Homework 1

Topics on Optimization and Machine Learning

Roberto Meroni
email: roberto.meroni@estudiantat.upc.edu

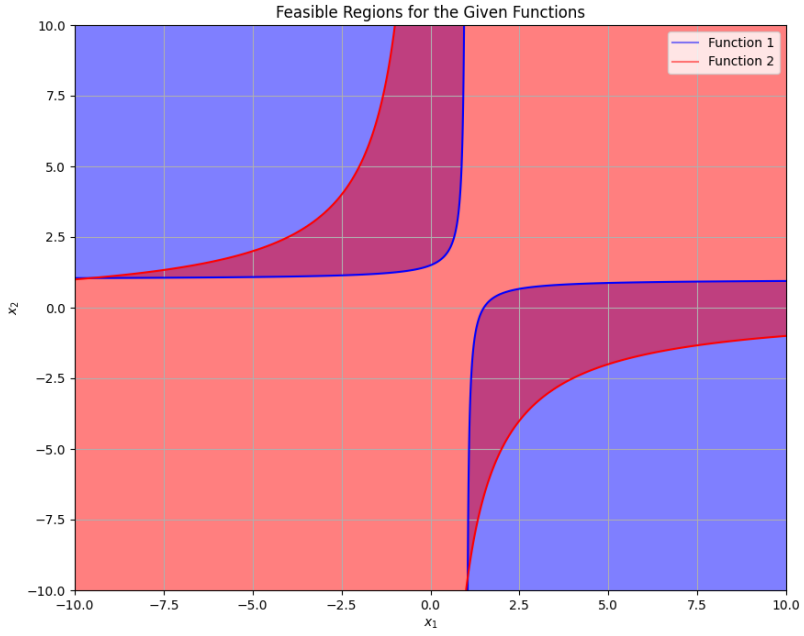
March 2024

Exercise 1

a Identify whether is convex or not.

a.1 Convex Set

Looking at the plot of the feasible region, it is clear that the set is not convex.



As an example, take two points $x_1 = [-5, 5]$, $x_2 = [5, -5]$. They both belong to the feasible set defined by the constraints, but some of the points of the segment connecting x_1 and x_2 do not belong to the feasible set (e.g., $x_3 = [0, 0]$).

a.2 Convex Function

To check if the function is convex, we check if the Hessian is a positive semi-definite matrix. To utilize this criterion, we first have to ensure that the function is twice differentiable.
The objective function $f(x_1, x_2) = e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right)$ is a multiplication between two smooth functions (exponential and polynomial), so $f(x_1, x_2) \in C^\infty$.
Now we calculate the Hessian, using *numdifftools.Hessian*:

```
Hessian of objective_function:
[[9. 6.]
 [6. 4.]]
```

and its eigenvalues, with *numpy.np.linalg.eigvals*:

```
Eigenvalues of the Hessian:
[ 1.3000000e+01 -7.9225515e-13]
```

If all the eigenvalues are ≥ 0, the function is convex. The eigenvalue $\lambda_2 = -7.9225515e - 13$ can probably be approximated to 0, but to ensure its value is non-negative we check by hand. To determine if a 2×2 matrix is positive semi-definite (PSD), we examine its principal minors:

- The top-left 1×1 minor, which is 9 in this case, must be non-negative.
- The determinant of the full 2×2 matrix should also be non-negative:

$$\text{Determinant} = (9)(4) - (6)(6) = 36 - 36 = 0$$

Given the non-negative top-left element and determinant, the matrix meets the PSD conditions, confirming it is indeed positive semi-definite.
The objective function is convex, but since it is not defined on a convex set we cannot ensure that a local minimum will also be a global minimum.

b **Find the minimum, e.g. use scipy.optimize.minimize (SLSQP as method) of python. Use as initial points x0 [0,0], [10,20], [-10,1], [-30,-30] and explain the difference in the solutions if any. Choose which ones give an optimal point/value and give how long take to converge to a result. Plot the objective curve and see whether the plot helps you to understand the optimization problem an results.**

b.1 Initial Guess: $x_0 = [0, 0]$

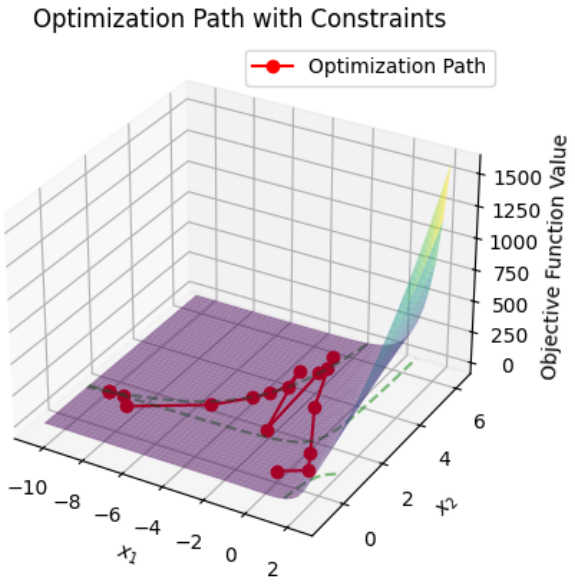


Figure 1: Visualization of the path followed by the solver

```
Initial guess: [0 0],
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.02355037962417486
       x: [-9.547e+00  1.047e+00]
     nit: 17
     jac: [ 1.840e-02 -2.284e-03]
    nfev: 54
```

```
    njev: 17, Time: 0.015645265579223633
```

b.2 Initial Guess: $x_0 = [10, 20]$

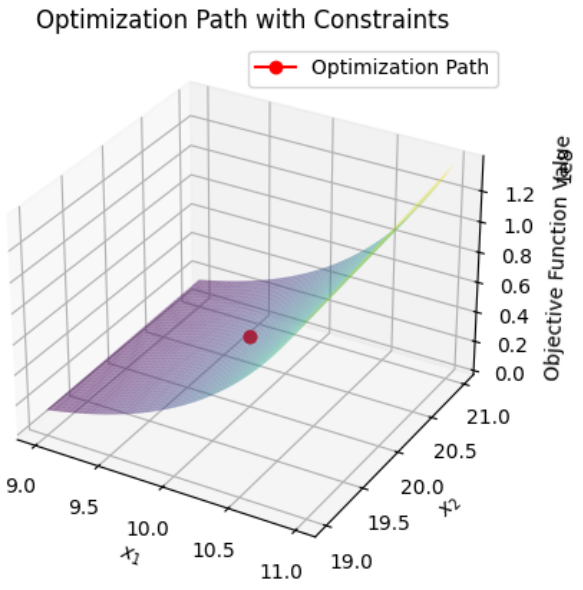


Figure 2: Visualization of the path followed by the solver

```
Initial guess: [10 20]
 message: Inequality constraints incompatible
 success: False
  status: 4
     fun: 44956016.68720051
       x: [ 1.000e+01  2.000e+01]
     nit: 1
     jac: [ 4.848e+07  2.687e+06]
    nfev: 3
    njev: 1, Time: 0.0008003711700439453
```

