インターネット計測とデータ解析 第8回

長 健二朗

2012年6月1日

前回のおさらい

多変量解析

- データセンシング
- 線形回帰
- 主成分分析
- 演習: 線形回帰

今日のテーマ

時系列データ

- インターネットと時刻
- ネットワークタイムプロトコル
- トラフィック計測
- 時系列解析
- 演習: 時系列解析
- **課題 2**

計測と時間

- 絶対時刻
  - 協定世界時 UTC (Universal Coordinated Time)
    - セシウム原子時計をもとに取り決められている標準時
- 相対時刻
  - 時刻の差分
- 時刻調整
  - 時計の時刻は前後に補正される
  - NTP では 128ms 未満の誤差は一度に、それ以上だと徐々に修正

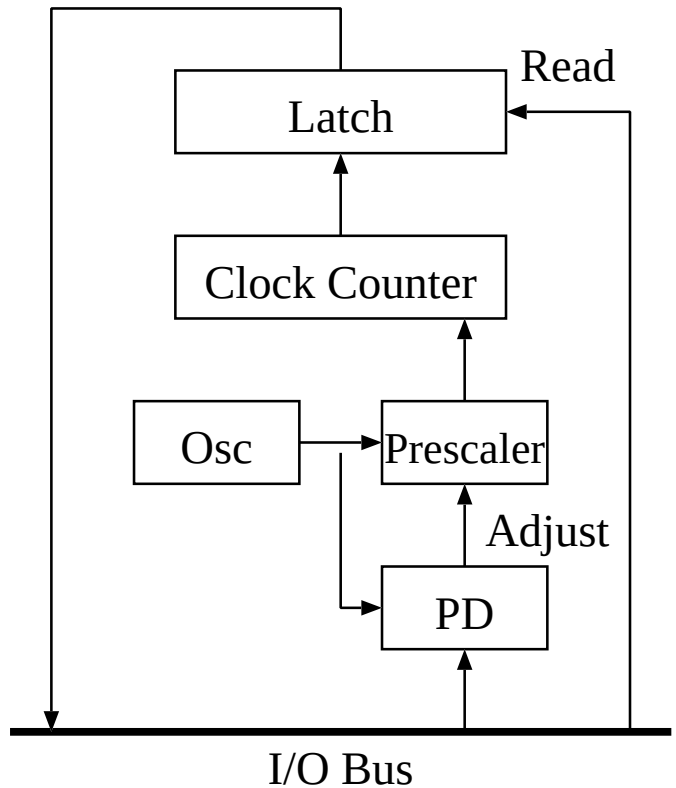
クロックの誤差

- クロックの誤差
  - 同期
    - 2 つのクロックの差
  - 正確さ
    - UTC からのずれ
  - 解像度
    - クロックの精度
  - スキュー
    - 時間とともに同期や正確さがずれる
- 時間粒度
  - PC クロック: 0.1-1sec/日ぐらいずれる
  - NTP: 10-100ms の正確さにクロックを同期
  - tcpdump などのタイムスタンプ:
    - 100usec-100msec (通常 $< 1msec$ だが保証なし)

PC のクロック

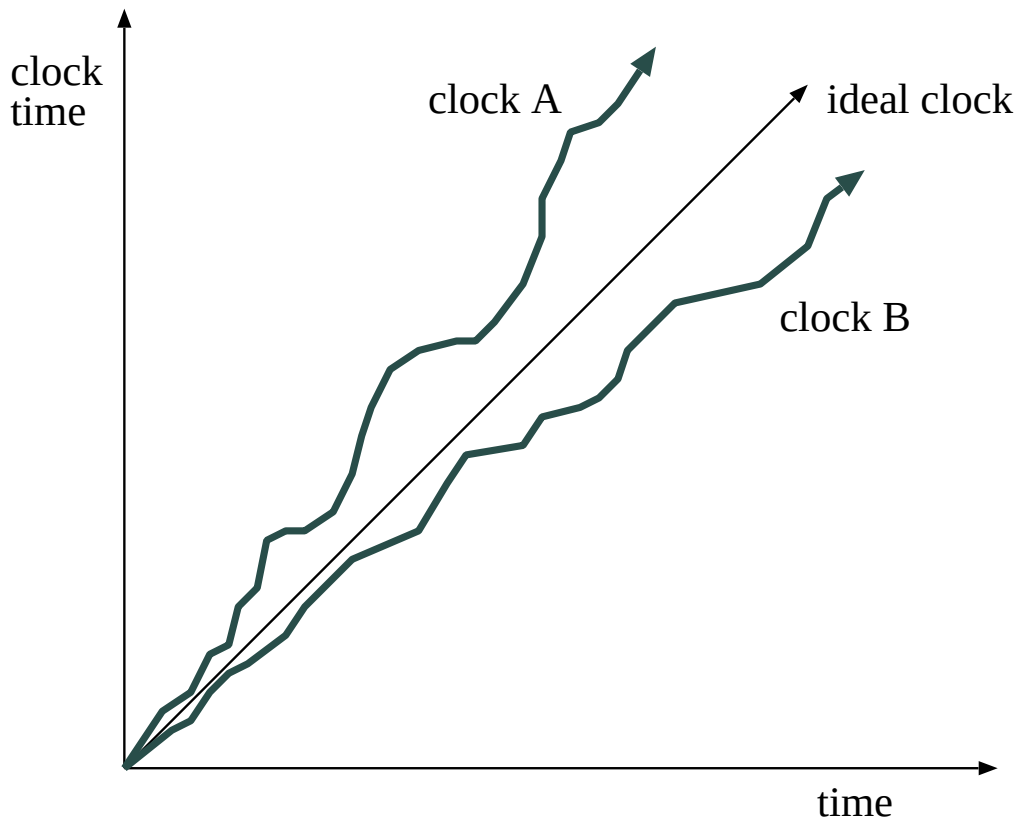
i8254 プログラムインターバルタイマー

- 16-bit フリーラニング ダウンカウンター
  - 1,193,182 Hz の水晶発振器を基にしている
  - カウンターがゼロになると割り込み信号を上げてカンターレジスタ値をリロード



クロックドリフト

- 水晶発振器のドリフト
  - ハードウエア仕様の許容誤差: 10^{-5}
    - 0.86 sec/day は許容誤差内
  - ドリフトは温度に大きく影響される



その他の PC クロック

- Pentium TSC (Time Stamp Counter)
  - CPU クロックで駆動される CPU 内蔵フリーラニングカウンター
  - 可変クロックやマルチ CPU で問題
- ACPI (Advanced Counfiguration and Power Interface)
  - パワー管理機能が提供するフリーラニングカウンター
- Local APIC (Advanced Programmable Interrupt Controller)
  - 各プロセッサに内蔵される割り込み機能付きタイマー
- HPET (High Precision Event Timer)
  - IA-PC の新しいタイマー仕様
  - 2005 年頃からチップセットに組み込み
- 外部クロック
  - GPS、CDMA など時刻情報を含む
    - インターフィスにより読み込みオーバーヘッド

