BlockBootstrap

Reng Chiz Der

2024-03-05

Task

Block Bootstrapping w/ random block size & randomization
- A key portion of this project will be constructing Monte Carlo simulations for asset returns
as well as interest rates (on cash) and yields (on bonds or other fixed income assets, as well as dividends).
- Demonstrate a multi-variable block bootstrap function with randomized block size and random noise parameters for the input time series.
- Use xts time series and assume 'wide' data construction.

Environment Setup

```
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: TTR
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(zoo)
library(xts)
library(ggplot2)
library(PerformanceAnalytics)
```

```
##
## Attaching package: 'PerformanceAnalytics'
```

```
## The following object is masked from 'package:graphics':
##
##     legend
```

Get Real-world Portfolio Data

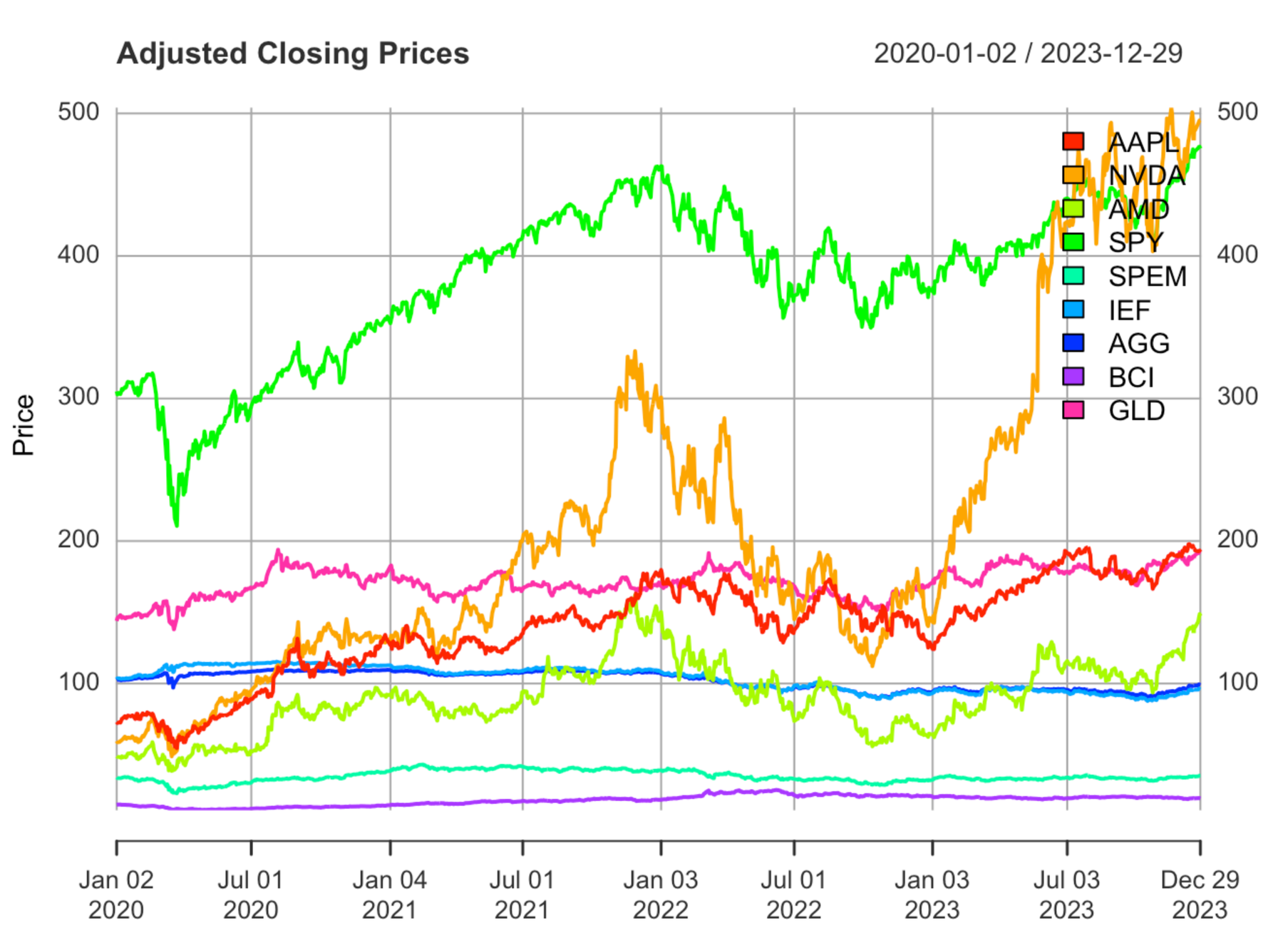
```
# A function to get real world data
fetch_and_preprocess <- function(tickers) {
  stocks <- lapply(tickers, function(ticker) {
    stock_data <- getSymbols(ticker, src = 'yahoo', from = "2020-01-01", to = "2024-01-01", auto.assign = FALSE)
    colnames(stock_data) <- gsub(".*\\.", "", colnames(stock_data)) # Remove the prefix
