Y \sim Exp(1)$ だから，X を生成するアルゴリズムは次のようにできる．詳細は演習問題とする．

アルゴリズム 3. 1. $U \sim Exp(1)$ を生成する．

2. $V \sim Exp(1)$ を生成する．

3. もし $U \geq \frac{1}{2}(V-1)^2$ ならば $X = V$ とする．

これを Scilab のコードとして実現したものを以下に示す．

コード 3. 棄却法による正規確率変数の生成．

```
clf()
n=20000;
U=-log(rand(1,n));V=-log(rand(1,n));       //U and V are Exp(1)-distributed.
V=V(U>=(V-1)^2/2);                                  // Acceptance-Rejection.
V=V.*((-1)^(bool2s(rand(1,size(V,'c'))>0.5)));  //Randomizing the sign.
X=- 6:0.1:6;
plot(X, (1/sqrt(2*(%pi)))*exp(-X^2/2));
histplot(X,V)
```

0.5.4 grand による確率変数の生成

- 2 項分布 (binomial distribution) `Y=grand(m,n,'bin',N,p)` によって，m 行 n 列の 2 項分布 $B(N,p)$ に従う確率変数を生成する．

- 負の2項分布 (negative binomial distribution) `Y=grand(m,n,'nbn',N,p)` によって，m 行 n 列の負の2項分布 $NB(N,p)$ に従う確率変数を生成する．

  generates random variates from the negative binomial distribution with parameters N (positive integer) and p (real in (0,1)) : number of failures occurring before N successes in independant Bernouilli trials with probability p of success. Related function(s) : cdfnbn.

- 指数分布 (exponential distribution) `Y=grand(m,n,'exp',Av)` によって，m 行 n 列の指数分布 $Exp(\lambda)$ に従う確率変数を生成する．ただし，Av は確率変数の平均，すなわち $Av = \lambda^{-1}$ である．

- 正規分布 (normal distribution) `Y=grand(m,n,'nor',Av,Sd)` によって，m 行 n 列の指数分布 $N(Av, Sd)$ に従う確率変数を生成する．

- 多次元正規分布 `Y=grand(n,'mn',m,Cov)` によって n 個の平均ベクトル m，共分散 Cov の任意次元の正規分布を生成する．ここで，次元を ℓ とすれば，m は ℓ 個の成分を持つベクトルであり Cov は $\ell \times \ell$ 行列である．

- 幾何分布 (geometric distribution) `Y=grand(m,n,'geom', p)` によって，m 行 n 列の幾何分布 $Geo(p)$ に従う確率変数を生成する．generates random variates from the geometric distribution with parameter p : number of Bernouilli trials (with probability succes of p) until a succes is met. p must be in [pmin,1] multinomial

- 多項分布 (multinomial distribution) `Y=grand(n,'mul',nb,P)` generates n observations from the Multinomial distribution : class nb events in m categories (put nb "balls" in m "boxes"). P(i) is the probability that an event will be classified into category i. P the vector of probabilities is of size m-1 (the probability of category m being 1-sum(P)). Y is of size m x n, each column Y(:,j) being an observation from multinomial distribution and Y(i,j) the number of events falling in category i (for the j th observation) (sum(Y(:,j)) = nb).

- ポアソン分布 `Y=grand(m,n,'poi',a)` によって，m 行 n 列の平均 a のポアソン分布 $Poi(a)$ に従う確率変数を生成する．

- ランダム置換 (random permutations) `Y=grand(n,'prm',vect)` によって，$\{1, 2, \cdots, n\}$ 上の置換の全体の上の一様分布に従う確率変数 (ランダムな置換) を生成する．

0.5.5 対称ランダムウォーク

最後に対称ランダムウォーク (Symmetric Random Walk, 以下 SRW) を生成するコードを与える．これらは，数値計算というよりランダムな道の視覚的なイメージを与えるものである．1 次元 SRW の道 $\{X_n,\, n = 0, 1, 2, \cdots\}$ は，$P(\xi_k = -1) = P(\xi = 1) = 1/2$ であるような i.i.d. 列 $\{\xi_k\}_{k=1,2,\ldots}$ を取って，$X_0 = 0$ かつ

$$X_n = \sum_{k=1}^{n} \xi_k$$

により定まる．以下は 10000 ステップの SWR の Scilab コードの例である．ただし，得られたグラフは横軸に時間，縦軸に SRW の位置をとっている．決して平面を動く SRW のシミュレーションではない事に注意されたい．

```
///// Symmetric Random Walk //////
clf();
t=(0:1:10000)';
xi=2*(rand(10000,1)>=0.5)-1;
S=[0;cumsum(xi)];
plot2d(t,S)
```

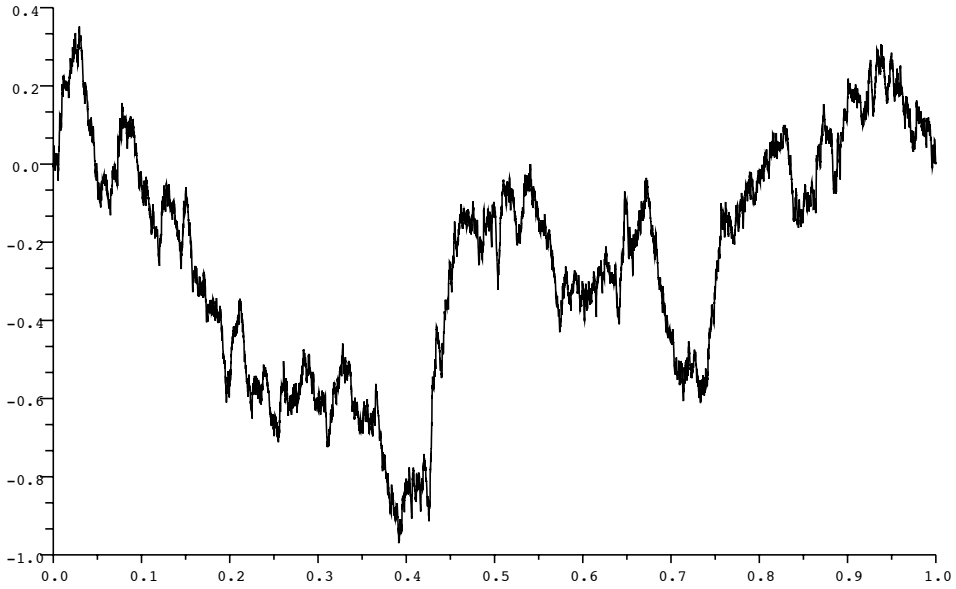
なお，SRW の時間と空間のスケールを

$$X_n^{(N)} = \frac{1}{\sqrt{N}} X_{Nn}$$

のように変換して $N \to \infty$ としたものがブラウン運動の道である．そこで，$N = 10000$ として，上のコードの最後の 1 行を

```
plot2d(t/10000,S/100)
```

とすれば，ブラウン運動を近似するものを見る事ができる．出力例を掲げる．



上記のコードの xi を 2 行にするだけで 2 次元 SRW をシミュレートする事ができる．出力されるグラフは時間パラメータはなく，平面上の SRW の道の軌跡である．

```
/// 2-dim. SRW ///
t=(0:1:10000)';
xi=2*(rand(2,10000)>=0.5)-1
S=[zeros(2,1),cumsum(xi,'c')];
plot2d(S(1,:),S(2,:))
```

0.5.6 Poisson 過程

月の満ち欠けは定期的に起こる．一方宇宙線がいつある計測器に飛んでくるかは全く予想がつかない．1 回飛んできてから次が飛んでくるまでの時間間隔は周期的である事の正反対で，「過去にどれだけの期間こなかったという事と今後いつ飛んでくるかという事は独立である」という性質を持つ確率変数と見なす事が自然である．このような性質を満たす確率変数は**指数分布**に従う．ある時刻から計測を始めて時刻 t までに飛んできた回数を $N(t)$ とすると，$t \to N(t)$ は 1 ずつ増えていく (ランダムな) 階段関数である．このような階段関数値の確率変数を Poisson 過程という．それを描画する Scilab のコードとその出力例を示す．

```
//// Poisson Process ///////
n=20;
lam=1.8
C=grand(1,n,"exp", lam);
t=[0, cumsum(C)];
N=0:n;
clf();
subplot(1,2,1)
plot2d2(t,N)
```