OS 時刻管理

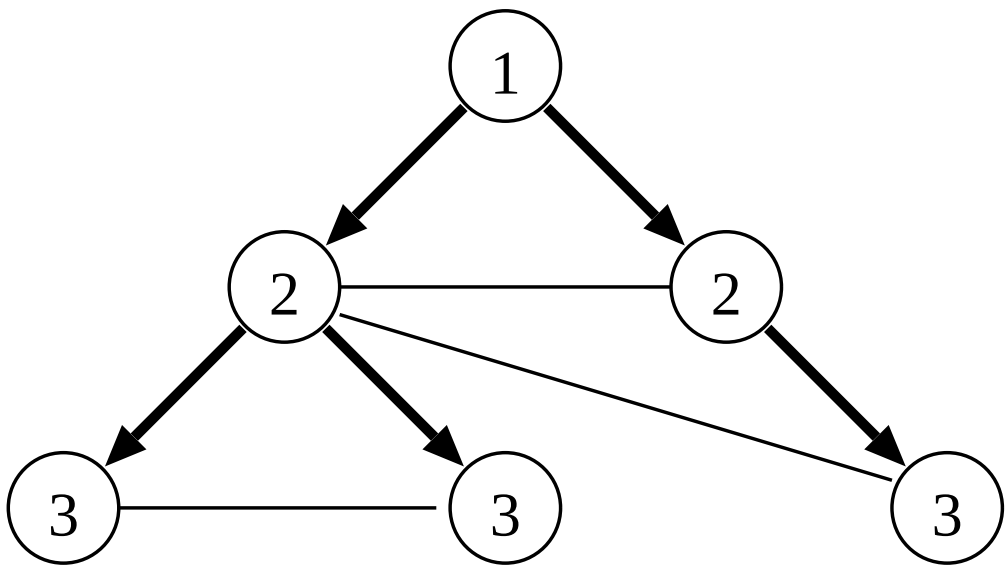
- OS はソフトウエアにより時刻を管理
  - 起動時にカレンダーチップから時刻を得る
  - ハードウエアクロック割り込み毎に時刻をアップデート
- 従来の UNIX では、デフォルトで 10ms ごとにクロック割り込みが発生するようにクロックカウンターを設定

UNIX gettimeofday

- 古い OS ではクロック割り込みの粒度しかなかった
- いまどきの OS ではより高精度の時刻を得られる
  - クロックカウンター値を読み出してソフトウエアクロックを補間
    - i8254 の解像度: 838ns (1 / 1193182)
  - OS 内部処理時間
    - i8254 レジスタアクセス: 1-10usec
    - struct timeval への変換: 1-100usec
  - ユーザ空間から OS 内部へのアクセス
    - システムコール オーバーヘッド: 10-500usec
    - プロセススケジューリングの影響: 1-100msec or more
- タイマーイベント ソフトウエア処理時間 (e.g., setitimer):
  - ソフトウエアタイマー割り込みから処理 (10msec by default)
  - プロセススケジューリングの影響を受ける

NTP (Network Time Protocol)

- インターネット上の複数サーバー間で時刻同期
  - プライマリサーバ: 直接 UTC ソースに繋がる
  - セカンダリサーバ: プライマリに同期
  - 3 段目以降のサーバ: セカンダリ以降に同期
- スケーラビリティ
  - 20-30 プライマリ、 2000 セカンダリを $< 30ms$ に同期
- さまざまな機能
  - 耐故障性、認証などをサポート



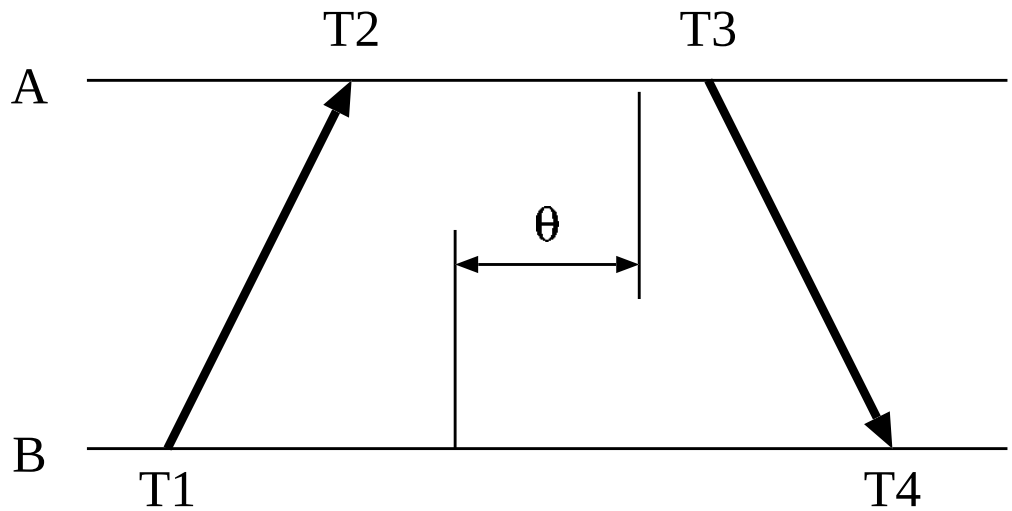
NTP 同期モード

- マルチキャスト (LAN 向け)
  - 定期的に時刻情報をマルチキャストで広報
- リモートプロシージャコール
  - クライアントが (複数) サーバーに時刻情報を要求
- ピアプロトコル
  - 複数のピアの間で同期

NTP ピアプロトコル

相手とのオフセットと通信遅延を計測

- $a = T2 - T1 \qquad b = T3 - T4$
- clock offset: $\theta = (a + b)/2$ (RTT が対称だと仮定)
- roundtrip delay: $\delta = a - b$

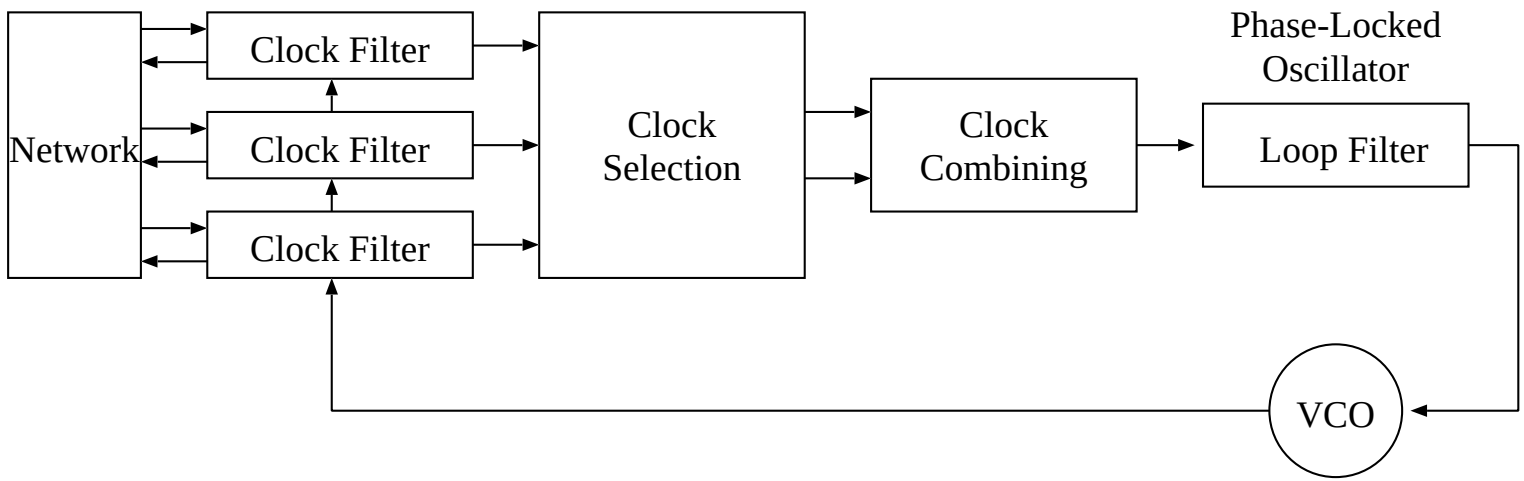


全てのメッセージに以下を含める

- T3: send time (current time)
- T2: receive time
- T1: send time in received message

