STEPHENS, JR., JAMES MILTON. Regime-Switching Nonlinear Optimal-Control. (Under the direction of Negash G. Medhin).

We consider an optimal-control problem with state constraints given by regime-switching nonlinear stochastic dynamics. Control elements are either vectors from, or measures (of total measure 1) defined on, a compact subset of Euclidean space; the former we call ordinary controls and the latter we call relaxed controls. We do not assume the underlying subset is convex.

There are three main results. We first discuss existence of solutions to the optimal-control problem. We show that there exist strong solutions to the state equation, and that any minimizing sequence of relaxed controls is weakly convergent. In the latter the Prokhorov Criterion is critical. We then discuss how applying the Skorokhod Coupling can produce strong solutions.

We next give two necessary characterizations of an optimal control: the Pontryagin Minimum Principle and the Bellman Dynamic Programming Principle. On the way to the former we derive a variational inequality, which we show to be linear in control after reformulation by a costate process. We then prove the optimal solution (control, state, and costate) minimizes a Hamiltonian. The second characterization is given in terms of a value function. Under mild regularity conditions, we demonstrate it satisfies the Hamilton-Jacobi-Bellman equation, a nonlinear second-order partial differential equation with integral terms. We also make explicit the connection between the two characterizations; we show the costate to be certain partial derivatives of the value function.

Finally, we apply our theory to the development of an alternative to the Black-Scholes pricing equation for options. We provide a numerical procedure for solving the associated Hamilton-Jacobi-Bellman equation: separate the variables, approximate the state variables by cubic splines, and apply a 4-stage Runge-Kutta algorithm. We also provide results for a variety of market conditions.
Regime-Switching Nonlinear Optimal-Control

by
James Milton Stephens, Jr.

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina
2023

APPROVED BY:

Min Kang
Tao Pang
Andrew Papanicolaou
Negash G. Medhin
Chair of Advisory Committee
DEDICATION

To the memory of my father, James, who died before seeing this dissertation completed; and to my loving mother, Mary, who never wavered in her support.
BIOGRAPHY

The author was born at Ft. Lewis, Washington. A child of the military, he grew up mainly in (then) West Germany and in North Carolina. He studied at North Carolina State University, majoring in Philosophy, after which he put his training in logic to good use as an operating-systems programmer and microprocessor performance-modeler in the technology industry in Research Triangle Park. His casual work in queueing theory led him to take up formal studies again, this time in Applied Mathematics.
ACKNOWLEDGEMENTS

My deepest and most heartfelt appreciation to my dissertation supervisor, Negash G. Medhin, for the most remarkable and persistent support any student has ever received, in times good, and in times tragic.
TABLE OF CONTENTS

List of Figures ........................................................................................................ vi

Chapter 1  The Optimal-Control Problem ............................................................... 1
  1.1 Examples ........................................................................................................... 2
  1.2 Elements of the Problem ............................................................................... 4
  1.3 Mathematical Formulation ............................................................................ 9
  1.4 The Ordinary-State Equation ....................................................................... 10
  1.5 The Optimal Relaxed-Control ..................................................................... 23

Chapter 2  Necessary Conditions for Optimality .................................................... 25
  2.1 Perturbations, Part 1: The Variational Inequality ....................................... 26
  2.2 Perturbations, Part 2: Pontryagin’s Minimum Principle ............................ 44
  2.3 Bellman’s Dynamic Programming Principle ............................................. 51
  2.4 The Costate Process and the Value Function, Revisited ......................... 56

Chapter 3  An Application in Financial Derivatives .............................................. 62
  3.1 A Précis of Financial Markets .................................................................... 63
  3.2 A Brief Introduction to the Black-Scholes Pricing Equation ..................... 65
  3.3 An Alternative Formulation Using Optimal Control ................................. 67
  3.4 Numerical Solutions to the Hamilton-Jacobi-Bellman Equation............... 74
  3.5 Our Alternative in Action .......................................................................... 80

References ............................................................................................................. 86

APPENDICES ........................................................................................................ 88
  Appendix A  Python Code for Solving the Hamilton-Jacobi-Bellman Equation . 89
  Appendix B  Configuration and Validation with Jupyter Notebooks ............... 116
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Natural Cubic Spline (Example)</td>
<td>75</td>
</tr>
<tr>
<td>3.2</td>
<td>Natural Cubic Spline Basis (Example)</td>
<td>76</td>
</tr>
<tr>
<td>3.3</td>
<td>Natural Cubic Spline Basis, First Derivatives (Example)</td>
<td>76</td>
</tr>
<tr>
<td>3.4</td>
<td>Natural Cubic Spline Basis, Second Derivatives (Example)</td>
<td>77</td>
</tr>
<tr>
<td>3.5</td>
<td>Natural Cubic Spline Basis, Third Derivatives (Example)</td>
<td>77</td>
</tr>
<tr>
<td>3.6</td>
<td>Value Function at ( t=0 )</td>
<td>84</td>
</tr>
<tr>
<td>3.7</td>
<td>Value Function per Initial Investment</td>
<td>85</td>
</tr>
<tr>
<td>B.1</td>
<td>Control Discretization, Value Function at ( t=0 )</td>
<td>117</td>
</tr>
<tr>
<td>B.2</td>
<td>Control Discretization, Value Function per Initial Investment</td>
<td>118</td>
</tr>
<tr>
<td>B.3</td>
<td>Control Discretization, Quote per Control Discretization</td>
<td>118</td>
</tr>
<tr>
<td>B.4</td>
<td>Time Discretization, Value Function at ( t=0 )</td>
<td>124</td>
</tr>
<tr>
<td>B.5</td>
<td>Time Discretization, Value Function per Initial Investment</td>
<td>125</td>
</tr>
<tr>
<td>B.6</td>
<td>Time Discretization, Quote per Time Discretization</td>
<td>125</td>
</tr>
<tr>
<td>B.7</td>
<td>Diffusion, Value Function at ( t=0 )</td>
<td>131</td>
</tr>
<tr>
<td>B.8</td>
<td>Diffusion, Value Function per Initial Investment</td>
<td>132</td>
</tr>
<tr>
<td>B.9</td>
<td>Diffusion, Quote per Diffusion</td>
<td>132</td>
</tr>
<tr>
<td>B.10</td>
<td>Expiry, Value Function at ( t=0 )</td>
<td>138</td>
</tr>
<tr>
<td>B.11</td>
<td>Expiry, Value Function per Initial Investment</td>
<td>139</td>
</tr>
<tr>
<td>B.12</td>
<td>Expiry, Quote per Expiry</td>
<td>139</td>
</tr>
<tr>
<td>B.13</td>
<td>Interest Rate, Value Function at ( t=0 )</td>
<td>145</td>
</tr>
<tr>
<td>B.14</td>
<td>Interest Rate, Value Function per Initial Investment</td>
<td>146</td>
</tr>
<tr>
<td>B.15</td>
<td>Interest Rate, Quote per Interest Rate</td>
<td>146</td>
</tr>
<tr>
<td>B.16</td>
<td>Jump Frequency, Value Function at ( t=0 )</td>
<td>152</td>
</tr>
<tr>
<td>B.17</td>
<td>Jump Frequency, Value Function per Initial Investment</td>
<td>153</td>
</tr>
<tr>
<td>B.18</td>
<td>Jump Frequency, Quote per Jump Frequency</td>
<td>153</td>
</tr>
<tr>
<td>B.19</td>
<td>Jump Intensity, Value Function at ( t=0 )</td>
<td>159</td>
</tr>
<tr>
<td>B.20</td>
<td>Jump Intensity, Value Function per Initial Investment</td>
<td>160</td>
</tr>
<tr>
<td>B.21</td>
<td>Jump Intensity, Quote per Jump Intensity</td>
<td>160</td>
</tr>
<tr>
<td>B.22</td>
<td>Regime Frequency, Value Function at ( t=0 )</td>
<td>166</td>
</tr>
<tr>
<td>B.23</td>
<td>Regime Frequency, Value Function per Initial Investment</td>
<td>167</td>
</tr>
<tr>
<td>B.24</td>
<td>Regime Frequency, Quote per Regime Frequency Scenario</td>
<td>167</td>
</tr>
<tr>
<td>B.25</td>
<td>Regime Intensity, Value Function at ( t=0 )</td>
<td>173</td>
</tr>
<tr>
<td>B.26</td>
<td>Regime Intensity, Value Function per Initial Investment</td>
<td>174</td>
</tr>
<tr>
<td>B.27</td>
<td>Regime Intensity, Quote per Regime Intensity Scenario</td>
<td>174</td>
</tr>
<tr>
<td>B.28</td>
<td>Strike, Value Function at ( t=0 )</td>
<td>181</td>
</tr>
<tr>
<td>B.29</td>
<td>Strike, Value Function per Initial Investment</td>
<td>181</td>
</tr>
<tr>
<td>B.30</td>
<td>Strike, Quote per Strike</td>
<td>182</td>
</tr>
</tbody>
</table>
“Control theory is the mathematical study of how to influence the behavior of a dynamical system to achieve a desired goal. In optimal control, the goal is to maximize or minimize the numerical value of a specified quantity that is a function of the behavior of the system.” (Berkovitz and Medhin 2013) The problem we tackle is the optimal control of a system with stochastic dynamics. In particular, we are interested in dynamics which are regime-switching nonlinear jump-diffusions and cost functionals which are determined by the entire system (including regime).

Our plan is as follows. In this chapter we introduce the optimal-control problem and show that a solution exists. In chapter 2 we derive a pair of necessary conditions for optimality. Finally, in chapter 3 we apply one of our necessary conditions to solve the practical problem of pricing a straight-forward financial derivative.

In particular, our plan for the present chapter is as follows. In section 1.1 we present a few examples from the literature. The examples start with a simple deterministic problem. We then extend that problem to the stochastic case by allowing the dynamics to be driven by a Brownian motion. Lastly, we motivate relaxed controls via an example with ‘pathological’ constraints on the control. In all cases, we claim no originality in the examples. In section 1.2 we present terms and notation that will be used throughout this dissertation. Of particular interest to the reader already familiar with the ordinary control of diffusions are the definitions of relaxed-control processes, the Markov chain that specifies the regime, and the Poisson random measures that describe the jumps in the state process. In section 1.3 we give detailed statements of our ordinary- and relaxed-control
problems. The bulk of the theory of this dissertation rests in sections 1.4 and 1.5. In the former we show there exists a unique strong solution of the ordinary-state equation. This result is modified easily then to cover the relaxed-state equation. In the latter section we show how to start to prove there exists a solution of the optimal-control problem. To this end we develop the definition of relaxed control given in section 1.2.

Our research rests on a foundation of knowledge in functional analysis and measure theory, probability theory, partial differential equations, and numerical analysis. Obviously there are many textbooks that cover that material. We list those we relied on the most in developing our foundation. All of the functional analysis and measure theory needed is in Friedman (1982). For probability theory we used Çınlar (2011) for the basic results and Kallenberg (2002) for the more advanced results. We required additional material for the topics of stochastic integration and Lévy processes. For the former, we found the $\delta - \epsilon$ approach in Friedman (2006) more instructive than the Hilbert-space approach of, for example, Protter (2004). When extending the theory of stochastic integration to general martingales we used Kuo (2006). All of the material specific to Lévy processes was found in Applebaum (2009). While we read widely on partial differential equations, we returned most often to McOwen (2003), supplemented by Crandall et al. (1992) for the theory of viscosity solutions and by Duistermaat and Kolk (2010) for the theory of generalized functions (also known as distributions). Finally, our primary reference for the numerical analysis we built upon was taken from Quarteroni et al. (2007).

The literature for optimal control is quite extensive. For easy introductions, and for ones that emphasize applications, there are Luenberger (1969) and Stengel (1994). Berkovitz and Medhin (2013) gives a comprehensive account of the theory of optimal control in the deterministic case. Transitioning to stochastic problems, Bensoussan (1988) develops both the Pontryagin Minimum Principle and the Bellman Dynamic Programming Principle for the diffusion model. Framstad et al. (2004) and Oksendal and Sulem (2007) study sufficient conditions for optimality for the jump-diffusion model: the former via the Pontryagin Minimum Principle while the later via the Bellman Dynamic Programming Principle. We were first introduced to relaxed controls in the case of the diffusion model in Bahlali (2008).

Finally, all computations were performed on an Apple iMac. The machine contained a 3.1 GHz 6-Core (12 logical cores) Intel Core i5 CPU with 12 MB L3 cache, and 16 GB 2667 MHz DDR4 of RAM. The software stack was as follows: Apple macOS Ventura (version 13.0.1), Python 3.9.13, Intel’s Math Kernel Library (MKL) 2021.4.0, NumPy 1.21.5, SciPy 1.9.1, and JupyterLab 3.4.4.

1.1 Examples

This section is self-contained, and is not essential to what follows. All terms and notation are specific to it. The reader already familiar with optimal control generally, and relaxed controls specifically,
EXAMPLE 1: Let \( n \) be a positive integer and \( i = 1, 2, \ldots, n \). Let \( x_i(t) \in \mathbb{R} \) denote the inventory level of some intermediate good and \( D_i > 0 \) its demand level. For now, assume \( D_i \) is known and constant. Let \( u_i(t) \in \mathbb{R} \) denote the production rate for the corresponding intermediate good. In vector notation, the state equation for the intermediate goods is

\[
x'(t) = u(t) - D \]

\[
x(0) = x_0
\]

Let \( \alpha_i(\cdot) \) denote the storage cost and \( \gamma_i(\cdot) \) the shortage penalty for each intermediate good. Let \( \psi(\cdot) \) denote the overhang at production-run completion, that is, at time \( T \). For convenience, assume \( \alpha_i(0) = \gamma_i(0) = 0 \) and \( \psi(0) = 0 \) (where the 0 argument to \( \psi \) is \( 0 \in \mathbb{R}^n \)). Clearly, the total cost of the production run is

\[
\int_0^T \sum_{i=1}^n \left[ \alpha_i(\max\{x_i(t), 0\}) + \gamma_i(\max\{-x_i(t), 0\}) \right] dt + \psi(x(T))
\]

Let \( U \subset \mathbb{R}^n \) denote the production-rate limits, that is \( u(t) \in U \) for each \( 0 \leq t \leq T \). The optimal-control problem then is to minimize over all reasonable \( u(t) \) the cost of the production run subject to the dynamics of production.

EXAMPLE 2: In case the demand in the previous example is not only not constant but also not deterministic, the above can be reformulated as follows. Let the demand \( \{D_t\} \) be given \( \mathbb{P} - a.s. \) as

\[
dD_t = b D_t \, dt + \sigma D_t \, dB_t
\]

\[
D_0 = \hat{D}
\]

where \( b \) and \( \sigma \) are model parameters, and \( \{B_t\} \) a Brownian motion. The state equation then satisfies \( \mathbb{P} - a.s. \)

\[
dx_i = (u_t - b D_t)dt - \sigma D_t dB_t
\]

\[
x_0 = \hat{x}
\]

The expectations of the total cost of the production run then is

\[
\mathbb{E}\left[ \int_0^T \sum_{i=1}^n \left[ \alpha_i(\max\{x_i^t, 0\}) + \gamma_i(\max\{-x_i^t, 0\}) \right] dt + \psi(x_T) \right]
\]

(Note that the vector component is not indicated by superscript.) Now the optimal-control problem
is to minimize over all reasonable \( \{ u_t \} \) the expectations of the cost of the production run subject to the stochastic dynamics of production.

**EXAMPLE 3:** It is not clear in the previous examples that there could arise in some optimal-control problems a \( U \) such that, while the objective (the cost of the production run in the examples above) has a clear minimum, there exists no control (production rate in the examples above) that actually attains that minimum. To illustrate such a case, consider the following toy problem (not related to production planning). Let \( x(t) \in \mathbb{R} \) and \( u(t) \in \{-1, 1\} \). The state equation is given as

\[
\begin{align*}
  x'(t) &= u(t) \\
  x(0) &= 0
\end{align*}
\]

and the objective is given as

\[
J = \int_0^T [x(t)]^2 \, dt
\]

Suppose the problem is to minimize this objective. Clearly the objective is bounded below by 0. Moreover, no control can obtain this minimum because that would imply \( x(t) = 0 \) on \([0, T]\), which in turn would require \( u(t) = 0 \) on \([0, T]\). However, for each \( n = 1, 2, 3, \ldots \); consider the control given by

\[
u_n(t) = (-1)^k \quad \text{if} \quad \frac{k}{n} T \leq t \leq \frac{k+1}{n} T, \quad 0 \leq k \leq n - 1
\]

Denote by \( x_n(t) \) the state controlled by \( u_n(t) \), and by \( J_n \) its associated objective. Then \( 0 \leq x_n(t) \leq \frac{1}{n} T \) and \( 0 \leq J_n \leq \frac{1}{n^2} T^3 \). That is, 0 is the minimum insofar as it is the greatest lower bound, but there is no control that attains this minimum.

### 1.2 Elements of the Problem

Let \( T > 0 \) be a fixed deterministic time. Unless otherwise stated, \( t \in [0, T] \). Let \( C \gg 1 \) be a constant. As there will be need in what follows for only finitely many such positive constants we take \( C \) to be a constant universal in application throughout this dissertation. Unless otherwise indicated all scalars, vectors, and matrices are real. If \( a \) is a vector and \( A \) a matrix, by \( a^i \) we mean the \( i^{th} \) component of \( a \); by \( A^{ij} \), \( A^i\* \), and \( A^*j \) we mean the \((i, j)^{th}\) component, \( i^{th} \) row, and \( j^{th} \) column, respectively, of \( A \); and by \( A^\top \) we mean the transpose of \( A \). Let \( N, K, D, L, \) and \( M \) be positive integers. Unless otherwise stated, \( n, k, d, l, \) and \( m \) are arbitrary positive integers between (inclusive) 1 and \( N, K, D, L, \) and \( M \), respectively.

For any topological space \( X \) we denote by \( \mathcal{B}(X) \) the Borel \( \sigma \)-algebra on the space and call any
element of $\mathcal{B}(X)$ a Borel set. If $X$, $Y$, and $Z$ are topological spaces we denote the set of $Z$-valued continuous function on $X \times Y$ by $C(X \times Y; Z)$. If these are metric spaces, this notation is extended in a natural way to continuously differentiable functions. For example, a function is in $C^{0,1}(X \times Y; Z)$ if both it and its partial derivative in the second argument are jointly continuous. We remark that the superscript 0 does not imply compact support or rapidly decreasing to 0, as is sometimes the case; we use a subscript to denote compact support.

Let $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, \mathbb{P})$ be a filtered probability space. We write $\{\mathcal{F}_t\}$ for the filtration and $\mathcal{F}_t$ for the $\sigma$-algebra at time $t$. Likewise we write $\{X_t\}$ for a process and $X_t$ for its value at a particular $t$. We always assume that $\{\mathcal{F}_t\}$ is both complete and right-continuous, that is, $\mathcal{F}_0$ contains all subsets of null sets and $\mathcal{F}_t = \bigcap_{\varepsilon > 0} \mathcal{F}_{t+\varepsilon}$. As one can always augment in a straightforward manner any filtration such that it satisfies these conditions, our assumption that $\{\mathcal{F}_t\}$ does indeed satisfy these conditions is conventional.

