Edgar Joel Arévalo Chavarín 215861967

Seminario de Solución de Problemas de InteligenciaArtificial I

Practica 2

Introduccion:

La búsqueda aleatoria combina el algoritmo unidimensional de Metrópolis-Hastings con el multidimensional muestreador de Gibbs, de tal manera que el nivel de ruido se puede controlar adaptativamente de acuerdo al panorama de la función. Existe un buen equilibrio entre la exploración y la explotación en todo el espacio de búsqueda. Una estrategia de búsqueda local puede acoplarse a la búsqueda aleatoria con el fin de intensificar en las regiones prometedoras.

Es una técnica que permite establecer una ecuación en forma de una línea recta. Esta nos permite hacer estimaciones acerca del valor que asumirá la variable dependiente según el valor que tome el valor independiente.

Dando una ecuación de una recta (y = m + bx) para poder predecir valores de algún problema.

Esto funcionan dando valores a la m y b, para ver cómo se van acercando a los otros puntos que tengamos en la gráfica.

Para poder establecer un valor m y b que se acerque, necesitamos una función de costo.

Lo que busca el gradiente descendente es encontrar un mínimo de la función, ya sea local o no.

Para saber cuando un valor se acerca más que otro es tomar un valor y restarle la derivada de la función en ese valor multiplicado por un alpha, que es un número de cuando se irá desplazando.

Desarrollo:

Para la implementación del algoritmo de la busqueda aleatoria se opto por realizar el siguiente algoritmo:

def aleatorySearch(excercise, limit_inf, limit_sup, start = 0, ended = 50)

```
  #this is equal to Infinite  when you make a comparation, this is always biger
  f_best = +1.0/0.0
  x_best = y_best = 0
  start.step(ended) do |i|
    xi = limit_inf + ( (limit_sup - limit_inf) * Random.rand(0.0..1.0) )
    yi = limit_inf + ( (limit_sup - limit_inf) * Random.rand(0.0..1.0) )
    if excercise == 2
      f_val = f(xi, 2, excercise)
    else
      f_val = f(xi, yi, excercise)
    end
    if f_val < f_best
      f_best = f_val
      x_best = xi
      y_best = yi
    end
  end
  if excercise == 2
    min = [x_best]
  else
    min = [x_best, y_best]
  end
  {minimo: min, costo: f_best }
end
```

Mientras que para el algoritmo de Gradiente Descendiente se opto por separarlo en 2D y 3D de las siguientes maneras:

3D

```
def gradient3Dim(limit_iterations, init_x = 0, init_y = 0)
```

```
  puts "h"
  h = Random.rand(0.0..0.01)
  puts h
  puts "dx and dy"
  dx = Random.rand(0.0..0.015)
  puts dx

  x = init_x
  y = init_y
  (0..limit_iterations).map do |i|
    x = x - (h * ( f(x + (dx / 2), y, 1) - f(x - (dx / 2), y, 1) ) / dx )
    y = y - (h * ( f(x, y + (dx / 2), 1) - f(x, y - (dx / 2), 1) ) / dx )
  end
  {minimo: [x, y], costo: f(x, y, 1) }
end
```

2D

```
def gradient2Dim(limit_iterations)
  puts "h"
  h = Random.rand(0.0..0.5)
  puts h
  puts "dx"
  dx = Random.rand(0.0..0.5)
  puts dx

  xi = 0.7
  (0..limit_iterations).map do |i|
    inc = f(xi + (dx / 2), 2, 2)
```

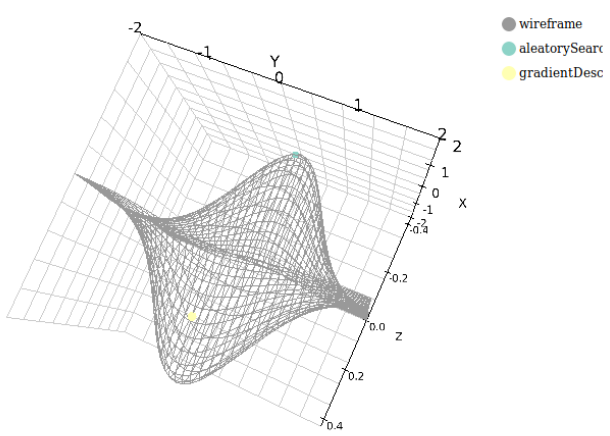
```
    dec = f(xi - (dx / 2), 2, 2)
    grad_x = ( inc - dec ) / dx
    xi -= (h * grad_x)
  end
  {minimo: [xi], costo: f(xi, 2, 2) }
end
```

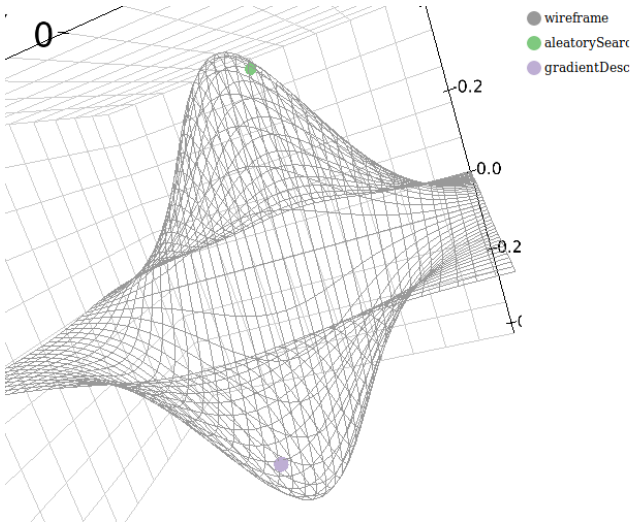
Para realizarlos bastaba con simplemente seguir el algoritmo.

Resultados:

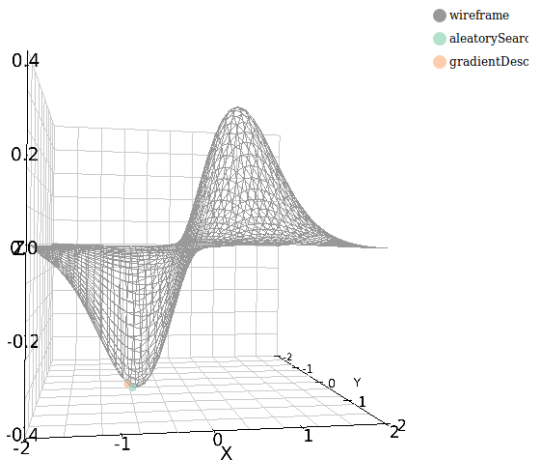
$f(x,y)=x\,e^{-x^2-y^2}; x,y\in[-2,2]$

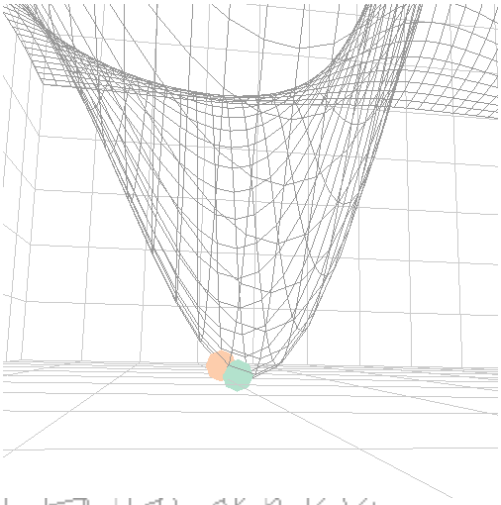
Busqueda aleatoria	Gradiente desendiente
Minimo:[-0.7344943281829575, -0.09432600047641326] costo: -0.4244537130583126	Minimo=>[1.3325299859682438, 0.0] costo=>0.22569815197589285
Minimo=>[-0.8936806247340394, 0.23724243131541245] costo=>-0.38008672032993945	Minimo=>[1.1801110084397113, 0.0] costo=>0.29315509274366447
Minimo=>[-0.7585901263298167, 0.036837382342350455] costo=>-0.42608789062900954	Minimo=>[-0.8188903719102716, 0.0] costo=>-0.4187896552059087
Minimo=>[-0.7376635481939049, 0.118844294695168] costo=>-0.42208906538266494	Minimo=>[-0.8072474316850297, 0.0] costo=>-0.42072600911487334

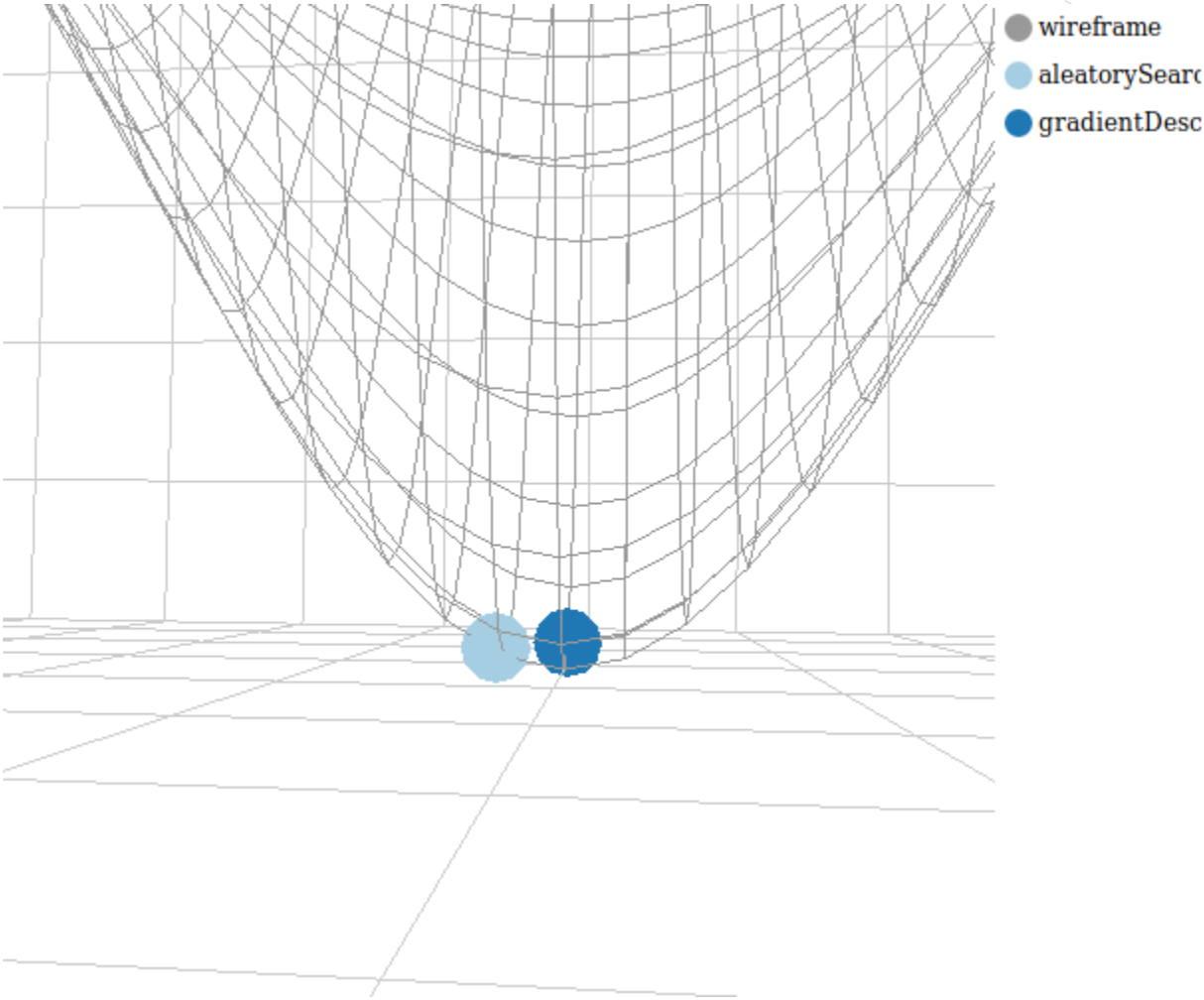




Edgar Joel Arévalo Chavarín 215861967



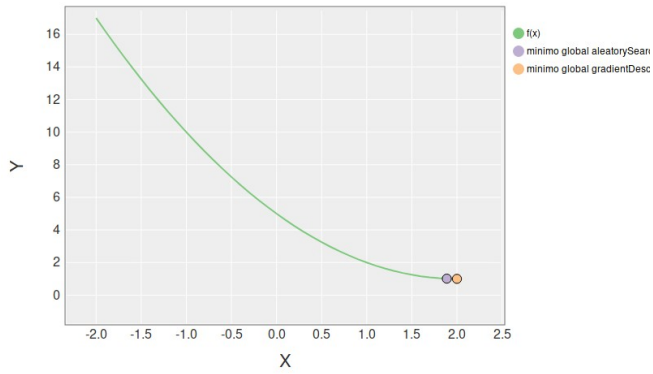


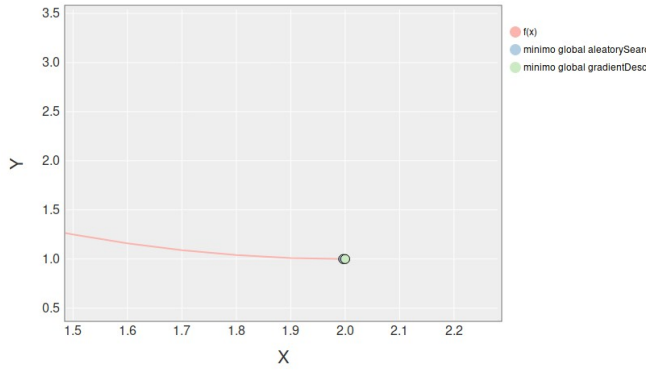


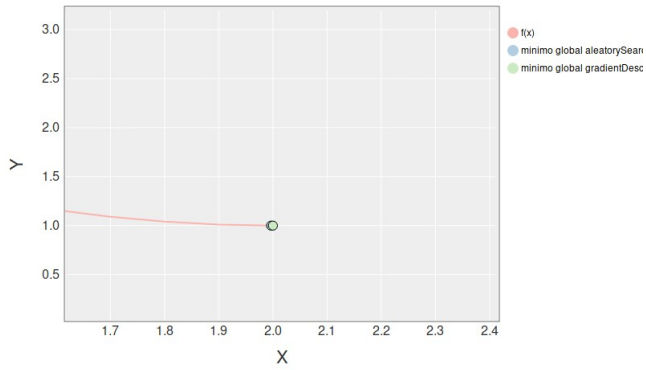
Edgar Joel Arévalo Chavarín 215861967

$$f(x)=\sum_{i=1}^{d}(x_i-2)^2, d=2$$

Busqueda aleatoria	Gradiente desendiente
Minimo=>[1.8879453862112414] costo=>1.0125562364713478	Minimo=>[1.9999999999999996] costo=>1.0
Minimo=>[1.9963759199997995] costo=>1.0000131339558478	Minimo=>[2.0] costo=>1.0
Minimo=>[1.9917909417837847] costo=>1.0000673886367972}	Minimo=>[1.9999999999999998] costo=>1.0







Conclución

¿Qué método es mejor? ¿Por qué?

El mejor algoritmo es el gradiente decendiente en cuanto a certeza se refiere, siempre y cuando se defina bien el punto de inicio, lo que requiere revisar la grafica en primera insatancia, sin embargo si el problema es de 2D este no tendra muchos problemas, y llegara casi de inmediato al resultado. De igualmanera si se coloca mal, no quiere decir que sea malo porque no encontro exactamente el punto que deseabamos, solo que encontro el mas apto según la pendiente en la que inicio.

Mientras que el algoritmo de busqueda aleatoria, por defecto se tiene que especificar el rango de la funcion en los que se realizara la busqueda, esto indica que forzosamente se debe de ver la grafica,

aunque de igual manera, si se definen unos limites, este encontrara el mejor resultado cercano a estos, sin embargo es cuestion de probabilidad e incluzo suerte que este nos de exactamente la mejor solución cosa que el algoritmo de gradiente descendiente si es mas capaz en ello.