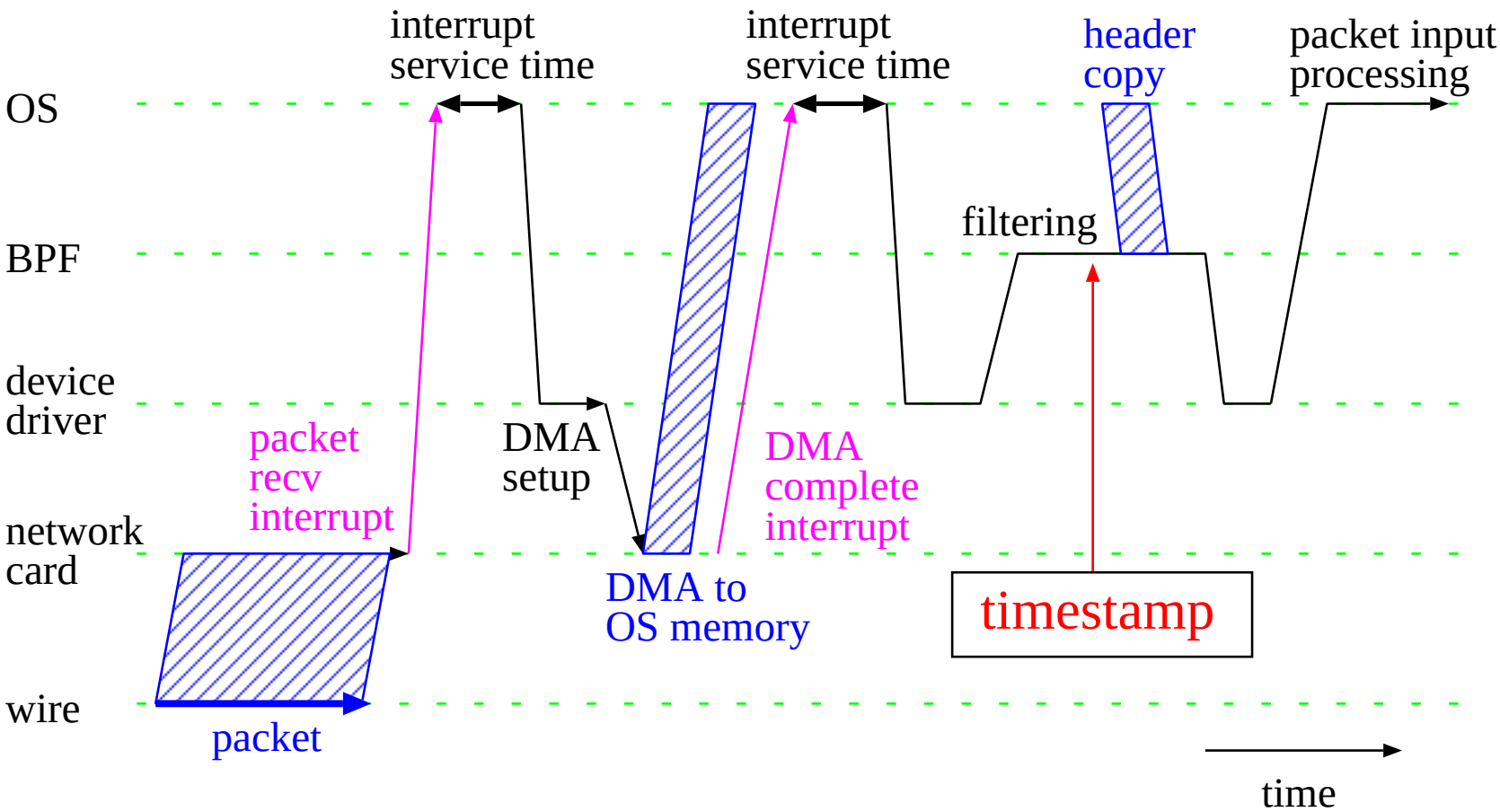
NTP システムモデル

- クロックフィルタ
  - 各ピアからの時刻情報を時系列に平滑化
- クロック選択
  - 互いに一致しているクロックを抜き出す
  - インターセクションアルゴリズム: 外れ値の除外
  - クラスタリング: 最善値の選択
- クロック統合
  - 推定値を 1 個に統合



BSD UNIX の BPF タイムスタンプ

- 通常、割り込み処理 2 回の後タイムスタンプ
  - recv packet, DMA complete



ネットワークトラフィックの時系列解析

時間とともに変化する動的な挙動の解析

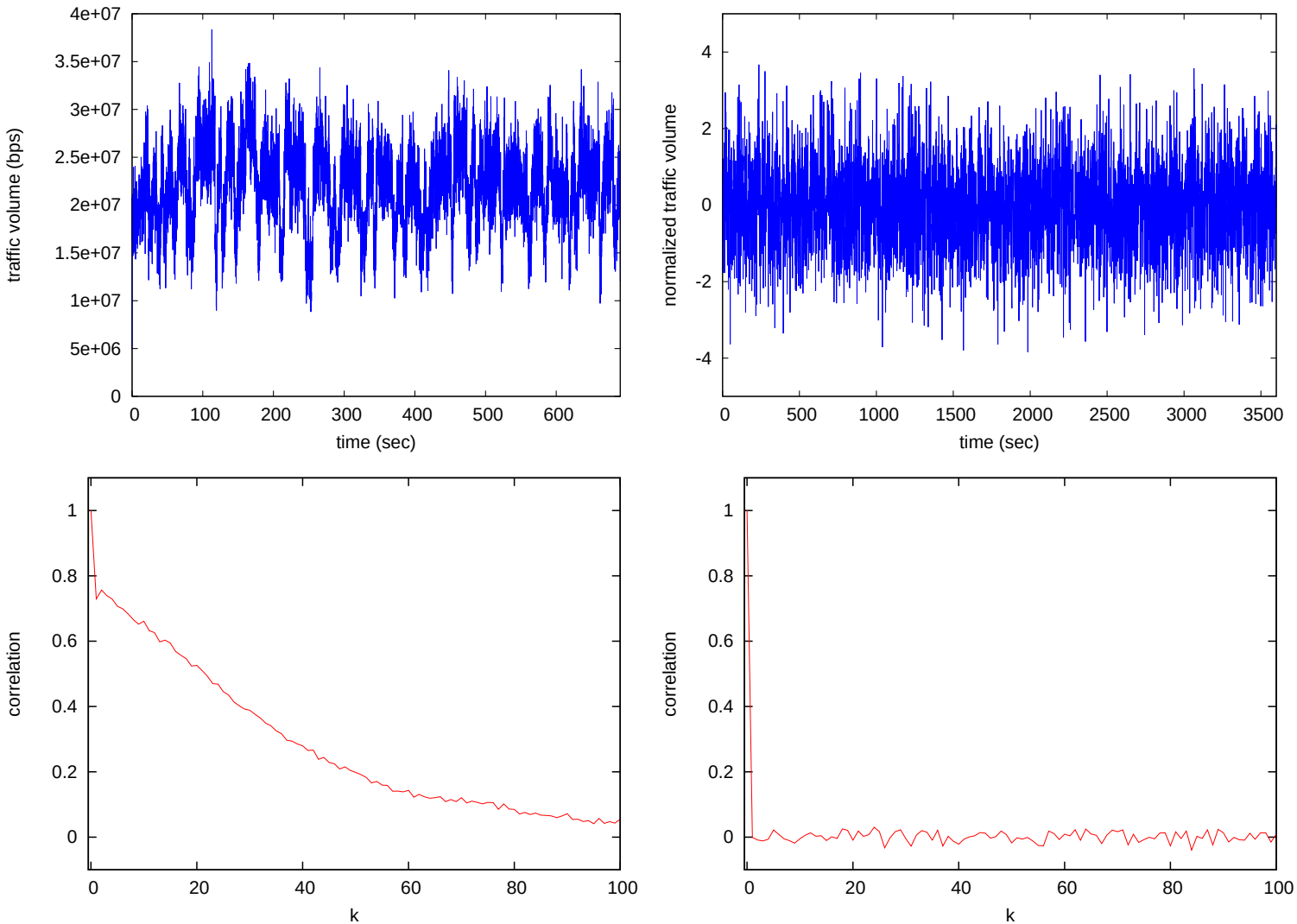
- 数学的な取り扱いは難しい
- 限られたツール

トピック

- 自己相関 (autocorrelation)
- 定常過程 (stationary process)
- 長期記憶 (long-range dependence)
- 自己相似トラフィック (self-similar traffic)

