# 人口動態の確率的モデル —
# 二項分布，正規分布，ポアソン分布

小波秀雄

June, 2015

この章では確率的に増減が決まる過程について，人口の推移，あるいは生物の個体数の推移を題材として解説します．この扱いは，たとえば伝染病の感染のシミュレーションや物の移動などにおいても重要な手法です．

## 1　決定論的モデルと確率的モデル

### 1.1　人口動態を記述するモデル

人口学や人口政策の分野においてもっとも重要なデータは**合計特殊出生率 (TFR)**[*1] という数値です．これはひとりの女性が子供を産むことができる年代（おおむね 16 歳から 50 歳）の間に平均して何人の子供を産むかというデータのことです．子供を産むには男親も必要です．したがって人口が維持されるための条件は TFR が 2 以上であることです[*2]．

また，進化生態学の分野では，**適応度 (fitness)** という言葉が中心的な概念となっています．これはある生物の 1 個体の子が繁殖年齢にまで生き残る数のことです．TFR とは異なり，1 個体当たりの値として定義するので，適応度が 1 のときに，個体数は増加も減少もしない平衡状態になります．

さて，これらの人口（個体数）の時間的変化をモデル化する時には，一般に**決定論**

[*1] total fertility rate

[*2] TFR が 2 のときに人口が維持されるという主張は，男女の比が 1:1 であることを前提にしています．効率を考えると雄の数は少なくてもよさそうなものですが，進化的な考察から多くの動物の雌雄の比は 1:1 が最も安定していることが分かっています．

**的モデル**が使われます．たとえば，TFR が 1.42 であるとし，また平均して女性が 30 歳のときに子供を産むものと仮定して，1 世代ごとに人口は $1.42/2 = 0.76$ 倍になるとして計算していくことができます．このやり方で，現在の何年後に半減するかという計算を正確に行うことができます．そのようなアプローチの仕方を**決定論的な方法**といいます．

一方，子どもが生まれるかどうかは，個々の母親にとっては確率的な事象です．子どもの数も必ず整数であって， たとえば 1.8 人の子どもというのは平均値としては意味があっても，個々の事象にはあり得ません．そういった確率的な要素を考慮して将来予測を行う方法を確率論的方法といいます．

対象としている人口が大きいときには，決定論的方法はよい精度で将来を予測できますが，たとえば絶滅とか小集団の進化といった場合には，確率論的な方法による予測が意味を持ちます．この文章では，主に確率論的な扱いに焦点を当てて，シミュレーションのプログラムを考えてみることにします．

### 1.1.1 決定論的な方法

次の式は，さまざまな変化においてよく現れる指数関数的な増減を表す一般式です．

$$x(t) = x_0 e^{kt} \tag{1}$$

時間の単位を年として，30 年後に人口が $1.42/2 = 0.71$ 倍になるわけですから，

$$x(30) = x_0 e^{30k} = x_0 \times 0.71$$

となり，$e^{30k} = 0.71$ となるので，この両辺の対数をとれば，

$$30k = \log(0.71)$$

より，$k = -0.0114163$ が得られます．これを式 (1) に代入し，$x_0 = 20000$ としてプロットしてみれば，人口が減少していくようすを見ることが出来ます．

また，$e^{kt} = 1/2$ として $t$ を求めれば，人口が半減するのにかかる時間を求められます．計算してみると約 60 年になります．日本の実際の TFR はもっと小さく 1.3 程度ですので，今 20 歳前後の人がお年寄りになるころには，毎年生まれる人は今の半分になっているだろうということになります．

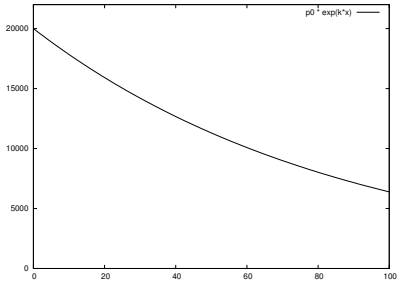


図 1　初期人口 2 万人，TFR = 1.42 のときの 100 年間の人口の推移

### 1.1.2　決定論的な計算の問題

式 (1) は，次の微分方程式の解です．

$$\frac{dx}{dt} = kx \tag{2}$$

これは $x$ の時刻 $t$ における変化率が，$x$ に比例するというモデルをそのまま微分方程式で記述したものです．これを解いて，1 世代後の子どもの数が 0.71 になるという条件を使えば $k$ の値が得られます．

この種の微分方程式を使った解法は決定論的であり，初期状態を設定すれば何度実行しても同じ答が得られます．しかし，実際の現象を再現する上で，決定論的な方法は弱点を抱えています．

上の計算が可能なのは，人口が非常に大きな数字であるときに限られることに注意しましょう．60 年で 100 万人が半減するとしたら，それは「ほぼ正確に」50 万人であるといえるでしょうが，たとえば上の例で初期人口が 20 であったとすると，60 年後の出生数はちょうど 10 人になるでしょうか？実際にはその数字には幅があるはずです．

また極端な話，たった 1 人の状態から人口が 1/2 に減ったとしたら，そのときにいるのは半分の人でしょうか？このケースで実際にありうることは，誰かがまだ生き残っているか，あるいは絶滅しているかという状況のはずです[*3]．

