QR

May 4, 2022

[25]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[26]:
```
# inventory is the amount of inventory on hand.
# days_left is the number of days left until our order arrives.
# If the order hasn't been placed, this is set to -1.
# cost is the total cost paid
state = {'inventory' : 100, 'days_left' : -1, 'cost' : 0}

params = {'demand_mean' : 5, 'demand_std' : 3,
          'lead_time' : 10, # tau in the slides
          'stockout_penalty' : 0.1, # p in the slides
          'holding_cost' : 0.01, # h in the slides
          'K' : 1, 'c' : 0.1 # order cost parameters
         }
```

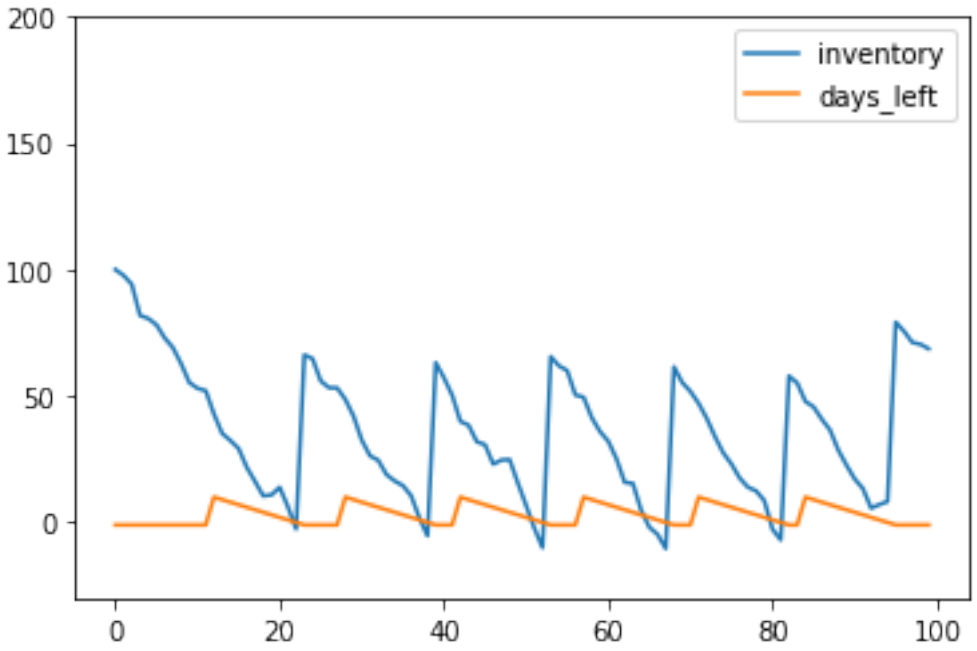
[31]:
```
def step(state, Q, R, params):
    # Check whether an order has arrived
    if state['days_left'] == 0:
        # If it has, unload the truck and add the contents to our inventory
        state['inventory'] += Q
        # Mark the state to say that a new order hasn't been placed
        state['days_left'] = -1
    # Otherwise, if an order is in transit, it gets one day closer
    elif state['days_left']>0:
        state['days_left'] -= 1

    # Simulate the demand on this day, and fill it from our inventory.
    # This may make our inventory negative.
    # Also calculate stockout costs
    demand = np.random.normal(params['demand_mean'], params['demand_std'])

    # Calculate stockout costs
    onhand = np.max([0,state['inventory']])
    backordered = np.max([0,demand - onhand])
    state['cost']+= params['stockout_penalty'] * backordered
```

```
    # Fill demand from inventory, backordering if needed
    state['inventory'] -= demand

    # Pay holding cost
    state['cost'] += params['holding_cost']*np.max([state['inventory'],0])

    # If our inventory is below the reorder level, and an order is
    # not on its way, then place an order
    if state['inventory'] < R and state['days_left'] == -1:
        state['days_left'] = params['lead_time']
        state['cost'] += params['K'] + params['c']*Q

    return state
```

