```r
library("ggplot2")
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library("readxl")

library("fractaldim")
```

```
## Loading required package: abind
```

```r
library("somebm")
library("stats")
```

```r
oil_data <- read_xls("oil_complete.xls") #2589 rows
names(oil_data)[2] <- "fut_price"
names(oil_data)[3] <- "spot_price"
names(oil_data)[4] <- "fut_vol"
names(oil_data)[5] <- "spot_vol"
```

```r
genNoise <- function(data, size) {

  i <- 1
  hurst <- NULL

  while (i < length(data) - size - 1) {
    j <- i + size
    hurst <- c(hurst, 2 - fd.estim.hallwood(data[i:j])$fd)
    i <- i + 1

  }

  noise <- NULL

  for (i in 1:length(hurst)) {
    if (hurst[i] < 0)
      noise <-
        c(noise, fbm(0, 1)[1])
    else
      noise <- c(noise, fbm(hurst[i], 1)[1])

  }

  return (noise)

}
```

```r
genJumps <- function(T, la, j_mu, j_var) {

  jump_count <- rpois(1, T * la) # in T units, how many jumps will there be

  x <- runif(jump_count, 0, 1)
  case_when(x < 0.5 ~ -1, x > 0.5 ~ 1)

  jump_vals <- x*rnorm(jump_count, j_mu, j_var) # what will the value of the jumps be

  # determine time of poisson jumps

  units <- runif(jump_count, 1, 100)
  units <- round (T * units / sum(units), 0)
  if (sum(units) != T) units[length(units)] <- (units[length(units)] + T - sum(units))
  jump_times <- sample(units)

  final_data <- c()
  pjumps <- c()

  for (i in 1:length(jump_times)) {

    final_data <- c(final_data, rep(jump_vals[i], jump_times[i]))
    pjumps <- c(pjumps, rep(jump_vals[i], jump_times[i]))

  }

  final_data <- final_data

  return (final_data)

}
```

To improve our model, we looked into implementing Poisson jump shifts. The algorithm for simulating Poisson jumps is simple. First, to determine how many jumps there will be, N_T is found:

$$N_T \sim Po(T\lambda)$$

Then, simulate N_T uniforms to determine the time between jumps:

$$p_{1\ldots N_T} \sim U(0, a)$$

Where a is some arbitrary value. However, the total of the lengths between the jumps must be equal to T. Therefore, we normalize the data to sum to T.

$$U_i = \frac{Tp_i}{\sum_{i=1}^{N_T} p_i}$$

This ensures the following sum holds:

$$\sum_{i=1}^{N_T} U_i = T$$

Addionallity, the total time elapsed before the next jump is:

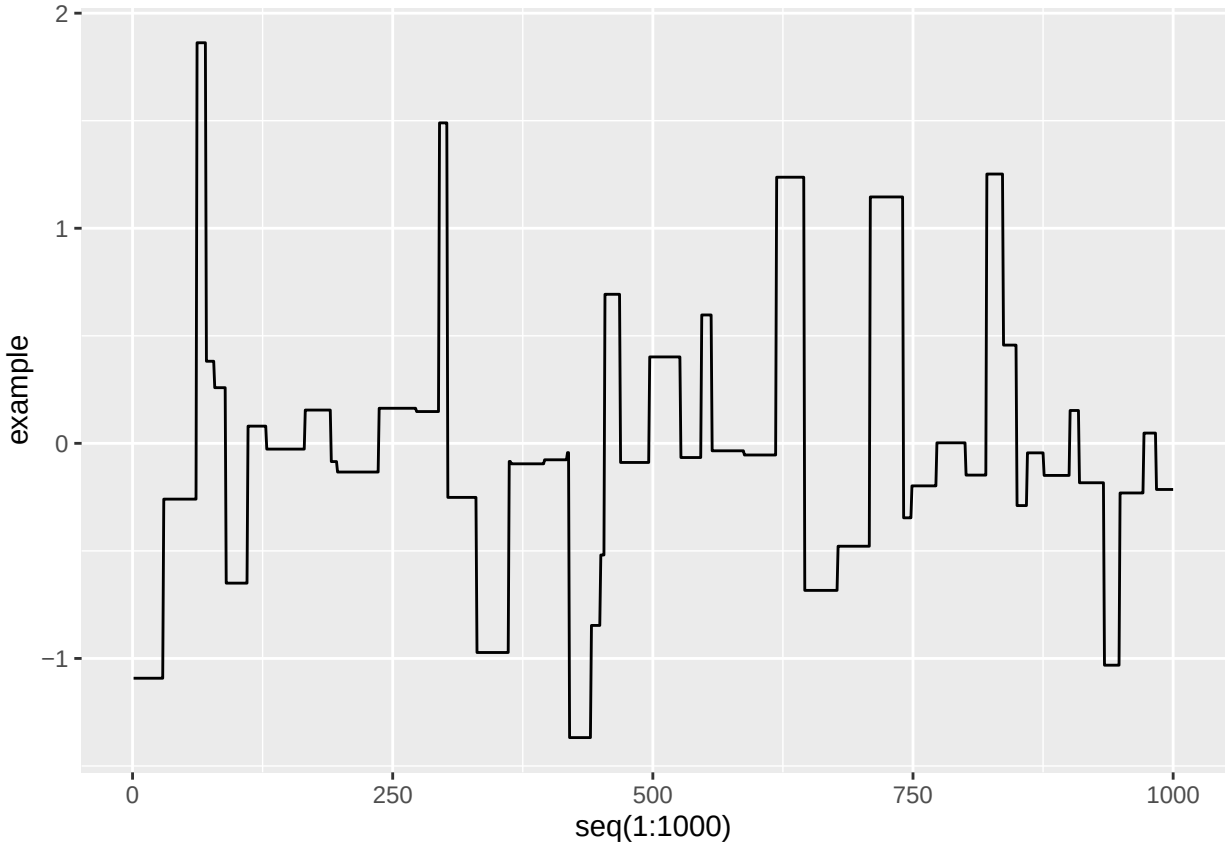
$$F(U_i) = \sum_{j=1}^{i} U_j$$

To actually compute this, whole integers are needed. Because rounding may make $\sum_{i=0}^{N_t} P_i \neq T$, a buffer is added to the last value equal to $B = T - \sum_{i=0}^{N_t} Y_i$ to ensure the equivalence holds. We then sampled all the values to randomize the order, so the buffer wasn't always at the end. Finally we simulate the size of the jumps:

$$Y_{1...T} \sim N(\mu, \sigma^2)$$

This is what a example of a sample may look like for $T = 1000$, $\lambda = 0.05$

```
example <- genJumps(1000, 0.05, 0, 1)
ggplot() + geom_line(aes(x = seq(1:1000), y = example))
```



Because the price of oil is subject to random political events that may quickly change the price, adding a poisson jump process to our simulation will help us to form confidence intervals of how we expect the price to perform over a period of time. After we implement the jumps, we must account for the intraday noise, or Brownian motion. For simple Brownian Motion, $X_i \sim N(0, 1)$, with

$$S_n = \sum_{i=1}^{n} X_i$$

$$E(S_n) = \sqrt{n}$$

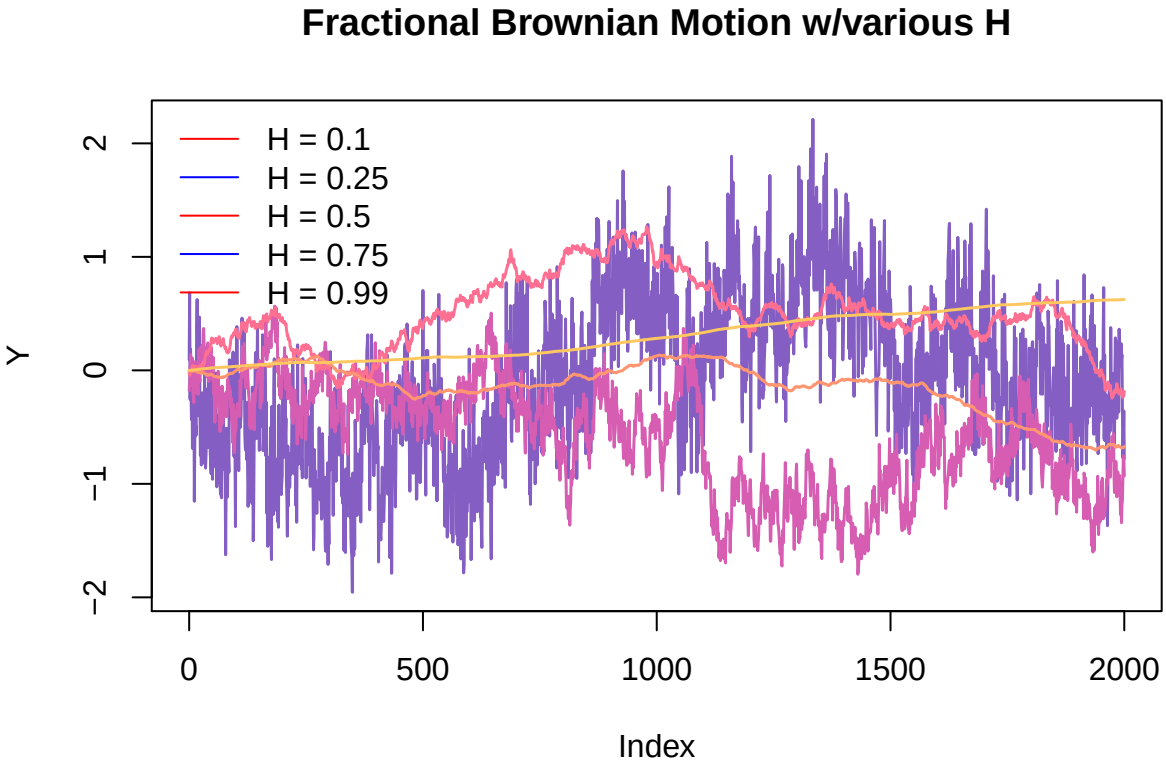
Therefore, the interpretation is that over n samples, the expected distance from the origin is $\sqrt{n}$. However, for oil, we believed this was an oversimplification of the price behavior. Oil is very connected to macroeconomic forces, creating cyclicality in supply and demand. A company like Apple may be expected to grow on average for 40 years, but this assumption does not hold for oil, thus we believe the Brownian Motion used to simulate our volatility should reflect that. In our research, we came across Hurst Exponents. For some sum S_n of arbitrary random variables Y_i

$$S_n = \sum_{i=1}^{n} X_i$$

$$E(S_n) = n^H$$

The Hurst Exponent describes the degree of dispersion a time series exercises, with the domain of $H : [0, 1]$. $H = 0.5$ would be standard Brownian Motion. $H = 1$ would dsuggest the series will move far from the origin, meaning the series is trend following. $H = 0$ would suggest the series is mean reverting or returns to the starting point over time.

```
set.seed(10)

df <- NULL

n <- 2000

h1 <- fbm(0.1, n)
h2 <- fbm(0.25, n)
h3 <- fbm(0.5, n)
h4 <- fbm(0.75, n)
h5 <- fbm(0.99, n)

ts.plot(main = "Fractional Brownian Motion w/various H",ts(h1, start = c(0,1)),
        ts(h2, start = c(0,1)),
        ts(h3, start = c(0,1)),
        ts(h4, start = c(0,1)),
        ts(h5, start = c(0,1)),
    col = c('#845EC2', '#D65DB1', '#FF6F91', '#FF9671', '#FFC75F'),
    xlab = "Index",
    ylab = "Y",
    lwd = 1.5)

legend("topleft", bty="n", lty=c(1,1), col=c("red","blue"),
       legend=c("H = 0.1", "H = 0.25", "H = 0.5", "H = 0.75 ", "H = 0.99"))
```



The Hurst exponent is found empirically by looking at how self-similar the data is. First, we found the Hurst Exponent of the last 100 trading days, using the library fractaldim. Then, we generated a sample of random data with a Hurst exponent equal to the one found empirically. To do this, we used the somebm library. In conclusion, our simulated price at time T is equal to the sum of all previous noise plus all previous poisson jumps plus the initial starting value:

$$X_T = X_1 + \sum_{t=1}^{T-1} B_t + \sum_{t=1}^{T-1} \left[\sum_{i=0}^{N_T} Y_i \, \mathbf{1}_{F(U_i) \leq t} \right]$$

```
la_real <- 0.03592
T <- nrow(oil_data)
init <- 85.5

fut_jumps <- filter(oil_data, abs(fut_vol) > 0.05)
fj_vol <- abs(fut_jumps$fut_vol)
j_mu <- mean(fj_vol)
j_var <- var(fj_vol)

jumps <- exp(genJumps(T, la_real, j_mu, j_var))
noise <- cumsum(genNoise(oil_data$fut_price, 100))

b <- length(jumps)-length(noise)+1
jumps <- jumps[b:length(jumps)]

sim <- init + jumps*noise
```

```
dis <- seq(1:length(jumps))
p = ggplot() + geom_line(aes(x = dis, y = sim))
p
```


50
0
−50
sim
0
500
1000
1500
2000
2500
dis