Use of Markov Decison Models for Risk Management

Prof. Dr. Michael Koller, ETH Zurich

October 10, 2022

Contents

1 Intro

e-mail michael.koller@math.ethz.ch mobile: +447826943343

Additional resources from me, published at Springer:

1) Stochastic Models in Life Insurance, https://link.springer.com/book/10.1007/978-3-642-28439-7
2) Life Insurance Risk Management Essentials, https://link.springer.com/book/10.1007/978-3-642-20721-1

Aim of the presentation

- Understand the conceptual framework used in modelling with Markov Chains,
- Investigate a concrete problem using the methodology, and
- Implement the correspong algoriths in python

2 Markov Chains

$(X_t)_{t\in\mathbb{N}} : (\Omega, \mathscr{A}, P) \to \{1,2,\dots n\}$ is called **Markov Chain**

iff

$$P[X_{t_{m+1}} = i_{m+1} | X_{t_1} = i_1, \dots X_{t_m} = i_m] \quad = \quad P[X_{t_{m+1}} = i_{m+1} | X_{t_m} = i_m]$$

for

$$t_1 < t_2 < \cdots < t_m < t_{m+1}$$

and

$$i_1, i_2, \dots, i_m, i_{m+1} \in \{1, 2, \dots n\}$$

We **define**

$$p_{ij}(s,t) \quad := \quad P[X_t = j | X_s = i]$$

Question: How does $p_{ij}(s,t)$, $p_{jk}(t,u)$ and $p_{ik}(s,u)$ relate?

$\mathscr{S} := \{1, 2, \dots, n\}$

Thm: Chapman-Kolmogorov For $s < t < u$ and $i, k \in \mathscr{S}$ we have the following

$$\begin{aligned} p_{ik}(s,u) &= \sum_{j \in \mathscr{S}} p_{ij}(s,t)\, p_{jk}(t,u) \\ P(s,u) &= P(s,t) P(s,u) \end{aligned}$$

Proof

$$\begin{aligned} p_{ik}(s,u) &= P[X_u = k | X_s = i] \\ &= P[X_u = k \cap \Omega | X_s = i] \\ &= P[X_u = k \cap \cup_{j \in \mathscr{S}} \{X_t = j\} | X_s = i] \\ &= \sum_{j \in \mathscr{S}} P[X_u = k, X_t = j | X_s = i] \end{aligned}$$

Now lets look at $P[X_u = k, X_t = j | X_s = i]$

$$\begin{aligned} P[X_u = k, X_t = j | X_s = i] &= \frac{P[X_u = k, X_t = j, X_s = i]}{P[X_s = i]} \\ &= \frac{P[X_u = k, X_t = j, X_s = i]}{P[X_s = i]} \times \frac{P[X_t = j, X_s = i]}{P[X_t = j, X_s = i]} \\ &= P[X_t = j | X_s = i] \times P[X_u = k | X_t = j, X_s = i] \\ &= P[X_t = j | X_s = i] \times P[X_u = k | X_t = j] \\ &= p_{ij}(s,t)\, p_{jk}(t,u) \end{aligned}$$

$$\begin{aligned} p_{ik}(s,u) &= P[X_u = k | X_s = i] \\ &= P[X_u = k \cap \Omega | X_s = i] \\ &= P[X_u = k \cap \cup_{j \in \mathscr{S}} \{X_t = j\} | X_s = i] \\ &= \sum_{j \in \mathscr{S}} P[X_u = k, X_t = j | X_s = i] \\ &= p_{ij}(s,t)\, p_{jk}(t,u) \end{aligned}$$

3 Cash Flows and Reserves for Markov Models

3.1 Life Insurance Markov Model

We are given

a) State Space (finite) $\mathscr{S}$

b) Discount factor $v = \frac{1}{1+i}$

c) Transition Probabilities $P(t, t+1)$

d) Benefit functions $a_i^{Pre}(t)$ and $a_{ij}^{Post}(t)$

What is the Cash flow induced

We define $I_i(t) = \chi_{X_t = i}$. Which mean that you would get the following annuity at time t: $\sum_{i \in S} a_i^{pre}(t) \times I_i(t)$. Which transition (death) benefit would you get at time t? You get $a_{ij}^{post}(t)$ if you are in state i at time t and in state j at $t+1$? ie $a_{ij}^{post}(t) \times I_i(t) \times I_j(t+1)$. In sum for the death benefit $\sum_{i,j \in S^2} a_{ij}^{post}(t) \times I_i(t) \times I_j(t+1)$.

So the cash flow at times t can be calculated as

$$A(t) = \sum_{i \in S} a_i^{pre}(t) \times I_i(t) + \sum_{i,j \in S^2} a_{ij}^{post}(t) \times I_i(t) \times I_j(t+1) \tag{1}$$

$$\tilde{A}(t) = \sum_{i \in S} a_i^{pre}(t) \times I_i(t) + \sum_{i,j \in S^2} a_{ij}^{post}(t) \times I_i(t) \times I_j(t+1) \times v \tag{2}$$

Reserve

$V_j(t) = \mathbb{E}[PV(FutureCashFlows) | X_t = j]$

What is the value of this insurance cash flow?

We define the **Mathematical Reserve** as

$$\begin{aligned} V_j(t) &= \mathbb{E}[PV\, of\, future CF\, | X_t = j] && (3)\\ &= \mathbb{E}[\sum_{\tau=0}^{\infty} v^\tau \tilde{A}(t+\tau) | X_t = j] && (4)\\ & && (5) \end{aligned}$$

3.2 Value of Cash Flows

In order to calculate the mathematical reserve you can substitute $\tilde{A}$ in the formula below and ultimately what you need to calculate (keeping in mind the linearity of the $\mathbb{E}$ functional as the following quantities:

$$\begin{aligned} \mathbb{E}[I_i(t+\tau) | X_t = j] &= p_{ji}(t, t+\tau) && (6)\\ \mathbb{E}[I_i(t+\tau) \times I_k(t+\tau+1) | X_t = j] &= ? && (7) \end{aligned}$$

How do we do this?

$$\mathbb{E}[I_i(t+\tau) \times I_k(t+\tau+1) | X_t = j] = P[X_{t+\tau+1} = k, X_{t+\tau} = i | X_t = j] \tag{8}$$

Now we do the same as in the proof of the Chapman-Kolmogorov-Equation]

$$\begin{aligned} \mathbb{E}[I_i(t+\tau) \times I_k(t+\tau+1) | X_t = j] &= P[X_{t+\tau+1} = k, X_{t+\tau} = i | X_t = j] && (9)\\ &= \frac{P[X_{t+\tau+1} = k, X_{t+\tau} = i, X_t = j]}{P[X_t = j]} && (10)\\ &= \frac{P[X_{t+\tau+1} = k, X_{t+\tau} = i, X_t = j]}{P[X_t = j]} \times \frac{P[X_t = j, X_{t+tau} = i]}{P[X_t = j, X_{t+tau} = i]} && (11)\\ &= P[X_{t+\tau} = i | X_t = j] \times P[X_{t+\tau+1} = k | X_t = j, X_{t+\tau} = i] && (12)\\ &= p_{ji}(t, t+\tau) \times p_{ik}(t+\tau, t+\tau+1) && (13) \end{aligned}$$

If we put now all things together we can can calcuate the mathematical reserves as follows

$$
\begin{aligned}
V_j(t) &= \mathbb{E}[\sum_{\tau=0}^{\infty} v^\tau \tilde{A}(t+\tau) | X_t = j] && (14)\\
&= \sum_{\tau=0}^{\infty} v^\tau \left(\sum_{i \in S} a_i^{pre}(t+\tau) \times p_{ji}(t,t+\tau) + \sum_{i,j \in S^2} a_{ij}^{post}(t+\tau) p_{ji}(t,t+\tau) \times p_{ik}(t+\tau, t+\tau+1) \times v \right) && (15)
\end{aligned}
$$

Remark: With this formula we can also calculate the expected cash flows at time t as follows:

$$
\mathbb{E}[A(t+\tau) | X_t = j] = \sum_{i \in S} a_i^{pre}(t+\tau) \times p_{ji}(t,t+\tau) + \sum_{i,j \in S^2} a_{ij}^{post}(t+\tau) p_{ji}(t,t+\tau) \times p_{ik}(t+\tau, t+\tau+1) \quad (16)
$$

Thiele Difference Equation this is the relationship between the mathematical reserves between times t and $t+1$. The relationship is as follows:

$$
V_j(t) = a_j^{pre}(t) + v \sum_{k \in S} p_{jk}(t,t+1) \times \left(a_{jk}^{post}(t) + V_k(k+1) \right) \quad (17)
$$

To prove this equation we split the time-sum into $\tau = 0$ and the rest. For $\tau = 0$ we get

$$
a_j^{pre}(t) + v \sum_{k \in S} p_{jk}(t,t+1) \times a_{jk}^{post}(t) \quad (18)
$$

as per above.

In a second step we need to consider (**NEW AND IMPROVED FORMULA**:)

$$
\begin{aligned}
&\sum_{\tau=1}^{\infty} v^\tau \left(\sum_{i \in S} a_i^{pre}(t+\tau) \times p_{ji}(t,t+\tau) + \sum_{(i,k) \in S^2} a_{ik}^{post}(t+\tau) p_{ji}(t,t+\tau) \times p_{ik}(t+\tau, t+\tau+1) \times v \right) && (19)\\
&= \sum_{\tau=1}^{\infty} v^\tau \sum_{i \in S} p_{ji}(t,t+\tau) \times \left(a_i^{pre}(t+\tau) + \sum_{k \in S} a_{ik}^{post}(t+\tau) p_{ik}(t+\tau, t+\tau+1) \times v \right) && (20)
\end{aligned}
$$

We can now calculate the quantity $p_{ji}(t,t+\tau)$ as follows, by means of the Chapman-Kolmorgorov equation

$$
p_{ji}(t,t+\tau) = \sum_{l \in S} p_{jl}(t,t+1) \times p_{li}(t+1,t+\tau) \quad (21)
$$

$$
\begin{aligned}
&\sum_{\tau=1}^{\infty} v^\tau \left(\sum_{i \in S} a_i^{pre}(t+\tau) \times p_{ji}(t,t+\tau) + \sum_{i,j \in S^2} a_{ij}^{post}(t+\tau) p_{ji}(t,t+\tau) \times p_{ik}(t+\tau, t+\tau+1) \times v \right)\\
&= \sum_{\tau=1}^{\infty} v^\tau p_{ji}(t,t+\tau) \times \left(\sum_{i \in S} a_i^{pre}(t+\tau) + \sum_{i,j \in S^2} a_{ij}^{post}(t+\tau) p_{ik}(t+\tau, t+\tau+1) \times v \right)\\
&= \sum_{\tau=1}^{\infty} v^\tau \left(\sum_{l \in S} p_{jl}(t,t+1) \times p_{li}(t+1,t+\tau) \right) \times \left(\sum_{i \in S} a_i^{pre}(t+\tau) + \sum_{i,j \in S^2} a_{ij}^{post}(t+\tau) p_{ik}(t+\tau, t+\tau+1) \times v \right)\\
&= \sum_{l \in S} p_{jl}(t,t+1) \times v \times \left(\sum_{\tau=0}^{\infty} v^\tau p_{li}(t+1,t+1+\tau) \times \left(\sum_{i \in S} a_i^{pre}(t+1+\tau) + \sum_{i,j \in S^2} a_{ij}^{post}(t+1+\tau) p_{ik}(t+1+\tau, t+1+\tau+1) \times v \right) \right)\\
&= \sum_{l \in S} p_{jl}(t,t+1) \times v \times V_l(t+1)
\end{aligned}
$$

Remarks:

1) For one life we have a recursion of reals $A_x = q_x \times v + p_x \times v \times A_{x+1}$. In case of MR of a Markov model we have a recursion of vectors.
2) To solve it one needs boundary conditions as per before with $V_j(\omega) = 0 \forall j \in S$
3) Thiele Difference Equations leads to the same results as for the classical life insurance we have seen.

4 3. Markov Chain Decision Models

How does the framework change? Given

We are given

a) State Space (finite) $\mathscr{S}$

a2) We are given a finite set of actions $\mathscr{A}$. An is taken at time t w/o knowledge of the future of the MC (previsible)

b) Discount factor $v = \frac{1}{1+i}$

c) Transition Probabilities $P(t,t+1)(a)$ for each $a \in \mathscr{A}$

d) Benefit functions $a_i^{Pre}(t,a)$ and $a_{ij}^{Post}(t,a)$ for each $a \in \mathscr{A}$

Task

Optimise the action to maximise the Value at the beginning given $X_t = i$:

$\vec{b} = argmax\{V(t,\vec{a})|X_t = i)|\vec{a} \in \mathscr{A}^{\mathbb{N}}\}$

For a given $\vec{a}$ the mathematical reserve $V(t,\vec{a}|X_t = i)$ is defined as the reserve $V_i(t)$ the markov model using the following assumptions:

a) State Space (finite) $\mathscr{S}$

b) Discount factor $v = \frac{1}{1+i}$

c) Transition Probabilities $P(t,t+1)(\vec{a}[t])$

d) Benefit functions $a_i^{Pre}(t,\vec{a}[t])$ and $a_{ij}^{Post}(t,\vec{a}[t])$

We define

$V_i^*(t) = V_i(t,\vec{b})$

We proof (very similar to Thiele using previsibility of decision) that the following recursion holds (Thiele-Bellman recusion).

Thiele-Bellman Difference Equation

$V_i^*(t) = max\{a_i^{Pre}(t,a) + \sum_{j \in S} v \times p_{ij}(t,t+1)(a) \times (a_{ij}^{post}(t,a) + V_j^*(t+1))|a \in \mathscr{A}\}$

subject to Boundary Conditions

$V_i^*(\omega) = 0$

5 Concrete Task

We have seen the theory regarding Markov optimisation and the related Thiele-Bellman Difference Equation. Please explain the concept and what what are the similarities and differences to the Markov model without decision choices. Provide an idea how to prove the Thiele-Bellman Difference Equation, ie

$$V_i^*(t) = max\{a_i^{Pre}(t,a) + \sum_{j \in S} v \times p_{ij}(t,t+1)(a) \times (a_{ij}^{post}(t,a) + V_j^*(t+1))|a \in \mathscr{A}\}$$

subject to boundary conditions: $V_i^*(\omega) = 0$.

Based on the above concept solve the following task>

- We consider a disability insurance policy. It is know that there are strategies to lower the claim frequency by preventive measures, both during the time where people are healthy (such as motivate them do do more sports and having a more healthy life style) and also when there become disables (ie provide measures the quickly reintegrate them back into the workforce).
- The aim of the task is to find an optimal strategy to reduce the total costs of the insurance cover
- It is expected that you define the respective markov models formally and find the optimal solution per age and state including the respective optimal mathematical reserves.

5.1 Base Model

In the following we define the disability model (in terms of states and transition probabilities). We consider 10 States $(\star, \dagger, \diamondsuit_1, \ldots \diamondsuit_8)$, as per the lecture, where people can reactivate in states $\diamondsuit_1, \ldots \diamondsuit_7$. The probabilities are given as follows (you can use them as python code)

Mortality Healthy def mua(x): return(np.exp(-7.85785 + 0.01538*x* + *0.000577355*x**2))

Mortality Disabled def mui(x): return(mua(x)+0.008)

Disability Incicdence def sigma(x): return(3.e-4 * (8.4764-1.0985*x* + *0.055*x**2))

Reaktivation def alpha(x,k): return(max(0,0.773763-0.01045*(x-k + 1)))

def rx(x,k,n=8): if (k>= n): return(0.) return(np.exp(-0.94*(k-1))* alpha(x, k))

We consider a disability annuity of 12'000 pa against single premium payment

x0 = 25 s = 65 iRate = 0.02 Annuity = 12000

5.2 Possible Strategies

For each state and age there are four possible choices for the strategy as follows:

- No action; then the above model and assumptions are used
- For people all active people one can reduce disability incidence rate (sigma(x)) by 17.5% (RedToInval = 0.175) for an annual cost of 150 (CostActive = 150)
- For all active people with an age ($<$ 45) one can reduce incidence rate by 70% for an annual cost of 5
- For disabled people one can increase reactivation probability by 20% for a cost of 1500.

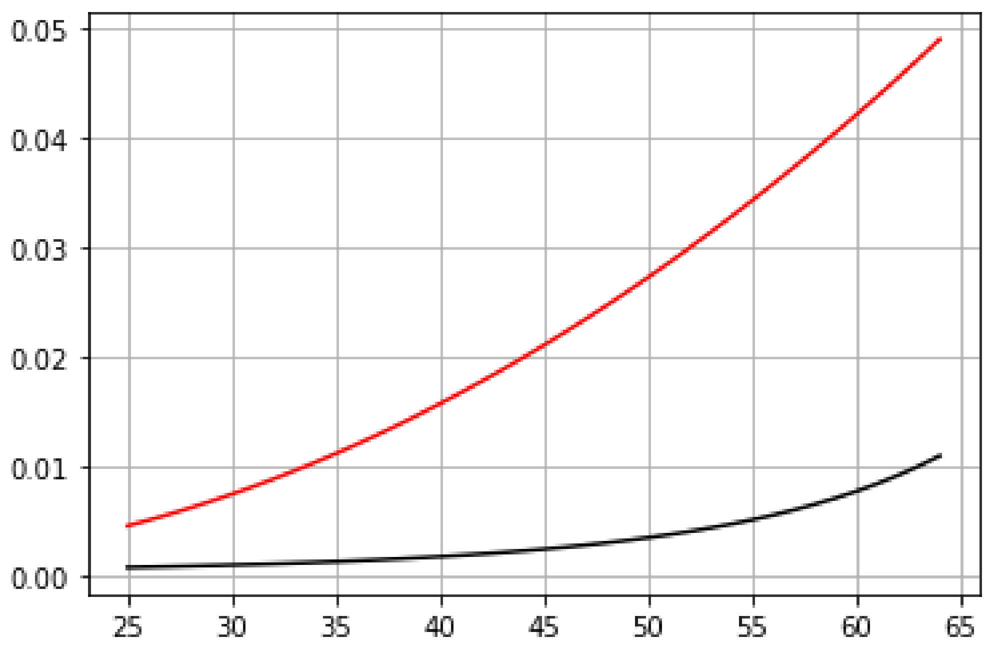
5.3 Expected Results

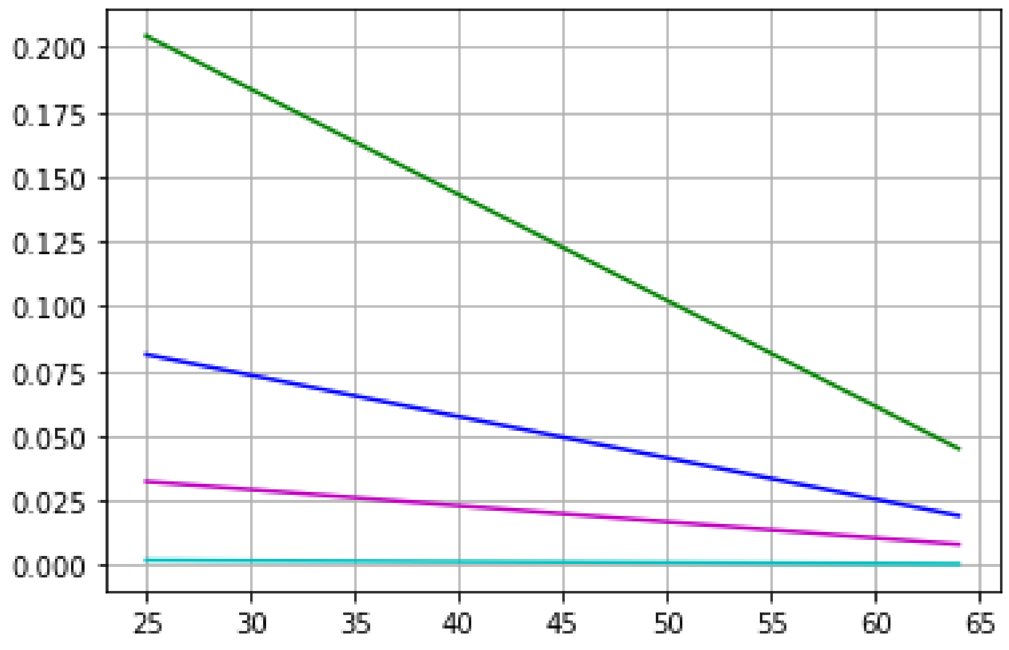
- Discussion of you results
- Sample Reserves for certain ages and stages to evidence that the algorithm works
- A graphic as the one below showing graphically the optimal strategy. In the figure x-axis: age, y-axis: state (0 healthy, 1-8: disabled and 9 dead) color corresponds to chosen strategy)

6 Concrete Solution

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
import copy

class Markov:
    def __init__(self):
        self.iNrStates = None
        self.iMaxTime  = None
        self.dPij = [] # for each time a matrix ie dPij[k] matrix at time␣
 ↪k
        self.dPre = [] # Vector vector of annuities at time t
        self.dPost= []
        self.dv   = []
        # Outputs
        self.dDK  = []
        self.dDKDistr  = []
        self.dCF  = []
        self.bCalculated = False
        self.bCFCalculated = False
        self.bCalculatedDistr = False
        self.iStart = None
        self.iStop  = None
        self.fDistrLow = -1000
        self.fDistrHigh = 150000
        self.iNrBuckets = 10000
        self.fBucketWidth = (self.fDistrHigh-self.fDistrLow)/self.
↪iNrBuckets
        self.fBucketWidthRound = self.fBucketWidth / 2.

    def vDefineModel(self,iNrStates,iMaxTime=1200):
        self.iNrStates = iNrStates
        self.iMaxTime = iMaxTime
        for i in range(iMaxTime):
            tempPij = np.zeros([iNrStates,iNrStates])
            tempPost = np.zeros([iNrStates,iNrStates])
            tempPre = np.zeros([iNrStates])
            tempDK = np.zeros([iNrStates])
            tempCF = np.zeros([iNrStates])
            self.dPij.append(tempPij)
            self.dPost.append(tempPost)
            self.dPre.append(tempPre)
            self.dDK.append(tempDK)
            self.dCF.append(tempCF)
        tempv = np.zeros([iMaxTime])
        self.dv=tempv

    def iBucketNr(self, fValue):
        if fValue < self.fDistrLow:
            return(0)
        iBNR = (int(min(self.iNrBuckets-1,(fValue-self.fDistrLow)/self.
↪fBucketWidth)))
        return(iBNR)

    def fValueOfBucket(self, iBucket):
        return(self.fBucketWidth*min(self.iNrBuckets-1,iBucket)+self.
↪fDistrLow)

    def vCreateDistModel(self):
        print("You Know that you can call me only once everything is␣
↪done")
        print("We are using (states/buckets):",self.iNrStates,self.
↪iNrBuckets)
        for i in range(self.iMaxTime):
            tempDK = np.zeros([self.iNrStates,self.iNrBuckets])
            self.dDKDistr.append(tempDK)

    def vSetDiscount(self,fIRate):# you set v
        vTemp = 1./(1.+fIRate)
        for i in range(self.iMaxTime):
            self.dv[i] = vTemp
        self.bCalculated = False
        self.bCFCalculated = False

    def vSetPij(self,t,i,j,fValue):# you set p_{ij}(t,t+1)
        self.dPij[t][i,j] = fValue
        self.bCalculated = False
        self.bCFCalculated = False

    def vSetPre(self,t,i,j,fValue):# you set a_{i}^{pre}(t)
        self.dPre[t][i] = fValue
        self.bCalculated = False
        self.bCFCalculated = False

    def vSetPost(self,t,i,j,fValue):# you set a_{ij}^{post}(t)
        self.dPost[t][i,j] = fValue
        self.bCalculated = False
        self.bCFCalculated = False

    def doComplementStates(self,default=None, eps = 0.0001):
        iState = self.iNrStates -1
        if default != None:
            iState = default
        for i in range(self.iNrStates):
            bFound = False
            bStop = False
            for t in range(self.iStop,self.iStart):
                fTot = sum(self.dPij[t][i,:])
                #print(i,t,"-->",fTot
                if abs(fTot-1.) >= eps:
                    if fTot >1+eps:
                        bStop = True
                        print("Error time",t,"State",i,"Value:",fTot)
                    bFound=True
                    self.dPij[t][i,iState] += 1. - fTot
            if bFound:
                print("Check P(Omega) = 1 failed for iState=",i,"Target␣
↪State",iState)
            if bStop:
                stop()


    def doCalculateDK(self,iStart,iStop,iAge,iState):
        self.iStop = iStop
        self.iStart = iStart
        self.bCalculated = True
        for i in range(self.iMaxTime):
            self.dDK[i] *= 0.

        for i in range(self.iStart-1, self.iStop-1,-1):
            #print("Calc Time", i)
            for j in range(self.iNrStates):
                self.dDK[i][j] = self.dPre[i][j]
                for k in range(self.iNrStates):
                    self.dDK[i][j] += self.dv[i]*self.dPij[i][j,k]*(self.
↪dPost[i][j,k]+self.dDK[i+1][k])

    def doCalculateCF(self,iStart,iStop,iAge,iState,bTrace=False):
        self.iStop = iStop
        self.iStart = iStart
        self.bCFCalculated = True
        for i in range(self.iMaxTime):
            self.dCF[i] *= 0.

        CurrentP = np.mat(np.identity(self.iNrStates))
        if bTrace:
            print("----- ----- ----- ----- ")
        for i in range(self.iStop, self.iStart):
            if bTrace:
                print("----- ----- ----- ----- ")
                print(" Time ", i)
                print("CF BoP", self.dCF[i])
            for k in range(self.iNrStates):
                for l in range(self.iNrStates):
                    self.dCF[i][k] += CurrentP[k,l] * self.dPre[i][l]
            if bTrace:
                print("CF BoP after Pre", self.dCF[i])
            NextP = np.mat(self.dPij[i])
            if bTrace:
                print("+++++ +++++ +++++ ")
                print("CurrentP\n", CurrentP)
                print("+++++ +++++ +++++ ")
                print("Next P\n", NextP)
                print("+++++ +++++ +++++ ")

            for k in range(self.iNrStates):
                for l in range(self.iNrStates):
                    for m in range(self.iNrStates):
                        self.dCF[i+1][k] += CurrentP[k,l] * NextP[l,m] *␣
↪self.dPost[i][l,m]
            if bTrace:
                print("CF EoP t", self.dCF[i])
                print("CF EoP t+1", self.dCF[i+1])

            CurrentP = CurrentP * NextP # This is Chapman Kolmogorov
            if bTrace:
                print("+++++ +++++ +++++ ")
                print("CurrentP EoP\n", CurrentP)
                print("+++++ +++++ +++++ ")

    def doCalculateDKDistr(self,iStart,iStop,iAge,iState,default=None):
        self.iStop = iStop
        self.iStart = iStart
        self.bCalculatedDistr = True
        self.vCreateDistModel()
        print("default is",str(default))
        self.doComplementStates(default=default)
        for i in range(self.iMaxTime):
            self.dDKDistr[i] *= 0.
        # Set Boundary Conditions
        iIndexSwitch = self.iBucketNr(0)
        print("switch index:",iIndexSwitch)
        for j in range(self.iNrStates):
            value = 0.
            for l in range(self.iNrBuckets):
                if l > iIndexSwitch:
```

```
                              value = 1.
                    self.dDKDistr[self.iStart][j,l] = value
        # Calculation
        for i in range(self.iStart-1, self.iStop-1,-1):
            print("Dirst DK Calc Time", i)
            for j in range(self.iNrStates):
                for k in range(self.iNrStates):
                    for l in range(self.iNrBuckets):
                        dNewXTPlusOne = (self.fValueOfBucket(l) - self.
↪dPre[i][j])/self.dv[i] - self.dPost[i][j,k]
                        self.dDKDistr[i][j,l] += self.dPij[i][j,k]*(self.
↪dDKDistr[i+1][k,self.iBucketNr(dNewXTPlusOne)])


    def dGetDK(self,iStart,iStop,iAge,iState):
        if (iStart != self.iStart or iStop != self.iStop or not(self.
↪bCalculated)):
            self.doCalculateDK(iStart,iStop,iAge,iState)
        return(self.dDK[iAge][iState])

    def dGetCF(self,iStart,iStop,iAge,iState):
        if (not(self.bCFCalculated) or self.iStart != iStart or self.
↪iStop != iStop ):
            self.doCalculateCF(iStart,iStop,iAge,iState)
        return(self.dCF[iAge][iState])

    def dGetDKDistr(self,iStart,iStop,iAge,iState,fValue,default=None):
        if (iStart != self.iStart or iStop != self.iStop or not(self.
↪bCalculatedDistr)):
            temp = self.dGetDK(iStart,iStop,iAge,iState) # To be on the␣
↪safe side
            self.
↪doCalculateDKDistr(iStart,iStop,iAge,iState,default=default)
        return(self.dDKDistr[iAge][iState,self.iBucketNr(fValue)])

    def PrintDKs(self,iStart,iStop):
        for i in range(iStop,iStart+1):
            strTemp = " %3d :"%(i)
            for j in range(self.iNrStates):
                 strTemp += "  %7.4f "%(self.dGetDK(iStart,iStop,i,j))
            print(strTemp)

    def PlotDKs(self,iStart,iStop,figNr=1):
        x = []
        y = []
        for i in range(iStop,iStart+1):
            x.append(i)
            ytemp = np.zeros(self.iNrStates)
            for j in range(self.iNrStates):
                ytemp[j] = self.dGetDK(iStart,iStop,i,j)
            y.append(ytemp)
        plt.figure(figNr)
        plt.plot(x,y)
        plt.grid(True)

    def PlotCFs(self,iStart,iStop,figNr=2,bLines=True):
        import matplotlib.colors as mcolors
        if bLines:
            x=[]
            y=[]
            plt.figure(figNr)

            for j in range(self.iNrStates):
                x=[]
                y=[]
                for i in range(iStop,iStart+1):
                    x.append(i)
                    y.append(self.dGetCF(iStart,iStop,i,j))
                plt.plot(x,y)
            plt.grid(True)
        else:
            A= []
            for i in mcolors.TABLEAU_COLORS.keys():
                A.append(i)
            for i in mcolors.BASE_COLORS.keys():
                A.append(i)

            xBar =[]
            hBar =[]
            bBar =[]
            cBar =[]
            y = []
            for i in range(iStop,iStart+1):
                for j in range(self.iNrStates):
                    xBar.append(i+(j)*1./self.iNrStates)
                    hBar.append(self.dGetCF(iStart,iStop,i,j))
                    bBar.append(0)
                    cBar.append(A[j])

            plt.figure(figNr)
            plt.bar(xBar,hBar,bottom=bBar, width = 1./self.
↪iNrStates,color=cBar)
            plt.grid(True)

    def PlotDKDistr(self,iStart,iStop, iSteps = None, iStates = [0],␣
↪iDeltaT = 5, figNr=10, eps = 0.01,legTitle="",default=None):
        if iSteps == None:
            iSteps = []
            for i in range(iStop,iStart,iDeltaT):
                iSteps.append(i)
            iSteps.append(iStart)
        for i in iSteps:
            for j in iStates:
                x = []
                y = []
                for k in range(self.iNrBuckets):
                    xLoc = eps + self.fValueOfBucket(k)
                    yLoc = self.
↪dGetDKDistr(iStart,iStop,i,j,xLoc,default=default)
                    x.append(xLoc)
                    y.append(yLoc)

                plt.figure(figNr)
                plt.plot(x,y)
                plt.grid(True)
                mylegend = legTitle + "Age %d - State %d"%(i,j)
                plt.title(mylegend)
                figNr+=1


def mua(x):
    return(np.exp(-7.85785 + 0.01538*x + 0.000577355*x**2))

def mui(x):
    return(mua(x)+0.008)

def sigma(x):
    return(3.e-4 * (8.4764-1.0985*x + 0.055*x**2))

def alpha(x,k):
    return(max(0,0.773763-0.01045*(x-k + 1)))

def rx(x,k,n=8):
    k+=1
    if (k>= n): return(0.)
    return(np.exp(-0.94*(k-1)) * alpha(x, k))



x=[]
ix =[]
qx=[]
rxv=[]
rxv1=[]
rxv2=[]
rxv5=[]

for i in range(25,65):
    x.append(i)
    qx.append(mua(i))
    ix.append(sigma(i))
    rxv.append(rx(i,1))
    rxv2.append(rx(i,3))
    rxv1.append(rx(i,2))
    rxv5.append(rx(i,6))

plt.figure(1)
plt.plot(x,qx,"k",x,ix,'r')
plt.grid(True)


plt.figure(2)
plt.plot(x,rxv,'g',x,rxv1,'b',x,rxv2,'m',x,rxv5,'c')
plt.grid(True)
```

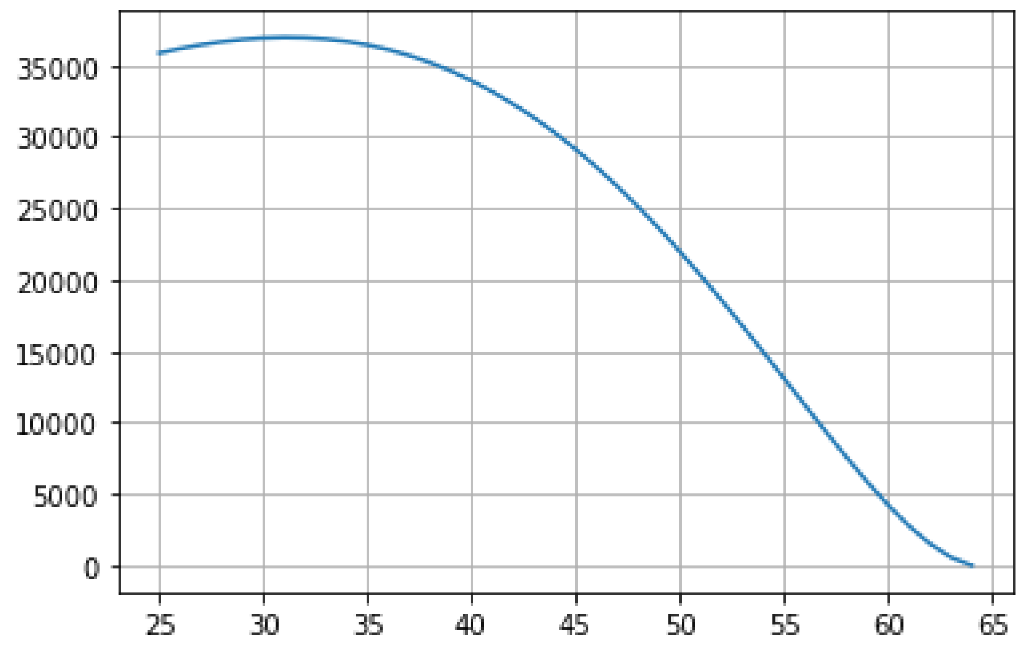


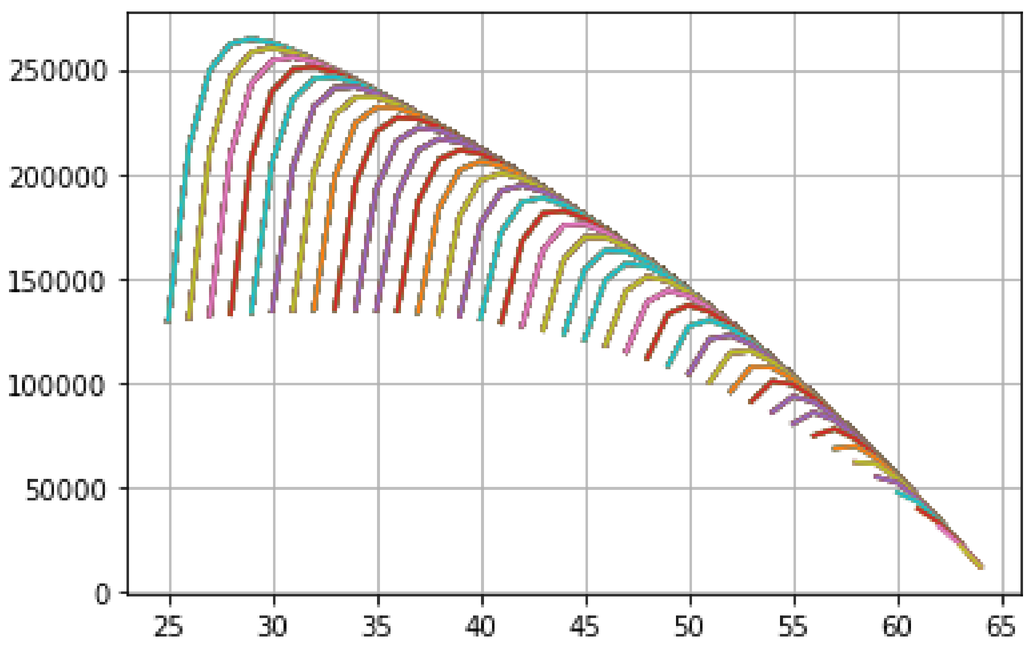


```
[4]: x0  = 25
     s   = 65
     iRate = 0.02
     Annuity = 12000
     CostActive = 150
     RedToInval = 0.175
     CostDisab = 1500
     RedFromInval = 1.2
     CostActive40 = 5.0
     RedActive40 = 0.30
```

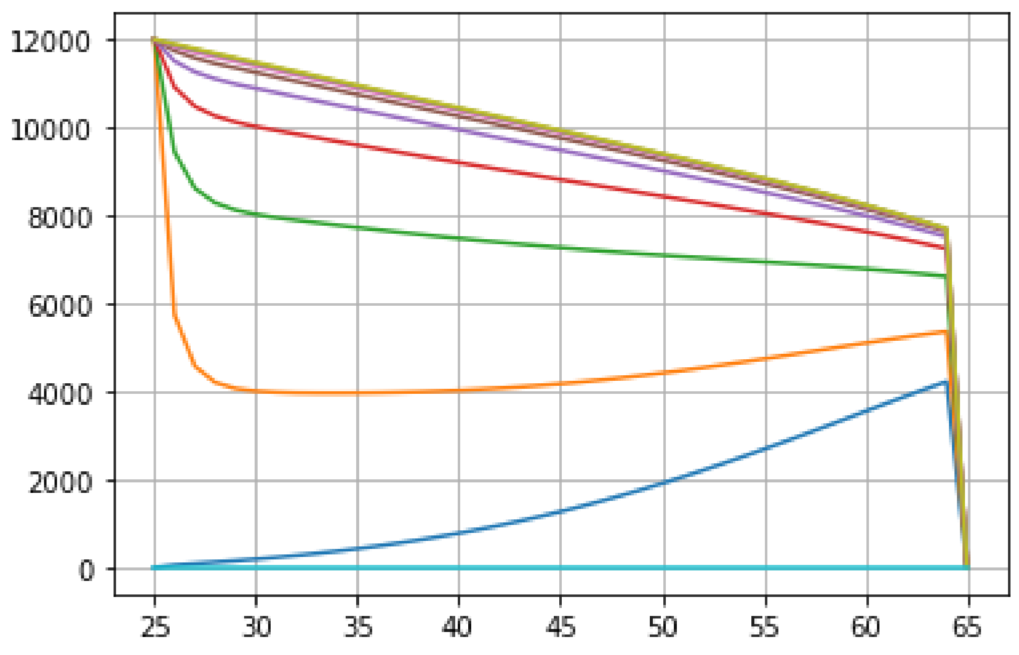
```
iStart = s
iStop = x0

symM = Markov()
symM.vDefineModel(8+2)
symM.vSetDiscount(iRate)

for i in range(iStop,iStart):
    symM.vSetPij(i,0,0, 1. - sigma(i) - mua(i))
    symM.vSetPij(i,0,1, sigma(i))
    symM.vSetPij(i,0,9, mua(i))
    for j in range(8):
        if j<7:
            symM.vSetPij(i,1+j,1+j+1, 1.- mui(i)- rx(i,j))
            symM.vSetPij(i,1+j,0, rx(i,j))
        else:
            symM.vSetPij(i,8,8, 1.- mui(i))

        symM.vSetPij(i,1+j,9, mui(i))
        symM.vSetPre(i,1+j,1+j,Annuity)


symMOpt = Markov()
symMOpt.vDefineModel(8+2)
symMOpt.vSetDiscount(iRate)

for i in range(iStop,iStart):
    symMOpt.vSetPij(i,0,0, 1. - sigma(i) - mua(i))
    symMOpt.vSetPij(i,0,1, sigma(i))
    symMOpt.vSetPij(i,0,9, mua(i))
    symMOpt.vSetPre(i,0,0,0)
    for j in range(8):
        if j<7:
            symMOpt.vSetPij(i,1+j,1+j+1, 1.- mui(i)- RedFromInval*rx(i,j))
            symMOpt.vSetPij(i,1+j,0, RedFromInval*rx(i,j))
        else:
            symMOpt.vSetPij(i,8,8, 1.- mui(i))

        symMOpt.vSetPij(i,1+j,9, mui(i))

        symMOpt.vSetPre(i,1+j,1+j,Annuity+CostDisab)

symMOpta = Markov()
symMOpta.vDefineModel(8+2)
symMOpta.vSetDiscount(iRate)

for i in range(iStop,iStart):
    symMOpta.vSetPij(i,0,0, 1. - sigma(i)*(1-RedToInval) - mua(i))
    symMOpta.vSetPij(i,0,1, sigma(i)*(1-RedToInval))
    symMOpta.vSetPij(i,0,9, mua(i))
    symMOpta.vSetPre(i,0,0,CostActive)
    for j in range(8):
        if j<7:
            symMOpta.vSetPij(i,1+j,1+j+1, 1.- mui(i)- rx(i,j))
```

```
                symMOpta.vSetPij(i,1+j,0, rx(i,j))
            else:
                symMOpta.vSetPij(i,8,8, 1.- mui(i))

            symMOpta.vSetPij(i,1+j,9, mui(i))

            symMOpta.vSetPre(i,1+j,1+j,Annuity)


symMOpt2 = Markov()
symMOpt2.vDefineModel(8+2)
symMOpt2.vSetDiscount(iRate)

for i in range(iStop,iStart):
    if i < 45:
        symMOpt2.vSetPij(i,0,0, 1. - sigma(i)*RedActive40 - mua(i))
        symMOpt2.vSetPij(i,0,1, sigma(i)*RedActive40)
        symMOpt2.vSetPij(i,0,9, mua(i))
    else:
        symMOpt2.vSetPij(i,0,0, 1. - sigma(i) - mua(i))
        symMOpt2.vSetPij(i,0,1, sigma(i))
        symMOpt2.vSetPij(i,0,9, mua(i))

    symMOpt2.vSetPre(i,0,0,CostActive40)
    for j in range(8):
        if j<7:
            symMOpt2.vSetPij(i,1+j,1+j+1, 1.- mui(i)- rx(i,j))
            symMOpt2.vSetPij(i,1+j,0, rx(i,j))
        else:
            symMOpt2.vSetPij(i,8,8, 1.- mui(i))

        symMOpt2.vSetPij(i,1+j,9, mui(i))

        symMOpt2.vSetPre(i,1+j,1+j,Annuity)




x = []
y = []

for i in range(iStop,iStart):
    x.append(i)
    y.append(symM.dGetDK(iStart,iStop,i,0))

plt.figure(1)
plt.plot(x,y)
plt.grid(True)

plt.figure(2)
for i in range(iStop,iStart):
    x = []
    y = []
    for j in range(i,iStart):
        x.append(j)
        state = min(8,1+(j-i))
        y.append(symM.dGetDK(iStart,iStop,j,state))
        plt.plot(x,y)
plt.grid(True)
```





[5]:
```
symM.PlotCFs(iStart,iStop)
```



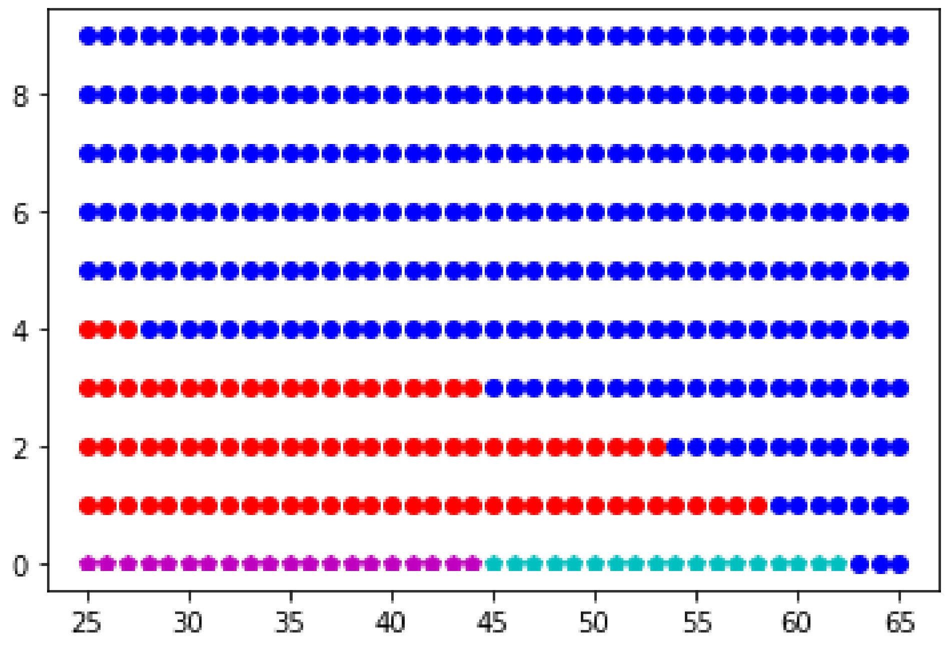
[6]:
```
import os
class MarkovOpt:
    def __init__(self,symMSet):
        self.psymModels =symMSet
        self.iNrStates = None
        self.iMaxTime  = None
        self.dPij = [] # for each time a matrix ie dPij[k] matrix at time␣
 ↪k
        self.dPre = [] # Vector vector of annuities at time t
        self.dPost= []
        self.dv   = []
        # Outputs
        self.dDK  = []
        self.dCF  = []
        self.iOptStrategy = []
        self.bCalculated = False
        self.bCFCalculated = False
        self.iStart = None
        self.iStop  = None
        bCheckConsistency = True
        for i in range(len(symMSet)):
            if symMSet[0].iNrStates != symMSet[i].iNrStates:␣
 ↪bCheckConsistency = False
        if not bCheckConsistency:
            print("Consistency Error -> abort")
            os.abort()
        self.vDefineModel(self.psymModels[0].iNrStates,self.psymModels[0].
 ↪iMaxTime)
        self.bMin = True # This is not implemented
        self.fStart = 1.e20

    def vDefineModel(self,iNrStates,iMaxTime=1200):
        self.iNrStates = iNrStates
        self.iMaxTime = iMaxTime
        for i in range(iMaxTime):
            tempPij = np.zeros([iNrStates,iNrStates])
            tempPost = np.zeros([iNrStates,iNrStates])
            tempPre = np.zeros([iNrStates])
            tempDK = np.zeros([iNrStates])
            tempCF = np.zeros([iNrStates])
            tempStrat = np.zeros([iNrStates],dtype=np.int16)
            self.dPij.append(tempPij)
            self.dPost.append(tempPost)
            self.dPre.append(tempPre)
            self.dDK.append(tempDK)
            self.dCF.append(tempCF)
            self.iOptStrategy.append(tempStrat)
        tempv = np.zeros([iMaxTime])
        self.dv=tempv


    def dGetDK(self,iStart,iStop,iAge,iState):
        if (iStart != self.iStart or iStop != self.iStop or not(self.
↪bCalculated)):
            self.doCalculateDK(iStart,iStop,iAge,iState)
        return(self.dDK[iAge][iState], self.iOptStrategy[iAge][iState])

    def doCalculateDK(self,iStart,iStop,iAge,iState, myMin=-1.e-20,␣
↪bChatter = False):
        self.iStop = iStop
        self.iStart = iStart
        self.bCalculated = True
        for i in range(self.iMaxTime):
            self.dDK[i] *= 0.
            self.iOptStrategy[i] *=0

        NrStrategies = len(self.psymModels)

        for i in range(self.iStart-1, self.iStop-1,-1):
            if bChatter:
                print("===== Recursion t =", i, " ================== ")
            for j in range(self.iNrStates):
                if bChatter:
                    print("--> Analysing State", j, "\n   ␣
↪**********************")
                self.dDK[i][j] = self.fStart
                for l in range(NrStrategies):
                    CandDK = self.psymModels[l].dPre[i][j]
                    for k in range(self.iNrStates):
                        CandDK += self.psymModels[l].dv[i]*self.
↪psymModels[l].dPij[i][j,k]*(self.psymModels[l].dPost[i][j,k]+self.
↪dDK[i+1][k])
                    if CandDK < self.dDK[i][j]:
                        self.dDK[i][j] = CandDK
                        self.iOptStrategy[i][j] = l
                    if bChatter:
                        print("     Strategy %d  Value %10.2f ␣
↪OptStartegy %d  ValueOpt %10.2f"%(l, CandDK, self.iOptStrategy[i][j],␣
↪self.dDK[i][j] ))
```

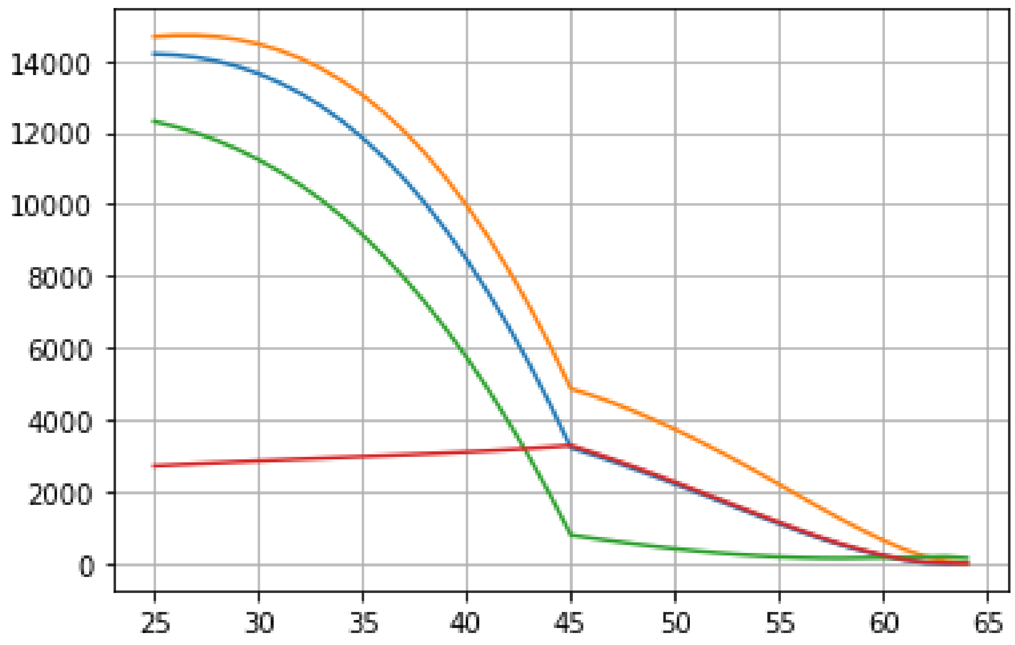
```
    def PrintDKs(self,iStart,iStop,iMaxStates=200):
        for i in range(iStop,iStart+1):
            strTemp = " %3d :"%(i)
            for j in range(min(iMaxStates,self.iNrStates)):
                 strTemp += "  %9.2f [%2d]"%(self.
↪dGetDK(iStart,iStop,i,j))
            print(strTemp)

    def PlotStrategies(self,iStart,iStop):
        Sign = ["bo","ro","cp","mp","yo"]
        plt.figure(1)
        for i in range(iStop,iStart+1):
            for j in range(self.iNrStates):
                x=[i]
                y=[j]
                alpha, beta = self.dGetDK(iStart,iStop,i,j)
                plt.plot(x,y,Sign[int(beta)])

    def PlotDeltaDK(self,iStart,iStop,iState=0,iFigNr=2):
        plt.figure(iFigNr)
        for l in range(len(self.psymModels)):
            x=[]
            y=[]
            for i in range(iStop,iStart):
                x.append(i)
                y.append(self.psymModels[l].
↪dGetDK(iStart,iStop,i,iState)-self.dGetDK(iStart,iStop,i,iState)[0])
            plt.plot(x,y)
        plt.grid(True)

    def ReturnOptimalStrategy(self):
        psymOpt = copy.deepcopy(self.psymModels[0])
        for i in range(self.iStart-1, self.iStop-1,-1):
            for j in range(self.iNrStates):
                lOpt = self.iOptStrategy[i][j]
                psymOpt.vSetPre(i,j,0,self.psymModels[lOpt].dPre[i][j])
                for k in range(self.iNrStates):
                    psymOpt.vSetPij(i,j,k,self.psymModels[lOpt].
↪dPij[i][j,k])
                    psymOpt.vSetPost(i,j,k,self.psymModels[lOpt].
↪dPost[i][j,k])
        return(psymOpt)
```

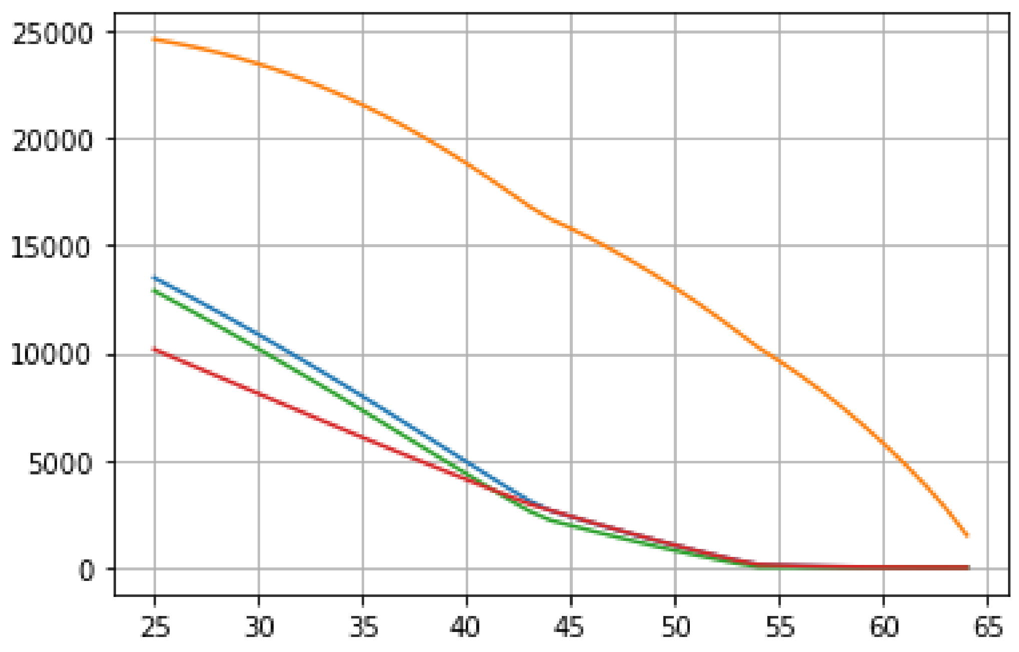
```
[7]: MaOpt = MarkovOpt([symM,symMOpt,symMOpta,symMOpt2])
     MaOpt.PlotStrategies(iStart,iStop)
     MaOpt.PlotDeltaDK(iStart,iStop,iState=0)
     MaOpt.PlotDeltaDK(iStart,iStop,iState=2,iFigNr=3)
     MaOpt.PrintDKs(iStart,iStop,iMaxStates=5)
     psymNewMA = MaOpt.ReturnOptimalStrategy()
     print("====")
     #print(repr(psymNewMA))
     MaOpt2 = MarkovOpt([symM])
     print("----")
     MaOpt2.PrintDKs(iStart,iStop,iMaxStates=5)
```

```
  25 :   21686.11 [ 3]  100375.97 [ 1]  200530.20 [ 1]  251190.72 [ 1]  272466.69 [ 1]
  26 :   22021.45 [ 3]  102330.36 [ 1]  199564.16 [ 1]  248473.73 [ 1]  268931.38 [ 1]
  27 :   22349.90 [ 3]  104195.05 [ 1]  198474.96 [ 1]  245591.54 [ 1]  265284.70 [ 1]
  28 :   22670.25 [ 3]  105963.12 [ 1]  197259.38 [ 1]  242587.86 [ 1]  261468.53 [ 0]
  29 :   22981.30 [ 3]  107627.47 [ 1]  195912.86 [ 1]  239460.95 [ 1]  257533.60 [ 0]
  30 :   23281.84 [ 3]  109180.78 [ 1]  194430.76 [ 1]  236207.30 [ 1]  253478.77 [ 0]
  31 :   23570.67 [ 3]  110615.51 [ 1]  192808.32 [ 1]  232823.35 [ 1]  249300.92 [ 0]
  32 :   23846.62 [ 3]  111923.93 [ 1]  191040.69 [ 1]  229305.41 [ 1]  244996.84 [ 0]
  33 :   24108.58 [ 3]  113098.10 [ 1]  189122.91 [ 1]  225649.76 [ 1]  240563.23 [ 0]
  34 :   24355.46 [ 3]  114129.85 [ 1]  187049.90 [ 1]  221852.54 [ 1]  235996.73 [ 0]
  35 :   24586.28 [ 3]  115010.81 [ 1]  184816.48 [ 1]  217909.82 [ 1]  231293.86 [ 0]
  36 :   24800.12 [ 3]  115732.37 [ 1]  182417.34 [ 1]  213817.57 [ 1]  226451.07 [ 0]
  37 :   24996.19 [ 3]  116285.69 [ 1]  179847.04 [ 1]  209571.64 [ 1]  221464.69 [ 0]
  38 :   25173.82 [ 3]  116661.69 [ 1]  177100.02 [ 1]  205167.77 [ 1]  216330.93 [ 0]
  39 :   25332.50 [ 3]  116850.55 [ 1]  174170.56 [ 1]  200601.59 [ 1]  211045.89 [ 0]
  40 :   25471.91 [ 3]  116841.84 [ 1]  171051.91 [ 1]  195868.57 [ 1]  205605.55 [ 0]
  41 :   25591.94 [ 3]  116623.53 [ 1]  167736.60 [ 1]  190962.94 [ 1]  200005.72 [ 0]
  42 :   25692.76 [ 3]  116179.61 [ 1]  164214.71 [ 1]  185878.11 [ 1]  194240.84 [ 0]
  43 :   25775.20 [ 3]  115430.64 [ 1]  160469.89 [ 1]  180604.47 [ 1]  188304.43 [ 0]
  44 :   25840.60 [ 3]  114344.44 [ 1]  156379.04 [ 1]  175124.70 [ 1]  182186.71 [ 0]
  45 :   25892.61 [ 2]  112607.79 [ 1]  151906.40 [ 1]  169291.44 [ 0]  175869.31 [ 0]
  46 :   24824.10 [ 2]  110607.88 [ 1]  147207.79 [ 1]  163260.52 [ 0]  169368.75 [ 0]
  47 :   23673.19 [ 2]  108335.83 [ 1]  142277.09 [ 1]  157029.10 [ 0]  162679.70 [ 0]
  48 :   22441.72 [ 2]  105782.48 [ 1]  137107.87 [ 1]  150591.37 [ 0]  155796.51 [ 0]
  49 :   21132.48 [ 2]  102938.28 [ 1]  131693.33 [ 1]  143941.17 [ 0]  148713.16 [ 0]
  50 :   19749.33 [ 2]   99793.20 [ 1]  126026.29 [ 1]  137071.94 [ 0]  141423.20 [ 0]
  51 :   18297.34 [ 2]   96336.71 [ 1]  120099.03 [ 1]  129976.58 [ 0]  133919.66 [ 0]
  52 :   16782.94 [ 2]   92557.75 [ 1]  113903.35 [ 1]  122647.48 [ 0]  126195.04 [ 0]
  53 :   15214.73 [ 2]   88422.67 [ 1]  107430.46 [ 1]  115076.34 [ 0]  118241.13 [ 0]
  54 :   13605.47 [ 2]   83833.94 [ 1]  100639.19 [ 0]  107253.98 [ 0]  110048.90 [ 0]
  55 :   11967.64 [ 2]   78883.41 [ 1]   93403.46 [ 0]   99170.21 [ 0]  101608.37 [ 0]
  56 :   10316.22 [ 2]   73555.05 [ 1]   85863.52 [ 0]   90813.76 [ 0]   92908.46 [ 0]
  57 :    8669.13 [ 2]   67831.01 [ 1]   78005.79 [ 0]   82171.98 [ 0]   83936.81 [ 0]
  58 :    7047.70 [ 2]   61691.45 [ 1]   69815.16 [ 0]   73230.75 [ 0]   74679.53 [ 0]
  59 :    5477.71 [ 2]   55101.74 [ 0]   61274.86 [ 0]   63974.21 [ 0]   65120.99 [ 0]
  60 :    4000.85 [ 2]   47718.85 [ 0]   52359.65 [ 0]   54384.07 [ 0]   55243.31 [ 0]
  61 :    2655.12 [ 2]   39832.02 [ 0]   43039.15 [ 0]   44433.56 [ 0]   45026.14 [ 0]
  62 :    1487.14 [ 2]   31368.78 [ 0]   33268.83 [ 0]   34090.51 [ 0]   34439.93 [ 0]
  63 :     556.11 [ 0]   22194.22 [ 0]   22973.60 [ 0]   23307.32 [ 0]   23449.11 [ 0]
  64 :       0.00 [ 0]   12000.00 [ 0]   12000.00 [ 0]   12000.00 [ 0]   12000.00 [ 0]
  65 :       0.00 [ 0]       0.00 [ 0]       0.00 [ 0]       0.00 [ 0]   0.00 [ 0]
------------------------------
  25 :   35869.64 [ 0]  129923.27 [ 0]  214019.47 [ 0]  255421.42 [ 0]   273282.54 [ 0]
  26 :   36178.37 [ 0]  131082.08 [ 0]  212545.99 [ 0]  252470.31 [ 0]   269671.82 [ 0]
  27 :   36442.34 [ 0]  132111.27 [ 0]  210939.63 [ 0]  249398.95 [ 0]   265948.29 [ 0]
  28 :   36657.27 [ 0]  133003.92 [ 0]  209196.02 [ 0]  246204.00 [ 0]   262109.01 [ 0]
  29 :   36818.87 [ 0]  133753.10 [ 0]  207310.81 [ 0]  242882.10 [ 0]   258150.98 [ 0]
  30 :   36922.89 [ 0]  134351.91 [ 0]  205279.61 [ 0]  239429.83 [ 0]   254071.16 [ 0]
  31 :   36965.13 [ 0]  134793.46 [ 0]  203098.04 [ 0]  235843.73 [ 0]   249866.43 [ 0]
  32 :   36941.48 [ 0]  135070.91 [ 0]  200761.70 [ 0]  232120.30 [ 0]   245533.63 [ 0]
  33 :   36847.94 [ 0]  135177.44 [ 0]  198266.20 [ 0]  228255.99 [ 0]   241069.50 [ 0]
  34 :   36680.63 [ 0]  135106.28 [ 0]  195607.12 [ 0]  224247.21 [ 0]   236470.74 [ 0]
  35 :   36435.85 [ 0]  134850.71 [ 0]  192780.04 [ 0]  220090.30 [ 0]   231733.96 [ 0]
  36 :   36110.12 [ 0]  134404.05 [ 0]  189780.50 [ 0]  215781.53 [ 0]   226855.68 [ 0]
  37 :   35700.17 [ 0]  133759.64 [ 0]  186604.04 [ 0]  211317.12 [ 0]   221832.35 [ 0]
  38 :   35203.04 [ 0]  132910.90 [ 0]  183246.17 [ 0]  206693.20 [ 0]   216660.30 [ 0]
  39 :   34616.06 [ 0]  131851.25 [ 0]  179702.33 [ 0]  201905.83 [ 0]   211335.79 [ 0]
  40 :   33936.94 [ 0]  130574.12 [ 0]  175967.94 [ 0]  196950.96 [ 0]   205854.91 [ 0]
  41 :   33163.82 [ 0]  129072.96 [ 0]  172038.32 [ 0]  191824.43 [ 0]   200213.66 [ 0]
  42 :   32295.31 [ 0]  127341.19 [ 0]  167908.73 [ 0]  186521.94 [ 0]   194407.88 [ 0]
  43 :   31330.54 [ 0]  125372.20 [ 0]  163574.31 [ 0]  181039.06 [ 0]   188433.24 [ 0]
  44 :   30269.27 [ 0]  123159.27 [ 0]  159030.05 [ 0]  175371.18 [ 0]  182285.21 [ 0]
  45 :   29111.92 [ 0]  120695.60 [ 0]  154270.79 [ 0]  169513.49 [ 0]   175959.07 [ 0]
  46 :   27859.73 [ 0]  117974.21 [ 0]  149291.18 [ 0]  163460.95 [ 0]   169449.83 [ 0]
  47 :   26514.76 [ 0]  114987.92 [ 0]  144085.60 [ 0]  157208.25 [ 0]   162752.25 [ 0]
  48 :   25080.12 [ 0]  111729.25 [ 0]  138648.14 [ 0]  150749.78 [ 0]   155860.73 [ 0]
  49 :   23560.01 [ 0]  108190.37 [ 0]  132972.54 [ 0]  144079.54 [ 0]   148769.32 [ 0]
  50 :   21959.95 [ 0]  104362.99 [ 0]  127052.11 [ 0]  137191.10 [ 0]   141471.63 [ 0]
  51 :   20286.93 [ 0]  100238.23 [ 0]  120879.64 [ 0]  130077.56 [ 0]   133960.77 [ 0]
  52 :   18549.64 [ 0]   95806.51 [ 0]  114447.30 [ 0]  122731.39 [ 0]   126229.26 [ 0]
  53 :   16758.71 [ 0]   91057.40 [ 0]  107746.54 [ 0]  115144.39 [ 0]   118268.93 [ 0]
  54 :   14927.07 [ 0]   85979.35 [ 0]  100767.91 [ 0]  107307.52 [ 0]   110070.82 [ 0]
  55 :   13070.28 [ 0]   80559.52 [ 0]   93500.92 [ 0]   99210.79 [ 0]   101625.02 [ 0]
```

```
56 :   11207.00 [ 0]   74783.49 [ 0]   85933.82 [ 0]   90843.06 [ 0]   92920.52 [ 0]
57 :    9359.54 [ 0]   68634.89 [ 0]   78053.37 [ 0]   82191.83 [ 0]   83945.00 [ 0]
58 :    7554.53 [ 0]   62095.06 [ 0]   69844.56 [ 0]   73243.02 [ 0]   74684.61 [ 0]
59 :    5823.89 [ 0]   55138.21 [ 0]   61290.29 [ 0]   63980.65 [ 0]   65123.66 [ 0]
60 :    4206.03 [ 0]   47733.04 [ 0]   52365.62 [ 0]   54386.56 [ 0]   55244.34 [ 0]
61 :    2747.56 [ 0]   39834.57 [ 0]   43040.22 [ 0]   44434.01 [ 0]   45026.32 [ 0]
62 :    1506.19 [ 0]   31368.78 [ 0]   33268.83 [ 0]   34090.51 [ 0]   34439.93 [ 0]
63 :     556.11 [ 0]   22194.22 [ 0]   22973.60 [ 0]   23307.32 [ 0]   23449.11 [ 0]
64 :       0.00 [ 0]   12000.00 [ 0]   12000.00 [ 0]   12000.00 [ 0]   12000.00 [ 0]
65 :       0.00 [ 0]       0.00 [ 0]       0.00 [ 0]       0.00 [ 0]   0.00 [ 0]
```






25000
20000
15000
10000
5000
0
25
30
35
40
45
50
55
60
65