    adjusted_data <- stock_data[, "Adjusted", drop = FALSE] # Select only the adjusted closing value
    if ("Adjusted" %in% colnames(stock_data)) {
      colnames(adjusted_data) <- ticker
      return(adjusted_data)
    } else {
      stop("Adjusted closing value not found for", ticker)
    }
  })

  # Preprocess the data
  for (i in seq_along(stocks)) {
    if (!is.null(stocks[[i]])) {
      # Calculate the number of missing values
      missing_values <- colSums(is.na(stocks[[i]]))

      # Replace missing values with the previous day's values
      stocks[[i]] <- na.locf(stocks[[i]])
    }
  }

  # Combine the data into a single xts object
  combined_data <- do.call(merge, stocks)
  return(combined_data)
}

# Stocks
stocks <- c('AAPL', 'NVDA', 'AMD')
stock_data <- fetch_and_preprocess(stocks)

# S&P500, Emerging Market ETF
etfs <- c('SPY', 'SPEM')
etf_data <- fetch_and_preprocess(etfs)

# fixed income (bonds): IEF (treasury bond ETF), AGG (US Aggregate Bond ETF)
fis <- c('IEF', 'AGG')
fi_data <- fetch_and_preprocess(fis)

comms <- c('BCI')
comm_data <- fetch_and_preprocess(comms)

# SPDR  Gold Share ETF
gold <- c('GLD')
gold_data <- fetch_and_preprocess(gold)

# Long volatility
long_vol <- c('VIXY')
long_vol_data <- fetch_and_preprocess(long_vol)
```

