

# Boltzmann Univariate Marginal Distribution Algorithm

Presenta: Sergio Ivvan Valdez Peña

Trabajo conjunto con: Arturo Hernández y Salvador Botello



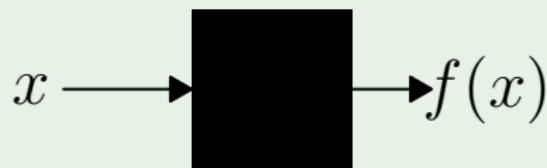
Centro de Investigación en Matemáticas, CIMAT A.C.

Junio 2015

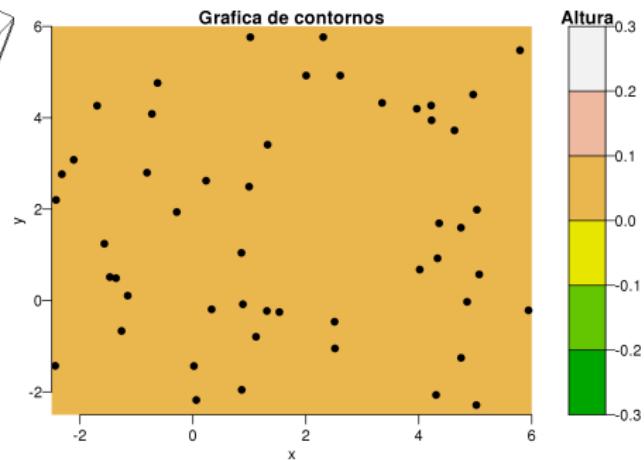
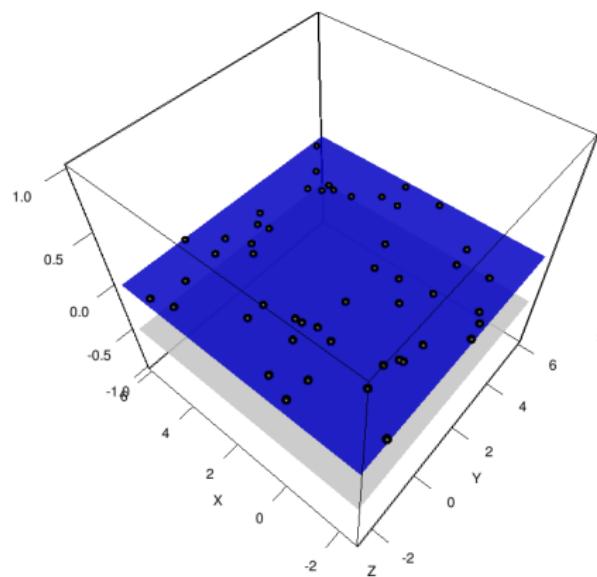
Los algoritmos de estimación de distribución son algoritmos de caja negra (solo utilizan la evaluación de la función objetivo). Para maximizar o minimizar una función. En este caso en particular veremos una algoritmo de estimación de distribución en dominios continuous en los reales.

## Problema

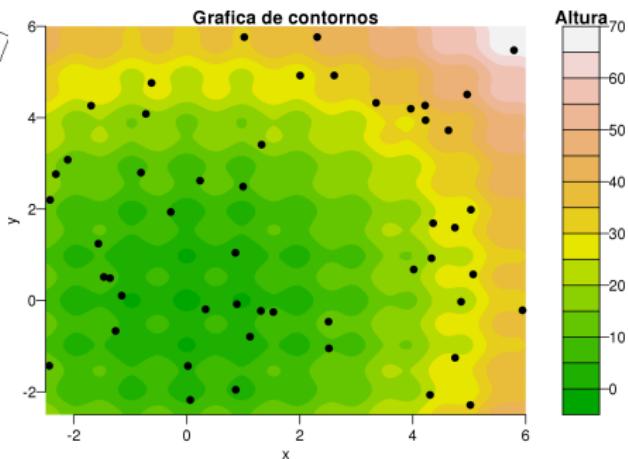
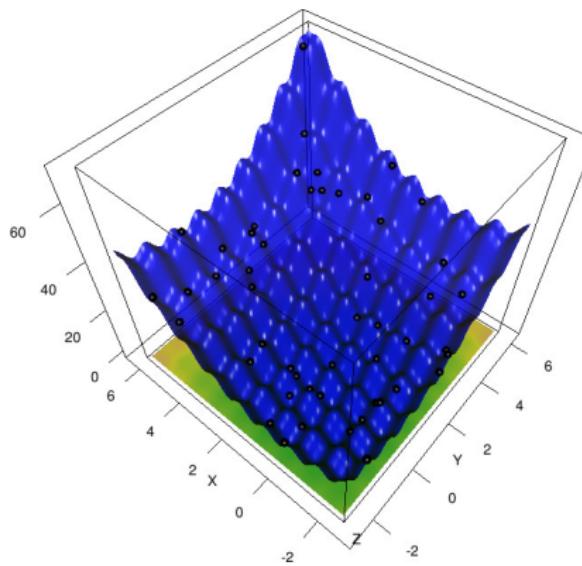
$$\min/\max \quad f(x), x \in \mathbf{R}^n$$



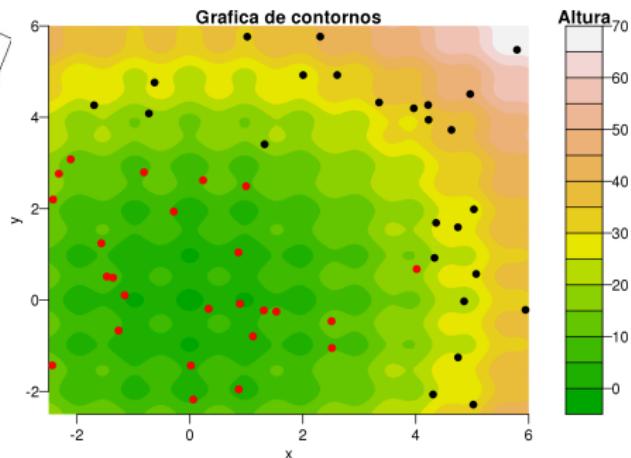
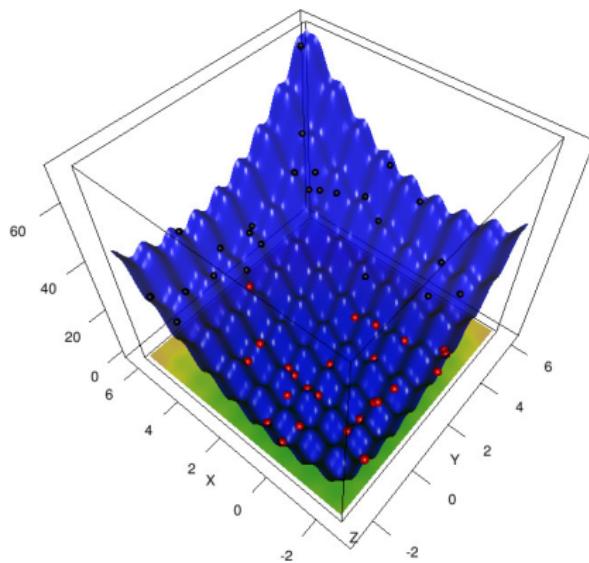
# Ejemplo de EDA. Inicialización



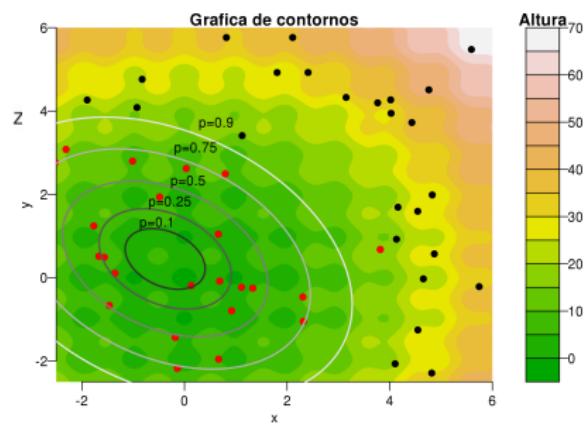
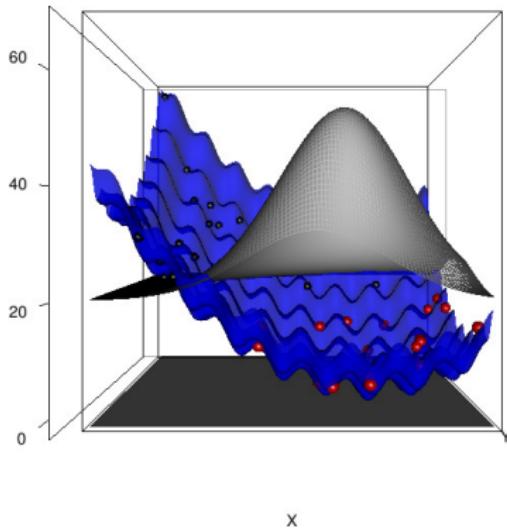
# Ejemplo de EDA. Evaluación



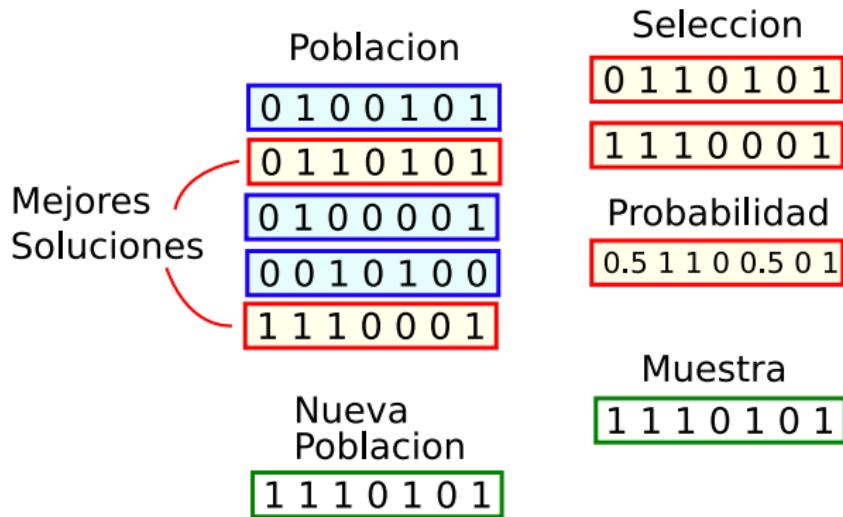
# Ejemplo de EDA. Selección (suponiendo minimización)



# Ejemplo de EDA. Estimación

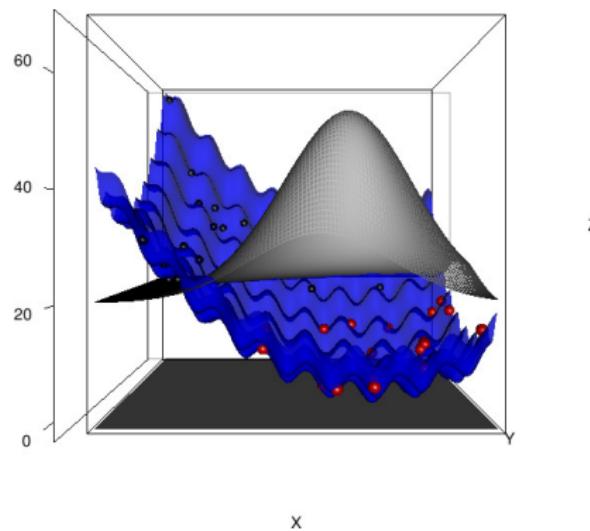


Los EDAs inicialmente fueron propuestos para dominios binarios por Mühlenbein en 1996. (Aunque ya había algoritmos similares) La idea principal es que el conjunto seleccionado contiene con mayor probabilidad los valores de las variables del óptimo.



## EDAs continuos

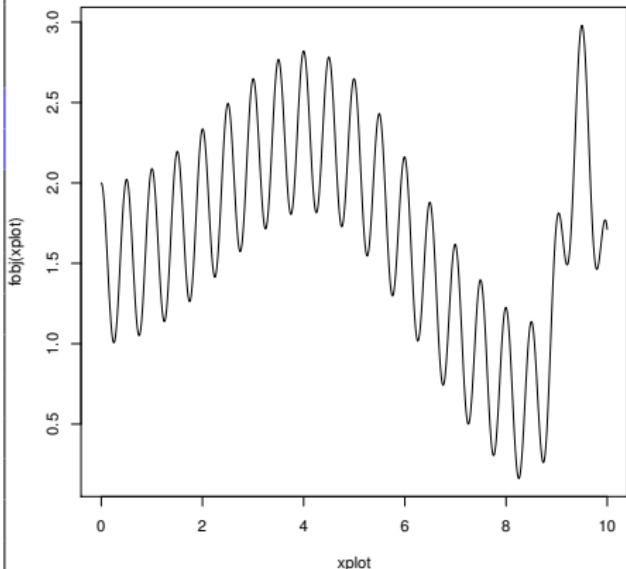
En caso de los reales, lo que encontraremos serán regiones más probables. Alrededor del 2002 Larrañaga et al. presentaron un algoritmo llamado  $UMDA_g$  (Univariate Marginal Distribution Algorithm) Gaussiano. La idea es simple, definir una distribución Gaussiana a partir del conjunto seleccionado.



# *UMDA<sub>g</sub>*, problema

Supongamos que queremos maximizar una función:

```
1 fobj<-function (x)
2 {
3     f=0.5*cos (4.0*pi*x)+1.5*
4         exp(0.12*x*sin (0.5*x))+2*
5         exp(-(x-9.5)^2/0.25)
6     return (f)
7 }
8
9 lim=c(0,10)
10 xplot=seq(lim [1],lim [2],
11             length.out=1000)
12
13 plot(xplot,fobj(xplot),type='l')
```

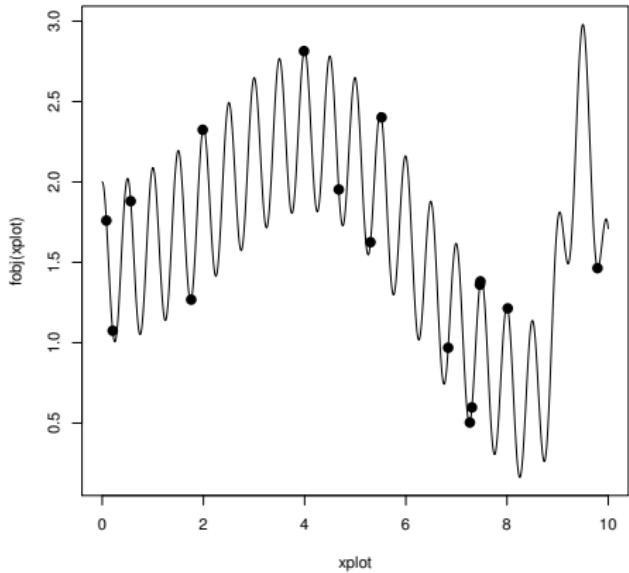


Listing 1 : Función Objetivo en R

# *UMDA<sub>g</sub>*, inicialización

Generamos una población inicial en el dominio de búsqueda

```
2 evalua<-function(func ,X)
4 {
5   F=NULL
6   for ( i in 1:length(X))
7     F=c(F, func(X[ i ]))
8   return(F)
9 }
10 npop=16
11 X=runif(npop ,lim [1] ,lim [2])
12 F=evalua(fobj ,X, pch=16)
13 points(X,F)
```

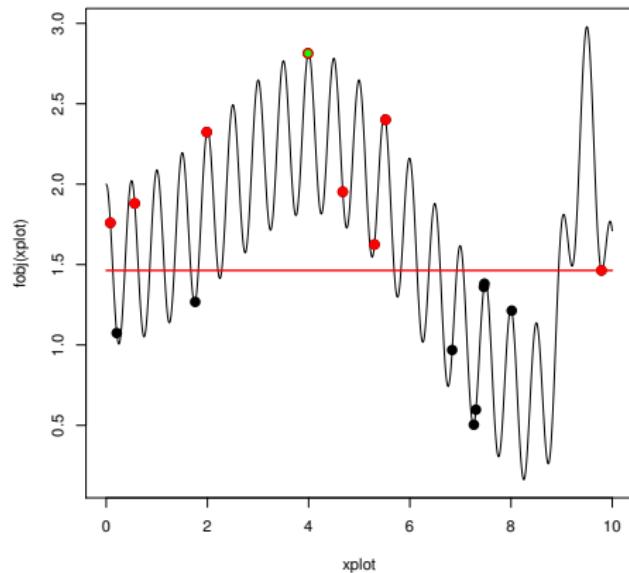


Listing 2 : Evaluación, código R.

