## Análisis de rendimiento y mejora del tiempo de ejecución de un programa

|  |  |
| --- | --- |
| 50.000 combinaciones de lotería. Objetivo: Análisis de rendimiento del programa. | 1ª mejora. Objetivo: Obtén cada combinación de una sola vez. |
| library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Establecemos una semilla de generación aleatoria determinada para que los rdos sean siempre iguales.**  set.seed(1000)  **# Hacemos una función que saque una bola y compruebe que no ha salido ya en la combinación actual.**  sacar\_bola <- function(combi,nbola){  bola <-sort(sample(seq(1,50,1), 1, replace = TRUE))    **# Comprobamos que este número no ha salido ya en la combinación actual**  for (comprobar\_bola in 1:nbola){  if (bola == combi[comprobar\_bola]){  bola <-sort(sample(seq(1,50,1), 1, replace = TRUE)) # Si la bola ya ha salido sacamos otra.  }  }  return(bola)  }  **# Sacamos la combinación ganadora**  combi\_ganadora<- matrix(0,nrow=1,ncol=5,byrow=T) # Creamos una matriz para la combi ganadora.  for (nbola in 1:5){  combi\_ganadora[nbola]<- sacar\_bola(combi\_ganadora,nbola)  }  **# Sacamos las combinaciones apostadas y comprobamos cuantos aciertos tenemos en cada una de ellas.**  combinaciones <- 50000  apuestas<- matrix(0,nrow=combinaciones,ncol=5,byrow=T) # Creamos una matriz para las apuestas.  aciertos<-matrix(0,nrow=combinaciones,ncol=1,byrow=T) # Creamos un vector para los aciertos.  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = combinaciones, width=300) # width es el nº de pixeles de la barra.  for (combinacion in 1:combinaciones){    **# Obtenemos las apuestas realizadas (sacamos una combinación)**  combi<- matrix(0,nrow=1,ncol=5,byrow=T) # Creamos una matriz donde guardaremos la combi apostada  for (nbola in 1:5){  combi[nbola]<- sacar\_bola(combi,nbola)  }  apuestas[combinacion,]<-combi    **# Comprobamos los aciertos que tenemos entre nuestras apuestas y la combinación ganadora.**  for (bola\_apostada in 1:5){  for (bola\_premiada in 1:5){ # comparamos cada bola\_apostada con cada bola\_premiada  if(apuestas[combinacion,bola\_apostada]==combi\_ganadora[bola\_premiada]){  aciertos[combinacion]<- aciertos[combinacion]+1  }  }  }  setWinProgressBar(barra\_progreso, combinacion, title=paste(round(combinacion/combinaciones\*100,0), "% realizado"))  }  close(barra\_progreso)  **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  library(plyr)  aciertos<-count(aciertos)  aciertos  print(proc.time()-tiempo) # Detiene el cronómetro (elapsed es el tiempo que ha tardado el programa)  }) | library(profvis)  profvis({  tiempo <- proc.time()  **# Establecemos una semilla de generación aleatoria determinada para que los rdos sean siempre iguales.**  set.seed(1000)  **# Sacamos la combinación ganadora. Al usar replace False no pueden salir dos números iguales.**  combi\_ganadora <- sort(sample(seq(1,50,1), 5, replace = FALSE)) # sample(1:50, 5, replace = FALSE) mejor opción  **# Sacamos las combinaciones que apostamos y comprobamos el nº de aciertos en cada una de ellas.**  combinaciones <- 50000  apuestas<- matrix(0,nrow=combinaciones,ncol=5,byrow=T) # Creamos una matriz para las apuestas.  aciertos<-matrix(0,nrow=combinaciones,ncol=1,byrow=T) # Creamos un vector para los aciertos.  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = combinaciones, width=300) # width es el nº de pixeles de la barra.  for (combinacion in 1:combinaciones){    **# Obtenemos las apuestas realizadas**  apuestas[combinacion,]<-sort(sample(seq(1,50,1), 5, replace = FALSE)) # La combinación sale ordenada    **# Comprobamos los aciertos que tenemos entre nuestras apuestas y la combinación ganadora.**  for (bola\_apostada in 1:5){  **# Aunque las combinaciones estén ordenadas, no tiene xq coincidir el nº en la misma columna.**  for (bola\_premiada in 1:5){ # Por lo que comparamos cada bola\_apostada con cada bola\_premiada  if(apuestas[combinacion,bola\_apostada]==combi\_ganadora[bola\_premiada]){  aciertos[combinacion]<- aciertos[combinacion]+1  }  }  }  setWinProgressBar(barra\_progreso, combinacion, title=paste(round(combinacion/combinaciones\*100,0), "% realizado"))  }  close(barra\_progreso)  **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  library(plyr)  aciertos<-count(aciertos)  aciertos  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 24,14 segundos. | Tiempo de ejecución 5,79 segundos (reducimos el 76% del tiempo). |
| Si hacemos un análisis de rendimiento, observamos que el programa está constantemente llamando a sacar\_bola sort Debemos centrar nuestros esfuerzos en esa parte del código si queremos optimizarlo. | El programa es ahora mucho más uniforme. El análisis de rendimiento indica que gran parte del tiempo se la lleva la función count (marcada en rojo). Pero poco podemos hacer ahí. Probablemente podremos mejorar algo de tiempo en la comprobación de los aciertos. |
| 2ª mejora. Objetivo: Vectoriza la comprobación de aciertos. | 3ª mejora. Objetivo: Vectoriza todo, salvo la obtención de combinaciones. |
| library(profvis)  profvis({  tiempo <- proc.time()  **# Establecemos una semilla de generación aleatoria determinada para que los rdos sean siempre iguales.**  set.seed(1000)  **# Sacamos la combinación ganadora y la guardamos en un vector de 50 columnas.**  # Si sale el nº 32, en la columna 32 pondremos un 1.  # Esto evitará recorrer los números de la combinación para comprobar los aciertos.  combi\_ganadora <- matrix(0, nrow=1, ncol=50, byrow=TRUE)  combinacion <- sort(sample(seq(1,50,1), 5, replace = FALSE)) # sample(1:50, 5, replace = FALSE) # Mejor opción  combi\_ganadora[combinacion]<-1  **# Creamos una matriz de 50.000 filas por 50 columnas para guardar las apuestas.**  combinaciones <- 50000  apuestas <- matrix(0, nrow=combinaciones, ncol=50, byrow=TRUE)  # Creamos una función que saque combinaciones y las guarde en la matriz apuestas.  # Usamos Lapply (for optimizado) para llamar a la función que saque combinaciones y las meta en la matriz.  unlist(lapply(1:combinaciones, function(fila) c(apuestas[fila,sort(sample(seq(1,50,1), 5, replace = FALSE))]<<-1)))  # sapply(1:combinaciones, function(fila) c(apuestas[fila, sample(1:50, 5, replace = FALSE)]<<-1)) # Mejor opción  **# Comprobamos los aciertos que hemos tenido. Apuestas y combi\_ganadora son matrices de 1 y 0.**  # Si multiplicamos el vector combi\_ganadora por la matriz de apuestas, en el caso de que coincida  # el nº apostado con el ganador tendremos un 1, y en el resto de casos un 0.  aciertos <- t(apuestas) \* as.vector(combi\_ganadora)  aciertos<-t(aciertos)  **# Sumamos las filas para obtener el número de aciertos de cada apuesta**  num\_aciertos <- rowSums(aciertos)  **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  library(plyr)  aciertos<-count(num\_aciertos)  aciertos  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time()  **# Establecemos una semilla de generación aleatoria determinada para que los rdos sean siempre iguales.**  set.seed(1000)  **# Sacamos la combinación ganadora y la guardamos en un vector de 50 columnas.**  combi\_ganadora <- matrix(0, nrow=1, ncol=50, byrow=TRUE)  combinacion <- sort(sample(seq(1,50,1), 5, replace = FALSE)) # sample(1:50, 5, replace = FALSE) # Mejor opción  combi\_ganadora[combinacion]<-1  **# Para maximizar la eficiencia hay que eliminar el bucle lapply del código anterior.**  # Para ello generamos 50.000 \* 50 nº aleatorios entre 0 y 1  combinaciones <- 50000  apuestas <- runif(combinaciones\*50)  # Convertimos el vector en una matriz  apuestas <- matrix(apuestas, nrow=combinaciones, ncol=50, byrow=TRUE)  **# Para elegir los 5 nº de la combinación de entre los 50, nos quedamos con los 5 késimos mayores.**  # Nos quedamos con los 5 nº aleatorios más grandes (les asignamos un 1 dentro y al resto un 0).  # apuestas[order(apuestas, decreasing = TRUE)[1:5],]<-1 # ESTO solo FUNCIONA para una fila aislada  # Se podrían ordenar las filas de la matriz, pero perderíamos la posición de cada número apuestas <-t(apply(t(apuestas),2,sort)).  for (fila in 1:combinaciones){  vector<-apuestas[fila,]  vector[order(vector, decreasing = TRUE)[1:5]]<-1  vector[vector!=1]<-0  apuestas[fila,]<-vector  }  **# Comprobamos los aciertos que hemos tenido. Apuestas y combi\_ganadora son matrices de 1 y 0.**  aciertos <- t(apuestas) \* as.vector(combi\_ganadora)  aciertos<-t(aciertos)  **# Sumamos las filas para obtener el número de aciertos de cada apuesta**  num\_aciertos <- rowSums(aciertos)  **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  library(plyr)  aciertos<-count(num\_aciertos)  aciertos  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 4,34 segundos (reducimos el 21% del tiempo) | Tiempo de ejecución 1,57 segundos (reducimos el 66% del tiempo, el 93% del inicial). |
| El análisis de rendimiento nos indica que la línea unlist(lapply) requiere 2,07 segundos, el 45% del tiempo del programa. ¿Es una solución ineficiente aunque hayamos mejorado el tiempo? Lapply se presta a paralelizar la solución (**2,67** seg si hacemos el cluster en el programa, **0,25** seg si ya está hecho), aunque no guarda las combinaciones. | El análisis de rendimiento nos indica que las líneas que más tiempo llevan son la generación de nº aleatorios runif (0,04 segundos), convertirlo a una matriz (0,01 seg) y calcular la frecuencia de aciertos con count (0,2 segundos). El resto de líneas conllevan un tiempo uniforme e instantáneo. Es, por lo tanto, un programa óptimo. |

|  |
| --- |
| 4ª mejora. Objetivo: Vectoriza todo el programa. |
| library(profvis)  library(plyr)  profvis({  tiempo <- proc.time()  **# Establecemos una semilla de generación aleatoria determinada para que los rdos sean siempre iguales.**  set.seed(1000)  **# Sacamos la combinación ganadora y la guardamos en un vector de 50 columnas.**  combi\_ganadora <- matrix(0, nrow=1, ncol=50, byrow=TRUE)  combinacion <- sample(1:50, 5, replace = FALSE)  combi\_ganadora[combinacion]<-1  **# Sacamos 50.000 combinaciones.**  apuestas<-t(replicate(50000,rank(runif (50)))) # sacamos 50.000 permutaciones de 1 a 50 en cada línea.  apuestas[apuestas<=45]<-0# los que sean menor o igual a 45 son 0.  apuestas[apuestas>45]<-1# los que sean mayores de 45 los igualamos a 1.  **# Comprobamos los aciertos que hemos tenido**. Apuestas y combi\_ganadora son matrices de 1 y 0.  aciertos <- t(apuestas) \* as.vector(combi\_ganadora)  aciertos<-t(aciertos)  **# Sumamos las filas para obtener el número de aciertos de cada apuesta**  num\_aciertos <- rowSums(aciertos)  **# Calculamos la frecuencia de los aciertos** (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)  aciertos<-count(num\_aciertos)  aciertos  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 1,5 segundos. |

|  |  |
| --- | --- |
| Otra manera de hacerlo | Otra manera de hacerlo |
| library(profvis)  library(plyr)  profvis({    tiempo <- proc.time()    set.seed(1000)  **# Sacamos la combinación ganadora y la guardamos en un vector de 50 columnas.**  combi\_ganadora <- matrix(0, nrow=1, ncol=50, byrow=TRUE)  combinacion <- sample(seq(1,50,1), 5, replace = FALSE)  combi\_ganadora[combinacion]<-1    **# Defino una función que convierte un vector de enteros en un vector de 50 posiciones**  combinacion\_to\_vec <- function(comb){  result=c(1:50)\*0  result[comb] = 1  return(result)  }    **# Genero la secuencia de posibles números para no volver a generarla en cada iteración**  bolas = seq(1,50,1)    **# Sacamos 50.000 combinaciones de 5 numeros enteros sin reemplazo y los convierto a vector de 50 posiciones**  apuestas<-t(replicate(50000,combinacion\_to\_vec(sample(bolas,5,replace=FALSE))))    **# Comprobamos los aciertos que hemos tenido. Apuestas y combi\_ganadora son matrices de 1 y 0.**  aciertos <- t(apuestas) \* as.vector(combi\_ganadora)  aciertos<-t(aciertos)    **# Sumamos las filas para obtener el número de aciertos de cada apuesta**  num\_aciertos <- rowSums(aciertos)    **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  aciertos<-count(num\_aciertos)  aciertos    print(proc.time()-tiempo)  }) | library(profvis)  library(plyr)  profvis({    tiempo <- proc.time()  set.seed(1000)    **# Sacamos la combinación ganadora y la guardamos en un vector de 50 columnas.**  combi\_ganadora <- matrix(0, nrow=1, ncol=50, byrow=TRUE)  combinacion <- sample(seq(1,50,1), 5, replace = FALSE)  combi\_ganadora[combinacion]<-1  combinacion\_base <- c(rep(0, times = 45),rep(1, times = 5))    **# Sacamos 50.000 combinaciones de 45 ceros y de 5 unos en cada línea.**  apuestas<-t(replicate(50000,sample(combinacion\_base, 50)))    **# Comprobamos los aciertos que hemos tenido. Apuestas y combi\_ganadora son matrices de 1 y 0.**  aciertos <- t(apuestas) \* as.vector(combi\_ganadora)  aciertos<-t(aciertos)    **# Sumamos las filas para obtener el número de aciertos de cada apuesta**  num\_aciertos <- rowSums(aciertos)    **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  aciertos<-count(num\_aciertos)  aciertos    print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 0,66 segundos. | Tiempo de ejecución 0,45 segundos. |
| Otra manera de hacerlo | La manera más eficiente de hacerlo |
| library(profvis)  library(plyr)  profvis({    tiempo <- proc.time()    set.seed(1000)  **# Generamos la secuencia una sola vez**  secuencia <- seq(1,50,1)    **# Sacamos la combinación ganadora.**  combi\_ganadora <- sample(secuencia, 5, replace = FALSE)    **# Sacamos las combinaciones que apostamos y comprobamos el nº de aciertos en cada una de ellas.**  combinaciones <- 50000  apuestas<-matrix(0,nrow=combinaciones,ncol=5,byrow=T) # Creamos una matriz para las apuestas.  aciertos<-matrix(0,nrow=combinaciones,ncol=1,byrow=T) # Creamos un vector para los aciertos.    for (combinacion in 1:combinaciones){  **# Obtenemos las apuestas realizadas**  apuestas[combinacion,]<-sample(secuencia, 5, replace = FALSE) # La combinación sale ordenada  **# Comprobamos los aciertos que tenemos entre nuestras apuestas y la combinación ganadora.**  aciertos[combinacion,] <- sum(apuestas[combinacion,] %in% combi\_ganadora)  }    **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  aciertos<-count(aciertos)  aciertos    print(proc.time()-tiempo)  }) | library(profvis)  library(plyr)  profvis({    tiempo <- proc.time()    set.seed(1000)  **# Sacamos la combinación ganadora. Al usar replace False no pueden salir dos números iguales.**  combi\_ganadora <- sample.int(50, 5, replace = FALSE)    **# Sacamos las combinaciones que apostamos.**  combinaciones <- 50000  apuestas <- matrix(replicate(combinaciones, sample.int(50, 5, replace = FALSE)), ncol=5, byrow = T)    **# Comprobamos los aciertos que tenemos entre nuestras apuestas y la combinación ganadora.**  aciertos <- matrix((apuestas %in% combi\_ganadora), ncol = 5, byrow = T)    **# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**  aciertos <- rowSums(aciertos)  aciertos <- count(aciertos)  aciertos    print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 0,29 segundos. | Tiempo de ejecución 0,17 segundos. |

**Ahora probemos con Python**

|  |  |
| --- | --- |
| Python usando solo módulos del lenguaje | Podemos mejorar la función sacar bola con recursividad |
| import random  import time  start\_time = time.time()  combinaciones = 50000  n\_bolas = 5  def sacar\_bola\_slow(combi, n\_bola):  "Esta forma produce bolas repetidas y hay que buscarlas"  bola = random.randint(0, 50)  for bola\_comp in range(n\_bola):  if bola == combi[bola\_comp]:  bola = random.randint(0, 50)  return bola  combi\_ganadora = []  for i in range(n\_bolas):  combi\_ganadora.append(sacar\_bola\_slow(combi\_ganadora, i))  apuestas = []  aciertos = []  for combinancion in range(combinaciones):  combi\_apostada = []  for n\_bola in range(n\_bolas):  combi\_apostada.append(sacar\_bola\_slow(combi\_apostada, n\_bola))  apuestas.append(combi\_apostada)    aciertos\_combinacion = 0  for bola\_ganadora in combi\_ganadora:  for bola\_apostada in combi\_apostada:  if bola\_ganadora == bola\_apostada:  aciertos\_combinacion += 1  aciertos.append(aciertos\_combinacion)  for num\_aciertos in range(n\_bolas):  print(f"{num\_aciertos} : {aciertos.count(num\_aciertos)}")  print("--- %s seconds ---" % (time.time() - start\_time)) | import random  import time  start\_time = time.time()  combinaciones = 50000  n\_bolas = 5  def sacar\_bola(combi):  "Eliminamos el bucle de comprobación con recursividad"  bola = random.randint(0, 50)  if bola in combi:  bola = sacar\_bola(combi)  return bola  combi\_ganadora = []  for i in range(n\_bolas):  combi\_ganadora.append(sacar\_bola(combi\_ganadora))  apuestas = []  aciertos = []  for combinancion in range(combinaciones):  combi\_apostada = []  for n\_bola in range(n\_bolas):  combi\_apostada.append(sacar\_bola(combi\_apostada))  apuestas.append(combi\_apostada)    aciertos\_combinacion = 0  for bola\_ganadora in combi\_ganadora:  for bola\_apostada in combi\_apostada:  if bola\_ganadora == bola\_apostada:  aciertos\_combinacion += 1  aciertos.append(aciertos\_combinacion)  for num\_aciertos in range(n\_bolas):  print(f"{num\_aciertos} : {aciertos.count(num\_aciertos)}")  print("--- %s seconds ---" % (time.time() - start\_time)) |
| Tiempo de ejecución 0,43 segundos. | Tiempo de ejecución 0,37 segundos. |
| Podemos sustituir los dos bucles de comprobación de aciertos por sets | Introducimos numpy |
| import random  import time  start\_time = time.time()  combinaciones = 50000  n\_bolas = 5  def sacar\_bola(combi):  "Eliminamos el bucle de comprobación con recursividad"  bola = random.randint(0, 50)  if bola in combi:  bola = sacar\_bola(combi)  return bola  combi\_ganadora = []  for i in range(5):  combi\_ganadora.append(sacar\_bola(combi\_ganadora))  apuestas = []  aciertos = []  for combinancion in range(combinaciones):  combi\_apostada = []  for n\_bola in range(5):  combi\_apostada.append(sacar\_bola(combi\_apostada))  apuestas.append(combi\_apostada)    # Calculamos la intersección entre combi\_apostada y combi\_ganadora  aciertos\_combinacion = len(set(combi\_apostada).intersection(combi\_ganadora))  aciertos.append(aciertos\_combinacion)  for num\_aciertos in range(5):  print(f"{num\_aciertos} : {aciertos.count(num\_aciertos)}")  print("--- %s seconds ---" % (time.time() - start\_time)) | import numpy as np  import random  import time  start\_time = time.time()  combinaciones = 50000  combinacion = np.random.choice(50, 5, replace=False)  combi\_ganadora = np.zeros(50)  combi\_ganadora[combinacion] = 1  aciertos = np.zeros(combinaciones)  for i in range(combinaciones):  combinacion = np.random.choice(50, 5, replace=False)  combi\_apostada = np.zeros(50)  combi\_apostada[combinacion] = 1    # Igual que en R (multiplicamos los vectores combi\_ganadora con combi\_apostada  aciertos[i] = combi\_ganadora@combi\_apostada  unique, counts = np.unique(aciertos, return\_counts=True)  print(f"{unique} {counts}")  print("--- %s seconds ---" % (time.time() - start\_time)) |
| Tiempo de ejecución 0,30 segundos. | Tiempo de ejecución 2,41 segundos. |

|  |
| --- |
| Eliminamos el último bucle definiendo las apuestas previamente |
| import numpy as np  import random  import time  start\_time = time.time()  combinaciones = 50000  combinacion = np.random.choice(50, 5, replace=False)  combi\_ganadora = np.zeros(50)  combi\_ganadora[combinacion] = 1  apuestas = np.zeros((combinaciones, 50))  apuestas[:, :5] = 1  [np.random.shuffle(apuesta) for apuesta in apuestas] # Hacemos que los 1 se mezclen a lo largo del vector  num\_aciertos = apuestas @ combi\_ganadora  unique, counts = np.unique(num\_aciertos, return\_counts=True)  print(f"{unique} {counts}")  print("--- %s seconds ---" % (time.time() - start\_time)) |
| Tiempo de ejecución 1,06 segundos. |
| 4ª mejora. Objetivo: Vectoriza todo el programa. |
| Maximizamos la eficiencia |
| import numpy as np  import random  import time  start\_time = time.time()  combinaciones = 50000  combinacion = np.random.choice(50, 5, replace=False)  combi\_ganadora = np.zeros(50)  combi\_ganadora[combinacion] = 1  apuestas = np.zeros((combinaciones, 50))  apuestas\_num = np.random.rand(combinaciones, 50).argpartition(5,axis=1)[:,:5] # Sacamos 5 nº aleatorios por fila  index = np.tile(np.expand\_dims(np.arange(combinaciones), axis=0).transpose(), (1, 5)) # Genero una matriz de coordenadas de filas  apuestas[index, apuestas\_num] = 1 # Asigno un 1 a los nº que han salido. Necesito pasar 2 matrices (matriz filas, matriz columnas)  num\_aciertos = apuestas @ combi\_ganadora  unique, counts = np.unique(num\_aciertos, return\_counts=True)  print(f"{unique} {counts}")  print("--- %s seconds ---" % (time.time() - start\_time)) |
| Tiempo de ejecución 0,064 segundos. |

# Cálculo del Alfa de Jensen

Importa los datos del fichero DAX.csv y calcula el Alpha de Jensen para todos los activos del índice.

Verás que en el mismo archivo encontrarás también los datos del índice (DAX) y de la renta fija (Bund) alemán.

α=Rentabilidad\_cartera-(Rentabildad\_activo\_libre\_riesgo+ β (Rentabilidad\_mercado- Rentabildad\_activo\_libre\_riesgo))

β= covarianza (Rentabilidad\_cartera, Rentabilidad\_mercado) / varianza (Rentabilidad\_mercado)

El alfa de Jensen es una medida de calidad sobre la gestión realizada (ya sea en un fondo, cartera de activos, o una única empresa). Indica el exceso de rentabilidad obtenida para un nivel de riesgo determinado.

El Alfa explica la diferencia entre la rentabilidad esperada, es decir, la que corresponde al riesgo sistemático asumido, y la realmente obtenida por el gestor. En función de que el gestor supere, iguale o esté por debajo del rendimiento esperado tendrá un Alfa positivo, neutro o negativo.

**Cómo vectorizar progresivamente una instrucción en R**

rent\_activos[2,1]<-log(activos[2,1]/activos[1,1]) # 1 activo un día

rent\_activos[2,1:length(activos)] <- log(activos[2,1:length(activos)]/activos[2-1,1:length(activos)]) # Todos los act un solo día (día 2)

rent\_activos[2:5,1:length(activos)] <- log(activos[2:5,1:length(activos)]/activos[(2:5)-1,1:length(activos)]) # Todos los act días 2 a 5

# Todos los activos todos los días

rent\_activos[2:dim(activos)[1],1:length(activos)] <-log(activos[2:dim(activos)[1],1:length(activos)]/activos[(2:dim(activos)[1])-1,1:length(activos)])

**Cómo vectorizar progresivamente una instrucción en Python (con index)**

rent\_activos[0, 0] = np.log(matrix\_price[1,0]/matrix\_price[0,0]) # 1 activo un día

rent\_activos[1, :] = np.log(matrix\_price[1,:] / matrix\_price[0,:]) # Todos los act un solo día (día 2)

rent\_activos[1:6, :] = np.log(matrix\_price[1:6,:] /matrix\_price[0:5,:]) # Todos los act días 2 a 5

rent\_activos[1:, :] = np.log(matrix\_price[1:,:] / matrix\_price[0:-1,:]) # Todos los activos todos los días

**Cómo vectorizar progresivamente una instrucción en Python (con pandas)**

import pandas as pd

retornos = np.log(serie).diff() # Cuando es solo un activo

retornos = np.log(df).diff() # Cuando es un DF con muchos activos

|  |  |
| --- | --- |
| Objetivo: Calcula el Alpha de Jensen. | Objetivo: Creamos una función que calcule el Alpha de un activo en un día. |
| library(profvis)  profvis({  tiempo <- proc.time()  **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  DAX <- datos[,length(datos)-1] # Extraemos el índice.  rango\_fechas <- as.Date(datos[,1])  **# Calculamos la rentabilidad de los 10 primeros días para usarlos como datos iniciales.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(rent\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(rent\_activos) <- datos[,1] # Ponemos nombres a las filas.  rent\_BUND <- rep(0,length(BUND)) # Genero un vector donde guardaré las rentabilidades de la renta fija.  rent\_DAX <- rep(0,length(DAX)) # Genero un vector donde guardaré las rentabilidades del índice.  alpha\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(alpha\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(alpha\_activos) <- datos[,1] # Ponemos nombres a las filas.  for (dia in 2:10){ # Calculamos la rentabilidad de los 10 primeros días (activos, Bund y Dax)  for (activo in 1:length(activos)){  rent\_activos[dia,activo] <- log(activos[dia,activo]/activos[dia-1,activo])  }  rent\_BUND[dia] <- log(BUND[dia]/BUND[dia-1])  rent\_DAX[dia] <- log(DAX[dia]/DAX[dia-1])  }  **# Teniendo ya datos históricos, calculamos los Alphas para cada día con un bucle.**  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = dim(activos)[1], width=300)  for (dia in 11:dim(activos)[1]){  **# Calculamos la rentabilidad de los activos, la renta fija y el índice para una fecha concreta.**  for (activo in 1:length(activos)){  rent\_activos[dia,activo] <- log(activos[dia,activo]/activos[dia-1,activo])  }  rent\_BUND[dia] <- log(BUND[dia]/BUND[dia-1])  rent\_DAX[dia] <- log(DAX[dia]/DAX[dia-1])    **# Calculamos la varianza del DAX**  varianza\_ind<-var(rent\_DAX[2:dia]) # Excluimos la 1ª línea porque contiene ceros.  for (activo in 1:length(activos)){  **# Calculamos la covarianza entre el activo y el índice**  cov\_act\_ind <- cov(rent\_activos[2:dia,activo],rent\_DAX[2:dia])  **# Sacamos la Beta del activo. β=cov(Rc,Rm)/σRm**  if (varianza\_ind==0){  beta <-0  }else{  beta <- cov\_act\_ind/varianza\_ind  }  **# Calculamos el Alpha del activo. α=Rc-(Rf+β(Rm-Rf))**  alpha\_activos[dia,activo]<- rent\_activos[dia,activo]-(rent\_BUND[dia]+beta\*(rent\_DAX[dia]-rent\_BUND[dia]))  }  setWinProgressBar(barra\_progreso, dia, title=paste(round(dia/dim(activos)[1]\*100,0), "% realizado"))  }  close(barra\_progreso)  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time()  **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  DAX <- datos[,length(datos)-1] # Extraemos el índice.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  rent\_DAX <- rep(0,length(DAX))  alpha\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(alpha\_activos)<-names(activos)  rownames(alpha\_activos) <- datos[,1]  **# Calculamos la rentabilidad de los activos, renta fija e índice de manera vectorial.**  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_BUND[2:dim(activos)[1]] <- log(BUND[2:dim(activos)[1]]/BUND[2:dim(activos)[1]-1])  rent\_DAX[2:dim(activos)[1]] <- log(DAX[2:dim(activos)[1]]/DAX[2:dim(activos)[1]-1])  **# Generamos una función que calcule el Alpha para un solo activo.**  calcular\_alpha<-function(activo,rent\_activos,rent\_DAX,rent\_BUND,dia){  **# Calculamos la varianza del DAX**  varianza\_ind<-var(rent\_DAX[2:dia])    **# Calculamos la covarianza entre el activo y el índice**  cov\_act\_ind <- cov(rent\_activos[2:dia,activo],rent\_DAX[2:dia])    **# Sacamos la Beta del activo. β=cov(Rc,Rm)/σRm**  if (varianza\_ind==0){  beta <-0  }else{  beta <- cov\_act\_ind/varianza\_ind  }    **# Calculamos el Alpha del activo. α=Rc-(Rf+β(Rm-Rf))**  alpha<- rent\_activos[dia,activo]-(rent\_BUND[dia]+beta\*(rent\_DAX[dia]-rent\_BUND[dia]))  return(alpha)  }  **# Invocamos a la función para que calcule el alpha de todos los activos, todos los días.**  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = dim(activos)[1], width=300)  for (dia in 11:dim(activos)[1]){  for (activo in 1:length(activos)){  alpha\_activos[dia,activo]<-calcular\_alpha(activo,rent\_activos,rent\_DAX,rent\_BUND,dia)  }  setWinProgressBar(barra\_progreso, dia, title=paste(round(dia/dim(activos)[1]\*100,0), "% realizado"))  }  close(barra\_progreso)  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 19,6 segundos | Tiempo de ejecución 13,81 segundos. |
| Objetivo: Cambia la función para que calcule al alpha de todos los activos de un día. | Objetivo: Quita la función y vectoriza todo (menos varianza y covarianza). |
| library(profvis)  profvis({  tiempo <- proc.time()  **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  DAX <- datos[,length(datos)-1] # Extraemos el índice.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  rent\_DAX <- rep(0,length(DAX))  alpha\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(alpha\_activos)<-names(activos)  rownames(alpha\_activos) <- datos[,1]  **# Calculamos la rentabilidad de los activos, renta fija e índice.**  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_BUND[2:dim(activos)[1]] <- log(BUND[2:dim(activos)[1]]/BUND[2:dim(activos)[1]-1])  rent\_DAX[2:dim(activos)[1]] <- log(DAX[2:dim(activos)[1]]/DAX[2:dim(activos)[1]-1])  **# Generamos una función que calcule los Alpha de todos los activos en un mismo día.**  calcular\_alpha<-function(dia,rent\_activos,rent\_DAX,rent\_BUND){  **# Calculamos la varianza del DAX**  varianza\_ind<-var(rent\_DAX[2:dia])    **# Calculamos la covarianza entre el activo y el índice**  cov\_act\_ind<-as.data.frame(matrix(0,nrow=1,ncol=length(rent\_activos),byrow=F))  cov\_act\_ind[1:length(cov\_act\_ind)] <- cov(rent\_activos[2:dia,1:length(activos)],rent\_DAX[2:dia])    **# Sacamos la Beta del activo. β=cov(Rc,Rm)/σRm**  beta<-as.data.frame(matrix(0,nrow=1,ncol=length(rent\_activos),byrow=F))  if (varianza\_ind==0){  beta <-0  }else{  beta[1:length(beta)] <- cov\_act\_ind[1:length(cov\_act\_ind)]/varianza\_ind  }    **# Calculamos el Alpha del activo. α=Rc-(Rf+β(Rm-Rf))**  alpha<-as.data.frame(matrix(0,nrow=1,ncol=length(rent\_activos),byrow=F))  alpha[1:length(activos)]<- rent\_activos[dia,1:length(rent\_activos)]-(rent\_BUND[dia]+beta[1:length(beta)]\*(rent\_DAX[dia]-rent\_BUND[dia]))    return(alpha)  }  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = dim(activos)[1], width=300)  for (dia in 11:dim(activos)[1]){  alpha\_activos[dia,]<-calcular\_alpha(dia,rent\_activos,rent\_DAX,rent\_BUND)  setWinProgressBar(barra\_progreso, dia, title=paste(round(dia/dim(activos)[1]\*100,0), "% realizado"))  }  close(barra\_progreso)  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time()  **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  DAX <- datos[,length(datos)-1] # Extraemos el índice.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  rent\_DAX <- rep(0,length(DAX))  varianza\_DAX <-rep(0,length(DAX))  cov\_act\_ind<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  beta<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  alpha\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(alpha\_activos)<-names(activos)  rownames(alpha\_activos) <- datos[,1]  **# Calculamos la rentabilidad de los activos, renta fija e índice.**  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_BUND[2:dim(activos)[1]] <- log(BUND[2:dim(activos)[1]]/BUND[2:dim(activos)[1]-1])  rent\_DAX[2:dim(activos)[1]] <- log(DAX[2:dim(activos)[1]]/DAX[2:dim(activos)[1]-1])  **# Calculamos la varianza del DAX**  #varianza\_DAX[2:length(rent\_DAX)]<-var(rent\_DAX[2:length(rent\_DAX)]) # No funciona, usa el vector entero.  for (dia in 11:dim(activos)[1]){  varianza\_DAX[dia]<-var(rent\_DAX[2:dia]) # Excluimos la 1ª línea porque contiene ceros.  } # Este for únicamente lleva 0,06 segundos (no es ineficiente).  **# Calculamos la covarianza entre el activo y el índice**  #cov\_act\_ind[2:dim(activos)[1],1:length(activos)] <- cov(rent\_activos[2:dim(activos)[1],1:length(activos)],rent\_DAX[2:dim(activos)[1]])  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = dim(activos)[1], width=300)  for (dia in 11:dim(activos)[1]){  cov\_act\_ind[1:length(cov\_act\_ind)] <- cov(rent\_activos[2:dia,1:length(activos)],rent\_DAX[2:dia])  setWinProgressBar(barra\_progreso, dia, title=paste(round(dia/dim(activos)[1]\*100,0), "% realizado"))  }  close(barra\_progreso)  **# Sacamos la Beta del activo. β=cov(Rc,Rm)/σRm**  beta[2:dim(activos)[1],1:length(activos)] <- cov\_act\_ind[2:dim(activos)[1],1:length(activos)]/varianza\_DAX[2:length(rent\_DAX)]  **# Calculamos el Alpha del activo. α=Rc-(Rf+β(Rm-Rf))**  alpha\_activos[2:dim(activos)[1],1:length(activos)]<- rent\_activos[2:dim(activos)[1],1:length(activos)]-(rent\_BUND[2:dim(activos)[1]]+beta[2:dim(activos)[1],1:length(activos)]\*(rent\_DAX[2:dim(activos)[1]]-rent\_BUND[2:dim(activos)[1]]))  alpha\_activos<-alpha\_activos[11:dim(activos)[1],1:length(activos)] # Eliminamos los primeros valores (ceros).  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 26,43 segundos (empeora la eficiencia del programa). | Tiempo de ejecución 18,52 segundos, el bucle de la covarianza es donde está el problema. |

|  |
| --- |
| Objetivo: Vectoriza todo el programa, menos la covarianza. |
| library(profvis)  library(devtools)  library(RollingWindow) # install\_github("andrewuhl/RollingWindow") # <https://github.com/andrewuhl/RollingWindow>  # También puedes usar library(roll) en lugar de RollingWindow  profvis({  tiempo <- proc.time()  **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  activos<- datos[,2:n\_activos] # Extraemos los activos quitando la última columnasy la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  DAX <- datos[,length(datos)-1] # Extraemos el índice.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  rent\_DAX <- rep(0,length(DAX))  varianza\_DAX <-rep(0,length(DAX))  cov\_act\_ind<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  beta<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  alpha\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(alpha\_activos)<-names(activos)  rownames(alpha\_activos) <- datos[,1]  **# Calculamos la rentabilidad de los activos, renta fija e índice.**  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_BUND[2:dim(activos)[1]] <- log(BUND[2:dim(activos)[1]]/BUND[2:dim(activos)[1]-1])  rent\_DAX[2:dim(activos)[1]] <- log(DAX[2:dim(activos)[1]]/DAX[2:dim(activos)[1]-1])  **# Calculamos la varianza del DAX**  varianza\_DAX<-RollingVar(rent\_DAX,window = 10)  **# Calculamos la covarianza entre el activo y el índice**  for (activo in 1:dim(activos)[2]){ # RollingCov solo acepta vectores, por lo que tenemos que recorrer los activos con un for,  cov\_act\_ind[,activo] <- RollingCov(rent\_DAX, rent\_activos[,activo], window = 10)  }  **# Sacamos la Beta del activo. β=cov(Rc,Rm)/σRm**  beta[11:dim(activos)[1],1:length(activos)] <- cov\_act\_ind[11:dim(activos)[1],1:length(activos)]/varianza\_DAX[11:length(rent\_DAX)]  **# Calculamos el Alpha del activo. α=Rc-(Rf+β(Rm-Rf))**  alpha\_activos[11:dim(activos)[1],1:length(activos)]<- rent\_activos[11:dim(activos)[1],1:length(activos)]-(rent\_BUND[11:dim(activos)[1]]+beta[11:dim(activos)[1],1:length(activos)]\*(rent\_DAX[11:dim(activos)[1]]-rent\_BUND[11:dim(activos)[1]]))  alpha\_activos<-alpha\_activos[11:dim(activos)[1],1:length(activos)] # Eliminamos los primeros valores (ceros).  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 0,09 segundos. |

La vectorización tiene una limitación, no permite vectorizar operaciones o funciones que trabajen con ventanas de datos. Lo cual obliga a buscar funciones que sí estén preparadas y optimizadas para trabajar así como RollingVar, pero encontramos de nuevo limitaciones a la hora de trabajar con estas funciones. RollingCov trabaja únicamente con vectores, por lo que si queremos calcular la cov de varios activos tenemos que usar obligatoriamente un bucle for.

Sin embargo, roll sí puede trabajar con matrices, por lo que es una alternativa lógica.

|  |
| --- |
| Objetivo: Vectoriza todo el programa |
| library(profvis)  profvis({  tiempo <- proc.time()  **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  activos<- datos[,2:n\_activos] # Extraemos los activos quitando la última columnasy la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  DAX <- datos[,length(datos)-1] # Extraemos el índice.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  rent\_DAX <- rep(0,length(DAX))  varianza\_DAX <-rep(0,length(DAX))  cov\_act\_ind<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  beta<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  alpha\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(alpha\_activos)<-names(activos)  rownames(alpha\_activos) <- datos[,1]  **# Calculamos la rentabilidad de los activos, renta fija e índice.**  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_BUND[2:dim(activos)[1]] <- log(BUND[2:dim(activos)[1]]/BUND[2:dim(activos)[1]-1])  rent\_DAX[2:dim(activos)[1]] <- log(DAX[2:dim(activos)[1]]/DAX[2:dim(activos)[1]-1])  **# Calculamos la varianza del DAX**  CumSumDAX=cumsum(rent\_DAX)  CumAvgDAX=(CumSumDAX[window\_width:length(rent\_DAX)]-CumSumDAX[1:(length(CumSumDAX)-window\_width+1)])/window\_width  SquaredDAX=rent\_DAX\*\*2  CumSumSqaredDAX=cumsum(SquaredDAX)  CumAvgSquaredDAX=(CumSumSqaredDAX[window\_width:length(CumSumSqaredDAX)]-CumSumSqaredDAX[1:(length(CumSumSqaredDAX)-window\_width+1)])/window\_width  CumVarDAX=CumAvgSquaredDAX-CumAvgDAX\*\*2    **# Calculamos la covarianza entre el activo y el índice**  CumSumAssets=cumsum(rent\_activos)  CumAvgAssets=(CumSumAssets[window\_width:dim(CumSumAssets)[1],1:dim(CumSumAssets)[2]]-CumSumAssets[1:(dim(CumSumAssets)[1]-window\_width+1),1:dim(CumSumAssets)[2]])/window\_width  CumSumProduct=cumsum(rent\_activos\*rent\_DAX)  CumSumProductAVG=(CumSumProduct[window\_width:dim(CumSumProduct)[1],1:dim(CumSumProduct)[2]]-CumSumProduct[1:(dim(CumSumProduct)[1]-window\_width+1),1:dim(CumSumProduct)[2]])/window\_width  CumCovar=CumSumProductAVG-CumAvgAssets\*CumAvgDAX    **# Sacamos la Beta del activo. β=cov(Rc,Rm)/σRm**  beta=CumCovar/sqrt(CumVarDAX)    **# Calculamos el Alpha del activo. α=Rc-(Rf+β(Rm-Rf))**  alpha<- rent\_activos[window\_width:dim(rent\_activos)[1],1:dim(rent\_activos)[2]]-(rent\_BUND[window\_width:length(rent\_BUND)]+beta\*(rent\_DAX[window\_width:length(rent\_DAX)]-rent\_BUND[window\_width:length(rent\_BUND)]))  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 0,16 segundos. |

No siempre, conseguir vectorizar todo el programa, supone un ahorro de tiempo.

# Cálculo de la Frontera de Markowitz

Importa los datos del fichero indices.csv y calcula la frontera de Markowitz.

Haz lo mismo con el fichero de 30 activos del DAX y analiza el escalado de rendimiento.

|  |  |
| --- | --- |
|  |  |

La frontera de carteras eficiente es el conjunto de carteras más eficientes de un mercado, es decir, las que ofrecen una mayor rentabilidad esperada según los diferentes niveles de riesgo que se pueden asumir (o el menor riesgo para una rentabilidad esperada).

Se representa gráficamente como una curva, en dónde cualquier cartera que no se encuentre sobre la línea de la frontera no será eficiente, y por lo tanto estará corriendo riesgos innecesarios o recibiendo una rentabilidad inferior a la que podría obtener, con respecto al riesgo que está asumiendo.

La frontera de carteras eficiente representa la relación óptima que encontramos en una cartera de inversión entre volatilidad y rentabilidad, es decir, entre los beneficios que el inversor podrá obtener y los riesgos que deberá afrontar para hacerlo.

|  |  |  |
| --- | --- | --- |
| Objetivo: Programa la frontera de Markowitz con 5 activos y 50.000 simulaciones. | Objetivo: Vectoriza el cálculo de la rentabilidad diaria. | Objetivo: Vectoriza todo el programa. Reduce la duración por debajo del segundo. |
| library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos**  datos <- read.csv("indices.csv", sep=';',stringsAsFactors = FALSE)  activos<- datos[,2:length(datos)] # Extraemos los activos quitando la primera.  **# Calculamos la rentabilidad de los activos.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(rent\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(rent\_activos) <- datos[,1] # Ponemos nombres a las filas.  for (dia in 2:dim(activos)[1]){  for (activo in 1:length(activos)){  rent\_activos[dia,activo]<-log(activos[dia,activo]/activos[dia-1,activo])  }  }  rent\_activos<-rent\_activos[2:dim(activos)[1],1:length(activos)] # Quitamos la primera línea dado que son ceros.  **# Calculamos la matriz de correlaciones**  # matriz\_correlaciones<-cor(rent\_activos)  **# Calculamos la matriz de varianzas / covarianzas.**  matriz\_var\_covarianzas<-cov(rent\_activos) # Obtenemos diréctamente la matriz de varianzas covarianzas  **# calculamos la rentabilidad, riesgo y eficiencia de N carteras aleatorias.**  num\_simulaciones <- 50000  matriz\_pesos <- as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(matriz\_pesos)<-names(activos)  rentabilidad\_carteras <- c(1:num\_simulaciones)  riesgo\_carteras <- c(1:num\_simulaciones)  eficiencia\_carteras <- c(1:num\_simulaciones)  progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = num\_simulaciones, width=300)  for (cartera in 1:num\_simulaciones){  **# Sacamos los pesos de la cartera**  pesos <- runif(dim(activos)[2], min = 0, max = 100)  pesos[1:dim(activos)[2]]<-(pesos[1:dim(activos)[2]]/sum(pesos)) # Los pesos deben sumar 100%.  matriz\_pesos[cartera,1:dim(activos)[2]]<- pesos    **# Calculamos la rentabilidad diaria del periodo para cada activo: LN(precio final/ precio inicial)/nº de datos**  rent\_diaria <-as.data.frame(matrix(0,nrow=1,ncol=length(activos),byrow=F))  names(rent\_diaria)<-names(activos) # Ponemos nombres a las columnas    for (activo in 1:length(activos)){ # Una manera más lenta de hacerlo.  rent\_diaria[activo]<-log(activos[dim(activos)[1],activo]/activos[1,activo])/dim(rent\_activos)[1]  }  **# Calculamos la rentabilidad de la cartera según los pesos (suma producto pesos con rentabilidad diaria)**  rentabilidad\_carteras[cartera]<-sum(pesos\*rent\_diaria)    **# calculamos el riesgo de la cartera (desviación), en función de los pesos.**  vector<-matriz\_var\_covarianzas%\*%pesos # Multiplicamos la matriz de cov/var por el vector fila de pesos.  riesgo\_carteras[cartera]<-(t(vector)%\*%pesos)^0.5 # Riesgo = vector anterior por vector pesos elevado a 0,5    **# calculamos la eficiencia de la cartera (pendiente), en función de los pesos.**  # (y2 - y1)/(x2-x1) donde Y1 e Y1 son el origen (0,0)  eficiencia\_carteras[cartera] <- (rentabilidad\_carteras[cartera])/(riesgo\_carteras[cartera])  setWinProgressBar(progreso, cartera, title=paste(round(cartera/num\_simulaciones\*100,0), "%"))  }  close(progreso)  **# Graficamos la frontera de Markowitz (riesgo y rentabilidad para cada vector de pesos)**  plot(riesgo\_carteras,rentabilidad\_carteras, cex = .5, pch=19)  **# Localizamos la cartera con mayor rentabilidad, menor riesgo y mayor eficiencia.**  matriz\_pesos$rentabilidad\_cartera <-rentabilidad\_carteras # Unimos todos los datos en un único DF.  matriz\_pesos$riesgo\_cartera <-riesgo\_carteras  matriz\_pesos$eficiencia\_cartera <-eficiencia\_carteras  max\_rentabilidad<-matriz\_pesos[matriz\_pesos$rentabilidad\_cartera==max(matriz\_pesos$rentabilidad\_cartera),]  min\_riesgo<-matriz\_pesos[matriz\_pesos$riesgo\_cartera==min(matriz\_pesos$riesgo\_cartera),]  max\_eficiencia<-matriz\_pesos[matriz\_pesos$eficiencia\_cartera==max(matriz\_pesos$eficiencia\_cartera),]  max\_rentabilidad  min\_riesgo  max\_eficiencia  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos**  datos <- read.csv("indices.csv", sep=';',stringsAsFactors = FALSE)  activos<- datos[,2:length(datos)] # Extraemos los activos quitando la primera.  **# Calculamos la rentabilidad de los activos.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(rent\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(rent\_activos) <- datos[,1] # Ponemos nombres a las filas.  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_activos<-rent\_activos[2:dim(activos)[1],1:length(activos)] # Quitamos la primera línea dado que son ceros.  **# Calculamos la matriz de correlaciones**  # matriz\_correlaciones<-cor(rent\_activos)  **# Calculamos la matriz de varianzas / covarianzas.**  matriz\_var\_covarianzas<-cov(rent\_activos) # Obtenemos diréctamente la matriz de varianzas covarianzas  **# calculamos la rentabilidad, riesgo y eficiencia de N carteras aleatorias.**  num\_simulaciones <- 50000  matriz\_pesos <- as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(matriz\_pesos)<-names(activos)  rentabilidad\_carteras <- c(1:num\_simulaciones)  riesgo\_carteras <- c(1:num\_simulaciones)  eficiencia\_carteras <- c(1:num\_simulaciones)  progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = num\_simulaciones, width=300)  for (cartera in 1:num\_simulaciones){  **# Sacamos los pesos de la cartera**  pesos <- runif(dim(activos)[2], min = 0, max = 100)  pesos[1:dim(activos)[2]]<-(pesos[1:dim(activos)[2]]/sum(pesos)) # Los pesos deben sumar 100%.  matriz\_pesos[cartera,1:dim(activos)[2]]<- pesos    **# Calculamos la rentabilidad diaria del periodo para cada activo: LN(precio final/ precio inicial)/nº de datos**  rent\_diaria <-as.data.frame(matrix(0,nrow=1,ncol=length(activos),byrow=F))  names(rent\_diaria)<-names(activos) # Ponemos nombres a las columnas    rent\_diaria<-t(rent\_diaria)  activos<-t(activos) # Calculamos la rentabilidad vectorialmente.  rent\_diaria[1:dim(activos)[1]]<-log(activos[1:dim(activos)[1],dim(activos)[2]]/activos[1:dim(activos)[1],1])/dim(rent\_activos)[1]  rent\_diaria<-t(rent\_diaria)  activos<-t(activos) # Devolvemos la matriz de activos a su posición original    **# Calculamos la rentabilidad de la cartera según los pesos (suma producto pesos con rentabilidad diaria)**  rentabilidad\_carteras[cartera]<-sum(pesos\*rent\_diaria)    **# calculamos el riesgo de la cartera (desviación), en función de los pesos.**  vector<-matriz\_var\_covarianzas%\*%pesos # Multiplicamos la matriz de cov/var por el vector fila de pesos.  riesgo\_carteras[cartera]<-(t(vector)%\*%pesos)^0.5 # Riesgo = vector anterior por vector pesos elevado a 0,5    **# calculamos la eficiencia de la cartera (pendiente), en función de los pesos.**  # (y2 - y1)/(x2-x1) donde Y1 e Y1 son el origen (0,0)  eficiencia\_carteras[cartera] <- (rentabilidad\_carteras[cartera])/(riesgo\_carteras[cartera])  setWinProgressBar(progreso, cartera, title=paste(round(cartera/num\_simulaciones\*100,0), "%"))  }  close(progreso)  **# Graficamos la frontera de Markowitz (riesgo y rentabilidad para cada vector de pesos)**  plot(riesgo\_carteras,rentabilidad\_carteras, cex = .5, pch=19)  **# Localizamos la cartera con mayor rentabilidad, menor riesgo y mayor eficiencia.**  matriz\_pesos$rentabilidad\_cartera <-rentabilidad\_carteras # Unimos todos los datos en un único DF.  matriz\_pesos$riesgo\_cartera <-riesgo\_carteras  matriz\_pesos$eficiencia\_cartera <-eficiencia\_carteras  max\_rentabilidad<-matriz\_pesos[matriz\_pesos$rentabilidad\_cartera==max(matriz\_pesos$rentabilidad\_cartera),]  min\_riesgo<-matriz\_pesos[matriz\_pesos$riesgo\_cartera==min(matriz\_pesos$riesgo\_cartera),]  max\_eficiencia<-matriz\_pesos[matriz\_pesos$eficiencia\_cartera==max(matriz\_pesos$eficiencia\_cartera),]  max\_rentabilidad  min\_riesgo  max\_eficiencia  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos**  datos <- read.csv("indices.csv", sep=';',stringsAsFactors = FALSE)  activos<- datos[,2:length(datos)] # Extraemos los activos quitando la primera.  **# Calculamos la rentabilidad de los activos.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(rent\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(rent\_activos) <- datos[,1] # Ponemos nombres a las filas.  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_activos<-rent\_activos[2:dim(activos)[1],1:length(activos)] # Quitamos la primera línea dado que son ceros.  **# Calculamos la matriz de correlaciones**  # matriz\_correlaciones<-cor(rent\_activos)  **# Calculamos la matriz de varianzas / covarianzas.**  matriz\_var\_covarianzas<-cov(rent\_activos) # Obtenemos diréctamente la matriz de varianzas covarianzas  **# Calculamos la rentabilidad diaria del periodo para cada activo: LN(precio final/ precio inicial)/nº de datos**  rent\_diaria <-as.data.frame(matrix(0,nrow=1,ncol=length(activos),byrow=F))  names(rent\_diaria)<-names(activos) # Ponemos nombres a las columnas  rent\_diaria<-t(rent\_diaria)  activos<-t(activos) # rentabilidad vectorial: traspongo precios para que los activos sean filas y no columnas.  rent\_diaria[1:dim(activos)[1]]<-log(activos[1:dim(activos)[1],dim(activos)[2]]/activos[1:dim(activos)[1],1])/dim(rent\_activos)[1]  rent\_diaria<-t(rent\_diaria)  activos<-t(activos)  **# Sacamos la matriz de pesos para cada cartera**  num\_simulaciones <- 50000  pesos <- runif(dim(activos)[2]\*num\_simulaciones, min = 0, max = 100) # pesos de todas simulaciones de una vez.  pesos <- matrix(pesos, nrow=num\_simulaciones, ncol=dim(activos)[2], byrow=TRUE)  pesos[,1:dim(activos)[2]]<-(pesos[,1:dim(activos)[2]]/rowSums(pesos)) # Escalamos los pesos a 1  **# Calculamos la rentabilidad de la cartera en función de los pesos**  # Para cada conjunto de pesos queremos hacer suma producto de pesos con rentabilidad diaria.  rentabilidad\_carteras <- c(1:num\_simulaciones) # vector para guardar las rentabilidades de cada cartera.  matriz\_intermedia <- matrix(0,nrow=dim(activos)[1],ncol=dim(activos)[2],byrow=F)  matriz\_intermedia<-t(as.vector(rent\_diaria)\*t(pesos)) # Sin as.vector da error al multiplicar dos matrices.  rentabilidad\_carteras<-rowSums(matriz\_intermedia) # Sumamos las filas para sacar rentabilidad de cada cartera.  **# calculamos el riesgo de la cartera (desviación), en función de los pesos.**  riesgo\_carteras <- c(1:num\_simulaciones) # Generamos un vector donde guardaremos el riesgo de cada cartera.  # Multiplicamos la matriz de var/covar por cada fila de la matriz de pesos.  matriz\_intermedia <- pesos[1:num\_simulaciones,]%\*%matriz\_var\_covarianzas    # Multiplicamos cada fila de la matriz intermedia \* cada fila de la matriz pesos y el rdo lo elevamos a 0.5.  matriz\_intermedia <- matriz\_intermedia \* pesos  riesgo\_carteras <- rowSums(matriz\_intermedia)  riesgo\_carteras <- riesgo\_carteras^0.5  **# calculamos la eficiencia de la cartera (pendiente), en función de los pesos.**  eficiencia\_carteras <- c(1:num\_simulaciones)  eficiencia\_carteras[1:num\_simulaciones] <- rentabilidad\_carteras[1:num\_simulaciones]/riesgo\_carteras[1:num\_simulaciones]  **# Graficamos la frontera de Markowitz (riesgo y rentabilidad para cada vector de pesos)**  plot(riesgo\_carteras,rentabilidad\_carteras, cex = .5, pch=19)  **# Localizamos la cartera con mayor rentabilidad, menor riesgo y mayor eficiencia.**  pesos<-as.data.frame(pesos)  names(pesos)<-names(rent\_activos) # Ponemos nombres a las columnas  pesos$rentabilidad\_cartera <-rentabilidad\_carteras # Unimos todos los datos en un único DF.  pesos$riesgo\_cartera <-riesgo\_carteras  pesos$eficiencia\_cartera <-eficiencia\_carteras  max\_rentabilidad<-pesos[pesos$rentabilidad\_cartera==max(pesos$rentabilidad\_cartera),]  min\_riesgo<-pesos[pesos$riesgo\_cartera==min(pesos$riesgo\_cartera),]  max\_eficiencia<-pesos[pesos$eficiencia\_cartera==max(pesos$eficiencia\_cartera),]  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 5 activos: 11 minutos. Con 30 activos: 46 minutos.  Obtener la cartera eficiente de otro conjunto de datos es tan sencillo como cambiar el CSV de origen. datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE) | Tiempo de ejecución 5 activos: 16 minutos. Con 30 activos 10 horas con 20 minutos. | Tiempo de ejecución 5 activos: 0,44 segundos, 30 activos: 1,67 segundos. |

|  |  |
| --- | --- |
| Frontera de Markowitz con 5 activos | Frontera de Markowitz con 30 activos |

**¿Sabrías decir porqué se grafica mal la frontera con 30 activos?**

El procedimiento de generación de números aleatorios vectorizado asigna un porcentaje a cada uno de los activos. Posteriormente se transforma a una base 100 por lo que, en la mayoría de simulaciones, todos los activos tienen un porcentaje asignado. De ahí que exista un núcleo central.

**¿Se te ocurre alguna solución?**

|  |  |
| --- | --- |
| Todos los activos 1 año | 10 activos, 1 año |
| 5 activos aleatorios, 1 año | 5 activos, preseleccionados, 1 año |

|  |
| --- |
| Objetivo: Solventa el problema de la generación de la frontera para 30 activos. |
| library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  datos<-datos[,-length(datos)]  datos<-datos[dim(datos)[1]:(dim(datos)[1]-260),] # Extraemos los datos de un único año.  datos<-datos[dim(datos)[1]:1,] # El orden de los datos está invertido, tenemos que arreglarlo antes de hacer ningún cálculo  activos<- datos[,2:length(datos)] # Extraemos los activos quitando la primera columna.  **# Calculamos la rentabilidad de los activos.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(rent\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(rent\_activos) <- datos[,1] # Ponemos nombres a las filas.  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_activos<-rent\_activos[2:dim(activos)[1],1:length(activos)] # Quitamos la primera línea dado que son ceros.  **# Necesitamos un criterio de preselección de activos. Por ejemplo: rentabilidad / riesgo (varianza)**  rentabilidad\_ponderada\_riesgo<-log(activos[1,1:length(activos)]/activos[dim(activos)[1],1:length(activos)])/ diag(var(rent\_activos))    # Nos quedamos con los N activos que tengan mejor relación rentabilidad riesgo  n\_empresas<-5  seleccion<-order(rentabilidad\_ponderada\_riesgo, decreasing = F)[1:n\_empresas]  activos<-activos[,seleccion]  rent\_activos<-rent\_activos[,seleccion]  **# Calculamos la matriz de correlaciones**  # matriz\_correlaciones<-cor(rent\_activos)  **# Calculamos la matriz de varianzas / covarianzas.**  matriz\_var\_covarianzas<-cov(rent\_activos) # Obtenemos diréctamente la matriz de varianzas covarianzas  **# Calculamos la rentabilidad diaria del periodo para cada activo: LN(precio final/ precio inicial)/nº de datos**  rent\_diaria <-as.data.frame(matrix(0,nrow=1,ncol=length(activos),byrow=F))  names(rent\_diaria)<-names(activos) # Ponemos nombres a las columnas  rent\_diaria<-t(rent\_diaria)  activos<-t(activos) # rentabilidad vectorial: traspongo precios para que los activos sean filas y no columnas.  rent\_diaria[1:dim(activos)[1]]<-log(activos[1:dim(activos)[1],dim(activos)[2]]/activos[1:dim(activos)[1],1])/dim(rent\_activos)[1]  rent\_diaria<-t(rent\_diaria)  activos<-t(activos)  **# Sacamos la matriz de pesos para cada cartera**  num\_simulaciones <- 50000  pesos <- runif(dim(activos)[2]\*num\_simulaciones, min = 0, max = 100) # pesos de todas simulaciones de una vez.  pesos <- matrix(pesos, nrow=num\_simulaciones, ncol=dim(activos)[2], byrow=TRUE)  pesos[,1:dim(activos)[2]]<-(pesos[,1:dim(activos)[2]]/rowSums(pesos))  **# Calculamos la rentabilidad de la cartera en función de los pesos**  # Para cada conjunto de pesos queremos hacer suma producto de pesos con rentabilidad diaria.  rentabilidad\_carteras <- c(1:num\_simulaciones) # vector para guardar las rentabilidades de cada cartera.  matriz\_intermedia <- matrix(0,nrow=dim(activos)[1],ncol=dim(activos)[2],byrow=F)  matriz\_intermedia<-t(as.vector(rent\_diaria)\*t(pesos)) # Sin as.vector da error al multiplicar dos matrices.  rentabilidad\_carteras<-rowSums(matriz\_intermedia) # Sumamos las filas para sacar rentabilidad de cada cartera.  **# calculamos el riesgo de la cartera (desviación), en función de los pesos.**  riesgo\_carteras <- c(1:num\_simulaciones) # Generamos un vector donde guardaremos el riesgo de cada cartera.  # Ahora multiplicamos la matriz de var/covar por cada fila de la matriz de pesos.  matriz\_intermedia <- pesos[1:num\_simulaciones,]%\*%matriz\_var\_covarianzas  # Ahora multiplicamos cada fila de la matriz intermedia \* cada fila de la matriz pesos y el rdo lo elevamos a 0.5.  # Ojo, la multiplicación es matricial de dos vectores.  matriz\_intermedia <- matriz\_intermedia \* pesos  riesgo\_carteras <- rowSums(matriz\_intermedia)  riesgo\_carteras <- riesgo\_carteras^0.5  **# calculamos la eficiencia de la cartera (pendiente), en función de los pesos.**  eficiencia\_carteras <- c(1:num\_simulaciones)  eficiencia\_carteras[1:num\_simulaciones] <- rentabilidad\_carteras[1:num\_simulaciones]/riesgo\_carteras[1:num\_simulaciones]  **# Graficamos la frontera de Markowitz (riesgo y rentabilidad para cada vector de pesos)**  plot(riesgo\_carteras,rentabilidad\_carteras, cex = .5, pch=19)  **# Localizamos la cartera con mayor rentabilidad, menor riesgo y mayor eficiencia.**  pesos<-as.data.frame(pesos)  names(pesos)<-names(rent\_activos) # Ponemos nombres a las columnas  pesos$rentabilidad\_cartera <-rentabilidad\_carteras # Unimos todos los datos en un único DF.  pesos$riesgo\_cartera <-riesgo\_carteras  pesos$eficiencia\_cartera <-eficiencia\_carteras  max\_rentabilidad<-pesos[pesos$rentabilidad\_cartera==max(pesos$rentabilidad\_cartera),]  min\_riesgo<-pesos[pesos$riesgo\_cartera==min(pesos$riesgo\_cartera),]  max\_eficiencia<-pesos[pesos$eficiencia\_cartera==max(pesos$eficiencia\_cartera),]  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 5 activos: 0,44 segundos, 30 activos: 1,67 segundos. |

|  |  |
| --- | --- |
| Decreasing = T con 10 activos | Decreasing = F con 10 activos |
| Decreasing = T con 5 activos | Decreasing = F con 5 activos |
| Decreasing = T con 3 activos | Decreasing = F con 3 activos |

Abrir documento generación de carteras óptimas y proponerles el reto.

# Ratio de Sharpe

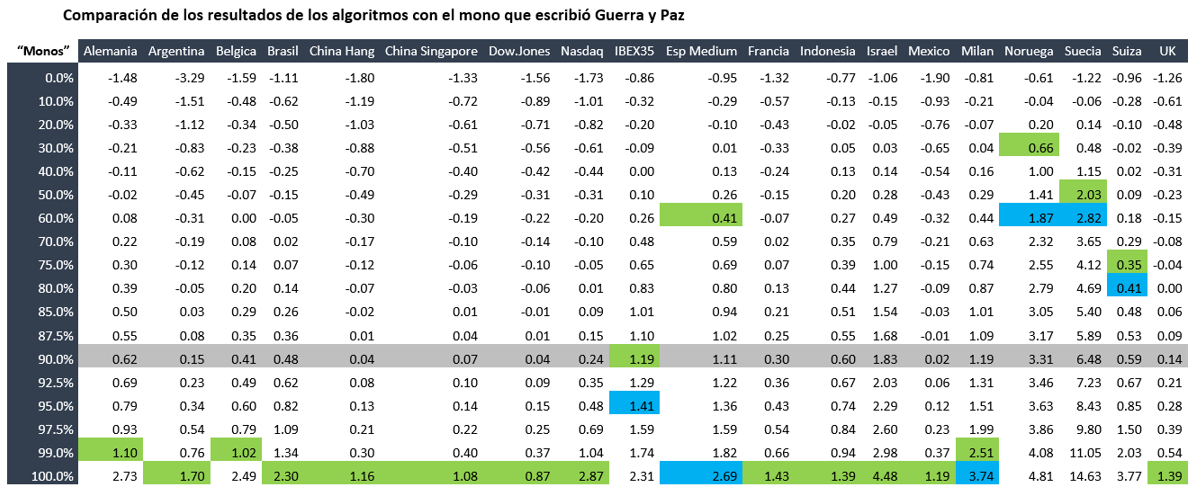
El ratio de Sharpe proporciona el exceso de rentabilidad sobre el activo sin riesgo por unidad de riesgo asumida (riesgo diversificable + no diversificable).

Cuanto mayor sea el ratio de Sharpe, mejor habrá sido la gestión del fondo en el pasado.

|  |  |  |
| --- | --- | --- |
| Objetivo: Calcula el ratio de Sharpe | Objetivo: Vectoriza todo menos el cálculo de la volatilidad | Igual que el anterior, pero un poco más eficiente |
| library(profvis)  profvis({    tiempo <- proc.time()    **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  rango\_fechas <- as.Date(datos[,1])    **# Calculamos la rentabilidad de los 10 primeros días para usarlos como datos iniciales.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(rent\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(rent\_activos) <- datos[,1] # Ponemos nombres a las filas.    rent\_BUND <- rep(0,length(BUND)) # Genero un vector donde guardaré las rentabilidades de la renta fija.  sharpe\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(sharpe\_activos)<-names(activos) # Ponemos nombres a las columnas  rownames(sharpe\_activos) <- datos[,1] # Ponemos nombres a las filas.    for (dia in 2:10){ # Calculamos la rentabilidad de los 10 primeros días (activos, Bund y Dax)  for (activo in 1:length(activos)){  rent\_activos[dia,activo] <- log(activos[dia,activo]/activos[dia-1,activo])  }  rent\_BUND[dia] <- log(BUND[dia]/BUND[dia-1])  }    **# Teniendo ya datos históricos, calculamos el ratio de Sharpe para cada día con un bucle.**  barra\_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = dim(activos)[1], width=300)  for (dia in 11:dim(activos)[1]){    **# Calculamos la rentabilidad de los activos y la renta fija para una fecha concreta.**  for (activo in 1:length(activos)){  rent\_activos[dia,activo] <- log(activos[dia,activo]/activos[dia-1,activo])  }  rent\_BUND[dia] <- log(BUND[dia]/BUND[dia-1])    for (activo in 1:length(activos)){    **# Calculamos la volatilidad del activo (desviación típica).**  volatilidad<-sqrt(var(rent\_activos[2:dia,activo])) # Excluimos la 1ª línea porque contiene ceros.    **# Calculamos el ratio de Sharpe**  if (!is.nan((rent\_activos[dia,activo]-rent\_BUND[dia]) / volatilidad)){  sharpe\_activos[dia,activo]<- (rent\_activos[dia,activo]-rent\_BUND[dia]) / volatilidad  }  }  setWinProgressBar(barra\_progreso, dia, title=paste(round(dia/dim(activos)[1]\*100,0), "% realizado"))  }  close(barra\_progreso)    print(proc.time()-tiempo)  }) | library(profvis)  library(RollingWindow)  options(warn=-1)  profvis({  tiempo <- proc.time()    **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  volatilidad<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  sharpe\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(sharpe\_activos)<-names(activos)  rownames(sharpe\_activos) <- datos[,1]    **# Calculamos la rentabilidad de los activos y renta fija.**  rent\_activos[2:dim(activos)[1],1:length(activos)] <- log(activos[2:dim(activos)[1],1:length(activos)]/activos[2:dim(activos)[1]-1,1:length(activos)])  rent\_BUND[2:dim(activos)[1]] <- log(BUND[2:dim(activos)[1]]/BUND[2:dim(activos)[1]-1])  **# Calculamos la volatilidad**  for (activo in 1:dim(activos)[2]){ # RollingCov solo acepta vectores,  volatilidad[,activo] <- sqrt(RollingVar(rent\_activos[,activo], window = 10))  }  volatilidad[is.na(as.data.frame(volatilidad))]<-0    **# Calculamos el ratio de Sharpe**  sharpe\_activos<- (rent\_activos-rent\_BUND) / volatilidad  sharpe\_activos[is.na(sharpe\_activos)]<-0  sharpe\_activos[is.infinite(as.matrix(sharpe\_activos))]<-0    print(proc.time()-tiempo)  }) | library(profvis)  library(roll)  profvis({  tiempo <- proc.time()    **# Cargamos los datos y extraemos por un lado los activos, la renta fija y el índice Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  activos<- datos[,2:n\_activos] # Extraemos los activos quitando las dos últimas columnas y la primera.  BUND <- datos[,length(datos)] # Extraemos la renta fija.  **# Generamos las matrices de datos donde guardaremos los resultados.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  volatilidad<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  rent\_BUND <- rep(0,length(BUND))  sharpe\_activos<-as.data.frame(matrix(0,nrow=dim(activos)[1],ncol=length(activos),byrow=F))  names(sharpe\_activos)<-names(activos)  rownames(sharpe\_activos) <- datos[,1]    **# Calculamos la rentabilidad de los activos y renta fija.**  rent\_activos[2:dim(activos)[1],1:length(activos)] = log(activos[2:dim(activos)[1],1:length(activos)]/activos[(2:dim(activos)[1])-1,1:length(activos)])  rent\_BUND[2:length(BUND)] = log(BUND[2:length(BUND)]/BUND[(2:length(BUND))-1])    **# Calculamos la volatilidad**  fun\_vola = function(activo){  resultado = roll\_sd(x=activo, width = 30)  return(resultado)  }  volatilidad = sapply(rent\_activos, FUN = fun\_vola)    **# Calculamos el ratio de Sharpe**  sharpe\_activos = (rent\_activos - rent\_BUND) / volatilidad  names(sharpe\_activos) = names(activos)  rownames(sharpe\_activos) = datos[,1]    print(proc.time()-tiempo)  }) |
| Tiempo de ejecución 14,69 segundos | Tiempo de ejecución 0,13 segundos | Tiempo de ejecución 0,06 segundos |

# Prueba de aleatoriedad – confirmación del benchmark

Programa 50.000 inversiones aleatorias para obtener un benchmark artificial con el que comparar los resultados del algoritmo (cuando el benchmark de referencia haya obtenido rentabilidades negativas en el año).



