Tarea 2 - Simulación Estocástica

Fabián Ramírez

Una versión ejecutable de los códigos los puede encontrar haciendo click en la siguiente GitHub

Problema 1

Implementar el algoritmo gradiente estocástico con:

$$h(x,y) = (x\sin(20y) + y\sin(20x))^2\cosh(\sin(10x)x) + (x\cos(10y) - y\sin(10x))^2\cosh(\cos(20y)y)$$

definida en el ejemplo anterior, y considere los siguientes escenarios:

1. $$\gamma_k = \frac{1}{\log(1+k)}, \quad \lambda_k = \frac{1}{\log(1+k)^{1/10}}$$
2. $$\gamma_k = \frac{1}{100\log(1+k)}, \quad \lambda_k = \frac{1}{100\log(1+k)}$$
3. $$\gamma_k = \frac{1}{1+k}, \quad \lambda_k = \frac{1}{(1+k)^{1/2}}$$
4. $$\gamma_k = \frac{1}{1+k}, \quad \lambda_k = \frac{1}{(1+k)^{1/10}}$$

Problema 2

Considere la función:

$$h(x) = \{\cos(50x) + \sin(20x)\}^2, \quad 0 < x < 1$$

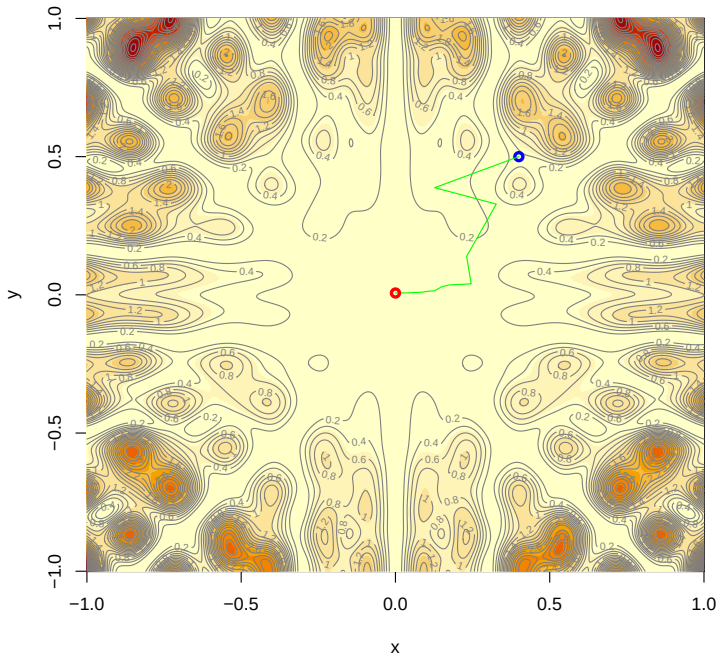
Implementar el algoritmo Simulated Annealing (SA) considerando g como $\mathrm{U}(-\xi,\xi)$, $T_k = \frac{1}{\log(1+k)}$ y $\xi = \sqrt{T_k}$.

Solución Problema 1:

Para empezar procedemos a mostrar una rutina que permite aplicar el método del gradiente descendiente pero no de forma estocástica, esto para familiarizarnos un poco con el método. Obteniendo los siguientes resultados:

$\theta^{(\mathbf{K})}$	$h(\theta^{(\mathbf{K})})$	iter(K)
(1.70284773316561e-14,0.0066352687506327)	2.71946974516173e-28	150

Además presentamos el siguiente mapa de calor del dominio de la h donde vemos como se van aproximando las estimaciones al mínimo teórico.



Donde el color azul denota la estimación inicial que es igual a (0.65,0.8) y el color rojo denota el valor obtenido en la última iteración del algoritmo.

El archivo `grad-determinista.R` contiende los códigos de esta simulación.

