TarEngV020

March 4, 2025

Tarif Engine

Classes

TarEng

Demogr

Economics

Tariff

General Parameters (configurable via bDefaultsInput)

t0: currrent year (default - current year)

g: gender (0 - man; 1 - woman - default 0)

x: current age

b: start age of benefits (default b=x)

s: end age benefits (default - omega)

Other Defaults: ATar, QxCH, iTech

GTar

Inputs:

fB: Death Benefit Level

fM: Maturity Benefit

ATar

Inputs:

fR: Annuity Benefit Start

fDeltaR: Linear increase relative to fR

iNrIncreases: Nr of Benefit Increases

Read Me

The library TarLib aims to provide all necessary functionality to calculate cash flows and mathematical reserves for a variety of life insurance products, in particular capital insurance on one life and annuities on one and two lives. The whole librariy is written in a abstract way such that it is easy to amend it for other types of prodcuts.

The library consists of the following classes:

- **TarEng** This is the top class and the main aim is to Dispatch the requests to the respective modules and to provide the resuylts in a structured way. The inputs are provided via the function ParseTask, where a semicolon reparated string is given and processed and the input being written on self->pInp and the output to self->pOut. For the documentation of the member variables see the constructor. Remark, that all Input and Output is encapsulated in the MyIO Class.
- **MyIo** This is an abstraction of input and output including reset and printing itself
- **Markov** Markov Life Insurance Class for implementation of actual Tariffs
- **ATar** One Life Annuity. Output Reserves and Cash Flows; uses classical approach
- **A2Tar** Ditto 2 life Annuity; uses Markov Class
- **GTar** Ditto Capital Insurance; uses classical approach

Implemented Tariffs/Demoraphics and Economies

self.symDirTariffs = {“A”:ATar,“A2”:A2Tar,“G”:GTar,“A..”:ATar,“AR.”:ATar,“A.E”:ATar,“ARE”:ATar,“A.L”:AT

self.symDirDemogr = {“CH”:QxCH}

self.symDirEco ={“iTech”:ITech}

Parameters ATar

- **Gender 0/1** self.gender = int(self.psymParent.pInp.Param[“g”])
- **Current Age** self.x = int(self.psymParent.pInp.Param[“x”])
- **Start Age Benefit Payment** self.b = int(self.psymParent.pInp.Param[“b”])
- **End Age Benefit Payment+1** self.s = int(self.psymParent.pInp.Param[“s”])
- **Benefit Level** self.fLevel = float(self.psymParent.pInp.Param[“fLevel”])
- Tokens=[“iNrIncreases”,“alpha”,“fR”,“iExp”]: **See Implementation** fR level Annuity

Parameters GTar - **Gender 0/1** self.gender = int(self.psymParent.pInp.Param[“g”]) - **Current Age** self.x = int(self.psymParent.pInp.Param[“x”]) - **Start Age Benefit Payment** self.b = int(self.psymParent.pInp.Param[“b”]) - **End Age Benefit Payment+1** self.s = int(self.psymParent.pInp.Param[“s”]) - **Benefit Level** self.fLevel = float(self.psymParent.pInp.Param[“fLevel”]) - **Maturity Benefit at age s** self.gender = int(self.psymParent.pInp.Param[“fM”]) - **Death Benefit between b and s** self.gender = int(self.psymParent.pInp.Param[“fB”])

Parameters A2Tar

Same as per above mutatis mutandis. Some notes: the ages are in respect to the age of person number one and all Benefits indicated “1” (and “2” respectively refer to the state where the indicated person is alive and the partner death. The symbol “12” referrs to both alive and dead(jopint life status). Note the age of person “2” in respect to person “1” is given by deltaxy with $\Delta_{xy} = y - x$

- self.x = int(self.psymParent.pInp.Param[“x”])
- self.s = int(self.psymParent.pInp.Param[“s”])
- self.b = int(self.psymParent.pInp.Param[“b”])
- self.fLevel = float(self.psymParent.pInp.Param[“fLevel”])
- self.gender1 = int(self.psymParent.pInp.Param[“g”])
- self.gender2 = int(self.psymParent.pInp.Param[“g2”])
- self.deltaxy = int(self.psymParent.pInp.Param[“deltaxy”])

- Tokens=[“fM12”,“fM1”,“fM2”,“fR1”,“fR2”,“fR12”,“fB1”,“fB2”,“fB12”]

Examples

Whole of life staring age 20 dfault interest Rate a = TarEng()

```
a.ParseTask("G;iTech;g=0;x=20;s=110;fB=0;fM=1000000")
```

```
a.vPlot()
```

Deferred Annuity

Current Age 55, start payout 60, end payment at 89, Annuity of 12000

```
strTask = "A;iTech;g=0;x=55;b=60;s=90;fR=12000"
```

```
a.ParseTask(strTask)
```

```
a.vPlot()
```

Widows pension 12000 with woman 3 years younger

```
strTask = "A2;iTech;g=0;g2=1;deltaxy=3;x=20;s=120;fR2=12000"
```

```
a.ParseTask(strTask)
```

```
a.vPlot()
```

Example Output See below

[170]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import datetime

bExaMode = True

class MyIO:
    def __init__(self, strName):
        self.me = strName

    def getvariablenames(self):
        members = [attr for attr in dir(self) if not \
                 callable(getattr(self, attr)) and not attr.startswith("__")]
        return(members)

    def clean(self):
        strName = self.me
        members = self.getvariablenames()
        for i in members:
            strTask = "self."+i+"= None"
            exec(strTask)
```

```
        self.me = strName

    def print(self):
        instance_vars = vars(self)
        for v_name, v_value in instance_vars.items():
            print("%s --> %s"%(repr(v_name),repr(v_value)))

    def vstrPrint(self):
        instance_vars = vars(self)
        strOut = "Trace %s \n------------ \n"%(self.me)
        for v_name, v_value in instance_vars.items():
            strOut+= "%s --> %s \n"%(repr(v_name),repr(v_value))
        strOut +="------------ \n"
        return(strOut)

class TarEng():
    def __init__(self,i=0.025, t0= 2025, omega=120,nCF=100,bDefaultsInput =␣
 ↪True,bLevel=False,bBatch=False,bTrace=False):
        self.strVersion = "Tariff Engine V0.20 M Koller"
        # Tarif dictory stringName:ClassReference. All available Tarif Routines
        self.symDirTariffs = dict()
        # Demographics dictory stringName:ClassReference. All available␣
 ↪MOrtality Rates
        # Currently only one Qx Table
        self.symDirDemogr = dict()
        # Economics dictory stringName:ClassReference. All available Economics
        self.symDirEco = dict()
        # Input Class
        self.pInp = MyIO("Input")
        # Output Class
        self.pOut = MyIO("Output")
        # Trace Class
        self.pTrace = MyIO("Trace")
        # What now follows are Global Parameter Class and initialisation
        self.pGPar = MyIO("GlobalParameter")
        self.pGPar.iTech = i # Technical Interest Ratse
        self.pGPar.vTech = 1./(1.+i) # Corresponding Discount
        self.pGPar.t0 = t0 # Start Year for generational table
        self.pGPar.omega=omega # Latest Age Omega
        self.pGPar.nCF=nCF # Number of Cash FLows to be calculated
        # Here we add all available Traif Modules to self.symDirTariffs
        self.AddModules()
        # Default Behavior and resetting I/O
        self.bDefaultsInput = bDefaultsInput
        self.pInp.Tarif = None
        self.pInp.Econ = None
        self.pInp.Demo = None
```

