digital signal processing

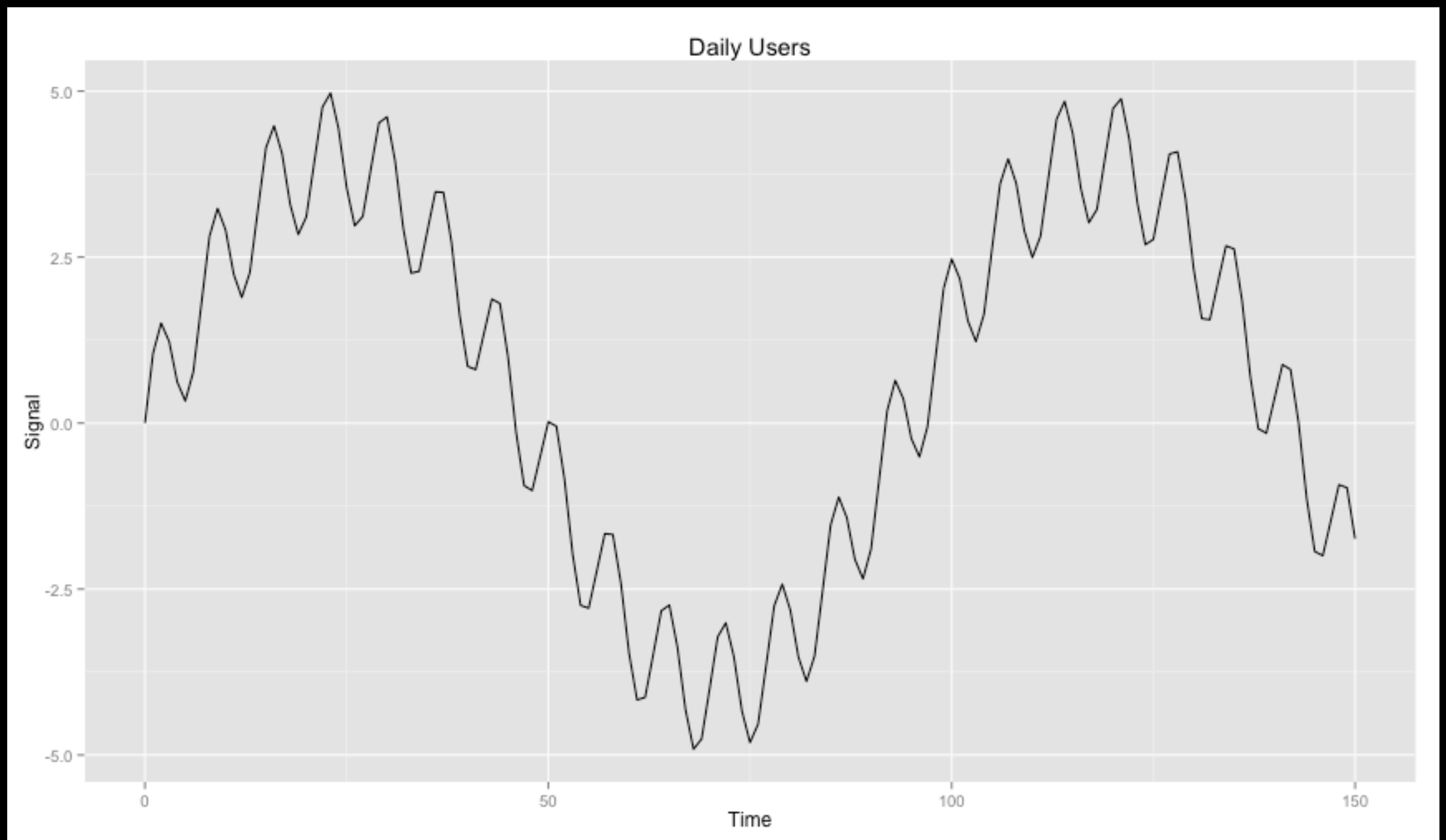
in hadoop

with scalding

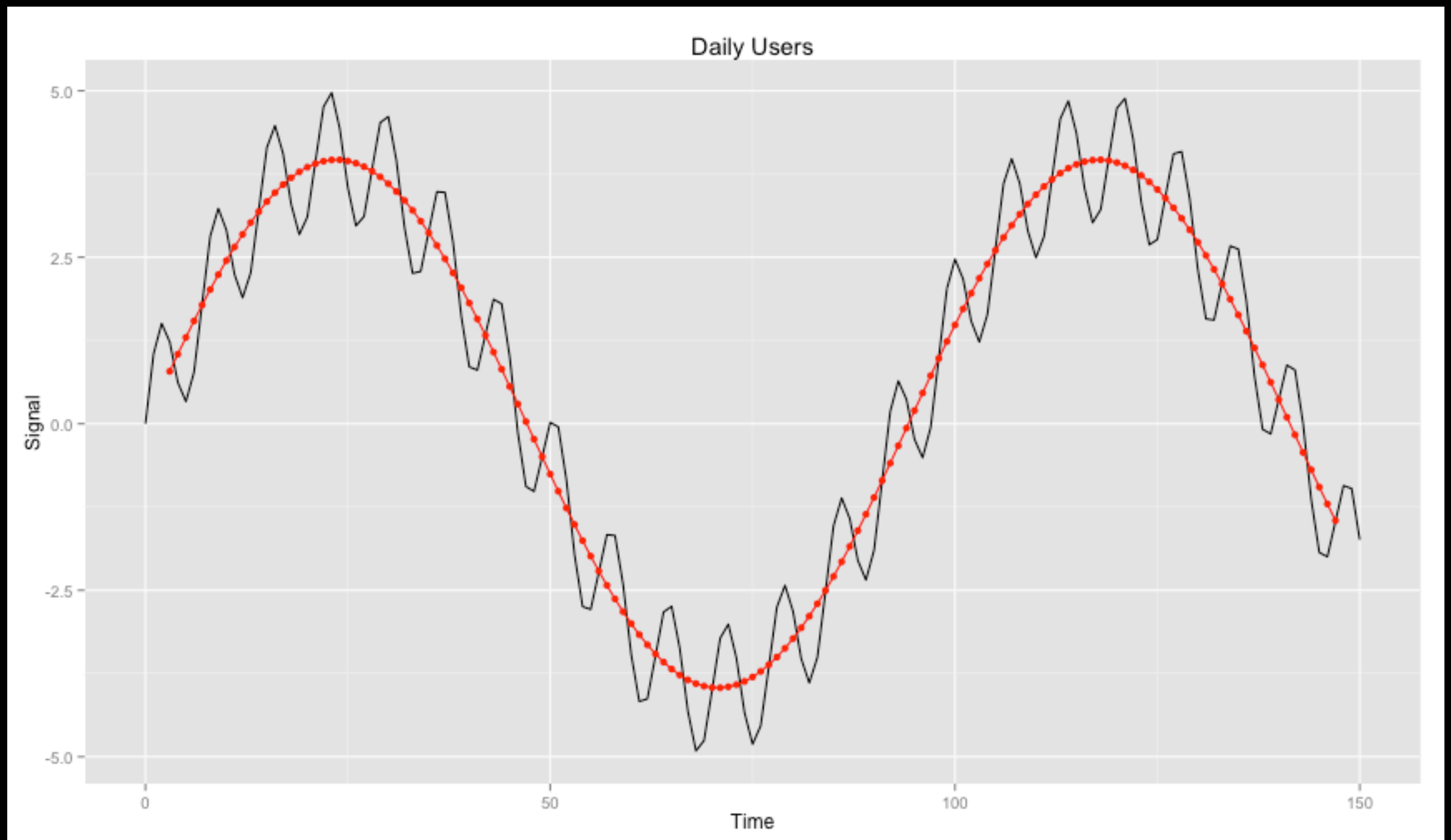
Adam Laiacano
adam@tumblr.com
@adamlaiacano

Overview

- Intro to digital signals and filters
  - sampling
  - frequency domain
  - FIR / IIR filters
- Very quick intro to Scalding
- Filtering tons of signals at once
- Application: Finding trending blogs on tumblr



1 sample / day



7-day average

1 sample / day

Some Definitions

Signal - Any series of data (Volts, posts, etc) that is measured at regular intervals.

Sampling period, T_s - Time between samples (my example was Ts = 1 day)

Sampling frequency f_s - 1 / Ts

Nyquist frequency - Highest frequency that can be represented = $f_s/2$

Filter - A system to reduce or enhance certain aspects (phase, magnitude) of a signal.

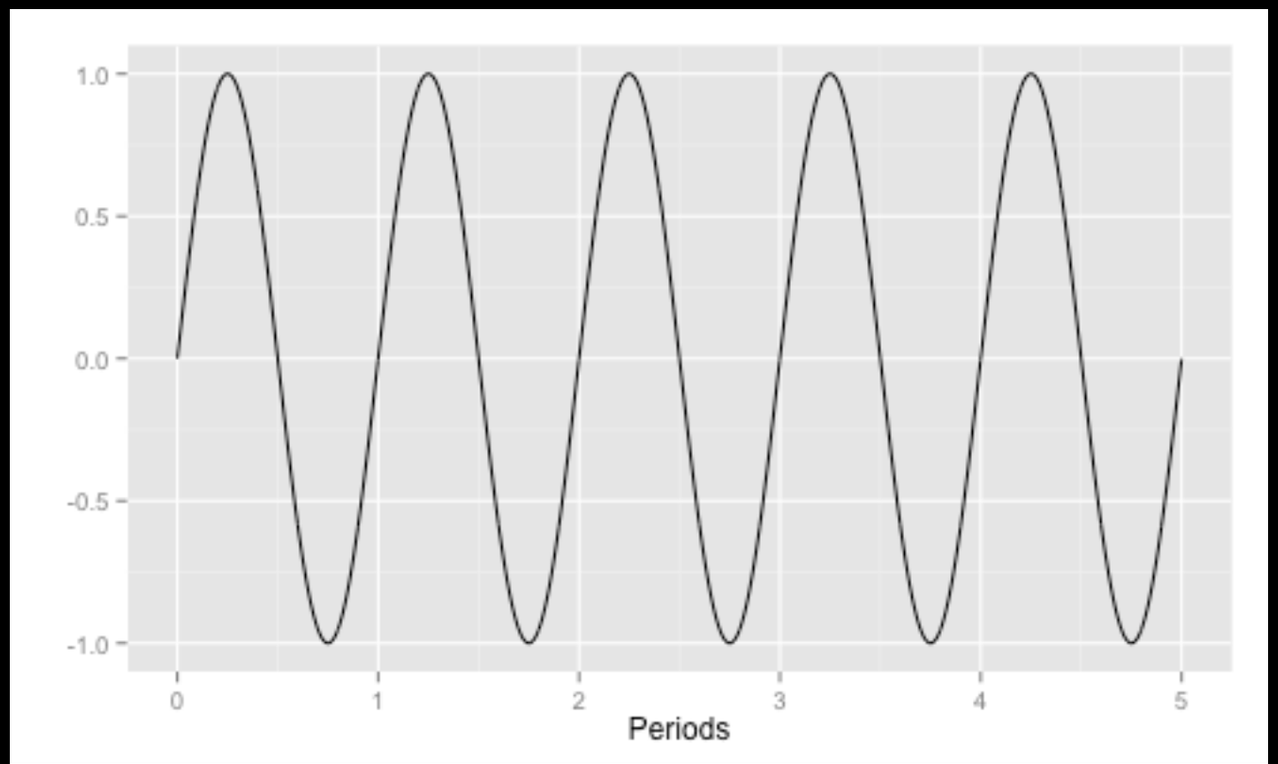
Stopband - The frequency range we want to eliminate

Passband - The frequency range we want to preserve

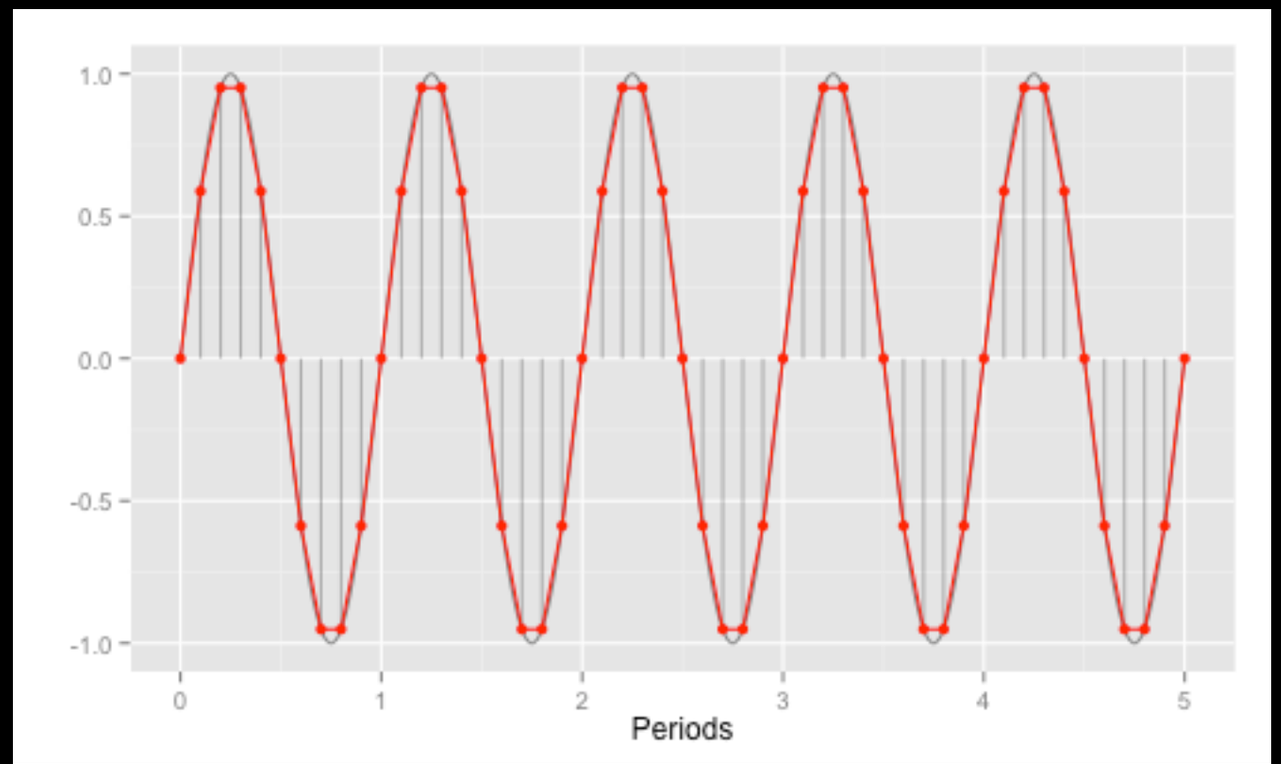
Cutoff frequency, f_c - The boundary of the stopband/passband