[*3] 皇室が存続するかどうかが，ごく少数の家で男の子が生まれるかどうかにかかっていたという近年の状況は，まさにそういう状況です．

このように，ひとりひとりの運命を扱って考えようとすると，確率的に状況を捉えるしかありません．その結果集団全体の将来の状態はある決まったものにはならず，人数にはばらつきが生じます．つまり，最初に確定した人口から出発したとしても，何世代かたった後の人口はなんらかの分布をもつ確率変数になるわけです．

国の人口のようにきわめて大きな集団では，結果のばらつきが全体に比べて非常に小さいために，決定論的な扱いをしてもかまいません．しかし，小さな集団の場合，その分布の揺らぎは相対的に大きく，その結果，偶発的な絶滅がどのような確率で起きるのかといったことが問題になります．つまり，小さな集団で人口の推移を再現しようとした場合には，確率的なモデルを考える必要があるのです．

# 2　変化を確率論的に扱う—二項分布

## 2.1　二項分布で考える

次の問題を考えてください．

> **20 人の女性が 1 年後までにいずれも 0.7 の確率で子を産むものとします (新婚さんかな？)．双子は考えません．妊娠は確率的な現象ですから，20 人のうち 1 人も子どもを産まない可能性もあるし，全員が子どもを産むかも知れません．1 年後の赤ちゃんの数はどうなるでしょうか？**

決定論的には，この問題の答はいうまでもなく $20 \times 0.7 = 14$ 人です．一方，確率論的には，14 という値は期待値です．期待値が 14 というのは，上記のシチュエーションが非常に多数回起きたとした場合を考えると，その平均は 14 になるだろうという意味です．

確率論的には，これはある事象の確率が 0.7 であり，20 回の試行の後で，その事象が実現した数がどうなるかということですから，典型的な二項分布の問題です．

ここで二項分布の式を思い出しましょう．

$$f(x) = {}_nC_x p^x (1-p)^{n-x} \tag{3}$$

二項分布の期待値 $\mu$ と分散 $\sigma^2$ は，次のように $n$ と $p$ から導かれます．

$$\mu = np \text{ (期待値)} \tag{4}$$
$$\sigma^2 = np(1-p) \text{ (分散)} \tag{5}$$

このように，二項分布は試行回数 $n$ と確率 $p$ の 2 つのパラメータで特徴づけられる確率分布であり，$B[n,p]$ と表現されます．B は Binominal distribution の頭文字です．

もとの問題に戻りましょう．式 (3) に従って $x = 0, 1, \ldots, 20$ について確率を計算してプロットしてみると図 2 のようになります．

図から分かるように，生まれる子どもの数は，期待値の 14 人をピークとして，そのまわりに広がった分布をしています．期待値の 14 人ちょうどである確率は 0.19 程度にすぎません．また，分散 $\sigma^2 = 20 * 0.7 * 0.3 = 4.2$ より標準偏差は 2.05 で．ざっと $14 \pm 2$ 程度の範囲に大半が分布している[*4]ことがグラフからも分かります．

### 2.1.1 分布の形は多数回の後にしか現れない

気を付けておかねばならないことは，図 2 のヒストグラムで表されるような分布は，実際には近似的にしか現れないということです．現実に起きることは，たとえば

$$12, 12, 15, 13, 14, 14, 12, 13, 9, \cdots$$

のような値がばらばらと現れてきて，それを累計した度数分布によるヒストグラムがこの図のような分布に近づいていくのです．次節ではそれをシミュレーションで見ていきます．

## 2.2 二項分布に従う乱数を発生させる

二項分布の性質をシミュレーションで確認してみましょう．そのためには，次の実験を多数回繰り返してみて，結果の分布を知るというやり方をとります．

1. カウンタをゼロにセットする．
2. 4 までを $n$ 回繰り返す．
3. 確率 $p$ で勝つじゃんけんをする．
4. 勝ったらカウンタを 1 つ増やす．
5. 繰り返しを終えたら，カウンタの数字を出力する．

これを実装したメソッド (`binominal_rand`) を作ってください．このメソッドは仮引数として $n$ と $p$ をとります．そうしてから，そのメソッドを多数回 (たとえば 10000 回) 実行して結果を集め，どのような分布になるかを確認してください．また，結果の平均と分散を計算して，式 (4),(5) と比較してみましょう．

以上のアルゴリズムは，2 つのパラメータ $n, p$ で決まる二項分布に従う乱数を発生させるものになっています．

---

[*4] ある確率変数が平均 $\pm$ 標準偏差の範囲に含まれる確率は，確率分布の形によって異なります．正規分布の場合，$[\mu - \sigma, \mu + \sigma]$ の範囲に収まる確率は 68% 程度で，二項分布でもだいたいこの目安で考えておくとよいでしょう．

実際に，$n = 20, p = 0.7$ としてこのメソッドを 10000 回走らせ，出てきた数をカウントしてみると，図 3 のような分布が得られました．理論的に計算した図 2 と，ちょっと見にはわからない結果になっています．