We say $\{X_t\}$ is a càdlàg, càglàd, or continuous process whenever $\mathbb{P} - a.s.$ its paths are right-continuous with left hand limits, left-continuous with right hand limits, or continuous, respectively. If for each $t$ the map $\omega \rightarrow X_t(\omega)$ is $\mathcal{F}_t$-measurable we say the process is adapted. If for each $t$ and $s \in [0, t]$ the map $(s, \omega) \rightarrow X_s(\omega)$ is measurable with respect to $\mathcal{B}([0, t]) \otimes \mathcal{F}_t$, we say the process is progressive. Whenever the map $(t, \omega) \rightarrow X_t(\omega)$ is measurable with respect to the $\sigma$-algebra generated by all càdlàg adapted processes we say the process is optional, and whenever it is measurable with respect to the $\sigma$-algebra generated by all càglàd adapted processes we say the process is predictable. We remark that the predictable $\sigma$-algebra can also be generated by all continuous adapted processes. Clearly all predictable processes are optional, and all optional processes are progressive. In the sequel we shall be very interested in the processes

$$\{X_t\} := \left\{ \lim_{\varepsilon \downarrow 0} X_{t-\varepsilon} \right\}$$

$$\Delta X_t := \{X_t - X_{t-}\}$$

where we understand $\mathbb{P} - a.s.$ that $X_0 = X_0$ and thus $\Delta X_0 = 0$. Of great utility is that fact that if $\{X_t\}$ is a càdlàg adapted process then $\{X_{t-}\}$ is predictable and $\{\Delta X_t\}$ is optional. If $\{X_t\}$ is a càglàd adapted process then $\{\Delta X_t\}$ is also predictable.

In what follows, we have need of several processes. Specifically, we use a Markov chain to model regime switches, and an associated Poisson process to track the number of switches between particular regime pairs. Brownian motion is used to model the diffusion in the state variable, while a Poisson random measure (that is, another Poisson process) is used to model jumps of a continuum’s range of sizes.

Before continuing, we make one remark. For each of the martingales described below, both the quadratic variation process and the Doob-Meyer Decomposition compensator process are well defined. We denote the value at a particular $t$ of the former by $[\cdot]_t$ and the latter $\langle \cdot \rangle_t$. The
quadratic variation process is used, for example, when applying the Itô Formula; the Doob-Meyer Decomposition compensator process is used when computing the Itô Isometry, and thus determines the class of acceptable integrands under the theory of integration used in this dissertation.

Let \( I = \{1, 2, \ldots, M\} \). By \( \{a_t\} \) we mean an \( I \)-valued continuous-time stationary Markov chain with infinitesimal generator \( G \in \mathbb{R}^{M \times M} \) and \( \mathbb{P} - a.s. \) fixed starting state. By

\[
\mathcal{N}^{ij}_t := \sum_{0 \leq s \leq t} 1_{\{a_s = i\}} 1_{\{a_s = j\}}
\]

\[
\mathcal{N}^{ij}_t := \{\mathcal{N}^{ij}_t - G_{ij} \int_0^t 1_{\{a_s = i\}} \, ds\}
\]

where \( i \in I, j \in I, \) and \( i \neq j \); we mean the Poisson process that keeps count of the number of transitions from regime \( i \) to regime \( j \) over the interval \([0, t]\) and its associated compensated Poisson process, respectively. We remark that \( \{\mathcal{N}^{ij}_t\} \) is a centered, purely discontinuous, square-integrable martingale with

\[
[\mathcal{N}^{ij}_t]_t = \mathcal{N}^{ij}_t
\]

\[
\langle \mathcal{N}^{ij}_t \rangle_t = G^{ij} \int_0^t 1_{\{a_s = i\}} \, ds
\]

In what follows we will especially interested in counting the number of transitions into a particular regime. Let

\[
\{\Upsilon^{m}_t\} := \left\{ \sum_{i=1, i \neq m}^{M} \mathcal{N}^{im}_t \right\}
\]

\[
\{\mathcal{N}^{m}_t\} := \left\{ \sum_{i=1, i \neq m}^{M} \mathcal{N}^{im}_t \right\}
\]

denote the Poisson process that keeps count of the number of transitions into regime \( m \) from any other regime over the interval \([0, t]\) and its associated compensated Poisson process, respectively. We shall write \( \{d\mathcal{N}^{m}_t\} \) for the differential form of \( \{\mathcal{N}^{m}_t\} \) and note

\[
\{d\mathcal{N}^{m}_t\} = \{d\Upsilon^{m}_t - G^{a_t, m} 1_{\{a_t \neq m\}} \, dt\}
\]

By \( \{x_t\} \) we mean an \( \mathbb{R}^N \)-valued process, which we call the state process. We call the compact set \( U \subset \mathbb{R}^K \) the ordinary-control set, its elements \( \nu \) ordinary controls, and any càglàd adapted, hence predictable, \( U \)-valued process \( \{v_t\} \) an ordinary-control process. We call the set \( \mathcal{P}(U) \) of all measures (of total measure 1) on \( (U, \mathcal{B}(U)) \) the relaxed-control set, its elements \( \nu \) relaxed controls, and any
càglàd adapted, hence predictable, $\mathcal{P}(U)$-valued process $\{v_t\}$ a relaxed-control process. We denote by $\mathcal{O}$ and $\mathcal{R}$ the set of all ordinary- and relaxed-control processes, and by $\{x_t^u\}$ and $\{x_t^\nu\}$ the state process controlled by the ordinary- and relaxed-control processes $\{v_t\}$ and $\{\nu_t\}$, respectively. If for a given ordinary-control process $\{v_t\}$ there exist Borel measurable $U$-valued functions $v(\cdot, \cdot, m)$ defined on $[0, T] \times \mathbb{R}^N$ such that $P-a.s.$ holds for all $t$, then we call that ordinary-control process a Markov ordinary-control process. Likewise, if for a given relaxed-control process $\{\nu_t\}$ there exist Borel measurable $\mathcal{P}(U)$-valued functions $\nu(\cdot, \cdot, m)$ defined on $[0, T] \times \mathbb{R}^N$ such that $P-a.s.$ holds for all $t$, then we call that relaxed-control process a Markov relaxed-control process.

By $\mathcal{B}_d t$ we mean an $\mathbb{R}$-valued standard Brownian motion. We remark that $\mathcal{B}_d t$ is a centered, continuous, square-integrable martingale with

$$[\mathcal{B}_d t]_t = \langle \mathcal{B}_d t \rangle_t = t$$

We write $\{d\mathcal{B}_d t\}$ for the differential form of $\{\mathcal{B}_d t\}$.

Let $E \subset \mathbb{R}$ be a bounded open set and $E_0 := E \setminus \{0\}$. By $N^l(\cdot, \cdot)$ we mean a Poisson random measure on $([0, T] \times E_0, \mathcal{B}([0, T] \times E_0))$, so that for any $A \in \mathcal{B}(E_0)$ such that the closure of $A$ is bounded away from the origin, $\{N^l(t, A)\}$ is a Poisson process with intensity $\lambda^l(A) = \mathbb{E}[N^l(1, A)]$; this process counts the number of jumps of signed magnitude $A$ that have occurred over the interval $[0, t]$ in the $l^{th}$ source of jump noise. In what follows, we shall be especially interested in its associated compensated Poisson process

$$\{\tilde{N}^l(t, A)\} := \{N^l(t, A) - \lambda^l(A)t\}$$

We remark that $\{\tilde{N}^l(t, A)\}$ is also a centered, purely discontinuous, square-integrable martingale with

$$[\tilde{N}^l(\cdot, A)]_t = N^l(t, A)$$

$$\langle \tilde{N}^l(\cdot, A) \rangle_t = \lambda^l(A)t$$

We write $\{\tilde{N}^l(dt, d\zeta)\}$ for the differential form of $\{\tilde{N}^l(t, A)\}$ and note

$$\{\tilde{N}^l(dt, d\zeta)\} = \{N^l(dt, d\zeta) - \lambda^l(d\zeta)dt\}$$
We are now in a better position to describe the filtration \( \{ \mathcal{F}_t \} \). We assume the processes \( \{ B^d_t \} \), \( \{ \tilde{N}^i(t, \cdot) \} \), and \( \{ \tilde{T}^m_t \} \) are mutually independent, \( \{ \mathcal{F}_t \} \)-adapted, and have increments at each \( t \) independent of \( \mathcal{F}_t \). With this filtration in mind, we mention several spaces of \( \mathbb{R} \)-valued processes. By \( L^2_{opt}([0, T] \times \Omega) \), or \( L^2_{pred}([0, T] \times \Omega) \), we mean the space of optional, or predictable, processes \( \{ X_t \} \) such that \( \mathbb{E} \int_0^T |X_t|^2 \, dt < \infty \); by \( L^2_{pred}([0, T] \times \tilde{\Omega} \times \Omega) \) we mean the space of predictable processes \( \{ X(t, \cdot) \} \) such that \( \mathbb{E} \int_0^T |X(t, \zeta)|^2 \lambda(d\zeta) \, dt < \infty \); and by \( L^2_{pred}([0, T] \times \tilde{\Omega}) \) we mean the space of predictable processes \( \{ X_t \} \) such that \( \mathbb{E} \int_0^T |X_t|^2 G^m 1_{i \neq m} \, dt < \infty \). By \( L^2(\Omega, \mathcal{F}) \) we mean the space of random variables \( X \) on \( \mathcal{F} \) such that \( \mathbb{E}|X|^2 < \infty \). When we write that some vector- or matrix-valued process or variable belongs to one of these spaces, we mean component-wise inclusion.

The following functions are used in what follows to define the state equations. Specifically, the following are the coefficient functions in the state equation.

\[
\begin{align*}
b(t, x, v, m) &\in C^{0,1,1}([0, T] \times \mathbb{R}^N \times U; \mathbb{R}^N) \\
\sigma(t, x, v, m) &\in C^{0,1,1}([0, T] \times \mathbb{R}^N \times U; \mathbb{R}^{N \times D}) \\
\gamma(t, x, v, m) &\in C^{0,1,0}([0, T] \times \mathbb{R}^N \times U \times E_0; \mathbb{R}^{N \times L}) \\
\eta(t, x, v, m) &\in C^{0,1,1}([0, T] \times \mathbb{R}^N \times U; \mathbb{R}^{N \times M})
\end{align*}
\]

The following functions are used to define the cost functionals.

\[
\begin{align*}
h(t, x, v, m) &\in C^{0,1,1}([0, T] \times \mathbb{R}^N \times U; \mathbb{R}) \\
g(t, v, m) &\in C^1(\mathbb{R}^N; \mathbb{R})
\end{align*}
\]

It is conventional, and convenient, to impose the following conditions on the coefficients. The linear growth condition is given by

\[
\begin{align*}
|b(t, x, v, m)|^2 &\leq C \left( 1 + |x|^2 + |v|^2 \right) \\
|\sigma^{sd}(t, x, v, m)|^2 &\leq C \left( 1 + |x|^2 + |v|^2 \right) \\
\int_{E_0} |\gamma^{sl}(t, x, v, m, \zeta)|^2 \lambda(d\zeta) &\leq C \left( 1 + |x|^2 + |v|^2 \right) \\
|\eta^{sm}(t, x, v, m)|^2 &\leq C \left( 1 + |x|^2 + |v|^2 \right)
\end{align*}
\]  

and the Lipschitz condition by

\[
\begin{align*}
|b(t, x, u, m) - b(t, y, v, m)|^2 &\leq C \left( |x - y|^2 + |u - v|^2 \right) \\
|\sigma^{sd}(t, x, u, m) - \sigma^{sd}(t, y, v, m)|^2 &\leq C \left( |x - y|^2 + |u - v|^2 \right)
\end{align*}
\]  

(1.7)

(1.8)
\[ \int_{E_0} \left| \gamma^*(t, x, u, m, \zeta) - \gamma^*(t, y, v, m, \zeta) \right|^2 \lambda^l(d\zeta) \leq C \left( |x - y|^2 + |u - v|^2 \right) \]

\[ \left| \eta^m(t, x, u, i) - \eta^m(t, y, v, i) \right|^2 G^m_1 \Delta M(t, i) \leq C \left( |x - y|^2 + |u - v|^2 \right) \]

\[ |h(t, x, u, m) - h(t, y, v, m)|^2 \leq C \left( |x - y|^2 + |u - v|^2 \right) \]

\[ |g(x, m) - g(y, m)|^2 \leq C \left( |x - y|^2 \right) \]

We will also make use in the sequel of the random vector \( \zeta_0 \in L^2(\Omega, \mathcal{F}_0) \) to denote the initial value of the state process.

### 1.3 Mathematical Formulation

The optimal-control problem is essentially a constrained optimization problem, with the state equation the constraint. We consider the problem formulated for both ordinary and relaxed controls. We remark that in what follows we assume that \( \{x^u_t\} \) and \( \{x^v_t\} \) make sense, that is, that (1.9) and (1.12) have \( \lambda \) solutions; in fact, we will show in section 1.4 that there exists a unique càdlàg adapted solution for each of the equations. We describe the ordinary-control problem here and the relaxed-control problem below.

For any ordinary-control process \( \{v_t\} \) the state process \( \{x^v_t\} \) satisfies \( \mathbb{P} - a.s. \)

\[ dx^v_t = b(t, x^v_t, v_t, \alpha_{t-})dt + \sum_{d=1}^D \alpha^d(t, x^v_t, v_t, \alpha_{t-})dB^d_t \]

\[ + \sum_{l=1}^L \int_{E_0} \gamma^l(t, x^v_t, v_t, \alpha_{t-}, \xi) N^l(dt, d\xi) \]

\[ + \sum_{m=1}^M \eta^m(t, x^v_t, v_t, \alpha_{t-})d\tilde{Y}^m_t \]

\[ x^v_0 = \zeta_0 \]

The objective is defined as

\[ f(\{v_t\}) := \mathbb{E} \left[ \int_0^T h(t, x^v_t, v_t, \alpha_{t-})dt + g(x^v_T, \alpha_T) \right] \] (1.10)

Without loss of generality, we take optimal to mean minimal, and as a result call (1.10) the cost functional. To be clear, if there exists an ordinary-control process \( \{u_t\} \) such that

\[ f(\{u_t\}) = \inf_{\{v_t\} \in \mathcal{O}} f(\{v_t\}) \] (1.11)
then and only then do we call \{u_t\} optimal.

For any relaxed-control process \{v_t\} the state process \{x^v_t\} satisfies \(P-a.s\.

\begin{align*}
\mathrm{d}x^v_t &= \int_U b(t, x^v_t, a, \alpha \tau) v_t(\mathrm{d}a) \mathrm{d}t + \sum_{d=1}^D \int_U \sigma^{sd}(t, x^v_t, a, \alpha \tau) v_t(\mathrm{d}a) \mathrm{d}B^d_t \\
&+ \sum_{l=1}^L \int_{E_0} \int_U \gamma^{sl}(t, x^v_t, a, \alpha \tau, \zeta) v_t(\mathrm{d}a) \tilde{N}^l(\mathrm{d}t, \mathrm{d}\zeta) \\
&+ \sum_{m=1}^M \int_U \eta^{sm}(t, x^v_t, a, \alpha \tau) v_t(\mathrm{d}a) \mathrm{d}\tilde{\Upsilon}^m_t
\end{align*}
(1.12)

\[x^v_0 = \xi_0\]

The objective is defined as

\[J(\{v_t\}) := \mathbb{E}\left[\int_0^T \int_U h(t, x^v_t, a, \alpha \tau) v_t(\mathrm{d}a) \mathrm{d}t + g(x^y_T, \alpha_T)\right]\]
(1.13)

Without loss of generality, we take optimal to mean minimal, and as a result call (1.13) the cost functional. To be clear, if there exists a relaxed-control process \{\mu_t\} such that

\[J(\{\mu_t\}) = \inf_{\{v_t\} \in \mathbb{R}} J(\{v_t\})
\]
(1.14)
then and only then do we call \{\mu_t\} optimal.

We remark that (1.11) and (1.14) are generalizations of the controlled-diffusion problem. If \{\alpha_t\} = \{1\}, \gamma(\cdot) = 0, and \eta(\cdot) = 0, then the ordinary-control problem is the controlled-diffusion problem. By allowing \{\alpha_t\} to take multiple values we introduce regime switching, that is, we introduce random changes to the coefficients (1.5) and (1.6). If additionally \eta_{nm}(\cdot) is nontrivial then regime switches into regime \(m\) produce a jump in the \(n^{th}\) component of the state process. Similarly if \(\gamma_{nl}(\cdot)\) is nontrivial then the \(l^{th}\) source of jump noise will produce a jump in the \(n^{th}\) component of the state process. In both of these cases it is important to note that the integrators are centered, purely discontinuous, square-integrable martingales and so compensate for these jumps in a particularly tractable manner.

### 1.4 The Ordinary-State Equation

We begin with a constructive proof for the existence of a strong solution to the ordinary-state equation (1.9). A strong solution exists when for any prescribed filtered probability space \((\Omega, \mathcal{F}, \{\mathcal{F}_t\}, \mathbb{P})\); \{\mathcal{F}_t\}-adapted \{B^d_t\}, \{\tilde{N}^l(t, \cdot)\}, and \{\tilde{\Upsilon}^m_t\}; and random variable initial value \(\xi_0 \in L^2(\Omega, \mathcal{F}_0)\); there
exists a càdlàg $L^2_{o.p.}([0,T] \times \Omega)$ process $\{X_t\}$ such that $\mathbb{P} - a.s. \{X_t\}$ satisfies (1.9). By contrast, a weak solution given $\mathcal{Q}$, a distribution (of total measure 1) on $\mathbb{R}^N$, exists if there are filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, \mathbb{P}); \{\mathcal{F}_t\}$-adapted $\{B^d_t\}, \{\tilde{N}^l(t, \cdot)\}$, and $\{\Upsilon^m_t\}$; and càdlàg $L^2_{o.p.}([0,T] \times \Omega)$ process $\{X_t\}$ such that $\mathbb{P} - a.s. \{X_t\}$ satisfies (1.9) and $\mathbb{P} \circ X_0^{-1} = \mathcal{Q}$.

**Theorem 1.** There exists a càdlàg $L^2_{o.p.}([0,T] \times \Omega)$ solution to the state equation (1.9).

**Proof.** This proof is similar to the Picard Successive Approximation, which proves under similar hypotheses the existence of a solution to an ordinary (deterministic) differential equation. We first define an approximating sequence of càdlàg $L^2_{o.p.}([0,T] \times \Omega)$ processes. We next compute second-moment estimates of the increments of the approximating sequence. Thirdly we show that $\mathbb{P} - a.s.$ the approximating sequence converges uniformly on $[0,T]$ to a càdlàg $L^2_{o.p.}([0,T] \times \Omega)$ process. Finally we confirm that this limiting process satisfies the state equation (1.9).

(Step 1.) Let $j = 1, 2, 3, \ldots; s \in [0, T];$ and $\{X^j_s\}$ be a $j$-indexed sequence of $s$-indexed processes defined by $\mathbb{P} - a.s.$

$$X^j_s := \xi_0 + \int_0^s b(t, X^j_{t-}, v_t, \alpha_{t-})dt + \sum_{d=1}^D \int_0^s \sigma^d(t, X^j_{t-}, v_t, \alpha_{t-})dB^d_t$$
$$+ \sum_{l=1}^L \int_0^s \int_{E_0} \gamma^l(t, X^j_{t-}, v_t, \alpha_{t-}, \zeta) \tilde{N}^l(t, d\zeta)$$
$$+ \sum_{m=1}^M \int_0^s \eta^m(t, X^j_{t-}, v_t, \alpha_{t-})d\Upsilon^m_t$$

(1.15)

where we take $X^0_s := \xi_0$.