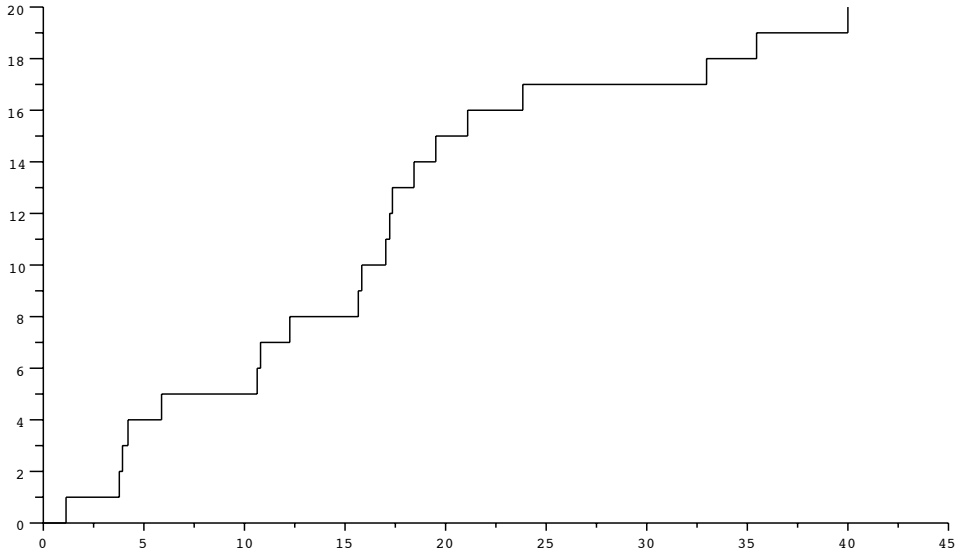
0.6 マルコフ連鎖

0.6.1 マルコフ連鎖とは—-遷移確率と不偏分布

ある $L \times L$ 正方行列 $P = (p(x,y))_{1 \le x,y \le r}$ が確率行列であるとはすべての i, j に対して $p(x,y) \ge 0$ かつ $\sum_{y=1}^{r} p(x,y) = 1$ が成り立つ事である．すなわち，確率行列のすべての成分は 0 以上 1 以下であり，かつ $\mathbf{1}$ をすべての成分を 1 とする縦ベクトルとすると

$$P\mathbf{1} = \mathbf{1} \tag{3}$$

である．言い換えると任意の確率行列は固有値 1 をもち，対応する右固有ベクトルは $\mathbf{1}$ である．



集合 $E = \{1, 2, \cdots, r\}$ に値をとる確率変数の列 $\{X_k\}$ $k = 0, 1, \cdots$ がある確率行列 P を遷移確率行列に持つマルコフ連鎖であるとは

$$P(X_{k+1} = y | X_k = x) = p(x, y), \qquad x, y \in E$$

が成立する事である．このように各遷移確率 $p(x, y)$ を行列で表す事のメリットは n ステップでの遷移確率を考えてみるとすぐに了解できる．Partition Rule より

$$\begin{aligned} P(X_{k+2} = z | X_k = x) &= \sum_{y \in E} P(X_{k+2} = z | X_{k+1} = y) P(X_{k+1} = y | X_k = x) \\ &= \sum_{y \in E} p(x, y) p(y, z) = P^2(x, z) \end{aligned}$$

である．つまり 2 ステップの遷移確率は P^2 から計算できるのである．一般の n ステップの場合も同様に

$$P(X_{k+n} = y | X_k = x) = P^n(x, y) \tag{4}$$

である．

最初に，$1, 2, 3$ 上の遷移確率行列

$$P = \begin{pmatrix} 0.2 & 0.4 & 0.4 \\ 0.3 & 0.3 & 0.4 \\ 0.5 & 0.1 & 0.4 \end{pmatrix}$$

で与えられ，$X(1) = 1$ であるようなマルコフ連鎖の N 個のサンプルを Scilab を用いて生成してみよう．このコードでは離散確率分布 $\mu = (\mu_1, \mu_2, \mu_3)$ をもつ確率変数を 1 行のコード

```
 mu=(0.3, 0.4, 0.3)
[m,X]=max(bool2s(rand(1,1)*ones(1,3)<cumsum(mu)));
```

によって構成する方法を応用している．結果的にかなり短いコードであるが，やや tricky でもある．これとは別にループ文を用いてわかりやすいコードを自分で書いてみる事を奨める．

```
/// Markov Chain Generation

P=[0.2, 0.4, 0.4;0.3, 0.3, 0.4;0.5, 0.1, 0.4];
/// Transition Probability
n=10;  /// The number of steps.
N=20;  /// The number of samples.

X=[ones(N,1),zeros(N,n-1)]; /// The chain starts at 1.
for k=2:n;
  PP=P(X(:,k-1),:);
  W=(bool2s(rand(N,1)*ones(1,3)<cumsum(PP,'c')));
  [m,x]=max(W,'c');
  X(:,k)=x;
end
X
```

以下が実行例である．

```
--> X  =

    1.    2.    2.    1.    2.    2.    2.    1.    3.    1.
    1.    3.    3.    1.    3.    1.    2.    2.    2.    1.
```

```
1.    1.    2.    3.    3.    1.    3.    1.    2.    3.
1.    2.    2.    1.    2.    3.    1.    2.    2.    1.
1.    3.    1.    1.    2.    1.    2.    2.    2.    3.
1.    3.    1.    3.    3.    1.    1.    2.    2.    1.
1.    3.    2.    1.    2.    3.    3.    1.    3.    1.
1.    3.    1.    3.    1.    3.    1.    1.    3.    1.
1.    3.    1.    3.    1.    2.    2.    3.    3.    3.
1.    1.    2.    2.    2.    3.    3.    1.    3.    3.
1.    2.    2.    2.    2.    3.    3.    3.    1.    2.
1.    3.    1.    3.    1.    3.    1.    2.    1.    2.
1.    3.    1.    1.    2.    1.    2.    1.    3.    1.
1.    1.    3.    1.    3.    3.    1.    3.    1.    2.
1.    2.    2.    2.    3.    1.    2.    1.    2.    3.
1.    2.    3.    2.    1.    3.    3.    1.    2.    3.
1.    2.    1.    3.    3.    3.    1.    3.    1.    3.
1.    2.    2.    3.    1.    1.    3.    3.    2.    1.
1.    3.    1.    2.    2.    3.    3.    1.    3.    1.
1.    3.    1.    2.    3.    3.    1.    2.    3.    1.
```

もし $E = \{1, 2, \cdots, r\}$ 上の確率分布 $\pi = (\pi_1, \cdots, \pi_r)$ が存在して，初期分布が π であるマルコフ連鎖の分布がどの時点においても π であるとき，π をそのマルコフ連鎖の不変測度 (invariant measure) という．マルコフ連鎖の遷移確率 $p(x, y)$ と不変分布 π との関係を表してみよう．初期分布を π としたとき，次のステップでマルコフ連鎖が $y \in E$ にいる確率は $\sum_{x \in E} \pi(x) p(x, y)$ である．π が不変分布であるならば，それば $\pi(y)$ に等しい．したがって，すべての $y \in E$ に対して

$$\pi(y) = \sum_{x \in E} \pi(x) p(x, y)$$

である．これを π を横ベクトル $\pi = (\pi_1, \cdots, \pi_r)$，遷移確率 P を行列と思って表すと

$$\pi P = \pi \tag{5}$$

である．言い換えると，不変分布 π は遷移確率行列 P の固有値 1 の左固有ベクトルとして特徴づけられる．

与えられた遷移確率行列に対して，そのマルコフ連鎖の不変分布はScilabのコマンド spec を用いれば容易に求める事ができる．Scilab の Help で spec の項を見ると

```
evals=spec(A)
returns in vector evals the eigenvalues.
[R,diagevals] =spec(A)
returns in the diagonal matrix evals the eigenvalues and in R the right
eigenvectors.
```

とある．

```
P=[0.2, 0.4, 0.4;0.3, 0.3, 0.4;0.5, 0.1, 0.4];
[R,diagevals] =spec(P)
```

と入力すると

```
 diagevals  =

    1.    0      0
    0   - 0.1    0
    0     0    - 3.237D-16
 R  =

    0.5773503    0.6701663  - 0.4850713
    0.5773503    0.2094270  - 0.4850713
    0.5773503  - 0.7120516    0.7276069
```

と返してくる．各列ベクトルがそれぞれ固有値1, -0.1, - 3.237D-16 に対応する右固有ベクトルである．1列目のベクトルはもちろん $\mathbf{1}$ と考えてよい，すなわち式 (3) に対応している．不変分布，すなわち固有値1に対応した左固有ベクトルを求めるためには，$\pi P = \pi \iff P^t \pi^t = \pi^t$ だから，転置行列 P^t の右固有ベクトルを求めればよい．

```
/// Markov Chain ////
```

```
clear
m=3;
stacksize('max')

P=rand(m,m);
r=sum(P,'c').^(-1);
P=diag(r)*P
/// まず確率行列をさだめる.
/// 乱数を利用して一つの確率行列を定めた.

[lvector, eigenvalues]=spec(P');
pi=lvector(:,1)'/sum(lvector(:,1))
/// pi P=pi の解として不変分布 pi を定める.
/// pi は確率分布なので, 正規化したものとして得られる.

nu=rand(1,m); mu=nu/sum(nu)
A=zeros(10,m);
for k=1: 10;
A(k,:)=mu*P^(k-1);
end
A
/// 任意の初期分布 mu に対して 10 ステップまでのマルコフ連鎖の分布を求める.
///A の k 行目が k ステップ目のマルコフ連鎖の分布である.
```