Note that a dictionary is a "mutable object" and if you change its contents inside a function, then the dictionary will also be changed if we refer to it outside of the function. So this means that instead of returning `state` from `step(state, params)` and setting `state = step(state,params)`, we could have simply done `step(state,params)`

[28]:
```
state = {'inventory' : 100, 'days_left' : -1, 'cost' : 0}
Q = 75
R = 50
inventory = []
days_left = []
cost = []

T = 100
for t in range(T):
    inventory.append(state['inventory'])
    days_left.append(state['days_left'])
    cost.append(state['cost'])
    state = step(state,Q,R,params)

plt.plot(range(T),inventory)
plt.plot(range(T),days_left)
plt.legend(['inventory','days_left','cost'])
plt.ylim(-30,200)
```

[28]:
```
(-30, 200)
```



1 Lecture 19 Demo, Question 1

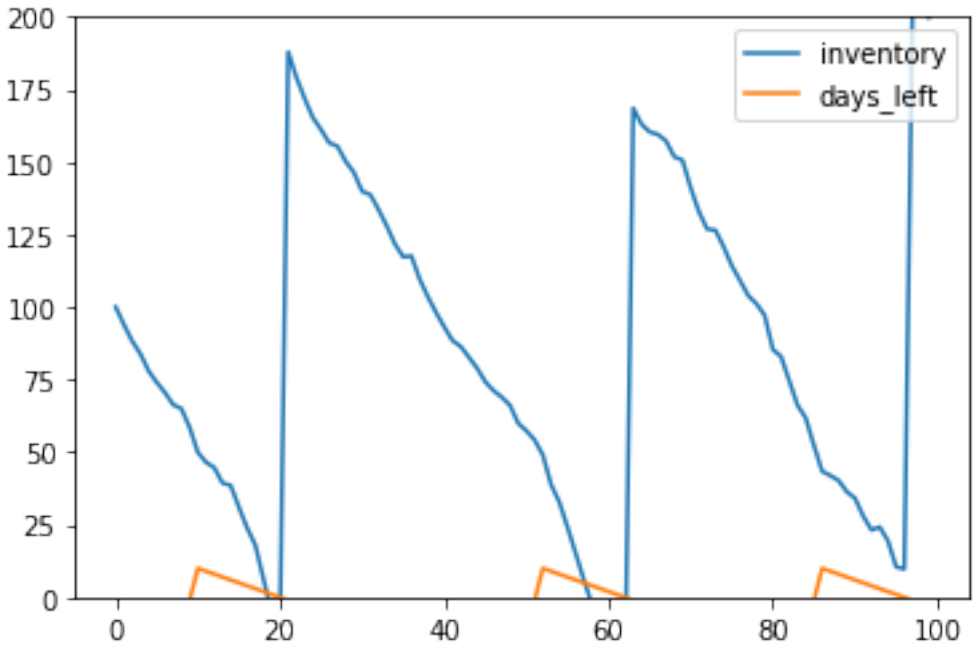
If I increase Q to 200, what will happen to the picture?

- a. Steeper blue triangles
- b. Higher blue triangles
- c. Wider blue triangles
- d. a and b
- e. a and c
- f. b and c
- g. all of these

[29]:
```
state = {'inventory' : 100, 'days_left' : -1, 'cost' : 0}
Q = 200
R = 50
inventory = []
days_left = []
cost = []

T = 100
for t in range(T):
    inventory.append(state['inventory'])
```

```
    days_left.append(state['days_left'])
    cost.append(state['cost'])
    state = step(state,Q,R,params)

plt.plot(range(T),inventory)
plt.plot(range(T),days_left)
#plt.plot(range(T),cost)
plt.legend(['inventory','days_left','cost'])
plt.ylim(0,200)
```

[29]: (0, 200)



