POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Gestione del Rischio di Tasso di Interesse

Applicazioni di Programmazione Stocastica in ambito Asset-Liability Management

Relatore
prof. Paolo Brandimarte

Candidato
Daniele Plutino

Anno Accademico 2020-2021

Sommario

Questa tesi affronta come argomento principale la gestione del rischio di tasso di interesse all'interno di un problema di *asset-liability management*. In particolare, lo scopo è quello di costruire modelli di programmazione stocastica che siano in grado di fornire una strategia di investimento che permetta di non risultare insolventi. I modelli costruiti, a livello teorico, possono essere adattati sia ad un contesto di istituzione finanziaria (ad esempio banche, compagnie assicurative o fondi pensione) sia ad un contesto di investimenti individuali (*goal-based investing*).
Per lo sviluppo degli argomenti teorici si sono utilizzati diversi paper con temi relativi sia all'*asset-liability management* sia alla programmazione stocastica. I modelli più semplici sono stati presi dal libro del Prof. Brandimarte [1], mentre il modello più complesso è quello utilizzato dalla compagnia assicurativa giapponese The Yasuda Fire and Marine Insurance Company, Ltd., descritto in dettaglio in una serie di articoli riportati in bibliografia ([3], [2], [4]).
Per implementare i codici che risolvessero i modelli costruiti è stato utilizzato il software MATLAB nella sua release R2021a. In particolare sono state utilizzate alcune funzioni del pacchetto *Optimization toolbox*, mentre i pacchetti *Financial Toolbox* e *Financial Instruments Toolbox* sono stati utilizzati anche per alcune nozioni teoriche.
La struttura dell'elaborato consta di due parti: la prima in cui vi è un'introduzione al problema e la costruzione di modelli non elaborati per la sua risoluzione e la seconda nella quale si costruisce ed implementa un modello più complesso e si fanno alcune osservazioni sui risultati ottenuti. In particolare la prima parte si divide in due capitoli:

- il primo capitolo consta di un'introduzione al problema della gestione di un portafoglio con asset e passività e introduce brevemente i rischi a cui si è soggetti in questo contesto;
- il secondo capitolo introduce dei semplici modelli di programmazione stocastica (con dei brevi richiami su essa) che permettano di formalizzare e risolvere il problema introdotto nel primo capitolo.

La seconda parte consta, invece, di un unico capitolo: il capitolo tre contiene la costruzione teorica e numerica del modello più complicato e, a seguito della sua implementazione si fanno delle considerazioni sui risultati ottenuti e si traggono delle conclusioni.

Indice

Elenco delle tabelle

Elenco delle figure

Parte I

Prima Parte

Capitolo 1

Introduzione al problema

1.1 Il concetto di ALM

Le istituzioni finanziarie, banche, compagnie assicurative e fondi pensione soprattutto, si trovano quotidianamente a dover decidere come allocare le proprie risorse economiche. Spesso tali decisioni vengono prese con lo scopo di ridurre, o almeno mitigare, i rischi finanziari. Tipicamente un'istituzione finanziaria deve fronteggiare la possibilità che si osservi uno sbilanciamento tra il valore dei propri *asset*, cioè degli strumenti finanziari in grado di portare un guadagno, e quello delle proprie passività (*liabilities*), cioè di tutto ciò che l'istituzione finanziaria deve, per contratto, ad altri attori finanziari presenti sul mercato. L'espressione *asset-liability management*, spesso abbreviata in ALM, sta proprio ad indicare la gestione delle proprie risorse economiche da parte delle istituzioni finanziarie in modo da ridurre i rischi finanziari dovuti a un eventuale squilibrio tra *asset* e *liabilities*: in particolare l'istituzione finanziaria ha come scopo quello di riuscire a pagare tutte le proprie passività in tempo e un'ottimale gestione dei propri *asset* e dei flussi di cassa è il mezzo per raggiungere tale obiettivo.
Per comprendere meglio il significato del problema descritto è possibile considerare due esempi molto semplici:

- un fondo pensione eroga periodicamente (mensilmente in genere) un certo ammontare di denaro ai suoi clienti e deve assicurarsi di essere in grado di farlo in tempo ad ogni scadenza per tutti coloro che hanno un contratto;
- una compagnia di assicurazioni deve essere certa di riuscire a pagare i propri assicurati secondo contratto, nel caso in cui si realizzino le condizioni per cui ogni cliente era assicurato.

In entrambi i casi è evidente come l'istituzione finanziaria debba organizzare al meglio le proprie risorse economiche per essere in grado di pagare le passività al momento giusto. In questo senso si introduce il concetto di *solvency*: con tale termine si indica la capacità di una compagnia di ripagare i propri debiti entro le scadenza o in generale di adempiere ai propri doveri finanziari per tempo. Dunque lo scopo di una strategia di *asset-liability management* è quello di non far diventare la compagnia insolvente. In una buona strategia

di ALM, l'allocazione degli *asset* deve essere organizzata in modo da ottenere stabilità nel lungo periodo.
I due diversi esempi mettono anche in luce quanto vasto sia il campo dell'ALM: risulta evidente infatti come i flussi di cassa in uscita, ovvero le passività, possano essere, a seconda dei casi, deterministiche o stocastiche. Un fondo pensione generalmente eroga una pensione fissa ai clienti, mentre è chiaro come una compagnia assicurativa non abbia certezze nella conoscenza né dei tempi né delle quantità dei propri flussi in uscita.

1.2 Fattori di rischio

Lo squilibrio tra il valore degli *asset* e quello delle *liabilities* è spesso il risultato di cambiamenti nel panorama finanziario. A tale cambiamento possono contribuire diversi fattori, ciascuno dei quali può rappresentare una fonte di rischio per un'istituzione finanziaria. I principali fattori di rischio nell'ambito dell'*asset-liability management* sono descritti di seguito.

1.2.1 Rischio di mercato

Con rischio di mercato si fa riferimento alla possibilità di variazioni di fattori che interessano l'intero mercato finanziario: tra questi i più rilevanti sono i cambiamenti nei prezzi o nei tassi di mercato, dovuti alla loro volatilità. Tali variazioni di questi macro-fattori costituiscono una delle cause principali di aumento o decrescita nel valore di una posizione finanziaria. In particolare, poiché queste variazioni coinvolgono l'intero mercato, il rischio di mercato è un fattore di rischio nei confronti del quale non è possibile coprirsi tramite diversificazione. Per tale motivo il rischio di mercato è anche definito rischio sistematico. Il rischio di mercato può essere a sua volta scomposto in diverse tipologie di rischio:

- rischio di tasso di interesse;
- rischio azionario;
- rischio di posizione in merci (*commodity risk*);
- rischio valuta.

Diamo una breve spiegazione soltanto del tipo di rischio più interessante dal punto di vista dell'analisi che segue nei prossimi capitoli, cioè il rischio di tasso di interesse.

Rischio di tasso di interesse

Il rischio di tasso di interesse fa riferimento ai rischi associati a variazioni nei tassi di interesse e a come tali variazioni possano avere un impatto sui flussi di cassa futuri di un'istituzione finanziaria. Spesso gli *asset* e le *liabilities* facenti parte di un portafoglio sono strettamente dipendenti dai tassi di interesse del mercato: pertanto una variazione nel tasso di interesse può comportare una perdita. Ad esempio, nel caso in cui sul mercato si verificasse un aumento nel tasso di interesse, si avrebbe una corrispondente diminuzione

nel valore di un *bond* o di un analogo investimento di tipo *fixed-income*. Ciò è dovuto al fatto che quando un tasso di interesse sale, il costo-opportunità dovuto al possesso di quel *bond* sale: infatti se i tassi salgono, il tasso che si guadagna su un *bond* già sul mercato è meno consistente rispetto a quello che è possibile guadagnare su un nuovo *bond* appena emesso.

1.2.2 Rischio di liquidità

Con il termine liquidità si indica la facilità con cui un certo *asset* o un bene può essere convertito velocemente in denaro (cioè con cui può essere comprato o venduto sul mercato) senza impattare il proprio prezzo di mercato. Dunque il rischio di liquidità indica la possibilità che una posizione contenga investimenti poco liquidi, che cioè non possono essere venduti (o comprati) con facilità nel breve termine per prevenire o contenere eventuali perdite. Il rischio di liquidità insorge quando un investitore o un'istituzione finanziaria non è in grado di soddisfare le proprie obbligazioni a breve termine.

1.2.3 Rischio di credito

Il rischio di credito consiste nella possibilità di una perdita dovuta all'incapacità di un debitore di ripagare un prestito ricevuto o di soddisfare obblighi contrattuali. A livello pratico si riferisce alla possibilità che un'istituzione che presta del denaro possa non ricevere indietro il principale e/o gli interessi che le spettano dalla controparte che ha ricevuto il prestito: ciò può causare un'interruzione nei flussi di cassa in entrata.

1.2.4 Rischio operativo

Come definito dall'accordo Basilea II

> *il rischio operativo è definibile come il rischio di perdite derivanti dalla inadeguatezza o dalla disfunzione di procedure, risorse umane e sistemi interni, oppure da eventi esogeni.*

Dunque, il rischio operativo racchiude tutte le incertezze che una compagnia fronteggia nel portare avanti le proprie attività quotidiane: in questo senso i sistemi e le procedure aziendali, nonché i dipendenti stessi, costituiscono una possibile fonte di rischio. Pertanto il rischio operativo è a tutti gli effetti classificabile come rischio umano.

1.3 Goal-based investing

Il problema di *asset-liability management* può essere anche adattato al contesto di un singolo investitore. Spesso un privato investe il proprio denaro con lo scopo di incrementare la propria disponibilità economica per soddisfare delle esigenze economiche in specifici periodi di tempo. In questo contesto, il problema di ALM viene definito *goal-based investing* (GBI), cioè la questione consiste nel riuscire a gestire le proprie ricchezze attraverso investimenti che permettano di ottenere obiettivi specifici in determinati momenti. Alcuni esempi tipici di obiettivi per cui investire possono essere:

- il finanziamento dell'istruzione dei propri figli;
- la realizzazione di un certo ammontare di denaro per il pensionamento;
- il garantirsi un ingresso periodico fisso (mensile ad esempio) per integrare la pensione.

In questo genere di problema gli *asset* sono rappresentati dal capitale in possesso dell'investitore, mentre le passività sono costituite dai suoi diversi obiettivi: se il proprio scopo è garantirsi una pensione integrativa di 500 euro al mese, questi flussi di cassa possono essere considerate come *liability*, in quanto è necessario riuscire ad avere almeno 500 euro al termine di ogni mese per essere il linea con gli obiettivi prefissati.

Capitolo 2

Modelli di programmazione stocastica multistadio

2.1 Richiami di programmazione stocastica

Prima di costruire dei modelli di programmazione stocastica che formalizzino dal punto di vista matematico un problema di asset-liability management più o meno articolato, si danno dei brevi richiami di cosa si intende per ottimizzazione stocastica. In particolare si analizza la programmazione stocastica lineare con ricorso nel caso di due stadi e nel caso multistadio.

2.1.1 Programmazione stocastica lineare a due stadi con ricorso

Nel contesto della programmazione a due stadi ci si trova di fronte ad un problema con due istanti di tempo distinti in cui vanno prese delle decisioni che dipendono anche da alcune variabili aleatorie, o fattori di rischio, che si realizzano nel tempo. In particolare, nella programmazione con ricorso le informazioni che determinano le decisioni sono dinamiche nel tempo, pertanto tali decisioni non possono essere prese tutte all'inizio come si farebbe in un problema di programmazione multiperiodale senza incertezza.
Un problema di programmazione stocastica a due stadi con ricorso può essere rappresentato da un albero come quello in figura 2.1: la radice dell'albero corrisponde al primo stadio in cui vengono prese le decisioni iniziali. Il secondo stadio è invece rappresentato dalle foglie dell'albero in cui, dopo aver osservato la realizzazione dei fattori di rischio $\boldsymbol{\xi}$, si prendono le cosiddette decisioni di ricorso. Ogni foglia dell'albero costituisce uno dei possibili S *outcome* di realizzazione delle variabili aleatorie del problema in questione: ovviamente a ciascuna di esse è associata una probabilità e tali probabilità sommano a 1.

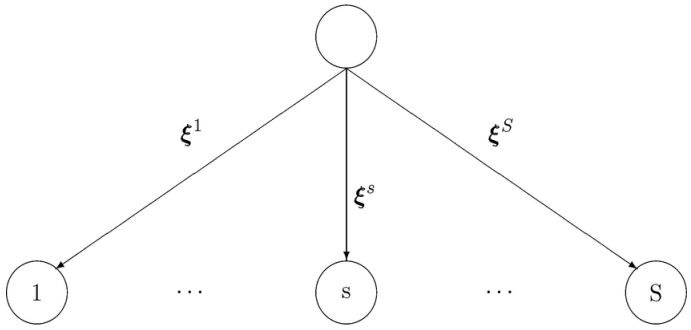


Figura 2.1. Albero per un problema di programmazione stocastica a due stadi

Formalmente un modello di programmazione stocastica lineare a due stadi con ricorso è caratterizzato da:

- le variabili decisionali al primo stadio $\mathbf{x} \geq \mathbf{0}$ che devono soddisfare dei vincoli lineari immediati del tipo $\mathbf{Ax} = \mathbf{b}$ e che hanno un certo impatto (sempre lineare) $\mathbf{c}^\intercal\mathbf{x}$ sull'obiettivo;

- le variabili aleatorie ξ che rappresentano i fattori di rischio del problema;

- le variabili decisionali al secondo stadio $\mathbf{y}(s) \geq \mathbf{0}$ che dipendono dalla realizzazione s dei fattori di rischio e che, quindi, all'istante iniziale sono di fatto delle variabili aleatorie;

- i vincoli tra gli stadi della forma $\mathbf{Wy}(s) + \mathbf{T}(s)\mathbf{x} = \mathbf{h}(s)$ che esprimono la dipendenza delle decisioni al secondo stadio da quelle al primo stadio e dalla realizzazione delle variabili aleatorie (essendo dei vincoli che contengono delle variabili aleatorie sarebbe necessario richiedere che tali vincoli siano soddisfatti quasi ovunque, ma nel caso di variabili aleatorie rappresentate da un insieme di scenari è sufficiente imporre che essi valgano in ognuno di tali scenari);

- l'impatto delle decisioni al secondo stadio sull'obiettivo $\mathbf{q}(s)^\intercal\mathbf{y}(s)$.

L'obiettivo di un problema di questo genere è quello di ottimizzare la somma tra l'impatto delle decisioni al primo stadio e il valore atteso dell'impatto delle decisioni al secondo stadio. Generalmente in un problema di questo tipo l'incertezza è modellata tramite un albero di scenari come quello in figura 2.1, cioè il vettore dei fattori di rischio $\boldsymbol{\xi}$ è discretizzata in un numero finito di punti $\{1, \ldots, s, \ldots, S\}$, a ciascuno dei quali è associata una probabilità $\pi_s \in (0,1)$. Qualora il problema sia di minimizzazione dei costi, indicando

con il pedice s la dipendenza dallo scenario, il modello che lo rappresenta può essere formalizzato come segue:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \mathbf{c}^\intercal \mathbf{x} + \sum_{s=1}^{S} \pi_s \mathbf{q}_s^\intercal \mathbf{y}_s \\ \text{tale che} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{W}\mathbf{y}_s + \mathbf{T}_s \mathbf{x} = \mathbf{h}_s, \quad \forall s \in S. \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{y}(s) \geq \mathbf{0}. \end{aligned}$$

Nei casi più complessi, che vanno oltre lo scopo di questo elaborato, anche la matrice di ricorso $\mathbf{W}$ può dipendere dalla realizzazione dei fattori di rischio.

2.1.2 Programmazione stocastica lineare multistadio con ricorso

I problemi di programmazione stocastica multistadio costituiscono la generalizzazione dei problemi a due stadi della sezione precedente: in questo caso la sequenza di decisioni da prendere nel tempo è costituita da più di due istanti decisionali (in generale si indica con T l'orizzonte temporale del problema), pertanto le variabili decisionali stesse sono annidate tra loro. Anche in questo caso, discretizzando i fattori di rischio in ogni intervallo di tempo a un insieme finito di scenari, il problema può essere rappresentato da un albero con un numero di livelli pari al numero di istanti di tempo del problema.
Un problema di questo tipo è caratterizzato dal seguente schema:

- le variabili decisionali al primo stadio $\mathbf{x}_0$ vengono prese nella radice dell'albero: esse devono soddisfare un vincolo deterministico iniziale della forma $\mathbf{A}_{00}\mathbf{x}_0 = \mathbf{b}_0$ e hanno un impatto sull'obiettivo pari a $\mathbf{c}_0^\intercal \mathbf{x}_0$;
- al secondo livello dell'albero si osserva la realizzazione dei fattori di rischio nel primo periodo, la quale fornisce dei dati $\mathbf{A}_{10}, \mathbf{A}_{11}, \mathbf{b}_1, \mathbf{c}_1$: dopo aver osservato tali dati si prende la decisione $\mathbf{x}_1$ che deve soddisfare il vincolo $\mathbf{A}_{10}\mathbf{x}_0 + \mathbf{A}_{11}\mathbf{x}_1 = \mathbf{b}_1$ e ha un impatto sull'obiettivo pari a $\mathbf{c}_1^\intercal \mathbf{x}_1$;
- tale struttura si ripete per tutti i periodi fino all'istante temporale $T-1$ (dove T è l'orizzonte temporale del problema);
- alla scadenza T si osservano le realizzazioni $\mathbf{A}_{T,T-1}, \mathbf{A}_{TT}, \mathbf{b}_T, \mathbf{c}_T$ e si prende l'ultima decisione $\mathbf{x}_T$ che ha un impatto pari a $\mathbf{c}_T^\intercal \mathbf{x}_T$ e deve soddisfare il vincolo $\mathbf{A}_{T,T-1}\mathbf{x}_{T-1} + \mathbf{A}_{TT}\mathbf{x}_T = \mathbf{b}_T$.

In questo schema i vincoli e l'impatto di ogni decisione (cioè le grandezze $\mathbf{A}, \mathbf{b}$ e $\mathbf{c}$) dipendono in qualche modo dal processo stocastico $\boldsymbol{\xi}_t, t = 1, \ldots, T$. Supponendo, di nuovo, che il problema sia di minimizzazione dei costi, esso può essere formalizzato come segue:

$$\min_{\mathbf{A}_{00}\mathbf{x}_0 = \mathbf{0}} \mathbf{c}_0^\intercal \mathbf{x}_0 + \mathbb{E}\left[\min_{\substack{\mathbf{A}_{10}\mathbf{x}_0 + \mathbf{A}_{11}\mathbf{x}_1 = \mathbf{b}_1 \\ \mathbf{x}_1 \geq \mathbf{0}}} \mathbf{c}_1^\intercal \mathbf{x}_1 + \mathbb{E}\Big[\cdots + \mathbb{E}\Big[\min_{\substack{\mathbf{A}_{T,T-1}\mathbf{x}_{T-1} + \mathbf{A}_{TT}\mathbf{x}_T = \mathbf{b}_T \\ \mathbf{x}_T \geq \mathbf{0}}} \mathbf{c}_T^\intercal \mathbf{x}_T \Big] \Big] \right] \quad (2.2)$$

Anche in questo caso, ponendosi nella radice dell'albero, le decisioni $\mathbf{x}_1, \ldots, \mathbf{x}_T$ sono a tutti gli effetti delle variabili aleatorie, in quanto esse dipendono dalla realizzazione del processo stocastico $\boldsymbol{\xi}_t$: in questo senso le decisioni si dicono non-anticipative.

2.2 Applicazione ad un semplice problema di ALM

2.2.1 Descrizione del modello

Si consideri ora un primo semplice esempio di modellazione attraverso la programmazione stocastica di un problema di asset-liability management in cui vi è un'unica passività da soddisfare alla scadenza dell'ultimo periodo temporale e i rendimenti degli asset hanno soltanto 2 possibili scenari. Il problema, descritto in seguito, è preso dal capitolo 15 del libro di Brandimarte [1]. Di seguito la descrizione dettagliata del problema nella sua forma più generale.

- Si ha una ricchezza iniziale W_0, che può essere investita in un certo insieme $\mathcal{A}$ di asset.

- Ogni asset $i \in \mathcal{A}$ ha un rendimento incerto R_i che può assumere due valori diversi con la stessa probabilità pari a $\frac{1}{2}$. Gli asset da considerare devono avere rendimenti tali che per qualsiasi coppia di asset dell'insieme $\mathcal{A}$ nessuno dei due domini l'altro e che non ci siano opportunità di arbitraggio. Inoltre i rendimenti degli asset si "muovono" insieme nel senso che o si verifica lo scenario positivo o quello negativo per tutti gli asset dell'insieme $\mathcal{A}$.

- Si vuole riuscire a generare una ricchezza in grado di soddisfare una certa passività L al termine dell'orizzonte temporale di T anni.

- Il portafoglio può essere ribilanciato al termine di ogni anno, cioè agli istanti di tempo $t = 0,1,\ldots,T$.

Lo scopo è quello di soddisfare la passività ed eventualmente avere un surplus, contenendo il rischio di "andare corti", cioè di avere uno shortfall sulla passività. Per modellare tale avversione al rischio una scelta semplice consiste nel considerare una funzione di utilità che sia lineare a tratti: il punto in cui cambia la pendenza della retta è il punto in cui la ricchezza finale W_T è esattamente pari alla passività L. Inoltre il coefficiente r che penalizza lo shortfall ω_- è maggiore rispetto al coefficiente q che "premia" il surplus ω_+: questa scelta per la pendenza dei due tratti di retta permette di rappresentare la necessità di pagare la passività piuttosto che la volontà di avere un profitto. Tale funzione di utilità è rappresentata in figura 2.2.

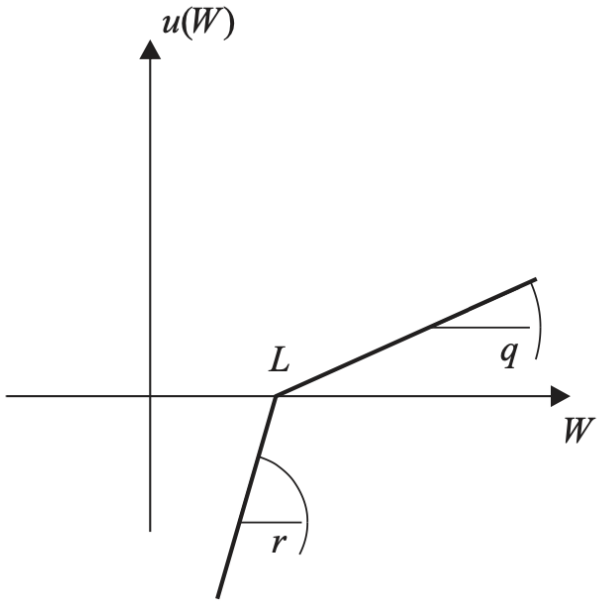
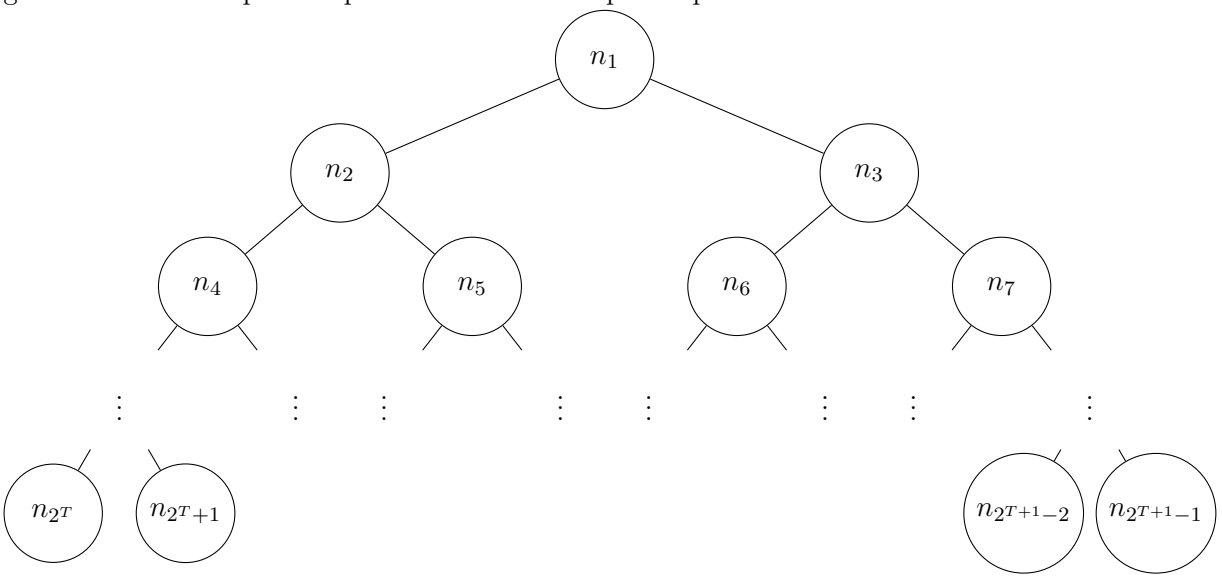


Figura 2.2. Funzione di utilità lineare a tratti.

Per rappresentare l'incertezza sui rendimenti degli investimenti è necessario costruire un albero di scenari, in cui ogni scenario della ricchezza terminale è rappresentato da una foglia e ogni livello dell'albero costituisce un istante di tempo in cui il portafoglio può essere ribilanciato. L'albero in questo caso è binario, dal momento che gli investimenti possono avere soltanto due possibili esiti. Di seguito la rappresentazione grafica di un generico albero di questo tipo con orizzonte temporale pari a T anni.



Il nodo radice n_1 è quello in cui si stabilisce l'allocazione iniziale ($t = 0$) delle risorse economiche. I nodi $\mathcal{I} = \{n_2, n_3, \ldots, n_{2^T-1}\}$ sono i nodi intermedi in cui il portafoglio può essere ribilanciato. Le foglie dell'albero formano l'insieme dei possibili scenari per la ricchezza terminale $\mathcal{S} = \{n_{2^T}, n_{2^T+1}, \ldots, n_{2^{T+1}-1}\}$ in cui si confronta la ricchezza ottenuta con la passività da pagare e si calcola l'utilità. In questo caso, poiché ogni nodo si ramifica in 2 scenari di probabilità $\frac{1}{2}$, gli scenari sono tutti equiprobabili con probabilità $\frac{1}{2^T}$. Sono indicati con un apice s le grandezze dipendenti dallo scenario $s \in \mathcal{S}$: le notazioni ω_-^s e ω_+^s rappresentano rispettivamente lo shortfall e il surplus nello scenario s. Inoltre p^s indica la probabilità che si realizzi lo scenario terminale s, cioé di raggiungere il corrispondente nodo foglia.
Le variabili decisionali del problema sono costituite dalle diverse quantità di denaro allocate in ciascun asset $i \in \mathcal{A}$ in ogni nodo che non sia una foglia: tale grandezza è indicata con $x_{i,n}$, dove $n \in \{n_1, \ldots, n_{2^T-1}\}$ rappresenta il nodo dell'albero.
Le variabili aleatorie del problema, cioè i fattori di rischio, sono rappresentate dai rendimenti degli asset: la notazione $R_{i,n}$ indica il rendimento moltiplicativo dell'investimento i nel periodo che termina nel nodo n.
Indicando con la notazione $a(n)$ il nodo antecedente, cioè il genitore, del nodo n, il problema di ALM si può formalizzare come segue:

$$
\begin{aligned}
\max_{\mathbf{x}} \quad & \sum_{s\in\mathcal{S}} p^s(q\omega_+^s - r\omega_-^s) \\
\text{tale che} \quad & \sum_{i\in\mathcal{A}} x_{i,n_1} = W_0, \\
& \sum_{i\in\mathcal{A}} R_{i,n} x_{i,a(n)} = \sum_{i\in\mathcal{A}} x_{i,n}, \quad \forall n \in \{n_1, \ldots, n_{2^T-1}\} \\
& \sum_{i\in\mathcal{A}} R_{i,s} x_{i,a(s)} = L + \omega_+^s - \omega_-^s, \quad \forall s \in \mathcal{S} \\
& x_{i,n} \geq 0, \quad \forall i \in \mathcal{A}, \forall n \in \{n_1, \ldots, n_{2^T-1}\} \\
& \omega_+^s, \omega_-^s \geq 0, \quad \forall s \in \mathcal{S}.
\end{aligned}
$$

Nei nodi in cui il portafoglio può essere ribilanciato si può pensare a tale ribilanciamento come una sequenza di due step:

- tutto il portafoglio viene liquidato e si ottiene il denaro corrispondente alla somma delle posizioni nei diversi asset;
- si rialloca tutto il denaro a disposizione negli asset secondo la politica che risolve il problema di ottimizzazione.

2.2.2 Implementazione del modello in MATLAB

Tale modello è stato implementato in MATLAB nella versione R2021a con i seguenti dati:

- orizzonte temporale di $T = 3$ anni;

- due possibili asset in cui investire con rendimenti moltiplicativi rispettivamente pari a 1.25 e 1.14 nello scenario positivo e 1.06 e 1.12 nello scenario negativo (si noti come nessuno dei due investimenti domina l'altro: il primo ha un rendimento più volatile, mentre il secondo rendimento oscilla poco);
- ricchezza iniziale di $W_0 = \$55000$;
- passività da pagare alla maturità pari a $L = \$80000$;
- coefficiente di penalizzazione dello shortfall $r = 4$;
- coefficiente di premio del surplus $q = 1$.

Tali dati chiaramente non hanno alcuna origine reale ma sono soltanto utili a verificare come questo semplice problema lineare di ALM può essere risolto computazionalmente. Il codice implementato è il seguente:

```
clear all
format bank
%% PROBLEM DATA
rootNode = 1;
tradingNodes = 2:7;
leafNodes = 8:15;
numNodes = 15;
numScenarios = 8;
numTradingNodes = length(tradingNodes);
antecedentNode = [NaN 1 1 2 2 3 3 4 4 5 5 6 6 7 7];
numAssets = 2;
returns = [ NaN NaN;
    repmat([1.25 1.14; 1.06 1.12], 7,1)];
wealth_zero = 55000;
target = 80000;
shortfallCoeff = 4;
surplusCoeff = 1;
probs = 1/numScenarios;

%%
probALM = optimproblem('ObjectiveSense','max');
hold = optimvar('hold', numAssets, numNodes,  'LowerBound', 0);
shortfall = optimvar('shortfall', numScenarios, 1,...
    'LowerBound', 0);
surplus = optimvar('surplus', numScenarios, 1,  'LowerBound', 0);
% objective is expected 'utility'
probALM.Objective = dot(probs, sum(surplusCoeff*surplus -...
    shortfallCoeff*shortfall));
% initial wealth
probALM.Constraints.initWealth = sum(hold(:,rootNode)) ==...
    wealth_zero;
% rebalancing at trading nodes
rebalance = optimconstr(numTradingNodes,1);
```

```
for k = 1:numTradingNodes
    thisNode = tradingNodes(k);
    prevNode = antecedentNode(thisNode);
    rebalance(k) = sum(hold(:,thisNode)) ==...
        dot(returns(thisNode,:),hold(:,prevNode));
end
probALM.Constraints.rebalance = rebalance;
% check result at leaf nodes
check = optimconstr(numScenarios,1);
for k = 1:numScenarios
    thisNode = leafNodes(k);
    prevNode = antecedentNode(thisNode);
    check(k) = dot(returns(thisNode,:), hold(:,prevNode)) ==...
        target + surplus(k) - shortfall(k);
end
probALM.Constraints.check = check;
showproblem(probALM)
solALM = solve(probALM);

%% show solution
sol = solALM.hold'
```

Il problema è risolto da MATLAB attraverso l'uso della funzione *linprog* che risolve i problemi di programmazione lineare. La soluzione trovata è la seguente:

```
sol =

      41479.27      13520.73
      65094.58       2168.14
      36743.22      22368.03
      83839.90             0
             0      71428.57
             0      71428.57
      64000.00             0
             0             0
             0             0
             0             0
             0             0
             0             0
             0             0
             0             0
```

Riscriviamo la soluzione ottenuta in output su MATLAB in maniera più chiara e comprensibile per un eventuale investitore che si trovi a dover affrontare il problema. La tabella 2.1 indica ad ogni istante di tempo in che modo investire nei due asset il denaro che si possiede a seconda dello scenario che si realizza.

Istante di tempo	Nodo	$ in asset 1	$ in asset 2
$t = 0$	n_1	41479.27	13520.73
$t = 1$	n_2	65094.58	2168.14
	n_3	36743.22	22368.03
$t = 2$	n_4	83839.90	0
	n_5	0	71428.57
	n_6	0	71428.57
	n_7	64000.00	0

Tabella 2.1. Politica ottimale di investimento che risolve il modello.

Vediamo meglio come leggere la tabella. La soluzione dice che all'istante iniziale la strategia ottimale è quella di dividere i $55000 iniziali investendone $41479.27 nel primo asset e $13520.73 nel secondo. Dopo il primo intervallo di tempo la strategia ottimale da seguire è la seguente:

- nel caso in cui si realizzi lo scenario favorevole e si finisca nel nodo n_2 investire $65094.58 nel primo asset e $2168.14 nel secondo;
- nel caso in cui si realizzi lo scenario sfavorevole e si finisca nel nodo n_3 investire $36743.22 nel primo asset e $22368.03 nel secondo.

Si noti come la disponibilità di denaro totale nel nodo n_2 è maggiore rispetto a quella nel nodo n_3, perché i rendimenti sono stati maggiori per entrambi gli asset. Allo stesso modo è possibile leggere la strategia da applicare al termine del secondo intervallo temporale. Nella tabella non compare l'istante di tempo $t = T = 3$ in quanto, come mostra l'ouput di MATLAB, al termine dell'orizzonte temporale il portafoglio non è più ribilanciato, ma si paga la passività e si osserva un eventuale surplus di denaro.
Risulta interessante osservare lo shortfall ed il surplus nei nodi foglia dell'albero, cioè al termine dell'orizzonte temporale. Lo shortfall risulta essere:

```
solALM.shortfall =

             0
             0
             0
             0
             0
             0
             0
      12160.00
```

mentre il surplus è dato da:

```
solALM.surplus =

     24799.88
       8870.30
       1428.57
             0
```

```
1428.57
      0
      0
      0
```

Pertanto negli stati terminali si ha un surplus in ben quattro casi su otto possibili scenari, mentre si ha uno shortfall, che risulta anche piuttosto consistente, soltanto nel nodo più sfavorevole. Negli altri casi si è comunque solventi ma senza avere alcun surplus.

2.3 Applicazione ad un problema generalizzato di ALM

2.3.1 Descrizione del modello

A partire dal modello descritto nella sezione precedente è possibile definire un problema di ALM più complesso e generale. In particolare possono essere considerate le seguenti complicazioni rispetto al modello precedente.

- Si può considerare un flusso di passività periodiche anziché un'unica passività al termine dell'orizzonte temporale: in particolare tale flusso può essere deterministico (e in tal caso è rappresentato con la notazione $L_0, L_1, \ldots, L_T$) oppure stocastico (e in questo secondo caso si usa la notazione L^n con n nodo dell'albero).
- È possibile aggiungere un costo di transazione percentuale $c \in (0,1)$: ciò comporta il fatto che, vendendo un asset, anziché incassare il suo prezzo di mercato per intero, si incassa una frazione pari a $1 - c$; analogamente, acquistando un asset, si paga il prezzo di mercato moltiplicato per un fattore pari a $1 + c$.
- Si può introdurre come utilità una generica funzione non lineare della ricchezza finale $u(W)$.

Tale modello ha in comune con quello di prima la struttura ad albero dovuta all'aleatorietà dei rendimenti degli investimenti che viene rappresentata al termine di ogni periodo da un insieme di scenari. Anche in questo caso l'albero di scenari considerato è binario. Un altro aspetto comune con il modello di prima è la funzione obiettivo: anche in questo caso si vuole massimizzare l'utilità attesa della ricchezza finale, seppur essa non sia necessariamente lineare a tratti. La differenza principale con il primo modello consiste nel fatto che, avendo introdotto un costo di transazione, non è più sufficiente limitarsi a considerare il denaro allocato in ciascun asset: diventa necessario infatti tenere conto per ciascun asset dell'ammontare di denaro posseduto in esso, di quanto ne viene acquistato e di quanto ne viene venduto in ogni nodo. Pertanto le variabili decisionali necessarie alla formalizzazione matematica del problema sono le seguenti:

- x_i^n che indica la quantità di asset i posseduta nel nodo n dopo aver ribilanciato il portafoglio;
- y_i^n che indica la quantità di asset i venduta nel nodo n;
- z_i^n che indica la quantità di asset i acquistata nel nodo n.

Chiaramente, come nel caso precedente, queste variabili sono definite per i nodi che non sono foglie, dal momento che queste ultime costituiscono il termine dell'orizzonte temporale.
Un'altra differenza rispetto al modello precedente sta nel fatto che, anziché considerare un ammontare di denaro a disposizione per gli investimenti, in questo caso si presuppone che all'istante $t = 0$ si abbia già tutto il denaro allocato negli asset disponibili. A causa dell'introduzione dei costi di transazione risulta anche necessario tenere conto dei prezzi degli asset piuttosto che del

loro rendimento moltiplicativo: il prezzo dell'asset i nel nodo n dell'albero è indicato con P_i^n. Indicando con la notazione $\bar{h}_i^{n_1}$ il denaro allocato inizialmente (cioè al tempo $t = 0$) nell'asset i e mantenendo le notazioni del modello precedente per ciò che non varia, il modello può essere formalizzato come segue:

$$
\begin{aligned}
\max_x \quad & \sum_{s\in\mathcal{S}} p^s u(W^s) \\
\text{tale che} \quad & x_i^{n_1} = \bar{h}_i n_1 + z_i^{n_1} - y_i^{n_1}, \quad \forall i \in \mathcal{A} \\
& x_i^n = x_i a(n) + z_i^n - y_i^n, \quad \forall i \in \mathcal{A}, \forall n \in \{n_2, \dots, n_{2^T-1}\} \\
& (1-c)\sum_{i\in\mathcal{A}} P_i^n z_i^n - (1+c)\sum_{i\in\mathcal{A}} P_i^n z_i^n = L^n, \quad \forall n \in \{n_1, \dots, n_{2^T-1}\} \\
& W^s = (1-c)\sum_{i\in\mathcal{A}} P_i^s x_i^{a(s)} - L^s, \quad \forall s \in \mathcal{S} \\
& W^s \geq 0, \quad \forall s \in \mathcal{S} \\
& x_i^n, y_i^n, z_i^n \geq 0, \quad \forall i \in \mathcal{A}, \forall n \in \{n_1, \dots, n_{2^T-1}\}
\end{aligned}
$$

2.3.2 Implementazione del modello in MATLAB

Tale modello è stato implementato in MATLAB nella versione R2021a con dei dati che lo rendessero confrontabile con il problema della sezione precedente. In particolare, quindi i dati sono:

- orizzonte temporale di $T = 3$ anni;
- un insieme di due possibili asset in cui investire con prezzo iniziale di \$100 per entrambi e rendimenti moltiplicativi rispettivamente pari a 1.25 e 1.14 nello scenario positivo e 1.06 e 1.12 nello scenario negativo, con i due scenari di nuovo equiprobabili (si noti come anche in questo caso nessuno dei due investimenti domina l'altro e che non c'è uno scenario in cui favorevole per uno dei due e sfavorevole per l'altro);
- ricchezza iniziale di $W_0 =$ \$55000 suddivisa equamente in 275 unità del primo asset e 275 unità del secondo asset all'interno del portafoglio;
- passività deterministiche da pagare in ogni periodo, compreso l'istante $t = 0$, pari a $L =$ \$15000 per periodo (pertanto vi sono 4 flussi in uscita ai tempi $t =$ 0,1,2,3);
- costo percentuale di transazione del 5% sia per gli acquisti sia per le vendite;
- funzione di utilità data dalla radice quadrata della ricchezza terminale.

Tali dati, di nuovo, non hanno alcuna origine reale. Il codice implementato è il seguente:

```
%% PROBLEM DATA
rootNode = 1;
tradingNodes = 2:7;
leafNodes = 8:15;
numNodes = 15;
numScenarios = 8;
numTradingNodes = length(tradingNodes);
antecedentNode = [NaN 1 1 2 2 3 3 4 4 5 5 6 6 7 7];
numAssets = 2;
initHold = [275 275];
```

```
probs = repmat(1/numScenarios, numScenarios, 1);
transCostPercentage = 0.05;
high = [1.25 1.14];
low = [1.06 1.12];
prices = zeros(numNodes, numAssets);
prices (1,1) = 100;
prices (1,2) = 100;
prices (2,1) = prices (1,1)*high(1);
prices (2,2) = prices (1,2)*high(2);
prices (3,1) = prices (1,1)*low(1);
prices (3,2) = prices (1,2)*low(2);
prices (4,1) = prices (2,1)*high(1);
prices (4,2) = prices (2,2)*high(2);
prices (5,1) = prices (2,1)*low(1);
prices (5,2) = prices (2,2)*low(2);
prices (6,1) = prices (3,1)*high(1);
prices (6,2) = prices (3,2)*high(2);
prices (7,1) = prices (3,1)*low(1);
prices (7,2) = prices (3,2)*low(2);
prices (8,1) = prices (4,1)*high(1);
prices (8,2) = prices (4,2)*high(2);
prices (9,1) = prices (4,1)*low(1);
prices (9,2) = prices (4,2)*low(2);
prices (10,1) = prices (5,1)*high(1);
prices (10,2) = prices (5,2)*high(2);
prices (11,1) = prices (5,1)*low(1);
prices (11,2) = prices (5,2)*low(2);
prices (12,1) = prices (6,1)*high(1);
prices (12,2) = prices (6,2)*high(2);
prices (13,1) = prices (6,1)*low(1);
prices (13,2) = prices (6,2)*low(2);
prices (14,1) = prices (7,1)*high(1);
prices (14,2) = prices (7,2)*high(2);
prices (15,1) = prices (7,1)*low(1);
prices (15,2) = prices (7,2)*low(2);
liability = [15000; repmat(15000, 2, 1); repmat(15000, 4, 1);...
    repmat(15000, 8, 1)];
%%
probALM = optimproblem('ObjectiveSense','max');
hold = optimvar('hold', numAssets, numNodes-numScenarios,...
    'LowerBound', 0);
purchase = optimvar('purchase', numAssets, numNodes-...
    numScenarios, 'LowerBound', 0);
sell = optimvar('sell', numAssets, numNodes-numScenarios,...
    'LowerBound', 0);
terminalWealth = optimvar('terminalWealth', numScenarios, 1,...
    'LowerBound', 0);
% objective is expected 'utility' (utility=sqrt)
probALM.Objective = dot(probs, sqrt(terminalWealth));
```

```
% initial wealth
initWealth = optimconstr(numAssets,1);
for k=1:numAssets
    initWealth(k) = hold(k, rootNode)==initHold(k) +...
        purchase(k, rootNode) - sell(k, rootNode);
end
probALM.Constraints.initWealth = initWealth;
% equilibrium at trading nodes for every asset
equilibrium = optimconstr(numAssets, numTradingNodes);
for k = 1:numAssets
    for j = 1:numTradingNodes
        thisNode = tradingNodes(j);
        prevNode = antecedentNode(thisNode);
        equilibrium(k,j) = hold(k,thisNode) == hold(k,...
            prevNode) + purchase(k, thisNode) -...
            sell(k, thisNode);
    end
end
probALM.Constraints.equilibrium = equilibrium;
% transactions to match liabilities
transaction = optimconstr(numNodes-numScenarios, 1);
for n=1:numNodes-numScenarios
    transaction(n) = (1-transCostPercentage)*dot(prices(n,:),...
        sell(:,n))-(1+transCostPercentage)*dot(prices(n,:),...
        purchase(:,n)) == liability(n);
end
probALM.Constraints.transaction = transaction;
% final wealth
finalWealth = optimconstr(numScenarios,1);
for k = 1:numScenarios
    thisNode = leafNodes(k);
    prevNode = antecedentNode(thisNode);
    finalWealth(k) = terminalWealth(k) ==...
        (1-transCostPercentage)*dot(prices(thisNode,:),...
        hold(:,prevNode)) - liability(k+7);
end
probALM.Constraints.finalWealth = finalWealth;
showproblem(probALM)
x0.hold = zeros(numAssets, numNodes-numScenarios);
x0.purchase = zeros(numAssets, numNodes-numScenarios);
x0.sell = zeros(numAssets, numNodes-numScenarios);
x0.terminalWealth = zeros(numScenarios, 1);
solALM = solve(probALM, x0);

%% show solution
solHold = round(solALM.hold,2)
solPurchase = round(solALM.purchase,2)
solSell = round(solALM.sell,2)
```

Il problema è risolto da MATLAB attraverso l'uso della funzione *fmincon* che risolve alcuni problemi di programmazione non lineare. La soluzione trovata è la seguente:

```
solPurchase =

             0             0
             0             0
             0             0
             0             0
             0             0
             0             0
             0             0


solSell =

        152.23          5.67
             0        138.50
         69.75         74.97
             0        121.49
             0        123.66
             0        123.66
         27.21        101.50


solHold =

        122.77        269.33
        122.77        130.83
         53.02        194.37
        122.77          9.33
        122.77          7.16
         53.02         70.70
         25.82         92.86
```

Prima di riportare la soluzione in forma tabulare, è subito possibile osservare dall'output di MATLAB che la soluzione del problema non prevede in alcun nodo dell'albero di acquistare nessuno dei due asset a disposizione. Ciò è dovuto al fatto che in questo modello la ricchezza iniziale è già tutta investita nei due asset, pertanto la soluzione evidenzia come in ogni scenario di ciascun livello sia sufficiente soltanto vendere una certa quantità dei due asset che permetta di pagare la passività del relativo livello. Nella tabella 2.2 si riportano pertanto soltanto le variabili *solHold* e *solSell*.

Istante di tempo	Nodo	Asset 1		Asset 2	
		Sell	Hold	Sell	Hold
$t = 0$	n_1	152.23	122.77	5.67	269.33
$t = 1$	n_2	0	122.77	138.50	130.83
	n_3	69.75	53.02	74.97	194.37
$t = 2$	n_4	0	122.77	121.49	9.33
	n_5	0	122.77	123.66	7.16
	n_6	0	53.02	123.66	70.70
	n_7	27.21	25.82	101.50	92.86

Tabella 2.2. Politica ottimale di investimento che risolve il modello.

La tabella questa volta non mostra quantità di denaro ma unità di asset che vengono vendute in ogni nodo e che si possiedono dopo il ribilanciamento. La vendita iniziale di entrambi gli asset nel nodo radice è dovuta al fatto che la prima passività si ha subito al tempo $t = 0$, a differenza del modello precedente in cui c'era solo una passività terminale e il denaro non era già allocato nei diversi asset. La soluzione evidenzia come, a differenza del modello precedente, in questo caso non possiamo immaginare di liquidare tutto il portafoglio ogni volta che questo deve essere ribilanciato: ciò è ovviamente dovuto alla presenza di costi di transazione che prima erano assenti. Pertanto viene liquidata solo una certa porzione di portafoglio necessaria a pagare la passività immediata.

2.4 Russell-Yasuda-Kasai per il problema di ALM

2.4.1 Descrizione del modello

Un modello più completo dei precedenti, il quale risulta anche essere utilizzato nella pratica, è il modello di Russell-Yasuda-Kasai formulato in origine per una compagnia assicurativa giapponese. Il modello completo è descritto dagli articoli [3], [2], [4] e formalizzato completamente nell'appendice dell'articolo [3]. La versione del problema implementata in questa tesi contiene delle semplificazioni teoriche per allinearlo ai modelli precedenti. In particolare si fanno le seguenti due ipotesi:

- le passività sono deterministiche e la prima si ha soltanto al termine del primo periodo (cioè $L_0 = 0$);
- non si hanno né *payout* né flussi di cassa in entrata (*deposit inflow*) .

Pertanto il modello presente nell'articolo [3], sotto queste assunzioni, prende la seguente forma:

$$\max_{\mathbf{X}} \quad \sum_{s\in\mathcal{S}} p^s V^s - r\sum_{n\in\mathcal{N}} p^n \omega_-^n \tag{2.5a}$$

$$\text{tale che} \quad V^n = \sum_{i\in\mathcal{A}} R_{i,n} X_{i,a(n)}, \quad \forall n = 2,\dots,2^{T+1}-1 \tag{2.5b}$$

$$\sum_{i\in\mathcal{A}} X_{i,n} = V^n - L^n, \quad \forall n \in \mathcal{N}\setminus\mathcal{S} \tag{2.5c}$$

$$V^n - L^n = \omega_+^n - \omega_-^n \quad \forall n = 2,\dots,2^{T+1}-1 \tag{2.5d}$$

$$V_1 = \omega_+^1 = W_0, \tag{2.5e}$$

$$\omega_-^1 = 0, \tag{2.5f}$$

$$\omega_-^n, \omega_+^n \geq 0, \quad \forall n \in \mathcal{N} \tag{2.5g}$$

$$X_{i,n} \geq 0, \quad \forall i \in \mathcal{A}, \forall n \in \mathcal{N}\setminus\mathcal{S} \tag{2.5h}$$

La notazione di tale modello è la seguente:

- $\mathcal{N}$ è l'insieme dei nodi dell'albero;
- $\mathcal{S}$ è l'insieme dei nodi foglia dell'albero, cioè degli scenari terminali;
- W_0 è la ricchezza iniziale;
- T è l'orizzonte temporale, che coincide con l'altezza dell'albero;
- L^n è la passività che deve essere pagata nel nodo n dell'albero (in realtà le passività sono deterministiche pertanto esse dipendono soltanto dal livello dell'albero e non dal singolo nodo);
- $r > 0$ è il coefficiente di penalizzazione dello shortfall;
- la variabile decisionale $\mathbf{X}$ rappresenta l'ammontare di denaro che si possiede nei diversi asset, cioè $X_{i,n}$ è il denaro investito nell'asset i nel nodo n dell'albero;
- il rischio è rappresentato dalle variabili aleatorie che individuano i rendimenti: $R_{i,n}$ è il rendimento moltiplicativo dell'asset i nel periodo che termina nel nodo n dell'albero;
- V^n sta ad indicare il valore del portafoglio che si detiene nel nodo n prima di pagare la passività L^n; si noti come il portafoglio nel nodo radice abbia valore pari alla ricchezza iniziale W_0 come evidenziato dal vincolo 2.5e;
- ω_-^n e ω_+^n rappresentano rispettivamente lo shortfall e il surplus nel nodo n dell'albero; si noti come lo shortfall iniziale è nullo (vincolo 2.5f), mentre il surplus iniziale coincide con la ricchezza di partenza (2.5e), dal momento che al tempo $t = 0$ non c'è alcuna passività da pagare;
- infine p^n rappresenta la probabilità di arrivare nel nodo n dell'albero partendo dalla radice, pertanto p^s indica la probabilità dello scenario terminale s.

Dunque in questo modello la funzione obiettivo 2.5a contiene un primo termine che va a premiare il valore del portafoglio al termine dell'orizzonte temporale del problema, cioè dopo aver pagato tutte le passività, e un secondo termine che penalizza eventuali shortfall in qualsiasi nodo dell'albero, cioè in qualsiasi scenario di ogni livello. Questo tipo di obiettivo serve nella pratica ad evitare di privilegiare strategie con un valore atteso finale molto grande che abbiano, però, una certa probabilità di risultare insolventi in un istante intermedio qualora si realizzi uno degli scenari meno favorevoli. Per tale motivo, spesso il coefficiente r che penalizza l'insolvenza è

scelto in modo tale da essere maggiore rispetto a quello che premia la ricchezza finale, che in questo caso è pari a 1.
Il vincolo 2.5b stabilisce che il valore del portafoglio nel nodo n è dato dalla somma dei prodotti di ciascun rendimento nel periodo che termina in n per il denaro investito nei diversi asset nel nodo genitore $a(n)$. Il vincolo 2.5c stabilisce che la quantità totale di denaro che è possibile investire in un certo nodo è data dal valore della propria posizione a cui va sottratta la passività da pagare in quel nodo. L'equazione 2.5d definisce invece il concetto di surplus e shortfall: in ogni nodo chiaramente al più uno dei due sarà non nullo. Infatti se la differenza tra il valore del portafoglio e la passività è positiva allora si ha un surplus positivo e lo shortfall risulta nullo, viceversa se tale differenza è negativa è lo shortfall a risultare positivo, mentre il surplus rimane nullo. I vincoli successivi sono vincoli sullo stato iniziale del sistema. L'ultimo vincolo indica che non è possibile investire una quantità negativa negli asset, cioè che non è permessa la vendita allo scoperto di questi ultimi.
In questa sezione viene implementato il modello di Russell-Yasuda-Kasai soltanto considerando scenari binari in ogni nodo, in modo da poterlo confrontare con i modelli costruiti nelle sezioni 2.2 e 2.3.

2.4.2 Implementazione del modello in MATLAB

Per implementare il modello appena costruito nel caso di un albero binario è stata creata una funzione chiamata *binary_yasuda* la quale richiedi in input i seguenti oggetti:

- l'orizzonte temporale (o il numero di periodi) T del problema;
- il numero di asset a disposizione in cui investire;
- la ricchezza iniziale a disposizione W_0;
- il coefficiente r di penalizzazione dello shortfall;
- i vettori contenenti gli scenari positivo e negativo di ciascun asset;
- la passività da pagare al termine di ciasun periodo.

In output è restituita la strategia di investimento che suggerisce cosa fare a secondo dello scenario che si realizza. In particolare per ogni nodo dell'albero, oltre alla quantità di denaro da investire in ciascun asset, si ottiene il valore della posizione, lo shortfall e il surplus.

```
function solALM = binary_yasuda (numPeriods,numAssets,...
    wealth0,shortfallCoeff,high,low,liability)
    num_scenarios_per_period = zeros(1, numPeriods+1);
    for i = 1:numPeriods+1
        num_scenarios_per_period(i) = 2.^(i-1);
    end
    numNodes = sum(num_scenarios_per_period);
    rootNode = 1;
    tradingNodes = 2:2^(numPeriods)-1;
    leafNodes = 2^(numPeriods):2^(numPeriods+1)-1;
    numScenarios = num_scenarios_per_period(end);
    numTradingNodes = length(tradingNodes);
    antecedentNode = zeros(1,numNodes);
    antecedentNode(1) = NaN;
    for i = 2:numNodes
        antecedentNode(i) = floor(i/2);
```

```
    end
    probs = zeros(1,numNodes);
    for i=1:numNodes
        probs(i) = 1./num_scenarios_per_period(floor(log2(i))+1);
    end
    liabilities = zeros(numNodes,1);
    for i = 2:numNodes
        liabilities (i) = liability;
    end
    returns = zeros(numNodes, numAssets);
    returns(1,:) = ones(1,numAssets);
    for i = 2:numNodes
        if mod(i,2)==0
            returns(i,:) = high;
        else
            returns(i,:) = low;
        end
    end
    returns = transpose (returns);
    probALM = optimproblem('ObjectiveSense','max');
    value = optimvar('value', 1, numNodes);
    asset_value = optimvar('asset_value', numAssets,...
        numNodes-numScenarios, 'LowerBound', 0);
    shortfall = optimvar('shortfall', 1, numNodes,...
        'LowerBound', 0);
    surplus = optimvar('surplus', 1, numNodes,  'LowerBound', 0);

    probALM.Objective = dot(probs(2^numPeriods:end),...
        value(2^numPeriods:end))-dot(probs,...
        shortfallCoeff*shortfall);

    initials = optimconstr(3,1);
    initials (1) = value(1) == wealth0;
    initials (2) = shortfall (1) == 0;
    initials (3) = surplus(1) == wealth0;
    probALM.Constraints.initials = initials;

    V = optimconstr(1, numNodes-1);
    for k = 1:numNodes-1
        V(k) = value(k+1) == dot(asset_value(:,...
            antecedentNode(k+1)),returns(:,k+1));
    end
    probALM.Constraints.V = V;

    budget = optimconstr(1, numNodes-numScenarios);
    for k = 1:numNodes-numScenarios
        budget(k) = sum(asset_value(:,k)) == value(k) -...
            liabilities(k);
    end
```

```
66      probALM.Constraints.budget = budget;
67
68      balance = optimconstr(1, numNodes-1);
69      for k=1:numNodes-1
70          balance(k) = value(k+1) - liabilities(k+1) ==...
71              surplus(k+1) - shortfall (k+1);
72      end
73      probALM.Constraints.balance = balance;
74      showproblem(probALM)
75      solALM = solve(probALM);
76  end
```

Con lo scopo di confrontare tale modello con i risultati dei due modelli studiati in precedenza (in particolar modo con il primo), la funzione *binary_yasuda* è stata implementata con i seguenti dati:

- $numPeriods = 3$;
- $numAssets = 2$;
- $wealth0 = 55000$;
- $shortfallCoeff = 4$;
- $high = [1.25, 1.14]$;
- $low = [1.06, 1.12]$;
- $liability = 27000$.

La scelta di una passività costante pari a \$27000 è stata dettata dal fatto che nel primo modello si aveva una passività di \$80000 al termine dei 3 periodi e si è usata l'approssimazione $80000/3 \approx 27000$.

```
1  binary_yasuda(3,2,55000, 4, [1.25 1.14], [1.06 1.12], 27000)
```

La soluzione ottenuta con tali dati ha una struttura data dalla figura 2.3

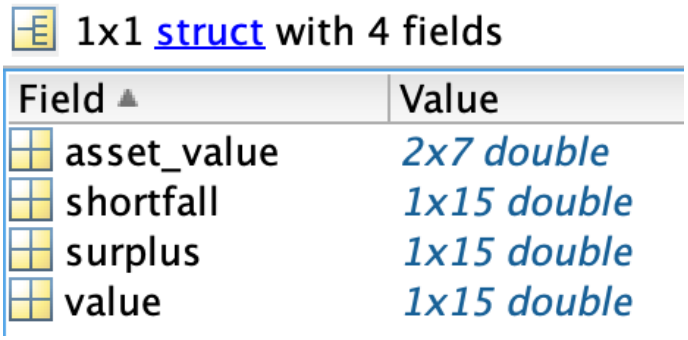


Figura 2.3. Struttura della soluzione ottenuta per il problema di Russell-Yasuda-Kasai

La variabile che risulta di maggior interesse è chiaramente *asset_value* poiché è quella che definisce la politica di investimenti. Essa assume i seguenti valori, che vengono riportati in forma tabulare come per gli altri modelli nella tabella 2.3:

Istante di tempo	Nodo	$ in asset 1	$ in asset 2
$t = 0$	n_1	55000	0
$t = 1$	n_2	31929	9821
	n_3	31300	0
$t = 2$	n_4	0	24107
	n_5	17844	0
	n_6	12125	0
	n_7	6178	0

Tabella 2.3. Politica ottimale di investimento che risolve il modello di Russell-Yasuda-Kasai.

Chiaramente la politica è molto diversa rispetto a quello ottenuta con il primo modello dal momento che in questo modello si ha un vettore di passività. Risulta interessante andare a confrontare il surplus e lo shortfall dei due modelli negli scenari terminali. Per il modello 1 lo shortfall si aveva soltanto nell'ultimo nodo dell'albero, cioè nello scenario terminale meno favorevole. In questo caso, avendo passività distribuite nel tempo si potrebbe risultare insolventi anche prima della maturità. Lo shortfall che si ottiene con la politica ottimale è dato da

```
ans.shortfall' =

             0
             0
             0
             0
             0
             0
             0
             0
             0
          4694.64
          8085.06
         11843.75
         14147.50
         19277.50
         20451.32
```

Ciò significa che non si risulta mai insolventi nei nodi intermedi, tuttavia negli scenari terminali si realizza quasi sempre una situazione di insolvenza, a meno di trovarsi nei due casi più favorevoli. Ovviamente più sfavorevole risulta lo scenario terminale, più grande sarà il corrispondente shortfall. Questo risultato dimostra come riuscire a pagare un'unica passività alla fine dell'orizzonte temporale sia chiaramente più semplice rispetto a dover pagare periodicamente un certo ammontare, anche se la somma totale delle passività è all'incirca analoga nei due casi.
Chiaramente se risolviamo il problema con una passività periodica leggermente inferiore anche con questo modello si riesce ad essere sempre solventi come dimostra il caso sottostante.

```
1 binary_yasuda(3,2,55000, 4, [1.25 1.14], [1.06 1.12], 22000)
```

```
ans.shortfall' =
```

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

Parte II

Seconda Parte

Capitolo 3

Problema di ALM con generazione di scenari tramite il modello CIR

Per tutti i modelli analizzati nel capitolo precedente sono stati implementati dei codici i cui dati erano fittizi e avevano come unico scopo quello di mostrare che i problemi descritti fossero risolvibili numericamente. In particolare erano irreali i dati relativi ai rendimenti dei diversi asset presi in considerazione e, di conseguenza, gli alberi di scenari da essi ricavati. In questo capitolo viene implementato il modello Russell-Yasuda-Kasai su un albero generato con dati non fittizi, ma basati sul modello CIR implementato con parametri stimati da dati reali. In aggiunta ai modelli precedenti sarà anche possibile decidere la struttura dell'albero, ovvero il numero di scenari possibili al termine di ogni periodo. Gli strumenti in cui è possibile investire non saranno più dei generici asset con un certo rendimento moltiplicativo ma saranno dei bond, cioè delle obbligazioni.

3.1 Il modello di Cox-Ingersoll-Ross

Nel modello che verrà costruito in questo capitolo si considereranno come possibili investimenti nel problema di ALM dei bond. Il prezzo di un bond dipende dall'evoluzione nel tempo del tasso di interesse a cui tale bond viene scontato: in questa tesi si è scelto di utilizzare come modello per l'evoluzione dei tassi di interesse a breve termine quello elaborato nel 1985 da Cox, Ingersoll e Ross, anche chiamato modello CIR. Uno dei motivi per cui è stata fatta tale scelta sta nel fatto che il CIR è un modello a un fattore, cioè esso descrive l'evoluzione nel tempo dei tassi di interesse come se essi dipendessero da un'unica fonte di rischio, che può essere considerata come un rischio di mercato.
Il modello CIR è una estensione del modello di Vasicek e si basa su un processo di diffusione secondo cui il tasso di interesse istantaneo $r(t)$ segue la seguente equazione differenziale stocastica (detta processo CIR):

$$dr(t) = \gamma[\bar{r} - r(t)]dt + \sigma\sqrt{r(t)}dW(t). \tag{3.1}$$

In tale modello:

- $W(t)$ è un processo di Wiener che di fatto modella il fattore di rischio di mercato da cui dipende il tasso di interesse;
- $\sigma > 0$ rappresenta la volatilità istantanea del tasso di interesse;
- $\bar{r} > 0$ è la media a lungo temine del processo;
- $\gamma > 0$ è la velocità con cui il tasso si avvicina alla media (*speed of reversion*);
- $\gamma[\bar{r} - r(t)]$ è il fattore di *drift*, il quale garantisce la proprietà di *mean-reversion*, cioè di convergenza nel lungo termine del tasso di interesse alla sua media $\bar{r}$: tutte le traiettorie future del tasso di interesse evolveranno nel lungo periodo intorno alla media;
- il fattore dato dalla deviazione standard $\sigma\sqrt{r(t)}$ evita la possibilità che ci siano tassi di interesse negativi.

Si enuncia di seguito, senza dimostrarlo, un risultato sul modello CIR che risulta estremamente utile dal punto di vista dell'implementazione in MATLAB del codice che simula il modello di evoluzione dei tassi.

Teorema 1. *La legge di transizione da r_0 a r_t per il modello CIR della forma (3.1) può essere espressa da:*

$$r_t = \frac{\alpha(1 - e^{-\gamma t})}{4\gamma}\chi'^2_\nu(\lambda), \tag{3.2}$$

dove $\alpha = \sigma^2$ e $\chi'^2_\nu(\lambda)$ è una variabile aleatoria chi-quadrato non centrale con $\nu = \frac{4\bar{r}\gamma}{\alpha}$ gradi di libertà e parametro di non centralità dato da $\lambda = \frac{4\gamma e^{-\gamma t}}{\alpha(1-e^{-\gamma t})} r_0$.

In MATLAB è stata creata la seguente funzione *Sample_Rate_CIR* che ha lo scopo di campionare un dato numero di osservazioni dall'evoluzione del modello CIR su un periodo di ampiezza temporale data. In particolare tale funzione prende in input i parametri $r_0, \bar{r}, \gamma, \alpha$ del modello, l'ampiezza dell'intervallo temporale dt e il numero di osservazioni che si vogliono generare *numRepl* e restituisce le osservazioni desiderate.

```
function chi = Sample_Rate_CIR(r0,rbar,gamma,alpha,dt,numRepl)
expg = alpha*(1-exp(-gamma*dt))/(4*gamma);
nu = 4*rbar*gamma/alpha;
lambdaC = exp(-gamma*dt)/expg;
chi = zeros(numRepl,1);

for i = 1:numRepl
    chi(i) = expg*ncx2rnd(nu,lambdaC*r0);
end
end
```

Un altro dei motivi che avvalorano la scelta del modello CIR come processo evolutivo dei tassi di interesse è la semplicità con cui può essere prezzato un bond una volta che è noto il tasso ottenuto da tale modello. Infatti, assumendo l'ipotesi di non arbitraggio, il prezzo di un bond può essere calcolato tramite il processo CIR secondo quanto evidenziato dal risultato della seguente proposizione, che non viene dimostrata.

Proposizione 1. *Sia r_t il tasso di interesse ricavato dal modello CIR con parametri $\gamma, \bar{r}$ e σ. Sia $\alpha = \sigma^2$ e $h = \sqrt{\gamma^2 + 2\alpha}$. Allora il prezzo al tempo t di un bond a cedola zero (zero-coupon bond) con maturità T e valore facciale pari a 1 è*

$$P_{t,T}(r_t) = A_{t,T} \exp(-B_{t,T} r_t), \tag{3.3}$$

$$A_{t,T} = \Big(\frac{2h\exp\left((\gamma+h)(T-t)/2\right)}{2h+(\gamma+h)(\exp\left((T-t)h\right)-1))}\Big)^{\frac{2\gamma\bar{r}}{\alpha}}, \tag{3.4}$$

$$B_{t,T} = \frac{2(\exp\left((T-t)h\right)-1}{2h+(\gamma+h)(\exp\left((T-t)h\right)-1))}. \tag{3.5}$$

Pertanto il prezzo del bond a cedola zero è esponenziale affine nel tasso di interesse.
Di seguito la funzione *zero_CIR_price* che è stata creata in MATLAB e che prende in input un vettore di *zero-coupon bond* (in particolare il loro valore facciale e il loro *time-to-maturity*), i parametri del CIR e il tasso di interesse e restituisce in output il prezzo degli zeri passati in input.

```
function prices = zero_CIR_price(face,time_to_mat,gamma,rbar,...
    alpha,r_t)
prices = zeros(length(time_to_mat),1);
h = sqrt(gamma^2+2*alpha);
for i=1:length(prices)
    A = ((2*h*exp((gamma+h)*(time_to_mat(i))/2))/(2*h+...
        (gamma+h)*(exp((time_to_mat(i))*h)-1)))^...
        (2*gamma*rbar/alpha);
    B = (2*(exp((time_to_mat(i))*h)-1))/(2*h+(gamma+h)*...
        (exp((time_to_mat(i))*h)-1));
    prices(i) = A*exp(-B*r_t);
end
prices = prices.*face;
end
```

3.2 Bond a cedola zero e bond con cedola

Nella sezione precedente è stata mostrata una formula esplicita che permette, noto il tasso di interesse ottenuto tramite il CIR, di calcolare il prezzo di un bond a cedola zero. Nel problema di asset-liability management che ci si propone di risolvere, i bond a cedola zero costituirebbero la categoria di asset ideali per coprirsi dal rischio di tasso di interesse. Tali bond possono, però, presentare alcuni problemi:

- non sempre gli zeri sono disponibili sul mercato;
- quando sono disponibili generalmente hanno tempi di maturità molto lunghi (10,15,30 anni), pertanto ciò può risultare non conveniente per pagare delle passività nel breve termine;
- spesso sono molto cari.

Queste motivazioni inducono a considerare, al posto di bond a cedola zero, i cosiddetti *coupon-bearing bond* ovvero quei bond che periodicamente (generalmente ogni 6 mesi) staccano una cedola data da una percentuale del valore facciale, la quale viene incassata dall'investitore.
Dunque, nel modello che verrà implementato, saranno usati come possibili asset dei coupon con cedola. Pertanto è necessario capire in che modo sia possibile calcolare il prezzo di un tale tipo di bond. Si consideri ad esempio un bond con cedola con le seguenti caratteristiche:

- valore facciale F;

- maturità di T anni;
- cedola a tasso costante c pagata semestralmente.

In base a questi dati, è possibile rappresentare il bond attraverso il vettore dei flussi di cassa che esso genera: ogni 6 mesi verrà staccata una cedola di valore pari a $F\frac{c}{2}$ e alla maturità, insieme all'ultima cedola, verrà restituito anche il valore facciale F. Dunque il vettore dei flussi di cassa sarà:

$$\left[\left(F\frac{c}{2}, T_1\right), \ldots, \left(F\frac{c}{2}, T_{m-1}\right), \left(F(1+\frac{c}{2}), T_m\right)\right], \tag{3.6}$$

dove sono stati indicati con $T_1, \ldots, T_m$ gli istanti di tempo in cui vengono staccate le cedole.
Avendo osservato ciò, è possibile ora pensare il bond con cedola come un vettore di bond a cedola zero rappresentati in (3.6). Pertanto il suo prezzo ad un determinato tempo t sarà dato dalla somma dei prezzi al tempo t degli zeri che lo compongono. Indicando con $P_c(t,T)$ il prezzo al tempo t del bond con tasso di cedola c, valore facciale F e maturità T e con $P_z(t,T)$ il prezzo al tempo t del bond a cedola zero con valore facciale F e maturità T, si avrà

$$P_c(t,T) = \frac{c}{2}\sum_{i=1}^{m} P_z(t,T_i) + P_z(t,T_m), \tag{3.7}$$

dove

$$P_z(t,T_i) = FZ(t,T_i), \quad \forall i = 1, \ldots, m$$

e $Z(t,T)$ indica il tasso di sconto al tempo t di uno *zero-coupon bond* con maturità T (ad esempio, se si considera la capitalizzazione nel continuo è pari a $exp(-r(t,T)\cdot(T-t))$.
Pertanto è sufficiente saper prezzare un bond a cedola zero per riuscire a prezzare un qualsiasi bond che paga delle cedole a tasso costante. In MATLAB è stata costruita una funzione chiamata *bon_CIR_price* che permette di calcolare il prezzo di un qualsiasi bond con cedola, dato il tasso di interesse ottenuto dal modello CIR.

```
function prices = bond_CIR_price(face,time_to_mat,coupon,...
    period,gamma,rbar,alpha,r_t)
prices = zeros(length(time_to_mat),1);
m = floor(time_to_mat.*period); %num_coupons
for k = 1:length(prices)
    for i = 1:m(k)
        prices (k) = prices (k) + zero_CIR_price(face(k)*...
            coupon(k)/2,i/period, gamma,rbar,alpha,r_t);
    end
    prices(k) = prices(k) + zero_CIR_price(face(k),...
        time_to_mat(k), gamma,rbar,alpha,r_t);
end
end
```

Tale funzione, seguendo quanto espresso dalla formula (3.7), al suo interno chiama la funzione *zero_CIR_price* introdotta nella sezione precedente, la quale calcola il prezzo di un bond a cedola zero. Pertanto ora si è in grado di calcolare il prezzo di un qualsiasi bond in qualsiasi istante, se è noto il tasso a cui scontare i flussi.

3.3 Costruzione dell'albero di scenari tramite *moment matching*

Il passo successivo per riuscire ad implementare il modello nella sua forma più generale è quello di riuscire a creare un albero di scenari. Per costruire il modello di Russell-Yasuda-Kasai (2.5) con una struttura ad albero arbitraria, cioè con ramificazioni qualsiasi e non unicamente binarie, sono stati definiti in MATLAB i seguenti oggetti:

- una classe *ScenarioTreeClass* che caratterizza la struttura di un albero;
- una classe *NodeClass* che caratterizza i nodi all'interno di un albero;
- una classe *PathGeneratorClass* che definisce un generatore qualsiasi di scenari;
- una sottoclasse *CIRPathGeneratorClass* di *PathGeneratorClass* che caratterizza un generatore di scenari ottenuti a partire dal modello evolutivo di Cox, Ingersoll e Ross.

Diamo una breve descrizione di come sono caratterizzati gli oggetti delle classi *NodeClass* e *ScenarioTreeClass*.
Un oggetto *NodeClass* è individuato da:

- un predecessore, anche detto antecedente o nodo genitore;
- un vettore di successori o nodi figli;
- una probabilità;
- una probabilità condizionata al nodo genitore;
- un indice temporale che ne indica il livello;
- uno stato (non necessariamente osservabile);
- un insieme di valori osservabili.

Un esempio di struttura di un oggetto di classe *NodeClass* è dato dalla figura 3.1.

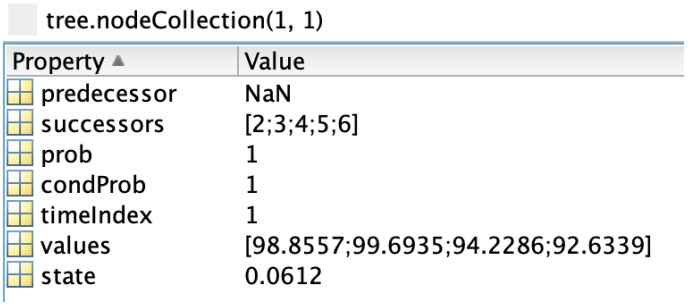


Figura 3.1. Esempio di struttura di un nodo.

Un oggetto che appartiene alla classe *ScenarioTreeClass* è, invece, identificato da:

- la struttura della ramificazione, ovvero il numero di scenari per ogni livello, specificata tramite un vettore di interi;
- l'orizzonte temporale;
- i parametri del modello CIR che lo genera;

- una collezione di nodi;
- un vettore che indica il numero di nodi ad ogni livello;
- l'indice del nodo iniziale di ciascun livello;
- il numero di nodi totali nell'albero;
- il numero di stati per ogni nodo.

Un esempio di struttura di un oggetto di classe *ScenatrioTreeClass* è dato dalla figura 3.2.

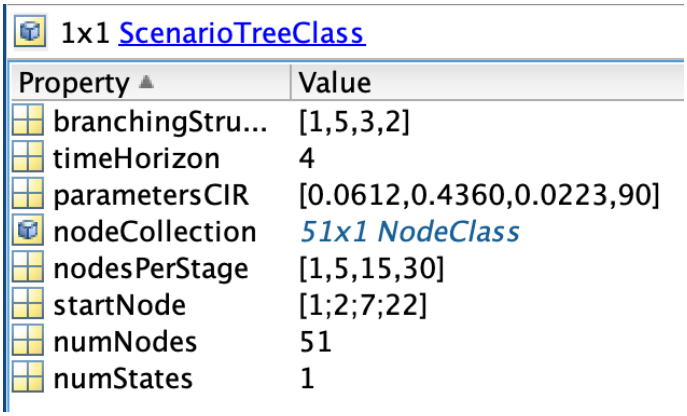


Figura 3.2. Esempio di struttura di un albero.

Per questioni di spazio e leggibilità i codici che definiscono queste quattro classi definite per implementare il modello e i relativi metodi si trovano in appendice A.

Una volta definiti questi oggetti rimane da capire in che modo sia possibile generare un albero di scenari per il tasso di interesse a cui scontare i bond, partendo soltanto dal modello CIR. In particolare la difficoltà sta nel fatto che, fissato un certo $t > 0$, la variabile aleatoria $r(t)$ è continua, mentre per costruire l'albero è necessario generare un numero finito di scenari. Per capire in che modo generare gli scenari e quale sia la probabilità di ogni scenario si è utilizzato il metodo del *moment matching* di cui diamo una breve spiegazione, ma che è descritto dettagliatamente nel paper [5].
L'idea alla base del metodo del *moment matching* è di creare un insieme di punti tale che i momenti della distribuzione discreta si discostino il meno possibile dai momenti della distribuzione originale che si vuole discretizzare. In particolare si tengono in considerazione i primi quattro momenti centrali della distribuzione: il valore atteso, la deviazione standard, la skewness (asimmetria) e la curtosi; per una variabile aleatoria multivariata si tiene conto anche della matrice di covarianza. Per completezza ricordiamo che la skewness e la curtosi di una generica distribuzione $\mathbf{X}$ sono rispettivamente date da:

$$s = \frac{\mathbb{E}[(\mathbf{X} - \mu)^3])}{\sigma^3}, \tag{3.8}$$

$$k = \frac{\mathbb{E}[(\mathbf{X} - \mu)^4])}{\sigma^4}, \tag{3.9}$$

dove μ indica la media della distribuzione e σ la sua deviazione standard. Chiaramente risulta pressoché impossibile creare un insieme di punti tali da ottenere gli stessi valori per tutti e

quattro i momenti e per la covarianza soprattutto a causa del fatto che il numero di punti in cui discretizzare la distribuzione è arbitrario. Pertanto, per ovviare a questo problema, si costruisce un modello di ottimizzazione quadratica che minimizza la distanza dei momenti della distribuzione discreta da quelli noti della distribuzione che si intende approssimare. Nel caso che interessa questa tesi, si vuole approssimare una variabile aleatoria univariata, ovvero il tasso di interesse ottenuto dal modello CIR. Pertanto il modello di ottimizzazione descritto nel paragrafo 3.1 del paper [5] può essere semplificato come segue.
Sia data una variabile aleatoria continua e siano M_k, con $k = 1,2,3,4$ i primi quattro momenti della sua distribuzione continua. Sia N il numero di scenari con cui si vuole approssimare la distribuzione in questione. Indichiamo con x_i, con $i = 1,\dots,N$ gli scenari della distribuzione discreta approssimante, con p_i, $i = 1,\dots,N$ le relative probabilità e con m_k, $k = 1,2,3,4$ i momenti della distribuzione approssimante. Detto w_k il peso da dare ad ogni momento nella funzione obiettivo, il modello di ottimizzazione che permette di discretizzare la distribuzione in N punti è il seguente:

$$\min_{\mathbf{x},\mathbf{p}} \quad \sum_{k=1}^{4} w_k (m_k - M_k)^2 \tag{3.10a}$$

$$\text{tale che} \quad m_1 = \sum_{i=1}^{N} x_i p_i, \tag{3.10b}$$

$$m_2^2 = \sum_{i=1}^{N} (x_i - m_1)^2 p_i, \tag{3.10c}$$

$$m_k^k m_2^k = \sum_{i=1}^{N} (x_i - m_1)^k p_i, \quad k = 3,4, \tag{3.10d}$$

$$\sum_{i=1}^{N} p_i = 1, \tag{3.10e}$$

$$p_i \geq 0 \quad \forall i = 1,\dots N. \tag{3.10f}$$

Il vincolo 3.10b definisce la media m_1 degli scenari, il vincolo 3.10c la loro deviazione standard m_2 e il vincolo 3.10d definisce la skewness m_3 e la curtosi m_4.
Nell'implementazione MATLAB è stato anche aggiunto il vincolo che i punti della discretizzazione debbano essere distinti l'uno dall'altro. Di seguito il codice che risolve questo problema prendendo in input il numero di scenari da generare e il vettore dei primi q momenti della distribuzione da approssimare (con eventuali *bound* che la distribuzione in questione può avere) e restituendo in output gli N scenari, le relative probabilità e i primi q momenti della distribuzione discreta ottenuta. Il problema è stato implementato dando peso decrescente rispetto all'ordine dei momenti: cioè il vettore dei pesi usato per i momenti è [4,3,2,1].

```
1 function [scenarios,probabilities,approx_moments] =...
2     Scenarios_generator(numScenarios,moments,bounds)
3 %   This function takes in input an integer number of scenarios
4 %   and a vector of moments of a distribution and gives in output
5 %   the support points and their probabilities (the discretized
6 %   distribution)
7
8 weights = length(moments):-1:1;
9
```

```
prob = optimproblem('ObjectiveSense','min');
scenarios = optimvar('scenarios', numScenarios, 1,...
    'LowerBound', bounds(1), 'Upperbound', bounds(2));
probabilities = optimvar('probabilities', numScenarios, 1,...
    'LowerBound', 0, 'UpperBound', 1);
approx_moments = optimvar('approx_moments', length(moments), 1);

% objective is weighted difference of moments
prob.Objective = dot(weights, (approx_moments-moments).^2);

% probability sum = 1
sum_probs = optimconstr(1,1);
sum_probs = (sum(probabilities)==1);
prob.Constraints.sum_probs = sum_probs;

% scenarios must be different points
different_scenarios = optimconstr (numScenarios-1,1);
for k = 1:numScenarios-1
    different_scenarios(k) = (scenarios(k)<=scenarios(k+1));
end
prob.Constraints.different_scenarios = different_scenarios;

% matching mean
mean = optimconstr(1,1);
mean = (approx_moments(1) == dot(scenarios, probabilities));
prob.Constraints.mean = mean;

% matching stdev
std = optimconstr(1,1);
std = (approx_moments(2)^2 ==...
    dot((scenarios-approx_moments(1)).^2, probabilities));
prob.Constraints.std = std;

% matching higher order moments
matching = optimconstr(length(moments)-2, 1);
for k = 1:length(moments)-2
    matching(k) = (approx_moments(k+2)*approx_moments(2)^(k+2)...
        == dot(probabilities, (scenarios-approx_moments(1)).^...
        (k+2)));
end
prob.Constraints.matching = matching;


if (bounds(1)>-inf && bounds(2)<inf)
    x0.scenarios = linspace(bounds(1), bounds(2), numScenarios);
else
    x0.scenarios = zeros(numScenarios, 1);
end

```

```
x0.probabilities = repmat(1/numScenarios, numScenarios, 1);
x0.approx_moments = moments;

sol = solve(prob, x0);
scenarios = sol.scenarios;
probabilities = sol.probabilities;
approx_moments = sol.approx_moments;
end
```

A questo punto si ha un modo per discretizzare una variabile aleatoria di cui si conoscono i primi quattro momenti.
Dal modello CIR, tuttavia, non si è in grado di ricavare una forma esplicita per i primi quattro momenti del tasso di interesse $r(t)$ nell'intervallo temporale $[0, t]$. Infatti tale tasso risulta essere una variabile aleatoria chi-quadrato moltiplicata per una costante, per cui non si ha una forma esplicita per i suoi momenti. Dunque, per ovviare a questo ulteriore problema la strategia adottata è stata quella di simulare il processo CIR un numero molto grande di volte e considerare i momenti che si ottengono campionando al posto dei momenti della distribuzione continua "vera". Nell'implementazione del modello finale si sono usate 100000 replicazioni per stimare i momenti della distribuzione in ogni nodo dell'albero. Di seguito è riportato il codice della funzione *CIRmoments* che prende in input i parametri del modello CIR e l'ampiezza *dt* dell'intervallo di tempo su cui si vuole simulare e restituisce in output il vettore contenente i primi quattro momenti del campione generato e anche i valori massimo e minimo di tale campione.

```
function [moments,low,up] = CIRmoments(r0,rbar,gamma,alpha,dt)
% This functions gets in input the parameters of a CIR model
% and the time interval dt; it samples 100000 realizations of the
% CIR model and it computes the first 4 moments of the observed
% data
NumObs = 100000;
rng('default');
Sample = Sample_Rate_CIR (r0,rbar,gamma,alpha,dt,NumObs);

low = min(Sample);
up = max(Sample);
edges = low:0.0005:up;
[N,edges] = histcounts(Sample, edges);
rates = zeros(size(N));
for i=1:length(rates)
    rates(i)=(edges(i)+edges(i+1))/2;
end
rate_probs = N./NumObs;

moments = zeros(4,1);
moments(1) = mean(Sample);
moments(2) = std(Sample);
moments(3) = skewness(Sample, 0);
moments(4) = kurtosis(Sample, 0);
end
```

A questo punto, grazie all'introduzione delle classi e delle funzioni descritte in questa sezione, si è in grado di generare completamente un albero di scenari per i tassi di interesse, supponendo

che questi ultimi abbiano un'evoluzione che segue il modello di Cox-Ingersoll-Ross. La funzione che adempie a questo compito richiede in input soltanto i parametri del modello CIR e la struttura che si vuole dare all'albero, cioè il numero di ramificazioni che partono da ciascun nodo dell'albero a seconda del livello in cui esso si trova. Per questioni implementative si ha sempre un 1 nella prima entrata del vettore *branches*, il quale rappresenta la radice dell'albero stesso. La funzione, chiamata *Scenarios_Tree_generator*, è la seguente:

```
function tree = Scenarios_Tree_generator(branches,r0,rbar,...
    gamma,alpha,dt)
parameters = [rbar,gamma,alpha,dt];
state = r0;
Path = CIRPathGeneratorClass;
branchingStructure = branches;
tree = ScenarioTreeClass;
tree.FillTree(branchingStructure, state, parameters, Path);
end
```

Adesso la costruzione dell'albero di tassi di interesse è ultimata. Ciò che in realtà risulta interessante dal punto di vista del problema di gestione di un portafoglio con asset e passività non è tanto il tasso di interesse quanto il prezzo di ciascun asset in cui è possibile investire. Pertanto, dato un certo albero di tassi di interesse e un certo insieme di possibili bond in cui investire, è necessario riuscire a calcolare il prezzo di ciascuno dei bond in ogni nodo dell'albero stesso. A questo scopo è stata creata un'altra funzione, chiamata *compute_prices* la quale prende in input un albero in cui i tassi di interesse costituiscono gli stati dei nodi, i dati sui bond che si vogliono prezzare (in particolare valore facciale, data in cui si vuole prezzare, tasso di cedola e maturità) e i parametri relativi al modello CIR e restituisce in output, utilizzando la funzione *bond_CIR_price* introdotta in precedenza, l'albero aggiornato in cui i valori di ogni nodo sono costituiti dal vettore dei prezzi dei *coupon-bearing bond* nello scenario rappresentato dal nodo.

```
function tree = compute_prices(tree, Face, SettleNum,...
    MaturityNum, Coupon, period, gamma, alpha, dt)
for i = 1:tree.numNodes
    for j = 1:tree.timeHorizon
        if j < tree.timeHorizon
            if (i >= tree.startNode(j) && i <tree.startNode(j+1))
                periods = j;
                break
            end
        else
            periods = j;
            break
        end
    end
    time_to_m = yearfrac(SettleNum+dt*(j-1), MaturityNum);
    tree.nodeCollection(i).values = bond_CIR_price(Face,...
        time_to_m,Coupon,period,gamma,...
        tree.nodeCollection(i).state,alpha,...
        tree.nodeCollection(i).state);
end
end
```

3.4 Costruzione del modello di programmazione stocastica

Come preannunciato all'inizio del capitolo, il modello di programmazione stocastica che si vuole implementare sull'albero generato è quello di Russell-Yasuda-Kasai 2.5 introdotto nella sezione 2.4. Per semplicità lo riportiamo, senza specificare di nuovo la notazione.

$$\max_X \quad \sum_{s\in\mathcal{S}} p^s V^s - r\sum_{n\in\mathcal{N}} p^n \omega_-^n \tag{3.11a}$$

$$\text{tale che} \quad V^n = \sum_{i\in\mathcal{A}} R_{i,n} X_{i,a(n)}, \quad \forall n = 2,\dots,2^{T+1}-1 \tag{3.11b}$$

$$\sum_{i\in\mathcal{A}} X_{i,n} = V^n - L^n, \quad \forall n \in \mathcal{N}\setminus\mathcal{S} \tag{3.11c}$$

$$V^n - L^n = \omega_+^n - \omega_-^n \quad \forall n = 2,\dots,2^{T+1}-1 \tag{3.11d}$$

$$V_1 = \omega_+^1 = W_0, \tag{3.11e}$$

$$\omega_-^1 = 0, \tag{3.11f}$$

$$\omega_-^n, \omega_+^n \geq 0, \quad \forall n \in \mathcal{N} \tag{3.11g}$$

$$X_{i,n} \geq 0, \quad \forall i \in \mathcal{A}, \forall n \in \mathcal{N}\setminus\mathcal{S} \tag{3.11h}$$

Anche in questo caso si suppone che la passività sia deterministica e che sia uguale per ogni periodo, supponendo che sia una sorta di pensione integrativa che si vuole ottenere o che si abbia un'uscita fissa al termine di ciascun intervallo temporale.
A differenza di quanto fatto nella sezione 2.4, qui non è più sufficiente costruire una funzione che risolva il problema nel caso di un albero binario, ma è necessario costruirne una che lo risolva su un albero qualsiasi. Pertanto la funzione *yasuda* che segue non fa altro che risolvere il problema su un albero generato con modello CIR tramite le funzioni precedenti e che contiene nei valori di ogni nodo i prezzi dei bond a disposizione come asset. In input sono pertanto richiesti:

- l'albero di scenari contenente i prezzi;
- la ricchezza iniziale W_0;
- il coefficiente r di penalizzazione dello shortfall;
- la passività da pagare.

L'output è ancora una volta la soluzione del modello che contiene per ogni nodo i seguenti quattro valori:

- l'ammontare da investire in ciascun asset;
- il valore del portafoglio;
- il surplus;
- lo shortfall.

```
1 function solALM = yasuda(tree,wealth0,shortfallCoeff,liability)
2     numPeriods = tree.timeHorizon-1;
3     numAssets = length(tree.nodeCollection(1).values);
4     num_scenarios_per_period = tree.nodesPerStage;
5     numNodes = tree.numNodes;
6     rootNode = 1;
```

```
    tradingNodes = 2:tree.startNode(end)-1;
    leafNodes = tree.startNode(end):numNodes;
    numScenarios = num_scenarios_per_period(end);
    numTradingNodes = length(tradingNodes);
    antecedentNode = zeros(1,numNodes);
    for i = 1:numNodes
        antecedentNode(i) = tree.nodeCollection(i).predecessor;
    end
    probs = zeros(1,numNodes);
    for i=1:numNodes
        probs(i) = tree.nodeCollection(i).prob;
    end
    liabilities = zeros(numNodes,1);
    for i = 2:numNodes
        liabilities (i) = liability;
    end
    returns = zeros(numNodes, numAssets);
    returns(1,:) = ones(1,numAssets);
    for i = 2:numNodes
        returns(i,:) = tree.nodeCollection(i).values./...
            tree.nodeCollection(antecedentNode(i)).values;
    end
    returns = transpose(returns);
    probALM = optimproblem('ObjectiveSense','max');
    value = optimvar('value', 1, numNodes);
    asset_value = optimvar('asset_value', numAssets, numNodes-...
        numScenarios, 'LowerBound', 0);
    shortfall = optimvar('shortfall', 1, numNodes,
        'LowerBound', 0);
    surplus = optimvar('surplus', 1, numNodes,  'LowerBound', 0);

    probALM.Objective = dot(probs(2^numPeriods:end),...
         value(2^numPeriods:end))-dot(probs,...
         shortfallCoeff*shortfall);

    initials = optimconstr(3,1);
    initials (1) = value(1) == wealth0;
    initials (2) = shortfall (1) == 0;
    initials (3) = surplus(1) == wealth0;
    probALM.Constraints.initials = initials;

    V = optimconstr(1, numNodes-1);
    for k = 1:numNodes-1
        V(k) = value(k+1) == dot(asset_value(:,...
            antecedentNode(k+1)),returns(:,k+1));
    end
    probALM.Constraints.V = V;

    budget = optimconstr(1, numNodes-numScenarios);
```

```
    for k = 1:numNodes-numScenarios
        budget(k) = sum(asset_value(:,k)) == value(k) -...
            liabilities(k);
    end
    probALM.Constraints.budget = budget;

    balance = optimconstr(1, numNodes-1);
    for k=1:numNodes-1
        balance(k) = value(k+1) - liabilities(k+1) ==...
            surplus (k+1) - shortfall (k+1);
    end
    probALM.Constraints.balance = balance;
    showproblem(probALM)
    solALM = solve(probALM);
end
```

Giunti a questo punto non rimane che costruire un'ultima funzione, detta *solver*, la quale compie nell'ordine i seguenti passi:

1. crea l'albero di scenari per i tassi di interesse partendo dal modello CIR;
2. aggiunge ad ogni nodo dell'albero i relativi prezzi dei bond;
3. calcola la soluzione del modello di Russell-Yasuda-Kasai.

Tale funzione richiede in input i parametri del modello CIR, la struttura che si vuole dare all'albero, i dati sui bond, la ricchezza iniziale e il coefficiente di penalizzazione dello shortfall e restituisce in output la soluzione del problema di asset-liability management.

```
function [tree,sol] = solver(branches,r0,rbar,gamma,alpha,dt,...
    wealth0,shortfallCoeff,liabilities,Settle,Maturity,Face,...
    Coupon)

SettleNum = datenum(Settle);
MaturityNum = datenum(Maturity);
time_to_m = yearfrac(SettleNum, MaturityNum);
tree = Scenarios_Tree_generator(branches, r0,rbar,gamma,...
    alpha,dt);
tree = compute_prices(tree, Face, SettleNum, MaturityNum,...
    gamma, alpha, dt);
sol = yasuda(tree,wealth0,shortfallCoeff,liabilities);
end
```

3.5 Scelta dei parametri

Per implementare il codice che risolve il problema presentato è necessario stabilire in che modo stimare i parametri del modello di Cox-Ingersoll-Ross da mettere come input della funzione *solver*. Poiché la stima dei parametri è aldilà dello scopo di questa tesi, che è incentrata sugli aspetti di programmazione stocastica, si è preso in considerazione il paper [6] per le stime dei suddetti parametri. In tale paper viene considerato un dataset composto da una serie temporale

di osservazioni giornaliere (prendendo soltanto i giorni lavorativi) del tasso PRIBOR 3M nell'intervallo temporale dal 3 gennaio 1994 al 4 ottobre 2007, il quale ha un andamento riportato in figura 3.3.

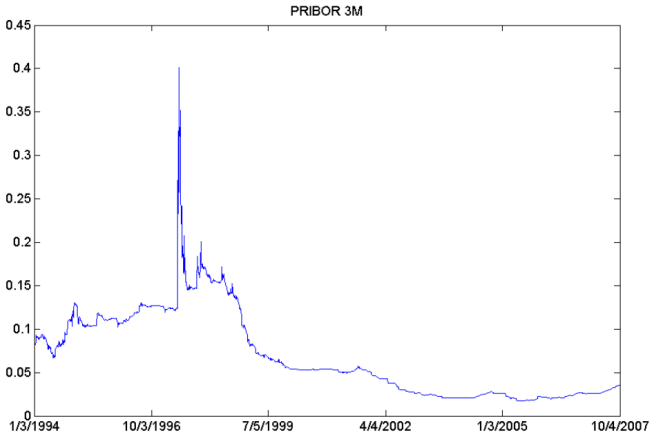


Figura 3.3. Serie temporale del tasso PRIBOR 3M dal 3 gennaio 1994 al 4 ottobre 2007. Il tasso è dato in forma percentuale.

Tale serie è stata sfrutta per stimare i parametri del modello CIR con il quale evolve tale tasso. In particolare per implementare il problema sono state scelte le stime ottenute attraverso il metodo di massima verosimiglianza con la funzione di MATLAB *ncx2pdf*. Tali stime sono riportate nella tabella 3.1

Stima di γ	Stima di $\bar{r}$	Stima di σ
0.4360	0.0612	0.1492

Tabella 3.1. Stime dei parametri del modello CIR.

3.6 Implementazione del modello

Riportiamo la funzione che calcola il prezzo di un bond a cedola zero dato il tasso di interesse modellato dal CIR:

$$P_{t,T}(r_t) = A_{t,T}\exp(-B_{t,T}r_t),$$

$$A_{t,T} = \left(\frac{2h\exp\left((\gamma+h)(T-t)/2\right)}{2h+(\gamma+h)(\exp\left((T-t)h\right)-1))}\right)^{\frac{2\gamma\bar{r}}{\alpha}},$$

$$B_{t,T} = \frac{2(\exp\left((T-t)h\right)-1}{2h+(\gamma+h)(\exp\left((T-t)h\right)-1))},$$

$$h = \sqrt{\gamma^2+2\alpha}.$$

Prima di implementare il modello con diversi insiemi di dati in input è importante sottolineare come con i dati in tabella 3.1 la funzione $P_{t,T}(r_t)$ risulta decrescente rispetto alla maturità T

in tutto l'intervallo $[0, +\infty)$. Pertanto, se venissero scelti dei bond a cedola zero come possibili strumenti per coprirsi rispetto al rischio di tasso di interesse, la soluzione direbbe in ogni scenario di acquistare il bond con scadenza più breve dal momento che esso risulterebbe quello con il valore maggiore. Anche per questo motivo, oltre a quelli elencati nelle precedenti sezioni, sono stati considerati bond con cedola come possibili asset del problema.
Dal momento che le stime dei parametri del modello CIR sono state ottenute osservando i valori del tasso PRIBOR trimestrale, si è scelto di usare un intervallo di tempo di 90 giorni in tutta l'analisi. Chiaramente tale modello non è confrontabile con i precedenti dal momento che si stanno considerando dei bond come strumenti di investimento mentre prima si avevano dei generici asset di cui era noto il rendimento moltiplicativo. Tuttavia come primo passo si considerano dei dati paragonabili a quelli degli altri modelli: in particolare si considera un intervallo temporale di 3 periodi di 90 giorni con ramificazione binaria in ogni periodo. Inoltre si sceglie un budget iniziale di \$55000 e una passività costante nel tempo di \$27000 al termine di ogni periodo. L'insieme degli asset è costituito da quattro bond con i seguenti dati:

- valore facciale di \$100 per ognuno;
- vettore delle maturità [1,2,3,4] misurate in anni;
- vettore dei tassi di cedola annuale $[0.05, 0.06, 0.04, 0.04]$ pagate semestralmente.

Il codice che implementa il modello con tali dati è il seguente:

```
close all
clear
branches = [1,2,2,2];
r0=0.0612;
rbar=0.0612;
gamma=0.436;
alpha=0.1492^2;
dt=90;
wealth0 = 55000;
shortfallCoeff = 4;
liabilities = 27000;
Settle = '31-Dec-2017';
Maturity = ['31-Dec-2018'; '31-Dec-2019'; '31-Dec-2020';
    '31-Dec-2021'];
Face = repmat(100, 4, 1);
Coupon = [0.05; 0.06; 0.04; 0.04];
Period = 2;
[tree,sol] = solver(branches,r0,rbar,gamma,alpha,dt,wealth0,...
    shortfallCoeff,liabilities,Settle,Maturity,Face,...
    Coupon, Period);
```

La soluzione che si ottiene risolvendo tale modello evidenzia subito una differenza rispetto ai modelli visti nel capitolo precedente: infatti, avendo introdotto come asset dei bond, il cui valore può aumentare ma anche diminuire a seconda dell'evoluzione dei tassi di interesse, anziché degli asset con rendimento positivo anche negli scenari meno favorevoli, non sempre si riesce a garantire la capacità di essere solventi. Lo shortfall che si ottiene mettendo il pratica la politica ottimale è il seguente:

```
sol.shortfall' =
```

```
            0
            0
            0
            0
            0
            0
            0
     25893.47
     25899.59
     25946.55
     25963.44
     25981.11
     25981.11
     26264.05
     26267.31
```

Ciò evidenzia come in realtà in tutti gli scenari terminale si risulti insolventi e questo è chiaramente dovuto ai dati inseriti nell'input del problema: avendo infatti un capitale iniziale di \$55000 non risulta plausibile riuscire a generare un flusso in entrata pari a \$27000 ogni 90 giorni utilizzando degli strumenti come i bond.
Si considera adesso un problema con dei dati meno stringenti e si analizza la soluzione al variare della struttura ad albero passata in input al risolutore. Siano:

- il numero di periodi pari a $T = 4$ (ricordiamo che ogni periodo coincide sempre con un intervallo di tempo di 90 giorni);
- la ricchezza iniziale pari ancora a \$55000;
- la passività pari a $L = \$15000$ per periodo.

Implementiamo tale problema prima con la struttura ad albero binaria $1 - 2 - 2 - 2 - 2$ e poi con una struttura più complessa che contenga un maggior numero di scenari, della forma $1 - 7 - 5 - 3 - 2$.

Implementando il modello con la struttura ad albero binario si ha la seguente soluzione (si riportano la politica di investimento e lo shortfall in ogni nodo):

```
sol.asset_value' =

            0            0     55000.00            0
     39685.04            0            0            0
     39543.76            0            0            0
            0            0            0     25294.81
            0            0            0     25213.03
     25242.57            0            0            0
            0            0     24819.11            0
     10205.96            0            0            0
     10066.70            0            0            0
     10214.91            0            0            0
      9810.71            0            0            0
      9957.90            0            0            0
      9957.85            0            0            0
      9832.44            0            0            0
```

```
       9722.35            0            0            0

sol.shortfall' =

            0
            0
            0
            0
            0
            0
            0
            0
            0
            0
            0
            0
            0
            0
            0
       4641.06
       4641.92
       4777.98
       4778.11
       4624.01
       4624.93
       5021.62
       5021.96
       4891.35
       4892.18
       4891.38
       4892.20
       4976.87
       4976.98
       5084.02
       5084.14
```

Pertanto si risulta sempre solventi nei primi 3 periodi, ma sempre insolventi nell'ultimo, indipendentemente dallo scenario che si realizza. Negli scenari meno favorevoli si ha chiaramente un'insolvenza più grave, ma comunque si riesce sempre a coprire una buona parte anche dell'ultima passività (lo shortfall non si avvicina mai nemmeno al 50% della passività). Calcoliamo lo shortfall atteso nello scenario terminale:

```
mean_shortfall=0;
for i=tree.startNode(end):tree.numNodes
    mean_shortfall = mean_shortfall+...
        (tree.nodeCollection(i).prob*sol.shortfall(i));
end
```

```
mean_shortfall =
```

```
        4737.40
```

Per il modello implementato con la struttura ad albero $1-7-5-3-2$ chiaramente la politica di investimento cambia dal momento che si ha un numero molto maggiore di scenari per ciascun livello. Inoltre con la struttura più complessa si ha un numero di scenari terminali pari a 210. Per questioni di spazio non viene stampata tutto il vettore degli shortfall nello stato terminale (la soluzione si trova nell'appendice B). Si osserva tuttavia che, a differenza della struttura precedente, tale albero permette di risultare solvente in alcuni scenari (chiaramente quelli più favorevoli) e che in altri scenari lo shortfall risulta essere molto piccolo. Anche in questo caso, però, la maggior parte degli scenari terminali prevede una situazione di insolvenza, in alcuni casi non troppo grave, in altri casi molto grave (non si riesce a coprire neanche una piccola parte dell'ultimo flusso in uscita nei casi peggiori). Anche in questo caso si è sempre solventi nei primi 3 periodi. Calcoliamo lo shortfall atteso al termine dell'ultimo periodo:

```
mean_shortfall=0;
for i=tree.startNode(end):tree.numNodes
    mean_shortfall = mean_shortfall+...
        (tree.nodeCollection(i).prob*sol.shortfall(i));
end
```

```
mean_shortfall =

        4886.79
```

Lo shortfall atteso è leggermente superiore rispetto a quello ottenuta con la precedente struttura ad albero.

3.7 Conclusioni e possibili sviluppi

In generale è possibile risolvere il problema con qualsiasi struttura ad albero, anche con un numero di periodi ben superiore a 4. Chiaramente al crescere dell'orizzonte temporale T e della ramificazione dell'albero si può incorrere in problemi computazionali dal momento che ad ogni nodo deve essere risolto un problema di ottimizzazione per generare i nodi figli. Nella pratica potrebbe essere più conveniente generare degli scenari soltanto per un periodo di tempo e al termine dell'intervallo di tempo generare un nuovo insieme di possibili scenari per l'intervallo di tempo successivo che dipendano soltanto dallo scenario che si è realizzato, senza costruire tutto l'albero fin dall'inizio. GIà per generare l'albero con la struttura $1-7-5-3-2$ che contiene 358 nodi sono necessari all'incirca tre minuti, pertanto creare soltanto i nodi necessari ridurrebbe molto i costi computazionali.
Inoltre il modello può chiaramente essere raffinato introducendo delle ulteriori complicazioni, quali possono essere:

- passività stocastiche, che magari dipendono dal tasso di interesse in maniera diversa rispetto agli asset;
- asset che non siano necessariamente bond (per esempio degli swap);
- flussi di cassa in entrata (*deposit inflow*) che rappresentino nuovi depositi dei clienti nel caso di una banca;
- altri modelli per rappresentare l'evoluzione del tasso di interesse (anche modelli a più fattori) e altri modi per stimarne eventuali parametri.

Appendice A

Codici MATLAB secondari

A.1 Classe *ScenarioTreeClass*

```
classdef ScenarioTreeClass < handle


    properties
        branchingStructure  % a numeric array describing...
        % branching structure
                            % e.g., [1, 3, 3, 2]; NOTE the...
                            % leading 1
        timeHorizon         % number of time periods
        parametersCIR        % vector of the parameters rbar,...
        % gamma, alpha ,dt of the CIR
        nodeCollection      % an array of node objects
        nodesPerStage
        startNode
        numNodes
        numStates           % the number of state variables...
        % (if any) per node
    end

    methods
        %%      ------------------------------------------
        function obj = FillTree(obj, branchingStructure,...
        state0, parameters, PathGenerator)
            obj.branchingStructure = branchingStructure;
            obj.timeHorizon = length(branchingStructure);
            obj.parametersCIR = parameters;
            obj.numStates = length(state0);
            obj.SetTreeStructure
            % Fill nodes: values, states, and conditional...
            % probs using the path generator
            nodeIdx = 1;
```

```
32              obj.nodeCollection(nodeIdx).state = state0;
33              obj.nodeCollection(nodeIdx).condProb = 1;
34              obj.nodeCollection(nodeIdx).prob = 1;
35              for t = 1:(obj.timeHorizon-1)
36                  for nodeIdx = obj.startNode(t) :...
37                  (obj.startNode(t)+obj.nodesPerStage(t)-1)
38                      [stateOut, probsOut] = ...
39                          PathGenerator.Transition...
40                           (obj.nodeCollection(nodeIdx).state,...
41                           obj.branchingStructure(t+1), ...
42                          obj.parametersCIR);
43                      for k = 1:obj.branchingStructure(t+1)
44                          auxIdx =...
45                          obj.nodeCollection(nodeIdx).successors(k);
46                          obj.nodeCollection(auxIdx).state =...
47                          stateOut(k);
48                          obj.nodeCollection(auxIdx).condProb =...
49                          probsOut(k);
50                          obj.nodeCollection(auxIdx).prob =...
51                          probsOut(k)*...
52                          obj.nodeCollection(nodeIdx).prob ;
53                      end
54                  end
55              end
56
57          end
58
59          %       ------------------------------------------------
60          % fanList is an array of FanClass objects
61          function FillTreeIndependentFans(obj, fanList)
62              obj.timeHorizon = length(fanList);
63              obj.branchingStructure = zeros(obj.timeHorizon,1);
64              obj.branchingStructure(1) = 1;
65              for t=2:obj.timeHorizon
66                  obj.branchingStructure(t) =...
67                  length(fanList(t).values);
68              end
69              % Find number of values per node
70              obj.numValues = length(fanList(1).values{1});
71              obj.numStates = 0; % no state variable needed...
72              % for simulation
73              obj.SetTreeStructure
74              % Fill nodes: values and conditional probabilities;...
75              % state is left empty
76              nodeIdx = 1;
77              obj.nodeCollection(nodeIdx).values =...
78              fanList(1).values{1};
79              obj.nodeCollection(nodeIdx).condProb = 1; % must...
80              % be scalar at the root node
```

```
81              obj.nodeCollection(nodeIdx).prob = 1;
82              for t = 2:obj.timeHorizon
83                  for j = 1:obj.nodesPerStage(t-1)
84                      predNode =...
85                      obj.nodeCollection(nodeIdx + 1).predecessor;
86                      predProb = obj.nodeCollection(predNode).prob;
87                      for k = 1:obj.branchingStructure(t)
88                          nodeIdx = nodeIdx + 1;
89                          obj.nodeCollection(nodeIdx).values =...
90                          fanList(t).values{k};
91                          obj.nodeCollection(nodeIdx).condProb =...
92                          fanList(t).condProbs(k);
93                          obj.nodeCollection(nodeIdx).prob =...
94                          predProb*fanList(t).condProbs(k);
95                          obj.nodeCollection(nodeIdx).state = [];
96                      end
97                  end
98              end
99          end
100
101         %%      ------------------------------------------
102         function SetTreeStructure(obj)
103             % set number of tree nodes in each period...
104             % (including t=0)
105             obj.nodesPerStage = cumprod(obj.branchingStructure);
106             % set index of start node per stage
107             obj.startNode = [0;transpose(cumsum(...
108             obj.nodesPerStage(1:end-1)))]+1;
109             % find total number of nodes and create cell...
110             % array of node objects
111             obj.numNodes=sum(obj.nodesPerStage);
112             aux(obj.numNodes,1) = NodeClass;
113             obj.nodeCollection = aux;
114             % set predecessor and time
115             obj.nodeCollection(1).predecessor = NaN;
116             obj.nodeCollection(1).timeIndex = 1;
117             for t=2:obj.timeHorizon
118                 for j=obj.startNode(t) : (obj.startNode(t)+...
119                 obj.nodesPerStage(t)-1)
120                     obj.nodeCollection(j).predecessor =...
121                     obj.startNode(t-1) + floor((j-...
122                     obj.startNode(t))/obj.branchingStructure(t));
123                     obj.nodeCollection(j).timeIndex = t;
124                 end
125             end
126             % set successors
127             for t = 1:obj.timeHorizon-1
128                 %nodesNow = (obj.startNode(t) :...
129                 obj.startNode(t)+obj.nodesPerStage(t)-1);
```

```
130                 nodesAfter = (obj.startNode(t+1) :...
131                 obj.startNode(t+1)+obj.nodesPerStage(t+1)-1);
132                 columnSize = obj.nodesPerStage(t);
133                 rowSize = round(obj.nodesPerStage(t+1)/...
134                 obj.nodesPerStage(t));
135                 nodeMatrix = reshape(nodesAfter, rowSize,...
136                 columnSize);
137                 for j = 1:obj.nodesPerStage(t)
138                     obj.nodeCollection(obj.startNode(t)+j-...
139                     1).successors = nodeMatrix(:,j);
140                 end
141             end
142         end
143 
144         %%      ------------------------------------------------
145         function nodeList = FindNodesTime(obj, t)
146             nodeList = find([obj.nodeCollection(:).timeIndex]...
147             == t);
148         end
149 
150         %%      ------------------------------------------------
151         function node = GetNode(obj, nodeIdxIn)
152             node = obj.nodeCollection(nodeIdxIn);
153         end
154 
155         %%      ------------------------------------------------
156         function [probsArray, condProbsArray] =...
157         MakeProbsArray(obj)
158             probsArray = [obj.nodeCollection(:).prob];
159             condProbsArray = [obj.nodeCollection(:).condProb];
160         end
161 
162         %%      ------------------------------------------------
163         function predecessorsArray =...
164         MakePredecessorsArray(obj)
165             predecessorsArray =...
166             [obj.nodeCollection(:).predecessor];
167         end
168 
169         %%      ------------------------------------------------
170         function timesArray = MakeTimesArray(obj)
171             timesArray = [obj.nodeCollection(:).timeIndex];
172         end
173 
174         %%      ------------------------------------------------
175         function valuesMatrix = MakeValuesMatrix(obj)
176             % should improve by making sure that values are...
177             % column vectors
178             % rows correspond to nodes
```

```
179             valuesMatrix = zeros(obj.numValues, obj.numNodes);
180             for j = 1:obj.numNodes
181                 valuesMatrix(:,j) =...
182                 obj.nodeCollection(j).values(:);
183             end
184             valuesMatrix = transpose(valuesMatrix);
185         end
186 
187         %%      ------------------------------------------
188         function nodeIdx = FindPredecessor(obj, nodeIdxIn)
189             nodeIdx = obj.nodeCollection(nodeIdxIn).predecessor;
190         end
191 
192         %%      ------------------------------------------
193         function nodeList = FindSuccessors(obj, nodeIdxIn)
194             nodeList = obj.nodeCollection(nodeIdxIn).successors;
195         end
196 
197         %%      ------------------------------------------
198         function [nodeList, prob] = FindPathToNode(obj,...
199         nodeIdxIn)
200             t = obj.nodeCollection(nodeIdxIn).timeIndex;
201             nodeList = zeros(t, 1);
202             currentNode = nodeIdxIn;
203             nodeList(t) = nodeIdxIn;
204             prob = obj.nodeCollection(currentNode).condProb;
205             while ~isnan(obj.nodeCollection(...
206             currentNode).predecessor)
207                 currentNode = obj.nodeCollection(...
208                 currentNode).predecessor;
209                 t = t-1;
210                 nodeList(t) = currentNode;
211                 prob = prob * obj.nodeCollection(...
212                 currentNode).condProb;
213             end
214         end
215 
216         %%      ------------------------------------------
217         function scenarioList = FindScenarios(obj)
218             leafNodes = obj.FindNodesTime(obj.timeHorizon);
219             numScenarios = length(leafNodes);
220             scenarioList(numScenarios,1) = ScenarioClass;
221             valueMatrix = zeros(obj.numValues, obj.timeHorizon);
222             % values per scenario: vars on rows and times on...
223             % columns
224             for j = 1:numScenarios
225                 [scenarioList(j).nodeList, scenarioList(j).prob]...
226                =  obj.FindPathToNode(leafNodes(j));
227                 for t = 1:obj.timeHorizon
```

```
228                     valueMatrix(:,t) = obj.nodeCollection(...
229                     scenarioList(j).nodeList(t)).values;
230                 end
231                 scenarioList(j).valueMatrix = valueMatrix;
232             end
233         end
234 
235         %%      ------------------------------------------
236         function nodeMap = FindNodeMap(obj)
237             % the node map is a matrix with rows corresponding...
238             % to scenarios and
239             % columns corresponding to time instants, giving the
240             % corresponding node index
241             leafNodes = obj.FindNodesTime(obj.timeHorizon);
242             numScenarios = length(leafNodes);
243             nodeMap = zeros(numScenarios, obj.timeHorizon);
244             for j = 1:numScenarios
245                 nodeMap(j,:) = obj.FindPathToNode(leafNodes(j));
246             end
247         end
248 
249 
250         %%      ------------------------------------------
251         function inverseNodeMap = FindInverseNodeMap(obj)
252             % the inverse node map is a cell array of ragged...
253             % vectors, giving for each node the list of ...
254             % scenarios to which it belongs
255             nodeMap = obj.FindNodeMap;
256             inverseNodeMap = cell(obj.numNodes,1);
257             for j = 1:obj.numNodes
258                 time = obj.nodeCollection(j).timeIndex;
259                 inverseNodeMap{j} = find(nodeMap(:,time) == j);
260             end
261         end
262 
263 
264     end  % METHODS
265 
266 end
```

A.2 Classe *NodeClass*

```
1 classdef NodeClass < handle
2     % Node class for use in scenario trees
3     % It only has the constructor method
4 
5     properties
6         predecessor     % index of predecessor node in the tree
7                         % NaN for the root node
```

```
        successors      % an array of index nodes
        prob            % the UNconditional probability of...
        % that node
        condProb        % the conditional probability of...
        % that node
        timeIndex       % time instant (starting from 1 for...
        % root node)
        values          % an array of values
        state           % an array of values
    end

    methods

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Constructor: use default values
        function obj = NodeClass()
            obj.predecessor = NaN;
            obj.successors = [];
            obj.prob = NaN;
            obj.condProb = NaN;
            obj.timeIndex = NaN;
            obj.values = [];
            obj.state = [];
        end

    end % METHODS

end % CLASS
```

A.3 Classe *PathGeneratorClass*

```
classdef (Abstract) PathGeneratorClass < handle
    % Abstract class defining a scenario generator

    % The concrete class must provide a Transition method with
    % Input: valuesIn, stateIn, howMany
    % Output: valuesOut, stateOut, probsOut
    methods (Abstract)
        Transition(obj, stateIn, howMany, parameters)
    end


end % CLASS
```

A.4 Classe *CIRPathGeneratorClass*

```
classdef CIRPathGeneratorClass < PathGeneratorClass

```

```
3       methods
4           function [stateOut, probsOut] = Transition(obj, r0,...
5           numScenarios, parameters)
6               rbar = r0;
7               gamma = parameters(2);
8               alpha = parameters(3);
9               dt = parameters(4);
10              [moments,low,up] = CIRmoments (r0,rbar,gamma,...
11              alpha,dt);
12              bounds = [low, up];
13              [stateOut, probsOut]=Scenarios_generator(...
14              numScenarios,moments,bounds);
15          end
16      end
17  end
```

Appendice B

Soluzioni dei modelli di ottimizzazione

B.1 Soluzione del modello Russell-Yasuda-Kasai con struttura $1-7-5-3-2$

La variabile *asset_value* della soluzione del modello è rappresentata nella seguente tabella, dove ogni riga rappresenta un diverso nodo.

Bond 1	Bond 2	Bond 3	Bond 4
39849.6611	0	15150.3389	0
42571.1198	0	0	0
40711.7356	0	0	0
15226.8306	0	0	24126.2994
18417.8983	0	0	19638.2784
21507.6233	0	0	15059.6399
25438.51	0	0	9033.9953
30400.2089	0	0	0

Bond 1	Bond 2	Bond 3	Bond 4
1168.9002	0	25635.6049	0
4462.7182	0	0	21616.6464
6006.6693	0	0	19584.5911
7062.6753	0	0	17872.7721
21982.548	0	0	0
0	0	0	26789.3067
0	0	0	26170.9159
0	0	0	25685.4701
0	0	0	25054.4929
0	0	23465.2307	0
0	0	0	28690.6001
0	0	25308.0958	0
142.7161	0	0	22820.6862
6702.532	0	0	13505.9574
0	0	14629.0829	0
0	0	0	27781.9939
0	0	0	24398.9453
2398.0185	0	0	19763.1229
8124.767	0	0	11481.3233
0	0	14492.6826	0
0	0	0	26471.8993
0	0	0	23226.8138
4971.0959	0	0	16200.3538
0	0	18897.2584	0
14428.7014	0	0	0
0	0	0	24125.2603
4499.7447	0	0	16873.1568
0	0	19702.0087	0
0	0	17890.8819	0
14335.4029	0	0	0
9493.5273	0	0	9162.0267
17441.0999	0	0	0
0	0	16674.0017	0
15802.7071	0	0	0
13897.0153	0	0	0
0	0	0	13805.0959
10458.1769	0	0	0
5331.8883	0	0	0
14825.1203	0	0	0
9672.9126	0	0	0
3264.024	0	0	0

Bond 1	Bond 2	Bond 3	Bond 4
14770.4074	0	0	0
9408.8498	0	0	0
0	0	0	3175.4195
14691.9496	0	0	0
9003.9662	0	0	0
0	0	0	2864.3667
8123.5961	0	0	0
8120.9578	0	0	0
0	8018.9062	0	0
12240.416	0	0	0
9622.327	0	0	0
3705.0951	0	0	0
13206.9688	0	0	0
9309.6577	0	0	0
3069.5804	0	0	0
13738.3659	0	0	0
8893.6711	0	0	0
2118.1753	0	0	0
14308.9295	0	0	0
8454.5416	0	0	0
1196.9883	0	0	0
9213.7069	0	0	0
8809.2549	0	0	0
8408.6676	0	0	0
14759.5143	0	0	0
11409.306	0	0	0
4715.7505	0	0	0
10373.6728	0	0	0
10187.2593	0	0	0
9865.9771	0	0	0
11451.4535	0	0	0
6455.9731	0	0	0
0	0	0	0
8168.7924	0	0	0
4395.5026	0	0	0
0	0	0	0
71.0391	0	0	0
37.2724	0	0	0
0	0	0	0
14390.9693	0	0	0
10767.8164	0	0	0

Bond 1	Bond 2	Bond 3	Bond 4
4508.5909	0	0	0
12628.1183	0	0	0
7727.2692	0	0	0
1116.9674	0	0	0
10797.9596	0	0	0
5913.2555	0	0	0
0	0	0	0
7599.8907	0	0	0
3999.486	0	0	0
0	0	0	0
0.1595	0	0	0
0.065306	0	0	0
0	0	0	0
13658.7359	0	0	0
9605.461	0	0	0
3226.2213	0	0	0
12032.2825	0	0	0
6728.6306	0	0	0
57.9429	0	0	0
9791.1624	0	0	0
5219.8186	0	0	0
0	0	0	0
4512.5456	0	0	0
4118.2158	0	0	0
3739.3672	0	0	0
70.1389	0	0	0
0	0	0	34.4312
0	0	0	0
0	0	0	11975.3549
7438.4304	0	0	0
1090.6728	0	0	0
9995.5694	0	0	0
5361.8118	0	0	0
0	0	0	0
5171.0891	0	0	0
4979.0906	0	0	0
4792.6851	0	0	0
3483.6381	0	0	0
3253.8853	0	0	0
3148.058	0	0	0
117.3924	0	0	0

Bond 1	Bond 2	Bond 3	Bond 4
116.8841	0	0	0
0	0	0	0
5609.7704	0	0	0
3069.5652	0	0	0
0	0	0	0
2847.2281	0	0	0
2770.0308	0	0	0
2688.9637	0	0	0
2238.6944	0	0	0
2238.6725	0	0	0
2238.6533	0	0	0
0	0	0	1553.5608
0	0	0	1553.5044
0	0	0	1553.4642
15.8615	0	0	0
12.0905	0	0	0
0	0	0	0

Tabella B.1. Politica di investimento che risolve il problema di Russell-Yasuda-Kasai con struttura dell'albero $1-7-5-3-2$.

Lo shortfall nei nodi foglia è dato da:

```
sol.shortfall(tree.startNode(end):end)'=

            0
         5.30
      4316.32
      4316.39
      9404.44
      9404.45
            0
         1.39
      5040.55
      5049.00
     11532.65
     11532.68
            0
         0.74
      5247.86
      5248.49
     11207.18
     12844.08
            0
         0.24
      5592.26
```

```
 5592.69
 9241.00
11322.47
 6240.34
 6240.34
 6245.10
 6249.29
 6363.52
 6364.10
 2757.37
 2771.70
 5289.53
 5289.56
11179.23
11182.11
 1740.09
 1740.09
 5539.49
 5540.56
11806.47
11807.83
 1211.48
 1211.48
 5907.19
 5907.91
12774.39
12774.54
  523.57
  524.33
 6295.77
 6303.16
13728.66
13728.67
 5384.90
 5384.95
 5790.11
 5797.65
 6188.36
 6200.09
  213.77
  213.77
 3458.98
 3459.29
10114.78
10115.00
 4482.28
 4482.38
 4662.09
 4662.73
```

```
 4973.41
 4974.15
 3432.98
 3433.16
 8366.33
 8366.33
15000.00
15000.00
 6702.65
 6703.22
10442.92
10442.96
15000.00
15000.00
14925.19
14925.19
14960.74
14960.74
15000.00
15000.00
  593.03
  596.78
 4086.04
 4086.43
10325.37
10325.39
 2266.77
 2266.78
 7083.38
 7085.04
13822.97
13823.03
 4062.12
 4062.39
 8900.32
 8904.82
15000.00
15000.00
 7252.91
 7253.60
10831.02
10831.06
15000.00
15000.00
14999.83
14999.83
14999.93
14999.93
15000.00
```

```
15000.00
 1282.36
 1282.36
 5233.03
 5233.41
11638.14
11638.29
 2834.32
 2834.48
 8074.62
 8074.62
14938.53
14938.54
 5046.20
 5048.08
 9581.31
 9581.62
15000.00
15000.00
10300.69
10300.79
10701.18
10701.71
11087.65
11088.20
14925.14
14925.14
14958.96
14976.58
15000.00
15000.00
 1926.05
 4746.60
 7395.79
 7395.84
13854.20
13854.25
 4846.75
 4847.70
 9440.85
 9440.86
15000.00
15000.00
 9624.25
 9626.87
 9819.53
 9821.68
10004.36
10004.40
```

```
11334.24
11334.37
11571.07
11571.23
11680.33
11680.67
14872.94
14872.94
14873.48
14873.49
15000.00
15000.00
 9310.52
 9310.54
11825.02
11825.03
15000.00
15000.00
12023.97
12023.97
12092.74
12092.75
12165.50
12165.79
12628.59
12628.60
12628.62
12628.62
12628.64
12628.64
12595.20
13580.21
12595.27
13580.26
12595.30
13580.29
14982.41
14982.44
14986.59
14986.61
15000.00
15000.00
```

Bibliografia

[1] Paolo Brandimarte. *An Introduction to Financial Market, A Quantitative Approach.* Wiley, 2018.

[2] David R. Cariño and William T. Ziemba. Formulation of the Russell-Yasuda Kasai Financial Planning Model. *Operations Research*, 46(4):433:449, 1998.

[3] David R. Cariño, Terry Kent, David H. Myers, Celine Stacy, Mike Sylvanus, Andrew L. Turner, Kouji Watanabe, and William T. Ziemba. The Russell-Yasuda Kasai Model: An Asset/Liability Model for a Japanese Insurance Company Using Multistage Stochastic Programming. *INFORMS Journal on Applied Analytics*, 24(1):29–49, 1994.

[4] David R. Cariño, William T. Ziemba, and David H. Myers. Concepts, Technical Issues, and Uses of the Russell-Yasuda Kasai Financial Planning Model. *Operations Research*, 46(4): 450:462, 1998.

[5] Nalan Gülpinar, Berç Rustem, and Reuben Settegren. Simulation and optimization approaches to scenario tree generation. *Journal of Economic Dynamics e Control*, 28:1291:1315, 2004.

[6] Kamil Kladívko. Maximum likelihood estimation of the cox-ingersoll-ross process: The matlab implementation.

Sitografia

[1] *https://corporatefinanceinstitute.com/resources/knowledge/strategy/asset-and-liability-management-alm/*, 14 luglio 2021

[2] *https://www.investopedia.com/financial-term-dictionary-4769738*, 4 settembre 2021

[3] *https://www.investor.gov/introduction-investing/investing-basics/glossary/zero-coupon-bond*, 22 settembre 2021

[4] *https://en.wikipedia.org/wiki/Goal-based_investing*, 22 agosto 2021

[5] *https://it.mathworks.com/help/matlab/*, 12 settembre 2021