これに対して次のような結果が得られる.

```
 P  = 0.1518796    0.2374068    0.6107136
    0.3587982    0.3157721    0.3254296
    0.0001468    0.4170292    0.5828241
```

```
pi  = 0.1499424    0.3542281    0.4958296

mu  = 0.0529383    0.4342346    0.5128270

A  = 0.0529383    0.4342346    0.5128270
   0.1639181    0.3635509    0.4725309
   0.1554066    0.3507737    0.4938197
   0.1495325    0.3535963    0.4968711
   0.1496536    0.3543656    0.4959808
   0.1499479    0.354266     0.4957861
   0.1499568    0.3542232    0.4958200
   0.1499428    0.3542260    0.4958312
   0.1499417    0.3542282    0.4958301
   0.1499423    0.3542282    0.4958295
```

この結果から，わずか 10 ステップの後マルコフ連鎖の分布は不変分布にかなり近づく事がわかる．この性質を利用して任意の確率分布 π に (近似的に) 従う確率変数を生成する方法をマルコフ連鎖モンテカルロ法 (MCMC 法) という．後の章においてこの方法を詳述する．

上のプログラムの続きとして，遷移確率 P で定まるマルコフ連鎖をのサンプルを多数生成し，100 ステップ目においてそれらが E のどの点にいるか，各点ごとに割合を調べてみよう．

```
N=10000;        /// マルコフ連鎖のサンプル数
n=100;            /// ステップ数.

X=[ones(N,1),zeros(N,n-1)]; /// The chain starts at 1.
for k=2:n;
  PP=P(X(:,k-1),:);
  W=(bool2s(rand(N,1)*ones(1,3)<cumsum(PP,'c')));
```

```
  [mm,Y]=max(W,'c');
  X(:,k)=Y;
end
/// 前に述べたやり方でサンプルを生成する.
/// X(i,k) は i 番目のサンプルの k ステップ目の点.

mu_n=zeros(1,m);
for j=1:m;
  mu_n(j)=sum((X(:,n)==j)*1)/N;
end
mu_n
/// mu_n は N 個のサンプルの n ステップ目に E 上のそれぞれの点
///  にいる割合.
```

結果は,

```
mu_n  =  0.15121    0.35447    0.49432
```

であり，10000 個のサンプルの平均を見ると，10 ステップ目において E 上の各点にいる割合は P の不変分布に非常に近い事がわかる．あるいは，以下のようにマルコフ連鎖の一つのサンプルを非常に多くのステップ数観察して，E 上の各点にどれだけの割合滞在したか (それを経験分布という）を調べる：

```
n=50000;   /// マルコフ連鎖のステップ数.

X=[1,zeros(1,n-1)]; /// The chain starts at 1.
for k=2:n;
  PP=P(X(k-1),:);
```

```
  W=bool2s(rand(1)*ones(1,m)<cumsum(PP,'c'));
  [mm,Y]=max(W);
  X(k)=Y;
end

pii=zeros(1,m);
for x=1:m;  /// For each site of E={1,\cdots,m}
  pii(x)=sum(bool2s(X==x*ones(n)))/n;
end
pii
/// pii はマルコフ連鎖の経験分布.
```

結果として

```
P  = 0.4807292    0.1401419    0.3791289
    0.5962674    0.1234640    0.2802686
    0.4741179    0.4365404    0.0893417

pi  = 0.5041594    0.2186531    0.2771874

pii  = 0.50588    0.21586    0.27826
```

を得る．このようにマルコフ連鎖を長時間観察した際，その経験分布が不変分布に近づく現象を「マルコフ連鎖のエルゴード性」という．

0.6.2 MCMC 法—-マルコフ連鎖を利用した確率変数の生成法

前章において，マルコフ連鎖 $\{X_n\}$ は，多くの場合 n を大きくしていく時 X_n の分布はその不変分布 π に近づいていくという現象 (エルゴード性) を確かめた．この事実を利用して，与えられた分布に従う確率変数列を生成する方法をマルコフ連鎖モンテカルロ法 (Markov Chain

Monte Carlo Method, MCMC 法) という．MCMC 法とは，与えられた確率分布 π に対して，それを不変分布とするマルコフ連鎖 $\{X_n\}$ を生成し，十分大きな n の X_n をもって近似的に π に従う確率変数として生成する方法である．ここでは MCMC 法の一つである Metropolis 法について述べる．

この方法は，有限集合 E 上の確率 π がいわゆる**ギブス分布**である場合に非常に有効である．ギブス分布とは，E 上の関数 U および定数 $\beta > 0$ が与えられて E 上の確率 π が

$$\pi(x) = \frac{1}{Z_\beta} e^{-\beta U(x)}$$

の形で与えられるものである．ただし，Z_β は正規化定数，すなわち $Z_\beta = \sum_{x \in E} e^{-\beta U(x)}$ である．一般には Z_β を具体的に計算する事は困難であるため，$\pi(x)$ を x の関数として表す事はできない．したがって，今まで述べてきた逆関数法や棄却法をそのまま適用する事はできないのである．しかし，今から述べるように，MCMC 法 (Metropolis 法) を用いるとギブス分布に近似的に従う確率変数を生成する事ができる．

MCMC 法の原理は次の命題に要約される．

命題 4. $q = q(x, y)$ を任意の既約な E 上の遷移確率とし，任意の $x, y \in E, x \neq y$ に対して

$$\alpha(x, y) = \frac{\pi(y) q(y, x)}{\pi(x) q(x, y)} \wedge 1 \tag{6}$$

とおき，新しい E 上の遷移確率 $p = p(x, y)$ を

$$p(x, y) = \begin{cases} q(x, y)\alpha(x, y) & x \neq y \\ 1 - \sum_{z \neq x} q(x, z)\alpha(x, z) & x = y \end{cases} \tag{7}$$

と定義する．その時，π は p に関して可逆分布である，すなわち任意の $x, y \in E$ に対して

$$\pi(x) p(x, y) = \pi(y) p(y, x) \tag{8}$$

である．特に，π は p で定まるマルコフ連鎖の不変分布である．

証明：$x = y$ の時には主張は自明である．$x \neq y$ である時は

$$\begin{aligned} \pi(x)p(x,y) &= \pi(x)q(x,y)\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)} \wedge 1 \\ &= \pi(y)q(y,x) \wedge \pi(x)q(x,y) \end{aligned}$$

であるが，この右辺は x と y を置き換えても変わらない事に注意すればよい．最後に，式 (8) の両辺を x について和をとって得られる

$$\sum_{x \in E} \pi(x)p(x,y) = \sum_{x \in E} \pi(y)p(y,x) = \pi(y)$$

より，π は p で定まるマルコフ連鎖の不変分布である事がわかる．

□

最初にとった遷移確率 q から定まるマルコフ連鎖に対し，新しい遷移確率 p から定まるマルコフ連鎖は次のように定まる事がわかる：

アルゴリズム 4. *Metropolis* 法：

1. ひとつの遷移確率 $q(x,y)$ を選び $\alpha(x,y)$ を以下で定める．

$$\alpha(x,y) = \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)} \wedge 1$$

2. X_n に対して Y を分布 $q(X_n, \cdot)$ に従って生成する．

3. $U \sim U(0,1)$ を生成し，X_{n+1} を以下のように定める：

$$X_{n+1} = \begin{cases} Y, & if\ U \leq \alpha(X_n, Y) \\ X_n, & if\ U > \alpha(X_n, Y). \end{cases}$$

すなわち，遷移確率 p で定まるマルコフ連鎖は各ステップごとに x にいるマルコフ連鎖が遷移確率 q で $y \neq$ に移動すると決まった際，$\alpha(x,y)$ を求めて，その確率で実際に y に移動し，確率 $1 - \alpha(x,y)$ で次も x に留まる，と定めたものである．以上のアルゴリズムを Scilab で実現したものが次のコードである．

コード 4. Metropolis Algorithm.

```
Q=rand(3,3);
r=sum(Q,'c').^(-1);
Q=diag(r)*Q
   /// 遷移確率行列 Q は何を選んでもよい．ここではランダムに選んでみる．

pi=[0.3, 0.4, 0.3];
   /// pi を不変分布とする遷移確率行列を作る．

A=min(inv(diag(pi))*(Q'./Q)*diag(pi),1);
// 各 x,y ごとに alpha(x,y) を
//for x=1:3;
//  for y=1:3;
//    A(x,y)=min(pi(y)*Q(y,x)/(pi(x)*Q(x,y)),1);
//  end
//end
/// のように求めるものを，Scilab の特徴を生かして行列として 1 行で求めている．



R=A.*Q-diag(diag(A.*Q));
P=R+diag(ones(3,1)-sum(R,'c'))
   // MCMC 法で定めた新しい遷移確率．

n= 3;  /// マルコフ連鎖のステップ数．

nu=rand(1,3);
mu=nu/sum(nu)
mu_n=mu*P^n
/// mu は勝手に (ランダムに) 与えた初期分布である．mu_n は n ステップ目の
/// マルコフ連鎖の分布．
```

