IEM Research Orientation Programming Assignment

Orientation Financial Engineering and Management (FEM)

Authors: Thymen van der Poll, Berend Roorda

Introduction

Within economics and decision making, utility is a well-known measure for the relative value of for outcomes of a gamble. A common way to model the value assigned to a game as a whole is by the expected utility of all its outcomes: the famous 'Von Neumann-Morgenstern' Expected Utility (EU) framework. In a dynamic, tree-like setting, this means that prices can be determined backward recursively, as a consequence of the law of iterated expectations. In other words, a lottery of (sub-)lotteries, also called a compound lottery, can be identified with a lottery on the expected utility of the sub-lotteries.

There is an abundance of evidence, however, that risk attitudes towards a compound lottery, cannot be properly understood in terms of risk attitudes towards each of its sub-lotteries separately, contrary to the EU framework. The Allais-paradox showed that the systematic aspects in human decision making clearly violate the Independence Axiom underlying expected utility. Despite decades of empirical and theoretical research, there is still controversy whether these deviations should be interpreted as biases of the human mind, comparable to optical illusions, or biases in the Independence Axiom itself, rendering it as less rational than it seems to be at first sight. The approach in this assignment follows the latter view, by implementing a straightforward explanation in terms of '*Tuned Risk Aversion'*.

Assignment

In this assignment you will implement a "*Tuned Risk Aversion"* method. This is done in the following fashion:

(1) To get started, part one focusses on the implementation of a dynamic programming algorithm that just determines expected value.
(2) Part two extends the model of part one by embedding utility into the model. For this model the *Entropic Risk Measure* is used.
(3) In part 3, risk-aversion tuning parameters are introduced. The idea is to find the optimal set of risk-aversion tuning parameters for a given lottery.

Keep in mind that there is programmable overlap between the parts. That is, write procedures and functions that are general. We cannot emphasize this enough, as this is essential to the Dynamic Programming model where you will have to determine values using same the function, for different states and stages. **Only** code in the Assignment.pas file. Make sure your code is able to do that. Code should be clean, well-commented, free of bugs and errors and styled just as in the first two programming sessions. See the deliverables section.

1. Dynamic Programming

One of the most common methods of valuating a certain stock, or option, is using dynamic programming. Dynamic programming implies that there is a consistent recursive relation between the stages. In this first part of assignment you will just use the expected value of the outcome. You are given a vector of possible outcomes X_T at T,

$$X_T = [x_{T,1}, \dots, x_{T,n}].$$

This resembles the outcomes in the final nodes of a (recombining) binomial tree, with a given probability p for the up-branch, see Figure 1 for an example with n=3. Note that the number of periods T derives from the length n of X_T : we have n=T+1. The vectors of values at earlier time instants t are denoted as X_t, having t+1 entries. In particular, X_0 is the initial value in the tree.

In this part, X_t is just defined as the expected value of X_T, conditioned at time t. To prepare for the next parts, you have to compute it backwardly, starting with X_{T-1}, "up" to X_0 , in Delphi, according to the following rules.

In the *Assignment.pas* file, you will find X_T as *TValueArrayX*, where *TValueArrayX* is an array of double, defined in *Main.pas*. Then, there is constant probability for that the price goes up, denoted by *p*, for which we take 0.5 as default.

Write the values of X_t to *ResultArray,* an array of *TValueX*. The highest index of *ResultArray* is already filled with the values of X_T. Make sure that the dimension of ResultArray does not change, as this is used to visualize. So the columns in the triangular ResultArray correspond to X_0, X_1, …, X_T.

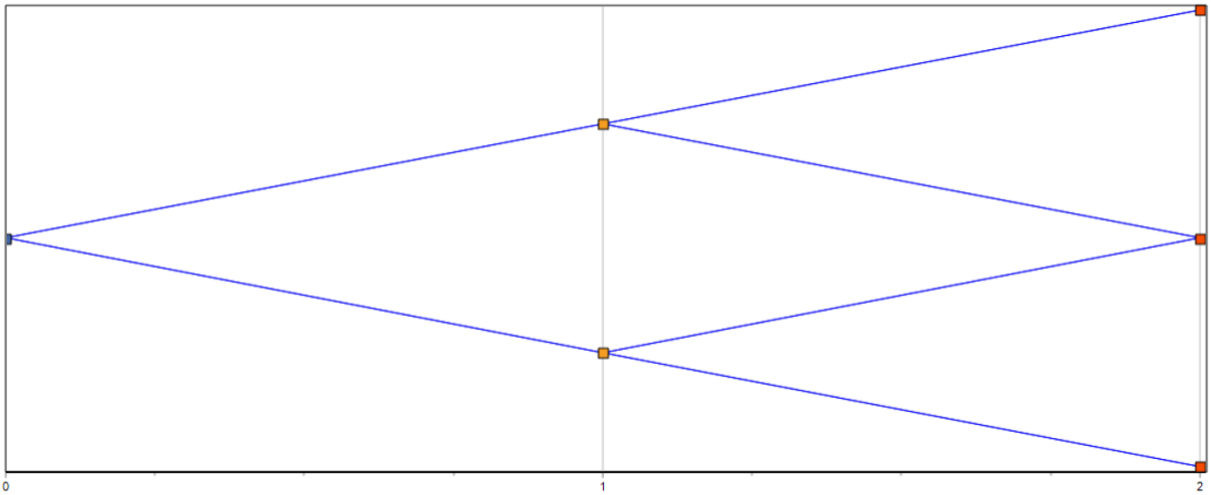

Figure 1, Structure of the relation between stages

2. Entropic Risk

Expected value is just a simple way to denote the value of a stock. There is no measure of risk involved. In this exercise, the Entropic Risk Measure is used:

$$Entropic\ Risk\ Measure\ \varphi(X) = -\frac{1}{\beta}\ln \mathrm{E}[\mathrm{e}^{-\beta X}]$$

For β=0, this is defined as just the expected value (verify, for yourselves, that this is also the limit for $\beta \to 0$). Also verify that phi(X)=c if X is the constant vector with all entries equal to c.

For the simplest case, with $X = [x_u, x_d]$, the expected value $\mathrm{E}[\mathrm{e}^{-\beta x}]$ is $p\mathrm{e}^{-\beta x_u} + (1-p)e^{-\beta x_d}$, where x_u and x_d are the values for the price going up and going down. This is exactly the expression that needs to be evaluated in each node of the binomial tree (*Figure 1*). To avoid numerical issues, we first subtract the mean $m = px_u + (1-p)x_d$, then evaluate the expression, and then add up the mean again. Here we use that $\varphi(X - m) = \varphi(X) - m$, where in $X - m$, m is subtracted from every entry in X (verify this equality for yourselves).

This function has already been implemented for you. The function header is:

function EntropicRiskMeasure(X_input:TValueArrayX; Beta, ProbabilityUp: double): TValueArrayX;

As you can see, the function takes as input X_input, which is an array of doubles and is basically a vector containing the values for X_t. In this function, an array is created with the results of the Entropic Risk Measure for X_t, a constant risk aversion budget β for which the input parameter is called Beta, and the probability of the price going up which for which the input parameter is ProbabilityUp. This function returns a vector containing the results of X_{t-1} stored in the array of double TValueArrayX. Obviously, this array has a length of *Length(X_input) – 1*.

This extends to the general case in the obvious way: given X_T, compute all entries of X_{T-1} as above, etc., until X_0. We can write, in vector notation, $X_{t-1} = \varphi_{t-1,\beta}(X_t)\ with\ \varphi_{t,\beta}$ the one-step entropic measure applied at time t.

Do this for the given X_T and β = 100. Again, write the values to *ResultArray*.

3. Tuned Risk Aversion

You have now built the model incorporating a recursive relation between stages. However, as argued in the lecture, the Allais paradox indicates that people tend to deviate from this rule. We leave aside here whether that is rational or not, but focus on incorporating non-recursiveness in the valuation scheme. Here we follow the idea of Tuned Risk Aversion (TRA): instead of applying the same parameter β in every step, we allow for different levels of risk aversion, only putting a limit on the sum of the risk parameters over all time steps, see the last sheet of the lecture. This limit is called the risk budget, denoted as B. The idea is to tune the timing of risk aversion to the position X, by determining the pattern that hurts the most. This leads to the optimization problem

$$\phi_B(\mathrm{X}_T) := \min\left\{\varphi_{0,\beta_0}\left(\ldots\left(\varphi_{T-1,\beta_{T-1}}(X_T)\right)\ldots\right)\middle|\beta_t \geq 0, \beta_0 + \ldots + \beta_{T-1} = \mathrm{B}\right\}.$$

Here β_t is a vector, of the same length as X_t, , that specifies the applied level of risk aversion in all nodes at time t. To help you, this problem starts with applying the entropic risk measure $\varphi(X)$ at $t = T$. Then, the 2-step can be formulated in the following fashion, where you clearly see that you apply the bits of risk aversion budget over a path.

$$\phi_T(\mathrm{X}_2) := min\{\varphi_{0,\beta_0}(\varphi_{\beta_u}(X_{uu}, X_{ud}), \varphi_{\beta_d}(X_{ud}, X_{dd}))|\ \beta_o + \beta_u \leq B, \beta_0 + \beta_d \leq B\}$$

Note that this is the same as

$$\phi_T(\mathrm{X}_2) := min\{\varphi_{0,\beta_0}(\varphi_{\beta_u}(X_{uu}, X_{ud}), \varphi_{\beta_d}(X_{ud}, X_{dd}))|\ \beta_0 + \beta_1 = B\}$$

In practice, this is the path you follow:

$$\phi_0\left(\lessgtr\right): \textit{Expected value}$$

$$\phi_h\left(\lessgtr\right): min\left\{\varphi_0\left(\begin{matrix}\varphi_h(<)\\ <\\ \varphi_h(<)\end{matrix}\right), \varphi_h\left(\begin{matrix}\varphi_h(<)\\ \varphi_h(<)\end{matrix}\right)\right\}$$

$$\phi_B\left(\lessgtr\right): min\left\{\varphi_0\left(\begin{matrix}\varphi_B(<)\\ <\\ \varphi_B(<)\end{matrix}\right), \varphi_h\left(\begin{matrix}\varphi_{B-h}(<)\\ <\\ \varphi_{B-h}(<)\end{matrix}\right), \ldots, \varphi_{B-h}\left(\begin{matrix}\varphi_h(<)\\ <\\ \varphi_h(<)\end{matrix}\right), \varphi_B\left(\begin{matrix}\varphi_0(<)\\ <\\ \varphi_0(<)\end{matrix}\right)\right\}$$

You have to solve this problem, keeping the following instructions. Assume a linear grid of K+1 points for all beta-values, i.e., they are in the set {0,B/K,…B}. Take B=100 and K=100, but keep the procedures and functions for general B and integer K. When beta takes a value kB/K in some node, we say that k 'bits' (of risk aversion) are applied in that node.

You are given an empty object *BetaStar*. *BetaStar* is an array of array of array of double. This object is used to contain the optimal number of bits to apply in each node (t,j), conditioned on the amount of bits *k* assumed to be still available in that node, for all nodes, and all k=0,…,K. The corresponding entry is *BetaStar*(k+1,t+1,j).

Also provide the results for k=K in each node in *ResultArray*. These values will be plotted on the chart. Explain the interpretation of these figures, firstly the initial value *BetaStar*(K+1,0,1), and then the values *BetaStar*(K+1,t,j) for t > 0.

The crux is to determine *BetaStar* in each node from *BetaStar* in both child-nodes, backwardly in time. Hint: first be sure about all entries in *BetaStar* corresponding to time T-1, then first solve for one node at T-2 on paper. To help you, a MATLAB-implementation of TRA has been provided as well. In the end, the results should look similar to *figure 2,* in which β^* represents

the optimal bits of risk aversion of the total budget B. Note that we are applying the budget of risk aversion per path, not per node, stage or state. Hence, we evaluate the function per path.

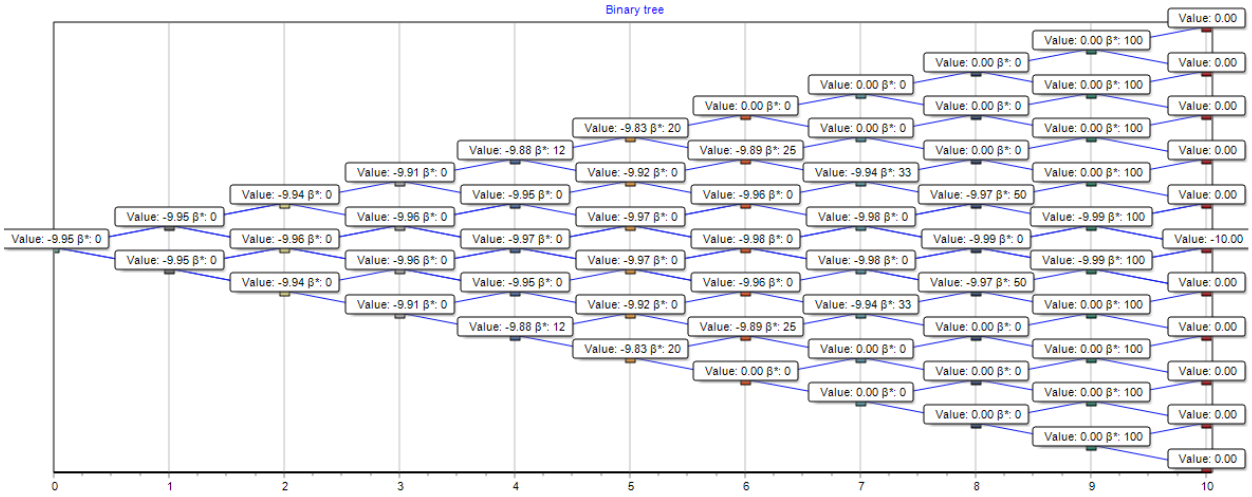

Figure 2, results of running the TRA algorithm

Compare the outcome *BetaStar*(K+1,0,1) with the outcome of applying the same number of bits of risk aversion in every step, B/T. Briefly comment on the difference.

Extra question, not obligatory: do you recognize a pattern in the outcome of Tuned Risk Aversion for the given position X_T? Can you generalize to other positions? Can you find positions for which the aforementioned difference is large?

4. Deliverables

As you may have noticed, canvas contains two separate grade modules. One is for programming. That is, you need to have this project signed off with respect to the Delphi code, just as the first and second programming assignment, during the programming sessions, however with the digital way of working due to the novel Corona virus, this will be done digitally as well. This is *"Programming Assignment 3"* and is about '*Have I coded according to the code conventions*'. As a guideline, you can use the Pascal style guide: https://edn.embarcadero.com/article/10280

The "*FEM – Programming"* assignment is the second module on canvas. This is about the results of the assignment. This is about how you implement the Dynamic Programming model, and how you use the Entropic Risk Function. Please deliver

- A zip file containing the whole project. Please make sure you upload the Delphi Project files (*.dpr and *.dproj), the resource file (*.res), the Delphi Form (*.dfm), and the Delphi Source file (*.pas). Also make sure that your program will run with those files located in the same directory.

- A report containing the explanation about the project, specifically the brief comment on the difference in the end result of the *BetaStar*-Array which is visualized in the tree.