インターネット計測とデータ解析 第 5 回

長 健二朗

2015 年 5 月 18 日

前回のおさらい

第 4 回 分布と信頼区間 (5/11)

- 正規分布
- 信頼区間と検定
- 分布の生成
- 演習: 信頼区間
- 課題 **1**

今日のテーマ

第 5 回 多様性と複雑さ

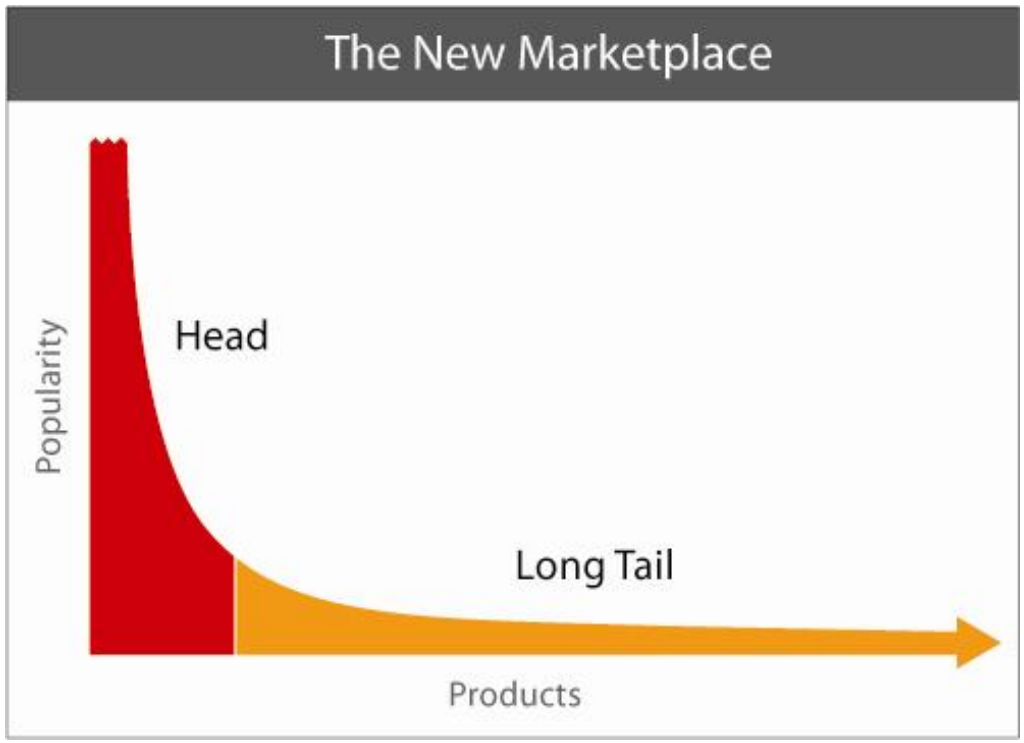
- ロングテール
- Web アクセスとコンテンツ分布
- べき乗則と複雑系
- 演習: べき乗則解析

ロングテール

オンライン小売サービスのビジネスモデル

- ヘッド: 少数の売れ筋商品、リアルの店舗の守備範囲
- テール: 多様な売上下位商品、オンライン店舗の売上の特徴

いまでは多様なニッチマーケットを指す言葉として広く使われる



source: http://longtail.com/

複雑さ

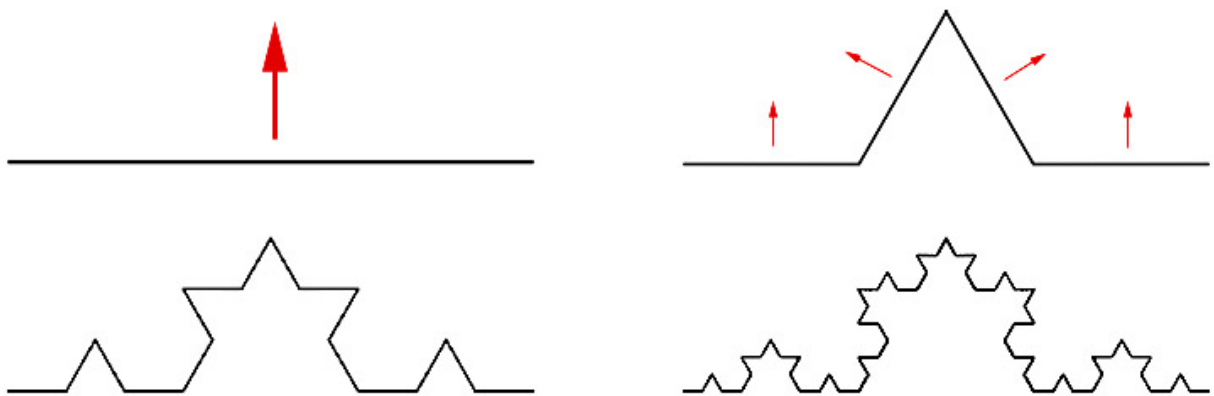
複雑さの科学

- 多数の因子が相互に影響して複雑な挙動を示すシステム
  - カオス、フラクタル、非線形力学など
- 世界は複雑系に満ちている
- 従来の還元主義的手法で解析が困難
  - 複雑な現象を複雑なまま理解する必要
- 90 年台から盛んに研究
  - 還元主義的手法で解ける未解決な問題が減ってきた
  - コンピュータによる解析やシミュレーション

べき乗則と複雑系

べき乗則

- べき乗則は複雑系を示す特徴のひとつ
  - べき乗則: 観測量がパラメータのべき乗に比例
  - 自己相似的 (フラクタル)
- さまざまな自然現象、社会現象、インターネットサービスで観測される
- スケールフリー: 特徴的なスケールを持たない



コッホ曲線の作成：海岸線に似たフラクタル図形

ジフ (Zipf) の法則

- 1930 年代に順位付けされたデータの出現頻度で発見された経験則
- シェアは順位に反比例
  - 出現頻度が k 番目に大きい要素が占める割合が $1/k$ に比例
- 社会科学や自然科学、データ通信でさまざまな現象が確認される
  - 英単語の出現頻度、都市の人口、富の分配など
  - ファイルサイズ、ネットワークトラフィックなど
- リニアスケールのグラフではロングテール、ログログスケールのグラフではヘビーテイルになる

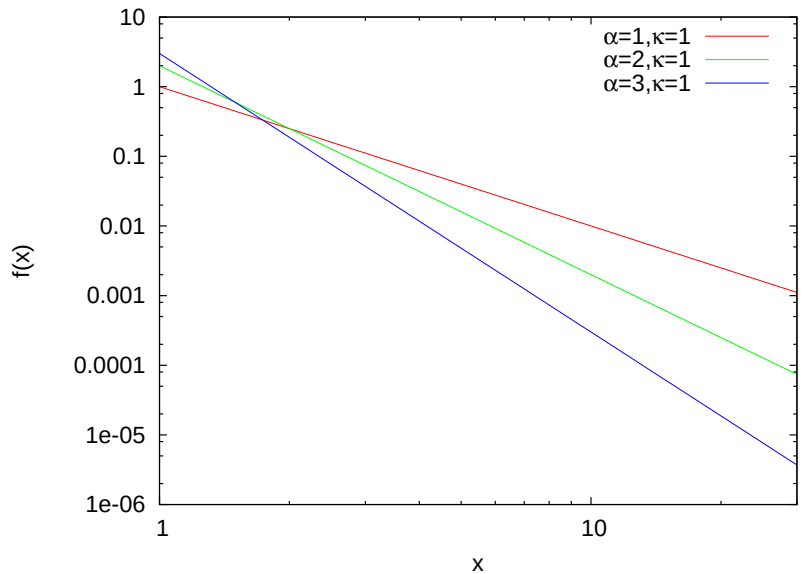
べき分布 (power-law distribution)

- べき分布: ある量が観測される確率がその大きさのべき乗に比例

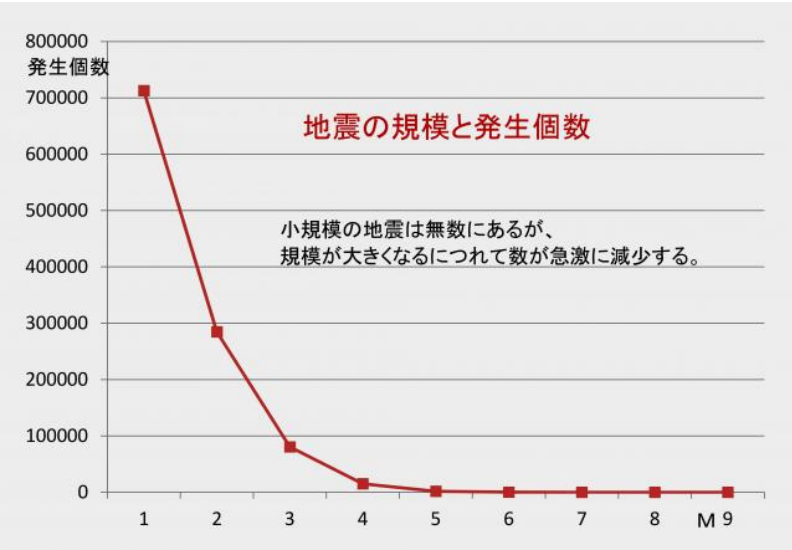
$$f(x) = ax^{-k} = a\frac{1}{x^k}$$

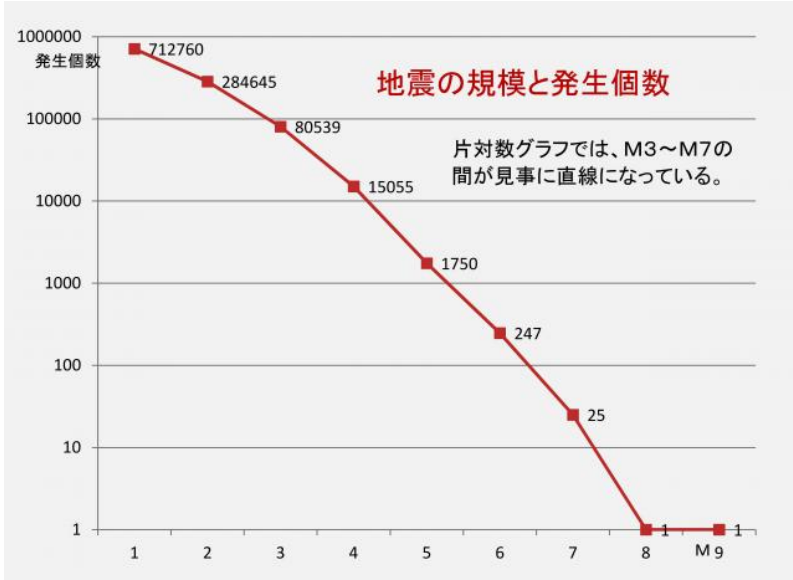
- 両対数グラフに書くと線形になる

$$\log f(x) = -k \log x + \log a$$



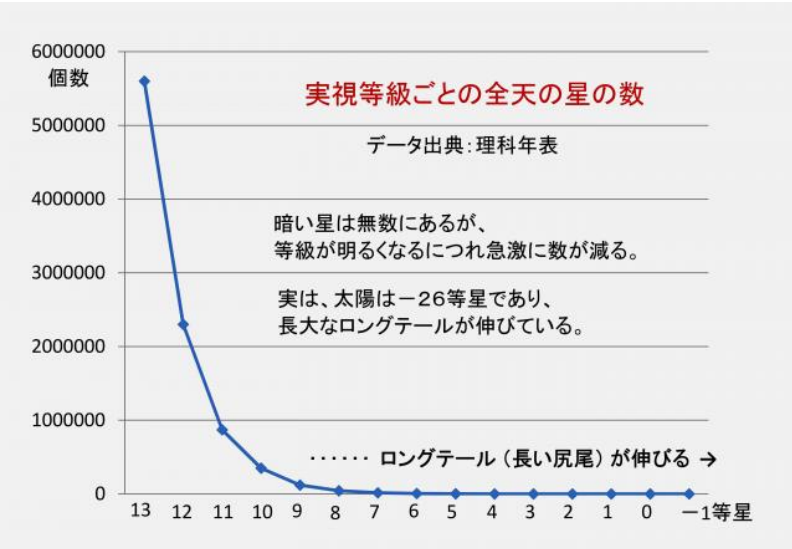
べき分布の例: 地震の規模と発生回数

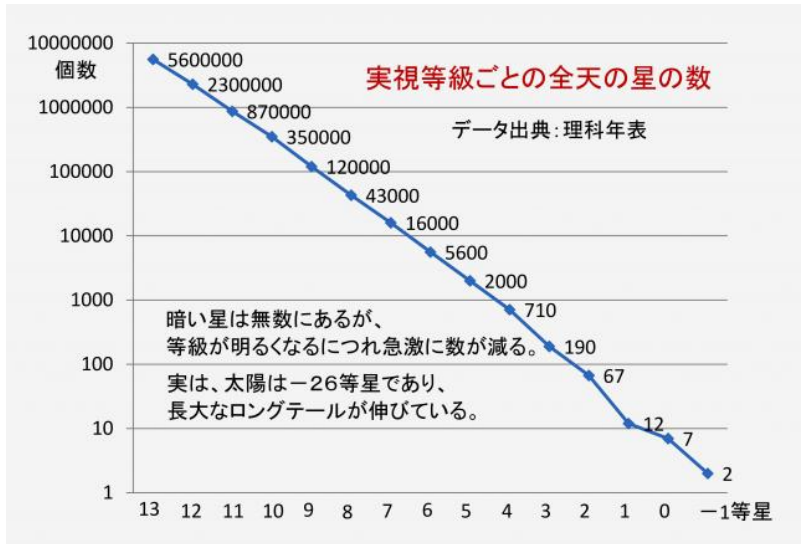




source: http://sitakisou.blog.fc2.com/blog-entry-963.html

べき分布の例: 星の等級と数





source: http://sitakisou.blog.fc2.com/blog-entry-963.html

インターネットの複雑さ

トポロジーの複雑さ

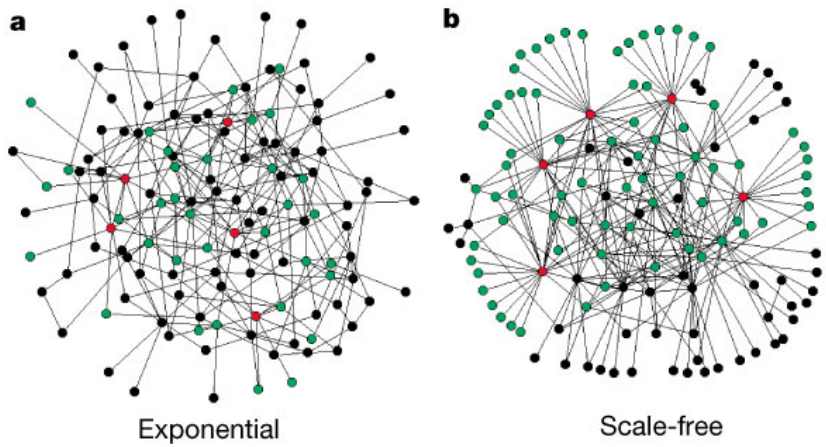
- スケールフリー: ノードの次数にべき乗則の偏り
  - 多数の小次数ノードと少数の大次数ノード
  - 平均的なサイズがない
- スモールワールド:
  - コンパクト: 任意のノード間の距離は短い
  - クラスタ: 友達の友達は友達

トラフィックの挙動 (時系列解析)

- 自己相似性
- 長期依存性

スケールフリーネットワーク

- ネットワークノードの次数分布がべき乗則に従う
  - ほとんどのノードの次数は 1 や 2
  - 一部のノードの次数は桁違いに大きい (ハブノード)
- スモールワールド
  - ハブノードを経由して任意のノードに短距離で到達
  - ハブノードは情報の拡散機能 (感染症の場合も)
- 発生の仕組み: preferential attachment: rich get richer
  - 次数の大きいノードに接続しやすい仕組み
- 耐故障性、耐攻撃性
  - ランダムなノードの故障には強い
  - ハブノードへの攻撃には弱い