|  |  |
| --- | --- |
| Objetivo: Programa un Benchmark sintético (50.000 simulaciones). | Objetivo: Vectoriza todo el programa y trata de bajar del minuto de cálculo. |
| library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos del Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  DAX<- datos[,2:n\_activos] # Extraemos los activos quitando las 2 últimas columnas y la primera.  div\_EUR <- rep(1,dim(DAX)[1]) # Fabricamos el tipo de cambio.  **# Cargamos los datos del FTSE**  datos <- read.csv("FTSE.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  FTSE<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_GBP <- datos[,length(datos)] # Extraemos el tipo de cambio.  FTSE<-FTSE\*div\_GBP # Pasamos todos los precios a euros.  **# Cargamos los datos del NASDAQ**  datos <- read.csv("NASDAQ.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  NASDAQ<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_USD <- datos[,length(datos)] # Extraemos el tipo de cambio.  NASDAQ<-NASDAQ\*div\_USD # Pasamos todos los precios a euros.  **# Cargamos los datos del HANG SENG**  datos <- read.csv("HANG SENG.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  HANGSENG<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_HKD <- datos[,length(datos)] # Extraemos el tipo de cambio.  HANGSENG<-HANGSENG/div\_HKD # Pasamos todos los precios a euros.  **# Cargamos los datos de BOVESPA**  datos <- read.csv("BOVESPA.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  BOVESPA<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_BRL <- datos[,length(datos)] # Extraemos el tipo de cambio.  BOVESPA<-BOVESPA\*div\_BRL # Pasamos todos los precios a euros.  **# Unimos todos los activos en un único data.frame**  benchmark\_global<-data.frame(DAX,FTSE,NASDAQ,HANGSENG,BOVESPA)  num\_simulaciones<-50000  **# Sacamos las fechas de compra y venta de cada una de las simulaciones.**  fecha\_compra<-matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE)  fecha\_venta<-matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE)  for (simulacion in 1:num\_simulaciones){  fecha\_compra[simulacion] <- round(runif(1, min = 1, max = dim(benchmark\_global)[1]))  fecha\_venta[simulacion] <- round(runif(1, min = fecha\_compra[simulacion], max = dim(benchmark\_global)[1]))  }  **# Obtenemos la cartera de inversión de cada simulación.**  # Hay dos opciones, 1ª que todos los activos tengan un % del dinero asignado.  # 2ª Que primero se obtenga el número de activos a invertir y luego se distribuya el dinero entre ellos.  # Optamos por la 1ª opción al estar construyendo un benchmark sintético.  pesos <- matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE)  sumatoria\_pesos <- matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE)  for (simulacion in 1:num\_simulaciones){  pesos[simulacion,]<-round(runif(dim(benchmark\_global)[2], min = 0, max = 100))  sumatoria\_pesos[simulacion]<-sum(pesos[simulacion,])  }  for (simulacion in 1:num\_simulaciones){ # Los activos de cada simulación no suman 100%, hay que dividirlos entre el total de la simulación para que sí lo hagan.  for (activo in 1:dim(benchmark\_global)[2]){  pesos[simulacion,activo]<-pesos[simulacion,activo]/sumatoria\_pesos[simulacion]  }  }  **# Calculamos el resultado de cada acción**  # rdo acción = (Precio venta - precio compra)\*número de acciones.  # precio compra = (precio \* divisa \* fecha de compra)  # precio venta = (precio \* divisa \* fecha de venta)  # número de acciones = dinero total \* % asignado a la acción / precio de compra.  dinero<-1000000  precios\_compra<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  num\_acciones<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  precios\_venta<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  rdo\_accion<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  for (simulacion in 1:num\_simulaciones){  for (activo in 1:dim(benchmark\_global)[2]){  precios\_compra[simulacion,activo]<-benchmark\_global[fecha\_compra[simulacion],activo]  num\_acciones[simulacion,activo]<-dinero\*pesos[simulacion,activo]/precios\_compra[simulacion,activo]  precios\_venta[simulacion,activo]<-benchmark\_global[fecha\_venta[simulacion],activo]  rdo\_accion[simulacion,activo]<-(precios\_venta[simulacion,activo]- precios\_compra[simulacion,activo])\*num\_acciones[simulacion,activo]  }  }  **# Agrupamos el rdo de cada activo para sacar el rdo de cada simulación**  rdo\_simulacion<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE))  for (simulacion in 1:num\_simulaciones){  rdo\_simulacion[simulacion]<-sum(rdo\_accion[simulacion,])  }  rdo\_simulacion<-t(rdo\_simulacion[1,]) # antes del for es un vector de 100x1, después una matriz de 100x100  **# Añadimos las columnas rdo\_simulación, fecha compra y fecha de venta al DF de pesos.**  pesos<-as.data.frame(pesos)  nombres\_activos<-c(names(DAX),names(FTSE),names(NASDAQ),names(HANGSENG),names(BOVESPA))  names(pesos)<-nombres\_activos # Ponemos nombres a las columnas  pesos$rdo\_simulacion<-rdo\_simulacion  pesos$fecha\_compra<-fecha\_compra  pesos$fecha\_venta<-fecha\_venta  **# Calculamos la rentabilidad de cada simulación**  rentabilidad<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE))  for (simulacion in 1:num\_simulaciones){  rentabilidad[simulacion]<-log((rdo\_simulacion[simulacion]+ dinero)/ dinero)  }  rentabilidad<-t(rentabilidad[1,])# antes del for, es un vector de 100x1 después una matriz de 100x100  pesos$rentabilidad<-rentabilidad  **# Sacamos la mejor simulación y los deciles de rentabilidad.**  mejor\_simulacion <- pesos[pesos$rentabilidad==max(pesos$rentabilidad),]  quantile(pesos$rentabilidad,c(0, .1, .2, .3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)) # Los deciles son el Benchmark sintético con el que puedes comparar tu rentabilidad obtenida.  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos del Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  DAX<- datos[,2:n\_activos] # Extraemos los activos quitando las 2 últimas columnas y la primera.  div\_EUR <- rep(1,dim(DAX)[1]) # Fabricamos el tipo de cambio.  **# Cargamos los datos del FTSE**  datos <- read.csv("FTSE.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  FTSE<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_GBP <- datos[,length(datos)] # Extraemos el tipo de cambio.  FTSE<-FTSE\*div\_GBP # Pasamos todos los precios a euros.  **# Cargamos los datos del NASDAQ**  datos <- read.csv("NASDAQ.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  NASDAQ<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_USD <- datos[,length(datos)] # Extraemos el tipo de cambio.  NASDAQ<-NASDAQ\*div\_USD # Pasamos todos los precios a euros.  **# Cargamos los datos del HANG SENG**  datos <- read.csv("HANG SENG.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  HANGSENG<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_HKD <- datos[,length(datos)] # Extraemos el tipo de cambio.  HANGSENG<-HANGSENG/div\_HKD # Pasamos todos los precios a euros.  **# Cargamos los datos de BOVESPA**  datos <- read.csv("BOVESPA.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  BOVESPA<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_BRL <- datos[,length(datos)] # Extraemos el tipo de cambio.  BOVESPA<-BOVESPA\*div\_BRL # Pasamos todos los precios a euros.  **# Unimos todos los activos en un único data.frame**  benchmark\_global<-data.frame(DAX,FTSE,NASDAQ,HANGSENG,BOVESPA)  num\_simulaciones<-50000  **# Sacamos las fechas de compra y venta de cada una de las simulaciones.**  **# Una manera de hacerlo sería:**  fechas <- round(runif(num\_simulaciones\*2, min = 1, max = dim(benchmark\_global)[1]))  fechas <- matrix(fechas, nrow=num\_simulaciones, ncol=2, byrow=TRUE) # Coloco las fechas en una matriz.  # fechas<-t(apply(t(fechas),2,sort)) # Ordeno las fechas (fecha de compra debe ser menor de venta). Funciona, pero esta línea tarda 1,91 seg.  fecha\_compra<-matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE) # Este sistema para ordenar las fechas tarda solo 0,03 seg.  fecha\_compra[1:num\_simulaciones]<-ifelse(fechas[1:num\_simulaciones,1]<fechas[1:num\_simulaciones,2], fecha\_compra[1:num\_simulaciones]<-fechas[1:num\_simulaciones,1], fecha\_compra[1:num\_simulaciones]<-fechas[1:num\_simulaciones,2])  fecha\_venta<-matrix(0, nrow=num\_simulaciones, ncol=1, byrow=TRUE)  fecha\_venta[1:num\_simulaciones]<-ifelse(fechas[1:num\_simulaciones,1]<fechas[1:num\_simulaciones,2], fecha\_venta[1:num\_simulaciones]<-fechas[1:num\_simulaciones,2], fecha\_venta[1:num\_simulaciones]<-fechas[1:num\_simulaciones,1])  **# Otra manera más sencilla de hacerlo sería:**  fecha\_compra<-round(runif(num\_simulaciones,min=1,max=dim(benchmark\_global)[1]),0)  fecha\_venta<- round(runif(num\_simulaciones,min=fecha\_compra,max=dim(benchmark\_global)[1]),0)  **# Obtenemos la cartera de inversión de cada simulación.**  # Hay dos opciones, 1ª que todos los activos tengan un % del dinero asignado.  # 2ª Que primero se obtenga el número de activos a invertir y luego se distribuya el dinero entre ellos.  # Optamos por la 1ª opción al estar construyendo un benchmark sintético.  pesos <- runif(dim(benchmark\_global)[2]\*num\_simulaciones, min = 0, max = 100) # pesos de todas simulaciones de una vez.  pesos <- matrix(pesos, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE)  pesos[,1:dim(benchmark\_global)[2]]<-(pesos[,1:dim(benchmark\_global)[2]]/rowSums(pesos))  **# Calculamos el resultado de cada acción**  # rdo acción = (Precio venta - precio compra)\*número de acciones.  # precio compra = (precio \* divisa \* fecha de compra)  # precio venta = (precio \* divisa \* fecha de venta)  # número de acciones = dinero total \* % asignado a la acción / precio de compra.  precios\_compra<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  precios\_compra[1:num\_simulaciones,]<-benchmark\_global[fecha\_compra[1:num\_simulaciones],]  dinero<-1000000  num\_acciones<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  num\_acciones<-dinero\*pesos/precios\_compra # Presuponemos que se puede comprar una fracción de acción.  precios\_venta<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  precios\_venta[1:num\_simulaciones,]<-benchmark\_global[fecha\_venta[1:num\_simulaciones],]  rdo\_accion<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  rdo\_accion<-(precios\_venta - precios\_compra)\*num\_acciones  **# Agrupamos el rdo de cada activo para sacar el rdo de cada simulación**  rdo\_simulacion<-rowSums(rdo\_accion)  **# Añadimos las columnas rdo\_simulación, fecha compra y fecha de venta al DF de pesos.**  pesos<-as.data.frame(pesos)  nombres\_activos<-c(names(DAX),names(FTSE),names(NASDAQ),names(HANGSENG),names(BOVESPA))  names(pesos)<-nombres\_activos # Ponemos nombres a las columnas  pesos$rdo\_simulacion<-rdo\_simulacion  pesos$fecha\_compra<-fecha\_compra  pesos$fecha\_venta<-fecha\_venta  **# Calculamos la rentabilidad de cada simulación**  pesos$rentabilidad<-log((pesos$rdo\_simulacion+1000000)/1000000) # sumamos el bº al capital inicial para calcular la rentabilidad.  **# Sacamos la mejor simulación y los deciles de rentabilidad.**  # Los deciles son el Bencmark sintético con el que puedes comparar tu rentabilidad obtenida.  mejor\_simulacion <- pesos[pesos$rentabilidad==max(pesos$rentabilidad),]  quantile(pesos$rentabilidad,c(0, .1, .2, .3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1))  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución: 74 minutos. | Tiempo de ejecución 5,86 segundos.  Ojo, con 1.000.000 de simulaciones consumimos muchísima RAM |

El que hayamos bajado el tiempo de ejecución a 5,86 segundos no implica que el resultado esté bien. Al utilizar runif para obtener los pesos de los activos para cada mono, estamos haciendo que prácticamente todos los monos tengan la misma cartera (un pequeño % en cada activo). La única diferencia es la fecha de compra y venta. Claramente, el benchmark sintético está mal construido.