## *UMDA<sub>g</sub>*, selección

Seleccionamos los individuos con mejor (**mayor para maximización**) valor objetivo (aptitud).

```
1 ls=sort(F, index.return=TRUE  
         , decreasing=TRUE)$ix  
2 umbral=F[ls[npop/2]]  
3 lines(lim, rep(umbral,2), col  
       ="red", lw=2)  
4 S=X[ls[1:(npop/2)]]  
5 Fs=F[ls[1:(npop/2)]]  
6 fbest=Fs[1]  
7 xbest=S[1]  
8 points(S, Fs, pch=16, cex=1.5,  
          col="red")  
9
```

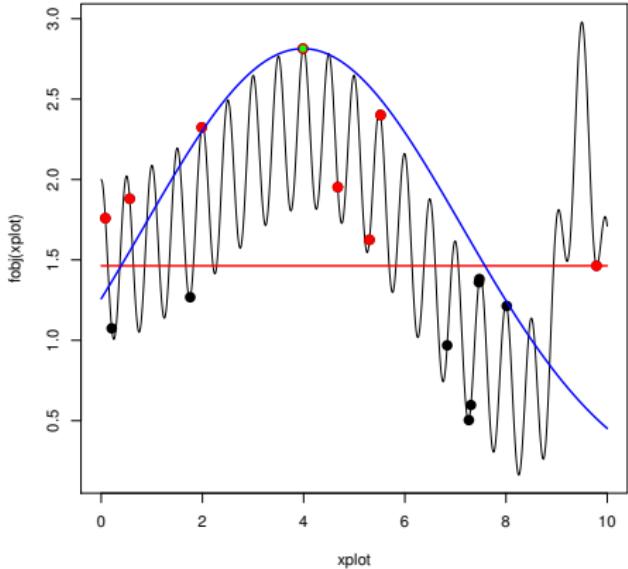


Listing 3 : Selección, código R.

# *UMDA<sub>g</sub>* estimación

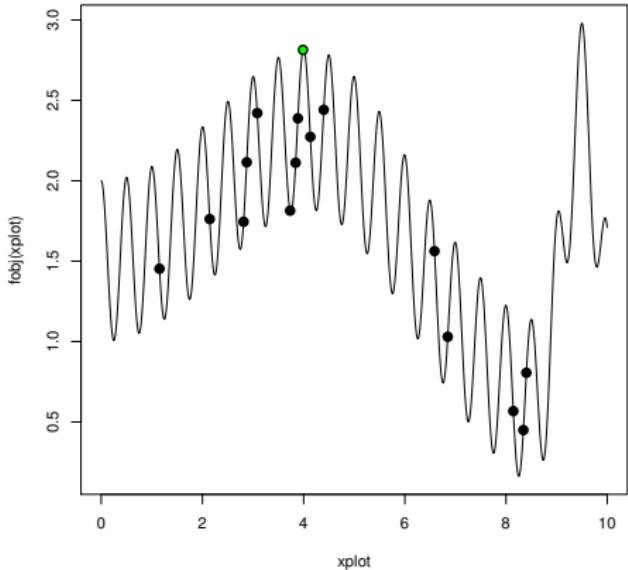
```
2 ls=sort(F, index.return=TRUE  
         , decreasing=TRUE)$ix  
umbral=F[ls[npop/2]]  
4 lines(lim, rep(umbral, 2), col  
       ="red", lw=2)  
S=X[ls[1:(npop/2)]]  
Fs=F[ls[1:(npop/2)]]  
6 points(S, Fs, pch=16, cex=1.5,  
         col="red")
```

Listing 4 : Estimación, código R.



# *UMDA<sub>g</sub>* muestreo

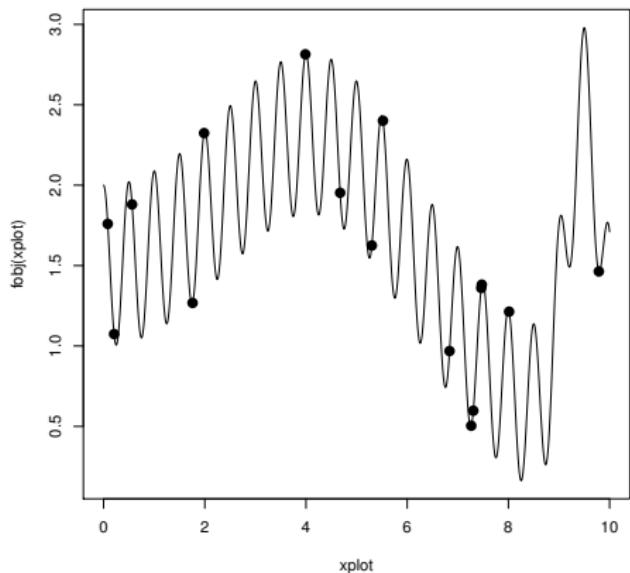
```
2 X=rnorm(npop,mu, sd)
3 X[X<lim[1]]=lim[1]
4 X[X>lim[2]]=lim[2]
5 F=evalua(fobj,X)
6 X[1]=xbest
7 F[1]=xbest
8 plot(xplot,fobj(xplot),type='l')
9 points(X,F,pch=16,cex=1.5)
10 points(xbest,fbest,pch=16,cex=1,col="green")
```



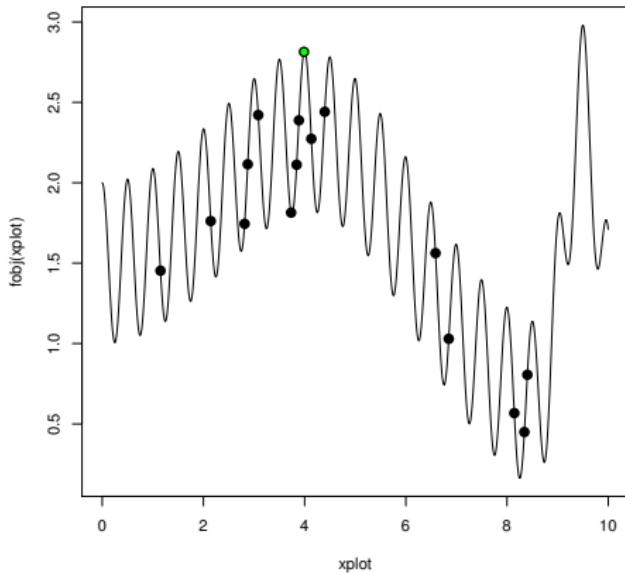
Listing 5 : Genera nueva población,  
código R.

# $UMDA_g$ mejora entre generaciones

Generación inicial



Segunda generación



# $UMDA_g$ pasos

Inicialización  
Evaluación

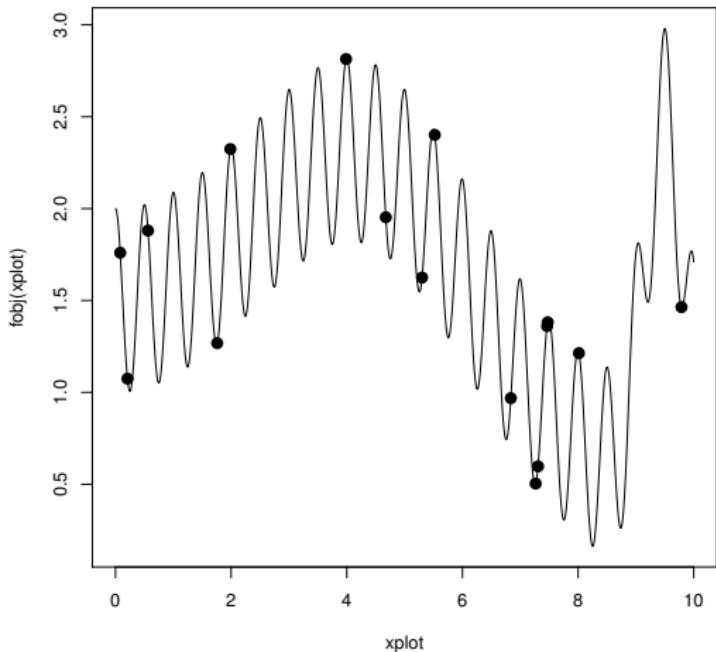
Selección

Estimación

Muestreo

Evaluación

Reemplazo

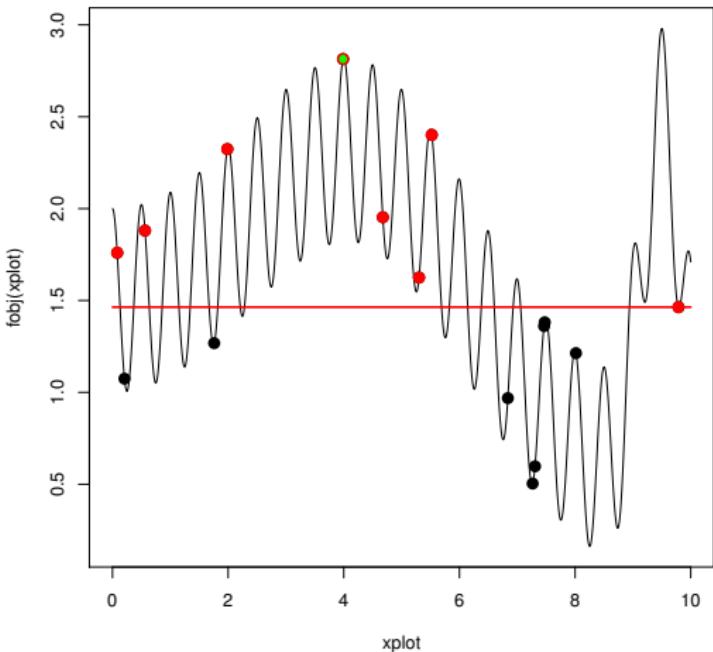


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



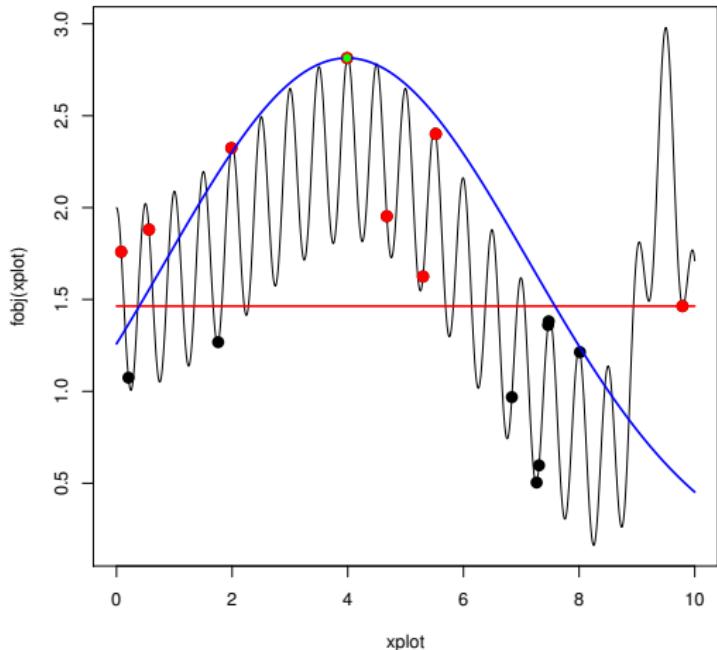
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

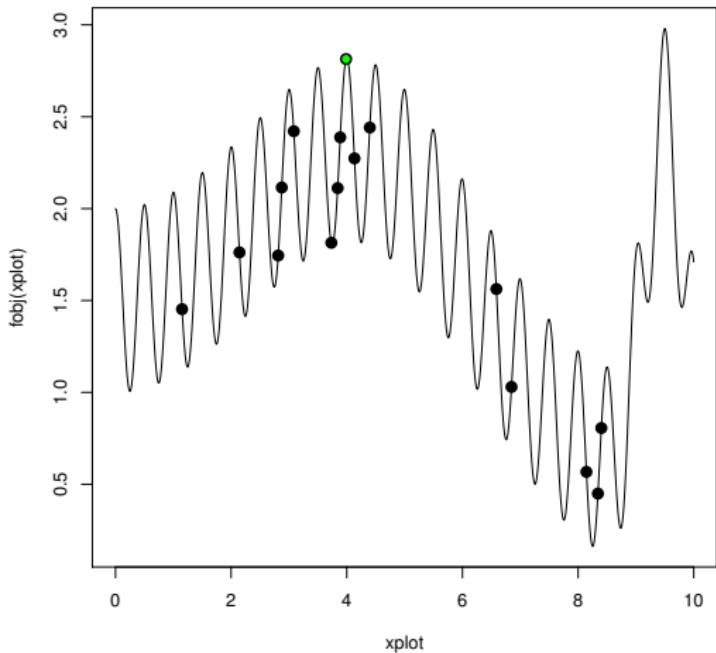
Selección

Estimación

Muestreo

Evaluación

Reemplazo

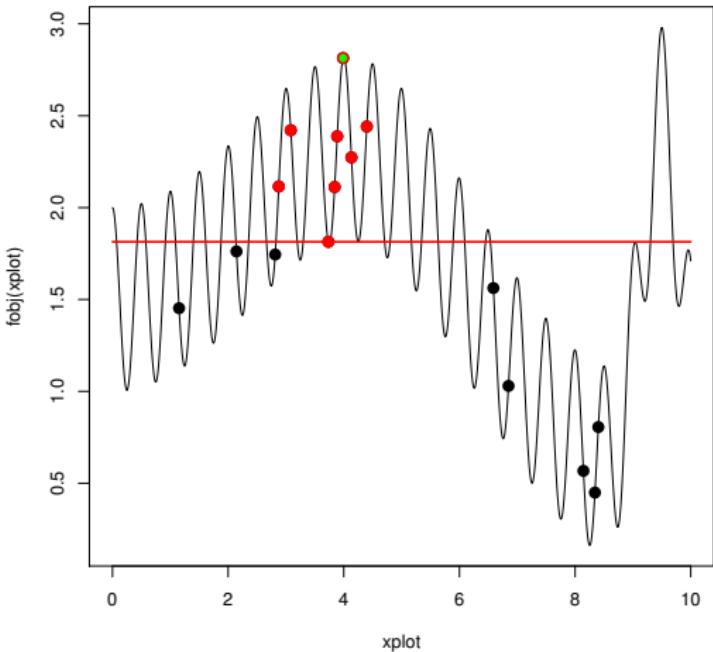


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



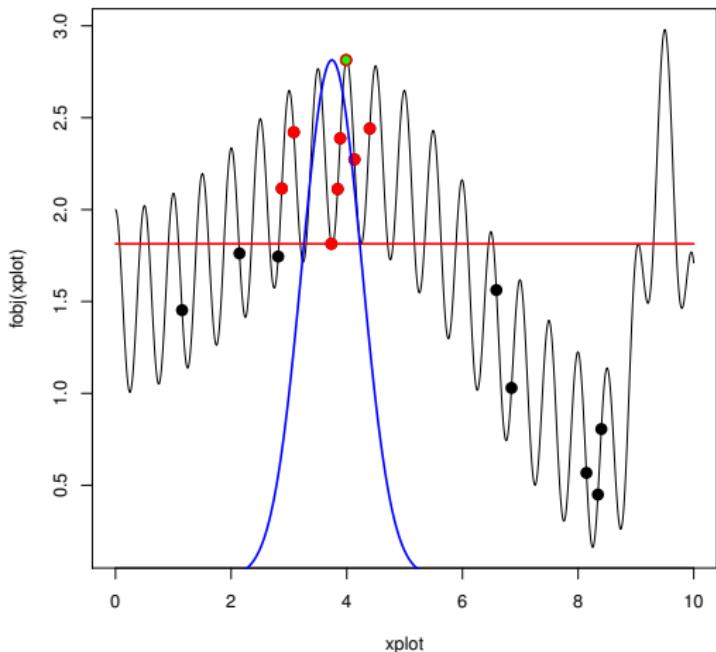
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

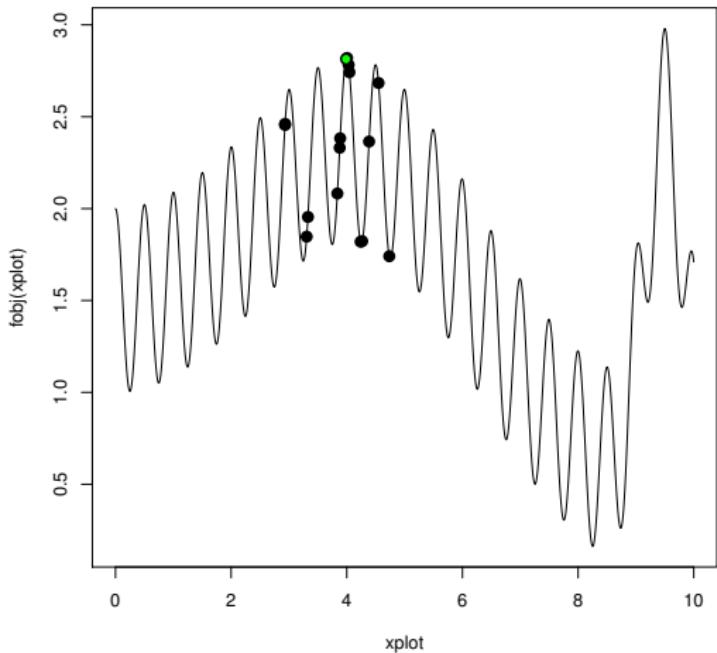
Selección

Estimación

Muestreo

Evaluación

Reemplazo

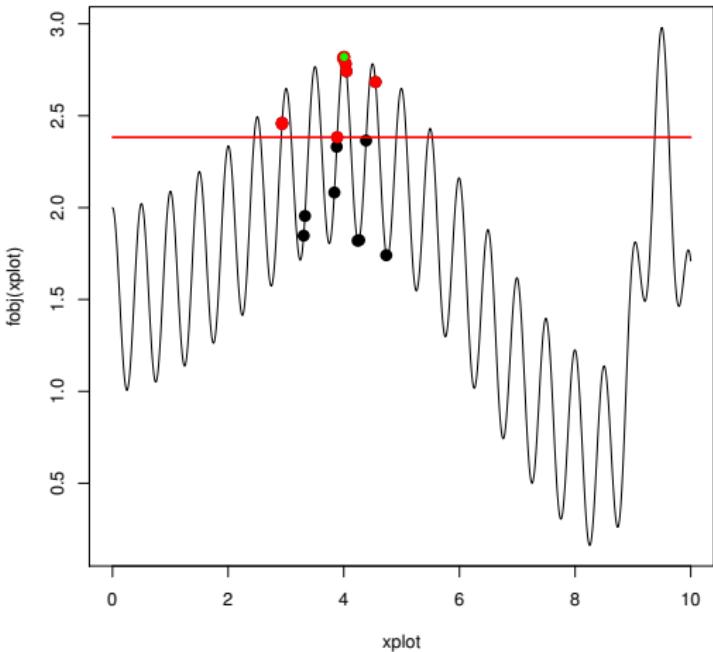


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



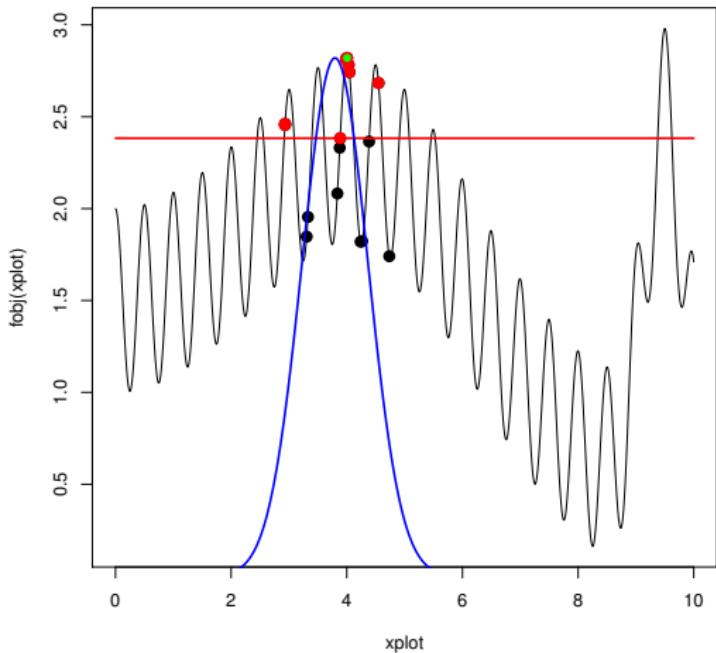
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



