TP2

Objectif du TP

Ce TP consiste en deux parties indépendantes afin d'illustrer les applications de "Machine Learning" dans la gestion d'actifs:

- Modèle de régression linéaire pour prédire le rendement futur du stock Amazon. (Voir CM3)
- Problème d'optimisation de Markowitz avec la régularisation L2. (Voir CM4)

Partie I: Modèle de régression linéaire pour prédire le rendement futur du stock Amazon

Cette partie consiste à construire le modèle de régression linéaire pour prédire le rendement futur du stock Amazon (NYSE). Mathématiquement, nous allons construire un modèle linéaire:

$$Y_{t,10} = \sum_{k=1}^{S} \beta_t F_t^{(k)} + \epsilon_t \ , \quad \epsilon_t \sim N\left(0, \sigma_t^2\right)$$

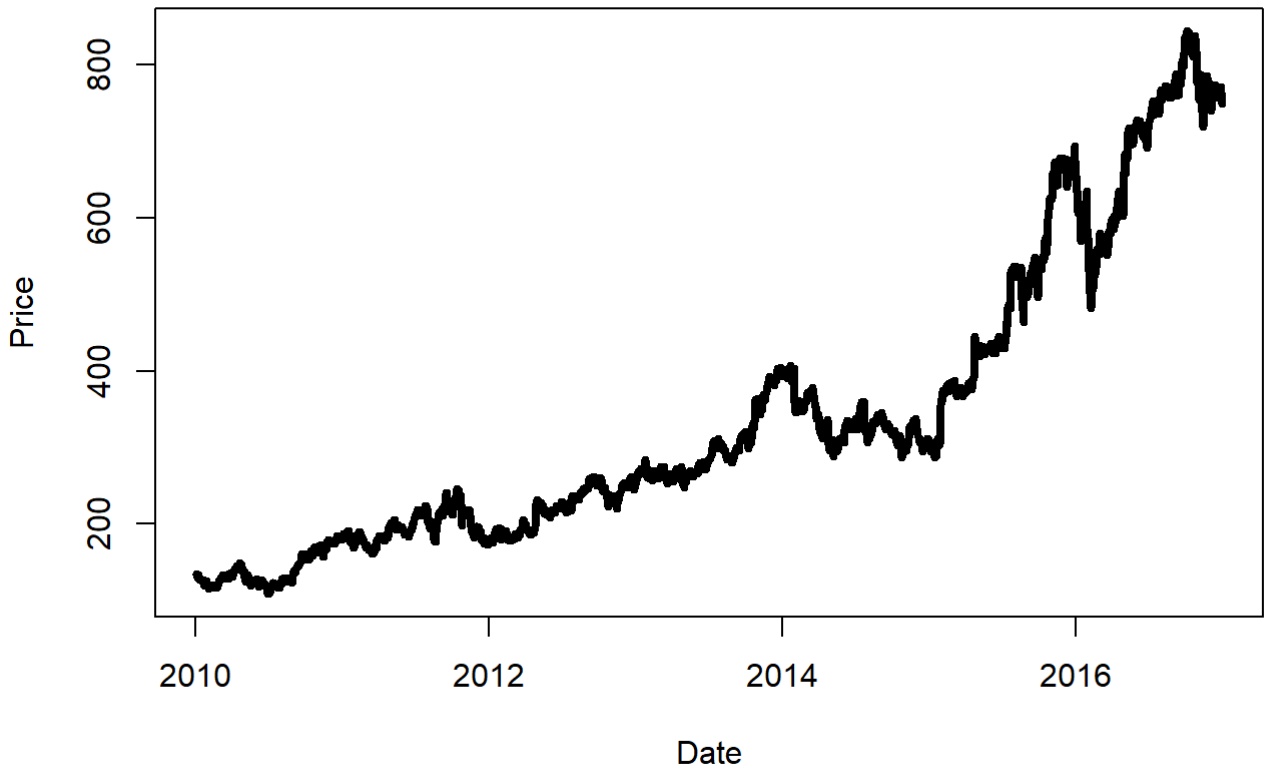
$Y_{t,10}$ dénote le rendement de 10 prochains jours à prédire, et $F_t^{(k)}$ dénote k-ème feature utilisé dans le modèle prédictif linéaire.

Préparation

Nous allons d'abord importer les prix historiques (close) du srock Amazon et calculer les rendements ainsi que les moyennes mobiles comme des features du modèle linéaire.

```
rm(list=ls())
set.seed(123)

#### Importer des donnees ####
library(openxlsx)
Data = read.xlsx("TP2_Amazon.xlsx", sheet = 1, detectDates = TRUE, skipEmptyRows = FALSE)
Price = Data$close
Date  = Data$date
plot(as.Date(Date, "%Y-%m-%d"), Price, lwd = 4, type = "l", xlab = 'Date')
```



Nous fournissons ci-dessous les codes de la fonction myreturn() qui peut calculer “backward returns” ou “forward returns” pour une certaine ***Period***:

- “bakward returns”: Pour une ***TimePosition*** choisie, “backward returns” avec un ***Lag*** sont definis comme:

$$R_{t,\ Lag} := \frac{P_t}{P_{t-Lag}} - 1, \quad TimePosition - Period < t \leq TimePosition$$

- “forward returns”: Pour une ***TimePosition*** choisie, “forward returns” avec un ***Lag*** sont definis comme:

$$Y_{t,\ Lag} := \frac{P_{t+Lag}}{P_t} - 1, \quad TimePosition \leq t < TimePosition + Period$$

```
#### Fonction genrerale pour calculer the backward (Direction = 0) returns ou forward (Direct
ion = 1) returns avec un "Lag" donnée, pour une certain période "Period".  ####
myreturn <- function(Price, TimePosition, Lag, Direction = 0, Period = 252) {
  if (Direction == 0) {
    # backward returns
    Ret = Price[(TimePosition-Period+1):TimePosition] / Price[(TimePosition-Period+1-Lag):(Ti
mePosition-Lag)] - 1
  }
  else {
    # forward returns
    Ret = Price[(TimePosition+Lag):(TimePosition+Period-1+Lag)] / Price[(TimePosition):(TimeP
osition+Period-1)] - 1
  }
  return (Ret)
}
```

Nous fournissons ci-dessous les codes de la fonction myma() qui peut calculer les moyennes mobiles avec une fenêtre de W jours pour une certaine ***Period***:

$$MA\left(R_t, W\right) = \frac{1}{W}\sum_{k=0}^{W-1} R_{t-k}, \quad TimePosition - Period < t \leq TimePosition$$

```
#### Fonction pour calculer compute moving average (MA)  ####
myma <- function(Data, TimePosition, W, Period = 252){
  R = Data[(TimePosition-Period+1):TimePosition]
  for (k in 1:(W-1)) {
    R = R + Data[(TimePosition-Period+1-k):(TimePosition-k)]
  }
  R = R / W
  return (R)
}
```

A l'aide des fonctions myreturn() et myma(), nous allons construire la variable expliquée et les variables explicatives de notre modèle linéaire:

- La variable expliquée $Y_{t,10}$ dénote le rendement de 10 prochains jours à prédire.
- Les variables explicatives sont des moyennes mobiles avec fenêtres differents (1, 2, 4, 8, 16, 32, 64, 128) sur les rendements journaliers.

Nous allons construire le moèle linéaire avec 2 ans des données pour une ***TomePosition*** choisie.

```
PredDays = 10           # On creera un modele de regression lineaire pour expliquer "10-day f
uture returns".
WinLen   = 252 * 2      # On choisit une fenetre de 2 ans pour faire le modèle de regression.
TimePosition = WinLen + 252 # On choisit une date comme "TimePosition" pour faire la regressi
on lineaire.

R = myreturn(Price, TimePosition, Lag = 1, Direction = 0, Period = WinLen + 128 - 1) # On com
pute les rendements journaliers (Backward returns avec lag = 1). / Remarque: on compute 127 e
xtra points pour calculer les moyennes mobiles des rendements dans la partie suiante.

Yt = myreturn(Price, TimePosition-WinLen+1, Lag = 10, Direction = 1, Period = WinLen) # On co
mpute les "10-day future returns". (Forward returns avec lag = 10).


# On compute les moyennes mobiles des rendements avec differents "Window" pour préparer des
"features" dans le modèle de regression.
Ftm1    = R[(length(R)-WinLen+1):length(R)]
Ftm2    = myma(R, length(R), W = 2, Period = WinLen)
Ftm4    = myma(R, length(R), W = 4, Period = WinLen)
Ftm8    = myma(R, length(R), W = 8, Period = WinLen)
Ftm16   = myma(R, length(R), W = 16, Period = WinLen)
Ftm32   = myma(R, length(R), W = 32, Period = WinLen)
Ftm64   = myma(R, length(R), W = 64, Period = WinLen)
Ftm128  = myma(R, length(R), W = 128, Period = WinLen)
M = cbind(Ftm1, Ftm2, Ftm4, Ftm8, Ftm16, Ftm32, Ftm64, Ftm128)
```

Questions

Q1: Correlation analysis

Nous allons calculer les coefficients de corrélation entre les variable expliquée $Y_{t,10}$ et nos vecteurs de features. Que pouvez-vous dire de ces corrélations?

```
Vec_Cor = #...# Hint: cor()
barplot(Vec_Cor)
```

Q2: Preprocessing

Notre modèle de régression suppose un processus homoscédastique. Donc, il est nécessaire de standardiser la variable expliquée et les vecteurs de features dans le processus de régression.

- Pour la variable expliquée $Y_{t,10}$, il suiffit d'avoir une moyenne nulle.
- Pour les vecteurs de features F_{tmK}, la normalisation de chaque facteur d'agit d'avoir une moyenne nulle et une variance unitaire.

```
Ycen = #...#
Mnorm =  #...# Hint: scale()
```

Q3: Régression linéaire

Nous pouvons maintenant construire un modèle linéaire en utilisant tous les features. Que pouvez-vous conclure du résumé du modèle?

```
linModelFull = #...# Hint: lm()
summary(linModelFull)
```

Q4: Méthode Backward pour la sélection des variables

Nous allons appliquer la méthode Backward pour la sélectiondes variables. A chaque étape, nous enlevons la variable dont la valeur t-statistics est la plus petite. Laisser nous appelons ce modèle réduit ***linModelRed***. Que pouvez-vous conclure en comparant les features sélectionnées aux analyse de corrélation? Comment savoir si le modèle réduit préserve la même qualité d'ajustement que le modèle complet? (Hint: Nous pouvons utiliser anova () pour comparer les deux modèles.)

```
linModelRed = #...# Hint: lm()
summary(linModelRed)

# F-test: model comparison
#...# Hint: anova()
```

Q5: Ridge and Lasso

Nous pouvons utiliser le package glmnet pour effectuer des ridge et des lasso régressions. Lisez le document glmnet et complétez les codes suivants. Comparez les deux figures et interprétez-les.

```
library(glmnet)

rdg = #...# Hint: cv.glmnet() pour ridge
lso = #...# Hint: cv.glmnet() pour lasso

plot(rdg$glmnet.fit, "lambda", label = TRUE, lwd = 4)
plot(lso$glmnet.fit, "lambda", label = TRUE, lwd = 4)
```

Nous pouvons extraire les variables optimales pour la regression Lasso en utilisant la validation croisée. Nous refaisons un modèle linéaire standard sur les variables extraites. Comparez la qualité du modèle Lasso au modèle complet en utilisant les statistiques F de l'ANOVA

```
coefLso = coef(lso, s = 'lambda.min') # Lasso parameter from minimizing cross validation erro
r

# refit the selected factors
linModelLasso = #...# Hint: lm()
summary(linModelLasso)

# F-Test to confirm
#...# Hint: anova()
```

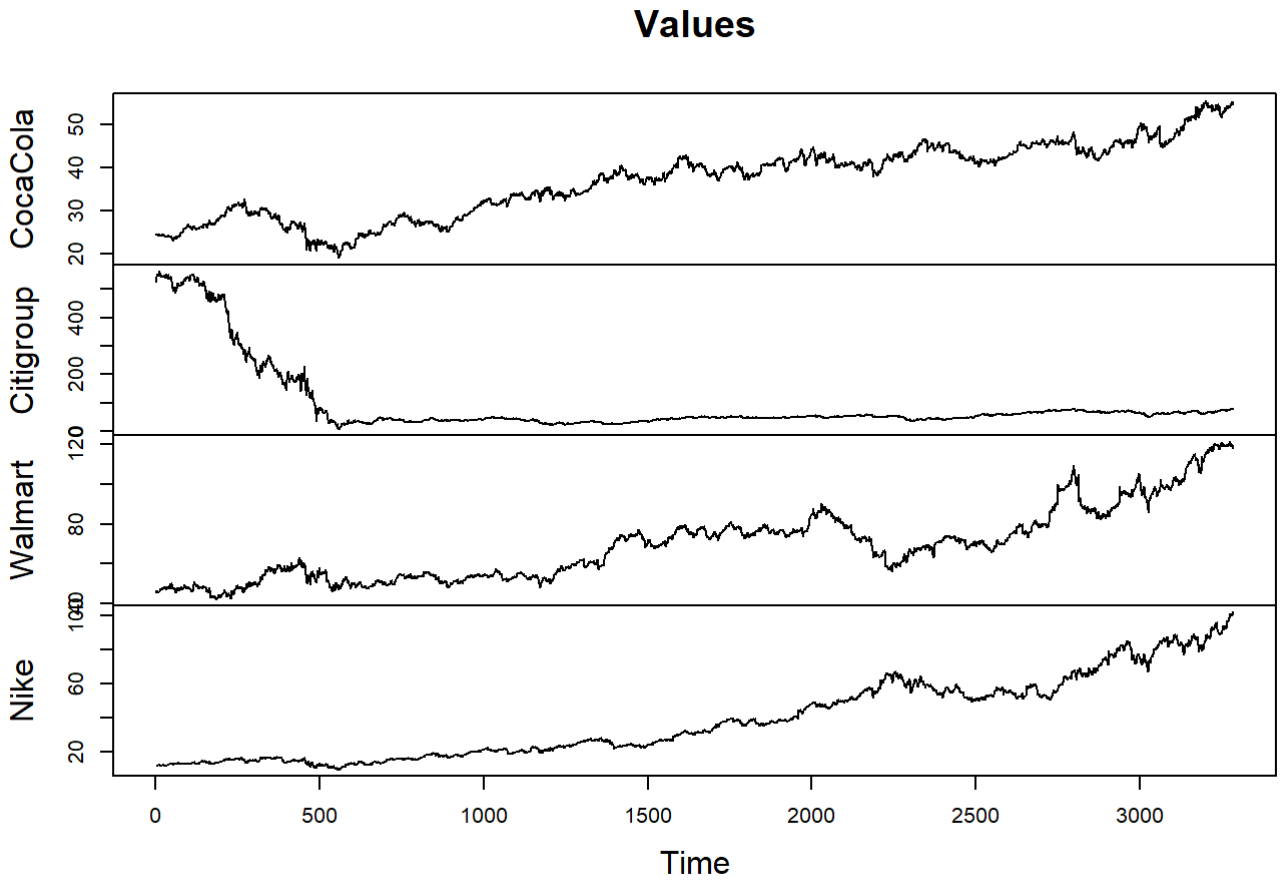
Partie II: Problème d'optimisation de Markowitz avec la régularisation L2

Dans cet exercice, nous allons utiliser les données des 4 stocks (Coca Cola, Citi Group, Walmart, Nike) aux Etats-Unis pour illustrer la technique de régularisation L2 dans l'optimisation du portefeuille (Mean-variance optimisation de Markowitz). Nous disposons d'une base de données de 3286 lignes (nombre d'observations) et 4 colonnes (nombre de sociétés).

```
rm(list=ls())
#### Importer les donnees ####
Stocks = read.csv(file="TP2_MVO.csv", header=TRUE, sep=";")
Date  = Stocks$Index
Values = Stocks[c('CocaCola', 'Citigroup', 'Walmart', 'Nike')]
print(dim(Values))
```

```
[1] 3286    4
```

```
#### Tracer les prix historiques ####
plot.ts(Values, lwd = 1, type = "l")
```



II.1/ Tracer la frontière efficiente via la méthode de simulation

L'objectif de cette partie est de générer 5000 portefeuilles fictifs (les combinaisons aléatoires de 4 stocks) pour visualiser la frontière efficiente dans la théorie de Markowitz. Nous calculons d'abord le rendement annuel composé (CGAR - compound annual growth rate) et la volatilité annuelle pour chaque actif avec les fonctions dans TD1.

```
#### Fonction pour calculer les rendements ####
Compute_Return <- function(Price){
  Price_Clean = Price[!is.na(Price)]
  n = length(Price_Clean)
  ret = (Price_Clean[2:n]/Price_Clean[1:(n-1)]) - 1
  ret = c(0, ret)
  return (ret)
}

#### Fonction pour calculer le CGAR ####
Compute_CAGR <- function(Price, Multiplicator = 252){
  Price_Clean = Price[!is.na(Price)]
  n = length(Price_Clean)
  ret = (Price_Clean[n]/Price_Clean[1])^(Multiplicator/(n-1))-1
  return (ret)
}

#### Fonction pour calculer la volatilite ####
Compute_Vol <- function(Price, Multiplicator = 252){
  Price_Clean = Price[!is.na(Price)]
  n = length(Price_Clean)
  ret = Compute_Return(Price_Clean)
  mu = mean(ret)
  sigma_daily = sqrt(sum((ret - mu)^2)/(n-1))
  sigma = sqrt(Multiplicator)*sigma_daily
  return (sigma)
}
```

Q1: Utiliser la fonction 'apply' et 'Compute_CAGR' pour calculer le CAGR de chaque stock

```
Mus = #...#
```

Q2: Utiliser la fonction 'apply' et 'Compute_Return' pour calculer les rendements de chaque stock et puis calculer la matrice de covariance (annuelle) avec la fonction 'cov'

```
Return = #...#
Sigma = #...#
```

Dans la partie suivante, nous allons simuler les allocations de stocks pour construire les portefeuilles fictifs. Pour le faire, nous pouvons simuler 4 chiffres de 0 à 1 et puis nous les normalisons afin d'avoir la somme 100%. Nous simulons au total 5000 portefeuilles fictifs.

Q3: Calculer le rendement du portefeuille et la volatilité du portefeuille fictif avec l'opérateur '%*%' (produit matriciel)

```
Num_asset = dim(Return)[2] ## Nombre d'actifs
Num_portfolios = 5000 ## Nombre de simulation

List_Ret = c()
List_Vol = c()
List_SR = c()

for(Ind in 1:Num_portfolios){
  ## Les allocations du portefeuille sont générées aléatoirement entre 0 et 1 et puis nous les normalisons afin d'avoir la somme 100%.
  Weights = runif(Num_asset)
  Weights = as.matrix(Weights/sum(Weights))

  ## Nous devons calculer le rendement et la volatilité du portefeuille simulé avec l'opérateur %*% (produit matriciel).
  Ret_portfolio = #...#
  Vol_portfolio = #...#

  SR_portfolio = Ret_portfolio / Vol_portfolio

  List_Ret = c(List_Ret, as.numeric(Ret_portfolio))
  List_Vol = c(List_Vol, as.numeric(Vol_portfolio))
  List_SR = c(List_SR, as.numeric(SR_portfolio))
}

#### Tracer la frontiere efficiente ####
xlim = c(0, 0.2)
ylim = c(-0.05, 0.2)

plot(List_Vol, List_Ret, type = 'p', pch= 16, cex = 0.5, xlab = 'Vol', ylab = 'Ret', xlim = xlim, ylim = ylim)
```

Chaque point tracé signifie un portefeuille fictif. Donc, ce nuage de points est la totalité des combinaisons possibles entre les quatre stocks. Nous pouvons voir clairement l'existence de la frontière efficiente.

II.2/ Tracer la frontière efficiente via la programmation quadratique

L'objectif de cette partie est de refaire l'exercice précédente et de retrouver la frontière efficiente via la programmation quadratique.

Nous allons utiliser la forme γ-problème du problème d'optimisation de Markowitz (CM4) et nous rappelons que γ est le coefficient de trade-off entre le rendement espéré et la volatilité.

γ-problème:

$$w^\star(\gamma) = \arg\min \frac{1}{2} w^T \Sigma w - \gamma w^T \mu$$

$$\text{s.c.} \quad w^T \mathbf{1} = \sum_i w_i = 1$$

$$w_i \geq 0, \forall i$$

γ-problème peut s'écrire sous forme de la programmation quadratique (CM4).

La programme quadratique:

$$w^\star = \arg\min \frac{1}{2} w^T Q w - w^T R$$
$$\text{s.c. } Sw \leq T$$

avec

$$Q = \Sigma \quad R = \gamma \mu$$

$$\begin{matrix} w^T \mathbf{1} = 1 \\ w_i \geq 0, \forall i \end{matrix} \quad \Leftrightarrow \quad \underbrace{\begin{bmatrix} -(1,1,\ldots,1) \\ (1,1,\ldots,1) \\ -I_n \end{bmatrix}}_{S} w \leq \underbrace{\begin{bmatrix} -1 \\ 1 \\ \mathbf{0}_n \end{bmatrix}}_{T} \quad \Leftrightarrow \quad Sw \leq T$$

Nous pouvons utiliser la fonction ‘solve.QP’ dans le package ‘quadprog’ afin de réaliser la programmation quadratique en R. Nous devons lire attentivement le document de la fonction ‘solve.QP’ (A l’aide de la commande ‘?quadprog::solve.QP’) et nous décidons les valeurs des arguments: **Dmat**, **Amat**, **dvec**, **bvec**. Nous pouvons profiter de l’argument **meq** pour avoir directement la contrainte d’égalité.

Nous allons résoudre le problème d’optimisation sous forme de la programmation quadratique pour les différents valeurs de γ.

Q4: Déterminer les arguments pour la programmation quadratique et calculer le rendement du portefeuille et la volatilité du portefeuille avec l’opérateur ‘%*%’

```
library(quadprog)
gammas = seq(from = 0, to = 1, by = 0.01)

List_Ret_QP = c()
List_Vol_QP = c()
List_SR_QP = c()

for(Ind in gammas){
  #### Déterminer les arguments de la programmation quadratique  ####
  Dmat = #...#
  Amat = #...#
  dvec = #...#
  bvec = #...#
  meq = #...#

  qp.out = solve.QP(Dmat=Dmat, dvec=dvec, Amat=Amat, bvec=bvec, meq = meq)
  Weights =qp.out$solution

  #### Calculer le rendement du portefeuille et la volatilite du portefeuille avec l'opérateu
r '%\*%' ####
  Ret_portfolio = #...#
  Vol_portfolio = #...#

  SR_portfolio = Ret_portfolio / Vol_portfolio

  List_Ret_QP = c(List_Ret_QP, as.numeric(Ret_portfolio))
  List_Vol_QP = c(List_Vol_QP, as.numeric(Vol_portfolio))
  List_SR_QP = c(List_SR_QP, as.numeric(SR_portfolio))
}

#### Tracer la frontiere efficiente ####
xlim = c(0, 0.2)
ylim = c(-0.05, 0.2)

plot(List_Vol, List_Ret, type = 'p', pch= 16, cex = 0.5, xlim = xlim, ylim = ylim, xlab = 'Vo
latility', ylab = 'Return')
par(new = TRUE)
plot(List_Vol_QP, List_Ret_QP, type = 'l', col = 'red', pch= 16, lwd = 4, xlim = xlim, ylim =
ylim, xlab = NA, ylab = NA)
```

Nous pouvons voir que la frontière efficiente est bien collée avec le nuage de points.

II.3/ L'allocation du portefeuille au cours du temps

Dans cette partie, nous fixons la valeur de $\gamma = 0.02$ et nous allons rebalancer le portefeuille tous les 22 jours (environ 1 mois). A chaque date de rebalancement, nous utilisons un an de données passées pour estimer le rendement espéré et la volatilité.

Q5: Nous devons utiliser les réponses de Q1 $\sim$ Q4 afin de remplir les trous dans les codes ci-dessous.

```
gamma = 0.02
Num_Observation = dim(Return)[1]
Index_Rebalancement = seq(500, Num_Observation - 22, by = 22)
Weights = matrix(nrow = Num_Observation, ncol = 4)

for(Ind in Index_Rebalancement){
  Values_tmp = Values[(Ind-252):Ind, ]
  Return_tmp = Return[(Ind-252):Ind, ]

  #### Calculer les CAGR et les volatilite avec un an de données ####
  Mus_tmp = #...#
  Sigma_tmp = #...#

  #### Déterminer les arguments de la programmation quadratique  ####
  Dmat = #...#
  dvec = #...#
  Amat = #...#
  bvec = #...#
  meq = #...#

  qp.out = solve.QP(Dmat=Dmat, dvec=dvec, Amat=Amat, bvec=bvec, meq = meq)
  Weights[(Ind+1):(Ind+22), ] = matrix(rep(qp.out$solution, 22), ncol = Num_asset,  byrow = T
RUE)
}

Weights = Weights[rowSums(is.na(Weights)) != ncol(Weights), ]

#### Tracer les allocations du portefeuille au cours du temps ####
library(ggplot2)
Time = as.numeric(rep(seq(1, dim(Weights)[1]), each = dim(Weights)[2]))
Value = round(as.vector(t(Weights)), 4)
Group = rep(c('CocaCola', 'Citigroup', 'Walmart', 'Nike'), times = dim(Weights)[1])
Data_Graph = data.frame(Time, Value, Group)

ggplot(Data_Graph, aes(x=Time, y=Value, fill=Group)) + geom_area()
```

II.4/ La technique de la régulation L2

Dans cette partie, nous allons voir l'application de la technique de la régularisation L2. L'objectif est de résoudre le problème d'instabilité dans l'optimisation mean-variance de Markowitz et réduire les turnovers du portefeuille. Nous rappelons que le problème d'optimisation (γ-problème) avec la régularisation L2 peut s'écrire sous forme:

$$\begin{aligned} w^\star = \arg\min \tfrac{1}{2} w^T \Sigma w - \gamma w^T \mu + \rho \|w - w_{old}\|_2^2 \\ \text{s.c.} \quad w^T \mathbf{1} = 1 \\ w_i \geq 0, \forall i \end{aligned}$$

w_{old} signifie l'allocation de la date de rebalancement précédente.

Nous pouvons aussi écrire la fonction d'objectif du problème ci-dessus sous forme de la programmation quadratique:

$$w^\star = \arg\min \tfrac{1}{2} w^T \Sigma w - \gamma w^T \mu + \rho (w - w_{old})^T (w - w_{old})$$

Nous pouvons le développer:

$$w^\star = \arg\min \frac{1}{2} w^T \Sigma w - \gamma w^T \mu + \rho w^T w - 2\rho w^T w_{old} + \rho w_{old}^T w_{old}$$

Comme $\rho w_{old}^T w_{old}$ est un terme constant, nous pouvons le négliger.

Finalement, la version de la programmation quadratique du problème initial est :

$$\begin{aligned} w^\star = \arg\min \frac{1}{2} w^T (\Sigma + 2\rho I_n) w - w^T (\gamma \mu + 2\rho w_{old}) \\ \text{s.c.} \quad w^T \mathbf{1} = 1 \\ w_i \geq 0, \forall i \end{aligned}$$

Q6: Nous pouvons prendre les codes de Q5 et actualiser les valeurs de Dmat, dvec afin d'ajouter la régularisation L2 dans l'optimisation. Nous pouvons tester différentes valeurs de ρ pour voir l'impact de la régularisation

```
#### Loop MVO via quadprog + Regularisation (Long-only, No leverage)
Num_Observation = dim(Return)[1]
Index_Rebalancement = seq(500, Num_Observation - 22, by = 22)
Weights_Ridge = matrix(nrow = Num_Observation, ncol = 4)
Weights_Old = rep(0.25, 4) #### Les allocations initiales du portefueille

for(Ind in Index_Rebalancement){
  Values_tmp = Values[(Ind-252):Ind, ]
  Return_tmp = Return[(Ind-252):Ind, ]

  #### Calculer les CAGR et les volatilite avec un an de données ####
  Mus_tmp = #...#
  Sigma_tmp = #...#

  rho = 0.05

  #### Déterminer les arguments de la programmation quadratique  ####
  Dmat = #...#
  dvec = #...#
  Amat = #...#
  bvec = #...#
  meq = #...#

  qp.out = solve.QP(Dmat=Dmat, dvec=dvec, Amat=Amat, bvec=bvec, meq = meq)
  Weights_Ridge[(Ind+1):(Ind+22), ] = matrix(rep(qp.out$solution, 22), ncol = Num_asset,  byr
ow = TRUE)
  Weights_Old = qp.out$solution
}

Weights_Ridge = Weights_Ridge[rowSums(is.na(Weights_Ridge)) != ncol(Weights_Ridge), ]

#### Tracer les allocations du portefeuille au cours du temps ####
library(ggplot2)
Time = as.numeric(rep(seq(1, dim(Weights_Ridge)[1]), each = dim(Weights_Ridge)[2]))
Value = round(as.vector(t(Weights_Ridge)), 4)
Group = rep(c('CocaCola', 'Citigroup', 'Walmart', 'Nike'), times = dim(Weights_Ridge)[1])
Data_Graph <- data.frame(Time, Value, Group)

ggplot(Data_Graph, aes(x=Time, y=Value, fill=Group)) + geom_area()
```

Nous devons voir que:

- Quand ρ est très petit, la régularisation L2 n'a pas d'impact sur les résultats d'optimisation.
- Quand ρ est très grand, la partie de la régularisation L2 est dominante dans la fonction d'objectif et nous pouvons avoir un effet de limiter les turnovers.