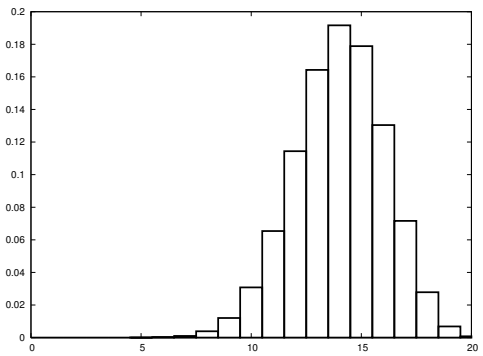


図 2　$n = 20, p = 0.7$ のときの二項分布

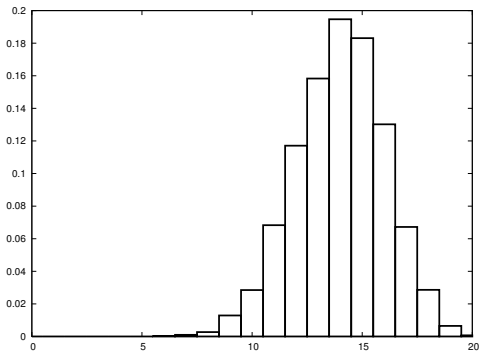


図 3　$n = 20, p = 0.7$ のときの二項分布をシミュレーションで作り出した結果

## 2.3　二項分布乱数で人口変化を追う

さて，前節で作った二項分布乱数を使って，人口の世代変化を追跡することを考えます．単純化のために，有性生殖ではなく，親 1 個体だけで子が産める生物を想定してください．考えるのは次の課題です．

**最初の親個体の数と適応度が与えられたものとして，世代を追って人口が変わっていく様子を確率的に追跡しなさい．**

最初の個体数を $p_0 = 100$, 適応度を $f = 0.7$ とし，また追跡する世代数を $n = 100$ とします．

1. $p \leftarrow p_0$
2. 世代数と人口を出力
3. 4 から 7 を $n$ 回繰り返す
4. $B[p, f]$ に従う乱数を発生させる
5. 世代数と人口を出力
6. 発生した数を $p$ に代入

7. 4 に戻る.

この手続きの中で，二項分布に従う乱数を計算する部分は 2.2 節で取り上げたメソッドを使って下さい．また，世代数を数え上げるには `times`，`for`, `while` のいずれかを使えます.

以上のシミュレーションを 6 回繰り返して得た結果のプロットを図 4 に示します．図から見て取れるように，絶滅に至るのが何世代目であるかは一定ではなく，10 世代から 16 世代ぐらいの間に分布しています.

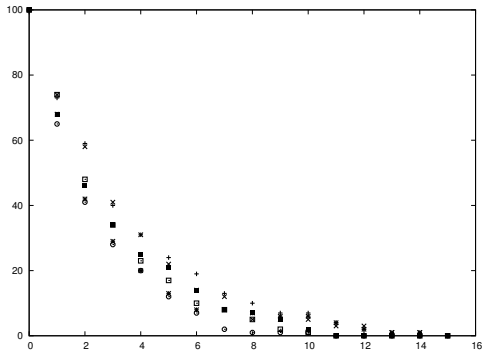


図 4　確率論的に求めた人口減少のようす．初期人口 100, 適応度 0.7 として 16 世代にわたって追跡.

## 2.4　二項分布乱数で大きな人口変化を追うと？

2.3 節の設定で初期人口を非常に大きな数，たとえば 10,000,000 人とした場合，集団の人口がどのように世代変化していくかを追跡してみましょう．これは実際にやってみると，上のやり方では致命的な問題が生じます．それは計算時間がかかりすぎるということです．いまどきの CPU を載せたパソコンでは，最初の 1 世代の計算に 10 秒ほどかかるはずです.

この種のシミュレーションにおいては，何度も同じ計算を繰り返して，結果がどのように分布するかを知りたいのですから，1 世代の計算に 10 秒もかかるようでは，使い物になりません.

計算時間がかかる原因は，二項分布に従う乱数生成のアルゴリズムにあります．このアルゴリズムは，$B[n,p]$ に従う乱数を生成するのに，$[0,1)$ の範囲を乱数を $n$ 回発

生させて，それぞれが $p$ よりも小さかった回数を数えるというものですから，$n$ が大きくなると，乱数発生の回数もそのまま増えるということになるわけです．何かよい工夫はないものでしょうか？

# 3　正規分布で二項分布を近似する

## 3.1　正規分布による正規分布の近似

二項分布 $B[n,p]$ は，$n \gg 1$ のときには，正規分布に近似できます．このことから，一定の条件を満たせば二項分布乱数の代わりに正規分布 $N[\mu,\sigma^2]$ に従う乱数を使うという手段が考えられます．ここで平均 $\mu$，分散 $\sigma^2$ はそれぞれ次のように $n,p$ と関係付けられます．

$$\mu = np \tag{6}$$
$$\sigma^2 = np(1-p) \tag{7}$$

つまり，二項分布 $B[n,p]$ に従う乱数の代わりに，上の平均と分散をもつ正規分布に従う乱数（正規分布乱数）が得られればいいということになります．

一方，正規分布乱数のアルゴリズムとしては，標準正規分布 $N[0,1]$ に従う乱数を発生させるものがよく知られています．また，標準正規分布の確率変数 $z$ と，一般の正規分布の確率変数 $x$ は次の式を使って変換ができますので，前者の乱数をこれに従って変換してやれば，任意の正規分布乱数が得られます．

