newsvendor

April 19, 2022

1 Newsvendor Optimization

Complete Enumeration, Inverting the CDF, Bisection Search, Derivative-free optimization with `scipy.optimize`

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats
```

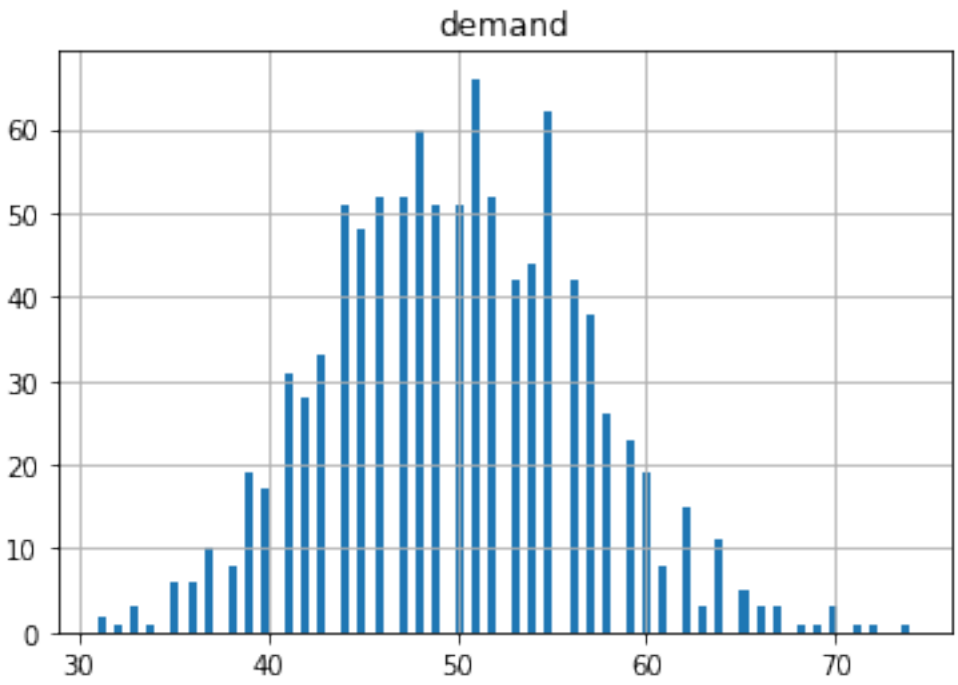
2 Generate Demand Data

[36]:
```
# This is for the first example in class, of holiday lights
#demand = pd.DataFrame(np.exp(2*np.random.normal(size=10000)),␣
 ↪columns=['demand'])

# A discrete demand distribution
mu = 50
demand = pd.DataFrame(np.random.poisson(mu,size=1000), columns=['demand'])
```

[37]:
```
demand.hist(bins=100)
```

[37]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f7f69f2e910>]],
      dtype=object)
```



3 Complete Enumeration

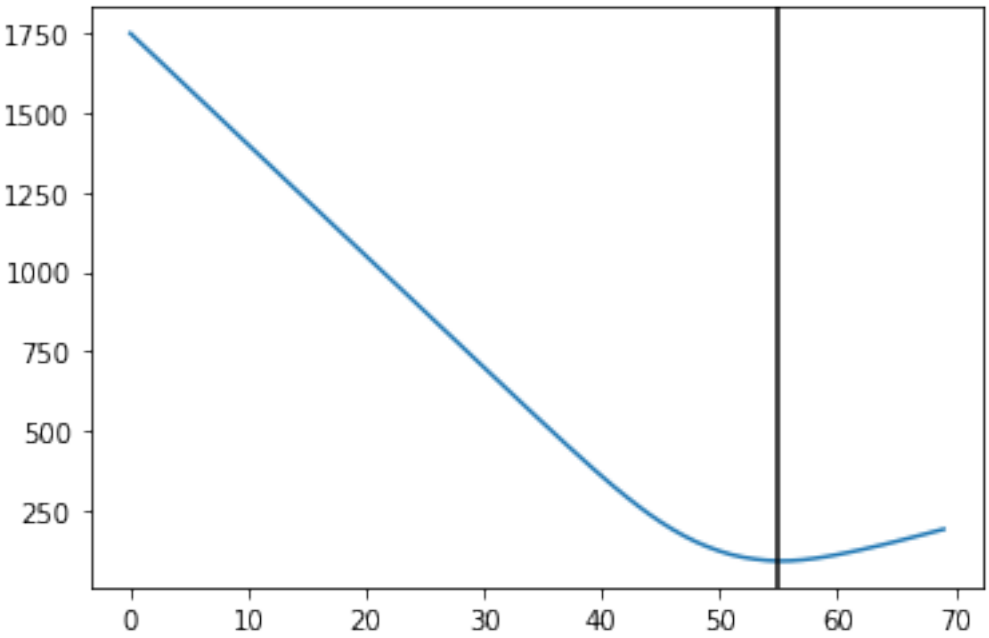
[38]:
```
co = 10
cu = 35
maxq = 70

# Expected cost of an inventory level q
def cost(q):
    cost_overage = co*(q-demand).clip(lower=0).mean()[0]
    cost_underage = cu*(demand-q).clip(lower=0).mean()[0]
    return cost_overage + cost_underage
```

[40]:
```
# Calculate where the cost is minimized via complete enumeration
q_list = range(maxq)
cost_list = [cost(q) for q in q_list]
qstar = q_list[np.where(cost_list == np.min(cost_list))[0][0]]

plt.plot(q_list,cost_list)
plt.axvline(qstar,color='black')
#plt.xlim(0,10)
#plt.ylim(200,300)
qstar
```

[40]: 55



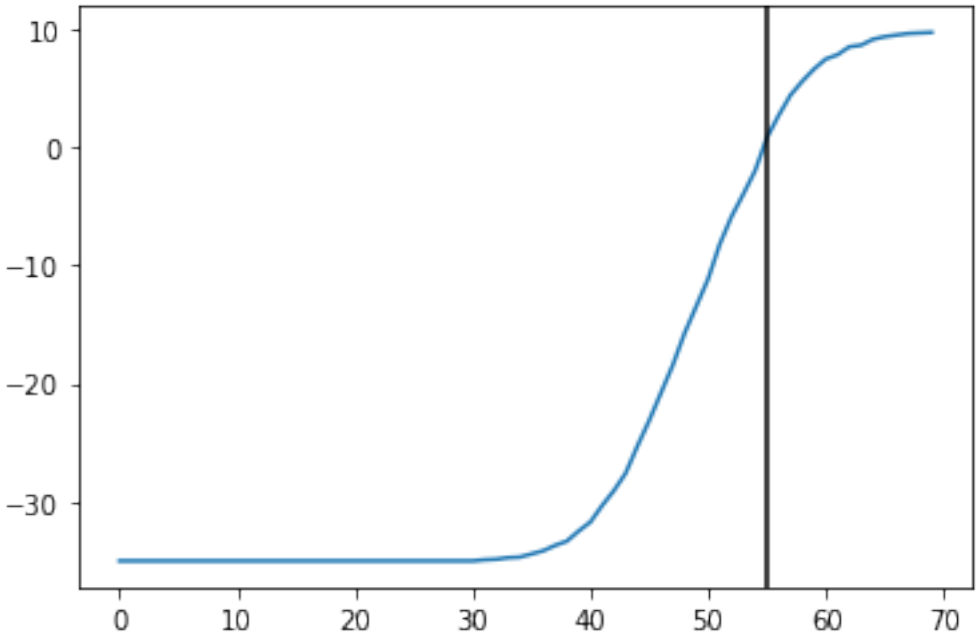
4 Derivative

[41]:
```
def cdf(q):
    return (demand <= q).mean()[0] # P(D <= Q)

def derivative(q):
    # This is the derivative we calculated in the slides
    return -cu + (cu + co) * cdf(q)
```

[42]:
```
# The optimal point is where the derivative is 0
q_list = range(maxq)
grad_list = [derivative(q) for q in q_list]
plt.plot(q_list,grad_list)
#plt.axhline(0,color='orange')
plt.axvline(qstar,color='black')
#plt.xlim(0,10)
```

[42]: <matplotlib.lines.Line2D at 0x7f7f786fd850>



5 Approximate Q* by inverting the cdf

[43]:
```
a = cu / (cu + co) # Newsvendor quantile
a
```

[43]: 0.7777777777777778

[21]:
```
# First example, where the log of demand is normal with standard deviation 2 and␣
 ↪mean 0.
# To handle a general non-zero mean of m and standard deviation s, use stats.
 ↪lognorm(s,scale=np.exp(m))
stats.lognorm(s=2).ppf(a)
```

[21]: 4.615496045093771

[27]:
```
cost(4)
```

[27]: 247.53531520516498

[28]:
```
cost(5)
```

[28]: 247.23339658094702

[44]:
```
# Example with Poisson demand
stats.poisson(mu).ppf(a) # ppf is the inverse cdf
# This gives a slightly different answer from complete enumeration when D has a␣
 ↪discrete distribution.
# There isn't a q such that cdf(q) = a.
# As a result, our answer is off by 1 from complete enumeration. In practice,␣
 ↪this usually doesn't matter,
# unless we are only ordering 1 or 2 items.
```

[44]:
```
55.0
```

6 Bisection Search

[45]:
```
L = 0 # lower bound on qstar
U = 1 # this will be an upper bound on qstar
a = cu / (cu + co) # Newsvendor quantile


while cdf(U) < a: # while the derivative of the cost is negative
    U *= 2
    n = n+1

while U > L + 1:
    q = np.round((U + L)/2)
    if cdf(q) >= a: # the derivative of the cost is positive
        U = q
    else: # the derivative of the cost is negative
        L = q

qstar = U
qstar
```

[45]:
```
55.0
```

[34]:
```
# Version for continuous demand

L = 0 # lower bound on qstar
U = 1 # this will be an upper bound on qstar
a = cu / (cu + co) # Newsvendor quantile

# Find U large enough that cdf(U) >= a
while cdf(U) < a: # while the derivative of the cost is negative
    U *= 2

#while U > L + 1:
while U-L > .0001:
```

```
    #q = np.round((U + L)/2)
    q = (U+L)/2
    if cdf(q) >= a: # the derivative of the cost is positive
        U = q
    else: # the derivative of the cost is negative
        L = q

qstar = U
qstar
```

[34]: 4.688232421875

7 Finding qstar using scipy.optimize (Derivative-free optimization)

[46]:
```
import scipy.optimize
scipy.optimize.minimize(fun=cost, x0 = np.random.rand()*100)
```

[46]:
```
      fun: 90.7200000077868
 hess_inv: array([[0.03042296]])
      jac: array([0.07992744])
  message: 'Desired error not necessarily achieved due to precision loss.'
     nfev: 369
      nit: 4
     njev: 119
   status: 2
  success: False
        x: array([55.])
```

8 Stochastic Gradient Descent

[80]:
```
def stochastic_gradient(d,q):
    if q > d: # we ordered too many and had overage
        return co
    elif q < d: # we orded too few and had underage
        return -cu
    else: # q == d, which should happen with probability 0
        print('Q is equal to D!  Derivative undefined')
        return (co + cu)/2
```

[106]:
```
q = np.random.rand() # Choose a random starting point
hist = pd.DataFrame(columns=['q','g', 'step'])
for n in range(10000):
    d = demand.sample().values[0][0] # Choose a random demand from the data
    g = stochastic_gradient(d,q)
```

```
    step = (n+1)**-.5 # Any sequence that decreases slowly
    hist = hist.append({ 'q' :q, 'g' : g, 'step' : step} ,ignore_index=True)
    q = q - step * g
```

[107]:
```
hist
```

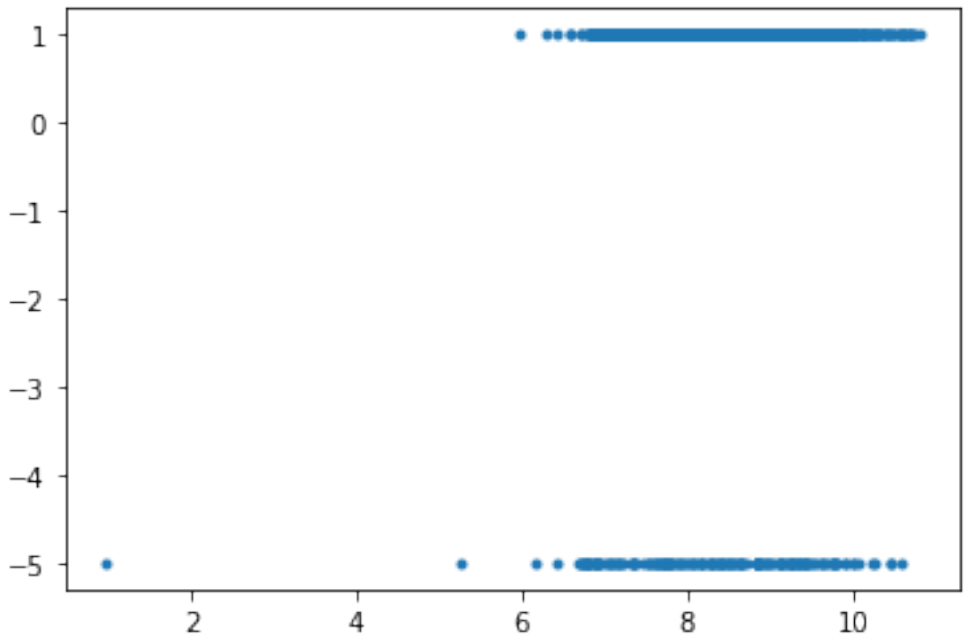
[107]:

	q	g	step
0	0.561783	-5.0	1.000000
1	5.561783	1.0	0.707107
2	4.854677	1.0	0.577350
3	4.277326	1.0	0.500000
4	3.777326	1.0	0.447214
…	…	…	…
9995	7.646220	1.0	0.010002
9996	7.636218	1.0	0.010002
9997	7.626217	1.0	0.010001
9998	7.616216	1.0	0.010001
9999	7.606215	1.0	0.010000

[10000 rows x 3 columns]

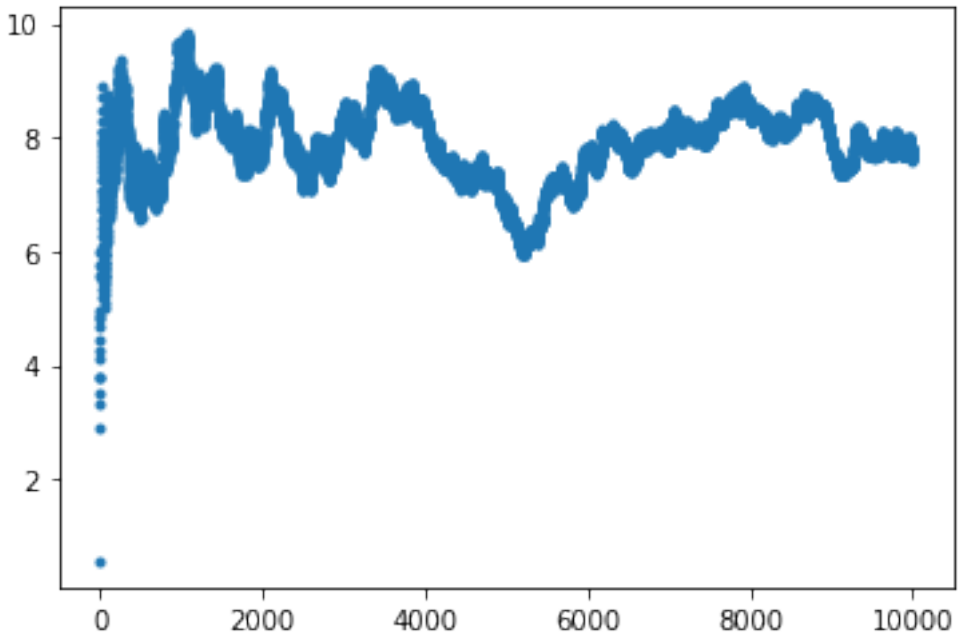
[104]:
```
plt.plot(hist['q'],hist['g'],'.')
```

[104]: [<matplotlib.lines.Line2D at 0x7fa12896be50>]



[108]:
```
plt.plot(hist['q'],'.')
```

[108]: [<matplotlib.lines.Line2D at 0x7fa0f8a48910>]



[ ]: