Convex Optimization in Python

Shahrokh Hamidi
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
Email: shahrokh.hamidi@uwaterloo.ca

I. Abstract

In this tutorial, well-known convex optimization problems that have applications in different areas of science and engineering have been coded in Python3 and the results have been presented. The goal is to provide a Python-based approach for solving convex optimization problems.

II. Introduction

Convex optimization is a sub-field of mathematical optimization which has found numerous applications in different branches of science and engineering and is a powerful tool to address complex problems in a well-organized manner.

On the other hand, Python is a versatile computer programming language capable of handling complex and advanced scientific problems efficiently. Therefore, it has been considered as the programming language to develop and subsequently solve the convex optimization problems presented in this tutorial.

The tutorial contains exercises and examples from the convex optimization book *"S. Boyd and L. Vandenberghe, Convex Optimization. Cambridge: Cambridge University Press, 2004"* that I have coded in Python3.

The code can be downloaded from my github page at *"https://github.com/Shahrokh-Hamidi/Convex-Optimization"* and *"https://github.com/Shahrokh-Hamidi/Filter-Design-Convex-Optimization"*.

III. Convex Optimization Examples

A. Time Series Analysis

In this section, the application of the convex optimization problems in time series analysis is investigates. The data has been generated based on an Auto Regressive (AR) model of order 1

$$y_t = A_0 y_{t-1} + \epsilon_t, \tag{1}$$

where $\epsilon_t \in \mathcal{N}(0, \sigma^2)$ is a normally distributed parameter with mean 0 and standard deviation equal to σ. It should be noted that, when $A_0 = 1$, the model describes the random walk process. In order to obtain

an AR model, the parameter A_0 should be chosen as $|A_0| < 1$. The analysis is based on the following optimization problem

$$\min_{\mathbf{x}} \quad ||\mathbf{y} - \mathbf{x}||_2 + \lambda ||\mathbf{D}\mathbf{x}||_2, \tag{2}$$

where $\mathbf{y}$ represents the noisy time series and the matrix $\mathbf{D}$ is defined as

$$\begin{bmatrix} 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ & & & & \dots & \\ 0 & 0 & \dots & 1 & -2 & 1 \end{bmatrix}$$

. In fact, the $\mathbf{D}\mathbf{x}$ term is a regularizer responsible for removing the high frequency elements from the signal which results in removing the noisy components.

The code is given as follows.

```
#time series analysis


import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import matplotlib



n = 2000
y = np.zeros(n)
A0 = 1
for i in range(1,n):
    y[i] = A0*y[i-1] + np.random.normal(0,0.05)

n = len(y)


A = np.hstack((np.array([1, -2, 1]), np.zeros(n-3)))

D = []
for i in range(len(A)):
    D.append(np.roll(A,i))

D = np.array(D)



lambda_ = 10
x = cp.Variable(n)

cost = cp.norm(y - x, 2) + lambda_* cp.sum(cp.abs(D@x))       #cp.tv(x)
constr = []
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = True)



x = x.value

plt.rcParams['savefig.dpi'] = 300

plt.plot(y, 'k-', lw = 0.3)
```

```
plt.plot(x, 'r')
matplotlib.rc('font', size=14)
plt.title('Time Series Analysis')

plt.xlabel('t', fontsize = 16)
plt.ylabel('y[t]', fontsize = 16)
matplotlib.style.use('ggplot')
plt.show()
```

Fig. 1 illustrates the result of the code for the *time series analysis* code.

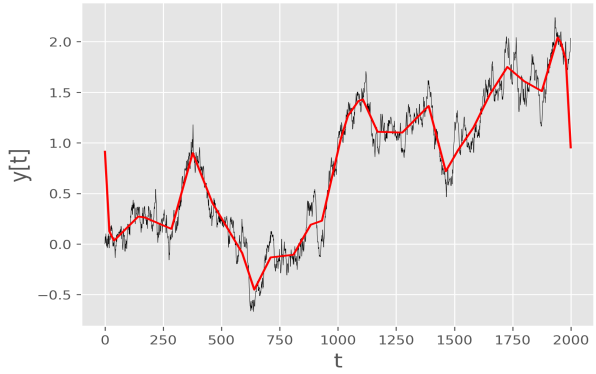

Fig. 1. The result for the *time series analysis*.

B. *Chebyshev Center*

The optimization problem for the Chebyshev center is describe as

$$\begin{aligned} \max_{r} \quad & r \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + r \leq \mathbf{b}, \\ & r \geq 0. \end{aligned} \tag{3}$$

The optimization problem presented in (3) is a Linear Programming (LP) problem.

```
#Chebyshev Center
import numpy as np
import matplotlib.pyplot as plt
import scipy
import cvxpy as cp


n = 2
px = np.array([0, .5, 2, 3, 1])
py = np.array([0, 1, 1.5, .5, -.5])

px = np.hstack((px, px[0]))
py = np.hstack((py, py[0]))

px_diff = px[1:] - px[:-1]
py_diff = py[1:] - py[:-1]

px_avg = 0.5*(px[1:] + px[:-1])
py_avg = 0.5*(py[1:] + py[:-1])
```

```
A = []
for i in range(0, len(px)-1):
    p = np.array([px_diff[i], py_diff[i]])
    p = p/np.linalg.norm(p)
    A.append([-p[1], p[0]])

A = np.array(A)

b = []
for i in range(0, len(px)-1):
    p = np.array([px_avg[i], py_avg[i]])
    b.append(A[i,:].dot(p))

#plt.plot(px, py)

m = A.shape[-1]
x = cp.Variable(m)
r = cp.Variable(1)

constr = []
constr += [A@x + r <= b]

prob = cp.Problem(cp.Maximize(r), constr)
prob.solve(verbose = False)



x = x.value
r = r.value


#---- Display

N = 100

theta = np.linspace(0,2*np.pi, N).reshape(1,-1)

x1 = r*np.cos(theta).squeeze() + x[0]
x2 = r*np.sin(theta).squeeze() + x[1]


plt.plot(px, py)
plt.plot(x1, x2, 'r--')
plt.plot(x[0], x[1], 'ko')
plt.title('Chebyshev Center')
plt.axis('off')
plt.show()
```

Fig. 2 illustrates the result of the code for the *Chebyshev Center* problem.

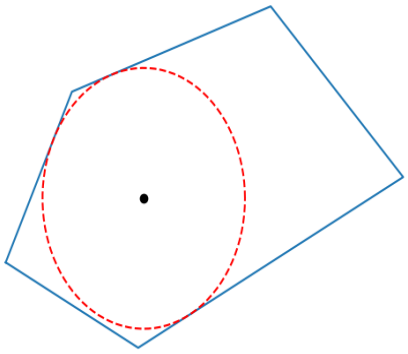

Fig. 2. The result for the *Chebyshev Center*.

C. *Chebyshev Center of a 2D Polyhedron*

In this section the optimization problem for the Chebyshev center, given in (3), has been rewritten in a slightly different form as

$$\begin{aligned} \max_{r} \quad & r \\ \text{s.t.} \quad & \mathbf{a_i^T x_c} + r||\mathbf{a_i}||_2 \leq \mathbf{b} \ \text{ for } \ \mathrm{i} \in \{1, 2, \cdots, \mathrm{N}\}, \\ & r \geq 0, \end{aligned} \tag{4}$$

in which $\mathbf{x_c}$ is the center point. The optimization problem expressed in (4) is a LP problem similar to the optimization problem presented in (3).

```
#Chebyshev center of a 2D polyhedron
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp

#%matplotlib qt



a1 = np.array([ 2, 1])
a2 = np.array([ 2, -1])
a3 = np.array([-1,  2])
a4 = np.array([-1, -2])
b = np.ones(4)


m = 2
c = cp.Variable(m)
r = cp.Variable(1)

constr = []

constr += [a1@c + np.linalg.norm(a1,2)*r <= b[0]]
constr += [a2@c + np.linalg.norm(a2,2)*r <= b[1]]
constr += [a3@c + np.linalg.norm(a3,2)*r <= b[2]]
constr += [a4@c + np.linalg.norm(a4,2)*r <= b[3]]
```

```
prob = cp.Problem(cp.Maximize(r), constr)
prob.solve(verbose = False)



c = c.value
r = r.value


#---- Display

N = 100

theta = np.linspace(0,2*np.pi, N).reshape(1,-1)
x = np.linspace(-2,2,100)




x1 = r*np.cos(theta).squeeze() + c[0]
x2 = r*np.sin(theta).squeeze() + c[1]

plt.plot(x, b[0]/a1[1] - a1[0]*x/a1[1], 'k')
plt.plot(x, b[1]/a2[1] - a2[0]*x/a2[1], 'k')
plt.plot(x, b[2]/a3[1] - a3[0]*x/a3[1], 'k')
plt.plot(x, b[3]/a4[1] - a4[0]*x/a4[1], 'k')
plt.plot(x1, x2, 'r--')
plt.plot(c[0], c[1], 'bo')
plt.title('Chebyshev center of a 2D polyhedron')
plt.ylim(-1,1)

plt.show()
```

Fig. 3 illustrates the result of the code for the *Chebyshev center of a 2D polyhedron* problem.

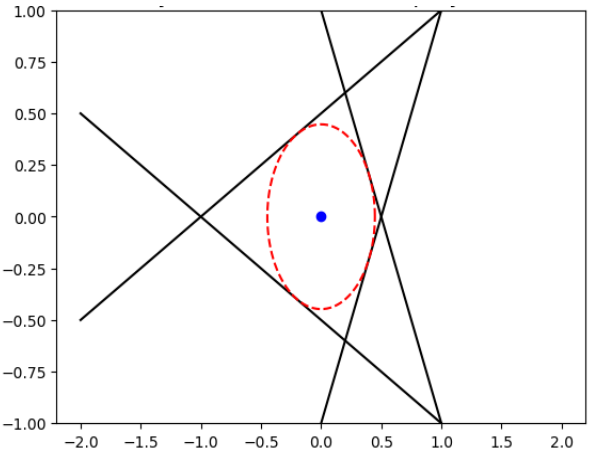

Fig. 3. The result for the *Chebyshev center of a 2D polyhedron*.

D. *Fastest Mixing Markov Chain on a Graph*

The transition probability for Markov chain is described as

$$\begin{aligned} & p_{ij} = P[x(t+1) = i | x(t) = j], \\ & \sum_j p_{ij} = 1 \ \text{ for } \ \mathrm{i} \in \{1, 2, \cdots, \mathrm{N}\}, \\ & p_{ij} \geq 0, \\ & p_{ij} = p_{ji}, \end{aligned} \tag{5}$$

where $P[x(t+1) = i | x(t) = j]$ describes the transition probability from state j at time t to state i at time $t+1$.

```
#Fastest mixing Markov chain on a graph
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp

#%matplotlib qt


n = 5
E = np.array([[0, 1, 0, 1, 1],
              [1, 0, 1, 0, 1],
              [0, 1, 0, 1, 1],
              [1, 0, 1, 0, 1],
              [1, 1, 1, 1, 0]])


P = cp.Variable((n,n), symmetric = True)



constr = []
constr += [P@np.ones(n) == np.ones(n)]
constr += [P >= 0]
constr += [P[E==0] == 0]

cost = cp.norm(P - np.ones((n,n))/n)
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = True)


print(f'the optimal value is {np.round(P.value,2)}')
```

The optimal result is given as

```
P =  [[0.    0.38  0.    0.38  0.25]
 [ 0.38  0.    0.38  0.    0.25]
 [0.    0.38  0.    0.38  0.25]
 [ 0.38  0.    0.38  0.    0.25]
 [ 0.25  0.25  0.25  0.25  0.  ]]
```

E. Fitting a Convex Function to Given Data

In this section the problem of fitting a convex function to the set of given data is considered. The problem can be addressed using least-squares method. The optimization problem can be written as the following Quadratic Programming (QP) problem

$$\begin{aligned} \min_{\hat{y},g} \quad & \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \\ \text{s.t.} \quad & \hat{y}_j \geq \hat{y}_i + g_i^T (u_j - u_i) \ \text{ for } \ \mathrm{i,j} \in \{1, 2, \cdots, \mathrm{m}\}. \end{aligned} \tag{6}$$

The code has been given as follows.

```
#Fitting a convex function to given data
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp

#%matplotlib qt



lambda_ = 1

yns = np.array([[ 5.2057354 ],
       [ 5.16852954],
       [ 4.46931747],
       [ 3.16764149],
       [ 3.21867268],
       [ 2.75587742],
       [ 1.63606279],
       [ 0.72527756],
       [ 0.24583927],
       [-0.58044829],
       [-0.87676552],
       [-0.82548372],
       [-0.79731423],
       [-0.05948396],
       [-0.04975525],
       [ 0.70500264],
       [ 0.82600211],
       [ 0.1403059 ],
       [ 0.51054544],
       [ 0.38582234],
       [ 0.83860086],
       [ 0.41632982],
       [ 0.81154682],
       [ 0.23060127],
       [ 0.84177419],
       [ 0.34454159],
       [ 0.37408514],
       [ 0.86597229],
       [ 0.2120701 ],
       [ 0.71788999],
       [ 0.80995603],
       [ 1.06910934],
       [ 0.64850169],
       [ 1.09248439],
```

```
       [ 0.76143045],
       [ 1.2122857 ],
       [ 1.17728916],
       [ 0.84659501],
       [ 0.95866895],
       [ 1.82113178],
       [ 1.80159358],
       [ 1.63543887],
       [ 1.77429526],
       [ 2.52647668],
       [ 2.65227764],
       [ 3.75011515],
       [ 4.05642222],
       [ 4.62476811],
       [ 4.91230273],
       [ 5.80689459],
       [ 7.02609346]])

u = np.array([[0.  ],
       [0.04],
       [0.08],
       [0.12],
       [0.16],
       [0.2 ],
       [0.24],
       [0.28],
       [0.32],
       [0.36],
       [0.4 ],
       [0.44],
       [0.48],
       [0.52],
       [0.56],
       [0.6 ],
       [0.64],
       [0.68],
       [0.72],
       [0.76],
       [0.8 ],
       [0.84],
       [0.88],
       [0.92],
       [0.96],
       [1.  ],
       [1.04],
       [1.08],
       [1.12],
       [1.16],
       [1.2 ],
       [1.24],
       [1.28],
       [1.32],
       [1.36],
       [1.4 ],
       [1.44],
       [1.48],
       [1.52],
       [1.56],
       [1.6 ],
       [1.64],
       [1.68],
       [1.72],
       [1.76],
```

```
        [1.8 ],
        [1.84],
        [1.88],
        [1.92],
        [1.96],
        [2.  ]])


n = len(yns)
y = cp.Variable((n,1))
g = cp.Variable((n,1))


constr = []

constr += [y@np.ones((1,n)) >=  np.ones((n,1))@y.T + cp.multiply((np.ones((n,1))@g.T), (u@np.
                                                    ones((1,n)) - np.ones((n,1))@u.T))]

cost = cp.norm(y - yns)
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = False)



y = y.value

plt.figure()
plt.plot(u, yns, 'o', mfc= 'none')
plt.plot(u, y, 'b')
plt.xlabel('u', fontsize = 16)
plt.ylabel('y', fontsize = 16)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.title('Fitting a convex function to given data')
plt.xlim(-0.5,2.5)
plt.ylim(-1,8)
plt.grid()
plt.show()
```

Fig. 4 shows the result of the code for the *Fitting a convex function to given data* problem.

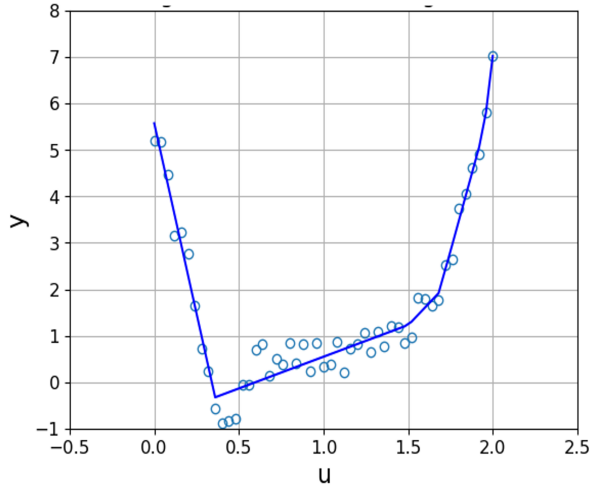

Fig. 4. The result for the *Fitting a convex function to given data* problem.

F. Matrix Fractional Minimization Using SOCP

The problem of matrix fractional minimization is presented in this section and is cast as Second Order Cone Programming (SOCP) problem

```
#Matrix fractional minimization using SOCP
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp

#%matplotlib qt




A = np.array([[-0.432564811528221,  1.06676821135919, 0.815622288876143,  -2.17067449430526,
                                                0.428183273045163,  0.623233851138494,  0.
                                               781181617878392,  1.47247993441992],
[-1.66558437823810, 0.0592814605236054, 0.711908323500893,  -0.0591878245211912,  0.
                                               895638471211752,  0.799048618147778,  0.
                                               568960645723274,  0.0557438318378432],
[0.125332306474831, -0.0956484054836690,  1.29024975493248, -1.01063370647425,  0.
                                               730957338429453,  0.940889940727780,  -0.
                                               821714291696256, -1.21731745370455],
[0.287676420358549, -0.832349463650023, 0.668600505682040,  0.614463048895481,  0.
                                               577857346330798,  -0.992091735543795, -0.
                                               265606851332549, -0.0412271336864321],
[-1.14647135068146, 0.294410816392640,  1.19083807424337, 0.507740785341986,  0.
                                               0403140316184403, 0.212035152165055,  -1.
                                               18777701646980,  -1.12834386432023],
[1.19091546564300,  -1.33618185793780,  -1.20245711477394,  1.69242987019052, 0.
                                               677089187597305,  0.237882072875579,  -2.
                                               20232071732344,  -1.34927754310249],
[1.18916420165210,  0.714324551818952,  -0.0197895577687705,  0.591282586924176,  0.
                                               568900205200723,  -1.00776339167827,  0.
                                               986337391002023,  -0.261101623061621],
```

```
[-0.0376332765933176, 1.62356206444627, -0.156717298831981, -0.643595202682526, -0.
                                                 255645415631965, -0.742044752133604, -0.
                                                 518635066344746, 0.953465445504819],
[0.327292361408654, -0.691775701702287, -1.60408556200116,  0.380337251713910,  -0.
                                                 377468955522361, 1.08229495315533, 0.
                                                 327367564080834,  0.128644430046645],
[0.174639142820925, 0.857996672828263,  0.257304234677490,  -1.00911552434079,  -0.
                                                 295887110003557, -0.131499702945274, 0.
                                                 234057012847185,  0.656467513885396],
[-0.186708577681439,  1.25400142160253, -1.05647292808148,  -0.0195106695302893,  -1.
                                                 47513450585526,  0.389880489687039,  0.
                                                 0214661388790945, -1.16781936472664],
[0.725790548293303, -1.59372957644748,  1.41514148587234, -0.0482207891453123,  -0.
                                                 234004047656033, 0.0879871065797930, -1.
                                                 00394446674772,  -0.460605179506150],
[-0.588316543014189,  -1.44096443190102,  -0.805090404196880, 4.31918416255450e-05, 0.
                                                 118444837054121,  -0.635465225479316, -0.
                                                 947146064738541, -0.262439952838333],
[2.18318581819710,  0.571147623658178,  0.528743010962225,  -0.317859451247688, 0.
                                                 314809043395056,  -0.559573302196241, -0.
                                                 374429195029166, -1.21315206849391],
[-0.136395883086596,  -0.399885577715363, 0.219320672667622,  1.09500373878749, 1.
                                                 44350824434982, 0.443653489503667,  -1.
                                                 18588621380853,  -1.31943699810954],
[0.113931313520810, 0.689997375464345,  -0.921901624355539, -1.87399025764096,  -0.
                                                 350974738327742, -0.949903798547645, -1.
                                                 05590292352369,  0.931217514995436]])


b  =np.array([0.0112448963841337,-0.645145815691170,0.805728793112376,0.231626010780437,-0.
                                                 989759671682004,1.33958570061039,0.
                                                 289502034538413,
              1.47891705768128,1.13802801285837,-0.684138585136340,-1.29193604496594,-0.
                                                                 0729262762636467,-0.
                                                                 330598879892764,-0.
                                                                 843627639154800,
              0.497769664182782,1.48849047090348])


B  = np.array([[-0.546475894767623, 0.469383119866330,  0.288807018730340,  1.95738475514751,
                                                  -0.321004692181292, 1.22744798900972, 0.
                                                 485497707312810,  -1.27050020370838],
               [-0.846758163883060,  -0.903566942617776, -0.429303004551915, 0.
                                                                 504542353592166,  1.
                                                                 23655565160192, -0.
                                                                 696204800032889, -0.
                                                                 00500507375553139, -1.
                                                                 663606645282977],
               [-0.246336528084900,  0.0358796387294769, 0.0558011901764726, 1.
                                                                 86452902048530, -0.
                                                                 631279656725146, 0.
                                                                 00752448652301445,  -0.
                                                                 276217859354759, -0.
                                                                 703554261536755],
               [0.663024145855908, -0.627531219966832, -0.367873566740638, -0.
                                                                 339811777414964, -2.
                                                                 32521112888377,  -0.
                                                                 782893044378287, 1.
                                                                 27645247367439, 0.
                                                                 280880488523302],
               [-0.854197374468980,  0.535397954249106,  -0.464973367171118, -1.
                                                                 13977940231323,  -1.
                                                                 23163653332502,  0.
```

```
                                                    586938559214431,  1.
                                                    86340061318454, -0.
                                                    541209329916194],
 [-1.20131481533904, 0.552883517423822,  0.370960583848952,  -0.
                                                    211123483380258, 1.
                                                    05564838790246, -0.
                                                    251207374568882, -0.
                                                    522559301636399, -1.
                                                    33353072973639],
 [-0.119869428057387,  -0.203690479567358, 0.728282931551495,  1.
                                                    19024493625120, -0.
                                                    113223989369025, 0.
                                                    480135822842601,  0.
                                                    103424446937315,  1.
                                                    07268626789014],
 [-0.0652940148415865, -2.05432468055661,  2.11216016977150, -1.11620875778561
                                                    ,  0.379223622685033,  0.
                                                    668155034433641,  -0.
                                                    807649130897181, -0.
                                                    712085452494356],
 [0.485295555916544, 0.132560731417280,  -1.35729774309675,  0.635274134747122
                                                    ,  0.944199726747308,  -0.
                                                    0783211962734119,  0.
                                                    680438583748946,  -0.
                                                    0112855612306856],
 [-0.595490902619476,  1.59294070376602, -1.02261014433421,  -0.
                                                    601412126269725, -2.
                                                    12042668822421,  0.
                                                    889172618412599,  -2.
                                                    36458984794158,  -0.
                                                    0008170291956958 36],
 [-0.149667743824475,  1.01841178862471, 1.03783419871876, 0.551184711824902,
                                                      -0.644678915541937, 2.
                                                    30928748595239, 0.
                                                    990114872049490,  -0.
                                                    249436284695434],
 [-0.434751931152533,  -1.58040249930316,  -0.389799548476831, -1.
                                                    09984045471081,  -0.
                                                    704301728433609, 0.
                                                    524638679771098,  0.
                                                    218899120881177,  0.
                                                    396575318711652],
 [-0.0793302230234206, -0.0786619193594521,  -1.38126562401984,  0.
                                                    0859905932937184, -1.
                                                    01813721639907,  -0.
                                                    0117873239513068,  0.
                                                    261662460161402,  -0.
                                                    2640133549222 43],
 [1.53515226612215,  -0.681656860002363, 0.315542632772365,  -2.00456332159079
                                                    ,  -0.182081868411385, 0.
                                                    913140817761371,  1.
                                                    21344449497535, -1.
                                                    66401087693059],
 [-0.606482859277266,  -1.02455305742903,  1.55324256851535, -0.
                                                    493087917659697, 1.
                                                    52101323900559, 0.
                                                    0559406788884020, -0.
                                                    274666986456781, -1.
                                                    02897509954380],
 [-1.34736267385024, -1.23435347798426,  0.707893884632476,  0.462048011799193
                                                    ,  -0.0384387638867116,  -1.
                                                    10706989482601,  -0.
                                                    133134450813529, 0.
```

```
                                                    2430947002245651]])


def Form_I():

    x = cp.Variable(n)

    constr = []

    constr += [x >= 0]

    cost = cp.matrix_frac(A@x + b, np.eye(m) + (B@cp.diag(x))@B.T)
    prob = cp.Problem(cp.Minimize(cost), constr)
    prob.solve(verbose = False)


    x = x.value
    print(f'Form_I Objective : {cost.value}')



def Form_II():

    x = cp.Variable(n)
    Y = cp.Variable((n,n))

    constr = []

    constr += [x >= 0]
    constr += [Y == cp.diag(x)]

    cost = cp.matrix_frac(A@x + b, np.eye(m) + (B@Y)@B.T)
    prob = cp.Problem(cp.Minimize(cost), constr)
    prob.solve(verbose = False)


    x = x.value
    print(f'Form_II Objective : {cost.value}')


def Form_III():

    x = cp.Variable(n)
    v = cp.Variable(m)
    w = cp.Variable(n)

    constr = []

    constr += [x >= 0]
    constr += [v + B@w == A@x + b]

    cost = cp.sum_squares(v) + cp.matrix_frac(w, cp.diag(x))
    prob = cp.Problem(cp.Minimize(cost), constr)
    prob.solve(verbose = False)


    x = x.value
    print(f'Form_II Objective : {cost.value}')
```

```
def Form_IV():

    x = cp.Variable(n)
    v = cp.Variable(m)
    w = cp.Variable(n)
    Y = cp.Variable((n,n))


    constr = []

    constr += [x >= 0]
    constr += [v + B@w == A@x + b]
    constr += [Y == cp.diag(x)]
    cost = cp.sum_squares(v) + cp.matrix_frac(w, Y)
    prob = cp.Problem(cp.Minimize(cost), constr)
    prob.solve(verbose = False)


    x = x.value
    print(f'Form_II Objective : {cost.value}')


m, n = A.shape

Form_I()
Form_II()
Form_III()
Form_IV()
```

