Convex Optimization

Prof. Amiri

Electrical Engineering Department

Naser Kazemi 99102059

Practical Homework 1

December 1, 2023

Convex Optimization

Practical Homework 1

Naser Kazemi 99102059

Problem 1: *DCP Representation with CVXPY*

Solution

Since **dom** $f = \mathbb{R}^2_{++}$, we can express $f(x,y) = \frac{1}{xy}$ as:

$$f(x,y) = \left(\frac{1}{\sqrt{xy}}\right)^2$$

Now let g_1, g_2 and g_3 be defined as:

$$g_1(u,v) = \sqrt{uv} \ \ , \ \ g_2(u) = \frac{1}{u} \ \ , \ \ g_3(u) = u^2$$

There for we can present $f(x,y)$ as $f(x,y) = g_3(g_2(g_1(x,y)))$, which is the composition of 3 convex functions, and therefor it is convex itself.
Here you can see the CVXPY implementation of this problem, for minimizing the function $f(x,y)$.

```python
import cvxpy as cp
from cvxpy import power, inv_pos, geo_mean

x = cp.Variable()
y = cp.Variable()

def g_1(u, v): return geo_mean(cp.hstack([x, y]))
def g_2(u): return inv_pos(u)
def g_3(u): return power(u, 2)
objective = cp.Minimize(g_3(g_2(g_1(x, y))))
constraints = [0 <= x, 0 <= y]

prob = cp.Problem(objective, constraints)
prob.solve()

print("status: ", prob.status)
print("optimal value: ", prob.value)
print("optimal var: ", x.value, y.value)

```

With output:

```
status:  optimal
optimal value:  2.205505320857822e-06
optimal var:  673.3578772552374 673.3578772552374

```

Problem 2: *Optimizing a Set of Disks*

Solution

(1)

The only difference between these to problems is their objectives. So let's first we are solving the problem fro the sum of areas objectives. The objective function $\sum_{i=1}^{n} \pi r_i^2$ is convex w.r.t variables $c_1, \cdots, c_n$ and $r_1, \cdots, r_n$. Now as for the constraints, the two discs D_i and D_j having intersection, means that the distance between their centers is less than the sum of their ratios. i.e $\|c_i - c_j\|_2 \leq r_i + r_j$, which is convex, since the norm of difference is convex, and the sum is linear. The other constraints are the fixed discs, with fixed centers and radius, which are obviously convex. Therefor this optimization problem can be expressed as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \pi r_i^2 \\ \text{s.t.} \quad & c_i = c_i^{fix}, \;\; r_i = r_i^{fix}, \;\; i \in \{1, \cdots, k\} \\ & r_i \geq 0, \;\; i \in \{1, \cdots, n\} \\ & \|c_i - c_j\|_2 \leq r_i + r_j, \;\; (i,j) \in I \end{aligned}$$

As for the second objective, the function to be minimized is $\sum_{i=1}^{n} 2\pi r_i$ is linear, and hence convex. The constraints is exactly the same as the first problem. Therefor this optimization problem can be expressed as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} 2\pi r_i \\ \text{s.t.} \quad & c_i = c_i^{fix}, \;\; r_i = r_i^{fix}, \;\; i \in \{1, \cdots, k\} \\ & r_i \geq 0, \;\; i \in \{1, \cdots, n\} \\ & \|c_i - c_j\|_2 \leq r_i + r_j, \;\; (i,j) \in I \end{aligned}$$

(2)

The code for this problem is:

```python
import cvxpy as cp
from disks_data import *
import numpy as np

C = cp.Variable((n, 2))
R = cp.Variable(n)

constraints = [R >= 0]
constraints += [C[:k, :] == Cgiven[:k, :]]
constraints += [R[:k] == Rgiven[:k]]

for i in range(len(Gindexes)):
    constraints += [cp.norm(C[Gindexes[i, 0], :]
                  - C[Gindexes[i, 1], :])
                  <=R[Gindexes[i, 0]] + R[Gindexes[i, 1]]]

area_objective = cp.Minimize(np.pi * cp.sum_squares(R))
perimeter_objective = cp.Minimize(2 * np.pi * cp.sum_squares(R))

min_area_problem = cp.Problem(area_objective, constraints)
min_perimeter_problem = cp.Problem(perimeter_objective,
    constraints)

min_area_problem.solve()
print('Optimal area: ', min_area_problem.value)
plot_disks(C.value, R.value, Gindexes, name="areas.png")

min_perimeter_problem.solve()
print('optimal perimeter: ', min_perimeter_problem.value)
plot_disks(C. value, R.value, Gindexes, name="perimeters.png")

