

インターネット計測とデータ解析 第3回

長 健二郎

2013年4月24日

前回のおさらい

第2回 データとばらつき (4/17)

- ▶ 要約統計量 (平均、標準偏差、分布)
- ▶ サンプルング
- ▶ グラフによる可視化
- ▶ 演習: gnuplot によるグラフ描画

今日のテーマ

第3回 データの収集と記録

- ▶ ネットワーク管理ツール
- ▶ データフォーマット
- ▶ ログ解析手法
- ▶ 演習: ログデータと正規表現

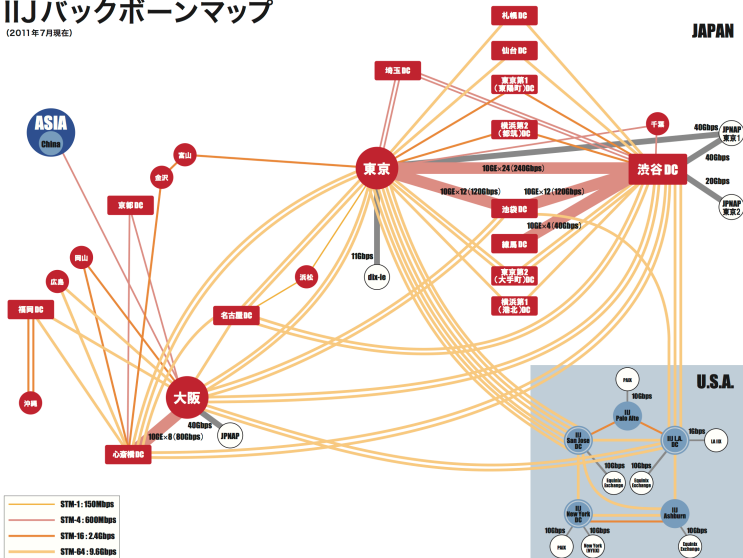
ネットワーク管理ツール

ネットワークの構成: 日本のある ISP の場合

東京-大阪を中心に地域拠点を冗長構成で接続

IIJバックボーンマップ

(2011年7月現在)



ルータ

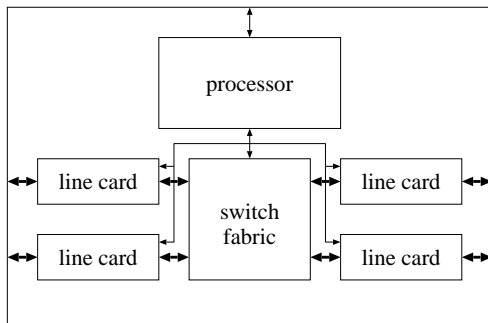
ルータ: ネットワークを接続する装置

- ▶ 機能
 - ▶ 経路制御、パケットフォワーディング、管理機能
- ▶ ルータの分類
 - ▶ コアルータ、エッジルータ、ブロードバンドルータなど



ルータのアーキテクチャ

- ▶ fast path: ハードウェアサポート
- ▶ slow path: ソフトウェア処理
 - ▶ ICMP などの制御用パケットは通常スローパスで処理



一般的なネットワーク管理ツール

ネットワークの管理用ツール (もともと計測ツールではない)

- ▶ ping
 - ▶ 到達性、ラウンドトリップタイム
- ▶ traceroute
 - ▶ 経路観測
- ▶ tcpdump
 - ▶ パケットキャプチャリング
- ▶ SNMP
 - ▶ ルータの状態把握

ping

- ▶ ネットワーク到達性確認ツール
- ▶ ICMP-echo request/reply
- ▶ 制約
 - ▶ 到達性がある \neq ネットワークの正常動作
 - ▶ ICMP は遅延計測に適さない場合がある

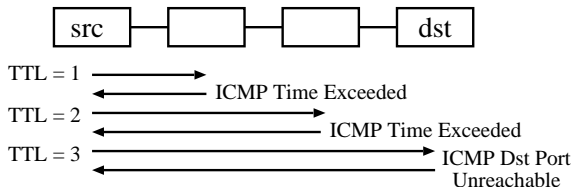
ping sample output

```
% ping -c 10 www.ait.ac.th
PING www.ait.ac.th (202.183.214.46): 56 data bytes
64 bytes from 202.183.214.46: icmp_seq=0 ttl=114 time=112.601 ms
64 bytes from 202.183.214.46: icmp_seq=1 ttl=114 time=106.730 ms
64 bytes from 202.183.214.46: icmp_seq=2 ttl=114 time=106.173 ms
64 bytes from 202.183.214.46: icmp_seq=3 ttl=114 time=111.704 ms
64 bytes from 202.183.214.46: icmp_seq=4 ttl=114 time=112.412 ms
64 bytes from 202.183.214.46: icmp_seq=5 ttl=114 time=114.603 ms
64 bytes from 202.183.214.46: icmp_seq=6 ttl=114 time=111.755 ms
64 bytes from 202.183.214.46: icmp_seq=7 ttl=114 time=115.273 ms
64 bytes from 202.183.214.46: icmp_seq=8 ttl=114 time=106.525 ms
64 bytes from 202.183.214.46: icmp_seq=9 ttl=114 time=111.562 ms

--- www.ait.ac.th ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max/stddev = 106.173/110.934/115.273/3.142 ms
```

traceroute

- ▶ IP パケットのループ検出のための TTL (time-to-live) を利用
 - ▶ ルータはパケット転送時に TTL を 1 減らす
 - ▶ TTL が 0 になると ICMP TIME EXCEEDED を送信者に返す
- ▶ 制約
 - ▶ 経路は時間とともに変化する可能性
 - ▶ 非対称な経路も存在する
 - ▶ 行きのパスしか分からない
 - ▶ 通常ルータはインターフェイス毎に IP アドレスを持つ
 - ▶ IP アドレスだけでは同一ルータか判定できない



traceroute sample output

```
% traceroute www.ait.ac.th
traceroute to www.ait.ac.th (202.183.214.46), 64 hops max, 40 byte packets
 1  202.214.86.129 (202.214.86.129)  0.687 ms  0.668 ms  0.730 ms
 2  jc-gw0.IIJ.Net (202.232.0.237)  0.482 ms  0.390 ms  0.348 ms
 3  tky001ix07.IIJ.Net (210.130.143.233)  0.861 ms  0.872 ms  0.729 ms
 4  tky001bb00.IIJ.Net (210.130.130.76)  10.107 ms  1.026 ms  0.855 ms
 5  tky001ix04.IIJ.Net (210.130.143.53)  1.111 ms  1.012 ms  0.980 ms
 6  202.232.8.142 (202.232.8.142)  1.237 ms  1.214 ms  1.120 ms
 7  ge-1-1-0.tokenf-cr2.ix.singtel.com (203.208.172.209)  1.338 ms  1.501 ms
    1.480 ms
 8  p6-13.sngtp-cr2.ix.singtel.com (203.208.173.93)  93.195 ms  203.208.172.
229 (203.208.172.229)  88.617 ms  87.929 ms
 9  203.208.182.238 (203.208.182.238)  90.294 ms  88.232 ms  203.208.182.234
(203.208.182.234)  91.660 ms
10  203.208.147.134 (203.208.147.134)  103.933 ms  104.249 ms  103.986 ms
11  210.1.45.241 (210.1.45.241)  103.847 ms  110.924 ms  110.163 ms
12  st1-6-bkk.csloxinfo.net (203.146.14.54)  131.134 ms  129.452 ms  111.408
ms
13  st1-6-bkk.csloxinfo.net (203.146.14.54)  106.039 ms  105.078 ms  105.196
ms
14  202.183.160.121 (202.183.160.121)  111.240 ms  123.606 ms  112.153 ms
15  * * *
16  * * *
17  * * *
```

tcpdump

- ▶ パケットキャプチャリングのためのツール
 - ▶ パケットの先頭 N バイトを記録
- ▶ 柔軟なフィルタリング機能
 - ▶ 例: 特定のホストからの TCP SYN パケット
- ▶ 詳細な解析が可能
- ▶ 制約
 - ▶ データサイズが大きい
 - ▶ 高速ネットワークでは技術的に困難

