Multidimensional Cointegration

Oliver Schwartz, Princeton University Daniel Gitelman, Princeton University

4/28/2021

Getting data

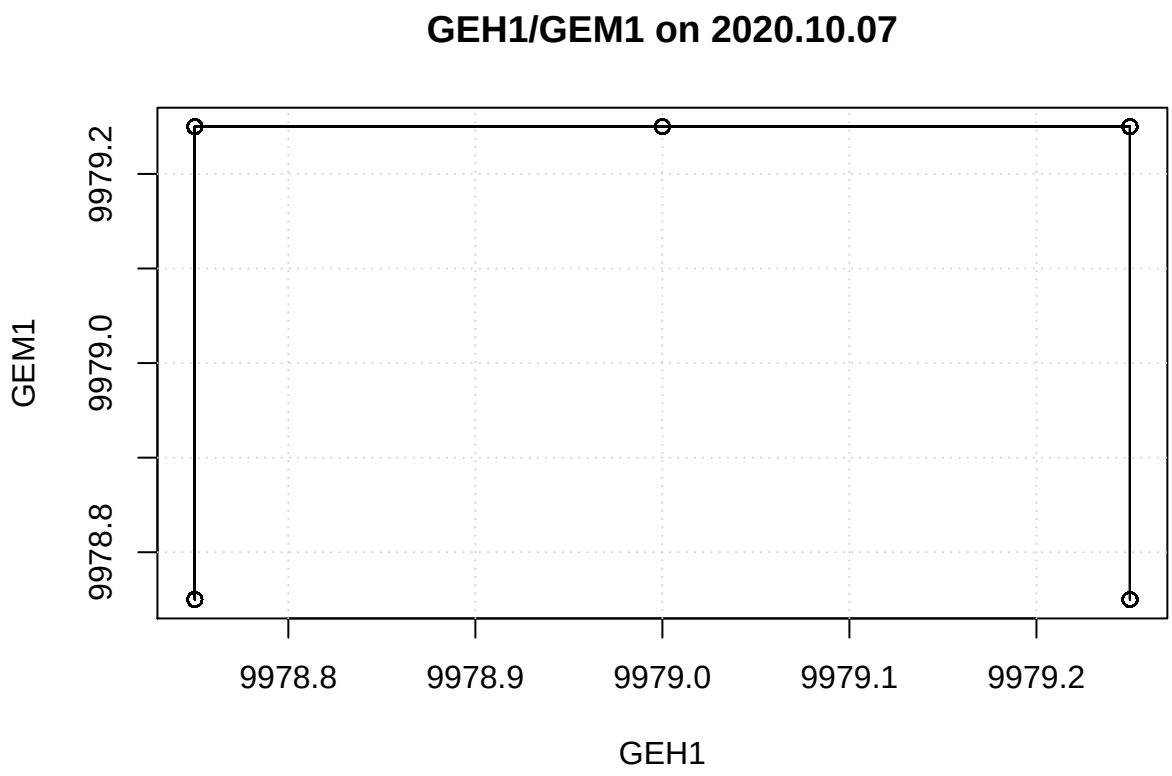
First, we will choose a set of 8 closely related products. We will use Eurodollars maturing March 2021 through to December 2022 - i.e. `GEH1, GEM1, GEU1, GEZ1, GEM2, GEU2, GEZ2`.

We sample the quote midpoint prices at 1-minute intervals for the entire 23-hour trading day, for any day in October 2020 (our data set).

```
# Constants.
INST <- "GE"
SYMS <- c()
for (dvy in c("H1", "M1", "U1", "Z1", "H2", "M2", "U2", "Z2")) {
  SYMS <- c(SYMS, paste0(INST, dvy))
}
NSYM <- length(SYMS)
DATE <- "2020.10.07"
NSEC <- 23 * 3600
TMIN <- "-0D07:00:00"

# Get midpoint data for all symbols.
mid_by_sym <- list()
for (sym in SYMS) {
  q1 <- sprintf("([] time:%s+1000000000 * til %d)", TMIN, NSEC)
  q2 <- sprintf(
    "select time, mid:0.5*(bid+ask) from quote where sym=`%s, date=%s",
    sym, DATE
  )
  q3 <- sprintf("aj[`time; %s; %s]", q1, q2)
  mid_by_sym[[sym]] <- try_execute(q3)
}

# Get the ticksize.
TICKSZ <- try_execute(paste0("select minpxincr from instinfo where inst=`",
                             INST))$minpxincr

# For convenience, let's create a matrix of midpoint prices.
mids <- matrix(data=NA, nrow=NSEC, ncol=NSYM)
for (i in 1:NSYM) {
  mids[,i] <- mid_by_sym[[SYMS[i]]]$mid
}
```

Let's plot a few symbols to visualize the price relationship.

```
plot(mids[,1], mids[,2], type="o",
     main=sprintf("%s/%s on %s", SYMS[1], SYMS[2], DATE), xlab=SYMS[1], ylab=SYMS[2])
grid()
```



Rolling statistics

Our assumption is that all 8 prices move back and forth along some subspace, for instance a 1-dimensional line or a 2-dimensional plane. By introducing more degrees of freedom, we model the prices as moving along a higher dimensional subspace in $\mathbb{R}^8$.

First, we must compute exponential rolling averages using a 1-hour window time. At a time n, the exponential average is given by A_n/B_n where:

$$A_n = \sum_{j=1}^{n} e^{-(n-j)/m} x_j$$

$$B_n = \sum_{j=1}^{n} e^{-(n-j)/m}$$

We adjust m according to our desired window size. For simplicity, we will use 1 hour. The smaller this value, the quicker the weights decay, and the less past values influence the exponential average.

This yields the following recursive update rules, which makes this computation constant at every time step,

i.e. O(1).

$$\begin{aligned} A_n &= e^{-(n-j)/m} x_j = x_n + \sum_{j=1}^{n-1} e^{-(n-j)/m} x_j \\ &= x_n + e^{-1/m} \cdot \sum_{j=1}^{n-1} e^{-(n-j-1)/m} x_j \\ &= x_n + e^{-1/m} \cdot A_{n-1} \end{aligned}$$

Similarly, for B_n:

$$\begin{aligned} B_n &= \sum_{j=1}^{n} e^{-(n-j)/m} \\ &= 1 + \sum_{j=1}^{n-1} e^{-(n-j)/m} \\ &= 1 + e^{-1/m} \cdot B_{n-1} \end{aligned}$$

Computing exponential correlations is a little more involved. Given the exponential average at some time $t = n$ for some instrument x, $EMA_x(n) = \frac{A_n}{B_n}$, the exponential covariance Cov_e is simply the weighted mean from the beginning of time relative to the current exponential average:

$$Cov_e = \sum_{j=1}^{n} e^{-(n-j)/m} \Big(x_j - EMA_x(n) \Big) \Big(y_j - EMA_y(n) \Big)$$

Similarly, the exponential standard deviation would be:

$$\sigma_e = \sqrt{\sum_{j=1}^{n} e^{-(n-j)/m} \Big(x_j - EMA_x(n) \Big)^2}$$

To find the correlation, we simply normalize the covariance by dividing by the standard deviations of both assets.

```
# Compute the rolling statistics for two time series.
# Returns a list with:
# - ema: exponentially weighted average for given window size.
# - std: rectangular standard deviation for given window size.
rolling_stats <- function(x, y, wlen) {

  # Initialize values to compute exp mean, exp std.
  N <- length(x)
  ema_x <- c(x[1])      # Exponential average.
  ema_y <- c(y[1])
  std_x <- c(0)         # Standard deviation.
  std_y <- c(0)
  corr <- c(0)          # Correlation of assets.

  # For ema calculation.
  A_x <- x[1]
  A_y <- y[1]
  B_x <- 1
  B_y <- 1
```

```
  for (t in 2:N) {

    # EMA updates.
    A_x <- x[t] + exp(-1 / wlen) * A_x
    A_y <- y[t] + exp(-1 / wlen) * A_y
    B_x <- 1 + exp(-1 / wlen) * B_x
    B_y <- 1 + exp(-1 / wlen) * B_y
    ema_x[t] <- A_x / B_x
    ema_y[t] <- A_y / B_y

    # Exponential weights.
    js <- 1:t
    weights <- exp(-(t-js)/wlen)

    # Standard deviation and correlation using exponential weights.
    x_sub_ema <- x[1:t] - ema_x[t]
    y_sub_ema <- y[1:t] - ema_y[t]
    std_x[t] <- sqrt(sum(weights * x_sub_ema^2))
    std_y[t] <- sqrt(sum(weights * y_sub_ema^2))
    corr[t] <- sum(weights * x_sub_ema * y_sub_ema) / (std_x[t] * std_y[t])

    # # Standard dev and correlation.
    # start_idx = max(1,t-wlen)
    # std_x[t] <- sqrt(mean((x[start_idx:t]-ema_x[t])**2))
    # std_y[t] <- sqrt(mean((y[start_idx:t]-ema_y[t])**2))
    # cov <- mean((y[start_idx:t] - ema_y[t]) * (x[start_idx:t] - ema_x[t]))
    # corr[t] <- cov / (std_x[t] * std_y[t])
  }

  return(list(
    ema_x=ema_x,
    ema_y=ema_y,
    std_x=std_x,
    std_y=std_y,
    corr=corr
  ))
}
```

For every unique pair of instruments, compute their rolling statistics (exponential average and exponential correlation).

```
pairwise_stats <- list()
WINDOW_LEN <- 3600
PATH <- sprintf("cache/%s_%s_pairwise_stats.rds", INST, DATE)

# Check if we have already cached this result.
if (file.exists(PATH)) {
  pairwise_stats <- readRDS(PATH)
} else {

  # For every pair of assets, get the rolling correlations, means, & stdevs.
  for (i in 1:(NSYM-1)) {
    for (j in (i+1):NSYM) {
      start.time <- Sys.time()
```

```
      key <- paste0(SYMS[i], ",", SYMS[j])
      pairwise_stats[[key]] <- rolling_stats(
        x=mid_by_sym[[SYMS[i]]]$mid,
        y=mid_by_sym[[SYMS[j]]]$mid,
        wlen=WINDOW_LEN
      )
      end.time <- Sys.time()
      print(sprintf("Inner loop: %.2f minutes",
                    difftime(end.time, start.time, units="mins")))
    }
  }

  # Save to cache.
  saveRDS(pairwise_stats, file=PATH)
}
```

Signal construction

At every point in time, we have computed the correlation matrix for this group of assets. Our assumption is that the eigenvectors of this correlation matrix are either mean-reverting or random walks. We assume that the largest eigenvectors are always the random walk, and the later ones (with smaller singular values) are mean-reverting. In theory, it might be possible that the first and third components are mean reverting while the second is a random walk; however, testing this would require testing 2^n subsets which is impractical. Nevertheless, this assumption makes sense. In the random walk direction, you expect the variance to keep increasing in time, while in a mean reverting direction you expect it to reach a maximum and stabilize.

At every point in time, we want to compute the eigenvectors of the correlation matrix. That is, given our correlation matrix $\hat{\Sigma}$, we do eigen-decomposition to get eigenvectors V. Because the correlation matrix is symmetric, the eigenvectors are orthogonal to one another. Therefore, they form an orthonormal basis. We can multiply by the matrix of normalized eigenvectors (an orthonormal matrix) to get a vector $\vec{\xi}$. Each component of this vector represents how much of each eigenvector this particular $\vec{\xi}$ contains:

$$\vec{\xi} = V\vec{x}$$

By assumption, we assume that some components of this vector are mean-reverting, so we set them to zero. Then, we apply the inverse linear transformation (i.e. back into the standard basis). Since V is orthonormal, $V = V^{-1}$:

$$\vec{x}^* = DV \begin{bmatrix} \xi_1 \\ \xi_2 \\ 0 \\ \vdots \end{bmatrix} + \underline{\vec{x}}$$

Here, D is the diagonal matrix of standard deviations, and $\underline{\vec{x}}$ is the exponential mean of prices. This converts from the eigenbasis to the standard basis, and un-normalizes by re-scaling by volatility and adding back the exponential mean.

```
missing <- 0
signals <- matrix(data=NA, nrow=NSEC, ncol=NSYM)

for (t in 1:NSEC) {

  # Progress tracking.
  if (t %% 10000 == 0) {
    print(t)
  }
```

```
  # Get z-scores, exponential means, stdevs of all prices for time t.
  zs <- vector(mode="numeric", length=NSYM)
  emas <- vector(mode="numeric", length=NSYM)
  stds <- vector(mode="numeric", length=NSYM)

  # Create an 8x8 matrix and fill it with correlations.
  m <- matrix(1, nrow=NSYM, ncol=NSYM)
  for (i in 1:(NSYM-1)) {
    for (j in (i+1):NSYM) {
      key <- paste0(SYMS[i], ",", SYMS[j])
      m[i,j] <- pairwise_stats[[key]]$corr[t]
      m[j,i] <- m[i,j]

      # Get the z-score.
      zs[i] <- (mid_by_sym[[SYMS[i]]]$mid[t] -
                pairwise_stats[[key]]$ema_x[t]) / pairwise_stats[[key]]$std_x[t]
      zs[j] <- (mid_by_sym[[SYMS[j]]]$mid[t] -
                pairwise_stats[[key]]$ema_y[t]) / pairwise_stats[[key]]$std_y[t]

      # Get exponential means, stdevs, and prices.
      emas[i] <- pairwise_stats[[key]]$ema_x[t]
      emas[j] <- pairwise_stats[[key]]$ema_y[t]
      stds[i] <- pairwise_stats[[key]]$std_x[t]
      stds[j] <- pairwise_stats[[key]]$std_y[t]
    }
  }

  # If there are any missing  correlation values, don't calculate the signal.
  if (any(is.na(m))) {
    missing <- missing + 1
  } else {

    # Get eigenvectors & project onto eigenvector space.
    V <- eigen(m)$vectors
    proj <- V %*% zs

    # Set mean reverting directions to 0 to project onto reverting subspace.
    proj[3:8,] <- 0

    D <- diag(stds)
    p_star <- D %*% V %*% proj + emas

    ## TODO: plot p_star against the prices as per lec10b slide 23.

    signals[t,] <- t(p_star) - mids[t,]
  }
}
```

```
## [1] 10000
## [1] 20000
## [1] 30000
## [1] 40000
## [1] 50000
```

```
## [1] 60000
## [1] 70000
## [1] 80000
```

Let's populate missing signal values with the previous non-missing value.

```
signals <- na.locf(signals, na.rm=FALSE)
```

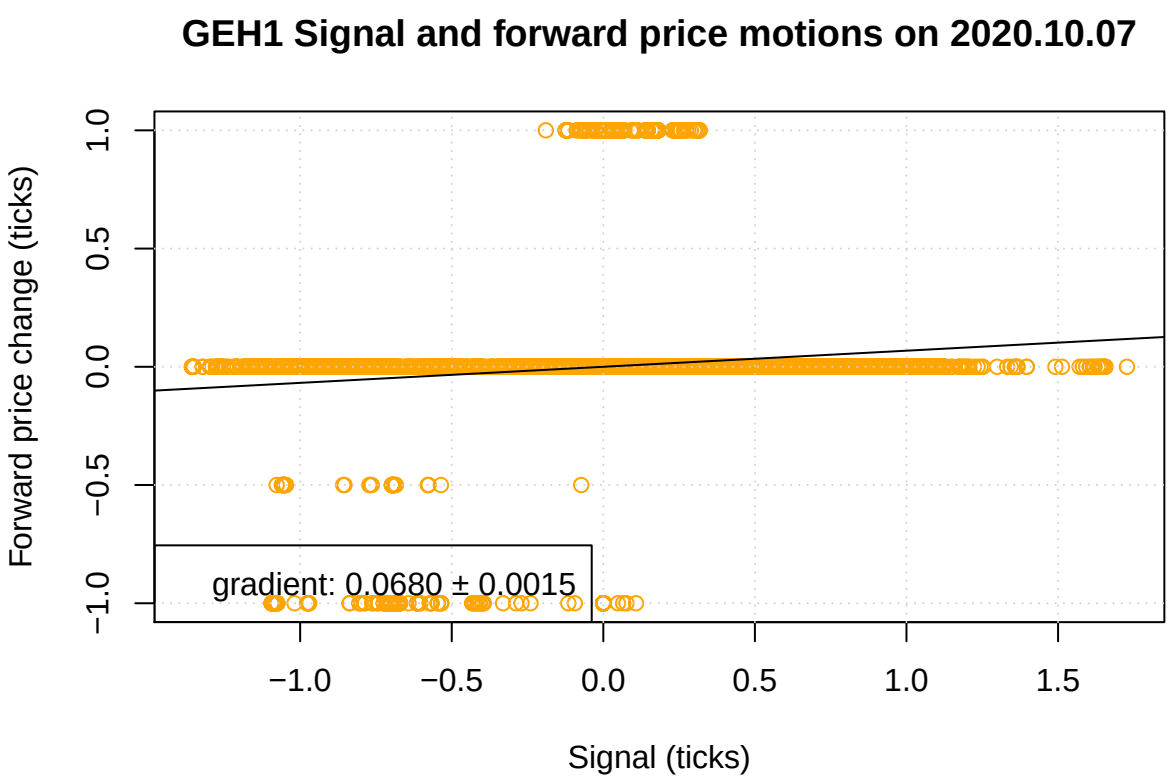
TODO: look at signular values over time, plot them. If they stabilize –> mean reverting If they keep increasing –> random walk.

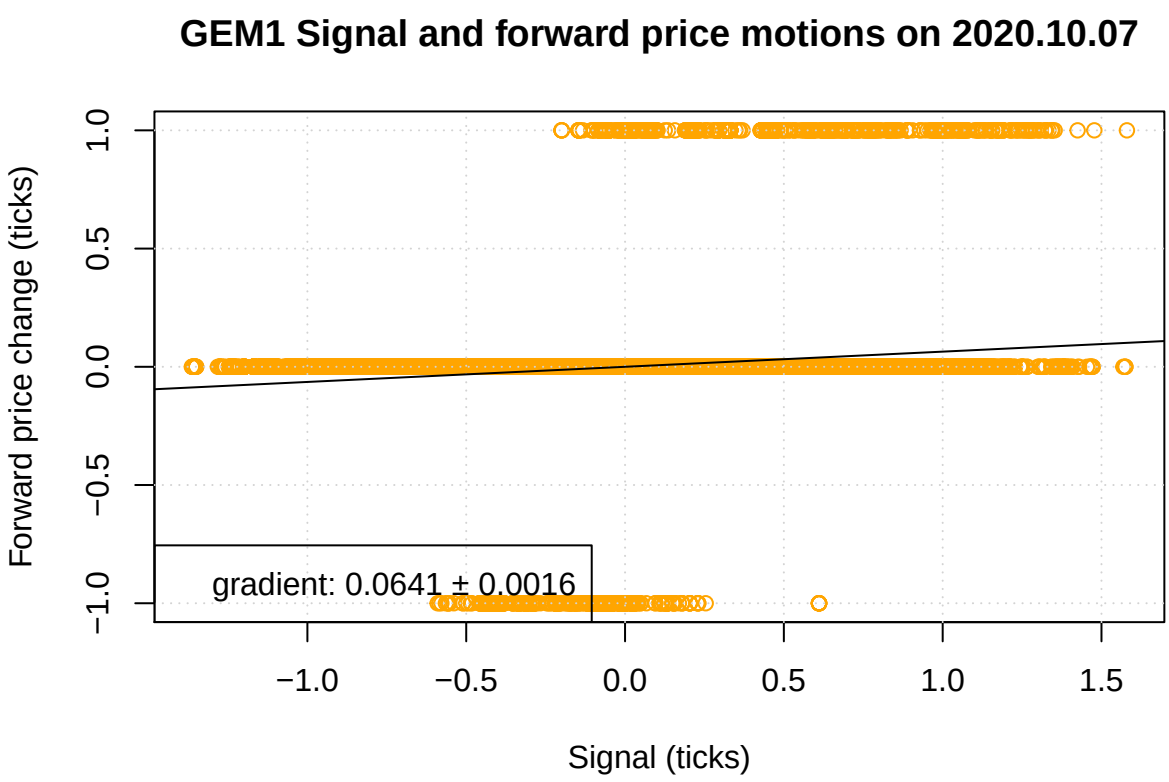
To evaluate the strength of this signal, let's look at the correlation of forward price changes at a given lag with the signal, for each asset.

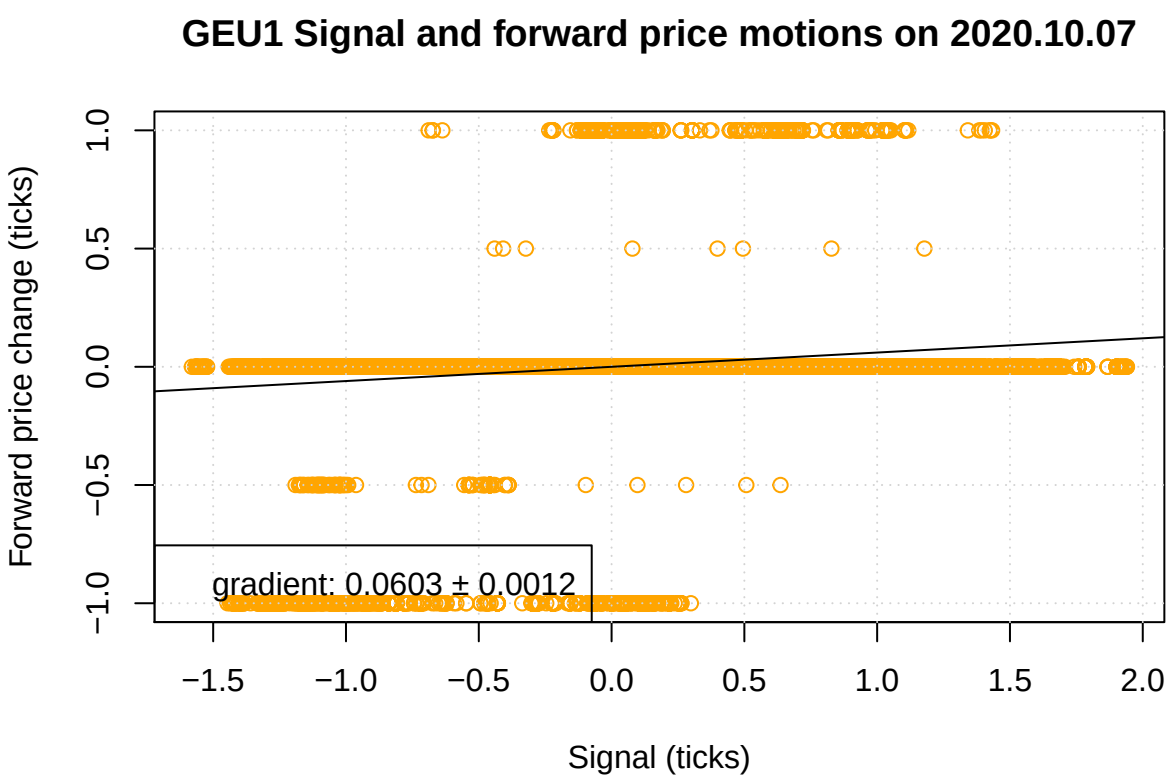
```
LAG <- 120

mid_diff <- diff(mids, lag=LAG) / TICKSZ
signals_shifted <- head(signals, -LAG) / TICKSZ # Drop last LAG elements.
```

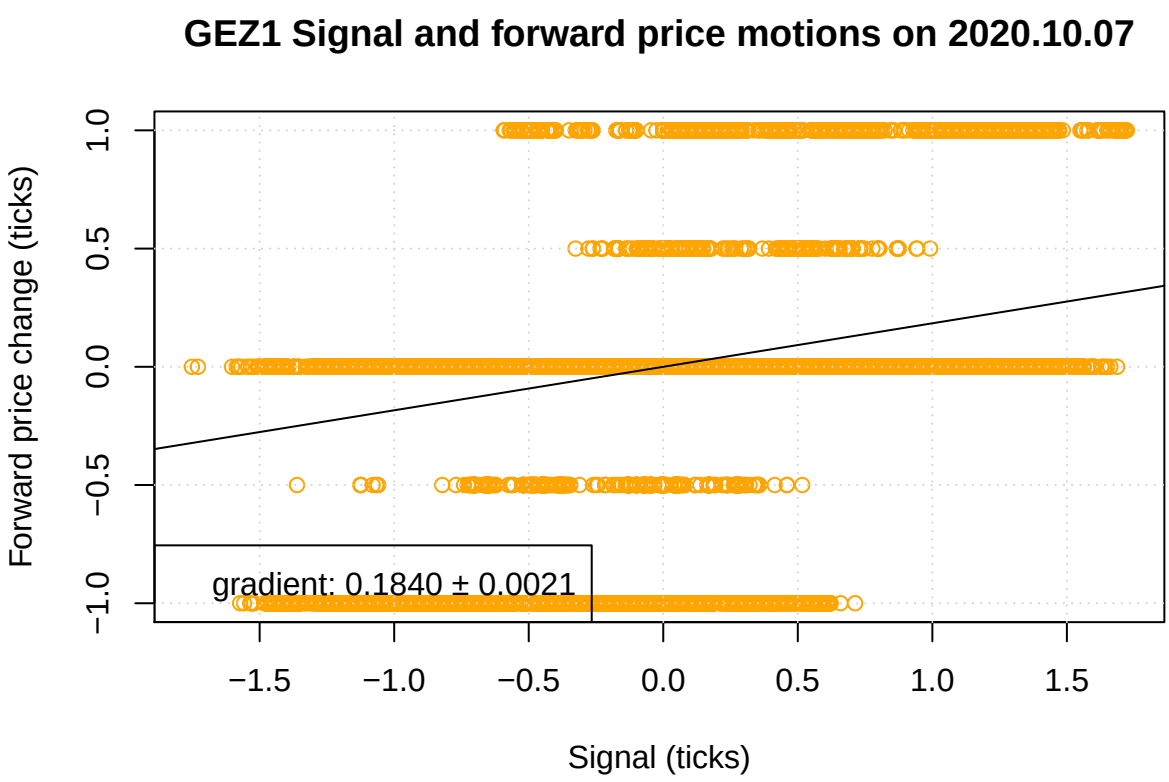
Let's regress the forward price change at a 120s lag against the signal. We will force an intercept of 0 and look at the gradient of the regression.

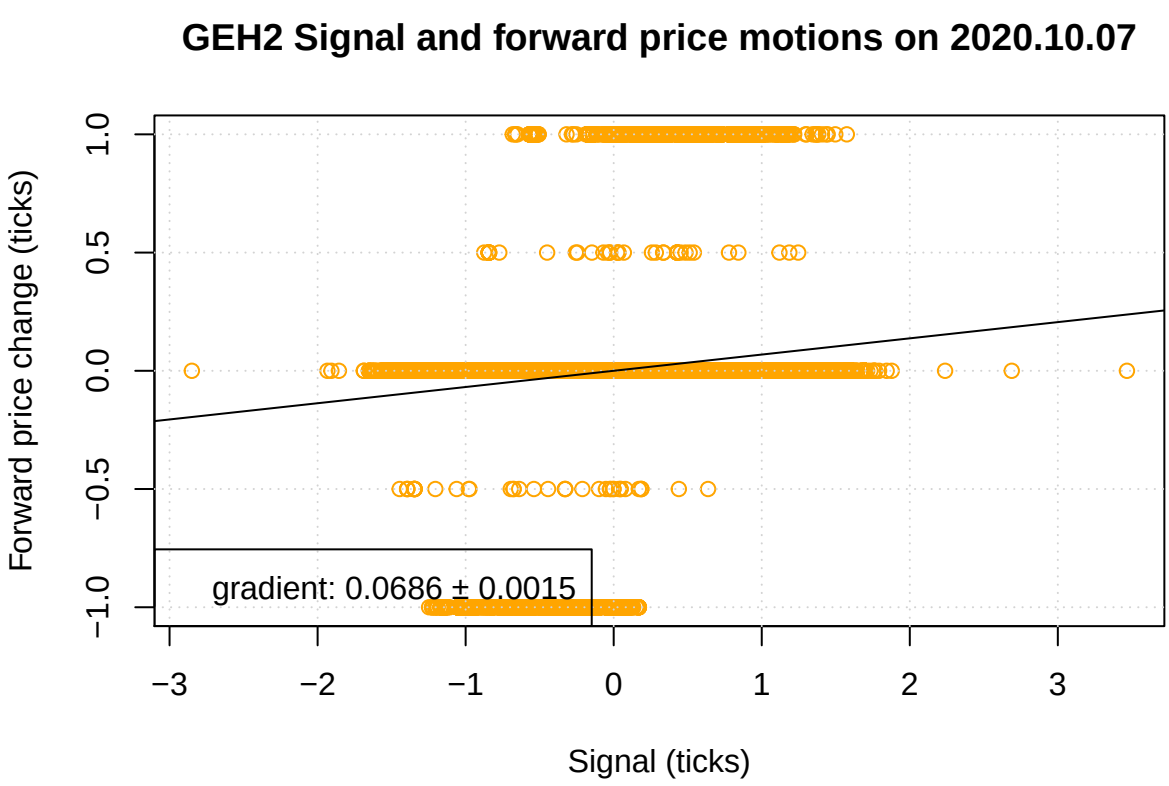
```
for (i in 1:NSYM) {
  y <- mid_diff[,i]
  x <- signals_shifted[,i]

  plot(x, y, xlab="Signal (ticks)", ylab=sprintf(
       "Forward price change (ticks)", LAG),
       main=sprintf("%s Signal and forward price motions on %s", SYMS[i], DATE),
       col="orange")
  grid()
  model = lm(y ~ x + 0)
  abline(model)
  gradient = summary(model)$coefficients[1,1]
  stderr <- summary(model)$coefficients[1,2]
  legend(x="bottomleft", legend=c(sprintf("gradient: %.4f \u00B1 %.4f",
                                          gradient, stderr)))
}
```