Consider the equation for $\{X^1_s\}$. Observe that the integrators of the third, fourth, and fifth terms on the right-hand side are càdlàg martingales, and that each integrand is predictable to conclude that the integrals are well defined and that $\{X^1_s\}$ is a càdlàg adapted, hence optional, process. Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, and take expectation on both sides to get

$$\mathbb{E} |X^1_s|^2 \leq \beta \mathbb{E} |\xi_0|^2 + \beta \mathbb{E} \left| \int_0^s b(t, X^0_{t-}, v_t, \alpha_{t-})dt \right|^2$$
$$+ \beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \sigma^d(t, X^0_{t-}, v_t, \alpha_{t-})dB^d_t \right|^2$$
$$+ \beta \mathbb{E} \sum_{l=1}^L \left| \int_0^s \int_{E_0} \gamma^l(t, X^0_{t-}, v_t, \alpha_{t-}, \zeta) \tilde{N}^l(t, d\zeta) \right|^2$$
$$+ \beta \mathbb{E} \sum_{m=1}^M \left| \int_0^s \eta^m(t, X^0_{t-}, v_t, \alpha_{t-})d\Upsilon^m_t \right|^2$$
where $\beta = 2 + D + L + M$. Apply the Cauchy-Schwarz Inequality for Integrals to the second term and the Itô Isometry to the third, fourth, and fifth terms on the right-hand side to obtain

$$
\mathbb{E}\left|X_s^1\right|^2 \leq \beta \mathbb{E}|\xi_0|^2 + \beta T \mathbb{E} \int_0^s \left| b(t, X_t^0, v_t, \alpha_{t-}) \right|^2 dt
$$

$$
+ \beta \mathbb{E} \sum_{d=1}^D \int_0^s \left| \alpha^{s,d}(t, X_t^0, v_t, \alpha_{t-}) \right|^2 dt
$$

$$
+ \beta \mathbb{E} \sum_{l=1}^L \int_0^s \int_{E_0} \left| \gamma^{s,l}(t, X_t^0, v_t, \alpha_{t-}, \zeta) \right|^2 \lambda^l(d\zeta) dt
$$

$$
+ \beta \mathbb{E} \sum_{m=1}^M \int_0^s \left| \eta^{s,m}(t, X_t^0, v_t, \alpha_{t-}) \right|^2 \mathbb{E} \left| G^{\alpha_{t-},m} 1_{\{ \alpha_{t-} \neq m \}} \right| dt
$$

Apply (1.7) and combine like terms to produce

$$
\mathbb{E}\left|X_s^1\right|^2 \leq \beta \mathbb{E}|\xi_0|^2 + \beta (T + \beta - 2) \mathbb{E} \int_0^s C \left( 1 + |X_t^0|^2 + |v_t|^2 \right) dt
$$

Recall that the control set $U$ is compact, which implies

$$
C \left( 1 + |v_t|^2 \right) \leq C (1 + C) \leq C^3
$$

and thus

$$
\mathbb{E}\left|X_s^1\right|^2 \leq \beta \mathbb{E}|\xi_0|^2 + \beta (T + \beta - 2) C^3 T + \beta (T + \beta - 2) C \int_0^s \mathbb{E}\left|X_t^0\right|^2 dt
$$

$$
\leq C \mathbb{E}|\xi_0|^2 + C^6 + C^3 \int_0^s \mathbb{E}\left|X_t^0\right|^2 dt
$$

which for convenience is written as

$$
\mathbb{E}\left|X_s^1\right|^2 \leq C^6 (1 + \mathbb{E}|\xi_0|^2) + C^6 \int_0^s \mathbb{E}\left|X_t^0\right|^2 dt
$$

(1.16)

Substitute for $X_t^0$ and compute the integral to get

$$
\mathbb{E}\left|X_s^1\right|^2 \leq C^6 (1 + \mathbb{E}|\xi_0|^2) + C^6 \mathbb{E}\left|\xi_0\right|^2 s
$$

$$
\leq C^6 (1 + \mathbb{E}|\xi_0|^2) + C^{12} (1 + \mathbb{E}|\xi_0|^2) s
$$

Thus $\{X_s^1\}$ is square integrable. As the right-hand side is uniformly bounded in $s$ we conclude that $\{X_s^1\}$ is a càdlàg $L^2_{opt}([0, T] \times \Omega)$ process.
Assume that \( \{X^j_t\} \) is a càdlàg \( L^2_{opt}([0, T] \times \Omega) \) process with

\[
E \left| X^j_s \right|^2 \leq C^6 \left( 1 + E |\xi_0|^2 \right) \sum_{\kappa=0}^j \frac{[C^6 s]^{\kappa}}{\kappa!} \quad (1.17)
\]

Observe that \( \{X^j_s\} \) is necessarily predictable and so by the argument given above for \( \{X^1_s\} \) we conclude that \( \{X^{j+1}_s\} \) is a càdlàg adapted, hence optional, process. Mimic calculation (1.16) and apply (1.17) to get

\[
E \left| X^{j+1}_s \right|^2 \leq C^6 \left( 1 + E |\xi_0|^2 \right) + C^6 \int_0^s E \left| X^j_t \right|^2 \, dt
\]

\[
\leq C^6 \left( 1 + E |\xi_0|^2 \right) + C^6 \int_0^s \sum_{\kappa=0}^j \frac{[C^6 t]^{\kappa}}{\kappa!} \, dt
\]

\[
= C^6 \left( 1 + E |\xi_0|^2 \right) + C^6 \left( 1 + E |\xi_0|^2 \right) \sum_{\kappa=0}^j \frac{[C^6(s+1)]^{\kappa+1}}{(s+1)!} \left( \frac{C^6 s^\kappa}{\kappa!} \right)
\]

\[
= C^6 \left( 1 + E |\xi_0|^2 \right) \sum_{\kappa=0}^j \frac{[C^6 s]^{\kappa}}{\kappa!}
\]

Thus \( \{X^{j+1}_s\} \) is square integrable. As the right-hand side is uniformly bounded in \( s \) we conclude that \( \{X^j_s\} \) is a càdlàg \( L^2_{opt}([0, T] \times \Omega) \) process. Apply the Principle of Mathematical Induction to deduce that, for each \( j \), \( \{X^j_s\} \) is in fact a càdlàg \( L^2_{opt}([0, T] \times \Omega) \) process.

(Step 2) Consider the equation for the increment process \( \{X^1_s - \xi_0\} \). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, and take expectation on both sides to get

\[
E \left| X^1_s - \xi_0 \right|^2 \leq (\beta - 1) E \left\| \int_0^s b(t, X^0_{t-}, v_t, \alpha_{t-}) \, dt \right\|^2
\]

\[
+ (\beta - 1) E \sum_{d=1}^D \left\| \int_0^s \sigma^d(t, X^0_{t-}, v_t, \alpha_{t-}) dB_t \right\|^2
\]

\[
+ (\beta - 1) E \sum_{l=1}^L \left\| \int_0^s \gamma^l(t, X^0_{t-}, v_t, \alpha_{t-}, \zeta) \tilde{N}_t \right\|^2 \, dt, \, d\zeta
\]

\[
+ (\beta - 1) E \sum_{m=1}^M \left\| \int_0^s \eta^m(t, X^0_{t-}, v_t, \alpha_{t-}) d\tilde{Y}^m_t \right\|^2
\]
Apply the Cauchy-Schwarz Inequality for Integrals to the first term and the Itô Isometry to the second, third, and fourth terms on the right-hand side; and then use (1.7) and combine like terms to obtain

\[
\mathbb{E}\left|X_s^1 - \xi_0\right|^2 \leq (\beta - 1)(T + \beta - 2)\mathbb{E}\int_0^s C\left(1 + \left|X_t^0\right|^2 + |\nu_t|^2\right)dt \\
\leq (\beta - 1)(T + \beta - 2)C^3\int_0^s \left(1 + \mathbb{E}\left|X_t^0\right|^2\right)dt \\
\leq C^5\left(1 + \mathbb{E}\left|\xi_0\right|^2\right)s
\]

(1.18)

Next consider the equation for the increment process \(\{X_s^2 - X_s^1\}\). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, and take expectation on both sides to get

\[
\mathbb{E}\left|X_s^2 - X_s^1\right|^2 \leq (\beta - 1)\mathbb{E}\left|\int_0^s b(t, X_{t-}^1, v_t, \alpha_{t-}) - b(t, X_{t-}^0, v_t, \alpha_{t-})dt\right|^2 \\
+ (\beta - 1)\mathbb{E}\sum_{d=1}^D \left|\int_0^s \sigma^{sd}(t, X_{t-}^1, v_t, \alpha_{t-}) - \sigma^{sd}(t, X_{t-}^0, v_t, \alpha_{t-})dB_t^d\right|^2 \\
+ (\beta - 1)\mathbb{E}\sum_{l=1}^L \left|\int_0^s \int_{E_0} \gamma^{sl}(t, X_{t-}^1, v_t, \alpha_{t-}, \zeta) - \gamma^{sl}(t, X_{t-}^0, v_t, \alpha_{t-}, \zeta)\tilde{N}^l(dt, d\zeta)\right|^2 \\
+ (\beta - 1)\mathbb{E}\sum_{m=1}^M \left|\int_0^s \eta^{sm}(t, X_{t-}^1, v_t, \alpha_{t-}) - \eta^{sm}(t, X_{t-}^0, v_t, \alpha_{t-})d\tilde{Y}_t^m\right|^2
\]

Apply the Cauchy-Schwarz Inequality for Integrals to the first term and the Itô Isometry to the second, third, and fourth terms on the right-hand side; and then use (1.8) and combine like terms to produce

\[
\mathbb{E}\left|X_s^2 - X_s^1\right|^2 \leq (\beta - 1)(T + \beta - 2)\mathbb{E}\int_0^s C\left|X_t^1 - X_t^0\right|^2 dt
\]
which for convenience is written as

\[ \mathbb{E} \left| X_s^2 - X_s^1 \right|^2 \leq C^5 \int_0^s \mathbb{E} \left| X_t^1 - X_t^0 \right|^2 \, dt \]  

(1.19)

Substitute (1.18) and compute the integral to get

\[ \mathbb{E} \left| X_s^2 - X_s^1 \right|^2 \leq C^5 \int_0^s (1 + \mathbb{E} |\xi_0|^2) t \, dt \]

\[ \leq (1 + \mathbb{E} |\xi_0|^2) \frac{[C^5 s]^2}{2!} \]

Assume that

\[ \mathbb{E} \left| X_s^{j+1} - X_s^j \right|^2 \leq (1 + \mathbb{E} |\xi_0|^2) \frac{[C^5 s]^{j+1}}{(j+1)!} \]  

(1.20)

Mimic calculation (1.19) and apply (1.20) to produce

\[ \mathbb{E} \left| X_s^{j+2} - X_s^{j+1} \right|^2 \leq C^5 \int_0^s \mathbb{E} \left| X_t^{j+1} - X_t^j \right|^2 \, dt \]

\[ \leq C^5 \int_0^s (1 + \mathbb{E} |\xi_0|^2) \frac{[C^5 t]^{j+1}}{(j+1)!} \, dt \]

\[ \leq C^5 (1 + \mathbb{E} |\xi_0|^2) \frac{[C^5 (j+1)]^{j+2}}{(j+1)!} \left( \frac{s^{j+2}}{j+2} \right) \]

\[ \leq (1 + \mathbb{E} |\xi_0|^2) \frac{[C^5 s]^{j+2}}{(j+2)!} \]

Apply the Principle of Mathematical Induction to deduce that, for each \( j \), (1.20) does in fact hold.

We should like to improve upon (1.20). Consider the equation for the increment process \( \{ X_s^{j+1} - X_s^j \} \). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, take the supremum and then the expectation on both sides to get

\[ 0 \leq - \left( \mathbb{E} \sup_{0 \leq s \leq T} \left| X_s^{j+1} - X_s^j \right|^2 \right) \]

\[ + (\beta - 1) \mathbb{E} \sup_{0 \leq s \leq T} \left( \int_0^s b(t, X_t^{j-1}, \nu_t, \alpha_{t-}) - b(t, X_t^{j-1}, \nu_t, \alpha_{t-}) \, dt \right)^2 \]

\[ + (\beta - 1) \mathbb{E} \sum_{d=1}^D \int_0^s \sigma^{*d}(t, X_t^{j-1}, \nu_t, \alpha_{t-}) \]
Apply the Cauchy-Schwarz Inequality for Integrals to the second term, and the Doob Martingale Inequality and the Itô Isometry to the third, fourth, and fifth terms on the right-hand side to produce

\[
0 \leq -\left\{ E \sup_{0 \leq s \leq T} |X_s^{j+1} - X_s^j|^2 \right\} \\
+ (\beta - 1) T \sum_{d=1}^D \int_0^T \left| \sigma^d(t, X_t^j, v_t, \alpha_t, \zeta) \right|^2 dt \\
+ 4(\beta - 1) \sum_{m=1}^M \int_0^T \left| \eta^m(t, X_t^j, v_t, \alpha_t) \right|^2 dt \\
+ 4(\beta - 1) \sum_{m=1}^M \int_0^T \left| \gamma^m(t, X_t^j, v_t, \alpha_t, \zeta) \right|^2 \lambda(\zeta) d\zeta
\]

Use (1.8), combine like terms, and substitute (1.20) to get

\[
0 \leq -\left\{ E \sup_{0 \leq s \leq T} |X_s^{j+1} - X_s^j|^2 \right\} \\
+ 4(\beta - 1)(T + \beta - 2) \int_0^T C |X_t^j - X_t^{j-1}|^2 dt
\]
\[
\leq -\left( \mathbb{E} \sup_{0 \leq s \leq T} |X_{s}^{j+1} - X_{s}^{j}|^{2} \right) + 4 C^{5} \int_{0}^{T} \mathbb{E} \left| X_{t}^{j} - X_{t}^{j-1} \right|^{2} dt \\
\leq -\left( \mathbb{E} \sup_{0 \leq s \leq T} |X_{s}^{j+1} - X_{s}^{j}|^{2} \right) + 4 C^{5} \int_{0}^{T} \left( 1 + \mathbb{E} |\xi_{0}|^{2} \right) \frac{[C^{5}T]^{j+1}}{j!} dt
\]

Compute the integral to conclude that

\[
\mathbb{E} \sup_{0 \leq s \leq T} |X_{s}^{j+1} - X_{s}^{j}|^{2} \leq 4 \left( 1 + \mathbb{E} |\xi_{0}|^{2} \right) \frac{[C^{5}T]^{j+1}}{(j+1)!}
\]

(1.21)

holds for each \( j \).

(Step 3.) Consider the \( j \)-indexed sequence of events

\[
\left\{ \sup_{0 \leq s \leq T} \left| X_{s}^{j+1} - X_{s}^{j} \right| \geq \frac{1}{2j} \right\}
\]

Use the Chebyshev-Markov Inequality and (1.21) to obtain

\[
\mathbb{P} \left\{ \sup_{0 \leq s \leq T} \left| X_{s}^{j+1} - X_{s}^{j} \right| \geq \frac{1}{2j} \right\} \leq \left( \frac{1}{2j} \right)^{2} \mathbb{E} \sup_{0 \leq s \leq T} \left| X_{s}^{j+1} - X_{s}^{j} \right|^{2} \\
\leq 4^{j} \left[ 4 \left( 1 + \mathbb{E} |\xi_{0}|^{2} \right) \frac{[C^{5}T]^{j+1}}{(j+1)!} \right] \\
= \left( 1 + \mathbb{E} |\xi_{0}|^{2} \right) \frac{[4C^{5}T]^{j+1}}{(j+1)!}
\]

Observe that

\[
\sum_{\kappa=1}^{\infty} \left( 1 + \mathbb{E} |\xi_{0}|^{2} \right) \frac{[4C^{5}T]^{\kappa+1}}{(\kappa+1)!} \leq \left( 1 + \mathbb{E} |\xi_{0}|^{2} \right) e^{4C^{5}T}
\]

and apply the Borel-Cantelli Lemma to deduce

\[
\mathbb{P} \left\{ \sup_{0 \leq s \leq T} \left| X_{s}^{j+1} - X_{s}^{j} \right| \geq \frac{1}{2j} \text{ i.o.} \right\} = 0
\]

Observe also that \( \sum_{j=1}^{\infty} \frac{1}{2j} < \infty \) to reckon that \( \mathbb{P} - a.s. \) the series

\[
X_{s}^{1} + \sum_{\kappa=1}^{\infty} \left( X_{s}^{\kappa+1} - X_{s}^{\kappa} \right)
\]

converges absolutely and uniformly on \([0, T]\). Denote this limit by \( \{x_{s}^{j}\} \). It is clear that this process
is a càdlàg adapted, hence optional, process, and that \( P - a.s. \)

\[
x_s^v = X_1^s + \lim_{j \to \infty} \sum_{

\kappa=1}^{j-1} (X_{

\kappa+1}^s - X_s^\kappa)
\]

\[
= \lim_{j \to \infty} X_j^s
\]

Square both sides, take expectation, apply the Fatou Lemma, and use (1.17) to get

\[
E \left| x_s^v \right|^2 = E \left| \lim_{j \to \infty} X_j^s \right|^2
\]

\[
\leq \lim_{j \to \infty} E \left| X_j^s \right|^2
\]

\[
\leq \lim_{j \to \infty} C^6 \left( 1 + E |\xi_0|^2 \right) \sum_{

\kappa=0}^{j} \frac{[C^6 \kappa]^K}{\kappa!}
\]

\[
= C^6 \left( 1 + E |\xi_0|^2 \right) e^{C^6

\]

Thus \( \{ x_s^v \} \) is square integrable. As the right-hand side is uniformly bounded in \( s \) we conclude that \( \{ x_s^v \} \) is a càdlàg \( L^2_{opt}([0, T] \times \Omega) \) process.