tcpdump sample output

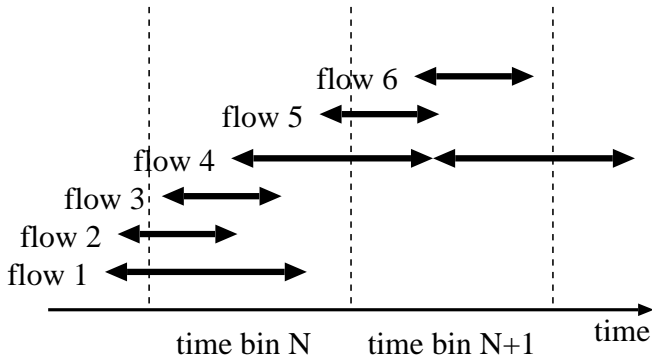
```
18:45:29.767497 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  S 3304970307:3304970307(0) win 65535 <mss 1460,nop,nop,sackOK,nop, \  
  wscale 1,nop,nop,timestamp 710778973 0>  
18:45:29.770038 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  S 3129218301:3129218301(0) ack 3304970308 win 65535 <mss 1460,nop, \  
  ywscale 1,nop,nop,timestamp 2523776361 710778973,nop,nop,sackOK>  
18:45:29.770090 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  . ack 1 win 33304 <nop,nop,timestamp 710778973 2523776361>  
18:45:29.787084 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  P 1:521(520) ack 1 win 33304 <nop,nop,timestamp 710778975 2523776361>  
18:45:29.791392 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  P 1:222(221) ack 521 win 33304 <nop,nop,timestamp 2523776363 710778975>  
18:45:29.887024 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  . ack 222 win 33304 <nop,nop,timestamp 710778985 2523776363>  
18:45:34.792726 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  F 222:222(0) ack 521 win 33304 <nop,nop,timestamp 2523776864 710778985>  
18:45:34.792763 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  . ack 223 win 33304 <nop,nop,timestamp 710779475 2523776864>  
18:45:42.528539 IP 202.214.86.132.50052 > 202.210.220.18.80: \  
  F 521:521(0) ack 223 win 33304 <nop,nop,timestamp 710780249 2523776864>  
18:45:42.531088 IP 202.210.220.18.80 > 202.214.86.132.50052: \  
  . ack 522 win 33303 <nop,nop,timestamp 2523777637 710780249>
```

SNMP (Simple Network Management Protocol)

- ▶ SNMP
 - ▶ リモートから情報問い合わせ、情報の格納、トラップの設定
 - ▶ UDP の利用 (信頼性がない)
- ▶ 標準化されたトラフィック統計情報
 - ▶ ほとんどのルータ、スイッチ、ホスト OS に実装
 - ▶ 多くの管理ツールが存在
- ▶ MIB (Management Information Base)
 - ▶ SNMP オブジェクトの木構造データベース
 - ▶ 例: interfaces.ifTable.ifEntry.ifOutOctets
 - ▶ 標準 MIB とプライベート MIB
 - ▶ get, set, get-next to access MIB
- ▶ 制約
 - ▶ 標準化されている情報は限られている
 - ▶ オブジェクトのサポートを後から追加することは難しい
 - ▶ アクセスの効率が悪い

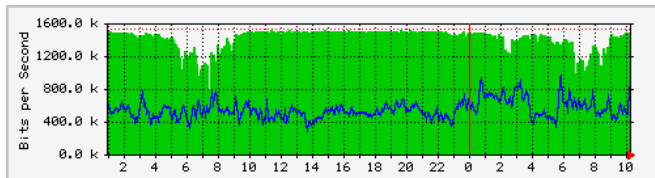
フロー計測

- ▶ SNMP によるインターフェイスカウンタ値による計測の限界
 - ▶ 総量は分かるが、それ以上の情報取得が困難
- ▶ フローベースの計測
 - ▶ 5 tuples (protocol, srcaddr, dstaddr, srcport, dstport), AS, etc
 - ▶ プロトコル: NetFlow、sFlow、IPFIX、...
- ▶ サンプリングによるデータ量削減も可能



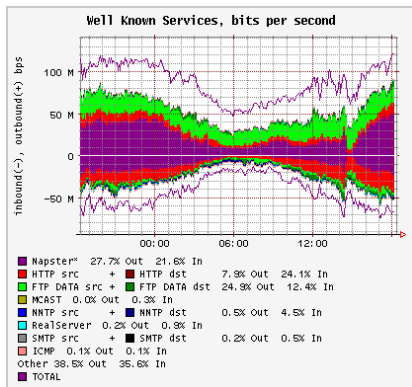
MRTG

- ▶ SNMP データをグラフ化するツール
- ▶ 古い時系列データを集約しデータ量を一定にする仕組み
 - ▶ daily, weekly, monthly, yearly
- ▶ inbound/outbound traffic
 - ▶ 他の時系列データにも利用可能



RRDtool

- ▶ RRDtool: MRTG の作者による後継ツール
 - ▶ 柔軟な設定が可能に
 - ▶ 任意の時系列データを扱えるよう工夫
- ▶ flowscan による RRDtool を使った NetFlow データのグラフ



from caida web site

ネットワーク管理ツールのまとめ

- ▶ もともと管理用ツールで計測ツールではない
- ▶ 多くの計測で利用されている
- ▶ 利用に際しては仕組みと制約を理解する必要がある

データフォーマット

いろいろなログ

- ▶ web server accesslog
- ▶ mail log
- ▶ syslog
- ▶ firewall log
- ▶ IDS log
- ▶ その他 あらゆる記録