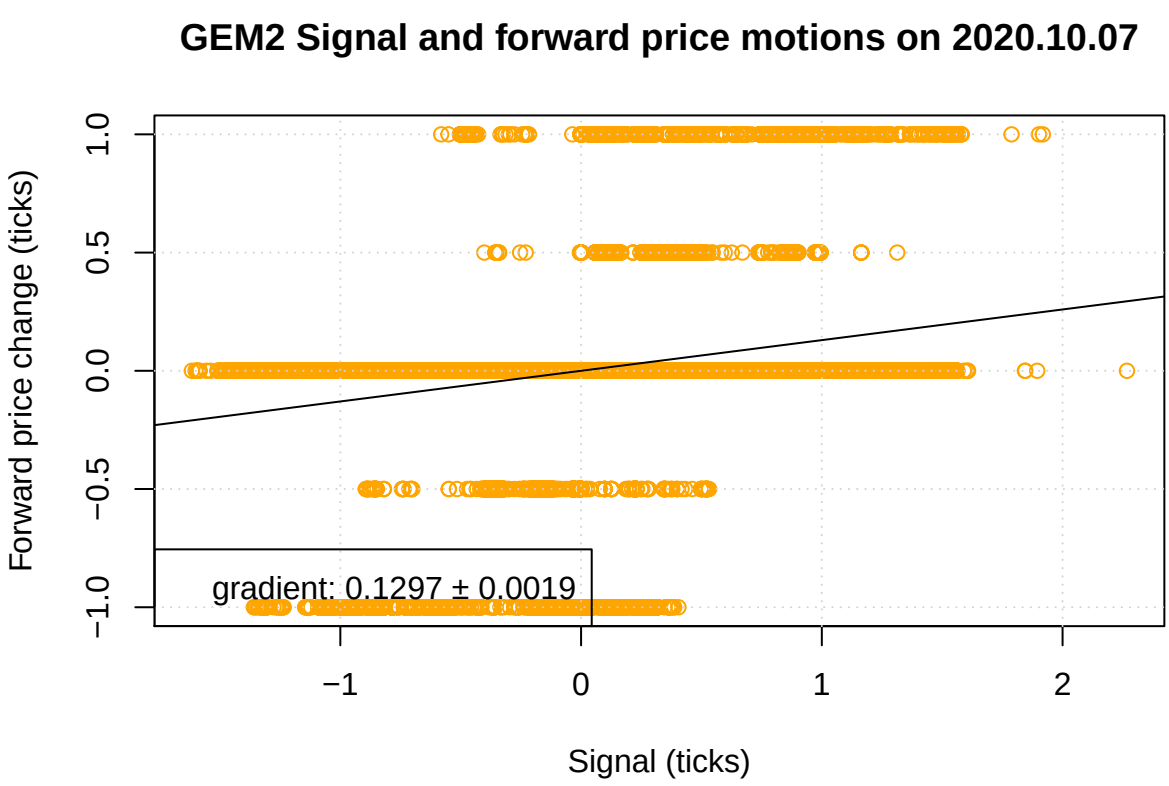

GEH1 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.0680 ± 0.0015
−1.0
−0.5
0.0
0.5
1.0
1.5
Signal (ticks)

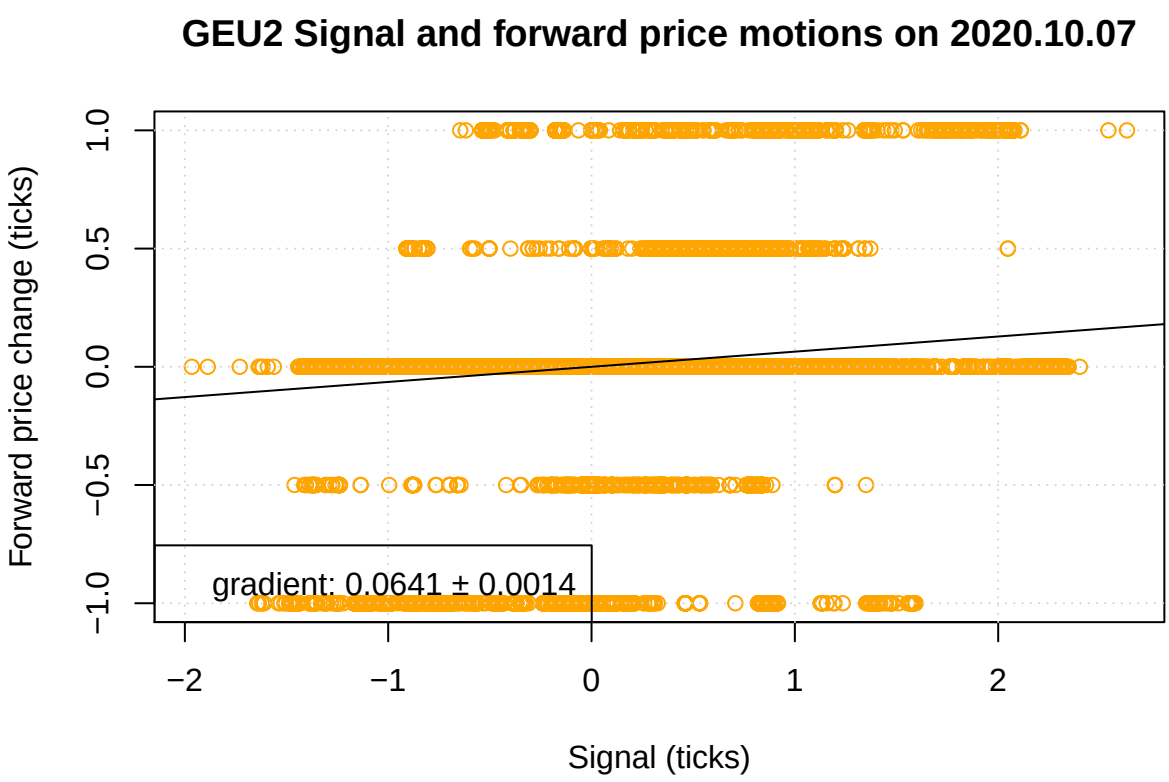

GEM1 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.0641 ± 0.0016
−1.0
−0.5
0.0
0.5
1.0
1.5
Signal (ticks)

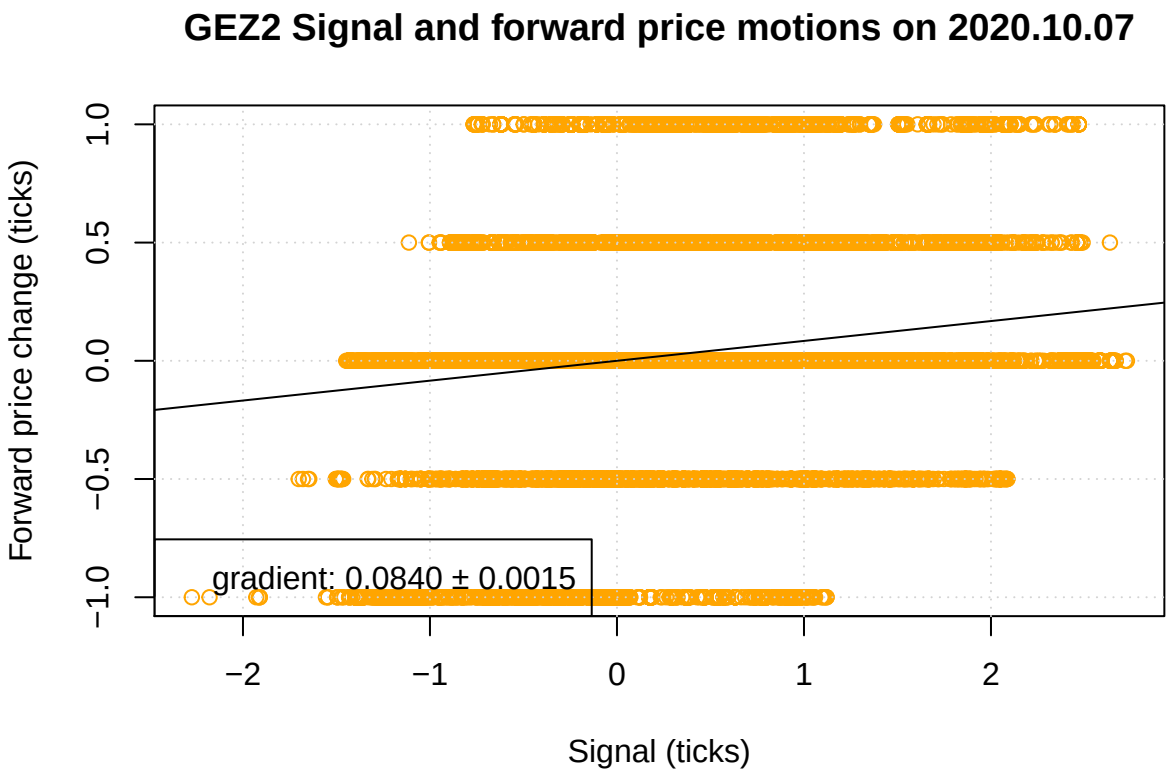

GEU1 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.0603 ± 0.0012
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Signal (ticks)


GEZ1 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.1840 ± 0.0021
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Signal (ticks)


GEH2 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.0686 ± 0.0015
−3
−2
−1
0
1
2
3
Signal (ticks)


GEM2 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.1297 ± 0.0019
−1
0
1
2
Signal (ticks)

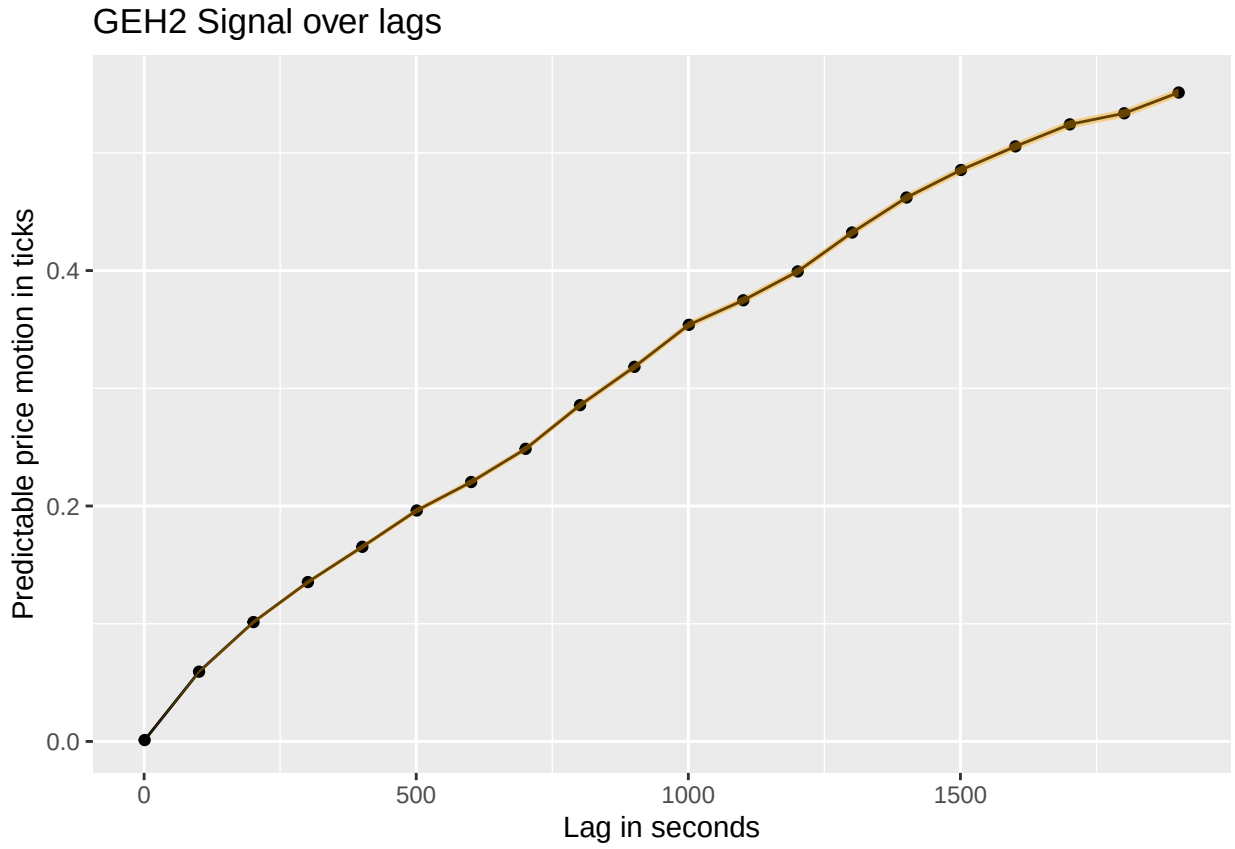

GEU2 Signal and forward price motions on 2020.10.07
Forward price change (ticks)
1.0
0.5
0.0
−0.5
−1.0
gradient: 0.0641 ± 0.0014
−2
−1
0
1
2
Signal (ticks)