ネットワークトラフィックの自己相関

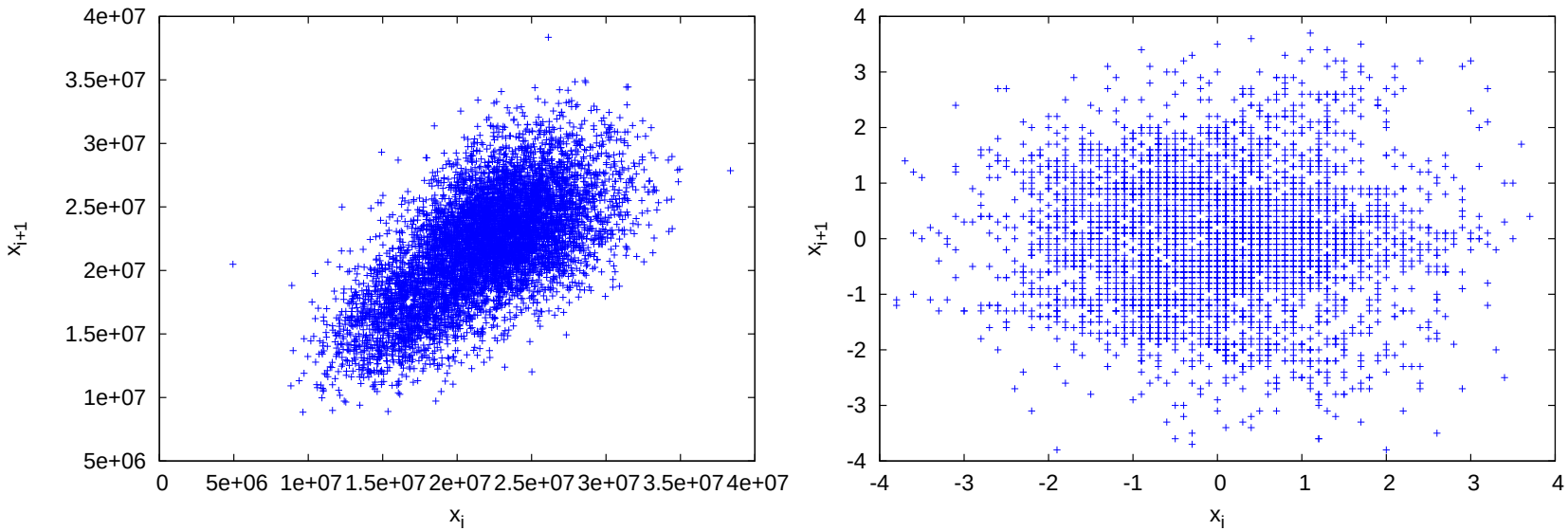
- 過去の状態の影響 (トレンド) と周期性 (日、週、季節)
- 自己相関 (autocorrelation): 同一変数の異なる時間の値の相関



(左) 実トラフィック (右) 乱数から生成したトラフィック (上) 時系列グラフ (下) 自己相関

自己相関とラグプロット

- ラグ (lag) プロット: x_i と x_{i+k} の散布図
  - 自己相関の存在を確認する簡単な方法
  - k を大きくすると長周期の繰り返しパターンを発見可能



ラグプロットの例: (左) 実トラフィック (右) 乱数から生成したトラフィック

自己相関

- 確率過程 (stochastic process)

$$\{x(t), t \in T\}$$

- 自己相関 (autocorrelation): 同一変数の時刻 t_1 の値と t_2 の値の相関
- 自己相関関数 (autocorrelation function)

$$R(t_1, t_2) = E[x(t_1)x(t_2)]$$

- 自己共分散 (autocovariance)

$$Cov(t_1, t_2) = E((x(t_1) - \mu_{t_1})(x(t_2) - \mu_{t_2})] = E[x(t_1)x(t_2)] - \mu_{t_1}\mu_{t_2}$$

定常過程 (stationary process)

- 時系列 X_t が定常過程
  - 平均が変化しない: $E(X_t) = \mu$
  - かつ自己共分散が k にのみ依存

$$\gamma_k = Cov(X_t, X_{t+k}) = E((X_t - \mu)(X_{t+k} - \mu))$$

$$\gamma_0 = Var(X_t) = E((X_t - \mu)^2)$$

- 自己相関係数 (autocorrelation coefficient)
  - 自己共分散を分散で正規化
  - 過去からの影響を示す

$$\rho_k = \frac{\gamma_k}{\gamma_0}$$

ホワイトノイズ

ホワイトノイズ: 定常過程で自己相関係数が 0

$$\rho_k = 0 \ (k \neq 0)$$

IID 過程 (independent identically distributed process)

- 平均と分散が一定のホワイトノイズ
  - 確率過程の話に必ず出てくる
- X_t が互いに独立で同じ分布に従う
  - independent: X_t が互いに独立 (無相関)
  - identically distributed: X_t が同じ分布に従う

非定常過程

- 非定常
  - 平均または自己共分散が時間とともに変化
- 数学的な扱いが困難
  - 一般には時系列の差分を取って定常化する必要
- 定常判定
  - パワースペクトル密度を調べ
    - べき指数が 1.0 より大きい場合は非定常
- ネットワークでは非定常なトラフィックが観測される
  - 輻輳、DoS/flooding 等の攻撃

パワースペクトル密度 (power spectral dencity)

- 定常過程のパワースペクトル密度は自己相関関数のフーリエ変換
  - 時間領域から周波数領域への変換
  - 時系列データを sin,cos の重ね合わせで表現

$$S(f) = \int_{-\infty}^{\infty} R(\tau) e^{-2\pi i f \tau} d\tau$$

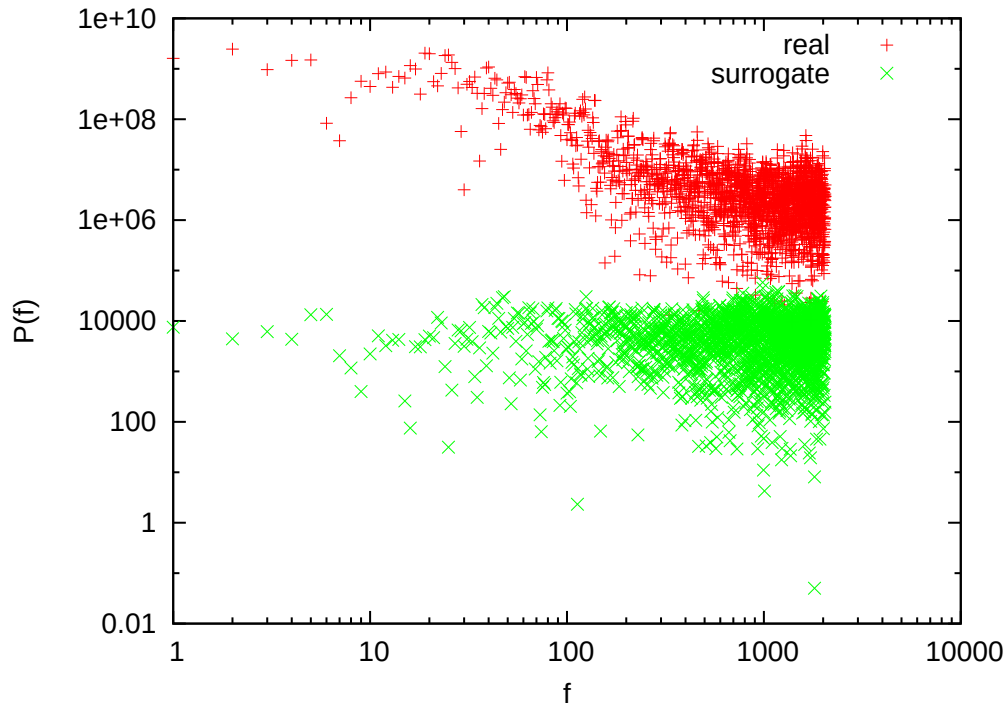
  - パワースペクトル密度

$$P(f) \equiv |S(f)|^2 + |S(-f)|^2,\ 0 \leq f < \infty$$

- パワースペクトル密度は各周波数成分の平均パワーを示す

パワースペクトル密度の性質

- ホワイトノイズ (無相関): $P(f) \sim const$
- 自己相似 (長期記憶): $P(f) \sim f^{-\alpha}, 0 < \alpha \leq 1.0$
- 1/f ゆらぎ (パワーが周波数に反比例): $\alpha = 1.0$
- 非定常: $\alpha > 1.0$



