HW15

June 14, 2020

1 HW15

1.1 Alireza Darvishi 96109674

1.1.1 5.12

[1]:
```
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt
from ls_perm_meas_data import *
y=np.reshape(y,m)

x = cp.Variable(n)
first_objective = cp.Minimize(cp.norm1(A @ x - y))
first_problem = cp.Problem(first_objective,[])
print("first estimate error: ",first_problem.solve())
rsq = (A@x.value-y)**2
plt.hist(rsq**0.5);
sorted_idx = np.argsort(-rsq)
k_largest_idx = sorted_idx[:k]
keeping_idx = sorted_idx[k:]

second_objective = cp.Minimize(cp.norm2(A[keeping_idx,:] @ x - y[keeping_idx]))
second_problem = cp.Problem (second_objective,[])
second_problem.solve()
y_estimate = A[k_largest_idx,:]@x.value
order_of_y_estimete = np.argsort(y_estimate)
order_of_y_k_largest = np.argsort(y[k_largest_idx])

newA = np.zeros(A.shape)
newA[k:,:]=A[keeping_idx,:]
newA[:k,:]=A[k_largest_idx,:][order_of_y_estimete]

newy = np.zeros(y.shape)
newy[k:]=y[keeping_idx]
newy[:k]=y[k_largest_idx][order_of_y_k_largest]
# newy[order_of_y_k_largest]=y[order_of_y_estimete]
```

```
# A[order_of_y_k_largest,:]=A[order_of_y_estimete,:]

final_objective = cp.Minimize(cp.norm1(newA @ x - newy))
final_problem = cp.Problem(final_objective,[])
print("final estimate error: ",final_problem.solve())
rsq = (newA@x.value-newy)**2
plt.figure();
plt.hist(rsq**0.5);
```

```
first estimate error:  2041.6648818003052
final estimate error:  75.81172584135858
```

1.1.2 13.3

[2]:
```
import cvxpy as cp
import numpy as np

G = np.array([[0.3, -0.1, -0.9], [-0.6, 0.3, -0.3], [-0.3, 0.6, 0.2]])
v = np.random.normal(0, 1, 3)
I = np.eye(3)
Z = cp.Variable((3,3))
constrints = [cp.norm(Z,axis=1)<=1]
objective = cp.Minimize(cp.max(cp.norm(I+G@Z,axis=1)))
prob = cp.Problem(objective, constrints)
prob.solve()
print(np.linalg.inv(I+Z.value@G)@Z.value)
print("optimal value is:",np.round(prob.value**2,4))
```

```
[[-1.30309479  3.68758732  3.45616496]
 [ 0.21119619 -0.89652267 -3.36088163]
 [15.52709908  7.32358762 -9.1525585 ]]
optimal value is: 0.1647
```

