File - inventory-prob-dp-bigger-shortage.py

```python
# Python code for larger probabilistic inventory by dynamic programming

import numpy

hugeNumber = float("inf")
unused     = -1000

stages             = 5
startInventory     = 1
inventoryCapacity  = 10
productionCapacity = 10

setupCost    = 10.0
variableCost = 2.0
holdingCost  = 1.0
salvageValue = 2.0
shortageCost = 40.0

minDemand = 0
maxDemand = 4

demandProb = numpy.array([[unused]*5,
                          [0.20, 0.20, 0.20, 0.20, 0.20],
                          [0.10, 0.20, 0.30, 0.20, 0.10],
                          [0.30, 0.30, 0.15, 0.10, 0.05],
                          [0.30, 0.25, 0.20, 0.15, 0.10],
                          [0.20, 0.20, 0.20, 0.20, 0.20]])

f = numpy.zeros([stages + 2, inventoryCapacity + 1])
x = numpy.zeros([stages + 1, inventoryCapacity + 1], dtype=int)

# Set the value of ending up in each final state (not zero in this case)
for i in range(inventoryCapacity+1) :
    f[stages+1][i] = -salvageValue*i   # Negative "cost" = benefit for each leftover unit

for t in range(stages,0,-1) :

    for i in range(inventoryCapacity+1) :

        # Minimum production always 0 because we don't have to meet demand
        maxProduction = min(productionCapacity, inventoryCapacity - i + minDemand)

        value    = hugeNumber

        for p in range(maxProduction+1) :

            # Compute production cost
            productionCost = variableCost*p
            if p > 0 :
                productionCost += setupCost

            moveValue = productionCost
            # Accumulate expected value of this decision
            for d in range(minDemand,maxDemand+1) :
                j = i + p - d
                if j < 0 :
                    shortagePenalty = -j*shortageCost
                    j = 0
                else :
                    shortagePenalty = 0
                moveValue += demandProb[t,d]*(holdingCost*j + shortagePenalty + f[t+1,j])

            if moveValue < value :
                value = moveValue
                bestMove = p

        # End of p loop

```

```
69          f[t,i] = value
70          x[t,i] = bestMove
71
72      # End of i loop
73
74  # End of t loop
75
76  print("Optimal expected cost is " + str(f[1,startInventory]))
77  print("Period 1: produce " + str(x[1,startInventory]))
78  for t in range(2,stages+1) :
79      print("Period " + str(t) + ":")
80      for i in range(inventoryCapacity + 1) :
81          print("    If inventory=" + str(i) + ", produce " + str(x[t,i]))
82
```