

Steepest Descent Optimization



Optimization
Techniques Project

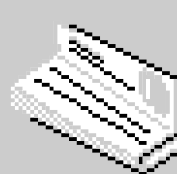
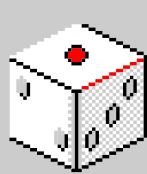
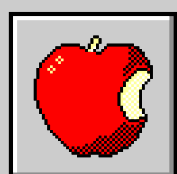


Ragheed Almasry - 412117664

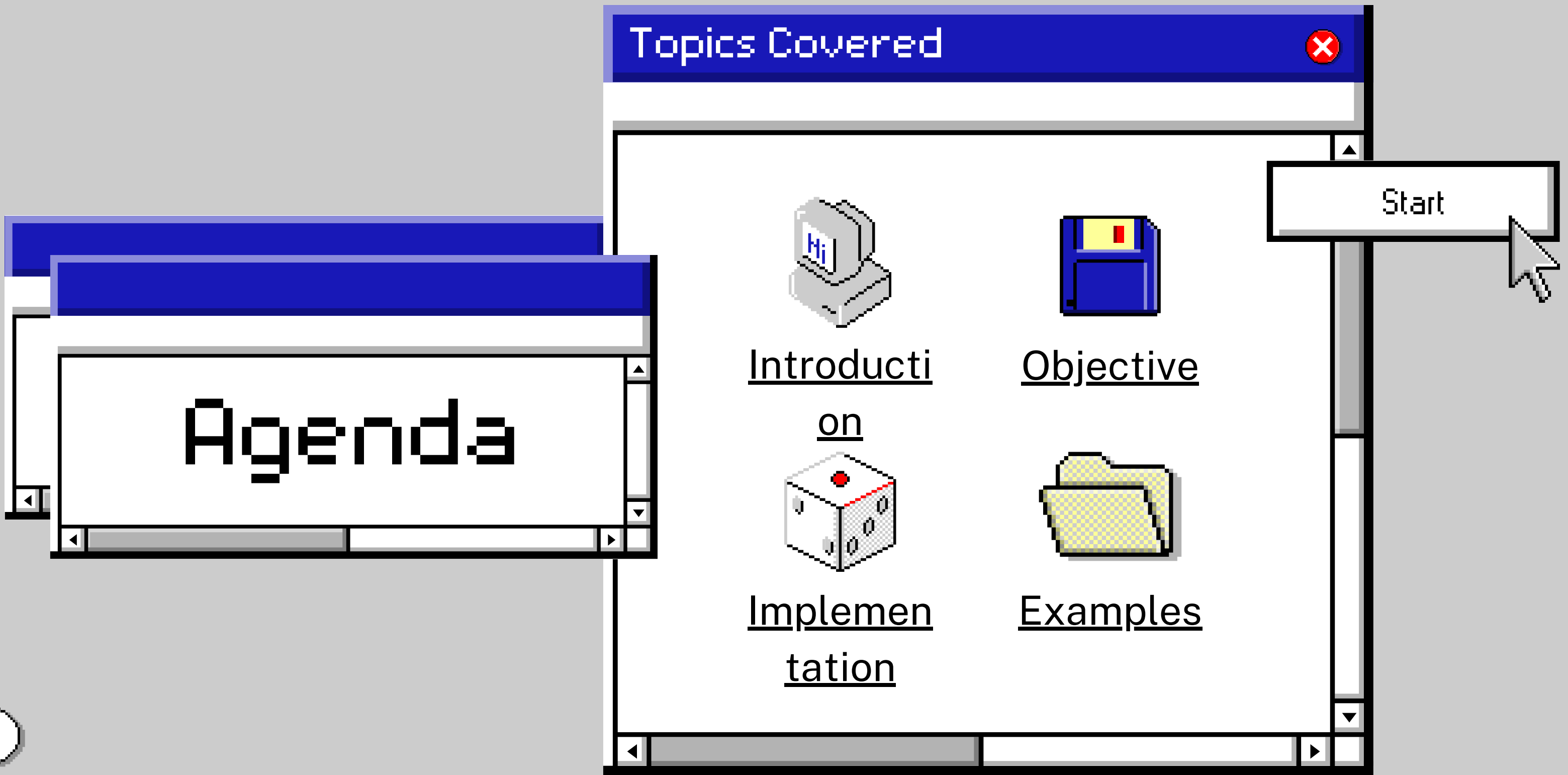
Anas Alarbied - 411116054

Mustafa Sallat - 412117552

Instructor: Dr. Ali Mustafa



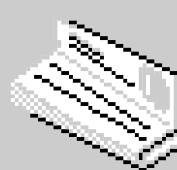
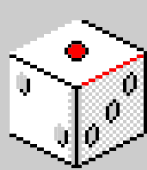
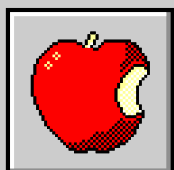
11:11PM



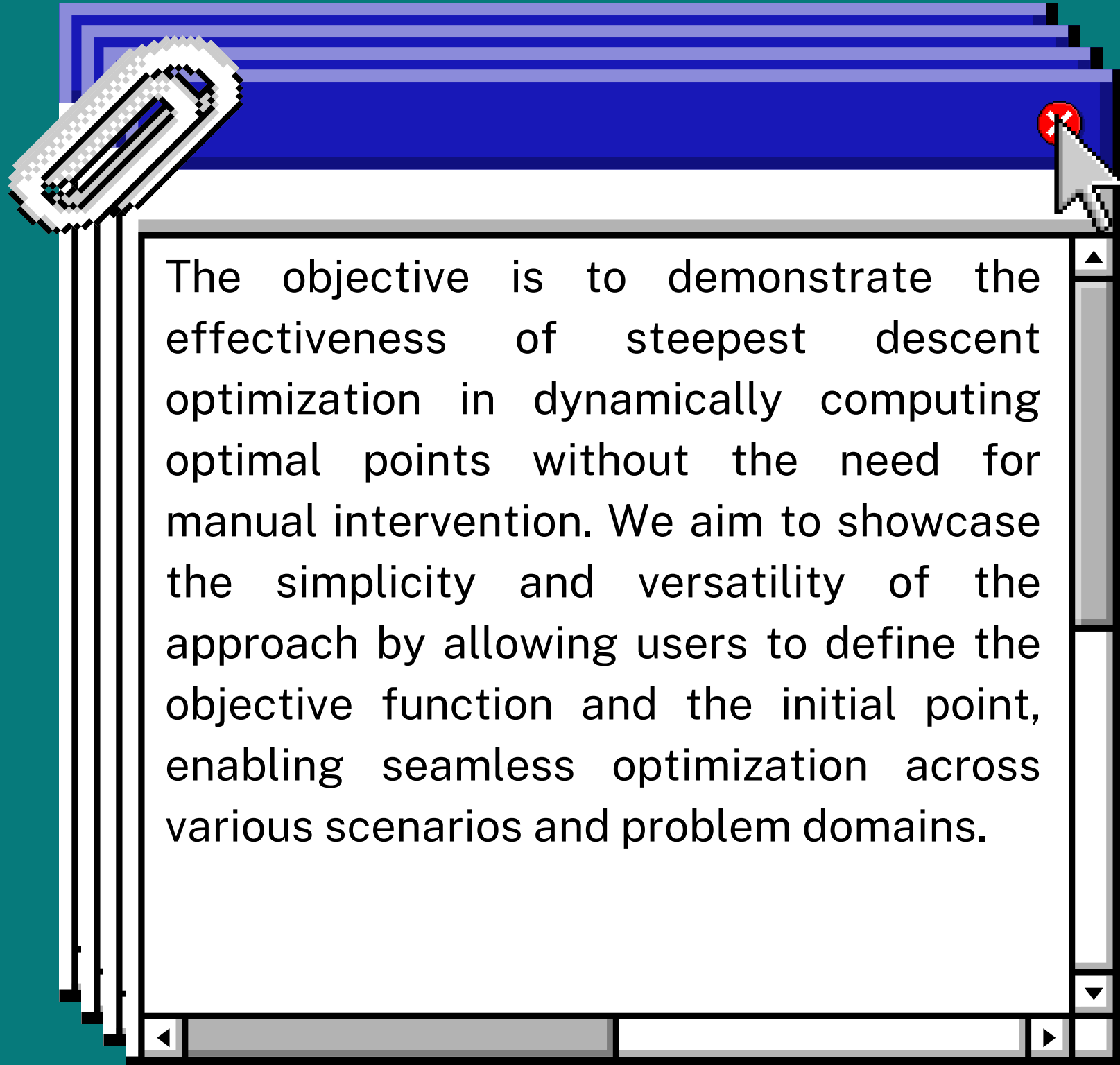
Introduction



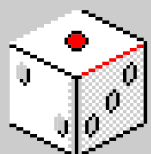
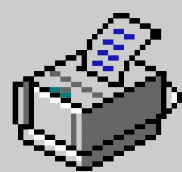
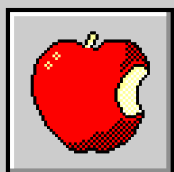
Steepest descent optimization is a versatile technique used to find the minimum of a function. The steepest descent aims to converge to the optimal point where the function reaches its minimum value. In this presentation, we explore the implementation of steepest descent optimization, focusing on dynamically computing the optimal point by defining only the objective function and the initial point.



objective



The objective is to demonstrate the effectiveness of steepest descent optimization in dynamically computing optimal points without the need for manual intervention. We aim to showcase the simplicity and versatility of the approach by allowing users to define the objective function and the initial point, enabling seamless optimization across various scenarios and problem domains.





Implementation

We implement steepest descent optimization using Python and the Autograd library. The implementation involves several key steps:





1. Defining The Objective Function and The Initial Point



```
1 def f(x):  
