Group Project on Financial Data

Initial Setup

```
knitr::opts_chunk$set(warning = FALSE, message = FALSE)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(readr)
library(quadprog)
set.seed(123)
```

Introduction

In this project, we work with financial time-series data, specifically, the prices of stocks in the S&P 500 index. Our goal in this project is to obtain an "optimal" portfolio of stocks, which we will do in three separate stages.

1. **Dimension reduction** on the stocks to obtain reduced factors/principal portfolios
2. Modelling of prices of factors using **Gaussian Process Regression**
3. Portfolio optimisation using **Markowitz Mean-Variance Optimization Model**

This full, 3 step weight estimation procedure is consolidated into the tested and documented "StockPriceMod" R package which we developed, which is available on github here. This notebook will also explain each of the three stages of the procedure on a sample of the dataset, before performing the full experiment on the year of 2020.

Dataset

We obtain a dataset of stock prices from the Standard and Poor's 500 (S&P 500) index through Yahoo Finance. The total number of dates we obtained is from 2020-01-02 to 2022-12-30, with the Open, High, Low, Close, Adjusted Close Prices (in US dollars), and the Volume and Ticker. We have a total of 499 unique Stock tickers, noting that althought the S&P 500 refers to the 500 largest companies, there may be different classes of shares/stocks issued and due to some data API issues we were not able to obtain the full stock lists.

```
# Load the data from our prepared CSV file
stock_prices <- read_csv("./data/sp500_stock_data.csv",show_col_types = FALSE)
head(stock_prices)
```

```
## # A tibble: 6 x 8
##   Date        Open  High   Low Close `Adj Close`  Volume Ticker
##   <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>   <dbl> <chr>
## 1 2020-01-02  178.  180.  177.  180         152. 3601700 MMM
## 2 2020-01-03  177.  179.  176.  178.        151. 2466900 MMM
## 3 2020-01-06  177.  179.  176.  179.        151. 1998000 MMM
## 4 2020-01-07  178.  179.  177.  178.        150. 2173000 MMM
## 5 2020-01-08  178   182.  178.  181.        153. 2758300 MMM
## 6 2020-01-09  182.  182.  180.  181.        153. 2746300 MMM
```

```
tail(stock_prices)
```

```
## # A tibble: 6 x 8
##   Date        Open  High   Low Close `Adj Close`  Volume Ticker
##   <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>   <dbl> <chr>
## 1 2022-12-22  144.  145.  142.  145.        144. 1541800 ZTS
## 2 2022-12-23  145.  146.  144.  146.        144. 1017900 ZTS
## 3 2022-12-27  146.  146.  144.  145.        144.  957900 ZTS
## 4 2022-12-28  145.  147.  144.  144.        143. 1443900 ZTS
## 5 2022-12-29  145.  149.  145.  148.        147. 1298900 ZTS
## 6 2022-12-30  147.  148.  145.  147.        145. 1249500 ZTS
```

```
length(unique(stock_prices$Ticker)) # Number of unique stocks
```

```
## [1] 499
```

For simplicity, we focus on the dates between 2020-01-02 to 2020-12-31, i.e., all of the days in the year 2020. In particular, due to the effects of the COVID pandemic during this time, we note that the stock prices at this time were highly unstable and exhibited high volatility, and so this time period may serve as an interesting testing ground for our method. We furthermore focus ourselves only on the end of day closing prices for the stocks.

```
# Using all of 2020
start_date <- as.Date("2020-01-02")
end_date <- as.Date("2020-12-31")

# Filter for specific date range and select only Date, Close, and Ticker
stock_prices_filtered <- stock_prices %>%
  filter(Date >=  start_date & Date <= end_date) %>%
  select(Date, Close, Ticker)

# Reshape data to a wide format
wide_data <- stock_prices_filtered %>%
  pivot_wider(names_from = Ticker, values_from = Close)

wide_data <- wide_data[, !apply(is.na(wide_data), 2, any)]
head(wide_data)
```

```
## # A tibble: 6 x 495
##   Date         MMM   AOS   ABT  ABBV   ACN  ADBE   AMD   AES   AFL     A   APD
##   <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2020-01-02  180   47.8  86.9  89.6  210.  334.  49.1  20.0  53.3  85.9  231.
## 2 2020-01-03  178.  47.3  85.9  88.7  210.  332.  48.6  19.8  53.0  84.6  226
## 3 2020-01-06  179.  47.7  86.3  89.4  208.  334.  48.4  20.0  52.8  84.8  226.
## 4 2020-01-07  178.  47.3  85.9  88.9  204.  333.  48.2  20.1  52.3  85.1  227.
## 5 2020-01-08  181.  47.3  86.2  89.5  204.  338.  47.8  20.1  52.5  85.9  228.
## 6 2020-01-09  181.  47.0  86.4  90.2  206.  340.  49.0  20.3  52.5  87.3  234.
```

```
## # i 483 more variables: AKAM <dbl>, ALB <dbl>, ARE <dbl>, ALGN <dbl>,
## #   ALLE <dbl>, LNT <dbl>, ALL <dbl>, GOOGL <dbl>, GOOG <dbl>, MO <dbl>,
## #   AMZN <dbl>, AMCR <dbl>, AEE <dbl>, AAL <dbl>, AEP <dbl>, AXP <dbl>,
## #   AIG <dbl>, AMT <dbl>, AWK <dbl>, AMP <dbl>, AME <dbl>, AMGN <dbl>,
## #   APH <dbl>, ADI <dbl>, ANSS <dbl>, AON <dbl>, APA <dbl>, AAPL <dbl>,
## #   AMAT <dbl>, APTV <dbl>, ACGL <dbl>, ADM <dbl>, ANET <dbl>, AJG <dbl>,
## #   AIZ <dbl>, T <dbl>, ATO <dbl>, ADSK <dbl>, ADP <dbl>, AZO <dbl>, ...
```

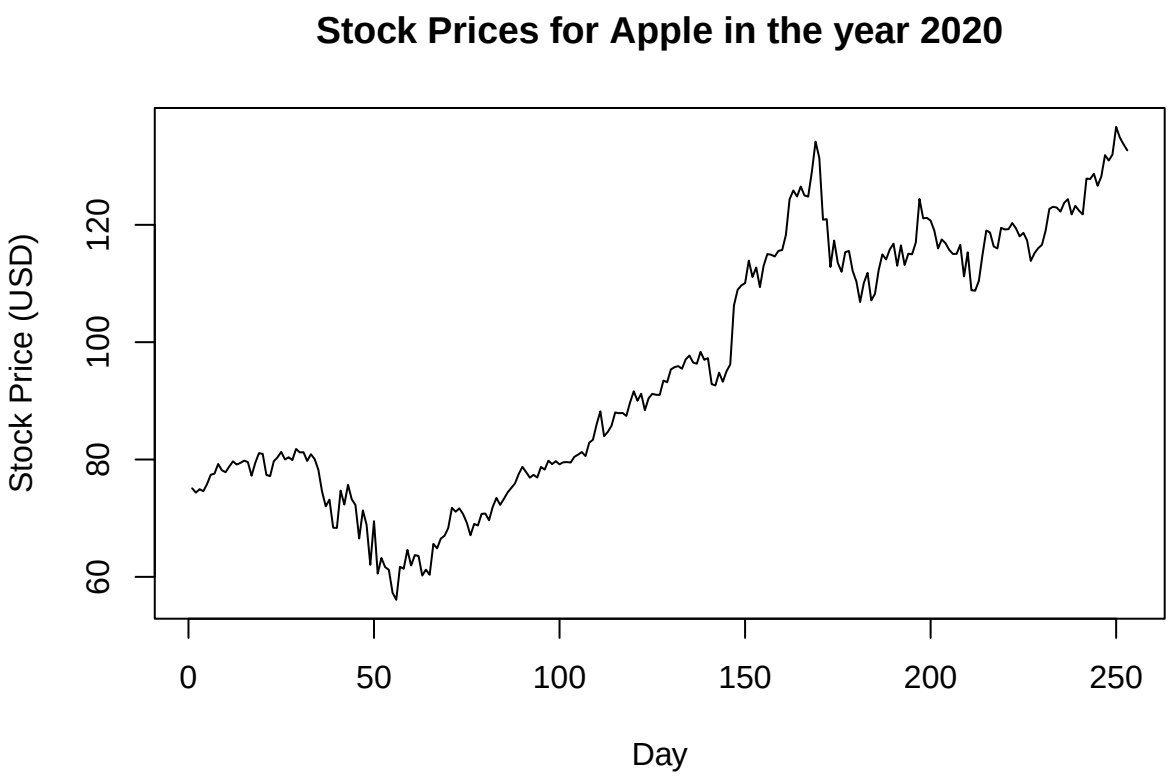
```r
nrow(wide_data)
```

```
## [1] 253
```

To help in our PCA and further analysis we pivot our data to a "wide" format, i.e., where the rows are the time series days and the columns are the individual stocks. In total, we have an $\mathbb{R}^{253\times 495}$ data matrix, where 253 are the number of trading days in the year 2020 and 495 are the number of stocks.

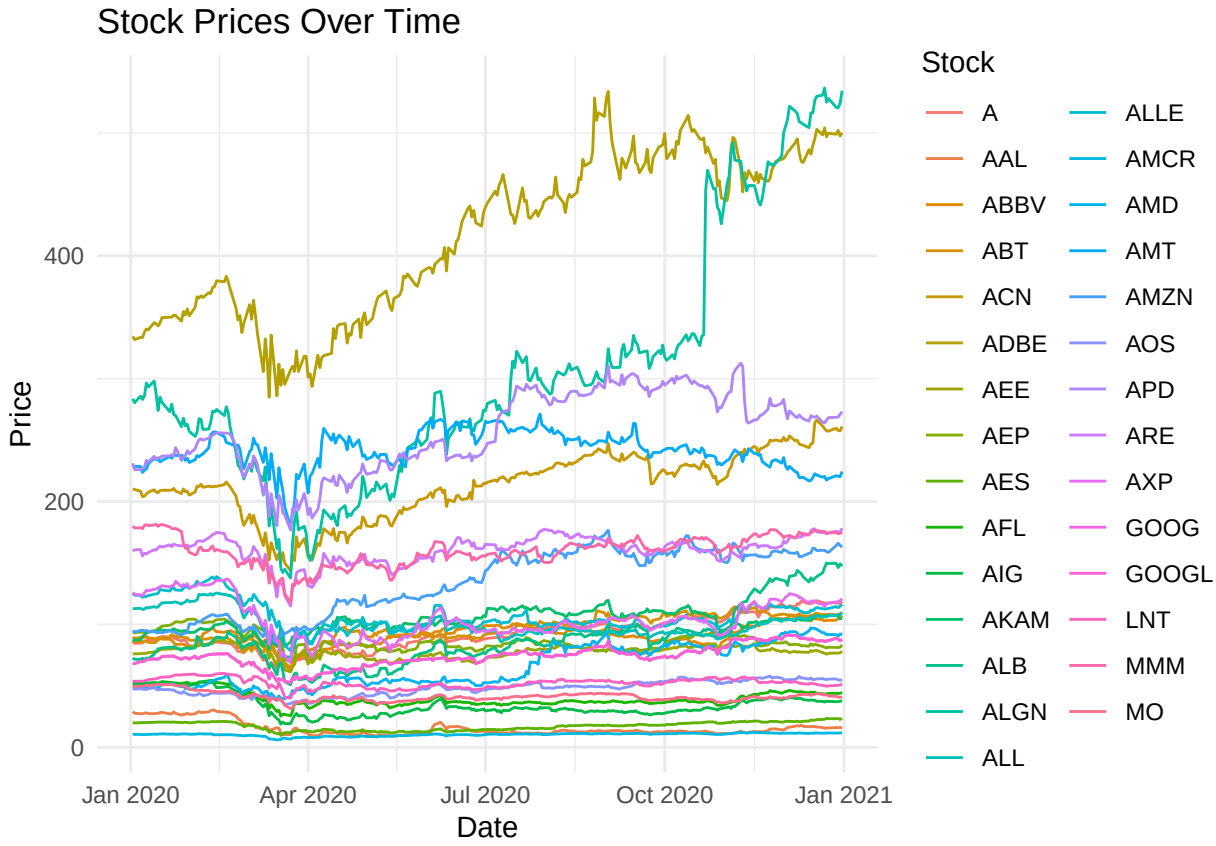
We can quickly look at the stock prices for Apple (ticker $AAPL).

```r
plot.ts(wide_data$AAPL, ylab = "Stock Price (USD)", xlab= "Day")
title("Stock Prices for Apple in the year 2020")
```




```r
long_data <- wide_data[,1:30] %>%
  pivot_longer(
    cols = -Date,  # -Date means to keep the Date column as is
    names_to = "Stock",  # This will be the new column for stock names
    values_to = "Price"  # This will be the new column for stock prices
  )

ggplot(long_data, aes(x = Date, y = Price, color = Stock)) +
  geom_line() +
  theme_minimal() +
  labs(title = "Stock Prices Over Time", x = "Date", y = "Price")
```



Markowitz' Mean-Variance Model

<! https://sites.math.washington.edu/~burke/crs/408/fin-proj/mark1.pdf –>

We define the **return** of an asset (our stock) which has price P_t at time t and price P_{t-1} at time $t-1$ as being $R_t = \frac{P_t}{P_{t-1}}$.

The well-known **Markowitz's Mean-Variance Optimisation** is the basis of our Portfolio strategy. In this setting, the return on assets are modelled as random variables, and the goal is to choose a portfolio of **weighting factors** through an optimality criterion. Specifically, we have n stocks which we weight in our portfolio with a set of weighting factors $\{w_i\}_{i=1}^p$. The idea then is to maximize the expected return and to minimize the volatility at the same time. Mathematically speaking, we aim to maximize

$$\mathbb{E}[R] = \sum_i w_i \mathbb{E}[R_i]$$

subject to minimize

$$\sigma^2 = \sum_{i,j} w_i w_j \sigma_i \sigma_j \rho_{ij}$$

where $\{R_i\}_i$ is the percentage return on the underlying assets; $\{w_i\}_i$ is the respective proportion that sum to 1; $\{\sigma_i\}$ is the standard deviation of the return on the ith underlying asset and ρ_{ij} is the correlation between ith return and jth return.

Portfolio Strategy

We wish to get estimates of the above objects, and since our data is time series and therefore not independent and identically distributed, we will update our estimates sequentially. Also, since there are too many (in our

case 500 stocks) variables, we will use dimension reduction techniques. The reduced factors could also vary from time to time. Combined these observations, we do the following strategy:

1. Once we decide to reallocate the portfolio upon meeting some predefined general criteria C, do
2. Update the reduced factors where each factor is a linear combination of the underlying stocks.
3. Update the estimates of expectation and covariance between reduced factors.
4. Get the new weights $\{w_i\}$ by maximizing $\mathbb{E}[R]$ subject to minimize σ^2 and weights sum to 1.

Principal Components Analysis

We will use Principal Component Analysis(PCA) as a dimension reduction technique.z

```
#Looking at 10 days
start_day <- 100
end_day <- 110

# Calculate logarithmic returns
log_returns_monthly <- wide_data[start_day:end_day, ] %>%
  mutate(across(-Date, ~log(. / .[1]) + 1)) %>%
  select(-Date)

log_returns_monthly
```

```
## # A tibble: 11 x 494
##      MMM   AOS   ABT  ABBV   ACN  ADBE   AMD   AES   AFL     A   APD  AKAM   ALB
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  1     1     1     1     1     1     1     1     1     1     1     1     1
##  2  1.04  1.05  1.01  0.992 1.02  0.996 0.992 0.991 1.03  1.00  0.989 0.999 1.00
##  3  1.03  1.04  1.03  0.992 1.03  1.01  0.972 1.00  1.01  1.01  1.01  1.03  1.01
##  4  1.03  1.06  1.06  1.02  1.03  1.03  1.01  0.969 1.01  1.02  1.02  1.06  1.02
##  5  1.02  1.05  1.04  1.00  1.03  1.03  1.01  0.974 1.02  1.04  1.02  1.05  1.03
##  6  1.04  1.07  1.03  1.00  1.05  1.04  1.01  1.02  1.03  1.05  1.03  1.05  1.05
##  7  1.06  1.11  1.02  1.00  1.05  1.03  0.991 1.06  1.05  1.05  1.04  1.03  1.05
##  8  1.07  1.12  0.986 1.03  1.03  1.02  0.989 1.07  1.08  1.06  1.03  1.02  1.07
##  9  1.10  1.15  1.00  1.03  1.06  1.04  0.998 1.10  1.11  1.05  1.04  0.989 1.12
## 10  1.09  1.13  1.03  1.04  1.07  1.05  0.996 1.15  1.15  1.05  1.05  1.03  1.15
## 11  1.09  1.09  1.01  1.06  1.05  1.05  1.06  1.07  1.10  1.04  1.05  1.03  1.13
## # i 481 more variables: ARE <dbl>, ALGN <dbl>, ALLE <dbl>, LNT <dbl>,
## #   ALL <dbl>, GOOGL <dbl>, GOOG <dbl>, MO <dbl>, AMZN <dbl>, AMCR <dbl>,
## #   AEE <dbl>, AAL <dbl>, AEP <dbl>, AXP <dbl>, AIG <dbl>, AMT <dbl>,
## #   AWK <dbl>, AMP <dbl>, AME <dbl>, AMGN <dbl>, APH <dbl>, ADI <dbl>,
## #   ANSS <dbl>, AON <dbl>, APA <dbl>, AAPL <dbl>, AMAT <dbl>, APTV <dbl>,
## #   ACGL <dbl>, ADM <dbl>, ANET <dbl>, AJG <dbl>, AIZ <dbl>, T <dbl>,
## #   ATO <dbl>, ADSK <dbl>, ADP <dbl>, AZO <dbl>, AVB <dbl>, AVY <dbl>, ...
```

```
# Perform PCA on logarithmic returns
pca_result <- prcomp(log_returns_monthly, scale = TRUE, center= TRUE)


explained_variance <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumulative_variance <- cumsum(explained_variance)

# Find the number of components that explain at least the threshold variance
num_components <- which(cumulative_variance >= 0.9)[1]
print(num_components)
```

```
## [1] 4
```

```
# Return the principal components
y <- pca_result$x[, 1:num_components]
```

State-space Model and Gaussian Process Regression(GPR) Modelling

As mentioned in the previous sections, we do PCA first to get independent components. In this section we will try to predict the market value change in short future. Each components' behaviour is viewed as independent time series, and we will fit each time series with the simple linear state-space model:

$$X_t = \phi X_{t-1} + V$$

$$Y_t = X_t + W$$

where $W \sim \mathcal{N}(0, \sigma_w^2)$ and $V \sim \mathcal{N}(0, \sigma_v^2)$.

Kalman Filter

Note that using Kalman Filter with initiate distribution $\mathcal{N}(0, 1)$ is equivalent to a Gaussian process regression with the following relation: $\phi = \exp(-\frac{1}{\gamma})$ and $\sigma_v^2 = 1 - \exp(-\frac{2}{\gamma})$.

The `Kalman` function takes inputs ϕ, σ_v^2, σ_w^2, m_0, σ_0^2 and the number of prediction days n, it returns a the predicted mean, predicted variance, updated mean and updated variance. It is a general Kalman Filter funtion and will be then implemented using our relation to GPR.

```
kalman <- function(y,phi,Sigmav,Sigmaw,m0,Sigma0,n=2){

  T <- length(y)

  #initialization
  mu.p <- rep(NA,T+n)
  Sigma.p <- rep(NA,T+n)
  mu.f <- rep(NA,T)
  Sigma.f <- rep(NA,T)



  #forward recursion time1
  mu.p[1] <- m0
  Sigma.p[1] <- Sigma0
  mu.f[1] <- m0 + (y[1]-m0)*(Sigma0/(Sigma0+Sigmaw))
  Sigma.f[1] <- Sigma0-(Sigma0^2/(Sigma0+Sigmaw))

  #forward recursion time 2:T
  for (t in 2:T){

    #prediction
    mu.p[t] <- phi*mu.f[t-1]
    Sigma.p[t] <- phi^2 * Sigma.f[t-1] + Sigmaw

    #update
    deno <- Sigmaw + Sigma.p[t]
    mu.f[t] <- Sigmaw*mu.p[t]/deno + Sigma.p[t]*y[t]/deno
```

```
    Sigma.f[t] <- Sigmaw*Sigma.p[t]/deno
  }
  #predict for T+1:T+n
  for (t in (T+1):(T+n)){
    if (t == T+1){
      mu.p[t] <- phi*mu.f[t-1]
      Sigma.p[t] <- phi^2 * Sigma.f[t-1] + Sigmaw
    }
    else{
      mu.p[t] <- phi*mu.p[t-1]
      Sigma.p[t] <- phi^2 * Sigma.p[t-1] + Sigmaw
    }
  }
  return (list(mu.f=mu.f,Sigma.f=Sigma.f,mu.p=mu.p,Sigma.p=Sigma.p))
}
```

We then implement it with GPR:

```
kf.gp <- function(y,gamma,Sigmaw,m0=0,Sigma0=1,n=2){
  T=length(y)
  #update Sigmav and phi
  phi <- exp(-1/gamma)
  Sigmav <- 1-exp(-2/gamma)
  result <- kalman(y,phi=phi,Sigmav=Sigmav,Sigmaw=Sigmaw,m0=m0,Sigma0=Sigma0,n)

  return (list(mu.p=result$mu.p, Sigma.p=result$Sigma.p,
               mu.f=result$mu.f, Sigma.f=result$Sigma.f))

}

kf.loglikelihood1 <- function(y,mu.p,Sigma.p,Sigmaw,m0=0,Sigma0=1){
  T <- length(y)
  likelihood <- rep(NA,T)

  #at time 1
  likelihood[1] <- log(dnorm(y[1],mean=m0,sd = sqrt(Sigma0 + Sigmaw)))

  #time 2:T
  for (t in 2:T){
    likelihood[t] <- log(dnorm(y[t],mean=mu.p[t],sd=sqrt(Sigmaw+Sigma.p[t])))
  }
  return (sum(likelihood))
}
```

There are two functions here: • `kf.gp` simply applies the Kalman Filter on observed y, with σ_v^2 and ϕ as functions of hyper parameter γ, computing the predictive and updated distributions. • `kf.loglikelihood` computes the loglikelihood of the observed y in a iteration manner by noting

$$\log(p(y_{1:T})) = p(y_1) + \sum_{t=2}^{T} p(y_t|y_{1:t-1})$$

and

$$p(y_t|y_{1:t-1}) = \mathcal{N}(y_t; m_{t|t-1}, \sigma^2_{t|t-1} + \sigma^2_w)$$

Optimization to get hyperparameters' MLE

In real life we never observe the hyper parameters γ and σ_w^2. We propose using `optim` to optimize against σ_w^2 and γ against the loglikehood computed using `kf.loglikelihood`.

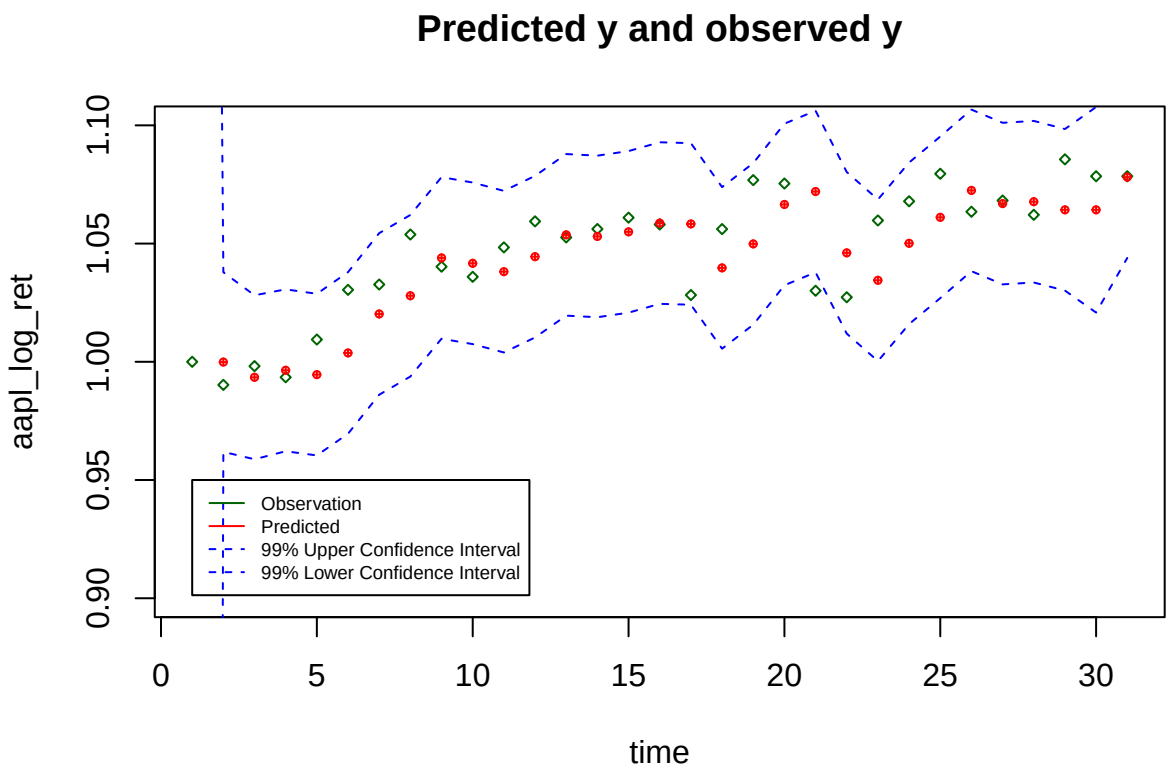
```
kf.loglikelihood <- function(y,gamma,Sigmaw,m0=0,Sigma0=1){
  o <- kf.gp(y=y,gamma=gamma,Sigmaw=Sigmaw,m0=m0,Sigma0=Sigma0)
  mu.p <- o$mu.p
  Sigma.p <- o$Sigma.p
  result <- kf.loglikelihood1(y=y,mu.p=mu.p,Sigma.p=Sigma.p,Sigmaw=Sigmaw,m0=m0,Sigma0=Sigma0)
  return (result)
}

optim_parm <- function(y){
  opt_param <- optim(par = c(5,0.5),
                     fn = function(parm) -1*kf.loglikelihood(y,parm[1],  parm[2]))
  return(list(gamma = opt_param$par[1],
              Sigmaw=opt_param$par[2]))
}
```

```
log_returns_monthly <- wide_data[1:30, ] %>%
  mutate(across(-Date, ~log(. / .[1]) + 1)) %>%
  select(-Date)

aapl_log_ret <- log_returns_monthly$AAPL
optim_hyperparam = optim_parm(aapl_log_ret)
results = kf.gp(aapl_log_ret,gamma=optim_hyperparam$gamma,Sigmaw = optim_hyperparam$Sigmaw,n=0)
mu.p <- results$mu.p
Sigma.p <- results$Sigma.p
se.p <- sqrt(Sigma.p)

alpha=0.01
cv99 = qnorm(1-alpha/2)
CIupper.p <- mu.p + cv99*se.p
CIlower.p <- mu.p - cv99*se.p
time = 1:(length(aapl_log_ret)+1)
aapl_log_ret <- c(aapl_log_ret,aapl_log_ret[length(aapl_log_ret)])
plot(time,aapl_log_ret,cex=0.5,col='darkgreen',pch=5,ylim=c(.9,1.1),main='Predicted y and observed y')
points(time,mu.p,cex=0.5,col='red',pch=10)
points(time,CIupper.p,col='blue',type ='l',lty=2,lwd=1)
points(time,CIlower.p,col='blue',type ='l',lty=2,lwd=1)
legend(1,.95,legend= c('Observation','Predicted','99% Upper Confidence Interval','99% Lower Confidence
```



Portfolio Optimisation

```r
#Only predict 1 day

# Initialize an empty dataframe to store the time series data
time_series_df <- data.frame(matrix(ncol = 2, nrow = num_components))
rownames(time_series_df) <- paste("Component", 1:num_components, sep = "")
colnames(time_series_df)[1] <- "mu"
colnames(time_series_df)[2] <- "sigma2"


for (m in 1:num_components) {
  # Obtain optimized hyperparameters for the m-th component
  optim_hyperparam <- optim_parm(y[, m])

  # Run the Gaussian Process with Kalman filter
  gp_result <- kf.gp(y[, m], optim_hyperparam$gamma, optim_hyperparam$Sigmaw, n = 0)

  # Add the time series to the dataframe
  time_series_df[m, ] <- c(tail(gp_result$mu.p, 1) , tail(gp_result$Sigma.p, 1))
}

time_series_df
```

```
##                    mu     sigma2
## Component1 17.5750938 144.609443
## Component2 -0.8638087  41.518120
## Component3  2.2988612  10.169838
## Component4 -0.6473105   9.574817
```

Readjust the predicted cumulative returns ($r_{T+1} = 1 + \log \frac{P_{T+1}}{P_0} = 1 + \log(P_{T+1}) - \log(P_0)$) as follows:

$$z_{T+1=} \log \frac{P_{T+1}}{P_T} = \log(P_{T+1}) - \log(P_T) - \log(P_0) + \log(P_0) = r_{T+1} - r_T$$

As we use the predicted daily log returns z_{T+1} for our portfolio optimisation.

```
pred_returns <- time_series_df$mu - tail(y,1) #z_T+1
#covar_mat <- diag(pca_result$sdev[1:num_components] ** 2)
covar_mat <- diag(time_series_df[[2]])
Amat <- matrix(0, num_components, num_components)
Amat[,1] <- 1
bvec <- rep(0,num_components)
bvec[1] <- 1
lambda <- 1

QP_result <- solve.QP(2*covar_mat, lambda * pred_returns, matrix(1, nrow=num_components,ncol=1), c(1), r
QP_result$solution
```

```
## [1] 0.02278848 0.12481907 0.19105017 0.66134228
```

Experiments

We utilise our developed package, "StockPriceMod" which is available on github here to perform our portfolio optimisation over the year 2020. This package can be install with the command `devtools::install_github('Shermjj/StockPriceMod')`.

```
rm(list = ls()) #First clear the environment of the previously defined functions due to conflicts
library(StockPriceMod)
library(tidyverse)
library(readr)
library(quadprog)
set.seed(123)
```