File - Inventory-discounted-poisson-stockout.py

```
# Python code for probabilistic inventory with stockouts, Poisson demands,
# and present value

# We need to compute Poisson distributions
import numpy
from binomialPoisson import *

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

stages             = 12
startInventory     = 0
inventoryCapacity  = 50
productionCapacity = 45

setupCost    = 10000.0
variableCost = 2000.0
holdingCost  = 0.0        # No direct holding cost
salvageValue = 1800.0
sellingPrice = 3000.0
discountRate = 0.005

meanDemand = numpy.array([unused,
                          10.0, 10.5, 11.1, 12.2, 12.3, 12.4,
                          11.8, 10.9,  9.8,  9.0,  8.5,  8.1])

# This is the highest demand we could ever possibly meet
maxSupply = inventoryCapacity + productionCapacity

beta = 1.0/(1.0 + discountRate)

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)
# We are doing this as a max problem, so the terminal value is positive
for i in range(inventoryCapacity+1) :
    f[stages+1,i] = salvageValue*i

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

    # We can never meet a demand above maxSupply, so lump all values greater
    # than that in with maxSupply
    demandProb = poisson(meanDemand[t], maxSupply)

    for i in range(inventoryCapacity + 1) :

        maxProduction = min(productionCapacity, inventoryCapacity - i)

        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(maxSupply + 1) :
                sales = min(d, i + p)
                j = i + p - sales
                moveValue += demandProb[d]*\
                             (sellingPrice*sales - holdingCost*j + beta*f[t+1,j])

            if moveValue > value :
                value = moveValue
```

```
                bestMove = p

        # End of p loop

        f[t,i] = value
        x[t,i] = bestMove

    # End of i loop

# End of t loop

print("Optimal EMV is " + str(f[1,startInventory]))
print("Period 1: produce " + str(x[t, startInventory]))
for t in range(2, stages + 1):
    print("Period " + str(t) + ":")
    sumOfxt = sum(x[t,:])
    for i in range(inventoryCapacity + 1):
        print("    If inventory=" + str(i) + ", produce " + str(x[t,i]))
        sumOfxt -= x[t,i]
        if sumOfxt <= 0 :
            print("    If inventory>"+ str(i) + ", produce 0")
            break
```