Signals

Sampling

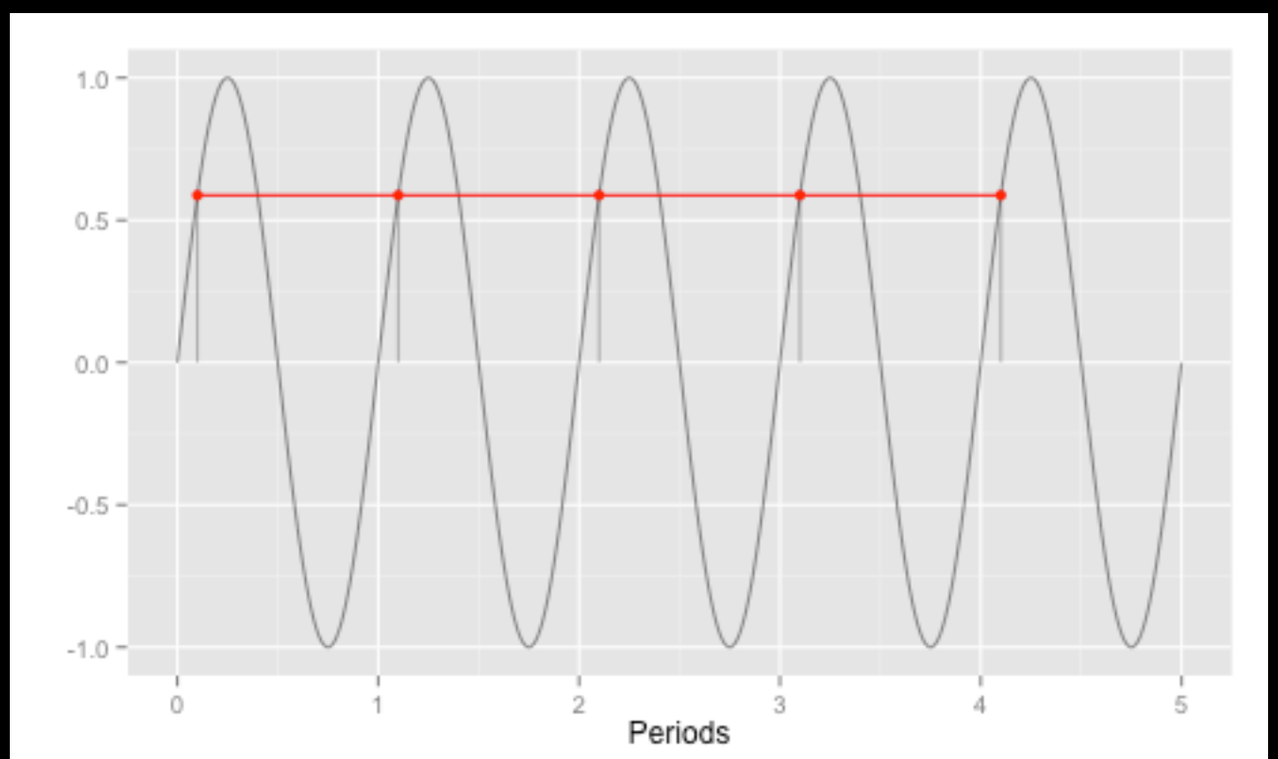


Orignal Analog

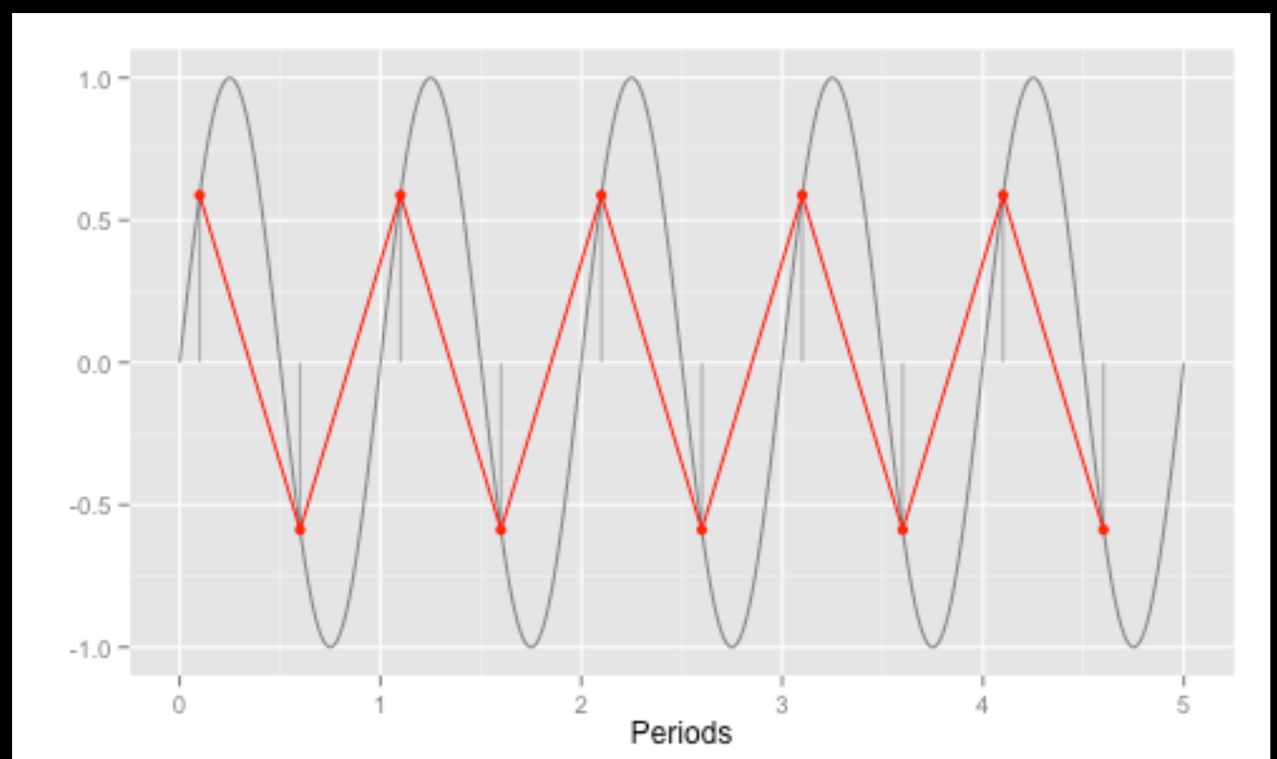


10 samples/period

Sampling

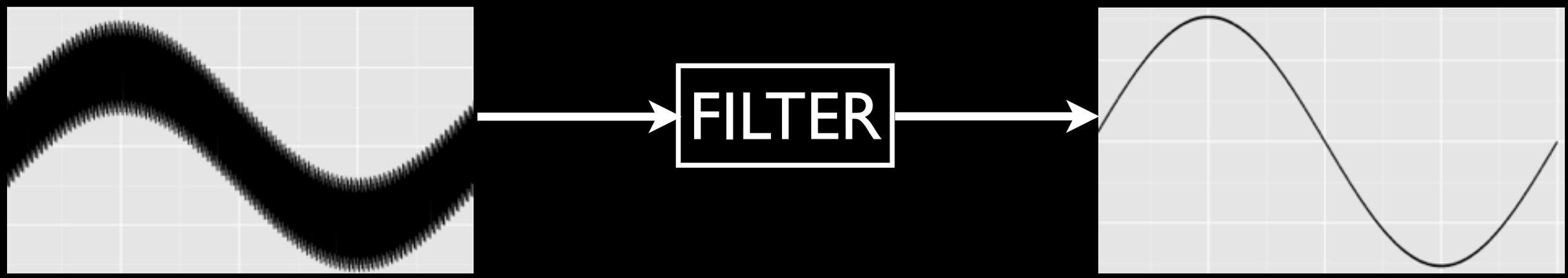


1 sample/period

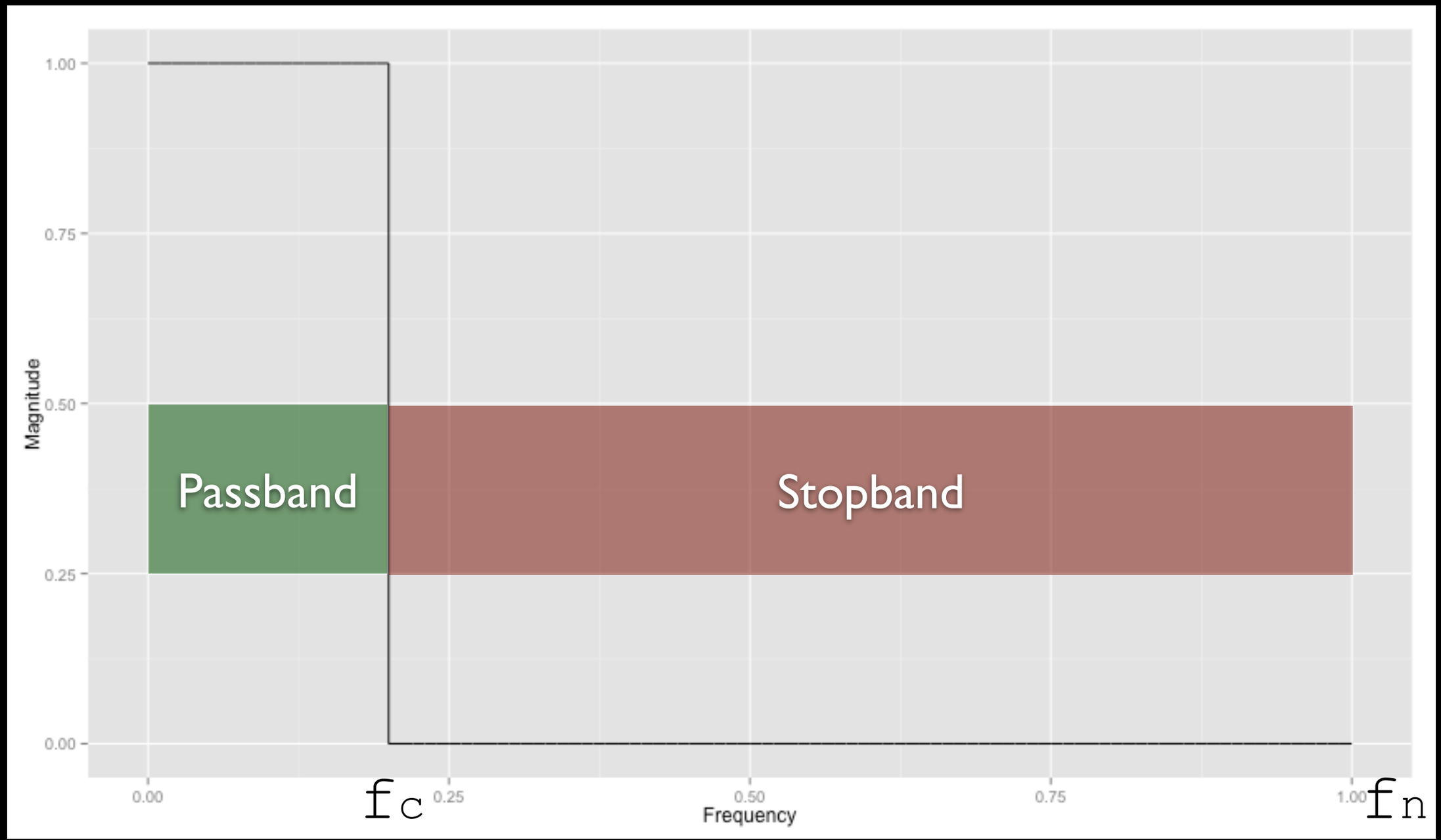


2 samples/period

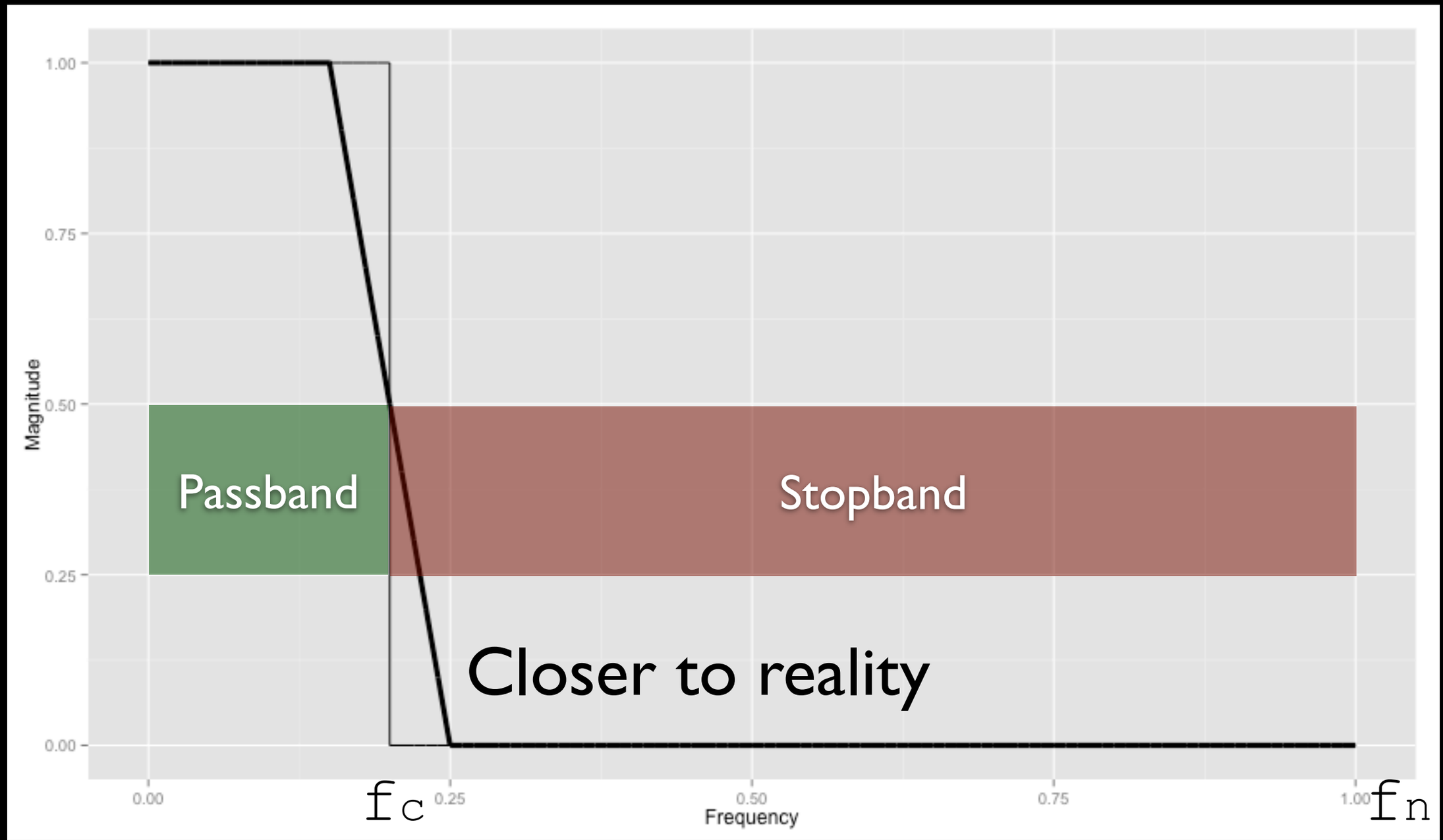
Filters


FILTER

Low-Pass Filter



Low-Pass Filter



Moving Average Filter

$$y_t = \sum_{i=0}^{N-1} \frac{1}{N} x_{t-i}$$

```
y[t] = 1/7 * x[t] +
       1/7 * x[t-1] +
       1/7 * x[t-2] +
       ...
       1/7 * x[t-6]
```

FIR Digital Filter

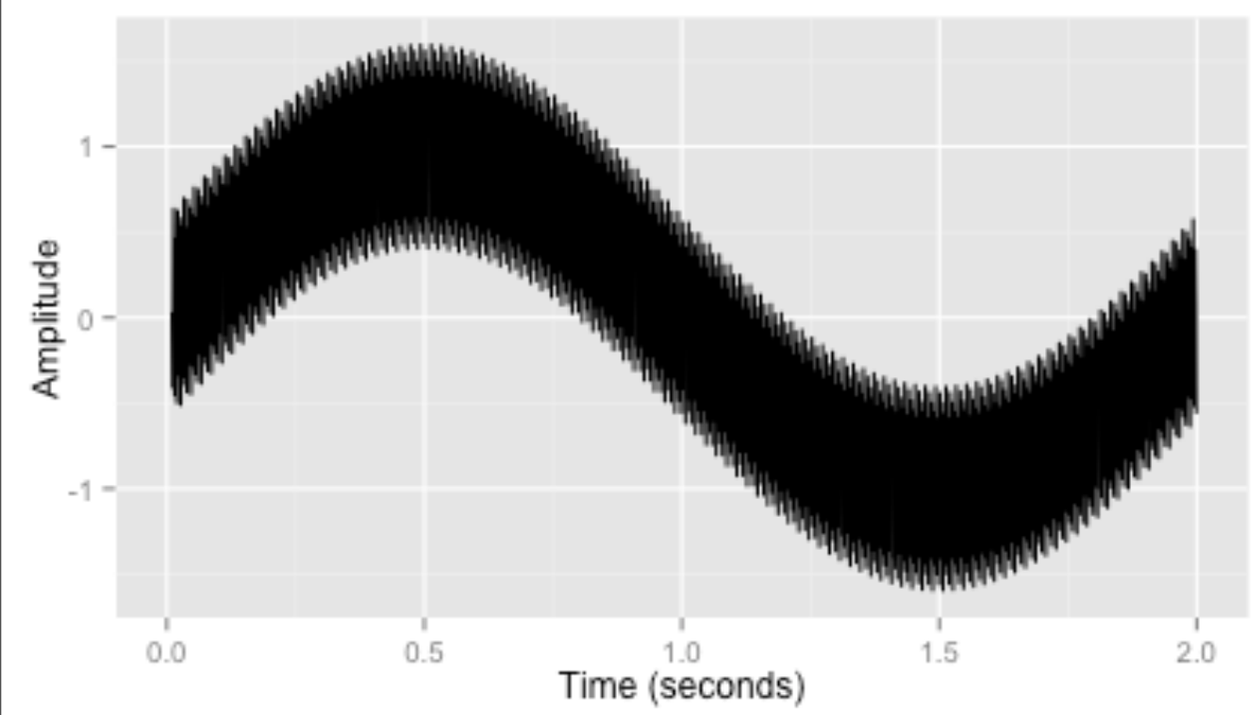
$$y_t = \sum_{i=0}^{N-1} h_i x_{t-i}$$

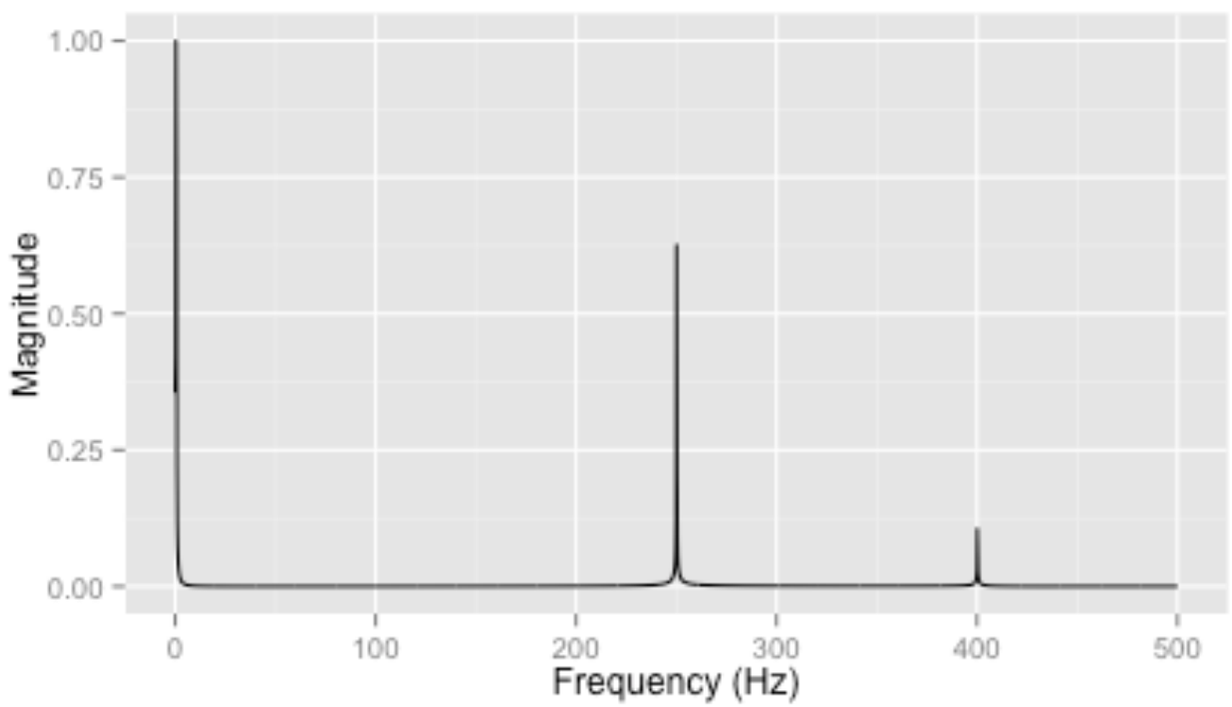
```
y[t] = h[0] * x[t] +
       h[1] * x[t-1] +
       h[2] * x[t-2] +
       ...
       h[N-1] * x[t-N-1]
```

R code:

```
y <- filter(x, h)
```

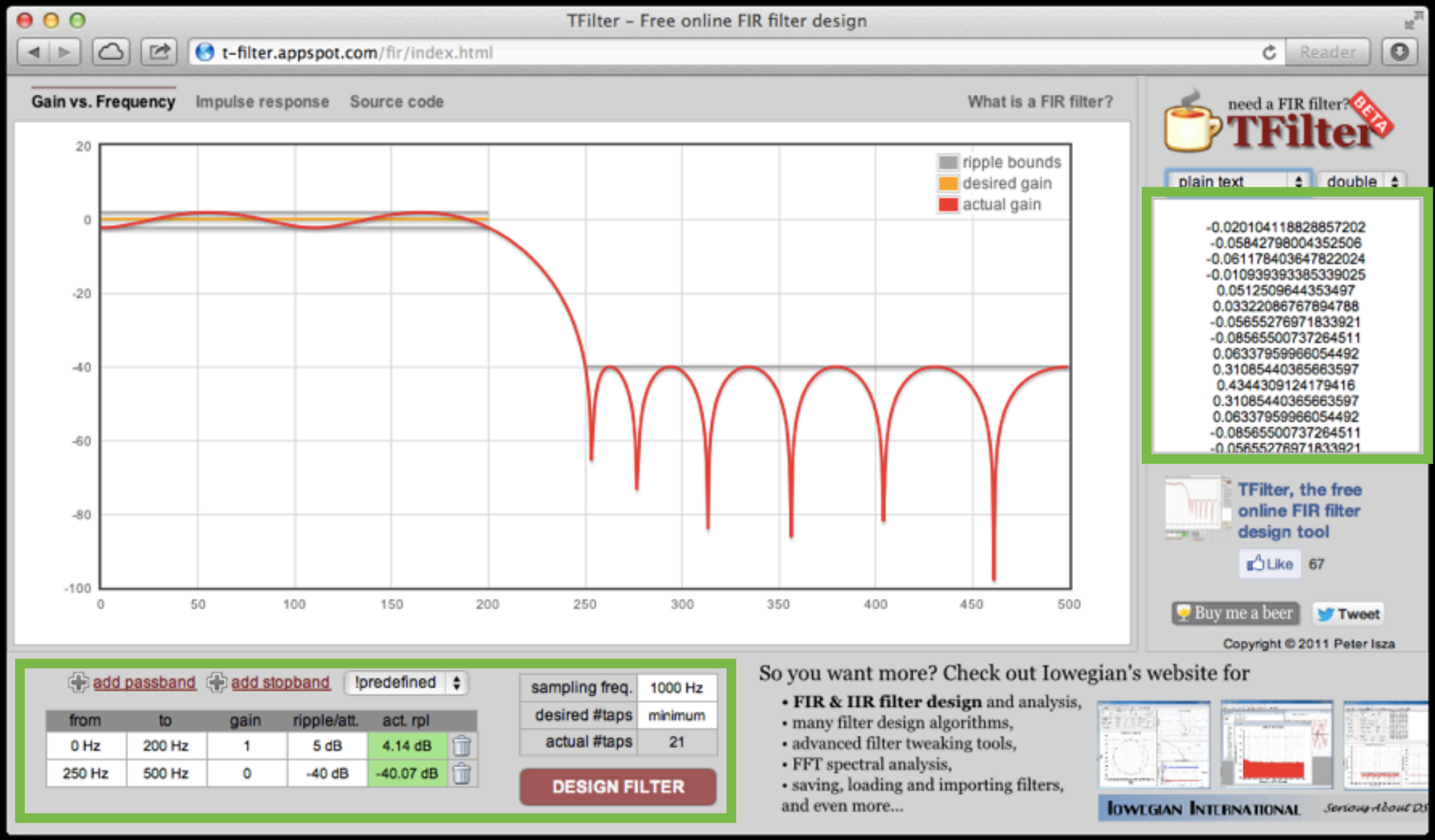
Frequency Domain



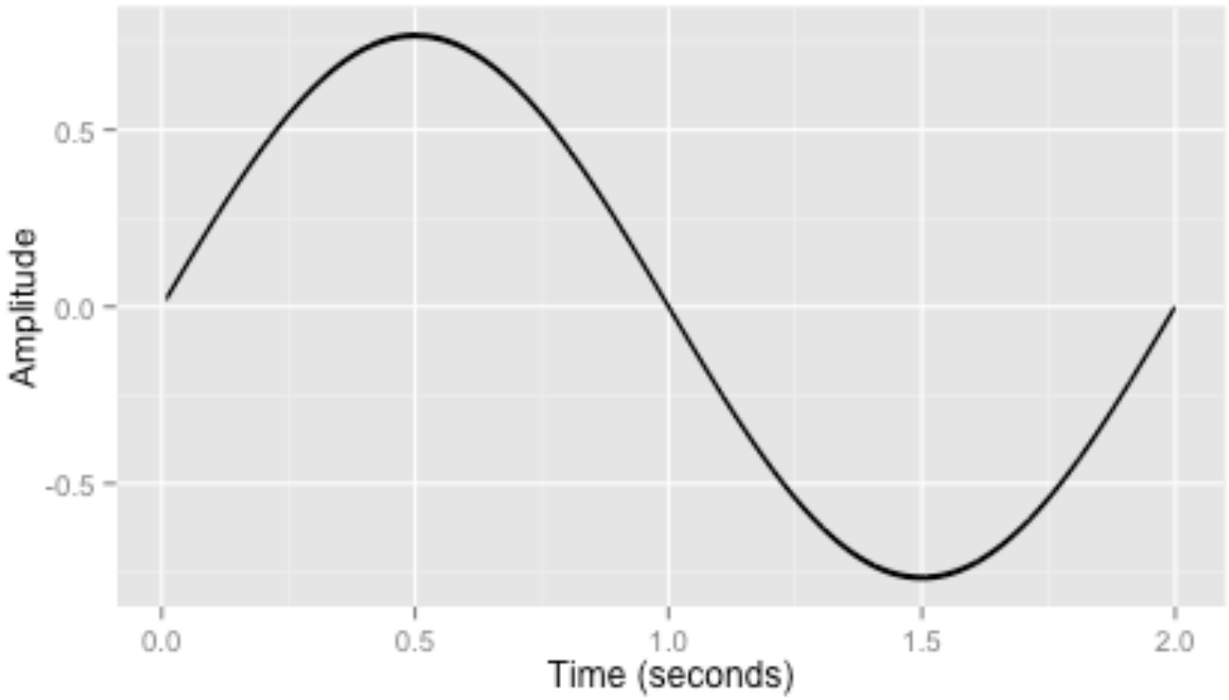


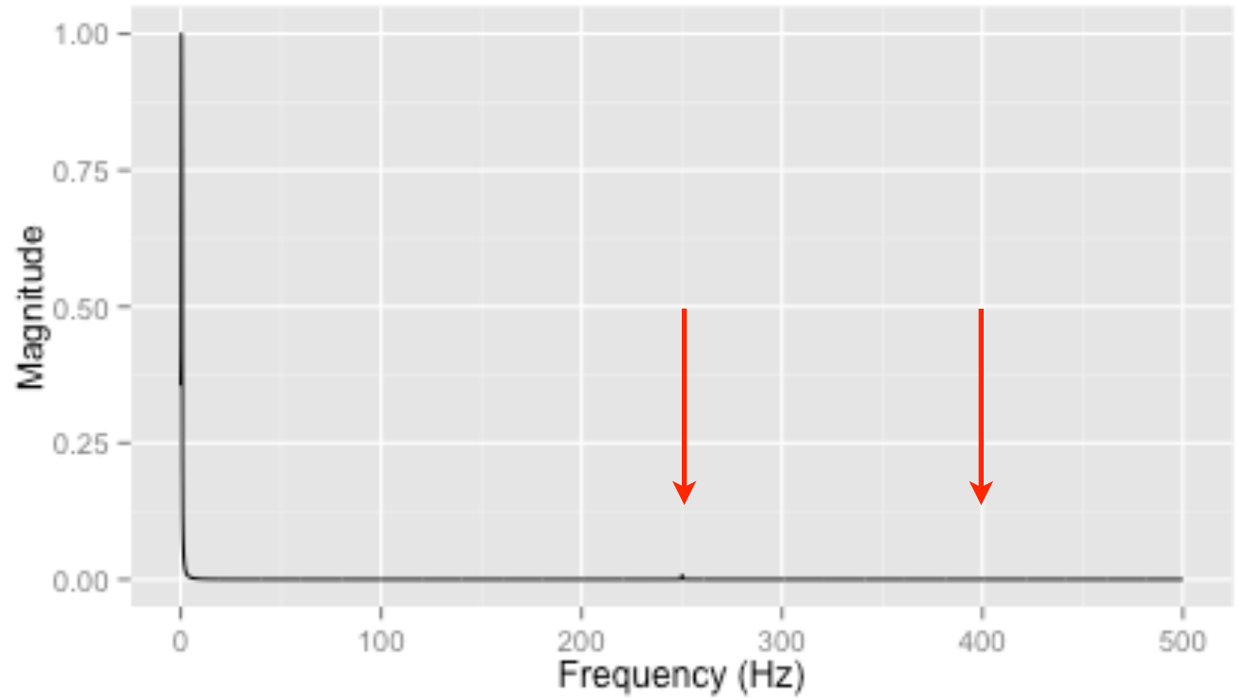

```
x = 1.0 * sin(0.5*2*pi*t) +
    0.5 * sin(250*2*pi*t) +
    0.1 * sin(400*2*pi*t)
```

Frequency Domain



Frequency Domain





21-point low-pass filter with 250Hz cutoff

```
h = [-0.0201, -0.0584, -0.0612, -0.0109,  0.0513,
      0.0332, -0.0566, -0.0857,  0.0634,  0.3109,
      0.4344,  0.3109,  0.0634, -0.0857, -0.0566,
      0.0332,  0.0513, -0.0109, -0.0612, -0.0584,
     -0.0201]
```

http://t-filter.appspot.com/fir/index.html

FIR vs IIR

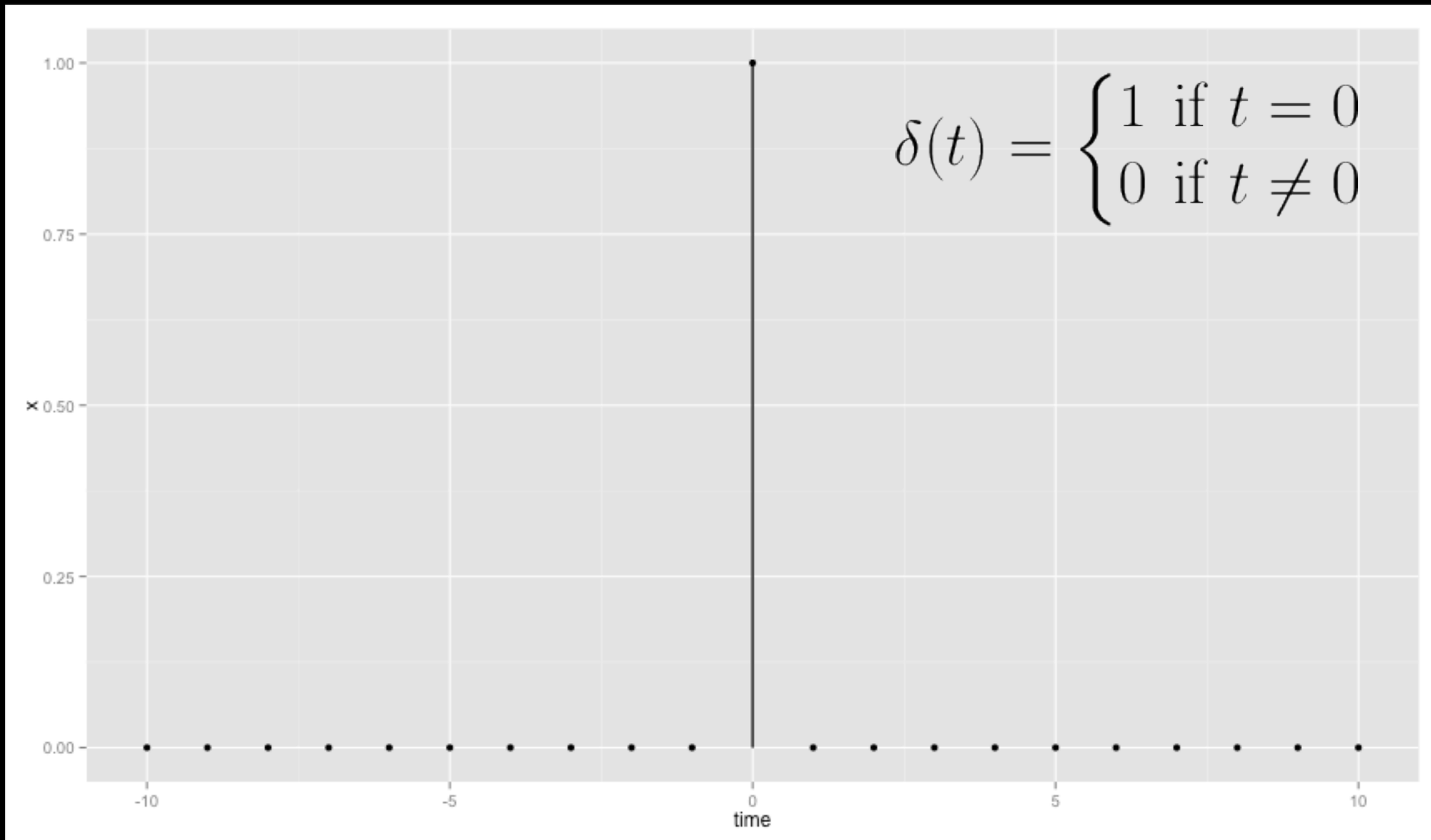
$$y_t = \sum_{i=0}^{N-1} h_i x_{t-i}$$

```
y[t] = h[0] * x[t] +
       ...
       h[N-1] * x[t-N-1]
```

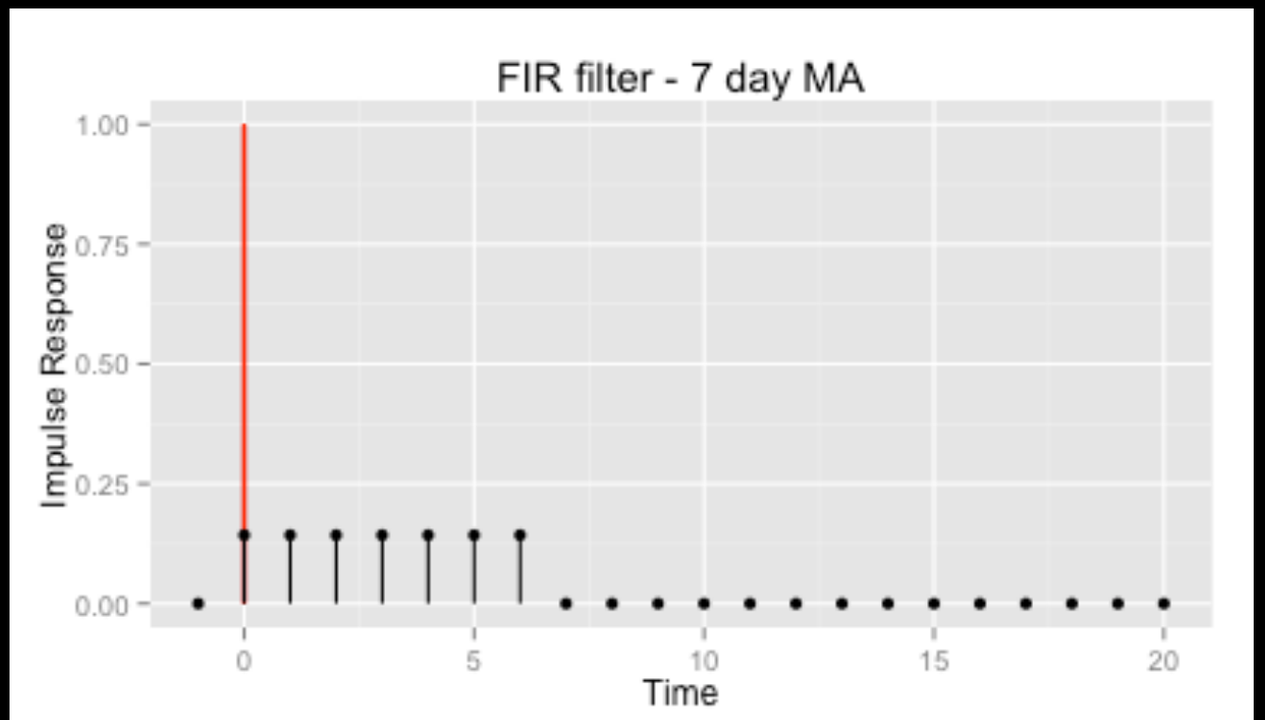
$$y_t = \sum_{i=0}^{N} h_i x_{t-i} - \sum_{j=1}^{M} g_i y_{t-i}$$

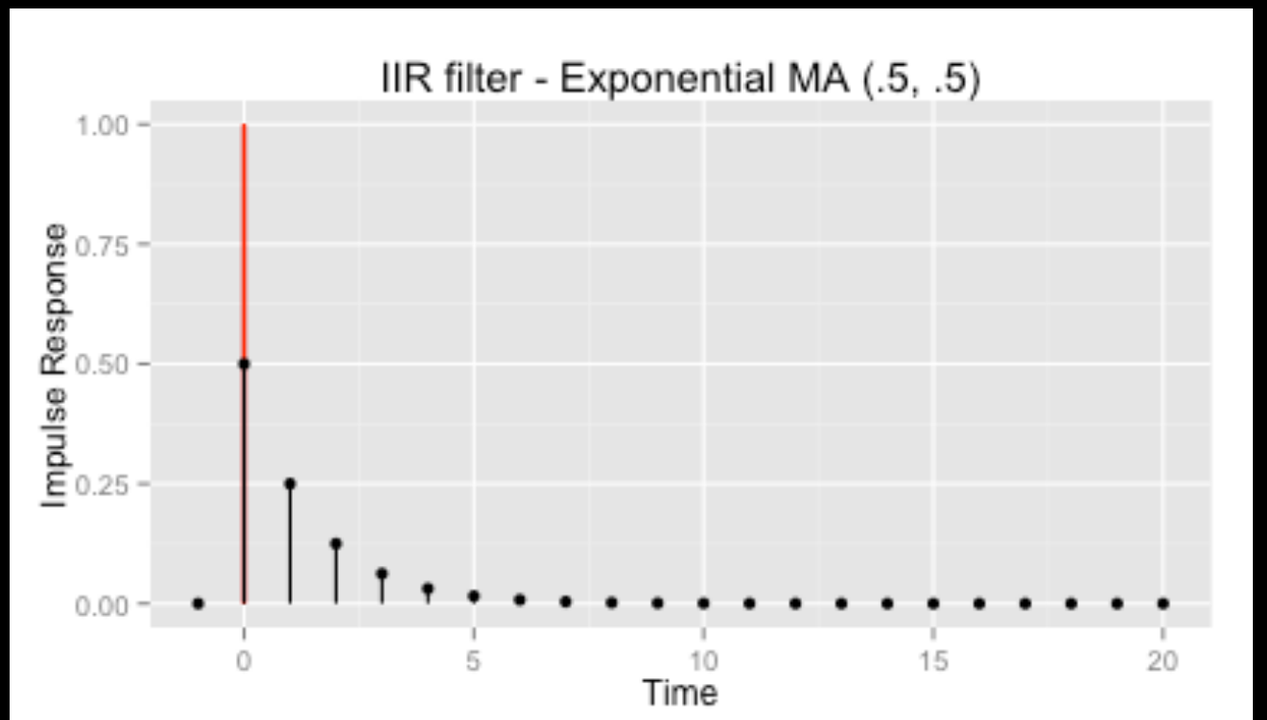
```
y[t] = h[0] * x[t] +
       ...
       h[N-1] * x[t-N-1] -

       g[1] * y[t-1] -
       ...
       g[M] * y[t-M]
```

Delta Function



FIR vs IIR

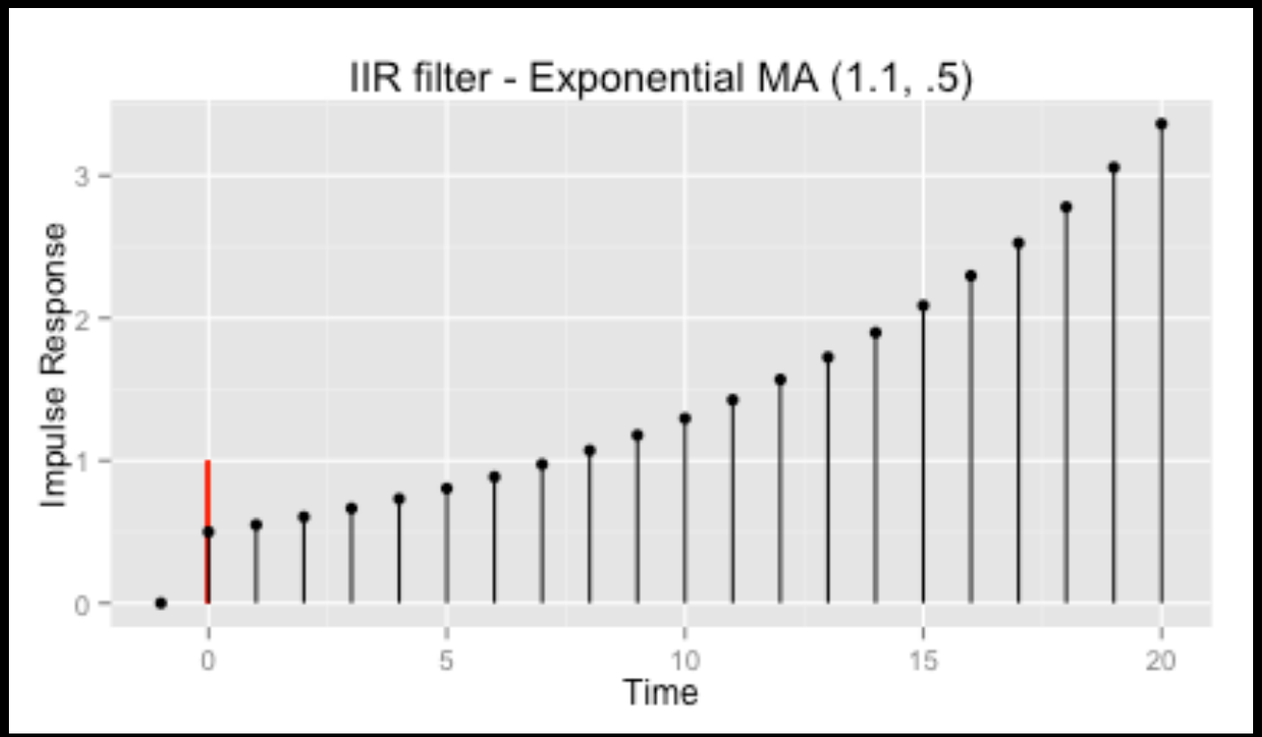





```
y[t] = 1/7 * x[t] +
       1/7 * x[t-1] +
       ...
       1/7 * x[t-6]
```

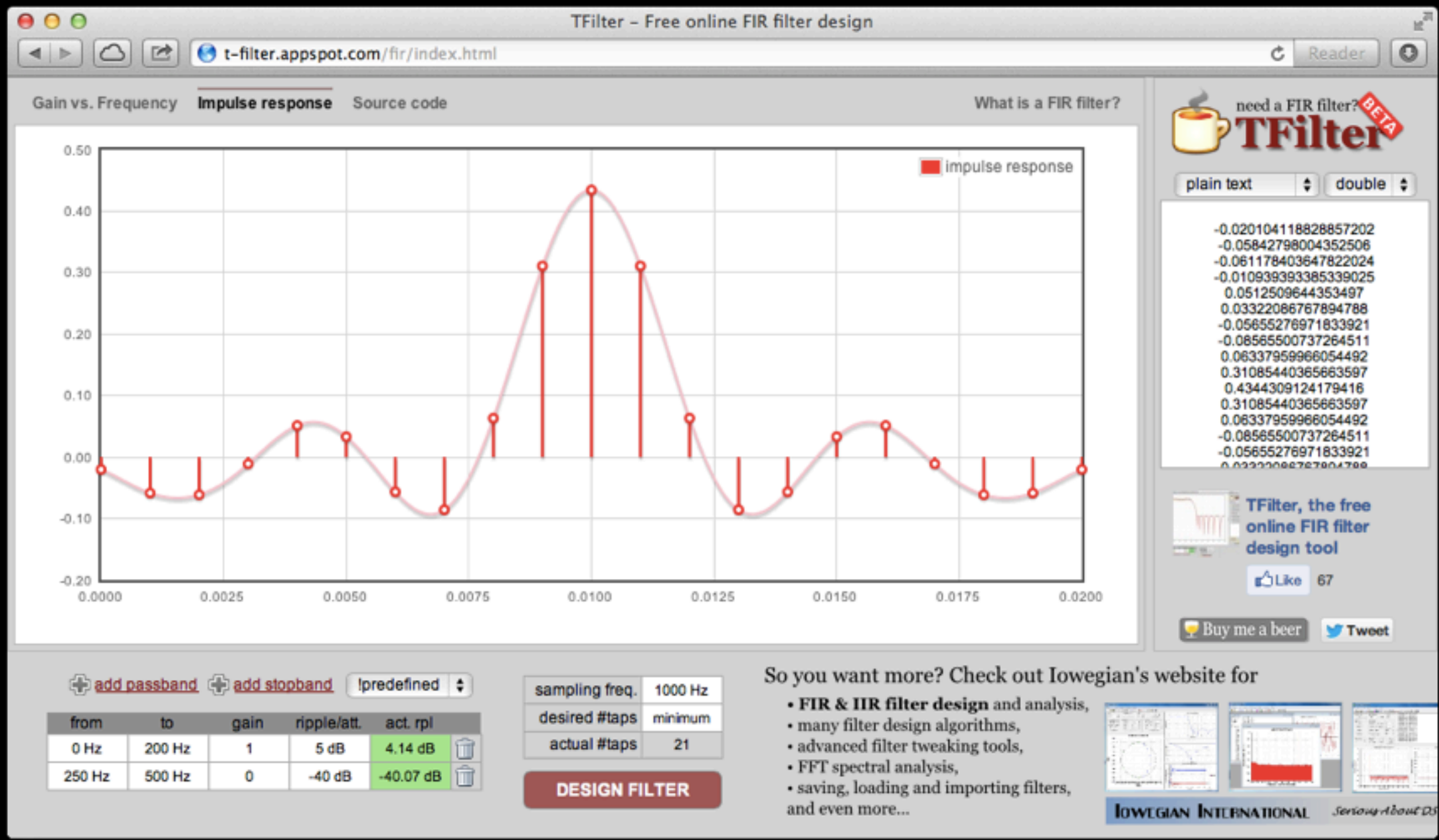
```
y[t] = 1/2 * x[t] +
       1/2 * y[t-1]
```

IIR - be careful!




```
y[t] = 0.5 * x[t] +
       1.1 * y[t-1]
```

Impulse Response



Recap - FIR Filters

- FIR filters are weighted sums of previous input.
- Can think of them as a generalized Moving Average
- Required to apply:
  - Filter `h` of length N
  - Previous `N` inputs `x`


Scalding

Super General Overview

- DSL on top of Cascading, written in scala
- Cascading: Workflow language for dealing with lots of data. Often in hadoop.
- Similar to pig or hive, but easier to extend (no UDFs! one language!).
- Feels like "real programming" - compiler! types!
- Is awesome

Less General Overview

- Similar to split/apply/combine paradigm (plyr, pandas)
- Load data into Pipes (like data.frames)
- Each pipe has one or more Fields (columns)
- Perform row-wise operations with `map` (d$a+d$b)
- Perform field-wise operations in `groupBy`

Hello World

```
1 package com.tumblr.jobs
2
3 import com.twitter.scalding._
4 import com.tumblr.lucille2._       // our internal library
5
6 class MyCoolJob(args : Args) extends Job(args) {
7     val posts = LoadPosts()
8         .groupBy('blogId) {group => group.size('postCount)}
9         .map('postCount -> 'fewPosts) {pc : Int => pc < 50}
10        .filter('fewPosts) {fp : Boolean => fp == false}
11        .project('blogId, 'postCount)
12        .write(Tsv("/results/postCounts"))
13 }
```

Scalding Resources

- The best resource is the Scalding wiki page.
  https://github.com/twitter/scalding/wiki/Fields-based-API-Reference

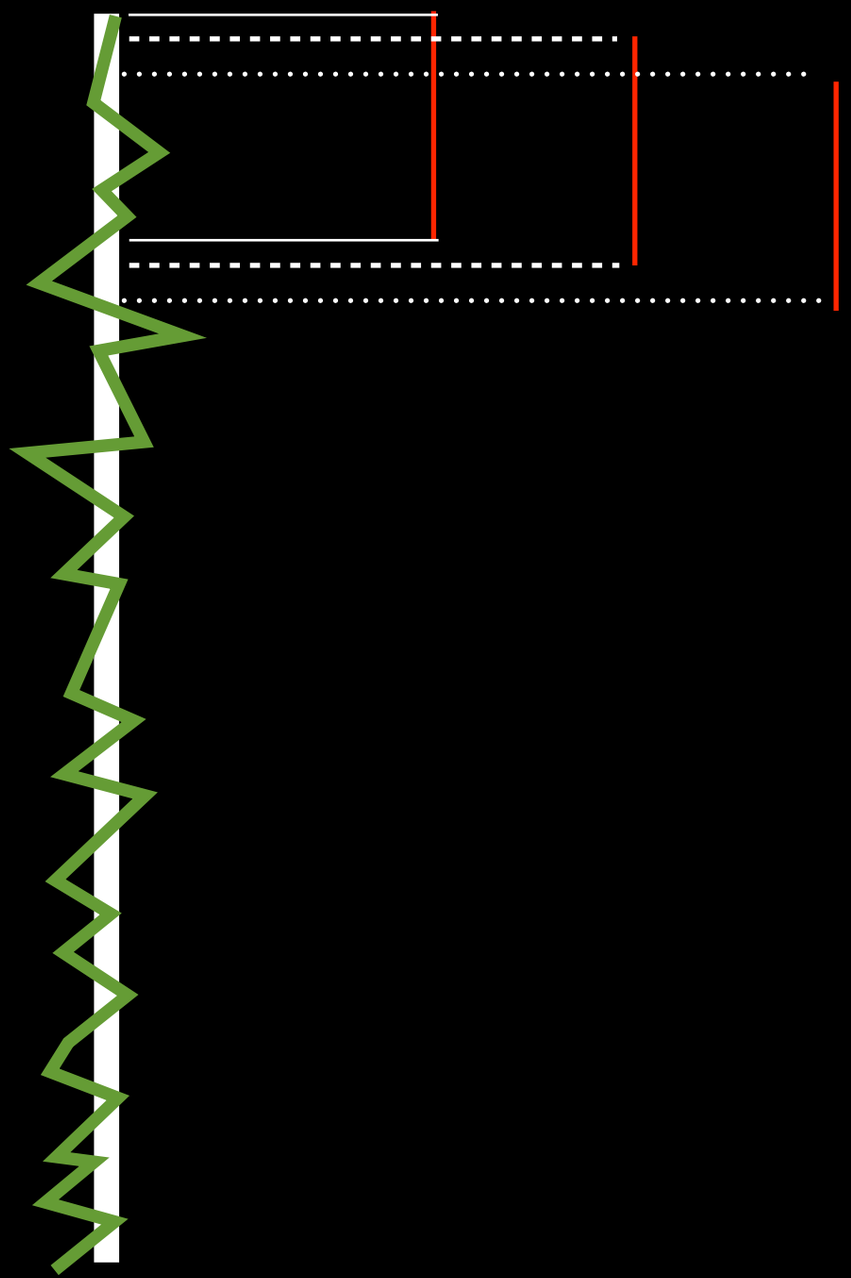
- Edwin Chen's post about recommendations.
  http://blog.echen.me/2012/02/09/movie-recommendations-and-more-via-mapreduce-and-scalding/

- Source code is FULL of undocumented features!
  https://github.com/twitter/scalding

```
import Matrix._
```

Data Vector

Sliding Filter

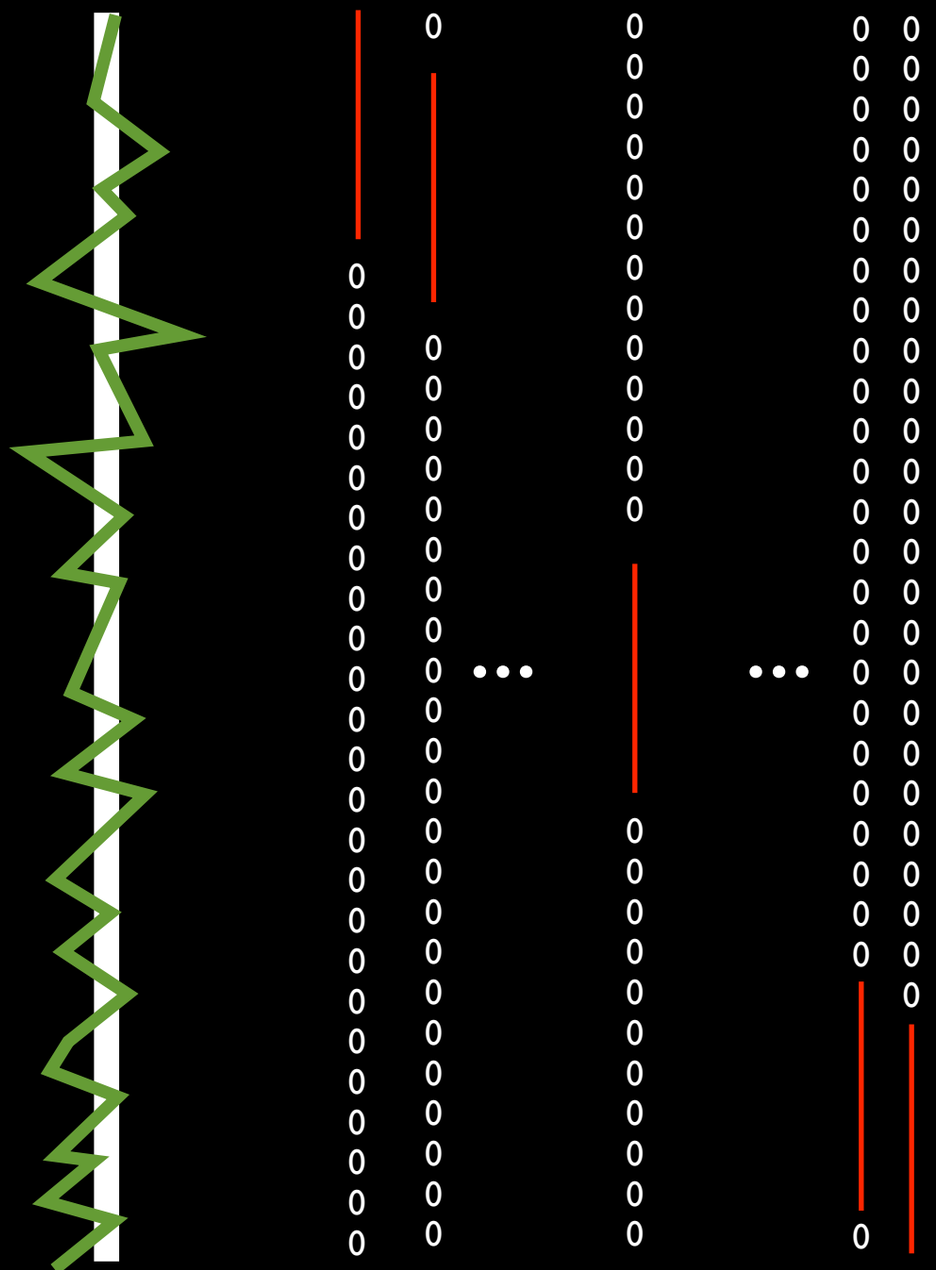


$$y = \sum_{i=0}^{N} h_i \widehat{x}_i = \mathbf{h} \cdot \widehat{\mathbf{x}}$$

$\widehat{\mathbf{x}}$ = Sliding subset of input

Data Vector

Sliding Filter



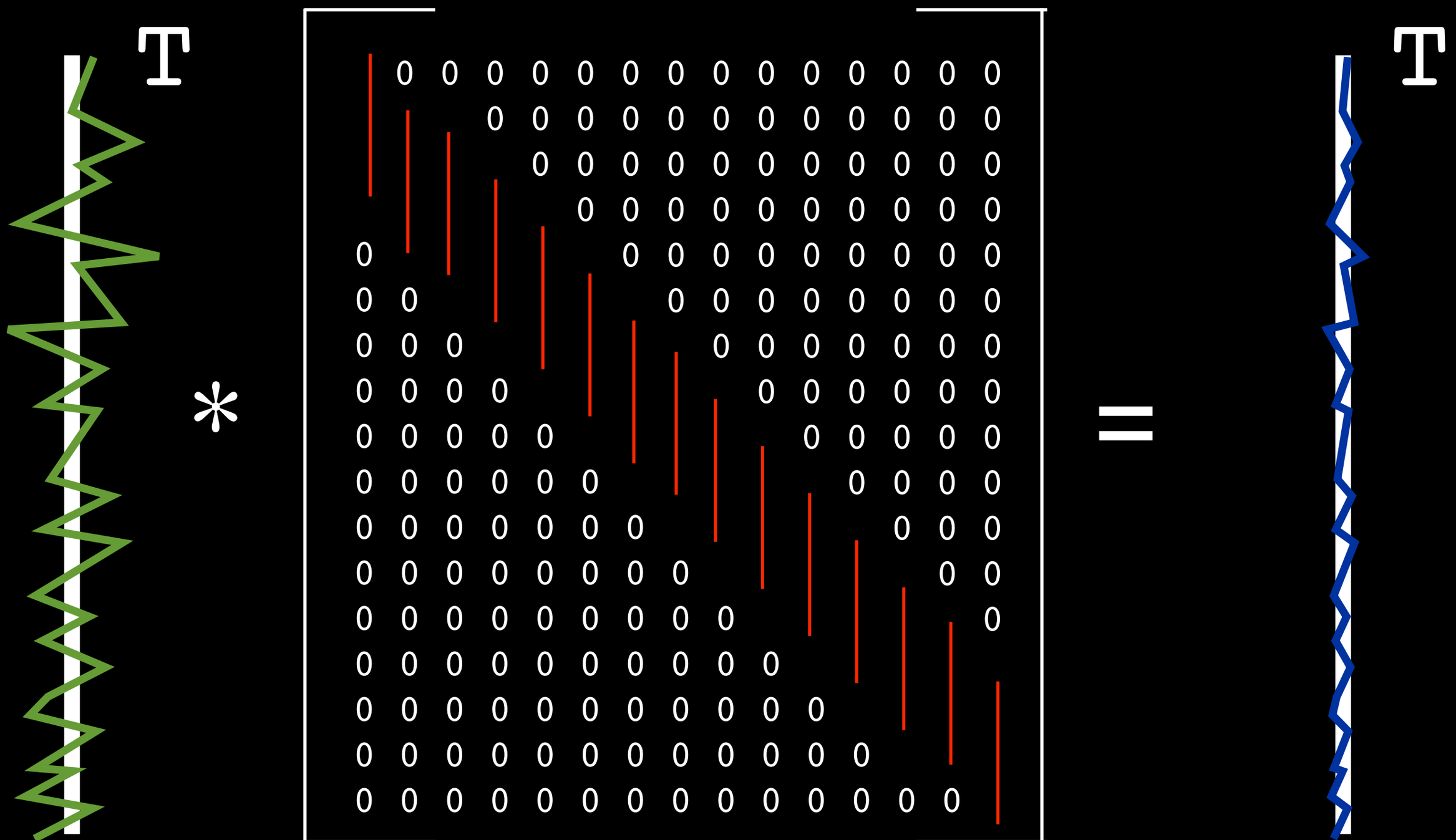
$$y = \sum_{i=0}^{M} \hat{h}_i x_i = \hat{\mathbf{h}} \cdot \mathbf{x}$$

