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>, ...
```

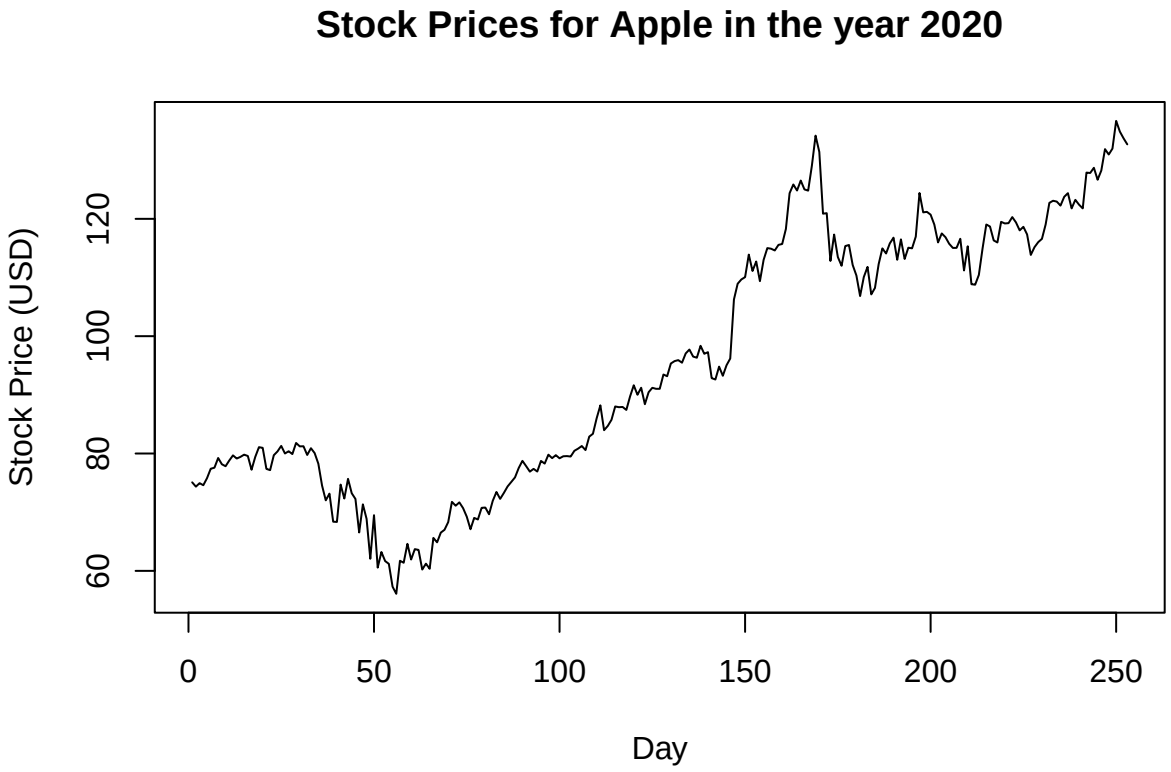
```
nrow(wide_data)
```

```
## [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).

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



Data Preprocessing

https://gregorygundersen.com/blog/2022/02/06/log-returns/

As opposed to raw prices, it is usually standard in financial analysis to transform to the **return** of an asset (our stock), which we define as being $r_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1$, where the stock has price P_t at time t and price P_{t-1} at time $t-1$. In order to obtain unbounded support of our returns, we further transform to the **log-returns**, which is defined as $z_t = \log(1 + r_t) = \log(\frac{p_t}{p_{t-1}})$. Finally, in our case, due to the use of the SSM modelling, we will use the **cumulative log-returns** instead, which we define as:

$$r_{0:t} = 1 + \log \frac{p_t}{p_0} = 1 + \log p_t - \log p_0$$

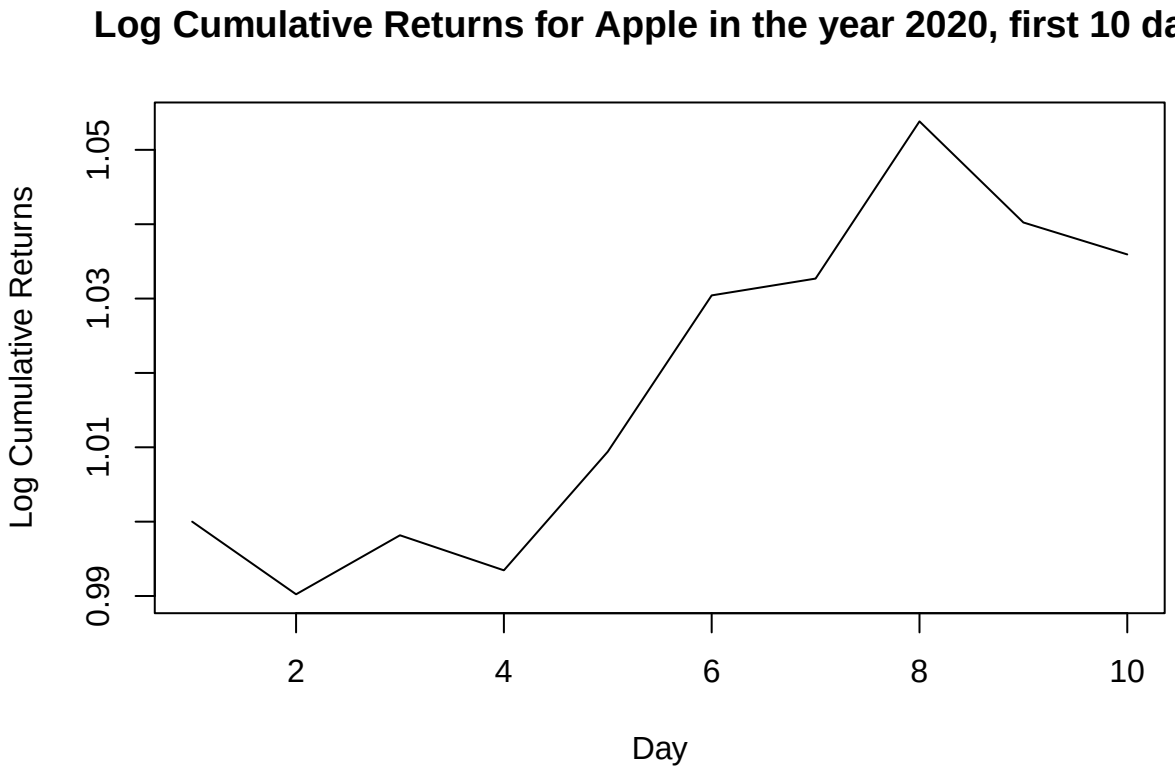
```r
#Looking at first 10 days of the year
start_day <- 1
end_day <- 10

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

head(log_cu_returns)
```

```
## # A tibble: 6 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 0.991 0.991 0.988 0.990 0.998 0.992 0.990 0.988 0.993 0.984 0.978 0.995 0.986
## 3 0.992 0.997 0.993 0.998 0.992 0.998 0.985 1     0.990 0.987 0.977 0.999 0.984
## 4 0.988 0.991 0.987 0.993 0.970 0.997 0.983 1.00  0.981 0.990 0.981 1.03  0.998
## 5 1.00  0.989 0.991 1.00  0.972 1.01  0.974 1.00  0.984 1.00  0.987 1.04  0.980
## 6 1.01  0.985 0.994 1.01  0.981 1.02  0.997 1.01  0.984 1.02  1.01  1.06  1.00
## # 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>, ...
```

```r
plot.ts(log_cu_returns$AAPL, ylab = "Log Cumulative Returns", xlab= "Day")
title("Log Cumulative Returns for Apple in the year 2020, first 10 days")
```



As we can see, the log cumulative returns are significantly more stable and easier to compare across stocks as we use the same starting base for each time window.

Principal Components Analysis

We will use Principal Component Analysis(PCA) as a dimension reduction technique. Specifically, our goal here is to find some linear dependencies between our stocks, known as **principal components**, which are orthogonal and maximise the variance. We will use the `prcomp` function from the base R `stats` package. We note that we also need to do centering, and optionally normalisation of the variables could be performed. We find that normalisation helps in the numerical stability of our procedure, so we perform that here, done automatically through the `scale` and `center` arguments in `prcomp`.

As we are in the $n < p$ setting, the number of PCs which we obtain is limited to $n = 10$. We perform PCA and show some of the components of the 10 PCs.

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

head(pca_result$rotation)
```

```
##               PC1          PC2          PC3          PC4          PC5
## MMM   0.040509000  0.006555956  0.088461385 -0.006869578  0.009402407
## AOS  -0.010308726  0.005317343  0.049042489  0.092321096 -0.107836376
## ABT  -0.004450003  0.048321850 -0.030177022 -0.088475842  0.141678441
## ABBV -0.017639096 -0.013556547  0.040804255 -0.038729233  0.176263971
## ACN  -0.002574362  0.077825551  0.009647837  0.116963859 -0.017359079
## ADBE  0.061006932  0.001891109  0.033103945  0.002229658 -0.019466467
##               PC6         PC7          PC8           PC9          PC10
## MMM  -0.007785326  0.06584364  0.082718141 -0.0349762301  0.063060364
## AOS  -0.078708715  0.05027738 -0.055230154  0.1238552029 -0.006809753
## ABT   0.001763373 -0.01379931  0.017620738  0.0433633798  0.013664600
## ABBV -0.033201889 -0.01669778  0.013114268  0.0490620670 -0.083451076
## ACN  -0.014926790  0.01678336  0.005882421  0.0100194337  0.013987827
## ADBE -0.031836784  0.02714022  0.014650997 -0.0005192823 -0.063119661
```

As we are interested in both reducing the number of PCs (dimension reduction) and maximising the amount of information, we face a tradeoff in determining the number of PCs to use. We show both the cumulative variance for the 11 PCs and a Scree Plot showing the proportion of variance explained for each PC. For a simple rule of thumb, we choose the number of PCs using the cutoff of 90% of cumulative variance explained, which in this case is the first 5 PCs.

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

```
##  [1] 0.4880975 0.6749094 0.7958053 0.8577381 0.9045194 0.9381545 0.9629781
##  [8] 0.9820628 1.0000000 1.0000000
```

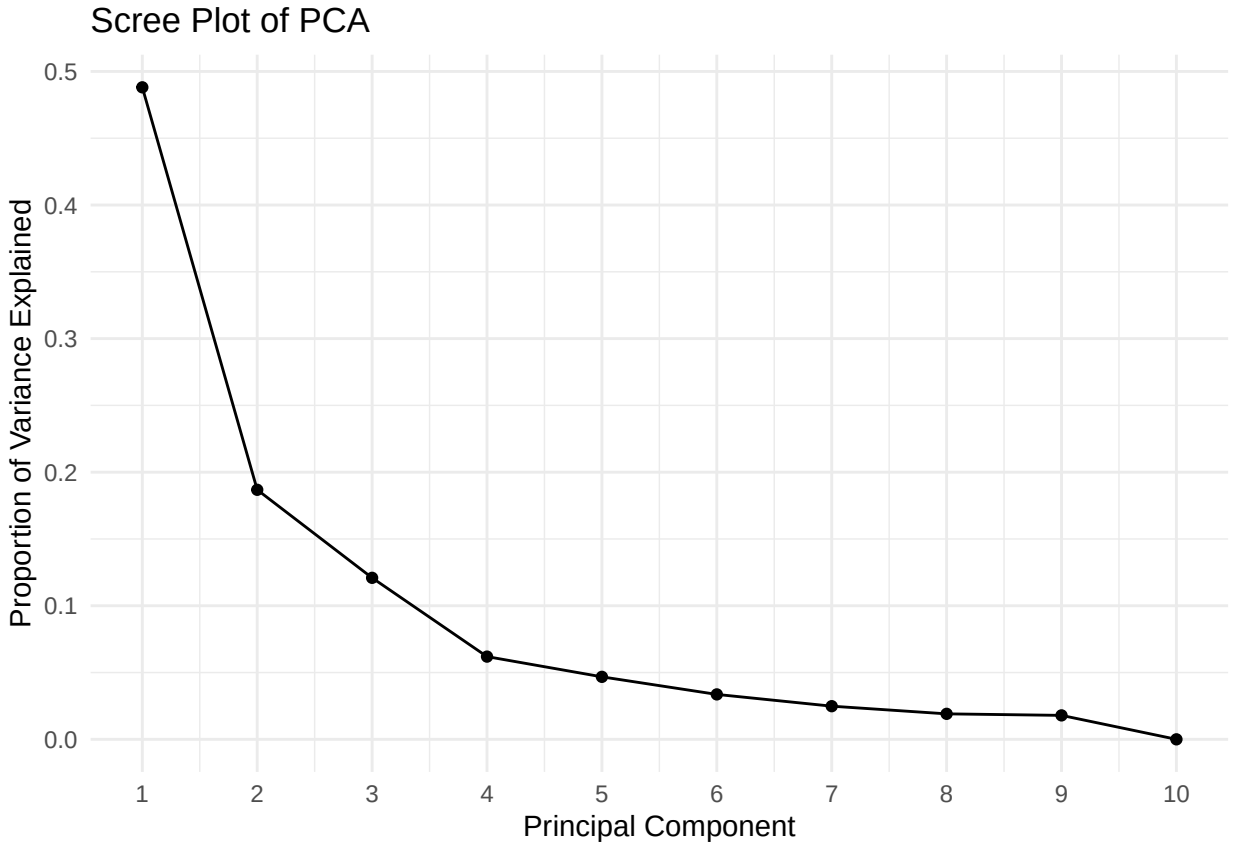
```
# Find the number of components that explain at least the threshold variance
num_components <- which(cumulative_variance >= 0.9)[1]
print(num_components)
```

```
## [1] 5
```

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

```r
pca_df <- tibble(
  Principal_Component = seq_along(explained_variance),
  Variance_Explained = explained_variance
)

ggplot(pca_df, aes(x = Principal_Component, y = Variance_Explained)) +
  geom_line() +
  geom_point() +
    scale_x_continuous(breaks = seq_along(explained_variance)) +
  theme_minimal() +
  labs(title = "Scree Plot of PCA", x = "Principal Component", y = "Proportion of Variance Explained")
```



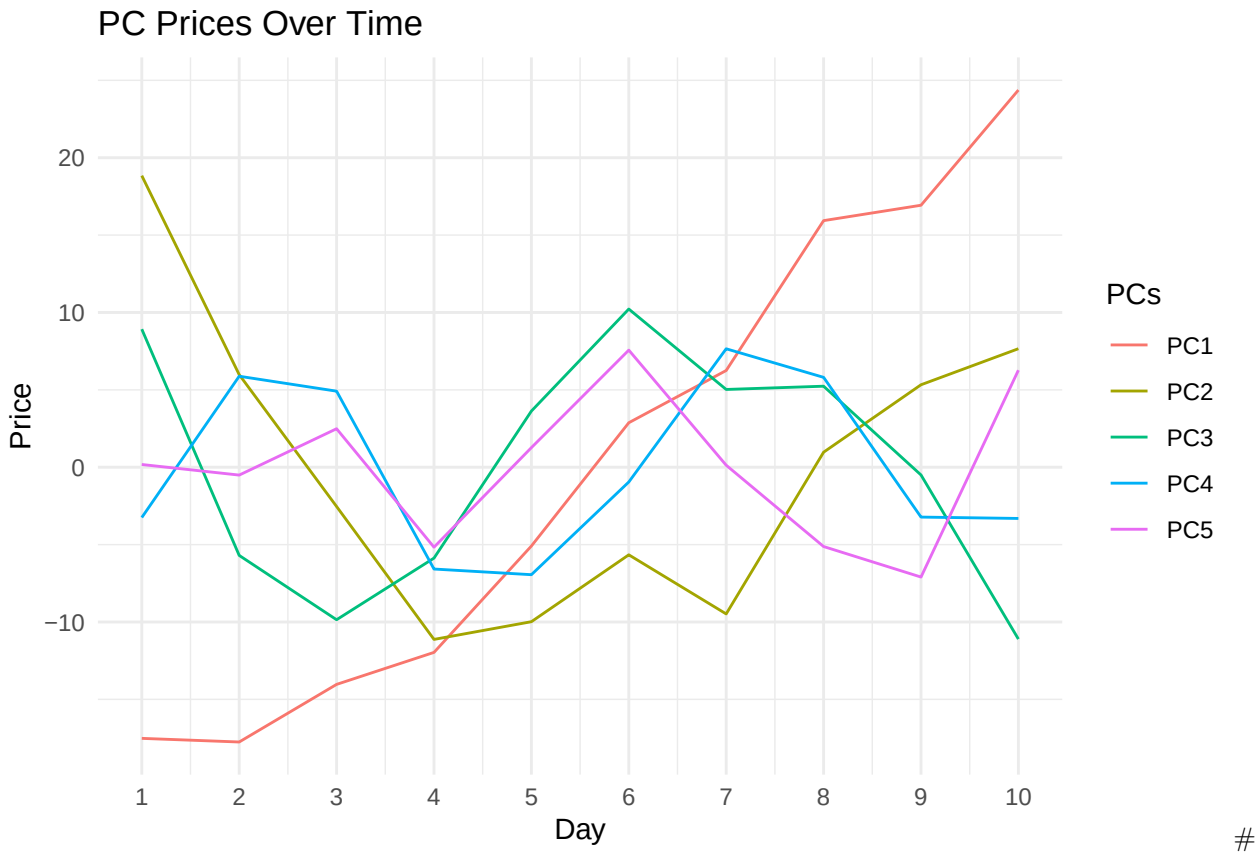
As we can see from the Scree Plot, there is a quick dropoff in the proportion of variance explained past the first few PCs, implying that we do not need much PCs to retain the information in the dataset.

We further plot a time series of the first 5 PCs for the 10 days.

```r
y_df <- data.frame(
  Date = 1:10, y
)

long_data <- y_df %>%
  pivot_longer(
    cols = -Date,  # -Date means to keep the Date column as is
    names_to = "PCs",  # This will be the new column for stock names
    values_to = "Price"  # This will be the new column for stock prices
  )
```

```r
ggplot(long_data, aes(x = Date, y = Price, color = PCs)) +
  geom_line() +
  scale_x_continuous(breaks = 1:max(long_data$Date)) +
  theme_minimal() +
  labs(title = "PC Prices Over Time", x = "Day", y = "Price")
```



#

SSM and GPR Modelling In this section we will try to predict the log returns of each PCs using a State Space Model, or equivalently, the Gaussian Process Regression. Each PC's behaviour is viewed as an independent time series (due to the orthogonality of the PCs), 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

We note that using the Kalman Filter with initial 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 function and will be then implemented using our relation to GPR.

```r
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 log-likelihood 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 with the log-likelihood 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]))
}
```

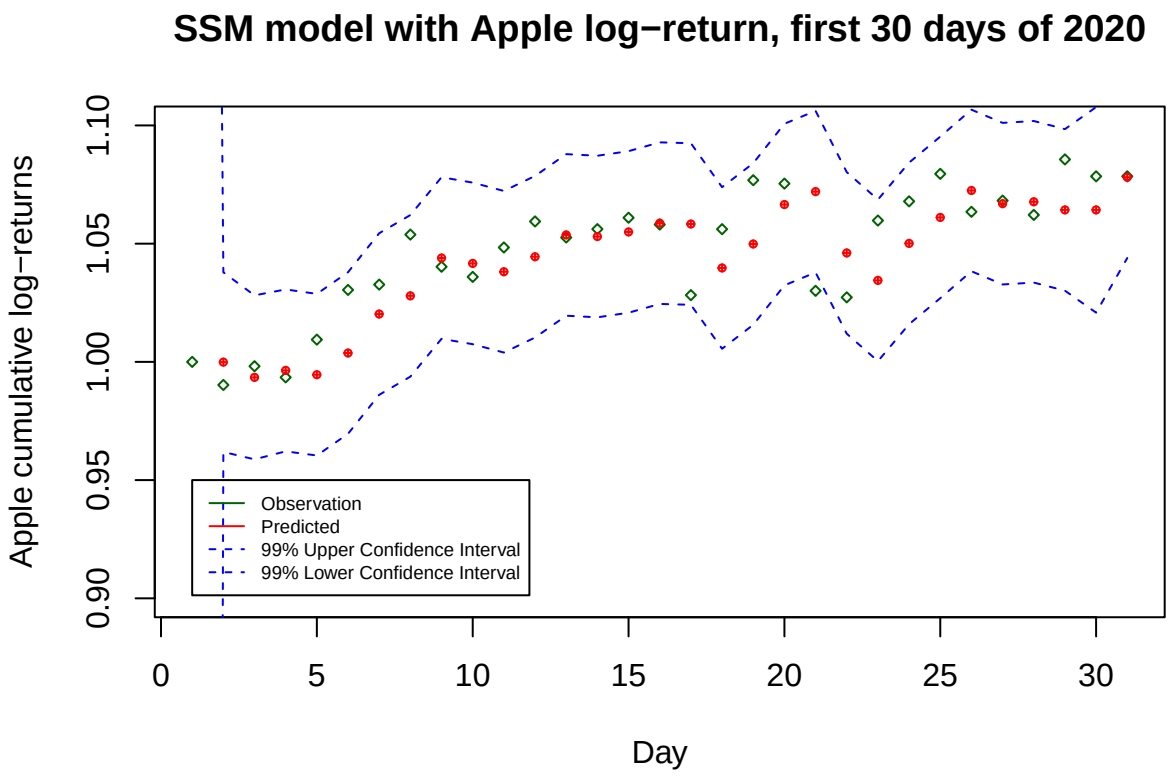
We plot here the SSM/GPR modelling for the log-returns for the first 30 days of Apple stock, along with the 99% upper and lower confidence intervals.

```
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
```

```r
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='SSM model with Apple log-retu
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

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

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.

We will use the same 5 PCs that we obtained previously in PCA, and over each PC, model with our SSM individually. We predict for the next day and obtain the mean and variance of the prediction.

```
# 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("PC", 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
## PC1  1.869980e+01 100.92437
## PC2  5.605874e+00  80.23354
## PC3 -6.203526e+00  43.89531
## PC4 -4.155165e-01  14.05056
## PC5  8.510026e-07  10.48528
```

We note here that we readjust the predicted cumulative log-returns ($r_{0:T+1} = 1 + \log \frac{P_{T+1}}{P_0} = 1 + \log(P_{T+1}) - \log(P_0)$) back to the log-returns as follows:

$$z_{T+1} = \log \frac{P_{T+1}}{P_T} = \log(P_{T+1}) - \log(P_T) - \log(P_0) + \log(P_0) = r_{0:T+1} - r_{0:T}$$

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(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), m
QP_result$solution
```

```
## [1] 0.02716953 0.05674442 0.18293053 0.50007058 0.23308494
```

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)
```