```
        self.bLevel = bLevel
        self.bBatch = bBatch
        self.bTrace = bTrace
        self.pInp.Param = dict()


    def AddModules(self):
        self.symDirTariffs = {"A":ATar,"A2":A2Tar,"G":GTar,\
                              "A..":ATar,"AR.":ATar,"A.E":ATar,"ARE":ATar,"A.L":
↪ATar,"ARL":ATar,\
                              "ART":ATar,"G..":GTar,"F..":GTar,\
                              "T..":GTar,"L..":GTar,"W..":A2Tar}
        self.symDirDemogr = {"CH":QxCH}
        self.symDirEco ={"iTech":ITech}

    def vPostProcess(self):
        Mapper={"A..":self.AMod,"AR.":self.AMod,"A.E":self.AMod,"ARE":self.
↪AMod, "A.L":self.AMod,"ARL":self.AMod,\
                "ART":self.AMod,"G..":self.GMod,"F..":self.GMod,\
                "T..":self.GMod,"L..":self.GMod,"W..":self.WMod}
        if self.pInp.Tarif in Mapper.keys():
            Mapper[self.pInp.Tarif]()
        return()

    def AMod(self):
        if self.pInp.Tarif == "AR." or self.pInp.Tarif == "A.." or self.pInp.
↪Tarif == "ART":
            self.pInp.Param["alpha"] = 0.
            self.pInp.Param["iNrIncreases"] = 0
            self.pInp.Param["iExp"] = 0
        if self.pInp.Tarif == "AR." or self.pInp.Tarif == "ARE" or self.pInp.
↪Tarif == "ARL" or self.pInp.Tarif == "ART":
            self.pInp.Param["b"] = self.pInp.Param["x"]
        if self.pInp.Tarif == "ARL" or self.pInp.Tarif == "A.L":
            self.pInp.Param["iExp"] = 0
            self.pInp.Param["iNrIncreases"] = 1000
        if self.pInp.Tarif == "ARE" or self.pInp.Tarif == "A.E":
            self.pInp.Param["iExp"] = 1
            self.pInp.Param["iNrIncreases"] = 1000
        if self.pInp.Tarif != "ART":
               self.pInp.Param["s"] = self.pGPar.omega

    def GMod(self):
        self.pInp.Param["b"] = self.pInp.Param["x"]
        if self.pInp.Tarif == "G..":
            self.pInp.Param["fM"] = 1
            self.pInp.Param["fB"] = 1
```

```
        if self.pInp.Tarif == "L..":
            self.pInp.Param["fM"] = 1
            self.pInp.Param["fB"] = 1
            self.pInp.Param["s"] = self.pGPar.omega
        if self.pInp.Tarif == "T..":
            self.pInp.Param["fM"] = 0
            self.pInp.Param["fB"] = 1
        if self.pInp.Tarif == "F..":
            self.pInp.Param["fM"] = 1
            self.pInp.Param["fB"] = 0

    def WMod(self):
        self.pInp.Param["s"] = self.pGPar.omega
        self.pInp.Param["b"] = self.pInp.Param["x"]
        Tokens=["fM12","fM1","fM2","fR1","fR12","fB1","fB2","fB12"]
        for i in Tokens:
            self.pInp.Param[i] = 0
        self.pInp.Param["fR2"] = 1
        self.pInp.Param["g2"] = 1-int(self.pInp.Param["g"])


    def ParseTask(self, strInput,strSep=";",bPrintOut=True):
        if self.bBatch: bPrintOut=False
        #print(self.pInp.Tarif)
        #strInput = self.vPreProcess(strInput)
        Tokens=strInput.split(strSep)
        self.pInp.strInp = strInput
        #print(self.pInp.Tarif)
        for i in Tokens:
            #print(self.pInp.Tarif)
            SubTok = i.split("=")
            if len(SubTok) == 1:
                if SubTok[0] in self.symDirTariffs.keys():
                    self.pInp.Tarif = SubTok[0]
                if SubTok[0] in self.symDirDemogr.keys():
                    self.pInp.Demo = SubTok[0]
                if SubTok[0] in self.symDirEco.keys():
                    self.pInp.Econ = SubTok[0]
            else:
                self.pInp.Param[SubTok[0]] = SubTok[1]
        #print(self.pInp.Tarif)
        if self.bDefaultsInput:
            #if self.pInp.Tarif == None:
            #    self.pInp.Tarif = "A"
            if self.pInp.Demo == None:
                self.pInp.Demo = "CH"
            if self.pInp.Econ == None:
```

```
                self.pInp.Econ = "iTech"
            self.vDefaultMapper()


        # init objects
         self.psymE = self.symDirEco[self.pInp.Econ](self)
         self.psymD = self.symDirDemogr[self.pInp.Demo](self)
         self.psymT = self.symDirTariffs[self.pInp.Tarif](self)

         self.vPostProcess()

       # do Stuff
         if not self.bLevel:
             self.pInp.Param["fLevel"] = 1.
         #else:
         #    print("Task:",strInput)
         #    print(repr(self.pInp.Param))

         self.psymE.vDoCalc()
         self.psymD.vDoCalc()
         self.psymT.vDoCalc()

       # print
         if bPrintOut: self.vPrint()
         if self.bTrace: self.TraceIO()


    def vDefaultMapper(self):
        strOut = ""
        today = datetime.date.today()
        self.pGPar.t0 = today.year
        dDefaults={"x":15,"g":0,"g2":1,"deltaxy":-3,"fM":1,"fB":1}
        for i in dDefaults.keys():
            if i not in self.pInp.Param.keys():
                self.pInp.Param[i] = dDefaults[i]
        strOut +="Set year to:"+str(self.pGPar.t0)+"\n"
        if "b" not in self.pInp.Param.keys():
            self.pInp.Param["b"] = self.pInp.Param["x"]
            strOut +="Overrule b (missing) \n"
        if "s" not in self.pInp.Param.keys():
            self.pInp.Param["s"] = self.pGPar.omega
            strOut +="Overrule s (missing) \n"

        self.pTrace.vDefaultMapperMsg  = strOut

    def strPrint(self):
        strStruct2a = "%7s: %20s %20s \n"
```

```
        strSep= (len(strStruct2a %("Age","DK/MR","E[CF]"))-1)*"-"+"\n"
        strParam = ["g","g2","x","deltaxy","b","s"]
        strOut = ""
        strOut += strSep
        strOut += self.pInp.Tarif +"\n"
        strOut += strSep
        strStruct1 = "%-15s: %10s \n"
        strStruct2 = "%7d: %20.6f %20.6f \n"