Scilbal でシミュレーションを試みると以下の結果を得た．

```
Q  = 0.0965179    0.3768874    0.5265947
     0.2807076    0.4294808    0.2898116
     0.3078084    0.4057739    0.2864177

P  = 0.3179148    0.3742767    0.3078084
     0.2807076    0.4294808    0.2898116
     0.3078084    0.3864155    0.3057761

mu  = 0.4211674    0.4246213    0.1542113

mu_n  =  0.3000008    0.3999987    0.3000004
```

この場合わずか 3 ステップで，マルコフ連鎖の分布は不変分布に非常に近い事がわかる．

この節の最後に，なぜ π がギブス分布の場合に MCMC 法が特に有効な手段であるのかを説明しておこう．有限集合 E 上の関数 U および任意のパラメータ $\beta > 0$ に対して E 上のギブス分布 π を

$$\pi_\beta(x) = \frac{1}{Z_\beta} e^{-\beta U(x)} \tag{9}$$

とする．ただし，Z_β は (11) 同様 $Z_\beta = \sum_{x \in E} e^{-\beta U(x)}$ である．

ここでは各 $x \in E$ の近傍の数は x によらず一定な N であるとし，遷移確率 $q(x, y)$ として x の近傍 y に一様な確率で遷移する確率を定めるものとする．すなわち，

$$q(x,y) = \begin{cases} \frac{1}{N} & \text{もし } y \text{ は } x \text{ の近傍である場合} \\ 0 & \text{その他の場合} \end{cases}$$

であるとする．この時 $q(x, y) = q(y, x)$ である．また，

$$\frac{\pi_\beta(y)}{\pi_\beta(x)} = e^{-\beta_n(U(y)-U(x))}$$

であるから，メトロポリスアルゴリズムは以下のようになる．

アルゴリズム 5. ギブス分布に対するメトロポリス法：

0. E 上の遷移確率 q を

$$q(x,y) = \begin{cases} \frac{1}{N} & \text{もし } y \text{ は } x \text{ の近傍である場合} \\ 0 & \text{その他の場合} \end{cases}$$

により定める．

1. X_n に対して Y を分布 $q(X_n, \cdot)$ に従って生成する．

2. $\xi \sim U(0,1)$ を生成し，X_{n+1} を以下のように定める：

$$X_{n+1} = \begin{cases} Y, & \text{if } \xi \leq e^{-\beta \cdot \max\{U(Y)-U(X_n),0\}} \\ X_n, & \text{if } \xi > e^{-\beta \cdot \max\{U(Y)-U(X_n),0\}}. \end{cases}$$

ここで，アルゴリズムを定めるために正規化定数 Z_β は必要ない．言い換えれば，Z_β が具体的にわからなくてもアルゴリズムを定める事ができる．これが，ギブス分布の場合に MCMC 法が特に有効である理由である．

関数 U とそこからメトロポリス法によって定まるマルコフ連鎖の関係を見ておこう．$x \in E$ にいるマルコフ連鎖は，各ステップごとにまず x の近傍の点のうちの 1 点 $y \in E$ を等確率で選ぶ．もし $U(y) \leq U(x)$ ならば，そのまま次のステップは y に進むが，$U(y) > U(x)$ ならば確率 $e^{-\beta(U(y)-U(x))}$ で y に進み，残りの確率で x に留まる．すなわち，$U(y)$ の値の大きい y には行きにくくなる．したがって，関数 $z = U(x)$ のグラフを E 上の「標高」と捉えると，次のような直感と合致する認識を得る事ができる．

- マルコフ連鎖は「登る」のはしんどいので，登らずに休む機会が多い．その傾向は標高差が大きいほど，また β が大きいほど強くなる．
- 「下る」時は休まず下る．

一方，このマルコフ連鎖の不変分布は π であるから，長いステップ数の間にはマルコフ連鎖

は $\pi(x)$ の最も大きい点，すなわち $U(x)$ の最小値をとる点 x_0 に最も長い時間滞在する．したがって，図 () の x_1 のように，x_0 に到達するためには山を登らなければならない点から出発するマルコフ連鎖は次のような挙動をする．U の極大値をとる点を「峠」と呼ぶ事にする．

命題 5. 准安定性 *(metastability)* と呼ばれる次の事実が成り立つ．

1. x_1 から出発するマルコフ連鎖は休み休みゆっくり山を登る．峠 x_2 に到着したらあとは休みなく移動し，いつか x_0 に到着する．

2. β が大きい時，x_1 から峠 x_2 にいたる時間 τ の期待値 $E[\tau]$ は β について指数オーダーである．正確には

$$\lim_{\beta\to\infty}\frac{1}{\beta}\log E[\tau] = U(x_2) - U(x_1)$$

である．一方，x_2 から x_0 にいたる時間の期待値は $O(1)$ である．

もちろん **2.** の結論は自明ではなく，深い解析に基づいて理解される結果である．

0.7　格子確率モデルのシミュレーション

0.7.1　イジング模型—MCMC 法 (Metropolis 法) による実現

イジング模型 (Ising model) とは強磁性体の相転移や臨界現象を説明するために導入されたモデルである．イジング模型を一言で言えば，ある確率に従う平面の格子上の 1 または-1 に値を取る確率変数（スピン変数）の集まりである．ただし，各格子上のスピン変数はコイン投げのように独立ではなく，隣り合うスピン変数の値に影響を受けあいながら決まる．したがって空間的な相互作用が存在するモデルである．

このようなモデルは，単に磁性体の研究のみならず，さまざまな理工学の問題を考察する上で有用なモデルである．例えば，交通渋滞を調べるモデルとして各格子上に車がいる事をスピン変数 1 とし，いない事を 0 で表現してそれの確率的な時間発展を調べる事も可能になる．あるいは格子の各点に黒または白の色がのっているデジタル画像のノイズ除去の問題を考察する舞台にもなる．ここでは前節に述べた Metropolis 法の考え方を用いてイジング模型をシミュレートしてみよう．

ここでは2次元のイジング模型を考える．まず記号や言葉の準備から始める．平面内の(非常に大きいが) 有限の格子系を $\Lambda = \{x = (x_1, x_2); x_1, x_2 \in \mathbb{Z},\ 0 \leq x_1, x_2 \leq L\}$ とする．$x = (x_1, x_2) \in \Lambda$ と Λ における x の最近接の点，すなわち $(x_1, x_2 - 1)$, $(x_1, x_2 + 1)$, $(x_1 - 1, x_2)$, $(x_1 + 1, x_2) \in \Lambda$ のいずれかとの組み $\langle x, y\rangle$ をボンドと呼ぶ．ボンドの集まりを $\Lambda*$ と書く．各 $x \in \Lambda$ におけるスピン σ_x は $\{-1, 1\}$ に値を取るとする．各点でのスピンの集まり $\sigma = (\sigma_x)_{x\in\Lambda} \in \{-1, 1\}^L$ を配置という．配置のすべての集まり $\{-1, 1\}^L$ を配置空間という．

h, J を正の定数とする．一つの配置 $\sigma = (\sigma_x)_{x\in\Lambda}$ に対して配置のエネルギー $H(\sigma)$ を

$$H(\sigma) = -\frac{J}{2} \sum_{\langle x,y\rangle\in\Lambda*} \sigma_x\sigma_y - \frac{h}{2} \sum_{x\in\Lambda} \sigma_x \tag{10}$$

とし，配置 σ が起こる確率 $P(\sigma)$ を，任意の $\beta > 0$ に対してギブス分布

$$P(\sigma) = \frac{1}{Z_\beta} e^{-\beta H(\sigma)} \tag{11}$$

により定める．Z_β は正規化定数，すなわち $Z_\beta = \sum_\sigma e^{-\beta H(\sigma)}$ である．β は物理学的には温度の逆数，$\beta = 1/T$ の意味を持つ値である．