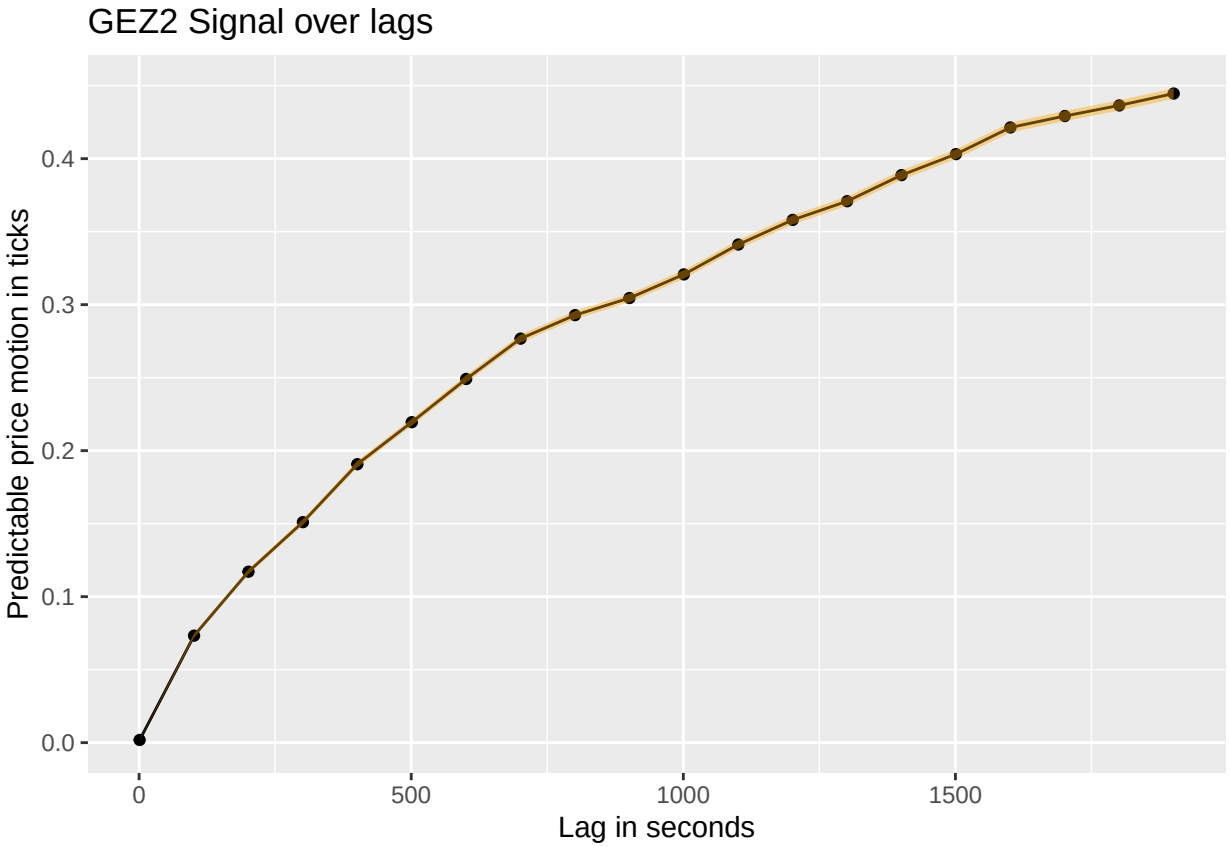


Now, let's pick a few instruments and plot how the signal evolves over lag.

```
LAG_SEQ <- seq(1, 2000, by=100)

# For a series of lags, plot the signal strength for `size` random instruments.
plot_strength_by_lag <- function(size) {

  # Sample the instruments randomly.
  indices <- sample(1:NSYM, size=size, replace=FALSE)

  # Save the coefficients and errors in them.
  coefs <- matrix(data=NA, nrow=length(LAG_SEQ), ncol=size)
  stderrs <- matrix(data=NA, nrow=length(LAG_SEQ), ncol=size)

  for (i in 1:length(LAG_SEQ)) {
    lag <- LAG_SEQ[i]
    y <- diff(mids, lag=lag) / TICKSZ
    x <- head(signals, -lag) / TICKSZ # Drop last LAG elements.

    for (j in 1:size) {
      idx <- indices[j]
      model = lm(y[,idx] ~ x[,idx] + 0)
      coefs[i,j] <- summary(model)$coefficients[1,1]
      stderrs[i,j] <- summary(model)$coefficients[1,2]
    }
  }
```

```
  for (i in 1:size) {

    # Plot error bands with cumulative standard error.
    data <- data.frame(x=LAG_SEQ, y=coefs[,i], err=stderrs[,i])
    p <- ggplot(data, aes(x=x, y=y)) +  geom_point() + geom_line()
    p <- p + geom_ribbon(aes(ymin=y-err, ymax=y+err),
                         linetype=2, alpha=0.4, fill="orange")
    p <- p + ggtitle(sprintf("%s Signal over lags", SYMS[indices[i]]))
    p <- p + labs(x="Lag in seconds", y="Predictable price motion in ticks")
    print(p)
  }
}

plot_strength_by_lag(2)
```




GEZ2 Signal over lags
Predictable price motion in ticks
0.0
0.1
0.2
0.3
0.4
0
500
1000
1500
Lag in seconds