なぜログ解析をするのか？

- ▶ 現状の把握
 - ▶ 新しい発見: 技術の進歩や利用形態の変化
 - ▶ そのうえで将来予測
- ▶ セキュリティ上の問題や機器故障、それらの兆候の把握
- ▶ 解析技術の向上
 - ▶ 自動化
- ▶ 障害のレポート、問題への対応
- ▶ 記録の必要性
 - ▶ 法的理由、その他

そもそも解析されないログには意味がない
(ログを取るだけで安心してはいけない)

ログ解析の問題

- ▶ 膨大なデータ量
- ▶ 必要な情報や精度の欠如、時刻情報や内容の信憑性
- ▶ (収集システムの障害などによる) 記録の欠落
- ▶ さまざまなフォーマットが存在
- ▶ 解析には時間と労力が必要
- ▶ 解析は難しいという思い込み

ログの管理

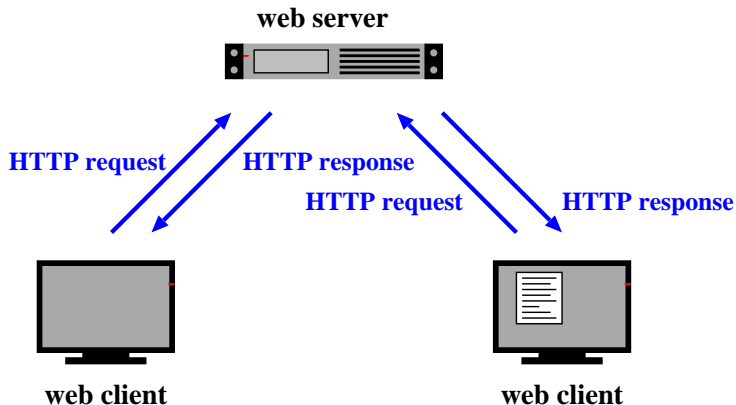
- ▶ ログ収集
 - ▶ プログラミング (syslog API の利用など)
 - ▶ 収集システム構築
- ▶ ログローテーション
 - ▶ 古いデータを一定期間保存した後削除
 - ▶ ログサイズ、時間、データの古さ
 - ▶ ローテーション時にデータを失わないよう工夫
- ▶ RRD (Round Robin Database)
 - ▶ 古いログを集約することで、データサイズを一定にする
 - ▶ 例: 5分粒度で1週間、2時間粒度で1カ月、1日粒度で1年
- ▶ 可視化
 - ▶ グラフ化して web に貼ることで状況の把握を容易に

さまざまなフォーマット

- ▶ web server access log
- ▶ mail log
- ▶ DHCP server log
- ▶ syslog

Web サーバへのアクセス

- ▶ HTTP プロトコル
- ▶ リクエスト/レスポンス



web server access log

- ▶ Apache Common Log Format
 - ▶ client_IP client_ID user_ID time request status_code size
- ▶ Apache Combined Log Format
 - ▶ Common Log Format に referer と User-agent を追加
 - ▶ client_IP client_ID user_ID time request status_code size
referer user-agent
- ▶ その他 カスタマイズ可能

client_IP: アクセス元の IP アドレス

client_ID: クライアントの識別子

user_ID: 認証ユーザ名

time: 時刻

request: リクエストの最初の行

status_code: レスポンスステータス

size: 送信バイト数 (ヘッダーは含まず) 0 バイトだと "-"

referer: リクエストのリンク元

user-agent: リクエスト元のブラウザの種類やバージョン

例 Combined Log Format:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] \  
"GET /apache_pb.gif HTTP/1.0" 200 2326 \  
"http://www.example.com/start.html" \  
"Mozilla/4.08 [en] (Win98; I ;Nav)"
```

mail log

受信、送信などのメール処理毎にログが取られる
例:

```
Oct 27 13:32:54 server3 sm-mta[24510]: m9R4WsBe024510:\
  from=<client@example.com>, size=2403, class=0, nrcpts=1 \
  msgid=<201012121547.oBCF1PX6032787@example.com>, \
  proto=ESMTP, daemon=MTA, relay=mail.example.co.jp [192.0.2.1] \
Oct 27 14:43:04 server3 sm-mta[24511]: m9R4WsBe024510: \
  to=<user@example.co.jp>, delay=01:10:10 xdelay=00:00:00, \
  mailer=local, pri=32599, dsn=2.0.0, stat=Sent
```

- ▶ 時刻
- ▶ ホスト名
- ▶ プロセスオーナー [プロセス番号]
- ▶ Queue ID: メールの内部 ID
- ▶ ...
- ▶ nrcpts: 受信者数
- ▶ relay: 次の送信先サーバ
- ▶ dsn: Delivery Status Notification, RFC3463
 - ▶ 2.X.X:Success, 4.X.X:Persistent Transient Failure,
5.X.X:Permanent Failure
- ▶ stat: Message Status
 - ▶ Sent, Deferred, Bounced, etc

DHCP server log

SYSLOG: メッセージの記録

```
Oct 28 15:04:32 server33 dhcpd: DHCPDISCOVER from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPOFFER on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPREQUEST for 192.168.2.101 \
  from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPACK on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
Oct 28 15:09:32 server33 dhcpd: DHCPREQUEST for 192.168.2.101 \
  from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:09:32 server33 dhcpd: DHCPACK on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
```

dhcpd.leases: 割り当てた IP アドレスの個別情報

```
lease 192.168.100.161 {
  starts 4 2010/12/09 23:13:39;
  ends 5 2010/12/10 00:13:39;
  tstp 5 2010/12/10 00:13:39;
  binding state free;
  hardware ethernet 5c:26:0a:17:06:00;
}
```

- ▶ UNIX 系 OS で任意のメッセージを通知したり保存する仕組み
 - ▶ もともとメールの処理記録保存用だったが広く使われるようになった
 - ▶ 他のサーバに転送も可能
 - ▶ ログのローテーション機能のサポート
- ▶ Windows Event Log

Web クローラ

クローラ (crawler) によるデータの収集

- ▶ クローラ: 自動でデータを収集するプログラム
- ▶ web クローラ: web ページを自動巡回する
 - ▶ 検索用データベースやインデックスの作成など
 - ▶ ページ内のリンクを辿って次のページへ
- ▶ いろいろツールが存在
 - ▶ 急激なアクセスは攻撃と思われるので注意

ログ解析手法

- ▶ 思いつくことを色々試す、グラフ化する
 - ▶ 手を動かしている内に分かる事、思いつく事が多い
- ▶ 処理スクリプトとコマンドラインツール (grep, sort, uniq, sed, awk, etc)
- ▶ 大量データを効率よく処理する工夫
- ▶ 繰り返し行う処理はできるだけ自動化する
 - ▶ いっぽうで自動化した処理を過信しないこと

大量データの扱い

- ▶ ナイーブにやると膨大なデータの読み込みや処理テーブルが必要
 - ▶ データ構造やアルゴリズムを勉強しておく役立つ
- ▶ 大量データを扱う工夫
 - ▶ 集計に不要な情報の削除
 - ▶ 時間的、空間的に集約
 - ▶ 必要に応じて分割処理
 - ▶ 必要に応じて分散並列処理
- ▶ 中間ファイルに変換する、分割処理する
- ▶ 処理に必要なメモリ量の見積り
 - ▶ データ構造を工夫する
 - ▶ 一度に処理するサイズ、次元を押える工夫
- ▶ 全体の処理時間の見積り
 - ▶ 小さいデータセットで試行
 - ▶ スケールするアルゴリズム
- ▶ メモリサイズと処理時間のトレードオフに配慮

正規表現 (regular expression)

正規表現

- ▶ 文字列パターンの表記法、文字列の検索や置換に利用
- ▶ もともとは形式言語理論において正規言語を表すための手段
- ▶ 文字列のパターンマッチのための記法として普及
 - ▶ grep, expr, awk, vi, lex, perl, ruby, ...

Ruby の正規表現

```
Regexp class
regular expression literal: /regexp/opt
=~ operator: subject =~ /regexp/
match() method: /regexp/.match(subject)
string class: string.match(/regexp/)
```

Ruby の正規表現クイックレファレンス

[abc] A single character: a, b or c
[^abc] Any single character but a, b, or c
[a-z] Any single character in the range a-z
[a-zA-Z] Any single character in the range a-z or A-Z
^ Start of line
\$ End of line
\A Start of string
\z End of string
. Any single character
\s Any whitespace character
\S Any non-whitespace character
\d Any digit
\D Any non-digit
\w Any word character (letter, number, underscore)
\W Any non-word character
\b Any word boundary character
(...) Capture everything enclosed
(a|b) a or b
a? Zero or one of a
a* Zero or more of a
a+ One or more of a
a{3} Exactly 3 of a
a{3,} 3 or more of a
a{3,6} Between 3 and 6 of a

Ruby の正規表現クイックレファレンス (つづき)

```
options:  
i case insensitive  
m make dot match newlines  
x ignore whitespace in regex  
o perform #{...} substitutions only once
```

最長マッチと最短マッチ (最短マッチの方が高速)