The result is given as

```
Form_I Objective : 5.1824532944840636
Form_II Objective : 5.182475550172795
Form_II Objective : 5.182551305152927
Form_II Objective : 5.185477845279463
```

G. Maximum Determinant PSD Matrix Completion

This section has been dedicated to the problem of completing the unknown elements of a given matrix in way that the matrix is Positive Semi Definite (PSD). The optimization problem can be formulated as

$$\begin{aligned} \min_{\mathbf{A}} \quad & -\det\log(\mathbf{A}) \\ \text{s.t.} \quad & \mathbf{A}[i,j] = \mathbf{M}[i,j] \ \text{ for } \ \text{i,j} \in \Omega, \\ & \mathbf{A} \succeq 0, \end{aligned} \tag{7}$$

in which $\mathbf{M}$ is the given matrix and Ω contains the list of the indexes of the matrix that their values are known.

```
#Maximum determinant PSD matrix completion
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp
```

```
#%matplotlib qt


n = 4


A = cp.Variable((n,n), PSD = True)


constr = []

constr += [A[0,0] == 3]
constr += [A[1,1] == 2]
constr += [A[2,2] == 1]
constr += [A[3,3] == 5]
constr += [A[0,1] == 0.5]
constr += [A[0,3] == 0.25]
constr += [A[1,2] == 0.75]

cost = -cp.log_det(A)
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = True)


print(f'A = {A.value}')
```

The result for the problem is given as

```
A = [[3.          0.5         0.18749999    0.25]
 [0.5         2.          0.75        0.04166667]
 [0.18749999        0.75        1.          0.015625]
 [0.25        0.04166667    0.015625    5.]]
```

H. Polynomial Fitting

In this section, the polynomial fitting problem based on 3 different norms, namely, l_1, l_2, and l_∞ norm, is presented. The optimization problem is expressed as

$$\min_{\mathbf{x}} \quad ||\mathbf{Ax} - \mathbf{v}||_\infty. \tag{8}$$

The code for the polynomial fitting problem is given as follows.

```
#Polynomial Fitting
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp

#%matplotlib qt

n=6
m=40


u = np.linspace(-1,1,m)
v = 1/(5+40*u**2) + 0.1*u**3 + np.random.normal(0,0.01,m)



A = np.vander(u)[:,-n:]
```

```
plt.plot(u,v, 'o', mfc = 'none')



def Poly_fitting(method):

    x = cp.Variable(n)

    cost = cp.norm(A@x - v, method)
    prob = cp.Problem(cp.Minimize(cost))
    prob.solve()

    return x.value

method = 2
x_l2 = Poly_fitting(method)

method = 1
x_l1 = Poly_fitting(method)

method = 'inf'
x_inf = Poly_fitting(method)

plt.plot(u, v, 'bo', mfc = 'none', label = 'Data')
plt.plot(u, A@x_l1, 'k', label = 'l1 norm')
plt.plot(u, A@x_l2, 'r', label = 'l2 norm')
plt.plot(u, A@x_inf, 'g', label = '$l_\infty$ norm')
plt.xlabel('u', fontsize = 16)
plt.ylabel('v', fontsize = 16)
plt.rc('legend',fontsize=20) # using a size in points
#plt.rc('legend',fontsize='medium') # using a named size
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=14)
ax.yaxis.set_tick_params(labelsize=14)
plt.grid()
plt.legend()
#
plt.title('Polynomial Fitting', fontsize = 16)
plt.tight_layout()
plt.show()
```

The result for the for the *Polynomial Fitting* problem has been shown in Fig. 2.

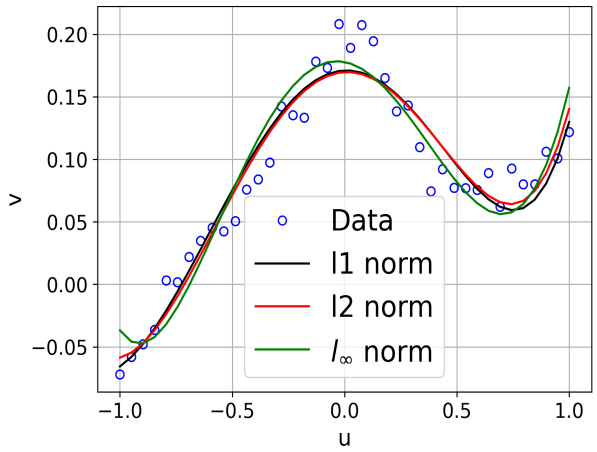

Fig. 5. The result for the *Polynomial Fitting* problem.

I. Quadratic Smoothing

The quadratic smoothing is a method that can be used for signal de-noising. The optimization problem for quadratic smoothing is given as

$$\min_{\mathbf{x}} \quad ||\mathbf{y}-\mathbf{x}||_2 + \lambda||\mathbf{D}\mathbf{x}||_2, \tag{9}$$

where $\mathbf{y}$ represents the noisy signal and the matrix $\mathbf{D}$ is defined as

$$\begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ & & & \dots & \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix}$$

. In fact, the $\mathbf{D}\mathbf{x}$ term is a regularizer responsible for removing the high frequency elements from the signal which results in removing the noisy components.

The code for the quadratic smoothing has been given as follows.

```
#Quadratic smoothing
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp

#%matplotlib qt


n = 4000
t = np.arange(0, n)

signal = 0.5*np.sin((2*np.pi/n)*t)*np.sin(0.01*t)

noisy_signal = signal + np.random.normal(0,0.1,n)


lambda_ = 500
D = np.zeros((n,n))
```

```
for i in range(n-1):
    D[i, i:i+2] = [-1,1]



x = cp.Variable(n)

cost = cp.norm(noisy_signal - x, 2) + lambda_*cp.norm(D@x,2)
prob = cp.Problem(cp.Minimize(cost))
prob.solve()

denoised_signal = x.value


plt.subplot(311)
plt.plot(t, signal, 'k', lw = 2,  label = 'signal')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.legend()
plt.title('Quadratic Smoothing', fontsize = 16)



plt.subplot(312)
plt.plot(t, noisy_signal, label = 'noisy_signal')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.legend()



plt.subplot(313)
plt.plot(t, denoised_signal, 'r', label = 'denoised_signal')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.legend()


plt.show()
```

Fig. 6 illustrates the result of the code for the *quadratic smoothing* problem.

J. Robust Regression Using the l_1, l_2, and Huber Penalty

In this section, the regression problem based on 3 different regularizers is presented. The code for the problem is given as follows.

```
#Robust regression using the l1 l2 huber penalty
import numpy as np
```

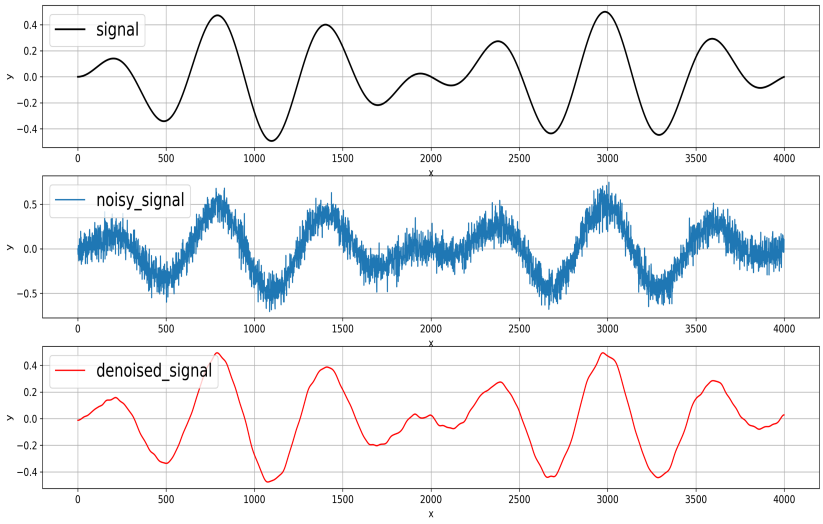

Fig. 6. The result for the *quadratic smoothing* problem.

```
import matplotlib.pyplot as plt
import cvxpy as cp

#%matplotlib qt

def DATA_gen():

    x = np.linspace(-10, 10, 30)

    y = x + np.random.normal(0, 2, len(x))
    data = np.hstack((x.reshape(-1,1), y.reshape(-1,1)))
    data = np.vstack((data, np.array([-10, 10])))
    data = np.vstack((data, np.array([10, -10])))

    return data



def cvx_opt(penalty):

    m = cp.Variable(1)
    b = cp.Variable(1)

    if penalty == 'l2':
        cost = cp.norm(data[:,1] - (m*data[:,0] + b), 2)

    if penalty == 'l1':
        cost = cp.norm(data[:,1] - (m*data[:,0] + b), 1)

    if penalty == 'huber':
        cost = cp.sum(cp.huber(data[:,1] - (m*data[:,0] + b), M = 4))

    prob = cp.Problem(cp.Minimize(cost))
    prob.solve()

    b = b.value
    m = m.value
```

```
    return m, b



data = DATA_gen()

penalty = 'l2'
m2, b2 = cvx_opt(penalty)

penalty = 'l1'
m1, b1 = cvx_opt(penalty)

penalty = 'huber'
mh, bh = cvx_opt(penalty)


x = np.linspace(-10, 10, 100)
plt.plot(x, m2*x+b2, 'k--', label = 'l2 norm')
plt.plot(x, m1*x+b1, label = 'l1 norm')
plt.plot(x, mh*x+bh, label = 'huber')
plt.plot(data[:,0], data[:,1], 'ko', mfc = 'none')
plt.legend()

plt.title('Robust regression using the l1, l2, and huber penalty')

plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.show()
```

Fig. 7 illustrates the result of the code for the Robust regression using the l1 l2 huber penalty problem. As it is clear form Fig. 7, l_2-based regularizer is susceptible to outliers. In contrast, the l_1-based regularizer is the most robust one when it comes to outliers.

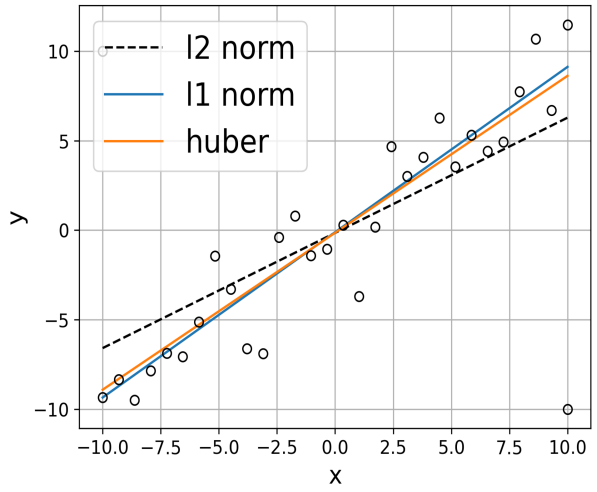

Fig. 7. The result for the *robust regression using the* l_1, l_2, *and huber penalty* functions.