例: (赤) 実トラフィック (緑) 乱数から生成したトラフィック

短期記憶と長期記憶

自己共分散は各々の時差 k の影響を個別に示す。
全体を見るために全ての時差 k について自己共分散の総和を取る

- 短期記憶性
  - $\sum_k \rho(k)$ が有限

$$\sum_{k=0}^{\infty} |\rho(k)| < \infty$$

  - $\rho(k)$ が指数関数と同様か、より早く減衰
  - 特徴
    - 平均値周辺でゆらぐ
    - 遠い過去の影響はない
- 長期記憶性
  - $\sum_k \rho(k)$ が発散

$$\sum_{k=0}^{\infty} |\rho(k)| = \infty$$

  - 自己相関係数が双曲線的に減衰
  - 特徴
    - 平均から大きく外れた値が観測される

自己相似トラフィック

ネットワークトラフィックは厳密な自己相似ではないが、場合によって他より良いモデルを与える

- スケールフリー
- 長期記憶
- 自己共分散がべき的に減衰

$$\rho(k) \sim k^{-\alpha} \quad (k \to \infty) \quad 0 < \alpha < 1$$

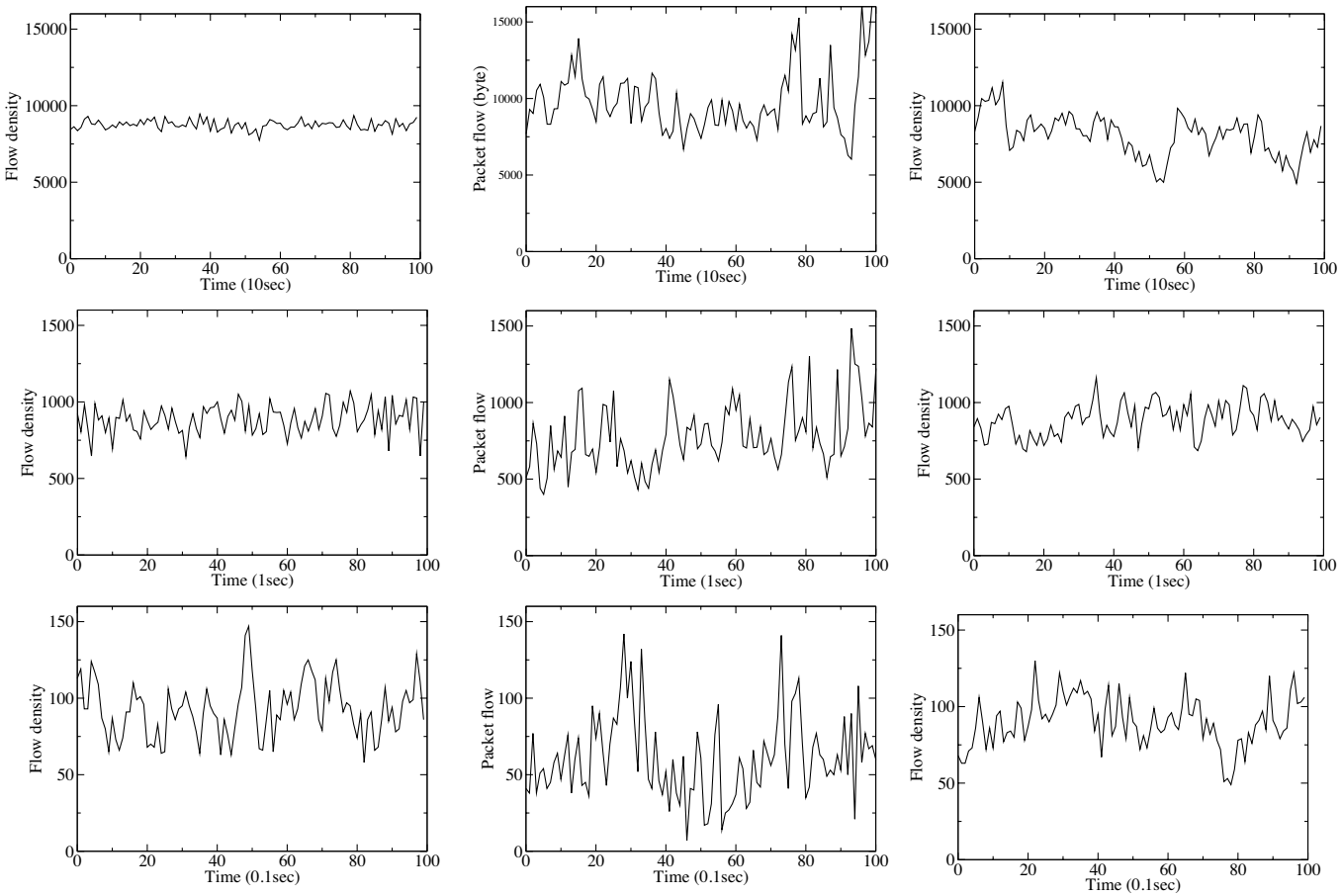
- 同様にパワースペクトル密度もべき的に減衰
  - 低周波成分 (遠い過去) の影響が大きい

$$P(f) \sim |f|^{-\alpha} \quad (f \to 0)$$

- 分散が発散

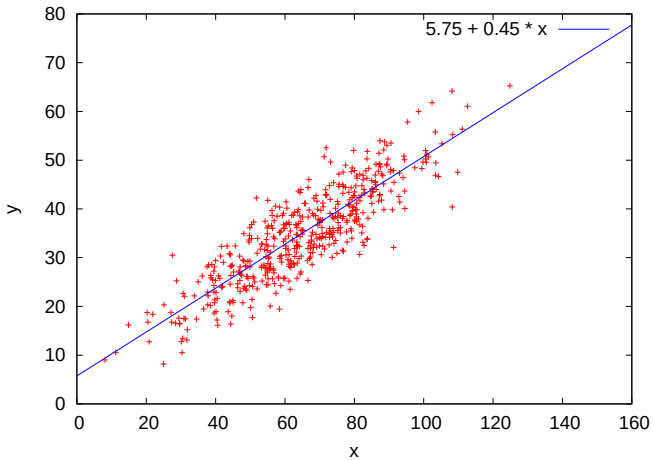
ネットワークトラフィックの自己相似性

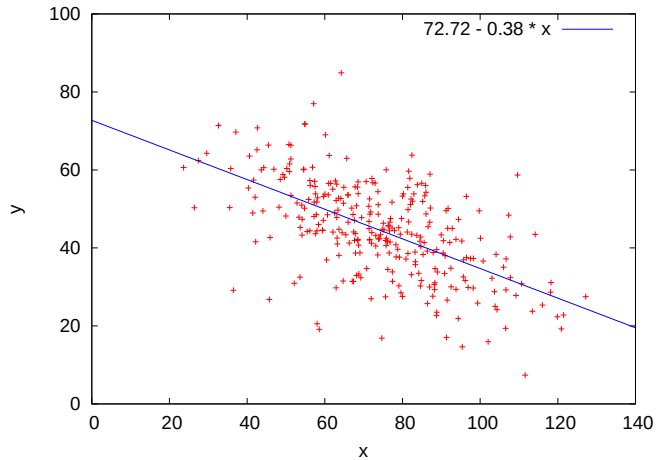
- (左) 指数関数モデル (中) 実トラフィック (右) 自己相似モデル
- 時間粒度: (上)10sec (中)1 sec (下)0.1 sec



前回の演習: 線形回帰の計算

- 前回のデータを使い回帰直線を計算する
  - correlation-data-1.txt, correlation-data-2.txt





data-1:r=0.87 (left), data-2:r=-0.60 (right)

