# Internet Measurement and Data Analysis (4)

Kenjiro Cho

2011-10-19

## review of previous class

items left off from previous class

- how to make good graphs
- exercise: graph plotting by gnuplot

Data recording and log analysis

- data format
- log analysis methods
- exercise: log data and regular expression

## today's topics

Class 4 Measuring the speed of the Internet

- bandwidth measurement
- inferring available bandwidth
- mean, standard deviation
- linear regression
- exercise: mean, standard deviation, linear regression
- **assignment 1**

## Measuring the speed of the Internet

What is speed?

- distance traveled per unit time

$$v = \frac{\Delta P}{\Delta t}$$

speed of network

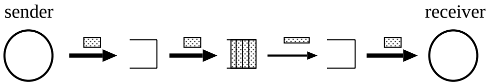
- the speed of light in vacuum, usually denoted by c: $3.0x10^8 m/s$
- the speed of light in an optical fiber: $2.1x10^8 m/s$

data transfer rate: efficiency rather than speed

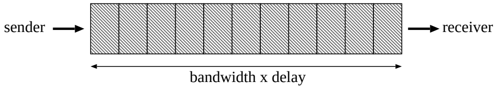
- bit rate: bits per second
- related terms (often confused with bit rate)
  - bandwidth capacity, bandwidth
  - throughput

## throughput measurement

- throughput measurement sites: provide services to measure throughput
  - assuming that the access network is the bottleneck
    - other possibilities: ISP boundaries, etc
  - interference of cross-traffic
- throughput measurement tools
  - Iperf, netperf, ixChariot, etc
- congestion: quality degradation by traffic concentration
  - increased delay due to queued packets at intermediate routers

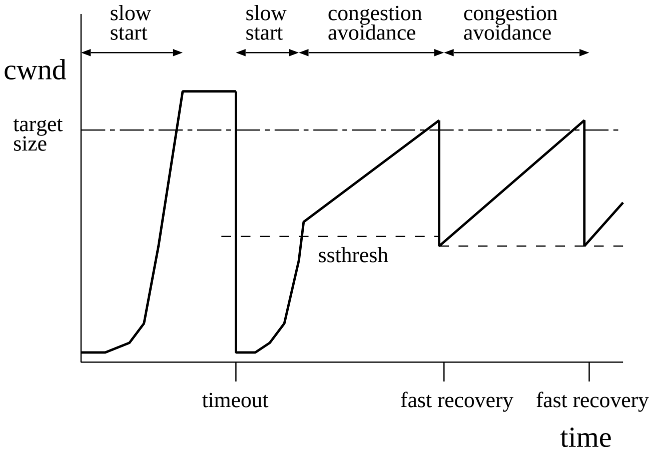


- TCP is often better than UDP
  - UDP overflows all buffers but TCP adapts to available bandwidth
  - TCP adapts itself to the available bandwidth



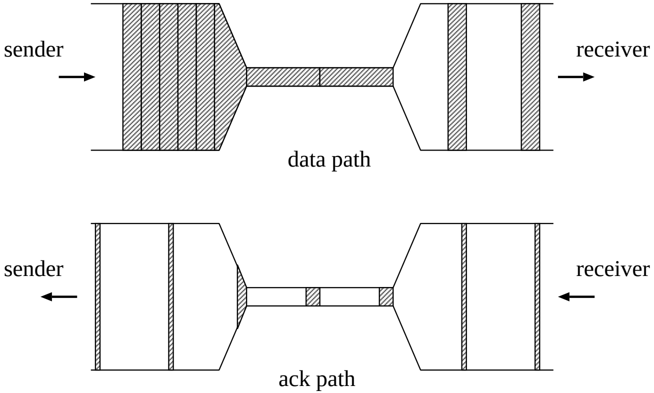
## TCP congestion control

- congestion window controls volume of packets in flight
  - slow start/congestion avoidance
  - retransmit timeout
  - fast retransmit/fast recovery



## TCP self-clocking

- reception of ack triggers next packet transmission
- adapts to bottleneck bandwidth