さて，今までさまざまな確率分布を考え，それに従う確率変数を生成する問題を考えてきた．ここでも，このギブス分布に従う確率変数，すなわちイジングモデルを生成に取り組もう．MCMC 法を導入した際，ギブス分布のシミュレーションには MCMC 法が有効であるという事実を述べた．ここでは，その理由を述べつつこのアイデアを実行していこう．すなわち，ここでの配置空間に値をとるマルコフ連鎖であって，その不変分布が (11) であるものを構成する．まず，最初に考える遷移確率 $q(\sigma, \sigma')$ を導入する．そのために，配置空間における近傍系を定義する．$\sigma' \in \{-1, 1\}^L$ が $\sigma \in \{-1, 1\}^L$ の近傍であるとは，ある格子の一つの点 $s \in \Lambda$ があって，s 以外の点 y における $\sigma'(y)$ と $\sigma(y)$ は一致しており，s においてのみスピンが反転 (flip) しているようなものであるとする，すなわち

$$\sigma' \sim \sigma \iff \begin{cases} \sigma'(s) = -\sigma(s) & \exists s \in \Lambda \\ \sigma'(s) = \sigma(s) & \forall y \neq s \end{cases} \tag{12}$$

とする．配置空間における近傍を以上のように定めると，遷移確率 q として σ からその近傍 σ' に等確率に遷移するものをとる．すなわち

$$q(\sigma, \sigma') = \begin{cases} \frac{1}{\sigma \text{の近傍の個数}}, & if\ \sigma' \sim \sigma \\ 0, & otherwise \end{cases}$$

よって，$\sigma \sim \sigma'$ であるとき，式 (6) により，

$$\begin{aligned} \alpha(\sigma, \sigma') &= \frac{\pi(y)}{\pi(x)} \wedge 1 \\ &= e^{-\beta(H(\sigma')-H(\sigma))} \wedge 1 = e^{-\beta \cdot \max\{H(\sigma')-H(\sigma), 0\}} \end{aligned}$$

である．ここで，π が分母と分子に現われるため，正規化定数 Z_β が消去されている事に注意されたい．このように，ギブス分布の正規化定数が具体的にわからない場合にも α を求める事ができるので，その分布に従う乱数をシミュレートが可能になるのである．

まとめると，遷移確率 $p(\sigma, \sigma') = q(\sigma, \sigma') e^{-\beta \cdot \max\{H(\sigma')-H(\sigma), 0\}}$ によって定まる配置空間上のマルコフ連鎖の不変分布はギブス分布 π である，すなわち以下のアルゴリズムに従う配置空間上のマルコフ連鎖の不変分布がギブス分布 (11) である事がわかった．

ただし，これから直ちにこのアルゴリズムによって生成されるマルコフ連鎖 X_n の分布が $n \to \infty$ とした時必ず π に近づくと保障されているわけではない．すなわち，このマルコフ連鎖のエルゴード性を検証する必要がある．次の結果が知られている．

アルゴリズム 6. イジング模型を不変分布とするマルコフ連鎖の生成：

1. Λ の一点 s を一様に選ぶ．

2. $X_n = \sigma$ に対して点 s のスピンを反転させた配置を σ' とする．

3. $U \sim U(0,1)$ を生成し，X_{n+1} を以下のように定める：

$$X_{n+1} = \begin{cases} \sigma', & \text{if } U \leq e^{-\beta \cdot \max\{H(\sigma')-H(\sigma), 0\}} \\ \sigma, & \text{if } U > e^{-\beta \cdot \max\{H(\sigma')-H(\sigma), 0\}}. \end{cases}$$

配置 σ に対して $s \in \Lambda$ のスピンを反転させた配置を σ' とすると

$$H(\sigma') - H(\sigma) = J\sigma_s \sum_{y;\langle y,s \rangle} \sigma_y + h\sigma_s = \sigma_s \Big(\sum_{y;\langle y,s \rangle} \sigma_y + h \Big)$$

である．上のアルゴリズムに従ってイジング模型をシミュレートした Scilab のコードの一例を次に示す．

コード 5. +1 境界条件付きの 2 次元イジング模型

```
clear
clf()
stacksize('max');
L=100;  // The size of the spin system is L*L.
b=10;   // b=beta, the inverse of the temperature.

J=1;
h=0.01;
// The energy H is defined by
// H(sigma)=-(J/2)sum_<x,y>sigma_x・sigma_y-(h/2)sum_x sigma_x
// and the P is given by
// P(sigma)=(1/Z_b)exp(-b H(sigma)).

n=10000;  // The steps of the Markov chain

S=ones(L+2,L+2);
S(2:L+1,2:L+1)=(rand(L,L)>=1/2)*2-1; //The initial state with 1-b.c..

// The state space of the M.C. is E={-1,1}^{L^2},
// For Q, we take the independent sampler, which is
// uniform on E.  Thus q(x,y)=const, for all x,y\in E.
// Therefore, the Metropolis alpha(x,y)=pi_y/pi_x.

for k=1:n-1;
W1=(rand(1)<(0:1/L:1))*1;
[m,e1]=max(W1,'c');
W2=(rand(1)<(0:1/L:1))*1;
[m,e2]=max(W2,'c');
// (e1,e2) takes values in {2,\dots,L+1}^2 randomly.
```

```
// You flip the spin at (e1,e2).

Y=S;
Y(e1,e2)=Y(e1,e2)*(-1);
// Y is the fliped config. of S.

H=J*S(e1,e2)*...
(S(e1-1,e2)+S(e1+1,e2)+S(e1,e2-1)+S(e1,e2+1))+h*S(e1,e2);
alpha=exp(-b*max(H,0));
U=rand(1,1);
S=bool2s(U<=alpha)*Y+bool2s(U>=alpha)*S;
end
Matplot(S*10)
```

補足：グラウバー力学の准安定性 (metastability).

ここで，グラウバー力学について前節に説明した准安定性をシミュレートしてみよう．ここで，標高を意味する関数は H である．H の最小値をとる点（配置）を求めてみよう．H が小さいためには $\sum_{\langle x,y\rangle\in\Lambda*}\sigma_x\sigma_y$ および $\sum_{x\in\Lambda}\sigma_x$ が大きければよい．前者は，各ボンド $\langle x,y\rangle$ について σ_x と σ_y が同符号であれば $\sigma_x\sigma_y$ が大きいから，そのようなボンドが多いほど大きい．後者は $\sigma_x=1$ である点 x が多いほど大きい．以上の事から $H(\sigma)$ が最小である配置はすべての点 x で $\sigma_x=1$ である配置 ⊞ である．したがって，β が大きい時 (温度が低い時)，メトロポリス法 (アルゴリズム (6)) によって定めたマルコフ連鎖はどのような初期配置から出発しても長時間の後ほとんどの時刻で H の最小点の配置 ⊞ にいるであろう．

そこで，初期配置としてすべてのスピンが -1 である配置 ⊟ を考えよう．この配置から出発するマルコフ連鎖はいつか必ず配置 ⊞ に到達する．では，温度が低温であるとき，すなわち β が非常に大きい時前節で述べた准安定性 (metastability) の現象，配置 ⊟ から出発してある峠の配置に到達するまでは非常に時間がかかり，峠から配置 ⊞ まではあっという間に到達するという現象が表れるか考えてみよう．

まず，配置のエネルギー H について容易に次の事を確かめる事ができる．

- $+1$ のスピンの数が一定の配置の中では，それらが一つの正方形をなすとき最も H の

値が低い．

- $0 \leq \ell < L$ に対し，$+1$ スピン達が 1 辺の長さが ℓ の一つの正方形であるような任意の任意の配置を $\sigma_{\ell\times\ell}$ とする．正方形は Λ の中のどこにあってもよい．その時

$$\ell \to H(\sigma_{\ell\times\ell}) - H(\boxminus) = J\cdot 4\ell - h\cdot\ell^2$$

である．これは

$$0 < h < 2J$$

の仮定の下で $\ell_c = \lceil \frac{2J}{h} \rceil$ において最大値をとる．$\lceil\cdot\rceil$ は小数点以下の切り上げを意味する．

これらの考察より，ここでの「峠」の配置 σ_c は，サイズが ℓ_c のひとつの正方形の内部およびその隣の一つの点で $+1$ スピンであり，外部で -1 スピンであるような配置である (図 ())．(正方形は Λ のどこにあってもよいので峠の配置はひとつではない．) したがって次のような准安定性の現象を見る事ができるであろう．

命題 6. エネルギー H について上の仮定が成り立つものとする．初期配置が $\boxminus$ であって，メトロポリス法 *(6)* によって時間発展するマルコフ連鎖が一つの峠の配置 σ_c に到達するまでの時間 τ の期待値は β について指数オーダーである．より正確には

$$\lim_{\beta\to\infty}\frac{1}{\beta}\log E[\tau] = H(\sigma_c) - H(\boxminus) = 4J\ell_c - h(\ell_c^2 - \ell_c + 1)$$