K. Simple QP with Inequality constraints

In this section a simple QP problem is presented and the code is given as follows.

```
#simple QP with inequality constraints
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp



P = np.array([[13, 12, -2], [12, 17, 6], [-2, 6, 12]])
q = np.array([[-22, -14.5, 13]])
r = 1
m = 3
#x_star = [1;1/2;-1];



x = cp.Variable(m)


constr = []

constr += [x <= 1]
constr += [x >= -1]

cost = 0.5*cp.quad_form(x, P) + q@x + r
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = True)


x = x.value

print(f'x_optimal : {x}')
```

The result for this problem is given as

```
x_optimal : [ 1.   0.5   -1.]
```

L. Total Variation Reconstruction Example

$$\min_{\mathbf{x}} \quad ||\mathbf{y} - \mathbf{x}||_2 + \lambda||\mathbf{D}\mathbf{x}||_2, \tag{10}$$

where $\mathbf{y}$ represents the noisy signal and the matrix $\mathbf{D}$ is defined as

$$\begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ & & & \dots & \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix}$$

. In fact, the $\mathbf{D}\mathbf{x}$ term is a regularizer responsible for removing the high frequency elements from the signal which results in removing the noisy components.

The code for the quadratic smoothing has been given as follows.

```
#Total variation reconstruction example
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp

#%matplotlib qt


n = 4000
t = np.arange(0, n)

signal = 0.5*np.sin(0.05*(2*np.pi/n)*t + np.pi/4)


for i in range(len(signal)):
    if n//4 <= i <= n//2:
        signal[i] = -signal[i]

    if  3*n//4 <= i:
        signal[i] = - signal[i]

noisy_signal = signal + np.random.normal(0,0.1,n)


lambda_ = 1
D = np.zeros((n,n))

for i in range(n-1):
    D[i, i:i+2] = [-1,1]



x = cp.Variable(n)

cost = cp.norm(noisy_signal - x, 2) + lambda_*cp.norm(D@x,1)
prob = cp.Problem(cp.Minimize(cost))
prob.solve()

denoised_signal = x.value
```

```
plt.subplot(311)
plt.plot(t, signal, 'k', lw = 2,  label = 'signal')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
plt.ylim(-0.6, 0.6)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.legend()
plt.title('Total variation reconstruction example', fontsize = 16)



plt.subplot(312)
plt.plot(t, noisy_signal, label = 'noisy_signal')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
plt.ylim(-0.6, 0.6)
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.legend()



plt.subplot(313)
plt.plot(t, denoised_signal, 'r', label = 'denoised_signal')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
ax = plt.gca()
plt.ylim(-0.6, 0.6)
ax.xaxis.set_tick_params(labelsize=12)
ax.yaxis.set_tick_params(labelsize=12)
plt.grid()
plt.legend()


plt.show()
```

Fig. 8 illustrates the result of the code for the *total variation reconstruction example* problem.

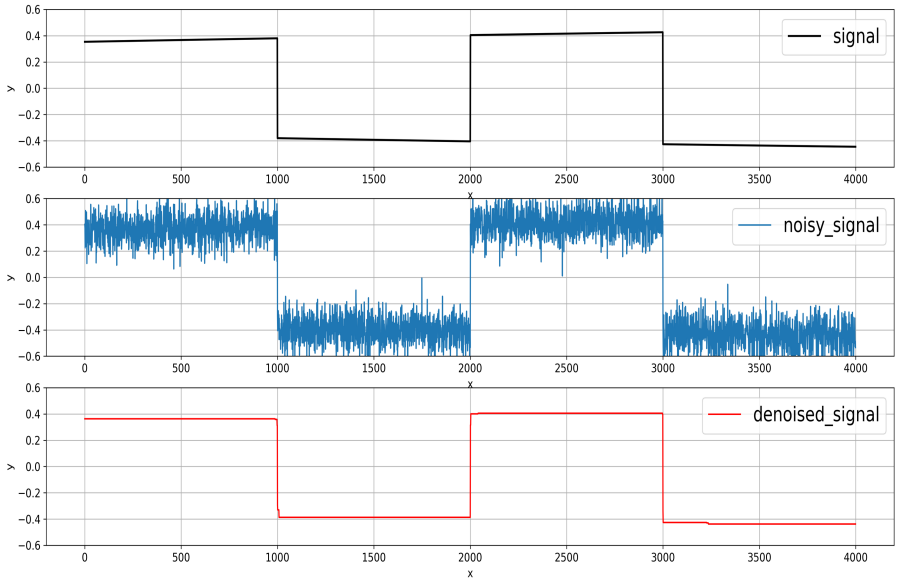

Fig. 8. The result for the *total variation reconstruction* example.

M. Maximum Entropy Distribution

This section has been dedicated to the maximum entropy distribution problem. The optimization problem is expressed as

$$\begin{aligned} \max_{\mathbf{p}} \quad & -\sum_{i=1}^{N} p_i \log p_i \\ \text{s.t.} \quad & \mathbf{Ap} \preceq \mathbf{b}, \\ & p_i \geq 0 \ \text{ for } \ \mathrm{i} \in \{0, 1, \cdots, \mathrm{N}\}, \\ & \sum_{i=1}^{N} p_i = 1. \end{aligned} \tag{11}$$

The code for this problem is given as follows.

```
#Maximum Entropy Distribution
import cvxpy as cp
import numpy as np
import scipy as scipy
import matplotlib.pyplot as plt


#%matplotlib qt


n = 100
u = np.linspace(-1,1,n)

A = np.array([[u], [-u], [u**2], [-u**2], [3 * ( u**3 ) - 2 * u], [-3 * ( u**3 ) + 2 * u], [
                                                ((u < 0)*1).tolist()]]).squeeze()
b = np.array([0.1, 0.1, 0.5, -0.5, -0.2, 0.3, 0.4])
```

```
p = cp.Variable(n)
constr = []
constr += [A@p <= b]
constr += [cp.sum(p) ==1]
constr += [p >= 0]

cost = cp.sum(cp.entr(p))
prob = cp.Problem(cp.Maximize(cost), constr)

prob.solve()

p = p.value

plt.plot(u, p)
plt.xlim(-1.3, 1.3)
plt.ylim(0, 0.05)
plt.xlabel('$u_i$', fontsize = 16)
plt.ylabel('$ Prob( X == u_i ) $', fontsize = 16)
#plt.rc('legend',fontsize=20) # using a size in points
#plt.rc('legend',fontsize='medium') # using a named size
ax = plt.gca()
ax.xaxis.set_tick_params(labelsize=14)
ax.yaxis.set_tick_params(labelsize=14)
plt.grid()
plt.legend()
#
plt.title('Maximum Entropy Distribution', fontsize = 16, color = 'k')
plt.tight_layout()
plt.show()
```

The result of the code for the *Maximum Entropy Distribution* problem has been presented in Fig. 9.

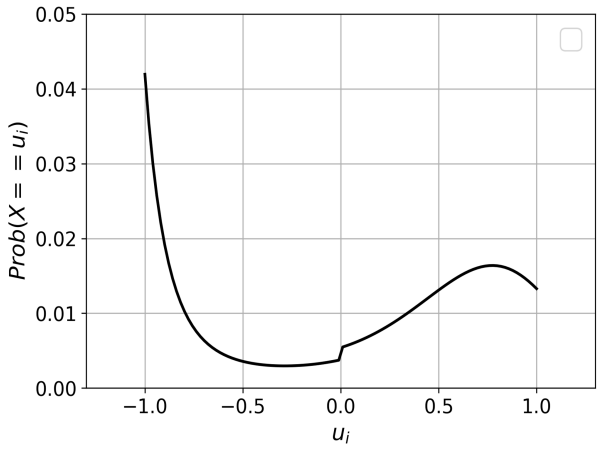

Fig. 9. The result for the *Maximum Entropy Distribution*.

N. *Analytical Center of a Polytope*

In this section, the analytical center of a polytope is formulated as a convex optimization problem. The problem can be cast as the following optimization problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & -\sum \log(\mathbf{b}-\mathbf{A}\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \preceq \mathbf{b}. \end{aligned} \tag{12}$$

The code for the problem has been given as follows.

```
#Analytical Center of a Polytope
import numpy as np
import matplotlib.pyplot as plt
import scipy

import cvxpy as cp


#%matplotlib qt

n = 2
px = np.array([0, .5, 2, 3, 1])
py = np.array([0, 1, 1.5, .5, -.5])

px = np.hstack((px, px[0]))
py = np.hstack((py, py[0]))

px_diff = px[1:] - px[:-1]
py_diff = py[1:] - py[:-1]

px_avg = 0.5*(px[1:] + px[:-1])
py_avg = 0.5*(py[1:] + py[:-1])


A = []
for i in range(0, len(px)-1):
    p = np.array([px_diff[i], py_diff[i]])
    p = p/np.linalg.norm(p)
    A.append([-p[1], p[0]])

A = np.array(A)

b = []
for i in range(0, len(px)-1):
    p = np.array([px_avg[i], py_avg[i]])
    b.append(A[i,:].dot(p))

#plt.plot(px, py)

m = A.shape[-1]
x = cp.Variable(m)

constr = []
constr += [A@x <= b]

prob = cp.Problem(cp.Minimize(-cp.sum(cp.log(b - A@x))), constr)
prob.solve(verbose = False)



xp = x.value
```

```
#-------- Display

plt.plot(px, py, 'k')
plt.plot(xp[0], xp[1], 'ro')
plt.title('Analytical Center of a Polytope')
plt.axis('off')
plt.show()
```

Fig. 10 illustrates the result of the code for the *analytical center of a polytope* problem.

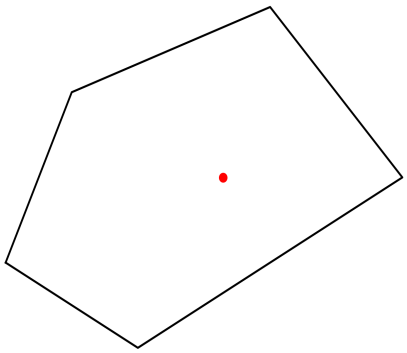

Fig. 10. The result for the *analytical center of a polytope*.