2 Lecture 19 Demo, Question 2

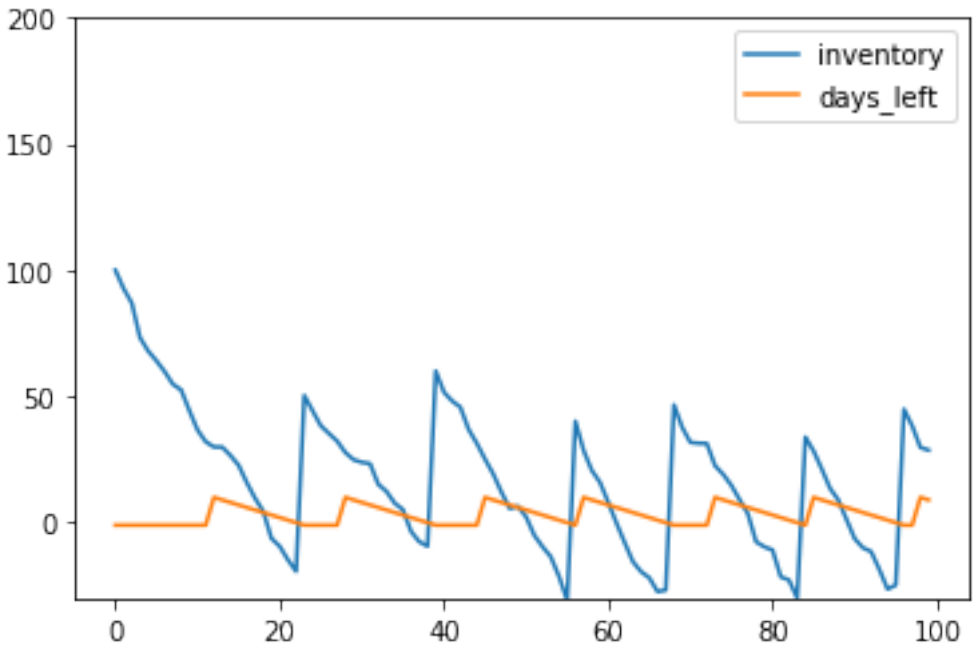
If I decrease R to 30, what will happen to the picture?

- a. Blue triangles whose lowest points are lower
- b. Blue triangles whose lowest points are higher
- c. Blue triangles whose highest points are higher
- d. Blue triangles whose highest points are lower
- e. a and c
- f. b and c

- g. a and d
- h. b and d

[30]:
```
state = {'inventory' : 100, 'days_left' : -1, 'cost' : 0}
Q = 75
R = 30
inventory = []
days_left = []
cost = []

T = 100
for t in range(T):
    inventory.append(state['inventory'])
    days_left.append(state['days_left'])
    cost.append(state['cost'])
    state = step(state,Q,R,params)

plt.plot(range(T),inventory)
plt.plot(range(T),days_left)
plt.legend(['inventory','days_left','cost'])
plt.ylim(-30,200)
```

