Práctica No 7: Búsqueda Local

Héctor Hernán Montes García

26 de septiembre de 2017

Resumen

En la presente práctica estudiamos una aplicación de búsqueda local para la maximización de una función bivariada sobre un conjunto acotado rectangular. La implementación del método se hace en R acompañada de apoyos gráficos para la mejor visualización de la forma en la que el método procede. Dado que el método es proclive a quedar estancado en óptimos locales, se realiza una implementación de un algoritmo de recocido simulado el cual tiene la ventaja de explorar más ampliamente el espacio de búsqueda permitiendo escapar de óptimos locales. Los métodos se comparan estudiando la cercanía de las soluciones encontradas con cada método al óptimo teórico, y se realizan también diversas experimentaciones para estudiar cómo afectan los parámetros de los métodos a la calidad de la solución.

1. Planteamiento del problema

El problema principal consiste en maximizar la función:

$$g(x,y) = \frac{(x+\frac{1}{2})^4 - 30x^2 - 20x + (y+\frac{1}{2})^4 - 30y^2 - 20y}{100} \tag{1}$$

Sobre el rectángulo definido por $-6 \leq x \leq 5$, y $-6 \leq y \leq 5$. Una gráfica de la función se ofrece en la figura 1

El problema fue pasado a Wolfram Alpha[1] para contar con una solución de referencia que sirviera de comparativo para nuestros métodos a implementar. La figura 2, muestra un esquema de la solución encontrada mediante Wolfram Alpha la cual arroja máximo global en el punto extremo $(5, 5)$ con un valor de función objetivo máximo de $\frac{1041}{800}$. Por otro lado, la figura 3 nos muestra el gráfico de contorno para la función objetivo. Como se puede apreciar, se nota también la presencia de un máximo local en las coordenadas $(-0{,}333023, -0{,}333023)$, donde la función objetivo alcanza el valor de 0,0666822.

La tarea consiste en resolver el problema antes expuesto usando el método de búsqueda local aportado en clase, el cual consiste de la elección aleatoria de un punto de coordenadas $X_0 = (x_0, y_0)$ sobre el rectángulo de búsqueda, que sirve como mejor solución inicial. Posteriormente se genera un segundo punto, mediante un movimiento aleatorio desde la solución inicial y en la dirección $(\Delta x_1, \Delta y_1)$, donde Δx_1 y Δy_1 se eligen de una distribución uniforme $(0, s)$ siendo s una longitud de paso máxima prefijada. Si en esta primera iteración del

[1]Nos referimos al sitio de internet https://www.wolframalpha.com, en donde se pueden resolver problemas matemáticos típicos.

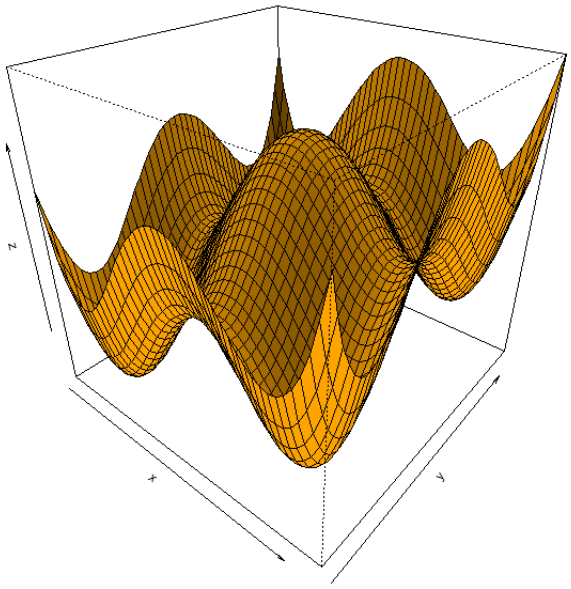

Figura 1: Gráfico de la función objetivo

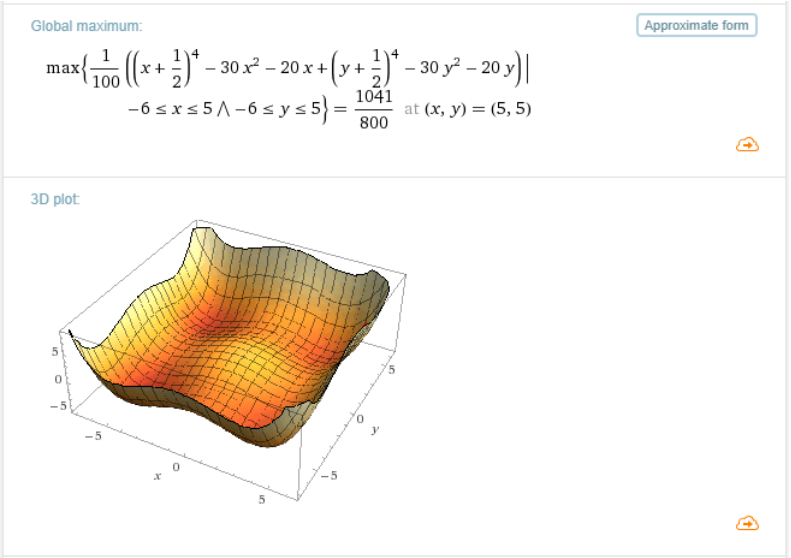

Figura 2: Solución aportada por Wolfram Alpha

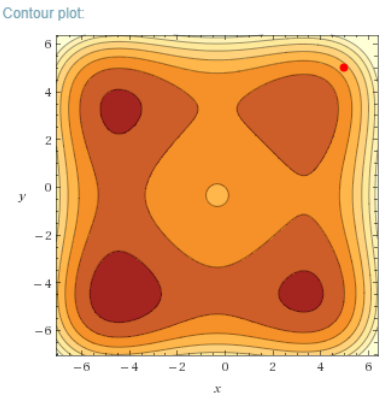

Figura 3: Gráfico de contorno para la función objetivo aportada por Wolfram Alpha

método se encuentra un punto $X_1 = (x_0+\Delta x_1, y_0+\Delta y_1)$ tal que $g(X_1) \geq g(X_0)$, este punto pasa a ser la mejor solución. El proceso se repite iterativamente por un número de pasos máximo t_{max}, en forma tal que se garantice que cualquier solución que no mejore o iguale la mejor obtenida en los pasos anteriores simplemente se deseche.

Un método de solución como éste, es denominado de búsqueda local, precisamente por la particularidad que tiene de quedar atrapado en óptimos locales, es decir, si el proceso generador de direcciones aleatorias de longitud de paso especificada no es capaz de encontrar en dicha vecindad de la solución actual, una mejor solución, entonces el método queda lógicamente atrapado en ese óptimo local.

La segunda cuestión es representar en forma gráfica el mecanismo de funcionamiento del método de búsqueda local, y la tercera cuestión es resolver el mismo problema mediante un algoritmo de recocido simulado. A continuación una descripción de la solución dada a los tres problemas.

2. Resultados

Todas las experimentaciones se han realizado en R, y para el efecto se usó una computadora con sistema operativo Windows 10 Pro de 64bits y procesadorx64 Intel(R) core(TM) i5-4210U, CPU@ 1.70GHz 2.40GHz y memoria instalada de 6.00GB. Una llamada a la función *detectcores(logical=FALSE)* de R, permitió además constatar que la computadora posee dos núcleos físicos disponibles. A continuación se describe las modificaciones más importantes realizadas al código suministrado, el cual debió ser adaptado para el caso de una función de dos variables.

2.1. Algoritmo de optimización por búsqueda local

Los siguientes son los cambios efectuados sobre el código original:

- El primer cambio fue ajustar la función *replica"* de forma que contemplara la creación de cuatro vecinos para el punto en cada iteración, en lugar de

dos, en concordancia con las instrucciones dadas en el problema. Cada vecino fue construido haciendo uso de los mismos valores de Δx y Δy excepto que variando sus signos para obtener puntos arriba, abajo, a la izquierda, y la derecha del punto actual.

- El segundo cambio fue ajustar los puntos obtenidos como vecinos de forma que ninguno violara las condiciones de factibilidad. Es decir, en los casos donde la componente x o y desbordaban los límites de variación establecidos, se aplicaba un ajuste al valor extremo más cercano. Este criterio nos pareció apropiado en virtud de su sencillez computacional.Además si la función objetivo crece en la dirección donde el punto se está desbordando, se aprovecha parte de esta información recortando la longitud del paso. Es decir el punto no se desecha del todo, lo cual nos parece un criterio razonable.

- Los valores de retorno de la función también fueron modificados, en forma tal que suministrara tanto la mejor solución encontrada después de cierta cantidad máxima de pasos como el valor de la función objetivo en ese punto.