Ahora nuestro objetivo sera simular el método de gradiente estocástico utilizando en primer lugar el ejemplo visto en clases. Para ello utilizamos la siguiente secuencia de comandos:

```
h.x = function(x)  (x[1]*sin(20*x[2]) + x[2]*sin(20*x[1]))^2*cosh(sin(10*x[1])*x[1])+␣
 ↪(x[1]*cos(10*x[2]) - x[2]*sin(10*x[1]))^2*cosh(cos(20*x[2])*x[2])
# Tomamos nuestra estimación inicial
theta_0 = c(0.65, 0.8)
# Número de iteraciones
iter = 58
```

```r
# Funciones gamma y lambda
gamma = function(k){
    return(1/(10*log(1+k)))
}
lambda = function(k){
    return(1/k)
}
```

```r
# Creamos el vector z
z=rsphere(iter,2)
```

```r
# La función diferencia
dif = function(h.x,theta,lambda,z,k){
    a=(h.x(theta[k,] + lambda(k)*z[k,]) - h.x(theta[k,]-lambda(k)*z[k,]))/
 ↪(2*lambda(k))*z[k,]
    return(a)
}
```

```r
# creamos la matriz de thetas
thetas = matrix(NA, ncol=2, nrow=iter+1)
colnames(thetas) = c('x','y')
thetas[1, ] = theta_0
```

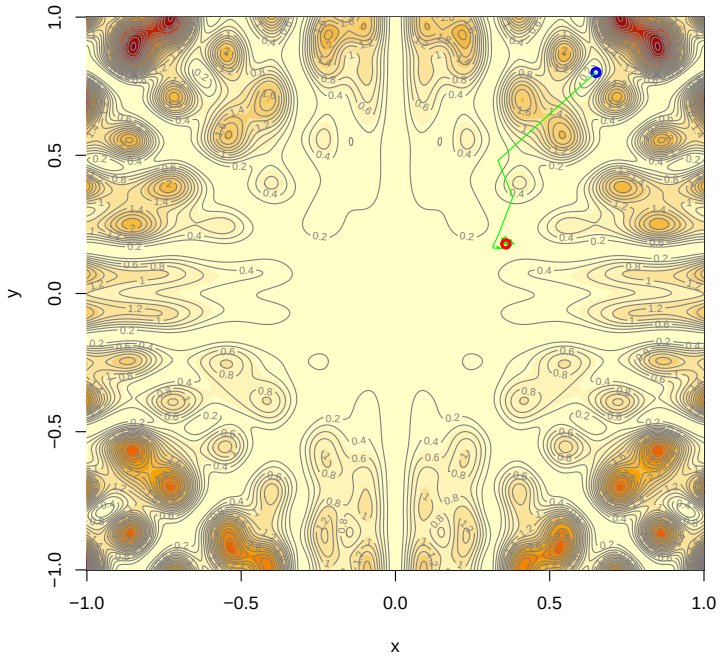
```r
for (k in 1:iter){
    thetas[k+1,]=thetas[k,]-gamma(k)*dif(h.x,thetas,lambda,z,k)
}
```

```r
h = function(x, y)  (x*sin(20*y) + y*sin(20*x))^2*cosh(sin(10*x)*x)+ (x*cos(10*y) -␣
 ↪y*sin(10*x))^2*cosh(cos(20*y)*y)
a = 1
x = seq(from=-a, to=a, length.out=1000)
y = seq(from=-a, to=a, length.out=1000)
z = outer(x, y, h)
image(x=x, y=y, z=z)
contour(x=x, y=y, z=z, add=TRUE, nlevels=40, col=gray(0.5))
points(theta_0[1],theta_0[2],col='blue',lwd = 3)
points(thetas[,1],thetas[,2],col='green',type='l')
points(thetas[iter,1],thetas[iter,2],col='red',lwd = 3)
```

Del cual obtenemos los siguientes resultados:

$\theta^{(\mathbf{K})}$	$h\left(\theta^{(\mathbf{K})}\right)$	iter(K)
`(0.357868061216711,0.179554651200241)`	`0.00033393848751944`	58

Y el siguiente mapa de calor:



Recordemos que en este caso $\gamma_k = \dfrac{1}{10\log(1+k)}$ y $\lambda_k = \dfrac{1}{k}$, es importante notas que el algoritmo va convergiendo a al mínimo que es $(0,0)$ pero no de la forma tan rápida como se muestra en clases, esto se debe a diferencias en las computadoras en la cual se ejecutan los códigos, los valores aleatorios que vienen de la esfera unitaria y errores numéricos.

Procedemos a aplicar el algoritmo para los distintas sucesiones γ_k y α_k. Es importante que para que el algoritmo alcance la convergencia, se debe pedir que:

$$\sum_{k=1}^{\infty} \gamma_k = \infty, \quad \sum_{k=1}^{\infty} \gamma_k^2 < \infty$$

y:

$$\sum_{k=1}^{\infty} \left(\frac{\gamma_k}{\lambda_k}\right)^2 < \infty$$

Teniendo esto en mente notemos que:

1.

$$\gamma_k = \frac{1}{\log(1+k)}, \quad \lambda_k = \frac{1}{\log(1+k)^{1/10}}$$

Notemos que $\sum_{k=1}^{\infty} \gamma_k^2 = \infty$. (ver justificación). Por lo tanto el algoritmo no converge para estas sucesiones.

2.

$$\gamma_k = \frac{1}{100\log(1+k)}, \quad \lambda_k = \frac{1}{100\log(1+k)}$$

Notemos que $\sum_{k=1}^{\infty} \gamma_k^2 = \infty$. (ver justificación). Por lo tanto el algoritmo no converge para estas sucesiones.

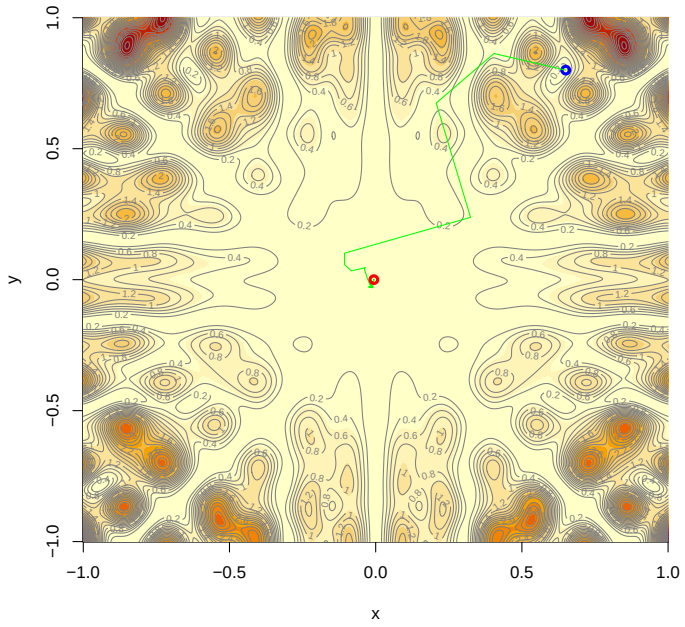
3.

$$\gamma_k = \frac{1}{1+k}, \quad \lambda_k = \frac{1}{(1+k)^{1/2}}$$

Note que estas sucesiones cumplen con todas las condiciones para la convergencia. Aplicando el algoritmo obtenemos los siguientes resultados:

$\theta^{(\mathbf{K})}$	$h(\theta^{(\mathbf{K})})$	iter(K)
`(-0.00626976748967397,0.00012134806211025)`	`3.921556708316e-05`	`100`

Y el siguiente mapa de calor:



Donde se puede ver claramente que en cada paso nos acercamos mas los puntos tales que se alcance el mínimo global.

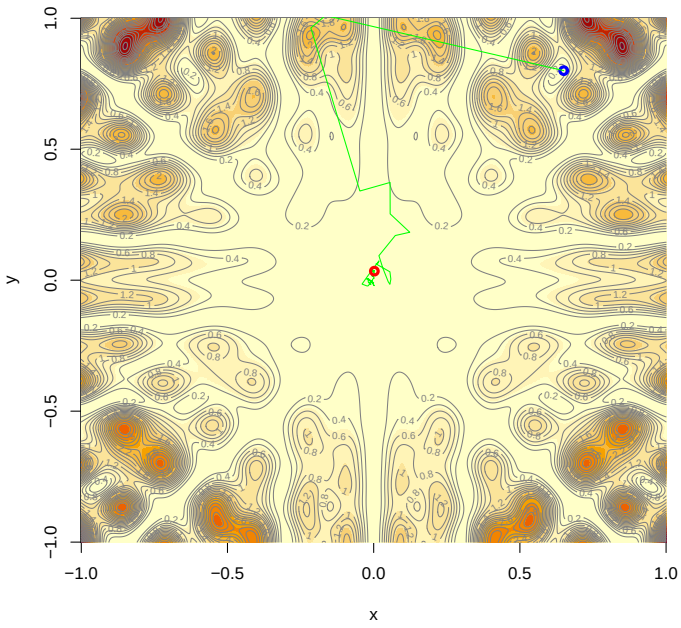
4.