$$z = \frac{x-\mu}{\sigma} \tag{8}$$
$$x = \mu + \sigma z \tag{9}$$

(8) は標準化変換，(9) は逆標準化変換の式です．

**正規分布乱数発生のためのメソッド**

標準正規分布に従う乱数を発生させるためのメソッドとしてはいくつか知られていますが，次のものはその中でも最もシンプルなものです．このソースは数学関数 `sqrt` や `log`, それに定数 `PI` を利用しているので，ソースの前のほうに `include Math` という行を置くことを忘れないで下さい．

Ruby 1.9.x から，質のよい乱数を発生させるために，ボックス—ミューラー法をよる正規分布乱数発生のアルゴリズムを実装した Random クラスが導入されました。

それを利用して，正規分布およびポアソン分布に従う乱数を発生させるクラスメソッドを定義しておきます。

```
# rand.rb
# Random クラスに新しいメソッドを追加する。
# このファイルは他から reruire './rand' して利用する。
class Random
  include Math
  # ボックス—ミューラー法による正規分布乱数発生
  # Box-Muller method
  def normal_rand(mu = 0,sigma = 1.0)
    a, b = self.rand(), self.rand()
    (sqrt(-2*log(a))*sin(2*PI*b) * sigma) + mu
  end

  # ポアソン分布に従う乱数を発生する
  def poisson_rand(mu)
    lambda = Math.exp(-mu)
    k = 0
    p = 1.0
    while p >= lambda
      p *= self.rand()
      k += 1
    end
    return k - 1
  end
end
```

上の rand.rb は，Random クラスにメソッドを追加しているだけなので，これを実行させても何も起きません。実際に実行させるサンプルプログラムを下に示します。

```
# randtest.rb
# 正規分布，ポアソン分布乱数発生のテスト
# 同じディレクトリに rand.rb を置いて，このプログラムを実行する
require './rand'
```

```
rnd = Random.new()   #
# 正規分布 N[mu,sigma^2] に従う乱数
mu = 100       # 平均
sigma = 10     # 標準偏差
10.times do
  puts rnd.normal_rand(mu,sigma)
end

# ポアソン分布に従う乱数
mu = 3.5
10.times do
  print rnd.poisson_rand(mu), " "
end
puts
```

この標準正規分布乱数を一般の正規分布でも使うためには，得られた乱数 $z$ を式 (9) に従って逆標準化変換してやります．

## 3.2 正規分布乱数で人口変化を追跡する

次の問題を考えてみます．

> **個体数 10,000 から出発して，適応度 0.9 で世代が交代して**
> **いったとき，20 世代目までの変化を追跡しなさい．**

この問題も，原理的には二項分布の問題です．最初の 10,000 の親個体それぞれが子どもを産む確率が 0.9 ですから，その次の世代の個体数 $x$ は ${}_{10000}\mathrm{C}_x 0.9^x(1-0.9)^{10000-x}$ によって決まる二項分布に従います．

言い換えれば 100000 人の親にひとりずつサイコロを振らせていって，0.9 の確率で子どもを産んでもらう，その結果何人の子どもが産まれるかということです．この考えにもとづくアルゴリズムはすでに扱いました．

しかし，前述のようにこのやり方は莫大な回数の乱数発生を伴います．そこで，二項分布ではなく正規分布に従う乱数を発生させて，1 世代後の子どもの数の計算を，1 回の乱数発生で済まそうというわけです．

二項分布の平均と標準偏差はそれぞれ次の形で，試行回数 $n$，確率 $p$ と関係づけられています．

$$\mu = np \tag{10}$$
$$\sigma = \sqrt{np(1-p)} \tag{11}$$

この平均と標準偏差は，分布の形を正規分布としても変わりません．したがって，正規分布 $N[\mu, \sigma^2]$ に従う乱数を発生させてやれば，一発で次の世代の子どもの数を得られるはずです．

ただし，私たちが持っているのは標準正規分布 $N[0, 1]$ に従う乱数を発生するメソッドだけですから，それによって得られた乱数を逆標準化変換して，実際の子どもの数を得ることになります．

もうひとつ，正規分布は実数を確率変数とする連続分布であるのに対し，二項分布は整数を確率変数とする離散分布であることにも注意しないといけません．したがって，得られた正規分布乱数は整数に変換して人口変化を求める必要があります．

以上を踏まえて，ソースを書く方針を考えてみます．

1. 正規分布乱数のメソッドを定義する．
2. 初期人口 (`pop0`) を 10000, 適応度 (`fitness`) を 0.9, 追跡する世代数 (`ngen`) を 20 にセットする．
3. 世代数 (`i`) を 0 にセット, 人口 (`pop`) を初期人口とする．
4. 世代数と人口を出力する．
5. 以下を `ngen` 回繰り返す．
6. 平均 (`mu`) を式 (10) に従って計算する．
7. 標準偏差 (`sigma`) も同様に式 (11) に従って計算する．
8. 正規分布乱数を発生させ，`mu` と `sigma` を使って逆標準化変換する．
9. 上の値を整数に変換して，`pop` に代入する．
10. 世代数と人口を出力する．