b.3 Initial Guess: $x_0 = [-10, 1]$

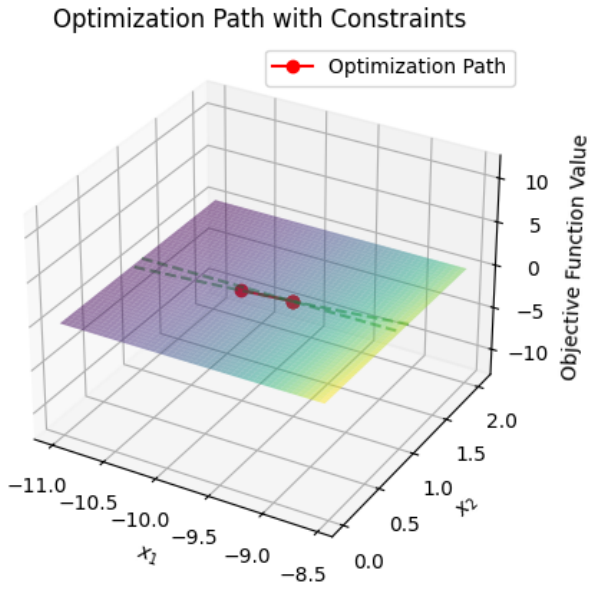


Figure 3: Visualization of the path followed by the solver

```
Initial guess: [-10   1],
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.0235503796241749
       x: [-9.547e+00  1.047e+00]
     nit: 3
     jac: [ 1.840e-02 -2.284e-03]
    nfev: 10
    njev: 3, Time: 0.001476287841796875
```

b.4 Initial Guess: $x_0 = [-30, -30]$

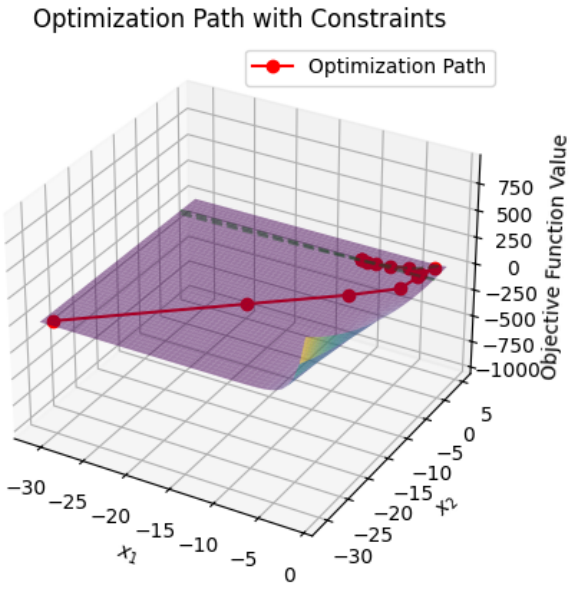


Figure 4: Visualization of the path followed by the solver

```
Initial guess: [-30 -30],
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.02355037962417503
       x: [-9.547e+00  1.047e+00]
     nit: 13
     jac: [ 1.840e-02 -2.284e-03]
    nfev: 41
    njev: 13, Time: 0.003003835678100586
```

Analysis Three out of four initial guesses led to successful optimization, converging to the same solution point with a minimal objective function value of approximately 0.023550379. The convergence from $[-10, 1]$ was notably the most efficient in terms of iterations and time, likely because the start point is already very close to the optimal point. For $x_0 = [10, 20]$, the solver could not find a solution. The starting point is out of the feasible region and the value of the gradient is extremely high, so handling the step size and the direction

in these conditions could be problematic. If the solver cannot identify a step that potentially reduces the violation of constraints, it may conclude that the problem is unsolvable from that starting position.
Even if $[-30, -30]$ is more far from the optimal point than $[0, 0]$, it converged faster. With big initial steps, the solver gets already in the feasible area and close to the optimal point, leading in less number of iterations needed. The consistent finding of the same optimal point suggests a strong likelihood of it being the global minimum, especially given the convergence from diverse starting points.

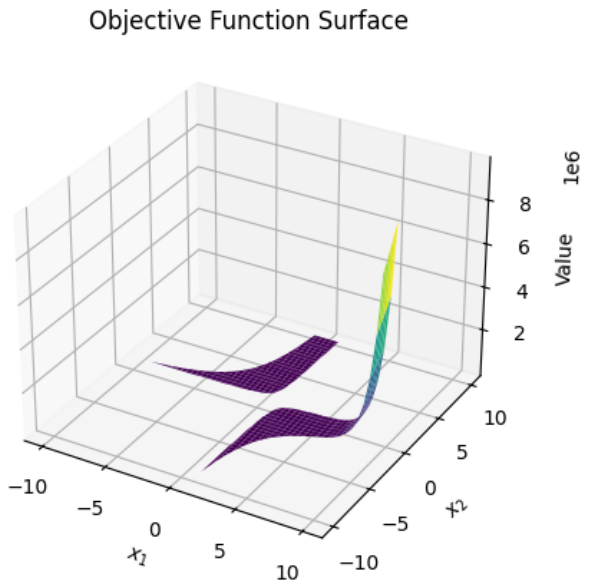


Figure 5: Visualization of the objective function's surface

c Give as input the Jacobian to the method, and repeat and check whether the method speeds up.

By passing the Jacobian as an input to *scipy.optimize.minimize*, the solver can save some computation time, and the execution should be faster.

```
Time difference for [0 0]: -0.012583732604980469
Time difference for [10 20]: -0.0008003711700439453
Time difference for [-10   1]: -0.0005545616149902344
Time difference for [-30 -30]: 0.00012922286987304688
```

For the first three initial guesses, the solver is indeed slightly faster. For the last one, interestingly, we see a slowdown.

```
Initial guess: [-30 -30],
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 3.0607730187565383
       x: [ 1.182e+00 -1.740e+00]
     nit: 45
     jac: [ 1.122e+01 -7.473e-01]
    nfev: 119
    njev: 44, Time: 0.003133058547973633
```

Analyzing the results, the solver returns a different solution from the previous one, with a different number of iterations (explaining the slowdown). The Jacobian should be approximately the same, but given how far the starting point is from the solution, even a tiny change can cause a significant change in step size or direction.

Exercise 2

a Identify whether is convex or not.

a.1 Convex Set

If each constraint is a convex set, the intersection is also a convex set.

1. The set defined by $-x_0 + 0.5 \leq 0$ (equivalently, $x_0 \geq 0.5$) is convex, as it defines a half-space in $\mathbb{R}^n$ (or a half-line in $\mathbb{R}$), which is always convex.

2. The set defined by $-x_0 - x_1 + 1 \leq 0$ (or $x_0 + x_1 \leq 1$) is convex. This constraint defines a half-space in $\mathbb{R}^2$, bounded by the line $x_0 + x_1 = 1$. Half-spaces are convex.

3. The set defined by $-x_0^2 - x_1^2 + 1 \leq 0$ represents the interior and boundary of a circle, so is a convex set.

4. Similar to the third item, the set defined by $-9x_0^2 - x_1^2 + 9 \leq 0$ represents the interior and boundary of an ellipse. Ellipses (and their interiors) are convex sets.

5. The set defined by $-x_0^2 + x_1 \leq 0$ (or $x_1 \leq x_0^2$) is not convex. This set includes all the points below the parabola $x_1 = x_0^2$. The region below a parabola is not convex because it does not contain all line segments connecting any two points within the set.

6. Similar to the fifth item, the set defined by $-x_1^2 + x_0 \leq 0$ (or $x_0 \leq x_1^2$) is not convex. It describes the region below the parabola $x_0 = x_1^2$, which does not satisfy the definition of a convex set.

Summary: The sets defined by the first four constraints are convex. The sets defined by the last two constraints are not convex, so the intersection might be not convex.

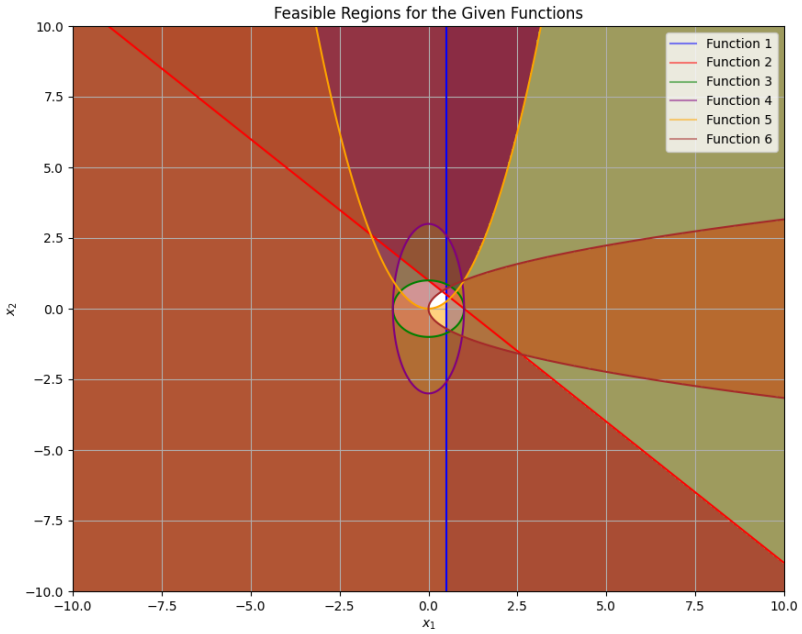


Figure 6: Feasible region (green area)

The feasible region (green area in Figure 6) is not a convex set.

a.2 Convex Function

The objective function $x_1^2 + x_2^2 \in C^\infty$, we can apply the Hessian matrix test:

```
Hessian of objective_function:
[[2. 0.]
 [0. 2.]]
Eigenvalues of the Hessian:
[2. 2.]
objective_function Hessian is: Positive definite
```

so the function is convex.

b Propose an initial point that is not feasible and an initial point that is feasible and check what happens with SLSQP as method.

b.1 Initial Guess: $x_0 = [5, 5]$

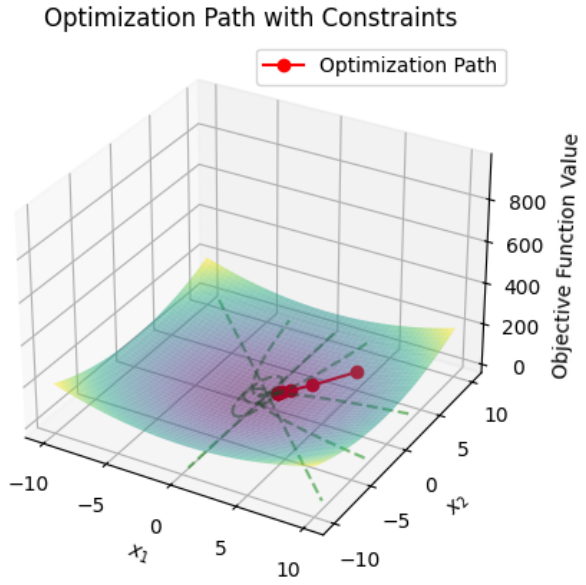


Figure 7: Visualization of the path followed by the solver

```
Initial guess: [5 5]
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 2.000002171193943
       x: [ 1.000e+00  1.000e+00]
     nit: 7
     jac: [ 2.000e+00  2.000e+00]
    nfev: 22
    njev: 7
Time: 0.004537820816040039
```

b.2 Initial Guess: $x_0 = [5, -2.5]$

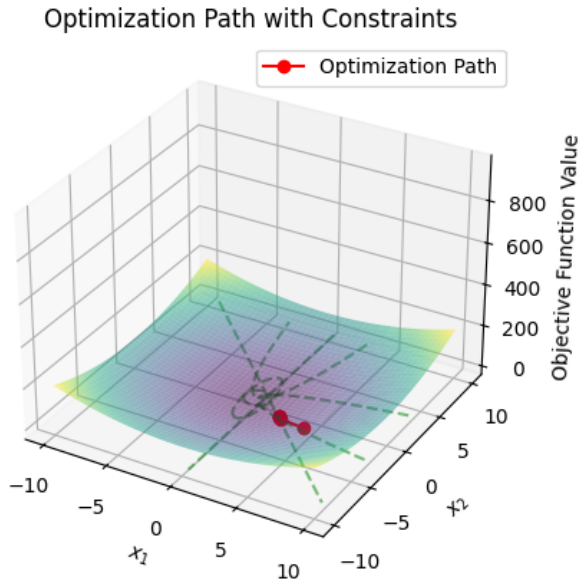


Figure 8: Visualization of the path followed by the solver

```
Initial guess: [ 5.  -2.5]
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 9.472135471952454
       x: [ 2.618e+00 -1.618e+00]
     nit: 5
     jac: [ 5.236e+00 -3.236e+00]
    nfev: 21
    njev: 5
Time: 0.0018379688262939453
```

b.3 Initial Guess: $x_0 = [5, -7.5]$

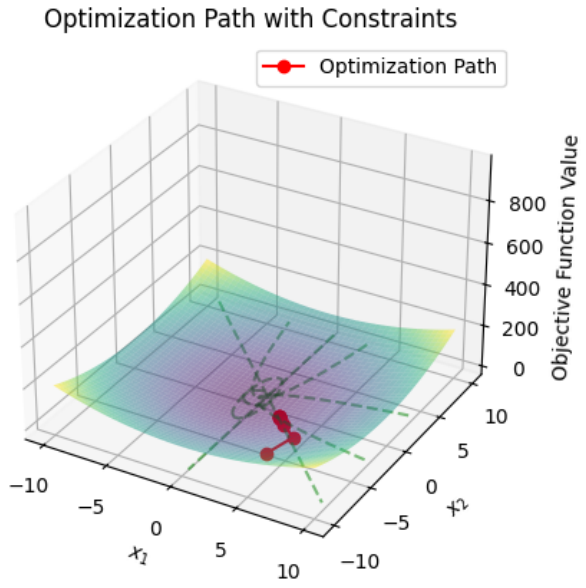


Figure 9: Visualization of the path followed by the solver

```
Initial guess: [ 5.  -7.5]
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 9.472135954887143
       x: [ 2.618e+00 -1.618e+00]
     nit: 9
     jac: [ 5.236e+00 -3.236e+00]
    nfev: 24
    njev: 8
Time: 0.009113073348999023
```

b.4 Initial Guess: $x_0 = [-100, -100]$

```
Initial guess: [-100 -100]
 message: Inequality constraints incompatible
 success: False
  status: 4
```

```
     fun: 0.009267617323262199
       x: [-7.000e-02 -6.609e-02]
     nit: 28
     jac: [-1.400e-01 -1.322e-01]
    nfev: 113
    njev: 26
Time: 0.01726984977722168
```

Analysis The feasible set is divided in two main areas.

- Starting with $x_0 = [5, 5]$, belonging to the upper area, leads to the global optimum in few iterations.
- With $[5, -2.5]$ as initial point, which belongs two the lower area, the solver returns a local minimum in few iterations, but it's not the global minimum.
- Using $x_0 = [5, -7.5]$, which is non-feasible for the constraints, but is close to the lower area, still returns the local minimum.
- Finally, for $x_0 = [-100, -100]$, which is very far from any feasible area, the solver cannot return a solution.

c Repeat giving as input the Jacobian. Check the convergence time (or number of steps).

```
Time difference for initial guess [5 5]:
    -0.0004226000019116327
Time difference for initial guess [ 5.  -2.5]:
    -0.0009024999890243635
Time difference for initial guess [ 5.  -7.5]:
    -0.0008933000062825158
Time difference for initial guess [-100 -100]:
    0.003865199993015267
```

Except for the last one (where a solution has not been returned), giving the Jacobi as an input to the solver makes again the computation faster. Here is a matter of milliseconds, so it would be interesting to see the effect on more complex and time demanding problems.

Exercise 3

a Say whether it is a COP

a.1 Convex Set

If each constraint is a convex set, the intersection is also a convex set.

1. The set defined by $x_1^2 + x_1x_2 + x_2^2 \leq 3$ is convex. This constraint describes a level set of a quadratic form, which is convex because the associated matrix is positive semidefinite.

2. The set defined by $3x_1 + 2x_2 \geq 3$ (equivalently, $-3x_1 - 2x_2 + 3 \leq 0$) is convex. This constraint defines a half-space in $\mathbb{R}^2$, which is always convex.

Summary: Both defined sets are convex, and thus, their intersection is also convex.

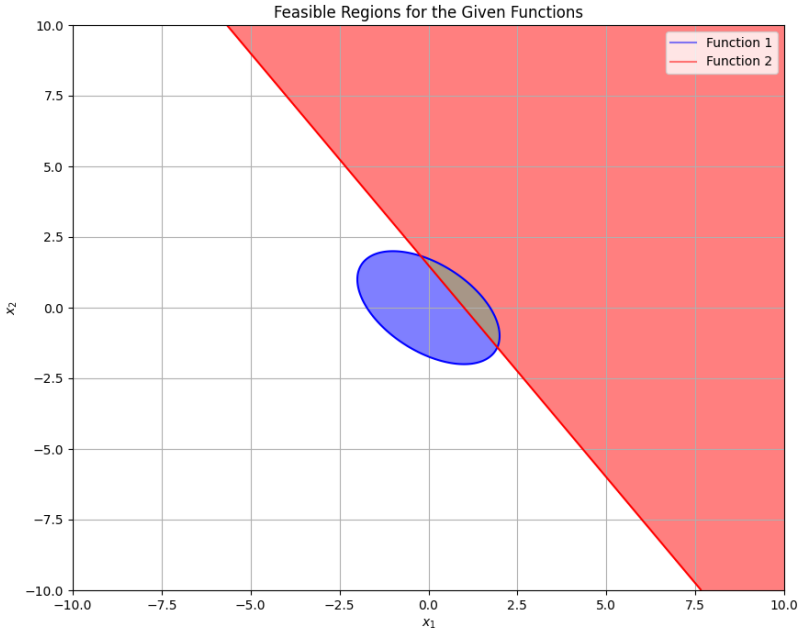


Figure 10: Feasible region (green area)

The feasible region (green area in Figure 10) is a convex set.

a.2 Convex Function

The objective function $x_1^2 + x_2^2 \in C^\infty$, we can apply the Hessian matrix test:

```
Hessian of objective_function:
[[2. 0.]
 [0. 2.]]
Eigenvalues of the Hessian:
[2. 2.]
objective_function Hessian is: Positive definite
```

so the function is convex.
Since both the set and the function are convex, a local minimum is also guaranteed to be a global minimum.

b Solve it using the scipy library. Check convergence.

Even for extreme starting points like $x_0 = [5000, 5000]$, $x_0 = [-1000, -1000]$, $x_0 = [-100000000, 100000]$, the algorithm converges to the optimal solution (global minimum) $x = [0.6923, 0.4615]$ in few iterations.

c Learn how to use the CVX toolbox to program it. Obtain also the Lagrange multipliers and provide the dual solution using CVX.

The solver doesn't directly accept the first constraint because the term $x_1 * x_2$ might lead to non-convexity, so it doesn't respect the Disciplined Convexity Programming (DCP) rules.
The constraint is initially given as:

$$x_1^2 + x_1 x_2 + x_2^2 \leq 3$$

It can be rewritten in a different, equivalent form by completing the square:

$$\left(x_1 + \frac{1}{2}x_2\right)^2 + \frac{3}{4}x_2^2 \leq 3$$

So that now it is DCP compliant and the solver can correctly compute a solution for this problem (Appendix C).

```
Solution: x1 = 0.6923095984422479, x2 =
    0.46153560262533166
Objective value: 0.6923076925859947
Dual variables: 9.995546904517202e-10,
    0.4615463540786608
```

- For the constraint $x_1^2 + x_1 x_2 + x_2^2 \leq 3$, the Lagrange multiplier is approximately 9.9955×10^{-10}. This value suggests that the constraint has almost no impact on the optimal objective value. In fact, from the feasible region (Figure 10), we can already intuit how moving away from the line defined by constraint number 2 (while still remaining in red area) increases either x_1 or x_2, worsening the objective function.
- For the constraint $3x_1 + 2x_2 \geq 3$, the Lagrange multiplier is approximately 0.4615. This nonzero value indicates that the constraint is active and influences the solution significantly.

In fact, relaxing the problem by removing the first constraint returns approximately the same solution.

```
Solution: x1 = 0.6923076923076924, x2 =
    0.46153846153846156
Objective value: 0.6923076923076925
Dual variables: 0.4615384615384616
```

Exercise 4

a Try to solve the problem by hand, giving the primal and dual solutions.

This is a convex optimization problem because the objective function $f(x) = x^2+1$ is convex (its second derivative is positive everywhere), and the constraint defines a convex set (an interval).
Since the problem is convex and there is at least one point that satisfies Slater's condition (for example $x = 3$), the problem has strong duality. This means that the solution obtained through the primal problem should be equal to the solution obtained through the dual problem.

a.1 Primal Solution.

The primal problem (Appendix B) is given as:

`minimize` $f(x) = x^2 + 1$
`subject to` $g(x) = (x-2)(x-4) \leq 0$

The constraint $(x-2)(x-4) \leq 0$ is satisfied for $2 \leq x \leq 4$.
Since x has to be always positive, the objective function is equivalent to:

`minimize` $f(x) = x$

So the primal solution is $x = 2$, with an objective value of $p^* = 5$

a.2 Dual Solution

The Lagrangian of the primal problem is:

$$L(x, \lambda) = f(x) + \lambda \cdot g(x) = x^2 + 1 + \lambda \cdot ((x-2)(x-4)), \quad \lambda \geq 0$$

which is a lower bound of $f(x)$, so the Lagrange dual function:

$$g(\lambda) = \inf_x \left(x^2 + 1 + \lambda \cdot ((x-2)(x-4))\right)$$

is a lower bound of the optimal value.
To find the infimum, we differentiate $L(x, \lambda)$ with respect to x and set the derivative equal to zero:

$$\frac{\partial L}{\partial x} = 2x + \lambda \cdot (2x - 6) = 0 \tag{1}$$

$$x^* = \frac{3\lambda}{\lambda + 1} \tag{2}$$

Substituting x^* into the original equation and simplifying gives:

$$g(\lambda) = \frac{-\lambda^2 + 9\lambda + 1}{\lambda + 1}$$

Since $g(\lambda)$ is a lower bound of the optimal value and the strong duality holds for this problem, maximizing $g(\lambda)$ should return the same objective value as the primal problem.

$$\frac{d}{d\lambda} g(\lambda) = 0 \quad \Longrightarrow \quad \lambda = -4, \lambda = 2 \tag{1}$$

Since by definition $\lambda \geq 0$, $\lambda = -4$ is discarded and the solution is $\lambda = 2$.

$$g(2) = p^* = 5 \tag{2}$$

b Use the CVX toolbox to program it. Obtain the Lagrange multipliers and the dual solution.

The primal problem is easily modeled with CVXPY and returns the outputs:

```
Solution: x = 1.9999999968611173
Objective value: 4.999999987444469
Lagrange multiplier: 2.0000322081789013
```

Formulating the dual problem and solving it in CVXPY is not as straightforward as for the primal one, as the objective function is not DCP compliant. However, keeping in mind that $\lambda \geq 0$, the dual problem aligns with the principles of Disciplined QuasiConvex Programming (DQCP), and can be solved in CVXPY with `qcp=True`.

```
Optimal lambda: 2.000034296330241
Optimal dual value: 4.999999999607925
```

The output confirms the holding of the strong duality between the two problems.

Exercise 5

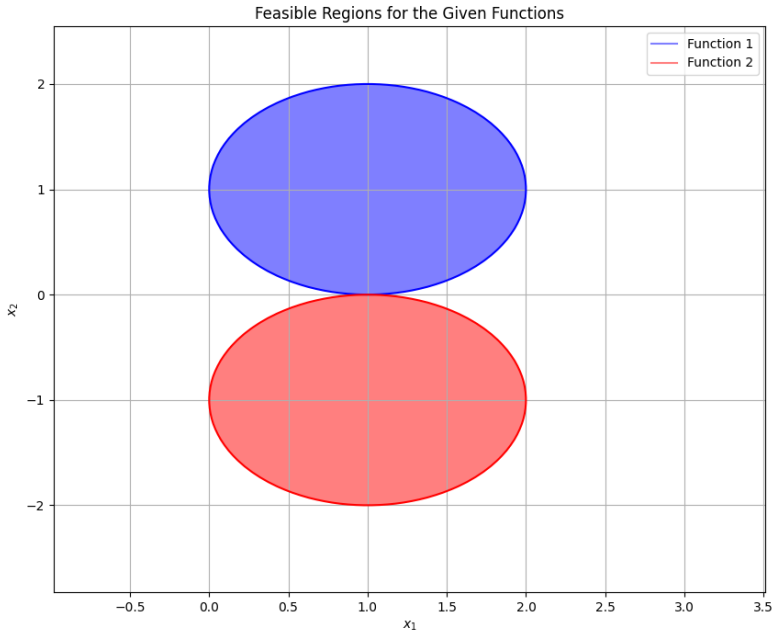


Figure 11: Feasible region (green area)

From the feasible region, it seems like the feasible set is only one point (intersection of the two constraints).
Let's verify it analytically by searching for intersections between the boundaries of the two ellipses.
Given the equations of two ellipses:

$$(x_0 - 1)^2 + (x_1 - 1)^2 = 1 \tag{1}$$
$$(x_0 - 1)^2 + (x_1 + 1)^2 = 1 \tag{2}$$

To find the intersection, observe that subtracting (2) from (1) gives:

$$(x_1 - 1)^2 - (x_1 + 1)^2 = 0 \tag{3}$$

Expanding both squares:

$$x_1^2 - 2x_1 + 1 - (x_1^2 + 2x_1 + 1) = 0 \tag{4}$$

Simplifying:

$$-4x_1 = 0 \tag{5}$$

Thus, we find:

$$x_1 = 0 \tag{6}$$

Having found x_1, we substitute $x_1 = 0$ back into either of the original ellipse equations to solve for x_0. Using equation (1):

$$(x_0 - 1)^2 + (0 - 1)^2 = 1 \tag{7}$$

$$(x_0 - 1)^2 + 1 = 1 \tag{8}$$

$$(x_0 - 1)^2 = 0 \tag{9}$$

Therefore:

$$x_0 = 1 \tag{10}$$

Thus, the ellipses intersect at exactly one point: $(1, 0)$, and this is the solution to the optimization problem.

a Use CVXPY to program the optimization problem.

The CVXPY solver, with this problem implementation (Appendix C), returns in fact the approximated value of the only point in the domain:

```
Optimal values:
x1 = 0.9999802295393706
x2 = 1.9912424211981957e-14
Minimum objective value = 0.9999604594696123
```

b If possible, provide the dual solution using CVX.

In this case the Slater's condition does not hold, so it's not sure whether strong duality holds or not.
Given the primal problem with the objective function to minimize $f(x) = x_1^2 + x_2^2$ and constraints:

$$g_1(x) = (x_1 - 1)^2 + (x_2 - 1)^2 - 1 \leq 0,$$
$$g_2(x) = (x_1 - 1)^2 + (x_2 + 1)^2 - 1 \leq 0,$$

the Lagrangian is given by:

$$L(x, \lambda) = x_1^2 + x_2^2 + \lambda_1 \left((x_1 - 1)^2 + (x_2 - 1)^2 - 1\right) + \lambda_2 \left((x_1 - 1)^2 + (x_2 + 1)^2 - 1\right)$$

where λ_1 and λ_2 are the dual variables associated with the constraints. The dual function $g(\lambda)$ is the infimum of the Lagrangian over x, i.e., $g(\lambda) = \inf_x L(x, \lambda)$, subject to $\lambda \geq 0$. The dual problem is then to maximize $g(\lambda)$ with respect to λ, under the constraint that $\lambda \geq 0$.
The infimum of the Lagrangian $L(x, \lambda)$ over x, after substituting the critical points back into L, is given by:

$$g(\lambda) = \inf_x L(x, \lambda) = \frac{-\lambda_1^2 + 2\lambda_1\lambda_2 + \lambda_1 - \lambda_2^2 + \lambda_2}{\lambda_1 + \lambda_2 + 1} = \frac{-(\lambda_1 - \lambda_2)^2 + \lambda_1 + \lambda_2}{\lambda_1 + \lambda_2 + 1}$$

This expression represents the dual function $g(\lambda)$, which is to be maximized with respect to λ_1 and λ_2, subject to $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$.
The formulation is not DCP compliant but, given the structure of the numerator and that the denominator is always positive ($\lambda_1, \lambda_2 \geq 0$), we can solve it as a Quasiconvex function:

```
Lambda 1: 188255.57602409448
Lambda 2: 188255.57602409448
Optimal dual value: 0.9999973440432279
```

Even if the Slater's condition is not met, the dual problem still returns a objective value approximately equal to the one returned from the primal problem.

Exercise 6

Let's analyze and compare the results of two different algorithms, the Gradient Descent (Appendix E) and the Newton Method (Appendix F). The convergence rate of the Gradient Descent method is linear and it only requires the first derivative (which is calculated in-place by the solver in our implementation). The Newton Method can converge faster (quadratically in the best scenario), because it also takes into account the curvature of the function. For this reason, the second derivative is also required, which is computed by the solver in this case.

a Function 1

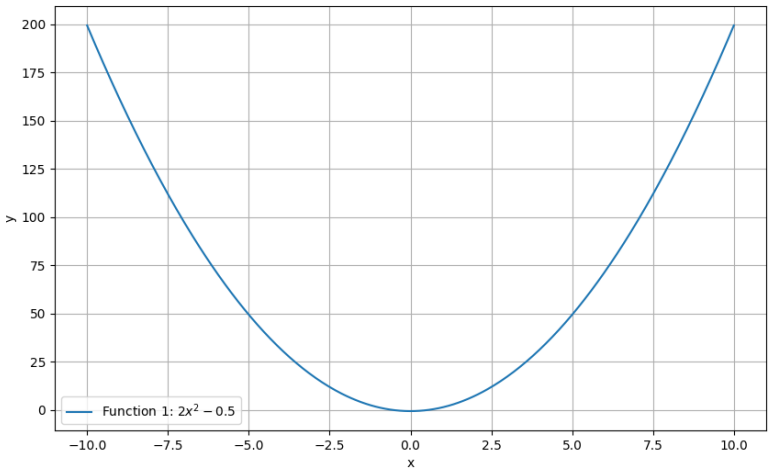


Figure 12: Function 1

The function is convex, both methods should return the global optimum.

```
Solver type: Gradient Descent
Starting point (x0): 3
Optimized variable (x): -0.0008388977607844657
Function value at optimal point (f(x)): [-0.49999859]
Number of iterations performed: 7
Time taken for optimization: 0.008015155792236328 seconds
Final accuracy: 0.0008388977607844657
```

```
Solver type: Newton
Starting point (x0): 3
Optimized variable (x): -5.5511240049099805e-09
Function value at optimal point (f(x)): -0.49999999999999994
Number of iterations performed: 2
Time taken for optimization: 0.0021941661834716797 seconds
Final accuracy: 5.5511240049099805e-09
```

Analysis

- For this simple convex function, Newton Method returns a better result (basically the theoretical optimum) in only 2 iterations, opposed to the 7 iterations of the Gradient Descent, confirming its faster convergence.
- It seems that the calculation of the second derivative does not significantly impact the solving time, though the observed durations are too short to draw definitive conclusions

b Function 2

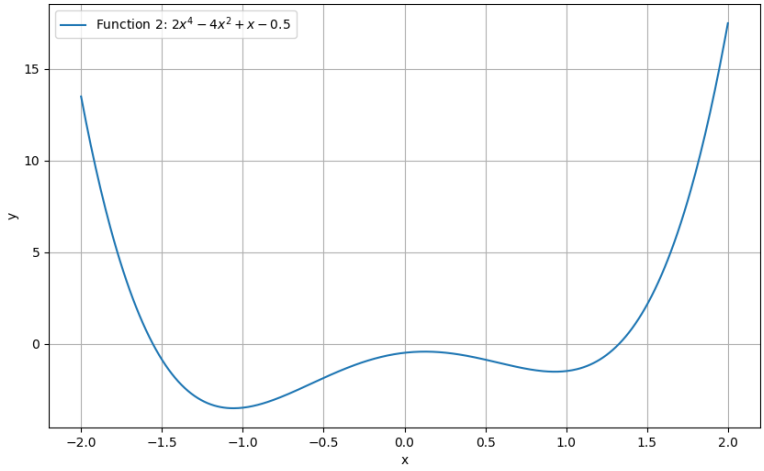


Figure 13: Function 2

The function is not convex, the local minimum returned by the solvers is not guaranteed to be a global minimum.

b.1 Starting point $x_0 = -2$

```
Solver type: Gradient Descent
Starting point (x0): -2
Optimized variable (x): -1.0578326991065417
Function value at optimal point (f(x)): [-3.52950593]
Number of iterations performed: 6
Time taken for optimization: 0.000518798828125 seconds
Final accuracy: 0.0003789291065416922
```

```
Solver type: Newton
Starting point (x0): -2
Optimized variable (x): -1.0574539265150036
Function value at optimal point (f(x)): -3.5295072825511826
Number of iterations performed: 5
Time taken for optimization: 0.0036668777465820312 seconds
Final accuracy: 1.5651500362423576e-07
```

b.2 Starting point $x_0 = -0.5$

```
Solver type: Gradient Descent
Starting point (x0): -0.5
Optimized variable (x): -1.0579679440589018
Function value at optimal point (f(x)): [-3.52950479]
Number of iterations performed: 5
Time taken for optimization: 0.0015430450439453125 seconds
Final accuracy: 0.0005141740589018617
```

```
Solver type: Newton
Starting point (x0): -0.5
Optimized variable (x): 0.930402924878493
Function value at optimal point (f(x)): -1.5334970165704984
Number of iterations performed: 6
Time taken for optimization: 0.00400853157043457 seconds
Final accuracy: 1.9878566948784928
```

b.3 Starting point $x_0 = 0.5$

```
Solver type: Gradient Descent
Starting point (x0): 0.5
Optimized variable (x): 0.9315737457517378
Function value at optimal point (f(x)): [-1.53348825]
Number of iterations performed: 5
Time taken for optimization: 0.001244215045683572 seconds
Final accuracy: 1.9890275157517379
```

```
Solver type: Newton
Starting point (x0): 0.5
Optimized variable (x): 0.9304029237694402
Function value at optimal point (f(x)): -1.5334970165704982
Number of iterations performed: 7
Time taken for optimization: 0.0037240982055664062 seconds
Final accuracy: 1.9878566937694402
```

b.4 Starting point $x_0 = 2$

```
Solver type: Gradient Descent
Starting point (x0): 2
Optimized variable (x): 0.9318157364245575
Function value at optimal point (f(x)): [-1.53348425]
Number of iterations performed: 4
Time taken for optimization: 0.001024007797241211 seconds
Final accuracy: 1.9892695064245576
```

```
Solver type: Newton
Starting point (x0): 2
Optimized variable (x): 0.9304159457907758
Function value at optimal point (f(x)): -1.5334970154877476
Number of iterations performed: 5
Time taken for optimization: 0.0035333633422851562 seconds
Final accuracy: 1.9878697157907759
```

Analysis

- For this non-convex function, different solutions are obtained from different starting points. The function has two local minima: in general, the local minimum returned is going to be the one closer to the starting point assigned to the method. Interestingly, for $x_0 = -0.5$ the two methods return two different local optimum, so the second derivative could have played a significant role in the solving path.
  Given the function:

  $$f(x) = 2x^4 - 4x^2 + x - 0.5$$

  First derivative ($f'(x_0)$):

  $$f'(x_0) = 8x_0^3 - 8x_0 + 1 = 4.0$$

  Second derivative ($f''(x_0)$):

  $$f''(x_0) = 24x_0^2 - 8 = -2.0$$

  For Gradient Descent, the first step is determined by:

  $$x_{\text{new}} = x - t \cdot f'(x_0)$$

  so from the starting point $x_0 = -0.5$, the solver moves to the left.

  For Newton's method, the first step is determined by:

  $$x_{\text{new}} = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

  Therefore, for the first step the solver moves to the right, with the concavity around the starting point being responsible for the change in direction with respect to Gradient Descent.

- The number of iterations is comparable between the two methods for all the starting points, showing that the quadratic convergence of Newton's method happens mostly in ideal scenarios.

- Final accuracies are only meaningful when the global optimum is reached. For starting point $x_0 = -2$, Newton turned out to be very close to the theoretical optimum, while Gradient Descent remains in the stopping criterion range of precision $\eta = 10^{-4}$.
- For every starting point, the in-place calculation of the second derivative seems to have a significant time impact, resulting in slower solving time for the Newton Method.

Exercise 7

a Specify the network utility maximization convex optimization problem.

```
# Define the capacities of the links
C = np.array([1, 2, 1, 2, 1])

# Define the matrix A based on the routes of each source
A = np.array([[1, 0, 1],
              [1, 1, 0],
              [0, 0, 0],
              [0, 0, 0],
              [0, 0, 1]])

# Number of sources
num_sources = A.shape[1]

# Decision variables for the rates of each source
x = cp.Variable(num_sources)

# Objective function: Sum of utilities of all sources
objective = cp.Maximize(cp.sum(cp.log(x)))

# Constraints: Ax <= C and x >= 0
constraints = [A @ x <= C, x >= 0]

# Problem definition
problem = cp.Problem(objective, constraints)

# Solve the problem
problem.solve()

# Results
x_optimal = x.value
lagrange_multipliers = constraints[0].dual_value
```

b Solve the problem using CVX.

The bottleneck of the problem is the L_1 link, which is used by both S_0 and S_2 and it has only 1 of capacity.

```
Optimal rates for each source: [0.42264971 1.57735029
    0.57735029]
Optimal value of the objective function: -0.95477125606744
```

As expected, the majority of the network rate is assigned to S_1, the only source that does not traverse the L_1 link.

c Obtain the Lagrange multipliers.

Since the problem is convex (maximization of a concave function over a convex set) and Slater's condition is respected (e.g. for $x = [0.1, 0.1, 0.1]$), strong duality is expected to hold.
Lagrange multipliers give us insights on how sensitive is the objective function to each constraint:

```
Lagrange multipliers for each link (dual solution):
    [1.73205063e+00 6.33974601e-01 1.41565492e-09 7.35642950e
    -10 3.25763413e-09]
```

As expected, since L_3 and L_4 are not part of any route, their value is approximately 0. The impact of L_5 is also negligible, since a relaxation of this constraint would not improve the objective function, given the bottleneck on the L_1 link. The bottleneck is also confirmed by λ_1 being the highest value.

Exercise 8

a Give the traffic allocated to each user and the percentage of time each link is used.

For the given constraints and objective (Appendix D), the results are:

```
Optimal traffic allocation (x): [0.16666663 0.33333336
    0.33333338]
Optimal time fractions (T): [0.49999999 0.16666663
    0.33333338]
```

b Dual problem

Since the problem is convex (maximization of a concave function over a convex set) and Slater's condition is respected (e.g. for $x = [0.1, 0.1, 0.1]$, $T = [0.25, 0.25, 0.25]$), strong duality is expected to hold.

```
Dual variables:
Constraint 1 dual variable: 2.999999270930079
Constraint 2 dual variable: 2.9999992496709327
Constraint 3 dual variable: 2.999999265944279
Constraint 4 dual variable: 2.999999279889642
Constraint 5 dual variable: [3.43454870e-08 1.63869302e-08
    1.64278155e-08]
Constraint 6 dual variable: [1.14056513e-08 3.26667962e-08
    1.63923931e-08]
```

All the Lagrange multipliers of constraints related to limitations of the traffic have approximately the same value.
Given that the objective is to maximize the traffic, relaxation of non-negativity constraints 5 and 6 would have no effect on the objective value.

A CVXPY implementation for exercise 3

```
# Define the variables
x1 = cp.Variable()
x2 = cp.Variable()

# Define the objective
objective = cp.Minimize(cp.square(x1) + cp.square(x2))

# Define the constraints
z = (x1 + 0.5*x2)
constraints = [cp.square(z) + 0.75*cp.square(x2) <= 3, 3*x1
    + 2*x2 >= 3]

# Define the problem and solve it
problem = cp.Problem(objective, constraints)
problem.solve()
```

B CVXPY implementation for exercise 4

```
# Define the variables
x = cp.Variable()

# Define the objective
objective = cp.Minimize(cp.square(x) + 1)

# Define the constraints
constraints = [cp.square(x) - 6*x + 8 <= 0]

# Define the problem and solve it
problem = cp.Problem(objective, constraints)
problem.solve()
```

C CVXPY implementation for exercise 5

```
# Define variables
x1 = cp.Variable()
x2 = cp.Variable()

# Objective function
objective = cp.Minimize(x1**2 + x2**2)

# Constraints
constraints = [
    cp.square(x1 - 1) + cp.square(x2 - 1) <= 1,
```

```
    cp.square(x1 - 1) + cp.square(x2 + 1) <= 1,
]

# Define problem
problem = cp.Problem(objective, constraints)

# Solve problem
problem.solve()
```

D CVXPY implementation for exercise 8

```
# Define the variables
x = cp.Variable(3) # Users 1, 2, and 3
T = cp.Variable(3) # Time fractions for the links

# Define the objective
objective = cp.Maximize(cp.sum(cp.log(x)))

# Define the constraints
constraints = [
    x[0] + x[1] <= T[0], # x1 + x2 <= T12
    x[0] <= T[1],        # x1 <= T23
    x[2] <= T[2],        # x3 <= T32
    T[0] + T[1] + T[2] <= 1, # T12 + T23 + T32 <= 1
    x >= 0, # Traffic allocated must be non-negative
    T >= 0, # Time fractions must be non-negative
]

# Define the problem and solve
problem = cp.Problem(objective, constraints)
problem.solve()
```

E Gradient Descent Algorithm

```
def gradient_descent(func, x0, grad_func=None, alpha=0.3,
    beta=0.8, precision=1e-4, max_iterations=1000, epsilon=1e
    -8):
    start_time = time.time()
    x = np.array([x0])
    for i in range(max_iterations):
        if grad_func is not None:
            grad = grad_func(x)
        else:
            grad = approx_fprime(x, func, epsilon)
```

```
        t = 1.0
        while func(x - t * grad) > func(x) - alpha * t * np.
            dot(grad, grad):
            t *= beta
        x_new = x - t * grad
        if np.abs(func(x_new) - func(x)) < precision:
            end_time = time.time()
            print_results(x0, x_new[0], func(x_new), i + 1,
                end_time - start_time, "Gradient Descent")
            return x_new[0], func(x_new), i + 1
        x = x_new

    end_time = time.time()
    print_results(x0, x[0], func(x), max_iterations,
        end_time - start_time, "Gradient Descent")
    return x[0], func(x), max_iterations
```

F Newton Method

```
def newton_method(f, x0, f_prime=None, f_double_prime=None,
    precision=1e-4, max_iterations=1000, epsilon=1e-8):
    start_time = time.time()
    x = x0
    for i in range(max_iterations):
        if f_prime is None:
            grad = approx_fprime(np.array([x]), f, epsilon)
                [0]
        else:
            grad = f_prime(x)

        if f_double_prime is None:
            H = nd.Hessian(f)([x])[0][0]
        else:
            H = f_double_prime(x)

        if H == 0:
            print("Hessian is zero, stop iteration.")
            break

        x_new = x - grad / H
        if np.abs(f(x_new) - f(x)) < precision:
            end_time = time.time()
            print_results(x0, x_new, f(x_new), i + 1,
                end_time - start_time, "Newton")
            return x_new, f(x_new), i + 1
        x = x_new
```