である．一方峠の配置 σ_c から安定配置 $\boxplus$ に到達する時間は $O(1)$ である．

この現象を観察できるような Scilab Code を以下に示す．

```
/// Simulated Annealing in Glauber dynamics


clear
clf()
```

```
stacksize('max');
L=50;  // The size of the spin system is L*L.

J=8.2;
h=4;
// The energy H is defined by
// H(sigma)=-(J/2)sum_<x,y>sigma_x · sigma_y-(h/2)sum_x sigma_x
// and the P is given by
// P(sigma)=(1/Z_b)exp(-b H(sigma)).

n=200000;  // The steps of the Markov chain

S=-ones(L+2,L+2);
/// (-1)-boundary condition.
subplot(4,6,1);
Matplot(S(:,:)*10)



for k=1:n;
W1=(rand(1)<(0:1/L:1))*1;
[m,e1]=max(W1,'c');
W2=(rand(1)<(0:1/L:1))*1;
[m,e2]=max(W2,'c');

Y=S;
Y(e1,e2)=Y(e1,e2)*(-1);

betaa=0.14;

H=J*S(e1,e2)*...
```

```
(S(e1-1,e2)+S(e1+1,e2)+S(e1,e2-1)+S(e1,e2+1))+h*S(e1,e2);
alpha=exp(-betaa*max(H,0));
U=rand(1,1);
S=bool2s(U<=alpha)*Y+bool2s(U>=alpha)*S;

m=int(n/24);
ell=int(k/m);
if k/m-ell==0;
subplot(4,6,ell+1);
Matplot(S(:,:)*10)
end;  // of if k/100-ell==0
if sum(S)==L^2;
  break;
end;
end  //for k=1:n-1;
```

0.8 モンテカルロシミュレーションによる最適化問題へのアプローチ

読者の皆さんは「巡回セールスマン問題」という言葉を聞いた事があるのではないかと思う．これは，セールスマンが所定の数の都市を 1 回づつすべて訪問してもとの場所に戻ってこなければならない時， どのような順序で訪問すれば移動距離を最短にできるか，という問題である．数学的には，すべての訪問の順序を集めた集合 E 上の，それぞれの順序に応じた移動距離という関数 f の最小値を求めるという単純な問題である．したがって，その答えは明らかである．n 都市を巡回する順序の組合せ $(n-1)!/2$ 通りのすべてについて移動距離を算出し，その中の最小値をとる順序を選び出せばよい．しかし，この答えは実用的には何の価値もないものである事がわかる．なぜなら，ここで考えたい都市の数 n は最低でも千のオーダーである．ところが，例えば $n = 30$ 程度でも $n!$ は巨大な数であり，この計算には天文学的時間がかかってしまうのである．何の工夫もせずすべての場合の関数の値を調べるのでは

なく，もっと効率的にこの問題の解を探す方法はないのだろうか．あるいは，厳密な最小値を求める事はあきらめて近似的に求める換わりに，計算時間を短くすませる事はできないだろうか．このように有限ではあるが巨大な数の元からなる集合 E 上の関数の最小値を取る点を効率的に探す問題を**最適化問題**と言う．巡回セールスマン問題に限らずさまざまな実用的な最適化問題の解法を探す学問分野をオペレーションリサーチ (OR) という．

ここでは，厳密に最短移動距離とそれを実現する巡回の順序を求める事をあきらめ，確率的にそれらを求める様々な方法考える．確率的に求める，という意味は乱数を発生させて最適値を推測し，その推測値が真の推測値に十分近い値である事を十分に高い確率で成立する事を大数の法則および中心極限定理によって保障するという事である．その意味では，これもモンテカルロシミュレーションの一つである．

0.8.1 焼きなまし法 (Simulated Annealing)

このようなモンテカルロシミュレーションを用いた最適化問題へのアプローチにはいくつかある．そのの一つが「焼きなまし法」(Simulated Annealing Method) である．実はこの焼きなまし法は，今までのべてきた MCMC 法の考え方から派生する方法である．この章ではこれについて述べていこう．

焼きなまし法とは，マルコフ連鎖の長時間挙動の性質を利用して最適化問題の階を推測する手法である．巨大な有限集合 E 上の関数 U の最小値を探すために，任意のパラメータ β に対して E 上のギブス分布

$$P_\beta(x) = \frac{1}{Z_\beta} e^{-\beta U(x)} \tag{13}$$

を考える．ただし，Z_β は (11) 同様 $Z_\beta = \sum_{x \in E} e^{-\beta U(x)}$ である．ここで注目すべき事実は，前に述べたように，β が大きいほど，$U(x)$ の小さい x 上の確率の重みが大きくなるという事である．簡単のために，U の最小値をとる点 x_0 は唯一つであると仮定しよう．β が非常に大きい状況では，x_0 の確率 $P(x_0)$ は 1 に非常に近く，それ以外の配置をとる確率はほとんど 0 である．言い換えれば，ギブス分布 P_β に従う確率変数の値は，$\beta \to \infty$ とした時，ほぼ確率 1 で x_0 である．焼きなまし法はこの事実に着目して x_0 を見つけ出す手法である．

絵

前章で述べた MCMC 法により，各 $\beta > 0$ ごとにギブス分布 P_β を不変分布とするマルコフ連鎖 X_n, $n = 1, 2, \cdots$ を構成する事ができる．マルコフ連鎖の分布は，$n \to \infty$ とする時不変分布に近づいて行くから，十分おおきな n に対して X_n をとることにより，近似的に P_β

に従う確率変数を生成する事ができる．一方，先ほど述べたように $\beta \to \infty$ とする時，P_β の分布の重みは U の最小値を取る点 x_0 に集中する，すなわち $P_\beta \to \delta_{x_0}$ である．以上より，$n \to \infty$ とする事と $\beta \to \infty$ とする事を同時に実現すれば，マルコフ連鎖の動きを見る事により x_0 を探し出す事ができるはずである．

そこで，β を n に依存して変化するものとし，$n \to \infty$ とする時 $\beta(n) \to \infty$ となるものをとる．それに応じて，Metropolis 法（命題 (4)）において定める遷移確率も，ここでは n に依存して定める．すなわち，(6) における π を各ステップ n ごとに $P_{\beta(n)}$ によっておきかえて遷移確率 $p_n(x,y)$ を定めるのである．(このように n ごとに異なる遷移確率によって定まるマルコフ連鎖を時間的に非斉次な (time-inhomogeneous) マルコフ連鎖という．)

MCMC 法の処方箋 (命題 (4) を思い出そう．ここでは各 $x \in E$ の近傍の数は x によらず N であるとし，遷移確率 $q(x,y)$ として x の近傍 y に一様な確率で遷移する確率を定めるものとする．すなわち，

$$q(x,y) = \begin{cases} \frac{1}{N} & \text{もし } y \text{ は } x \text{ の近傍である場合} \\ 0 & \text{その他の場合} \end{cases}$$

であるとする．この時 $q(x,y) = q(y,x)$ である．また，

$$\frac{P_{\beta(n)}(y)}{P_{\beta(n)}(x)} = e^{-\beta_n(U(y)-U(x))}$$

であるから，$b(n)$ を具体的にどのように取るかという問題をひとまず置くと，焼きなまし法のアルゴリズムは以下のようになる．

アルゴリズム 7. *Simulated Annealing*：

0. E 上の遷移確率 q を

$$q(x,y) = \begin{cases} \frac{1}{N} & \text{もし } y \text{ は } x \text{ の近傍である場合} \\ 0 & \text{その他の場合} \end{cases}$$

により定める．

1. X_n に対して Y を分布 $q(X_n,\cdot)$ に従って生成する．

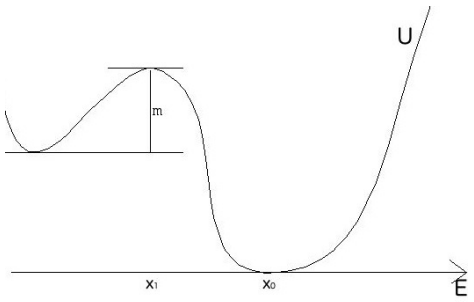
2. $\xi \sim U(0,1)$ を生成し，X_{n+1} を以下のように定める：

$$X_{n+1} = \begin{cases} Y, & \text{if } \xi \leq e^{-\beta_n \cdot \max\{U(Y)-U(X_n),0\}} \\ X_n, & \text{if } \xi > e^{-\beta_n \cdot \max\{U(Y)-U(X_n),0\}}. \end{cases}$$

すなわち x にいるマルコフ連鎖は，各ステップごとにまず x の近傍の点のうちの 1 点 y を等確率で選ぶ．もし $U(y) \leq U(x)$ ならば，そのまま次のステップは y に進むが，$U(y) > U(x)$ ならば確率 $e^{-\beta_n(U(y)-U(x)}$ で y に進み，残りの確率で x に留まる．したがって，$U(y)$ の値の大きい y には行きにくくなる．しかも，$b_n \to \infty$ なので，ステップ数が進むにつれて，ますますその傾向は強まる事になる．

では，このアルゴリズムにおいてどのように β_n を選べば，ほぼ確率 1 で $X_n \to x_0$ になるか考えてみよう．この問題に対して深い解析に基づく結果が知られている．ここでそれを細部にわたって紹介する事はできないが，いくつかの要点を記しておきたい．

