CSCI 5254 Homework 5*

Tuguluke Abulitibu

November 12, 2020 †

Chapter 6, Function fitting and interpolation

6.9

Using the definition of quasiconvex, all we need to show is the level set

$$\{t_i | \max_{i=1,\ldots,k} |\frac{p(t_i)}{q(t_i)} - y_i| \leq \alpha\}$$

to be convex. Since

$$-\alpha q(t_i) \leq p(t_i) - y_i q(t_i) \leq \alpha q(t_i), \ i = 1, \ldots, k$$

linear inequality (polyhedron), hence convex. therefore the original minimization problem is quasiconvex.

Additional

3.9

$$\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}.$$

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$

(a)

With the given hint: $z = (\Re x, \Im x)$, by the definition of Modulus of a complex number

$$\|x_j\| = \frac{\sqrt{z_j^2 + z_j^2}}{\sqrt{(\Re x_j)^2 + (\Im x_j)^2}}$$

Setup a system of equations using the vector breakdown of x for its $\Re$ and $\Im$ components:

$$\boxed{\begin{array}{ll} \text{minimize} & \|z\|_2 \\ \text{subject to} & \begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix} z = \begin{bmatrix} \Re b \\ \Im b \end{bmatrix} \\ & z = (\Re x, \Im x) \end{array}}$$

*The codes of this HW are between python and matlab, I used matlab whenever I can not debug the py file, it would be great if we have some hints for cvxpy, since there are so many online materials for mat

†Late submission due to family matters.

(b)

Introducing another auxiliary variable t, SOCP:

$$\boxed{\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \|\begin{bmatrix} z_i \\ z_{i+n} \end{bmatrix}\| \le t, \forall i \\ & \begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix} z = \begin{bmatrix} \Re b \\ \Im b \end{bmatrix} \end{array}}$$

(c)

As seen in figure, infinity norm is the circle one, same as the definition: minimizing the maximum pf x.

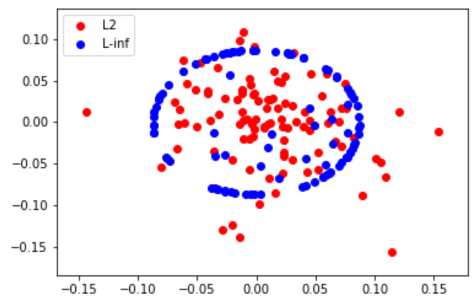

Figure 1: L

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

m = 30
n = 100
A = np.random.rand(m,n) + np.random.rand(m,n) * 1j
b = np.random.rand(m,1) + np.random.rand(m,1) * 1j

x  = cp.Variable((n,1), complex=True)
constraints  = [A@x - b == 0]
# objective = cp.Minimize(cp.norm(x_inf, p = "inf"))
objective = cp.Minimize(cp.norm(x, p = 2))

prob = cp.Problem(objective, constraints)
prob.solve()
print(x.value[:10])

x_inf  = cp.Variable((n,1), complex=True)
constraints  = [A@x_inf - b == 0]
objective = cp.Minimize(cp.norm(x_inf, p = "inf"))
# objective = cp.Minimize(cp.norm(x, p = 2))

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

```
prob.solve()
print(x_inf.value[:10])

X = [x.real for x in x.value]
Y = [x.imag for x in x.value]
X_inf = [x.real for x in x_inf.value]
Y_inf = [x.imag for x in x_inf.value]
plt.scatter(X,Y, color='red',label='L2')
plt.scatter(X_inf,Y_inf, color='blue',label='L-inf')
plt.legend(loc="upper left")
plt.savefig('prob_39.png')
plt.show()
```

4.1

(a)

$$x = \begin{cases} -2.3333 \\ 0.1667 \end{cases} \qquad \lambda = \begin{cases} 1.8994 \\ 3.4684 \\ 0.0931 \end{cases}$$

hence, KKT holds.

- primal feasibility $\begin{cases} x_1^* + 2x_2^* \leq u_1 \\ x_1^* + -4x_2^* \leq u_2 \\ 5x_1^* + 76x_2^* \leq 1 \end{cases}$
- dual feasibility $\lambda_i \geq 0$, $\forall i$
- complimentary slackness $\begin{cases} \lambda_1^*(x_1^* + 2x_2^* - u_1) = 0 \\ \lambda_2^*(x_1^* + -4x_2^* - u_2) = 0 \\ \lambda_3^*(5x_1^* + 76x_2^* - 1) = 0 \end{cases}$
- first order condition: $\begin{cases} 4x_2^* - x_1^* + 2\lambda_1^* - 4\lambda_2^* + 76\lambda_3^* = 0 \\ 2x_1^* - x_2^* - 1 + \lambda_1^* + \lambda_2^* + 5\lambda_2^* = 0 \end{cases}$

hold for the optimal primal and dual variables

```
import cvxpy as cp
import numpy as np
q1 = np.array([[1,-1/2],[-1/2,2]])
q2 = np.array([-1,0])
x = cp.Variable((2,1))
u1 = cp.Constant(-2)
u2 = cp.Constant(-3)

constraints = [x[0] + 2*x[1] <=u1,
               x[0] -4 *x[1] <=u2,
               5*x[0] + 76*x[1] <= 1]

objectives = cp.Minimize(cp.quad_form(x,q1)+q2@x)
prob = cp.Problem(objectives, constraints)
prob.solve()
x.value
```

(b)[1]

$$p^*_{pred} = p^* - \lambda_1^*\delta_1 - \lambda_2^*\delta_2$$

δ_1	δ_2	p^*_{pred}	$\leq$	p^*_{exact}
0	0	8.2222	$\leq$	8.2222
0	-0.1000	8.5691	$\leq$	8.7064
0	0.1000	7.8754	$\leq$	7.9800
-0.1000	0	8.4122	$\leq$	8.5650
-0.1000	-0.1000	8.7590	$\leq$	8.8156
-0.1000	0.1000	8.0653	$\leq$	8.3189
0.1000	0	8.0323	$\leq$	8.2222
0.1000	-0.1000	8.3791	$\leq$	8.7064
0.1000	0.1000	7.6854	$\leq$	7.7515

```
q1 = [1 -1/2; -1/2 2];
q2 = [-1 0];
A = [1 2; 1 -4; 5 76];
b = [-2 -3 1]';

cvx_begin
    variable x(2)
    dual variable lambda   %TODO
    minimize(quad_form(x, q1)+q2*x)
    subject to
    lambda: A*x <= b;  %TODO
cvx_end

% cvx_optval
% lambda
% x

% n = size(A,2);
% cvx_begin
%     variable x(n);
%     dual variable y;
%     minimize( c' * x );
%     subject to
%         y : A * x <= b;
% cvx_end

p_star = cvx_optval
array = [0 -1 1];
delta = 0.1;
pa_table = [];
for i = array
    for j = array
        p_pred = p_star - [lambda(1) lambda(2)]*[i; j]*delta;
        cvx_begin
            variable x(2)
%             dual variable lambda
            minimize(quad_form(x,q1)+q2*x)
            subject to
```

[1] I consulted the online solution on this one

```
            A*x <= b+[i;j;0]*delta
        cvx_end
        p_exact = cvx_optval;
        pa_table = [pa_table; i*delta j*delta p_pred p_exact]
    end
end
```

5.2

$$\min \max \|\frac{p(t_i)}{q(t_i)} - y_i\| \le s \Rightarrow -sq(t_i) \le p(t_i) - y_i q(t_i) \le sq(t_i)$$

is quasiconvex. Hence with bisection method

$$\boxed{\begin{array}{ll} \text{find} & a, b \\ \text{subject to} & \|a_0 + a_1 t_i + a_2 t_i^2 - y_i(1 + b_1 t_i + b_2 t_i^2)\| \le s(1 + b_1 t_i + b_2 t_i^2) \end{array}}$$

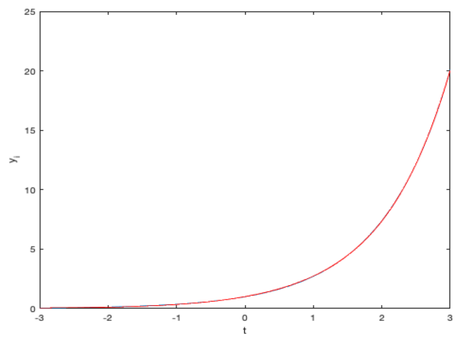

Figure 2: Problem 5.2 Data and the optimal rational function fit

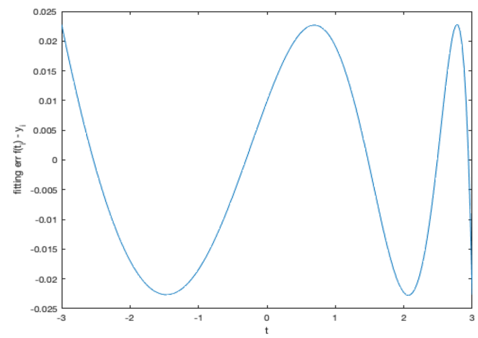

Figure 3: Problem 5.2 The fitting error, i.e. $f(t_i) - y_i$

```
top = 1000;
bottom = 0;
TOL = .001
k = 201
t=(-3:6/(k-1):3)'; % linspace(-3,3,k)
y=exp(t);
VDM=[ones(k,1) t t.^2];
% bisection method
% https://github.com/cvxr/CVX/blob/master/examples/filter_design/fir_lin_phase_lowpass_min_trans.m

while (top - bottom > TOL)
    cur = (top + bottom)/2
    cvx_begin quiet
    variable a(3)
    variable b(2)

    subject to
        abs(VDM*a-y.*(VDM*[1;b])) <= cur*VDM*[1;b]
    cvx_end
    if strcmp(cvx_status,'Solved')
        fprintf(1,'Problem is feasible for %3.4f',cur);
```

```
        a_star = a;
        b_star = b;
        top = cur;
    else
        fprintf(1,'Problem is not feasible for %3.4f',cur);
        bottom = cur
    end
end

y_star = VDM*a_star./(VDM*[1;b_star]);
% y_star
% a_star
% b_star

figure(1);
plot(t,y, t,y_star,'r');
xlabel('t');
ylabel('y_i');

figure(2);
plot(t, y_star-y);
xlabel('t');
ylabel('fitting err f(t_i) - y_i');
```

5.6[2]

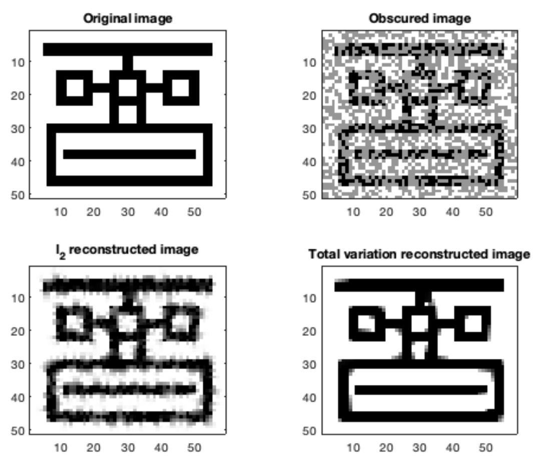

Figure 4: Total variation image interpolation

```
% tv_img_interp.m
% Total variation image interpolation.
% Defines m, n, Uorig, Known.

% Load original image.
Uorig = double(imread('tv_img_interp.png'));
```

[2] I consulted the online solution on this one

```
[m, n] = size(Uorig);
% Create 50% mask of known pixels.
rand('state', 1029);
Known = rand(m,n) > 0.5;
%%%%% Put your solution code here

% Calculate and define Ul2 and Utv.
% Placeholder:
cvx_begin
variable Ul2(m, n);
Ul2(Known) == Uorig(Known);
Ux = Ul2(2:end,2:end) - Ul2(2:end,1:end-1);
Uy = Ul2(2:end,2:end) - Ul2(1:end-1,2:end);
% L2 norm
minimize(norm([Ux(:); Uy(:)], 2));
cvx_end
cvx_begin
variable Utv(m, n);
Utv(Known) == Uorig(Known);
Ux = Utv(2:end,2:end) - Utv(2:end,1:end-1);
Uy = Utv(2:end,2:end) - Utv(1:end-1,2:end);
% L1 norm
minimize(norm([Ux(:); Uy(:)], 1)); %TODO
cvx_end
%%%%


% Graph everything.
figure(1); cla;
colormap gray;
subplot(221);
imagesc(Uorig)
title('Original image');
axis image;
subplot(222);
imagesc(Known.*Uorig + 256-150*Known);
title('Obscured image');
axis image;
subplot(223);
imagesc(Ul2);
title('l_2 reconstructed image');
axis image;
subplot(224);
imagesc(Utv);
title('Total variation reconstructed image');
axis image;
```

5.13

(a)

Since convex, by adding more info, say y us bounded from below (additional constraints):

$$\boxed{\begin{array}{ll} \text{minimize} & \sum_{i=1}^{M}(y_i - c^T x_i)^2 \\ \text{subject to} & c^T x_i \geq D, \\ & \text{for } i = M+1, \ldots, K \end{array}}$$

(b)

$$\frac{\|c_{true-\hat{c}}\|_2}{\|c_{true}\|_2} = 0.50, \ \frac{\|c_{true-\hat{c_{ls}}}\|_2}{\|c_{true}\|_2} = 1.33$$

```
## Data gen
n = 20 # number of variables
M = 25 # number of censored observations
K = 100 # total number of observations

np.random.seed(n*M*K)
X = np.random.randn(K*n).reshape(K, n)
c_true = np.random.rand(n)

# generating the y variable
y = X.dot(c_true) + .3*np.sqrt(n)*np.random.randn(K)

# ordering them based on y
order = np.argsort(y)
y_ordered = y[order]
X_ordered = X[order,:]

#finding boundary
D = (y_ordered[M-1] + y_ordered[M])/2.

# applying censoring
y_censored = np.concatenate((y_ordered[:M], np.ones(K-M)*D))

X_uncensored = X_ordered[:M, :]

## cvxpy
# with constraints
c = cp.Variable(shape=n)
objective = cp.Minimize(cp.sum_squares(X_uncensored*c - y_ordered[:M]))
constraints = [ X_ordered[M:,:]*c >= D]
prob = cp.Problem(objective, constraints)
result = prob.solve()
c_cvx = np.array(c.value).flatten()

# without constraints
c = cp.Variable(shape=n)
objective = cp.Minimize(cp.sum_squares(X_uncensored*c - y_ordered[:M]))
# constraints = [ X_ordered[M:,:]*c >= D]
prob = cp.Problem(objective)
result = prob.solve()
```

```
c_ols_uncensored = np.array(c.value).flatten()

print("norm(c_true - c_cvx)/norm(c_true):\
 {:.2f}".format(np.linalg.norm((c_true - c_cvx))/np.linalg.norm((c_true))))
print("norm(c_true - c_ols_uncensored)/norm(c_true): \
{:.2f}".format(np.linalg.norm((c_true - c_ols_uncensored))/np.linalg.norm((c_true)) ))
```

5.15

(a)

From HW matlab hint, we optimize the following convex problem

$$\boxed{\begin{array}{ll} \text{minimize} & \frac{1}{N}\sum_{i=1}^{N}(d_i - (x_i - y_i)^T P (x_i - y_i))^2 \\ \text{subject to} & P \succeq 0 \end{array}}$$

(b)

T he optimal mean squared distance error: $+1.24901e - 10$

```
%% data for learning a quadratic metric
% provides X, Y, d, X_test, Y_test, d_test
rand('seed',0);
randn('seed',0); %TODO
n = 5; % dimension
N = 100; % number of distance samples
N_test = 10;
X = randn(n,N);
Y = randn(n,N);
X_test = randn(n,N_test);
Y_test = randn(n,N_test);

P =randn(n,n);
P = P*P'+eye(n);
sqrtP = sqrtm(P);

d = norms(sqrtP*(X-Y)); % exact distances
d = pos(d+randn(1,N)); % add noise and make nonnegative
d_test = norms(sqrtP*(X_test-Y_test));
d_test = pos(d_test+randn(1,N_test));


[d_test, sort_ind] = sort(d_test);
X_test = X_test(:,sort_ind);
Y_test = Y_test(:,sort_ind);
diff = X_test-Y_test

clear P sqrtP;
% solution
cvx_begin
    variable P(n,n)
    minimize((1/N_test)*pow_pos(sum(d_test' - sqrt(diag(diff'*P*diff))),2)),
    subject to
    P>0
cvx_end
```