sdp_for_matrix_recovery

September 5, 2023

[102]:
```
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt

## Generate a random low-rank matrix:
np.random.seed(10)
m = 100
n = 50
rank = 10  ## Rank of the low-rank matrix
X_true = np.random.randn(m, rank) @ np.random.randn(rank, n)

# noise = 0.1 * np.random.randn(m, n)
# X_noisy = X_true + noise

U, S, Vt = np.linalg.svd(X_true)

np.set_printoptions(precision=3, suppress=True)
print("Singular Values (S): ", S)
```

```
Singular Values (S):  [96.091 89.433 81.229 74.632 67.663 59.273 48.967 48.459
42.805 30.96
  0.      0.      0.      0.      0.      0.      0.      0.      0.      0.
  0.      0.      0.      0.      0.      0.      0.      0.      0.      0.
  0.      0.      0.      0.      0.      0.      0.      0.      0.      0.
  0.      0.      0.      0.      0.      0.      0.      0.      0.      0.    ]
```

[103]:
```
## Nuclear norm:
X_nuc_norm = S.sum()
print("Nuclear norm of S = ", X_nuc_norm)
```

```
Nuclear norm of S =  639.511169667763
```

[104]:
```
## SDP to compute the nuclear norm:
W1 = cp.Variable((m,m), symmetric=True)
W2 = cp.Variable((n,n), symmetric=True)

C_mat = cp.bmat([[W1,        X_true],
                 [X_true.T, W2    ]])
```

```
obj = cp.Minimize((cp.trace(W1) + cp.trace(W2))/2)
constraints = [C_mat >> 0]

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

print("The estimated nuclear norm = ", prob.value)
```

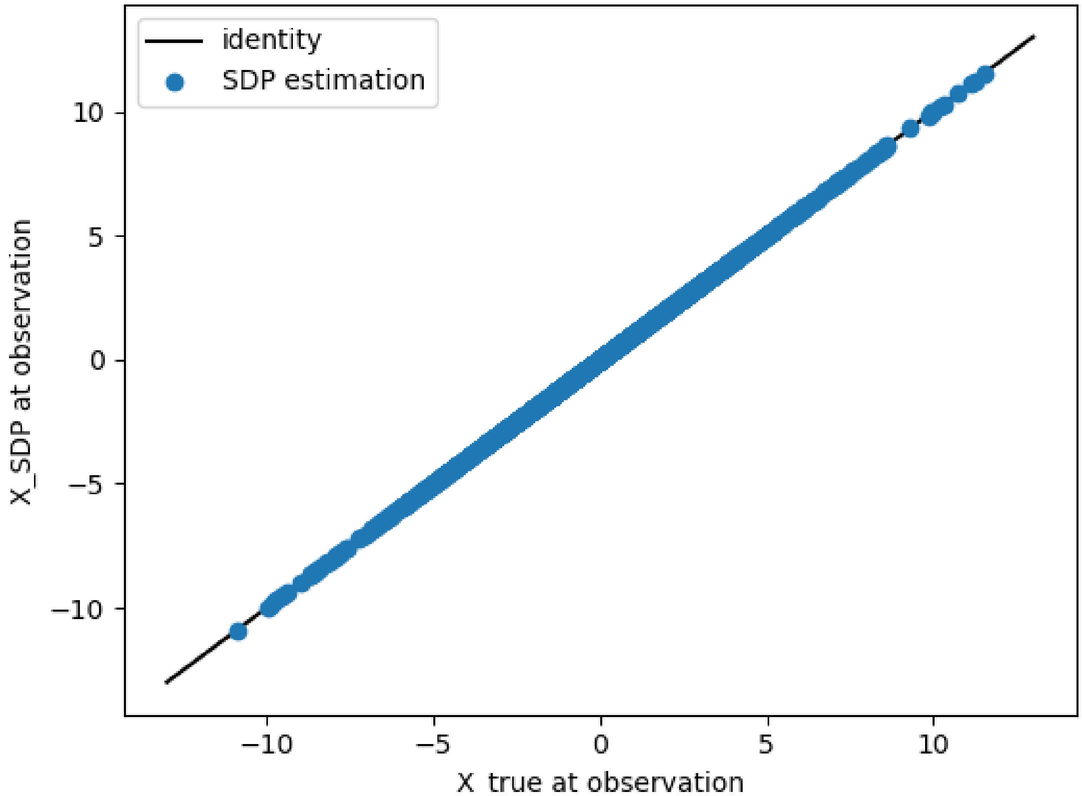
```
The estimated nuclear norm =  639.5118881660484
```