前回の演習: 回帰直線の計算スクリプト

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

# regular expression for matching 2 floating numbers
re = /([-+]?\d+(?:\.\d+)?)\s+([-+]?\d+(?:\.\d+)?)/

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
      y = $2.to_f

      sum_x += x
      sum_y += y
      sum_xx += x**2
      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**2)
b0 = mean_y - b1 * mean_x

printf "b0:%.3f b1:%.3f\n", b0, b1
```

前回の演習: 散布図に回帰直線を加える

```
set xrange [0:160]
set yrange [0:80]

set xlabel "x"
set ylabel "y"

plot "correlation-data-1.txt" notitle with points, \
5.75 + 0.45 * x lt 3
```

今回の演習: 自己相関

- 1 週間分のトラフィックデータから自己相関を計算する

```
# ruby autocorr.rb autocorr_5min_data.txt > autocorr.txt
# head -10 autocorr_5min_data.txt
2011-02-28T00:00 247 6954152
2011-02-28T00:05 420 49037677
2011-02-28T00:10 231 4741972
2011-02-28T00:15 159 1879326
2011-02-28T00:20 290 39202691
2011-02-28T00:25 249 39809905
2011-02-28T00:30 188 37954270
2011-02-28T00:35 192 7613788
2011-02-28T00:40 102 2182421
2011-02-28T00:45 172 1511718
# head -10 autocorr.txt
0 1.000
1 0.860
2 0.860
3 0.857
4 0.857
5 0.854
6 0.851
7 0.849
8 0.846
9 0.841
```

自己相関関数の求め方

タイムラグ k の自己相関関数

$$R(k) = \frac{1}{n}\sum_{i=1}^{n} x_i x_{i+k}$$

$k = 0$ の場合は、同一データの相関なので、$R(k)/R(0)$ で規格化する

$$R(0) = \frac{1}{n}\sum_{i=1}^{n} x_i^2$$

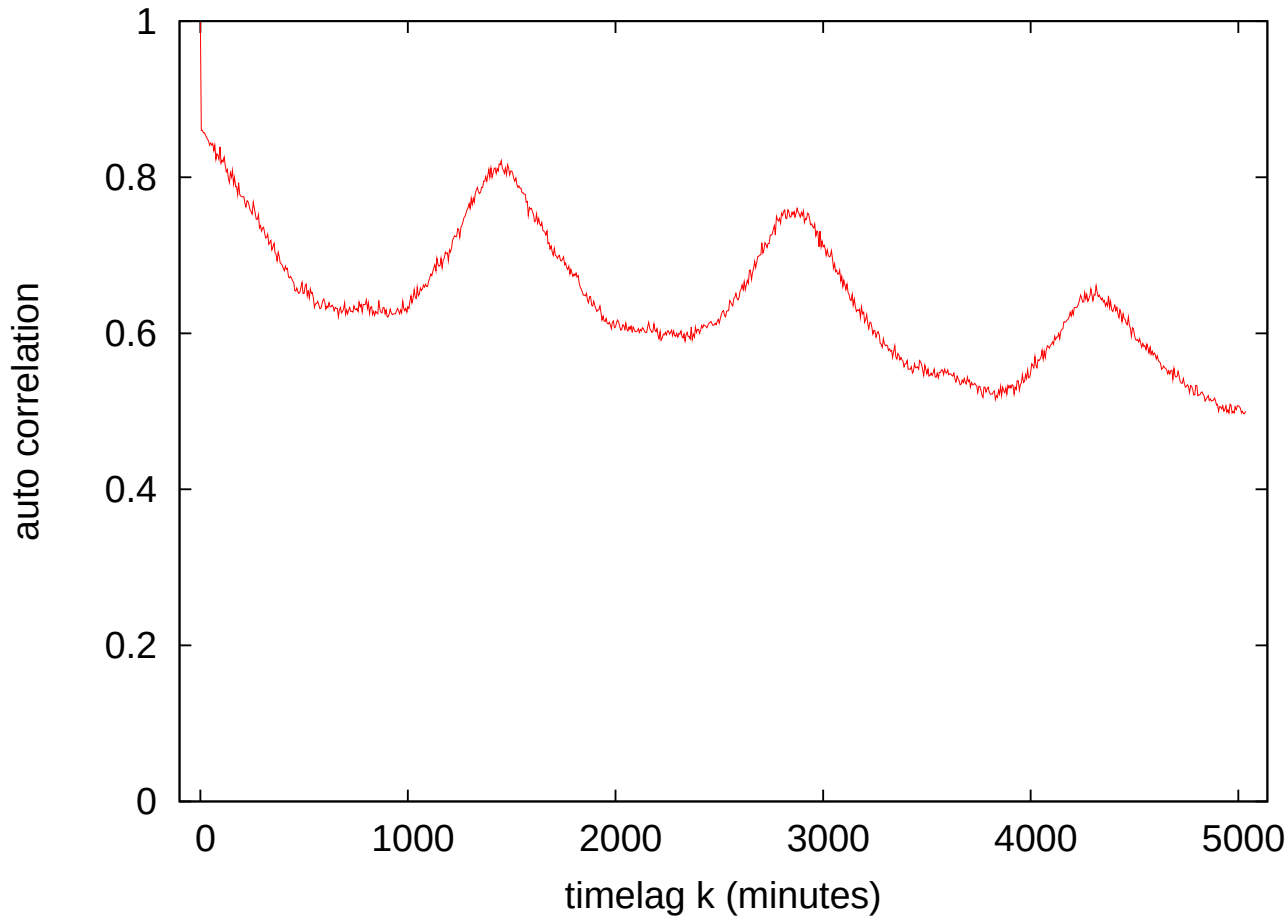
2n 個のデータ数が必要

自己相関関数スクリプト

```
# regular expression for matching 5-min timeseries
re = /^(\d{4}-\d{2}-\d{2})T(\d{2}:\d{2})\s+(\d+)\s+(\d+)/

v = Array.new() # array for timeseries
ARGF.each_line do |line|
  if re.match(line)
    v.push $3.to_f
  end
end

n = v.length # n: number of samples
h = n / 2 - 1 # (half of n) - 1

r = Array.new(n/2) # array for auto correlation
for k in 0 .. h # for different timelag
  s = 0
  for i in 0 .. h
    s += v[i] * v[i + k]
  end
  r[k] = Float(s)
end