```
"*"や"+"は最長マッチ、"*?"や"+?"は最短マッチ  
/<.*>/ .match("<a><b><c>") # => "<a><b><c>"  
/<.*?>/ .match("<a><b><c>") # => "<a>"
```

前回の演習: 要約統計量の計算

- ▶ 平均
- ▶ 標準偏差
- ▶ 中央値

- ▶ 市民マラソンのデータを使う: 出典 P. K. Janert “Gnuplot in Action”

<http://web.sfc.keio.ac.jp/~kjc/classes/sfc2013s-measurement/marathon.txt>

前回の演習: 平均の計算

- ▶ 各行から、完走時間 (分) と人数を読み合計、最後に総数で割る

```
# regular expression to read minutes and count
re = /^(d+)\s+(d+)/
```

```
sum = 0 # sum of data
n = 0 # the number of data
ARGF.each_line do |line|
  if re.match(line)
    min = $1.to_i
    cnt = $2.to_i
    sum += min * cnt
    n += cnt
  end
end
```

```
mean = Float(sum) / n
```

```
printf "n:%d mean:%.1f\n", n, mean
```

```
% ruby mean.rb marathon.txt
n:2355 mean:171.3
```

前回の演習: 標準偏差の計算

▶ アルゴリズム: $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$

```
# regular expression to read minutes and count
re = /^(d+)\s+(d+)/

data = Array.new
sum = 0 # sum of data
n = 0 # the number of data
ARGF.each_line do |line|
  if re.match(line)
    min = $1.to_i
    cnt = $2.to_i
    sum += min * cnt
    n += cnt
    for i in 1 .. cnt
      data.push min
    end
  end
end

mean = Float(sum) / n
sqsum = 0.0
data.each do |i|
  sqsum += (i - mean)**2
end

var = sqsum / n
stddev = Math.sqrt(var)
printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

前回の演習: 標準偏差の計算の改良

▶ アルゴリズムを改良: $\sigma^2 = \frac{1}{n} \sum_{i=1}^n x_i^2 - \bar{x}^2$

```
# regular expression to read minutes and count
re = /^(d+)\s+(d+)/

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

mean = Float(sum) / n
var = Float(sqsum) / n - mean**2
stddev = Math.sqrt(var)

printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev

% ruby stddev2.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```


前回の演習: 中央値の計算

- ▶ 各走者のタイムを配列に入れソート、中央値を取り出す

```
# regular expression to read minutes and count  
re = /^(d+)\s+(d+)/
```

```
data = Array.new
```

```
ARGF.each_line do |line|
```

```
  if re.match(line)
```

```
    min = $1.to_i
```

```
    cnt = $2.to_i
```

```
    for i in 1 .. cnt
```

```
      data.push min
```

```
    end
```

```
  end
```

```
end
```

```
data.sort! # just in case data is not sorted
```

```
n = data.length # number of array elements
```

```
r = n / 2 # when n is odd, n/2 is rounded down
```

```
if n % 2 != 0
```

```
  median = data[r]
```

```
else
```

```
  median = (data[r - 1] + data[r])/2
```

```
end
```

```
printf "r:%d median:%d\n", r, median
```

```
% ruby median.rb marathon.txt
```

```
r:1177 median:176
```

前回の演習: gnuplot

- ▶ gnuplot を使って簡単なグラフを書く

前回の演習: ヒストグラム

▶ 市民マラソンの完走タイムの分布

```
plot "marathon.txt" using 1:2 with boxes
```

グラフを見やすくする (右側)

```
set boxwidth 1
```

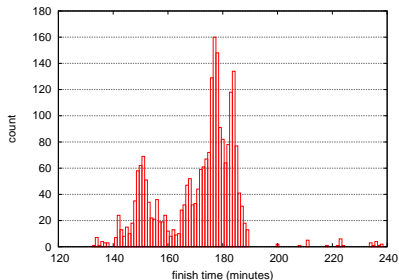
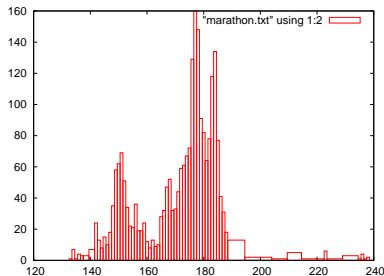
```
set xlabel "finish time (minutes)"
```

```
set ylabel "count"
```

```
set yrange [0:180]
```

```
set grid y
```

```
plot "marathon.txt" using 1:2 with boxes notitle
```



前回の演習: 完走時間の CDF の作成

元データ:

```
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
...
```

累積度数を追加:

```
# Minutes Count CumulativeCount
133 1 1
134 7 8
135 1 9
136 4 13
137 3 16
138 3 19
141 7 26
142 24 50
...
```

前回の演習: CDF (2)

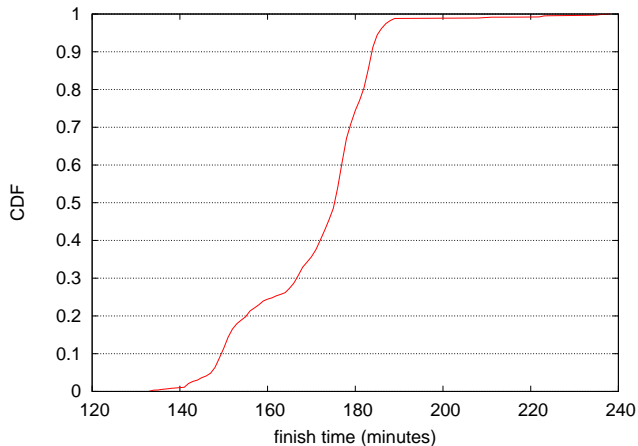
ruby code:

```
re = /^(\d+)\s+(\d+)/
cum = 0
ARGF.each_line do |line|
  begin
    if re.match(line)
      # matched
      time, cnt = $~.captures
      cum += cnt.to_i
      puts "#{time}\t#{cnt}\t#{cum}"
    end
  end
end
```

gnuplot command:

```
set xlabel "finish time (minutes)"
set ylabel "CDF"
set grid y
plot "marathon-cdf.txt" using 1:($3 / 2355) with lines notitle
```

前回の演習: 市民マラソンの完走時間 CDF



今日の演習: Web アクセスログ サンプルデータ

- ▶ apache log (combined log format)
- ▶ 米国 Splunk 社のチュートリアル用疑似データ (24 時間分)
- ▶ 約 500KB(zip 圧縮)、解凍後は約 9MB

http://www.iijlab.net/~kjc/classes/sfc2013s-measurement/access_combined.zip

サンプルアクセスログ

```
10.32.1.43 - - [10/Apr/2013:00:07:00] "GET /flower_store/product.screen?product_id=FL-DLH-02 HTTP/1.1" \
200 10901 "http://mystore.splunk.com/flower_store/category.screen?category_id=GIFTS&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
4361 3217

10.32.1.43 - - [10/Apr/2013:00:07:00] "GET /flower_store/category.screen?category_id=GIFTS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-12&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
1685 2466

177.23.21.50 - - [10/Apr/2013:00:07:45] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-18&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
2526 482

177.23.21.50 - - [10/Apr/2013:00:07:45] "GET /flower_store/category.screen?category_id=PLANTS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-18&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
2623 2099

233.77.49.54 - - [10/Apr/2013:00:07:58] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-11&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
662 2486

10.2.91.32 - - [10/Apr/2013:00:08:00] "GET /flower_store/category.screen?category_id=BALLOONS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-11&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
431 4899

10.2.91.32 - - [10/Apr/2013:00:08:00] "GET /flower_store/category.screen?category_id=BALLOONS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-11&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
3276 322

10.32.1.37 - - [10/Apr/2013:00:08:26] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-17&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
1494 4680

192.168.11.38 - - [10/Apr/2013:00:08:49] "GET /flower_store/category.screen?category_id=FLOWERS HTTP/1.1" \
200 10567 "http://mystore.splunk.com/flower_store/cart.do?action=purchase&itemId=EST-13&JSESSIONID=SD7SL1FFS"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.10) Gecko/20070223 CentOS/1.5.0.10-0.1.el4.centos Firefox/3.5.1"
48 / 53
```


今日の演習: リクエスト数の時系列プロット

- ▶ サンプル Web アクセスログを使う
- ▶ リクエスト数と転送バイト数を 5 分間隔で抽出
- ▶ 結果をプロット