[30]:
```
(-30, 200)
```



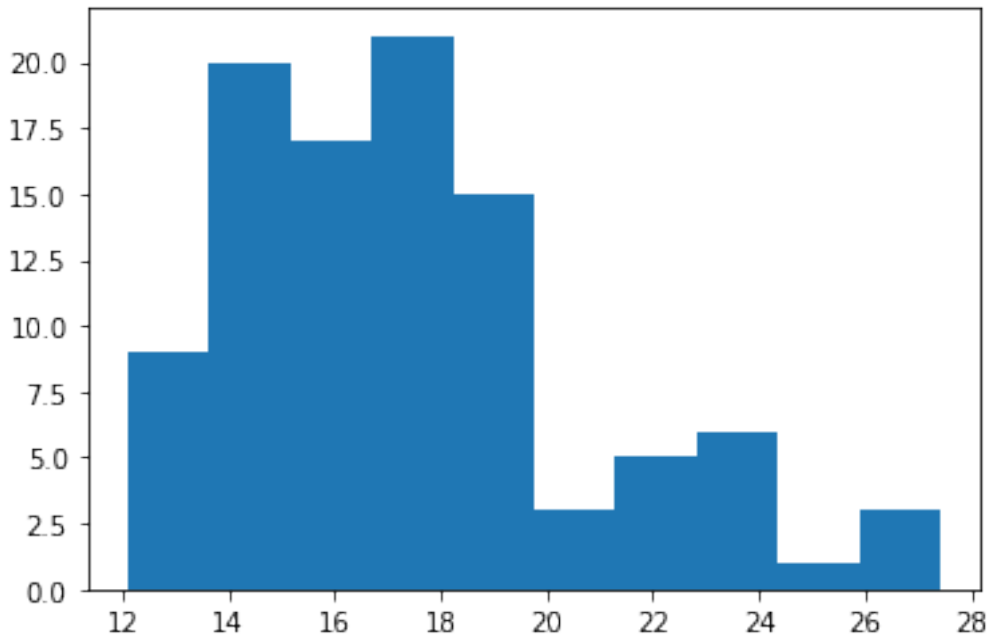
3 Let's find the optimal Q and R

[42]:
```
params = {'demand_mean' : 5, 'demand_std' : 8,
          'lead_time' : 10, # tau in the slides
          'stockout_penalty' : 1, # p in the slides
          'holding_cost' : 0.01, # h in the slides
          'K' : 1, 'c' : 0.5 # order cost parameters
         }
```

[33]:
```
def sim_cost(Q,R,params,T=1000):
    # T is how long to run the sim.
    # In general, we want to simulate a long time so the
    # starting inventory doesn't influence the avg cost too much,
    # and so that we run long enough to average out most of the
    # randomness.

    state = {'inventory' : 100, 'days_left' : -1, 'cost' : 0}
    for t in range(T):
        state = step(state,Q,R,params)
    return state['cost']/T # Average cost
```

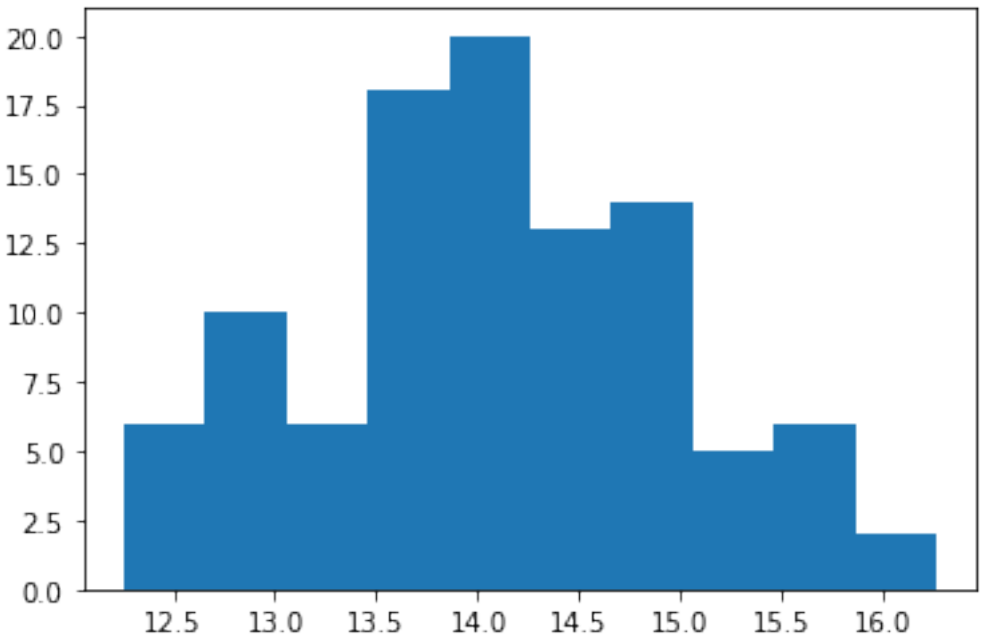
[34]:
```
# This gives a sense for the noise in the simulation.
# It needs to be run a long time to give an accurate answer.
# Run for 100 time periods
x = []
for n in range(100):
    x.append(sim_cost(70,30,params,100))
plt.hist(x)
```

[34]:
```
(array([ 9., 20., 17., 21., 15.,  3.,  5.,  6.,  1.,  3.]),
 array([12.10041295, 13.63264822, 15.16488349, 16.69711875, 18.22935402,
        19.76158929, 21.29382455, 22.82605982, 24.35829509, 25.89053035,
        27.42276562]),
 <a list of 10 Patch objects>)
```



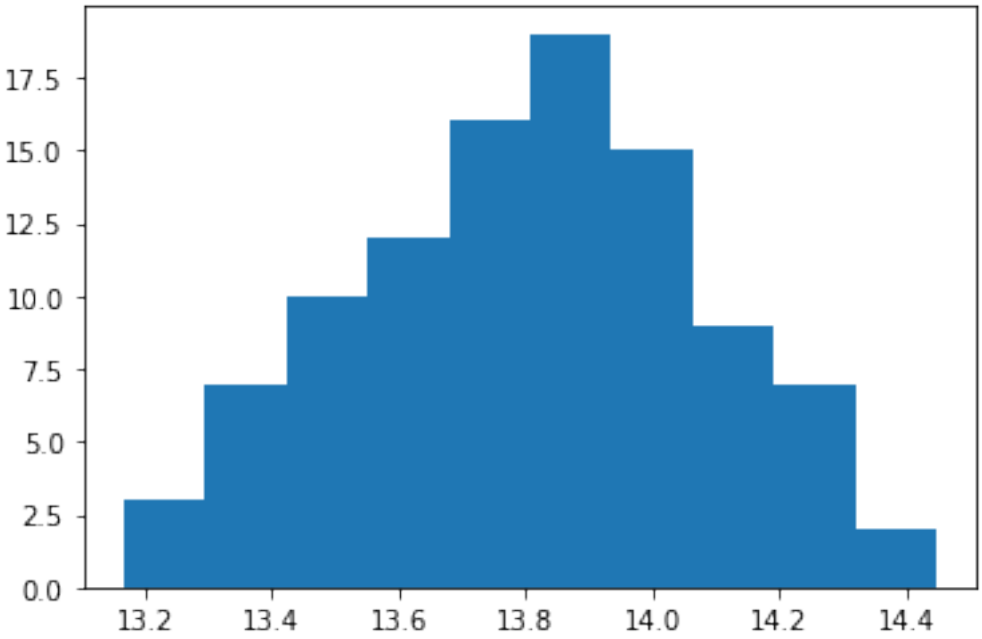
[22]:
```
# Run for 1000 time periods
x = []
for n in range(100):
    x.append(sim_cost(70,30,params,1000))
plt.hist(x)
```

[22]:
```
(array([ 6., 10.,  6., 18., 20., 13., 14.,  5.,  6.,  2.]),
 array([12.25359947, 12.65425538, 13.05491129, 13.4555672 , 13.85622312,
        14.25687903, 14.65753494, 15.05819085, 15.45884677, 15.85950268,
        16.26015859]),
 <a list of 10 Patch objects>)
```



[23]:
```
# Run for 10,000 time periods
x = []
for n in range(100):
    x.append(sim_cost(70,30,params,10000))
plt.hist(x)
```

[23]:
```
(array([ 3.,  7., 10., 12., 16., 19., 15.,  9.,  7.,  2.]),
 array([13.16734233, 13.29530305, 13.42326377, 13.55122448, 13.6791852 ,
        13.80714592, 13.93510664, 14.06306736, 14.19102808, 14.3189888 ,
        14.44694952]),
 <a list of 10 Patch objects>)
```



[43]:
```
# Random search
n = 500 # Number of possible solutions to try in random search
q_upper = 150
possible_solutions = pd.DataFrame()
possible_solutions['Q'] = np.random.uniform(0,q_upper,n)
possible_solutions['R'] = 0 # We'll fill this in momentarily
for index, row in possible_solutions.iterrows():
    Q = row['Q']
    R = np.random.uniform(0,Q)
    possible_solutions.loc[index,'R'] = R
    possible_solutions.loc[index,'cost'] = analytic_cost(Q,R,params)

min_cost = np.min(possible_solutions['cost'])
soln = possible_solutions[possible_solutions['cost']==min_cost]
best_Q = soln['Q'].values[0]
best_R = soln['R'].values[0]
```

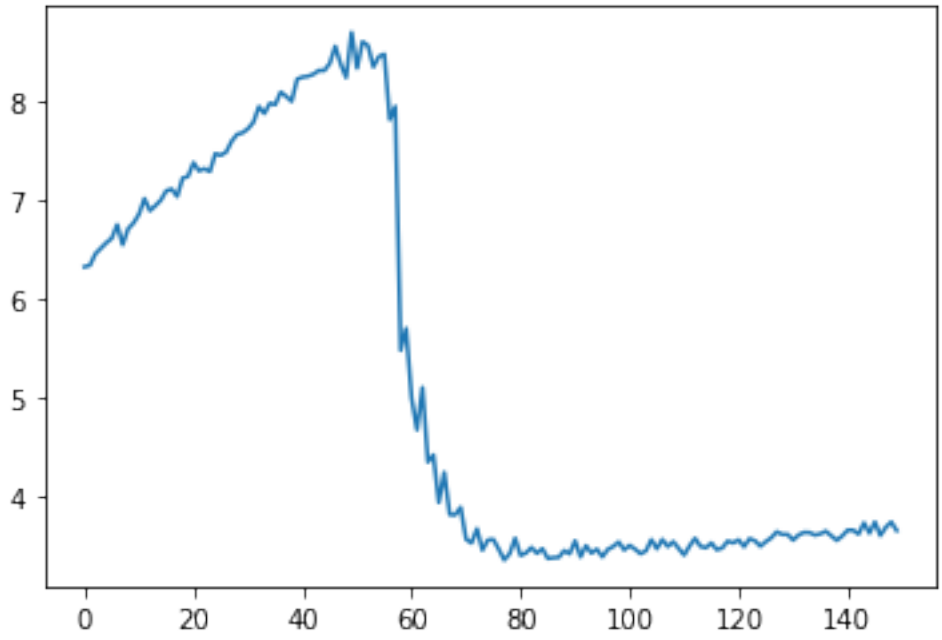
[44]:
```
print(f'The minimum cost is {min_cost}')
print(f'The optimal values are Q={best_Q}, R={best_R}')
```

```
The minimum cost is 3.2459252620196395
The optimal values are Q=102.39080188029766, R=85.8396488401183
```

[45]:
```
# It's ueful to understand how cost changes with Q and R
Q_list = range(0,q_upper)
Q_cost_list = [sim_cost(Q,best_R,params,10000) for Q in Q_list]
```