- Finalmente se hicieron modificaciones sobre la parte paralelizada, con el fin de ejecutar diversas búsquedas con cantidades diferentes de pasos: 10, 100, 1000, y 10000 pasos y para cada cantidad de pasos, se procedió a generar un gráfico de la distribución de las soluciones sobre el mapa de niveles de la función. Para el efecto se usó la librería *ggplot2* que ofrece mayor versatilidad para la manipulación de diferentes series de datos sobre un mismo gráfico (Se usa una serie de datos para los puntos solución y otra para la generación del mapa de niveles en sí mismo).

El código de la función réplica así como la parte paralelizada se puede consultar en el repositorio, sin embargo facilitamos la parte de la función réplica para rápida referencia. Conviene mencionar que para esta experimentación el parámetro de paso se ajustó al valor $s = 0{,}25$. La distribución de soluciones sobre el rectángulo de factibilidad se muestra en la secuencia de figuras 5, 6, 7 y 8

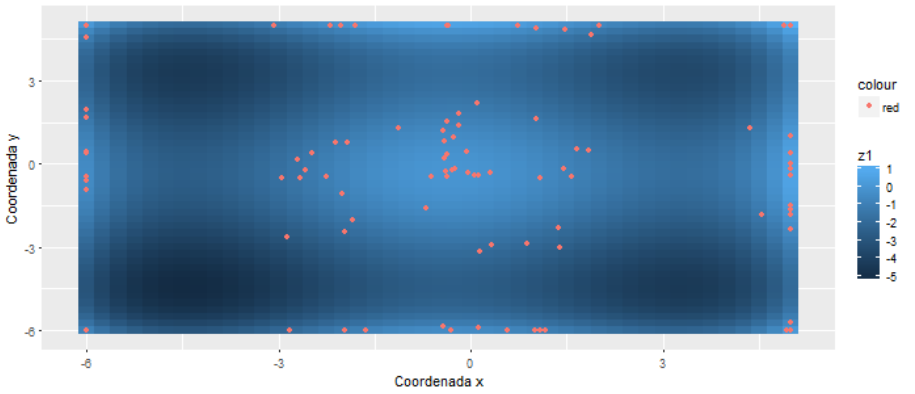

Figura 5: Soluciones con 10 pasos

```r
replica <- function(t) {
library(reshape2)
curr <- runif(2, low, high)
best <- curr
for (tiempo in 1:t) {

  delta_x <- runif(1, 0, step)
  delta_y <- runif(1, 0, step)

  x_l <- max(-low, curr[1]-delta_x)
  x_r <- min(high, curr[1]+delta_x)

  y_b <- max(-low, curr[2]-delta_y)
  y_t <- min(high, curr[2]+delta_y)

  gs <- outer(c(x_l,x_r),c(y_b,y_t),g)
  dimnames(gs)<-list(c(x_l,x_r),c(y_b,y_t))
  gsd<-melt(gs)
  curr<-as.numeric(gsd[which.max(gs),1:2])
  if (g(curr[1],curr[2]) > g(best[1],best[2])) {
    best <- curr
  }
}
return(c(best[1],best[2],g(best[1],best[2])))
}
```

Figura 4: Función réplica para búsqueda local sobre función bivariada

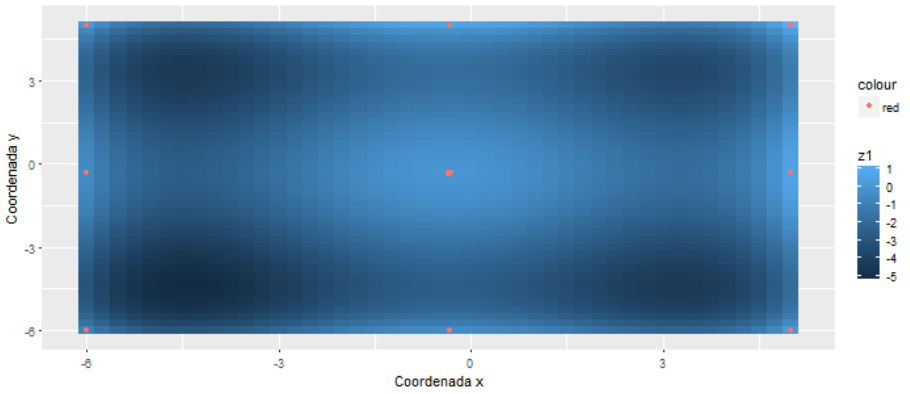

Figura 6: Soluciones con 100 pasos

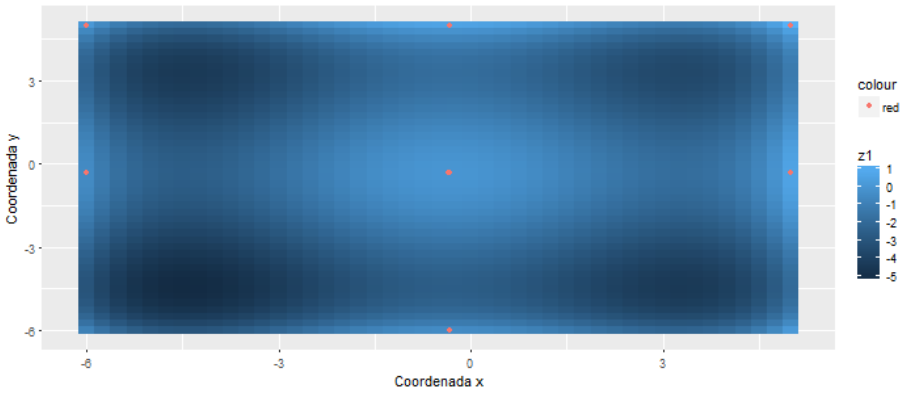

Figura 7: Soluciones con 1000 pasos

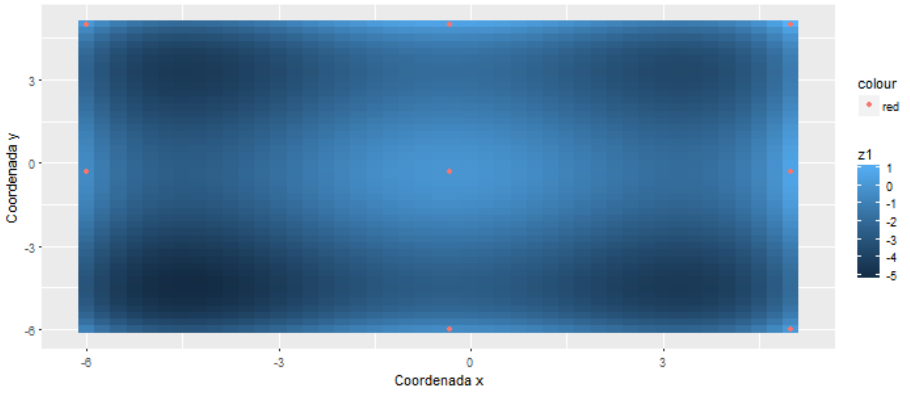

Figura 8: Soluciones con 1000 pasos

Aunque para diez pasos es notable la dispersión de las soluciones, para 100, 1000, y 10000 pasos, casi todas coinciden en los mismos máximos locales. Por supuesto no siempre se logra obtener el máximo global $(5, 5)$ o una solución cercana a éste, puesto que el método es proclive a quedar atrapado en mínimos locales. Sin embargo se nota un ligero adelgazamiento de las manchas rojas, conforme la cantidad de pasos incrementa, lo que indica que la precisión local del método mejora, es decir, las soluciones son menos dispersas entre sí.

2.2. Visualización del funcionamiento del método

Se realizaron varias animaciones *.gif* usando el paquete *magick* de R. Las mismas fueron logradas mediante código facilitado en clase, el cual fue adaptado al caso de una función bivariada en forma tal que la representación gráfica de la trayectoria de soluciones se realiza sobre un plano de curvas de nivel. Para ilustrar la fuerte dependencia que los métodos de búsqueda local poseen respecto a la solución inicial de partida, se muestran diferentes trayectorias que culminan o bien en un punto óptimo local, o bien en el global, dependiendo de la solución de partida. Para consultar las animaciones por favor remítase al repositorio a

las gráficas: *P7_N_.gif*, *P7_N_1_.gif*, *P7_N_2_.gif*, *P7_N_3_.gif*,y *P7_N_4_.gif*. A continuación una discusión del código usado:

```
low <- -6
high <- 5
step <- 0.1
t <- 100

x <- seq(-6, 5, 0.25)
y <-  x
z <- outer(x, y, g)
dimnames(z) <- list(x, y)
d <- melt(z)
names(d) <- c("x1", "y1", "z1")
library(reshape2)
curr <- runif(2,low,high)
best <- curr
digitos <- floor(log(t, 10)) + 1
```

Figura 9: Código para crear objeto que alimentará el mapa de niveles

Como se puede apreciar, primero inicializamos los parámetros del método. Note que en este caso estamos usando una longitud de paso un poco menor (0,1 en lugar de 0,25). Esto se debe a que se pudo constatar que a menor longitud de paso mayor precisión en la aproximación del óptimo local, pese a que esto casi siempre implica mayor número de iteraciones para alcanzar la convergencia, porque en promedio el método tiene que dar más pasos para acercarse del punto inicial al óptimo local. La ganancia en precisión es de esperarse, pues a pasos más cortos más finas las aproximaciones al óptimo local. Note que e trazado del mapa supone primero la creación del objeto d, el cual contiene la información de los puntos que se usan para estimar las curvas de nivel de la función objetivo. El objeto dígitos por su parte, cuenta la cantidad máxima de dígitos que posee el número que indica la cantidad límite de pasos y se usa para facilitar el rotulado de las gráficas que conforman la animación *.gif*. También puede apreciar que es requerida una solución aleatoria actual, para inicializar el método, la cual se genera como una pareja de números aleatorios uniformes con límites de variación iguales a los límites permitidos para cada componente.

```
1
2  for (tiempo in 1:t)
3  {
4    delta_x <- runif(1, 0, step)
5    delta_y <- runif(1, 0, step)
6
7    x_l <- max(low, curr[1] - delta_x)
8    x_r <- min(high, curr[1] + delta_x)
9
10   y_b <- max(low, curr[2] - delta_y)
11   y_t <- min(high, curr[2] + delta_y)
12
13   gs <- outer(c(x_l,x_r),c(y_b,y_t),g)
14   dimnames(gs)<-list(c(x_l,x_r),c(y_b,y_t))
15   gsd<-melt(gs)
16   curr<-as.numeric(gsd[which.max(gs),1:2])
17   if (g(curr[1],curr[2]) > g(best[1],best[2])) {
18    best <- curr
19   }
20   tl <- paste(tiempo, "", sep="")
21   while (nchar(tl) < digitos) {
22   tl <- paste("0", tl, sep="")
23   }
24   salida <- paste("p7_t", tl, ".png", sep="")
25   tiempo <- paste("Paso", tiempo)
26   d0<-data.frame(x=best[1],y=best[2])
27   p<-ggplot()+geom_tile(data=d,aes(x=x1,y=y1,fill=z1))+
28    geom_point(data=d0,aes(x=x,y=y))+xlab("Coordenada_x")+
29    ylab("Coordenada_y")+geom_vline(aes(xintercept = best[1],
          colour="green"))+
30    geom_hline(aes(yintercept = best[2],colour="green"))
31  png(salida, width=500, height=400)
32  print(p)
33  graphics.off()
34 }
```

Figura 10: Ciclo para trazado de trayectorias de soluciones

Para el caso del ciclo *for* note que contamos con dos números aleatorios diferentes para simular el desplazamiento en cada componente de la solución actual. En cada iteración se crean cuatro vecinos en las direcciones de los puntos cardinales de la solución actual. Las funciones *max* y *min* aplicadas sobre las componentes realizan la corrección para garantizar la factibilidad de cada punto. La función *outer* representa una forma sucinta de escribir las evaluaciones de los puntos vecino en la función objetivo. La nueva solución actual será aquella que mayor valor de función objetivo genere, pero sólo sustituirá a la mejor solución si la supera en el valor de la función objetivo. Una vez una iteración es realizada se ubica la nueva mejor solución sobre el mapa de niveles. Esto se hace usando la librería *ggplot2*, la cual ofrece la posibilidad de solapar los objetos geométricos *geom_tile* y *geom_point* en un único gráfico. Estos objetos representan el mapa de niveles y el punto solución respectivamente, y los parámetros dentro de la instrucción *aes* manipulan la serie de datos que se usan para su construcción así como otros atributos estéticos. Finalmente todas las gráficas se envían a salidas *png* con nombre estandarizados para facilitar su transformación en una imagen *.gif* a través del paquete *magick*. Esta parte no se discute acá por motivos de espacio pero puede ser consultada en el código fuente disponible en el repositorio.