Construct Portfolio and Prepocessing

```
# Create a sample portfolio of different assets
portfolio <- cbind(stock_data, etf_data, fi_data, comm_data, gold_data)
portfolio_na_counts <- colSums(is.na(portfolio))
portfolio_na_counts
```

```
## AAPL NVDA  AMD  SPY SPEM  IEF  AGG  BCI  GLD
##    0    0    0    0    0    0    0    0    0
```

```
plot(portfolio, main = "Adjusted Closing Prices", xlab = "Date", ylab = "Price", col = rainbow(ncol(portfolio)), legend.loc = "topright")
```




```
# Convert the data from adjusted closing price to daily return
portfolio_dr <- Return.calculate(portfolio, method = "simple")
portfolio_dr <- portfolio_dr * 100
# Remove first day w/o return
portfolio_dr <- portfolio_dr[-1, ]
```

Block Bootstrap Function

```
### block bootstrap w/ random block size + random noise params
random_size_block_bootstrap <- function(series) {
  ori_size <- length(series)

  # Initialize new series
  new_series <- numeric(0)

  while (length(new_series) < ori_size) {
    # Generate random block size restricted to at most half the size of the original series
    # Would ensure that the blocks towards the end of generated series not smaller
    block_size <- sample(1:(ori_size %/% 2), size = 1)

    # Get random index
    index <- sample(1:ori_size, size = 1)

    cat("Block size:", block_size, "Index:", index, "\n")

    # Circular sampling
    sampled_values <- numeric(block_size)
    for (j in 1:block_size) {
      sampled_index <- ((index - 1 + j - 1) %% ori_size) + 1  # Circular index calculation
      sampled_values[j] <- series[sampled_index]
    }

    # Append sampled series
    new_series <- c(new_series, sampled_values)
  }

  # Trim down new series to original series size
  new_series <- new_series[1:ori_size]

  # Random noise parameters
  series_std_dev <- sd(series)

  # Set noise to be a fraction (e.g. 10%) of the series' s.d.
  noise_frac <- 0.1
  noise_sd <- noise_frac * series_std_dev

  # Random noise with mean = 0 as expected return can be 0, and s.d. be fraction of the s.d. (volatility)
  new_series <- new_series + rnorm(length(new_series), mean = 0, sd = noise_sd)

  return(new_series)
}

# Function to perform block bootstrap independently on each asset
portfolio_bootstrap <- function(portfolio) {
  # Get the names of assets
  col_names <- colnames(portfolio)

  # Initialize data frame to store bootstrapped data for each column
  bootstrapped_data <- data.frame(matrix(NA, nrow = nrow(portfolio), ncol = length(col_names)))
  colnames(bootstrapped_data) <- col_names

  # Loop through each column
  for (col in col_names) {
    # Perform block bootstrap on the column
    bootstrapped_col <- random_size_block_bootstrap(portfolio[, col])

    # Store bootstrapped column in the data frame
    bootstrapped_data[[col]] <- bootstrapped_col
  }

  # Convert data frame to xts object
  bootstrapped_xts <- xts(bootstrapped_data, order.by = index(portfolio))

  return(bootstrapped_xts)
}
```

Apply Block Bootstrap on Data

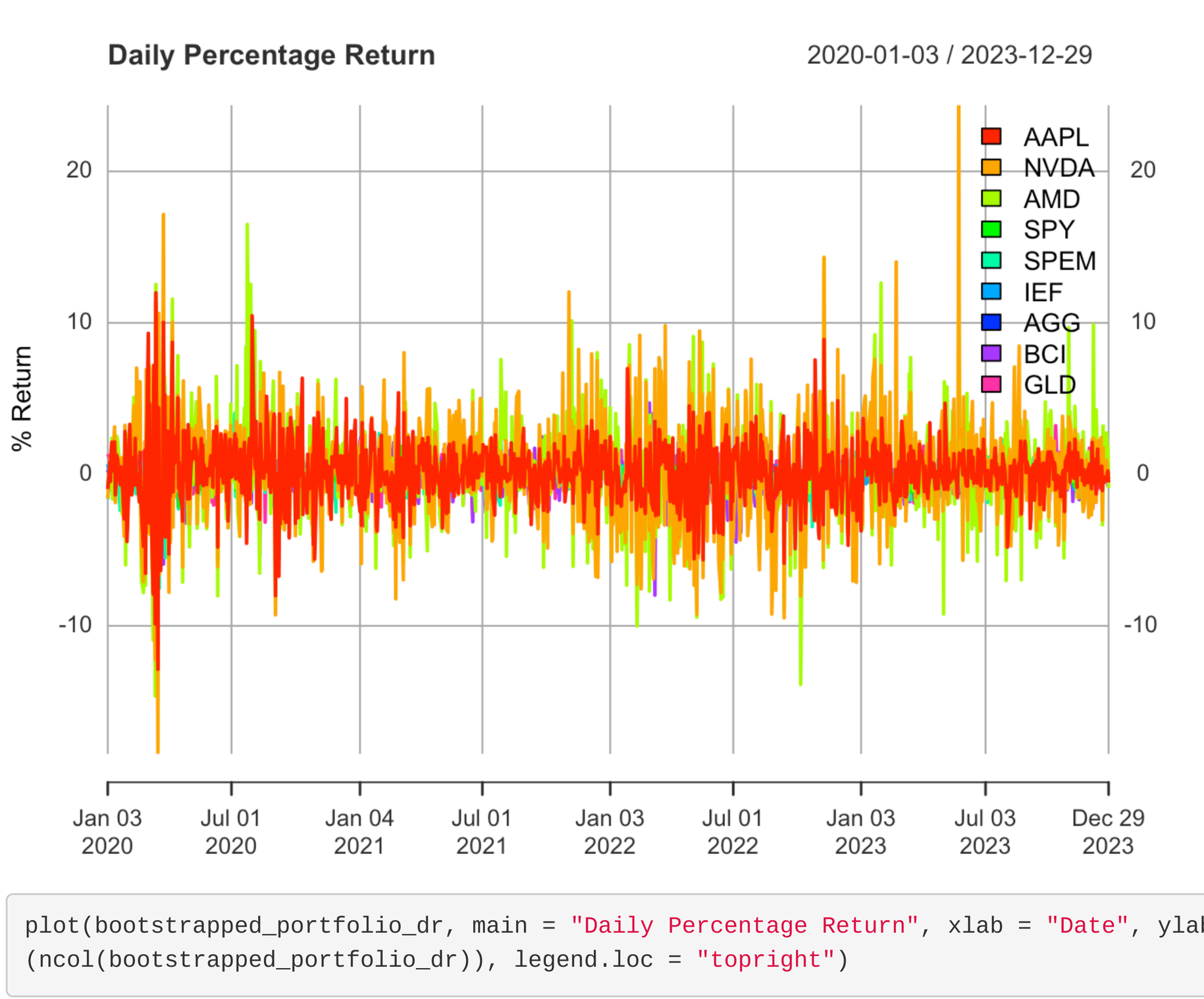
```
bootstrapped_portfolio_dr = portfolio_bootstrap(portfolio_dr)
```

```
## Block size: 232 Index: 534
## Block size: 54 Index: 450
## Block size: 472 Index: 842
## Block size: 217 Index: 485
## Block size: 278 Index: 286
## Block size: 296 Index: 234
## Block size: 121 Index: 600
## Block size: 336 Index: 916
## Block size: 392 Index: 267
## Block size: 132 Index: 451
## Block size: 156 Index: 113
## Block size: 291 Index: 319
## Block size: 313 Index: 724
## Block size: 280 Index: 619
## Block size: 323 Index: 554
## Block size: 407 Index: 231
## Block size: 180 Index: 262
## Block size: 75 Index: 34
## Block size: 391 Index: 672
## Block size: 25 Index: 448
## Block size: 422 Index: 312
## Block size: 128 Index: 558
## Block size: 76 Index: 10
## Block size: 6 Index: 409
## Block size: 347 Index: 768
## Block size: 400 Index: 874
## Block size: 297 Index: 1
## Block size: 443 Index: 837
## Block size: 158 Index: 441
## Block size: 245 Index: 85
## Block size: 310 Index: 823
## Block size: 165 Index: 612
## Block size: 266 Index: 710
## Block size: 397 Index: 293
## Block size: 320 Index: 603
## Block size: 451 Index: 554
## Block size: 489 Index: 313
## Block size: 482 Index: 568
## Block size: 428 Index: 241
## Block size: 51 Index: 276
## Block size: 257 Index: 563
```

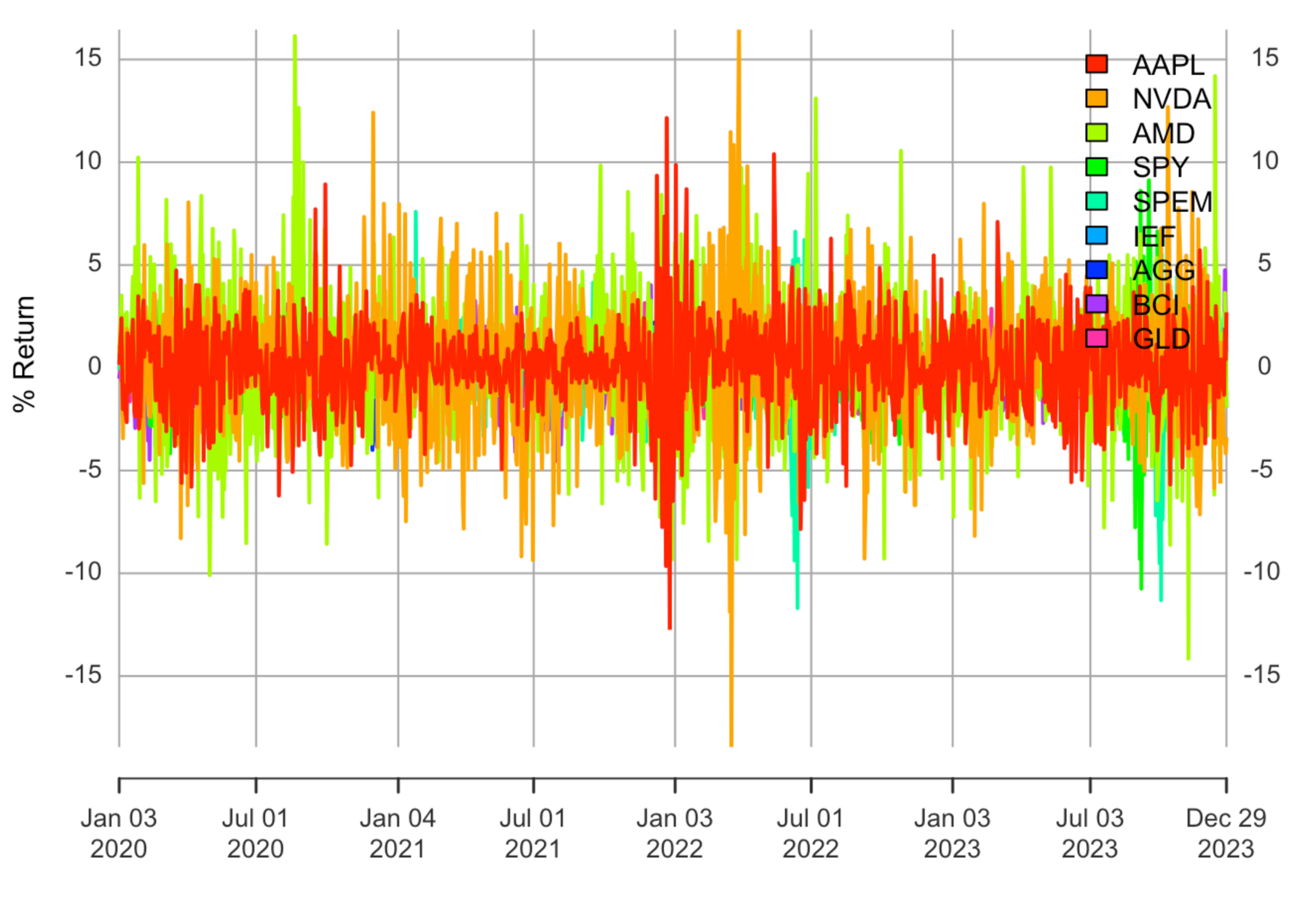
```
bootstrapped_portfolio_dr
```

```
##                    AAPL        NVDA         AMD        SPY        SPEM
## 2020-01-03  0.137836396  1.08946393  0.64593547  2.0740647 -0.09039788
## 2020-01-06  2.410099160  0.73175868  3.51904283  2.1716303  1.57481107
## 2020-01-07 -0.318705659 -1.70753208 -0.23348248  1.1953476  0.18012069
## 2020-01-08 -2.011375483 -3.43378672  0.31019124  1.1033742  0.94650539
## 2020-01-09 -0.853443783  0.85893473  2.57157203 -0.1941761 -0.25362744
## 2020-01-10 -1.824460642 -0.04450172  0.09366674  1.0635368  0.88715150
## 2020-01-13 -2.644003520  1.90173298  2.69756328 -1.2343690 -0.03009575
## 2020-01-14  1.679343332  0.51769613  0.81790684  1.7893205 -0.80660169
## 2020-01-15  1.455349398 -0.73229841 -0.61035960  0.5980822 -0.57664423
## 2020-01-16 -0.004419775  0.78552374 -0.73638225  0.6381530  0.58756665
##        ...
## 2023-12-15  2.999906856 -3.73993868  5.97197914 -0.9454815 -0.43353582
## 2023-12-18 -0.727177114  0.91970350  1.69473655  0.6288417  1.70871482
## 2023-12-19 -1.436328937  0.95720850  4.44979167 -1.1904315  1.83045419
## 2023-12-20  0.316469918  0.97015243 -4.75796986 -1.9209561  2.40192973
## 2023-12-21  0.485436482 -5.63433958  1.73334475 -0.8756631  1.77506728
## 2023-12-22 -0.248646730  1.22465116 -0.54006026 -1.7683044 -1.53953427
## 2023-12-26 -1.314329549 -3.69416897 -1.28308480 -1.0101782  2.64738201
## 2023-12-27  1.886857627 -3.43338341  3.64921204 -0.2620257  0.53543057
## 2023-12-28  0.411610484 -4.16498595  1.20646512  2.0279306 -0.62059767
## 2023-12-29  2.698047250 -3.37622407 -1.94321675 -1.9391106 -1.32746753
##                     IEF         AGG        BCI         GLD
## 2020-01-03  0.669516425  0.07725327 -0.5296680  0.37292102
## 2020-01-06 -0.130533790  0.03926506  0.4837323 -0.64570436
## 2020-01-07 -0.146889237 -0.01501335  0.5431510  0.23652949
## 2020-01-08 -0.280982549 -0.57629081  0.7570415  0.30738138
## 2020-01-09 -0.093729901 -0.50304078 -1.7509161  0.56585533
## 2020-01-10  0.167730946  0.10837998  0.6991889  0.43364340
## 2020-01-13 -0.058909595 -0.13683137  1.3044656  0.93663088
## 2020-01-14  0.171778260  0.39470613 -0.1607149  0.30657007
## 2020-01-15  0.234776711 -0.19192850  1.5501179 -0.32490981
## 2020-01-16 -0.235912060  0.34936130  0.4016166  0.34737903
##        ...
## 2023-12-15 -0.073076736  0.39205056  0.6306695  1.01727791
## 2023-12-18 -0.063124879 -0.55406086  0.3328010 -0.52713146
## 2023-12-19  0.008950552 -0.40420652  1.2506255  0.14996139
## 2023-12-20  0.049362128 -0.25166003  0.8042647  1.45678836
## 2023-12-21 -0.034913779 -0.48896298  0.4376809  0.05420196
## 2023-12-22  0.068029134 -0.04943559 -1.8615589  0.29499459
## 2023-12-26 -0.270737113 -0.91387031  2.0792734  0.80607124
## 2023-12-27  0.081002606  0.17348990  4.7456171 -0.53366125
## 2023-12-28 -0.156381640  0.07708772  2.4785724 -0.11461582
## 2023-12-29 -0.634071926  0.18676520  0.7840497 -0.03105335
```

```
plot(portfolio_dr, main = "Daily Percentage Return", xlab = "Date", ylab = "% Return", col = rainbow(ncol(portfolio_dr)), legend.loc = "topright")
```




```
plot(bootstrapped_portfolio_dr, main = "Daily Percentage Return", xlab = "Date", ylab = "% Return", col = rainbow(ncol(bootstrapped_portfolio_dr)), legend.loc = "topright")
```



Assumptions Made

1. The block bootstrapping is applied to each asset independently.
2. The randomized block size is random and have a maximum of N/2 where N is the total number of days.
3. The randomization does not follow distribution, for instance, like Stationary Bootstrap.
4. The random noise parameters follow a Normal Distribution

References

1. [https://asbates.rbind.io/2019/03/30/time-series-bootstrap-methods/]
2. [http://sfb649.wiwi.hu-berlin.de/fedc_homepage/xplore/ebooks/html/csa/node132.html#SECTION019142000000000000000]
3. [https://medium.com/@jcatankard_76170/block-bootstrapping-with-time-series-and-spatial-data-bd7d7830681e]
4. [https://stat.ethz.ch/R-manual/R-devel/library/boot/html/tsboot.html]