1. このアルゴリズムは β を n に関して変化しない場合の，生成したマルコフ連鎖 X_n のエルゴード性を大前提にしている．したがって，ここで β を n に応じて変化させるとしても，そのさせ方は急激ではなく穏やかなものであるべきである．実際，ある定数 $c > 0$ に対して $\beta_n = c\log(1+n)$ とする事が適切である事がわかる．



2. 関数 U のグラフの形状が $X_n \to x_0$ の収束のスピードにどのような影響をあたえるだろうか．例えば U のグラフが図 () のようである場合，極小点 x_1 から出発するマルコフ連鎖が最小点 x_0 に到着するためには，高さ m の「峠」を超えなければならない．先ほど述べたようにマルコフ連鎖は U の高い点には移動しにくい．したがって，m が大きいほど x_0 への到着が遅くなる，あるいはいつまでも x_0 に到着しない確率が大きくなるだろう．よって m の値が大きいほど β_n の発散するスピードは遅くするべきである事がわかる．実際，$c = \frac{1}{m}$ とする事が適切である事がわかる．

以上の事柄を命題としてまとめておこう．

命題 7. *(Stroock-Diaconis)* E 上の関数 U は唯一つの点 x_0 において最小値を取るものとする．また，U の *communication height(*E 上の各点から x_0 に到達するために超えなければならない峠の高さの最大値*)* を m とする．その時，焼きなまし法のアルゴリズム *(7)* において

$$\beta_n = \frac{1}{m}\log(1+n)$$

によって定まるマルコフ連鎖 $\{X_n\}$ に対して

$$\lim_{n\to\infty} P(|U(X_n) - U(x_0)| \geq \delta) = 0, \quad \forall \delta > 0$$

が成り立つ．

0.8.2 巡回セールスマン問題への適用

この章の序で述べたように，巡回セールスマン問題とはセールスマンが所定の数の都市を 1 回づつすべて訪問してもとの場所に戻ってこなければならない時，どのような順序で訪問すれば移動距離を最短にできるか，という問題である．都市の数を n とするとき，訪問の順序の全体の集合 E は $\{1, 2, \cdots, n\}$ の置換の全体 (すなわち置換群 S_n) である．一つの訪問の順序 X に対してそれに伴う移動距離 $U(X)$ の最小値を焼きなまし法を用いて推定してみよう．

まず置換の全体 S_n 上の近傍を決めよう．ここでは各 $X = (x_1, x_2, \cdots, x_n) \in S_n$ に対して X の近傍を次のように取る．2 つの数 $1 \leq a < b \leq n$ に対して X の $(x_a, \cdots, x_b)$ の部分を反転させて作った置換 $X_{a,b}$ の全体を X の近傍とする．正確に述べると，

$$X_{a,b} = (x_1, \cdots, x_{a-1}, x_b, x_{b-1}, \cdots, x_{a+1}, x_a, x_{b+1}, \cdots, x_n)$$

として，X の近傍の全体を $\{X_{a,b}; 1 \leq a < b \leq n\}$ とする．このように定めた近傍系に対してアルゴリズム (7) の考え方に従って巡回セールスマン問題の確率的近似解を求める Scilab によるプログラムの一例を以下に示す．

```
//// Travelling Salesman, by Simulated Annealing
```

```
n=10;          /// The number of sites a salesman visits.
N=2000;       /// The number of steps of the Markov chain.

/// n 点間の距離を乱数を用いて定める．点 k と点 j の距離を
/// kj 成分とする行列を Cost とする．Cost は対角成分が 0 の
/// 対称行列である．
A=grand(n,n,'geom', 0.2);
C=A+A';
Cost=C-diag(diag(C))

/// 上に定めた距離に対して，置換 X に対応する総距離 S を定める．
function S=dist(X)
  S=0;
  for j=1:n-1;
    S=S+Cost(X(j),X(j+1));
  end
  S=S+Cost(X(n),X(1))
endfunction

T=0.3*ones(1,N)+0.1*log([1:N]);
///Annealing Schedule.

X=zeros(n+1,N);
/// k 列目の 1〜n 行目がマルコフ連鎖の k ステップ目 X_k,
/// n+1 行目がその総距離 S_k を表す．

X(1:n,1)=grand(1,'prm',[1:n]');
// 初期変数 X_1 をランダムに選ぶ．

/// 以下，アルゴリズムに従いマルコフ連鎖を生成する．
```

```
/// (1,2, cdots, n) から 2 個の異なる数をランダムに選ぶ.
for k=1:N;
U=(rand(1)<cumsum(ones(1,n)/n))*1;
[m,a]=max(U);
A=ones(1,n);A(a)=0;
V=(rand(1)<cumsum(A/(n-1)))*1;
[m,b]=max(V);
c=gsort([a,b],'g','i');

/// X の一つの近傍 Y を選ぶ.
Y=[X(1:c(1)-1,k);X(c(2):-1:c(1),k);X(c(2)+1:n,k)];

/// 以下 Metropolis algorithm の核心部分.
if dist(Y)<dist(X(1:n,k));
   X(1:n,k+1)=Y;
   X(n+1,k+1)=dist(Y);
else
  if rand(1)<=exp(-(dist(Y)-dist(X(1:n,k))/T(k)));
  X(1:n,k+1)=Y;
  X(n+1,k+1)=dist(Y);
  else X(:,k+1)=X(:,k);
  end
end
end
X
```

プログラムの実行結果を見てみよう．各都市間の距離を与える行列 Cost が

```
Cost  =

    0.    21.    6.    14.    9.    12.    3.    7.    13.    8.
   21.    0.    13.    6.    9.    6.    31.    6.    9.    11.
    6.    13.    0.    14.    18.    19.    10.    2.    10.    21.
   14.    6.    14.    0.    8.    10.    7.    16.    20.    5.
    9.    9.    18.    8.    0.    11.    6.    21.    3.    3.
   12.    6.    19.    10.    11.    0.    7.    6.    12.    11.
    3.    31.    10.    7.    6.    7.    0.    2.    15.    12.
    7.    6.    2.    16.    21.    6.    2.    0.    3.    6.
   13.    9.    10.    20.    3.    12.    15.    3.    0.    4.
    8.    11.    21.    5.    3.    11.    12.    6.    4.    0.
```

によって与えられた．この時，焼きなまし法によって生成されるマルコフ連鎖 X_k，および対応する距離 S_k は $k = 19 \sim 27$ において

```
         column 19 to 27

    7.      7.      4.      4.      4.      4.      1.      1.      6.
    3.      3.      3.      2.      2.      2.      3.      3.      5.
    4.      4.      7.      8.      8.      8.      7.      7.      7.
    5.      5.      5.      10.     10.     10.     5.      5.      3.
    6.      6.      6.      6.      5.      6.      6.      6.      1.
   10.     10.     10.      5.      6.      5.     10.      8.      8.
    1.      8.      8.      7.      7.      7.      8.     10.     10.
    2.      2.      2.      3.      3.      3.      2.      2.      2.
    8.      1.      1.      1.      1.      1.      4.      4.      4.
    9.      9.      9.      9.      9.      9.      9.      9.      9.
  107.    115.    118.     95.     88.     95.     95.     95.     95.
```

であるが，$k = 492 \sim 501$ において

```
    column 492 to 501

    8.     8.     8.     5.     5.     5.     5.     5.     5.     5.
    3.     3.     3.     9.    10.    10.    10.    10.     7.     7.
    1.     1.     1.    10.     9.     9.     9.     9.     1.     1.
    7.     7.     7.     6.     6.     2.     2.     2.     3.     3.
    4.     4.     4.     4.     4.     8.     8.     8.     8.     8.
    6.     6.     6.     7.     7.     3.     3.     3.     2.     2.
    2.     2.    10.     1.     1.     1.     1.     1.     9.     9.
    5.     5.     9.     3.     3.     7.     7.     7.    10.    10.
    9.     9.     5.     8.     8.     4.     4.     4.     4.     4.
   10.    10.     2.     2.     2.     6.     6.     6.     6.     6.
   56.    56.    61.    61.    62.    61.    61.    61.    62.    62.
```

であり，$k = 1042 \sim 1051$ において

```
    column 1042 to 1051

    9.     9.     9.     5.     5.     5.     5.     5.     5.     5.
    8.     8.     8.    10.     9.     9.     9.     9.     9.    10.
    3.     3.     3.     6.     8.     8.     8.     8.     8.     4.
    1.     1.     1.     2.     3.     3.     3.     3.     3.     2.
    7.     7.     7.     4.     1.     1.     1.     1.     1.     6.
    4.     4.     4.     7.     7.     7.     7.     7.     7.     7.
    2.     2.     2.     1.     4.     4.     6.     6.     6.     1.
    6.     6.     6.     3.     2.     2.     2.     2.     2.     3.
   10.    10.    10.     8.     6.     6.     4.     4.     4.     8.
    5.     5.     5.     9.    10.    10.    10.    10.    10.     9.
```

```
    50.    50.    50.    50.    50.    50.    44.    44.    44.    44.
```