        strOut += strStruct1 %("omega",str(self.pGPar.omega))
        strOut += strStruct1 %("t0",str(self.pGPar.t0))
        strOut += strStruct1 %("iTech",str(self.pGPar.iTech))
        for i in strParam:
            bAvailable = (i in self.pInp.Param.keys())
            if bAvailable:
                strOut += strStruct1 %(i,str(self.pInp.Param[i]))
            else:
                strOut += strStruct1 %(i,"n/a")
        for i in self.pInp.Param.keys():
            bDone = (i in strParam)
            if not bDone:
                 strOut += strStruct1 %(i,str(self.pInp.Param[i]))

        strOut += strSep
        x= int(self.pInp.Param["x"])
        s= int(self.pInp.Param["s"])
        strOut += strStruct2a %("Age","DK/MR","E[CF]")
        strOut += strSep
        for i in range(x,min(self.pGPar.omega,s+1)):
            strOut += strStruct2 %(i,self.pOut.dDK[i],self.pOut.dCF[i-x])
        return(strOut)

    def vPrint(self):
        print(self.strPrint())

    def vPlot(self):
        x=int(self.pInp.Param["x"])
        periods = self.pGPar.nCF
        omega=self.pGPar.omega
        xmax = min(omega,x+periods)
        plt.figure(1)
        plt.plot(range(x,xmax),self.pOut.dDK[x:xmax])
        plt.grid(True)
        plt.figure(2)
        plt.plot(range(0,xmax-x),self.pOut.dCF[0:xmax-x])
        plt.grid(True)
```

```
    def TraceIO(self):
        strOut = ""
        for i in [self.pInp,self.pOut,self.pTrace,self.pGPar]:
            strOut += i.vstrPrint()
        print(strOut)

    def vClearInput(self):
        self.pInp.clean()
        self.pOut.clean()
        self.pTrace.clean()
        self.pInp.Tarif = None
        self.pInp.Econ = None
        self.pInp.Demo = None
        self.pInp.Param = dict()
        self.TraceIO()
```

[172]:
```
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+self.fBucketWidthRound)))
        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")
        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
            for t in range(self.iStop,self.iStart):
                fTot = sum(self.dPij[t][i,:])
                #print(i,t,"-->",fTot)
                if abs(fTot-1.) >= eps:
                    bFound=True
                    self.dPij[t][i,default] += 1. - fTot
            if bFound:
                print("Check P(Omega) = 1 failed for iState=",i,"Target␣
↪State",iState)

    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)
        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
```

[174]:
```
class ATar():
    def __init__(self,psymParent):
        self.psymParent=psymParent

        self.psymB = self.StdBenefit

        self.psymParent.pOut.dDK= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pOut.dDKPer= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pOut.dBenefitLevel= np.zeros(self.psymParent.pGPar.
 ↪omega)
        periods = self.psymParent.pGPar.nCF
        self.psymParent.pOut.dCF = np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pTrace.strExecTasks = []
        self.vUpdateParam()

    def vUpdateParam(self):
        self.nOmega = self.psymParent.pGPar.omega
        self.psymQx = self.psymParent.psymD.dQx
        self.dV = self.psymParent.pGPar.vTech
        self.nT0 = self.psymParent.pGPar.t0

    def StdBenefit(self,x,param=[]):
#iNrPayments: number of annuity payments
#fR: Annuity Benefit Start
#fDeltaR: Linear increase
#iNrIncreases: Nr of Benefit Increases
```

```
        dValue=0
        if x>=self.b and x<self.s:
           if self.iExp == 0:
               dValue = 1 + self.alpha *max(0,min(x-self.b,self.iNrIncreases))
           else:
               dValue = (1 + self.alpha) ** max(0,min(x-self.b,self.
↪iNrIncreases))

        return(dValue*self.fR)

    def CalcPV(self):

        dDK = self.psymParent.pOut.dDK
        dDKPer = self.psymParent.pOut.dDKPer
        gender = self.gender
        x = self.x
        s = self.s
        b = self.b
        fLevel = self.fLevel
        param = self.param
        PV = 0
        dDK[s] = PV
        n  = s-x
        for i in range(s-1,x-1,-1):
            t = self.nT0 + i - x
            qx = self.psymQx(gender,i,t)
            px = 1. - qx
            dBen = self.psymB(i,param=param) * fLevel
            self.psymParent.pOut.dBenefitLevel[i]=dBen
            PV = dBen + px * self.dV * PV # a_x = 1 + p_x v a_{x+1}
            dDK[i] = PV
            dDKPer[i-x] = PV


    def CalcCF(self):
        # Caculation of expected cash flows
        gender = self.gender
        x = self.x
        s = self.s
        b = self.b
        fLevel = self.fLevel
        param = self.param
        periods = self.psymParent.pGPar.nCF
        CF =  self.psymParent.pOut.dCF
        px = 1.
        for i in range(x,s):
            t = self.nT0 + i - x
```

```
            qx = self.psymQx(gender,i,t)
            n = i - x
            if n >= periods:
                break

            dBen = self.psymB(i,param=param) * fLevel
            CF[n] = px  * dBen
            px *= (1-qx)
            #print(i, px, qx)

    def PopulateParam(self):

        self.gender = int(self.psymParent.pInp.Param["g"])
        self.x = int(self.psymParent.pInp.Param["x"])
        self.b = int(self.psymParent.pInp.Param["b"])
        self.s = int(self.psymParent.pInp.Param["s"])
        #print(self.psymParent.pInp.Param)
        self.fLevel = float(self.psymParent.pInp.Param["fLevel"])
        Tokens=["iNrIncreases","alpha","fR","iExp"]
        for i in Tokens:
            if i in self.psymParent.pInp.Param.keys():
                dValue =  float(self.psymParent.pInp.Param[i])
            else:
                if i == "fR":
                    dValue = 1
                else:
                    dValue = 0
            strExec="self."+i+"="+str(dValue)
            self.psymParent.pTrace.strExecTasks.append(strExec)
            exec(strExec)

        self.param=[]

    def vDoCalc(self):
        self.vUpdateParam()
        #print ("Task :",self.psymParent.pInp.Param)
        self.PopulateParam()
        try:
            self.PopulateParam()
        except:
            print("Error Parameter")
            self.psymParent.TraceIO()
            return()
        self.CalcPV()
        self.CalcCF()
```

```
class A2Tar():
    def __init__(self,psymParent):
        self.psymParent=psymParent
        self.psymParent.pOut.dDK= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pOut.dDKPer= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pTrace.dDK12= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pTrace.dDK1= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pTrace.dDK2= np.zeros(self.psymParent.pGPar.omega+1)
        periods = self.psymParent.pGPar.nCF
        self.psymParent.pOut.dCF = np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pTrace.strExecTasks = []

        self.psymB1 = self.ConstantBenefit
        self.psymB2 = self.ConstantBenefit
        self.psymB12 = self.ConstantBenefit
        self.psymR1 = self.ConstantBenefit
        self.psymR2 = self.ConstantBenefit
        self.psymR12 = self.ConstantBenefit
        self.psymM1 = self.ConstantBenefit
        self.psymM2 = self.ConstantBenefit
        self.psymM12 = self.ConstantBenefit

        self.symM=Markov()
        self.symM.vDefineModel(4)
        self.QxLevelJoint = 1.
        self.QxLevelWidow = 1.

    def vUpdateParam(self):
        self.nOmega = self.psymParent.pGPar.omega
        self.psymQx = self.psymParent.psymD.dQx
        self.dV = self.psymParent.pGPar.vTech
        self.nT0 = self.psymParent.pGPar.t0
        self.dIrate = self.psymParent.pGPar.iTech

    def StdBenefit(self,x,param=[]):
       #iNrPayments: number of annuity payments
       #fR: Annuity Benefit Start
       #fDeltaR: Linear increase
       #iNrIncreases: Nr of Benefit Increases
        dValue=0
        if x>=self.b and x<self.s:
           dValue = 1 + self.fDeltaR *min(x-self.b,self.iNrIncreases)
        return(dValue*self.fR)

    def ConstantBenefit(self,x,param=[]):
        if x < self.b: return(0)
        return(1.)
```

```
    def ZeroBenefit(self,x,param=[]):
        return(0.)

    def PopulateParam(self):
        self.x = int(self.psymParent.pInp.Param["x"])
        self.s = int(self.psymParent.pInp.Param["s"])
        self.b = int(self.psymParent.pInp.Param["b"])
        self.fLevel = float(self.psymParent.pInp.Param["fLevel"])

        self.gender1 = int(self.psymParent.pInp.Param["g"])
        self.gender2 = int(self.psymParent.pInp.Param["g2"])
        self.deltaxy = int(self.psymParent.pInp.Param["deltaxy"])

        self.param=[]
        Tokens=["fM12","fM1","fM2","fR1","fR2","fR12","fB1","fB2","fB12"]
        for i in Tokens:
            if i in self.psymParent.pInp.Param.keys():
                dValue =  float(self.psymParent.pInp.Param[i])
            else:
                dValue = 0
            strExec="self."+i+"="+str(dValue)
            self.psymParent.pTrace.strExecTasks.append(strExec)
            exec(strExec)

    def vDoCalc(self):
        self.vUpdateParam()
        #print ("Task :",self.psymParent.pInp.Param)
        try:
            self.PopulateParam()
        except:
            print("Error Parameter")
            self.psymParent.TraceIO()
            return()
        self.CalcPV()
        self.CalcCF()

    def CalcPV(self):
        dDK= self.psymParent.pOut.dDK
        dDKPer = self.psymParent.pOut.dDKPer
        dDK12= self.psymParent.pTrace.dDK12
        dDK1=self.psymParent.pTrace.dDK1
        dDK2=self.psymParent.pTrace.dDK2
        x = self.x
        s = self.s
        fLevel = self.fLevel
        param = self.param
```

```
        n  = s-x
        gender1 = self.gender1
        gender2 = self.gender2
        deltaxy = self.deltaxy
        self.symM.vSetDiscounT(self.dIrate)
        tt=s-1
        self.symM.vSetPost(tt,0,0,self.fM12*self.psymM12(s,param=param)* fLevel)
        self.symM.vSetPost(tt,1,1,self.fM1*self.psymM1(s,param=param)* fLevel)
        self.symM.vSetPost(tt,2,2,self.fM2*self.psymM2(s,param=param)* fLevel)
        for i in range(x,s):
            t = self.nT0 + i - x
            iy = i + deltaxy
            qx = max(0,min(1,self.psymQx(gender1,i,t) * self.QxLevelJoint))
            qy = max(0,min(1,self.psymQx(gender2,iy,t)* self.QxLevelJoint))
            px = 1. - qx
            py = 1. - qy
            qxW = max(0,min(1,self.psymQx(gender1,i,t) * self.QxLevelWidow))
            qyW = max(0,min(1,self.psymQx(gender2,iy,t) * self.QxLevelWidow))
            pxW = 1. - qxW
            pyW = 1. - qyW
            tt = i
            self.symM.vSetPij(tt,0,0,px*py)
            self.symM.vSetPij(tt,0,1,px*qy)
            self.symM.vSetPij(tt,0,2,qx*py)
            self.symM.vSetPij(tt,0,3,qx*qy)
            self.symM.vSetPij(tt,1,1,pxW)
            self.symM.vSetPij(tt,1,3,qxW)
            self.symM.vSetPij(tt,2,2,pyW)
            self.symM.vSetPij(tt,2,3,qyW)
            self.symM.vSetPre(tt,0,0,self.fR12*self.psymR12(i,param=param)*␣
↪fLevel)
            self.symM.vSetPre(tt,1,1,self.fR1*self.psymR1(i,param=param)*␣
↪fLevel)
            self.symM.vSetPre(tt,2,2,self.fR2*self.psymR2(i,param=param)*␣
↪fLevel)
            self.symM.vSetPost(tt,0,1,self.fB12*self.psymB12(i,param=param)*␣
↪fLevel)
            self.symM.vSetPost(tt,0,2,self.fB12*self.psymB12(i,param=param)*␣
↪fLevel)
            self.symM.vSetPost(tt,0,3,self.fB12*self.psymB12(i,param=param)*␣
↪fLevel)
            self.symM.vSetPost(tt,1,3,self.fB1*self.psymB1(i,param=param)*␣
↪fLevel)
            self.symM.vSetPost(tt,2,3,self.fB2*self.psymB2(i,param=param)*␣
↪fLevel)
```

```
        for i in range(x,s):
            dDK1[i] = self.symM.dGetDK(self.nOmega,0,i,1)
            dDK2[i] = self.symM.dGetDK(self.nOmega,0,i,2)
            dDK12[i] = self.symM.dGetDK(self.nOmega,0,i,0)
            dDK[i]=dDK12[i]
            dDKPer[i-x] = dDK12[i]


    def CalcCF(self):
        CF =  self.psymParent.pOut.dCF
        x = self.x
        s = self.s
        fLevel = self.fLevel
        for i in range(x,s):
            CF[i-x]= self.symM.dGetCF(self.nOmega,x,i,0)

class GTar():
    def __init__(self,psymParent):
        self.psymParent=psymParent

        self.psymB = self.ConstantBenefit
        self.psymM = self.ConstantBenefit # We incialise actually A_x

        self.psymParent.pOut.dDK= np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pOut.dDKPer= np.zeros(self.psymParent.pGPar.omega+1)
        periods = self.psymParent.pGPar.nCF
        self.psymParent.pOut.dCF = np.zeros(self.psymParent.pGPar.omega+1)
        self.psymParent.pTrace.strExecTasks = []
        self.vUpdateParam()

    def vUpdateParam(self):
        self.nOmega = self.psymParent.pGPar.omega
        self.dV = self.psymParent.pGPar.vTech
        self.nT0 = self.psymParent.pGPar.t0
        self.psymQx = self.psymParent.psymD.dQx

    def ConstantBenefit(self,x,param=[]):
        if x < self.b: return(0)
        return(1.)

    def ZeroBenefit(self,x,param=[]):
        return(0.)

    def PopulateParam(self):
        self.gender = int(self.psymParent.pInp.Param["g"])
        self.x = int(self.psymParent.pInp.Param["x"])
        self.s = int(self.psymParent.pInp.Param["s"])
```

```
        self.b = int(self.psymParent.pInp.Param["b"])
        self.fLevel = float(self.psymParent.pInp.Param["fLevel"])

        Tokens=["fM","fB"]
        for i in Tokens:
            if i in self.psymParent.pInp.Param.keys():
                dValue =  float(self.psymParent.pInp.Param[i])
            else:
                dValue = 0
            strExec="self."+i+"="+str(dValue)
            self.psymParent.pTrace.strExecTasks.append(strExec)
            exec(strExec)

        self.fB = int(self.psymParent.pInp.Param["fB"])
        self.fM = int(self.psymParent.pInp.Param["fM"])
        self.param=[]

    def CalcPV(self):
        self.vUpdateParam()
        dDK = self.psymParent.pOut.dDK
        dDKPer = self.psymParent.pOut.dDKPer
        gender = self.gender
        x = self.x
        s = self.s
        fLevel = self.fLevel
        param = self.param
        # Calculation of present value by means of recursion
        PV = self.psymM(s,param=param) * self.fM * fLevel
        dDK[s] = PV
        n  = s-x
        for i in range(s-1,x-1,-1):
            t = self.nT0 + i - x
            qx = self.psymQx(gender,i,t)
            px = 1. - qx
            PV = qx * self.dV * self.psymB(i,param=param)*self.fB* fLevel + px␣
↪* self.dV * PV
            dDK[i] = PV
            dDKPer[i-x] = PV

    def CalcCF(self):
        # Caculation of expected cash flows
        gender = self.gender
        x = self.x
        s = self.s
        fLevel = self.fLevel
        param = self.param
        periods = self.psymParent.pGPar.nCF
```

```
        CF =  self.psymParent.pOut.dCF
        px = 1.
        for i in range(x,s):
            t = self.nT0 + i - x
            qx = self.psymQx(gender,i,t)
            n = i - x
            if n >= periods:
                break
            CF[n] = px * qx * self.psymB(i,param=param)* fLevel
            px *= (1-qx)
            #print(i, px, qx)

        n = s - x
        if n < periods:
            CF[n] = px * self.psymM(s,param=param)* fLevel

    def vDoCalc(self):
        #print ("Task :",self.psymParent.pInp.Param)
        try:
            self.PopulateParam()
        except:
            print("Error Parameter")
            self.psymParent.TraceIO()
            return()

        self.CalcPV()
        self.CalcCF()

class QxCH():
    def __init__(self,psymParent):
        self.psymParent=psymParent
        self.psymParent.pOut.strHelloQx = "Hello QX"
        self.dQx=self.Qx

    def Qx(self,gender,x,t,param =[]):
        # This is our default mortality
        if gender == 0:
            a =[2.34544649e+01,8.70547812e-02,7.50884047e-05,-1.67917935e-02]
        else:
            a =[2.66163571e+01,8.60317509e-02,2.56738012e-04,-1.91632675e-02]
        return(min(1,max(0,np.exp(a[0]+(a[1]+a[2]*x)*x+a[3]*t))))

    def QxNoReduction(self,gender,x,t,param =[]):
        # This is our default mortality
        t = self.psymParent.pGPar.t0
        if gender == 0:
            a =[2.34544649e+01,8.70547812e-02,7.50884047e-05,-1.67917935e-02]
```

```
        else:
            a =[2.66163571e+01,8.60317509e-02,2.56738012e-04,-1.91632675e-02]
        return(min(1,max(0,np.exp(a[0]+(a[1]+a[2]*x)*x+a[3]*t))))

    def QxMedieval(self,gender,x,t,param =[]):
        a=-9.13275
        b= 8.09432e-2
        c= -1.1018e-5
        value=math.exp(a+(b+c*x)*x)
        alpha = 7.26502413
        beta = 0.01342065
        return(max(0,min(1,alpha*value+beta)))

    def vDoCalc(self):
        strTokens ={"Std":self.Qx,"NoRed":self.QxNoReduction,"Medieval":self.
↪QxMedieval}
        if "QX" in self.psymParent.pInp.Param.keys():
            if self.psymParent.pInp.Param["QX"] in strTokens.keys():
                print("Update Mortality to:",self.psymParent.pInp.Param["QX"])
                self.dQx = strTokens[self.psymParent.pInp.Param["QX"]]


class ITech():
    def __init__(self,psymParent):
        self.psymParent=psymParent

    def vDoCalc(self):
        if "fI" in self.psymParent.pInp.Param.keys():
            i = float(self.psymParent.pInp.Param["fI"])
            self.psymParent.pGPar.iTech = i
            self.psymParent.pGPar.vTech = 1./(1.+i)


    def dV(self,nPeriod):
        return(self.psymParent.pGPar.vTech)
```

[176]:
```
def main_t():
    a = TarEng()
    a.ParseTask("G;iTech;g=0;x=20;s=110;fB=0;fM=1000000")

    strTask = "A;iTech;g=0;x=55;b=60;s=90;fR=12000;fDeltaR=0.05"
    a.ParseTask(strTask)

    strTask = "A2;iTech;g=0;g2=1;deltaxy=3;x=20;s=120;fR2=12000"
    a.ParseTask(strTask)
    a.vPlot()
```

```
import time
import sys
import matplotlib
import matplotlib.pyplot as plt


def GetTable(ppCF,iT=0.025, InpBuckets = [2,5,10,15,20,30,100]):
    cmap = matplotlib.colormaps['Blues']
    cmap2 = matplotlib.colormaps['Oranges']
    cmap3 = matplotlib.colormaps['Greens']
    dCF = np.zeros([100,3])
    for i in range(100):
        dCF[i,0]=np.sum(ppCF[i,ppCF[i,:]>=0])
        dCF[i,1]=np.sum(ppCF[i,ppCF[i,:]<0])
        dCF[i,2]=np.sum(ppCF[i,:])
    Label = ["Index"]
    buckets =[]
    iPrior = 0
    for i in InpBuckets:
        buckets.append(range(iPrior,i))
        Label.append("%d-%d"%(iPrior,i-1))
        iPrior=i
    buckets.append(range(0,100))
    iNrColbuckets = 1 + len(buckets)
    Label.append("%d-%d"%(0,99))
    #print(Label)
    #print(buckets)
    Data = []
    CellColor = []

    strName0 = ["L DK","P DK", "T DK"]
    strName1 = ["L d","P d", "T d"]
    strName2 = ["L DV01","P PV01", "T PV01"]

    CF = [dCF[:,0],dCF[:,1],dCF[:,2]]
    for l in range(3):
        lCF=np.reshape(CF[l],100)
        print(lCF.shape)
        #print(repr(lCF))
        LData0 = []
        LData1 = []
        LData2 = []
        LCol0 = []
        LCol1 = []
        LCol2 = []
        for i in range(1+len(buckets)):
            alpha = 1.*i/iNrColbuckets*0.8
```

```
            LCol0.append(cmap(alpha))
            LCol1.append(cmap2(alpha))
            LCol2.append(cmap3(alpha))
        LData0.append(strName0[l])
        LData1.append(strName1[l])
        LData2.append(strName2[l])
        for i in buckets:
            PVA=0
            PVB=0
            PVAD = 0
            for k in i:
                PVA += lCF[k]*(1+iT)**(-k)
                PVAD += k * lCF[k]*(1+iT)**(-k)
                PVB += lCF[k]*(1+iT+1.e-4)**(-k)
            LData0.append("%.0fM"%((PVA)*1.e-6))
            if PVA != 0:
                LData1.append("%.2f"%(PVAD/PVA))
            else:
                LData1.append("n/m")

            LData2.append("%.0fk"%((PVB-PVA)*1.e-3))
        Data.append(LData0)
        Data.append(LData1)
        Data.append(LData2)
        CellColor.append(LCol0)
        CellColor.append(LCol1)
        CellColor.append(LCol2)
    #print(repr(Data))
    return(Label,Data,CellColor)


def main():
    cmap = matplotlib.colormaps['Blues']
    cmap2 = matplotlib.colormaps['Oranges']
    cmap3 = matplotlib.colormaps['Greens']
    bFirst = True
    bArgs = False
    psymF = open("tarif.txt","w")
    bContinue = True
    psymParm = dict()
    psymParm["input"] = None
    psymParm["outdk"] = "dk.csv"
    psymParm["outcf"] = "cf.csv"
    psymParm["pic"] = "cf.pdf"
    bBatch = False
    if bArgs:
        print(sys.argv)
```

```
        if len(sys.argv) >=2:
            psymParm["input"] = sys.argv[1]
            bBatch = True
        if len(sys.argv) >=3:
            psymParm["outdk"] = sys.argv[2]
        if len(sys.argv) >=4:
            psymParm["outcf"] = sys.argv[3]
    while bContinue:
        if bBatch:
            strInput = "batch"
        else:
            strInput = input()
        if "clear" in strInput:
            a.vClearInput()
            continue
        if "stop" in strInput:
            bContinue = False
            continue
        if "batch" in strInput:
            a = TarEng(bLevel=True,bTrace=False,bBatch=True)
            if psymParm["input"] == None:
                strInput = input("Batch File?")
                psymParm["input"] = strInput
            t0 = time.time()
            psymF2 = open(psymParm["input"],"r")
            all= psymF2.read()
            psymF2.close()
            lines = all.split("\n")
            iNrLines = len(lines)
            if iNrLines >10:
                bBig = True
                res = np.zeros([a.pGPar.nCF,2])
                dk = np.zeros([100,2])
            else:
                bBig = False
                res = np.zeros([a.pGPar.nCF,iNrLines])
                dk = np.zeros([100,iNrLines])
            iC = 0
            iLC = 0
            dDKTot = 0
            for i in lines:
                if iLC <10 or iLC % 1000 == 0:
                    print(iLC,":>",i,"<")
                iLC +=1
                a.ParseTask(i)
                strOut = a.strPrint()
                psymF.write(strOut)
```

```
            dDK = a.pOut.dDKPer[0]
            dDKTot += dDK
            if bBig:
                if dDK >= 0: iIndex = 1
                else: iIndex = 0
                res[0:100,iIndex] += a.pOut.dCF[0:a.pGPar.nCF].transpose()
                dk[0:100,iIndex] += a.pOut.dDKPer[0:100].transpose()
            else:
                res[0:100,iC] = a.pOut.dCF[0:a.pGPar.nCF].transpose()
                dk[0:100,iC] = a.pOut.dDKPer[0:100].transpose()
                iC+=1
        np.savetxt(psymParm["outcf"], res, delimiter=";")
        np.savetxt(psymParm["outdk"], dk, delimiter=";")
        bContinue = False

        if bBig: iNrLines = 2

        xx=[]
        y=[]
        c=[]
        b=[]
        w=[]
        for i in range(0,a.pGPar.nCF):
            bH=0
            bL=0
            for j in range(0,iNrLines):
                fCF = res[i,j]
                if fCF>=0:
                    xOffset = 0
                    if bBig:
                        c.append(cmap(0.8))
                    else:
                        c.append(cmap(j*1./iNrLines))
                    fDeltaPlus = fCF
                    fDeltaMinus = 0
                    bPlus = True
                else:
                    xOffset = 0.5
                    if bBig:
                        c.append(cmap2(0.8))
                    else:
                        c.append(cmap2(j*1./iNrLines))
                    fDeltaPlus = 0
                    fDeltaMinus = fCF
                    bPlus = False

                xx.append(i+xOffset)
```

```
                    w.append(0.4)
                    if bPlus:
                        y.append(fCF)
                        b.append(bH)
                    else:
                        y.append(-fCF)
                        b.append(bL+fCF)
                    bH+=fDeltaPlus
                    bL+=fDeltaMinus
            fig= plt.figure(1)
            fig.set_size_inches(11.7,8.3)
            f,ax=plt.subplots(2,1)
            plt.subplot(2,1,1)
            plt.bar(xx,y,bottom=b,color=c,width=w,edgecolor="k",linewidth=0.2)
            plt.grid(True)
            ax=plt.subplot(2,1,2)
            MyLables =["x","y","z"]
            Data = [["A",10,20],
                    ["B",30,50]]
            MyLables,Data,CellColors =GetTable(res)
            ax.xaxis.set_visible(False)
            ax.yaxis.set_visible(False)
            ax.set_frame_on(False)
            table=ax.table(cellText=Data, colLabels␣
↪=MyLables,loc="center",cellColours=CellColors)
            table.auto_set_font_size(False)
            table.set_fontsize(8)
            #plt.subplots_adjust(left=0.1,top=0.4)

            plt.savefig(psymParm["pic"],dpi=1200)

            if dDKTot >1e9:
                print("DK Total %.3f bn"%(1.e-9*dDKTot))
            else:
                print("DK Total %.0f"%(dDKTot))
            print("Batch done --> exit (time elapsed %6.2f s)"%(time.time()-t0))

        else:
            if bFirst:
                a = TarEng()
                bFirst = False
            a.ParseTask(strInput)
            strOut = a.strPrint()
            psymF.write(strOut)
        #a.vPlot()
    psymF.close()
```

```python
if __name__ == "__main__":
    if bExaMode:
        main_t()
    else:
        main()
```

```
--------------------------------------------------
G
--------------------------------------------------
omega           :        120
t0              :       2025
iTech           :      0.025
g               :          0
g2              :          1
x               :         20
deltaxy         :         -3
b               :         20
s               :        110
fB              :          0
fM              :    1000000
fLevel          :        1.0
--------------------------------------------------
    Age:                 DK/MR                 E[CF]
--------------------------------------------------
     20:           9293.517035              0.000154
     21:           9527.323301              0.000166
     22:           9767.126233              0.000178
     23:          10013.091671              0.000192
     24:          10265.391308              0.000207
     25:          10524.203018              0.000223
     26:          10789.711224              0.000240
     27:          11062.107286              0.000258
     28:          11341.589934              0.000278
     29:          11628.365736              0.000299
     30:          11922.649611              0.000322
     31:          12224.665391              0.000347
     32:          12534.646437              0.000374
     33:          12852.836315              0.000403
     34:          13179.489540              0.000435
     35:          13514.872393              0.000468
     36:          13859.263822              0.000505
     37:          14212.956437              0.000545
     38:          14576.257606              0.000587
     39:          14949.490671              0.000633
     40:          15332.996285              0.000683
     41:          15727.133906              0.000737
     42:          16132.283442              0.000794
     43:          16548.847082              0.000857
```

```
44:          16977.251333            0.000925
45:          17417.949284            0.000998
46:          17871.423123            0.001077
47:          18338.186954            0.001162
48:          18818.789928            0.001254
49:          19313.819752            0.001353
50:          19823.906608            0.001460
51:          20349.727543            0.001576
52:          20892.011389            0.001701
53:          21451.544294            0.001837
54:          22029.175923            0.001982
55:          22625.826451            0.002140
56:          23242.494434            0.002310
57:          23880.265695            0.002493
58:          24540.323365            0.002691
59:          25223.959242            0.002904
60:          25932.586679            0.003134
61:          26667.755202            0.003382
62:          27431.167154            0.003649
63:          28224.696646            0.003936
64:          29050.411207            0.004245
65:          29910.596542            0.004578
66:          30807.784922            0.004936
67:          31744.787810            0.005320
68:          32724.733426            0.005733
69:          33751.110131            0.006175
70:          34827.816644            0.006650
71:          35959.220335            0.007157
72:          37150.225084            0.007700
73:          38406.350518            0.008280
74:          39733.824820            0.008898
75:          41139.693806            0.009556
76:          42631.949560            0.010256
77:          44219.682695            0.010998
78:          45913.263265            0.011784
79:          47724.556584            0.012614
80:          49667.181775            0.013488
81:          51756.822861            0.014406
82:          54011.604826            0.015367
83:          56452.550375            0.016368
84:          59104.137564            0.017408
85:          61994.984144            0.018483
86:          65158.692116            0.019587
87:          68634.896051            0.020714
88:          72470.572339            0.021857
89:          76721.684784            0.023007
90:          81455.266950            0.024151
91:          86752.075896            0.025276
```

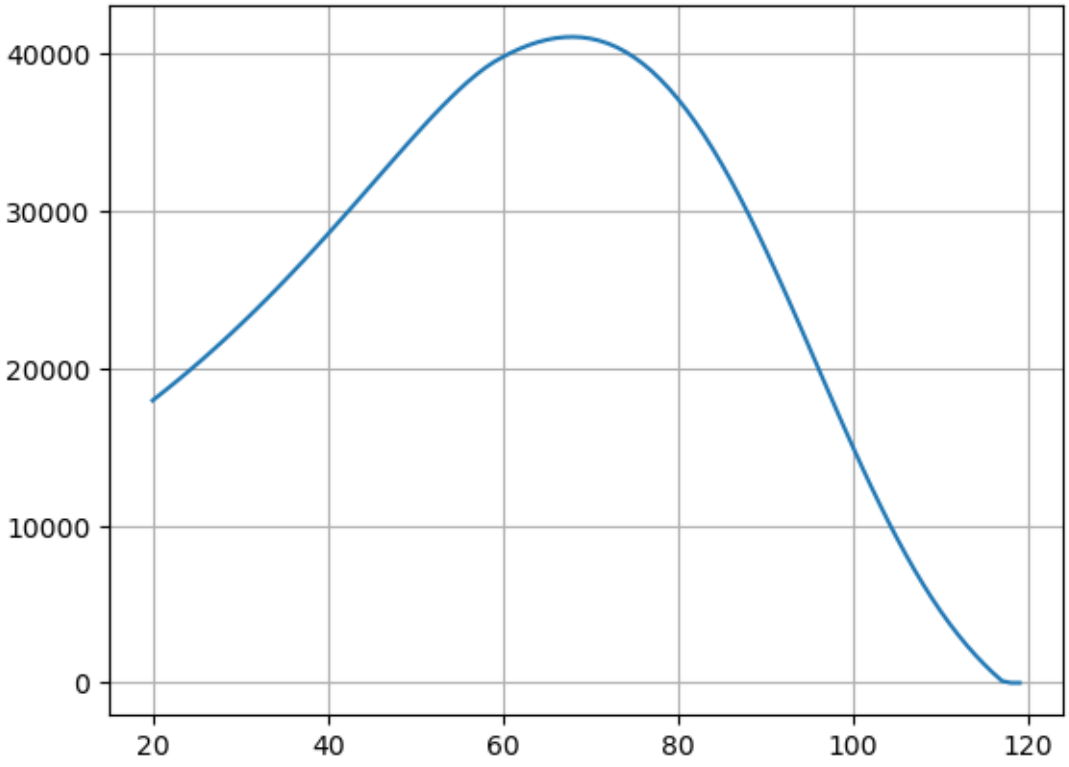
```
     92:          92709.999406              0.026367
     93:          99448.465199              0.027407
     94:         107114.194317              0.028376
     95:         115888.774601              0.029252
     96:         125998.723015              0.030013
     97:         137728.986819              0.030632
     98:         151441.248918              0.031085
     99:         167599.023721              0.031347
    100:         186802.471206              0.031391
    101:         209837.304533              0.031196
    102:         237744.427098              0.030742
    103:         271920.522870              0.030014
    104:         314265.618947              0.029004
    105:         367403.174949              0.027712
    106:         435014.261456              0.026147
    107:         522354.844280              0.024330
    108:         637073.379283              0.022293
    109:         790532.637698              0.020080
    110:        1000000.000000              0.085769

--------------------------------------------------
A
--------------------------------------------------
omega           :         120
t0              :        2025
iTech           :       0.025
g               :           0
g2              :           1
x               :          55
deltaxy         :          -3
b               :          60
s               :          90
fB              :           0
fM              :     1000000
fLevel          :         1.0
fR              :       12000
fDeltaR         :        0.05
--------------------------------------------------
    Age:                DK/MR                E[CF]
--------------------------------------------------
     55:         188829.577642              0.000000
     56:         194318.087054              0.000000
     57:         200031.007129              0.000000
     58:         205984.329455              0.000000
     59:         212195.765377              0.000000
     60:         218684.972594          11723.347925
     61:         213101.288744          11654.626095
     62:         207445.279892          11580.665210
```

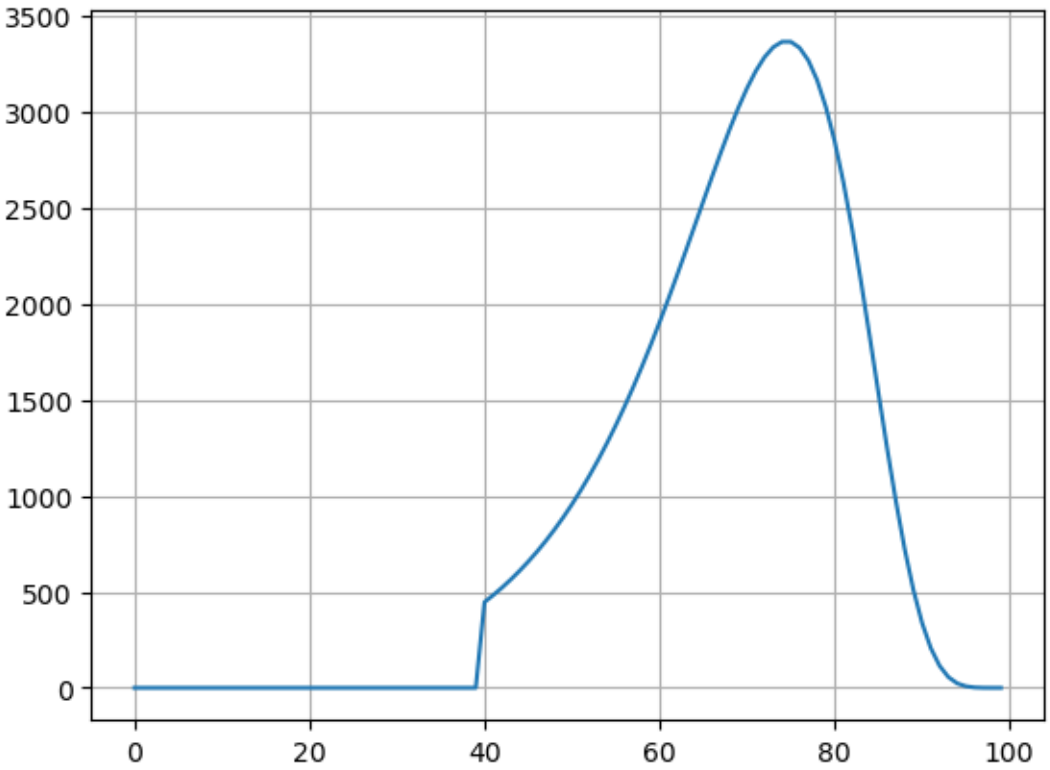
```
     63:        201717.443480         11501.092678
     64:        195918.171459         11415.515115
     65:        190047.706154         11323.518323
     66:        184106.087614         11224.667534
     67:        178093.090895         11118.507992
     68:        172008.151435         11004.565927
     69:        165850.276285         10882.350021
     70:        159617.938455         10751.353438
     71:        153308.951035         10611.056529
     72:        146920.316940         10460.930327
     73:        140448.049147         10300.440945
     74:        133886.955003         10129.055018
     75:        127230.376520          9946.246326
     76:        120469.876438          9751.503750
     77:        113594.857040          9544.340697
     78:        106592.095002          9324.306149
     79:         99445.170747          9090.997457
     80:         92133.764269          8844.074992
     81:         84632.780751          8583.278721
     82:         76911.257620          8308.446725
     83:         68930.988761          8019.535609
     84:         60644.779813          7716.642642
     85:         51994.218305          7400.029356
     86:         42906.800262          7070.146163
     87:         33292.195585          6727.657376
     88:         23037.350015          6373.465763
     89:         12000.000000          6008.735562
     90:             0.000000             0.000000

-------------------------------------------------
A2
-------------------------------------------------
omega          :        120
t0             :       2025
iTech          :      0.025
g              :          0
g2             :          1
x              :         20
deltaxy        :          3
b              :         60
s              :        120
fB             :          0
fM             :    1000000
fLevel         :        1.0
fR             :      12000
fDeltaR        :       0.05
fR2            :      12000
-------------------------------------------------
```

Age:	DK/MR	E[CF]
20:	17971.121492	0.000000
21:	18407.674309	0.000000
22:	18853.831303	0.000000
23:	19309.695518	0.000000
24:	19775.357972	0.000000
25:	20250.895778	0.000000
26:	20736.370024	0.000000
27:	21231.823421	0.000000
28:	21737.277658	0.000000
29:	22252.730456	0.000000
30:	22778.152267	0.000000
31:	23313.482589	0.000000
32:	23858.625843	0.000000
33:	24413.446765	0.000000
34:	24977.765255	0.000000
35:	25551.350610	0.000000
36:	26133.915075	0.000000
37:	26725.106627	0.000000
38:	27324.500893	0.000000
39:	27931.592094	0.000000
40:	28545.782896	0.000000
41:	29166.373032	0.000000
42:	29792.546525	0.000000
43:	30423.357339	0.000000
44:	31057.713245	0.000000
45:	31694.357675	0.000000
46:	32331.849271	0.000000
47:	32968.538827	0.000000
48:	33602.543247	0.000000
49:	34231.716113	0.000000
50:	34853.614338	0.000000
51:	35465.460364	0.000000
52:	36064.099194	0.000000
53:	36645.949498	0.000000
54:	37206.947841	0.000000
55:	37742.484927	0.000000
56:	38247.332562	0.000000
57:	38715.559771	0.000000
58:	39140.436190	0.000000
59:	39514.320519	0.000000
60:	39828.531307	446.429456
61:	40113.446938	482.778620
62:	40366.891188	521.859973
63:	40586.651351	563.857044
64:	40770.488538	608.960715
65:	40916.149460	657.368356

66:	41021.379803	709.282610
67:	41083.939223	764.909776
68:	41101.618046	824.457683
69:	41072.255661	888.132967
70:	40993.760640	956.137644
71:	40864.132547	1028.664827
72:	40681.485361	1105.893463
73:	40444.072430	1187.981921
74:	40150.312801	1275.060238
75:	39798.818710	1367.220856
76:	39388.424003	1464.507626
77:	38918.213152	1566.902895
78:	38387.550479	1674.312476
79:	37796.109157	1786.548335
80:	37143.899459	1903.308883
81:	36431.295668	2024.156836
82:	35659.061013	2148.494725
83:	34828.369921	2275.538322
84:	33940.826840	2404.288470
85:	32998.480858	2533.502162
86:	32003.835322	2661.664098
87:	30959.851677	2786.960502
88:	29869.946782	2907.257629
89:	28737.983014	3020.088153
90:	27568.250580	3122.649491
91:	26365.441586	3211.819007
92:	25134.615590	3284.191943
93:	23881.156538	3336.148557
94:	22610.721267	3363.957260
95:	21329.179977	3363.920120
96:	20042.549373	3332.565606
97:	18756.919469	3266.890437
98:	17478.375315	3164.647399
99:	16212.915190	3024.668578
100:	14966.367014	2847.203571
101:	13744.304917	2634.240117
102:	12551.968006	2389.761441
103:	11394.183397	2119.882771
104:	10275.295488	1832.802463
105:	9199.103280	1538.506109
106:	8168.807266	1248.180438
107:	7186.966983	973.332535
108:	6255.469833	724.670302
109:	5375.511183	510.876416
110:	4547.585083	337.484546
111:	3771.484330	206.116065
112:	3046.308359	114.323712
113:	2370.479094	56.186867

```
114:          1741.775817            23.608079
115:          1157.465323             8.016588
116:           615.083587             1.990415
117:           120.009975             0.288042
118:             0.000000             0.007773
119:             0.000000             0.000000
```




3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100