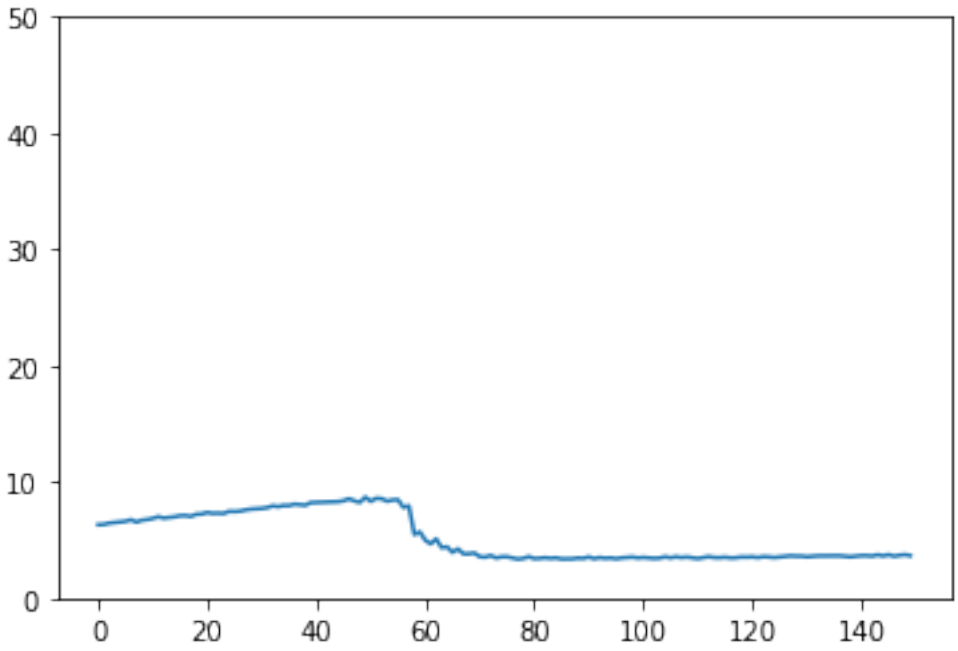
```
plt.plot(Q_list,Q_cost_list)
```

[45]: [<matplotlib.lines.Line2D at 0x7f88507dfdd0>]



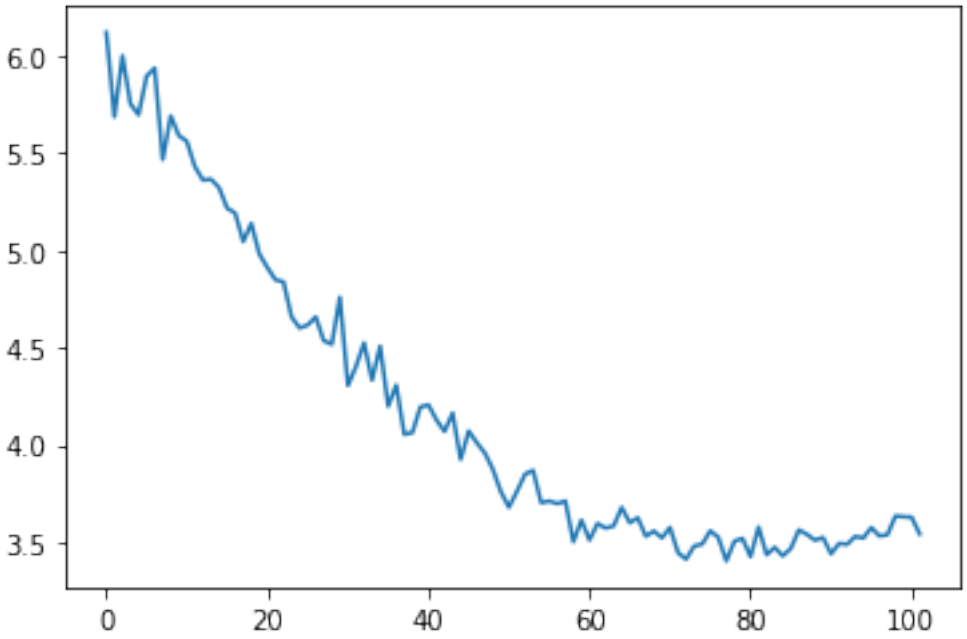
[46]:
```
plt.plot(Q_list,Q_cost_list)
plt.ylim(0,50)
```

[46]: (0, 50)



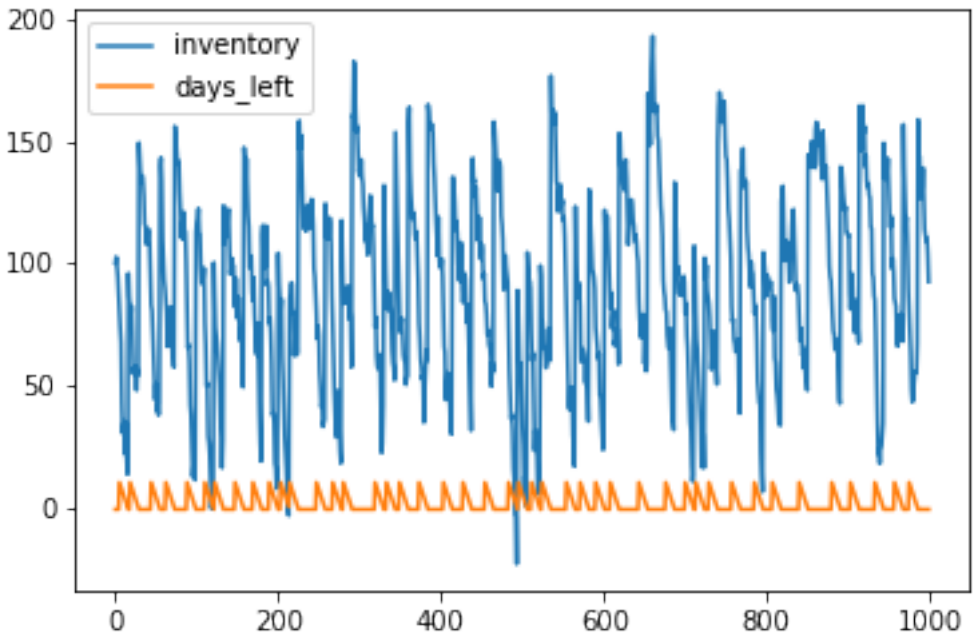
[47]:
```
R_list = range(0,int(best_Q))
R_cost_list = [sim_cost(best_Q,R,params,10000) for R in R_list]
plt.plot(R_list,R_cost_list)
```