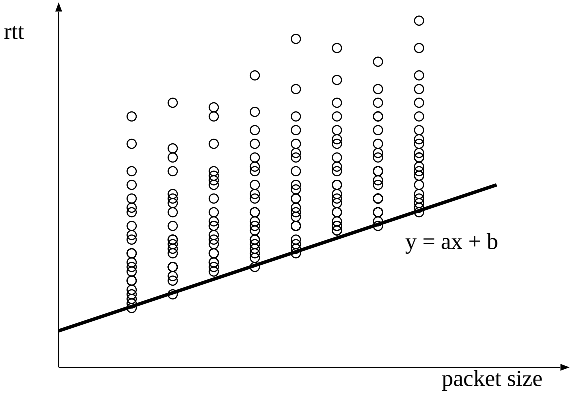


## bandwidth estimation

- infer the available bandwidth without filling the pipe with packets
- pathchar algorithm (there are other similar tools)
    - per hop measurement by TTL (as traceroute does)
    - repeated measurement with varying packet size
    - pick up minimum round-trip time for each packet size
    - linear regression to obtain propagation delay and bandwidth
- limitations
    - many measurements required
    - accumulation of errors
        - especially for narrow link behind fat link
    - only one-way

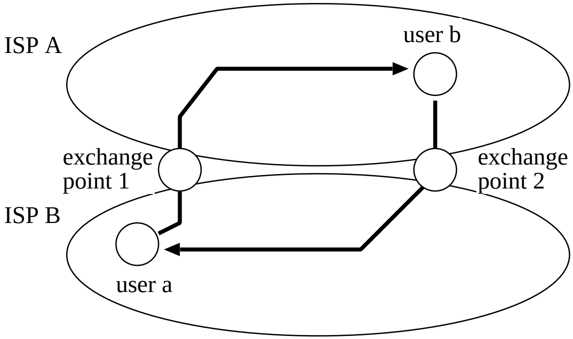
## pathchar algorithm

- linear regression to infer delay and capacity
  - $y = ax + b$
  - a: RTT-delta/packetsize-delta (bandwidth)
  - b: delay for packetsize 0 (propagation delay)



## asymmetric routes

- asymmetric route between 2 ISPs are common
  - packets forwarded via the nearest exchange point to other ISP
  - hot potato routing



## summary statistics

numbers that summarize properties of data

- measures of location: mean, median, mode
- measures of spread: range, variance, standard deviation

## measures of location

- mean: average, sensitive to outliers

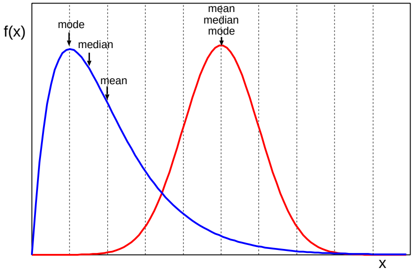
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- median: middle value (50th-percentile)

$$x_{median} = \begin{cases} x_{r+1} & \text{when } m \text{ is odd, } m = 2r+1 \\ (x_r + x_{r+1})/2 & \text{when } m \text{ is even, } m = 2r \end{cases}$$

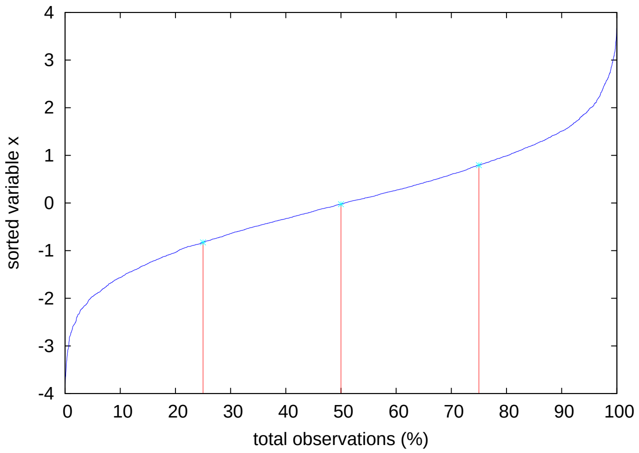
- mode: value with highest frequency

these are same if measurements have symmetric distribution



## percentiles

- *p*th-percentile:
  - $p\%$ of the observed values are less than $x_p$ in variable $x_i$
  - median $=$ 50th-percentile