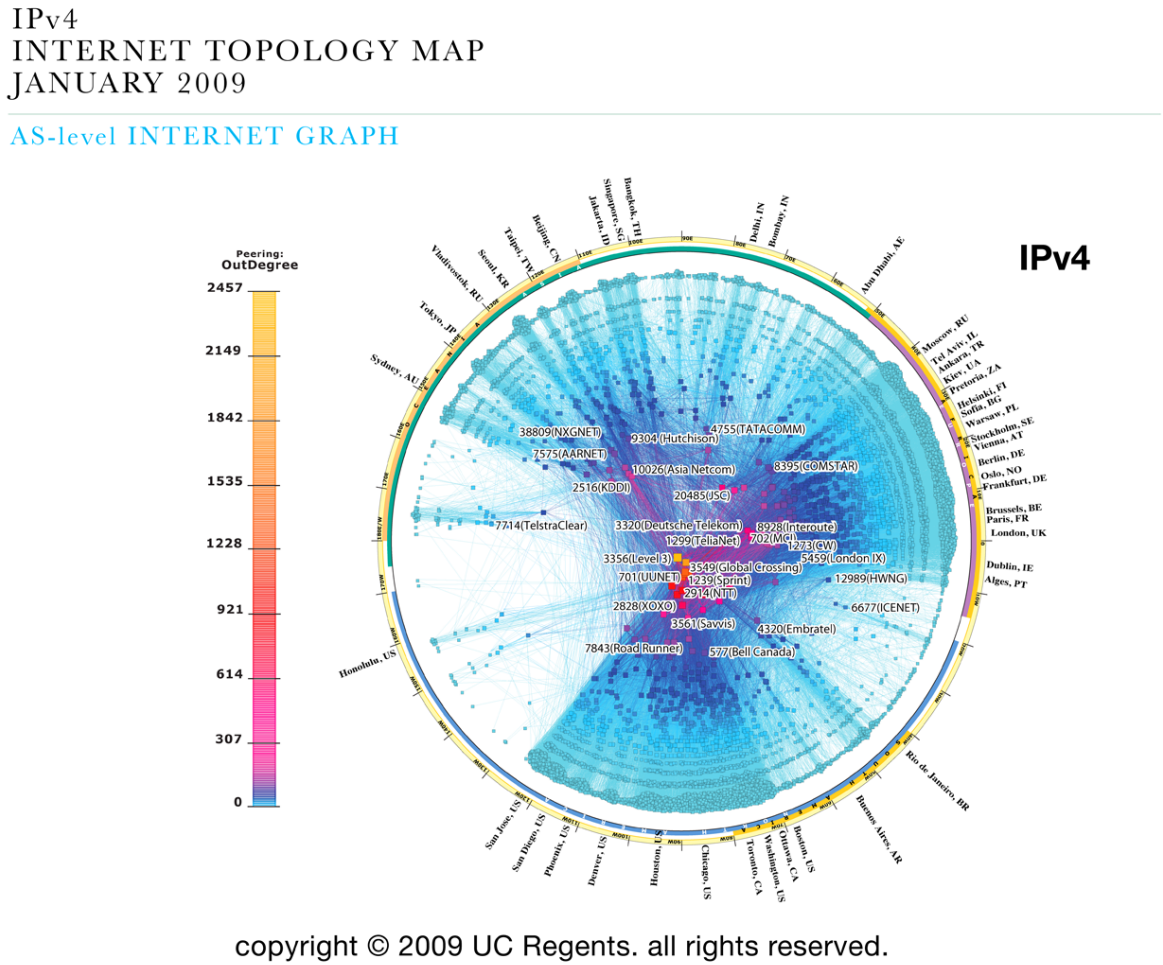


source: Error and attack tolerance of complex networks. R. Albert, H. Jeong, A. Barabasi. Nature 406, July 2000.

インターネットの AS 構造の例

CAIDA AS CORE MAP 2009/03

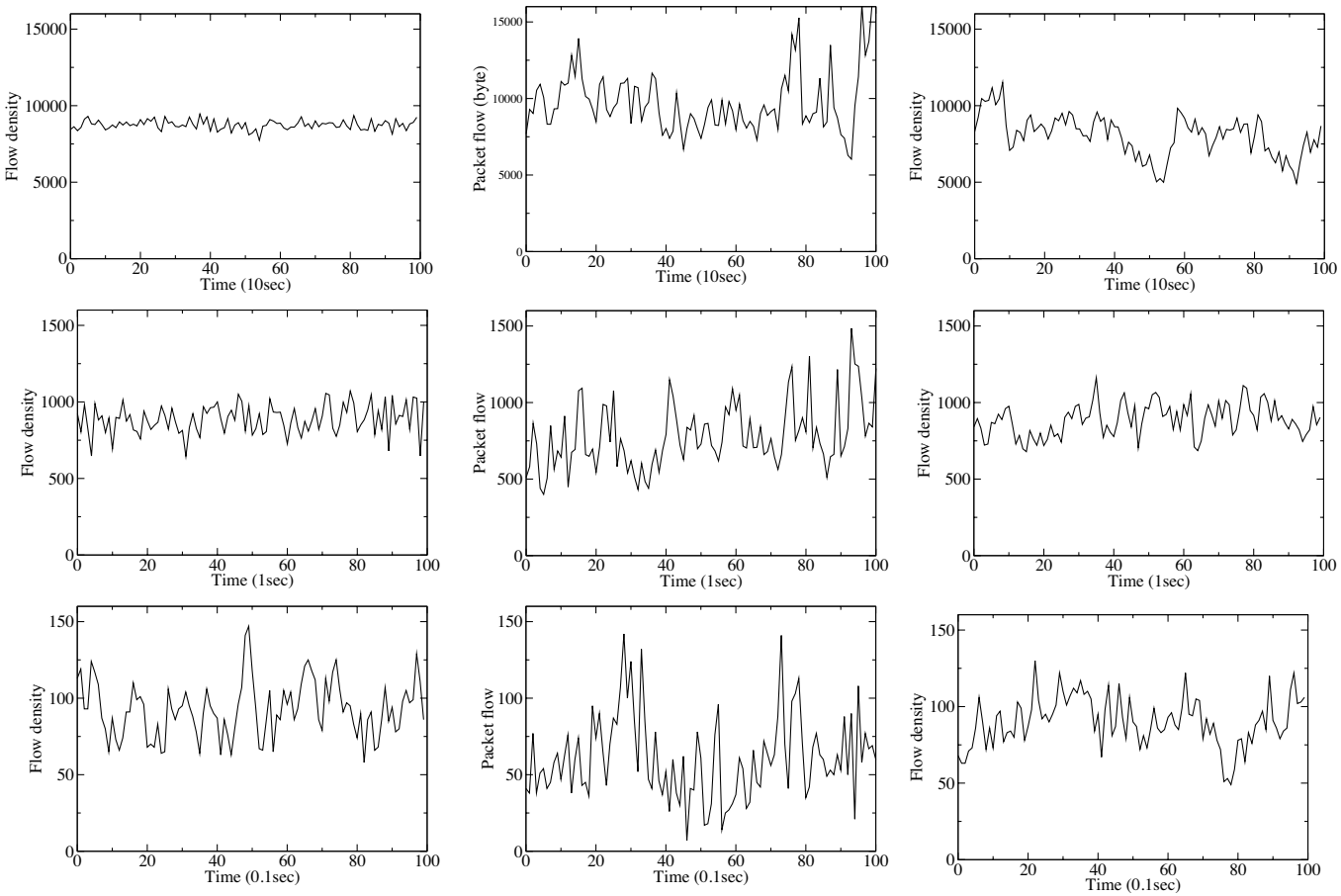
- AS の登録都市の経度、AS の out-degree



`http://www.caida.org/research/topology/as_core_network/`

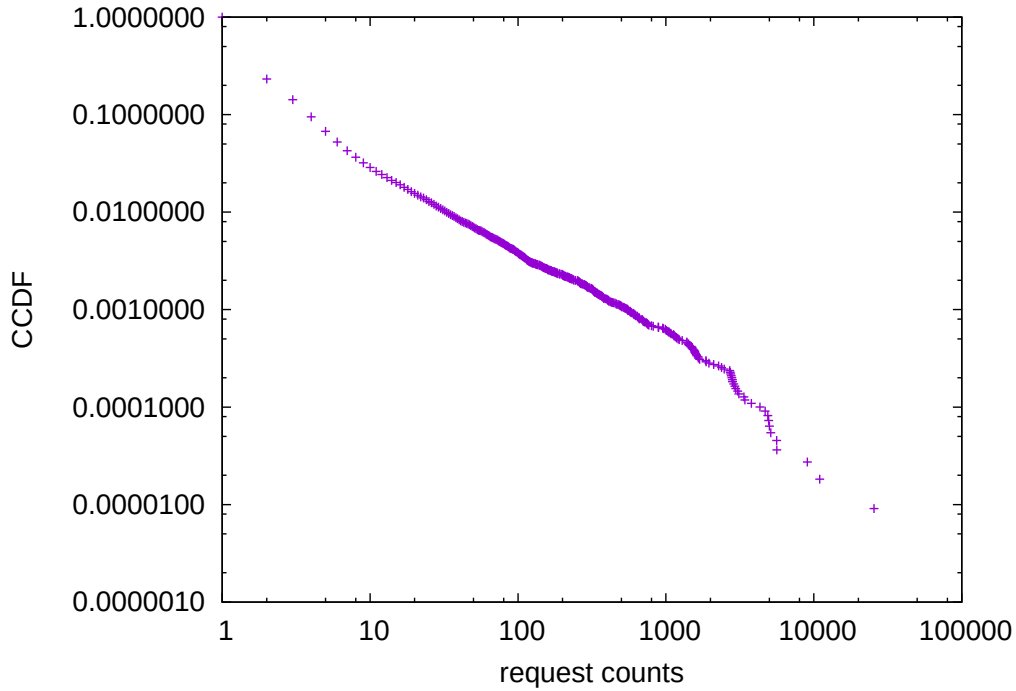
ネットワークトラフィックの自己相似性

- (左) 指数関数モデル (中) 実トラフィック (右) 自己相似モデル
- 時間粒度: (上)10sec (中)1 sec (下)0.1 sec



Web アクセスとコンテンツ分布

- Web の世界にもいたるところに、べき分布が存在
  - Web ページの被リンク数やアクセス数、検索キーワード



JAIST サーバのコンテンツ毎のアクセス数分布

さまざまな分布

- 二項分布
- ポアソン分布
- 正規分布
- 指数分布
- べき分布

二項分布 (binomial distribution)

- ベルヌーイ試行 (bernoulli trial): 試行の結果が 2 種類しかない試行
- 1 回の試行の成功率を p とし、n 回の試行の成功数 k の離散確率分布

確率関数 (PDF)

$$P[X = k] = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}$$

ここで

$$\left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!}$$

$$mean : E[X] = np, variance : Var[X] = np(1-p)$$

二項分布は n が大きくなるとポアソン分布で近似できる

ポアソン分布 (poisson distribution)

