Written by Yarden Nativ, mail: nativyardenac@gmail.com

7th Recitation 11.5.23

Optimization

Optimization

While discussing estimators, we were interested in finding parameters of a model by maximizing the likelihood or minimizing the cost function. When searching for an extremum we can either use brute force strategy by searching the best n parameters from a given n-dimensional grid. Alongside the high computational cost of a brute force strategy, brute force is highly dependent on the range of the parameters for each n.

In optimization we are interested in finding the conditions that give the maximum and minimum of a given function. Mathematically: Minimizing a function $f_0(x)$ subject to the constraints $g_i(x) \leq b_i, i = 1, \dots, m$ ($f_0: \mathbb{R}^n \to \mathbb{R}, g_i: \mathbb{R}^n \to \mathbb{R}$). By finding the minima we also find the maxima of an equivalent function:

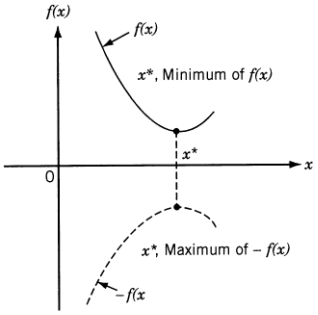


Figure 1.1 Minimum of $f(x)$ is same as maximum of $-f(x)$.

Generally, we are interested in finding the global optimum which represent best possible solution, and not the local optima which are better solutions than their neighborhood:

Class Discussion: Explain some basic ideas for optimization algorithms you are familiar with.

Written by Yarden Nativ, mail: nativyardenac@gmail.com

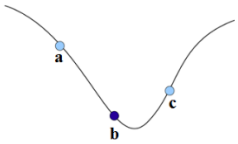
Line search methods:

For each variable in the function, optimize the function with respect to a single variable: $\frac{\partial f(x,y)}{\partial x}$ by using the expanded Fermat's theorem by which in extrema both the partial and the full derivatives are zero. The line search methods assume that f is unimodal in $[a,b]$ so that the minimum x^* lies inside. The line searches reduce the interval $[a,b]$ so that the minimum is still included within. An approximation of the minimum is found when the length of the interval is smaller than a pre-determined tolerance: $x^* \in [a^*,b^*]$ where $|b^* - a^*| < \epsilon$. A major advantage of line search methods is that differentiability is not essential for the calculations.

Golden search:

Initial bracketing interval:

- Consider to x values: $a < b$ when $f(a) > f(b)$
- Take successively larger steps beyond b until function starts increasing. This will result 3 values: $a < b < c$ when $f(a) > f(b)$ and $f(c) > f(b)$



Iterative step:

- If $[a,b] < [b,c]$:
  - Set $x = b + 0.38197 \cdot (c - b)$

  Else:

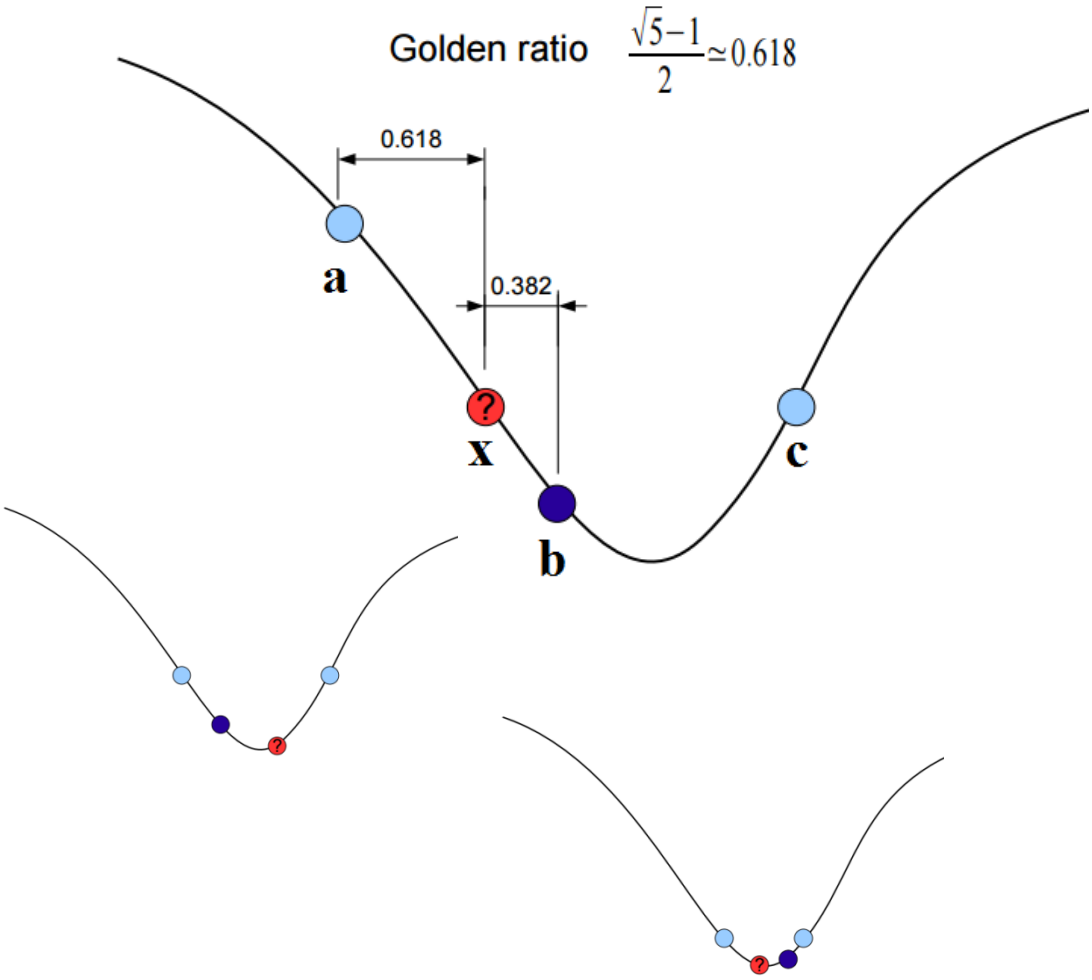
  - Set $x = b - 0.38197 \cdot (b - a)$
- Redefine a, b, c such that:
  - $a < b < c$
  - $f(b) < f(a)$ and $f(b) < f(c)$

Iterate until $|c - a| < \epsilon$

Class Discussion:

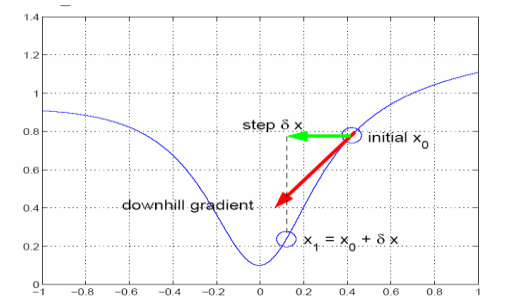
- For what kinds of function this method is less suitable?
- What are the method advantages and what are the disadvantages?

Written by Yarden Nativ, mail: nativyardenac@gmail.com



Gradient Descent:

Given a starting location x_0, examine $\frac{df}{dx}$ (or ∇f for multivariate) and move in the downhill direction to generate a new estimation $x_1 = x_0 + \delta x$:



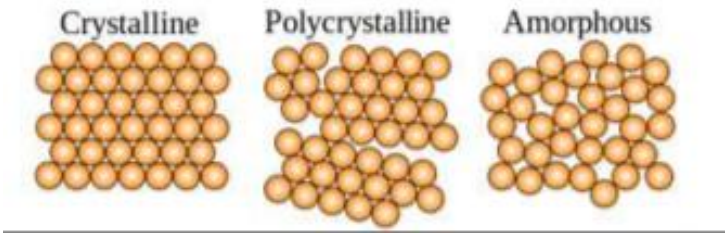
Class Discussion:

- For what kinds of function this method is less suitable?
- What are the method advantages and what are the disadvantages?

Written by Yarden Nativ, mail: nativyardenac@gmail.com

Simulated Annealing:

Annealing is a thermal process for obtaining low energy states of a solid in a heat bath from the field of crystallization. The process begins by increasing the temperature of the heat bath to a maximum value at which the solid melts. The next step is carefully decreasing the temperature of the heat bath until the particles arrange themselves in the ground state of the solid. Ground state is a minimum energy state of the solid. The ground state of the solid is obtained only if the maximum temperature is high enough and the cooling is done slowly.



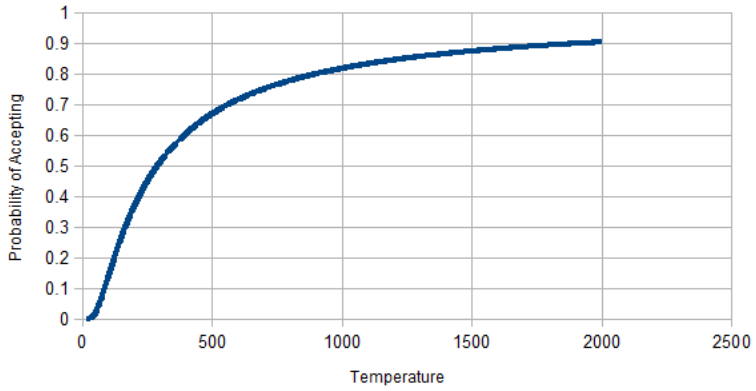
The method of **simulated annealing** is based on the same idea by following two equivalent steps. First, a successor function will return a “close” neighboring solution given the actual one. This will work as the “disturbance” for the particles of the system. Next, a target function will optimize depending on the current state of the system. This function will work as the energy of the system.

The search begins with a randomized value (state of the system). Using a loop, the successor functions move to neighboring values (states of the system) accepting the moves that decrease the energy. Bad moves are defined by a probability distribution which represents the “temperature of the system”. The higher the temperature, the more “bad moves” are accepted.

Decreasing the temperature slowly reduces the number of “accepted” bad moves until at very low temperatures the algorithm becomes a greedy hill-climbing algorithm, finding the best move within all possible moves.

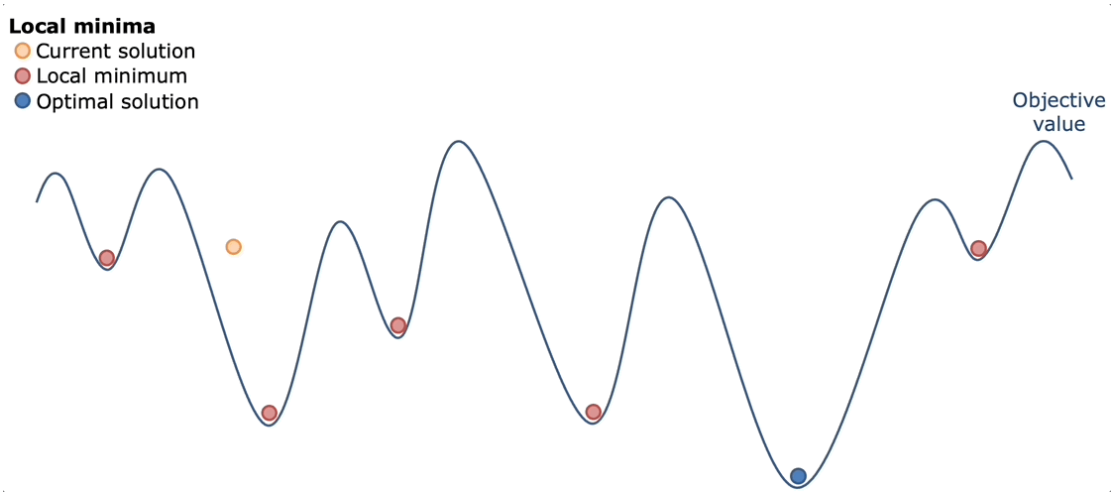
A common distribution from solid physics is used to decide if a bad move is accepted is the Boltzman distribution: $p(\gamma) = \frac{e^{-\frac{E_\gamma}{T}}}{Z(T)}$ $Z(T) = \sum_{\gamma'} e^{-\frac{E_{\gamma'}}{T}}$ when γ represents the current state of the system, E_γ the energy related with it and Z the normalization constant:

Written by Yarden Nativ, mail: nativyardenac@gmail.com

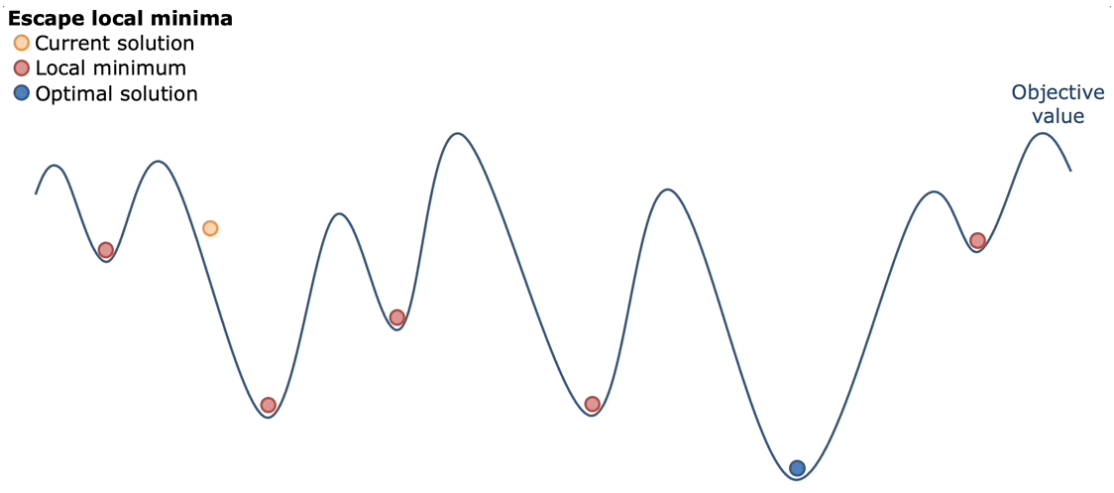


Demonstration:

Local search:



Simulated Annealing:



https://towardsdatascience.com/an-introduction-to-a-powerful-optimization-technique-simulated-annealing-87fd1e3676dd

Written by Yarden Nativ, mail: nativyardenac@gmail.com

Pseudo-Code example (link):

```
def accept(change, T):
    # Return boolean with probability e^(change/T)
    # (change / T) ~ 0 early on, increasing over time and rejecting
negative moves later on.

def simulatedAnnealing(state,temperatureSchedule):
    current = state
    for (t = 1;; t++):
        T = temperature(t) # Tempreature at time t
        if T = 0
            return current
        next = generateNewState();
        change = next.value - current.value
        if change > 0
            current = next
        else
            current = accept(change, T) ? next : current
```

Written by Yarden Nativ, mail: nativyardenac@gmail.com

Class Exercises for self-learning:

Golden Search

The file gs_data.mat contains a set of 20 observations drawn from a mixture of two normal distributions.

Assuming that the two distributions have unit variance and symmetric means μ. And -μ., the likelihood function for these data is:

$$L(\mu) = \prod_i \frac{1}{\sqrt{2\pi}}\left[e^{-\frac{1}{2}(x_i-\mu)^2} + e^{-\frac{1}{2}(x_i+\mu)^2}\right]$$

(The product should be calculated over all observations)

a) Bracket the maximum of the function. What are your initial values of the points a,b,c ?

b) Using the golden-search optimization strategy find the MLE for μ.

c) How many function evaluations did you need for steps a and b?

Simulated Annealing

You are asked to find the global minima of the following function with simulated annealing:

$$L(x,y) = 6sin(x+y) + 6sine(x-y) + 0.5 * x\text{^}2 + 0.5 * y\text{^}2 + 50$$

a) Create a function to randomize an initial solution (within a range of -10<=x,y<=10, with increments of 0.1, solutions should have an equal probability).

b) Create a function to estimate the cost (energy) of a given solution. Explain.

c) Create a function to randomly select a neighbor solution. The function should randomly select a step size (from 0.1 to 0.5 in increments of 0.05, uniform distribution) and a direction (8 directions: north (y+step), north west(y+step, x-step), west(x-step) etc.) and apply that step to the old solution to get the new solution.

d) Create a function to run simulated annealing on the given problem.

e) Set the timetable for the decrease in temperature, initial chances of accepting a bad move should be around 0.5 and should decrease to around 0.01. **Explain your selected timetable**.

f) Record the value of the cost function at each iteration, run your algorithm and plot the energy over iteration, **explain the result**!

g) Run your algorithm multiple times (at least 5) and record the result. Do you get the same result each time? **Explain**.

h) (bonus-8) add gaussian noise (mean-0, std-1) to the target function, run your algorithm again with the noise, plot the energy over iterations and explain the effect

on performance. Suggest a way to improve the algorithm to deal with the noise. You bonus points will be rewarded based on execution of the instruction, the understanding displayed by your explanation and the creativity of your suggestion.