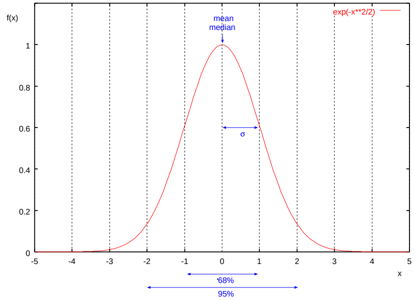
## measures of spread

common measures of the spread of a data set

- range: difference between the max and min
- variance:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- standard deviation: $\sigma$
  - most common measure of statistical dispersion
  - can be directly compared with mean
- for a normal distribution, 68% fall into $(mean \pm stddev)$ 95% fall into $(mean \pm 2stddev)$



## correlation

- covariance:

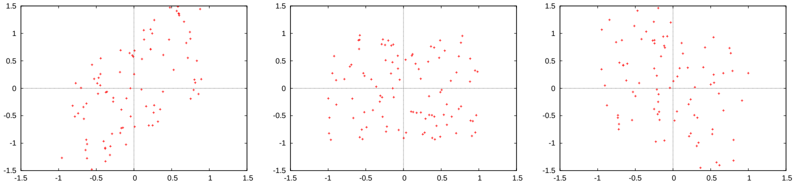
$$\sigma_{xy}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

- correlation coefficient:

$$\rho_{xy} = \frac{\sigma_{xy}^2}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

## scatter plots

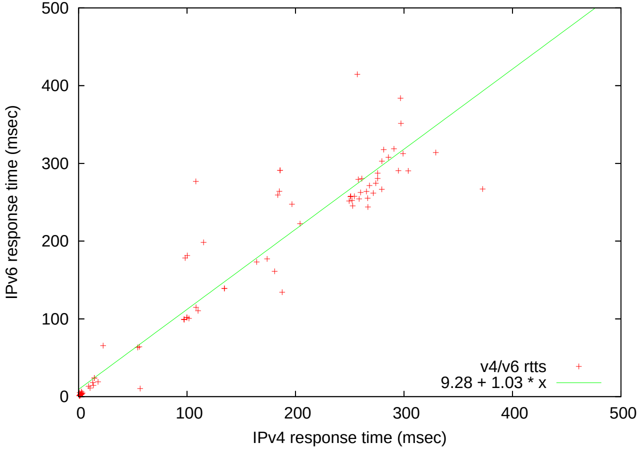
- explores relationships between 2 variables
  - X-axis: variable X
  - Y-axis: corresponding value of variable Y
- you can identify
  - whether variables X and Y related
    - no relation, positive correlation, negative correlation
  - whether the variation in Y changes depending on X
  - outliers
- examples: positive correlation 0.7 (left), no correlation 0.0 (middle), negative correlation -0.5 (right)



examples: positive correlation 0.7 (left), no correlation 0.0 (middle), negative correlation -0.5 (right)

## linear regression

- fitting a straight line to data
  - least square method: minimize the sum of squared errors



## least square method

a linear function minimizing squared errors

$$f(x) = b_0 + b_1 x$$

2 regression parameters can be computed by

$$b_1 = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n(\bar{x})^2}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$

where

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\sum xy = \sum_{i=1}^{n} x_i y_i \qquad \sum x^2 = \sum_{i=1}^{n} (x_i)^2$$

## a derivation of the expressions for regression parameters

The error in the $i$th observation: $e_i = y_i - (b_0 + b_1 x_i)$
For a sample of $n$ observations, the mean error is

$$\bar{e} = \frac{1}{n}\sum_i e_i = \frac{1}{n}\sum_i (y_i - (b_0 + b_1 x_i)) = \bar{y} - b_0 - b_1\bar{x}$$

Setting the mean error to 0, we obtain: $b_0 = \bar{y} - b_1\bar{x}$
Substituting $b_0$ in the error expression: $e_i = y_i - \bar{y} + b_1\bar{x} - b_1 x_i = (y_i - \bar{y}) - b_1(x_i - \bar{x})$
The sum of squared errors, $SSE$, is

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}[(y_i - \bar{y})^2 - 2b_1(y_i - \bar{y})(x_i - \bar{x}) + b_1^2(x_i - \bar{x})^2]$$

$$\begin{aligned}\frac{SSE}{n} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 - 2b_1\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x}) + b_1^2\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \\ &= \sigma_y^2 - 2b_1\sigma_{xy}^2 + b_1^2\sigma_x^2\end{aligned}$$

The value of $b_1$, which gives the minimum SSE, can be obtained by differentiating this equation with respect to $b_1$ and equating the result to 0:

$$\frac{1}{n}\frac{d(SSE)}{db_1} = -2\sigma_{xy}^2 + 2b_1\sigma_x^2 = 0$$

That is: $b_1 = \frac{\sigma_{xy}^2}{\sigma_x^2} = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n(\bar{x})^2}$

## previous exercise: plotting request counts over time

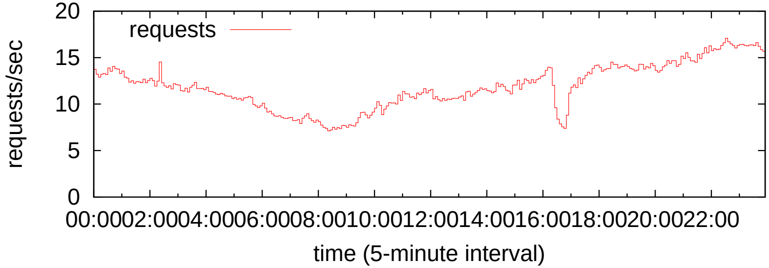
- use the sample data
- extract request counts and transferred bytes with 5 minutes bins
- plot the results

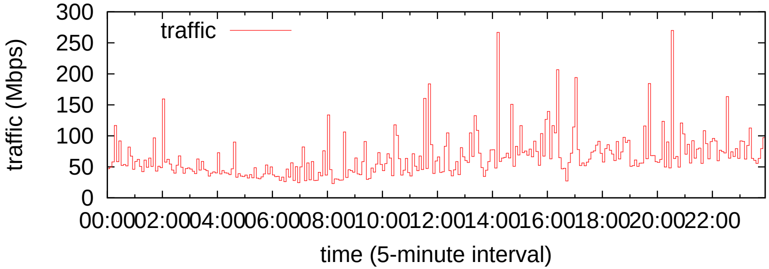
```
% ruby parse_accesslog.rb sample_access_log > access-5min.txt
% more access-5min.txt
2010-07-18T16:55 1 600572285
...
2010-07-18T23:55 463 2128020418
2010-07-19T00:00 4123 1766135158
2010-07-19T00:05 3963 1857342919
2010-07-19T00:10 3871 2171231118
2010-07-19T00:15 3965 4378143224
...
% gnuplot
gnuplot> load 'access.plt'
```

## previous exercise: extract request counts and transferred bytes with 5 minutes bins

```
#!/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 status is not success (2xx)
    next unless /2\d{2}/.match(status)
    parsed += 1
    # parse timestamp
    ts = DateTime.strptime(time, '%d/%b/%Y:%H:%M:%S %z')
    # 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}"
```

## previous exercise: plot graphs of request counts and transferred bytes





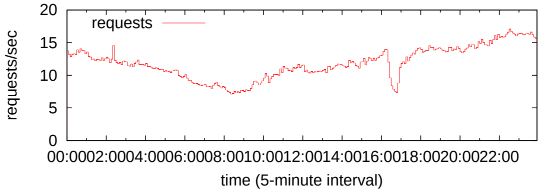
## previous exercise: gnuplot script

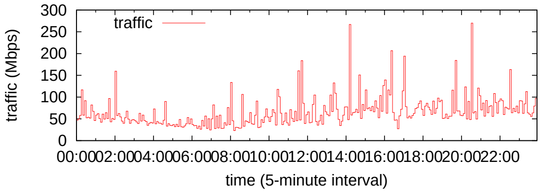
- put 2 graphs together using multiplot

```
set xlabel "time (5-minute interval)"
set xdata time
set format x "%H:%M"
set timefmt "%Y-%m-%dT%H:%M"
set xrange ['2010-07-19T00:00':'2010-07-19T23:55']
set key left top

set multiplot layout 2,1

set yrange [0:20]
set ylabel "requests/sec"
plot "access-5min.txt" using 1:($2/300) title 'requests' with steps

set yrange [0:300]
set ylabel "traffic (Mbps)"
plot "access-5min.txt" using 1:($3*8/300/1000000) title 'traffic' with steps

unset multiplot
```