まれにしか発生しない事象の時間あたりの発生回数はポアソン分布に従う

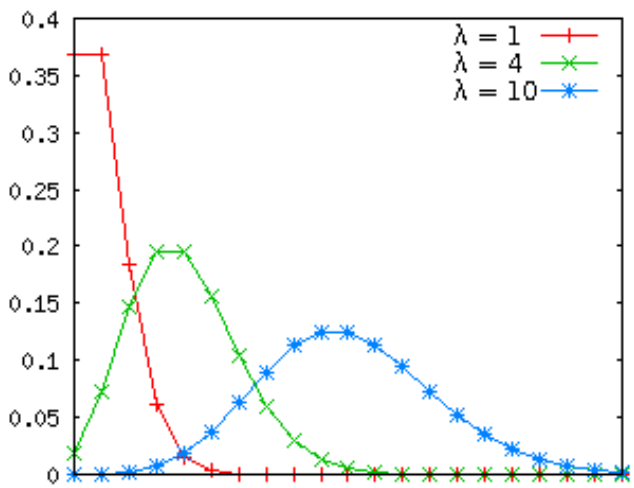
- 交通事故死亡者数や、遺伝子の突然変異数など

ポアソン分布はただひとつの平均パラメータ $\lambda > 0$ で表される

確率関数 (PDF)

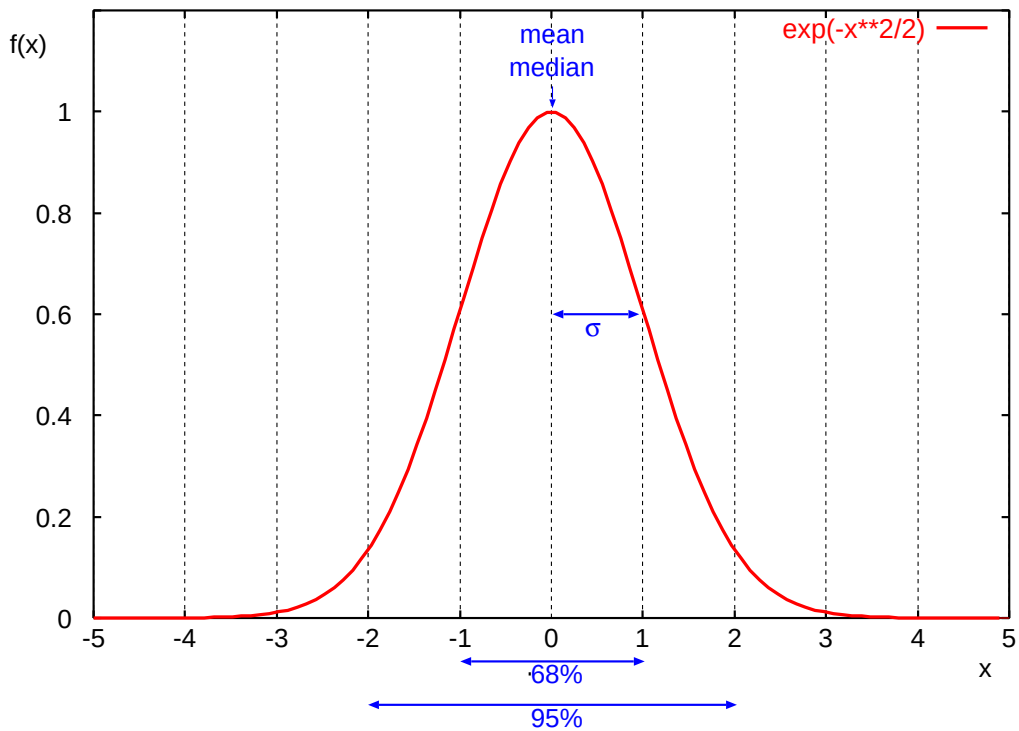
$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

$$mean : E[X] = \lambda, variance : Var[X] = \lambda$$



正規分布 (normal distribution) 1/2

- つりがね型の分布、ガウス分布とも呼ばれる
- 2 つの変数で定義: 平均 μ、分散 σ^2
- 乱数の和は正規分布に従う
- 標準正規分布: $\mu = 0, \sigma = 1$
- 正規分布ではデータの
  - 68%は $(mean \pm stddev)$
  - 95%は $(mean \pm 2stddev)$ の範囲に入る



正規分布 (normal distribution) 2/2

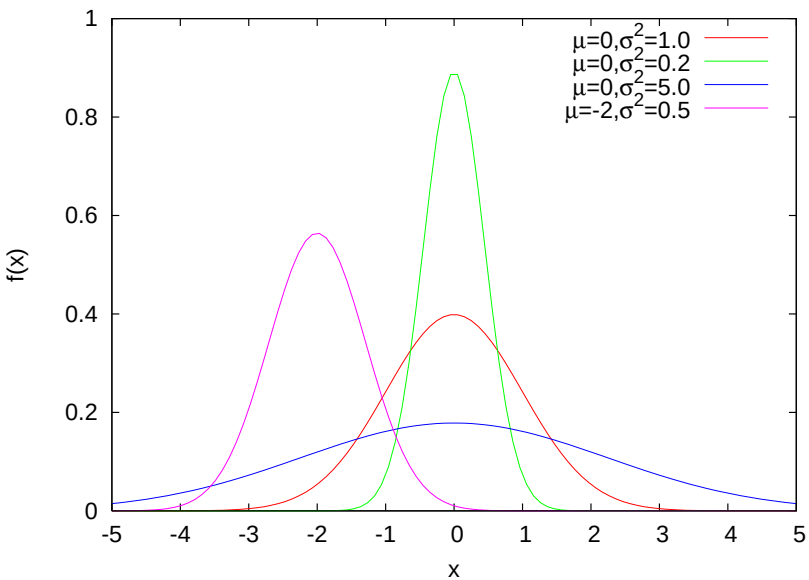
確率密度関数 (PDF)

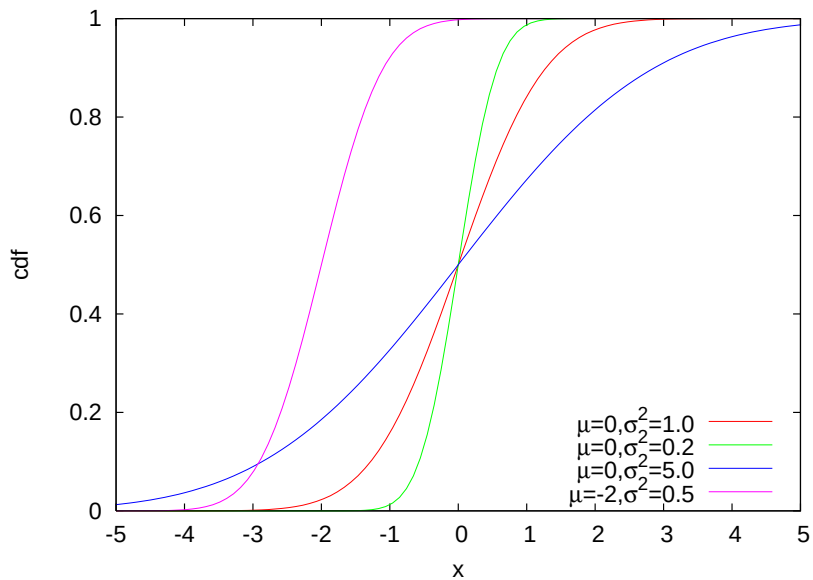
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

累積分布関数 (CDF)

$$F(x) = \frac{1}{2}(1 + erf\frac{x-\mu}{\sigma\sqrt{2}})$$

$$\mu : mean, \sigma^2 : variance$$





指数分布 (exponential distribution)

一定の確率で発生する独立事象の発生間隔は指数分布に従う

- 電話の発呼間隔や、TCP セッションの発生間隔など

確率密度関数 (PDF)

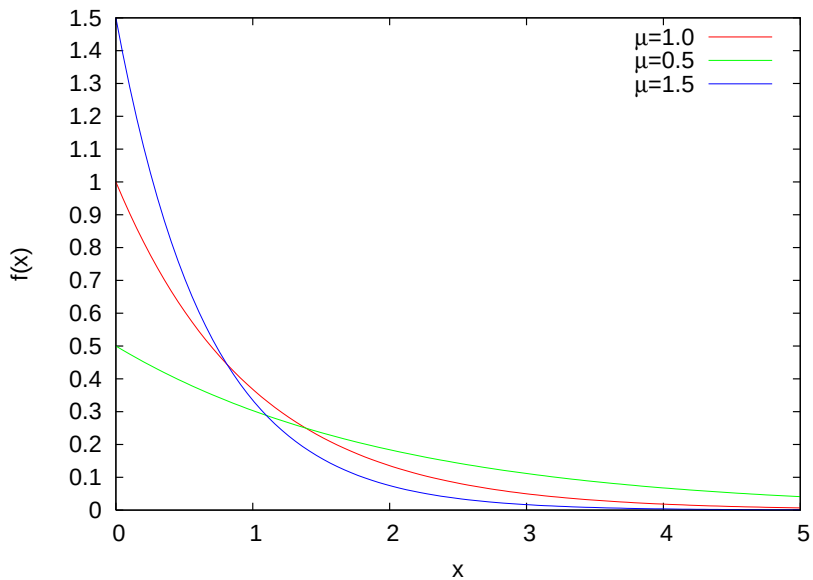
$$f(x) = \lambda e^{-\lambda x}, (x \geq 0)$$

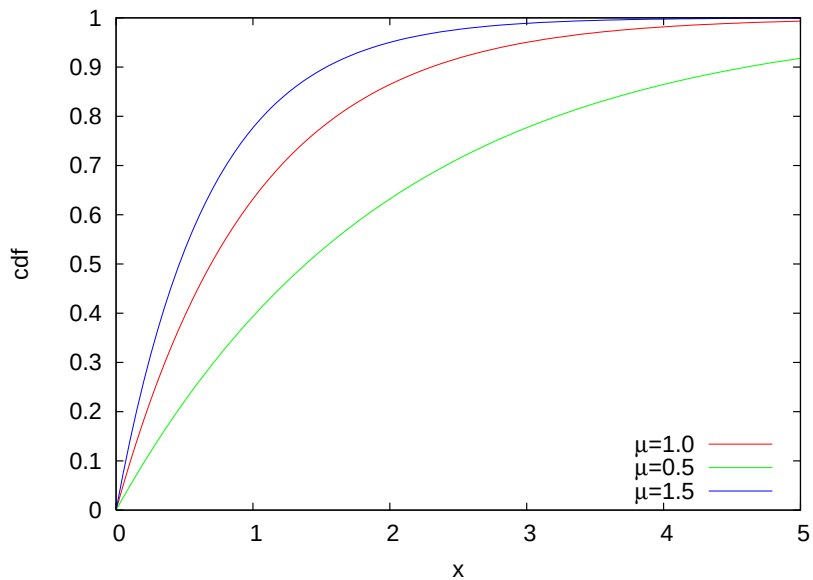
累積分布関数 (CDF)

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

$$\lambda > 0 : rate\ parameter$$

$$mean : E[X] = 1/\lambda, variance : Var[X] = \lambda^{-2}$$





パレート分布 (pareto distribution)

パレート分布: ネットワーク研究で最も使われる べき分布

確率密度関数 (PDF)

$$f(x) = \frac{\alpha}{\kappa}(\frac{\kappa}{x})^{\alpha+1}, (x > \kappa, \alpha > 0)$$

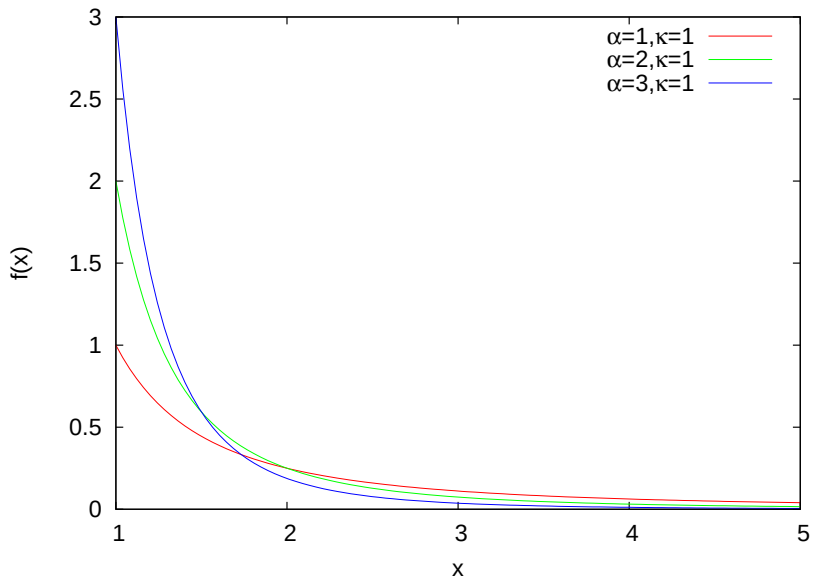
累積分布関数 (CDF)

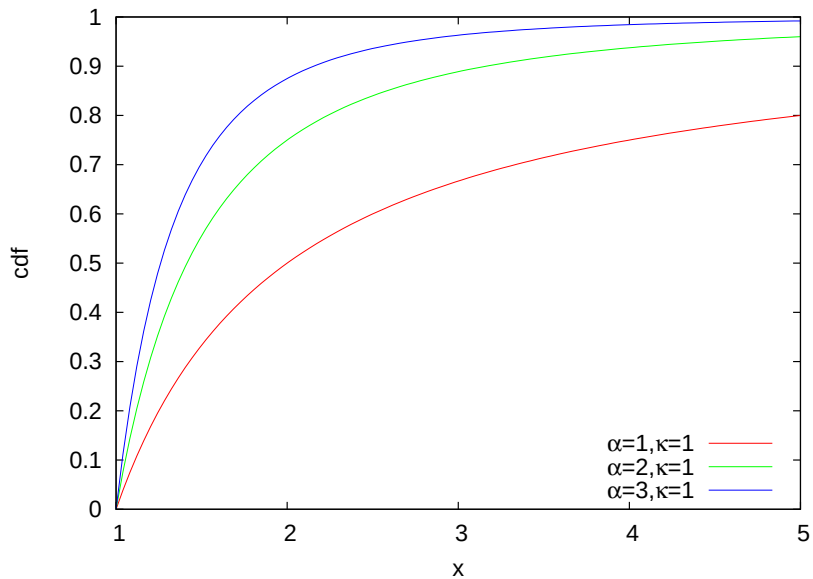
$$F(x) = 1 - (\frac{\kappa}{x})^{\alpha}$$

$$\kappa : minimum\ value\ of\ x, \alpha : pareto\ index$$

$$mean : E[X] = \frac{\alpha}{\alpha - 1}\kappa, (\alpha > 1)$$

if $\alpha \leq 2$, variance $\to \infty$. if $\alpha \leq 1$, mean and variance $\to \infty$.





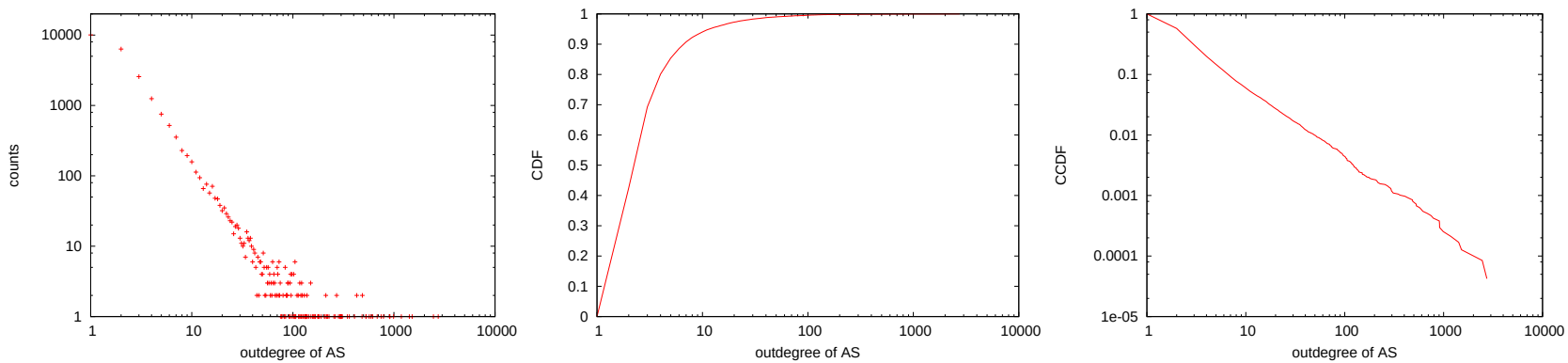
相補累積分布関数 (CCDF)

Complementary Cumulative Distribution Function (CCDF)
べき分布は分布のテイル部分 (値の大きい要素) に特徴

CCDF: x より大きい値の合計が全体に占める割合

$$F(x) = 1 - P[X <= x]$$

- CCDF はログログスケールで描画
  - テイル部分の分布や、スケールフリーな性質を見る



次数分布 (左) CDF(中) CCDF(右)

CCDF のプロット

CDF のプロット

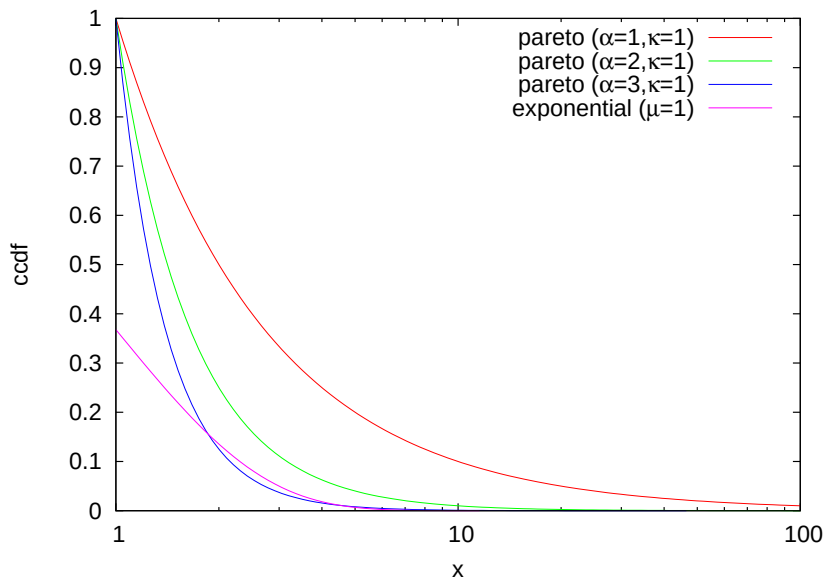
- $x_i, i \in \{1, \ldots, n\}$ を値順にソート
- $(x_i, \frac{1}{n} \sum_{k=1}^{i} k)$ をプロット
- Y 軸は通常リニアスケール

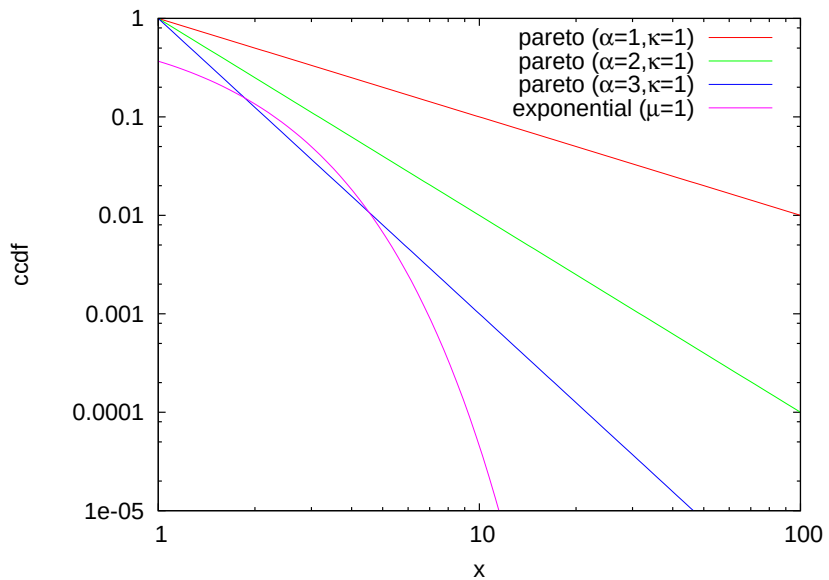
CCDF のプロット

- $x_i, i \in \{1, \ldots, n\}$ を値順にソート
- $(x_i, 1 - \frac{1}{n} \sum_{k=1}^{i-1} k)$ をプロット
- 通常 XY 軸ともログスケール

パレート分布の CCDF

- log-linear (左)
  - 指数分布が直線
- log-log (右)
  - パレート分布が直線





前回の演習: 正規乱数の生成

- 正規分布に従う疑似乱数の生成
  - 一様分布の疑似乱数生成関数 (ruby の rand など) を使って、平均 u、標準偏差 s を持つ疑似乱数生成プログラムを作成
- ヒストグラムの作成
  - 標準正規分布に従う疑似乱数を生成し、そのヒストグラム作成、標準正規分布であることを確認する
- 信頼区間の計算
  - サンプル数によって信頼区間が変化することを確認
    疑似正規乱数生成プログラムを用いて、平均 60, 標準偏差 10 の正規分布に従う乱数列を 10 種類作る。サンプル数 n $=$ 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 の乱数列を作る。
  - 標本から母平均の区間推定
    この 10 種類の乱数列のそれぞれから、母平均の区間推定を行え。信頼度 95%で、信頼区間 "$\pm 1.960 \frac{s}{\sqrt{n}}$" を用いよ。10 種類の結果をひとつの図にプロットせよ。 X 軸にサンプル数を Y 軸に平均値をとり、それぞれのサンプルから推定した平均とその信頼区間を示せ

box-muller 法による正規乱数生成

basic form: creates 2 normally distributed random variables, z_0 and z_1, from 2 uniformly distributed random variables, u_0 and u_1, in $(0, 1]$

$$z_0 = R\cos(\theta) = \sqrt{-2\ln u_0}\cos(2\pi u_1)$$
$$z_1 = R\sin(\theta) = \sqrt{-2\ln u_0}\sin(2\pi u_1)$$

polar form: 三角関数を使わない近似
u_0 and u_1: uniformly distributed random variables in $[-1, 1]$,
$s = u_0^2 + u_1^2$ (if $s = 0$ or $s \geq 1$, re-select u_0, u_1)

$$z_0 = u_0\sqrt{\frac{-2\ln s}{s}}$$
$$z_1 = u_1\sqrt{\frac{-2\ln s}{s}}$$

box-muller 法による正規乱数生成コード

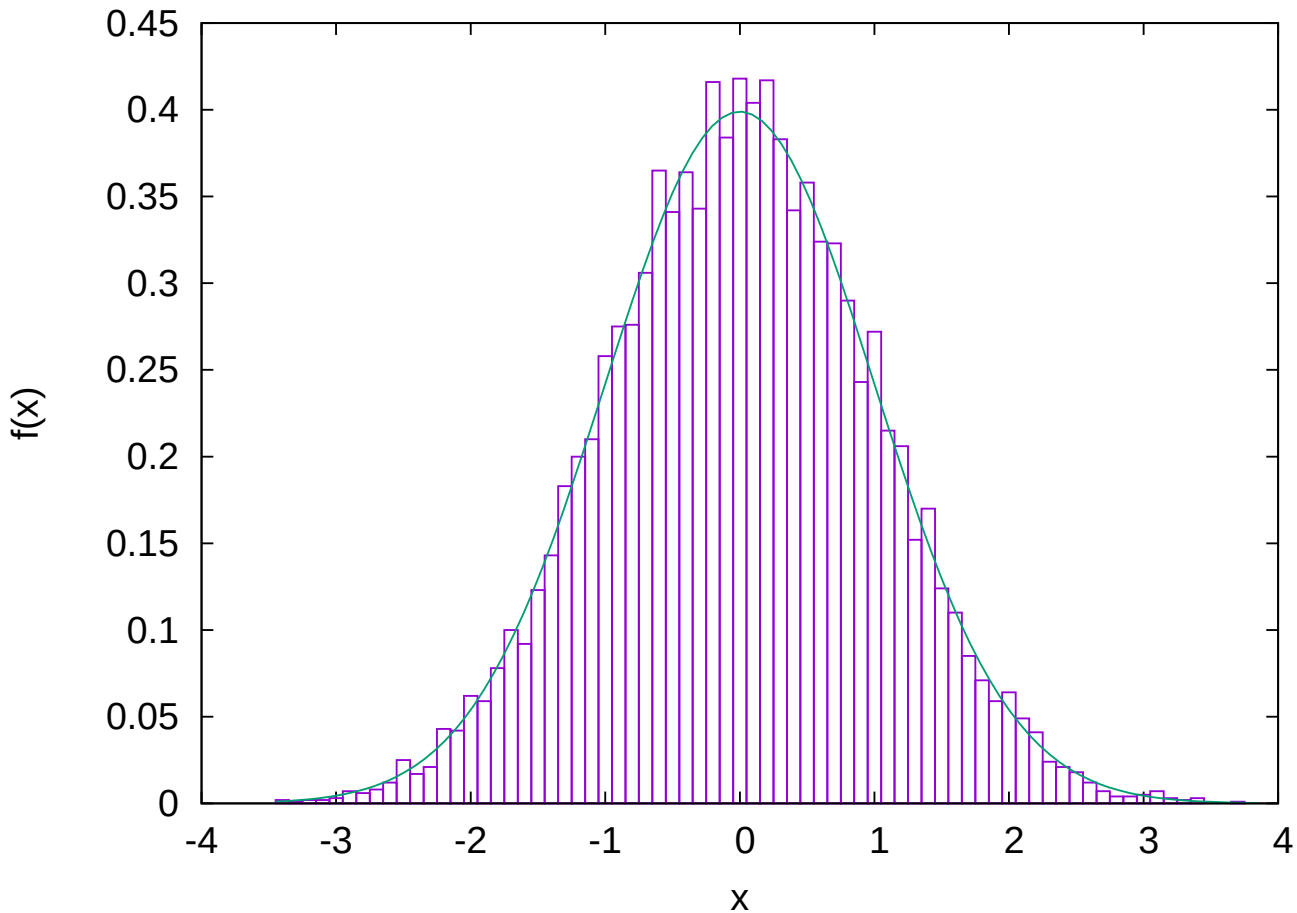
```
# usage: box-muller.rb [n [m [s]]]
n = 1 # number of samples to output
mean = 0.0
stddev = 1.0

n = ARGV[0].to_i if ARGV.length >= 1
mean = ARGV[1].to_i if ARGV.length >= 2
stddev = ARGV[2].to_i if ARGV.length >= 3

# function box_muller implements the polar form of the box muller method,
# and returns 2 pseudo random numbers from standard normal distribution
def box_muller
  begin
    u1 = 2.0 * rand - 1.0  # uniformly distributed random numbers
    u2 = 2.0 * rand - 1.0  # ditto
    s = u1*u1 + u2*u2    # variance
  end while s == 0.0 || s >= 1.0
  w = Math.sqrt(-2.0 * Math.log(s) / s) # weight
  g1 = u1 * w  # normally distributed random number
  g2 = u2 * w  # ditto
  return g1, g2
end
# box_muller returns 2 random numbers.  so, use them for odd/even rounds
x = x2 = nil
n.times do
  if x2 == nil
    x, x2 = box_muller
  else
    x = x2
    x2 = nil
  end
  x = mean + x * stddev  # scale with mean and stddev
  printf "%.6f\n", x
end
```

正規乱数のヒストグラム作成

- 標準正規乱数のヒストグラムを作成し、正規分布であることを確認する
- 標準正規乱数を 10,000 個生成し、小数点 1 桁のビンでヒストグラムを作成



ヒストグラムの作成

- 少数点以下 1 桁でヒストグラムを作成する

```
#
# create histogram: bins with 1 digit after the decimal point
#

re = /(-?\d*\.\d+)/ # regular expression for input numbers

bins = Hash.new(0)

ARGF.each_line do |line|
  if re.match(line)
    v = $1.to_f
    # round off to a value with 1 digit after the decimal point
    offset = 0.5    # for round off
    offset = -offset if v < 0.0
    v = Float(Integer(v * 10 + offset)) / 10
    bins[v] += 1 # increment the corresponding bin
  end
end

bins.sort{|a, b| a[0] <=> b[0]}.each do |key, value|
  puts "#{key} #{value}"
end
```

正規乱数のヒストグラムのプロット

```
set boxwidth 0.1
set xlabel "x"
set ylabel "f(x)"
plot "box-muller-hist.txt" using 1:($2/1000) with boxes notitle, \
     1/sqrt(2*pi)*exp(-x**2/2) notitle with lines
```

注: 標準正規分布の確率密度関数 (PDF)

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

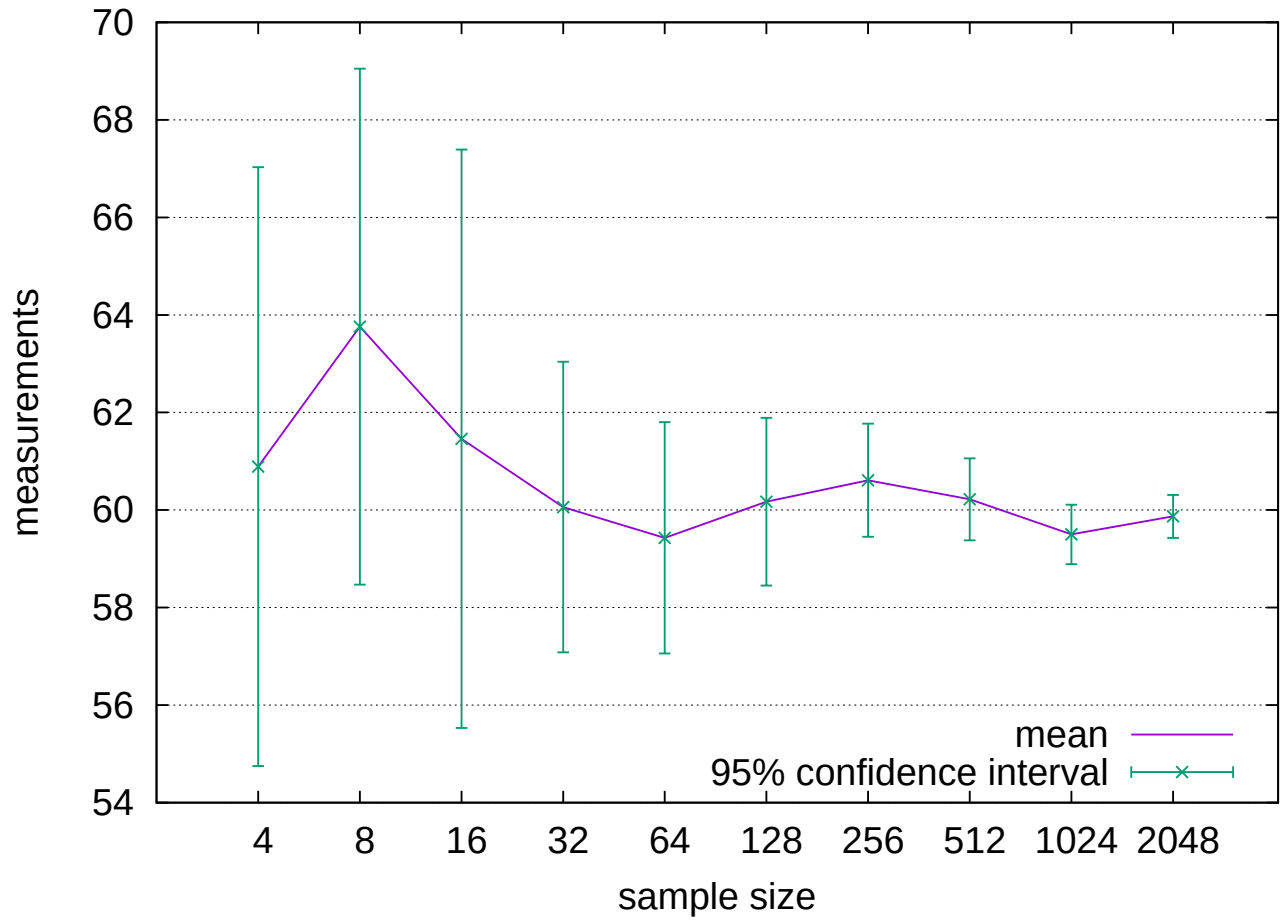
ヒストグラムの描画

```
$ ruby box-muller.rb 10000 > box-muller-data.txt
$ ruby box-muller-hist.rb box-muller-data.txt > box-muller-hist.txt
```

プロットには “box-muller-hist.plt” を使う

平均値の信頼区間とサンプル数の検証

サンプル数が増えるに従い、信頼区間は狭くなる



平均値の信頼区間のサンプル数による変化

信頼区間の描画

データの作成

```
$ ruby box-muller.rb 4 60 10 | ruby conf-interval.rb > conf-interval.txt
$ ruby box-muller.rb 8 60 10 | ruby conf-interval.rb >> conf-interval.txt
$ ruby box-muller.rb 16 60 10 | ruby conf-interval.rb >> conf-interval.txt
...
$ ruby box-muller.rb 2048 60 10 | ruby conf-interval.rb >> conf-interval.txt
```

描画には “conf-interval.plt” を使う

信頼区間の計算

```
# regular expression to read data
re = /^(\d+(\.\d+)?)/

z95 = 1.960 # z_{1-0.05/2}
z90 = 1.645 # z_{1-0.10/2}

sum = 0.0 # sum of data
n = 0 # the number of data
sqsum = 0.0 # su of squares
ARGF.each_line do |line|
    if re.match(line)
      v = $1.to_f
      sum += v
      sqsum += v**2
      n += 1
    end
end

mean = sum / n # mean
var = sqsum / n - mean**2 # variance
stddev = Math.sqrt(var) # standard deviation
se = stddev / Math.sqrt(n) # standard error
ival95 = z95 * se # intarval/2 for 95% confidence level
ival90 = z90 * se # intarval/2 for 90% confidence level

# print n mean stddev ival95 ival90
printf "%d %.2f %.2f %.2f %.2f\n", n, mean, stddev, ival95, ival90
```

信頼区間のプロット

```
set logscale x
set xrange [2:4192]
set key bottom
set xtics (4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048)

set grid ytics
set xlabel "sample size"
set ylabel "measurements"

plot "conf-interval.txt" title "mean" with lines, \
    "conf-interval.txt" using 1:2:4 title "95% confidence interval" with yerrorbars
```

課題 1: ホノルルマラソン 2014 完走時間分布のプロット

- ねらい: 実データから分布を調べる
- データ: 2014 年のホノルルマラソンの記録
  - http://www.pseresults.com/events/647/results
  - 完走者 21,815 人
- 提出項目
  1. 全完走者、男性完走者、女性完走者それぞれの、完走時間の平均、標準偏差、中間値
  2. それぞれの完走時間のヒストグラム
     - 3 つのヒストグラムを別々の図に書く
     - ビン幅は 10 分にする
     - 3 つのプロットは比較できるように目盛を合わせること
  3. それぞれの CDF プロット
     - ひとつの図に 3 つのプロットを書く
  4. オプション
     - 年代別や国別の CDF プロットなど自由
  5. 考察
     - データから読みとれることを記述
- 提出形式: レポートをひとつの PDF ファイルにして SFC-SFS から提出
- 提出〆切: 2015 年 5 月 27 日

ホノルルマラソンデータ

データフォーマット

Place	Num	Chip Time	Lname	Fname	Country	Division	Div Plc	Div Tot	Sex Plc	Sex Total	10Km	21Km	30Km	40Km	Pace
1	3	2:15:35	Chebet	Wilson	KEN	MElite	1	8	1	11507	0:31:50	1:08:45	1:37:47	2:09:49	5:11
2	6	2:16:04	Lonyangata	Paul	KEN	MElite	2	8	2	11507	0:31:50	1:08:45	1:37:47	2:10:05	5:12
3	7	2:16:27	Abraha	Geb	ETH	MElite	3	8	3	11507	0:31:49	1:08:44	1:37:46	2:10:24	5:13
4	5	2:16:37	Kolum	Benjamin	KEN	MElite	4	8	4	11507	0:31:50	1:08:44	1:37:47	2:10:32	5:13
5	4	2:17:54	Adhane	Yemane	ETH	MElite	5	8	5	11507	0:31:50	1:08:45	1:37:47	2:11:06	5:16
6	2	2:17:59	Chelimo	Nicholas	KEN	MElite	6	8	6	11507	0:31:51	1:08:46	1:37:48	2:11:32	5:16
7	25151	2:27:26	Harada	Taku	JPN	M30-34	1	1218	7	11507	0:32:26	1:11:15	1:43:20	2:20:27	5:38
8	8	2:28:23	Arile	Julius	KEN	MElite	7	8	8	11507	0:31:51	1:08:44	1:38:39	2:19:39	5:40
9	30300	2:29:52	Ito	Tatsuya	JPN	M30-34	2	1218	9	11507	0:34:36	1:13:04	1:44:37	2:22:16	5:43
10	F9	2:30:23	Chepkirui	Joyce	KEN	WElite	1	9	1	10308	0:34:37	1:13:07	1:44:50	2:22:43	5:45
...															

