

# インターネット計測とデータ解析 第9回

長 健二郎

2011年6月29日

# 前回のおさらい

## ロングテールとさまざまな分布

- ▶ 正規分布
- ▶ その他の主要な分布
- ▶ 信頼区間と検定
- ▶ 演習:分布の生成、信頼区間

# 今日のテーマ

インターネットの時間変化を計る

- ▶ インターネットと時刻
- ▶ ネットワークタイムプロトコル
- ▶ 時系列解析
- ▶ 演習:時系列解析

# 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

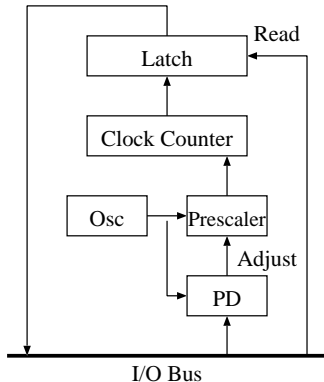
# クロックの誤差

- ▶ 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

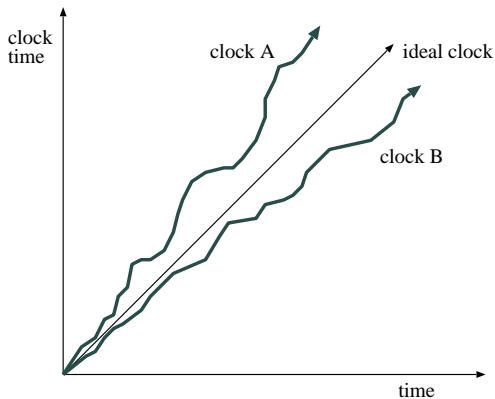
### i8254 プログラムインターバルタイマー

- ▶ 16-bit フリーランニング 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ウ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
  - ▶ ハードウェア仕様の許容誤差:  $10^{-5}$ 
    - ▶ 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 内蔵フリーランニングカウンター
  - ▶ 可変クロックやマルチ CPU で問題
- ▶ ACPI (Advanced Configuration and Power Interface)
  - ▶ パワー管理機能が提供するフリーランニングカウンター
- ▶ 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

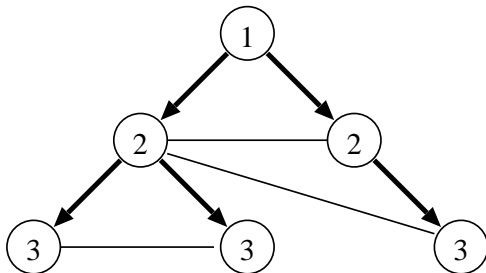
- ▶ OS はソフトウェアにより時刻を管理
  - ▶ 起動時にカレンダーチップから時刻を得る
  - ▶ ハードウェアクロック割り込み毎に時刻をアップデート
- ▶ 従来の UNIX では、デフォルトで 10ms ごとにクロック割り込みが発生するようにクロックカウンターを設定

## 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ェアクロックを補間
    - ▶ 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 ソフトウェア処理時間 (e.g., setitimer):
  - ▶ ソフトウェ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



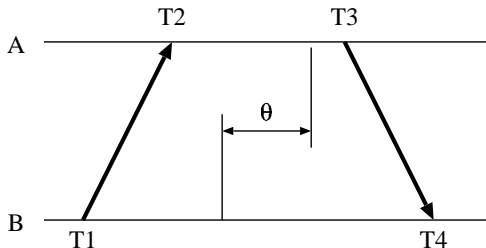
# NTP 同期モード

- ▶ 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

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

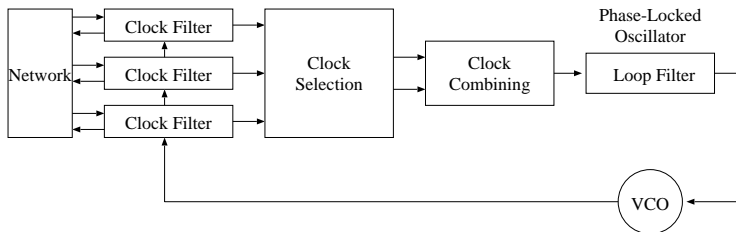


全てのメッセージに以下を含める

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

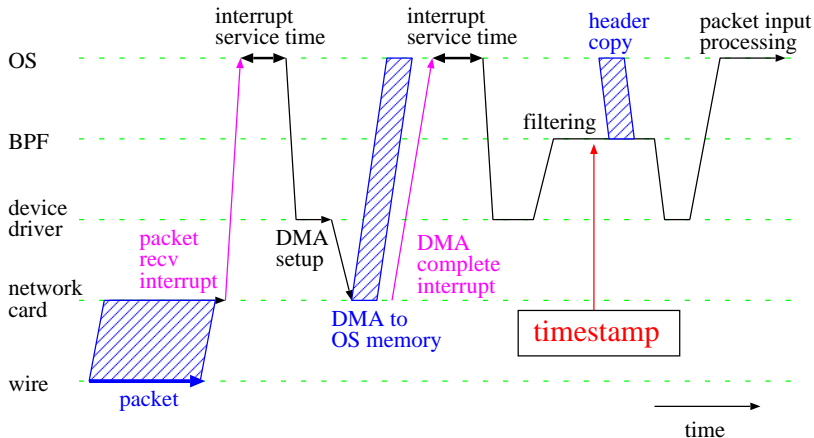
# NTP システムモデル

- ▶ 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
  - ▶ インターセクショナルアルゴリズム: 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

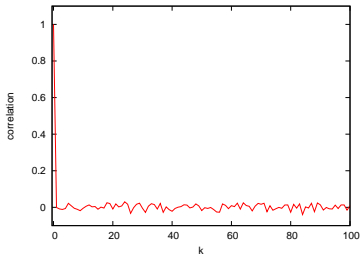
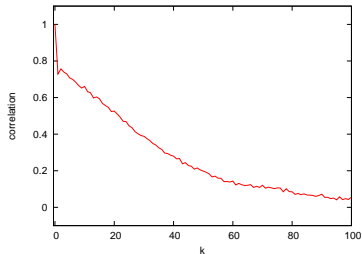
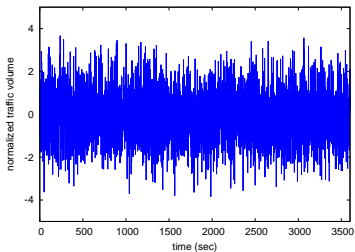
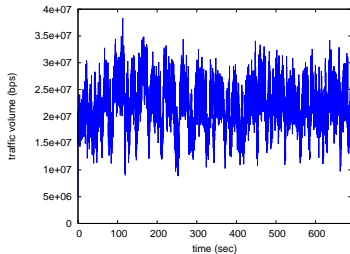
トピック

- ▶ 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

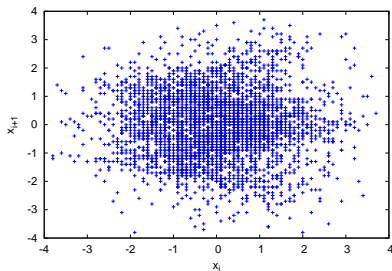
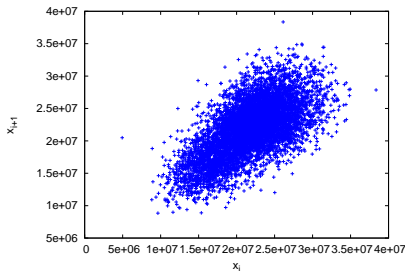
- ▶ 過去の状態の影響 (トレンド) と周期性 (日、週、季節)
- ▶ 自己相関 (autocorrelation): 同一変数の異なる時間の値の相関



(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)

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

# 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

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

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

- ▶ 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

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

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 density)

- ▶ 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

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

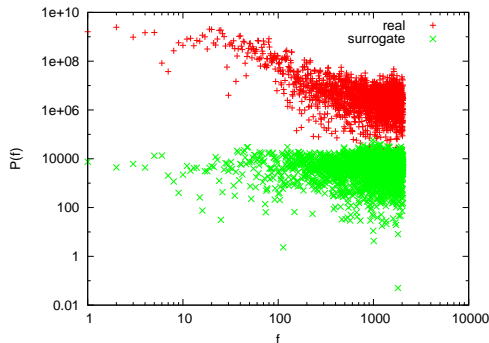
- ▶ パワースペクトル密度

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

- ▶ 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):  $P(f) \sim \text{const}$
- ▶ 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
  - ▶ 平均値周辺でゆらく
  - ▶ 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

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

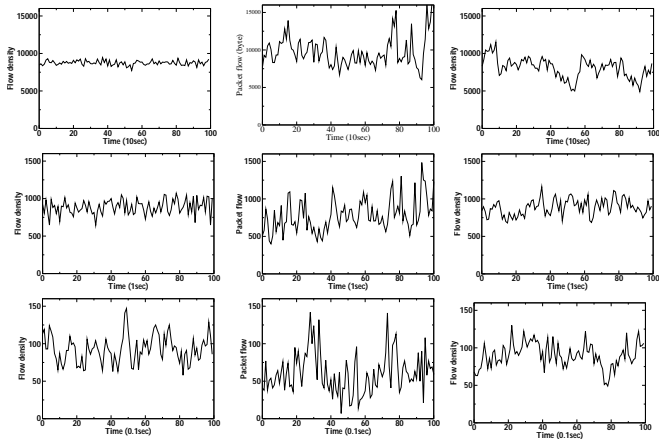
- ▶ 同様にパワースペクトル密度もべき的に減衰
  - ▶ 低周波成分 (遠い過去) の影響が大きい

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

- ▶ 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



## 演習 1: CCDF のプロット

- ▶ 演習用アクセスログ (2011-02-28/2011-03-06) からコンテンツ毎のアクセス数分布を求め、CCDF にプロットする

# コンテンツ毎のアクセス数の抽出スクリプト

```
# 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)
count = parsed = 0
ARGF.each_line do |line|
  count += 1
  if re.match(line)
    # match
    host, ident, user, time, request, status, bytes = $~.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
      parsed += 1
      # count contents by request
      contents[url] += 1
    else
      # match failed. print a warning msg
      $stderr.puts("request match failed at line #{count}: #{line.dump}")
    end
  else
    # match failed.
    $stderr.puts("match failed at line #{count}: #{line.dump}")
  end
end
contents.sort_by{|key, value| -value}.each do |key, value|
  puts "#{value} #{key}"
end
$stderr.puts "# #{contents.size} unique contents in #{parsed} successful requests"
$stderr.puts "# parsed:#{parsed} ignored:#{count - parsed}"
```

# CCDF への変換スクリプト

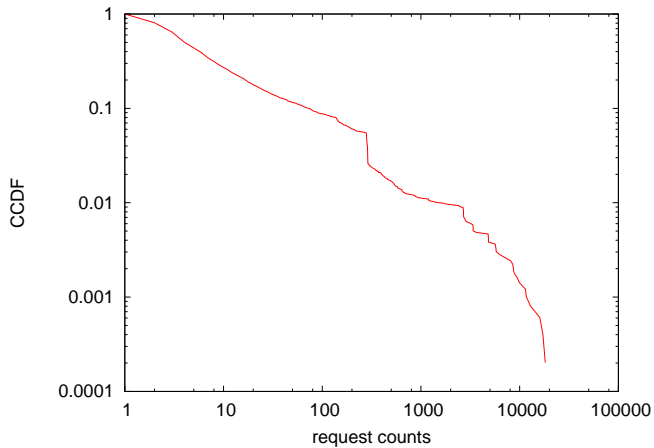
- ▶ コンテンツ毎のアクセス数を読み込んで、CCDF を計算

```
re = /^(\d+)/

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

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

# コンテンツアクセス数の CCDF



## 演習 2: 自己相関

- ▶ 演習用アクセスログ (2011-02-28/2011-03-06) から 5 分粒度の時系列を作る
- ▶ 以前作成した、1 時間粒度のスクリプトを以下のように修正

1-hour timebins:

```
# create the corresponding key for 1-hour timebins
key = ts.strftime("%Y-%m-%dT%H")
```

5-minutes timebins:

```
# 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}")
```



# 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+)/

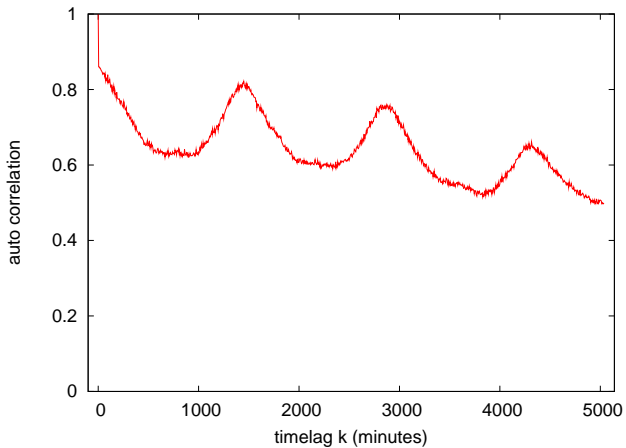
v = Array.new() # array for timeseries
ARGF.each_line do |line|
  if re.match(line)
    v.push $3.to_f
  end
end

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
    puts "#{k} #{r[k]}"
  end
end
```

# 自己相関プロット



# まとめ

## インターネットの時間変化を計る

- ▶ インターネットと時刻
- ▶ ネットワークタイムプロトコル
- ▶ 時系列解析
- ▶ 演習:時系列解析

# 次回予定

## 第 10 回 インターネットのトラフィック量を計る (7/6)

- ▶ トラフィック計測
- ▶ 演習:トラフィック量解析