Reporte práctica 7

Búsqueda local

A 26 de Septiembre de 2017

Búsqueda local

Cuando se necesita optimizar un proceso se hace uso del método de **búsqueda local**[1], la característica principal de este método es la realización de movimientos en el espacio, cada uno de estos movimientos representa una solución, la cual puede ir mejorando, en pocas palabras, la búsqueda local inicia con una solución inicial, la cual se reemplaza con una nueva hasta encontrar la mejor solución y que no exista una solución mejor.

Práctica

En esta práctica se realizó la maximización de una función bidimensional

$g(x,y) = \frac{((x+0.5)^4 - 30x^2 - 20x + (y+0.5)^4 - 30y^2 - 20y)}{100}$, con el método de búsqueda local.

El código original sirve para buscar el mínimo en una función unidimensional. A partir del código original se buscó la forma de encontrar el máximo para una función bidimensional.

El código original se modificó de la siguiente manera, como se muestra en la figura 1.

```
1. suppressMessages(library(lattice))
2. suppressMessages(library(doParallel))
3. suppressMessages(library(ggplot2))
4. registerDoParallel(makeCluster(detectCores(2))) # número de núcleos
   utilizables en la pc
```

Fig. 1. Fragmento modificado del código original.

Como se ve en la figura 1 se activaron los paquetes **lattice**, **doParallel** y **ggplot2**, y se agregaron al comando **suppressMessages(library())**, los cuales se utilizaron para la paralelización del código y la generación de los gráficos, también se creó un ***cluster*** que utiliza dos núcleos para llevar a cabo la paralelización, este número de núcleos se definió en base a la capacidad de la computadora.

```
1. low <- -2
2. high <- 3
3. step <- 0.25
4. replicas <- 3
5. t <- 25
6. wfa <- 0.0666822 # resultado de wolfram alpha
```

Fig. 2. Variables de código.

En la figura 2 se muestran las variables que se mantuvieron y las que se agregaron al código, estas son ***t*** para el tiempo y ***wfa*** que es el resultado obtenido en Wolfram Alpha.

```
1. LS <- function() {
2.
3.  resultado <- data.frame()
4.  x <- runif(1, low, high) # posición en x
5.  y <- runif(1, low, high) # posición en y
6.  best <- c(x, y) # mejor posición en x, y
```

Fig. 3. Función aplicada al código.

En la figura 3 se muestra la función ***LS()*** creada para el uso del paquete **doParallel**, se agregó un **data.frame** nombrado resultado para guardar los resultados de las operaciones. Modificando el código original el vector ***curr*** se cambió a ***x*** para las posiciones en el eje x y se agregó un vector ***y*** para las posiciones en el eje y, y se guardó un vector llamado ***best*** para guardar las mejores posiciones en ***x*** y en ***y***.

```
1. if (f(left, y) > f(right, y)) {
2.   bestx <- c(left, y) # mejor posición a la izquierda
3. } else {
4.   bestx <-c(right, y) # mejor posición a la derecha
5. }
6. if (f(x, up) > f(x, down)) {
7.   besty <- c(x, up) # mejor posición arriba
8. } else {
9.   besty <- c(x, down) # mejor posición abajo
```

Fig. 4. Función lógica para tomar posiciones en ***y***.

En la figura 4 se muestra como se agregaron los comandos lógicos **if** y **else** para poder elegir el mejor valor para la posición en ***y***, repitiendo el comando original y cambiando los valores de ***x*** por los de ***y***.

```
1. if (f(x, y) > f(best[1], best[2])) {
2.   best <- c(x, y)
3.    }
4.    res <- cbind(i, tiempo, best[1], best[2], f(best[1], best[2]))
5.    resultado <- rbind(resultado, res)
6.  }
7.  return(resultado)
8. }
```

Fig. 5. Definición de los valores del vector ***best***.