(Step 4.) Use (1.8) together with the results of the previous step to deduce \( P - a.s. \)

\[
\lim_{j \to \infty} b(\cdot, X_j^l, \cdot, \cdot, \cdot) = b(\cdot, x_s^v, \cdot, \cdot, \cdot)
\]

\[
\lim_{j \to \infty} \sigma(\cdot, X_j^l, \cdot, \cdot, \cdot) = \sigma(\cdot, x_s^v, \cdot, \cdot, \cdot)
\]

\[
\lim_{j \to \infty} \gamma(\cdot, X_j^l, \cdot, \cdot, \cdot, \cdot) = \gamma(\cdot, x_s^v, \cdot, \cdot, \cdot, \cdot)
\]

\[
\lim_{j \to \infty} \eta(\cdot, X_j^l, \cdot, \cdot, \cdot, \cdot) = \eta(\cdot, x_s^v, \cdot, \cdot, \cdot, \cdot)
\]

uniformly on \([0, T]\) and thus \( P - a.s. \)

\[
\lim_{j \to \infty} \int_0^T \left| \sigma^{sl}(t, X_i^l, v_t, \alpha_{t-}, \xi) - \sigma^{sl}(t, x_s^v, v_t, \alpha_{t-}) \right|^2 dt = 0
\]

\[
\lim_{j \to \infty} \int_0^T \left| \gamma^{sl}(t, X_i^l, v_t, \alpha_{t-}, \zeta) - \gamma^{sl}(t, x_s^v, v_t, \alpha_{t-}, \zeta) \right|^2 \lambda^l(d\zeta) dt = 0
\]

\[
\lim_{j \to \infty} \int_0^T \left| \eta^{sl}(t, X_i^l, v_t, \alpha_{t-}) - \eta^{sl}(t, x_s^v, v_t, \alpha_{t-}) \right|^2 \lambda_t^l dt = 0
\]
This implies the convergence in \( \mathbb{P} \)-measure of each of

\[
\sup_{0 \leq s \leq T} \left| \int_0^s \sigma^{sd}(t, X^j_{t-}, v_t, \alpha_{t-}) - \sigma^{sd}(t, X^v_{t-}, v_t, \alpha_{t-}) dB^d_t \right|
\]

\[
\sup_{0 \leq s \leq T} \left| \int_0^s \gamma^s(t, X^j_{t-}, v_t, \alpha_{t-}, \zeta) - \gamma^s(t, X^v_{t-}, v_t, \alpha_{t-}, \zeta) \tilde{N}^j_l(\text{d}t, \text{d}\zeta) \right|
\]

\[
\sup_{0 \leq s \leq T} \left| \int_0^s \eta^s(t, X^j_{t-}, v_t, \alpha_{t-}) - \eta^s(t, X^v_{t-}, v_t, \alpha_{t-}) d\tilde{\Gamma}^m_t \right|
\]

to 0 as \( j \) increases to infinity. Thus there exists a subsequence \( \{j_k\}_{k=1}^\infty \), with \( j_k < j_{k+1} \) for each \( \kappa \), such that \( \mathbb{P} - a.s. \)

\[
\lim_{\kappa \to \infty} \sup_{0 \leq s \leq T} \left| \int_0^s \sigma^{sd}(t, X^{j_k}_{t-}, v_t, \alpha_{t-}) - \sigma^{sd}(t, X^v_{t-}, v_t, \alpha_{t-}) dB^d_t \right| = 0
\]

\[
\lim_{\kappa \to \infty} \sup_{0 \leq s \leq T} \left| \int_0^s \gamma^s(t, X^{j_k}_{t-}, v_t, \alpha_{t-}, \zeta) - \gamma^s(t, X^v_{t-}, v_t, \alpha_{t-}, \zeta) \tilde{N}^j_l(\text{d}t, \text{d}\zeta) \right| = 0
\]

\[
\lim_{\kappa \to \infty} \sup_{0 \leq s \leq T} \left| \int_0^s \eta^s(t, X^{j_k}_{t-}, v_t, \alpha_{t-}) - \eta^s(t, X^v_{t-}, v_t, \alpha_{t-}) d\tilde{\Gamma}^m_t \right| = 0
\]

Apply (1.15) and this estimate to obtain \( \mathbb{P} - a.s. \)

\[
x^v_s = \lim_{\kappa \to \infty} X^{j_k+1}_s
\]

\[
= \xi_0 + \lim_{\kappa \to \infty} \int_0^s b(t, X^{j_k}_{t-}, v_t, \alpha_{t-}) \text{d}t + \lim_{\kappa \to \infty} \sum_{d=1}^D \int_0^s \sigma^{sd}(t, X^{j_k}_{t-}, v_t, \alpha_{t-}) dB^d_t
\]

\[
+ \lim_{\kappa \to \infty} \sum_{l=1}^L \int_0^s \gamma^l(t, X^{j_k}_{t-}, v_t, \alpha_{t-}, \zeta) \tilde{N}^j_l(\text{d}t, \text{d}\zeta)
\]

\[
+ \lim_{\kappa \to \infty} \sum_{m=1}^M \int_0^s \eta^m(t, X^{j_k}_{t-}, v_t, \alpha_{t-}) d\tilde{\Gamma}^m_t
\]

\[
= \xi_0 + \int_0^s b(t, x^v_{s-}, v_t, \alpha_{t-}) \text{d}t + \sum_{d=1}^D \int_0^s \sigma^{sd}(t, x^v_{s-}, v_t, \alpha_{t-}) dB^d_t
\]

\[
+ \sum_{l=1}^L \int_0^s \gamma^l(t, x^v_{s-}, v_t, \alpha_{t-}, \zeta) \tilde{N}^j_l(\text{d}t, \text{d}\zeta)
\]

\[
+ \sum_{m=1}^M \int_0^s \eta^m(t, x^v_{s-}, v_t, \alpha_{t-}) d\tilde{\Gamma}^m_t
\]

Thus \( \{x^v_s\} \) satisfies the state equation (1.9) and the proof is complete. \( \square \)

Two processes are indistinguishable if \( \mathbb{P} - a.s. \) the paths are identical for all \( t \in [0, T] \). Thus the
The following result is sometimes known as path-wise uniqueness of the strong solution.

**Theorem 2.** Any pair of càdlàg \( L_{opt}^2([0, T] \times \Omega) \) solutions to the state equation (1.9) are indistinguishable.

**Proof.** Let \( s \in [0, T] \); \( \{x_s\} \) and \( \{y_s\} \) any pair of càdlàg \( L_{opt}^2([0, T] \times \Omega) \) solutions to the state equation (1.9); and \( X_s := x_s - y_s \), so that we have \( \mathbb{P} \)-a.s.

\[
X_s = \int_0^s b(t, x_{t-}, v_t, \alpha_{t-}) - b(t, y_{t-}, v_t, \alpha_{t-}) \, dt \\
+ \sum_{d=1}^D \int_0^s \sigma^{sd}(t, x_{t-}, v_t, \alpha_{t-}) - \sigma^{sd}(t, y_{t-}, v_t, \alpha_{t-}) \, dB^d_t \\
+ \sum_{l=1}^L \int_0^s \gamma^{sl}(t, x_{t-}, v_t, \alpha_{t-}) - \gamma^{sl}(t, y_{t-}, v_t, \alpha_{t-}) \, \tilde{N}^l(t, d\zeta) \\
+ \sum_{m=1}^M \int_0^s \eta^{m}(t, x_{t-}, v_t, \alpha_{t-}) - \eta^{m}(t, y_{t-}, v_t, \alpha_{t-}) \, d\tilde{Y}^m_t
\]

Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, and take expectation on both sides to get

\[
\mathbb{E} |X_s|^2 \leq \beta \mathbb{E} \left[ \left| \int_0^s b(t, x_{t-}, v_t, \alpha_{t-}) - b(t, y_{t-}, v_t, \alpha_{t-}) \, dt \right|^2 \right] \\
+ \beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \sigma^{sd}(t, x_{t-}, v_t, \alpha_{t-}) - \sigma^{sd}(t, y_{t-}, v_t, \alpha_{t-}) \, dB^d_t \right|^2 \\
+ \beta \mathbb{E} \sum_{l=1}^L \left| \int_0^s \gamma^{sl}(t, x_{t-}, v_t, \alpha_{t-}) - \gamma^{sl}(t, y_{t-}, v_t, \alpha_{t-}) \, \tilde{N}^l(t, d\zeta) \right|^2 \\
+ \beta \mathbb{E} \sum_{m=1}^M \left| \int_0^s \eta^{m}(t, x_{t-}, v_t, \alpha_{t-}) - \eta^{m}(t, y_{t-}, v_t, \alpha_{t-}) \, d\tilde{Y}^m_t \right|^2
\]

where \( \beta = 1 + D + L + M \). Apply the Cauchy-Schwarz Inequality for Integrals to the first term and the Itô Isometry to the second, third, and fourth terms on the right-hand side to obtain

\[
\mathbb{E} |X_s|^2 \leq \beta T \mathbb{E} \int_0^s \left| b(t, x_{t-}, v_t, \alpha_{t-}) - b(t, y_{t-}, v_t, \alpha_{t-}) \right|^2 \, dt \\
+ \beta \mathbb{E} \sum_{d=1}^D \int_0^s \left| \sigma^{sd}(t, x_{t-}, v_t, \alpha_{t-}) - \sigma^{sd}(t, y_{t-}, v_t, \alpha_{t-}) \right|^2 \, dt
\]
\begin{align*}
+ \beta \mathbb{E} \sum_{l=1}^{L} \int_{0}^{s} \int_{E_0} |\gamma^l(t, x_{t-}, v_t, \alpha_{t-}, \zeta)|^2 \lambda^l(d\zeta)dt \\
+ \beta \mathbb{E} \sum_{m=1}^{M} \int_{0}^{s} |\eta^m(t, x_{t-}, v_t, \alpha_{t-})|^2 G^{\alpha_{t-} - m} 1_{\{\alpha \neq m\}} dt
\end{align*}

Use (1.8) and combine like terms to get
\begin{equation*}
\mathbb{E} |X_s|^2 \leq \beta (T + \beta - 1) \mathbb{E} \int_{0}^{s} C |x_{t-} - y_{t-}|^2 dt
\end{equation*}
and thus
\begin{equation*}
\mathbb{E} |X_s|^2 \leq C^3 \int_{0}^{s} \mathbb{E} |X_t|^2 dt
\end{equation*}

Apply the Gronwall Inequality to conclude
\begin{equation*}
\mathbb{E} |X_s|^2 = 0 \quad (1.22)
\end{equation*}

To improve this estimate consider again the equation for the process \{X_s\}. Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, take the supremum and then the expectation on both sides to get
\begin{align*}
0 &\leq -\left\{ \mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \right\} \\
+ \beta \mathbb{E} \sup_{0 \leq s \leq T} \left| \int_{0}^{s} b(t, x_{t-}, v_t, \alpha_{t-}) - b(t, y_{t-}, v_t, \alpha_{t-}) dt \right|^2 \\
+ \beta \mathbb{E} \sup_{0 \leq s \leq T} \sum_{d=1}^{D} \left| \int_{0}^{s} \sigma^{*d}(t, x_{t-}, v_t, \alpha_{t-}) - \sigma^{*d}(t, y_{t-}, v_t, \alpha_{t-}) dB_t^d \right|^2 \\
+ \beta \mathbb{E} \sup_{0 \leq s \leq T} \sum_{l=1}^{L} \left| \int_{0}^{s} \int_{E_0} \gamma^l(t, x_{t-}, v_t, \alpha_{t-}, \zeta) \right|^2 \lambda^l(d\zeta)dt
\end{align*}
$$-\gamma^s(t, y_{t-}, v_t, \alpha_{t-}, \zeta)\tilde{N}^l(dt, d\zeta)$$

$$+ \beta E \sup_{0 \leq s \leq T} \sum_{m=1}^{M} \left| \int_{0}^{s} \eta^m(t, x_{t-}, v_t, \alpha_{t-}) \right|^2$$

$$- \eta^m(t, y_{t-}, v_t, \alpha_{t-})d\tilde{Y}^l$$

Apply the Cauchy-Schwarz Inequality for Integrals to the second term, and the Doob Martingale Inequality and the Itô Isometry to the third, fourth, and fifth terms on the right-hand side to produce

$$0 \leq -E \sup_{0 \leq s \leq T} |X_s|^2$$

$$+ \beta T E \int_{0}^{T} \left| b(t, x_{t-}, v_t, \alpha_{t-}) - b(t, y_{t-}, v_t, \alpha_{t-}) \right|^2 dt$$

$$+ 4 \beta E \sum_{d=1}^{D} \int_{0}^{T} \left| \sigma^d(t, x_{t-}, v_t, \alpha_{t-}) - \sigma^d(t, y_{t-}, v_t, \alpha_{t-}) \right|^2 dt$$

$$+ 4 \beta E \sum_{l=1}^{L} \int_{0}^{T} \int_{E_0} \left| \gamma^l(t, x_{t-}, v_t, \alpha_{t-}, \zeta) - \gamma^l(t, y_{t-}, v_t, \alpha_{t-}, \zeta) \right|^2 \lambda^l(d\zeta) dt$$

$$+ 4 \beta E \sum_{m=1}^{M} \int_{0}^{T} \left| \eta^m(t, x_{t-}, v_t, \alpha_{t-}) - \eta^m(t, y_{t-}, v_t, \alpha_{t-}) \right|^2 G^{a_{t-}}(1_{a_{t-} \neq m}) dt$$

Use (1.8) and combine like terms to obtain

$$E \sup_{0 \leq s \leq T} |X_s|^2 \leq 4 \beta (T + \beta - 1) E \int_{0}^{T} C \left| x_{t-} - y_{t-} \right|^2 dt$$

and thus

$$E \sup_{0 \leq s \leq T} |X_s|^2 \leq 4 C^3 \int_{0}^{T} E|X_t|^2 dt$$

Substitute (1.22) to realize

$$E \sup_{0 \leq s \leq T} |X_s|^2 = 0$$
This implies
\[
P\left\{ \sup_{0 \leq s \leq T} |x_s - y_s| = 0 \right\} = 1
\]
and the proof is complete \(\Box\)

### 1.5 The Optimal Relaxed-Control

Example 3 in section 1.1 showed that an otherwise completely reasonable optimal-control problem can fail to have a solution. Upon investigation, Example 3 can be “fixed” if the control set \{-1, 1\} is expanded to include the point 0. More to the point, the “deficiency” in Example 3 is in the inability of the control variable to drive the dynamics of the state variable to 0, and yet, there exist controls that can switch between \(-1\) and 1 with such rapidity that the limit of the objective is the same as if the control variable could drive the dynamics to 0. Put differently, the set of directions (that is, the set of values on the right-hand side of the state equation) is not convex.

The main consequence of upgrading our controls from ordinary controls to relaxed controls is the convexification of the set of directions, with the follow-on consequence that our optimal-control problem has a solution (in the space of relaxed controls). To prove this is quite involved. Our intention in this section is to merely introduce the broad outlines of how such a proof would be carried out, and to take the first step.

The approach we introduce here follows that of Berkovitz and Medhin (2013), in which the proofs are shown in detail for the deterministic case. This approach is also followed for the controlled-diffusion case, most notably in Bahlali et al. (2006). The proofs rely on advanced tools from probability theory; Çınlar (2011) and Kallenberg (2002) was our constant companion.

Let \(M = \inf\{J(\nu_t) : \nu_t\} a\) relaxed control\}. Suppose \(\nu^j_t\) is a sequence of minimizing controls, that is, \(\lim_{j \to \infty} J(\nu^j_t) = M\). Our first task is to show that this converges in some sense.

**Theorem 3.** A minimizing sequence of relaxed controls weakly converges.

**Proof.** Let \(\mathcal{M}([0, T] \times U)\) denote the set of all locally finite measures on \([0, T] \times U\), endowed with the vague topology. (The vague topology is induced by mappings of the form \(m \mapsto \int f(\cdot)m(\cdot)\) for all nonnegative, compactly supported, continuous functions \(f\).) We rewrite the relaxed controls as a random elements \(\nu^j(\omega)(\cdot)\) in \(\mathcal{M}([0, T] \times U)\), which is to say, \(\nu^j(\omega)(\cdot)\) are random measures on \([0, T] \times U\).

Consider the random variables \(\nu^j(\omega)(B)\), with \(B\) an arbitrary compact subset of \([0, T] \times U\). The random variables are tight if
\[
\lim_{r \to \infty} \limsup_{j \to \infty} P\{\nu^j(\omega)(B) > r\} = 0
\]
Note that $\mathbb{P} - a.s.$

$$\nu^j(\omega)([0, T] \times U) = T$$

so tightness follows immediately.

By Prohorov’s Theorem, if $\nu^j(\omega)(B)$ is tight for every such compact $B$ in $[0, T] \times U$, then every subsequence of it has a further subsequence that converges in distribution, which is to say, the $\mathbb{P}$-induced measures on $\mathcal{M}([0, T] \times U)$ converge weakly, and the proof is complete. $\square$

Let $\{\mu_t\}$ denote the limit point from Theorem 3. What remains to be shown is that the corresponding state trajectories $\{x^\nu_t\}$ converge weakly to $\{x^\mu_t\}$. By Skorokhod Coupling, there exist a probability triple on which the minimizing subsequence converges $a.s.$ All that is left then it to verify that $J(\{\mu_t\}) = M$. 
In chapter 1 we introduced the optimal-control problem, and proved a couple of existence theorems. However, these results did not indicate how to find an optimal control in any given optimal-control problem. Nor did these results describe any features an optimal control might have, except of course optimality. What is needed are tests to determine if a candidate control is indeed an optimal control.

There are two standard approaches to developing techniques to enrich the theory of optimal control: the Pontryagin Minimum Principle and the Bellman Dynamic Programming Principle. The former approach establishes a test that determines if a candidate control fails to be an optimal control, that is, it establishes a necessary but not sufficient condition for a candidate control to be an optimal control. The approach mimics ordinary calculus. If the cost is determined by the state and control processes, and the state process is itself determined by the control process, then the cost is really determined by just the control process. If a candidate control is indeed an optimal control, then no other nearby control will produce a lower cost. If the cost depends continuously on the control, and if that dependence can be managed by a sufficiently nice limiting process, then the test is just an analogue of the first derivative test. This test can be reformulated via a Hamiltonian and used to construct directly or iteratively an optimal control, and hence calculate the minimal cost.

The second approach exploits the Markovian structure of the optimal-control problem. Es-
sentially it allows optimal-control problems to be divided into steps and an optimal control to be constructed via backwards induction. At the final time, or step, the cost is completely determined by the value of the state process. (There are no more decisions to be made.) At the previous step, given any state at that step, the cost is the sum of the cost of transitioning from that state to a final-time state, plus the cost of being at that final-time state. Continuing this chain backwards an optimal control can be constructed. In continuous time, this approach leads to the Hamilton-Jacobi-Bellman equation. Solving the Hamilton-Jacobi-Bellman equation not only gives an optimal control, but also the associated minimal cost.

Our plan for the present chapter is as follows. In section 2.1 we define a perturbation in control, and derive the variational equation and the variational inequality. Using these we then, in section 2.2, prove the Pontryagin Minimum Principle, the first of our necessary conditions for optimality. In section 2.3 we define the value function and show it to satisfy the Hamilton-Jacobi-Bellman equation. Lastly, in section 2.4 we show that the costate process from the Pontryagin Minimum Principle and the value function from the Bellman Dynamic Programming Principle are related via certain partial derivatives.

2.1 Perturbations, Part 1: The Variational Inequality

Let \{u_t\} and \{v_t\} be ordinary-control processes. Clearly any linear combination of ordinary-control processes is an ordinary-control process. In particular if \(0 < \theta \ll 1\) the ordinary-control process given by

\[ u^\theta_t := u_t + \theta v_t \]  (2.1)

will be called the perturbation of \{u_t\} in the direction of \{v_t\}, or simply, the perturbation \{u^\theta_t\}. If \{u_t\} is optimal, it follows immediately from (1.11) that

\[ J(\{u^\theta_t\}) - J(\{u_t\}) \geq 0 \]  (2.2)

Our first task is to show the convergence of state processes as the perturbation vanishes; essentially, the state process is continuous in perturbation. We remark that the convergence is in the sense of \(L^2(\Omega)\) and uniform in \(t\).

**Theorem 4.** Let \{u_t\} and \{v_t\} be ordinary-control processes; \{u^\theta_t\} the perturbation of \{u_t\} in the direction of \{v_t\}, as defined in (2.1); and \{x^u_t\} and \{x^\theta_t\} the solutions to the state equation (1.9) with ordinary-control processes \{u_t\} and \{u^\theta_t\}, respectively. Then

\[ \lim_{\theta \downarrow 0} \mathbb{E} \left[ \sup_{0 \leq r \leq T} \left| x^\theta_r - x^u_r \right|^2 \right] = 0 \]
Proof. Let \( s \in [0, T] \) and \( X_s := x^\theta_s - x^u_s \), so that we have \( \mathbb{P} - a.s. \)

\[
X_s = \int_0^s b(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - b(t, x^u_t, u_t, \alpha_{t-}) \, dt \\
+ \sum_{d=1}^D \int_0^s \sigma^d(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - \sigma^d(t, x^u_t, u_t, \alpha_{t-}) \, dB^d_t \\
+ \sum_{l=1}^L \int_0^s \gamma^l(t, x^\theta_t, u^\theta_t, \alpha_{t-}, \zeta), - \gamma^l(t, x^u_t, u_t, \alpha_{t-}, \zeta) \, \tilde{N}^l_t(d\xi, d\zeta) \\
+ \sum_{m=1}^M \int_0^s \eta^m(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - \eta^m(t, x^u_t, u_t, \alpha_{t-}) \, d\tilde{\gamma}^m_t
\]

Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, and take expectation on both sides to get

\[
\mathbb{E}|X_s|^2 \leq \beta \mathbb{E} \left\| \int_0^s b(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - b(t, x^u_t, u_t, \alpha_{t-}) \, dt \right\|^2 \\
+ \beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \sigma^d(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - \sigma^d(t, x^u_t, u_t, \alpha_{t-}) \, dB^d_t \right|^2 \\
+ \beta \mathbb{E} \sum_{l=1}^L \left| \int_0^s \gamma^l(t, x^\theta_t, u^\theta_t, \alpha_{t-}, \zeta) \\
- \gamma^l(t, x^u_t, u_t, \alpha_{t-}, \zeta) \, \tilde{N}^l_t(d\xi, d\zeta) \right|^2 \\
+ \beta \mathbb{E} \sum_{m=1}^M \left| \int_0^s \eta^m(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - \eta^m(t, x^u_t, u_t, \alpha_{t-}) \, d\tilde{\gamma}^m_t \right|^2
\]

where \( \beta = 1 + D + L + M \). Apply the Cauchy-Schwarz Inequality for Integrals to the first term and the Itô Isometry to the second, third, and fourth terms on the right-hand side to obtain

\[
\mathbb{E}|X_s|^2 \leq \beta \mathbb{E} \int_0^s \left| b(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - b(t, x^u_t, u_t, \alpha_{t-}) \right|^2 \, dt \\
+ \beta \mathbb{E} \sum_{d=1}^D \int_0^s \left| \sigma^d(t, x^\theta_t, u^\theta_t, \alpha_{t-}) - \sigma^d(t, x^u_t, u_t, \alpha_{t-}) \right|^2 \, dt \\
+ \beta \mathbb{E} \sum_{l=1}^L \int_0^s \left| \gamma^l(t, x^\theta_t, u^\theta_t, \alpha_{t-}, \zeta) \\
- \gamma^l(t, x^u_t, u_t, \alpha_{t-}, \zeta) \right|^2 \, \lambda^l(d\zeta) \, dt
\]
\[ + \beta \mathbb{E} \sum_{m=1}^{M} \int_{0}^{s} \left| \eta_{X_{s}^m(t, x_{t-}^\theta, u_{t-}^\theta, \alpha_{t-})} + \eta_{X_{s}^m(t, x_{t-}^\mu, u_{t-}, \alpha_{t-})} \right|^2 G^{a_{t-} - m} \mathbb{1}_{(a_{t-} \neq m)} \, dt \]

Use (1.8) and combine like terms, then use (2.1) to get

\[ \mathbb{E} |X_s|^2 \leq \beta (T + \beta - 1) \mathbb{E} \int_{0}^{s} C \left( |x_{t-}^\theta - x_{t-}^\mu|^2 + |\theta v_t|^2 \right) \, dt \]

Recall that the control set \( U \) is compact, which implies

\[ \mathbb{E} |X_s|^2 \leq \beta (T + \beta - 1) C \int_{0}^{s} \mathbb{E} |X_t|^2 \, dt + \theta^2 \beta (T + \beta - 1) C^2 T \]

\[ \leq C^3 \int_{0}^{s} \mathbb{E} |X_t|^2 \, dt + \theta^2 C^5 \]

Apply the Gronwall Inequality to produce

\[ \mathbb{E} |X_s|^2 \leq \theta^2 C^5 e^{C^3 T} \quad (2.3) \]

To improve this estimate consider again the equation for the process \( \{X_s\} \). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, take the supremum and then the expectation on both sides to get

\[ 0 \leq -\left\{ \mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \right\} \]

\[ + \beta \mathbb{E} \sup_{0 \leq s \leq T} \left| \int_{0}^{s} b(t, x_{t-}^\theta, u_{t-}^\theta, \alpha_{t-}) - b(t, x_{t-}^\mu, u_{t-}, \alpha_{t-}) \, dt \right|^2 \]

\[ + \beta \mathbb{E} \sup_{0 \leq s \leq T} \sum_{d=1}^{D} \left| \int_{0}^{s} \sigma^d(t, x_{t-}^\theta, u_{t-}^\theta, \alpha_{t-}) \right|^2 \]

\[ - \sigma^d(t, x_{t-}^\mu, u_{t-}, \alpha_{t-}) dB_t^d \]

\[ + \beta \mathbb{E} \sup_{0 \leq s \leq T} \sum_{l=1}^{L} \left| \int_{0}^{s} \gamma^l(t, x_{t-}^\theta, u_{t-}^\theta, \alpha_{t-}) \right|^2 \]

\[ - \gamma^l(t, x_{t-}^\mu, u_{t-}, \alpha_{t-}) \tilde{N}_l(dt, d\zeta) \]
Apply the Cauchy-Schwarz Inequality for Integrals to the second term, and the Doob Martingale Inequality and the Itô Isometry to the third, fourth, and fifth terms on the right-hand side to produce

\[
0 \leq - \left( \mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \right)
+ \beta T \mathbb{E} \int_0^T \left| b(t, x_t^\theta, u_t^\theta, \alpha_{t-}) - b(t, x_t^u, u_t, \alpha_{t-}) \right|^2 dt
+ 4 \beta \mathbb{E} \sum_{d=1}^D \int_0^T \left| \sigma^{*d}(t, x_t^\theta, u_t^\theta, \alpha_{t-}) - \sigma^{*d}(t, x_t^u, u_t, \alpha_{t-}) \right|^2 dt
+ 4 \beta \mathbb{E} \sum_{l=1}^L \int_0^T \int_{E_0} \left| \gamma^{*l}(t, x_t^\theta, u_t^\theta, \alpha_{t-}, \zeta) - \gamma^{*l}(t, x_t^u, u_t, \alpha_{t-}, \zeta) \right|^2 \lambda^l(d\zeta) dt
+ 4 \beta \mathbb{E} \sum_{m=1}^M \int_0^T \left| \eta^{*m}(t, x_t^\theta, u_t^\theta, \alpha_{t-}) - \eta^{*m}(t, x_t^u, u_t, \alpha_{t-}) \right|^2 G^{\alpha_{t-}, m} 1_{[\alpha_{t-} \neq m]} dt
\]

Use (1.8), combine like terms, and substitute (2.1) to obtain

\[
\mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \leq 4 \beta (T + \beta - 1) \mathbb{E} \int_0^T C \left( |x_t^\theta - x_t^u|^2 + |\theta u_t|^2 \right) dt
\]

and thus

\[
\mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \leq 4 \beta (T + \beta - 1) C \int_0^T \mathbb{E}|X_t|^2 dt + 4 \theta^2 \beta (T + \beta - 1) C^2 T
\]

\[
\leq 4 C^3 \int_0^T \mathbb{E}|X_t|^2 dt + 4 \theta^2 C^5
\]

where we have again used the fact that the control set $U$ is compact. Substitute (2.3) and compute
the integral to realize
\[
\mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \leq 4 C^3 \int_0^T \theta^2 C^5 e^{C^3 t} dt + 4 \theta^2 C^5 \\
\leq 4 \theta^2 C^8 T e^{C^3 T} + 4 \theta^2 C^5
\]

Let \( \theta \) decrease to 0 to complete the proof.

Our next task is to show the convergence of state processes as the perturbation vanishes is order-\( \theta \); essentially, the state process is differentiable in perturbation. We remark again that the convergence is in the sense of \( L^2(\Omega) \) and uniform in \( t \).

**Theorem 5.** Let \( \{u_t\} \) and \( \{v_t\} \) be ordinary-control processes; \( \{u_\theta^\theta\} \) the perturbation of \( \{u_t\} \) in the direction of \( \{v_t\} \), as defined in (2.1); and \( \{x^u_t\} \) and \( \{x^\theta_t\} \) the solutions to the state equation (1.9) with ordinary-control processes \( \{u_t\} \) and \( \{u^\theta_t\} \), respectively. Then

\[
\lim_{\theta \to 0} \mathbb{E} \left[ \sup_{0 \leq t \leq T} \left| \frac{x^\theta_t - x^u_t}{\theta} - z_t \right|^2 \right] = 0
\]

where \( \{z_t\} \) satisfies \( \mathbb{P} - a.s. \)

\[
dz_t = b_x(t, x^u_t, u_t, \alpha_{t-}) z_t dt + \sum_{d=1}^D \sigma_x^{sd}(t, x^u_t, u_t, \alpha_{t-}) z_t dB^d_t \\
+ \sum_{l=1}^L \int_{E_0} \gamma_x^l(t, x^u_t, u_t, \alpha_{t-}, \zeta) z_t \tilde{N}^l(dt, d\zeta) \\
+ \sum_{m=1}^M \eta_x^m(t, x^u_t, u_t, \alpha_{t-}) z_t d\tilde{\gamma}^m_t \tag{2.4}
\]

\[
+ b_v(t, x^u_t, u_t, \alpha_{t-}) v_t dt + \sum_{d=1}^D \sigma_v^{sd}(t, x^u_t, u_t, \alpha_{t-}) v_t dB^d_t \\
+ \sum_{l=1}^L \int_{E_0} \gamma_v^l(t, x^u_t, u_t, \alpha_{t-}, \zeta) v_t \tilde{N}^l(dt, d\zeta) \\
+ \sum_{m=1}^M \eta_v^m(t, x^u_t, u_t, \alpha_{t-}) v_t d\tilde{\gamma}^m_t
\]

\( z_0 = 0 \)

We call (2.4) the variational equation corresponding to the perturbation \( \{u^\theta_t\} \).

**Proof.** We first derive a particularly convenient form of the state equation for the quantity \( \frac{x^\theta_t - x^u_t}{\theta} - z_t \). We next compute a second-moment estimate of this quantity in terms of this quantity and some
Apply of the Fundamental Theorem of Calculus to the first, second, third, and fourth terms on the
determine that the second-moment estimate also vanishes, and does so uniformly on \([0, T]\).

(Step 1.) Let \(s \in [0, T]\) and \(X_s := \frac{x^0_s - x^u_s}{\theta} - z_t\), so that we have \(\mathbb{P} \text{ a.s.}
\)

\[
X_s = \frac{1}{\theta} \int_0^s b(t, x^0_{t^-}, u_t, \alpha_{t-}) - b(t, x^u_{t^-}, u_t, \alpha_{t-}) \, dt
\]

\[
+ \frac{1}{\theta} \sum_{d=1}^D \int_0^s \sigma^d(t, x^0_{t^-}, u^0_t, \alpha_{t-}) - \sigma^d(t, x^u_{t^-}, u_t, \alpha_{t-}) \, dB^d_t
\]

\[
+ \frac{1}{\theta} \sum_{l=1}^L \int_{E_0}^s \gamma^l(t, x^0_{t^-}, u^0_t, \alpha_{t-}, \zeta) - \gamma^l(t, x^u_{t^-}, u_t, \alpha_{t-}, \zeta) \, \tilde{N}^l_t \, dt \, d\zeta
\]

\[
+ \frac{1}{\theta} \sum_{m=1}^M \int_0^s \eta^m(t, x^0_{t^-}, u^0_t, \alpha_{t-}) - \eta^m(t, x^u_{t^-}, u_t, \alpha_{t-}) \, d\tilde{Y}^m_t
\]

\[
- b(x(t, x^u_{t^-}, u_t, \alpha_{t-}) z_t \, dt - \sum_{d=1}^D \sigma^d(x(t, x^u_{t^-}, u_t, \alpha_{t-}) z_t \, dB^d_t
\]

\[
- \sum_{l=1}^L \int_{E_0}^s \gamma^l(x(t, x^u_{t^-}, u_t, \alpha_{t-}) z_t \, \tilde{N}^l_t \, dt \, d\zeta
\]

\[
- \sum_{m=1}^M \eta^m(x(t, x^u_{t^-}, u_t, \alpha_{t-}) z_t \, d\tilde{Y}^m_t
\]

\[
- b_v(t, x^u_{t^-}, u_t, \alpha_{t-}) v_t \, dt - \sum_{d=1}^D \sigma^d_v(t, x^u_{t^-}, u_t, \alpha_{t-}) v_t \, dB^d_t
\]

\[
- \sum_{l=1}^L \int_{E_0}^s \gamma^l_v(t, x^u_{t^-}, u_t, \alpha_{t-}) v_t \, \tilde{N}^l_t \, dt \, d\zeta
\]

\[
- \sum_{m=1}^M \eta^m_v(t, x^u_{t^-}, u_t, \alpha_{t-}) v_t \, d\tilde{Y}^m_t
\]

Apply of the Fundamental Theorem of Calculus to the first, second, third, and fourth terms on the
right-hand side to reckon

\[
X_s = \int_0^s \int_0^1 b_x(t, x^u_{t^-} + \rho(x^0_{t^-} - x^u_{t^-}), u_t + \rho \theta v_t, \alpha_{t-}) \frac{1}{\theta} (x^0_{t^-} - x^u_{t^-}) \, d\rho \, dt
\]

\[
+ \int_0^s \int_0^1 b_v(t, x^u_{t^-} + \rho(x^0_{t^-} - x^u_{t^-}), u_t + \rho \theta v_t, \alpha_{t-}) v_t \, d\rho \, dt
\]

\[
+ \sum_{d=1}^D \int_0^s \int_0^1 \sigma^d_x(t, x^u_{t^-} + \rho(x^0_{t^-} - x^u_{t^-}), u_t + \rho \theta v_t, \alpha_{t-}) \frac{1}{\theta} (x^0_{t^-} - x^u_{t^-}) \, d\rho \, dB^d_t
\]
\[0 = \int_0^s \int_0^1 b_x(t, x_t^u + \rho(x_t^\theta - x_t^u), u_t + \rho \theta v_t, \alpha_t)z_t \, d\rho \, dt - \int_0^s \int_0^1 b_x(t, x_t^u + \rho(x_t^\theta - x_t^u), u_t + \rho \theta v_t, \alpha_t)z_t \, d\rho \, dt + \sum_{d=1}^D \int_0^s \int_0^1 \sigma_x^{\ast d}(t, x_t^u + \rho(x_t^\theta - x_t^u), u_t + \rho \theta v_t, \alpha_t)z_t \, d\rho \, dB_t^d \]
and group like terms to obtain

\[
X_s = \int_0^s \int_0^1 b_x(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-})X_{t-} \, d\rho \, dt \\
+ \sum_{d=1}^D \int_0^s \int_0^1 \sigma_{x_i}^d(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-})X_{t-} \, d\rho \, dB^d_t \\
+ \sum_{l=1}^L \int_0^s \int_0^1 \gamma_{x_i}^l(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) \, d\rho \, \tilde{N}_t \\
- \sum_{l=1}^L \int_0^s \int_0^1 \gamma_{x_i}^l(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) \, d\rho \, \tilde{N}_t \\
+ \sum_{m=1}^M \int_0^s \int_0^1 \eta_{x_i}^m(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-})X_{t-} \, d\rho \, \tilde{\tilde{N}}_t^m \\
- \sum_{m=1}^M \int_0^s \int_0^1 \eta_{x_i}^m(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-})X_{t-} \, d\rho \, \tilde{\tilde{N}}_t^m \\
\]

(2.5)
\[ + \sum_{m=1}^{M} \int_{0}^{s} \int_{0}^{1} \{ \eta_{x}^{m}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}) \\
- \eta_{x}^{m}(t, x_{t-}^{u}, \alpha_{t-}) \} \zeta_{t-} \rho d \tilde{\gamma}_{t}^{m} \\\n+ \sum_{d=1}^{D} \int_{0}^{s} \int_{0}^{1} \{ \sigma_{x}^{d}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}) \\\n- \sigma_{x}^{d}(t, x_{t-}^{u}, \alpha_{t-}) \} \rho d \tilde{B}_{t}^{d} \\\n+ \sum_{l=1}^{L} \int_{0}^{s} \int_{E_{0}} \int_{0}^{1} \{ \gamma_{x}^{l}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}, \zeta) \\\n- \gamma_{x}^{l}(t, x_{t-}^{u}, \alpha_{t-}, \zeta) \} \rho d \tilde{N}_{t}^{l}(d \tau, d \zeta) \\\n+ \sum_{m=1}^{M} \int_{0}^{s} \int_{0}^{1} \{ \eta_{x}^{m}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}) \\\n- \eta_{x}^{m}(t, x_{t-}^{u}, \alpha_{t-}) \} \rho d \tilde{\gamma}_{t}^{m} \]  

