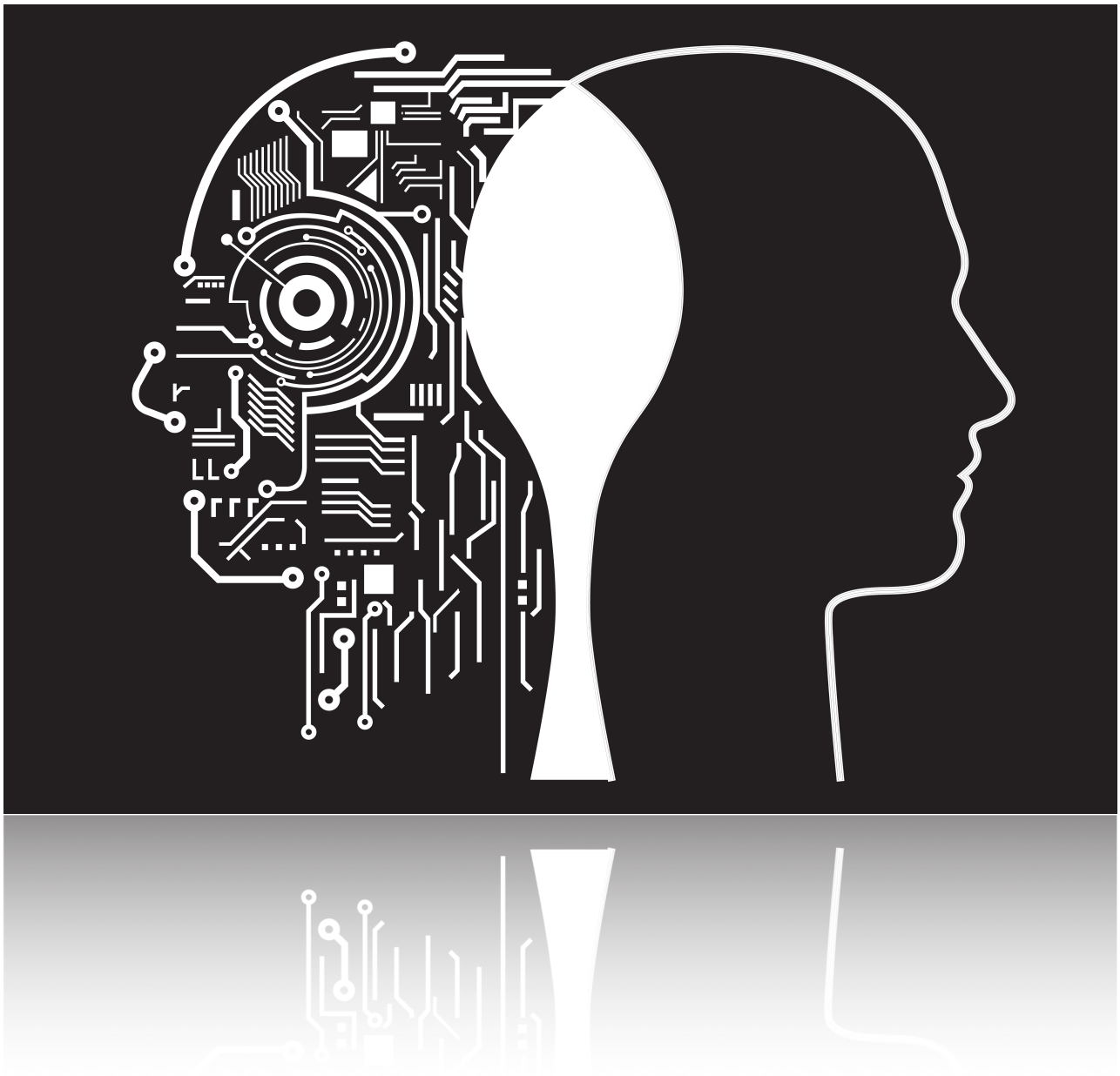
Function Maximization using SA Algorithm

Elements of Artificial Intelligence

Coursework I

Caetán Tojeiro Carpente

17 de noviembre de 2016



INDEX

Introduction

1 - What is Simulated Annealing (SA)?

Simulated Annealing is a probabilistic technique that allow us to approach the global optimum value of a given function.

2 - Origin

The idea comes from a published by *Metropolis* in 1953. In this paper appeared the process called annealing. The algorithm simulated the cooling of material in a heat bath.
Several years later, in 1982, Kirkpatrick took this idea and used it to optimization problems. This concept consists in use SA to search for feasible solutions and converge to an optimal solution.

Coursework

1 - Algorithm to implement

The algorithm which will be implemented is SA. To having a better view of the code is presented the following pseudocode[1]:

For t = 1 **to** ∞ **do** T = Schedule[t]

If T = 0 **then return** Current

Next = a randomly selected successor of Current

δc = c[Next] – c[Current]

if δc > 0 **then** Current = Next

else Current = Next only with probability exp(δc/ T)

[1] This pseudocode is taken of the slides of the subject, but changing the last expression **exp(-δc/ T)** for **exp(δc/ T)**

2 - Python code

The programming language that is used for this task is *Python*. The choice of this language is motivated because of the easiness that *Python* gives us to this kind of matters.

```
import random, numpy, math
import matplotlib.pyplot as plt


print "Introduce betta parameter (range[0,1])"
betta=input()

print "Introduce alpha parameter (range[0,1])"
alpha=input()

print "Introduce initial temperature (T)"
t_inicial=input()

def sim_anneal():
      y=0
      t_anterior=t_inicial/alpha #Multiply by alpha to start at initial
temperature
      p_actual=0
      while True:
            T=funcion_Temperatura(t_anterior)
            if T<=0.01 :
                  return p_actual,y
            t_anterior=T
            p_seguinte=funcion_vecinho(p_actual)
            custo=funcion_custo(p_seguinte)-funcion_custo(p_actual)
            if custo>0:
                  p_actual=p_seguinte
            else:
                  if (math.exp((custo)/T)>random.uniform(0,1)):
                        p_actual=p_seguinte


            y=funcion_custo(p_actual)
            print y

            plt.plot(p_actual, y, 'b^')

def funcion_Temperatura(temp):
      t=betta*temp  #Change the formula
      return t

def funcion_vecinho(p_actual):
      vecinho=random.uniform((-alpha*40)+p_actual,(alpha*40)+p_actual)
      if vecinho<0 or vecinho >40:
```

```
            return funcion_vecinho(p_actual)
      return vecinho

def funcion_custo(x):
      y=numpy.sin(0.15*x)+numpy.cos(x)
      return y


print sim_anneal()

plt.show()
```

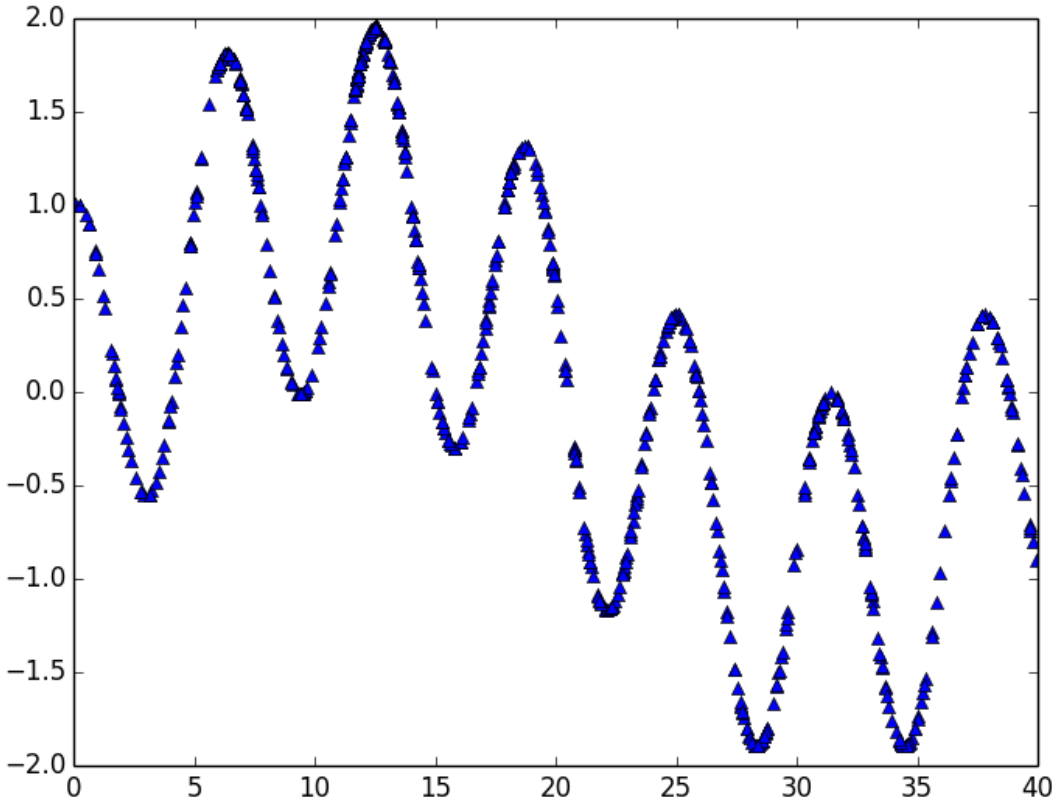
3 - Notes

- This code is only valid for this exercise, is not for general purpose.
- The user can choose β, T and α values introducing these by keyword.
- The decreasing temperature expression can be only changed by changing the code.

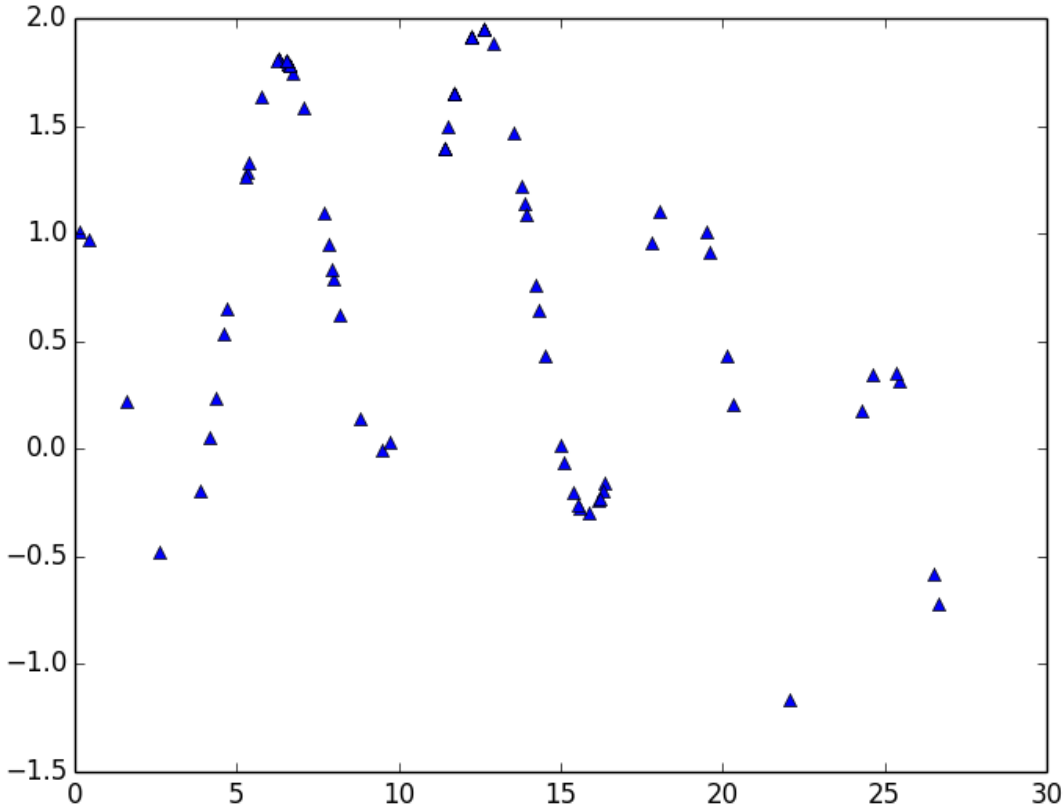
Variables choice

1 - Betta parameter (β)

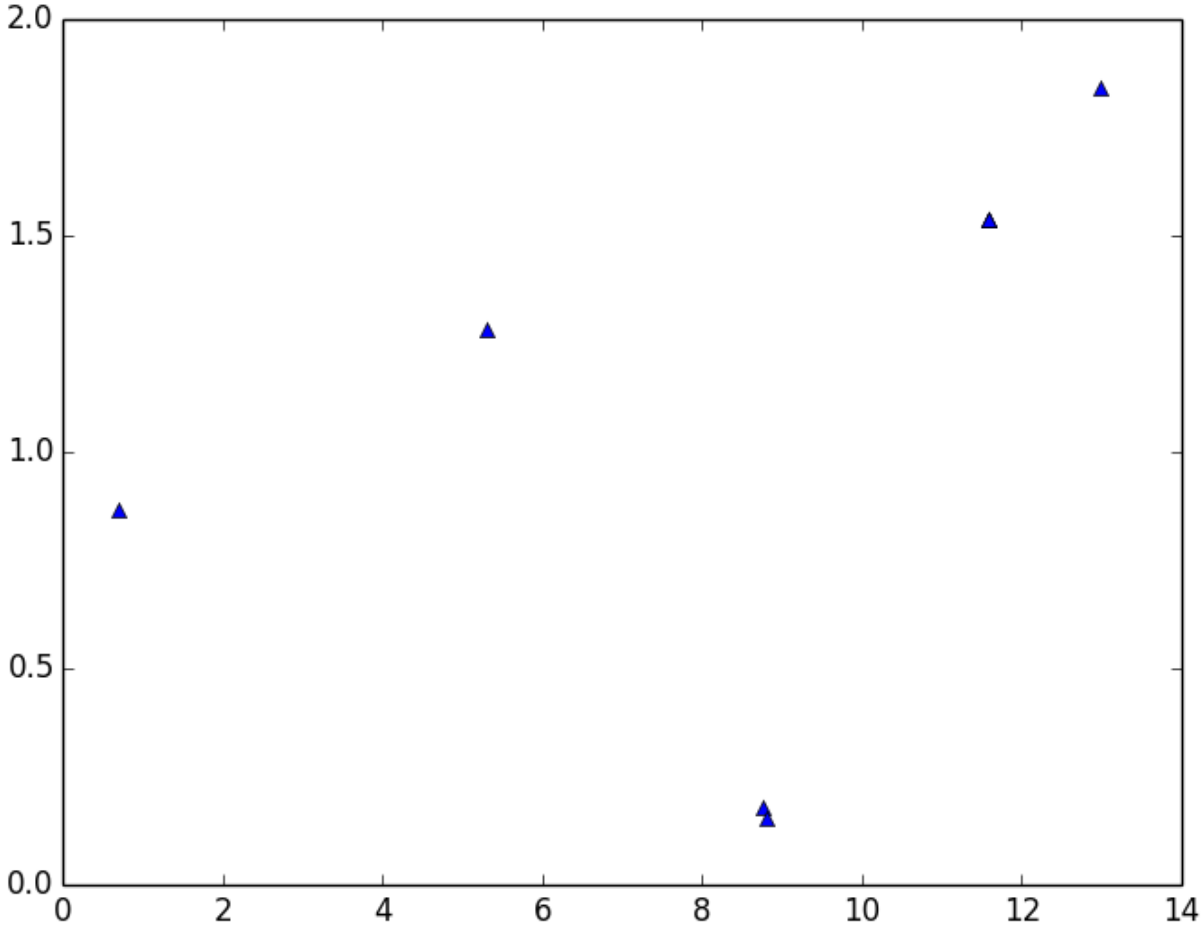
With this parameter you change the way of decrease temperature (T). The slower it is, the more samples are taken. It means more iterations of the algorithm, so it is more probable to get a x value that give us a good approach to the maximum value of the function y given.



Example for β=0.99 α=0.3 T=100



Example for β=0.7 α=0.3 T=100

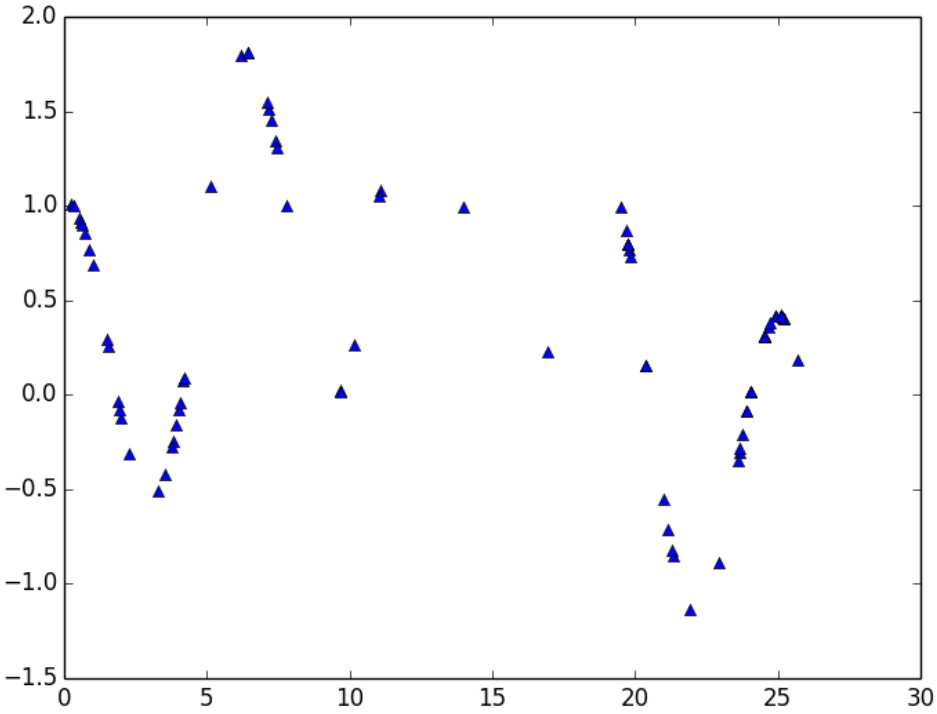


Example for β=0.3 α=0.3 T=100

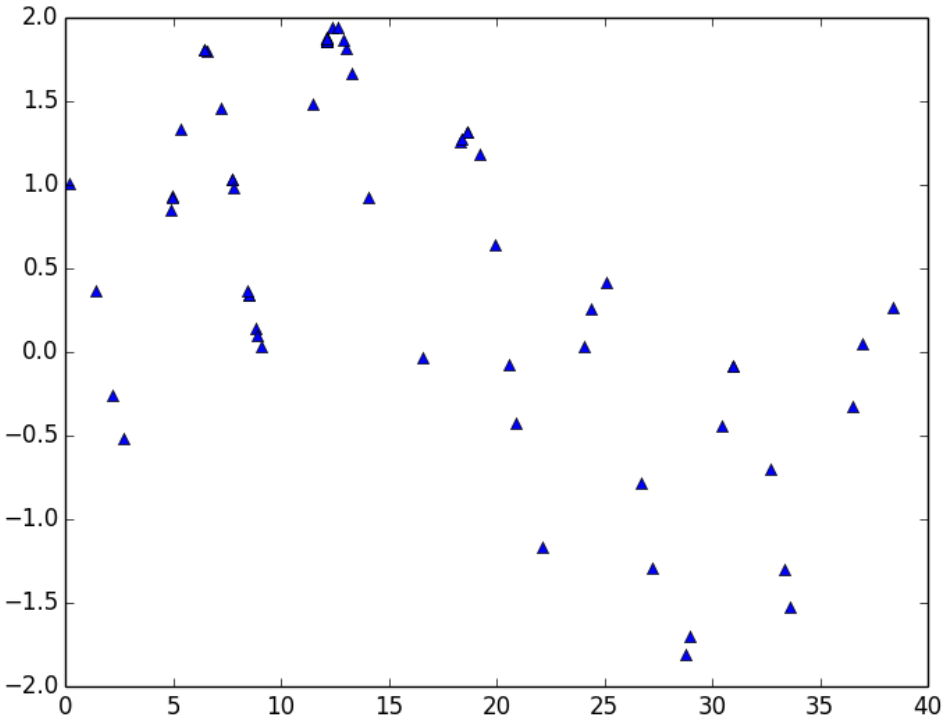
As we can see, if we do low the beta value (β) we have less samples and it is less probable to get a value near the maximum.

2 - Alpha parameter (α)

Changing alpha parameter (α) we are changing the neighborhood function. For low values of α we are taking a small range of possibles neighbors of the current point, so mostly we are going to get so close samples. On the other hand, if we select a high value of α we will take most of the range, so the algorithm lose some sense.



Example for β=0.9 α=0.1 T=100

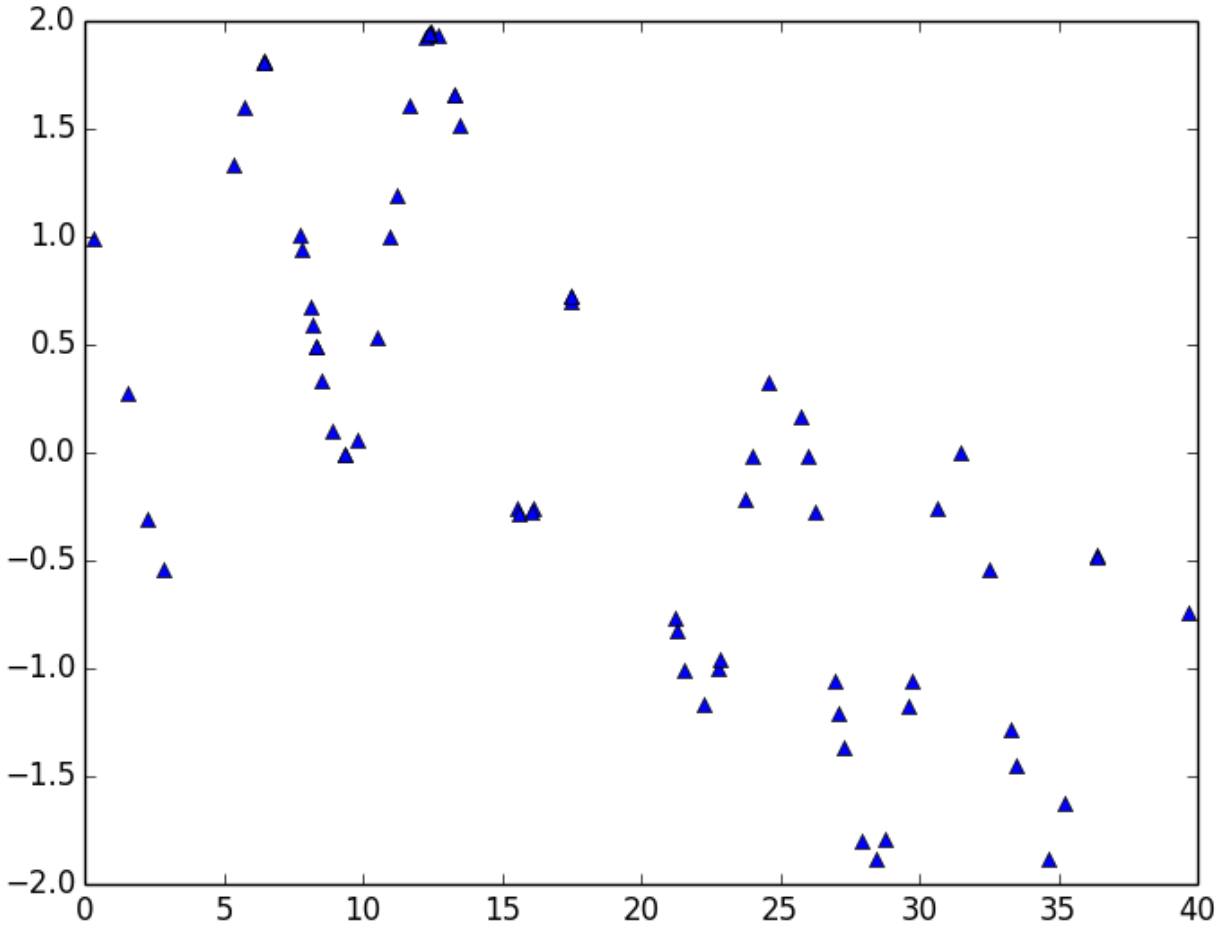


Example for β=0.9 α=0.8 T=100

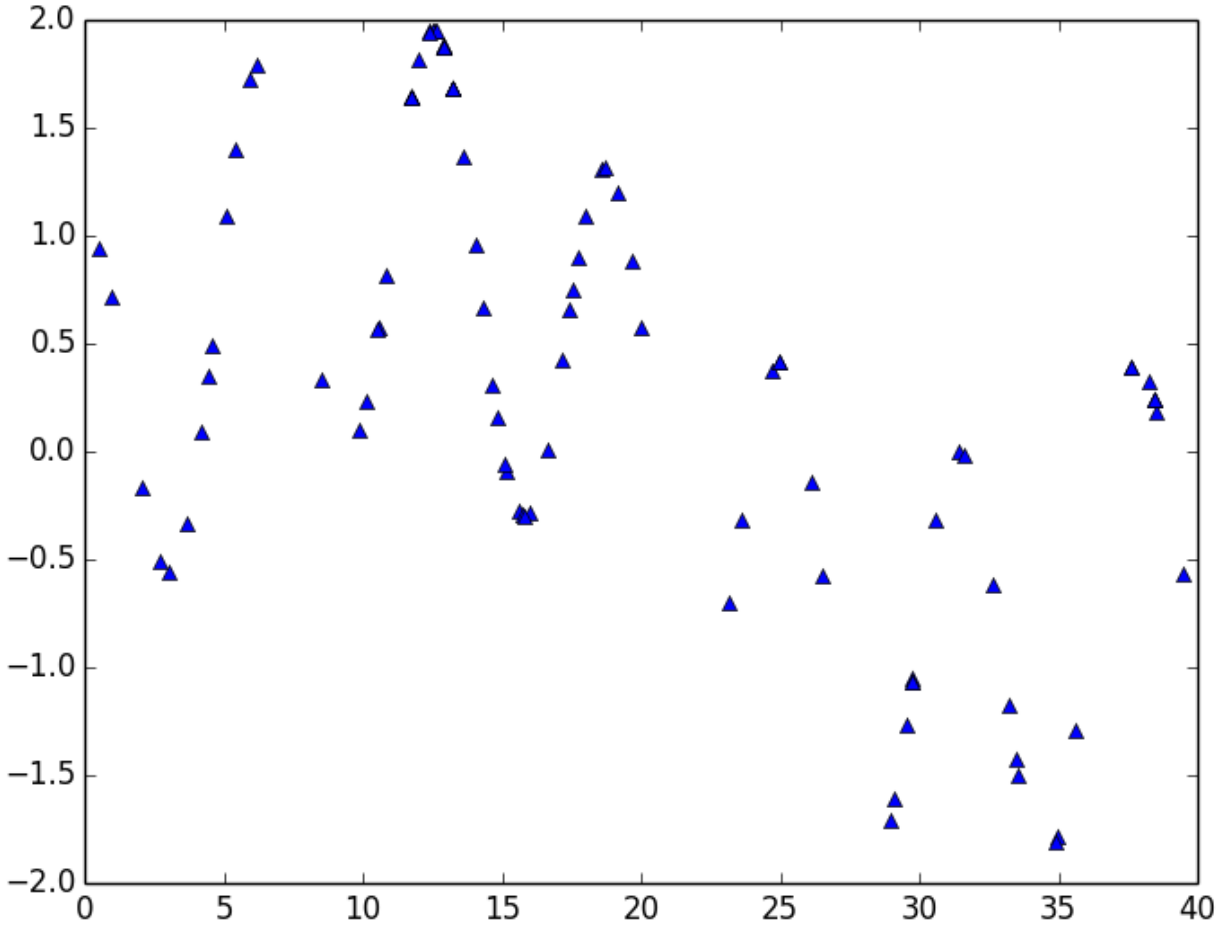
As we can see, changing α changes the distribution of the samples, but it doesn't affect almost nothing to get a good approach of maximum y.

3 - Temperature parameter (T)

We have to take care with this parameter. On one hand, if we select an small value we will have a small solution space so probably we will not get a good value for x and y. On the other hand, if we perform the algorithm with a high value of T, we will waste a lot of computational time and resources. Of course, everything of this depends on β parameter.



Example for β=0.9 α=0.4 T=300



Example for β=0.9 α=0.4 T=1000

We can see that there are hardly any difference between one graphic an the other.

4 - Notes

- We must take into account that this graphics are only simulations. Probably, if we run the algorithm again with the same values for β, T and α the graphic resultant will be different.

- All graphics shown here have been performed with the same math formula:

$$T = \beta * old\_temperature$$

- Changing the formula we will get different results. Mostly it will depends on how the decrement function behaves.

Conclusion

It can finish this task by saying that the most important parameter to work with is β. In other words, the expression to change the temperature (T) is with which you have to deal.

In addition, we can say that values of three parameters is a complex decision to take. Have an small solution space is good in terms of memory and time computation, but we have to be careful with this, because it can cause a detriment of the quality solution. We must to find a balanced solution.