|  |
| --- |
| Objetivo: Añade la habilidad de usar markowitz a los monos |
| library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro    **# Cantidad de monos**  num\_simulaciones<-50000  **# Cantidad de activos elegibles por mono para Markowitz**  num\_activos\_mono = 5  **# Ventana de cálculo de markowitz (120 aprox 6 meses)**  ventana\_mark = 240  **# Cantidad de simulaciones de cada mono para calcular markowitz**  num\_simulaciones\_mark\_mono <- 1000  **# Rango mínimo de días entre compra y venta**  min\_dias\_mono = 2  **# Dinero del que dispone el mono**  dinero<-1000000    **# Cargamos los datos del Dax**  datos <- read.csv("DAX.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-2  DAX<- datos[,2:n\_activos] # Extraemos los activos quitando las 2 últimas columnas y la primera.  div\_EUR <- rep(1,dim(DAX)[1]) # Fabricamos el tipo de cambio.    **# Cargamos los datos del FTSE**  datos <- read.csv("FTSE.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  FTSE<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_GBP <- datos[,length(datos)] # Extraemos el tipo de cambio.  FTSE<-FTSE\*div\_GBP # Pasamos todos los precios a euros.    **# Cargamos los datos del NASDAQ**  datos <- read.csv("NASDAQ.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  NASDAQ<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_USD <- datos[,length(datos)] # Extraemos el tipo de cambio.  NASDAQ<-NASDAQ\*div\_USD # Pasamos todos los precios a euros.    **# Cargamos los datos del HANG SENG**  datos <- read.csv("HANG SENG.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  HANGSENG<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_HKD <- datos[,length(datos)] # Extraemos el tipo de cambio.  HANGSENG<-HANGSENG/div\_HKD # Pasamos todos los precios a euros.    **# Cargamos los datos de BOVESPA**  datos <- read.csv("BOVESPA.csv", sep=';',stringsAsFactors = FALSE)  n\_activos <- length(datos)-1  BOVESPA<- datos[,2:n\_activos] # Extraemos los activos quitando la última columna y la primera.  div\_BRL <- datos[,length(datos)] # Extraemos el tipo de cambio.  BOVESPA<-BOVESPA\*div\_BRL # Pasamos todos los precios a euros.    **# Unimos todos los activos en un único data.frame**  benchmark\_global<-data.frame(DAX,FTSE,NASDAQ,HANGSENG,BOVESPA)    **# Calculamos la rentabilidad de los activos.**  rent\_activos<-as.data.frame(matrix(0,nrow=dim(benchmark\_global)[1],ncol=length(benchmark\_global),byrow=F))  names(rent\_activos)<-names(benchmark\_global)  rent\_activos[2:dim(benchmark\_global)[1],1:length(benchmark\_global)] <- log(benchmark\_global[2:dim(benchmark\_global)[1],1:length(benchmark\_global)]/benchmark\_global[2:dim(benchmark\_global)[1]-1,1:length(benchmark\_global)])  rent\_activos<-rent\_activos[2:dim(benchmark\_global)[1],1:length(benchmark\_global)]    activos\_disponibles = c(1:dim(benchmark\_global)[2])  fechas\_disponibles = c((ventana\_mark+1):dim(benchmark\_global)[1])    **# Generamos una función que calcule los activos de cada mono**  calcular\_mono <- function(){    fechas\_mono <- sample(fechas\_disponibles,2)  fecha\_compra\_mono <- sample((ventana\_mark+1):(dim(benchmark\_global)[1]-min\_dias\_mono),1)  fecha\_venta\_mono <- sample((fecha\_compra\_mono+min\_dias\_mono):dim(benchmark\_global)[1],1)  activos\_seleccionados\_mono = sample(activos\_disponibles,num\_activos\_mono)    activos\_mono<-benchmark\_global[(fecha\_compra\_mono-ventana\_mark):fecha\_compra\_mono,activos\_seleccionados\_mono]  rent\_activos\_mono<-rent\_activos[(fecha\_compra\_mono-ventana\_mark):fecha\_compra\_mono,activos\_seleccionados\_mono]    #Descomentar este bloque para descartar activos que tengan toda la rentabilidad a 0  # col\_sums=colSums(rent\_activos\_mono!=0)  # if (any(col\_sums==0)){  # activos\_seleccionados\_mono = activos\_seleccionados\_mono[-which(col\_sums==0)]  # if (length(activos\_seleccionados\_mono)<num\_activos\_mono){  # if (length(activos\_seleccionados\_mono)==0){  # print("todo cero")  # return(calcular\_mono())  # }  # activos\_mono<-benchmark\_global[(fecha\_compra\_mono-ventana\_mark):fecha\_compra\_mono,activos\_seleccionados\_mono]  # rent\_activos\_mono<-rent\_activos[(fecha\_compra\_mono-ventana\_mark):fecha\_compra\_mono,activos\_seleccionados\_mono]  # }  # }    matriz\_correlaciones\_mono<-cor(rent\_activos\_mono)  matriz\_var\_covarianzas\_mono<-cov(rent\_activos\_mono)    rent\_diaria\_mono <-as.data.frame(matrix(0,nrow=1,ncol=length(activos\_mono),byrow=F))  names(rent\_diaria\_mono)<-names(activos\_mono)    rent\_diaria\_mono<-t(rent\_diaria\_mono)  activos\_mono<-t(activos\_mono)  rent\_diaria\_mono[1:dim(activos\_mono)[1]]<- log(activos\_mono[1:dim(activos\_mono)[1],dim(activos\_mono)[2]] /activos\_mono[1:dim(activos\_mono)[1],1])/dim(rent\_activos\_mono)[1]  rent\_diaria\_mono<-t(rent\_diaria\_mono)  activos\_mono<-t(activos\_mono)    pesos\_mono <- runif(dim(activos\_mono)[2]\*num\_simulaciones\_mark\_mono, min = 0, max = 100)  pesos\_mono <- matrix(pesos\_mono, nrow=num\_simulaciones\_mark\_mono, ncol=dim(activos\_mono)[2], byrow=TRUE)  pesos\_mono[,1:dim(activos\_mono)[2]]<-(pesos\_mono[,1:dim(activos\_mono)[2]]/rowSums(pesos\_mono))    rentabilidad\_carteras\_mono <- c(1:num\_simulaciones\_mark\_mono)  matriz\_intermedia\_mono <- matrix(0,nrow=dim(activos\_mono)[1],ncol=dim(activos\_mono)[2],byrow=F)  matriz\_intermedia\_mono<-t(as.vector(rent\_diaria\_mono)\*t(pesos\_mono))  rentabilidad\_carteras\_mono<-rowSums(matriz\_intermedia\_mono)    riesgo\_carteras\_mono <- c(1:num\_simulaciones\_mark\_mono)  matriz\_intermedia\_mono <- pesos\_mono[1:num\_simulaciones\_mark\_mono,]%\*%matriz\_var\_covarianzas\_mono  matriz\_intermedia\_mono <- matriz\_intermedia\_mono \* pesos\_mono  riesgo\_carteras\_mono <- rowSums(matriz\_intermedia\_mono)  riesgo\_carteras\_mono <- riesgo\_carteras\_mono^0.5    eficiencia\_carteras\_mono <- c(1:num\_simulaciones\_mark\_mono)  eficiencia\_carteras\_mono[1:num\_simulaciones\_mark\_mono] <- (rentabilidad\_carteras\_mono[1:num\_simulaciones\_mark\_mono] /riesgo\_carteras\_mono[1:num\_simulaciones\_mark\_mono])    pesos\_mono=cbind(pesos\_mono,eficiencia\_carteras\_mono)  max\_eficiencia\_mono<-pesos\_mono[pesos\_mono[,dim(pesos\_mono)[2]]==max(pesos\_mono[,dim(pesos\_mono)[2]]),1:(dim(pesos\_mono)[2]-1)]    resultado\_mono = c(1:dim(benchmark\_global)[2])\*0  resultado\_mono[activos\_seleccionados\_mono] = unlist(max\_eficiencia\_mono[1:dim(activos\_mono)[2]])    #añado las fechas al final del resultado  resultado\_mono = c(resultado\_mono,fecha\_compra\_mono, fecha\_venta\_mono)    return(resultado\_mono)  }    pesos = t(replicate(num\_simulaciones,calcular\_mono()))    #quito las fechas del final del resultado  fecha\_compra = pesos[,dim(pesos)[2]-1]  fecha\_venta = pesos[,dim(pesos)[2]]  pesos = pesos[,1:(dim(pesos)[2]-2)]    **# Calculamos el resultado de cada acción**  # rdo acción = (Precio venta - precio compra)\*número de acciones.  # precio compra = (precio \* divisa \* fecha de compra)  # precio venta = (precio \* divisa \* fecha de venta)  # número de acciones = dinero total \* % asignado a la acción / precio de compra.    precios\_compra<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  precios\_compra[1:num\_simulaciones,]<-benchmark\_global[fecha\_compra[1:num\_simulaciones],]    num\_acciones<-dinero\*t(t(pesos)/t(precios\_compra)) # Presuponemos que se puede comprar una fracción de acción.  num\_acciones<-as.data.frame(num\_acciones)    precios\_venta<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  precios\_venta[1:num\_simulaciones,]<-benchmark\_global[fecha\_venta[1:num\_simulaciones],]    rdo\_accion<-as.data.frame(matrix(0, nrow=num\_simulaciones, ncol=dim(benchmark\_global)[2], byrow=TRUE))  rdo\_accion<-(precios\_venta - precios\_compra)\*num\_acciones    **# Agrupamos el rdo de cada activo para sacar el rdo de cada simulación**  rdo\_simulacion<-rowSums(rdo\_accion)    **# Añadimos las columnas rdo\_simulación, fecha compra y fecha de venta al DF de pesos.**  pesos<-as.data.frame(pesos)  nombres\_activos<-c(names(DAX),names(FTSE),names(NASDAQ),names(HANGSENG),names(BOVESPA))  names(pesos)<-nombres\_activos  pesos$rdo\_simulacion<-rdo\_simulacion  pesos$fecha\_compra<-fecha\_compra  pesos$fecha\_venta<-fecha\_venta    **# Calculamos la rentabilidad de cada simulación**  pesos$rentabilidad<-log((pesos$rdo\_simulacion+dinero)/dinero) # sumamos el bº al capital inicial para calcular la rentabilidad.    **# Sacamos la mejor simulación y los deciles de rentabilidad.**  # Los deciles son el Bencmark sintético con el que puedes comparar tu rentabilidad obtenida.  mejor\_simulacion <- pesos[pesos$rentabilidad==max(pesos$rentabilidad),]  quantile(pesos$rentabilidad,c(0, .1, .2, .3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1))  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución: 49,6 segundos. |

Pero aquí estamos sacando de nuevo 5 activos por cada cartera. ¿Cómo podríamos hacer que cada mono tuviera un número distinto de activos?

1. Dejando de usar runif y usando sample.
2. En vez de elegir los 5 activos con valor más alto, usando un umbral. Por ejemplo, podríamos usar el nº de activos que fuera superior al 0.99 en runif. Deberemos tener cuidado, porque habrá carteras que no tendrán activos, pero el resto tendrán 2, 3, 5, 7 activos… los que toquen. Y su número será aleatorio.