O. Bounding Correlation Coefficients

The bounding correlation coefficients is an interesting problem than can be formulated as the following optimization problem

$$\begin{aligned} \max_{\mathbf{R}} \quad & \mathbf{R}_{14} \\ \text{s.t.} \quad & 0.6 \leq \mathbf{R}_{12} \leq 0.9, \\ & 0.8 \leq \mathbf{R}_{13} \leq 0.9, \\ & 0.5 \leq \mathbf{R}_{24} \leq 0.7, \\ & -0.8 \leq \mathbf{R}_{34} \leq -0.4, \\ & \mathbf{R}_{11} = 1, \\ & \mathbf{R}_{22} = 1, \\ & \mathbf{R}_{33} = 1, \\ & \mathbf{R}_{44} = 1, \\ & \mathbf{R} \succeq 0. \end{aligned} \tag{13}$$

The code for the problem has been presented as follows.

```
#Bounding correlation coefficients
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp


#%matplotlib qt



m = 4
p = cp.Variable((m,m), PSD = True)
constr = []
constr += [p[0][1] <= 0.9]
constr += [0.6 <= p[0][1]]
constr += [p[0][2] <= 0.9]
constr += [0.8 <= p[0][2]]
constr += [p[1][3] <= 0.7]
constr += [0.5 <= p[1][3]]
constr += [p[2][3] <= -0.4]
constr += [-0.8 <= p[2][3]]
for i in range(m):
    constr += [p[i][i] == 1.]


prob = cp.Problem(cp.Maximize(p[0][3]), constr)
prob.solve(verbose = False)


print(f'p.value = {p.value}')
```

The result of the code is given as

```
R = [[ 1.          0.89999961   0.79999575   0.22992376]
 [ 0.89999961   1.          0.48779054  0.58355139]
 [ 0.79999575    0.48779054       1.         -0.40000125]
 [ 0.22992376  0.58355139   -0.40000125     1.]]
```

P. Euclidean Projection on a Half-Space

The problem of Euclidean projection on a half-space is presented in this section. The problem can be cast as the following convex optimization problem

$$\begin{aligned} \min_{\mathbf{R}} \quad & ||\mathbf{x} - \mathbf{x_0}||_2 \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \preceq \mathbf{b}. \end{aligned} \tag{14}$$

The code has been given as follows.

```
#Euclidean projection on a halfspace
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp


#%matplotlib qt
```

```
n = 2
px = np.array([-5, 5])
py = np.array([-10, 10])



px_diff = px[1:] - px[:-1]
py_diff = py[1:] - py[:-1]

px_avg = 0.5*(px[1:] + px[:-1])
py_avg = 0.5*(py[1:] + py[:-1])


A = []
for i in range(0, len(px)-1):
    p = np.array([px_diff[i], py_diff[i]])
    p = p/np.linalg.norm(p)
    A.append([-p[1], p[0]])

A = np.array(A)


b = []
for i in range(0, len(px)-1):
    p = np.array([px_avg[i], py_avg[i]])
    b.append(A[i,:].dot(p))

x0 = [-5, 5]
m = 2
x = cp.Variable(m)

constr = []
constr += [A@x <= b]

prob = cp.Problem(cp.Minimize(cp.norm(x - x0, 2)), constr)
prob.solve(verbose = False)



xp = x.value



#-------- Display

plt.plot(px, py, 'b', lw = 2)
plt.plot(xp[0], xp[1], 'k*', markersize = 10)
plt.plot(x0[0], x0[1], 'ro')
plt.xlim(-10,10)
plt.ylim(-10,10)
plt.title('Euclidean projection on a halfspace')
#plt.axis('off')
plt.show()
```

Fig. 11 shows the output of the code for the *Euclidean projection on a half-space* problem.

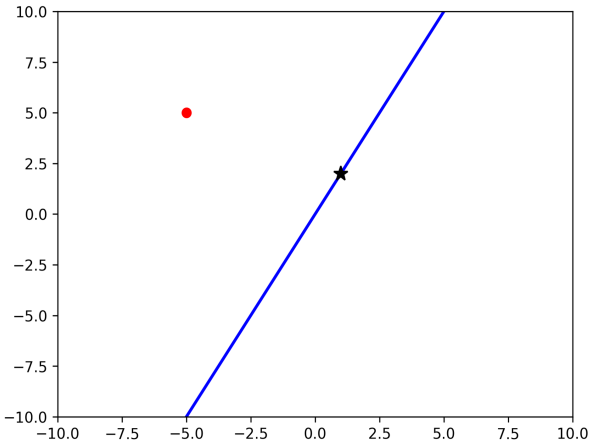

Fig. 11. The result for the *Euclidean projection on a half-space*.

Q. Euclidean Projection on Non-Negative Orthant

The problem of Euclidean projection on non-negative orthant which is special case of the Euclidean projection on a halfspace is presented in this section. The problem can be cast as the following convex optimization problem

$$\begin{aligned} \min_{\mathbf{R}} \quad & ||\mathbf{x} - \mathbf{x_0}||_2 \\ \text{s.t.} \quad & \mathbf{x} \succeq \mathbf{0}. \end{aligned} \tag{15}$$

The code has been given as follows.

```
#Euclidean projection on nonnegative orthant
import numpy as np
import matplotlib.pyplot as plt
import scipy

import cvxpy as cp


%matplotlib qt



x0 = np.array([5, -5])
m = 2
x = cp.Variable(m)

constr = []
constr += [x >= 0]

prob = cp.Problem(cp.Minimize(cp.norm(x - x0)), constr)
prob.solve(verbose = False)



xp = x.value



#-------- Display
```

```
plt.plot(xp[0], xp[1], 'k*')
plt.plot(x0[0], x0[1], 'ro')
plt.xlim(-10,10)
plt.ylim(-10,10)
#plt.axis('off')
```

Fig. 12 illustrates the result of the code for the *Euclidean projection on non-negative orthant* problem.

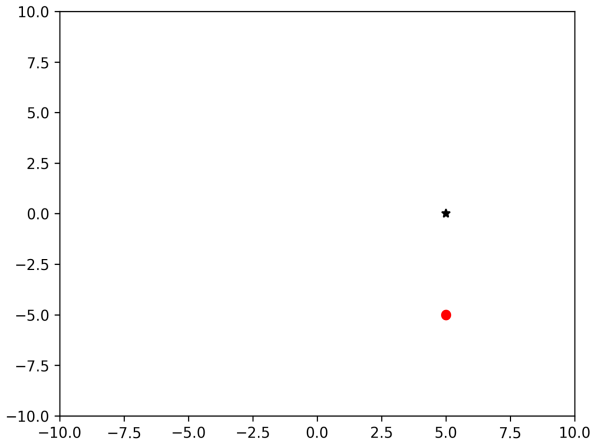

Fig. 12. The result for the *Euclidean projection on non-negative orthant*.

R. *Linear Discrimination*

The linear discrimination problem which is used in decision making problems can be cast as the following optimization problem

$$\begin{aligned} \min_{\mathbf{a},b} \quad & 0 \\ \text{s.t.} \quad & \mathbf{a}\mathbf{X} - b \succeq \mathbf{1}, \\ & \mathbf{a}\mathbf{Y} - b \preceq -\mathbf{1}, \end{aligned} \tag{16}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are $2 \times M$ matrices in which M is the number of data points. The code has been given as follows.

```
\begin{python}
#linear discrimination
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp


#%matplotlib qt
```

```
x = np.random.multivariate_normal([-3,-3], [[1,0],[0,1]],  10)
y = np.random.multivariate_normal([3,3], [[1,0],[0,1]],  10)



a = cp.Variable((1,x.shape[-1]))
b = cp.Variable(1)


constr = []

for i in range(x.shape[0]):
    constr += [a@x[i,:].reshape(-1,1) - b >= 1]
    constr += [a@y[i,:].reshape(-1,1) - b <= -1]
cost = 0
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = False)


# -------  Display

z = np.linspace(-5,5,100)
plt.plot(z, (b.value - a.value.squeeze()[0]*z)/a.value.squeeze()[1])
plt.plot(x[:,0], x[:,1], 'k*')
plt.plot(y[:,0], y[:,1], 'ro')
plt.title('Linear Discrimination')
plt.show()
```

Fig. 13 illustrates the result of the code for the *linear discrimination* problem.

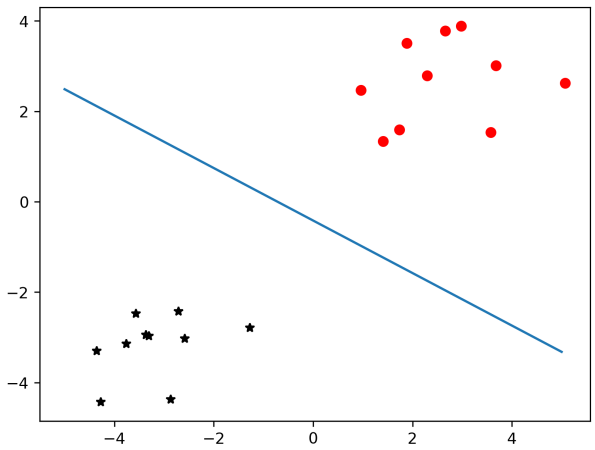

Fig. 13. The result for the *linear discrimination*.

S. *Linear Placement Problem*

The code for this problem is given as follows.

```
#Linear placement problem
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
```

```
import cvxpy as cp


#%matplotlib qt

fixed = np.array([[ 1,   1,  -1, -1,    1,   -1,  -0.2,  0.1], [ 1,  -1,  -1,  1, -0.5, -0.2,
                                                 -1,    1]]).T
M = fixed.shape[0]
N = 6

# first N columns of A correspond to free points,
# last M columns correspond to fixed points

A = np.array([[ 1,  0,  0, -1,  0,  0,    0,  0,  0,  0,  0, 0,  0, 0],
              [1,  0, -1,  0,  0,  0,    0,  0,  0,  0,  0,  0,  0,  0],
      [1,  0,  0,  0, -1,  0,    0,  0,  0,  0,  0,  0,  0,  0],
      [1,  0,  0, 0,  0,  0,   -1,  0,  0,  0,  0,  0,  0,  0],
      [1,  0,  0,  0,  0,  0,    0, -1,  0,  0,  0,  0,  0,  0],
      [1,  0, 0,  0,  0, 0,    0,  0,  0,  0, -1, 0,  0,  0],
      [1,  0,  0,  0,  0,  0,    0,  0,  0,  0,  0,  0,  0, -1],
      [0,  1, -1,  0,  0,  0,    0,  0,  0,  0,  0,  0,  0,  0],
      [0,  1,  0, -1,  0,  0,    0,  0,  0,  0,  0,  0,  0,  0],
      [0,  1,  0,  0,  0, -1,    0,  0,  0, 0,  0,  0,  0,  0],
      [0,  1,  0,  0, 0,  0,    0, -1,  0,  0,  0,  0,  0,  0],
      [0,  1,  0,  0,  0,  0,    0,  0, -1,  0,  0,  0,  0,  0],
      [0,  1,  0,  0,  0,  0,    0,  0,  0,  0,  0,  0, -1, 0],
      [0,  0,  1, -1,  0,  0,    0,  0,  0,  0,  0,  0,  0,  0],
      [0,  0,  1,  0,  0,  0,    0, -1,  0,  0,  0,  0,  0,  0],
      [0,  0,  1,  0,  0,  0,    0,  0,  0,  0, -1,  0,  0,  0],
      [0,  0,  0,  1, -1,  0,    0,  0,  0,  0,  0,  0,  0,  0],
      [0,  0,  0,  1,  0,  0,    0,  0, -1,  0,  0,  0,  0,  0],
      [0,  0,  0,  1,  0,  0,    0,  0,  0, -1,  0,  0,  0,  0],
      [0,  0,  0,  1,  0,  0,    0,  0,  0,  0,  0, -1,  0,  0],
      [0,  0,  0,  1,  0, -1,    0,  0,  0,  0,  0, -1,  0,  0] ,
      [0,  0,  0,  0,  1, -1,    0,  0,  0,  0,  0,  0,  0,  0],
      [0,  0,  0,  0,  1,  0,   -1,  0,  0,  0,  0,  0,  0,  0],
      [0,  0,  0,  0,  1,  0,    0,  0,  0, -1,  0,  0,  0,  0],
      [0,  0,  0,  0,  1,  0,    0,  0,  0,  0,  0,  0,  0, -1],
      [0,  0,  0,  0,  0,  1,    0,  0, -1,  0,  0,  0,  0,  0],
      [0,  0,  0,  0,  0,  1,    0,  0,  0,  0, -1,  0,  0,  0]])


x = cp.Variable((A.shape[-1], 2))
cost = cp.sum_squares(A@x)

constr = []
for i in range(A.shape[0]):
    constr += [x[N:,:] == fixed]

prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = False)



x = x.value

#------ Display

plt.plot(x[:N,0], x[:N,1], 'gs', label = 'free points')
plt.plot(x[N:,0], x[N:,1], 'rs', label = 'fixed points')

for i in range(A.shape[0]):
    ind = np.nonzero(A[i,:])
```

```
    for idx in ind:
        plt.plot(x[idx,0],x[idx,1], 'k-.', lw = 0.2)

plt.legend()
plt.title('Linear Placement Problem')
plt.xlim(-1.2,1.2)
plt.ylim(-1.2,1.2)
plt.show()
```

Fig. 14 illustrates the result of the code for the *Linear placement problem* code.

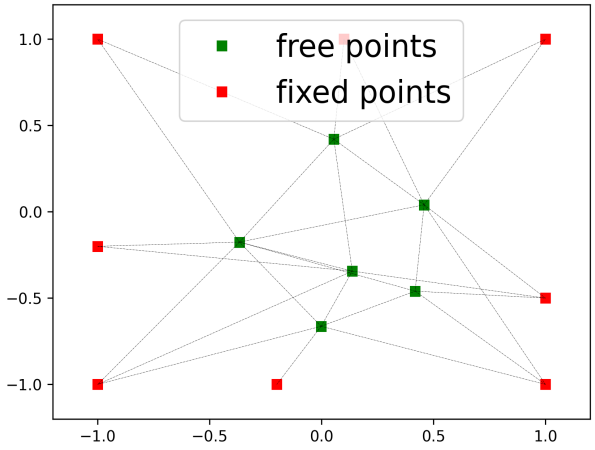

Fig. 14. The result for the *Linear placement problem*.

T. Maximum Volume Inscribed Ellipsoid

The maximum volume inscribed ellipsoid is addressed in this section. The problem can be cast as the following optimization problem.

$$\begin{aligned} \max_{\mathbf{B}} \quad & \log\det \mathbf{B} \\ \text{s.t.} \quad & \sup_{||\mathbf{u}||_2 \leq 1} \mathrm{I}_{\mathrm{C}}(\mathbf{B}\mathbf{u}+\mathbf{d}) \leq 0, \end{aligned} \tag{17}$$

The code for this problem is been given as follows.

```
#max volume inscribed ellipsoid

import numpy as np
import matplotlib.pyplot as plt
import scipy

import cvxpy as cp


n = 2
px = np.array([0, .5, 2, 3, 1])
py = np.array([0, 1, 1.5, .5, -.5])

px = np.hstack((px, px[0]))
py = np.hstack((py, py[0]))

px_diff = px[1:] - px[:-1]
```

```
py_diff = py[1:] - py[:-1]

px_avg = 0.5*(px[1:] + px[:-1])
py_avg = 0.5*(py[1:] + py[:-1])


A = []
for i in range(0, len(px)-1):
    p = np.array([px_diff[i], py_diff[i]])
    p = p/np.linalg.norm(p)
    A.append([-p[1], p[0]])

A = np.array(A)

b = []
for i in range(0, len(px)-1):
    p = np.array([px_avg[i], py_avg[i]])
    b.append(A[i,:].dot(p))

#plt.plot(px, py)

m = A.shape[-1]
X = cp.Variable((m,m), symmetric = True)
d = cp.Variable(m)

constr = []
for i in range(A.shape[0]):
    constr += [cp.norm( X@A[i,:].reshape(-1,1) , 2) + A[i,:].reshape(1,-1)@d <= b[i]]

prob = cp.Problem(cp.Maximize(cp.log_det(X)), constr)
prob.solve(verbose = False)



X = X.value
d = d.value


#---- Display

N = 100


theta = np.linspace(0,2*np.pi, N).reshape(1,-1)

ellipse = X@np.array([[np.cos(theta)], [np.sin(theta)]]).squeeze() + d.reshape(-1,1) @ np.
                                                    ones((1,N))

plt.plot(px, py, 'b', lw = 2)
plt.plot(ellipse[0,:], ellipse[1,:], 'r')
plt.title('Max Volume Inscribed Ellipsoid')
plt.axis('off')
plt.show()
```

Fig. 15 illustrates the result of the code for the *max volume inscribed ellipsoid* code.

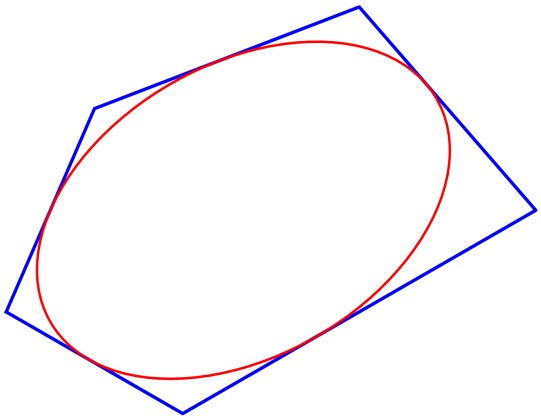

Fig. 15. The result for the max volume inscribed ellipsoid.

U. *One Free Point Localization*

In this section the problem of one free point localization is presented. The code for the problem is given as follows.

```
#One free point localization
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp


#%matplotlib qt




p = np.random.multivariate_normal([0,0], [[10,0],[0,10]],  10)


x = cp.Variable((1,p.shape[-1]))

constr = []
cost = cp.sum(cp.norm(x - p ,1))
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = False)



xp = x.value


# -------  Display
plt.plot(p[:,0], p[:,1], 'k*')
plt.plot(xp[:,0], xp[:,1], 'ro')
plt.title('One free point localization')
plt.show()
```

Fig. 16 illustrates the result of the code for the *One free point localization* code.

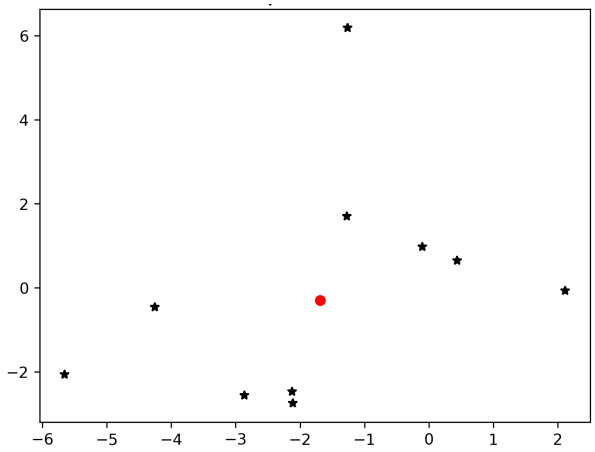

Fig. 16. The result for the *One free point localization*.

V. *Polynomial Discrimination*

The polynomial discrimination is described in this section. The code code for the problem is given as follows.

```
#polynomial discrimination
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp
import scipy.io

#%matplotlib qt



X = np.array([[-2.95855623e-02,  1.95754105e-01,  9.38421496e-02,
        -3.29467959e-02, -1.73989577e-01, -7.97793709e-01,
        -1.11389191e-01, -7.42729742e-01, -1.29151943e-01,
        -5.17979960e-01, -1.09758793e-01, -2.30802197e-01,
        -7.73237398e-03,  7.09756728e-02, -3.52385404e-01,
        -5.31747802e-01, -3.04772028e-01, -1.29998958e-01,
        -4.48261427e-02, -3.27872167e-03, -4.41183114e-01,
        -4.05461983e-01, -1.99804573e-01,  4.26455138e-03,
         2.62474869e-03, -3.51403372e-01, -3.96796643e-01,
        -2.97028945e-01, -1.39562473e-01, -5.57034030e-01,
        -4.39141850e-01,  8.26766799e-04, -8.98575132e-02,
         1.98784110e-01,  1.48327407e-01, -7.65239272e-01,
        -1.64675275e-01,  1.46751529e-01, -6.94701156e-02,
        -2.10891761e-01, -4.41468102e-01, -3.05412384e-01,
        -4.84305217e-02, -2.79176289e-01, -5.10060070e-01,
        -7.98311175e-02, -3.10935435e-01, -3.72311823e-01,
         2.19521351e-01, -4.94892629e-01,  1.41049092e-01,
        -3.87453856e-01,  1.34704940e-01, -3.12572867e-01,
        -6.26808596e-01, -3.25919656e-01, -6.47490319e-01,
        -1.60047770e-01, -1.02291418e-01, -2.23325151e-01,
         1.01700226e-01, -5.31054659e-01, -4.42743514e-01,
         3.72895510e-01, -2.93658107e-02],
       [-3.27235847e-01,  4.95076766e-01,  5.29947667e-02,
        -9.59639561e-03,  7.63301831e-01, -2.65502307e-01,
        -9.16862531e-02,  2.55875621e-01, -1.70749169e-01,
```

```
         2.63716029e-01,  8.63248949e-01, -5.52139923e-01,
         4.47627192e-03, -1.00834217e-01, -5.59729838e-01,
         3.16068644e-01,  6.43951955e-02,  7.36883561e-02,
         1.32683793e-01,  3.80192236e-01, -5.87038834e-01,
         8.70099909e-02,  3.12597219e-01,  2.65889204e-03,
         4.41683197e-02,  5.15480333e-01, -4.29987771e-01,
        -3.98981772e-01,  1.11941884e-01, -7.95167137e-02,
         4.92736689e-01, -8.85028346e-03, -3.67032553e-01,
         6.48520918e-01,  6.98918608e-01, -3.16115610e-01,
         2.87103155e-01, -6.41579927e-01, -1.60079670e-01,
        -5.80573323e-03,  2.21465550e-01,  7.80819944e-01,
         3.49373009e-01, -2.23253724e-01,  2.95930684e-01,
        -7.49587566e-01, -1.23183214e-01, -8.86820600e-02,
        -4.88104745e-01,  1.19707947e-01, -6.29274097e-01,
        -2.48284959e-01, -6.85654170e-01,  3.10257939e-01,
        -6.87885168e-02, -8.22196336e-01,  3.28320725e-01,
         4.06064597e-01, -1.32930751e-02, -2.41369680e-01,
        -7.44523222e-02, -5.16360847e-01, -4.14590950e-01,
        -8.18568152e-01,  4.40919531e-02]])


Y = np.array([[-1.10929648, -1.19714993, -1.46155265,  0.54850428,  1.35098877,
        -0.63978276,  1.02322679,  1.52509307,  0.61292653,  2.04638631,
         1.82855895,  1.47921071,  0.30167283,  1.73791846,  0.7515328 ,
        -1.25628825, -0.88484211,  0.53786931,  1.53502706,  1.18898028,
         0.91609724, -1.54078336,  0.16536052, -1.19906352, -1.46105915,
         1.27808057, -1.12648287,  0.3258603 ,  1.14215806,  0.28618332,
         1.47693679,  0.89203912, -0.68661538, -1.388819  ,  1.73895725,
         1.57556233,  1.28373784, -0.87911447,  1.40727042, -1.40937883,
        -1.28074118, -1.48632542,  0.38125655, -1.52045199, -1.66213125,
        -1.94551437,  0.21882479,  0.11292287, -1.41478813, -1.87993473,
        -1.00727723,  0.02675306, -0.50369109, -1.39723257, -1.22424294,
         1.30030558, -1.01635215,  1.0255251 , -0.08169464, -0.25732126,
         1.40042328, -2.00438398,  1.36791438, -0.89550151, -1.55659656,
        -0.70043184,  0.99463546,  1.61911896, -1.79102261,  0.82796362,
        -1.17658101,  0.07050389,  1.21510545, -0.26214263,  1.28184009,
        -1.80832372,  1.53041901, -0.66284338, -1.14343021,  0.8866055 ,
        -1.22024598, -0.86028255,  0.44238846, -0.39174046, -0.33583426,
        -1.0828518 ,  0.22247235,  0.06527953,  1.66426983,  0.95034167,
         0.29308652, -0.70090159, -0.7519563 , -1.04020311, -1.73012502,
         1.17321142, -0.01267105,  0.44442947,  1.43795889, -0.96444755,
        -1.61170243,  1.07293517, -1.390684  ,  1.86297062, -1.17721976,
        -0.85478808,  0.94232596,  0.90095948,  0.78233839,  0.6769013 ,
         0.57207325,  1.07070613,  0.94326805, -0.88960536, -0.70306538,
         0.49395402, -0.72786602, -1.88687969, -0.32508892,  1.30512574,
         0.33981014,  0.30344273,  0.23495848,  0.34879119,  0.29813592,
         0.7812839 ,  0.61364035,  0.38713553,  0.75876337,  0.70237945,
         0.39402689,  0.31522323,  0.39765858,  0.5236989 ,  0.74799895,
         0.28614548,  0.24846108,  0.64083345,  0.49758925,  0.48985264,
         0.28120179,  0.42798242,  0.58821225,  0.41714166,  0.30514283,
         0.62799254],
       [ 0.3852498 ,  0.48917503,  0.23041757,  1.26242449,  1.2942288 ,
         1.26090287, -1.40510591,  1.42402683, -1.4810577 ,  0.07343144,
        -0.61127391, -1.37203234, -1.17497071, -0.79741137, -1.8623418 ,
        -0.01485873,  0.84298797, -1.78331797,  0.98114637,  1.2863001 ,
        -1.61188272,  0.85716211, -1.97658857, -0.6671302 , -0.41711143,
         0.30076968, -0.14360928, -1.13532496,  1.58119757, -1.41148329,
         0.52105578, -1.80541092, -1.13739852,  1.38593001, -0.24846801,
        -1.21754355,  0.93499571,  1.89903227,  0.43621015, -0.82161215,
        -0.03838718, -0.59754699, -1.47367586,  0.88731639,  0.61900299,
         0.66901519, -1.27319787,  1.205941  , -0.87794017, -0.86446196,
        -0.49818959,  1.97003418, -1.00804074, -0.81891045,  0.4138192 ,
        -1.26725377,  0.88957589,  1.75638958,  1.23448912,  1.60107613,
```

```
        -1.42114194,  0.3921614 ,  0.43390617, -1.24962578, -0.21339644,
        -1.03449096,  0.73352648,  0.20857677,  0.35460486,  1.2492346 ,
         0.86176662,  1.3834841 , -1.54888459, -1.70639443, -0.39449858,
         1.02390852, -0.82895152,  1.15288164,  1.36591446,  1.52710936,
        -0.1884893 ,  0.72104447, -1.28836322,  1.15171548, -1.12005379,
        -1.62522906, -1.26085395, -1.13053544, -0.77038397, -1.33230553,
         1.34445451,  1.29038998, -0.8204131 ,  1.69555841, -0.93375822,
        -0.67643161, -1.66943251,  1.17823186,  0.89618357, -1.0573024 ,
        -0.7036574 ,  1.64458793,  0.92030731,  0.92614406,  1.48093904,
         0.9156026 , -1.68778436,  0.92584162, -1.55306823, -1.63437019,
         1.7825389 ,  1.02149349, -1.21931587,  1.30945838,  1.55670293,
        -1.17650937, -0.82744872,  0.11239135,  1.58052194,  0.14566471,
        -0.20296804, -0.03981107,  0.15737806, -0.06000793,  0.28542558,
         0.05675917, -0.40796816,  0.00328204, -0.25757154,  0.31645786,
         0.1985505 , -0.26469693,  0.07997021, -0.61982956,  0.47241106,
        -0.21855985, -0.22959073,  0.40903737, -0.64462399,  0.28872615,
        -0.10924733, -0.406085  ,  0.2233023 ,  0.4763915 ,  0.40876068,
        -0.59554049]])


m = 2
P = cp.Variable((m,m), PSD = True)
q = cp.Variable(m)
r = cp.Variable(1)


cost = 0

constr = []
for i in range(X.shape[-1]):
    constr += [(X[:,i].reshape(1,-1)@P)@X[:,i].reshape(-1,1) + q.T@X[:,i].reshape(-1,1) + r <
                                                        = -1]
    constr += [(Y[:,i].reshape(1,-1)@P)@Y[:,i].reshape(-1,1) + q.T@Y[:,i].reshape(-1,1) + r >
                                                        = 1]

prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = True)



P = P.value
q = q.value
r = r.value

points = np.vstack((np.linspace(-1,1,100).reshape(1,-1), np.linspace(-1,1,100).reshape(1,-1))
                                                    )
discr = (points.T@P)@points + points.T@q.reshape(-1,1) + r


#plt.plot(X[0,:], X[1,:], 'ro')
#plt.plot(Y[0,:], Y[1,:], 'go')
```

W. *Robust Linear Discrimination*

The robust linear discrimination problem is a more advanced version of what was presented as linear discrimination in previous sections. The problem can be cast as

$$\begin{aligned} \max_{t,\mathbf{a},b} \quad & t \\ \text{s.t.} \quad & \mathbf{aX} - b \succeq t\mathbf{1}, \\ & \mathbf{aY} - b \preceq -t\mathbf{1}, \\ & ||\mathbf{a}||_2 \leq 1. \end{aligned} \tag{18}$$

In (18), $\mathbf{X}$ and $\mathbf{Y}$ are $2 \times M$ matrices in which M is the number of data points. The code for the problem is given as follows.

```
#Robust Linear Discrimination
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp


#%matplotlib qt




x = np.random.multivariate_normal([-2,-2], [[1,0],[0,1]],  10)
y = np.random.multivariate_normal([2,2], [[1,0],[0,1]],  10)



a = cp.Variable((1,x.shape[-1]))
b = cp.Variable(1)
t = cp.Variable(1)

constr = []

for i in range(x.shape[0]):
    constr += [a@x[i,:].reshape(-1,1) - b >= t]
    constr += [a@y[i,:].reshape(-1,1) - b <= -t]
constr += [cp.norm(a,2) <= 1]
cost = t
prob = cp.Problem(cp.Maximize(cost), constr)
prob.solve(verbose = False)

a = a.value.squeeze()
b = b.value
t = t.value
# -------  Display

z = np.linspace(-5,5,100)
plt.plot(z, (b - a[0]*z)/a[1])
plt.plot(z, (t + b - a[0]*z)/a[1], 'r--')
plt.plot(z, (-t + b - a[0]*z)/a[1], 'r--')
plt.plot(x[:,0], x[:,1], 'k*')
plt.plot(y[:,0], y[:,1], 'ro')
plt.title('Robust Linear Discrimination')
plt.show()
```

Fig. 17 illustrates the result of the code for the *Robust Linear Discrimination* problem.

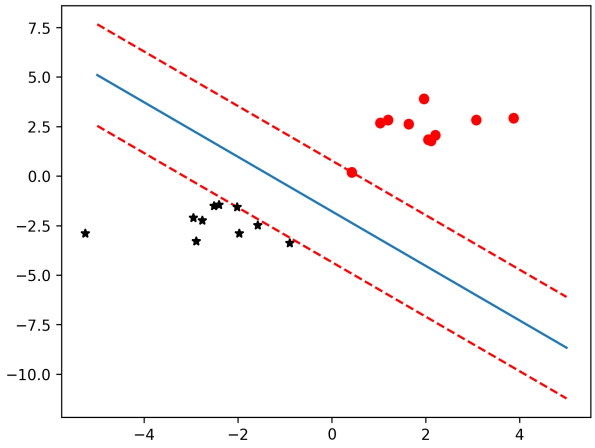

Fig. 17. The result for the *Robust Linear Discrimination*.

X. *Separating Ellipsoids in 2D*

In this section the problem of separating ellipsoids is cast as a convex optimization problem and is presented. The code for the problem is given as follows.

```
#Separating ellipsoids in 2D

import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import cvxpy as cp


#%matplotlib qt

n = 2
A = np.eye(n)
b = np.zeros(n)
C = np.array([[2, 1], [-.5, 1]])
d = np.array([-3, -3])


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

constr = []

constr += [cp.norm(A@x + b, 2) <= 1]
constr += [cp.norm(C@y + d, 2) <= 1]
constr += [x - y == w]
cost = cp.norm(w, 2)
prob = cp.Problem(cp.Minimize(cost), constr)
prob.solve(verbose = False)

w = w.value
x = x.value
```

```
y = y.value
# -------  Display


angle = np.linspace(0,2*np.pi,100).reshape(1,-1)
points = np.vstack((np.cos(angle), np.sin(angle)))

ellipse_1 = np.linalg.inv(A)@(points - b.reshape(-1,1))
ellipse_2 = np.linalg.inv(C)@(points - d.reshape(-1,1))

z = (x+y)/2.
m = (y[1] - x[1])/(y[0] - x[0])
m_ = (-90 + np.arctan(m)*180/np.pi)*np.pi/180
b_ = z[1] - m_*z[0]

t = np.linspace(-1.5, 2, 100)
plt.plot(t, m_*t + b_, 'm')
plt.plot([x[0], y[0]], [x[1], y[1]])
plt.plot(x[0], x[1], 'kx', markersize = 10)
plt.plot(y[0], y[1], 'rx', markersize = 10)
plt.plot(ellipse_1[0,:], ellipse_1[1,:], 'b--')
plt.plot(ellipse_2[0,:], ellipse_2[1,:], 'g--')
plt.xlim(-2,4)
plt.ylim(-2,4)
plt.title('Separating Ellipsoids in 2D')
plt.show()
```

Fig. 18 depicts the output of the code for the *Separating ellipsoids in 2D* problem.

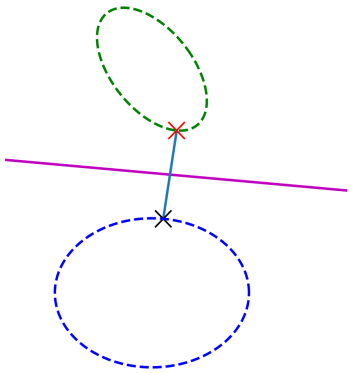

Fig. 18. The result for the *Separating ellipsoids in 2D*.

IV. Digital Filter Design

This section has been dedicated to Finite Impulse Response (FIR) digital filter design. The approach is based on the Chebyshev design which is described as

$$\begin{aligned} \min_{\mathbf{h},\delta_2} \quad & \delta_2 \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq H(\omega_k) \leq \delta_1, \quad 0 \leq \omega_k \leq \omega_p, \\ & -\delta_2 \leq H(\omega_k) \leq \delta_2, \quad \omega_s \leq \omega_k \leq \pi, \end{aligned} \tag{19}$$

where ω_p, ω_s are the pass-band and stop-band cut-off frequencies, respectively. In addition, $\pm 20\log_1 0\delta_1$ is the pass-band ripple. Furthermore, $-20\log_1 0\delta_2$ is the stop-band attenuation.

Finally $H(\omega) = \sum_{n=1}^{N} h[n]\exp(-j\omega n/N)$.

Digital filter design problem can be written in different forms. The convex optimization problem given in (19) is based on minimizing the stop-band attenuation.

A. *Low Pass Filter*

The Low Pass Filter (LPF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max|\mathbf{H}(\omega_\mathbf{i})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_\mathbf{k}) \leq \delta_1, \quad 0 \leq \omega_k \leq \omega_p, \\ & \omega_s \leq \omega_i \leq \pi. \end{aligned} \tag{20}$$

The code for the filter design has been given as follows.

```
#Low pass filter

import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:


    N  = 256
    Fs = 40
    fp = 3/Fs
    fs = 9/Fs
    filter_order = 10
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass




class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()
```

```
    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))
        index_p = np.where(Params.freq<Params.fp)[0].tolist()
        index_s = np.where(((Params.fs<Params.freq) & (Params.freq<1/2.)))[0].tolist()
        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):

    def __init__(self):
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
```

```
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', FontSize = 16)
        plt.ylabel('$IPR$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$Phase\;\;[rad]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()
```

```
if __name__ == '__main__':

    msg = 'Low'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 19 illustrates the result of the code for the LPF design.

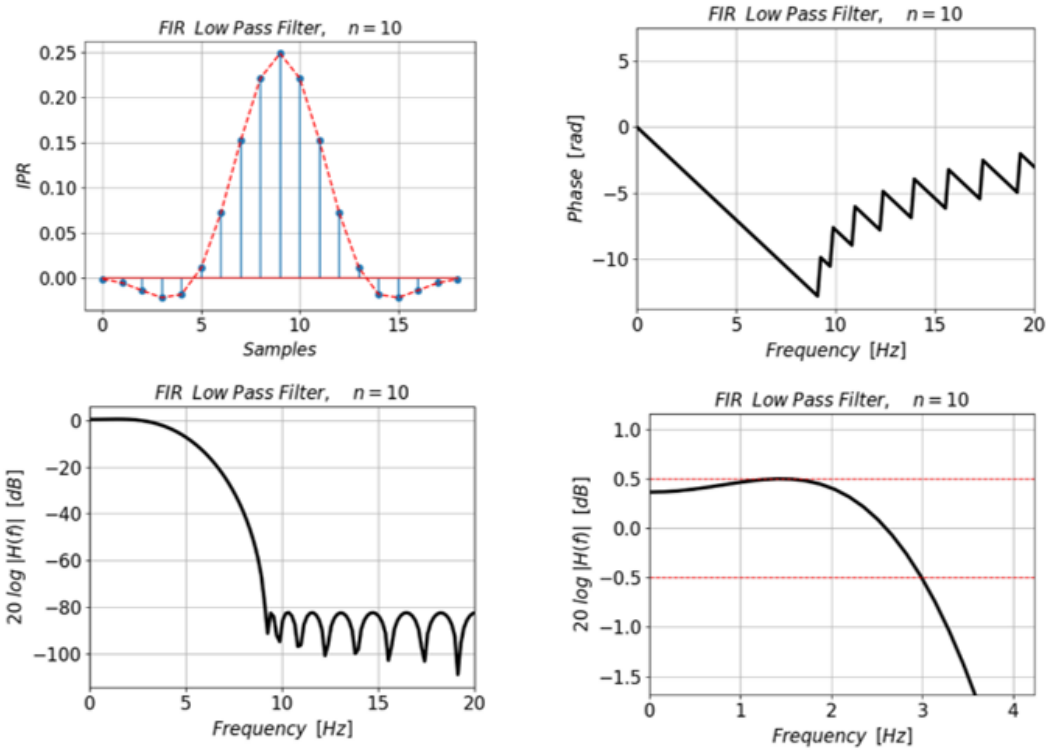

Fig. 19. The result for the LPF design.

B. *Band Pass Filter*

The Band Pass Filter (BPF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_{\mathbf{i}})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_{\mathbf{k}}) \leq \delta_1, \quad \omega_{p1} \leq \omega_k \leq \omega_{p2}, \\ & 0 \leq \omega_i \leq \omega_{s1} \cup \omega_{s2} \leq \omega_i \leq \pi. \end{aligned} \tag{21}$$

The code for the filter design is given as follows.

```
# Band pass filter
```

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:


    N  = 256
    Fs = 40
    fp1 = 4/Fs
    fp2 = 7/Fs
    fs1 = 2/Fs
    fs2 = 9/Fs
    filter_order = 10
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass






class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))
        index_p = np.where(((Params.fp1<Params.freq) & (Params.freq<Params.fp2)))[0].tolist()
        index_s1 = np.where(((0<Params.freq) & (Params.freq<Params.fs1)))[0].tolist()
        index_s2 = np.where(((Params.fs2<Params.freq) & (Params.freq<1/2.)))[0].tolist()
        index_s = np.hstack((index_s1, index_s2))
        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
```

```
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):

    def __init__(self):
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', FontSize = 16)
        plt.ylabel('$IPR$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()
```

```
    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$Phase\;\;[rad]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



if __name__ == '__main__':

    msg = 'Band'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 20 shows the result of the code for the BPF design.

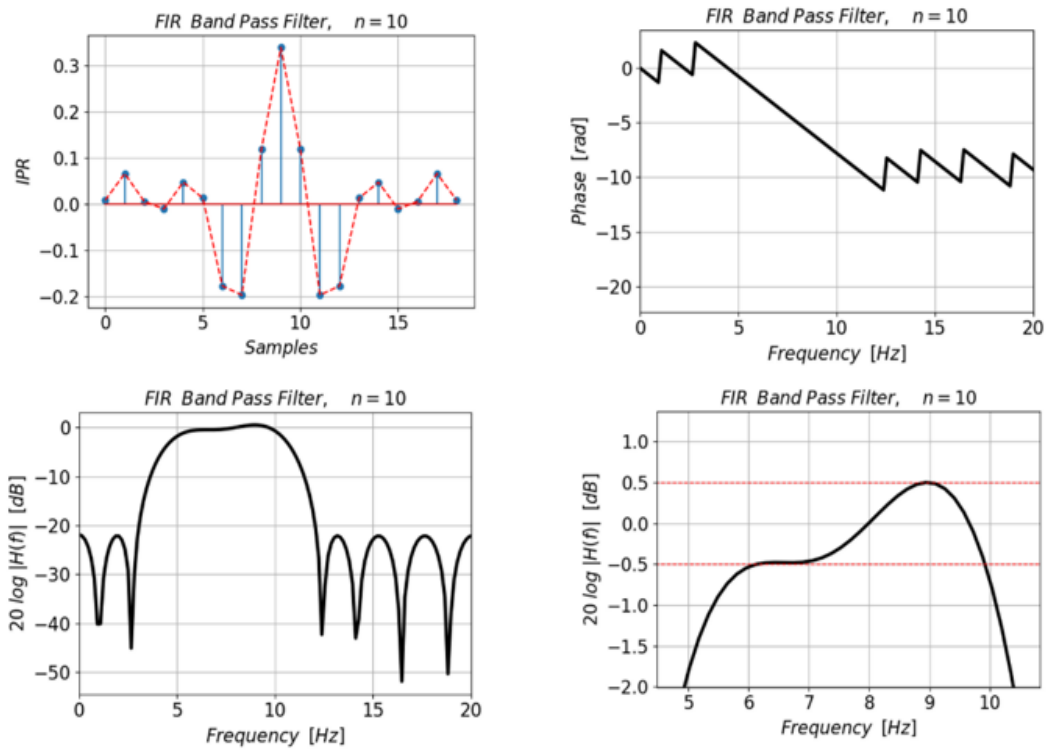

Fig. 20. The result for the BPF design.

C. *Band Stop Filter*

The Band Stop Filter (BSF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_{\mathbf{i}})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_{\mathbf{k}}) \leq \delta_1, \quad 0 \leq \omega_k \leq \omega_{s1} \cup \omega_{s2} \leq \omega_k \leq \pi, \\ & \omega_{p1} \leq \omega_i \leq \omega_{p2}. \end{aligned} \tag{22}$$

The code for the filter design is given as follows.

```
# Band stop filter


import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:
```

```
    N  = 256
    Fs = 40
    fs1 = 4/Fs
    fs2 = 7/Fs
    fp1 = 2/Fs
    fp2 = 9/Fs
    filter_order = 15
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass




class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))
        index_s = np.where(((Params.fs1<Params.freq) & (Params.freq<Params.fs2)))[0].tolist()
        index_p1 = np.where(((0<Params.freq) & (Params.freq<Params.fp1)))[0].tolist()
        index_p2 = np.where(((Params.fp2<Params.freq) & (Params.freq<1/2.)))[0].tolist()
        index_p = np.hstack((index_p1, index_p2))
        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):
```

```
    def __init__(self):
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', fontsize = 16)
        plt.ylabel('$IPR$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
```

```
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$Phase\;\;[rad]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        plt.tight_layout()
        plt.show()



if __name__ == '__main__':

    msg = 'Band-Stop'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 21 shows the result of the code for the BSF design.

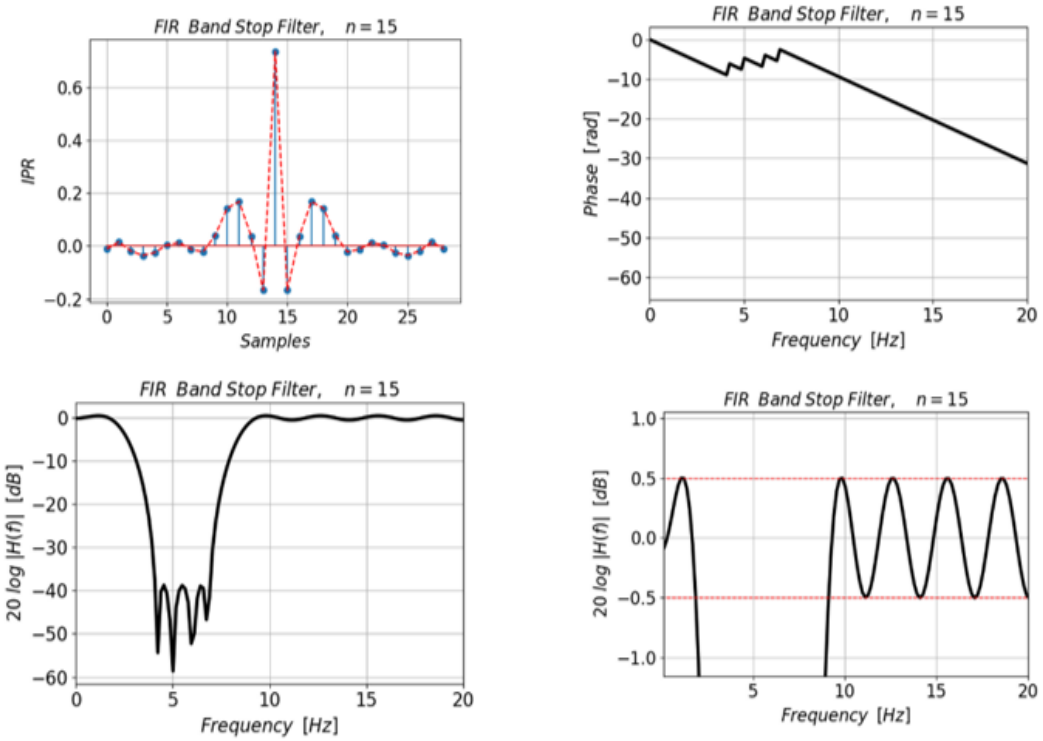

Fig. 21. The result for the BSF design.

D. *High Pass Filter*

The Low Pass Filter (LPF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_{\mathbf{i}})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_{\mathbf{k}}) \leq \delta_1, \quad \omega_p \leq \omega_k \leq \pi, \\ & 0 \leq \omega_i \leq \omega_s. \end{aligned} \tag{23}$$

The code for the filter design has been given as follows.

```
# High pass filter

import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:


    N  = 256
```

```
    Fs = 40
    fp = 15/Fs

    fs = 12/Fs
    filter_order = 10
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass




class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))

        index_s = np.where(((0<Params.freq) & (Params.freq<Params.fs)))[0].tolist()
        index_p = np.where(((Params.fp<Params.freq) & (Params.freq<1/2.)))[0].tolist()

        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):

    def __init__(self):
```

```
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', fontsize = 16)
        plt.ylabel('$IPR$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
```

```
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$Phase\;\;[rad]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



if __name__ == '__main__':

    msg = 'High'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 22 shows the result of the code for the HPF design.

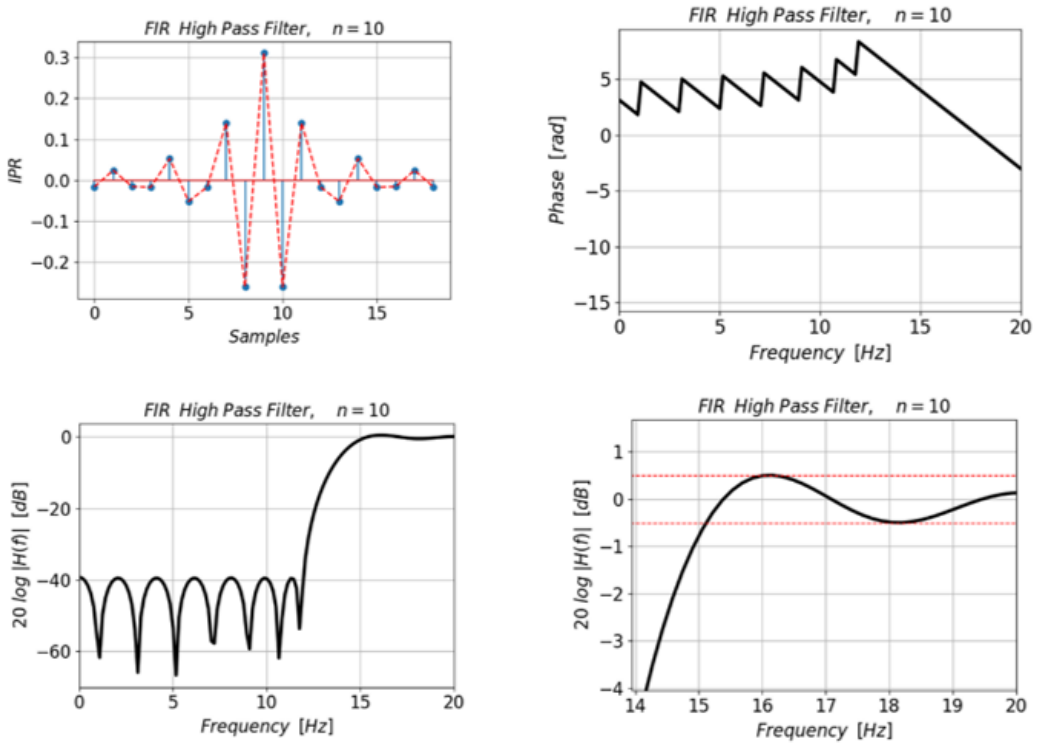

Fig. 22. The result for the HPF design.

Shahrokh Hamidi was born in Iran, in 1983. He received his B.Sc., M.Sc., and Ph.D. degrees all in Electrical and Computer Engineering. He is with the faculty of Electrical and Computer Engineering at the University of Waterloo, Waterloo, Ontario, Canada. His current research areas include statistical signal processing, mmWave imaging, Terahertz imaging, image processing, system design, multi-target tracking, wireless and digital communication systems, machine learning, optimization, and array processing.