```

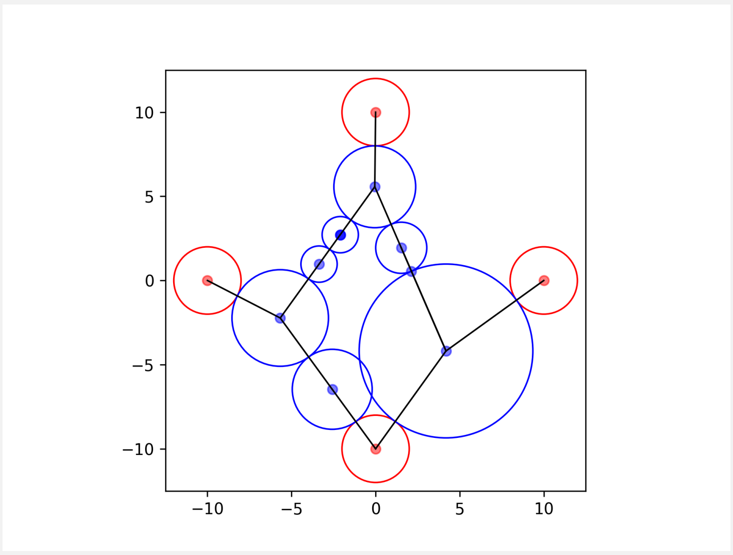
For the optimal value of the area we have:

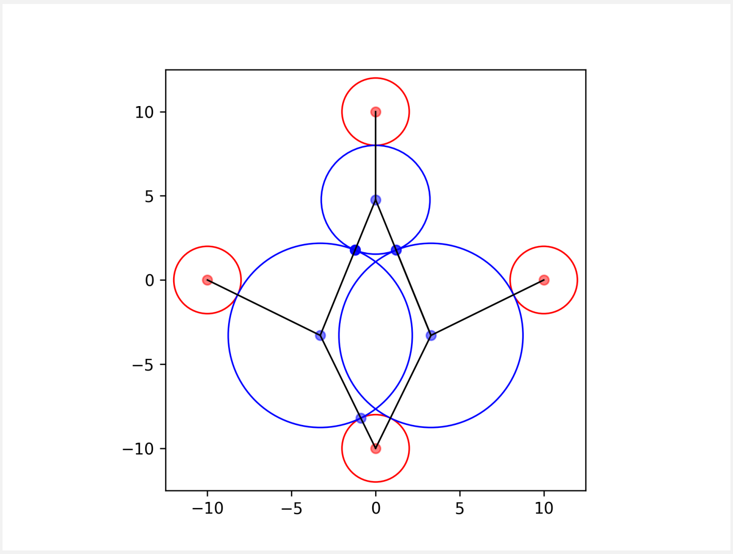
```
Problem status: optimal
Optimal perimeter:  210.76672783990597

```

As for the optimal value of the perimeter we have:

```
Problem status: optimal
Optimal area:  139.34597715732588

```

We observe that minimizing the perimeter, which is also the sum or the ℓ_1 norm of the radius, results in a scenario where a small number of disks have large radius while many disks have a radius of zero. This tendency is a characteristic of ℓ_1 norm minimization. On the other hand, when the goal is to minimize the area, which corresponds to the ℓ_2 norm squared of the radius, the outcome is a distribution with fewer large radius and none at zero.
The plots for the result of the area and perimeter minimization problems are presented below.





Problem 3: *Bandlimited Signal Recovery from Zero-Crossings*

Solution

(1)

The constraints are all in linear from. Our estimation can be represented as $\hat{y} = Ax$. Here, $x = (a, b)$ belongs to $\mathbb{R}^{2B}$ and denotes the coefficient vector for cosine and sine. The matrix A is defined by

$$A = \begin{bmatrix} C & S \end{bmatrix} \in \mathbb{R}^{n \times 2B},$$

where C and S are $n \times B$ matrices with entries defined as

$$C_{tj} = \cos\left(\frac{2\pi(f_{\min} + j - 1)t}{n}\right), \quad S_{tj} = \sin\left(\frac{2\pi(f_{\min} + j - 1)t}{n}\right),$$

for each i and j.
The consistency of $\hat{y}$'s signs with a given s requires the constraints $s_t a_t^T x \geq 0$ for $t = 1, \dots, n$, with $a_1^T, \dots, a_n^T$ representing the rows of A. Furthermore, normalization demands the equality constraint $\|\hat{y}\|_1 = s^T A x = n$, which is linear, due to the fact that we know the signs, and hence convex.
The optimization problem is convex, with a convex objective and linear constraints:

$$\begin{array}{ll} \text{minimize} & \|Ax\|_2 \\ \text{s.t.} & s_t a_t^T x \geq 0, \quad t = 1, \dots, n, \\ & s^T A x = n \end{array}$$

The estimation is obtained as $\hat{y} = Ax^*$, where x^* is the solution to the above problem.

(2)

The python implementation for this problem is:

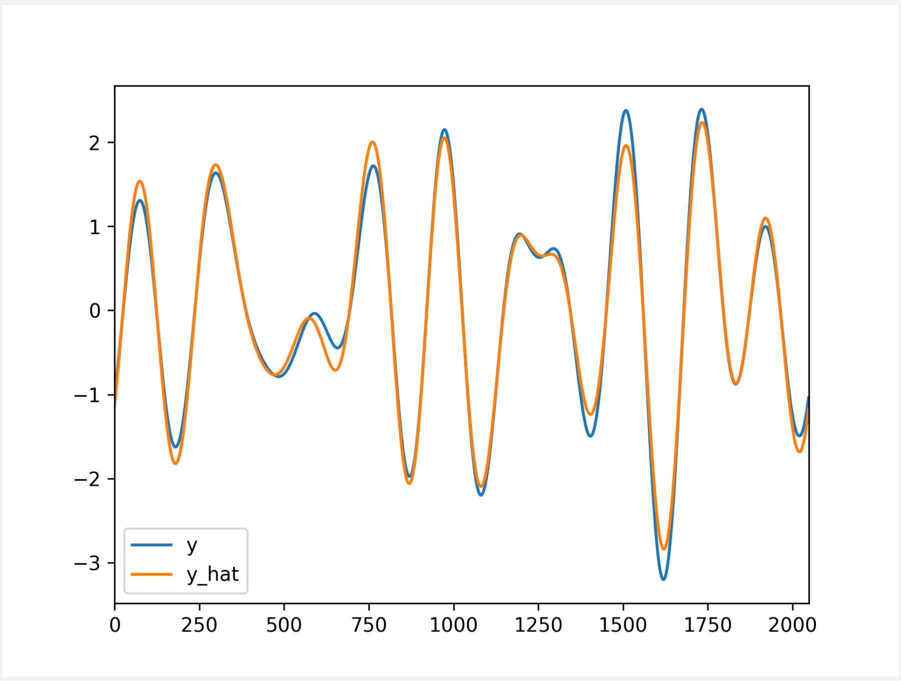
```python
x = cp.Variable(2 * B)

t_vector = np.arange(1, n + 1) / n
frequency_vector = f_min + np.arange(B)
C = np.cos(2 * np.pi * frequency_vector * t_vector[:, np.newaxis
    ])
S = np.sin(2 * np.pi * frequency_vector * t_vector[:, np.newaxis
    ])
A = np.hstack((C, S))

objective = cp.Minimize(cp.norm(A @ x))
constraints = [cp.multiply(s, A @ x) >= 0, s.T @ (A @ x) == n]
prob = cp.Problem(objective, constraints)
prob.solve()

```

```python
y_hat = A @ x.value
recovery_error = np.linalg.norm(y - y_hat) / np.linalg.norm(y)
print(f"Recovery Error: {recovery_error}")

plt.figure()
plt.plot(np.arange(0, n), y, label="y")
plt.plot(np.arange(0, n), y_hat, label="y_hat")
plt.xlim([0, n])
plt.legend(loc='lower left')
plt.savefig("recovery_error.png", dpi=300)
plt.show()
```

The recovery error is

```
Recovery Error: 0.12083136046028914
```

Which is a small value, and considering the amount of information, the quality of the recovered signal is so high.
The plot for the original and the recovered signal is:



Problem 4: *Fitting a Periodic Poisson Distribution to Data*

Solution

(1)

The log-likelihood for N_t independent events is given by $-\lambda_t + N_t \log \lambda_t - \log N_t!$. Summing over the event set yields the aggregate negative log-likelihood:

$$\sum_{t=1}^{24}(\lambda_t - N_t \log \lambda_t + \log N_t!).$$

To compute the ML estimate of λ_t, is equivalent to solving the following convex optimization problem:

$$\begin{aligned} &\text{minimize} \quad && \sum_{t=1}^{24}(\lambda_t - N_t \log \lambda_t + \log N_t!) \\ &\text{s.t. } \lambda \succeq 0 \end{aligned}$$

where $\lambda = (\lambda_1, \ldots, \lambda_{24})$. The constraint $\lambda_t \geq 0$ explicitly includes cases with $N_t = 0$.
Due to the separability of the problem with respect to t, optimization can proceed for each λ_t individually. When $N_t > 0$, we find that the optimal λ_t equals N_t. For $N_t = 0$, the minimization yields $\lambda_t = 0$. Consequently, the maximum likelihood (ML) estimate invariably equates to $\lambda_t = N_t$, intuitively suggesting that the observed frequency serves as the expected event rate.

(2)

We solve the following convex optimization problem:

$$\begin{aligned} &\text{minimize} \quad && \sum_{t=1}^{24}(\lambda_t - N_t \log \lambda_t) + \rho \sum_{t=1}^{23}\left((\lambda_{t+1} - \lambda_t)^2 - (\lambda_1 - \lambda_{24})^2\right) \\ &\text{s.t. } \lambda \succeq 0 \end{aligned}$$

With variable $\lambda \in R^{24}$.

(3)

As $\rho \to \infty$, because of the regularization term, λ_ts will be equal. And the corresponding convex optimization problem will be

$$\begin{aligned} &\text{minimize} \qquad 24\lambda^* - \left(\sum_{t=1}^{24} N_t\right) \log \lambda_t \\ &\text{s.t. } \lambda^* \geq 0 \end{aligned}$$

Where $\lambda = \mathbf{1}^T\lambda^*$. The solution to this problem is the simple Poisson Random variable with $\lambda_t = \lambda^* = \frac{\sum N_t}{24}$.

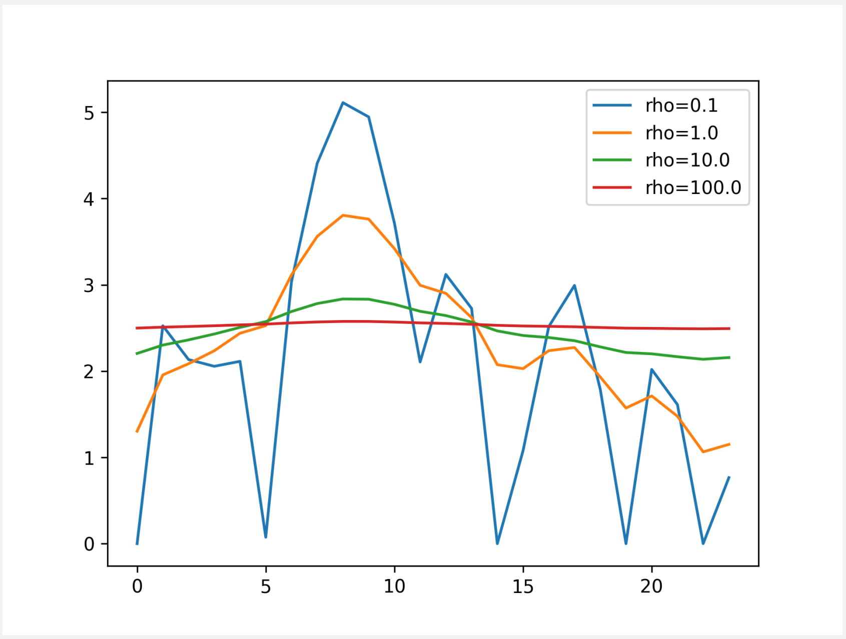
(4)

The python implementation for both this section and the next one is:

```python
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import factorial

T = 24
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])
Ntest = np.array([0, 1, 3, 2, 3, 1, 4, 5, 3, 1, 4, 3,
                  5, 5, 2, 1, 1, 1, 2, 0, 1, 2, 1, 0])

_lambda = cp.Variable(T)
rho = cp.Parameter(nonneg=True)

constraints = [_lambda >= 0]

objective = cp.Minimize(cp.sum(_lambda) - N @ cp.log(_lambda) +
   rho * (
    cp.sum_squares(cp.diff(_lambda)) + (_lambda[0] - _lambda[-1])
   **2))

prob = cp.Problem(objective, constraints)

rho_ls = [0.1, 1, 10, 100]
_lambda_ls = []
for r in rho_ls:
    rho.value = r
    prob.solve()
    _lambda_ls.append(_lambda.value)
    plt.plot(np.arange(T), _lambda.value, label="rho=%.1f" % r)
plt.legend()
plt.savefig("preodic_poisson.png", dpi=300)

```

```python
for lam, r in zip(_lambda_ls, rho_ls):
    logprob = np.sum(
        np.log(np.exp(-lam) * lam ** Ntest / factorial(Ntest)))
    print(f"Rho = {r}, log-likelihood = {logprob}")

```

The plot for the estimated rates is:



As concluded in the last part, the larger the ρ, the smoother the curve of λ.

(5)

Given the log-likelihoods below, the best regularization parameter is $\rho = 1$ since it has the highest log-likelihoods value.

```
Rho = 0.1, log-likelihood = -83.2937163304255
Rho = 1, log-likelihood = -37.748286645339405
Rho = 10, log-likelihood = -41.71490239551201
Rho = 100, log-likelihood = -43.76226937992665

```

End of Practical Homework 1