$$\gamma_k = \frac{1}{1+k}, \quad \lambda_k = \frac{1}{(1+k)^{1/10}}$$

Note que estas sucesiones cumplen con todas las condiciones para la convergencia. Aplicando el algoritmo obtenemos los siguientes resultados:

$\theta^{(\mathbf{K})}$	$h(\theta^{(\mathbf{K})})$	iter(K)
`(0.0022537307329987,0.0345189774141111)`	`1.07479467440969e-05`	`100`

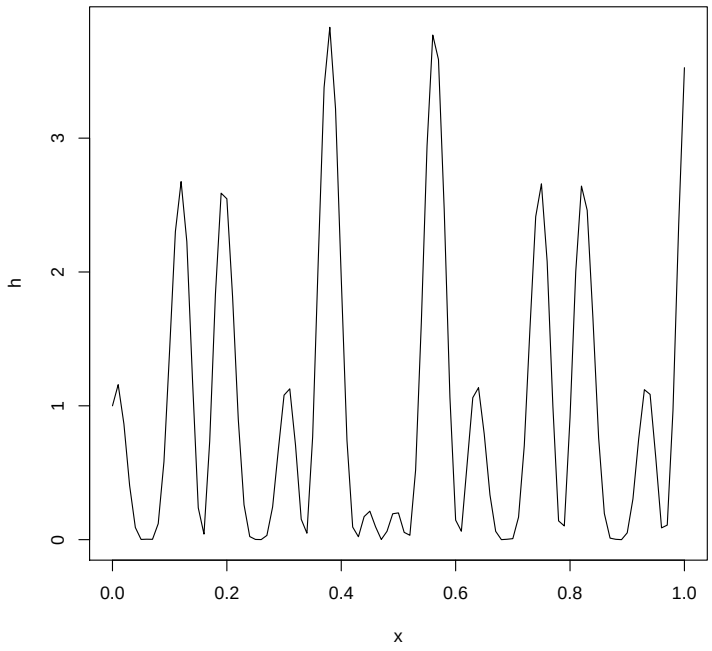
Y el siguiente mapa de calor:



Donde se puede ver claramente que en cada paso nos acercamos mas los puntos tales que se alcance el mínimo global.

Solución Problema 2:

Para empezar lo primero que haremos será graficar la función h:



Notemos que el máximo es aproximadamente $3{,}7$ por tanto idealmente este algoritmo debería aproximarse a ese valor. Procedemos a mostrar los códigos utilizados para implementar el algoritmo.

```
h = function(x) (cos(50*x) + sin(20*x))^2
```

```
# Dominio de la h
li = 0 #limite inferior
ls = 1 #limite superior
```

```
T = function(k){
    return(1/(log(k)+1))
}
```

```
dh = function(h,k,deltas,thetas){
    return(h(thetas[k]+deltas[k])-h(thetas[k]))
}
```

```
theta_0 = 0.5
```

```
iter=5000
```

```
#creamos un vector de semillas
semilla = seq(1,iter+2,1)
```

```
thetas = matrix(NA, ncol=1, nrow=iter+1)
colnames(thetas) = c('x')
thetas[1, ] = theta_0
```

```
# construimos los vectores delta:
deltas = matrix(NA, ncol=1, nrow=iter+1)
for (k in 1:(iter+2)){
    xi =sqrt(T(k))
    set.seed(semilla[k])
    deltas[k-1,] = runif(1,-xi,xi)
}
```

