Michael Koller
Feldstrasse 14
CH 8704 Herrliberg
Phone: +41 44 915 41 47
Switzerland

Herrliberg, July 18, 2012

Contents

1 Aim

The aim of `vamod` is to provide a library, where for a a given protfolio of policies and a given performance vector the respective cash flows for VA products can be calculated.
The driver routine, simulating the different preformace vectors is out of scope. The respective functunality is partially provided by `simlib`. This module is intended to be integrated in the `MARKOVLV` family

2 Products Modelled

There are the following VAs:

GMDB: This the VA analogon of a term insurance. Hence the loss in case of death is the difference between the agreed contractual death benefit and the respective funds value.

GMAB: This the VA analogon of a pure endowment insurance. Hence the loss in case of surviving the term of the insurance is the difference between the agreed contractual death benefit and the respective funds value.

GMIB: This is roughly speakting a GMIB where at maturity the respective guarantee is converted in an immediate payout annuity. This product will not be modelled in `vamod`

GMWB: This the VA analogon of an immediate or deferred annuity. Hence the loss are the annuityies which can not be taken out of the account value

3 Calculation Formulae

General Notation In the following section I summarise my understanding of the economics of this sort of contract. Assume the following:

- Person aged x_0 purchases such a GMWB and pays a single premium EE;
- Assume that the person starts to withdraw at age sw and that the income phase starts at age s;

- We use the following notation:

  $F(t)$: Funds value at time t. To be more precise we denote with $F(t)^-$ and $F(t)^+$ the value of the funds before and after withdrawal of the annuity, respectively. In consequence we have $F(t)^+ = F(t)^- - R(t)$.

  $GWB(t)$: GWB ("Guaranteed Withdrawal Balance") value at time t. To be more precise we denote with $GWB(t)^-$ and $GWB(t)^+$ the value of the funds before and after withdrawal of the annuity, respectively. In consequence we have $GWB(t)^+ = GWB(t)^- - R(t)$.

  $f(x)$: GAWA percentage if person starts to withdraw at age x;

  $GAWA(t)$: maximal allowable withdrawal benefit (usually $= GWB \times f(x)$).

  $R(t)$: actual amount withdrawn. Note that we have the following: $R(\xi) = 0$ for $\xi < sw$, $0 \leq R(\xi) \leq GAWA(x)$, for all $\xi \in [sw, s[$, and $R(\xi) = GAWA(x)$, for all $\xi \geq s$, assuming that the "for life option" is in place and identifying x and t in the obvious way, eg $x(t) = t - t_0 + x_0$

- With $\eta(t, \tau) \in \mathbb{R}$, we denote the fund performance during the time interval $[t, \tau]$, with $\tau > t$.

- We do not allow for changes in funds and lapses at this time and also do not consider the death of the person insured. This would add some complexity, where actually the annuities need to be weighted with the respective probabilities ${}_tp_x$ and in the same sense the respective death cover weighted with ${}_tp_x\, q_x$. For the moment assume that $\sum \tau \geq 0$ stands for "until death".

- By $X(t)$ we denote the loss at time t occurring from GMWB guarantees. It is obvious that under these premises the value of the total guarantee $Y = \sum_{\tau \geq 0} (1 + r(\tau))^{-\tau} X(\tau)$, where $r(\tau)$ represents the risk free interest between $[0, \tau]$.

As described above we have the following:

$$f(x) = \begin{cases} 5\% & \text{if } x \in [55, 74], \\ 6\% & \text{if } x \in [75, 84], \\ 7\% & \text{if } x \geq 85. \end{cases}$$

For the recursion we have at time $t_0 = 0$:

$$\begin{aligned} F(0) &= EE, \\ GWB(0) &= EE, \\ GAWA(0) &= f(sw) \times GWB(0). \end{aligned}$$

Afterwards from time $t - 1 \rightsquigarrow t$ we have the following:

$$\begin{aligned} F(t)^- &= (1 + \eta(t-1, t)) \times F(t-1)^+, \\ GWB(t)^- &= \max\{GWB(t-1)^+, \max_{k=0,1,\ldots,4}\{1 + \eta(t-1, t-1+\frac{k}{4})\} \times F(t-1)^+\}, \\ GAWA(t) &= \max(GAWA(t-1), f(sw) \times GWB(t)^-), \\ F(t)^+ &= \max(0, F(t)^- - R(t)), \\ GWB(t)^+ &= \max(0, GWB(t)^- - R(t)), \\ X(t) &= \max(0, R(t) - F(t)^-), \\ \pi(Y) &= E^Q\left[\sum_{\tau \geq 0} (1 + r(\tau))^{-\tau} X(\tau)\right]. \end{aligned}$$

If we now pick a given, mortality cover – the simplest one – namely the payment of $GWB(t)$ in case of death, we can calculate the value of the insurance option as follows:

$$\pi(Y) = E^Q\left[\sum_{\tau\geq 0}^{\infty}(1+r(\tau))^{-\tau}X(\tau)\times {}_\tau p_{x_0}\right],$$

Calculation of Funds and Losses As opposed to the specific setting, we aim to define the change in funds value more generally, namely:

$$\begin{aligned}
T(\omega) &= \text{Future life span. T=x: means person dies aged x}\\
F(t) &= \text{As above fund value at time t}\\
CF(t) &= \text{Cash flow at time t}\\
\eta(t-1,t) &= \text{Funds performance}\\
F(t)^- &= (1+\eta(t-1,t))\times F(t-1)^+,\\
F(t)^+ &= \max(0, F(t)^- - CF(t)),\\
X(t) &= \max(0, R(t) - F(t)^-),\\
\pi(Y) &= E^{Q\times S}\left[\sum_{\tau\geq 0}(1+r(\tau))^{-\tau}X(\tau)\right].
\end{aligned}$$

Note that S represents the probability measure with respect to the state the PHs are in.
Reference Quantity for benefits
With $R(t)$ we denote the ratcheted up funds value if ratcheting is present. With $G(t)$ we denote the guarantee value. We have the following:

$$\begin{aligned}
R(t+1) &= \max(R(t), S(t))\times\delta_{RA=1} + S(t)\times\delta_{RA=0}\\
G^{exp}(t) &= \begin{cases} 1 & \text{if} \quad x < x_0\\ (1+\alpha)^{x-x_0} & \text{if} \quad x_0 \leq x < x_1\\ (1+\alpha)^{x1-x_0} & \text{else}\end{cases}\\
G^{lin}(t) &= \begin{cases} 1 & \text{if} \quad x < x_0\\ (1+\alpha\times(x-x_0)) & \text{if} \quad x_0 \leq x < x_1\\ (1+\alpha\times(x1-x_0)) & \text{else}\end{cases}\\
G(t0 &= G^{lin}\times\delta_{lin=1} + G^{exp}\times\delta_{exp=1}\\
BE(t) &= \max(S(t), R(t), G(t))\\
CF(t) &= \sum_{i=1}^{n} BE(t)\times I_{Event\ i\ happens}(t)\times\beta_i
\end{aligned}$$

Note that i represents the cases “Death”, “Maturity”, “Annuity payment”, “Premium Payment”. As example for $i =$ “Death” we have $I_{Death}(t) = \delta_{T=t}$.

4 Structures

```
typdef struct VABENEFITS
{
  // Definition of Guarantee Vector
  double dStartValueGuarantee;
  double dIncreasePA;
  int    iStartGuaranteeAge;          // x_0
  int    iEndGuaranteeAge;            // x_1
  bool   bLinear;
```

```
  bool   bExponential;
  // Take also Fund Value into Account and make max - and how (eg Ratchet)
  bool   bMaxWithFunds; //Otherwise only guarantee
  int    iRatchet;  // 0 - no otherwise every iRatchet Periods
                                   // RA
  // Which Types of Benfits
  // note if Age < Current Age --> No Benefit
  int    iEndowmentAge; //0 - no endowment - otherwise maturity age
  int    iSTerm; // 0 - no term benefit otherwise s-age
  int    iSAnnuity; // Start age Annuity
  int    iSLastAnnuity; // Age at which Annuity ceases (\infty for lifelong)
  int    iSPrem; //Last Age with Premium
  // Levels of Benefits -- these are the beta's
  double dPctEndowment;              // F(t)
  double dPctTerm;                   // R(t)
  double dPctAnnuity;                // CF(t)
  double dPctPremium;                // X(t)
} VABENEFITS;

typedef struct VAINVESTMENT
{
  // This Structure is also for rolling forwards
  double dEE;
  double dSAA[NRFUNDS];
  int iAgeRiskFree; // Means if Age >=iAgeRiskFree All assets in risk free (asset 0)
  // This are the current Cash Flows and
  double dPerformance[NRFUNDS];
  double dCurrentVA;
  double dCurrentRatchet;
  double dCurrentCashFlow;
  double dCurrentLoss;
} VAINVESTMENT;

typedef struct VAPERSON
{
  long lId
  int iAge;
  int iGender;
  int iBirthYear;
  VABENEFITS * psymB;
  VAINVESTMENT * psymI;
} VAPERSON;
```

5 Classdefinition

```
// note that we use the following structures and objects to do
// VAPROJECT:
// 1. VAINFORCE to hold individual policydata
// 2. Simlib for simulation
// 3. GLMOD to do the exected actuarial cash flows and reserves

class VAINFORCE
{
```

```
 public:
  VAINFORCE();
  ~VAINFORCE();
  void vGotoStart();
  void nNext();
  int iAnalyseToken(char * pcString);
  VAPERSON * pGetPerson(long lId);
  VAPERSON * pNewPerson();
  VAPERSON * psymCurrentPers;

 private:
  VAPERSON * psymAllPers;

};

class VAPROJECT:VAINFORCE,SIMLIB,MARKOVLV
{
 public:
  VAPROJECT();
  ~VAPROJECT();
  double         dSetQx(long lTable, long lType, long lSex, long lTime, double dValue);
  double         dSetFx(long lTable, long lType, long lSex, long lTime, double dValue);
  double         dSetSx(long lTable, long lType, long lSex, long lTime, double dValue);
  double         dSetBaseYear(long lTable, long lType, long lSex, long lTime);
  double         dSetActualYear(long lTime);
  double         dSetDisc(long lTime, double dValue);

  void           vGenerateTrajectory();
  long           vGetState(long lTime);
  double         dGetRandCF(long lTime);
  double         dGetRandDK(long lTime, long lMoment);
  double         dGetMeanCF(long lTime, long lState, long lNrSim);
  double         dGetMeanDK(long lTime, long lState, long lNrSim);
  double  dGetDKDetail(long lTime, long lState); // Berechnet DK's fuer jeden State.
  double         dGetCFDetail(long lTime, long lState);
  void           vNewSeed(long lSeed);
  void           vResetMeanResults();
  long           lSeed;

  void           vAddDeath(long lSex, long lX, long lS, long lNTimes, double dLeist, dou
  void           vAddEndowment(long lSex, long lX, long lS, double dLeist, double dPraem
  void           vAddPremium(long lSex, long lX, long lS, double dLeist, double dPraem,

  void           vUpdateOperator();

  double         dGetQx(long lOrder, long lTafel, long lSex, long lTime, long lYear);
  double         dSetRelativeQxForTime(long lTime, double dValue); // eg x_0 + time we m
  int            iReadInforce(int iP, int iL, char * strFileName);
  void           vPrintTex(char * strName);
 private:
  bool           lValid;
  bool           bTildeCalc;
  LV_VECTOR  *psymQx[2];  // Tafel1/2 ; K oder R ; sex
  LV_VECTOR  *psymFx[2];
```

```
LV_VECTOR  *psymDisc;
LV_VECTOR      *psymSx;
LV_VECTOR  *psymRelQxTime;
LV_VECTOR      *psymTilde;
long           lBaseYear[2];
long  lActualYear;

FILE   *   psymTrace;
};
```

6 Implementation

In order to ensure efficient processes we note the following:

- In a normal environment one would project the cash flows per policy and weight it with the respective actuarial probabilities. This means that this approach is done per policy and simulation. There is however a possibility to save a lot of calculations.
- The first trick is to use the GLMOD approach where we base the calculation on a semi-markov approach with one state per age and gender.
- Since we are expected in present values one can in a first step we assume that the persons survive and die at the same time. This works since we have a tree structure. Moreover we can in a first step add all simulations together and divide the results only at the end by the number of simulations
- This is more or less the approch, as the GMxB's have been explained.

Hence the implementation is as follows:

1. Construct an instance of *GLMOD Note: It is likely easier to include a GLMOD structure in the new object, because one can better implement the functionality intrinsic to the VA's. This would also have the beauty that the object is directly generalised from OMARKOV*
2. For all $i \in \{sims\}$ call create first a capital market trajectory and do *DoOneSimiulation for Portfolio*
3. *DoOneSimiulation for Portfolio* does calculate the funds values and corresponding guarantee payment cash flows for all times $t \in T$. For each $t \in T$ one calculates rolls forwards are relevant quantities for each policy $p \in P$. The respective guarantee payments are added to the *GLMOD* instance via `vAddEndowment(long lSex, long lX, long lS, double dLeist, double dPraem, double dITechn);`, etc. *Note:* These functions to add at a particular time a death or survival benefit need to be refined in *GLMOD*. The functunality `vAddXXX` ensures that the respective benefits are added. If there are two simulations for arguments sake we just have the double, etc/
4. After having done the simulations over all $i \in \{sims\}$ the operators `dGetDK(long lTime);` and `dGetCF(long lTime);` are called to get the respective results for the *sum* of all simulations. Hence the respective results of the new object are $\frac{1}{card(\{sims\})} \times dGetDK(long\, lTime)$, etc.

Implementation details:

- The implementation of the new object `VAPROJECT` should form a functional perspective follow as close as possible the `OMARKOV` object, eg having the following functionalities: `void vSetInitState(long lInitState); void vGenerateTrajectory(); long vGetState(long lTime); double dGetRandCF(long lTime); double dGetRandDK(long lTime, long lMoment); double dGetMeanCF(long lTime, long lState, long lNrSim); double dGetMeanDK(long lTime, long lState, long lNrSim); void vNewSeed(long lSeed); void vResetMeanResults long lSeed;`

- Since we might be interested in both Premiums and Benefits separately, we should consider to use the first 200 states for benefits and the following 200 for premiums. It would also be advisable to enlarge the functionality to get the respective MR for annuities and mortatality benefits separately.