[47]: [<matplotlib.lines.Line2D at 0x7f8850824f50>]



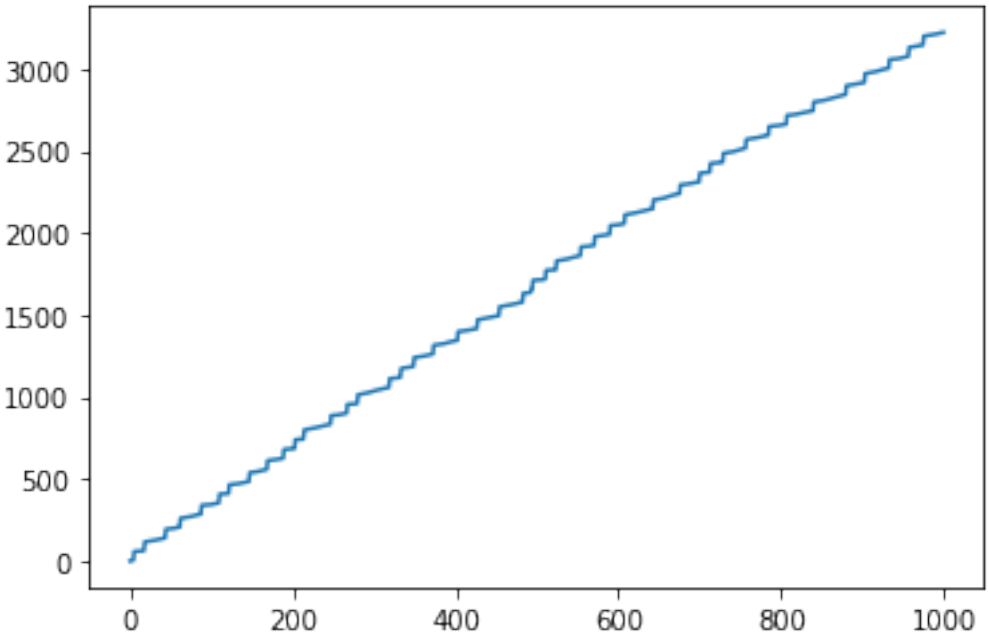
[48]:
```
# Show a sample path under the optimal Q,R values
state = {'inventory' : 100, 'days_left' : -1, 'cost' : 0}
Q = best_Q
R = best_R
inventory = []
days_left = []
cost = []

T = 1000
for t in range(T):
    inventory.append(state['inventory'])
    days_left.append(state['days_left'])
    cost.append(state['cost'])
    state = step(state,Q,R,params)

plt.plot(range(T),inventory)
plt.plot(range(T),days_left)
plt.legend(['inventory','days_left'])
```

[48]:
```
<matplotlib.legend.Legend at 0x7f8850818bd0>
```



[49]:
```
plt.plot(range(T),cost)
```

[49]:
```
[<matplotlib.lines.Line2D at 0x7f88507edad0>]
```


3000
2500
2000
1500
1000
500
0
0
200
400
600
800
1000

[ ]: