Internet Measurement and Data Analysis (5)

Kenjiro Cho

2016-05-09

review of previous class

Class 4 Distribution and confidence intervals (5/2)

- Normal distribution
- Confidence intervals and statistical tests
- Distribution generation
- exercise: confidence intervals
- **assignment 1**

today's topics

Class 5 Diversity and complexity

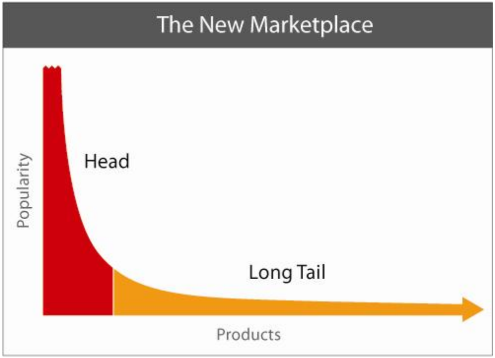
- Long tail
- Web access and content distribution
- Power-law and complex systems
- exercise: power-law analysis

long tail

a business model for online retail services

- head: a small number of bestseller items: for real stores
- tail: diverse low-sales items: covered by online stores

it is now widely used for diverse niche market



source: `http://longtail.com/`

complex systems

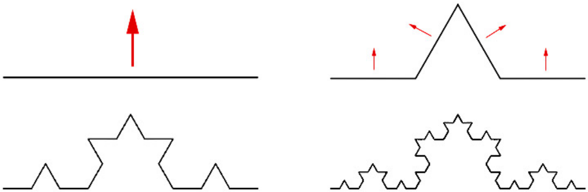
complex systems science

- a system with interfering components that as a whole exhibits complex behavior not obvious from the individual components
- the real world is full of complex systems
- difficult to analyze by traditional methods based on reductionism
  - need to understand a complex system as is, without decomposition
- many studies started in 1990's
  - few remaining problems that can be solved with reductionism
  - analysis and simulations enabled by computers

power-law and complex systems

power-law

- one of the characteristics of complex systems
  - power-law: observed variable changes in proportion to a power of some parameter
  - self-similarity (fractal)
- observed in various natural and social phenomena and Internet services
- scale-free: no typical scale



Koch curve: fractal image similar to coastline

Zipf's law

- an empirical law formulated in 1930's about frequency in ranked data
- the share is inversely proportional to its rank
  - the share of the kth ranked item is proportional to $1/k$
- observed in social science, natural science and data communications
  - the frequency of English words, the population of cities, wealth distribution, etc
  - file size, network traffic
- long-tail in a linear-scale plot, heavy-tail in a log-log plot

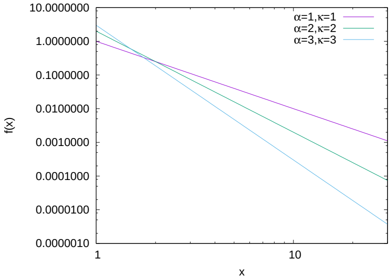
power-law distribution

- power-law distribution: the probability of observing a value is proportional to a power of the value

$$f(x) = ax^{-k}$$

- appears as a straight-line in a log-log plot

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



complexity of the Internet

complexity of topology (network science)

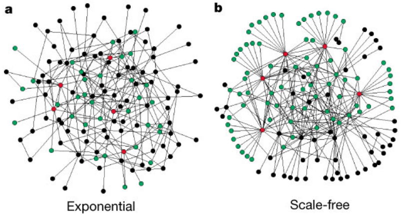
- scale-free: the degree distribution of nodes follows a power-law
  - many small-degree nodes and a small number of large-degree nodes
  - highest-degree nodes greatly exceed the average degree
- small-world:
  - compact: the average distance between 2 nodes is short
  - clusters: nodes are highly clustered

traffic behavior (time-series analysis)

- self-similarity
- long-range dependence

scale-free network

- the degree distribution of network nodes follows power-law
  - many small-degree nodes, small number of large-degree nodes
  - highest-degree nodes greatly exceed the average degree
- small-world
  - compact: the average distance between 2 nodes is short
  - clusters: nodes are highly clustered
- construction: preferential attachment: rich get richer
  - higher probability to attach to a high-degree node
- fault-tolerance, attack-tolerance
  - robust against random failures
  - vulnerable to an attack to a hub node

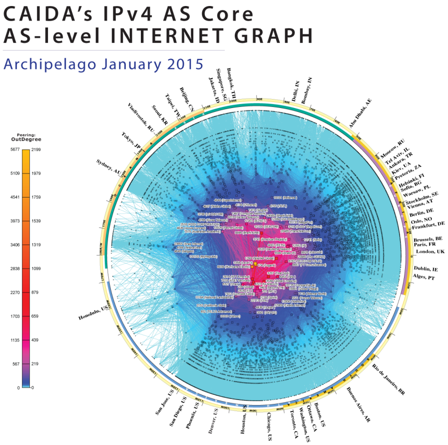


source: Error and attack tolerance of complex networks. R. Albert, H. Jeong, A. Barabasi. Nature 406, July 2000.

example: AS structure of the Internet

CAIDA AS CORE MAP 2015/01

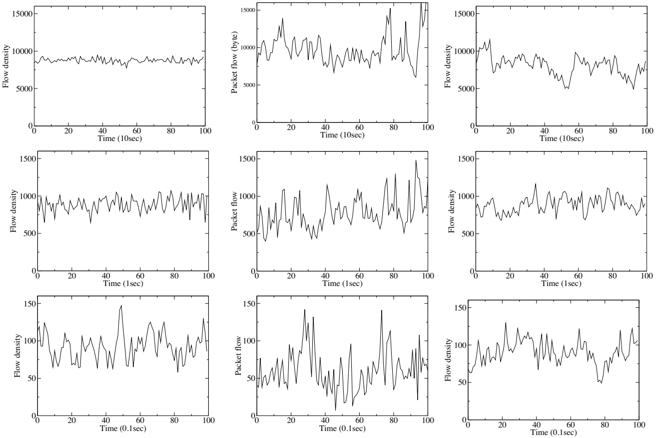
- visualization of AS topology using skitter/ark data
- longitude of AS (registered location), out-degree of AS



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

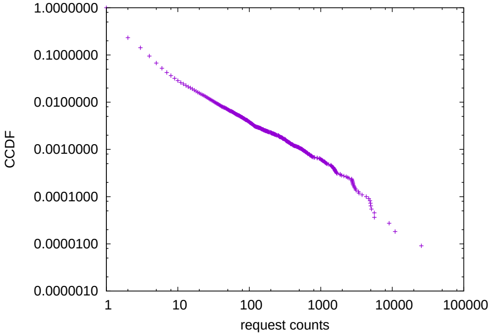
self-similarity in network traffic

- exponential model (left), real traffic (middle), self-similar model (right)
- time scale: 10sec (top), 1 sec (middle), 0.1 sec (bottom)



Web access and content distribution

- power-law can be observed everywhere on the web
  - the number of incoming links and access count of web page, occurrences of search keywords



content access count distribution of the JAIST web server

various distributions

- binomial distribution
- poisson distribution
- normal distribution
- exponential distribution
- power-law distribution

binomial distribution

- bernoulli trial: a trial is random and has only 2 outcomes
- discrete probability distribution of the number of success k for n trials, with the probability of success p for a trial

PDF

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

here

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

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

when n is large, a binomial distribution can be approximated by a poisson distribution

poisson distribution

the occurrence rate of rare events follows poisson distribution

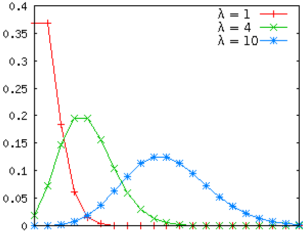
- death toll of traffic accidents, the number of mutations of DNA, etc

poisson distribution is expressed by a single expected value $\lambda > 0$

PDF

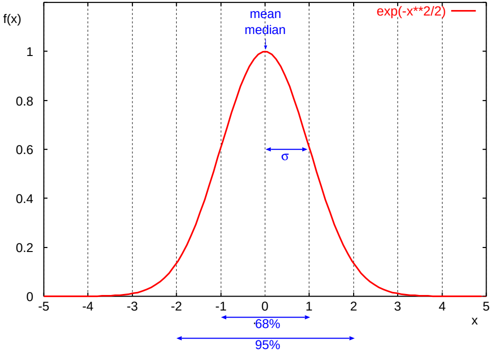
$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

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



normal distribution (1/2)

- also known as gaussian distribution
- defined by 2 parameters: $N(\mu, \sigma^2)$, μ:mean, σ^2:variance
- sum of random variables follows normal distribution
- standard normal distribution: $\mu = 0, \sigma = 1$
- in normal distribution
  - 68% within $(mean - stddev, mean + stddev)$
  - 95% within $(mean - 2 * stddev, mean + 2 * stddev)$



normal distribution (2/2)

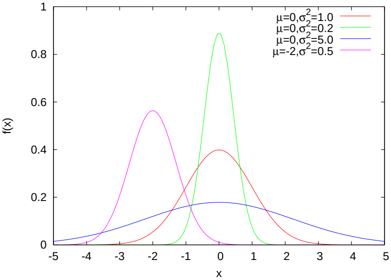
probability density function (PDF)

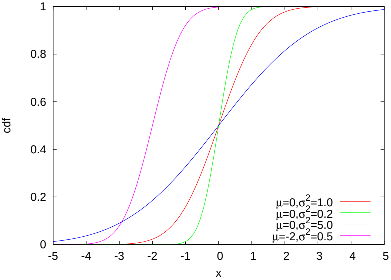
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

cumulative distribution function (CDF)

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

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





exponential distribution

the intervals of independent events occuring at a constant rate follow an exponential distribution

- call intervals in telephone systems, session intervals of TCP connections, etc

PDF

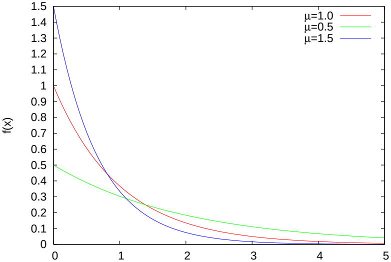
$$f(x) = \lambda e^{-\lambda x}, (x \geq 0)$$

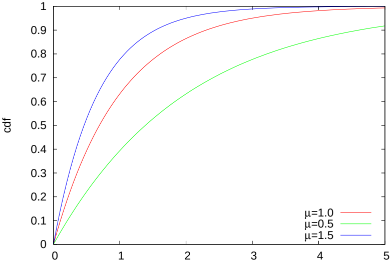
CDF

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

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

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





pareto distribution

most widely used power-law distribution in networking research

PDF

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

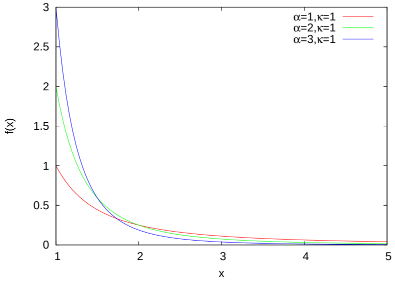
CDF

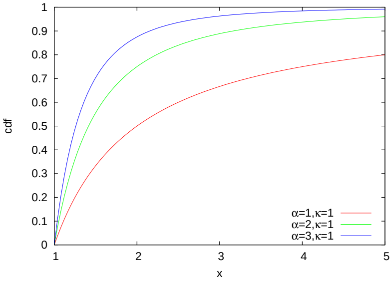
$$F(x) = 1 - (\frac{\kappa}{x})^{\alpha}$$

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

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

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





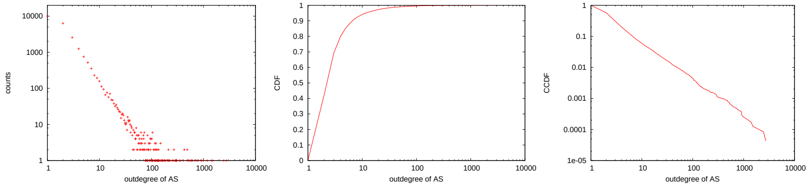
CCDF

Complementary Cumulative Distribution Function (CCDF)
in power-law distribution, the tail of distribution is often of interest

ccdf: probability of observing x or more

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

- plot ccdf in log-log scale
  - to see the tail of the distribution or scaling property



degree distribution(left) CDF(middle) CCDF(right)

plotting CCDF

to plot CDF

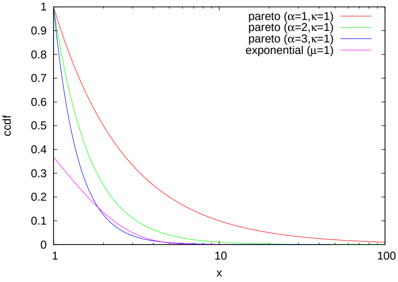
- sort $x_i, i \in \{1, \ldots, n\}$ by value
- plot $(x_i, \frac{1}{n} \sum_{k=1}^{i} k)$
- Y-axis is usually in linear scale

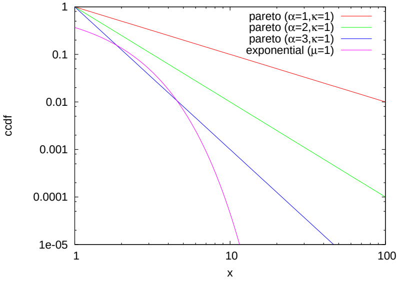
to plot CCDF

- sort $x_i, i \in \{1, \ldots, n\}$ by value
- plot $(x_i, 1 - \frac{1}{n} \sum_{k=1}^{i-1} k)$
- both X-axis and Y-axis are in log scale

CCDF of pareto distribution

- log-linear (left)
  - exponential distribution: straight line
- log-log (right)
  - pareto distribution: straight line





previous exercise: normally distributed random numbers

- generating pseudo random numbers that follow the normal distribution
  - write a program to generate normally distributed random numbers with mean u and standard deviation s, using a uniform random number generator function (e.g., rand in ruby)
- plotting a histogram
  - generate random numbers that follow the standard normal distribution, plot the histogram to confirm the standard normal distribution.
- computing confidence intervals
  - observe confidence interval changes according to sample size. use the normally distributed random number generator to produce 10 sets of normally distributed random numbers with mean 60 and standard deviation 10. sample size n = 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048
  - compute the confidence interval of the population mean from each sample set.
    use confidence level 95% and confidence interval "$\pm 1.960 \frac{s}{\sqrt{n}}$".
    plot the results of 10 sets in a single graph. plot sample size n on the X-axis in log-scale and mean and confidence interval on the Y-axis in linear scale

box-muller transform

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: approximation without trigonometric functions
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)

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

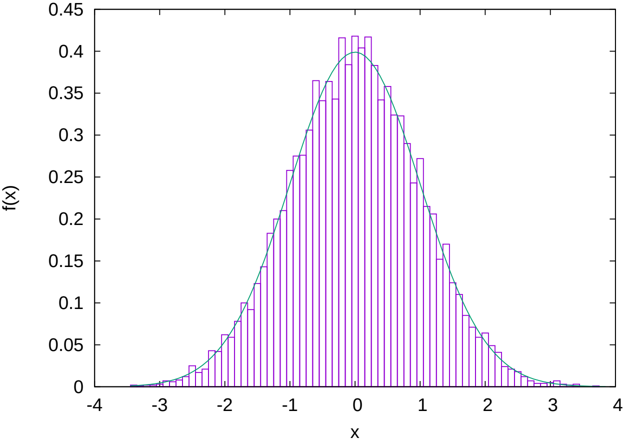
$$z_1 = u_1\sqrt{\frac{-2\ln s}{s}}$$

random number generator code by box-muller transform

```
# 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
```

plot a histogram of normally distributed random numbers

- plot a histogram of random numbers following the standard normal distribution, and confirm that they are normally distributed
- generate 10,000 random numbers from the standard normal distribution, use bins with one decimal place for the histogram



plotting a histogram

- plot a histogram using bins with one decimal place

```
#
# 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
```

plotting a histogram of the standard normal distribution

```
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 linetype 3
```

note: probability density function (PDF) of standard normal distribution

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

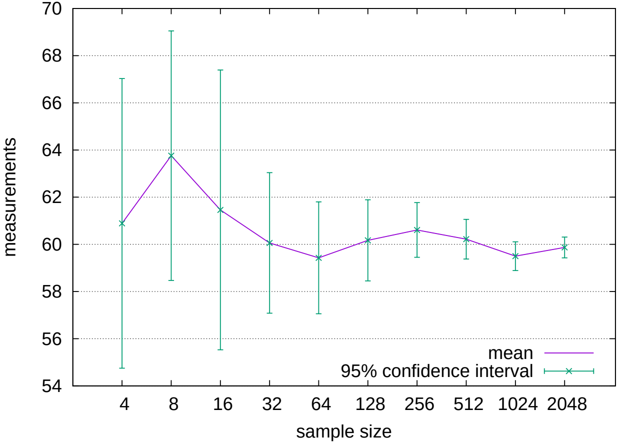
to plot a histogram

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

then, use “box-muller-hist.plt” for plotting

the confidence interval of sample mean and sample size

the confidence interval becomes narrower as the sample size increases



the confidence interval of sample mean and sample size

plotting the confidence intervals

to make data

```
$ 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
```

then, use “conf-interval.plt” for plotting

computing confidence intervals

```
# 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
```

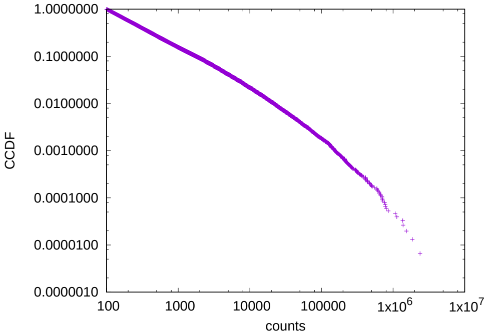
plotting confidence intervals

```
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 lt 3
```

today's exercise: CCDF plots

- plot the US surname distribution in CCDF
  - poopular surnames (100 or more) in the US Census in 2000
  - csv format: comma-separated-variables

```
% head us-surnames.csv
name,rank,count,prop100k,cum_prop100k,pctwhite,pctblack,pctapi,pctaian,pct2prace,pcthispanic
SMITH,1,2376206,880.85,880.85,73.35,22.22,0.40,0.85,1.63,1.56
JOHNSON,2,1857160,688.44,1569.30,61.55,33.80,0.42,0.91,1.82,1.50
WILLIAMS,3,1534042,568.66,2137.96,48.52,46.72,0.37,0.78,2.01,1.60
BROWN,4,1380145,511.62,2649.58,60.71,34.54,0.41,0.83,1.86,1.64
JONES,5,1362755,505.17,3154.75,57.69,37.73,0.35,0.94,1.85,1.44
...

% ./make_ccdf.rb us-surnames.csv > ccdf.txt
```



script to convert the counts to CCDF

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

re = /^[A-Z]+,\d+,(\d+),\d+/    # for US surnames
#re = /^\S+\s+(\d+)\s+\d+/      # for JAIST web server logs

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

$stderr.puts "# #{n} entries matched"
```

cumulative surname counts

```
% cat ccdf.txt
100 1236 1.0
101 1108 0.9918507822853413
102 1084 0.9845454965022977
103 1149 0.977398447956432
104 1084 0.969822840226543
105 1103 0.9626757916806773
106 1061 0.9554034719887124
107 1028 0.9484080674618088
108 1031 0.9416302391360247

...

1380145 1 2.637287286300083e-05
1534042 1 1.9779654647278377e-05
1857160 1 1.3186436431444903e-05
2376206 1 6.593218215722452e-06
```

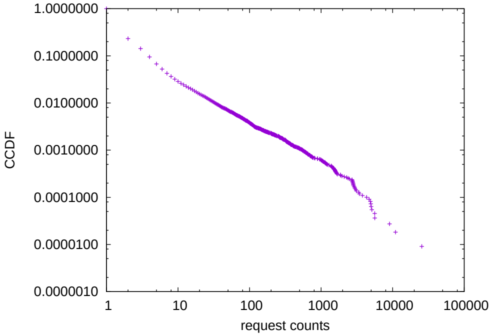
gnuplot script for plotting the counts in CCDF

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

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

another exercise: CCDF plots

extract the access count of each unique content from the JAIST server access log, plot the access count distribution in CCDF

```
first edit the regular expression in the script (e.g., vim make_ccdf.rb)
% ./count_contents.rb sample_access_log > contents.txt
% ./make_ccdf.rb contents.txt > ccdf.txt
```



extracting the access count of each unique content

```
# 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}"
```

access count of each unique content

```
% 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/nppluginmgr/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
```

cumulative access counts

```
% cat ccdf.txt
1 84414 1.0
2 9813 0.2315731022366253
3 5199 0.14224463601358184
4 3034 0.0949177537254331
5 1636 0.06729902688137779
6 1083 0.05240639764048316
7 663 0.04254776838138241
8 495 0.03651243024769468
9 367 0.03200640856417214
10 274 0.028665580366489807

...

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

assignment 1: the finish time distribution of a marathon

- purpose: investigate the distribution of a real-world data set
- data: the finish time records from honolulu marathon 2015
  - http://www.pseresults.com/events/741/results
  - the number of finishers: 21,554
- items to submit
  1. mean, standard deviation and median of the total finishers, male finishers, and female finishers
  2. the distributions of finish time for each group (total, men, and women)
     - plot 3 histograms for 3 groups
     - use 10 minutes for the bin size
     - use the same scale for the axes to compare the 3 plots
  3. CDF plot of the finish time distributions of the 3 groups
     - plot 3 groups in a single graph
  4. discuss differences in finish time between male and female. what can you observe from the data?
  5. optional
     - other analysis of your choice (e.g., discussion on differences among age groups)
- submission format: a single PDF file including item 1-5
- submission method: upload the PDF file through SFC-SFS
- submission due: 2016-05-17

honolulu marathon data set

data format (compacted to fit in the slide)

Place	Chip Time	Number	Lname	Fname	Country	Category	Cat Place	Cat Total	5K	10K		40K	Gndr Place	Gndr Total	Pace
1	2:11:43	3	Kiprotich	Filex	KEN	MElite	1	5	16:07	31:40	...	2:04:48	1	11346	5:02
2	2:12:46	1	Chebet	Wilson	KEN	MElite	2	5	16:07	31:41	...	2:05:57	2	11346	5:04
3	2:13:24	8	Limo	Daniel	KEN	MElite	3	5	16:06	31:41	...	2:06:13	3	11346	5:06
4	2:15:27	6	Kwambai	Robert	KEN	MElite	4	5	16:08	31:41	...	2:07:29	4	11346	5:10
5	2:18:36	4	Mungara	Kenneth	KEN	MElite	5	5	16:07	31:40	...	2:09:42	5	11346	5:18
6	2:27:58	11	Neuschwander	Florian	DEU	M30-34	1	1241	17:46	34:50	...	2:20:31	6	11346	5:39
7	2:28:34	F1	Chepkirui	Joyce	KEN	WElite	1	7	16:53	33:21	...	2:20:56	1	10207	5:40
8	2:28:42	28803	Takahashi	Koji	JPN	M25-29	1	974	16:54	33:22	...	2:20:52	7	11346	5:41
9	2:28:55	F5	Karimi	Lucy	KEN	WElite	2	7	16:54	33:22	...	2:20:58	2	10207	5:41
10	2:29:44	F6	Ochichi	Isabella	KEN	WElite	3	7	16:53	33:22	...	2:21:46	3	10207	5:43
...															

- Chip Time: finish time
- Number: bib number
- Category: MElite, WElite, M15-19, M20-24, ..., W15-29, W20-24, ...
  - note: 2 runners have "No Age" for Category, and num:18035 doesn't have cat/gender totals and its cat/gender placements are not reflected to the following entries
- Country: 3-letter country code: e.g., JPN, USA
- check the number of the total finishers when you extract the finishers

summary

Class 5 Diversity and complexity

- Long tail
- Web access and content distribution
- Power-law and complex systems
- exercise: power-law analysis

next class

Class 6 Correlation (5/16)

- Online recommendation systems
- Distance
- Correlation coefficient
- exercise: correlation analysis