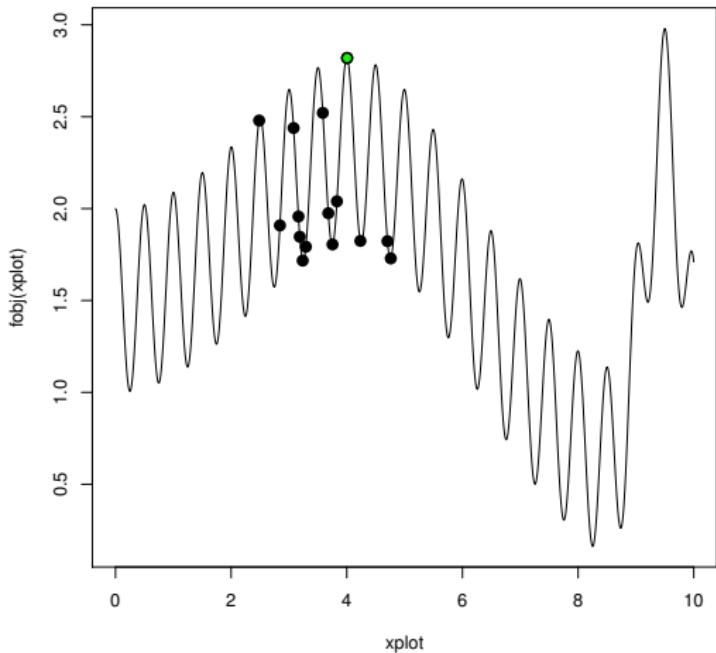
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo

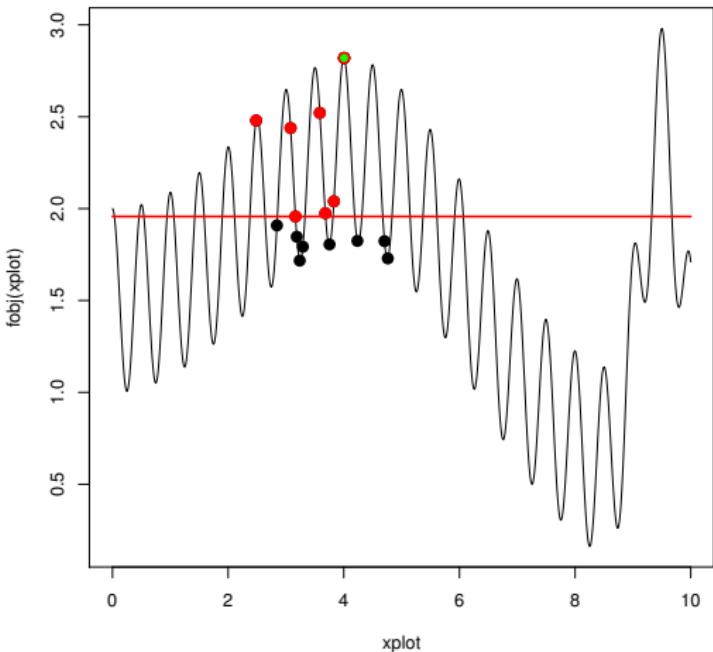


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



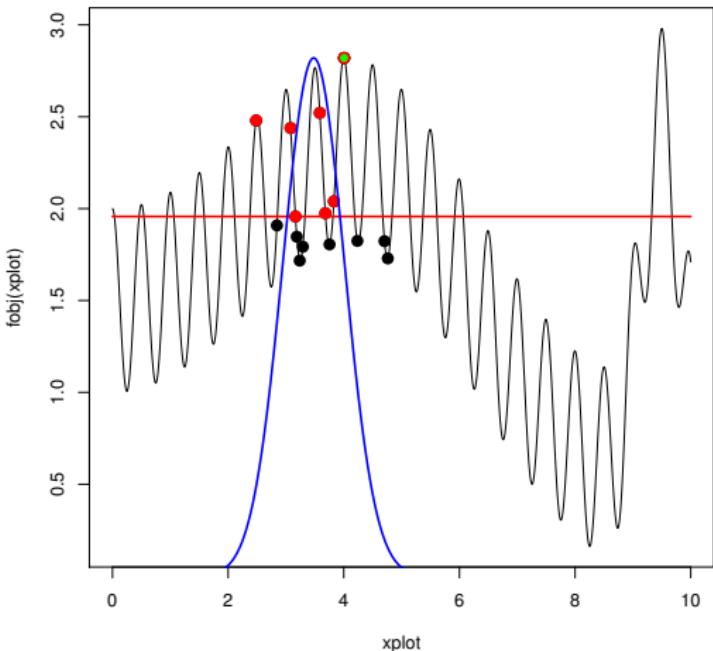
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

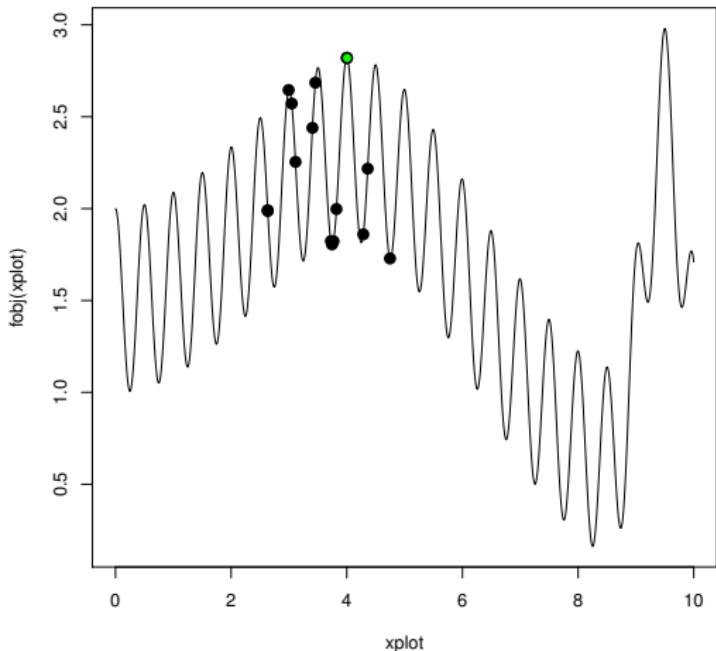
Selección

Estimación

Muestreo

Evaluación

Reemplazo

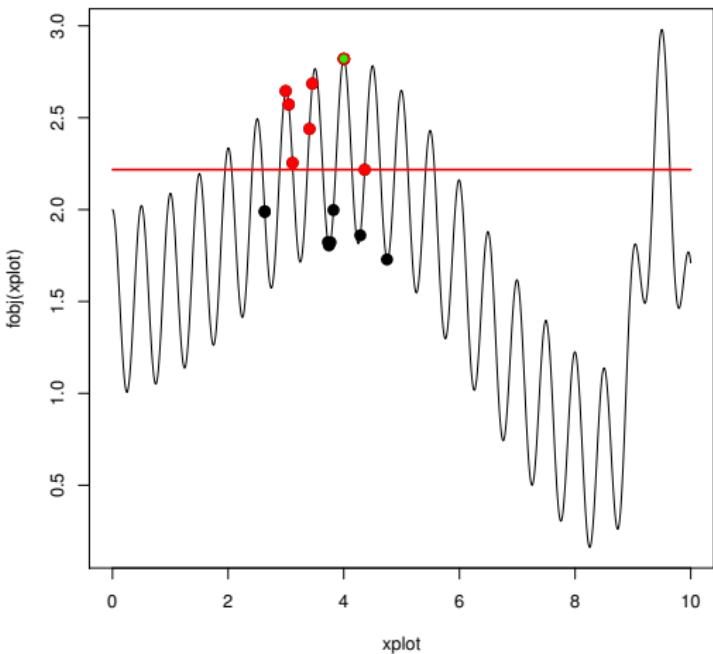


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

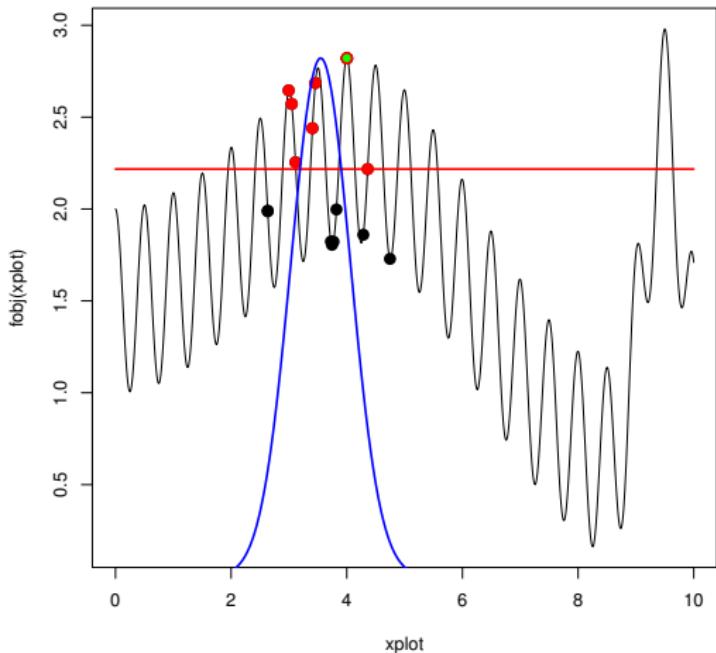
Selección

Estimación

Muestreo

Evaluación

Reemplazo



# $UMDA_g$ pasos

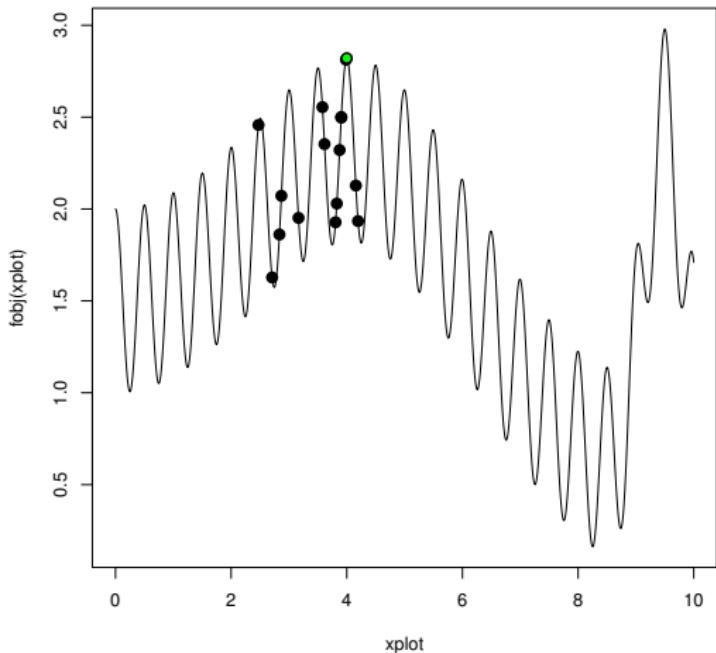
Inicialización

Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo

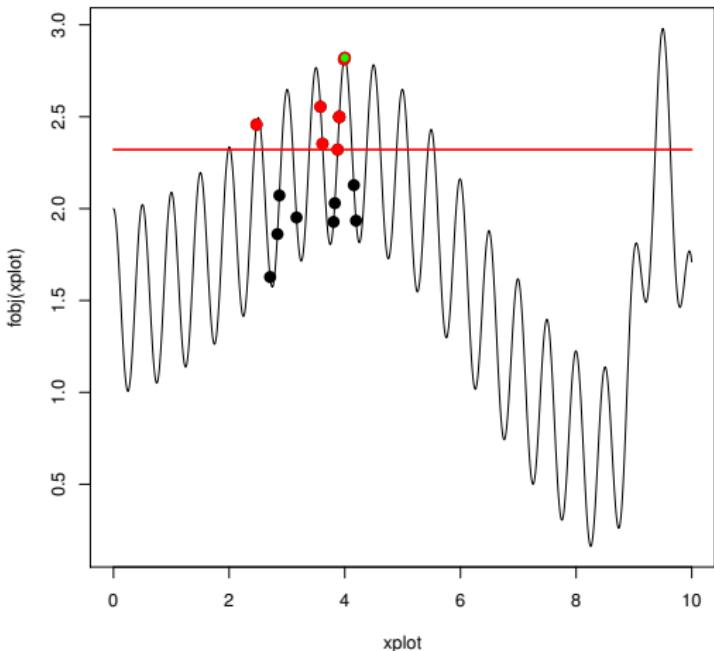


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



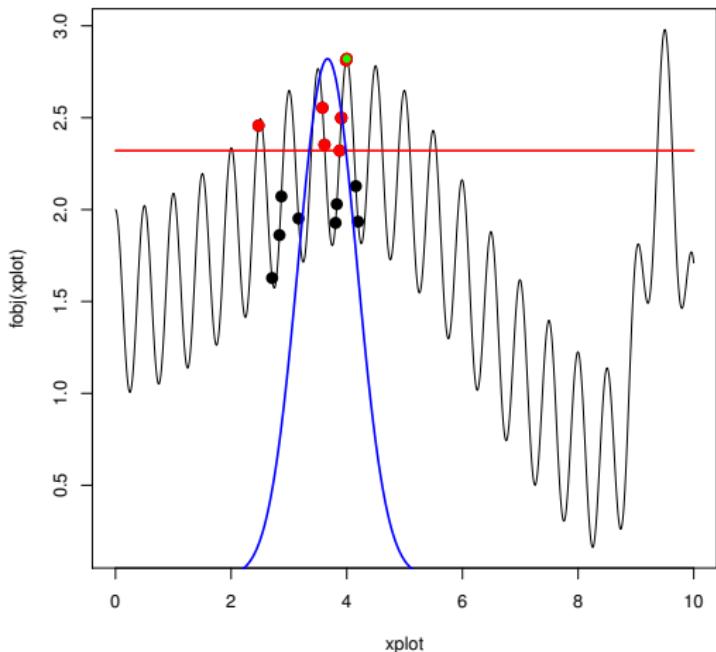
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



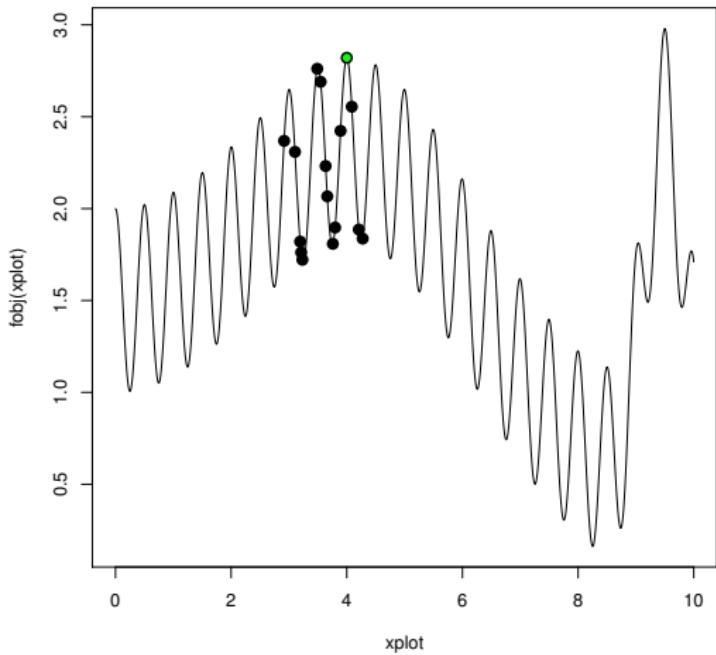
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo

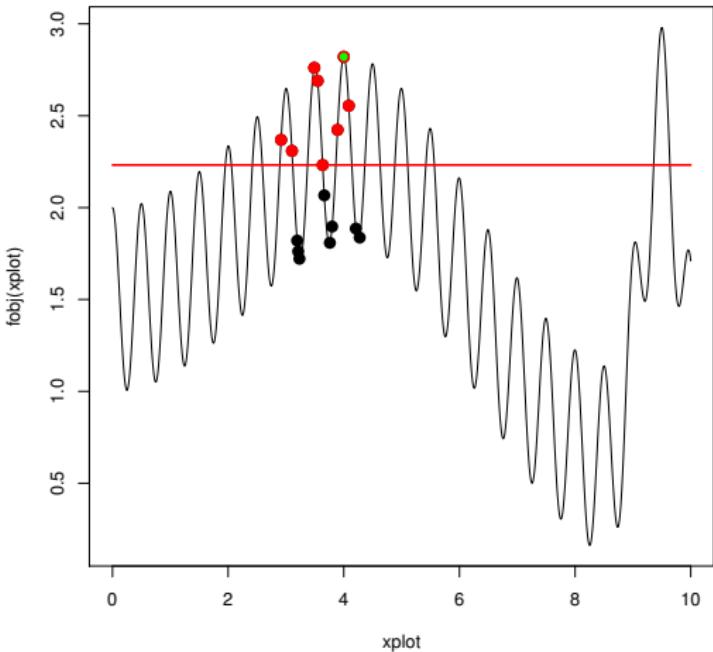


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



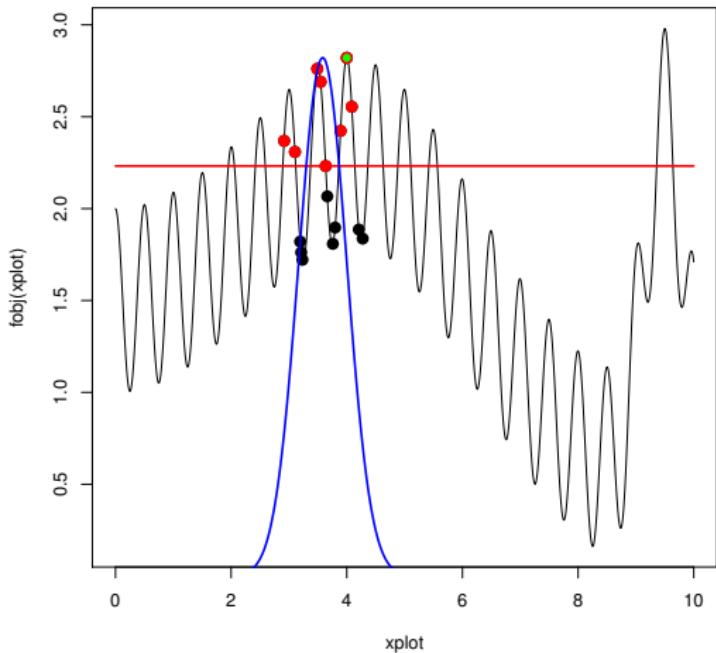
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

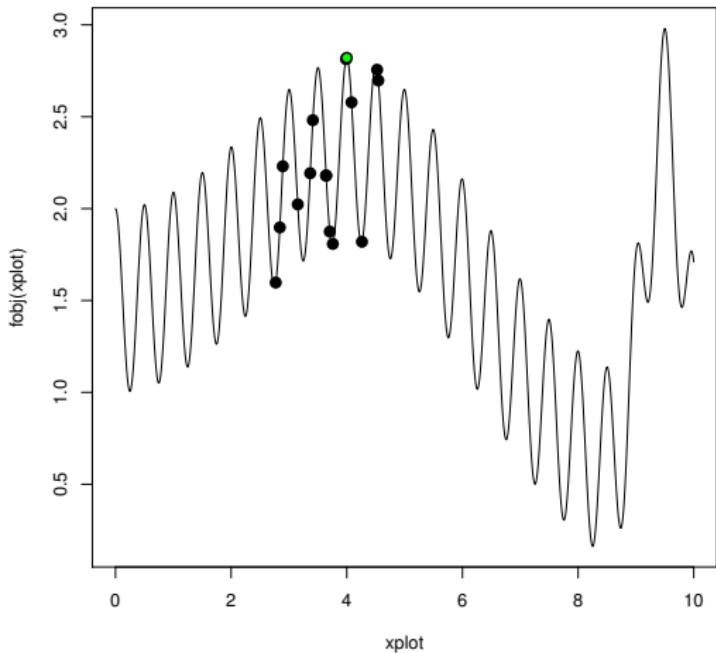
Selección

Estimación

Muestreo

Evaluación

Reemplazo

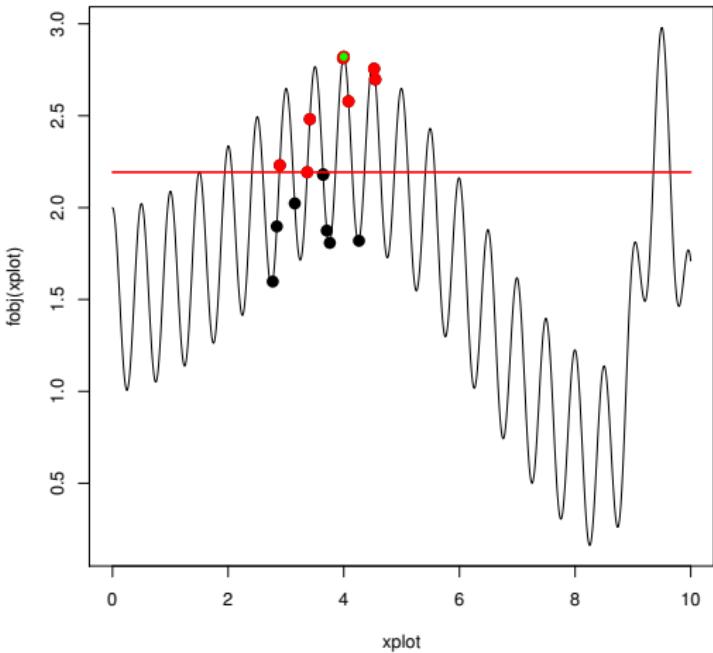


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



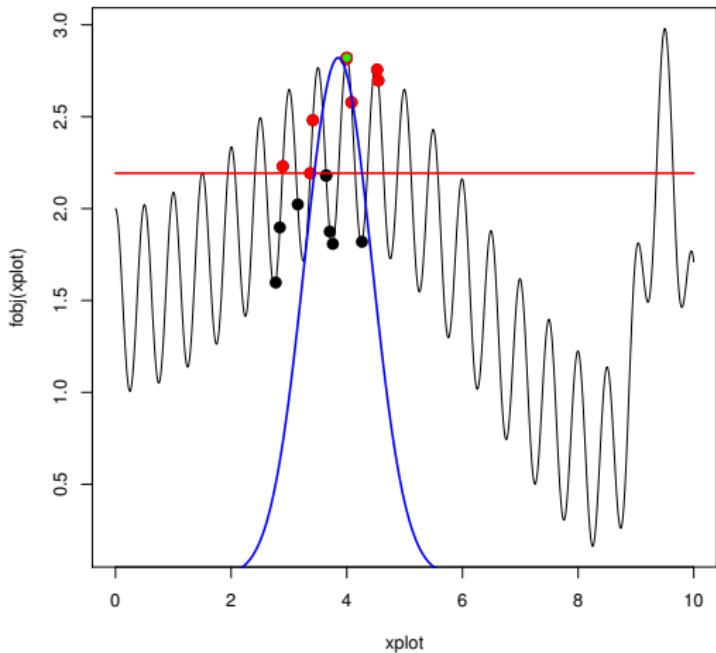
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

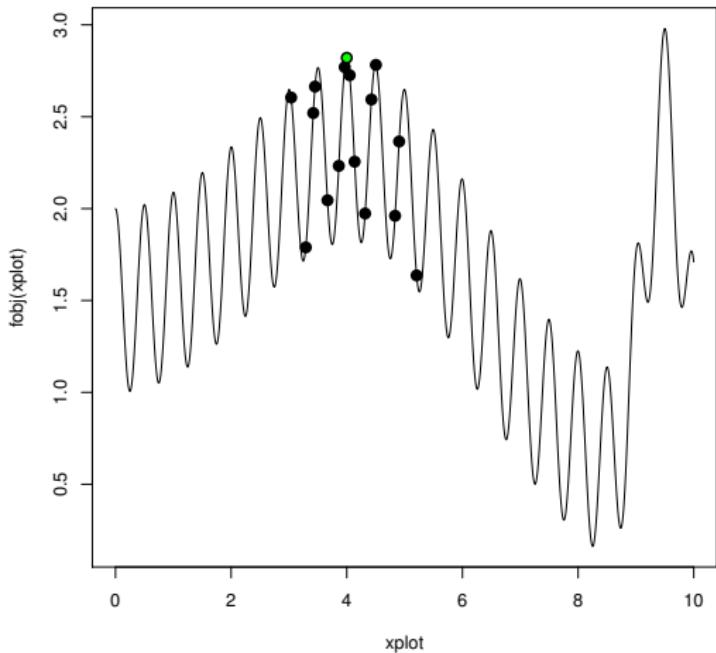
Selección

Estimación

Muestreo

Evaluación

Reemplazo

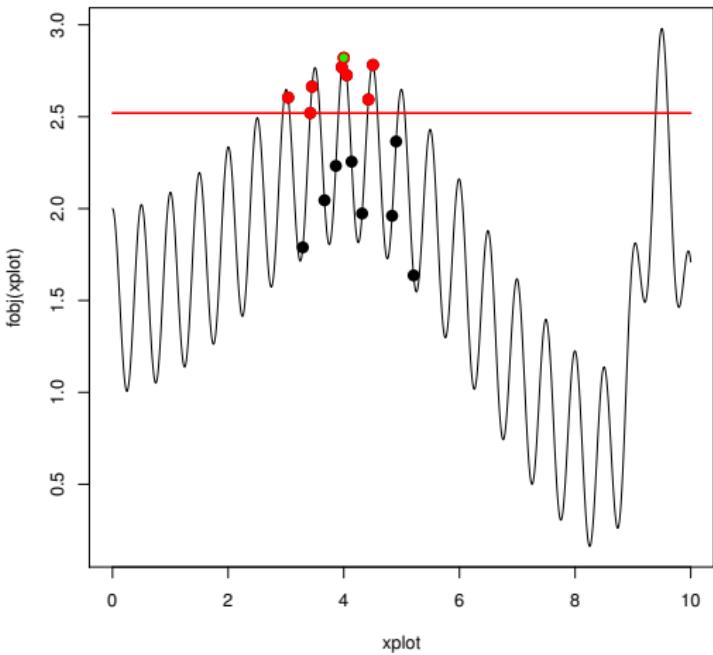


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



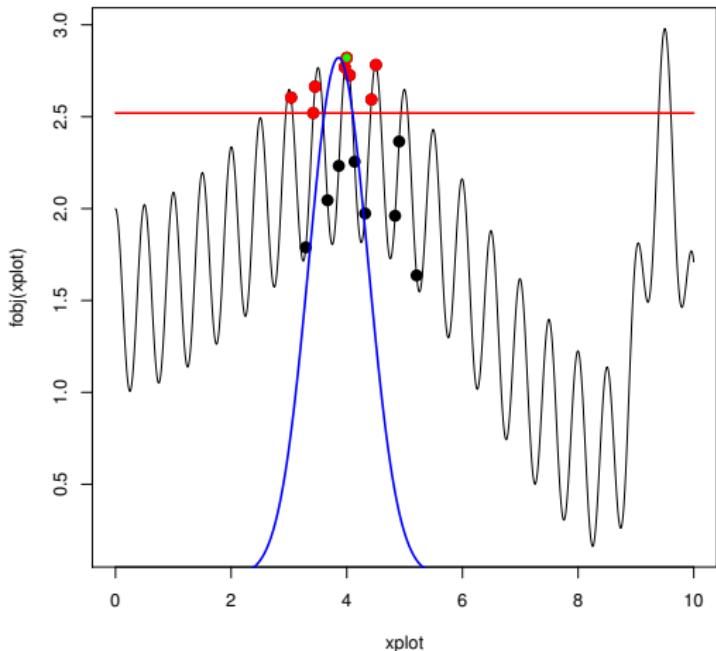
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

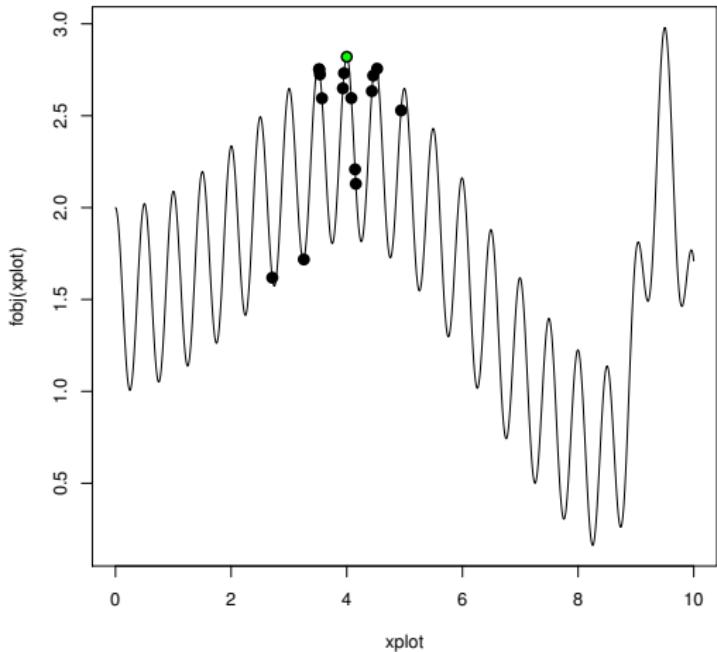
Selección

Estimación

Muestreo

Evaluación

Reemplazo

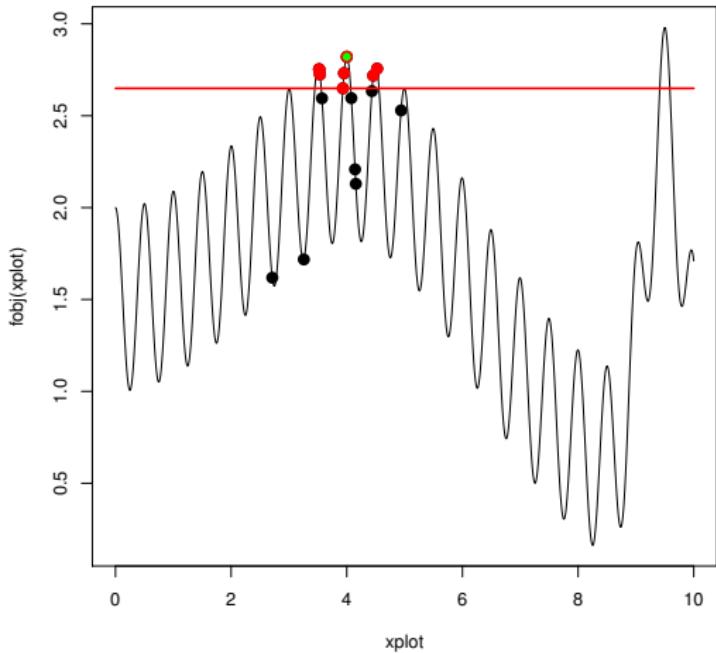


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

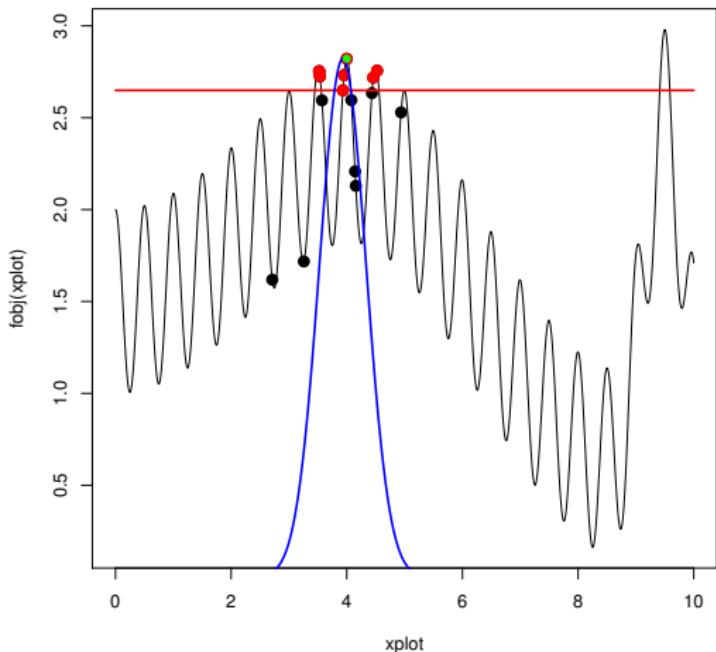
Selección

Estimación

Muestreo

Evaluación

Reemplazo



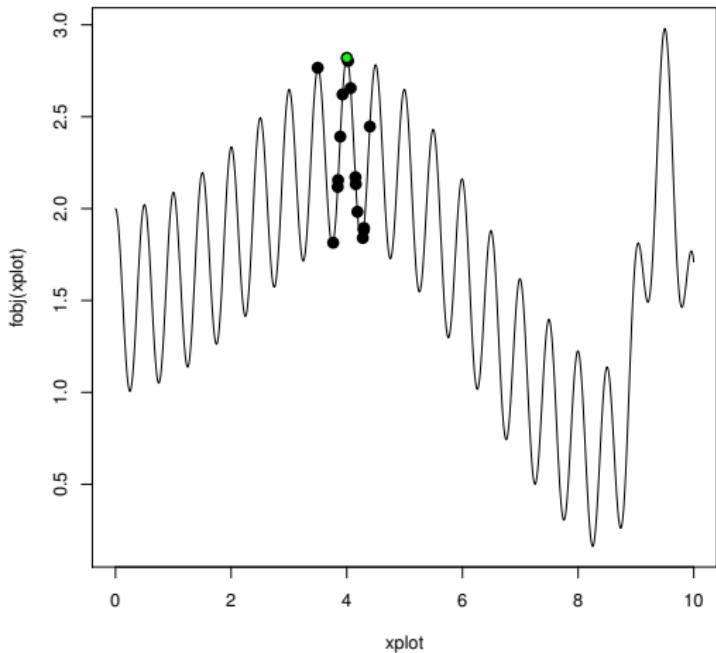
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo

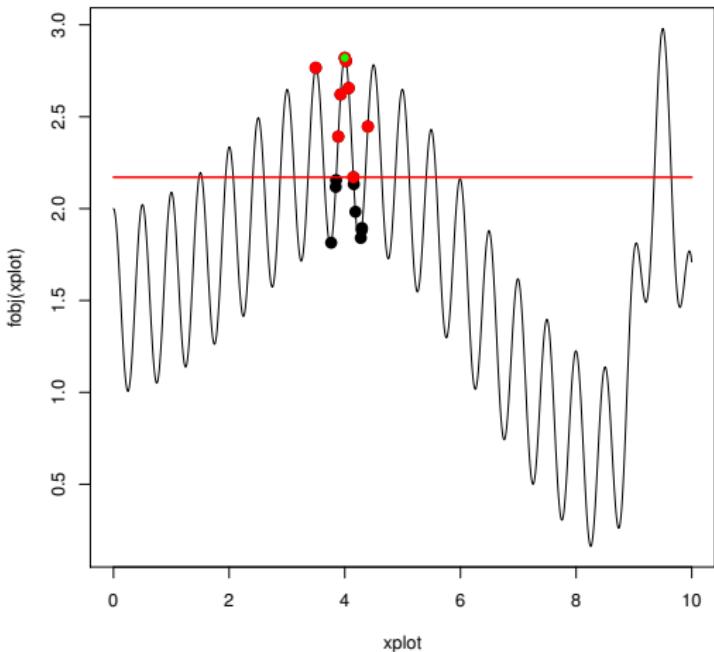


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

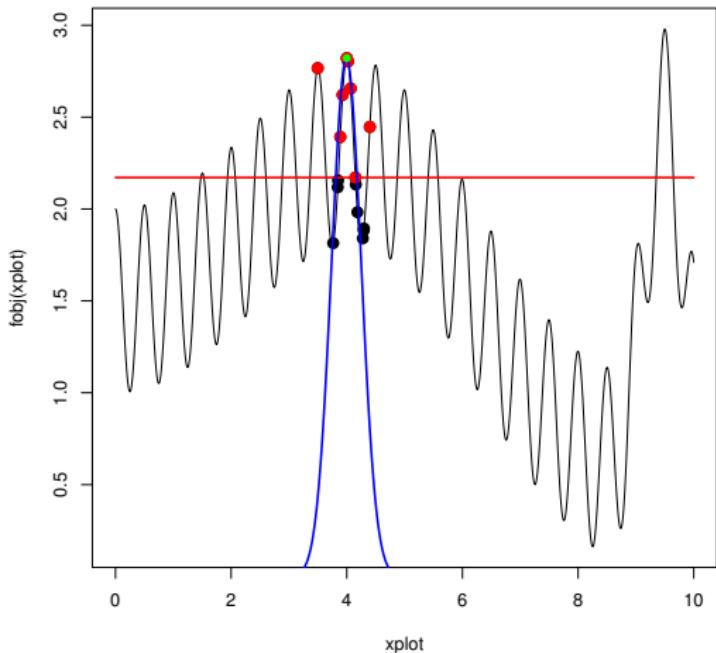
Selección

Estimación

Muestreo

Evaluación

Reemplazo



# $UMDA_g$ pasos

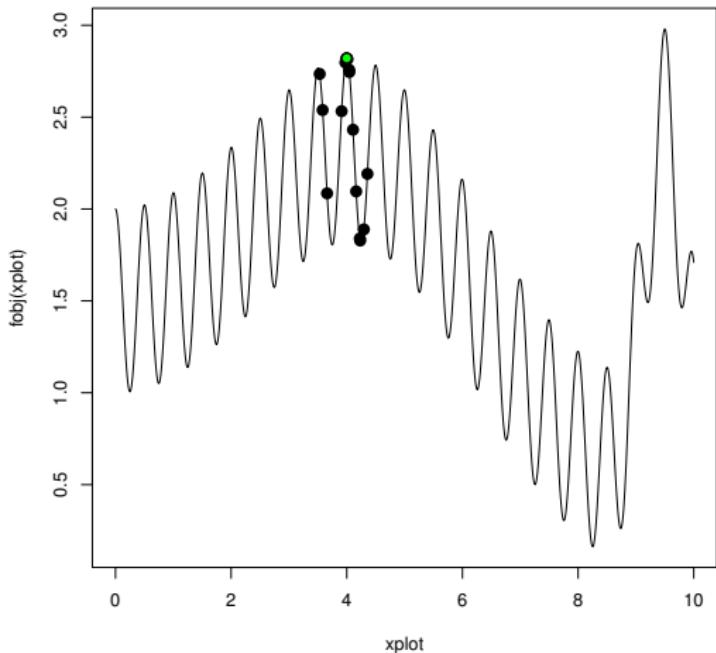
Inicialización

Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



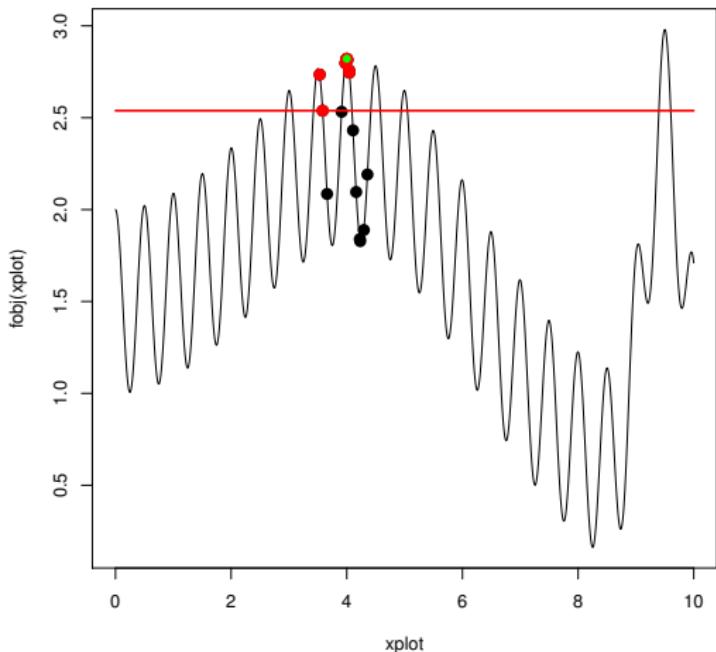
# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación

Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

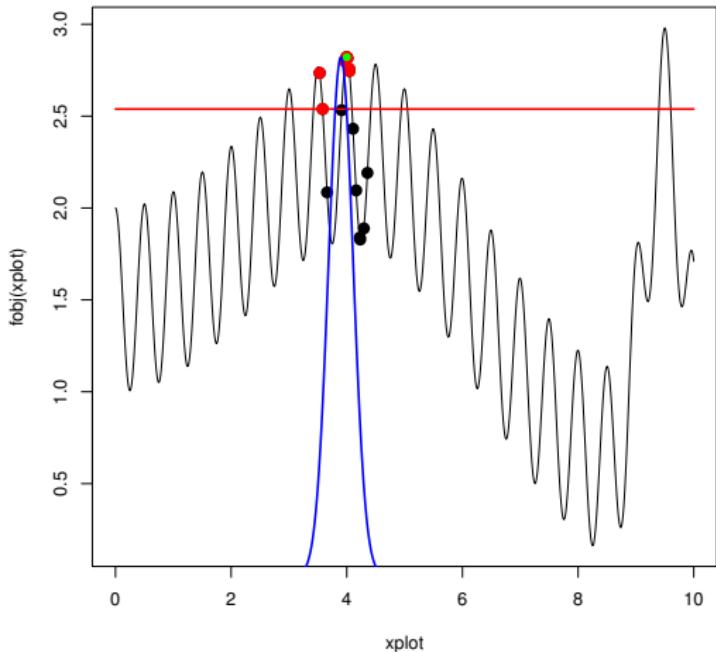
Selección

Estimación

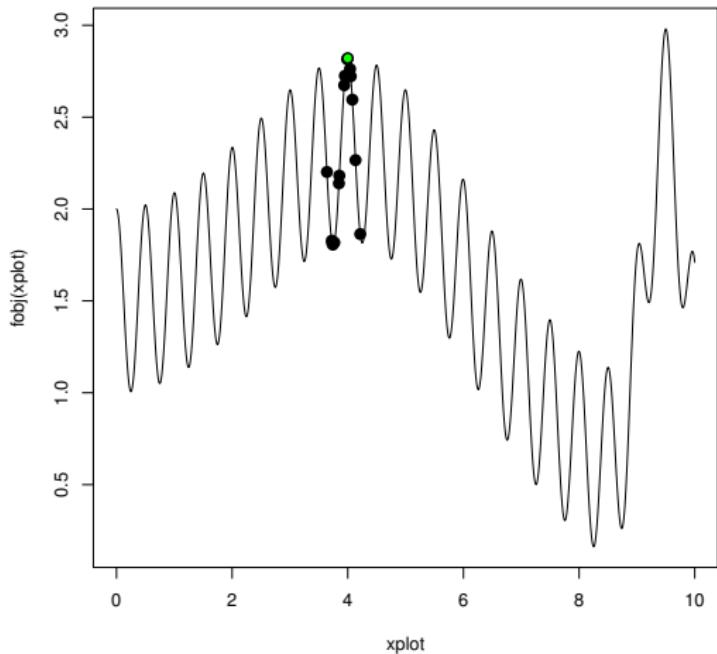
Muestreo

Evaluación

Reemplazo



# $UMDA_g$ pasos

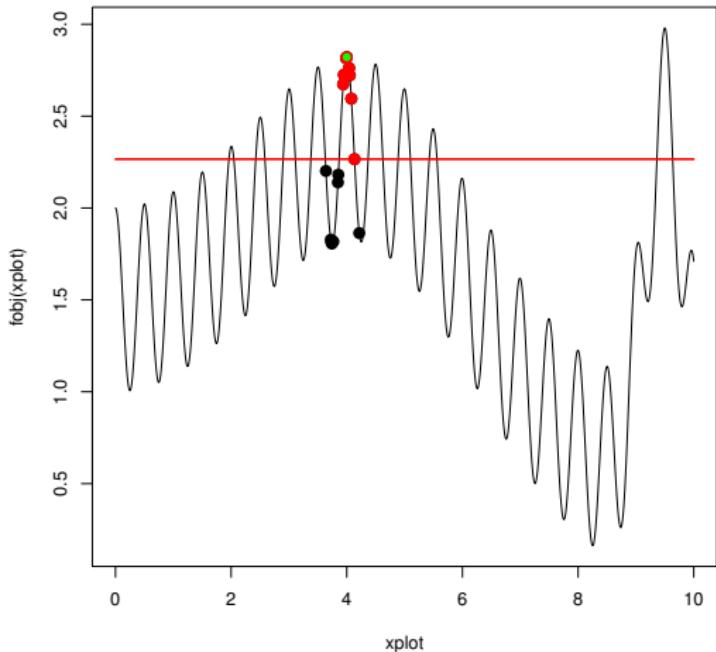


# $UMDA_g$ pasos

Inicialización  
Evaluación

Selección

Estimación  
Muestreo  
Evaluación  
Reemplazo



# $UMDA_g$ pasos

Inicialización

Evaluación

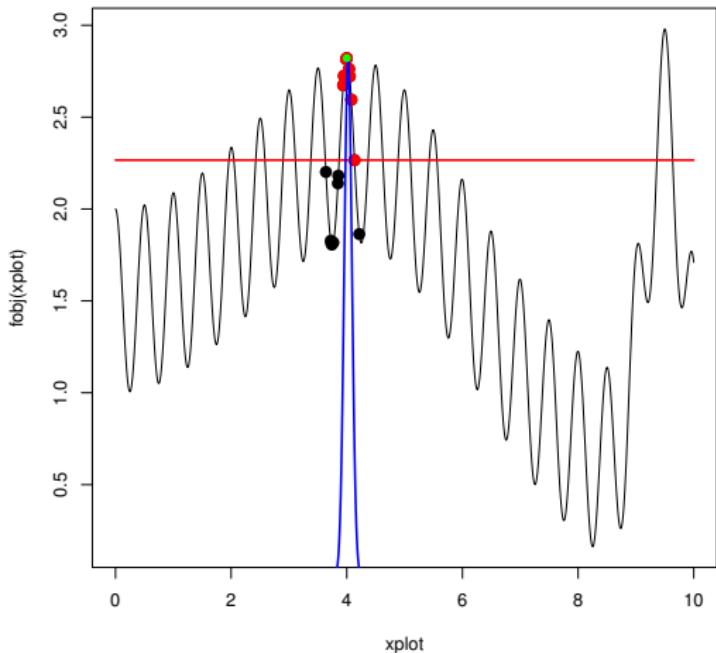
Selección

Estimación

Muestreo

Evaluación

Reemplazo



# Código en R

```
1 fobj<-function(x){  
2   f=0.5*cos(4.0*pi*x)+1.5*exp(0.12*x  
3     *sin(0.5*x))+2*exp(-(x-9.5)^2/  
4     0.25)  
5   return(f)  
6 }  
7 evalua<-function(func,X){  
8   F=NULL  
9   for (i in 1:length(X))  
10    F=c(F,func(X[i]))  
11  return(F)  
12 }
```

Listing 6 : Función objetivo y evaluador en una dimensión.

```
1 UMDAg<-function(fobj,ngen=50, npop=16, lim=  
2   c(0,10))  
3 {  
4   X=runif(npop,lim[1],lim[2])  
5   F=evalua(fobj,X)  
6   for (gen in 1:ngen)  
7   {  
8     ls=sort(F, index.return=TRUE, decreasing=  
9       TRUE)$ix  
10    umbral=F[ls[npop/2]]  
11    S=X[ls[1:(npop/2)]]  
12    Fs=F[ls[1:(npop/2)]]  
13    fbest=Fs[1]  
14    xbest=S[1]  
15    mu=mean(S)  
16    sd=sd(S)  
17    X=rnorm(npop-1,mu,sd)  
18    X[X<lim[1]]=lim[1]  
19    X[X>lim[2]]=lim[2]  
20    F=evalua(fobj,X)  
21    F=c(F,fbest)  
22    X=c(X,xbest)  
23  }  
24  return (list(xbest,fbest))  
25 }
```

Listing 7 : UMDAg completo en una dimensión.

# UMDA en $n$ dimensiones Código en R

```
1 fobj<-function(x){  
2   f=sum(0.5*cos(4.0*pi*x)  
3     +1.5*exp(0.12*x*sin  
4     (0.5*x))+2*exp(-(x  
5     -9.5)^2/0.25))  
6   return(f)  
7 }  
8 evalua<-function(func,X){  
9   F=NULL  
10  for(i in 1:nrow(X))  
11    F=c(F,func(X[i,]))  
12  return(F)  
13 }
```

Listing 8 : Función objetivo y evaluador en  $n$  dimensiones.

```
14 UMDAg<-function(fobj,nvar=10,ngen=100,npop=1600,lim=  
15   c(0,10)){  
16   X=matrix(runif(npop*nvar,lim[1],lim[2]),nrow=npop)  
17   F=evalua(fobj,X)  
18   for(gen in 1:ngen){  
19     ls=sort(F,index.return=TRUE,decreasing=TRUE)$ix  
20     umbral=F[ls[npop/2]]  
21     S=X[,ls[1:(npop/2),]]  
22     Fs=F[ls[1:(npop/2)]]  
23     fbest=Fs[1]  
24     xbest=S[,1]  
25     mu=colMeans(S)  
26     sd=apply(S,2,sd)  
27     X=apply(cbind(mu,sd),1,function(x) rnorm(npop-1,x  
28       [1],x[2]))  
29     X[X<lim[1]]=lim[1]  
30     X[X>lim[2]]=lim[2]  
31     F=evalua(fobj,X)  
32     F=c(F,fbest)  
33     X=rbind(X,xbest)  
34   }  
35   return(list(xbest,fbest))  
36 }
```

Listing 9 : UMDAg completo en  $n$  dimensiones

# UMDA en $n$ dimensiones Código en R

```
1 fobj<-function(x){  
2   f=sum(0.5*cos(4.0*pi*x)  
3     +1.5*exp(0.12*x*sin  
4     (0.5*x))+2*exp(-(x  
5     -9.5)^2/0.25))  
6   return(f)  
7 }  
8 evalua<-function(func,X){  
9   F=NULL  
10  for(i in 1:nrow(X))  
11    F=c(F,func(X[i,]))  
12  return(F)  
13 }
```

Listing 10 : Función objetivo y evaluador en  $n$  dimensiones.