となり，以後2000ステップまで同じ所に留まり続ける．では置換 $(5, 10, 4, 2, 6, 7, 1, 3, 8, 9)$ が本当にこの場合の最短経路なのか，このプログラムは教えてくれない．距離関数 S に対応する命題 (7) で定めた communication height がわからないため，ここで選んだ Cooling schedule T_k の選び方も命題に基づいて理論的に選ばれたものではない．しかし，マルコフ連鎖のステップが進むにつれて距離が減少する方向に向かっている事は見る事ができる．

0.8.3 反復重点サンプリング法

```
// The traveling salesman, Feb. 26, 2009
// In this version you perform adaptive sampling
// NN times.  On each sampling, you produce N
// Markov chains on n-sites, conditioned that
// the path visit each site only once.

clear;

n=6; // the number of sites.

// Part 1.
// First, we define cost matrix C, which is given
// here by random numbers.

A=grand(n,n,'geom', 0.2);
C=A+A';
for i=1:n
  C(i,i)=0;
end
```

```
CostMatrix=C;

//C=[0 9 9 9 23; 9 0 7 9 2; 9 7 0 8 8;
//9 9 8 0 2; 23 2 8 2 0]

// Part 2.
// Generation of Trajectories:
// In other words, we create N random permutations
// of {1,2,,,n}.
P=(1/(n-1))*(ones(n,n)-diag([ones(1,n)]));
// initial trans. prob.

NN=1;
for NN=1:10;  //We do adaptive sampling NN times.
// while min(min(P,1-P))<0.2;

N=720; // the number of samplings.
b=[ones(N,1) zeros(N,n-1)];
// For each path j, you b(j,k)=1 if the path have visited k.

B=zeros(N,1);
// If the path j visits a site more than twice, the B(j)=0.
for j=1:N;
  X(j,1)=1;
  for k=2:n;
    PP=P(X(j,k-1),:).*(1-b(j,:));
    if sum(PP)>0;
       PPP=PP./(sum(PP));
       [m,x]=max(bool2s(rand(1,1)*ones(1,n)<cumsum(PPP)));
       X(j,k)=x;
```

```
       b(j,x)=1;
       B(j)=1;
    else  B(j)=0; break;
    end
  end
end
// PPP is the transition law of where X(k-1) will visit
// next step, for each sampling.  You cannot visit
// where you have already visited.
// The problem here is that given transiton probability,
// there is not always a process which visits each site once.
// Such path is omitted by labeled B(j)=0.
// x is the integer-valued rancom variable with the law PPP.

A=find(B'==1);
// A is the number of path which safely returns to 1.
X(A,:) ;  // N samples of trajectories.

// Part 3.
// We compute the total cost S(X) for the route X.
// for  j=1:N
for k=1:n-1;
CC(A,k)=diag(C(X(A,k),X(A,k+1)));
end
// CC(A,k)=(C(X(i,k),X(i,k+1))_{i\in A}.
CC(A,n)=diag(C(X(A,n),X(A,1)));
S(A)=sum(CC(A,:),'c')+CC(A,n);
// S is the cost for route X.

rho=1/4;
```

```
N=sum(bool2s(A));
SS=sort(S(A));
SSS=SS(N-int(N*rho));
T=bool2s(S(A)<=SSS)';
// T is the number of the path X of which S(X) is the
// above the least 25 percent of all paths.



// Part 4.
// For each i,j, we count the number of paths X for which
// X take the route i to j.
for i=1:n;
P(i,1)=T*bool2s(X(A,n)==i)/sum(T);
[a,b]=find(X(A,:)==i);
K=gsort([a;b],'lc','i');
// a is some j, corresponding b is gives on which step
// path j visits i.
// K gives on which step the path j\in A visits i.
for k=2:n;
[c,d]=find(X(A,:)==k);
L=gsort([c;d],'lc','i');
R=(bool2s(K(2,:)+1==L(2,:)));
// R is the number of the path which goes the route i to k.
P(i,k)=(T*R')/sum(T);
end
end
// This P is give by the formula in the text [Rubinstein].
// P(i,k) is proportional to the amount of paths which
// go i \to k among the elite ones, i.e., S of the path
// is above the least 1/4.
```

```
// This P is not necessarily a prob. trans. matrix,
// since it does not hold that sum_j P(i,j)=1 always.
// The reason for that is i might be the last place
// to vist.  In spite of this fact, you do not need
// alter P to be probabilistic, since you will do it
// after a while.
// end   // end of while
PP=P.*[zeros(n,1) ones(n,n-1)];
P=[1-sum(PP,'c') zeros(n,n-1)]+PP;

NN=NN+1;
S(A)'

end
```

0.9 画像処理への応用

```
/// Image Restoration based on simulated annealing of
/// 2-dim. Ising model, Gibbs sampling (Barker algorithm)
///  with Dirichlet(1)-boundary dondition.

clf()
clear;
/// Part 1.  We create an image S as a sample of Ising model.
/// We create the sample via the Gibbs sampling.

L=30;  // The size of the spin system is L*L.
T=0.5;   // Temperature.
```

```
n=20;  // The steps (sweeps) of the Markov chain

S=ones(L+2,L+2);
S(2:L+1,2:L+1)=(rand(L,L)>=1/2)*2-1; //The initial state with 1-b.c..
subplot(2,2,1)
Matplot(S(:,:)*10)

for k=1:n-1;  // the number of sweeps.
  for e1=2:L+1; for e2=2:L+1;
  alpha= ...
  (1+exp(2*(S(e1-1,e2)+S(e1+1,e2)+S(e1,e2-1)+S(e1,e2+1))/T) )^(-1);
  U=rand(L,L);
S(e1,e2)=-bool2s(U(e1-1,e2-1)<=alpha)+bool2s(U(e1-1,e2-1)>=alpha);
end;end;
end;
subplot(2,2,2)
Matplot(S(:,:)*10)
/// This S is the original image as a sample of Ising spins.



/// Part 2. We create a image T with a binary channel noise.
/// This is a flip noise, independent on each site.

p=0.06;   //  The probability of the flip.
q=1-p;

V=2*(rand(L,L)>=0.06)-1;
T=S(2:L+1,2:L+1).*V;

subplot(2,2,3)
```

```
Matplot(T*10)

/// Part 3.
/// Our objective is to find the minima S of the functional
/// H(S,T)=-\sum_{s~t} S_sS_t+(1/2)log(p/q)\sum S_sT_s,
/// where T is the given image with the noise.
/// Hence, we perform the simulated annealing, namely the
/// Gibbs sampling with the slowly increasing inverse temperature beta(k).



n=20;  // The steps (sweeps) of the Markov chain
SS=ones(L+2,L+2);
SS(2:L+1,2:L+1)=T; //The initial state with 1-b.c..



for k=1:n-1;  // the number of sweeps.
for e1=2:L+1; for e2=2:L+1;
alpha= ...
(1+exp(2*log(1+k)*(SS(e1-1,e2)+SS(e1+1,e2)+SS(e1,e2-1)+SS(e1,e2+1) ...
-log(p/q)*T(e1-1,e2-1)) ) )^(-1);
U=rand(L,L);
SS(e1,e2)=-bool2s(U(e1-1,e2-1)<=alpha)+bool2s(U(e1-1,e2-1)>=alpha);
end;end;
end;

subplot(2,2,4)
Matplot(SS(:,:)*10)

SiteNoiseRate=sum(bool2s(T==S(2:L+1,2:L+1)))/L^2
RestoratedSiteRate=sum(bool2s(SS(2:L+1,2:L+1)==S(2:L+1,2:L+1)))/L^2
```

0.10　終わりに

後半のいくつかのコードによって，確率論と Scilab との相性の良さを感じてもらえたのではないかと思う．基本的な事柄で書ききれなかった事がたくさんあるが，長々とさまざまな事柄を紹介する事が本稿の目的ではないので，この辺で終わりとする．

いずれ次の機会には

- イジング模型や排他過程，パーコレーションなど統計力学的なモデル
- 数理ファイナンスで現われるさまざまなモデル
- 「巡回セールスマン問題」に代表される離散最適化問題に対する「確率的アルゴリズム」によるアプローチ

などの話題について，効率的なシミュレーションや数値計算の手法の紹介とともにその Scilab による実現について紹介したい．

Bibliography

[1] 大野修一, 「Scilab 入門」,
http://www.ecl.hiroshima-u.ac.jp/ ohno/scilab/introscilab/introscilab.html
(2006).

[2] 上坂吉則, 「MATLAB+Scilab プログラミング事典」, SoftBank (2007).