**Problem 1.1.1**Given: It+1(x’ + Δp) ≈ It+1(x’) + ∂It+1(x’) / ∂x’ . ∂W(x;p) / ∂pT . Δp

Here, p = [px, py]T

X’ = W(x;p) = x+p

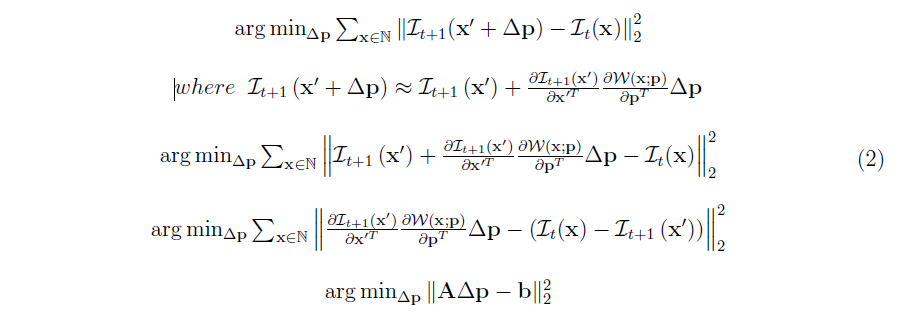
Hence, W(x;p) has two parameters p1 and p2

W(x;p) = = =

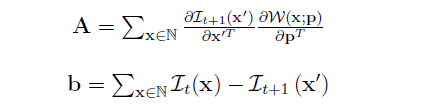
So, ∂W/∂pT = =

**Problem 1.1.2**  
Using first-order Taylor expansion we can linearize the objective function locally and on

rearranging the equation to minimize we get as below.



On comparing the last two above forms we get that:

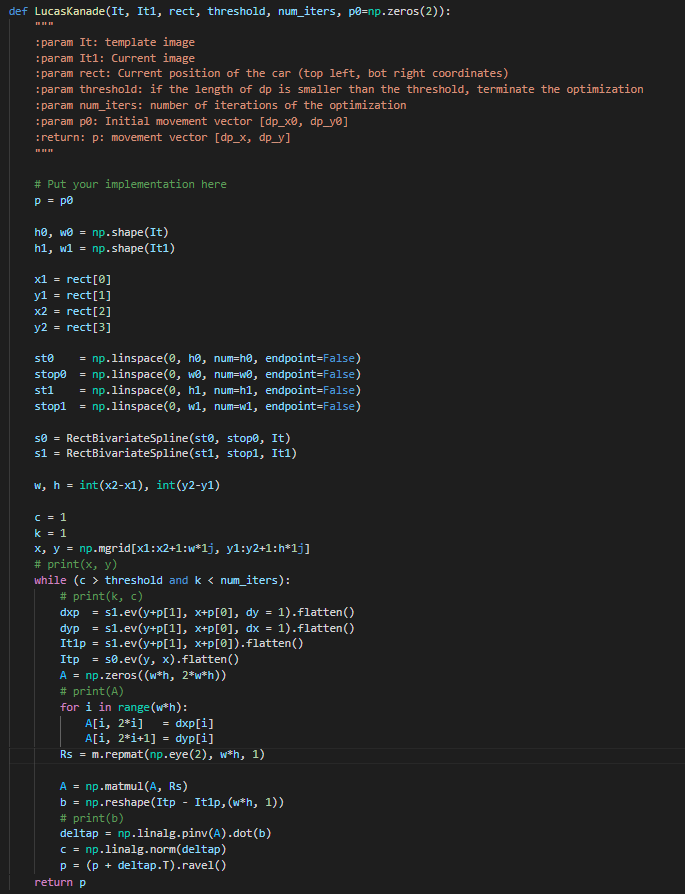


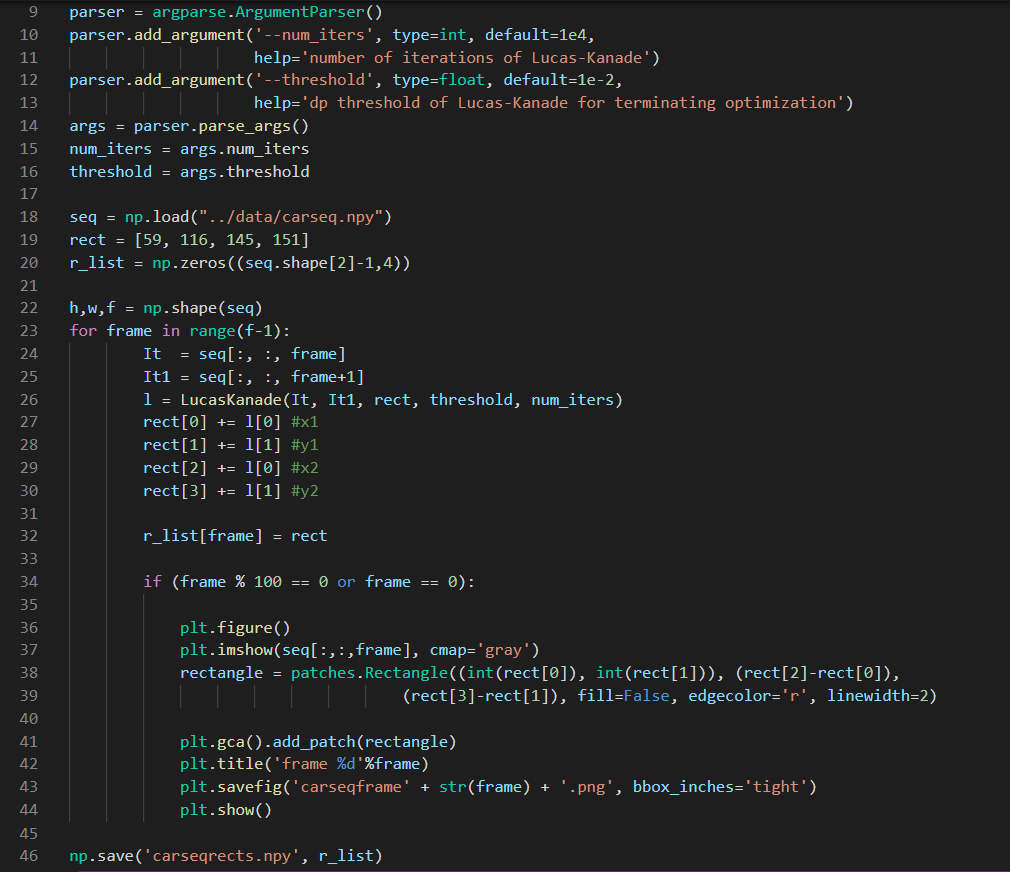
**A** represents the steepest descent images and **b** represents the error image.

**Problem 1.1.3**For minimizing ∆p in equation 2, we take its first derivative and equate it to zero and we get the following:



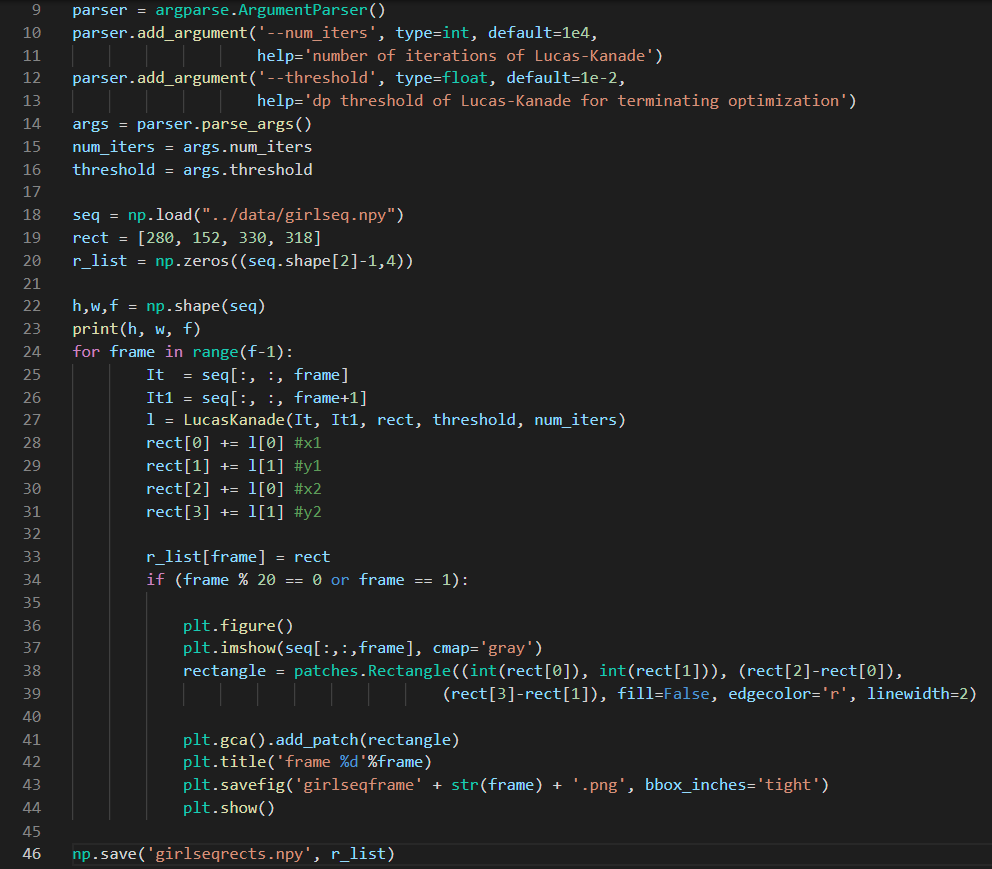
In order to solve for ∆p, **ATA** must be **invertible** or **non-singular matrix** or must **have a non-zero determinant** to obtain a unique solution for ∆p.

**Problem 1.2**

**Problem 1.3**testCarSequence.py ****

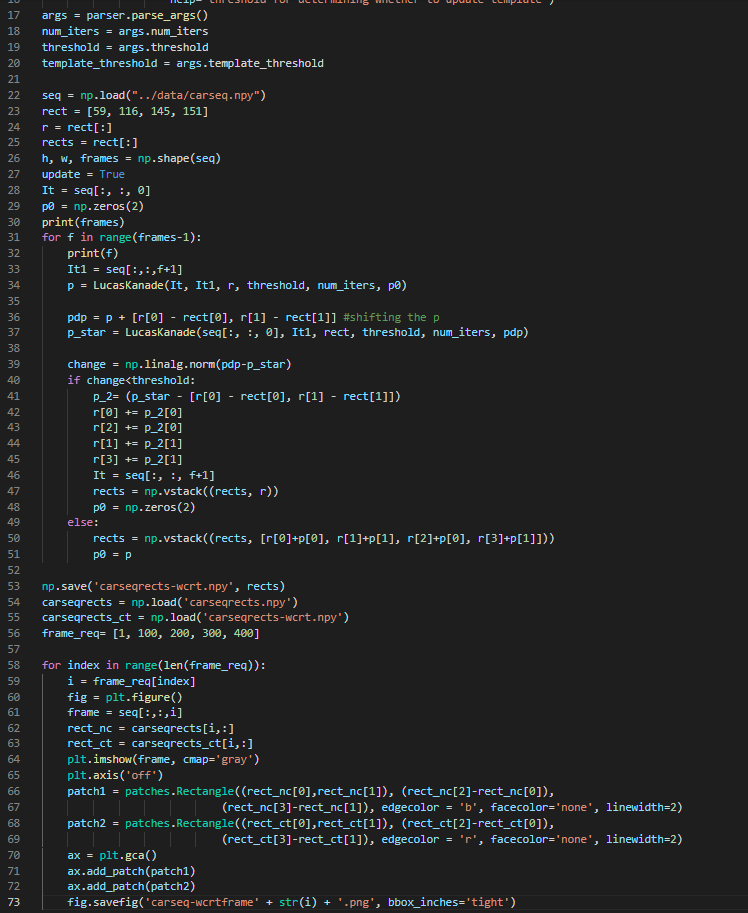
|  |  |
| --- | --- |
|  | |
|  |  |
|  |  |

testGirlSequence.py



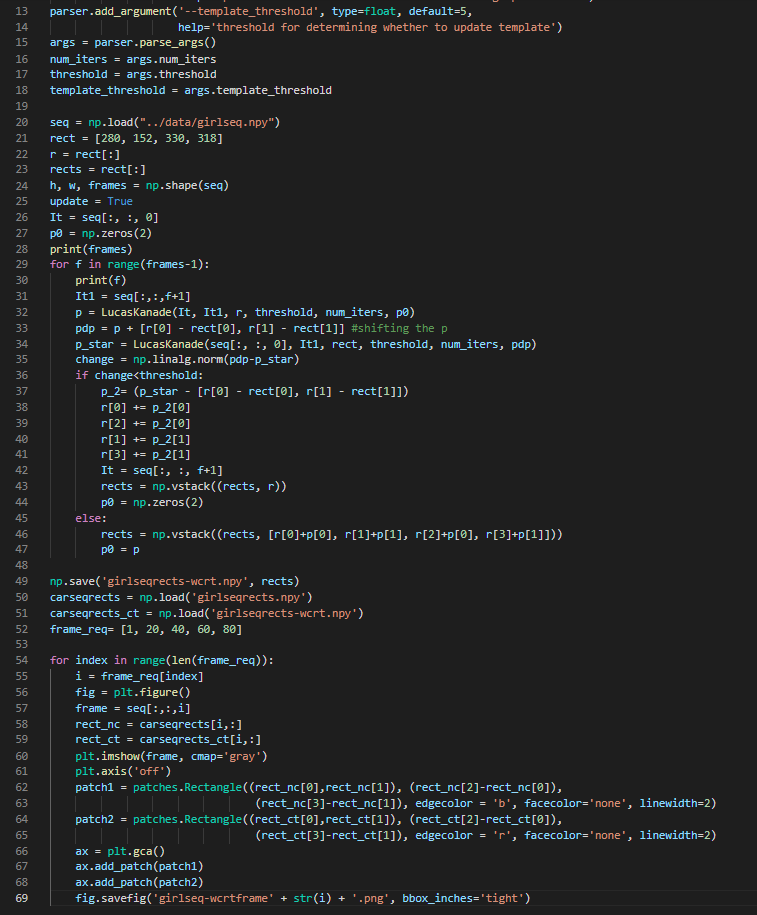
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Problem 1.4**testCarSequenceWithTemplateCorrection.py

****

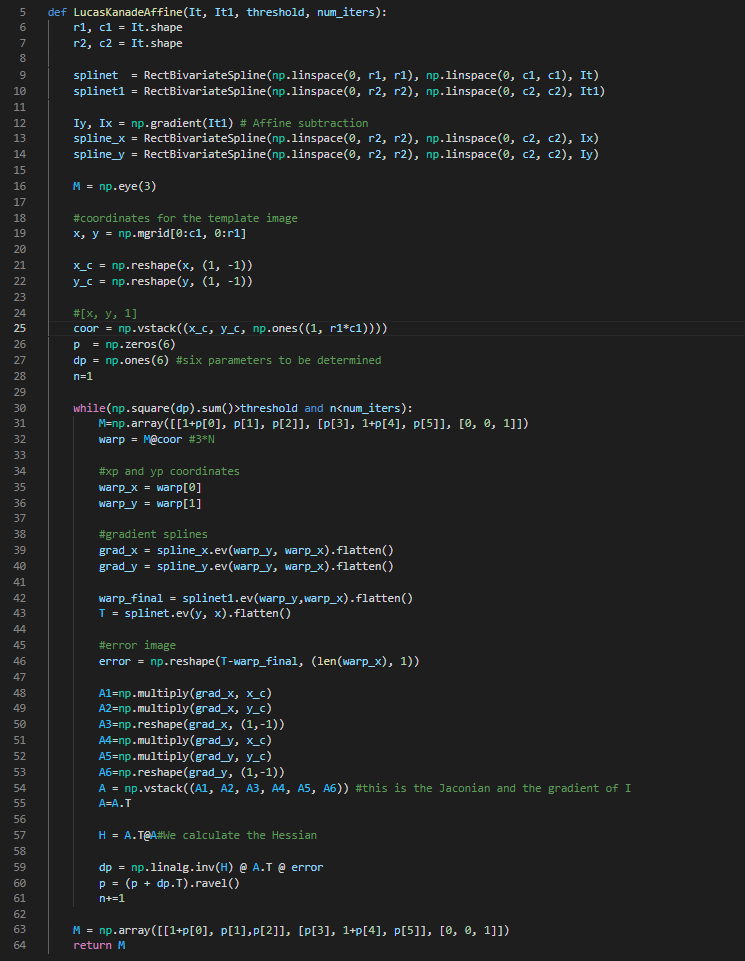
|  |  |
| --- | --- |
|  | **Frame 1** |
| **Frame 100** | **Frame 200** |
| **Frame 300** | **Frame 400** |

testGirlSequenceWithTemplateCorrection.py

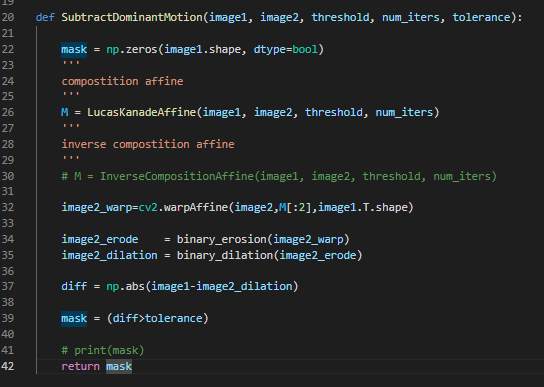


|  |  |
| --- | --- |
|  | **Frame 1** |
| **Frame 20** | **Frame 40** |
| **Frame 60** | **Frame 80** |

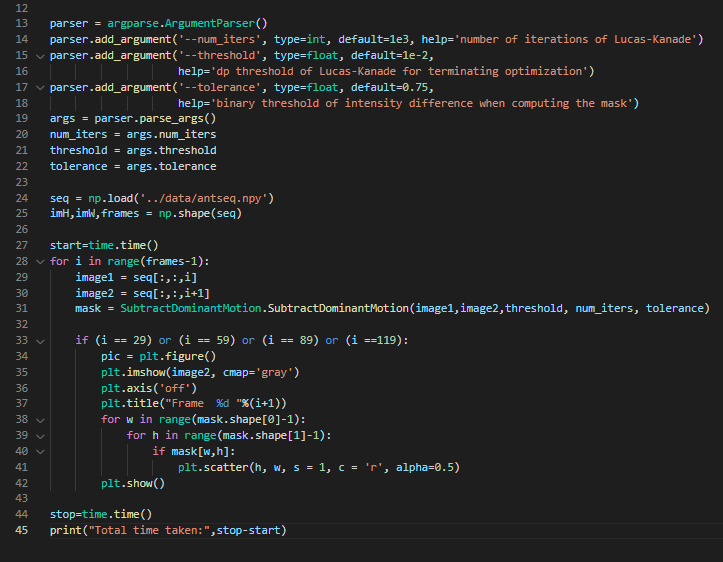
**Problem 2.1**LucasKanadeAffine.py



**Problem 2.2**SubtractDominantMotion.py

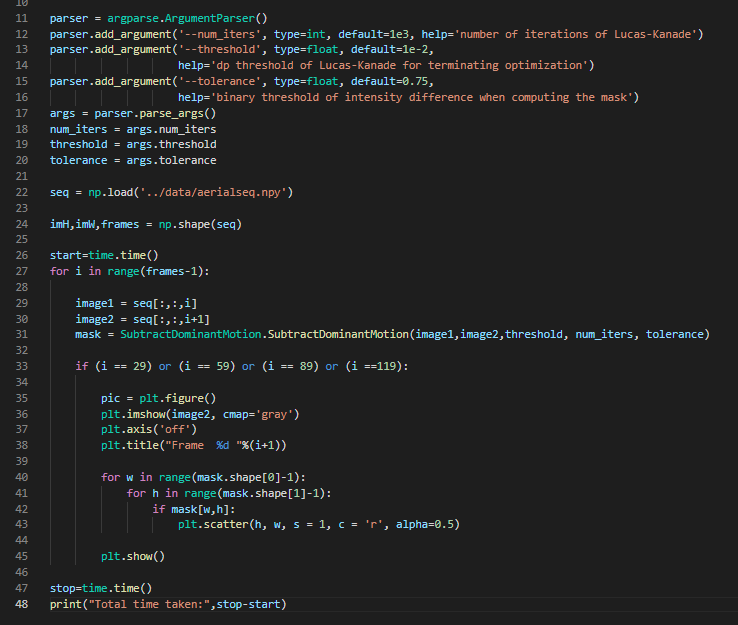


**Problem 2.3**testAntSequence.py



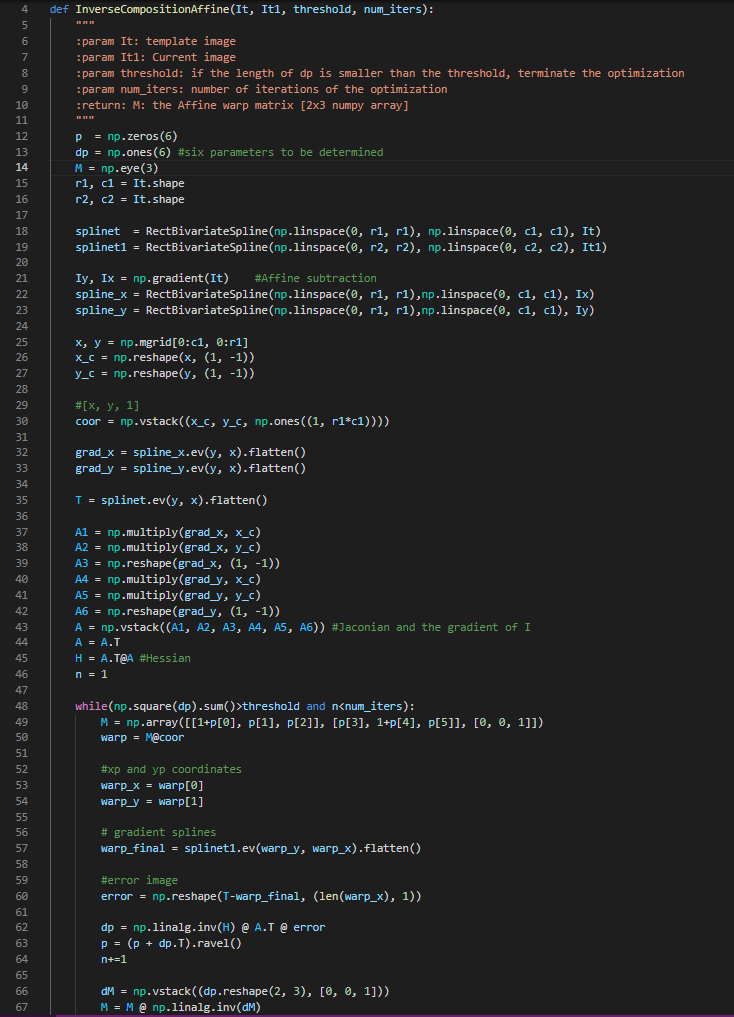
|  |  |
| --- | --- |
|  |  |
|  |  |

testArialSequence.py



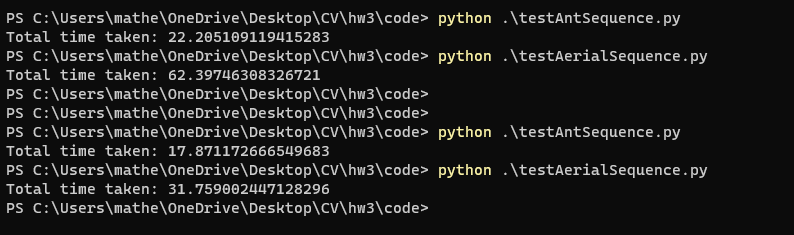
|  |  |
| --- | --- |
|  |  |
|  |  |

**Problem 3.1**InverseCompositionAffine.py



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

In the classical approach we need to update A and b in every iteration until Δp converges. A being a very large matrix D X 6 matrix, it requires more time for the convergence of Δp. However, with inverse compositional approach, A’ and (A’T A)-1A’T can be precomputed only once, and then it can be multiplied to updated b until Δp converges, which saves a huge amount of computational time and costs.



Here, the latter 2 computations are using the InverseCompositionAffine, while the former two correspond to without using Inverse compositional approach. We can clearly see the jump in performance by looking at the time each program took to compute.