Evaluación por renglones.  
Vector vs escalar

```
14 UMDAg<-function(fobj,nvar=10,ngen=100,npop=1600,lim=  
15   c(0,10)){  
16   X=matrix(runif(npop*nvar,lim[1],lim[2]),nrow=npop)  
17   F=evalua(fobj,X)  
18   for(gen in 1:ngen){  
19     ls=sort(F,index.return=TRUE,decreasing=TRUE)$ix  
20     umbral=F[ls[npop/2]]  
21     S=X[,ls[1:(npop/2),]]  
22     Fs=F[,ls[1:(npop/2)]]  
23     fbest=Fs[1]  
24     xbest=S[,1]  
25     mu=colMeans(S)  
26     sd=apply(S,2,sd)  
27     X=apply(cbind(mu,sd),1,function(x) rnorm(npop-1,x  
28       [1],x[2]))  
29     X[X<lim[1]]=lim[1]  
30     X[X>lim[2]]=lim[2]  
31     F=evalua(fobj,X)  
32     F=c(F,fbest)  
33     X=rbind(X,xbest)  
34   }  
35   return(list(xbest,fbest))  
36 }
```

Listing 11 : UMDAg completo en  $n$  dimensiones

# UMDA en $n$ dimensiones Código en R

```
1 fobj<-function(x){  
2   f=sum(0.5*cos(4.0*pi*x)  
3     +1.5*exp(0.12*x*sin  
4     (0.5*x))+2*exp(-(x  
5     -9.5)^2/0.25))  
6   return(f)  
7 }  
8 evalua<-function(func,X)->  
9   F=NULL  
10  for (i in 1:nrow(X))  
11    F=c(F,func(X[i,]))  
12  return(F)
```

Listing 12 : Función objetivo y evaluador en  $n$  dimensiones.

La población es una matriz

```
1 UMDAg<-function(fobj , nvar=10,ngen=100, npop=1600, lim=  
2   c(0,10) ){  
3   X=matrix(runif(npop+nvar,lim[1],lim[2]),nrow=npop)  
4   F=evalua(fobj ,X)  
5   for (gen in 1: ngen){  
6     ls=sort(F, index .return=TRUE, decreasing=TRUE)$ix  
7     umbral=F[ls [npop/2]]  
8     S=X[ls [1:( npop/2)],]  
9     Fs=F[ls [1:( npop/2)]]  
10    fbest=Fs [1]  
11    xbest=S [1,]  
12    mu=colMeans(S)  
13    sd=apply(S,2, sd)  
14    X=apply(cbind(mu, sd),1, function(x) rnorm(npop-1,x  
15      [1],x [2]))  
16    X[X<lim [1]]=lim [1]  
17    X[X>lim [2]]=lim [2]  
18    F=evalua (fobj ,X)  
19    F=c(F, fbest)  
20    X=rbind(X, xbest)  
21  }  
22  return (list (xbest ,fbest ))
```

Listing 13 : UMDAg completo en  $n$  dimensiones

# UMDA en $n$ dimensiones Código en R

```
1 fobj<-function(x){  
2   f=sum(0.5*cos(4.0*pi*x)  
3     +1.5*exp(0.12*x*sin  
4     (0.5*x))+2*exp(-(x  
5     -9.5)^2/0.25))  
6   return(f)  
7 }  
8 evalua<-function(func,X){  
9   F=NULL  
10  for (i in 1:nrow(X))  
11    F=c(F,func(X[i,]))  
12  return(F)  
13 }
```

Listing 14 : Función objetivo y evaluador en  $n$  dimensiones.

Conjunto seleccionado como matriz

```
1 UMDAg<-function(fobj ,nvar=10,ngen=100, npop=1600, lim=  
2   c(0,10) ){  
3   X=matrix ( runif(npop*nvar ,lim [1] ,lim [2]) ,nrow=npop)  
4   F=evalua(fobj ,X)  
5   for (gen in 1: ngen){  
6     ls=sort(F, index .return=TRUE, decreasing=TRUE)$ix  
7     umbral=F[ ls [ npop /2]]  
8     S=X[ ls [1:( npop /2) ],]  
9     Fs=F[ ls [1:( npop /2) ]]  
10    fbest=Fs [1]  
11    xbest=S [1 ,]  
12    mu.colMeans(S)  
13    sd=apply(S,2 ,sd)  
14    X=apply(cbind(mu, sd) ,1, function(x) rnorm(npop-1,x  
15      [1] ,x [2] ))  
16    X[X<lim [1]]=lim [1]  
17    X[X>lim [2]]=lim [2]  
18    F=evalua(fobj ,X)  
19    F=c(F, fbest)  
20    X=rbind(X, xbest)  
21  }  
22  return (list(xbest ,fbest))  
23 }
```

Listing 15 : UMDAg completo en  $n$  dimensiones

# UMDA en $n$ dimensiones Código en R

```
1 fobj<-function(x){  
2   f=sum(0.5*cos(4.0*pi*x)  
3     +1.5*exp(0.12*x*sin  
4     (0.5*x))+2*exp(-(x  
5     -9.5)^2/0.25))  
6   return(f)  
7 }  
8 evalua<-function(func,X){  
9   F=NULL  
10  for (i in 1:nrow(X))  
11    F=c(F,func(X[i,]))  
12  return(F)  
13 }
```

Listing 16 : Función objetivo y evaluador en  $n$  dimensiones.

Medias y varianzas en cada dimensión (columnas de la población).

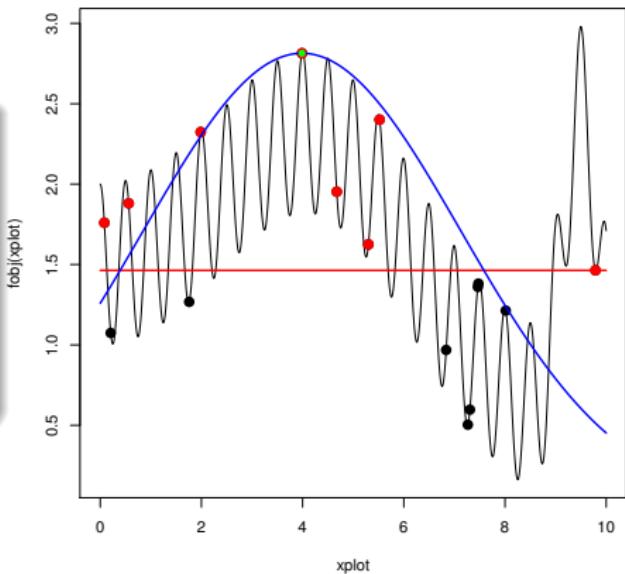
```
1 UMDAg<-function(fobj ,nvar=10,ngen=100, npop=1600, lim=  
2   c(0,10) ){  
3   X=matrix ( runif(npop*nvar ,lim [1] ,lim [2]) ,nrow=npop)  
4   F=evalua(fobj ,X)  
5   for (gen in 1: ngen){  
6     ls=sort(F, index .return=TRUE, decreasing=TRUE)$ix  
7     umbral=F[ ls [ npop /2]]  
8     S=X[ ls [ 1:( npop /2)] ,]  
9     Fs=F[ ls [ 1:( npop /2)] ]  
10    fbest=Fs[1]  
11    xbest=S[1 ,]  
12    mu.colMeans(S)  
13    sd=apply(S,2 ,sd)  
14    X=apply(cbind(mu ,sd),1 , function(x) rnorm(npop-1,x  
15      [1] ,x[2]))  
16    X[X<lim [1]]=lim [1]  
17    X[X>lim [2]]=lim [2]  
18    F=evalua(fobj ,X)  
19    F=c(F, fbest)  
20    X=rbind(X, xbest)  
21  }  
22  return ( list (xbest ,fbest ))  
23 }
```

Listing 17 : UMDAg completo en  $n$  dimensiones

# Comentarios

## Sobre $UMDA_g$

Aunque el  $UMDA_g$  es facil de implementar, tanto la estimación, selección y muestreo son computacionalmente baratos y simples. Sin embargo:  **$UMDA_g$  solo toma en cuenta la región donde cayeron mas puntos, y las detecta y muestrea como regiones promisorias.**



## ¿Porqué funcionan?

En el 2004 Mühlenbein et al. publicaron un artículo sobre la convergencia al óptimo de EDAs ideales.

## ¿Porqué funcionan?

La idea básica: si hay una población y conjunto seleccionado suficientemente grande (infinita), y la función de probabilidad es la densidad exacta del conjunto seleccionado, el EDA converge al óptimo global.

## ¿Porqué funcionan?

Lo mismo pasa con diferentes métodos de selección, el requisito es que la probabilidad del óptimo sea mayor en cada iteración.

# ¿Porqué funcionan?

La selección Boltzmann se refiere a que un individuo se puede seleccionar del dominio de búsqueda (población infinita) con probabilidad  $p(x) = \exp(\beta f(x))/Z$ . Para un problema de maximización,  $f(x)$  es la función objetivo mayor que cero,  $Z$  una constante de normalización y  $\beta$  regula que tanto *picuda* es la probabilidad.

# Boltzmann Univariate Marginal Distribution Algorithm

- Estimar y obtener muestras de una Gaussiana univariada es sencillo y barato computacionalmente.
- El desempeño del  $UMDA_g$  si bien no es el mejor es relativamente bueno para cierto tipo de problemas.
- Un problema notorio es que no utiliza información disponible del valor de función objetivo para determinar cuales son las mejores regiones, sino que las determina de acuerdo al numero de puntos en ellas.

# Boltzmann Univariate Marginal Distribution Algorithm

- Estimar y obtener muestras de una Gaussiana univariada es sencillo y barato computacionalmente.
- El desempeño del  $UMDA_g$  si bien no es el mejor es relativamente bueno para cierto tipo de problemas.
- Un problema notorio es que no utiliza información disponible del valor de función objetivo para determinar cuales son las mejores regiones, sino que las determina de acuerdo al numero de puntos en ellas.
- La distribución Boltzmann exacta ¡Converge al Óptimo Global! (con población infinita).
- La probabilidad mayor está en el óptimo y las zonas se pueden explorar proporcionalmente a  $\exp(\beta f(x))$ .
- NO SE PUEDE USAR LA BOLZTAMNN por lo de POBLACIÓN INFINITA.

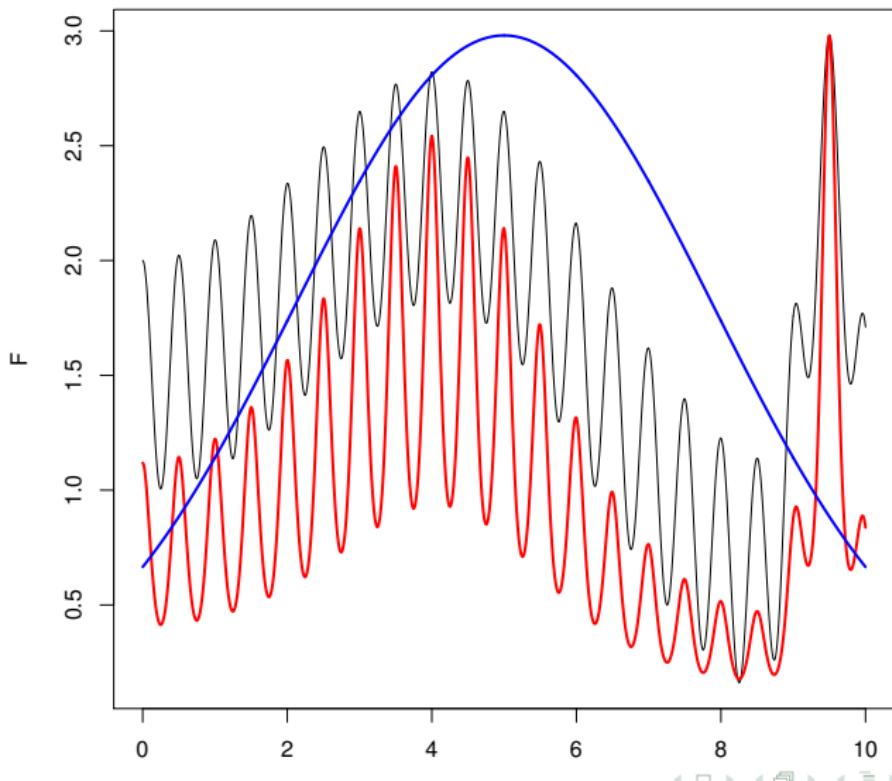
# Boltzmann Univariate Marginal Distribution Algorithm

- Estimar y obtener muestras de una Gaussiana univariada es sencillo y barato computacionalmente.
- El desempeño del  $UMDA_g$  si bien no es el mejor es relativamente bueno para cierto tipo de problemas.
- Un problema notorio es que no utiliza información disponible del valor de función objetivo para determinar cuales son las mejores regiones, sino que las determina de acuerdo al numero de puntos en ellas.

- La distribución Boltzmann exacta ¡Converge al Óptimo Global! (con población infinita).
- La probabilidad mayor está en el óptimo y las zonas se pueden explorar proporcionalmente a  $\exp(\beta f(x))$ .
- NO SE PUEDE USAR LA BOLZTAMNN por lo de POBLACIÓN INFINITA.

**¡La idea es equipar a un algoritmo que usa la Gaussiana con las ventajas de usar la Boltzmann!**

# Boltzmann Univariate Marginal Distribution Algorithm

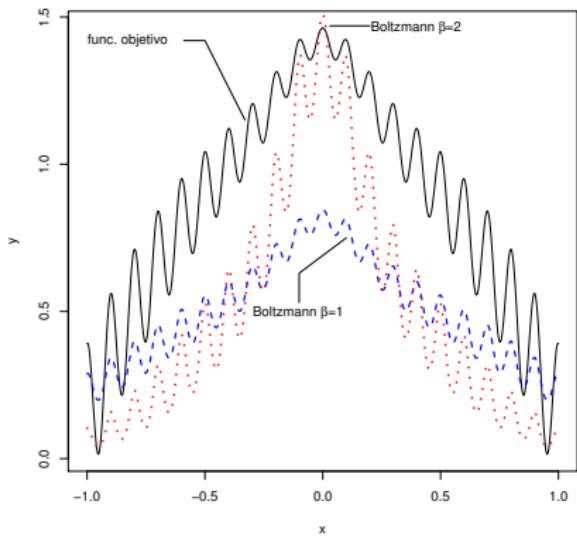


# Aproximar la Boltzmann con una Gaussiana

## La distribución Boltzmann:

Para  $\beta \geq 0$  definimos la densidad de probabilidad Boltzmann de una función  $g(x)$  como:

$$p_{\beta,g(x)} := \frac{e^{\beta g(x)}}{\int_y e^{\beta g(y)}} := \frac{e^{\beta g(x)}}{Z_g(\beta)} \quad (1)$$



### Características de la func. Boltzmann:

- Donde se incrementa/decremente la función objetivo, se incrementa/decremente la  $p$  exponencialmente (mismo comportamiento).
- Si para un punto  $x^*$ ,  $g(x^*) = \max g(x)$ , entonces  $p(x^*, t) = \max p(x, t)$ .
- Se puede regular la masa de probabilidad simplemente modificando la  $\beta$ .

Sin embargo no se puede utilizar la función de Boltzmann directamente, puesto que se necesita conocer la función  $g(x)$  en todos los puntos (población infinita).

# Aproximar la Boltzmann con una Gaussiana

Ya que no es posible utilizar la Boltzmann, la aproximamos con una normal.

# Aproximar la Boltzmann con una Gaussiana

Ya que no es posible utilizar la Boltzmann, la aproximamos con una normal.  
Modelo Univariado

$$Q(x) = \prod_{i=1}^n Q_i(x), \quad \text{donde} \quad Q_i(x) = Q(x_i, \mu_i, \nu_i) = \frac{e^{-\frac{(x_i - \mu_i)^2}{2\nu_i}}}{(2\pi\nu_i)^{1/2}} \quad (2)$$

# Aproximar la Boltzmann con una Gaussiana

Ya que no es posible utilizar la Boltzmann, la aproximamos con una normal.  
Modelo Univariado

$$Q(x) = \prod_{i=1}^n Q_i(x), \text{ donde } Q_i(x) = Q(x_i, \mu_i, \nu_i) = \frac{e^{-\frac{(x_i - \mu_i)^2}{2\nu_i}}}{(2\pi\nu_i)^{1/2}} \quad (2)$$

La divergencia de Kullback-Leibler se utiliza para saber qué tanto se puede explicar una distribución por medio de otra.

$$K_{Q,P} = \int_x Q_x \log \frac{Q_x}{P_x} dx, \quad (3)$$

# Aproximar la Boltzmann con una Gaussiana

Ya que no es posible utilizar la Boltzmann, la aproximamos con una normal.  
Modelo Univariado

$$Q(x) = \prod_{i=1}^n Q_i(x), \quad \text{donde} \quad Q_i(x) = Q(x_i, \mu_i, \nu_i) = \frac{e^{-\frac{(x_i - \mu_i)^2}{2\nu_i}}}{(2\pi\nu_i)^{1/2}} \quad (2)$$

La divergencia de Kullback-Leibler se utiliza para saber qué tanto se puede explicar una distribución por medio de otra.

$$K_{Q,P} = \int_x Q_x \log \frac{Q_x}{P_x} dx, \quad (3)$$

La idea es encontrar los parámetros  $\theta$  que minimizan  $K_{Q,P}$ . Para encontrar la distribución normal paramétrica más cercana a la Boltzmann.

$$\frac{\partial K_{Q,P}}{\partial \theta} = \int_x \left[ 1 + \log \frac{Q_x}{P_x} \right] \frac{\partial Q_x}{\partial \theta} dx. \quad (4)$$

# Aproximar la Boltzmann con una Gaussiana

Derivando e igualando a 0, con respecto de la media  $\mu$  y la varianza  $\nu$ , encontramos:

$$\frac{\partial K_{Q,P}}{\partial \mu} = -\frac{\beta}{\nu} \int_x Q_x(x - \mu) g(x) dx \approx -\frac{\beta}{\nu} \sum_{x_j \in X} (x_j - \mu) g(x_j). \quad (5)$$

$$\mu \approx \frac{\sum_j g(x_j) x_j}{\sum_j g(x_j)}. \quad (6)$$

$$\nu = \frac{\int_x g(x)(x - \mu)^2 Q_x dx}{\frac{3+4\sqrt{2}}{8\sqrt{2}\beta} + \int_x g(x) Q_x dx}, \quad (7)$$

$$\nu \approx \frac{\sum_{x_j \in X} g(x_j)(x_j - \mu)^2}{T' + \sum_{x_j \in X} g(x_j)}, \quad (8)$$

donde

$T'$  = es una constante relacionada con el tamaño de la muestra y el parámetro  $\beta$ .

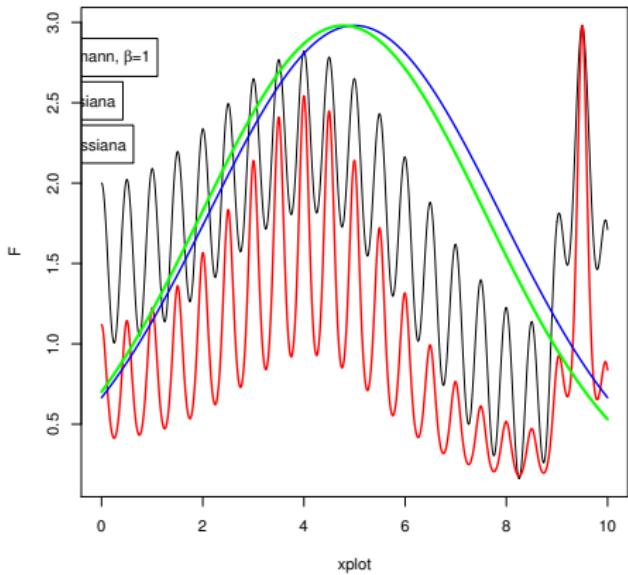
**¡Las importantes son las ecuaciones (6) y (7)!**

# Hasta aquí

Tenemos formulas para estimar la media y la varianza de una Gaussiana que approxima la Boltzmann

$$\mu \approx \frac{\sum_j g(x_j)x_j}{\sum_j g(x_j)}.$$

$$\nu \approx \frac{\sum_{x_j \in X} g(x_j)(x_j - \mu)^2}{T' + \sum_{x_j \in X} g(x_j)}.$$



# Características deseadas

**Las características deseadas en un algoritmo ideal y uno práctico son:**

## Ideal

Que converja  
Que el punto de convergencia sea el óptimo

## Práctico

Que converja  
Que el punto de convergencia sea una *buena aproximación* al óptimo

# Características deseadas

Las características deseadas en un algoritmo ideal y uno práctico son:

Ideal	Práctico
Que converja	Que converja
Que el punto de convergencia sea el óptimo	Que el punto de convergencia sea una <i>buena aproximación</i> al óptimo

---

La estrategia para lograrlo:

Ideal	Práctico
Incrementar el valor esperado	Incrementar la aproximación al valor esperado
Mantener una alta probabilidad en el óptimo	Mantener la mejor solución conocida (elitismo) y asignarle un valor de <i>pdf</i> alto.

# Método de selección

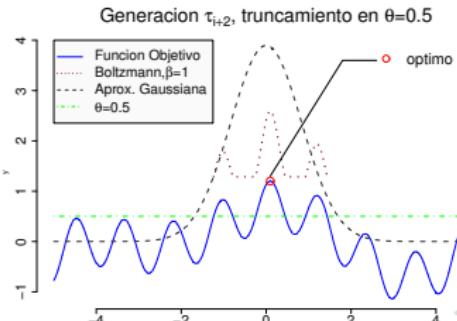
Proponemos utilizar un método de selección por truncamiento que:

- Asegura convergencia.
- La convergencia es a la mejor solución conocida (individuo elite).
- Asegura incrementar/mantener la aproximación al valor esperado al menos cada cierto número de generaciones (media del modelo), lo hace la mayoría de las veces, y asegura la convergencia de la aproximación al valor esperado al individuo elite (la media converge al mejor valor encontrado).

# Método de selección

## Método de selección por truncamiento.

- ① Para la generación inicial  $t = 0$ , sean  $g(x_i, 0)$  para  $i = 1..N$ , los valores objetivo de la población inicial. Se define:  $\theta_0 = \min g(x_i, 0)$ .
- ② Para  $t > 0$ , asignar:  
$$\theta_t = \min(g(x_i, t) | g(x_i, t) \geq \theta_{t-1})$$
.
- ③ Si para los individuos ordenados decrecientemente  $g(x_{N/2}) \geq \theta_t$ , asignar  
$$\theta_t = g(x_{N/2})$$
. Donde  $N$  es el tamaño de población.
- ④ Truncar la población tal que  $g(x_s, t) \geq \theta_t$ . Donde  $x_s$  son los individuos los cuales su valor de función objetivo es mayor o igual a  $\theta_t$ .



# Método de truncamiento

Con la población ordenada decreciente (caso de maximización).

Población

$t = 0$

$g(x_0)$
$g(x_1)$
$g(x_2)$
$g(x_3)$
$g(x_4)$
$g(x_5)$
...
$g(x_i)$
...
$g(x_n)$

Trunc.

en  $\theta_0$



# Método de truncamiento

Con la población ordenada decreciente (caso de maximización).

Población

$t = 0$

$g(x_0)$
$g(x_1)$
$g(x_2)$
$g(x_3)$
$g(x_4)$
$g(x_5)$
...
$g(x_i)$
...
$g(x_n)$

$t = 1$

$g(x_0)$
$g(x_1)$
$g(x_2)$
$g(x_3)$
$g(x_4)$
$g(x_5)$
...
$g(x_{\frac{n}{2}})$
...
$g(x_n)$

Trunc.

en  $\theta_1$



Trunc.

en  $\theta_0$



# Método de truncamiento

Con la población ordenada decreciente (caso de maximización).

Población

$t = 0$

$g(x_0)$
$g(x_1)$
$g(x_2)$
$g(x_3)$
$g(x_4)$
$g(x_5)$
...
$g(x_i)$
...
$g(x_n)$

Trunc.  
en  $\theta_0$

$t = 1$

$g(x_0)$
$g(x_1)$
$g(x_2)$
$g(x_3)$
$g(x_4)$
$g(x_5)$
...
$g(x_{\frac{n}{2}})$
...
$g(x_n)$

Trunc.  
en  $\theta_1$

$t = 2$

$g(x_0)$
$g(x_1)$
$g(x_2)$
$g(x_3)$
$g(x_4)$
...
$g(x_{\frac{n}{2}})$
...
$g(x_{n-1})$
$g(x_n)$

Trunc.  
en  $\theta_2$

- ① Dar el parámetro y criterio de paro:

**nsample**  $\leftarrow$  Número de individuos en la población.

**minvar**  $\leftarrow$  Mínima varianza permitida.

- ② Generar con la dist. uniforme la población inicial  $P_0$ , asignar  $t = 0$ .

- ③ Mientras  $v > minvar$  para todas las dimensiones

- ①  $t \leftarrow t + 1$

- ② Evaluar y seleccionar de acuerdo de al método de truncamiento.

- ③ Calcular  $\mu$  y  $v$  (para todas las dimensiones) utilizando el conjunto seleccionado (de tamaño  $nselec$ ), y las ecuaciones resultantes de minimizar  $K_{Q,P}$

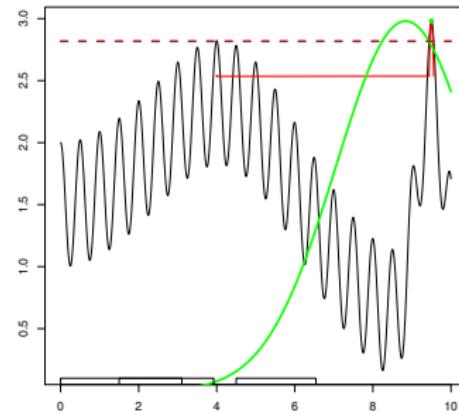
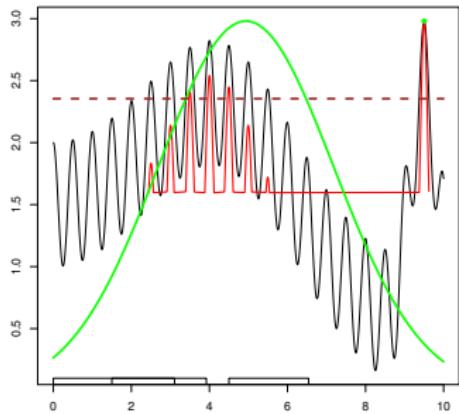
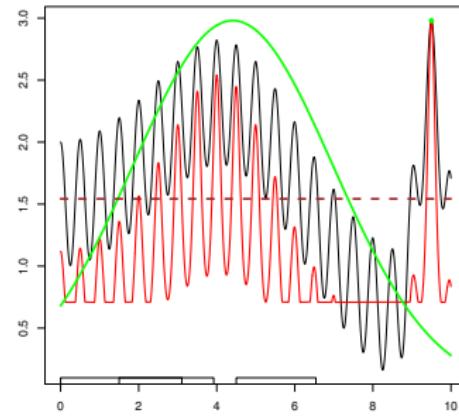
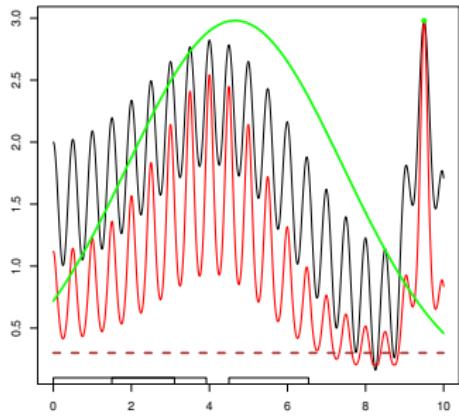
$$\mu \approx \frac{\sum_1^{nselec} x_i \bar{g}(x_i)}{\sum_1^{nselec} \bar{g}(x_i)}, \quad v \approx \frac{\sum_1^{nselec} \bar{g}(x_i)(x_i - \mu)^2}{1 + \sum_1^{nselec} \bar{g}(x_i)},$$

donde  $\bar{g}(x_i) = g(x) - g(x_{nselec}) + 1$ .

Nota: los individuos pueden estar ordenados para simplificar el cálculo, y  $g(x_{nselec})$  es el menor valor objetivo de los individuos seleccionados.

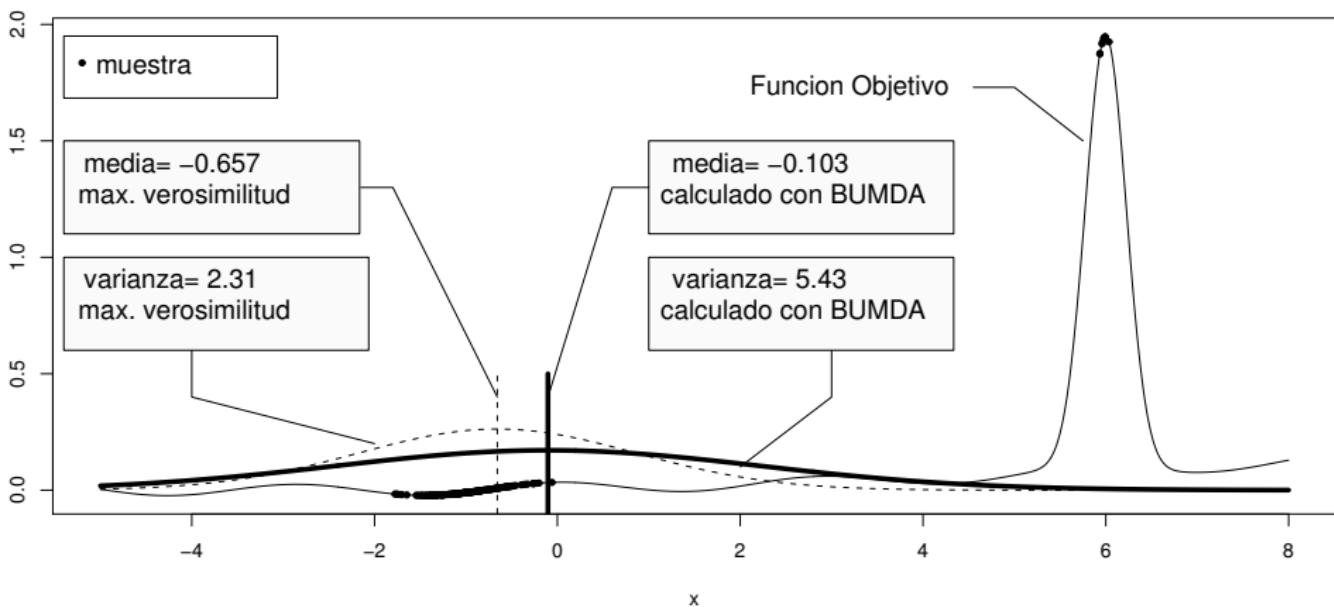
- ④ Generar  $nsample - 1$  individuos del nuevo modelo  $Q(x, \mu, v, t)$ , e insertar el individuo elite.
- ⑤ Regresar el individuo elite como la mejor aproximación al óptimo.

# Función del método de truncamiento



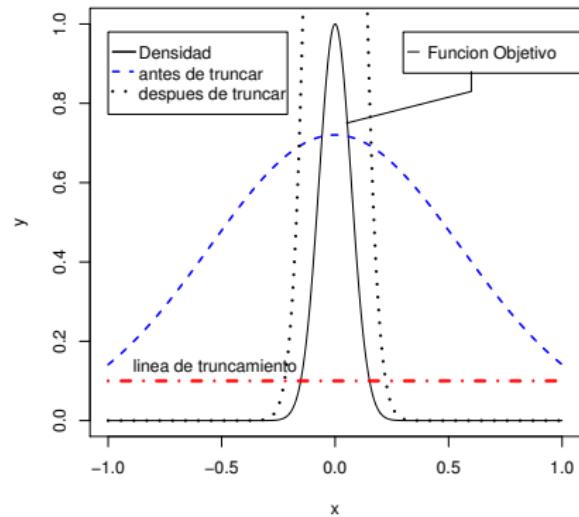
## Adaptación de los parámetros.

BUMDA vs Max. Verosimilitud



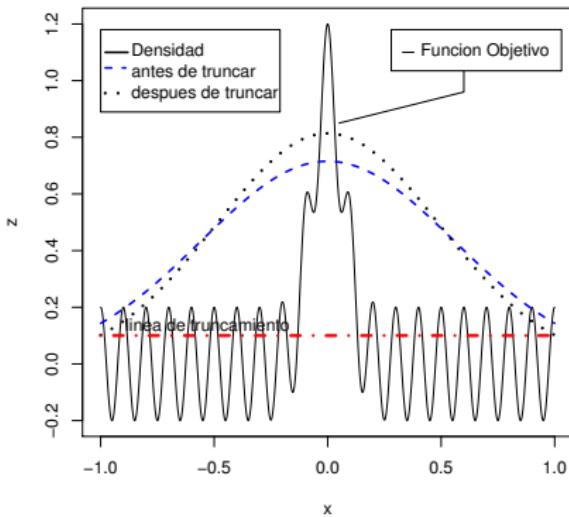
# Análisis del BUMDA

## Adaptación a la rugosidad de la función



(a)

$$\mu_{bf} = 0, v_{bf} = 0.306$$
$$\mu_{af} = 0, v_{af} = 0.006$$



(b)

$$\mu_{bf} = 0, v_{bf} = 0.311$$
$$\mu_{af} = 0, v_{af} = 0.240$$

# BUMDA en $n$ dimensiones Código en R

```
1 BUMDA<-function(fobj ,nvar=10,npop=1600,lim=c(0,10) ,
2 minsd=1e-6 )
3 {
4   X=matrix(runif(npop*nvar ,lim [1] ,lim [2]) ,nrow=npop)
5   F=evalua(fobj ,X)
6   umbral=-Inf
7   while (sum(sd>minsd )!=0)
8   {
9     ls=sort(F, index .return=TRUE, decreasing=TRUE)$ix
10    umbral=max(F[ls[npop/2]], min(F[F>=umbral]))
11    iumbral=which(F[ls]==umbral)
12    S=X[ls[1:i umbral],]
13    Fs=F[ls[1:i umbral]]
14    fbest=Fs[1]
15    xbest=S[1,]
16    G=Fs-min(Fs)+1
17    mu=colSums(S*G)/sum(G)
18    sd=sqrt(colSums((t(t(S)-mu))^2*G)/sum(G))
19    X=apply(cbind(mu, sd), 1,
20             function(x) rnorm(npop-1,x[1],x[2]))
21    X[X<lim[1]]=lim[1]
22    X[X>lim[2]]=lim[2]
23    F=evalua(fobj ,X)
24    F=c(F, fbest)
25    X=rbind(X, xbest)
26  }
27  return (list(xbest, fbest))
28 }
```

# BUMDA en $n$ dimensiones Código en R

Inicialización del umbral de truncamiento que garantiza convergencia

```
2 BUMDA<-function (fobj , nvar=10,npop=1600,lim=c(0,10) ,
3   minsd=1e-6 )
4 {
5   X=matrix( runif (npop*nvar , lim [1] , lim [2]) , nrow=npop)
6   umbral=Inf
7   while ( sum (sd>minsd) !=0 )
8   {
9     ls=sort (F, index . return=TRUE, decreasing=TRUE)$ix
10    umbral=max(F[ ls [npop / 2]], min(F[F>=umbral]))
11    iumbral=which(F[ ls ]==umbral)
12    S=X[ ls [1:i umbral], ]
13    Fs=F[ ls [1:i umbral]]
14    fbest=Fs [1]
15    xbest=S [1,]
16    G=Fs-min (Fs)+1
17    mu=colSums(S*G)/sum(G)
18    sd=sqrt (colSums(( t (t (S)-mu) ^2*G)/sum(G)))
19    X=apply (cbind(mu, sd) ,1 ,
20              function (x) rnorm (npop-1,x [1],x [2]))
21    X[X<lim [1]]=lim [1]
22    X[X>lim [2]]=lim [2]
23    F=evalua (fobj ,X)
24    F=c(F, fbest)
25    X=rbind(X, xbest)
26  }
27  return ( list (xbest , fbest))
28 }
```

# BUMDA en $n$ dimensiones Código en R

El criterio de paro está dado por la convergencia y no por generaciones

```
1 BUMDA<-function (fobj , nvar=10,npop=1600,lim=c(0,10) ,
2   minsd=1e-6 )
3 {
4   X=matrix( runif (npop*nvar , lim [1] , lim [2]) , nrow=npop)
5   F=evalua (fobj , X)
6   umbral=-Inf
7   while ( sum (sd>minsd ) !=0 )
8   {
9     ls=sort (F, index . return=TRUE, decreasing=TRUE)$ix
10    umbral=max(F[ ls [npop / 2]] , min(F[F>=umbral]))
11    iumbral=which(F[ ls ]==umbral)
12    S=X[ ls [1:i umbral] , ]
13    Fs=F[ ls [1:i umbral] ]
14    fbest=Fs [1]
15    xbest=S [1,]
16    G=Fs-min (Fs)+1
17    mu=colSums(S*G)/sum(G)
18    sd=sqrt (colSums(( t (t (S)-mu) ^2*G)/sum(G)))
19    X=apply (cbind(mu, sd) ,1 ,
20              function (x) rnorm (npop -1,x [1],x [2]))
21    X[X<lim [1]]=lim [1]
22    X[X>lim [2]]=lim [2]
23    F=evalua (fobj , X)
24    F=c(F, fbest)
25    X=rbind(X, xbest)
26  }
27  return ( list (xbest , fbest))
28 }
```

# BUMDA en $n$ dimensiones Código en R

```
1 BUMDA<-function( fobj ,nvar=10,npop=1600,lim=c(0,10) ,
2 minsd=1e-6 )
3 {
4   X=matrix( runif( npop*nvar ,lim [1] ,lim [2]) ,nrow=npop)
5   F=evalua( fobj ,X)
6   umbral=-Inf
7   while ( sum( sd>minsd ) !=0 )
8   {
9     ls=sort( F, index . return=TRUE, decreasing=TRUE)$ix
10    umbral=max( F[ ls [npop/2]] , min( F[F>=umbral] ))
11    iumbral=which( F[ ls ]==umbral)
12    S=X[ ls [1:i umbral] ,]
13    Fs=F[ ls [1:i umbral] ]
14    fbest=Fs[1]
15    xbest=S[1,]
16    G=Fs-min( Fs)+1
17    mu=colSums( S*G )/sum( G )
18    sd=sqrt( colSums( ( t( t( S )-mu ) ^2*G )/sum( G ) )
19    X=apply( cbind( mu, sd ) ,1 ,
20             function (x) rnorm( npop-1,x[1],x[2] ) )
21    X[X<lim [1]]=lim [1]
22    X[X>lim [2]]=lim [2]
23    F=evalua( fobj ,X)
24    F=c( F, fbest )
25    X=rbind( X, xbest )
26  }
27  return ( list ( xbest , fbest ) )
28 }
```

El umbral se utiliza para truncar el conjunto seleccionado.

# BUMDA en $n$ dimensiones Código en R

La función objetivo se traslada para asegurar que sea positiva y mayor que cero.

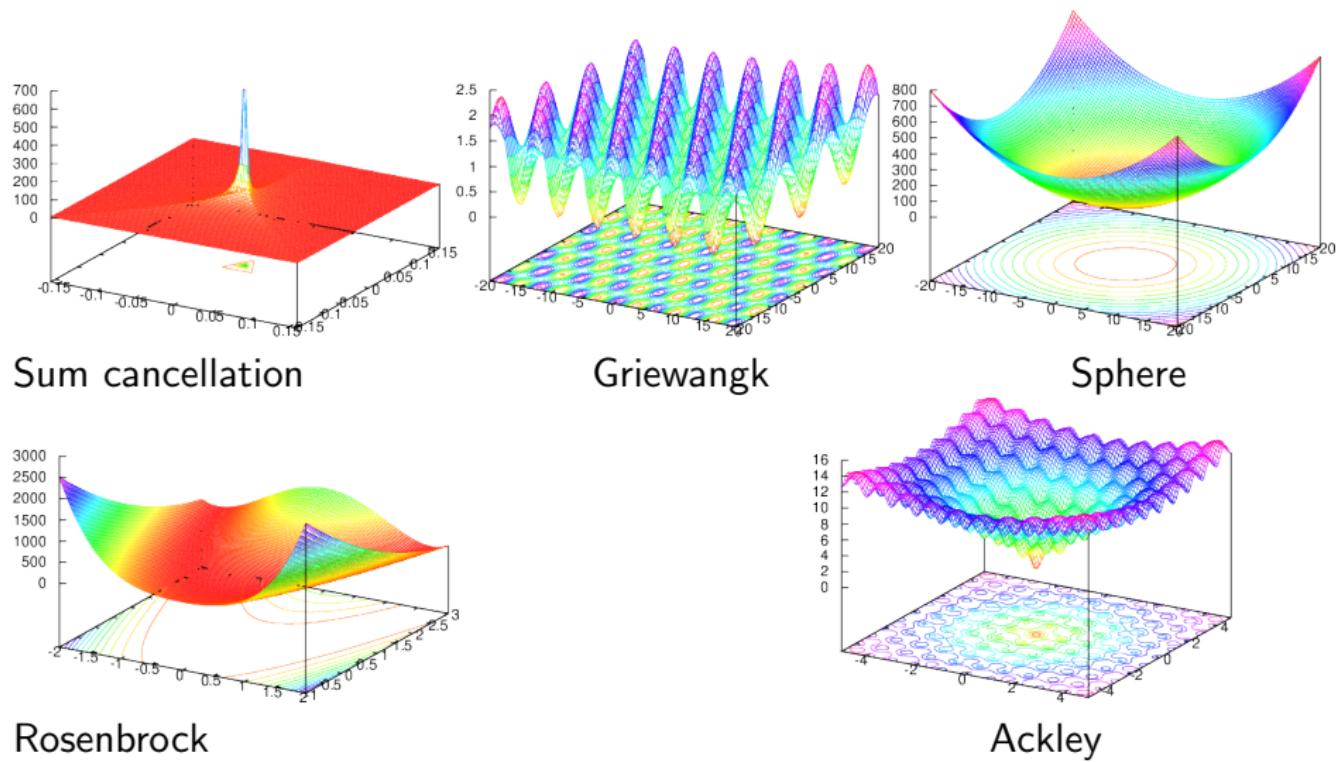
```
2 BUMDA<-function (fobj , nvar=10,npop=1600,lim=c(0,10) ,
3   minsd=1e-6 )
4   {
5     X=matrix ( runif ( npop*nvar , lim [1] , lim [2] ) , nrow=npop )
6     F=evalua ( fobj , X )
7     umbral=-Inf
8     while ( sum ( sd>minsd ) !=0 )
9     {
10       ls=sort ( F , index . return=TRUE , decreasing=TRUE ) $ ix
11       umbral=max ( F[ ls [ npop / 2 ] ] , min ( F[F>=umbral] ) )
12       iumbral=which ( F[ ls ]==umbral )
13       S=X[ ls [ 1:i umbral ] , ]
14       Fs=F[ ls [ 1:i umbral ] ]
15       fbest=Fs [ 1 ]
16       xbest=S [ 1 , ]
17       G=Fs-min ( Fs ) +1
18       mu=colSums ( S*G ) /sum ( G )
19       sd=sqrt ( colSums ( ( t ( t ( S ) -mu ) ^2*G ) /sum ( G ) ) )
20       X=apply ( cbind ( mu , sd ) , 1 ,
21                 function ( x ) rnorm ( npop -1,x [ 1 ] , x [ 2 ] ) )
22       X[X<lim [ 1 ]]=lim [ 1 ]
23       X[X>lim [ 2 ]]=lim [ 2 ]
24       F=evalua ( fobj , X )
25       F=c ( F , fbest )
26       X=rbind ( X , xbest )
27     }
28   return ( list ( xbest , fbest ) )
}
```

# BUMDA en $n$ dimensiones Código en R

Se calculan estimadores pesados de la media y la varianza.

```
1 BUMDA<-function (fobj , nvar=10,npop=1600,lim=c(0,10) ,
2   minsd=1e-6 )
3 {
4   X=matrix ( runif ( npop*nvar , lim [1] , lim [2] ) , nrow=npop )
5   F=evalua ( fobj , X )
6   umbral=-Inf
7   while ( sum ( sd > minsd ) !=0 )
8   {
9     ls=sort ( F , index . return=TRUE , decreasing=TRUE ) $ ix
10    umbral=max ( F[ ls [ npop / 2 ] ] , min ( F[F>=umbral] ) )
11    iumbral=which ( F[ ls ] == umbral )
12    S=X[ ls [ 1:i umbral ] , ]
13    Fs=F[ ls [ 1:i umbral ] ]
14    fbest=Fs [ 1 ]
15    xbest=S [ 1 , ]
16    G=Fs-min ( Fs ) +1
17    mu=colSums ( S*G ) /sum ( G )
18    sd=sqrt ( colSums ( ( t ( t ( S ) -mu ) ^2*G ) /sum ( G ) ) )
19    X=apply ( cbind ( mu , sd ) , 1 ,
20              function ( x ) rnorm ( npop-1,x [ 1 ] ,x [ 2 ] ) )
21    X[X<lim [ 1 ]]=lim [ 1 ]
22    X[X>lim [ 2 ]]=lim [ 2 ]
23    F=evalua ( fobj , X )
24    F=c ( F , fbest )
25    X=rbind ( X , xbest )
26  }
27  return ( list ( xbest , fbest ) )
28 }
```

# Funciones de prueba



# Test

Function	BUMDA	EMNA-B	BG-UMDA	$H_1 : \bar{F}_{BUMDA}$ better than $\bar{F}_{other}$	
	BUMDA	EMNA-B	BG-UMDA		
SumC 10d	$7.5E3 \pm 8.4E3$	$1E5 \pm 1.1E-7$	$5.8E4 \pm 2.3E4$	no	no
SumC 50d	$2.07 \pm 0.12$	$99910 \pm 160$	$1.39 \pm 0.1$	no	yes
Grie. 10d	$7.3E-7 \pm 1.7E-7$	$7.4E-7 \pm 1.1E-7$	$1.27E-4 \pm 4E-4$	no	no
Grie. 50d	$9E-7 \pm 8.4E-8$	$9.2E-7 \pm 5E-8$	$8.8E-7 \pm 7E-8$	no	no
Sphe. 10d	$7E-7 \pm 1.6E-7$	$7.5E-7 \pm 2.1E-7$	$5.9E-7 \pm 1.8E-7$	no	no
Sphe. 50d	$8.7E-7 \pm 8.1E-8$	$8.8E-7 \pm 1.1E-7$	$8.4E-7 \pm 8E-8$	no	no
Rose. 10d	$8.1 \pm 0.08$	$6.33 \pm 0.37$	$7.74 \pm 0.08$	no	no
Rose. 50d	$47.7 \pm 0.18$	$47.08 \pm 0.44$	$47.54 \pm 0.07$	no	no
Ackl. 10d	$8.3E-7 \pm 1.2E-7$	$8.4E-7 \pm 1E-7$	$8.3E-7 \pm 1.6E-7$	no	no
Ackl. 50d	$9.3E-7 \pm 4.3E-8$	$9.42E-7 \pm 4E-8$	$9.6E-7 \pm 4E-8$	no	yes

Table : Mean and standard deviation of best function value found in 20 runs for the Test problem 1. yes= the BUMDA is better than the other algorithm.

# Test

Function	BUMDA	EMNA-B	BG-UMDA	$H_1 : \bar{N}_{BUMDA}^{eval} < \bar{N}_{other}^{eval}$
	EMNA-B			BG-UMDA
SumC. 10d	$3E5 \pm 0$	$92520 \pm 840$	$300400 \pm 0$	NP
SumC. 50d	$3E5 \pm 0$	$301000 \pm 0$	$300400 \pm 0$	NP
Grie. 10d	<b><math>17262 \pm 384</math></b>	$134000 \pm 47000$	$229E3 \pm 64E3$	yes, p=5.8E-29
Grie. 50d	<b><math>39675 \pm 342</math></b>	$170100 \pm 1700$	$71880 \pm 420$	yes, p=0
Sphe. 10d	<b><math>14541 \pm 261</math></b>	$35200 \pm 420$	$35720 \pm 840$	yes, p=0
Sphe. 50d	<b><math>40695 \pm 325</math></b>	$192900 \pm 1600$	$82400 \pm 460$	yes, p=0
Rose. 10d	$3E5 \pm 0$	$300400 \pm 0$	$300400 \pm 0$	NP
Rose. 50d	$3E5 \pm 0$	$301000 \pm 0$	$300400 \pm 0$	NP
Ackl. 10d	<b><math>23257 \pm 287</math></b>	$43560 \pm 610$	$44000 \pm 530$	yes, p=0
Ackl. 50d	<b><math>58850 \pm 348</math></b>	$231800 \pm 4300$	$98920 \pm 530$	yes, p=0

Table : Average and standard deviation of evaluations for Test problem 1. **yes= the BUMDA is better than the other algorithm. NP= Comparison Not Possible.**

# Conclusiones

- Existen implementaciones simples de Algoritmos de Estimación de distribución para resolver problemas multimodales relativamente sencillos (como el UMDAg).

# Conclusiones

- Existen implementaciones simples de Algoritmos de Estimación de distribución para resolver problemas multimodales relativamente sencillos (como el UMDAg).
- Estas implementaciones pueden ser mejoradas tomando ciertas consideraciones:

# Conclusiones

- Existen implementaciones simples de Algoritmos de Estimación de distribución para resolver problemas multimodales relativamente sencillos (como el UMDAg).
- Estas implementaciones pueden ser mejoradas tomando ciertas consideraciones:
  - ▶ Que la probabilidad de las mejoras regiones sea la mas alta (como en la Boltzmann).

# Conclusiones

- Existen implementaciones simples de Algoritmos de Estimación de distribución para resolver problemas multimodales relativamente sencillos (como el UMDAg).
- Estas implementaciones pueden ser mejoradas tomando ciertas consideraciones:
  - ▶ Que la probabilidad de las mejores regiones sea la mas alta (como en la Boltzmann).
  - ▶ Que se pueda asegurar la convergencia al óptimo.

# Conclusiones

- Existen implementaciones simples de Algoritmos de Estimación de distribución para resolver problemas multimodales relativamente sencillos (como el UMDAg).
- Estas implementaciones pueden ser mejoradas tomando ciertas consideraciones:
  - ▶ Que la probabilidad de las mejores regiones sea la mas alta (como en la Boltzmann).
  - ▶ Que se pueda asegurar la convergencia al óptimo.
  - ▶ Que funciones con diferentes características se les explore diferente.

# Conclusiones

- Existen implementaciones simples de Algoritmos de Estimación de distribución para resolver problemas multimodales relativamente sencillos (como el UMDAg).
- Estas implementaciones pueden ser mejoradas tomando ciertas consideraciones:
  - ▶ Que la probabilidad de las mejores regiones sea la mas alta (como en la Boltzmann).
  - ▶ Que se pueda asegurar la convergencia al óptimo.
  - ▶ Que funciones con diferentes características se les explore diferente.
- Ya que se tienen distribuciones de probabilidad se pueden tener ventajas adicionales a otros evolutivos, por ejemplo: saber que regiones son donde mas probablemente se encuentra el óptimo, saber cual es la capacidad de exploración (varianza del algoritmo), en EDAs complejos se infieren correlaciones entre variables que pueden tener sentido físico. Se puede sesgar la búsqueda, insertar varianza, conocimiento a priori, etc. de manera mas natural que en otros algoritmos.

?

Presentación y código proximamente en:  
[www.cimat.mx/~ivvan/EMMN2015.zip](http://www.cimat.mx/~ivvan/EMMN2015.zip)