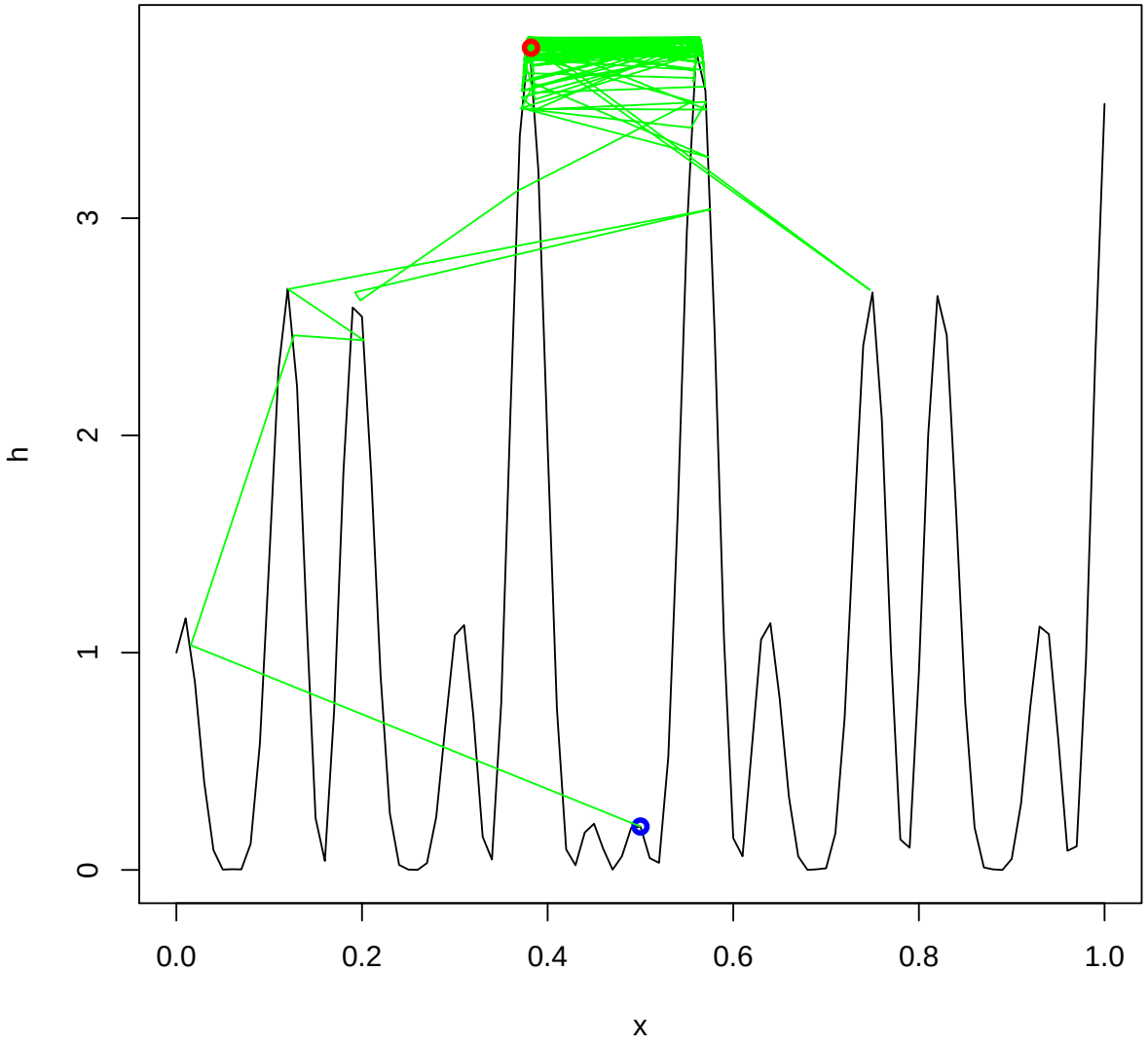
```
for (k in 1:iter){
    p = min(
        exp(
        dh(h,k,deltas,thetas)/T(k)
        ), 1
    )
set.seed(semilla[k])
p0 = runif(1,0,1)
if (p>=p0){
        a = thetas[k,] + deltas[k,]
        if (a >li & a <ls){
            thetas[k+1,] = a
        } else {
            thetas[k+1,] = thetas[k,]
        }
    } else {
        thetas[k+1,] = thetas[k,]
    }
}
```

```
plot(h,xlim=c(0,1))
points(theta_0,h(theta_0),col='blue',lwd = 3)
points(thetas,h(thetas),col='green',type='l')
points(thetas[iter+1],h(thetas[iter+1]),col='red',lwd = 3)
```

Obteniendo los siguientes resultados:

$\theta^{(\mathbf{K})}$	$h\left(\theta^{(\mathbf{K})}\right)$	iter(K)
0.382058861428591	3.78501435240289	5000

Y el siguiente gráfico:



Note en el gráfico que el algoritmo cumple con lo que promete, que es escapar de los extremos locales.