Math 260: Simulated Annealing

Max Wang

Nov. 24, 2020

1 Code Description

Due to the close connections between the procedures in the simulated annealing method, the entire functionality of the project is included in the single method `sim_anneal`. The parameters and the returns of `sim_anneal` are included in Tables 1 and 2, respectively.

Parameter	Meaning	Comment
`f`	1D function to locate the minimum for	Required
`x0`	Initial independent variable	Required
`T0`	Initial temperature	Required
`r_T`	Rate of temperature decrease	Default = 0.95
`x_iterations`	Number of accepted moves for a given step size	Default = 10
`step_iterations`	Number of step sizes for a given temperature	Default = 10
`tol`	Tolerance to determine steady-state condition	Default = 10e-5

Table 1: Parameters of `sim_anneal`

Return Variable	Meaning
`T_vals`	A list of `[T_k, x_opt_k, f_opt_k]` for all k-th temperatures
`x_vals`	All accepted moves
`f_vals`	Function values for all accepted moves
`x_opt`	`x` of the located minimum
`f_opt`	Function value of the located minimum
`step_sizes`	All step sizes taken in the process

Table 2: Returns of `sim_anneal`

Three loops (2 while-loops and 1 for-loop) are used to mimic the process of simulated annealing. In each iteration of the out while loop, temperature is fixed. For each iteration, an optimal x and $f(x)$ pair is determined for the given temperature. This loop also checks whether a steady state has been reached. In this case, a steady-state condition is met when the minimum function values for four successive temperatures differ from each other no more than the specified tolerance. In other words, updating the temperature will no longer result in substantial improvement.

The goal within each iteration of the outer while loop is to iteratively take random steps to minimize the function value. To do so, a for-loop controls the maximum sizes of the random steps to take. In each iteration of the for-loop, the maximum step size is the same. Between iterations,

the maximum step sizes are adjusted with the goal of equating the number of accepted moves and rejected moves.

The innermost while loop is where the random moves actually take place. Within each iteration, a random step is generated according to the fixed maximum step size. New x- and f-values are then generated, from which a change in f-values can be calculated. Now, if the change is negative, i.e. in the desired, this random step is automatically accepted. Otherwise, a Metropolis step is taken that compares a randomly generated number with a given probability, with the latter dependent on the fixed temperature. Therefore, there exists the possibility that a worse step is nonetheless accepted. A pseudo-code for the algorithm is included in Algorithm 1

Algorithm 1 Simulated Annealing

```
while not in steady state do
    for i = 0, 1, ··· step_iterations do
        while x_trial < x_iterations do:
            x' ← x_current + random step
            if f(x') < f(x) then
                f_opt = f(x')
                f_current = f(x')
            else
                if p < exp((f(x)−f(x'))/T) then
                    f_current = f(x')
                end if
            end if
        end while
        step size ← modified step size
    end for
    T ← modified T
    check steady-state condition
end while
```

One major challenge in implementing the algorithm is keeping track of what the optimum is and what we are working with. In the case of simulated annealing, these two quantities need not be the same, unlike in other optimization algorithms such as gradient descent. The principle is as follows. For each new temperature, the starting x- and f-values are always those of the optimum that we now have. Within a fixed temperature, however, x- and f-values are built upon the most recent accepted moves, which may not be the optimum if the previous accept move is the result of a successful Metropolis step.

To address this problem, variables `T_vals` and `x_vals` are used to differentiate the optimum from the working moves. Specifically, each item of the list `T_vals` is another list `[T_k, x_opt_k, f_opt_k]`, in which `x_opt_k` and `f_opt_k` are the optimal x- and f-values that are known up until `T_k`. In this way, when a new temperature $\texttt{T}_{\texttt{k+1}}$ is up, it is easy to retrieve the current optimum as the starting point. On the other hand, `x_vals` is a list of all the accepted moves regardless of temperature. Therefore, the latest addition to `x_vals` is always the x-value we are working with. For this reason, the list `x_vals` does need to be artificially appended with the current optimal x-value at the transition to a new temperature.

2 Discussion

Simulated annealing has a range of applications. In this project, the focus is to find the global minimum of a continuous, single-variable function $f(x)$. A major advantage of the simulated annealing method is its ability to bypass certain barriers in the search process and not get stuck in a local minimum. The random steps, as introduced before, are responsible for this feature. Here, two test functions are analyzed with the simulated annealing method, and the results shed light on both the power and limitations of this approach.

2.1 A simpler case

Function 1 is defined as:

$$f_1(x) = \frac{(x-13)(x-8)(x+10)(x-1)}{1000}$$

The plot of the function is shown in Figure 1 in red curve. Two local minima are visible. Also shown in the plot are the results of 50 trials of the simulated annealing algorithm, represented with black dots. The method successfully identifies the global minimum of the function in all 50 runs, demonstrating satisfactory robustness. More importantly, the starting location x_0 has been carefully placed around the unwanted local minimum at 12. Therefore, simulated annealing does have the potential to escape local barriers.

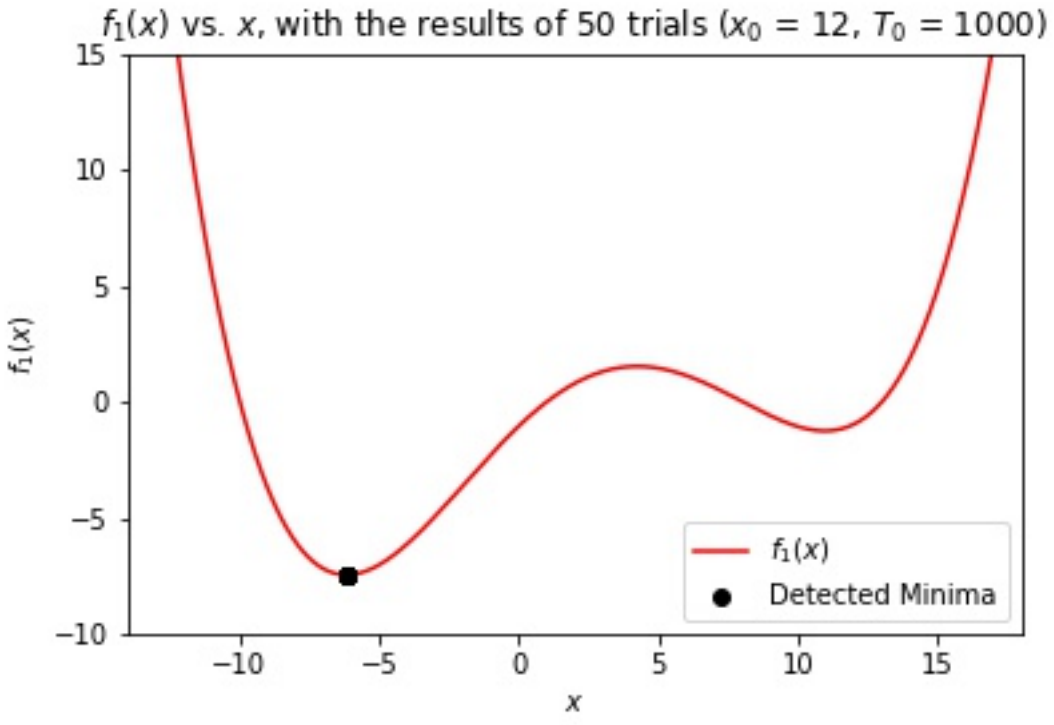

Figure 1: Test Function One: Plot and Results

For this test scenario, failures are almost never observed except when T_0 falls below 0.1. However, there is no reason to pick such small of a starting temperature unless the purpose is to break the code. After all, in a real annealing process, an extremely high temperature is the premise, so the simulated annealing method needs to respect this practice. Mathematically, temperature

matters because the Metropolis probability is $\exp(-\frac{df}{T})$, where df is the change in f-value between successive moves. Clearly, for a fixed df, a larger T will result in a higher probability of a bad move being accepted, thus a higher power to escape the local barrier.

Figure 2 provides one indirect way to visualize the randomness involved in the simulated annealing method. In the plot, the development of maximum step sizes within one iteration of the fixed-temperature while-loop is shown. Striking features of this plot include the large noise at the beginning and the continuously decreasing trend of the maximum step sizes. To explain these observations, we need to understand how the step sizes are being generated.

Recall that changing the step size requires comparing the number of accepted and rejected moves. If there are too many accepted moves, the Metropolis probability $\exp(-\frac{df}{T})$ is too high. For a fixed T, this translates to too small of a df and in turn too small of a dx. Therefore, large acceptance rate calls for larger step size, and vice versa. In the plot, the fact that maximum step size soars at the beginning is simply the result of a large T and a small df (recall that $x0$ is at a relatively flat area on the curve). As larger steps are taken, however, we end up in the two sides of the curve with extremely large slopes. The upshot is an extremely large df that mitigates the effect of the large T, thus reducing the step sizes. In this sense, while the simulated annealing process is random in nature, it is actually bounded in terms of how far-off it can go. The Metropolis step and the adjustable step sizes grant the method a self-correcting mechanism.

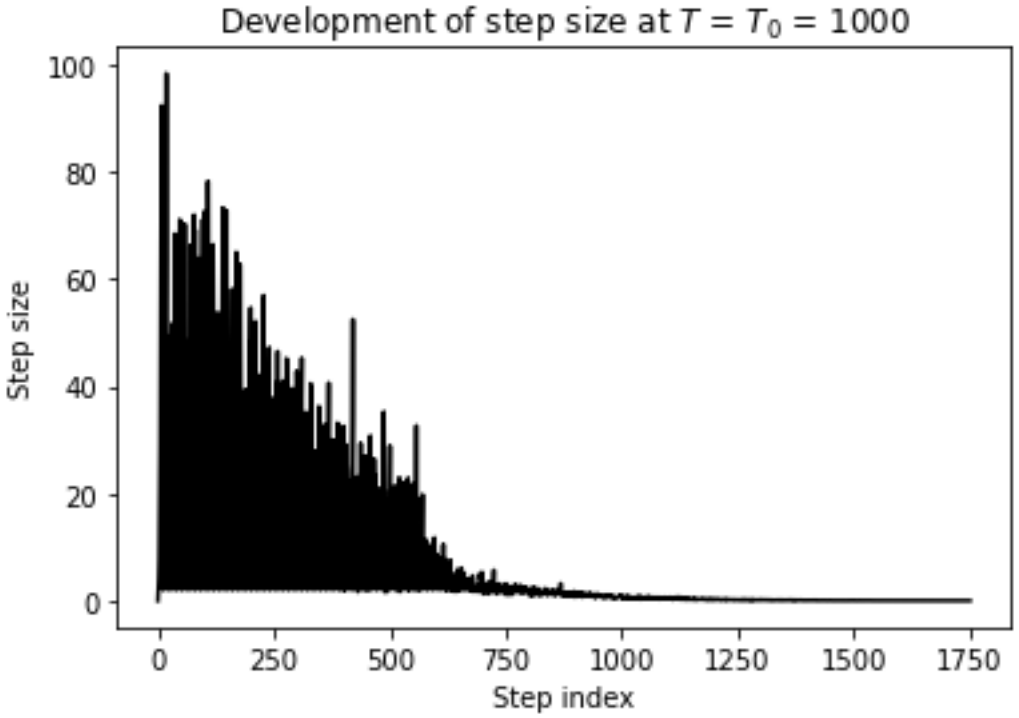

Figure 2: Test Function One: Step Size Progression

2.2 A more complex case

Function 2 is defined as:

$$f_2(x) = \frac{(x-1)(x-2)(x-3.05)x(x+1)(x+2)(x+3)(x+4)}{200}$$

The plot and the results for 50 trials are again shown in Figure 3.

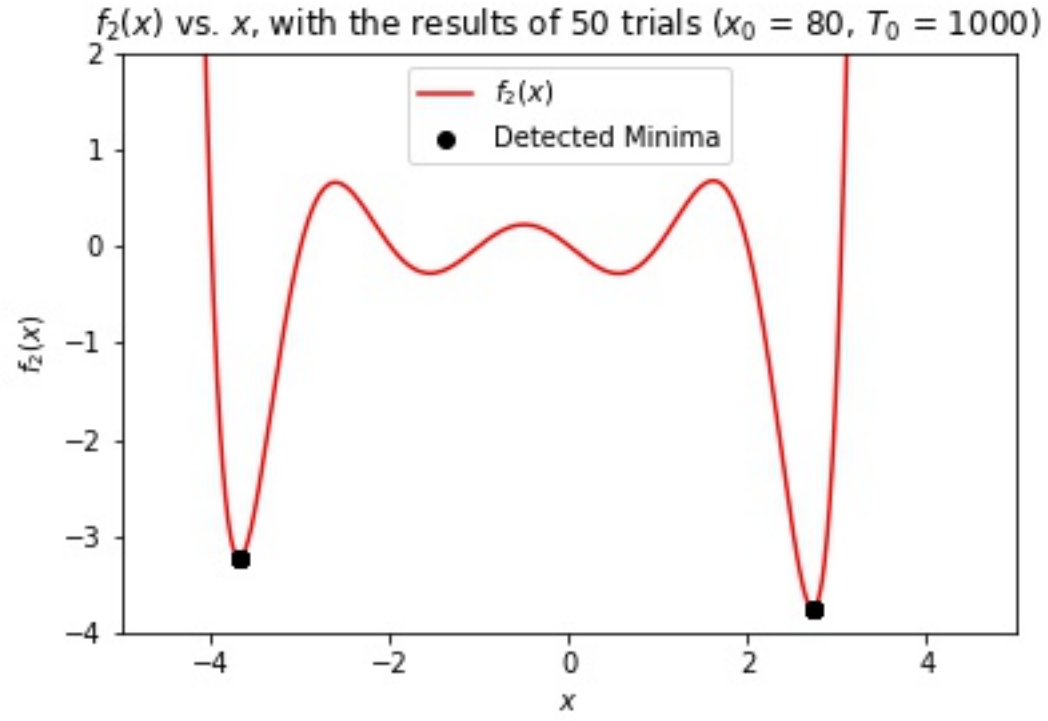

Figure 3: Test Function One: Plot and Results

In this case, there are failure cases in which the local minimum is identified as the final result, and no combination of the parameters has so far given a clean, correct result. The likely cause is the extreme similarity between the two local minima, as indicated by the quasi-symmetric shape of the curve. Detailed distribution of the results is include in Table **??**. It can be seen that a dominating percentage of trials are in fact successful, though the error can be difficult to eliminate in a short amount of time.

Location	Local Minimum (-3.69)	Global Minimum (2.79)
Counts	8	42
Percentage	16%	84 %

Table 3: Distribution of the 50 Trials for Test Function Two

The second test scenario invites a different way to view the simulated annealing method. While it might not always able to return the correct global optimum, it also almost never misses it. Out of all the combinations of parameters tested, the global minimum has always been detected. This sheds light on a potential way to use the simulated annealing algorithm without bothering too

much with fine-tuning all the parameters. Namely, a large number of simulated annealing trials can be performed, all at large initial temperatures, and the minimum value of all the trial results can be regarded as the global minimum. Of course, more research needs to be done to prove this modified approach, but simulated annealing, as an effective optimization method, has proven itself without doubt.