## exercise: mean, standard deviation, linear regression

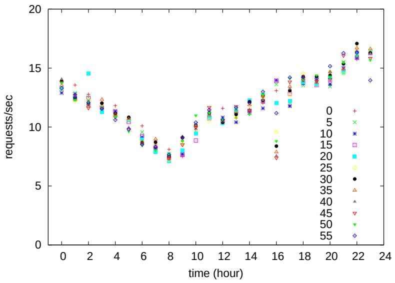
- use the 5-minute bin output from the previous class
  - remove data not for 7/19
- focus on the request counts (leave transferred bytes for the assignment)
- use 12 5-minute bins for an hour as 12 samples per hour

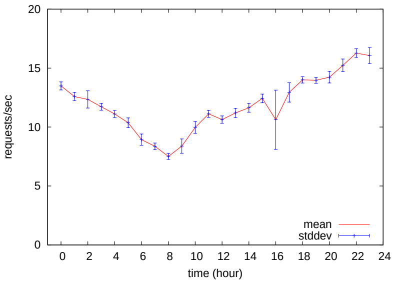




## graphs for request counts

- graph 1: 12 samples per hour, for 24 hours
- graph 2: mean and standard deviation for 24 hours





## compute mean and standard deviation

```
# extract a variable from each line
re = /^\S+\s+(\d+)\s+\d+/

# create an array for data
data = Array.new
ARGF.each_line do |line|
  if re.match(line)
    data.push $1.to_i
  end
end

# compute mean
sum = 0
data.each {|v| sum += v}
mean = Float(sum) / data.length

# compute standard deviation
sqsum = 0
data.each {|v| sqsum += (v - mean)**2}
var = Float(sqsum) /data.length
stddev = Math.sqrt(var)

puts "mean=#{mean} stddev=#{stddev}"
```

## creating data table for request counts

create an hourly data table for plotting

- row: hourly data (0 .. 23)
- column: hour samples(00 05 10 ... 55) mean stddev

```
#hour  00     05     10      ...     55     mean      stddev
 0    4123   3963   3871     ...    3987   4046.8    102.3
 1    4068   3871   3838     ...    3760   3774.9    106.2
 2    3833   3755   3580     ...    3628   3703.6    219.0
 3    3614   3433   3418     ...    3462   3515.5     86.2
...
22    4724   4790   4757     ...    4893   4882.2    113.4
23    4922   4932   4889     ...    4188   4818.9    203.8
```

## table creation code

```
re = /^(\d{4}-\d{2}-\d{2})T(\d{2}):(\d{2})\s+(\d+)\s+(\d+)/

hourly = Array.new(24){ Array.new(12) } # hourly[hour][min]
ARGF.each_line do |line|
  if re.match(line)
    day, hour, min, requests, bytes = $~.captures
    hourly[hour.to_i][min.to_i / 5] = requests.to_i
  end
end
hourly.each_index do |h|
  printf "%2d ", h
  sum = n = 0
  hourly[h].each_index do |m|
    printf "%6d ", hourly[h][m]
    sum += hourly[h][m]
    n += 1
  end
  mean = Float(sum) / n
  printf "%8.1f ", mean
  var = 0
  hourly[h].each_index do |m|
    var += (hourly[h][m] - mean) ** 2
  end
  var = var / n
  printf "%8.1f\n", Math.sqrt(var)
end
```