- Chip Time: 完走時間
- Category: MElite, WElite, M15-19, M20-24, ..., W15-29, W20-24, ...
  - "No Age"となっている人がいるので注意
- Country: 3-letter country code: e.g., JPN, USA
- 完走者を抽出したら、総数が合っているかチェックすること

課題 1 補足説明

- 要約統計量: 結果を表にすればよい
- ヒストグラム: X 軸は 10 分ビンの完走時間、Y 軸は各ビンの完走者数
- CDF プロット: X 軸は完走時間、Y 軸は CDF [0:1]
- ページ数: 3-6 ページ程度　(ソースコードは必要ない)

Chip Time を抜き出すサンプルコード

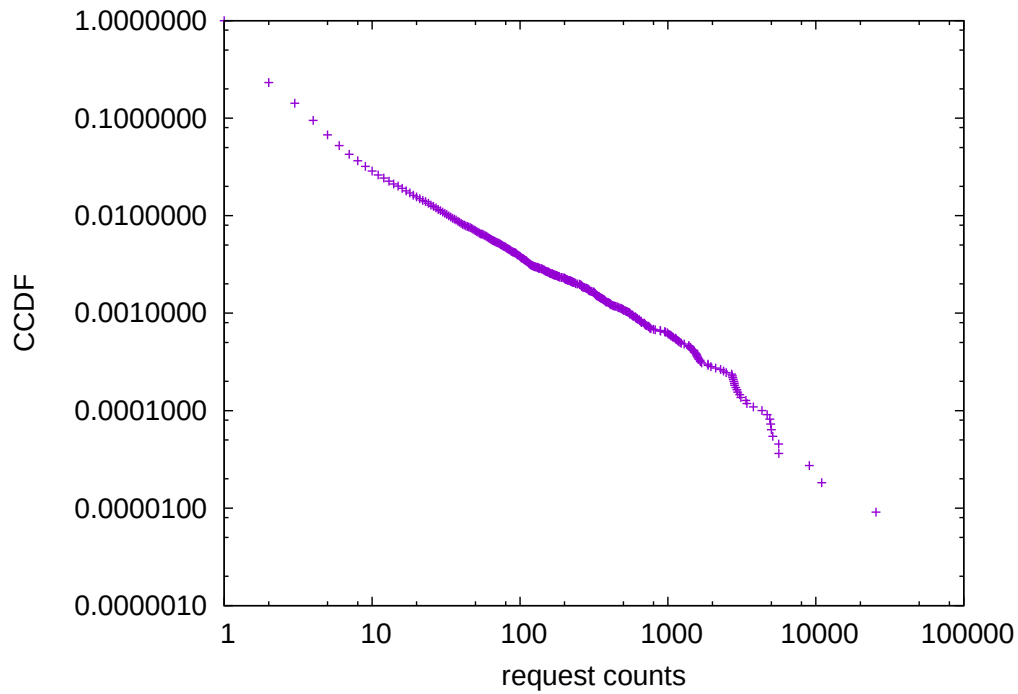
```
# regular expression to read chiptime
re = /^\d+\s+F?\d+\s+(\d{1,2}:\d{2}:\d{2})\s+/

ARGF.each_line do |line|
  if re.match(line)
    puts "#{$1}"
  end
end
```

今回の演習: CCDF のプロット

第 3 回演習で使った JAIST サーバのアクセスログのコンテンツ毎のアクセス数分布を CCDF プロットにする

```
% ./count_contents.rb sample_access_log > contents.txt
% ./make_ccdf.rb contents.txt > ccdf.txt
```



コンテンツ毎のアクセス数抽出スクリプト

```
# output: URL req_count byte_count
# regular expression for apache combined log format
#   host ident user time request status bytes referer agent
re = /^(\S+) (\S+) (\S+) \[(.*?)\] "(.*?)" (\d+) (\d+|-) "(.*?)" "(.*?)"/
# regular expression for request: method url proto
req_re = /(\w+) (\S+) (\S+)/
contents = Hash.new([0, 0])
count = parsed = 0
ARGF.each_line do |line|
  count += 1
  if re.match(line)
    host, ident, user, time, request, status, bytes, referer, agent = $~.captures
    # ignore if the status is not success (2xx)
    next unless /2\d{2}/.match(status)
    if req_re.match(request)
      method, url, proto = $~.captures
      # ignore if the method is not GET
      next unless /GET/.match(method)
      parsed += 1
      # count contents by request and bytes
      contents[url] = [contents[url][0] + 1, contents[url][1] + bytes.to_i]
    else
      # match failed.  print a warning msg
      $stderr.puts("request match failed at line #{count}: #{line.dump}")
    end
  else
    $stderr.puts("match failed at line #{count}: #{line.dump}") # match failed.
  end
end
contents.sort_by{|key, value| -value[0]}.each do |key, value|
  puts "#{key} #{value[0]} #{value[1]}"
end
$stderr.puts "# #{contents.size} unique contents in #{parsed} successful GET requests"
$stderr.puts "# parsed:#{parsed} ignored:#{count - parsed}"
```

コンテンツ毎のアクセス数

```
% cat contents.txt
/project/linuxonandroid/Ubuntu/12.04/full/ubuntu1204-v4-full.zip 25535 17829045
/project/morefont/xiongmaozhongwen.apk 10949 13535294486
/project/morefont/zhongguoxin.apk 9047 9549531354
/project/honi/some_software/Windows/Office_Plus_2010_SP1_W32_xp911.com.rar 5616
/project/morefont/fangzhengyouyijian.apk 5609 2879391721
/pub/Linux/CentOS/5.9/extras/i386/repodata/repomd.xml 5121 12213484
/pub/Linux/CentOS/5.9/updates/i386/repodata/repomd.xml 5006 10969621
/pub/Linux/CentOS/5.9/os/i386/repodata/repomd.xml 4953 6832653
/project/npppluginmgr/xml/plugins.md5.txt 4881 1369547
/project/winpenpack/X-LenMus/releases/X-LenMus_5.3.1_rev5.zip 4689 990250462

...

/pub/Linux/openSUSE/distribution/12.3/repo/oss/suse/x86_64/gedit-3.6.2-2.1.2.x8
/pub/sourceforge/n/nz/nzbcatcher/source/?C=D;O=A 1 1075
/ubuntu/pool/universe/m/mmass/mmass_5.4.1.orig.tar.gz 1 3754849
```

アクセス数を CCDF にするスクリプト

```
#!/usr/bin/env ruby

re = /^\S+\s+(\d+)\s+\d+/

n = 0
counts = Hash.new(0)
ARGF.each_line do |line|
  if re.match(line)
    counts[$1.to_i] += 1
    n += 1
  end
end

cum = 0
counts.sort.each do |key, value|
  comp = 1.0 - Float(cum) / n
  puts "#{key} #{value} #{comp}"
  cum += value
end
```

アクセス数の相補累積度数

```
% cat ccdf.txt
1 84414 1.0
2 9813 0.2315731022366253
3 5199 0.1422446360135818 4
4 3034 0.0949177537254331
5 1636 0.0672990268813779
6 1083 0.0524063976404831 6
7 663 0.0425477683813824 1
8 495 0.0365124302476946 8
9 367 0.0320064085641721 4
10 274 0.0286655803664898 07

...

5616 1 3.6412296432475344e-05
9047 1 2.730922232441202e-05
10949 1 1.8206148216237672e-05
25535 1 9.103074108174347e-06
```

gnuplot スクリプト

```
set logscale
set xlabel "request counts"
set ylabel "CCDF"

plot "ccdf.txt" using 1:3 notitle with points
```

まとめ

第 5 回 多様性と複雑さ

- ロングテール
- Web アクセスとコンテンツ分布
- べき乗則と複雑系
- 演習: べき乗則解析

次回予定

第 6 回 相関 (5/25)

- オンラインお勧めシステム
- 距離と類似度
- 相関係数
- 演習: 相関