# 4　ポアソン分布

ポアソン分布というのは，非常に多数の試行がなされる一方で，考える事象の確率が非常に小さく，結果的にそれほど大きくない期待値になるといった現象に適用される確率分布です．

たとえば，毎朝のラッシュの中ですごいイケメン男を見かける事象がどれだけあるかを考えてみましょう．この場合，あなたがラッシュで 1000 人の男性と遭遇するものとし，かつイケメンの存在率が 0.6% であるとしましょう．ちなみにこの程度のまれな比率は，イケメン度の偏差値にしてほぼ 75 に相当します．正規分布表で確かめてみて下さい．さてこの場合に，あなたが毎日出会う超イケメン男の数はどのように分布するでしょうか？

まず，この問題が二項分布に他ならないことはすぐに分かります．したがって式 (3) で $n = 1000, p = 0.006$ とすればいいわけです．期待値は $np = 6$ となり，

さらに，このイケメン男との毎日の遭遇数をシミュレートするような乱数はどんなふうにして発生させればいいでしょうか？これも簡単なことで，2.2 で紹介した二項分布乱数を使えばいいということになります．実際にソースを書いて，そのような乱数を発生させてみてください．

## 4.1　実例

次に 100 個の数が印刷されています．これをまず眺めてください．

> 1, 1, 1, 4, 1, 4, 0, 1, 1, 2, 1, 0, 2, 1, 2, 0, 0, 1, 2, 1, 1, 2, 2, 2, 2, 0, 0, 1, 1, 0, 2, 2,
> 2, 1, 0, 4, 1, 2, 0, 3, 1, 1, 2, 0, 3, 1, 2, 0, 0, 0, 1, 1, 0, 2, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
> 1, 2, 2, 2, 1, 2, 0, 1, 1, 1, 2, 0, 1, 0, 3, 1, 2, 3, 1, 2, 1, 0, 0, 2, 0, 2, 1, 0, 1, 1, 2, 1,
> 1, 0, 6, 1

0, 1, 2 という数が頻出し，たまにそれよりも大きな数が現れています．出方はランダムに見えます．もっと正確につかむために，それぞれの数の出現回数を数えて度数分布を調べてみましょう．

| 数 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 度数 | 28 | 39 | 25 | 4 | 3 | 0 | 1 | 0 |

度数の合計は 100 になっていますね．それではこれらの平均値を計算してみましょう．

$$\frac{0\times 28+1\times 39+2\times 25+\cdots+7\times 0}{100}=1.19$$

平均値は 1.19 と， 1.2 に近い値をとっています．適当な散り具合といい，期待値に近い平均値になっていることといい，なかなか使えそうな乱数系列が得られているようです．

このような数を発生するために使われるのがポアソン乱数です．ポアソン乱数はポアソン分布という確率分布に従って整数を発生する乱数で，少数のサンプルを扱うシミュレーションではよく使われます．次項以下，このポアソン乱数について調べていきましょう．

## 4.2 二項分布からポアソン分布へ

### 4.2.1 ポアソン分布の導入

数学的な取り扱いでは，ポアソン分布 (Poisson distribution) は二項分布において $p\to 0,\ n\to\infty$ ただし，$\mu=np$ は有限としたときに現れる確率分布です．

$$\lim_{p\to 0,n\to\infty}{}_nC_x p^x(1-p)^{n-x}=f(x)=\frac{\mu^x}{x!}e^{-u} \tag{12}$$

ここで $x$ は離散的な確率変数 $x=0,1,2,\ldots$，$\mu$ はすでに書いた通り期待値 (平均) です．

確率分布では，期待値以外に分散も重要なパラメータです．その値がどうなるかを考えてみましょう．二項分布における分散は式 (5) で与えられるのでした．

一方，ポアソン分布というのは二項分布において $n\to\infty, p\to 0$ としたときの極限です．このとき $(1-p)\to 1$ となりますから，結局次のように分散が得られます．

$$\lim_{p\to 0,n\to\infty} np(1-p)=np=\mu \tag{13}$$

すなわち，次のことが言えるわけです．

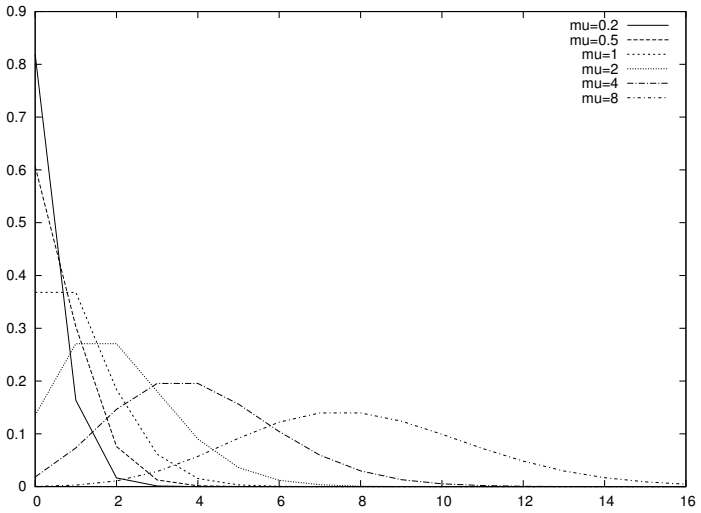


図 5　期待値 0.2, 0.5, 1,2,4,8 に対するポアソン分布

**ポアソン分布の分散 $\sigma^2$ は期待値 $\mu$ に等しい.**

また，ポアソン分布においては $n, p$ は現れず，期待値の $\mu$ だけをパラメータとして分布が決まることになります．したがってポアソン分布は $P[\mu]$ と表現されます.