```
% ruby parse_accesslog.rb access_combined.log > access-5min.txt
% more access-5min.txt
2013-04-10T00:05 11 116664
2013-04-10T00:10 12 137116
2013-04-10T00:15 15 168736
2013-04-10T00:20 9 102072
2013-04-10T00:25 10 105680
2013-04-10T00:30 13 139694
2013-04-10T00:35 12 129011
2013-04-10T00:40 9 95103
...
% gnuplot
gnuplot> load 'access.plt'
```

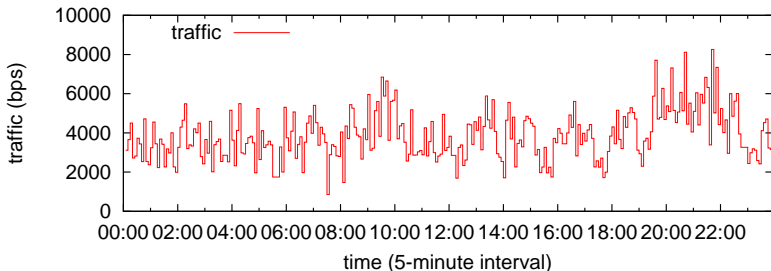
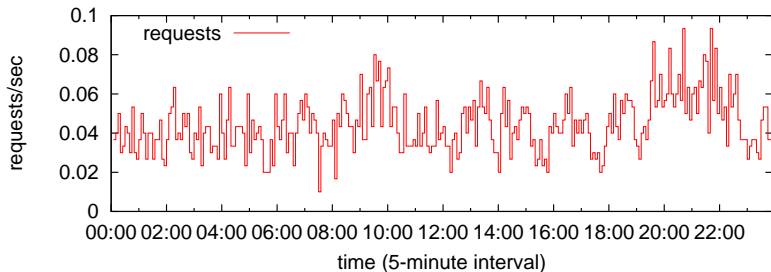
演習: 5 分間隔でリクエスト数と転送数を抽出

```
#!/usr/bin/env ruby
require 'date'

# regular expression for apache common log format
# host ident user time request status bytes
re = /^(S+) (S+) (S+) \[(.*?)\] "(.*?)" (\d+) (\d+|-)/

timebins = 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 = $~.captures
    # ignore if the request is not "GET"
    next unless request.match(/GET\s.*/i)
    # ignore if the status is not success (2xx)
    next unless status.match(/2\d{2}/i)
    parsed += 1
    # parse timestamp
    ts = DateTime.strptime(time, '%d/%b/%Y:%H:%M:%S')
    # create the corresponding key for 5-minutes timebins
    rounded = sprintf("%02d", ts.min.to_i / 5 * 5)
    key = ts.strftime("%Y-%m-%dT%H:#{rounded}")
    # count by request and byte
    timebins[key] = [timebins[key][0] + 1, timebins[key][1] + bytes.to_i]
  else
    # match failed
    $stderr.puts("match failed at line #{count}: #{line.dump}")
  end
end
timebins.sort.each do |key, value|
  puts "#{key} #{value[0]} #{value[1]}"
end
$stderr.puts "parsed:#{parsed} ignored:#{count - parsed}"
```

plot graphs of request counts and transferred bytes



gnuplot script

- ▶ multiplot 機能で2つのプロットをまとめる

```
set terminal postscript eps color solid 18
set xlabel "time (5-minute interval)"
set xdata time
set format x "%H:%M"
set timefmt "%Y-%m-%dT%H:%M"
set xrange ['2013-04-10T00:00':'2013-04-10T23:55']
set key left top

set multiplot layout 2,1

set yrange [0:0.1]
set ylabel "requests/sec"
plot "access-5min.txt" using 1:($2/300) title 'requests' with steps

set yrange [0:10000]
set ylabel "traffic (bps)"
plot "access-5min.txt" using 1:($3*8/300) title 'traffic' with steps

unset multiplot
```

まとめ

第3回 データの収集と記録

- ▶ ネットワーク管理ツール
- ▶ データフォーマット
- ▶ ログ解析手法
- ▶ 演習: ログデータと正規表現

次回予定

第 4 回 分布と信頼区間 (5/1)

- ▶ 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

参考文献

- [1] Ruby official site. <http://www.ruby-lang.org/>
- [2] gnuplot official site. <http://gnuplot.info/>
- [3] Mark Crovella and Balachander Krishnamurthy. *Internet measurement: infrastructure, traffic, and applications*. Wiley, 2006.
- [4] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.
- [5] Raj Jain. *The art of computer systems performance analysis*. Wiley, 1991.
- [6] Toby Segaran. (當山仁健 鴨澤眞夫 訳). 集合知プログラミング. オライリージャパン. 2008.
- [7] Chris Sanders. (高橋基信 宮本久仁男 監訳 岡真由美 訳). 実践パケット解析 第2版 — *Wireshark* を使ったトラブルシューティング. オライリージャパン. 2012.
- [8] あきみち、空閑洋平. インターネットのカタチ. オーム社. 2011.
- [9] 井上洋, 野澤昌弘. 例題で学ぶ統計的方法. 創成社, 2010.
- [10] 平岡和幸, 堀玄. プログラミングのための確率統計. オーム社, 2009.