2     return (x[0] - 4) ** 2 + (x[1] + 5) ** 2  
    + 1.8 * (x[0] - 4) * (x[1] - 5)
```



Figure 1: Objective Function $f(x)$

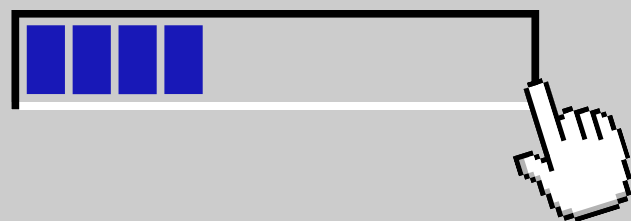
```
1 # Initial point  
2 initial_point = np.array([4.0, -7.0], dtype=float)
```



Figure 2: Initial Point



2. Computing The Gradient and Hessian Matrix Using Autograd



```
1 # Compute the gradient of the objective function
2 gradient_f = grad(f)
```



Figure 3: Gradient of
The Objective Function
 $\nabla f(x)$

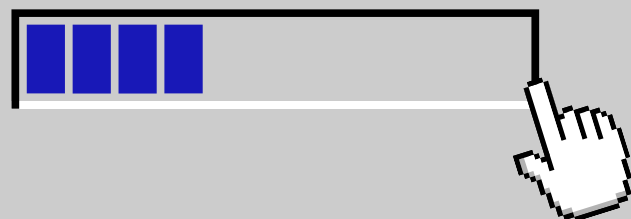
```
1 def steepest_descent(initial_point):
2     gradient = gradient_f(initial_point)
3     hessian_func = hessian(f)
4     hessian_matrix = hessian_func(initial_point)
```



Figure 4: Gradient of
Initial Point $\nabla f(x_0)$ and
The Hessian



3. Calculating c and k for the standard form of the quadratic function



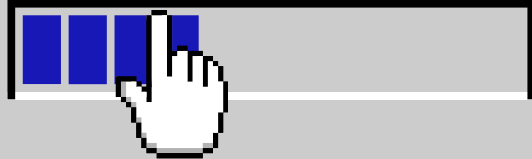
```
1 # Calculate c and k
2 c = gradient_f(np.zeros_like(initial_point))
3 k = f(np.zeros_like(initial_point))
```



Figure 5: Calculating
The Linear Part and The
Constant



4. Computing The Gradient Length, Learning Rate



```
1 # Calculate the length of the gradient
2 gradient_length = np.linalg.norm(gradient)
```



Figure 6: Gradient Length

```
1 def calculate_learning_rate(grad, hessian):
2     learning_rate = np.dot(grad, grad) / np.dot(grad,
3     np.dot(hessian, grad))
4     return learning_rate
```

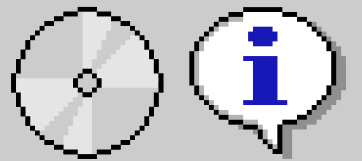


Figure 7: Learning Rate Function

```
1 learning_rate = calculate_learning_rate(gradient, hessian_matrix)
```



Figure 8: Learning Rate



5. Checking Function Convexity

```
1 # Check if the function is convex
2 is_convex = np.all(np.linalg.eigvals(hessian_matrix) >= 0)
```



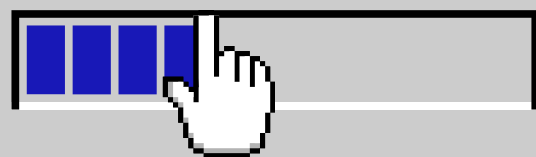
Figure 9: Check
Function Convexity

6. Finding Critical Point Using The Hessian Matrix

```
1 # Find the critical point using the Hessian
2 critical_point = -np.linalg.inv(hessian).dot(c)
```

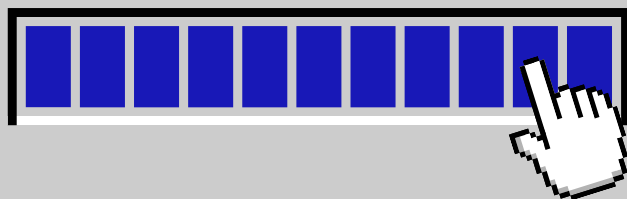


Figure 10: Critical Point
Using The Hessian





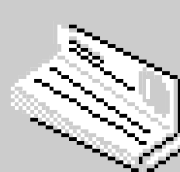
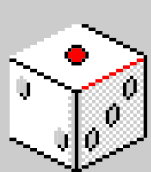
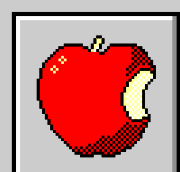
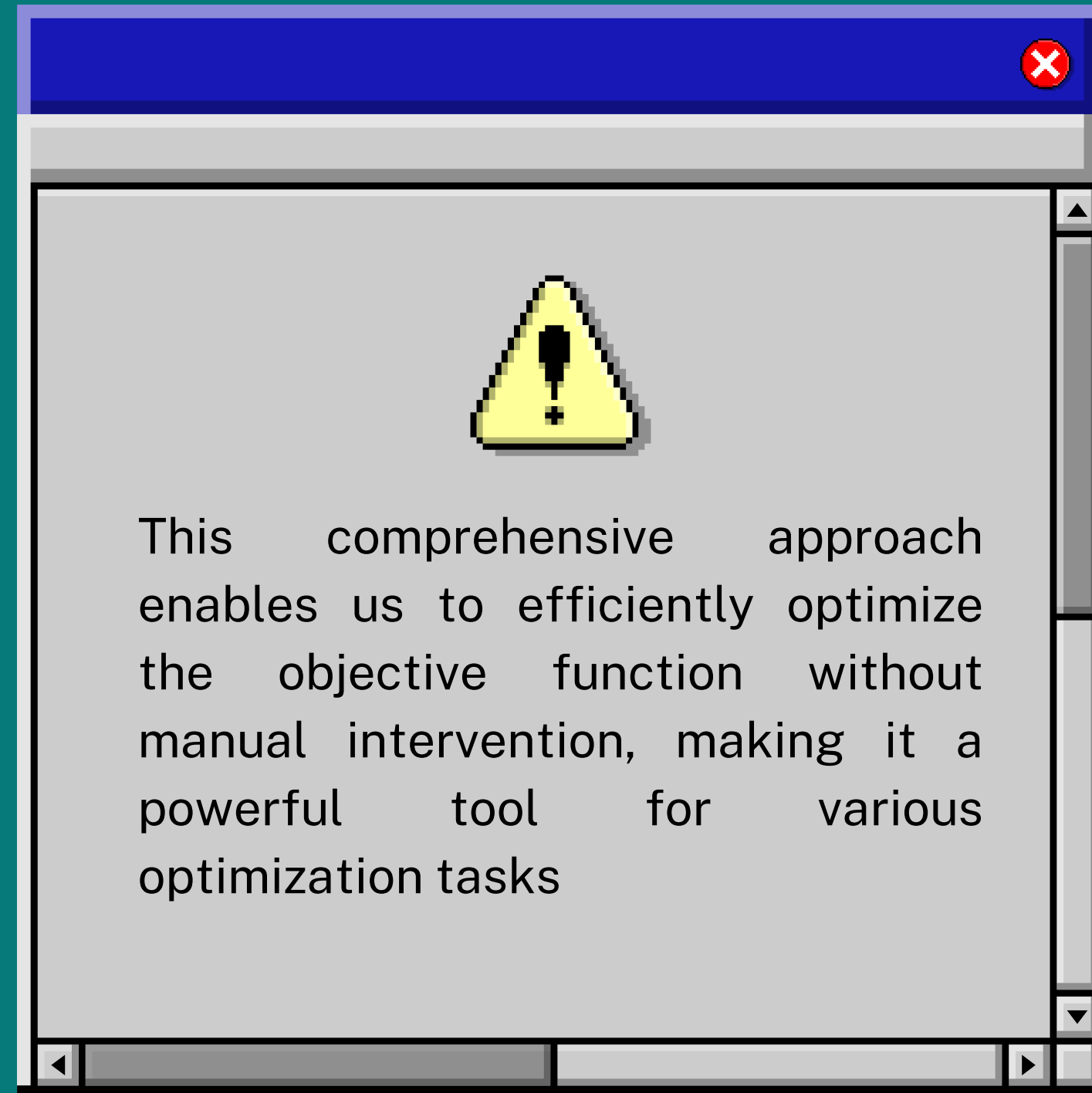
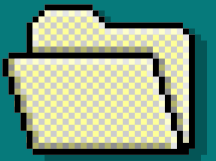
7. Computing The optimal Points Using Steepest Descent



```
1  # Steepest Descent Optimization Method
2  x_next = initial_point - learning_rate * gradient
```



Figure 11: Enter Caption



[Back to Agenda Page](#)

results



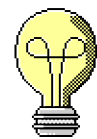
```
Constant k: 36.0
Linear part c: [-14.  54.]
Hessian matrix:
[[  8.   -17.75]
 [-17.75  44.5 ]]
The function is convex.
Critical point using Hessian: [-8.19541985 -4.48244275]
Gradient at initial point: [ 142.25 -328.5 ]
Gradient length: 357.97669267705123
Learning rate: 0.01934924201406845
Optimal point using Steepest descent: [ 1.24757032 -0.643774 ]
Minimum value: 13.473318492520413
```


results



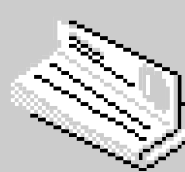
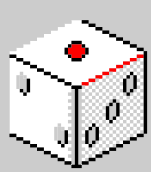
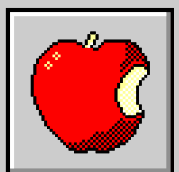
```
Constant k: 51.0
Linear part c: [-5.  15.5]
Hessian matrix:
[[ 2.   -1.75]
 [-1.75  2.   ]]
The function is convex.
Critical point using the Hessian: [-18.26666667 -23.73333333]
Gradient at initial point: [-7.25 20.75]
Gradient length: 21.980104640333266
Learning rate: 0.32364085494776945
Optimal point using Steepest Descent: [7.3463962  0.28445226]
Minimum value: 69.07050597667944
```


conclusion



I like that It so cool

We conduct experiments with different objective functions and initial points to evaluate the performance of the dynamically computed optimal points. The results demonstrate the versatility and effectiveness of steepest descent optimization in dynamically computing optimal points across various problem domains, providing valuable insights for practical applications in optimization tasks.





Thank you!

Ragheed Almasry - Anas Alarbied - Mustafa Sallat
Instructor: Dr. Ali Mustafa