# normalize by dividing by r0
if r[0] != 0.0
  r0 = r[0]
  for k in 0 .. h
    r[k] = r[k] / r0
    printf "%d %.3f\n", k, r[k]
  end
end
```

自己相関プロット

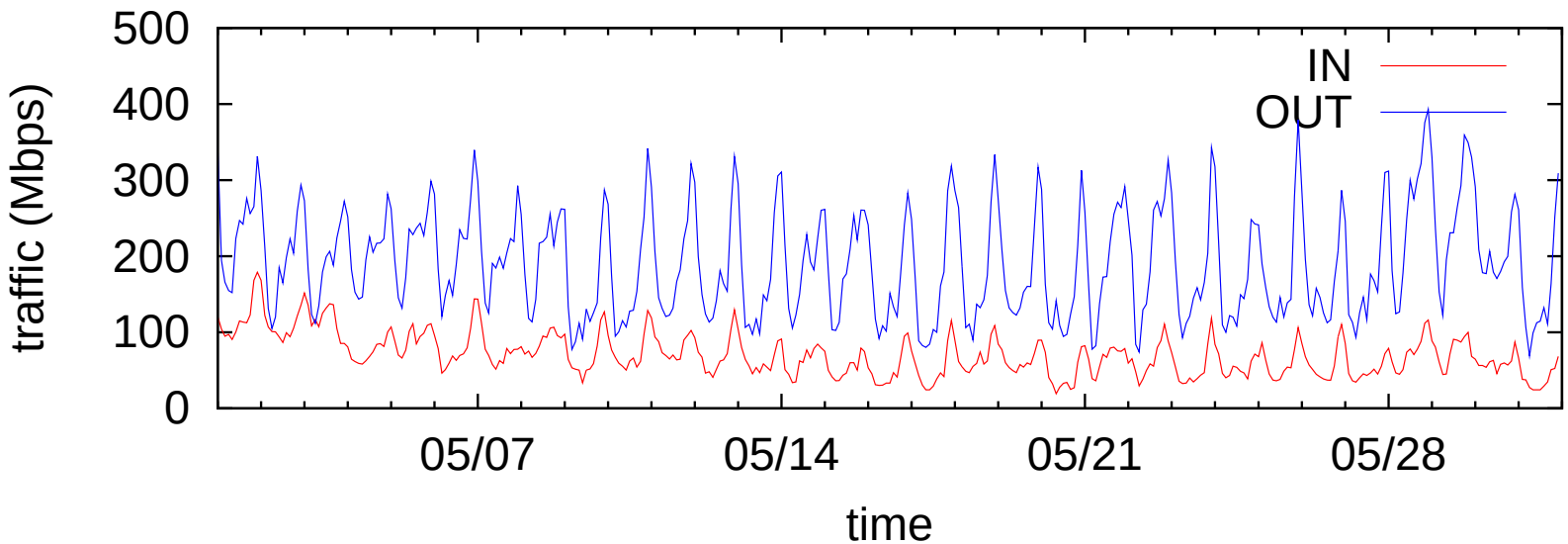
```
set xlabel "timelag k (minutes)"
set ylabel "auto correlation"
set xrange [-100:5140]
set yrange [0:1]
plot "autocorr.txt" using ($1*5):2 notitle with lines
```



今回の演習 2: トラフィック解析

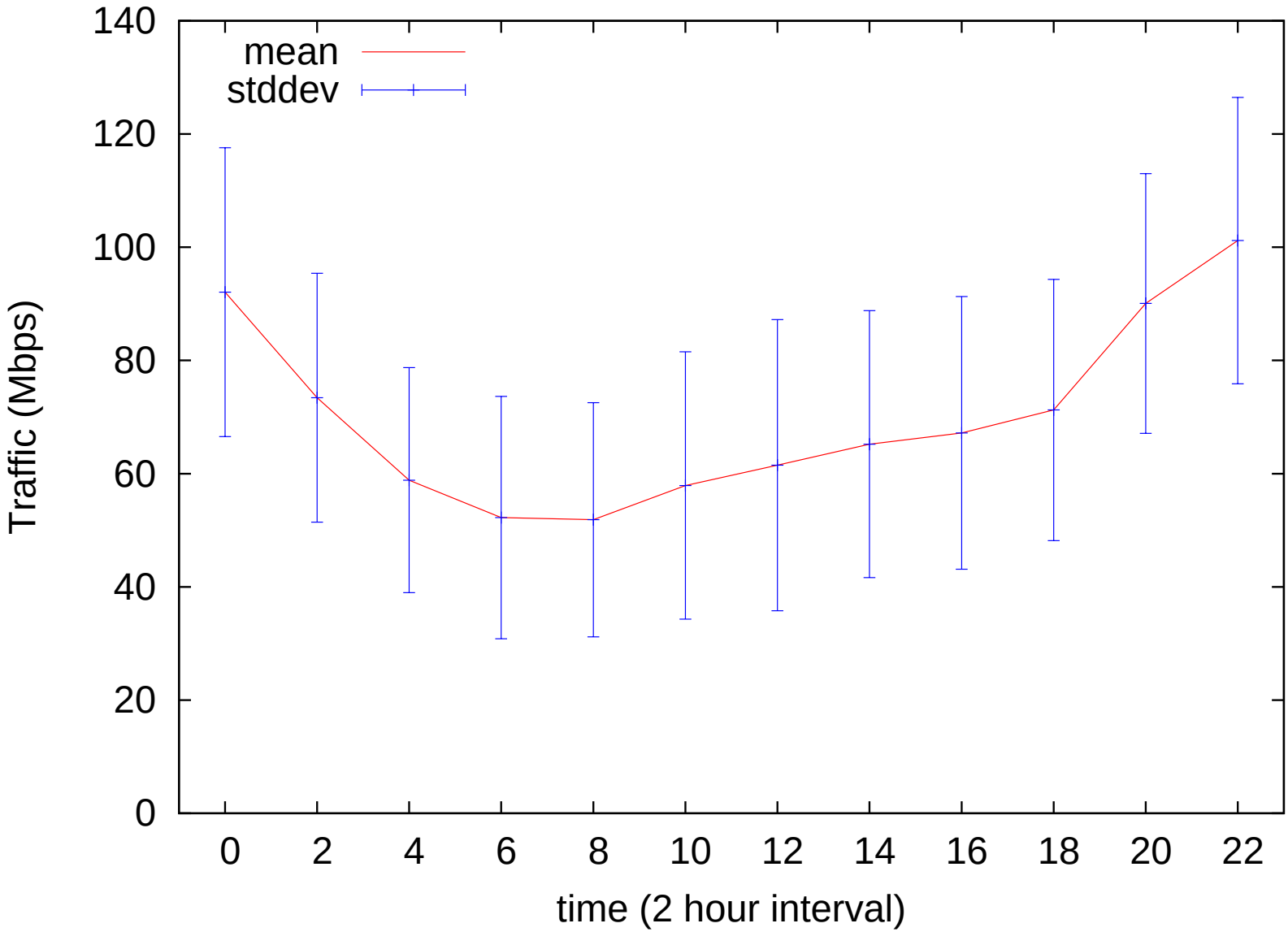
演習用データ: ifbps.txt

- あるブロードバンド収容ルータのインターフェイスカウンタ値
- 2011 年 5 月の 1ヶ月分、2 時間粒度
- format: time IN(bits/sec) OUT(bits/sec)
- 元データのフォーマットを変換してある
  - original format: unix_time IN(bytes/sec) OUT(bytes/sec)
- ここでは IN トラフィックを解析
  - OUT トラフィックは課題 2



今回の演習 2: 時間帯別トラフィック

- 時間毎の平均と標準偏差をプロット



今回の演習 2: 時間帯別トラフィック抽出スクリプト

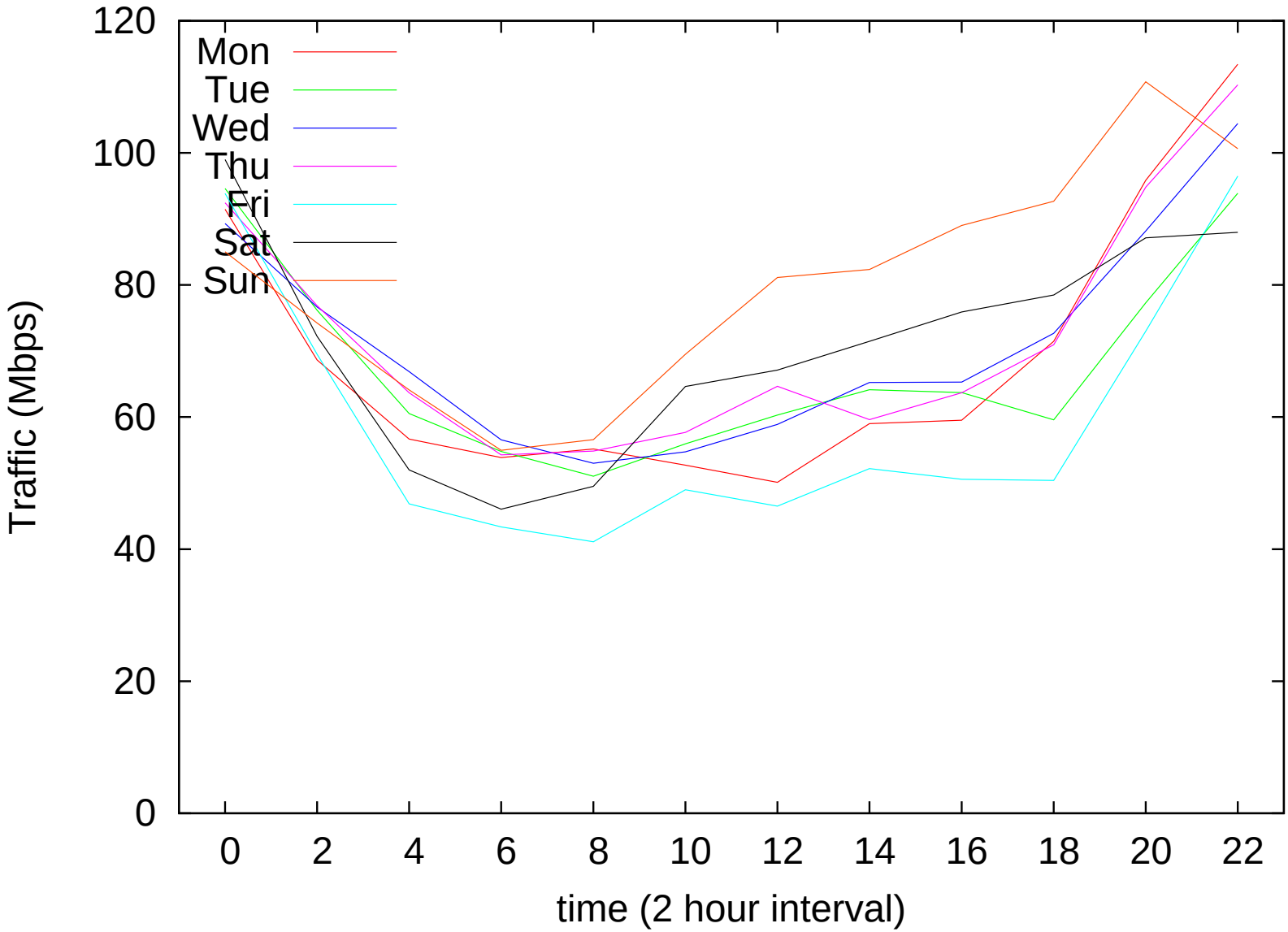
```
#   time  in_bps  out_bps
re = /^\d{4}-\d{2}-(\d{2})T(\d{2}):\d{2}:\d{2}\s+(\d+\.\d+)\s+\d+\.\d+/

# arrays to hold values for every 2 hours
sum = Array.new(12, 0.0)
sqsum = Array.new(12, 0.0)
num = Array.new(12, 0)

ARGF.each_line do |line|
  if re.match(line)
    # matched
    hour = $2.to_i / 2
    bps = $3.to_f

    sum[hour] += bps
    sqsum[hour] += bps**2
    num[hour] += 1
  end
end
printf "#hour\tn\tmean\t\tstddev\n"
for hour in 0 .. 11
  mean = sum[hour] / num[hour]
  var = sqsum[hour] / num[hour] - mean**2
  stddev = Math.sqrt(var)

  printf "%02d\t%d\t%.1f\t%.1f\n", hour * 2, num[hour], mean, stddev
end
```

今回の演習 2: 時間帯別トラフィックのプロット

```
set xlabel "time (2 hour interval)"
set xtic 2
set xrange [-1:23]
set yrange [0:]
set key top left
set ylabel "Traffic (Mbps)"

plot "hourly_in.txt" using 1:($3/1000000) title 'mean' with lines, \
"hourly_in.txt" using 1:($3/1000000):($4/1000000) title "stddev" with yerrorbars lt 3
```

今回の演習 2: 曜日別時間帯別トラフィック

- 曜日毎のトラフィックをプロット



今回の演習 2: 曜日別時間帯別トラフィックの抽出

```
#   time  in_bps  out_bps
re = /^\d{4}-\d{2}-(\d{2})T(\d{2}):\d{2}:\d{2}\s+(\d+\.\d+)\s+\d+\.\d+/

# 2011-05-01 is Sunday, add wdoffset to make wday start with Monday
wdoffset = 5

# traffic[wday][hour]
traffic = Array.new(7){ Array.new(12, 0.0) }
num = Array.new(7){ Array.new(12, 0) }

ARGF.each_line do |line|
  if re.match(line)
    # matched
    wday = ($1.to_i + wdoffset) % 7
    hour = $2.to_i / 2
    bps = $3.to_f

    traffic[wday][hour] += bps
    num[wday][hour] += 1
  end
end
printf "#hour\tMon\tTue\tWed\tThu\tFri\tSat\tSun\n"
for hour in 0 .. 11
  printf "%02d", hour * 2
  for wday in 0 .. 6
    printf " %.1f", traffic[wday][hour] / num[wday][hour]
  end
  printf "\n"
end
```

今回の演習 2: 曜日別時間帯別トラフィックのプロット

```
set xlabel "time (2 hour interval)"
set xtic 2
set xrange [-1:23]
set yrange [0:]
set key top left
set ylabel "Traffic (Mbps)"

plot "week_in.txt" using 1:($2/1000000) title 'Mon' with lines, \
"week_in.txt" using 1:($3/1000000) title 'Tue' with lines, \
"week_in.txt" using 1:($4/1000000) title 'Wed' with lines, \
"week_in.txt" using 1:($5/1000000) title 'Thu' with lines, \
"week_in.txt" using 1:($6/1000000) title 'Fri' with lines, \
"week_in.txt" using 1:($7/1000000) title 'Sat' with lines, \
"week_in.txt" using 1:($8/1000000) title 'Sun' with lines
```

今回の演習 2: 曜日間の相関係数行列

- 曜日間の相関係数行列を計算
  - 曜日間の各時間帯平均値を使う

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Mon	1.000	0.888	0.970	0.974	0.919	0.785	0.736
Tue	0.888	1.000	0.935	0.927	0.989	0.840	0.624
Wed	0.970	0.935	1.000	0.980	0.938	0.811	0.745
Thu	0.974	0.927	0.980	1.000	0.941	0.813	0.756
Fri	0.919	0.989	0.938	0.941	1.000	0.829	0.610
Sat	0.785	0.840	0.811	0.813	0.829	1.000	0.853
Sun	0.736	0.624	0.745	0.756	0.610	0.853	1.000

今回の演習 2: 曜日間の相関係数行列の計算

- 曜日別時間帯別で作った配列を使えばよい

```
n = 12
for wday in 0 .. 6
  for wday2 in 0 .. 6
    sum_x = sum_y = sum_xx = sum_yy = sum_xy = 0.0
    for hour in 0 .. 11
      x = traffic[wday][hour] / num[wday][hour]
      y = traffic[wday2][hour] / num[wday2][hour]

      sum_x += x
      sum_y += y
      sum_xx += x**2
      sum_yy += y**2
      sum_xy += x * y
    end
    r = (sum_xy - sum_x * sum_y / n) /
      Math.sqrt((sum_xx - sum_x**2 / n) * (sum_yy - sum_y**2 / n))
    printf "%.3f\t", r
  end
  printf "\n"
end
```

課題 2: トラフィック解析

- ねらい: 実時系列データから時間帯別や曜日別の情報を抽出
- 課題用データ: ifbps.txt (今回の演習 2 と同じ)
  - あるブロードバンド収容ルータのインターフェイスカウンタ値
  - 2011 年 5 月の 1ヶ月分、2 時間粒度
  - format: time IN(bits/sec) OUT(bits/sec)
- 提出項目
  1. OUT の時間帯別トラフィックプロット
     - 時間毎の平均と標準偏差をプロット
  2. OUT の曜日別時間帯別トラフィックプロット
     - 曜日毎のトラフィックをプロット
  3. OUT の曜日間の相関係数行列テーブル
     - 曜日間の各時間帯平均値を使い相関係数行列を計算
  4. オプション
     - その他の解析
  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: 2012 年 6 月 18 日

まとめ

時系列データ

- インターネットと時刻
- ネットワークタイムプロトコル
- トラフィック計測
- 時系列解析
- 演習: 時系列解析
- **課題 2**

次回予定

第 9 回 トポロジーとグラフ (6/8)

- 経路制御
- グラフ理論
- 最短経路探索
- 演習: 最短経路探索