# Numerical Optimization – Homework 3

Problem 1: Implement the stopping criteria provided on page 6 in the week 3 lecture slide in the MATLAB code random\_descentsearch. Test your code for tolerance values  and

Answer:

In this problem, we want to see how using different stopping criteria affects the performance of the random descent algorithm. We also want to understand how changing the **tolerance** value, the small number that tells the algorithm when to stop, influences its accuracy. According to the lecture notes, for a given tolerance, we can use any of the following stopping conditions:

* Gradient criterion:

If the gradient is almost zero (less than tolerance), it means the slope is flat and we’re probably near the minimum.

* Step size criterion:

It means if the new point ​ is almost the same as the previous one ​, the algorithm isn’t really moving anymore.

* Function value criterion:

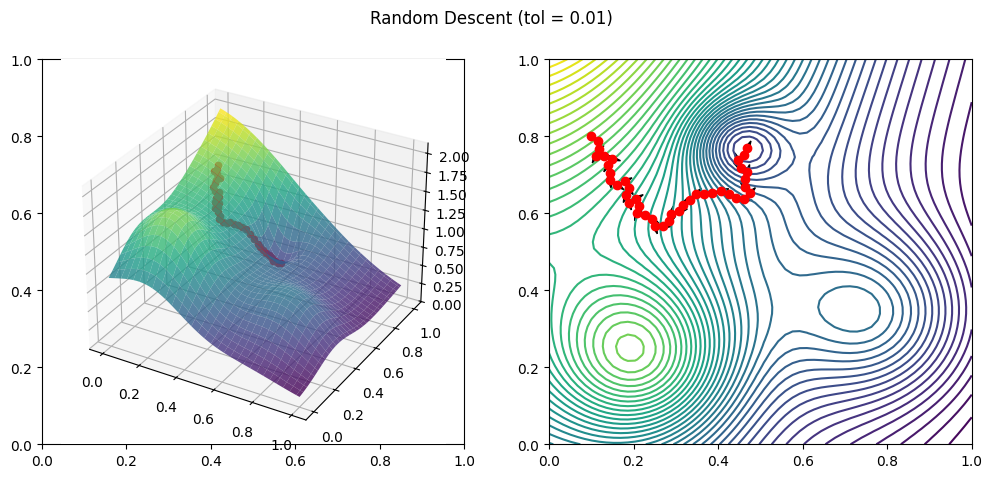
This checks if the improvement in the objective function is minimal, implying that progress has essentially stopped.

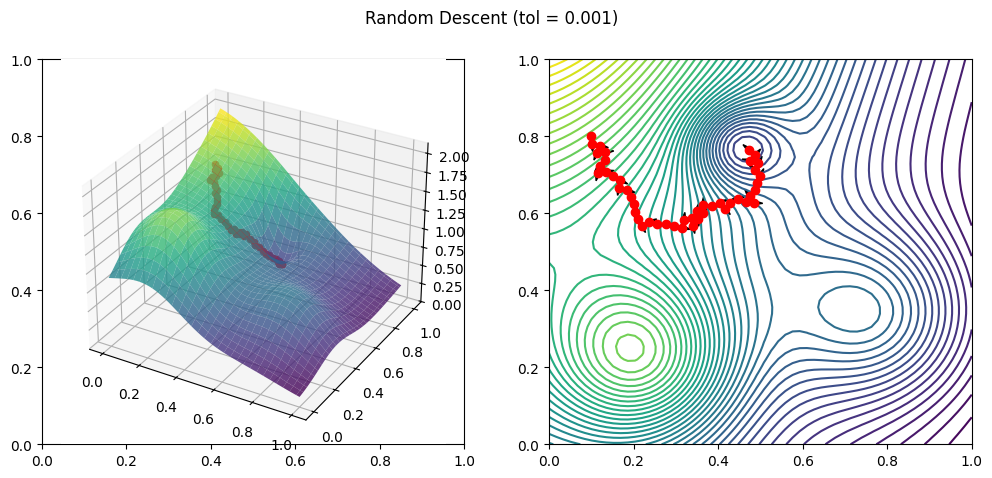
The objective function I used is the same as the one in the MATLAB code, and the following parameters have been implemented:

* Tolerance =
* Step size = 0.02
* Maximum iterations = 100

Results:

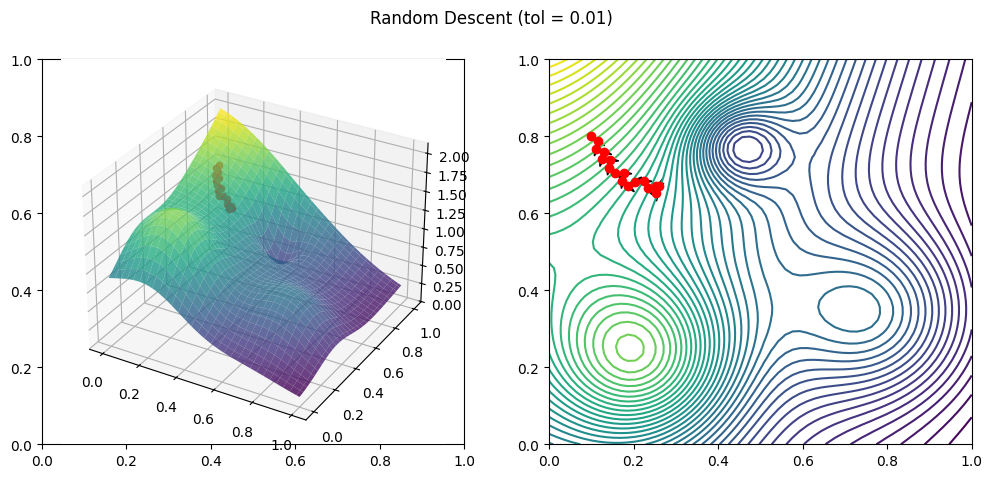
* Implementing Stopping criterion 1:

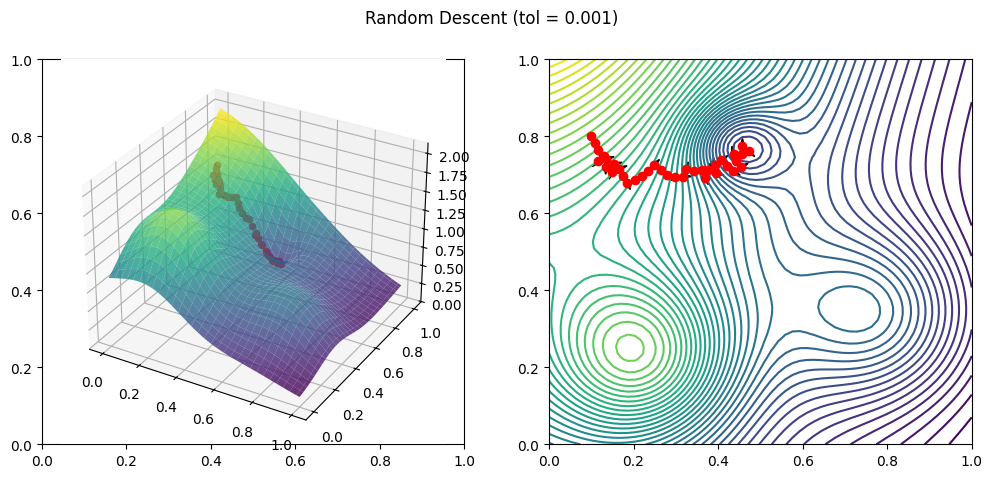




Using the gradient stopping criterion, the model was able to reach the local minimum in both values of tolerance.

* Implementing stopping criteria two and three:

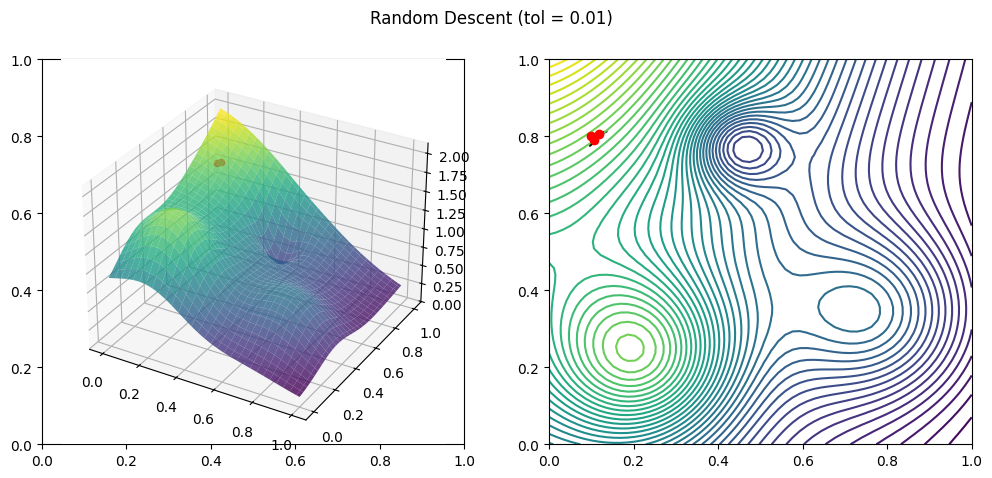


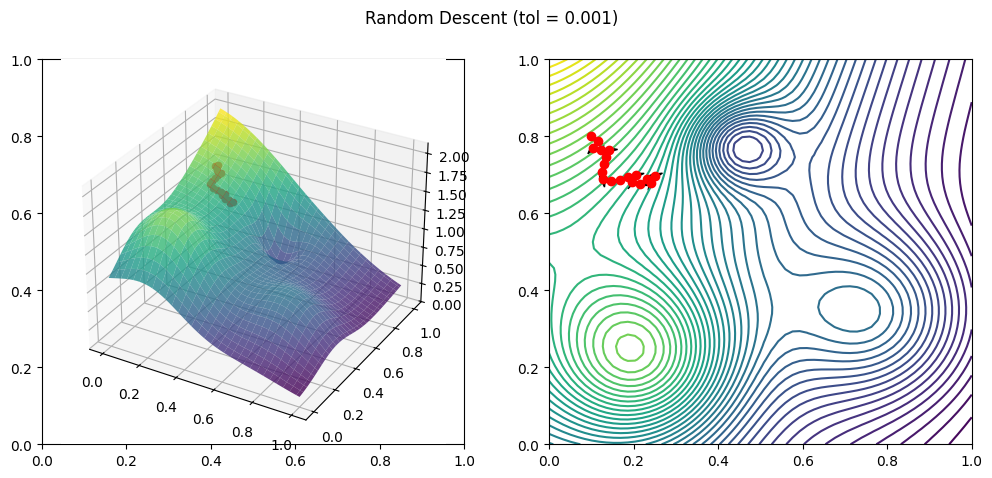


When the tolerance is larger (0.01), the algorithm stops earlier because it quickly satisfies the stopping condition. As a result, the path (red dots) shows fewer steps, and the algorithm doesn’t move all the way down to the true minimum. It stops when the improvement becomes smaller than the set tolerance, even though it hasn’t fully converged yet. On the other hand, when the tolerance is smaller (0.001), the algorithm continues for more iterations before stopping. The path is longer and it gets closer to the actual minimum since the condition for stopping is stricter. This makes the result more accurate but also increases the computation time.

As a conclusion, I believe that when only the **gradient-based stopping criterion** was used, the algorithm was able to find the minimum for both tolerance values. This makes sense because checking the gradient directly measures how flat the surface is. If the gradient is close to zero, we are truly near an optimum. In contrast, the step or function-value criteria can sometimes stop too early if the updates or improvements are small, even though the slope hasn’t fully flattened yet.

* Implementing all stopping criteria at once:





When all three stopping criteria are applied together, the algorithm becomes very sensitive to small numerical changes. Each condition: gradient size, step difference, and function change, has a different meaning, and combining them can sometimes make the algorithm stop too soon. For example, even if the gradient is still large (meaning we are far from the minimum), the step size or function change might already be small due to random direction updates or local flatness in the surface. In that case, one of the conditions gets satisfied early, so the algorithm thinks it has converged when it actually hasn’t. This problem becomes more noticeable with larger tolerance values, because all conditions are easier to satisfy. The algorithm may stop right after taking just a few steps, long before reaching the true minimum. On the other hand, using all conditions together with a very small tolerance makes the algorithm too strict, it might take too long to meet all conditions at once or even fail to stop properly if small numerical fluctuations keep one of the conditions slightly above the threshold.

Problem 2: Test your code in Problem 1 for finding a local minimizer for:

* A function involving 3 variables
* A function involving 4 variables.
* A function involving 5 variables.

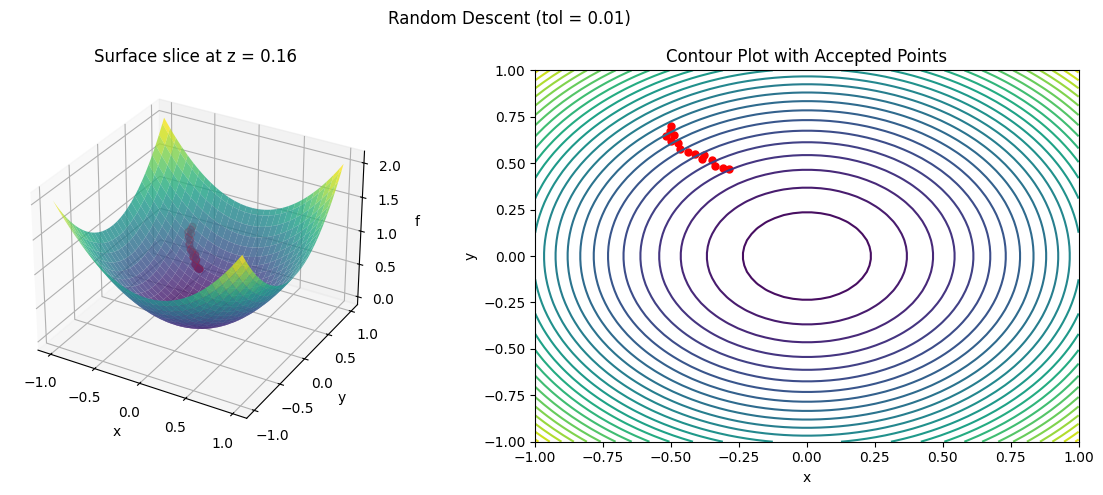
You can choose your own functions. For functions with 4 and 5 variables, you do not need to show the plots/animations.

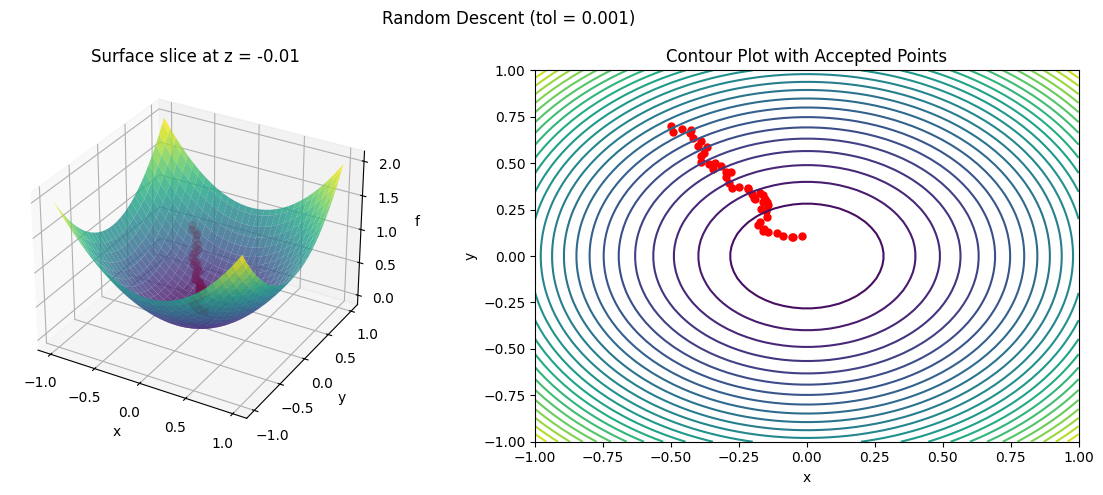
Answer:

In this part, I’m going to test the random descent algorithm on different functions with different tolerance values of and . Also, I’ve implemented the stopping criteria two and three in all of the algorithms.

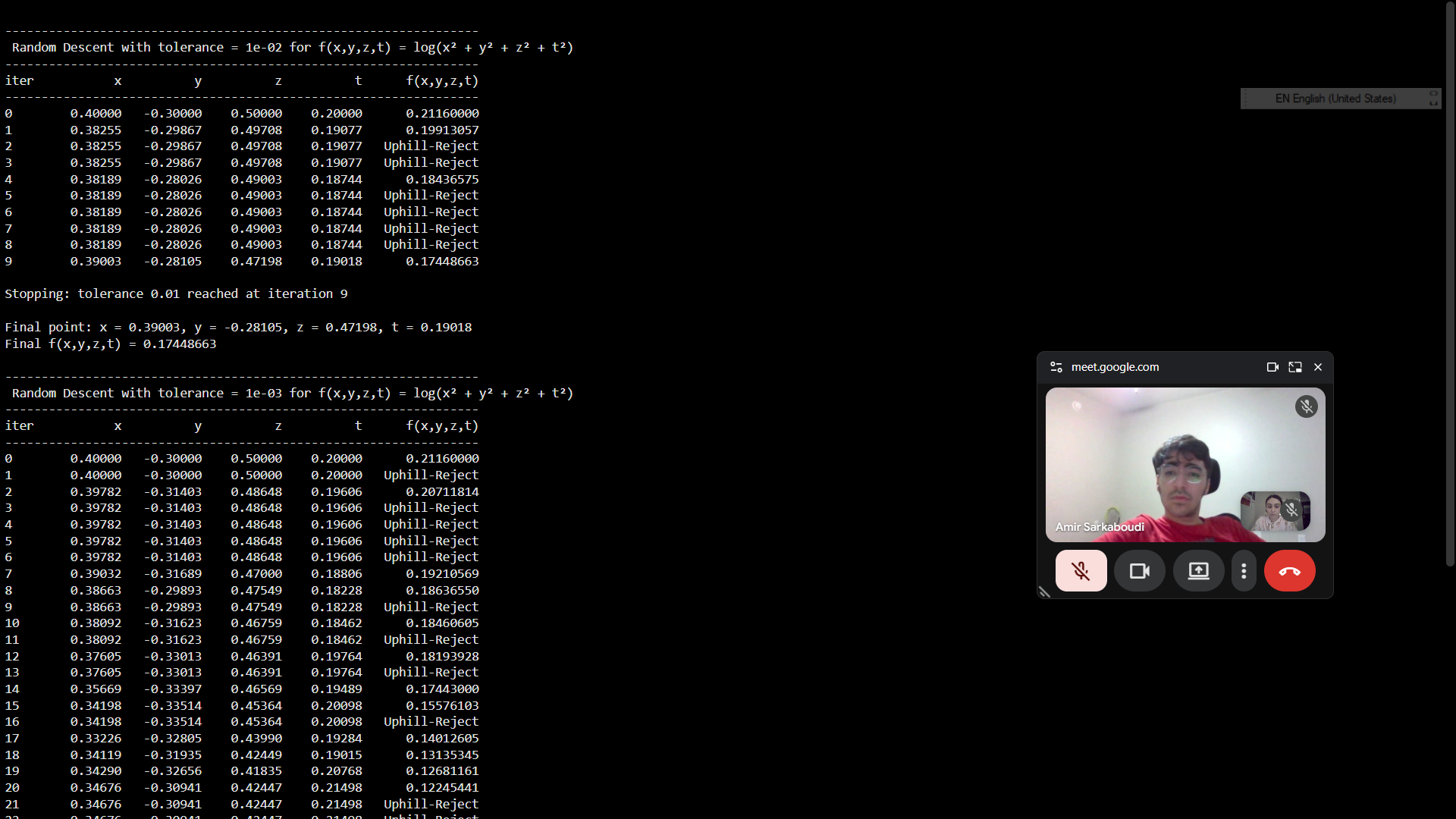
Quick heads-up: As a physics student, I was curious about trying something physical, so do not be surprised when you see my functions for 4 and 5 variables!:)

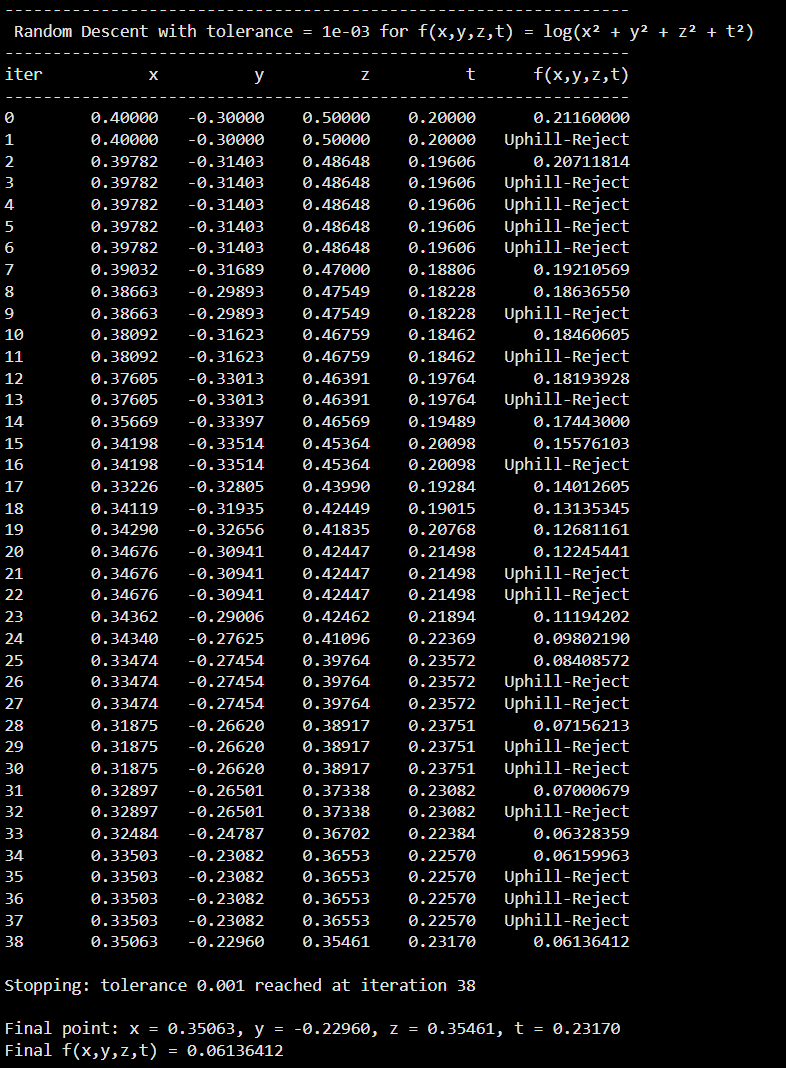
* 3 variables: Sphere function



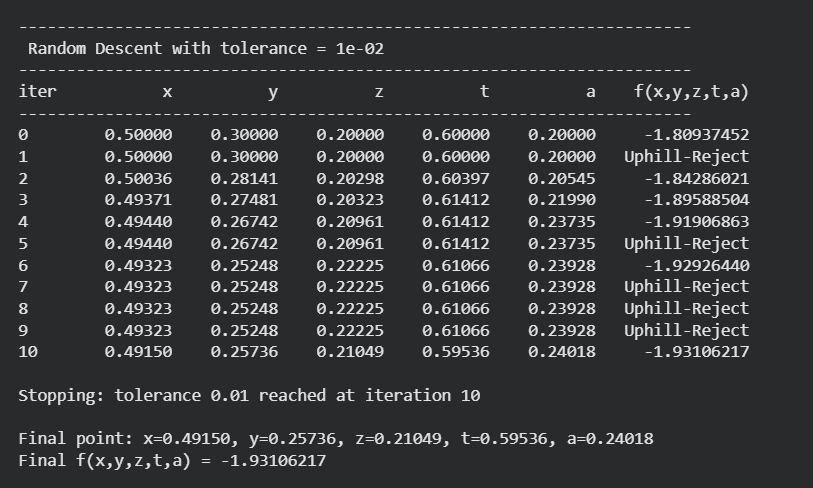


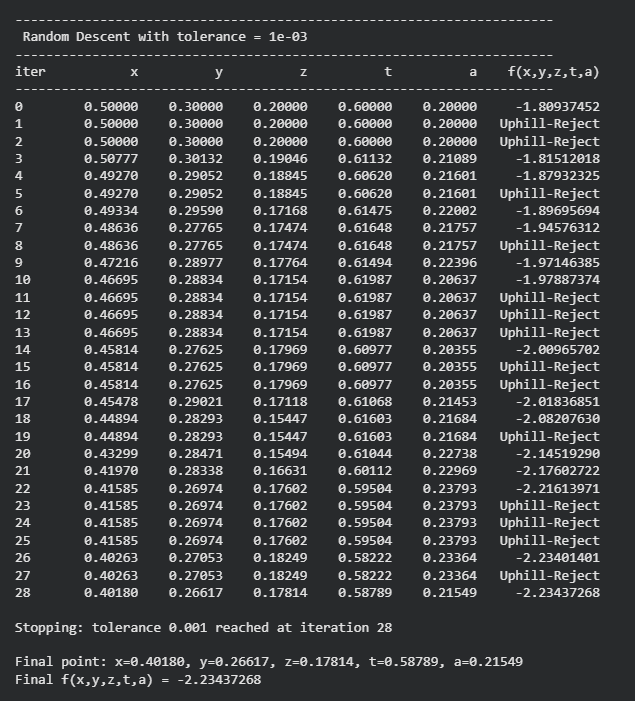
* 4 variables: Minkowski space time interval (This function represents the invariant separation between two events in four-dimensional spacetime)





* 5 variables: Kerr gravitational potential (It describes the spacetime curvature around a rotating, uncharged black hole, including both mass and angular momentum effects.)



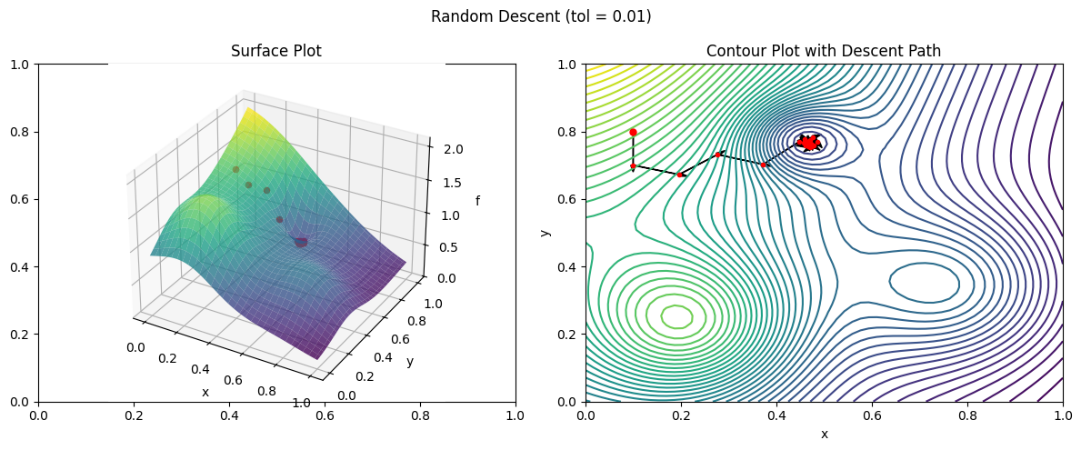


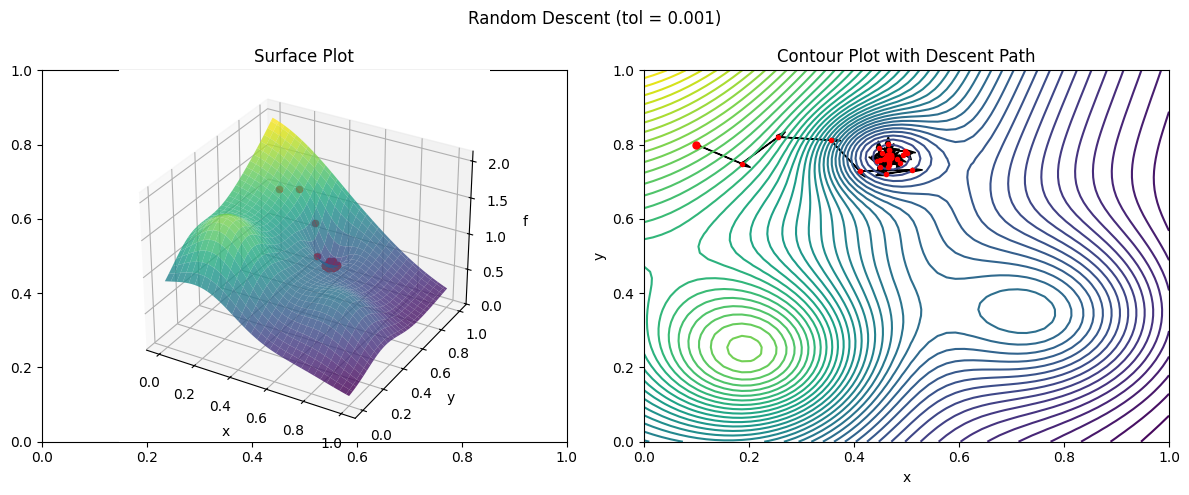
Problem 3: In the MATLAB code random\_descentsearch, we are using a fixed step value  at each step. Given a direction vector , implement a strategy in the code to find the largest  such that  in each step.

Answer:

In this problem, we need to use an adaptive step length instead of a fixed one. The method I used here, is one of the most well-know methods called backtracking search. I chose the same function as before, two-dimensional multi-Gaussian surface, and following the steps below:

* A random unit vector is generated
* The initial value of step length is 0.1
* The new point is tested
* If   shrinks by a factor(I chose ½), until
* Once a valid αₖ is found, the move is accepted and plotted.



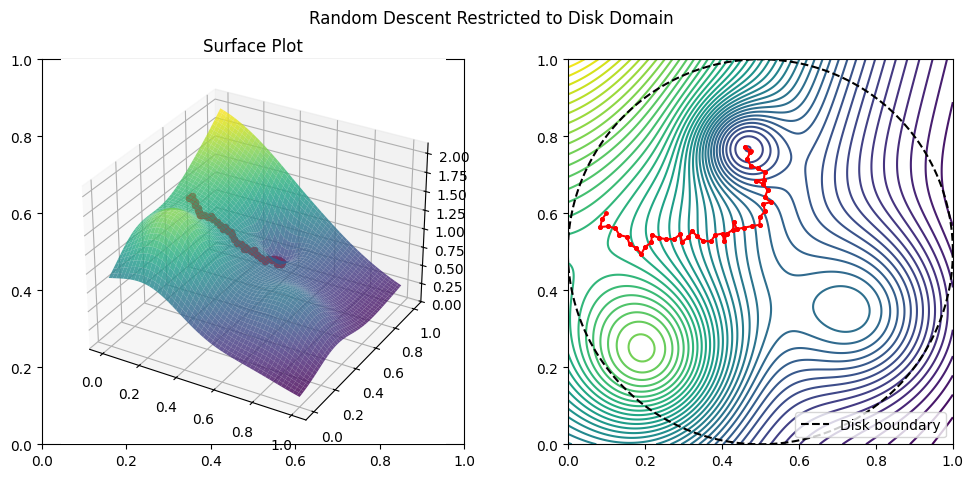


This plot shows how the random descent works when the step size changes automatically. In the beginning, the steps are bigger, so the points move faster across the surface. As the algorithm gets closer to a minimum, the steps become smaller, which helps it avoid jumping over the low point. That’s why the points at the end are very close together. This adaptive step size makes the method more stable and helps it find the minimum more smoothly than using a fixed step size.

Problem 4: Modify the MATLAB code random\_descentsearch such that the domain of the objective function is restricted to a disk

Answer:

In this problem, we need to restrict the search domain. That means before accepting a new point, we should check if it lies inside the mentioned disk. If not, we can discard or reduce the step so it stays within the boundary.



The plots above show how the random descent algorithm works when the search is limited to a circular area (the disk). The dashed black line marks the boundary of the disk, and all the red points are the accepted steps where the function value decreased. From the contour plot, we can see that the algorithm always stays inside the disk and gradually moves toward a low point inside it. On the surface plot, the red points show how the search moves downhill along the surface. Since the step size is fixed, some moves are smaller and some are rejected when they go outside the circle. Overall, the algorithm still finds a local minimum within the allowed area, showing that adding a boundary changes the path but keeps the method stable and controlled.