### 4.2.2　ポアソン分布の意味

ポアソン分布は，数学的にみると二項分布の極限なのですが，その現実的な意味はどうなるのでしょうか．その点について考えてみましょう.

二項分布では，$n$ 回の試行の結果として確率 $p$ の事象が何回出現するかということを表現しています．ポアソン分布は $n$ が大きく，かつ $p$ が小さいときの二項分布であるというわけですから，試行回数がすごく多く，かつ 1 回あたりの確率がとても小さいという状況です.

交通事故の統計は，このような状況に合致している例です．町には何千人，何万人というたくさんの人が住んでいて，だれもが交通事故に遭う可能性がありますが，その確率は非常に小さく，交通事故統計から見ると 1000 人のうちの数人程度が何らかの事故に遭っていうというレベルです．したがって，仮に 1 人が年間に交通事故に遭う確率が 0.5% とすると，2000 人が住む地域では年間に平均 10 人が事故に遭うこと

になります．もちろんちょうど 10 人になるわけではなく，8 人かも 9 人かもしれませんし，ひょっとしたら 0 人の可能性だってあるわけです．そのような確率を計算しようというのが，ここでの問題です．

例として事故数が 8 になる確率を計算してみましょう．二項分布を使うと次のようになります．

$n = 2000$, $p = 0.005$ として式 (3) を使います．事故数が 8 ですから，$x = 8$ となります．これらを代入すると次のようになります．

$$f(8) =_{2000} C_8 \times 0.005^8 \times (1 - 0.005)^{1992} \tag{14}$$

しかし，この式に出てくる二項係数 ${}_{2000}C_8$ の値を計算するのはとても大変です．$0.995^{1992}$ などというのも，いかにも誤差が生まれそうな式です．それでもなんとかして計算してみると， 0.1127 という値になります．

ポアソン分布の式 (12) の式を使って上の計算をやってみましょう．

$$f(8) = \frac{0.005^8}{8!} \times e^{-0.005} = 0.1126 \tag{15}$$

こちらの式は二項分布よりもずっと計算しやすくなっています．得られた値もほとんど一致しています．このように面倒な二項分布の代わりにずっと簡単なポアソン分布を利用することで，大幅に手間を省けるわけです．

もうひとつの特徴は，ポアソン分布にはパラメータとして $\mu$ しか現れないという点です．事象の確率 $p$ と試行回数 $n$ は不要というわけです．これも計算を簡単にすることにつながります．

## 4.3 ポアソン分布が適応できるケース

一定期間内に何回か起きるランダムな現象，あるいは一定の区間や面積の中にランダムに分布する何かといったものに対してポアソン分布は適用されます．

具体的例としては，たとえば 1 時間あたりに観察される流星の数，野球の 1 試合あたりのホームランの数，毎朝の通勤時間に見かける赤い車の数，学校の中の双子の数，その他いろいろなものを思い浮かべることができるでしょう．他にもさまざまなケースを自分で考えてみてください．

これらの例は，いずれも全体としてはほぼ一定の割合で起きていると考えられる事

象があって，そのうちの少数が観察されているようなケースです．このような分布はポアソン分布に従うことが多いのです．

## 4.4 ポアソン分布発生のアルゴリズム

### 4.4.1 素朴な方法

素朴な手法でポアソン分布を発生させるためのシミュレーションを実装してみることにします．`rand()` 関数を使うと，区間 $[0, 1)$ の中で一様分布する乱数が得られます．これは雨粒が 1 m の区間に均等に落ちてくるような状況を想定すると分かりやすい (図 6)．その区間の中の $p$ の長さの部分に落ちる確率は $p$ ですから，雨粒が全部で $n$ 個落ちるとすると，そのうちの $np$ 個が長さ $p$ の区間内に落ちることになります．

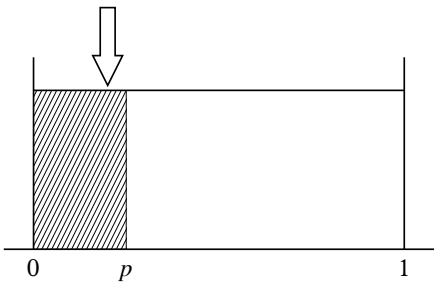


図 6　$[0, 1]$ の区間に等しい確率で雨粒が落ちるとき，$[0, p]$ の区間に落ちる確率は $p$ に等しい．

さて，ポアソン分布が適用される状況では，$n$ を大きくとり，$p = \mu/n$ は小さくなるようにします．たとえば $\mu = 2.5$ が与えられた時に $n = 500$ とすれば $p$ は 0.005 となります．つまり，数を増やしながら幅を狭めてやって，命中する数を一定に保つというわけです．