## plot commands

### For time-slots data

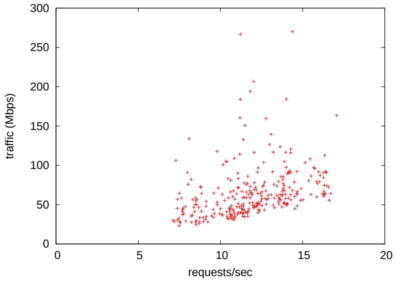
```
set xlabel "time (hour)"
set xrange [-1:24]
set yrange [0:20]
set xtic 2
set key bottom right
set ylabel "requests/sec"
plot "request-table.txt" using 1:($2/300) title '0' with points, \
"request-table.txt" using 1:($3/300) title '5' with points, \
"request-table.txt" using 1:($4/300) title '10' with points, \
"request-table.txt" using 1:($5/300) title '15' with points, \
"request-table.txt" using 1:($6/300) title '20' with points, \
"request-table.txt" using 1:($7/300) title '25' with points, \
"request-table.txt" using 1:($8/300) title '30' with points, \
"request-table.txt" using 1:($9/300) title '35' with points, \
"request-table.txt" using 1:($10/300) title '40' with points, \
"request-table.txt" using 1:($11/300) title '45' with points, \
"request-table.txt" using 1:($12/300) title '50' with points, \
"request-table.txt" using 1:($13/300) title '55' with points
```

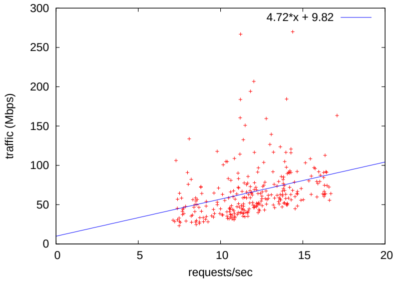
### For mean and stddev:

```
set xlabel "time (hour)"
set xrange [-1:24]
set yrange [0:20]
set xtic 2
set key bottom right
set ylabel "requests/sec"
plot "request-table.txt" using 1:($14/300) title 'mean' with lines, \
"request-table.txt" using 1:($14/300):($15/300) title 'stddev' with errorbars lt 3
```

# linear regression

- relationship of request count and traffic of 5-min bins
- linear regression by the least square method





## linear regression code

```
# extract 2 variables from each line
re = /^\S+\s+(\d+)\s+(\d+)/

# compute (y = b0 + b1*x) by the least square method
sum_x = sum_y = sum_xx = sum_xy = 0.0
n = 0
ARGF.each_line do |line|
    if re.match(line)
      x = $1.to_f / 300  # req/5-min to req/sec
      y = $2.to_f * 8 / 300 / 1000000 # bytes/5min to Mbps

      sum_x += x
      sum_y += y
      sum_xx += x * x
      sum_xy += x * y
      n += 1
    end
end

mean_x = Float(sum_x) / n
mean_y = Float(sum_y) / n
b1 = (sum_xy - n * mean_x * mean_y) / (sum_xx - n * mean_x * mean_x)
b0 = mean_y - b1 * mean_x

puts "b0=#{b0} b1=#{b1}"
```

## assignment 1

- assignment: compute mean and standard deviation of traffic, plot the results
  - similar to today's exercise but for traffic (not for request counts)
- to understand programming of statistical procedures and graph plotting
- data: use the 5-min bin outputs from the previous exercise
- items to submit
  1. traffic data table
  2. graph 1: a plot of 12 samples per hour for 24 hours
  3. graph 2: a plot of mean and standard deviation of traffic for 24 hours
- the results should look similar to ones for the request counts
  - you need to adjust the number of digits for table outputs
  - use "Mbps" for the unit of traffic in the graphs
- submission format: a single PDF file including 1 table, 2 graphs, and comments (if any)
- submission method: upload the PDF file through SFC-SFS
- submission due: 2011-11-5

## traffic data table

the table should look like:

```
#hour        00          05   ...          mean        stddev
 0   1766135158  1857342919   ...  2420355075.6   777770181.4
 1   2202831446  2322940598   ...  2120850506.4   521760120.1
 2   5980871926  2158091698   ...  2261318711.4  1161290997.4
 3   1741001140  1609648229   ...  1692879169.6   286988721.2
...
```

## summary

Class 4 Measuring the speed of the Internet

- bandwidth measurement
- inferring available bandwidth
- mean, standard deviation
- linear regression
- exercise: mean, standard deviation, linear regression
- **assignment 1**

## next class

Class 5 Measuring the structure of the Internet (10/26)

- Internet architecture
- network layers
- topologies
- graph theory
- exercise: topology analysis