Recipe Name: cyclic_min

Inputs:

func, the function whose local minimum will be returned

start, a sequence where each element of the sequence represents the value of the corresponding variable, with the whole sequence representing the starting point for cyclic minimization

num_vars, an integer representing the number of variables in *func*

step_size, a float representing how far to go along each variable for each iteration of cyclic minimization

Outputs:

local_mins, a sequence where each element of the sequence represents the value of its corresponding variable, with each variable represented in the same order as *start*, with the whole sequence representing the found local minimum

Steps:

1. Initialize *local_mins* to a copy of *start*
2. Initialize *temp_plus* to an empty sequence
3. Initialize *temp_min* to an empty sequence
4. *final_local* ← 0
5. While *final_local* is not equal to *num_vars*,
    a. For each number, *var*, from 0 to *num_vars* – 1,
        i. Initialize *found* to False
        ii. While *found* is not True, do the following
            1. *temp_plus* ← *local_mins*
            2. *temp_min* ← *local_mins*
            3. $temp\_plus_{var}$ ← $local\_mins_{var}$ + *step_size*
            4. $temp\_plus_{var}$ ← $local\_mins_{var}$ – *step_size*
            5. If *func*(*temp_min*) < *func*(*temp_plus*) and *func*(*temp_min*) < *func*(*local_mins*), then
                a. *local_mins* ← *temp_min*
                b. *final_local* ← 0
            6. Otherwise, if *func*(*temp_plus*) < *func*(*temp_min*) and *func*(*temp_plus*) < *func*(*local_mins*), then
                a. *local_mins* ← *temp_plus*
                b. *final_local* ← 0
            7. Otherwise,
                a. *found* ← True
                b. *final_local* ← *final_local* + 1
6. Return *local_mins*