これをシミュレートするためには，次のような動作をするメソッドを作ればよいでしょう．

1. $\mu$ を仮引数として受け取る．
2. $n$ に十分大きな値を代入する．
3. $p \leftarrow \mu/n$

4. $c \leftarrow 0$
5. 区間 $[0,1)$ の一様乱数 $r$ を発生させる．
6. $r$ は区間 $[0,p)$ に入るか？
7. yes $\rightarrow$ $c$ を 1 増やす．
8. 繰り返しが $n$ 回未満なら 5. へ戻る．
9. $c$ を返す．

以上の方針で書いてみたメソッドを下に示します．

```
def sim_poisson_rand(mu)
  n_trial = 500
  p = mu/n_trial.to_f
  count = 0
  n_trial.times{
    count += 1 if rand(0) < p
  }
  return count
end
```

このメソッドの動作を確認するには，次のようにします．

1. $\mu$ の 10 倍程度のサイズの配列 `freq` を用意しておく．
   これは頻度 (frequency) を記録するための配列．
2. 多数，たとえば 10000 回 (`n` としておく) 上記のメソッドを呼び出して，それが返して来た整数を添字とする要素を 1 ずつ増加させる．ちょうどたくさんの仕分け箱に玉を 1 つずつ入れていくように．
3. 最後に，配列のすべての要素を `n` で割った値を出力して，式 (12) のポアソン分布の理論値や，先述したポアソン乱数の出力と比較する．

### 4.4.2 ポアソン分布を発生させる高速アルゴリズム

実際に上述の素朴なアルゴリズムでポアソン分布乱数を発生させてみると，莫大な計算量が必要なことが分かります．これは多数の乱数を発生させることがアルゴリズムに含まれているためです．一般に確率論的なシミュレーションでは，多数回の試行

を行うことになるので，このアルゴリズムを使うととんでもない時間がかかることになってしまいます．

そこで，もっと数学的に洗練された議論をもとに，次のようなアルゴリズムが考案されています．

1. $\lambda \leftarrow e^{-\mu}$, $k \leftarrow 0$, $p \leftarrow 1$
2. $p \leftarrow p \times r$, ここで $r$ は区間 $[0, 1)$ の乱数
3. $k \leftarrow k + 1$
4. $p \geq \lambda$ の間 2. から繰り返す
5. $k - 1$ を返す．

# 5　小集団の絶滅

個体数 50 の集団があり，それぞれの適応度が 0.95 であったとします．

決定論的モデルでは，この集団の人口はゼロになることはありません．そのかわり 0.01 人とか $10^{-6}$ 人といった不合理な値になってしまいます．したがって絶滅するまでの時間を知りたければ，人口がたとえば 0.5 になったときといった基準を設けて，そこまでの時間を計算することになります．

確率論的モデルを使うとこの困難は避けることができ，集団はあるところで絶滅します．ただし毎回結果は異なります．つまり集団の寿命はある確率分布に従うわけです．したがって，何回も繰り返して実験を行って，その結果を集計して研究することになります．

こちらのほうがリアルに状況を反映するモデルであることはいうまでもありません．

## 5.1　絶滅のシミュレーション

図 7 は初期人口 500 の集団の適応度が 0.95 であったときに，どのように個体数が減少していくかを，ポアソン分布を使ってシミュレートしてみましょう．

このシミュレーションは次のような流れで実行されています．

1. 初期個体数 `pop0` を 500, 適応度 `fitness` を 0.95 とする．
2. `pop` に `pop0` を代入．
3. 世代数 (0) と人口 (`pop`) を出力．
4. 以下 9. までを 100 回繰り返す．
5. 次世代の数を表す変数 `next_pop` を 0 にする．
6. 以下 7. までを `pop` 回繰り返す．
7. `next_pop` に 期待値 `fitness` のポアソン乱数を加える．
8. `pop` に `next_pop` を代入．
9. 世代数 $(1, 2, \ldots)$ と人口を出力．

上の手続きの中心は 5. から 7. までの部分です．

この過程が意味しているのは，つぎのような処理です．

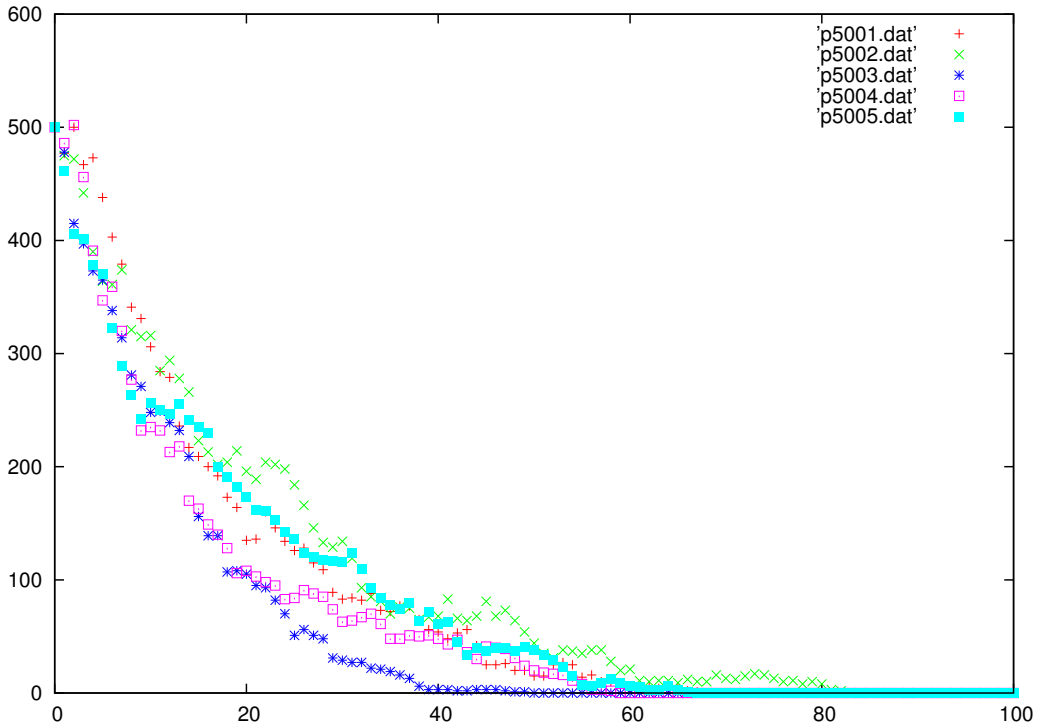


図 7　適応度 0.95 の集団の人口推移

1. いま存在しているすべての個体について，それぞれに子供を何人作るかを調べる．
2. 次世代の子供の数を総計した後に，それを新しい世代の親として計算を繰り返す．

したがって，ひとつの世代の計算は個体数の回数だけのループで実行し，それをさらに世代を繰り返すように繰り返すことになります．

# 6 集計のための Ruby スクリプトなど

## 6.1 二項分布データを作る

このアルゴリズムでは，いったんパスカルの三角形を生成してから，確率因子 $p(1-p)$ を各項に掛けています．

```
#!/usr/bin/env ruby
# generate array of binominal distribution
# parameter = n,p
def binomal_dist(n,p)
  ar = []
  v = [1]
  n.times{
    v2 = v.dup
    for i in 1 ... v.size
      v[i] += v2[i-1]
    end
    v << 1
  }
  for i in 0 .. n
    ar << v[i] * p ** i * (1-p) ** (n-i)
  end
  ar
end

if __FILE__ == $0 then
  p binary_dist(20,0.4)
end
```

## 6.2 整数データの度数分布表を作成する

```
#!/usr/bin/env ruby
# mkfreqtbl.rb
=begin
このプログラムは次のように整数が書き込まれたファイルから
度数分布表のデータを作りだす．カンマ，改行，空白がデータの区切りに
```

なっていることに注意.

```
1 3 4 2 3 4 5
6   8 5,9, 10,1
...
=end
freq_file = ARGV.shift
line = ""
File.open(freq_file) do |f|
  line = f.gets(nil)
end
data =  line.split(/[\s\,]+/).map{|v| v.to_i }
max = data.max
freq = Array.new(max+1,0)
data.each do |v|
  freq[v] += 1
end
for i in 0 .. max
  puts "#{i} #{freq[i]}"
end
```

## 6.3 浮動小数点データの度数分布表を作成する

```
#!/usr/bin/env ruby
# mkfreqtbl2.rb
=begin
このプログラムは浮動小数点データが書き込まれたファイルから
相対度数分布表のデータを作り出す.

1.1 3.2 4.1 2.1 3.0 4.1 5.1
6.1,  8.0 5.5, 9, 10,1.7
...
ファイルを読み込んだ後，変数の範囲と階級の幅を尋ねられるので,
そこでもう一度入力する.
=end
$stdout.sync = true
if ARGV.size < 1
  puts "USAGE: #{__FILE__} DATAFILE"
  exit
```

```
end
datafile = ARGV.shift
line = ""
File.open(datafile) do |f|
  line = f.gets(nil)
end

data = line.split(/[\s\,]+/).map{|v| v.to_f}
max = data.max
min = data.min

puts "Data:  Min = #{min}, Max = #{max}"
puts "Input: left_end right_end  n_div  [output_file]"
comline = gets
left = comline.split(" ")[0].to_f
right = comline.split(" ")[1].to_f
ndiv  = comline.split(" ")[2].to_i

if !(outfile = comline.split(" ")[3])
  outfile = File.basename(datafile,".*") + ".freq"
end
dx = ((right-left)/ndiv).to_f
freq = Array.new(ndiv+1,0)
data.each do |v|
  i = ((v-left)/dx).to_i
  freq[i] += 1
end
File.open(outfile,"w") do |f|
  for i in 0 .. ndiv
    x = left + dx * i + 0.5 * dx
    r = freq[i]/(data.size*dx)
    f.puts "#{x} #{r}"
    puts "#{x} #{r}"
  end
end
STDERR.puts "Frequency data was written in '#{outfile}'."
```