1.1.3 13.4

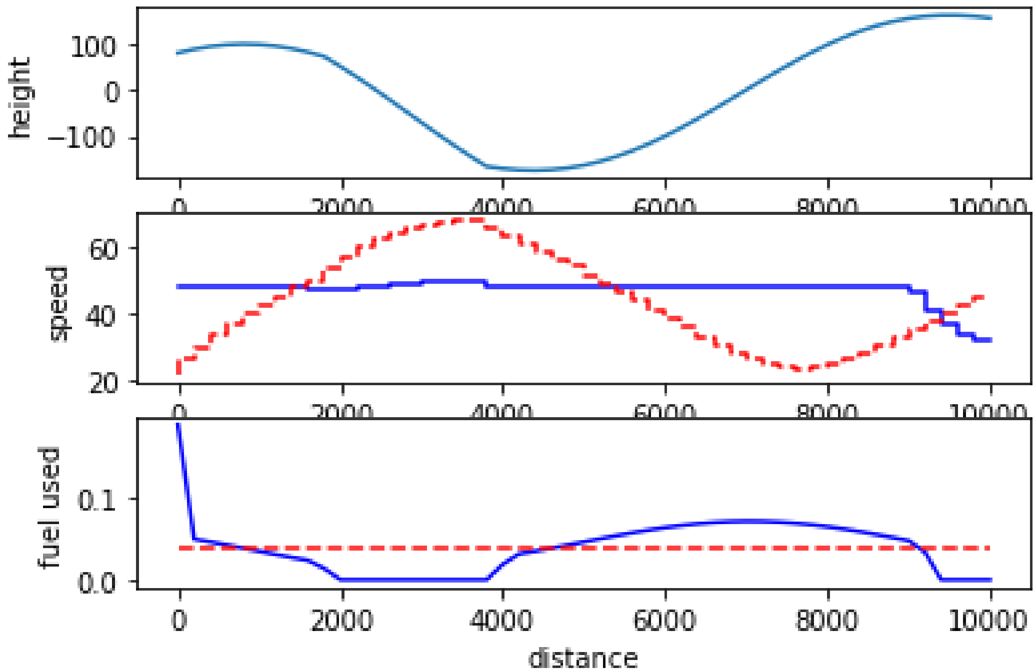
[3]:
```
from min_time_speed_data import *
import matplotlib.pyplot as plt
import cvxpy as cp
import numpy as np

f = cp.Variable(N+1)
f_ob = cp.Variable()
z = cp.Variable(N+1)
T = d*sum(cp.inv_pos(cp.sqrt(z[:-1])))

constraints = [f >= 0]
```

```
constraints += [sum(f) + (T*P)/eta <= F]
constraints += [eta*f[0] == 0.5*m*z[0]]
for i in range(N):
    constraints += [0.5*m*z[i+1] + m*g*h[i+1] == 0.5*m*z[i] + m*g*h[i] +
 ↪eta*f[i+1] - d*C_D*z[i]]
objective = cp.Minimize(T)
prob = cp.Problem(objective, constraints)
print(prob.solve())
s_opt = np.sqrt(z.value)
f_opt = f.value
prob2 = cp.Problem(objective, constraints + [cp.max(f)*(N+1) <= sum(f)])
print(prob2.solve())
sc = np.sqrt(z.value)
fc = f.value
plt.subplot(3,1,1)
plt.plot(np.array(range(0,N+1))*d,h)
plt.ylabel('height')
plt.subplot(3,1,2)
plt.step(np.array(range(0,N+1))*d,s_opt,'b')
plt.step(np.array(range(0,N+1))*d,sc,'--r')
plt.ylabel('speed')
plt.subplot(3,1,3)
plt.plot(np.array(range(0,N+1))*d, f_opt,'b');
plt.plot(np.array(range(0,N+1))*d, fc,'--r')
plt.xlabel('distance')
plt.ylabel('fuel used')
plt.show()
```

```
213.2619904455464
258.47941979011
```



1.1.4 14.8

[4]:
```
import cvxpy as cp
import numpy as np
p = np.array([1/3, 1/6, 1/3, 1/6])
R = np.array([[2, 2, 0.5, 0.5], [1.3, 0.5, 1.3, 0.5], [1, 1, 1, 1]])
x = cp.Variable(3)
y = cp.Variable(4)
constraints1 = [y == R.T @ x, sum(x) == 1]
objective = cp.Minimize(-p @ cp.log(y))
prob1 = cp.Problem(objective, constraints1)
prob1.solve()
print("best strategy is:",x.value,"and optimal value is:",-prob1.value)
constraints2 = constraints1 + [x[0] == 1, x[1] == 0, x[2] == 0]
prob2 = cp.Problem(objective, constraints2)
print("for strategy (1,0,0) : ", -prob2.solve())
constraints3 = constraints1 + [x[0] == 0, x[1] == 1, x[2] == 0]
prob3 = cp.Problem(objective, constraints3)
print("for strategy (0,1,0) : ", -prob3.solve())
constraints4 = constraints1 + [x[0] == 0.5, x[1] == 0.5, x[2] == 0]
prob4 = cp.Problem(objective, constraints4)
print("for strategy (0.5,0.5,0) : ", -prob4.solve())
constraints5 = constraints1 + [x[0] == 0, x[1] == 0, x[2] == 1]
```

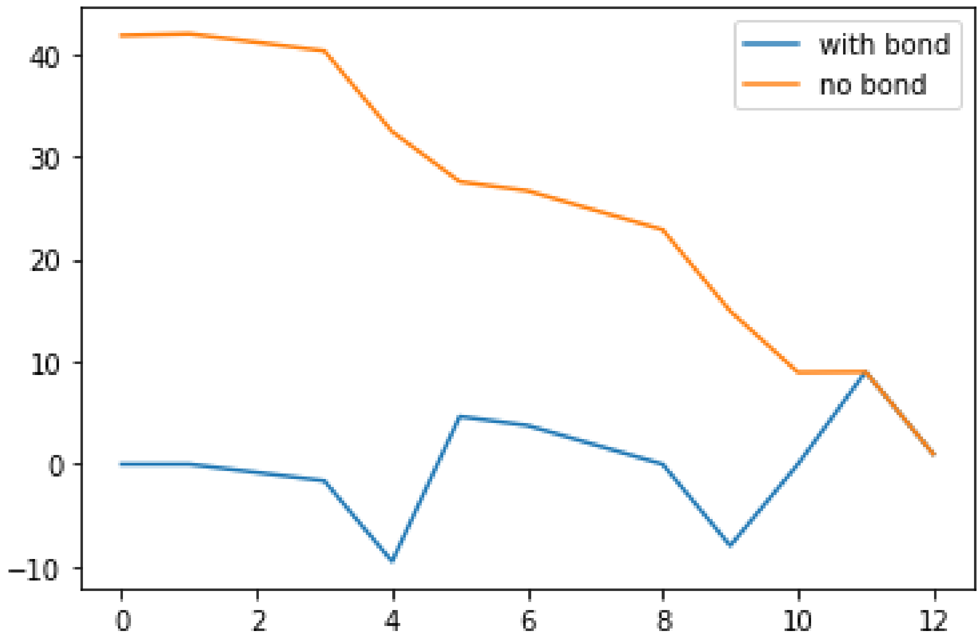
```
prob5 = cp.Problem(objective, constraints5)
print("for strategy (0,0,1) : ", -prob5.solve())
```

```
best strategy is: [0.49725105 0.19936803 0.30338093] and optimal value is:
0.06227157878135451
for strategy (1,0,0) :  -4.923437657700523e-10
for strategy (0,1,0) :  -0.056139550858751545
for strategy (0.5,0.5,0) :  0.05347098525410708
for strategy (0,0,1) :  -1.8126598794172385e-10
```

1.1.5 14.10

[5]:
```
import cvxpy as cp
import numpy as np
from opt_funding_data import *
x = cp.Variable(n)
B0 = cp.Variable()
B = cp.Variable(T)
I = A @ x
Sp = cp.Variable(T-1)
Sn = cp.Variable(T-1)
objective = cp.Minimize( P @ x + B0 )
constraints = [ x>=0 , B0>=0 , B[0] == (1+rp)*B0 , B[T-1]+I[T-1]-E[T-1]==0 ]
constraints += [Sp>=0,Sn>=0,B[1:]==(1+rp)*Sp -(1+rn)*Sn , B[:-1]+I[:-1]-E[:
↪-1]==Sp-Sn]
problem1=cp.Problem(objective,constraints)
problem1.solve()
print("optimal investment is:",np.round(problem1.value,4))
print("optimal x is:",np.round(x.value,4))
plt.plot(np.append(B0.value,B.value),label="with bond");
problem2 = cp.Problem(objective,constraints+[x==0])
problem2.solve()
print("optimal investment without bond is:",np.round(problem2.solve(),4))
plt.plot(np.append(B0.value,B.value),label="no bond");
plt.legend();
```

```
optimal investment is: 40.7495
optimal x is: [ 0.      18.9326  0.       0.      13.8493  8.9228]
optimal investment without bond is: 41.7902
```


with bond
no bond
40
30
20
10
0
−10
0
2
4
6
8
10
12

علیرضا درویشی - 96109674 - بهینه سازی محدب - تمرین 15

5.12

ابتدا مسئله را بدون جایگشت و با فرض $P=I$ و با نرم ℓ_1 حل می‌کنیم چون نسبت به داده‌های پرت، کم‌تر حساس است سپس می‌توان چندکار انجام داد. یکی اینکه پیشبینی‌هایی که با داده‌های اصلی متفاوت هستند را به عنوان آن‌هایی که جابجا شده‌اند در نظر بگیریم و جابجا کنیم. یکی اینکه آن‌هایی که باقی‌مانده زیادی دارند را حذف کنیم و مدل را فیت کنیم و پیشبینی دوباره انجام دهیم.

می‌توانیم این کارها را به صورت بازگشتی انجام دهیم تا جایی که دیگر داده‌ی پرت شناسایی نکنیم.

13.3

$$E u_i^2 \leq 1 \Rightarrow \operatorname{diag}(uu^T) \leq 1$$

$$\Rightarrow \operatorname{diag}\left[\sigma^2 F(I-GF)^{-1}(I-GF)^{-T}F^T\right] \leq 1$$

$$E y_i^2 = (yy^T)_{ii} = \left(\sigma^2 (I-GF)^{-1}(I-GF)^{-T}\right)_{ii}$$

$$\Rightarrow \min \quad \sigma^2 \max_{i=1,\dots,n} \quad (I-GF)^{-1}(I-GF)^{-T}$$

$$\text{st:} \quad \operatorname{diag}\left(F(I-GF)^{-1}(I-GF)^{-T}\right) \leq \frac{1}{\sigma^2}$$

اما این مسئله Convex نیست. برای حل این مشکل تغییر متغیر زیر را می‌دهیم.

$$X = F(I-GF)^{-1} \Rightarrow X(I-GF) = F \Rightarrow X = F + XGF \Rightarrow F = (I+XG)^{-1}X$$

از طرفی

$$I = (I-GF)(I-GF)^{-1} = (I-GF)^{-1} - GF(I-GF)^{-1}$$

$$\Rightarrow (I-GF)^{-1} = I + GX$$

Scanned with CamScanner

پس :

$$\min \quad \sigma^2 \max_{i=1,\dots,n} \left((I+Gx)(I+Gx)^T \right)_{ii}$$

$$\text{st} \quad \mathrm{diag}(\lambda X^T) \leq \frac{1}{\sigma^2}$$

13.4

$$T = \sum_{i=1}^{N} \frac{d}{s_i}$$

$$\Rightarrow \min \quad \sum_{i=1}^{N} \frac{d}{s_i}$$

$$\text{st:} \quad \frac{1}{2} m s_{i+1}^2 + mgh_{i+1} = \frac{1}{2} m s_i^2 + mgh_i + \eta f_i - dC_D s_i^2$$

$$\sum_{i=1}^{N} f_i + \frac{PT}{\eta} \leq F$$

$$f_i \geq 0$$

$$\eta f_0 = \frac{1}{2} m s_1^2$$

با وجود اینکه فرا مسئله محدب است اما چون در تساوی ، مسئله خطی نیست ، نمی‌توان از CVXPY برای حل مسئله استفاده کرد. با تغییر متغیر $Z_i = \sqrt{S_i}$ ، این مشکل برطرف می‌شود:

$$\min \quad \sum_{i=1}^{N} d/\sqrt{Z_i}$$

$$\text{st} \quad \frac{1}{2} m Z_{i+1} + mgh_{i+1} = \frac{1}{2} m Z_i + mgh_i + \eta f_i - dC_D Z_i$$

$$\sum f_i + \frac{P}{\eta} \sum \frac{d}{\sqrt{Z_i}} \leq F$$

$$f_i \geq 0$$

$$\eta f_0 = \frac{1}{2} m Z_1$$

14.8

a) اگر $x + tv$ انتخاب کنیم، چون $1^T v = 0$ پس در شرط صدق می کند و تابع هدف را بی کران زیاد می کند برای $t > 0$.

b)
$$\min \quad -\sum P_j \log y_j$$
$$\text{st} \quad y = R^T x$$
$$1^T x = 1$$

$$\Rightarrow L(x, y, v, \lambda) = -\sum P_j \log y_j + v^T(y - R^T x) + \lambda(1^T x - 1)$$

که اگر $Rv = \lambda 1$ باشد، کراندار از پایین است بعد از کمینه کردن روی x.

حال به ازای $y_j = \frac{P_j}{v_j}$ ، L کمینه می شود

$$\frac{\partial L}{\partial y_j} = -\frac{P_j}{y_j} + v_j = 0 \Rightarrow y_j = \frac{P_j}{v_j}, \quad v_j > 0$$

پس:

$$g(v, \lambda) = -\sum P_j \log\left(\frac{P_j}{v_j}\right) + 1 - \lambda$$

$$\text{st:} \quad Rv = 1\lambda$$

14.10 برای اینکه محدب بودن مسئله خراب نشود، بتوانیم از cvxpy استفاده کنیم، از متغیرهای کمکی S^+ و S^- کمک می گیریم:

$$B_t - E_t + I_t = S_t^+ - S_t^-, \quad S^+ \geq 0, \quad S^- \geq 0$$

$$, B_{t+1} = (1 + r_+) S_t^+ - (1 + r_-) S_t^-$$

پس:

$$\min \quad B_0 + \sum P_i x_i$$
$$\text{st} \quad Ax = I$$
$$x \geq 0, \quad B_0 \geq 0, \quad S^+ \geq 0, \quad S^- \geq 0$$
$$B_1 = (1 + r_+) B_0$$
$$B_T - E_T + I_T = 0$$
$$B_{t+1} = (1 + r_+) S_t^+ - (1 + r_-) S_t^-, \quad t = 1, \dots, T-1$$
$$B_t - E_t + I_t = S_t^+ - S_t^-$$