# Genera un Benchmark sintético

|  |  |
| --- | --- |
| Objetivo: Programa un Benchmark sintético. | Objetivo: Añade el Ibex con dividendos a la comparación anterior. |
| # Programa un Benchmark sintético  # No siempre dispondremos de un índice de referencia con el que compararnos.  # Por ejemplo, si seleccionamos 50 empresas del S&P500, ¿con qué nos vamos a comparar?  # Necesitamos crear un índice sintético.  # Reconstruye la composición del índice en cada instante de tiempo.  # Haz una estrategia de Hold & Sell (con proporciones iguales), según entren y salgan los activos del índice.  tiempo <- proc.time() # Inicia el cronómetro  **# Cargamos los datos de los activos**  datos <- read.csv("data\_ibex.csv", sep=',',stringsAsFactors = FALSE)  datos$X<-gsub("\_0", "", datos$X)  datos$X<-gsub("\_1", "", datos$X)  **# Sacamos la composición del índice durante el periodo 2003 - 2019 y su cotización**  fechas <- sort(as.Date(unique(datos$X.1)))  activos <- unique(datos$X)  composicion <- as.data.frame(matrix(0,nrow=length(fechas),ncol=length(activos),byrow=F))  colnames(composicion)<- activos  rownames(composicion)<- fechas    cotizacion <- as.data.frame(matrix(0,nrow=length(fechas),ncol=length(activos),byrow=F))  colnames(cotizacion)<- activos  rownames(cotizacion)<- fechas    invertido<- as.data.frame(matrix(0,nrow=length(fechas),ncol=length(activos),byrow=F))  colnames(invertido)<- activos  rownames(invertido)<- fechas    resultado <- as.data.frame(matrix(0,nrow=length(fechas),ncol=1,byrow=F))  rownames(invertido)<- fechas    for (activo in 1:length(activos)){  composicion[rownames(composicion) %in% datos$X.1[datos$X==activos[activo]],activo]<-1  cotizacion[rownames(composicion) %in% datos$X.1[datos$X==activos[activo]],activo]<- datos[datos$X.1 %in% datos$X.1[datos$X==activos[activo]] & datos$X == activos[activo], 3]  }    cotizacion<-cotizacion\*composicion    **# Calculamos la proporción que tenemos en cada activo del índice, en cada instante de tiempo.**  # Suponemos pesos iguales para cada activo.  proporcion\_invertida <- composicion / rowSums(composicion)  **# Calculamos la rentabilidad que habría obtenido el índice sintético.**  # Presuponiendo una estrategia buy & hold, según entran y salen los activos del índice.  dinero <- 100000  **# Realizo la inversión inicial**  invertido[1,]<-dinero\*proporcion\_invertida[1,]    **# Recorro los días calculando el resultado obtenido**  for (dia in 2:dim(cotizacion)[1]){    **# Compruebo si ha cambiado la composición del índice**  if (all(composicion[dia,]==composicion[dia-1,])==T) {    **# Calculo la rentabilidad que los activos han obtenido en el día**  rentabilidad<- log(cotizacion[dia,]/cotizacion[dia-1,])  rentabilidad[is.na(rentabilidad)]<-0    **# Actualizo el resultado**  resultado[dia,1]<- sum(invertido[dia-1,]\*rentabilidad)    **# Actualizo la inversión**  invertido[dia,]<- invertido[dia-1,] + (invertido[dia-1,]\*rentabilidad)    } else {    **# Ha cambiado la composición (ha entrado o salido algún valor). Hay que rehacer el índice.**  dinero <- sum(invertido[dia-1,])  invertido[dia,]<-dinero\*proporcion\_invertida[dia,]    **# Este día no obtenemos ningún resultado**  }  }  **# Comprobamos la evolución de nuestro capital**  evolucion\_inversion<-rowSums(invertido)    **# Cargamos los datos del índice**  datos\_indice <- read.csv("benchmark\_ibex.csv", sep=',',stringsAsFactors = FALSE)  **# Sacamos los datos del Ibex (sin dividendos).**  # filtrando las fechas que nos interesan y guardando la columna de cierre  ibex<- as.data.frame(datos\_indice[datos\_indice$X=="ibex" & datos\_indice$X.1 %in% rownames(composicion),3])  colnames(ibex)<-"Ibex"  rownames(ibex)<- fechas    **# Graficamos el Ibex,enfrentándo al índice sintético**.  rentabilidad\_ibex<-c(0,log(ibex[2:dim(ibex)[1],1]/ibex[1:(dim(ibex)[1]-1),1]))  evolucion\_ibex<-rep(0,length(evolucion\_inversion))  evolucion\_ibex[1]<-evolucion\_inversion[1]  for (dia in 2:dim(ibex)[1]){  evolucion\_ibex[dia]<- evolucion\_ibex[dia-1]\*(1+rentabilidad\_ibex[dia])  }    **# Calculamos el máximo y mínimo para graficar.**  maximo<-max(c(evolucion\_inversion, evolucion\_ibex))  minimo<-min(c(evolucion\_inversion, evolucion\_ibex))    grafico<-plot(evolucion\_inversion~as.Date(fechas), main="Comparación Ibex VS Sintético",col="darkblue", type="l", cex.lab=1, cex.axis=0.6, lwd = 2, xlab="La línea roja representa al Ibex sin dividendos \ny la línea azul al Sintético", ylab = "", ylim=c(minimo, maximo))  lines(evolucion\_ibex~as.Date(fechas), col="red")  print(proc.time()-tiempo) | # Añade el Ibex con dividendos a la comparación anterior.  # Si queremos incluir los datos del Ibex con dividendos, debemos darnos cuenta de que las fechas no coinciden.  # No tenemos los mismos datos en el Ibex con dividendos que en el Ibex sin dividendos.  # Debemos adaptarnos a la serie temporal más corta, si queremos hacer una comparación.  tiempo <- proc.time()  **# Cargamos los datos del índice y obtenemos la fecha mínima y máxima**  datos\_indice <- read.csv("benchmark\_ibex.csv", sep=',',stringsAsFactors = FALSE)  fecha\_inicio <- as.Date(min(datos\_indice[datos\_indice$X == "ibex\_div",2]))  fecha\_fin <- as.Date(max(datos\_indice[datos\_indice$X == "ibex\_div",2]))  fechas <- unique(datos\_indice[datos\_indice$X.1 >= fecha\_inicio & datos\_indice$X.1 <= fecha\_fin,2])  **# Sacamos los datos del Ibex (sin dividendos), filtrando las fechas que nos interesan y guardando la columna de cierre**  ibex<- as.data.frame(datos\_indice[datos\_indice$X=="ibex" & datos\_indice$X.1 %in% fechas,3])  colnames(ibex)<-"Ibex"  rownames(ibex)<- fechas    ibex\_div<- as.data.frame(datos\_indice[datos\_indice$X=="ibex\_div" & datos\_indice$X.1 %in% fechas,3])  colnames(ibex\_div)<-"Ibex con dividendos"  rownames(ibex\_div)<- fechas  **# Cargamos los datos de los activos**  datos <- read.csv("data\_ibex.csv", sep=',',stringsAsFactors = FALSE)  datos$X<-gsub("\_0", "", datos$X)  datos$X<-gsub("\_1", "", datos$X)    **# Elimino los datos de cotización que están fuera del rango de fechas**  datos<- datos[datos$X.1 >= fecha\_inicio,]  datos<- datos[datos$X.1 <= fecha\_fin,]  **# Sacamos la composición del índice durante el periodo 2003 - 2019 y su cotización**  fechas <- sort(as.Date(unique(datos$X.1)))  activos <- unique(datos$X)  composicion <- as.data.frame(matrix(0,nrow=length(fechas),ncol=length(activos),byrow=F))  colnames(composicion)<- activos  rownames(composicion)<- fechas    cotizacion <- as.data.frame(matrix(0,nrow=length(fechas),ncol=length(activos),byrow=F))  colnames(cotizacion)<- activos  rownames(cotizacion)<- fechas    invertido<- as.data.frame(matrix(0,nrow=length(fechas),ncol=length(activos),byrow=F))  colnames(invertido)<- activos  rownames(invertido)<- fechas    resultado <- as.data.frame(matrix(0,nrow=length(fechas),ncol=1,byrow=F))  rownames(invertido)<- fechas  for (activo in 1:length(activos)){  composicion[rownames(composicion) %in% datos$X.1[datos$X==activos[activo]],activo]<-1  cotizacion[rownames(composicion) %in% datos$X.1[datos$X==activos[activo]],activo]<- datos[datos$X.1 %in% datos$X.1[datos$X==activos[activo]] & datos$X == activos[activo], 3]  }  cotizacion<-cotizacion\*composicion  **# Calculamos la proporción que tenemos en cada activo del índice, en cada instante de tiempo.**  # Suponemos pesos iguales para cada activo.  proporcion\_invertida <- composicion / rowSums(composicion)  **# Calculamos la rentabilidad que habría obtenido el índice sintético.**  # Presuponiendo una estrategia buy & hold, según entran y salen los activos del índice.  dinero <- 100000  **# Realizo la inversión inicial**  invertido[1,]<-dinero\*proporcion\_invertida[1,]    **# Recorro los días calculando el resultado obtenido**  for (dia in 2:dim(cotizacion)[1]){    **# Compruebo si ha cambiado la composición del índice**  if (all(composicion[dia,]==composicion[dia-1,])==T) {    **# Calculo la rentabilidad que los activos han obtenido en el día**  rentabilidad<- log(cotizacion[dia,]/cotizacion[dia-1,])  rentabilidad[is.na(rentabilidad)]<-0    **# Actualizo el resultado**  resultado[dia,1]<- sum(invertido[dia-1,]\*rentabilidad)    **# Actualizo la inversión**  invertido[dia,]<- invertido[dia-1,] + (invertido[dia-1,]\*rentabilidad)    } else {    **# Ha cambiado la composición (ha entrado o salido algún valor). Hay que rehacer el índice.**  dinero <- sum(invertido[dia-1,])  invertido[dia,]<-dinero\*proporcion\_invertida[dia,]    **# Este día no obtenemos ningún resultado**  }  }    **# Comprobamos la evolución de nuestro capital**  evolucion\_inversion<-rowSums(invertido)  **# Graficamos el Ibex,enfrentándo al índice sintético.**  rentabilidad\_ibex<-c(0,log(ibex[2:dim(ibex)[1],1]/ibex[1:(dim(ibex)[1]-1),1]))  evolucion\_ibex<-rep(0,length(evolucion\_inversion))  evolucion\_ibex[1]<-evolucion\_inversion[1]    rentabilidad\_ibex\_div<-c(0,log(ibex\_div[2:dim(ibex\_div)[1],1]/ibex\_div[1:(dim(ibex\_div)[1]-1),1]))  evolucion\_ibex\_div<-rep(0,length(evolucion\_inversion))  evolucion\_ibex\_div[1]<-evolucion\_inversion[1]  for (dia in 2:dim(ibex)[1]){  evolucion\_ibex[dia]<- evolucion\_ibex[dia-1]\*(1+rentabilidad\_ibex[dia])  evolucion\_ibex\_div[dia]<- evolucion\_ibex\_div[dia-1]\*(1+rentabilidad\_ibex\_div[dia])  }    **# Calculamos el máximo y mínimo para graficar.**  maximo<-max(c(evolucion\_inversion, evolucion\_ibex, evolucion\_ibex\_div))  minimo<-min(c(evolucion\_inversion, evolucion\_ibex, evolucion\_ibex\_div))    grafico<-plot(evolucion\_inversion~as.Date(fechas), main="Comparación Ibex VS Sintético",col="darkblue", type="l", cex.lab=1, cex.axis=0.6, lwd = 2, xlab="La línea roja representa al Ibex sin dividendos, la negra al Ibex con dividendos \ny la línea azul al Sintético", ylab = "", ylim=c(minimo, maximo))  lines(evolucion\_ibex~as.Date(fechas), col="red")  lines(evolucion\_ibex\_div~as.Date(fechas), col="black")  print(proc.time()-tiempo) |
| Tiempo de ejecución: 67 segundos. | Tiempo de ejecución: 67 segundos. |

# Optimiza un programa que calcula la distribución de pérdidas de una cartera

|  |  |
| --- | --- |
| Estima la distribución de pérdidas de una cartera (50.000 simulaciones) | Objetivo: Vectoriza el programa para que tarde menos de 1 segundo. |
| library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  Cartera<-read.table("Cartera.csv",header=T,sep=";",stringsAsFactors = F,dec=".")#Importamos el archivo  analisis\_estadistico<-function(p){    **# Cálculo de escenarios**  simulacion\_carteras<-data.frame(0,length(Cartera$Grupo),n\_escenarios) # DF guardar simulaciones  perdidas\_por\_escenario<-1:n\_escenarios # Creamos un vector vacío donde guardar las pérdidas.    for (escenario in 1:n\_escenarios){    **# Sacamos el valor de Y y los pesos de cada cartera**  y<- rnorm(1, mean = 0, sd = 1) # Sacamos el factor común de todas las exposiciones.  Cartera$w<- rnorm(length(Cartera$Grupo), mean = 0, sd = 1) # pesos de cada cartera .    **# Sabiendo que Zi = raiz(p)\*y + raiz(1-p)\*w, calculamos Zi**  Cartera$zi<-0  for (fila in 1:length(Cartera$zi)){  Cartera$zi[fila]<-sqrt(p)\*y+sqrt(1-p)\*Cartera$w[fila]  }    **# Calculamos las carteras que han hecho default**  for (fila in 1:length(Cartera$zi)){  if (Cartera$zi[fila] < qnorm(Cartera$PD[fila], mean = 0, sd = 1)){ # qnorm punto en función de prob.  **# Guardamos las pérdidas de cada cartera para poder recuperar el escenario del 95%**  simulacion\_carteras[fila,escenario]<-Cartera$EAD[fila]\*Cartera$LGD[fila]  }else{  simulacion\_carteras[fila,escenario]<-0  }  }  perdidas\_por\_escenario[escenario]<-sum(simulacion\_carteras[,escenario])  }    **# Ordenamos los escenarios de menor a mayor pérdida.**  simulacion\_carteras<-simulacion\_carteras[,(order(perdidas\_por\_escenario))]    **# Recuperamos el escenario que corresponde al percentil 95% y lo metemos en el DF original.**  Cartera$escenario\_95<- simulacion\_carteras[,round(n\_escenarios\*0.95)]    **# Identificamos qué exposiciones dan las mayores pérdidas en dicho escenario**  Cartera[Cartera$escenario\_95>0,]    **# Comparamos el peso de la perdida de cada exposición en el escenario 95 con el peso EAD**.  Cartera$porcentaje\_EAD <- Cartera$EAD/sum(Cartera$EAD)  Cartera$porcentaje\_esc95 <- Cartera$escenario\_95/sum(Cartera$escenario\_95)    **# Hacemos la comparación a nivel de grupo y de segmento y guardamos los resultados.**  library(data.table)  Cartera<-data.table(Cartera) #Convierto el data frame en un data table.    **# Guardamos el % EAD y % escenario 95**  resultados[[length(resultados)+1]] <- Cartera[,lapply(.SD,sum),by=Grupo,.SDcols=9:10]  resultados[[length(resultados)+1]] <- Cartera[,lapply(.SD,sum),by=Segmento,.SDcols=9:10]  return (resultados)  }  **# Ejecutamos el programa**  resultados <- list()  n\_escenarios <- 50000  p<- c(0.3,0.1)  lapply(p,analisis\_estadistico)  print(proc.time()-tiempo)  }) | library(profvis)  profvis({  tiempo <- proc.time() # Inicia el cronómetro  n\_escenarios <- 50000  Cartera <- read.csv('Cartera.csv', header = T, dec = ".", stringsAsFactors = F, sep = ";")  Cartera$perd.esper <- Cartera$PD \* Cartera$LGD \* Cartera$EAD # pérdida esperada  Cartera$perd.default <- Cartera$EAD \* Cartera$LGD # pérdida asociada a default  Cartera$cuantil <- qnorm(Cartera$PD, mean = 0, sd = 1)  num\_exposic <- length(Cartera$LGD) # Número de exposiciones “filas” = 103  EAD\_total <- sum(Cartera$EAD) # EAD total  y <- rnorm(n\_escenarios, mean = 0, sd = 1) # vector con estado de la economía  xi <- rnorm(n\_escenarios\*num\_exposic, mean = 0, sd = 1) # N(0,1) # vector 103\*10000    **# Transformamos el vector en una matriz de 50.000 filas (escenarios) por 103 columnas**  xi <- matrix(xi, nrow=n\_escenarios, ncol=num\_exposic, byrow=TRUE)    **# Calcular las pérdidas de la cartera**  p <- 0.1  zi <- sqrt(p)\*y+sqrt(1-p)\*xi # filas = escenarios y columnas = carteras    # zi(filas escenarios,columnas exposiciones) < Cartera$cuantil(filas exposiciones, transponer)  # Cartera$cuantil es un vector de 4. Para poder compararlo con la matriz hay que hacer la traspuesta.  # Hay que hacer la traspuesta al rdo para que vuelva a su posición original.  def <- t(t(zi) < Cartera$cuantil) # Matriz de true o False  simulacion\_carteras <- t(def)\*Cartera$perd.default  perdidas\_por\_escenario <- colSums(simulacion\_carteras)    **# Ordenamos los escenarios de menor a mayor pérdida.**  simulacion\_carteras<-simulacion\_carteras[,(order(perdidas\_por\_escenario))]    **# Recuperamos el escenario que corresponde al percentil 95% y lo metemos en el DF original.**  Cartera$escenario\_95<- simulacion\_carteras[,round(n\_escenarios\*0.95)]    **# Comparamos el peso de la perdida de cada exposición en el escenario 95 con el peso EAD.**  Cartera$porcentaje\_EAD <- Cartera$EAD/sum(Cartera$EAD)  Cartera$porcentaje\_esc95 <- Cartera$escenario\_95/sum(Cartera$escenario\_95)    **# Hacemos la comparación a nivel de grupo y de segmento.**  Cartera<-data.table(Cartera) # Convierto el data frame en un data table.    **# Mostramos el % EAD y % escenario 95**  print(Cartera[,lapply(.SD,sum),by=Grupo,.SDcols=10:11])  print(Cartera[,lapply(.SD,sum),by=Segmento,.SDcols=10:11])  print(proc.time()-tiempo)  }) |
| Tiempo de ejecución: + de 2 horas | Tiempo de ejecución 0,82 segundos |

# Ejercicios voluntarios (no incrementan puntuación)

## Programa el ratio de Treynor

Mide el diferencial de rentabilidad obtenida sobre el activo libre de riesgo por unidad de riesgo sistemático (no diversificable).

Cuanto mayor sea el ratio de Treynor mejor habrá sido la gestión.



## Programa el índice de Modigliani

Indica la rentabilidad que obtendría el fondo en el supuesto de que tuviera todos los fondos tuvieran el mismo nivel de riesgo que su categoría.

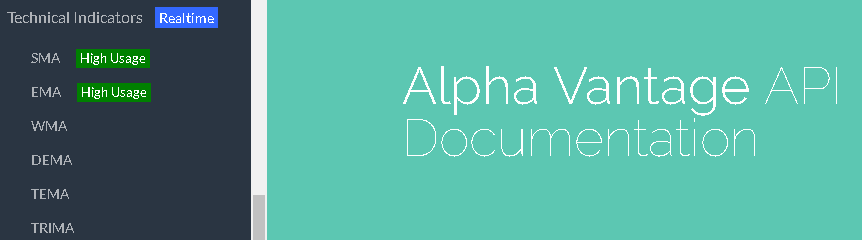
El índice homogeneiza el nivel de riesgo de los distintos fondos de forma que podamos comparar sus rentabilidades, o lo que es lo mismo: calcula la rentabilidad que tendrían los diferentes fondos en el caso de que tuvieran políticas de inversión similares. El nivel de riesgo que se utiliza es la volatilidad de la categoría.

Cuanto mayor es el índice de Modigliani, mejor ha sido la gestión de la cartera.



## Ratios técnicos (Alpha Vantage)

<https://www.alphavantage.co/documentation/>



Todos los indicadores tienen documentación en donde puedes consultar cómo programarlo. Por ejemplo:

