Sharif University of Technology
Electrical Engineering Department

Convex Optimization
CHW 2

Amir Hossein Yari
99102507

May 19, 2023

Contents

1. Maximum Likelihood Estimation

1. First Part

For each hour t, we have the following likelihood function:

$$L(\lambda_t) = P[N_t = n_t] = e^{-\lambda_t} \frac{\lambda_t^{n_t}}{n_t!}$$

$$\ln(L(\lambda_t)) = -\lambda_t + n_t \ln(\lambda_t) - \ln(n_t!)$$

Case 1: $N_t = 0$ (no events occurred at hour t) In this case, the likelihood function simplifies to $P[N_t = 0] = e^{-\lambda_t}$. To maximize this likelihood, we set λ_t to its minimum value of 0.
$\Rightarrow \lambda_t = 0$, if $n_t = 0$
Case 2: $N_t > 0$ (events occurred at hour t)

$$\frac{\partial}{\partial \lambda_t} \ln(L(\lambda_t)) = -1 + \frac{n_t}{\lambda_t} = 0$$

$$\Rightarrow n_t = \lambda_t$$

So in general it can be said $\lambda_t = n_t$

2. Second Part

$$\text{Maximize } \sum_{t=1}^{24} \ln(L(\lambda_t)) - \rho(\sum_{t=1}^{23} (\lambda_{t+1} - \lambda_t)^2 + (\lambda_1 - \lambda_{24})^2)$$

$$\text{Maximize } \sum_{t=1}^{24} -\lambda_t + n_t \ln(\lambda_t) - \ln(n_t!) - \rho(\sum_{t=1}^{23} (\lambda_{t+1} - \lambda_t)^2 + (\lambda_1 - \lambda_{24})^2)$$

The above function is concave. So negative of this function is convex and by minimize it, we achieve to convex optimization problem.
The optimization is without constraint, So by calculate gradient of objective function, we can calculate the optimal values.

3. Third Part

As a result of setting ρ to infinity, it increases the effect of the regularization term and all λ_t will be equal.

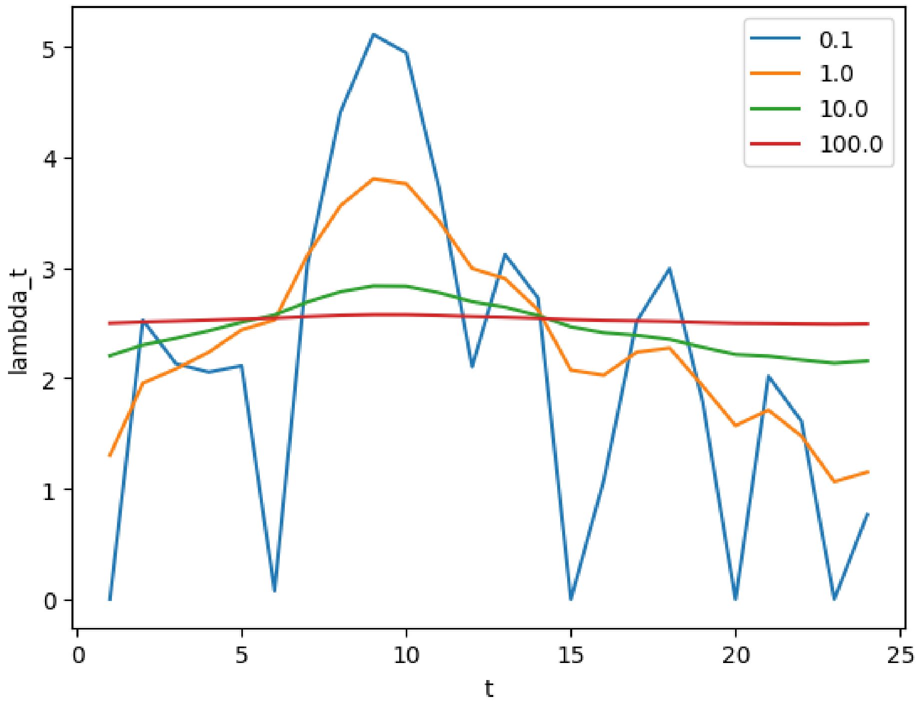
4. Fourth Part

```
# add required packages
import numpy as np
import cvxpy as cp
import matplotlib.pyplot as plt

def circular_differential_matrix(n):
circular_diff_matrix = np.zeros((n, n))
for i in range(n):
circular_diff_matrix[i, i] = -1
circular_diff_matrix[i, (i + 1) % n] = 1
return circular_diff_matrix

N = np.array([0,4,2,2,3,0,4,5,6,6,4,1,4,4,0,1,3,4,2,0,3,2,0,1])
rho = np.array([0.1,1,10,100])

for i in rho:
landa = cp.Variable(24)
obj_func = cp.Maximize(cp.sum(-landa + cp.multiply(N,cp.log(landa)) - cp.multiply(i,cp.
                                                sum((circular_differential_matrix(24)@landa)**
                                                2))))
constraints = []
prob = cp.Problem(obj_func, constraints)
prob.solve()
# plotting
plt.plot(range(1,25), landa.value, label = i)
plt.xlabel("t")
plt.ylabel("lambda_t")
plt.legend()
```



5. Fifth Part

The best ρ is 1.

```
rho( 0.1 ) : -54.89651503941711
rho( 1.0 ) : -7.090076886222096
rho( 10.0 ) : -11.056694980432539
rho( 100.0 ) : -13.104061134250635
```

2. Optimal Activity Levels

1. First Part

The optimization problem is as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} r_j(x_j) \\ \text{subject to} \quad & x \succeq 0 \\ & Ax \preceq c^{max} \end{aligned}$$

This is a convex optimization problem because the objective function is concave(maximize concave function(f) is equal to minimize convex function($-f$)) and the inequality constraints are linear so they are convex.
r_j has a lower bound $l_j = \min\{p_j x_j, p_j q_j + p_j^{disc}(x_j - q_j)\}$.
So $l_j \leqslant p_j x_j$ and $l_j \leqslant p_j q_j + p_j^{disc}(x_j - q_j)$.
Minimization preserve concavity, so optimization remains convex.
So, we can express problem as LP according to epigraph form as below.

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} l_j = \mathbf{1}^T l \\ \text{subject to} \quad & x \succeq 0 \\ & Ax \preceq c^{max} \\ & l_j \leqslant p_j x_j \\ & l_j \leqslant p_j q_j + p_j^{disc}(x_j - q_j) \end{aligned}$$

Two problems are equivalent.

2. Second Part

```python
# add required packages
import numpy as np
import cvxpy as cp

# initialize variables
A = np.array([[1, 2, 0, 1], [0, 0, 3, 1], [0, 3, 1, 1], [2, 1, 2, 5], [1, 0, 3, 2]])
c_max = np.array([100, 100, 100, 100, 100])
p = np.array([3, 2, 7, 6])
q = np.array([4, 10, 5, 10])
p_disk = np.array([2, 1, 4, 2])
```

```python
# solve problem
x = cp.Variable(4)
r_1 = cp.multiply(p,x)
r_2 = cp.multiply(p,q) + cp.multiply(p_disk,(x-q))
obj_func = cp.Maximize(cp.sum(cp.minimum(r_1,r_2)))
constraints = [A*x<=c_max, x>=0]
prob = cp.Problem(obj_func, constraints)
prob.solve()

# calculation of the demands of the problem and print it
print("optimal activity levels = ", x.value)
r = cp.minimum(r_1,r_2).value
print("revenue of each one = ", r)
total = np.sum(r)
print("total revenue = ", total)
avg_price = r / x.value
print("average price per unit for each activity = ", avg_price)
```

```
optimal activity levels =  [ 3.99999996 22.49999989 30.99999995  1.50000005]
revenue of each one =  [ 11.99999989  32.49999989 138.99999981   9.00000032]
total revenue =  192.49999991412406
average price per unit for each activity =  [3.         1.44444445 4.48387097 6.        ]
```

3. Optimal Vehicle Speed Scheduling

1. First Part

Total fuel consumption is $\sum_{i=1}^{n} \Phi(s_i)\, \frac{d_i}{s_i}$ and we want to minimize it.
So the optimization problem is :

$$\begin{aligned} \text{minimize} \;\; & \sum_{i=1}^{n} \Phi(s_i)\, \frac{d_i}{s_i} \\ \text{subject to} \;\; & s_i^{min} \leqslant s_i \leqslant s_i^{max} \\ & \tau_i^{min} \leqslant \tau_i \leqslant \tau_i^{max} \end{aligned}$$

We can formulate this problem as a convex problem by making a change of variables. If we define $t_i = \frac{d_i}{s_i}$, we have below convex optimization problem.

$$\begin{aligned} \text{minimize} \;\; & \sum_{i=1}^{n} \Phi(\frac{d_i}{t_i})\, t_i \\ \text{subject to} \;\; & \frac{d_i}{s_i^{max}} \leqslant t_i \leqslant \frac{d_i}{s_i^{min}} \\ & \tau_i^{min} \leqslant \sum_{i=1}^{n} t_i \leqslant \tau_i^{max} \end{aligned}$$

$\Phi(\frac{d_i}{t_i})\, t_i$ is perspective function of Φ and since Φ is convex and the objective function is positive weighted sum of convex functions, so the objective function is convex. The constraints are all linear in t, so they are convex.
Two problems are equivalent, so if we find $t_i^\star$ for second problem, according to $t_i = \frac{d_i}{s_i}$ we can find $s_i^\star$ of main problem.

2. Second Part

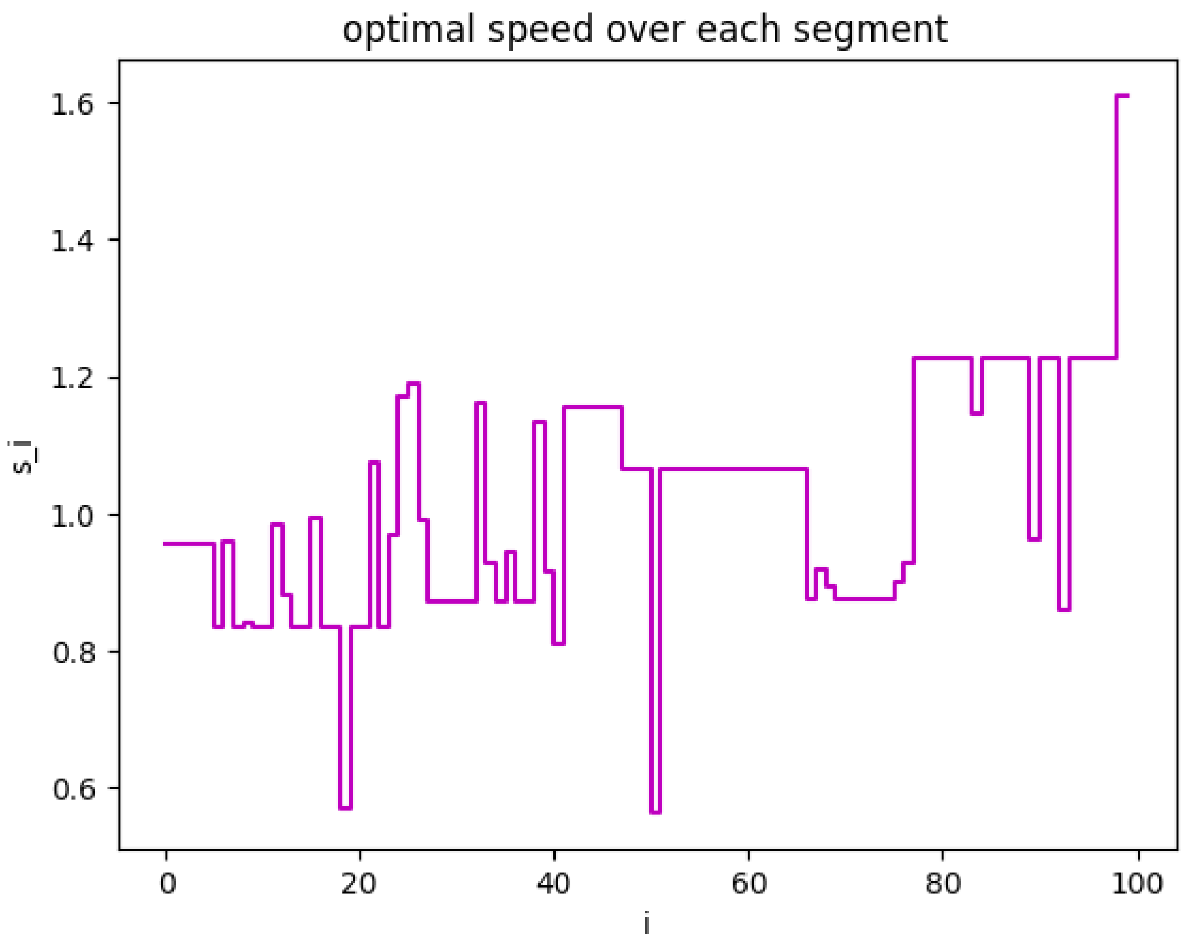
```
# add required packages
import cvxpy as cp
import matplotlib.pyplot as plt
from veh_speed_sched_data import *

# solve problem
t = cp.Variable(n)
obj_func = cp.sum(cp.multiply(a,cp.multiply(cp.multiply(d,d),cp.inv_pos(t))) + cp.
                                            multiply(b,d) + cp.multiply(c,t))
constraints = [t<=d/smin, t>=d/smax]
```

```
constraints += [tau_min[i] <= cp.sum(t[0:i+1]) for i in range(n)]
constraints += [tau_max[i] >= cp.sum(t[0:i+1]) for i in range(n)]
prob = cp.Problem(cp.Minimize(obj_func), constraints)
prob.solve()

# calculate optimal fuel consumption
s = d / t.value
print("The optimal fuel consumption is ", sum(s), "Kg")

# plot optimal speed over each segment
plt.step(range(n), s, "m")
plt.xlabel("i")
plt.ylabel("s_i")
plt.title("optimal speed over each segment")
```




```
The optimal fuel consumption is  2617.82519361969 Kg
```

4. Reformulate Constraint

- $\frac{1}{x} + \frac{1}{y} \leqslant 1$, $x \geqslant 0$, $y \geqslant 0$

We know that $\frac{1}{x}$ is not convex for $x \in R$ but it becomes convex by limiting the domain to R_{++}. So instead of coding $\frac{1}{x} + \frac{1}{y} \leqslant 1$, $x \geqslant 0$, $y \geqslant 0$, we can code as below.

```
import cvxpy as cp
x = cp.Variable(1)
y = cp.Variable(1)
constraints = [cp.inv_pos(x) + cp.inv_pos(y) <= 1]
prob = cp.Problem(cp.Minimize(obj_func), constraints)
prob.solve()
✓ 0.0s
```

The following image was taken from the www.cvxpy.org site and shows it covers all our constraint.

Function	Meaning	Domain	Sign	Curvature	Monotonicity
abs(x)	$\lvert x\rvert$	$x \in \mathbf{C}$	+ positive	∪ convex	↗ for $x \geq 0$ ↘ for $x \leq 0$
conj(x)	complex conjugate	$x \in \mathbf{C}$	± unknown	affine	None
entr(x)	$-x\log(x)$	$x > 0$	± unknown	∩ concave	None
exp(x)	e^x	$x \in \mathbf{R}$	+ positive	∪ convex	↗ incr.
huber(x, M=1) $M \geq 0$	$\begin{cases} x^2 & \lvert x\rvert \leq M \\ 2M\lvert x\rvert - M^2 & \lvert x\rvert > M \end{cases}$	$x \in \mathbf{R}$	+ positive	∪ convex	↗ for $x \geq 0$ ↘ for $x \leq 0$
imag(x)	imaginary part of a complex number	$x \in \mathbf{C}$	± unknown	affine	none
inv_pos(x)	$1/x$	$x > 0$	+ positive	∪ convex	↘ decr.

- $xy \geqslant 1$, $x \geqslant 0$, $y \geqslant 0$

$xy \geqslant 1$ is neither convex nor concave. So instead of coding $xy \geqslant 1$, $x \geqslant 0$, $y \geqslant 0$, we can code as below.

```python
import cvxpy as cp
x = cp.Variable(1)
y = cp.Variable(1)
constraints = [x >= cp.inv_pos(y)]
prob = cp.Problem(cp.Minimize(obj_func), constraints)
prob.solve()
```
✓ 0.0s

- $\frac{(x+y)^2}{\sqrt{y}} \leqslant x - y + 5,\ y \geqslant 0$

By dividing a convex function by a concave function, the result function is not guaranteed to have a specific convexity or concavity.
So instead of coding $\frac{(x+y)^2}{\sqrt{y}} \leqslant x - y + 5,\ y \geqslant 0$, we can code as below.

```python
import cvxpy as cp
x = cp.Variable(1)
y = cp.Variable(1)
constraints = [cp.quad_over_lin(x+y, cp.sqrt(y)) <= x-y+5]
prob = cp.Problem(cp.Minimize(obj_func), constraints)
prob.solve()
```
✓ 0.0s

- $x + z \leqslant 1 + \sqrt{xy - z^2},\ x \geqslant 0,\ y \geqslant 0$

xy is not concave, which causes problem. So according to $\sqrt{xy - z^2} = \sqrt{y(x - \frac{z^2}{y})}$.
So instead of coding $x + z \leqslant 1 + \sqrt{xy - z^2},\ x \geqslant 0,\ y \geqslant 0$, we can code as below.

```python
import cvxpy as cp
x = cp.Variable(1)
y = cp.Variable(1)
z = cp.Variable(1)
f = x - cp.quad_over_lin(z, y)
constraints = [(x + z) <= (1 + cp.geo_mean(cp.hstack([f, y])))]
prob = cp.Problem(cp.Minimize(obj_func), constraints)
prob.solve()
```
✓ 0.0s