En la figura 5 se definen los valores que tomará el vector ***best***, esto se hace usando un comando lógico donde si la función ***f(x, y)*** es mayor a la mejor posición en ***x*** e ***y***, ***best*** tomará los valores de ***f(x, y)***.

```
1. max <- foreach(i = 1:replicas, .combine=rbind) %dopar% LS() # usar
   comando foreach para aplicar a toda la función
2. names(max) <- c("replicas", "tiempo", "x", "y", "f(x, y)") # usar names
   para nombrar el objeto
3. stopImplicitCluster()
4.
5. max$replicas <- as.factor(max$replicas)
6. ggplot()+geom_line(data = max, aes(x = tiempo, y = max$`f(x,
   y)`, color = replicas), size=1)+
7.  geom_hline(yintercept = wfa,
   col = "black")+xlab("Pasos")+ylab("Máximos de f(x, y)")
```

Fig. 6. Comandos para graficar.

En la figura 6 se observa la creación de una función ***max***, la cual se usa con el comando **foreach** para hacer repeticiones, con una variable ***i*** se colocó el intervalo en el que se trabajó, se agregó la función **.combine = rbind** para que está nos arrojará los datos en forma de matriz acomodados por filas, al final del comando **foreach** se agregó la función ***LS()*** para que hiciera los cálculos en paralelo y guardar los datos en la función ***LS()***. Se utilizó la función **name** para darle nombre a las columnas de la matriz. Se utilizó de igual manera la función **as.factor**, para convertir en factor el número de réplicas y poder utilizarlo para graficar, con el comando **ggplot() + geom_line** para crear la gráfica de las repeticiones y se utilizó el comando **geom_hline** para agregar una recta horizontal en el valor de wólfram alpha.[3]

Resultados

Como resultado se obtuvieron gráficos en los que se muestran el tiempo contra el máximo en la función f(x, y). En la figura 7 se muestra la gráfica para una corrida del código.

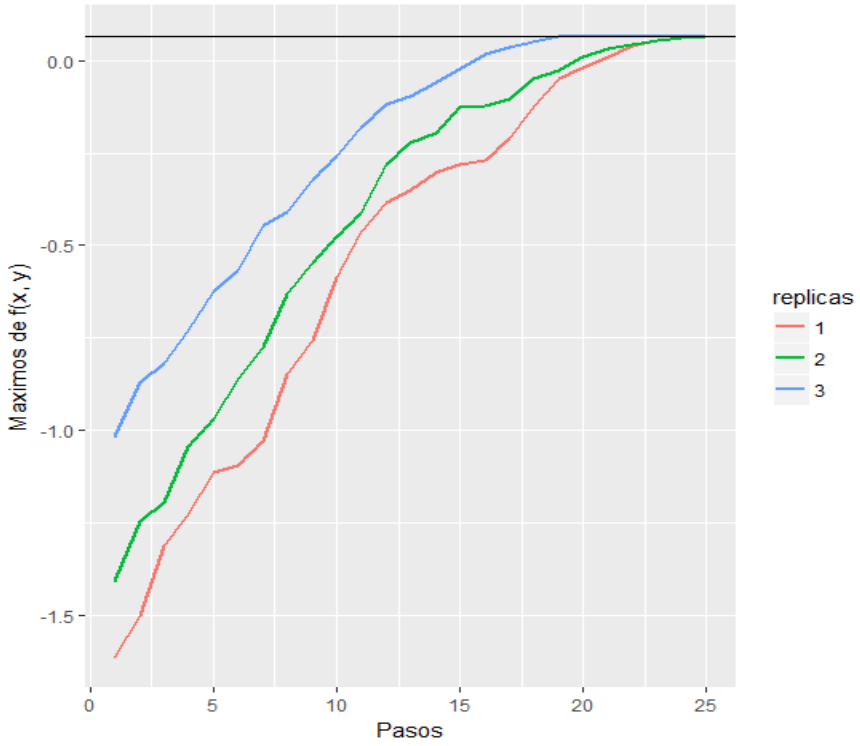

Fig. 7. Gráfica de puntos para tres repeticiones de ***f(x, y)***.

En la figura 7 se muestra la gráfica para tres corridas y se observa que el incremento corridas o repeticiones ayuda a mejorar el tiempo en el que se puede encontrar el máximo de una función como lo es la función ***f(x, y)***.

Conclusiones

De esta práctica se puede concluir que los métodos de búsqueda local son importantes cuando se necesita encontrar los máximos posibles o las mejores soluciones a algún problema. En la práctica 7 se puede concluir que la obtención de una mejor solución depende tanto del tiempo (número de pasos) como del número de repeticiones que se le dan al código, ya que al repetir el código encuentra la solución más rápidamente.

Reto 1

En el reto 1 hay que demostrar de manera visual como el código encuentra el valor máximo para la función ***g(x, y)***. Se utilizó el código original de la práctica 7 para graficar el mapa de calor donde

los valores máximos se encuentran en las zonas de color azul y los valores mínimos se encuentran en las zonas de color violeta.

En la figura 9 se muestran los paquetes que se activaron para el reto 1.

```
1. suppressMessages(library(lattice))
2. suppressMessages(library(latticeExtra))
3. suppressMessages(library(reshape2))
```

Fig. 9. Paquetes para utilizar en el reto 1.

En la figura 9 se muestran los paquetes que se utilizaron para el reto 1, se usaron **lattice**, **latticeExtra** y **reshape2** para hacer las gráficas e imágenes de este reto.

```
1. x <- seq(-6, 5, 0.25)
2. y <-  x
3. z <- outer(x, y, f)
4. dimnames(z) <- list(x, y)
5. d <- melt(z)
6. names(d) <- c("x", "y", "z")
7. levelplot(z ~ x * y, data = d)
```

Fig. 10. Variables para reto 1.

En la figura 10 se muestran las variables que se utilizaron, al igual que en la práctica 7, las variables se mantuvieron como en el código original.

```
1. trellis.device(device = "png", filename = paste("melt",
   tiempo, ".png"))
2. print(levelplot(z ~ x * y, data = d) + as.layer(xyplot(y ~ x, pch = 6,
   col = "red")))
3. dev.off()
4. graphics.off()
```

Fig. 11. Comando aplicado de **latticeExtra**.

En la figura 11 se muestran los comandos utilizados para generar múltiples imágenes del mapa de calor donde se agregó una figura para ver como avanzaba hasta el punto máximo del mapa de calor.[4, 5]

Resultados

Como resultados se obtuvieron varios mapas de calor en los cuales se mostraba el valor de la función ***f(x, y)***, de estos se crearon varias réplicas o repeticiones, las cuales se encuentran en el siguiente enlace https://github.com/PabloChavez94/Simulacion/tree/master/p7/reto1.

En estos gifs se puede ver cómo es que el valor de ***g(x, y)*** puede iniciar en cualquier punto del mapa de calor y el valor de ***g(x, y)*** se acerca al punto máximo o valor de wólfram alpha.

Conclusiones del reto 1

Del reto 1 se puede concluir que con el método de búsqueda local, no importa donde inicie la búsqueda del o de los máximos, siempre llegará al máximo local sin importar su ubicación, al llegar a este máximo ya no mejorará esa solución, esto en el caso del reto 1. También es importante decir que el número de repeticiones también hacen que encontrar o acercarse al máximo de una función es más rápido con el aumento de las repeticiones.

Bibliografía

1) https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node58.html
2) Blas Pelegrín Pelegrín, (2005), *Avances en localización de servicios y sus aplicaciones*, Ed. EDITUM.
3) Alboukadel Kassambara, (2013), *Guide to create beautiful graphics in R*.
4) https://www.stat.auckland.ac.nz/~ihaka/787/lectures-trellis.pdf
5) https://stat.ethz.ch/R-manual/R-devel/library/lattice/html/trellis.device.html