[105]:
```
## Define the observation:
## Some cases have very good recovery, e.g., seed = 40. Why?
np.random.seed(13)
A_obs = np.random.randint(2, size=(m, n))
Y_obs = X_true * A_obs

## SDP to solve the matrix recovery problem:
W1 = cp.Variable((m,m), symmetric=True)
W2 = cp.Variable((n,n), symmetric=True)
X = cp.Variable((m,n))

C_mat = cp.bmat([[W1,  X],
                 [X.T, W2]])

obj = cp.Minimize((cp.trace(W1) + cp.trace(W2))/2)
constraints = [C_mat >> 0]
constraints.append( X[A_obs == 1] == Y_obs[A_obs == 1] )

prob2 = cp.Problem(obj, constraints)
prob2.solve()

print("The optimal objective = ", prob.value)
```

```
The optimal objective =  639.5118881660484
```

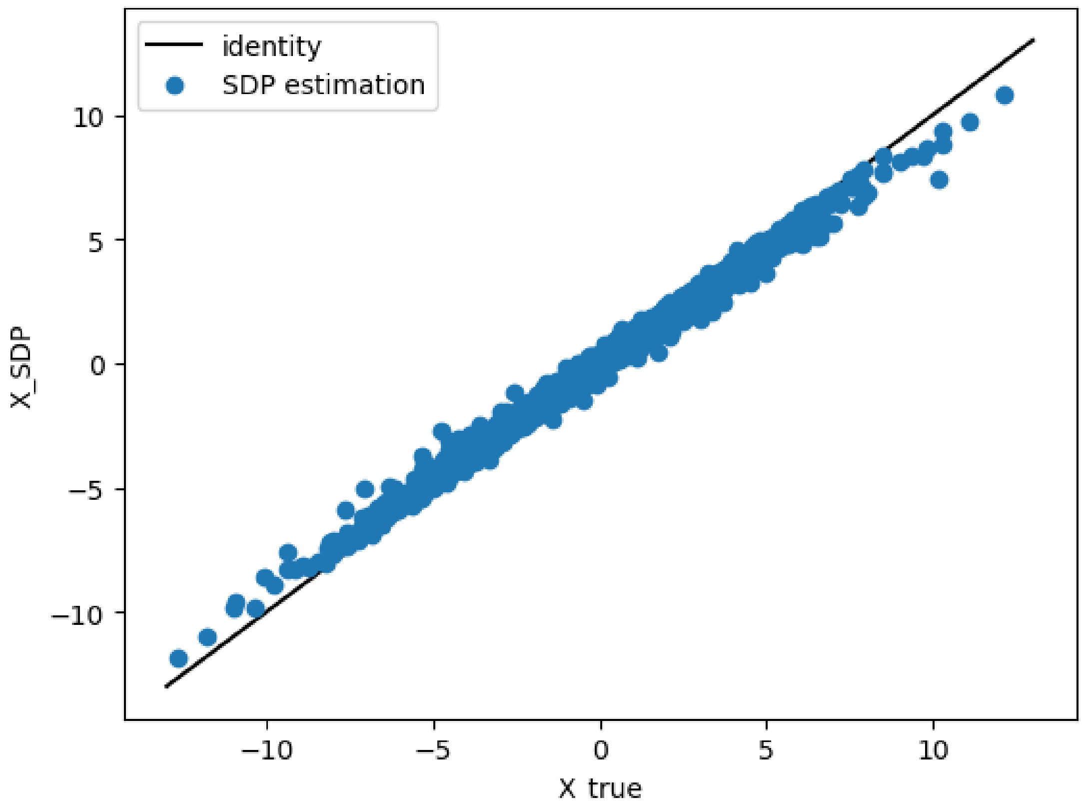
[106]:
```
## Check constraint:
xx = np.linspace(-13, 13, 100)
plt.plot(xx, xx, "k", label="identity")
plt.plot(X_true[A_obs == 1], X.value[A_obs == 1], "o", label="SDP estimation")
plt.xlabel("X_true at observation")
plt.ylabel("X_SDP at observation")
plt.legend()
```

[106]:
```
<matplotlib.legend.Legend at 0x12521e490>
```



[107]:
```
## Compare SDP solution to ground truth:
xx = np.linspace(-13, 13, 100)
plt.plot(xx, xx, "k", label="identity")
plt.plot(X_true[A_obs == 0], X.value[A_obs == 0], "o", label="SDP estimation")
plt.xlabel("X_true")
plt.ylabel("X_SDP")
plt.legend()
```

[107]: <matplotlib.legend.Legend at 0x14e1f5880>


identity
SDP estimation
X_SDP
X_true
10
5
0
−5
−10
−10
−5
0
5
10