$\hat{\mathbf{h}}$ = Sliding filter

Data Vector

Filter Matrix

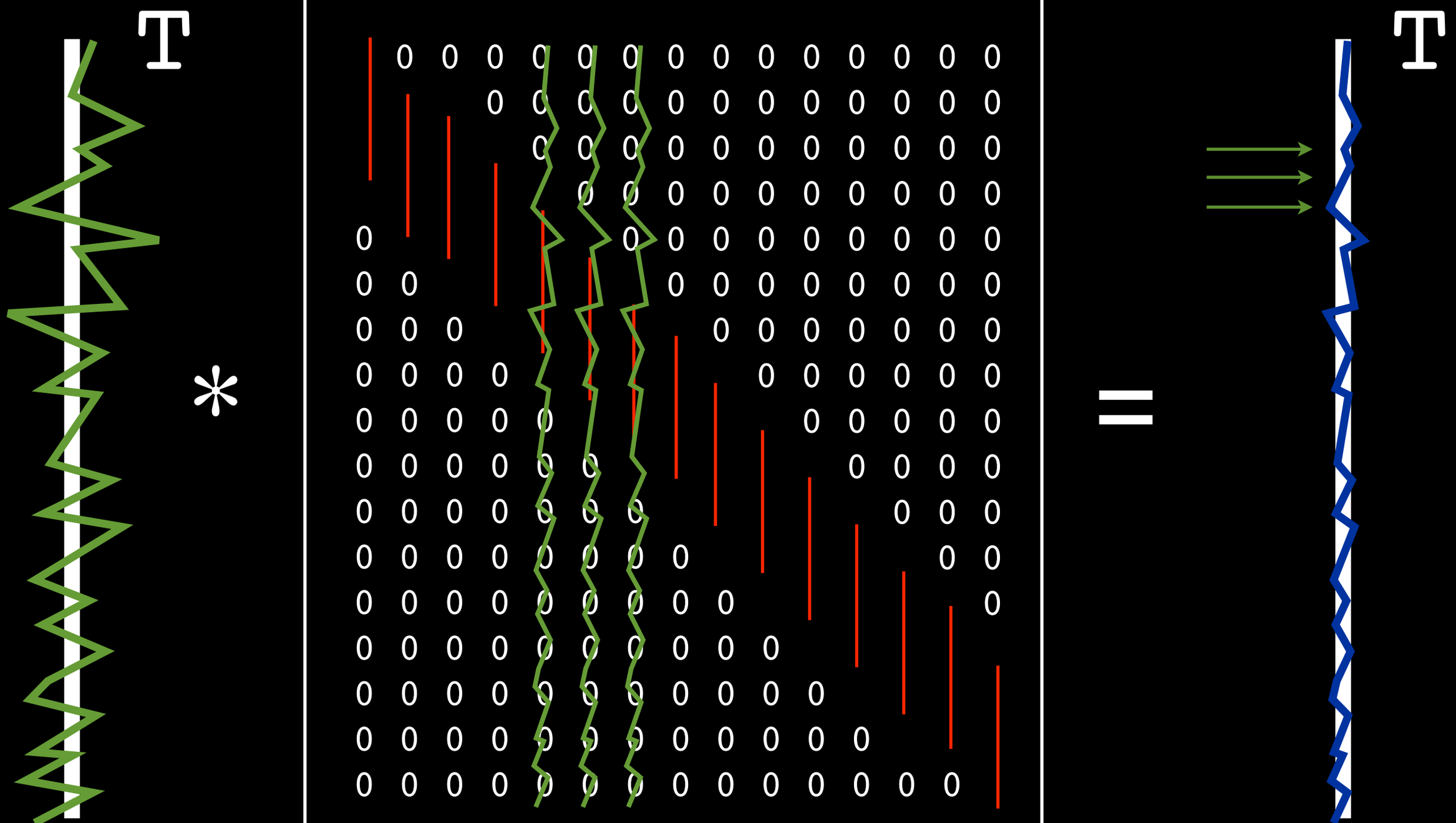
Filtered Output



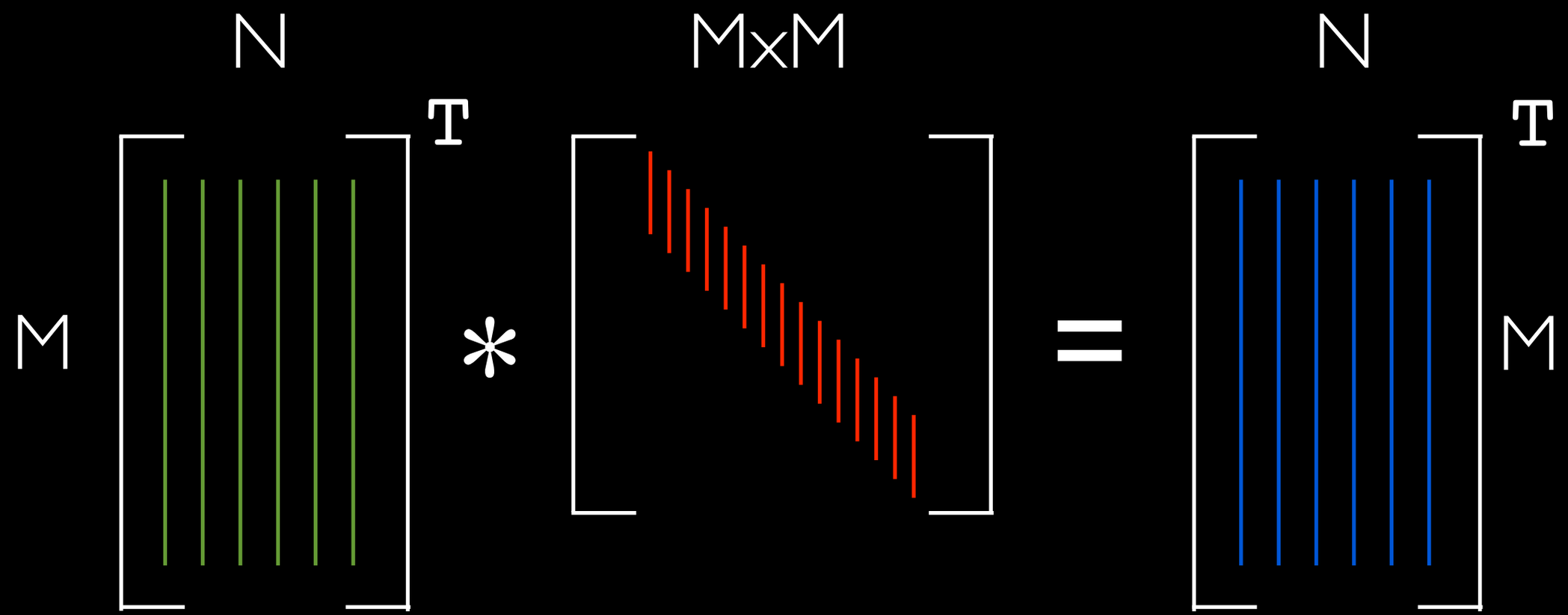
Data Vector

Filter Matrix

Filtered Output



$$X^T * H = Y^T$$

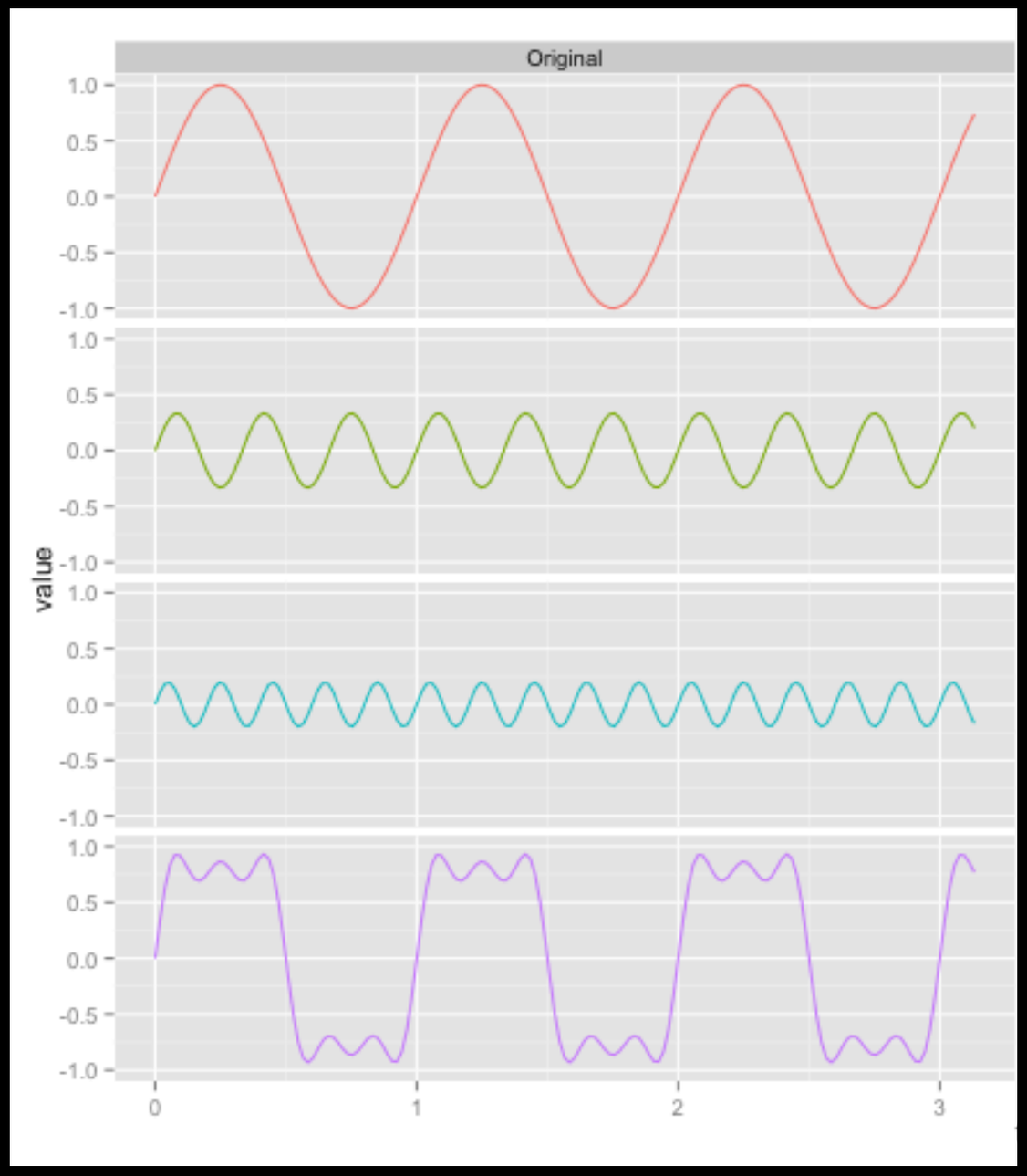


N = number of blogs
M = number of samples

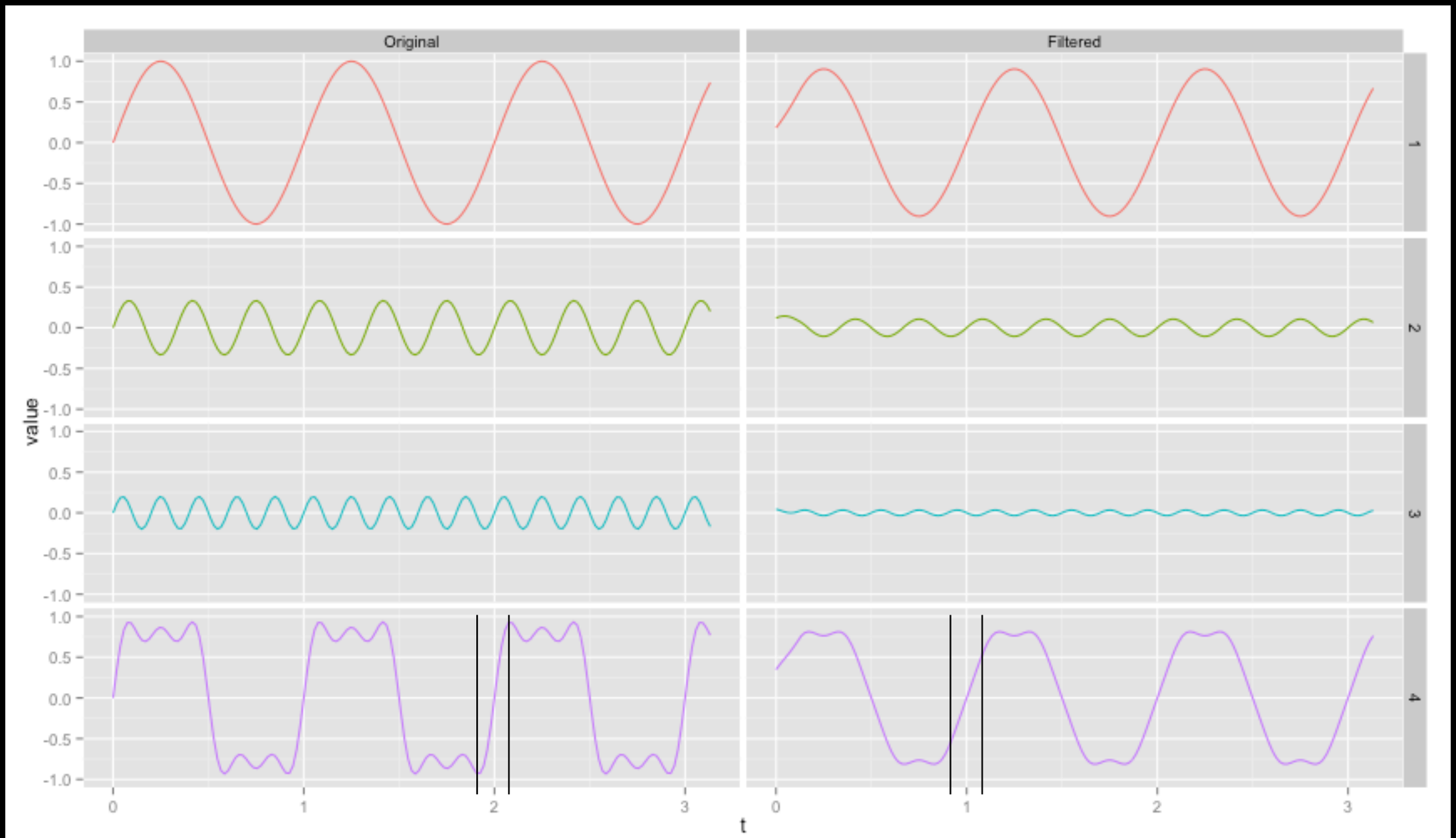
Matrix Filter: Square Waves

```r
t <- seq(0, 6*pi, length.out=1000)

x <- data.frame(
  x1 = sin(2*pi*t),
  x2 = 1/3 * sin(3 * 2*pi*t),
  x3 = 1/5 * sin(5 * 2*pi*t)
)
x$x4 <- with(x, x1+x2+x3)  # square wave!

X <- as.matrix(x)

H <- toeplitz(c(rep(1/7, 7), rep(0, nrow(x)-7)))

Xf <- t(t(X) %*% H)
```

Matrix Filter: Square Waves



Matrix Filter: Square Waves



`import Matrix._`

- Scalding has a Matrix library!
- Stores data in a Pipe as `('row, 'col, 'val)`
- Ideal for sparse matricies

- L0, L1, L2 norm, inverse, +, -, *
- QR Factorization: http://bit.ly/1hxWF17
- More!

Tumblr Social Graph

- 140+ Million Nodes
- 3.5 Billion Edges
- About 100GB of raw text data
- 3 columns: `fromId, toId, timestamp`

GOAL: Calculate followers / day for every blog, apply 1-week moving average.

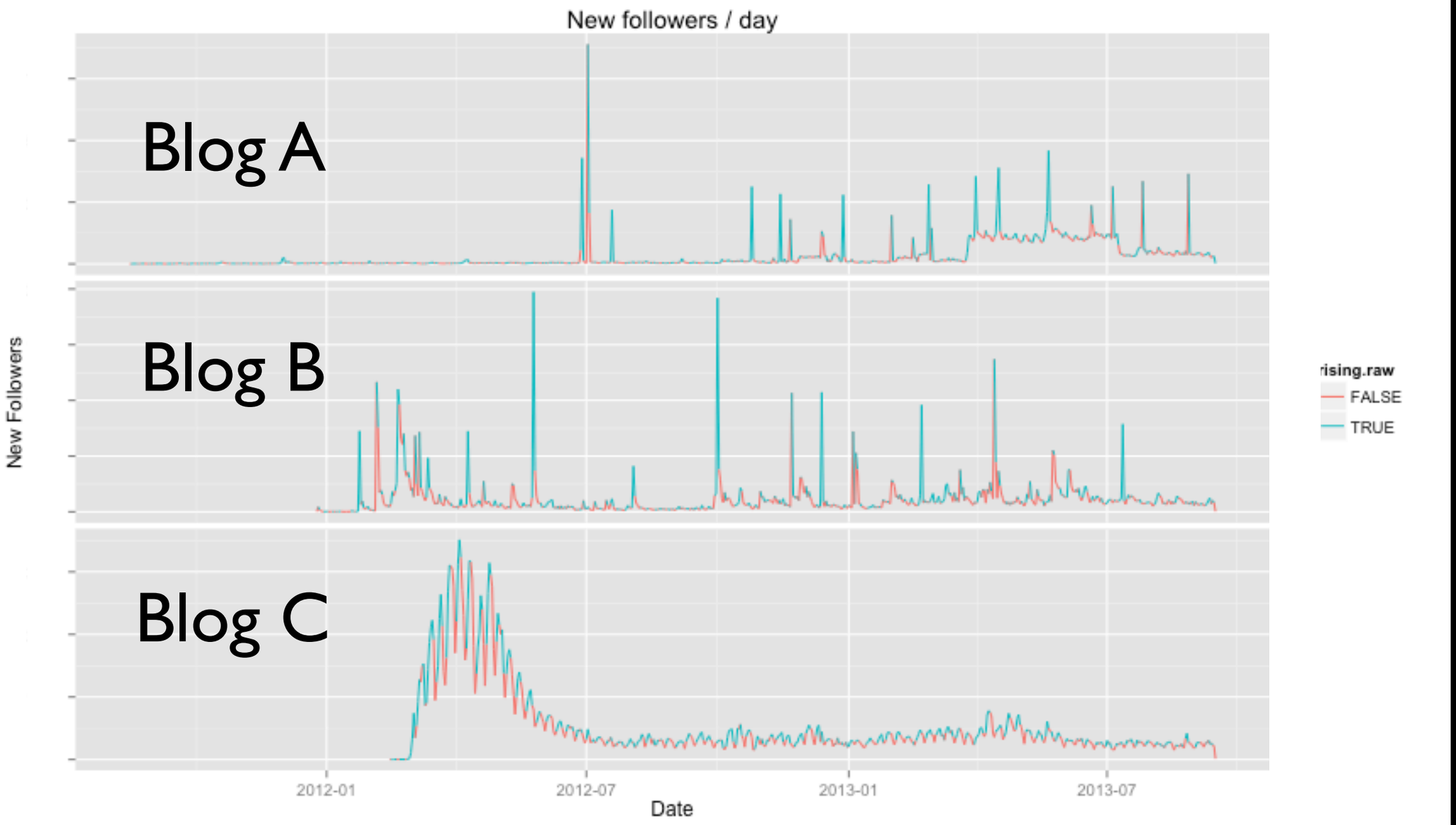
Apply Low-Pass filter to 140,000,000 blogs

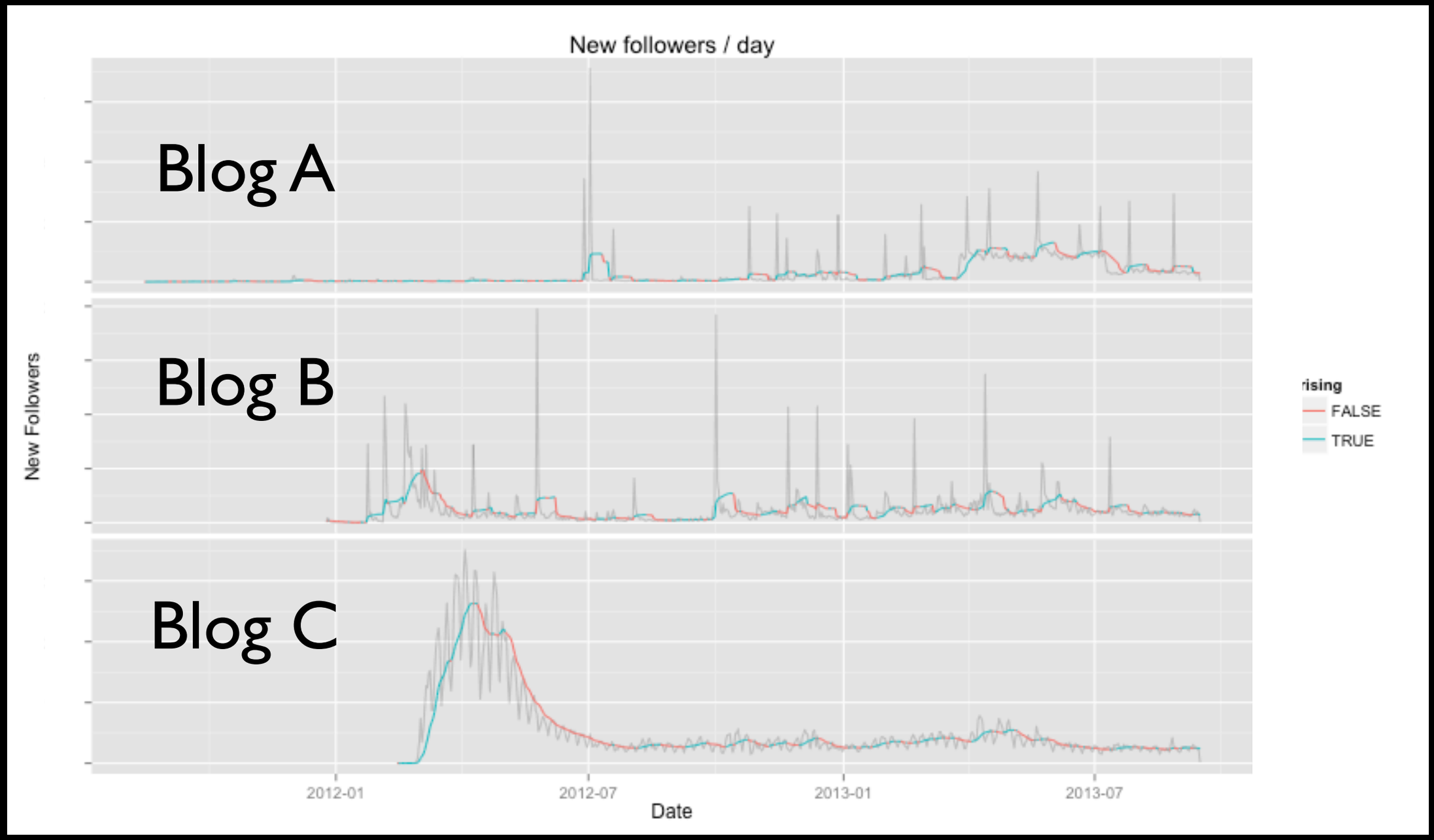
```
1 // Load followers table, count # of followers / day for the past year.
2 val timeWindow = 365   // days
3 val follows = FollowersTable()
4   .filter('ts) {ts : Time => ts >= timeWindow.days.ago}
5   .map('ts -> 'dt) {ts : Time => ymdFormat.format(ts)}
6   .groupBy('toId, 'dt) {_.size('follows)}
7   .toMatrix[String, Int, Double]('dt, 'toId, 'follows)   // (row, col, value)
8
9 // Build the filter matrix
10 val maFilter = Seq.fill[Double](10)(1/7)
11 val filter = toeplitz(maFilter, timeWindow)
12
13 // Apply filter and write results.
14 val result = (follows * filter).transpose
15   .write(Tsv("/path/to/results"))
```

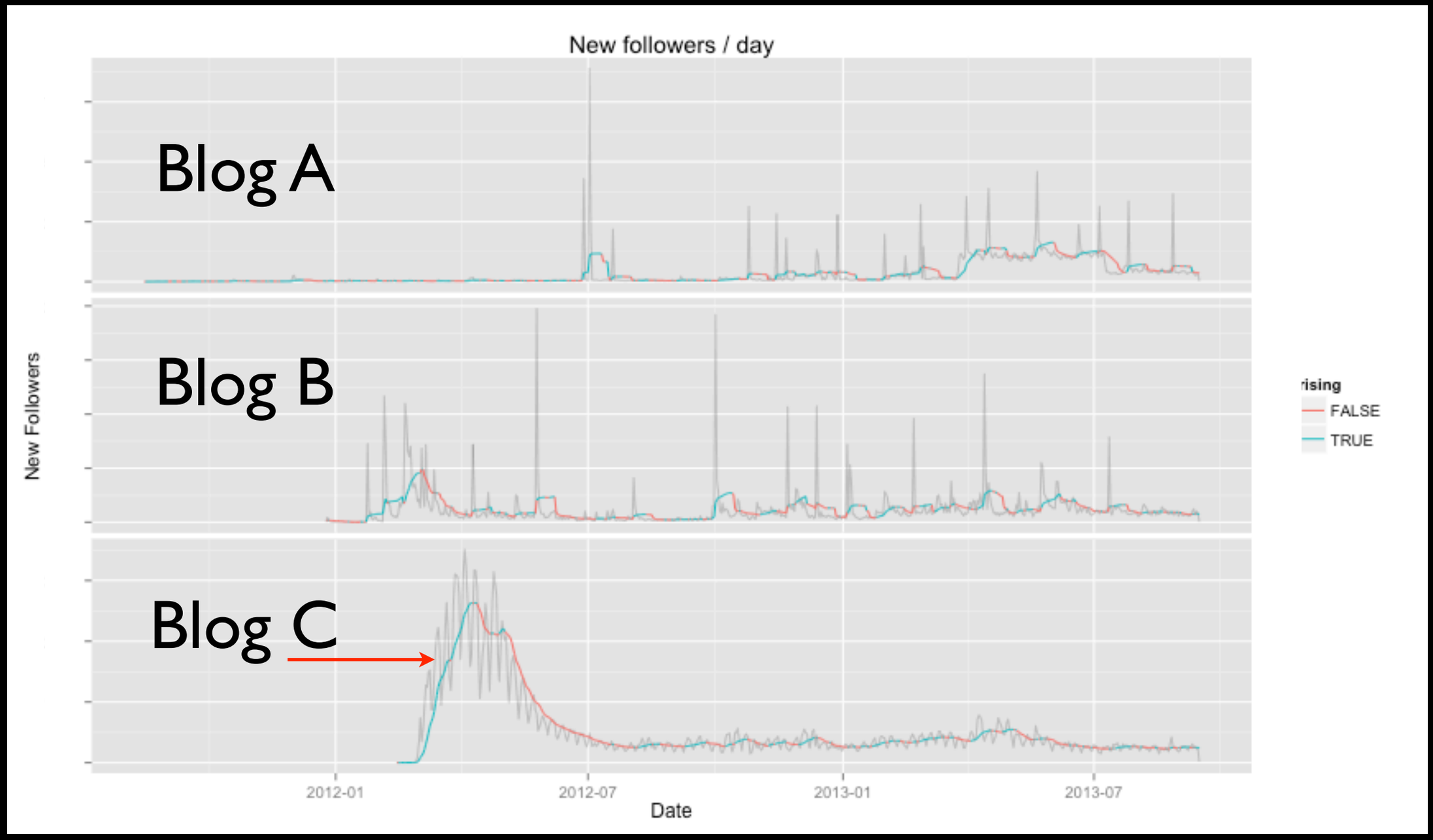
Find blogs who have accelerating follower counts for the most consecutive days.

"Accelerating":
New Followers Today > *New* Followers Yesterday

$$followers = x$$
$$growth = \frac{dx}{dt}$$
$$acceleration = \frac{d^2x}{dt^2}$$


New followers / day
Blog A
Blog B
Blog C
New Followers
2012-01
2012-07
2013-01
2013-07
Date
rising.raw
FALSE
TRUE


New followers / day
Blog A
Blog B
Blog C
New Followers
rising
FALSE
TRUE
2012-01
2012-07
2013-01
2013-07
Date

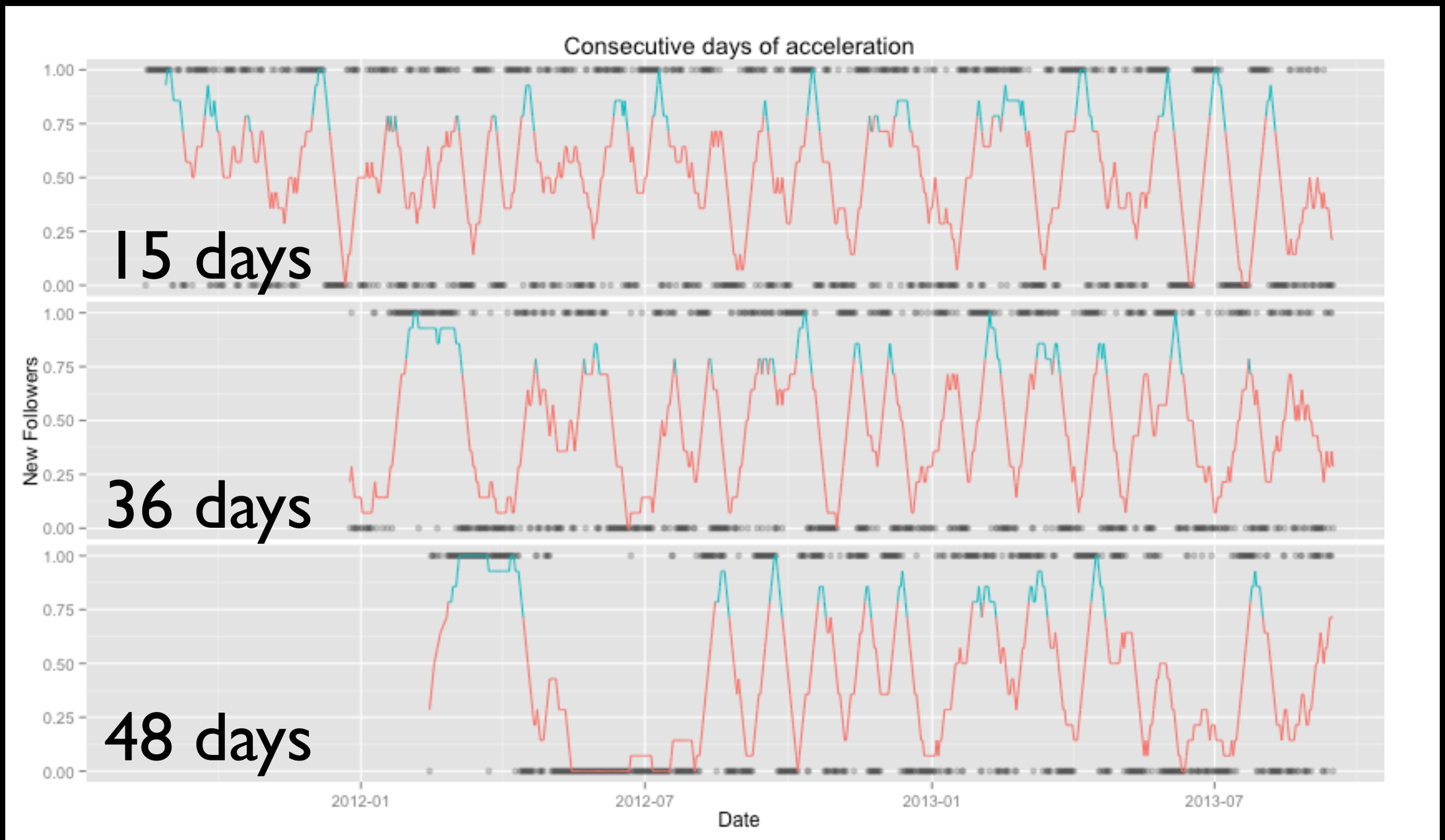

New followers / day
Blog A
Blog B
Blog C
New Followers
rising
FALSE
TRUE
2012-01
2012-07
2013-01
2013-07
Date

Days of consecutive acceleration

- Binary input: 1 if more followers today than yesterday, otherwise 0
- Filter the binary signal to produce a value between 0 and 1
- Anything above a threshold (0.75) is “accelerating”

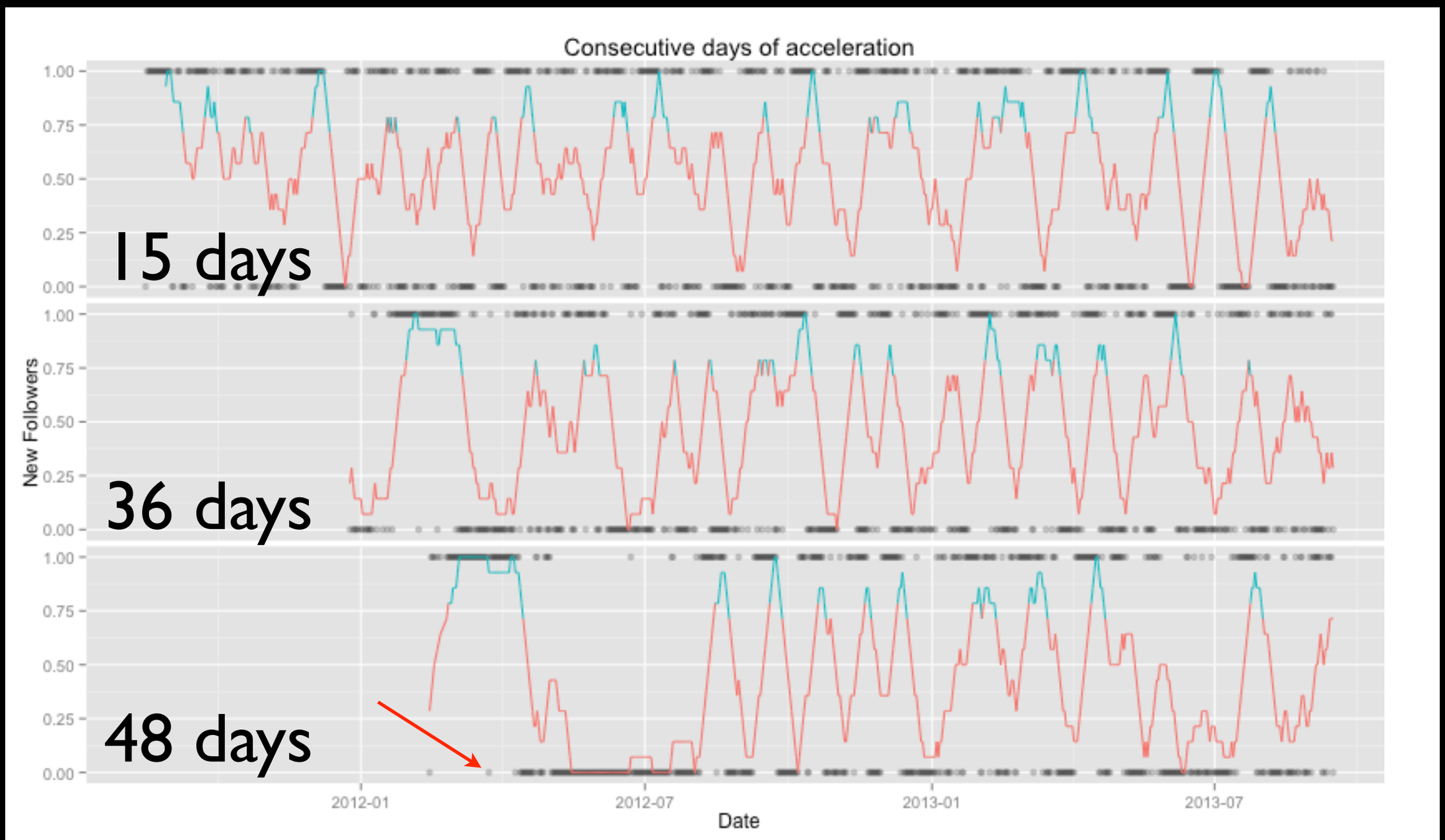
Days of consecutive acceleration

(filtered signal)



Days of consecutive acceleration

(filtered signal)



Thanks!

@adamlaiacano

adamlaiacano.tumblr.com

github.com/alaiacano/dsp-scalding