\((\text{Step 2})\) Consider the equation for \(X_{t}\) given by (2.5). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, and take expectation on both sides to yield

\[ 0 \leq -\mathbb{E}[X_{t}^{2}] \]

\[ +3 \beta \mathbb{E} \left| \int_{0}^{s} \int_{0}^{1} b_{x}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-})X_{t-} \rho d \tilde{\gamma}_{t}^{m} \right|^{2} \]

\[ +3 \beta \mathbb{E} \left| \int_{0}^{s} \int_{0}^{1} \sigma_{x}^{d}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}) \rho d \tilde{B}_{t}^{d} \right|^{2} \]

\[ +3 \beta \mathbb{E} \left| \int_{0}^{s} \int_{E_{0}} \int_{0}^{1} \gamma_{x}^{l}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}, \zeta) \rho d \tilde{N}_{t}^{l}(d \tau, d \zeta) \right|^{2} \]

\[ +3 \beta \mathbb{E} \left| \int_{0}^{s} \int_{0}^{1} \eta_{x}^{m}(t, x_{t-}^{u} + \rho(x_{t-}^{0} - x_{t-}^{u}), u_{t} + \rho \theta v_{t}, \alpha_{t-}) \rho d \tilde{\gamma}_{t}^{m} \right|^{2} \]
\[ X_t \cdot d\rho d\tilde{Y}^m_t \]

\[ + 3\beta \mathbb{E} \left| \int_0^s \int_0^1 \left\{ b_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \\
- b_x(t, x^u_{t-}, u_t, \alpha_{t-}) \right\} z_{t-} d\rho d\tau \right|^2 \]

\[ + 3\beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \int_0^1 \left\{ \sigma^\nu_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \\
- \sigma^\nu_x(t, x^u_{t-}, u_t, \alpha_{t-}) \right\} z_{t-} d\rho d\tilde{B}^d_t \right|^2 \]

\[ + 3\beta \mathbb{E} \sum_{l=1}^L \left| \int_0^s \int_{E_0} \int_0^1 \left\{ \gamma^v_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) \\
- \gamma^v_x(t, x^u_{t-}, u_t, \alpha_{t-}, \zeta) \right\} z_{t-} d\rho \tilde{N}^l(t, d\zeta) \right|^2 \]

\[ + 3\beta \mathbb{E} \sum_{m=1}^M \left| \int_0^s \int_0^1 \left\{ \eta^m_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \\
- \eta^m_x(t, x^u_{t-}, u_t, \alpha_{t-}) \right\} z_{t-} d\rho d\tilde{Y}^m_t \right|^2 \]

\[ + 3\beta \mathbb{E} \left| \int_0^s \int_0^1 \left\{ b_v(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \\
- b_v(t, x^u_{t-}, u_t, \alpha_{t-}) \right\} v_t d\rho d\tau \right|^2 \]

\[ + 3\beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \int_0^1 \left\{ \sigma^\nu_v(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \\
- \sigma^\nu_v(t, x^u_{t-}, u_t, \alpha_{t-}) \right\} v_t d\rho d\tilde{B}^d_t \right|^2 \]

\[ + 3\beta \mathbb{E} \sum_{l=1}^L \left| \int_0^s \int_{E_0} \int_0^1 \left\{ \gamma^v_v(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) \\
- \gamma^v_v(t, x^u_{t-}, u_t, \alpha_{t-}, \zeta) \right\} v_t d\rho \tilde{N}^l(t, d\zeta) \right|^2 \]
where $\beta = 1 + D + L + M$. Apply the Cauchy-Schwarz Inequality for Integrals to the second, sixth, and tenth terms and the Itô Isometry to the third, fourth, fifth, seventh, eighth, ninth, eleventh, twelfth and thirteenth terms on the right-hand side to get

\[
0 \leq -\mathbb{E}[X_s^2]
\]

\[
+ 3 \beta \mathbb{E} \sum_{m=1}^{M} \int_0^s \int_0^1 \{ \eta_{x}^m(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) - \eta_{x}^m(t, x_{t-}^u, u_t, \alpha_{t-}) \} v_t d\rho d\tilde{\gamma}_t^m
\]

\[
X_{t-}^2 d\rho dt
\]

\[
+ 3 \beta \mathbb{E} \sum_{l=1}^{L} \int_0^s \int_{E_0}^1 \{ |\gamma_x^l(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) - \gamma_x^l(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) |^2 d\rho \lambda^l(d\zeta) dt
\]

\[
+ 3 \beta \mathbb{E} \sum_{m=1}^{M} \int_0^s \int_0^1 \{ |\eta_x^m(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) - \eta_x^m(t, x_{t-}^u, u_t, \alpha_{t-}) |^2 d\rho dt
\]

\[
+ 3 \beta \mathbb{E} \sum_{d=1}^{D} \int_0^s \int_0^1 \{ |\sigma_x^d(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) - \sigma_x^d(t, x_{t-}^u, u_t, \alpha_{t-}) |^2 d\rho dt
\]

\[
+ 3 \beta \mathbb{E} \sum_{l=1}^{L} \int_0^s \int_{E_0}^1 \{ |\gamma_x^l(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) - \gamma_x^l(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) |^2 d\rho \lambda^l(d\zeta) dt
\]

\[
+ 3 \beta \mathbb{E} \sum_{m=1}^{M} \int_0^s \int_0^1 \{ |\eta_x^m(t, x_{t-}^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) - \eta_x^m(t, x_{t-}^u, u_t, \alpha_{t-}) |^2 d\rho G^{a_{t-m}}(a_{t-} \neq m) dt
\]
Note that (1.8) implies the derivatives of (1.5) are bounded. Collect the first, second, third, and fourth terms on the right-hand side to get

$$\mathbb{E}|X_s|^2 \leq 3\beta(T + \beta - 1)C\mathbb{E} \int_0^s |X_t|^2 dt + \kappa_s^\theta$$

$$\leq 3C^3 \int_0^s \mathbb{E}|X_t|^2 dt + \kappa_s^\theta$$

where

$$\kappa_s^\theta = 3\beta T\mathbb{E} \int_0^s \int_0^1 \left| \{b_x(t, x_{t_+}^u + \rho(x_{t_+}^\theta - x_{t_+}^u), u_t + \rho \theta v_t, a_{t_+}) \right| z_{t_+}^\theta \|^2 dpdt$$

$$+ 3\beta \sum_{d=1}^D \int_0^s \int_0^1 \left| \{\sigma_{x_d}^d(t, x_{t_+}^u, u_t, a_{t_+}) \right| z_{t_+}^\theta \|^2 dpdt$$

$$+ 3\beta \sum_{l=1}^L \int_0^s \int_{\mathcal{E}_l} \int_0^1 \left| \{\gamma_x^l(t, x_{t_+}^u, a_{t_+}, \zeta) \right| z_{t_+}^\theta \|^2 dp \lambda^l(d\zeta)dt$$

$$+ 3\beta \sum_{m=1}^M \int_0^s \int_0^1 \left| \{\eta_x^m(t, x_{t_+}^u, a_{t_+}) \right| z_{t_+}^\theta \|^2 dp \rho^m \gamma_{t_+}^m \{a_{t_+} \neq m \} dt$$
which is clearly uniformly bounded in $s$ and $\theta$. Apply the Gronwall Inequality to reckon

$$\mathbb{E}|X_s|^2 \leq \kappa^\theta T e^{3CT}$$  \hspace{2cm} (2.7)

(Step 3.) Observe that the product measure of each of the integrals in (2.6) is $\sigma$-finite and thus that

the Lebesgue Dominated Convergence Theorem holds for sequences of integrands that converge

merely in measure. Again use (1.8) to deduce that the derivatives of (1.5) are bounded and thus that

each integrand in (2.6) is dominated, either by $\{C|z|_s\}$ or $\{C|v_t\}$. Note that Theorem 4 implies

the convergence in $\mathbb{P}$-measure of $x^\theta_s - x^u_s$ to 0 as $\theta$ decreases to 0. Thus apply the Lebesgue Dominated

Convergence Theorem to (2.6) to conclude

$$\lim_{\theta \to 0} \kappa^\theta_s = 0$$  \hspace{2cm} (2.8)

(Step 4.) To improve this estimate consider again the equation for the process $\{X_s\}$, in particular

(2.5). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, take

the supremum and then the expectation on both sides to get

\begin{align*}
0 &\leq -\mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \\
&+ 3\beta \mathbb{E} \sup_{0 \leq s \leq T} \left| \int_0^s \int_0^1 \left\{ b_x(t, x^u_s - x^\theta_x, u_t + \rho \theta v_t, \alpha_t) \\
b_x(t, x^u_s, u_t, \alpha_t) \right\} v_t \right|^2 \mathbb{E} d\rho dt \\
&+ 3\beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \int_0^1 \left\{ \sigma_x^d(t, x^u_s - x^\theta_x, u_t + \rho \theta v_t, \alpha_t) \\
\sigma_x^d(t, x^u_s, u_t, \alpha_t) \right\} v_t \right|^2 \mathbb{E} d\rho dt
\end{align*}

\[ + 3\beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \int_0^1 \left\{ \gamma_x^d(t, x^u_s - x^\theta_x, u_t + \rho \theta v_t, \alpha_t) \\
\gamma_x^d(t, x^u_s, u_t, \alpha_t) \right\} v_t \right|^2 \mathbb{E} d\rho_\theta \lambda_1(d\zeta) dt \]

which is clearly uniformly bounded in $s$ and $\theta$. Apply the Gronwall Inequality to reckon

$$\mathbb{E}|X_s|^2 \leq \kappa^\theta T e^{3CT}$$  \hspace{2cm} (2.7)

(Step 3.) Observe that the product measure of each of the integrals in (2.6) is $\sigma$-finite and thus that

the Lebesgue Dominated Convergence Theorem holds for sequences of integrands that converge

merely in measure. Again use (1.8) to deduce that the derivatives of (1.5) are bounded and thus that

each integrand in (2.6) is dominated, either by $\{C|z|_s\}$ or $\{C|v_t\}$. Note that Theorem 4 implies

the convergence in $\mathbb{P}$-measure of $x^\theta_s - x^u_s$ to 0 as $\theta$ decreases to 0. Thus apply the Lebesgue Dominated

Convergence Theorem to (2.6) to conclude

$$\lim_{\theta \to 0} \kappa^\theta_s = 0$$  \hspace{2cm} (2.8)

(Step 4.) To improve this estimate consider again the equation for the process $\{X_s\}$, in particular

(2.5). Square both sides, apply the Cauchy-Schwarz Inequality for Sums to the right-hand side, take

the supremum and then the expectation on both sides to get

\begin{align*}
0 &\leq -\mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \\
&+ 3\beta \mathbb{E} \sup_{0 \leq s \leq T} \left| \int_0^s \int_0^1 \left\{ b_x(t, x^u_s - x^\theta_x, u_t + \rho \theta v_t, \alpha_t) \\
b_x(t, x^u_s, u_t, \alpha_t) \right\} v_t \right|^2 \mathbb{E} d\rho dt \\
&+ 3\beta \mathbb{E} \sum_{d=1}^D \left| \int_0^s \int_0^1 \left\{ \sigma_x^d(t, x^u_s - x^\theta_x, u_t + \rho \theta v_t, \alpha_t) \\
\sigma_x^d(t, x^u_s, u_t, \alpha_t) \right\} v_t \right|^2 \mathbb{E} d\rho dt
\end{align*}
\[X_t - d\rho dB_t^d\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \sum_{l=1}^L \int_0^s \int_0^1 \gamma_x^l(t, x_t^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}, \zeta)\]

\[X_t - d\rho N\mathbb{I}(dt, d\zeta)\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \sum_{m=1}^M \int_0^s \int_0^1 \eta_x^m(t, x_t^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-})\]

\[X_t - d\rho d\tilde{\varepsilon}_t^m\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \int_0^s \int_0^1 \left\{ b_x(t, x_t^u, u_t, \alpha_{t-}) \right\}^2\]

\[- b_x(t, x_t^u, u_t, \alpha_{t-}) z_t - d\rho dr\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \sum_{d=1}^D \int_0^s \int_0^1 \left\{ \sigma_x^d(t, x_t^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) \right\}

\[- \sigma_x^d(t, x_t^u, u_t, \alpha_{t-}) z_t - d\rho dB_t^d\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \sum_{l=1}^L \int_0^s \int_0^1 \left\{ \gamma_x^l(t, x_t^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) \right\}

\[- \gamma_x^l(t, x_t^u, u_t, \alpha_{t-}, \zeta) z_t - d\rho N\mathbb{I}(dt, d\zeta)\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \sum_{m=1}^M \int_0^s \int_0^1 \left\{ \eta_x^m(t, x_t^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) \right\}

\[- \eta_x^m(t, x_t^u, u_t, \alpha_{t-}) z_t - d\rho d\tilde{\varepsilon}_t^m\]

\[+ 3\beta \mathbb{E} \sup_{0 \le s \le T} \int_0^s \int_0^1 \left\{ b_v(t, x_t^u + \rho(x_{t-}^\theta - x_{t-}^u), u_t + \rho \theta v_t, \alpha_{t-}) \right\}

\[- b_v(t, x_t^u, u_t, \alpha_{t-}) v_t - d\rho dt\]

39
Apply the Cauchy-Schwarz Inequality for Integrals to the second, sixth, and tenth terms, and the Doob Martingale Inequality and the Itô Isometry to the third, fourth, fifth, seventh, eighth, ninth, eleventh, twelfth and thirteenth terms on the right-hand side to produce

\[
0 \leq -\left\{ \mathbb{E} \sup_{0 \leq s \leq T} |X_s| \right\}^2 + 3\beta \mathbb{E} \int_0^T \int_0^1 \left| \sigma^d_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \right|^2 \rho d\sigma d\tau
\]

\[
+ 12\beta \mathbb{E} \sum_{d=1}^D \int_0^T \int_{E_d} \left| \sigma^d_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \right|^2 \rho d\sigma d\tau
\]

\[
+ 12\beta \mathbb{E} \sum_{l=1}^L \int_0^T \int_{E_l} \left| \gamma^d_l(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}, \zeta) \right|^2 \rho d\lambda^l(d\zeta) d\tau
\]

\[
+ 12\beta \mathbb{E} \sum_{m=1}^M \int_0^T \int_{E_m} \left| \eta^m(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \right|^2 \rho G^{\alpha_{t-}-m}1_{\{\alpha_{t-}\neq m\}} d\tau
\]

\[
+ 3\beta \mathbb{E} \int_0^T \int_0^1 \left| \{ b_x(t, x^u_{t-} + \rho(x^\theta_{t-} - x^u_{t-}), u_t + \rho \theta v_t, \alpha_{t-}) \right|^2 \rho d\sigma d\tau
\]

\[
- b_x(t, x^u_{t-}, u_t, \alpha_{t-}) \right|^2 \rho d\sigma d\tau
\]
Recall that (1.8) implies the derivatives of (1.5) are bounded and collect the first, second, third, and fourth terms on the right-hand side to get

\[ \mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \leq 12 \beta (T + \beta - 1) C \mathbb{E} \int_0^T |X_{t-}|^2 \, dt + 4 \kappa_T^\theta \]
\[ \leq 12 C^3 \int_0^s \mathbb{E}|X_t|^2 \, dt + 4 \kappa_T^\theta \]

Substitute (2.7) and compute the integral to realize

\[ \mathbb{E} \sup_{0 \leq s \leq T} |X_s|^2 \leq 12 C^3 \int_0^T \kappa_T^\theta e^{3C^3T} \, dt + 4 \kappa_T^\theta \]
\[ \leq 12 \kappa_T^\theta C^3 T e^{3C^3T} + 4 \kappa_T^\theta \]
Let \( \theta \) decrease to 0 and apply (2.8) to complete the proof.

We now compute the change in cost due to an infinitesimal perturbation of an optimal control. This calculation is akin to differentiating the cost functional with respect to control, and determining it to be nonnegative; which is to say, a first derivative test.

**Theorem 6.** Let \( \{ u_t \} \) be an optimal ordinary-control process, that is, a solution to problem (1.11); \( \{ v_t \} \) an ordinary-control process; \( \{ u^\theta_t \} \) the perturbation of \( \{ u_t \} \) in the direction of \( \{ v_t \} \), as defined in (2.1); \( \{ x^u_t \} \) the solution to the state equation (1.9) with ordinary-control process \( \{ u_t \} \); and \( \{ z_t \} \) the variation corresponding to the perturbation \( \{ u^\theta_t \} \), as defined in (2.4). Then

\[
\mathbb{E} \left[ \int_0^T h(x, x^u_{t-}, u_t, \alpha_{t-})z_{t-} + h_v(t, x^u_{t-}, u_t, \alpha_{t-})v_t \, dt + g_x(x^u_T, \alpha_T)z_T \right] \geq 0
\]

We call this inequality the variational inequality corresponding to the perturbation \( \{ u^\theta_t \} \).

**Proof.** Use (2.2) and (1.10), rearrange inequality, and then apply the Fundamental Theorem of Calculus to the right-hand side to reckon

\[
0 \leq J(\{ u^\theta_t \}) - J(\{ u_t \})
\]

\[
= \mathbb{E} \left[ \int_0^T h(t, x^\theta_{t-}, u^\theta_t, \alpha_{t-}) \, dt + g(x^\theta_T, \alpha_T) \right]
\]

\[
- \mathbb{E} \left[ \int_0^T h(t, x^u_{t-}, u_t, \alpha_{t-}) \, dt + g(x^u_T, \alpha_T) \right]
\]

\[
= \mathbb{E} \int_0^T \int_0^1 h_x(t, x^u_{t-}, \alpha_{t-}) \, d\rho \, dt
\]

\[
+ \mathbb{E} \int_0^T \int_0^1 h_v(t, x^u_{t-}, \alpha_{t-}) \, d\rho \, dt
\]

\[
+ \mathbb{E} \int_0^T g_x(x^u_T, \alpha_T) \, d\rho
\]

Add the following equation

\[
0 = \mathbb{E} \int_0^T \int_0^1 h_x(t, x^u_{t-}, \alpha_{t-}) \, d\rho \, dt
\]

\[
- \mathbb{E} \int_0^T \int_0^1 h_v(t, x^u_{t-}, \alpha_{t-}) \, d\rho \, dt
\]

\[
+ \mathbb{E} \int_0^T g_x(x^u_T, \alpha_T) \, d\rho
\]
\[-E \int_0^1 g_x(x_T^u + \rho(x_T^\theta - x_T^u), \alpha_T)\theta z_T \, d\rho\]

group like terms, and divide by \(\theta\), which recall is strictly positive, to produce

\[
0 \leq E \int_0^T \int_0^1 h_x(t, x_T^u + \rho(x_T^\theta - x_T^u), u_t + \rho \theta v_t, \alpha_T) \, d\rho \, dt
\]

\[
+ E \int_0^T \int_0^1 h_v(t, x_T^u + \rho(x_T^\theta - x_T^u), u_t + \rho \theta v_t, \alpha_T) \, d\rho \, dt
\]

\[
+ E \int_0^1 g_x(x_T^u + \rho(x_T^\theta - x_T^u), \alpha_T)z_T \, d\rho
\]

\[
+ \theta^2
\]

where

\[
\kappa_T^\theta = E \int_0^T \int_0^1 h_x(t, x_T^u + \rho(x_T^\theta - x_T^u), u_t + \rho \theta v_t, \alpha_T)
\]

\[
\left\{ \frac{x_T^\theta - x_T^u}{\theta} - z_T \right\} \, d\rho \, dt
\]

\[
+ E \int_0^1 g_x(x_T^u + \rho(x_T^\theta - x_T^u), \alpha_T) \left\{ \frac{x_T^\theta - x_T^u}{\theta} - z_T \right\} \, d\rho
\]

Consider the equation for \(\kappa_T^\theta\). Square both sides, and apply the Cauchy-Schwarz Inequality for Sums and the Jensen Inequality to the right-hand side to obtain

\[
|\kappa_T^\theta|^2 \leq 2E \left| \int_0^T \int_0^1 h_x(t, x_T^u + \rho(x_T^\theta - x_T^u), u_t + \rho \theta v_t, \alpha_T)
\]

\[
\left\{ \frac{x_T^\theta - x_T^u}{\theta} - z_T \right\} \, d\rho \, dt \right|^2
\]

\[
+ 2E \left| \int_0^1 g_x(x_T^u + \rho(x_T^\theta - x_T^u), \alpha_T) \left\{ \frac{x_T^\theta - x_T^u}{\theta} - z_T \right\} \, d\rho \right|^2
\]

Note that (1.8) implies the derivatives of (1.6) are bounded. Use this fact together with the Cauchy-Schwarz Inequality for Integrals on the right-hand side to produce

\[
|\kappa_T^\theta|^2 \leq 2C^2 T E \int_0^T \left| \frac{x_T^\theta - x_T^u}{\theta} - z_T \right|^2 \, dt + 2C^2 E \left| \frac{x_T^\theta - x_T^u}{\theta} - z_T \right|^2
\]
\[ \leq 2 C^2 (T^2 + 1) \mathbb{E} \sup_{0 \leq t \leq T} \left| \frac{x_t^\theta - x_t^\mu}{\theta} - z_t \right|^2 \]

Let \( \theta \) decrease to 0 and apply Theorem 5 to realize

\[ \lim_{\theta \downarrow 0} \kappa_t^\theta = 0 \]

and thus

\[ 0 \leq \lim_{\theta \downarrow 0} \mathbb{E} \int_0^T \int_0^1 h_x(t, x_t^\mu + \rho(x_t^\theta - x_t^\mu), u_t + \rho \theta v_t, \alpha_{t-})z_t \, d\rho \, dt \]

\[ + \lim_{\theta \downarrow 0} \mathbb{E} \int_0^T \int_0^1 h_v(t, x_t^\mu + \rho(x_t^\theta - x_t^\mu), u_t + \rho \theta v_t, \alpha_{t-})v_t \, d\rho \, dt \]

\[ + \lim_{\theta \downarrow 0} \mathbb{E} \int_0^T \int_0^1 g_x(x_T^\mu + \rho(x_T^\theta - x_T^\mu), \alpha_T)z_T \, d\rho \]

Observe that the product measure of each of these integrals is finite and thus that the Lebesgue Dominated Convergence Theorem holds for sequences of integrands that converge merely in measure. Again use (1.8) to deduce that the derivatives of (1.6) are bounded and thus that each integrand above is dominated, either by \( \{C \mid z_{t-}\} \) or \( \{C \mid v_t\} \). Note that Theorem 4 implies the convergence in \( \mathbb{P} \)-measure of \( x_t^\theta - x_t^\mu \) to 0 as \( \theta \) decreases to 0. Thus apply the Lebesgue Dominated Convergence Theorem to complete the proof.

\[ \square \]

### 2.2 Perturbations, Part 2: Pontryagin's Minimum Principle

It is not clear that the result of Theorem 6 is linear in control, or more precisely, direction of the perturbation. It should be if our interpretation of it as a first derivative test is accurate. To that end, consider the expression

\[ \mathbb{E} \left[ \int_0^T h_x(t, x_t^\mu, u_t, \alpha_{t-})z_t \, dt + g_x(x_T^\mu, \alpha_T)z_T \right] \]

under the same assumptions and notation as Theorem 6. Clearly this is a continuous linear functional in \( z_t \). We observe that (2.4) is linear in \( z_t \), which together with the fact that \( \mathbb{P} - a.s. z_0 = 0 \) suggests that the solution \( \{z_t\} \) is linear in its forcing terms, and thus that the expression above is a continuous linear functional in the forcing terms of (2.4). The space of forcing terms is the Hilbert space formed by the \( (N + N D + N L + N M) \)-dimensional Cartesian product of \( L^2_{\text{pred}}([0, T] \times \Omega) \), \( L^2_{\text{pred}}([0, T] \times \mathbb{N}) \times \Omega \), and \( L^2_{\text{pred}}([0, T] \times \mathbb{N} \times \Omega) \). By the Riesz Representation Theorem there exists a unique element, which we denote \( \{(p_{t-}, q_t, r(t, \cdot), s_t)\} \) and name the costate process corresponding to the optimal ordinary-
control process \( \{u_t\} \), that characterizes this continuous linear functional. The following theorem provides an equation for the costate process.

**Theorem 7.** Let \( \{u_t\} \) be an optimal ordinary-control process, that is, a solution to problem (1.11); \( \{v_t\} \) an ordinary-control process; \( \{u^0_t\} \) the perturbation of \( \{u_t\} \) in the direction of \( \{v_t\} \), as defined in (2.1); \( \{x^m_t\} \) the solution to the state equation (1.9) with ordinary-control process \( \{u_t\} \); and \( \{z_t\} \) the variation corresponding to the perturbation \( \{u^0_t\} \), as defined in (2.4). If \( \{(p_t, q_t, r(t, s), s_t)\} \), that is, the costate process corresponding to the optimal ordinary-control process \( \{u_t\} \), satisfies \( \mathbb{P} - a.s. \)

\[
dp_t = -h^T(t, x^u_{t-}, u_t, \alpha_{t-}) dt - b^T(t, x^u_{t-}, u_t, \alpha_{t-}) p_{t-} dt
\]

\[
- \sum_{d=1}^D \langle \sigma^{*d} \rangle^T(t, x^u_{t-}, u_t, \alpha_{t-}) q^{*d}_{t-} dt
\]

\[
- \sum_{l=1}^L \int_{E_0} (\gamma^{*l})^T(t, x^u_{t-}, u_t, \alpha_{t-}, \zeta) r^{*l}(t, \zeta) \lambda^l(d\zeta) dt
\]

\[
- \sum_{m=1}^M (\eta^{*m}_x)^T(t, x^u_{t-}, u_t, \alpha_{t-}, s_{t-}^m G^{\alpha_{t-}, m} 1_{[\alpha_{t-}, \neq m]}) dt
\]

\[
+ \sum_{d=1}^D q^{*d}_t dB^d_t + \sum_{l=1}^L \int_{E_0} r^{*l}(t, \zeta) \hat{N}^l dt d\zeta + \sum_{m=1}^M s^m_t d\tilde{T}_t
\]

\[
p_T = g^T(x^u_T, \alpha_T)
\]

then

\[
0 = - \left\{ \mathbb{E} \left[ \int_0^T h_x(t, x^u_{t-}, u_t, \alpha_{t-}) z_{t-} dt + g_x(x^u_T, \alpha_T) z_T \right] \right\}
\]

\[
+ \mathbb{E} \int_0^T p^T_t b_v(t, x^u_{t-}, u_t, \alpha_{t-}) v_t dt
\]

\[
+ \mathbb{E} \sum_{d=1}^D \int_0^T (q^{*d}_t)^T \sigma^{*d}(t, x^u_{t-}, u_t, \alpha_{t-}) v_t dt
\]

\[
+ \mathbb{E} \sum_{l=1}^L \int_0^T \int_{E_0} (r^{*l})^T(t, \zeta) \gamma^{*l}_v(t, x^u_{t-}, u_t, \alpha_{t-}, \zeta) v_t \lambda^l(d\zeta) dt
\]

\[
+ \mathbb{E} \sum_{m=1}^M \int_0^T (s^m_t)^T \eta^{*m}_v(t, x^u_{t-}, u_t, \alpha_{t-}) v_t G^{\alpha_{t-}, m} 1_{[\alpha_{t-}, \neq m]} dt
\]

Moreover, the solution to (2.9) is the only such process satisfying (2.10). We call (2.9) the costate equation corresponding to the optimal ordinary-control process \( \{u_t\} \).
**Proof.** Apply the integration by parts formula to the \(L^2_p([0, T] \times \Omega)\)-product of \(p_t\) and \(z_t\) to yield
\[
d(p_t^T z_t) = z_t^T dp_t + p_t^T dz_t + d[p, z]_t
\]

(2.11)

Consider the first term on the right-hand side of (2.11) and substitute (2.9) to get
\[
z_t^T dp_t = -z_t^T h^T(t, x_{t-}^u, u_t, \alpha_{t-})dt - z_t^T b^T(t, x_{t-}^u, u_t, \alpha_{t-})p_{t-} dt
\]
\[
- \sum_{d=1}^{D} z_t^T \sigma_x^{sd}(t, x_{t-}^u, u_t, \alpha_{t-})q_{t-}^d dt
\]
\[
- \sum_{l=1}^{L} \int_{E_0} z_t^T \gamma_{x}^{sl}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) \lambda^l(t, \zeta) \lambda^l(d\zeta) dt
\]
\[
- \sum_{m=1}^{M} z_t^T \eta_{x}^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}) \zeta_s^m G^{a_{t-} - m} 1_{[a_{t-} \neq m]} dt
\]
\[
+ \sum_{d=1}^{D} z_t^T q_{t}^d dB_t^d + \sum_{l=1}^{L} \int_{E_0} z_t^T r_{x}^{sl}(t, \zeta) \tilde{N}^l(dt, d\zeta) + \sum_{m=1}^{M} z_t^T \zeta_s^m d\tilde{\tau}^m_t
\]

Integrate and take expectation on both sides to obtain
\[
E \int_0^T z_{t-}^T dp_t = -E \int_0^T z_{t-}^T h_{t-}^T(t, x_{t-}^u, u_t, \alpha_{t-}) dt
\]
\[
- E \int_0^T z_{t-}^T b_{t-}^T(t, x_{t-}^u, u_t, \alpha_{t-}) p_{t-} dt
\]
\[
- E \sum_{d=1}^{D} \int_0^T z_{t-}^T (\sigma_x^{sd})^T(t, x_{t-}^u, u_t, \alpha_{t-}) q_{t-}^d dt
\]
\[
- E \sum_{l=1}^{L} \int_{E_0} \int_0^T z_{t-}^T (\gamma_{x}^{sl})^T(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) r_{x}^{sl}(t, \zeta) \lambda^l(d\zeta) dt
\]
\[
- E \sum_{m=1}^{M} \int_0^T z_{t-}^T (\eta_{x}^{sm})^T(t, x_{t-}^u, u_t, \alpha_{t-}) \zeta_s^m G^{a_{t-} - m} 1_{[a_{t-} \neq m]} dt
\]

Consider the second term on the right-hand side of (2.11) and substitute (2.4) to produce
\[
p_t^T dz_t = p_t^T b_{x}(t, x_{t-}^u, u_t, \alpha_{t-}) z_{t-} dt + \sum_{d=1}^{D} p_t^T \sigma_x^{sd}(t, x_{t-}^u, u_t, \alpha_{t-}) z_{t-} dB_t^d
\]
\[
+ \sum_{l=1}^{L} \int_{E_0} p_t^T \gamma_{x}^{sl}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) z_{t-} \tilde{N}^l(dt, d\zeta)
\]
\[
+ \sum_{m=1}^{M} p_t^T \eta_{x}^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}) z_{t-} d\tilde{\tau}^m_t
\]

46
\[ + p_{t-}^\top b_v(t, x_{t-}^u, u_t, \alpha_{t-})v_t dt + \sum_{d=1}^D p_{t-}^\top \sigma_v^{sd}(t, x_{t-}^u, u_t, \alpha_{t-})v_t dB_t^d \]
\[ + \sum_{l=1}^L \int_{E_0} p_{t-}^\top \gamma_{v}^{sl}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) v_t \tilde{N}_t^l(dt, d\zeta) \]
\[ + \sum_{m=1}^M p_{t-}^\top n_v^{sm}(t, x_{t-}^u, u_t, \alpha_{t-})v_t d\tilde{Y}_t^m \]

Integrate and take expectation on both sides to obtain
\[
\mathbb{E} \int_0^T p_{t-}^\top dz_t = \mathbb{E} \int_0^T p_{t-}^\top b_v(t, x_{t-}^u, u_t, \alpha_{t-})z_{t-} dt
\]
\[ + \mathbb{E} \int_0^T p_{t-}^\top b_v(t, x_{t-}^u, u_t, \alpha_{t-})v_t dt \]

Consider the third term on the right-hand side of (2.11), substitute (2.9) and (2.4), and use (1.1) and (1.4) to get
\[
dl[p, z]_t = \sum_{d=1}^D (q_{t-}^{sd})^\top \sigma_v^{sd}(t, x_{t-}^u, u_t, \alpha_{t-})z_{t-} dt
\]
\[ + \sum_{d=1}^D (q_{t-}^{sd})^\top \sigma_v^{sd}(t, x_{t-}^u, u_t, \alpha_{t-})v_t dt \]
\[ + \sum_{l=1}^L \int_{E_0} (r_{t-}^{sl})^\top \gamma_{v}^{sl}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) z_{t-} N_t^l(dt, d\zeta) \]
\[ + \sum_{m=1}^M (s_{t-}^{sm})^\top n_v^{sm}(t, x_{t-}^u, u_t, \alpha_{t-})z_{t-} dt \]
\[ + \sum_{m=1}^M (s_{t-}^{sm})^\top n_v^{sm}(t, x_{t-}^u, u_t, \alpha_{t-})v_t d\tilde{Y}_t^m \]
\[ = \sum_{d=1}^D (q_{t-}^{sd})^\top \sigma_v^{sd}(t, x_{t-}^u, u_t, \alpha_{t-})z_{t-} dt \]
\[ + \sum_{d=1}^D (q_{t-}^{sd})^\top \sigma_v^{sd}(t, x_{t-}^u, u_t, \alpha_{t-})v_t dt \]
\[ + \sum_{l=1}^L \int_{E_0} (r_{t-}^{sl})^\top \gamma_{v}^{sl}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) z_{t-} \tilde{N}_t^l(dt, d\zeta) \]
Integrate and take expectation on both sides to obtain

\[
\mathbb{E} \int_0^T d[p, z]_t = \mathbb{E} \sum_{m=1}^M \int_0^T (s^m_t)^\top \gamma^m_x(t, x^u_{t-}, u_t, \alpha_{t-}) z_{t-} \mathcal{L}^i(d\zeta) dt + \mathbb{E} \sum_{m=1}^M \int_0^T (s^m_t)^\top \gamma^m_v(t, x^u_{t-}, u_t, \alpha_{t-}) v_t \tilde{N}^i dt
\]

Thus integrate and take expectation on both sides of equation (2.11); substitute on the right-hand side the calculations made above; and apply the final value (2.9) and the initial value (2.4) on the left-hand side to obtain

\[
\mathbb{E} g_3(x^u_T, \alpha_T) z_T = -\mathbb{E} \int_0^T z^\top_{t-} h^i(x^u_{t-}, u_t, \alpha_{t-}) dt
\]
that is, the ordinary-control problem, is defined as
Theorem 8.
Theorems 6 and 7.
and conclude that (2.10) holds.

Then
\[
\{ \text{the costate process corresponding to the optimal ordinary-control process} \}
\]
the solution to the state equation
Hamiltonian and optimality. They follow easily; the first from (1.9) and (2.9), and second from
It is clear from (1.5), (1.6), and (2.9) that (2.12) is well defined. The following theorems make clear
process satisfying (2.10) to complete the proof.

Apply the Riesz Representation Theorem to deduce that the solution to (2.9) is the only such
Let \( p \in \mathbb{R}^{N}, \ q \in \mathbb{R}^{N \times D}, \ r \in \mathbb{R}^{N \times L}, \) and \( s \in \mathbb{R}^{N \times M}. \) The Hamiltonian associated with (1.9) and
(1.10), that is, the ordinary-control problem, is defined as
\[
H(t, x, v, i, p, q, r, s) := h(t, x, v, i) + p^\top b(t, x, v, i)
\]
\[
+ \sum_{d=1}^{D} (q^{ad})^\top \sigma^{ad}(t, x, v, i)
\]
\[
+ \sum_{l=1}^{L} \int_{E_{0}} (r^{sl})^\top (t, \zeta) \gamma^{sl}(t, x, v, i, \zeta) \lambda^l(d\zeta)
\]
\[
+ \sum_{m=1}^{M} (s^{sm})^\top \eta^{sm}(t, x, v, i) G^{im} 1_{i \neq m}
\]
It is clear from (1.5), (1.6), and (2.9) that (2.12) is well defined. The following theorems make clear
the conjugate nature of the state and costate processes as well as the relationship between the
Hamiltonian and optimality. They follow easily; the first from (1.9) and (2.9), and second from
Theorems 6 and 7.

Theorem 8. Let \( \{ u_{t} \} \) be an optimal ordinary-control process, that is, a solution to problem (1.11); \( \{ x^{u}_{t} \} \)
the solution to the state equation (1.9) with ordinary-control process \( \{ u_{t} \}; \) and \( \{ (p_{t-}, q_{t}, r(t), s_{t}) \} \)
the costate process corresponding to the optimal ordinary-control process \( \{ u_{t} \}, \) as defined in (2.9).
Then \( \mathbb{P} - a.s. \)
\[
dx^{u}_{t} = H_{p}^\top(t, x^{u}_{t-}, u_{t}, \alpha_{t-}, p_{t-}, q_{t}, r(t_{-}), s_{t_{-}}) dt + \sum_{d=1}^{D} \sigma^{ad}(t, x^{u}_{t-}, u_{t}, \alpha_{t-}) dB^{d}_{t}
\]
\[
+ \sum_{l=1}^{L} \int_{E_{0}} \gamma^{sl}(t, x^{u}_{t-}, u_{t}, \alpha_{t-}, \zeta) \tilde{N}^{l}(dr, d\zeta)
\]
+ \sum_{m=1}^{M} \eta^{*m}(t, x_{t-}, u_t, \alpha_{t-}) d\tilde{\gamma}^{m}_t
\end{align*}

\begin{align*}
dp_t &= -H_x(t, x_{t-}, u_t, \alpha_{t-}, p_{t-}, q_t, r(t, \cdot), s_t) dt + \sum_{d=1}^{D} q_{t-}^d dB^d_t \\
&\quad + \sum_{l=1}^{L} \int_{E_0} r^{*l}(t, \zeta) \tilde{N}^l(dt, d\zeta) + \sum_{m=1}^{M} s_{t-}^{*m} d\tilde{\gamma}^{m}_t
\end{align*}

\begin{align*}
x_{0}^u &= \xi_0 \\
p_T &= g_x^{T}(x_{T}, \alpha_{T})
\end{align*}

**Theorem 9.** Let \( \{u_t\} \) be an optimal ordinary-control process, that is, a solution to problem (1.11); \( \{x_t^u\} \) the solution to the state equation (1.9) with ordinary-control process \( \{u_t\} \); and \( \{(p_{t-}, q_t, r(t, \cdot), s_t)\} \) the costate process corresponding to the optimal ordinary-control process \( \{u_t\} \), as defined in (2.9). Then

\[ E \int_0^T H_v(t, x_{t-}^u, u_t, \alpha_{t-}, p_{t-}, q_t, r(t, \cdot), s_t)(v_t - u_t) dt \geq 0 \]

for any ordinary-control process \( \{v_t\} \).

We now state and prove the first of our necessary characterizations of an optimal control. Its deterministic analog is known in the literature as the Pontryagin Minimum Principle. It is in essence the localization of Theorem 9.

**Theorem 10.** Let \( \{u_t\} \) be an optimal ordinary-control process, that is, a solution to problem (1.11); \( \{x_t^u\} \) the solution to the state equation (1.9) with ordinary-control process \( \{u_t\} \); and \( \{(p_{t-}, q_t, r(t, \cdot), s_t)\} \) the costate process corresponding to the optimal ordinary-control process \( \{u_t\} \), as defined in (2.9). Then

\[ H_v(t, x_{t-}^u, u_t, \alpha_{t-}, p_{t-}, q_t, r(t, \cdot), s_t)(v - u_t) \geq 0 \]

for any ordinary control \( v \), where the inequality is understood to hold for all \((t, \omega)\) in a set of full \((dt \otimes \mathcal{P})\)-measure.

**Proof.** Let

\[ X_t := H_v(t, x_{t-}^u, u_t, \alpha_{t-}, p_{t-}, q_t, r(t, \cdot), s_t)(v - u_t) \]

\[ \Lambda := \{(t, \omega) : X_t < 0\} \]

Apply (1.5) and (1.6) to reason that the partial derivative with respect to \( v \) of (2.12) exists and is continuous, and thus deduce that \( \{X_t\} \) is a predictable process. Denote by \( \Lambda_t \) the \( t \)-section of \( \Lambda \);
clearly \( \{1_{\Lambda_t}\} \) is a predictable process.

Let

\[
v_t := (1 - 1_{\Lambda_t}) u_t + 1_{\Lambda_t} v
\]

and note that this is a predictable \( U \)-valued process, hence an ordinary control. Apply Theorem 9 and the Fubini Theorem to get

\[
0 \leq \mathbb{E} \int_0^T H_{\nu}(t, x_t^u, u_t, \alpha_t, p_t, q_t, r(t, \cdot), s_t)(v_t - u_t) dt
\]

\[
= \int_{\Lambda} \text{X}_{\cdot} (dt \otimes \mathbb{P})
\]

\[
\leq 0
\]

This implies that \( \Lambda \) has \( (dt \otimes \mathbb{P}) \)-measure 0 and the proof is complete. \( \square \)

We remark that there is another version of the Pontryagin Minimum Principle. Under the assumptions and notation of Theorem 10 it can be shown that

\[
H(t, x_{t-}^u, u_t, \alpha_{t-}, p_{t-}, q_t, r(t, \cdot), s_t) \leq H(t, x_{t-}^u, v, \alpha_{t-}, p_{t-}, q_t, r(t, \cdot), s_t)
\]

The version in Theorem 10 is a consequence of how the perturbation (2.1) was defined. Since we follow Bensoussan (1988) in the definition of the perturbation, we arrive at the stated version of the Pontryagin Minimum Principle.

### 2.3 Bellman's Dynamic Programming Principle

Let \( \{v_t\} \) be an ordinary-control process, \( s \in [0, T), \xi \in \mathbb{R}^N \), and \( i \) a regime. By \( \mathbb{E}^\xi[i] \cdot \) we mean the conditional-expectation operator with conditions given \( \mathbb{P} - a.s. \) by \( x_s^\nu = \xi \) and \( \alpha_s = i \). In view of (1.10) and (1.11) the cost-to-go functional is defined as

\[
J(s, \xi, \{v_t\}, i) := \mathbb{E}^\xi[i] \left[ \int_s^T h(t, x_t^{\nu}, v_t, \alpha_{t-}) dt + g(x_T^\nu, \alpha_T) \right]
\]

and value function is defined as

\[
V(s, \xi, i) := \inf_{\{v_t\} \in \mathcal{O}} J(s, \xi, \{v_t\}, i)
\]

We introduce several terms from the study of Markov evolution systems. Unfortunately a rigorous
approach would take us too far afield. We proceed informally and direct the interested reader to Applebaum (2009) or Kallenberg (2002) for more details.

Consider the Markov process \( \{(x_t^v, \alpha_t)\} \). We associate with this process a two-parameter family of linear operators denoted \( \{\mathcal{E}^v_{s,t}\}_{t \geq s} \) and defined as

\[
(\mathcal{E}^v_{s,t} F)(s, \xi, i) := \mathbb{E}^{s \xi, i} \left[ F(t, x_t^v, \alpha_t) \right]
\]

with \( F \) any element of a sufficiently nice Banach space of functions of the form \([0, T] \times \mathbb{R}^N \times I \to \mathbb{R} \). We call \( \{\mathcal{E}^v_{s,t}\} \) the Markov evolution system associated with the Markov process \( \{(x_t^v, \alpha_t)\} \). Its infinitesimal generator \( \{\mathcal{A}^v_s\} \) is the one-parameter family of linear operators defined as

\[
(\mathcal{A}^v_s F)(s, \xi, i) := \lim_{\tau \downarrow s} \frac{\left( \mathcal{E}^v_{s,\tau} F \right)(s, \xi, i) - F(s, \xi, i)}{\tau - s}
\]

We remark that \( \{\mathcal{E}^v_{s,t}\} \) is a two-parameter family and \( \{\mathcal{A}^v_s\} \) is a one-parameter family the parameter because the state process \( \{x_t^v\} \) is a time-inhomogeneous Markov process. By an application of the Itô Formula it can be shown that

\[
(\mathcal{A}^v_s F)(s, \xi, i) = F_t(s, \xi, i) + (\mathcal{L}^v_s F)(s, \xi, i)
\]  

(2.15)

where \( \{\mathcal{L}^v_s\} \) is the one-parameter family of linear operators that characterizes the action in the state and regime-switching processes and is given by

\[
(\mathcal{L}^v_s F)(s, \xi, i) = F_z(s, \xi, i) b(s, \xi, v, i) + \frac{1}{2} \sum_{d=1}^D \sum_{n_1=1}^N \sum_{m_1=1}^N F_{x_{n_1} x_{n_2}}(s, \xi, i) (\sigma^{n_1 d} \sigma^{n_2 d})(s, \xi, v, i) \\
+ \sum_{l=1}^L \int_{E_0} \left\{ F(s, \xi + \gamma^l(s, \xi, v, i, \zeta), i) - F(s, \xi, i) \\
- F_z(s, \xi, i) \gamma^l(s, \xi, v, i, \zeta) \right\} \lambda^l(d\zeta) \\
+ \sum_{m=1}^M \left\{ F(s, \xi + \eta^m(s, \xi, v, i), m) - F(s, \xi, i) \\
- F_z(s, \xi, i) \eta^m(s, \xi, v, i) \right\} G^m 1_{i \neq m}
\]  

(2.16)

Giving a full description of the domain of \( \{\mathcal{A}^v_s\} \) is a delicate matter. It is evident that a necessary condition is that \( F(\cdot, \cdot, i) \in C^{1,2}([0, T] \times \mathbb{R}^N; \mathbb{R}) \).

We remark that for many control problems the value function is not sufficiently smooth for the calculations (2.15) and (2.16), and hence the analysis that follows. In these cases, the theory of viscosity solutions is employed to obtain similar results. For more details on viscosity solutions
see Crandall et al. (1992). For the application of such solutions to optimal control see Fleming and Soner (2006).

We now state a fundamental result concerning the Markovian nature of our problem; its deterministic analog is known in the literature as the Bellman Dynamic Programming Principle. For a proof see Fleming and Soner (2006).

**Theorem 11.** Let $s \in [0, T)$, $\tau \in (s, T)$, $\xi \in \mathbb{R}^N$, and $i$ be a regime. Assume that $V$ is in the domain of the infinitesimal generator of $\{ (x^v_t, \alpha_t) \}$; in particular $V(\cdot, \cdot, i) \in C^{1,2}([0, T] \times \mathbb{R}^N; \mathbb{R})$. Then

$$V(s, \xi, i) = \inf_{\{ v_t \} \in \mathcal{A}} \mathbb{E}^{s, \xi, i} \left[ \int_s^\tau h(t, x^v_t, \alpha_t) \, dt + V(\tau, x^v_\tau, \alpha) \right]$$

Moreover if $\{ u_t \}$ is an optimal Markov ordinary-control process the infimum is attained.

Under the assumptions and notation of Theorem 11, consider the expression

$$V(\tau, x^v_\tau, \alpha_\tau) - V(s, \xi, i)$$

If $\tau$ is such that $0 < \tau - s \ll 1$ we can estimate the expression via the Itô Formula, rewriting the result of Theorem 11 as a differential equation in $V(\cdot)$. The following theorem proves this, and is the second of our necessary characterizations of an optimal control. Its deterministic analog is known in the literature as the Hamilton-Jacobi-Bellman equation.

**Theorem 12.** Let $s \in (0, T)$, $\xi \in \mathbb{R}^N$, and $i$ be a regime. Assume that $V$ is in the domain of the infinitesimal generator of $\{ (x^v_t, \alpha_t) \}$; in particular $V(\cdot, \cdot, i) \in C^{1,2}([0, T] \times \mathbb{R}^N; \mathbb{R})$. Then

$$V_t(s, \xi, i) = - \sup_{v \in \mathcal{U}} \left[ h(s, \xi, v, i) + (\mathcal{L}_s^v V)(s, \xi, i) \right]$$

$$V(T, \xi, i) = g(\xi, i)$$

Moreover if $\{ u_t \}$ is an optimal Markov ordinary-control process the infimum is attained.

**Proof.** We first show that the differential inequality holds for an arbitrary control. We next show that the infimum is attained by $\{ u_t \}$. Finally we show that the final value is satisfied.

(Step 1.) Let $0 < \epsilon \ll 1$ and consider an arbitrary control $v$. Apply Theorem 11 and rearrange terms to get

$$0 \leq \mathbb{E}^{s, \xi, i} \left[ \int_s^{s+\epsilon} h(t, x^v_t, \alpha_t) \, dt + V(s+\epsilon, x^v_{s+\epsilon}, \alpha_{s+\epsilon}) - V(s, \xi, i) \right]$$

(2.17)
apply the Itô Formula to obtain

\[
0 = -\{ V(s + \epsilon, x_{s+\epsilon}^v, \alpha_{s+\epsilon}) - V(s, \xi, t) \} \\
+ \int_s^{s+\epsilon} V_t(t, x_{t-}^v, \alpha_{t-}) dt + \int_s^{s+\epsilon} V_x(t, x_{t-}^v, \alpha_{t-}) b(t, x_{t-}^v, v, \alpha_{t-}) dt \\
+ \sum_{d=1}^D \int_s^{s+\epsilon} V_{x_d}(t, x_{t-}^v, \alpha_{t-}) \sigma^{d}(t, x_{t-}^v, v, \alpha_{t-}) dB_t^d \\
+ \sum_{l=1}^L \int_s^{s+\epsilon} \int_{E_0} V_{x_l}(t, x_{t-}^v, \alpha_{t-}) \gamma^{sl}(t, x_{t-}^v, v, \alpha_{t-}, \zeta) \tilde{N}_l(\alpha, \zeta) dt d\zeta \\
+ \sum_{m=1}^M \int_s^{s+\epsilon} V_x(t, x_{t-}^v, \alpha_{t-}) \eta^{sm}(t, x_{t-}^v, v, \alpha_{t-}) d\tilde{\gamma}_t^m \\
+ \frac{1}{2} \sum_{d=1}^D \sum_{n_1=1}^N \sum_{n_2=1}^N \int_s^{s+\epsilon} V_{x_n_1 x_{n_2}}(t, x_{t-}^v, \alpha_{t-}) \left( \sigma^{n_1d} \sigma^{n_2d} \right)(t, x_{t-}^v, v, \alpha_{t-}) dt \\
+ \sum_{l=1}^L \int_s^{s+\epsilon} \int_{E_0} \left\{ V(t, x_{t-}^v + \gamma^{sl}(t, x_{t-}^v, v, \alpha_{t-}, \zeta), \alpha_{t-}) - V(t, x_{t-}^v, \alpha_{t-}) \\
- V_x(t, x_{t-}^v, \alpha_{t-}) \gamma^{sl}(t, x_{t-}^v, v, \alpha_{t-}, \zeta) \right\} \tilde{N}_l(\alpha, \zeta) dt d\zeta \\
+ \sum_{m=1}^M \int_s^{s+\epsilon} \left\{ V(t, x_{t-}^v + \eta^{sm}(t, x_{t-}^v, v, \alpha_{t-}), m) - V(t, x_{t-}^v, \alpha_{t-}) \\
- V_x(t, x_{t-}^v, \alpha_{t-}) \eta^{sm}(t, x_{t-}^v, v, \alpha_{t-}) \right\} d\tilde{\gamma}_t^m
\]

Use (1.1) and (1.4) to produce

\[
0 = -\{ V(s + \epsilon, x_{s+\epsilon}^v, \alpha_{s+\epsilon}) - V(s, \xi, t) \} \\
+ \int_s^{s+\epsilon} V_t(t, x_{t-}^v, \alpha_{t-}) dt + \int_s^{s+\epsilon} V_x(t, x_{t-}^v, \alpha_{t-}) b(t, x_{t-}^v, v, \alpha_{t-}) dt \\
+ \sum_{d=1}^D \int_s^{s+\epsilon} V_{x_d}(t, x_{t-}^v, \alpha_{t-}) \sigma^{d}(t, x_{t-}^v, v, \alpha_{t-}) dB_t^d \\
+ \sum_{l=1}^L \int_s^{s+\epsilon} \int_{E_0} V_{x_l}(t, x_{t-}^v, \alpha_{t-}) \gamma^{sl}(t, x_{t-}^v, v, \alpha_{t-}, \zeta) \tilde{N}_l(\alpha, \zeta) dt d\zeta \\
+ \sum_{m=1}^M \int_s^{s+\epsilon} V_x(t, x_{t-}^v, \alpha_{t-}) \eta^{sm}(t, x_{t-}^v, v, \alpha_{t-}) d\tilde{\gamma}_t^m \\
+ \frac{1}{2} \sum_{d=1}^D \sum_{n_1=1}^N \sum_{n_2=1}^N \int_s^{s+\epsilon} V_{x_n_1 x_{n_2}}(t, x_{t-}^v, \alpha_{t-}) \left( \sigma^{n_1d} \sigma^{n_2d} \right)(t, x_{t-}^v, v, \alpha_{t-}) dt 
\]
+ \sum_{l=1}^{L} \int_{s}^{s+\varepsilon} \int_{E_0} \left\{ V(t, x^v_t + \gamma^* t(t, x^v_{t-}, v, \alpha_{t-}, \zeta), \alpha_{t-}) - V(t, x^v_t, \alpha_{t-}) \\
- V_5(t, x^v_t, \alpha_{t-}) \gamma^* t(t, x^v_{t-}, v, \alpha_{t-}, \zeta) \right\} \tilde{N}_l^t (dt, d\zeta)
+ \sum_{l=1}^{L} \int_{s}^{s+\varepsilon} \int_{E_0} \left\{ V(t, x^v_t + \gamma^* t(t, x^v_{t-}, v, \alpha_{t-}, \zeta), \alpha_{t-}) - V(t, x^v_t, \alpha_{t-}) \\
- V_5(t, x^v_t, \alpha_{t-}) \gamma^* t(t, x^v_{t-}, v, \alpha_{t-}, \zeta) \right\} \lambda^t (d\zeta) dt
+ \sum_{m=1}^{M} \int_{s}^{s+\varepsilon} \int_{E_0} \left\{ V(t, x^v_t + \eta^* m(t, x^v_{t-}, v, \alpha_{t-}), m) - V(t, x^v_t, \alpha_{t-}) \\
- V_5(t, x^v_t, \alpha_{t-}) \eta^* m(t, x^v_{t-}, v, \alpha_{t-}) \right\} \tilde{Y}_m^t dt
+ \sum_{m=1}^{M} \int_{s}^{s+\varepsilon} \int_{E_0} \left\{ V(t, x^v_t + \eta^* m(t, x^v_{t-}, v, \alpha_{t-}), m) - V(t, x^v_t, \alpha_{t-}) \\
- V_5(t, x^v_t, \alpha_{t-}) \eta^* m(t, x^v_{t-}, v, \alpha_{t-}) \right\} G^{a_{t-}-m} 1_{[\alpha_{t-} \neq m]} dt

Substitute this estimate into (2.17) to get

\begin{align*}
0 & \leq \mathbb{E}^x \int_{s}^{s+\varepsilon} h(t, x^v_t, v, \alpha_{t-}) dt + \mathbb{E}^x \int_{s}^{s+\varepsilon} V(t, x^v_t, \alpha_{t-}) dt \\
& + \mathbb{E}^x \int_{s}^{s+\varepsilon} V_5(t, x^v_t, \alpha_{t-}) b(t, x^v_{t-}, v, \alpha_{t-}) dt \\
& + \frac{1}{2} \mathbb{E}^x \sum_{d=1}^{D} \sum_{n_1=1}^{N_1} \sum_{n_2=1}^{N_2} \int_{s}^{s+\varepsilon} V_{x_{n_1} x_{n_2}} (t, x^v_t, \alpha_{t-}) \\
& \quad \cdot (s_n^d, s_{n_2}^d) (t, x^v_{t-}, v, \alpha_{t-}) dt \\
& + \mathbb{E}^x \sum_{l=1}^{L} \int_{s}^{s+\varepsilon} \int_{E_0} \left\{ V(t, x^v_t + \gamma^* l(t, x^v_{t-}, v, \alpha_{t-}, \zeta), \alpha_{t-}) - V(t, x^v_t, \alpha_{t-}) \\
- V_5(t, x^v_t, \alpha_{t-}) \gamma^* l(t, x^v_{t-}, v, \alpha_{t-}, \zeta) \right\} \lambda^t (d\zeta) dt \\
& + \mathbb{E}^x \sum_{m=1}^{M} \int_{s}^{s+\varepsilon} \int_{E_0} \left\{ V(t, x^v_t + \eta^* m(t, x^v_{t-}, v, \alpha_{t-}), m) - V(t, x^v_t, \alpha_{t-}) \\
- V_5(t, x^v_t, \alpha_{t-}) \eta^* m(t, x^v_{t-}, v, \alpha_{t-}) \right\} G^{a_{t-}-m} 1_{[\alpha_{t-} \neq m]} dt
\end{align*}

Apply the Fubini Theorem and divide both sides by \( \varepsilon \) to produce

\begin{align*}
0 & \leq \frac{1}{\varepsilon} \int_{s}^{s+\varepsilon} \mathbb{E}^x h(t, x^v_t, v, \alpha_{t-}) dt + \frac{1}{\varepsilon} \int_{s}^{s+\varepsilon} \mathbb{E}^x V(t, x^v_t, \alpha_{t-}) dt
\end{align*}
Under the assumptions and notation of Theorem 7 we rewrite the costate equation (2.9) component-wise as

\[ dp_t^n = -h_{x_n}(t, x_{t-}^u, u_t, \alpha_{t-}) dt - b_{x_n}^\top(t, x_{t-}^u, u_t, \alpha_{t-}) p_{t-} dt \]
We remark that in what follows we shall write the partials of $V$ rather compactly. For example we shall in this proof generally denote the optimal Markov ordinary-control process by

$$(1.11) \quad V(t, x_t^u, u_t, \alpha_{t-})$$

produce $V_j(t, x_t^u, u_t, \alpha_{t-})$. Similarly $V_j(t, x_t^u, u_t, \alpha_{t-})$ means that we first take the partial with respect to $x$ and then the partial with respect to $t$; the result is an $\mathbb{R}$-valued function. $V_{x, t}$ means that we first take the partial with respect to the $j^{th}$ component of $x$ and then the gradient with respect to $x$; the result is an $\mathbb{R}^{1 \times N}$-valued function. Similarly $V_{xx, j}$ means that we first take the gradient with respect to $x$ and then the partial with respect to the $j^{th}$ component of $x$ of each of the components of the $x$-gradient; the result is also an $\mathbb{R}^{1 \times N}$-valued function.

**Theorem 13.** Let $\{u_t\}$ be an optimal Markov ordinary-control process, that is, a solution to problem (1.11); $\{x_t^u\}$ the solution to the state equation (1.9) with Markov ordinary-control process $\{u_t\}$; and $\{(p_t, q_t, r(t, \cdot), s_t)\}$ the costate process corresponding to the optimal Markov ordinary-control process $\{u_t\}$. Assume that $V$ is in the domain of the infinitesimal generator of $\{(x_t^u, \alpha_t)\}$ and moreover that $V(\cdot, \cdot, i) \in C^{1,2}(\{0, T\} \times \mathbb{R}^N; \mathbb{R})$, where $i$ is any regime. Then

$$p^n_{t-} = V_{x_n}(t, x_t^u, \alpha_{t-})$$

$$q^{nd}_{t-} = V_{x_n x}(t, x_t^u, \alpha_{t-}) \sigma^{sd}(t, x_t^u, u_t, \alpha_{t-})$$

$$r^{nl}(t, \zeta) = V_{x_n}(t, x_t^u + \gamma^{sl}(t, x_t^u, u_t, \alpha_{t-}, \zeta), \alpha_{t-}) - V_{x_n}(t, x_t^u, \alpha_{t-})$$

$$s^{nm}_{t-} = V_{x_n}(t, x_t^u + \eta^{sm}(t, x_t^u, u_t, \alpha_{t-}), m) - V_{x_n}(t, x_t^u, \alpha_{t-})$$

**Proof.** We first compute the differential of $V_{x_j}(\cdot)$. We next find an alternative expression for $V_{x_j}(\cdot)$. Finally we compare our calculation to (2.19). We remark that though $\{u_t\}$ has representation (1.2) we shall in this proof generally denote the optimal Markov ordinary-control process by $\{u_t\}$.

(Step 1) Let $j = 1, 2, \ldots, N; e_j \in \mathbb{R}^N$ be the vector such that $e_j^n = 1$ if $j = n$, else 0; and $X^n_j := V_{x_j}(t, x_t^u, \alpha_{t-})$. Take the differential of $X^n_j$ and apply the Itô Formula on the right-hand side to produce

$$dX^n_j = V_{x_j}(t, x_t^u, \alpha_{t-})dt + V_{x_j x}(t, x_t^u, \alpha_{t-})b(t, x_t^u, u_t, \alpha_{t-})dt$$
Use (1.1) and (1.4) to obtain

\[ dX_t^j = V_{x,t}(t, x_{t-}^u, \alpha_{t-}) dt + V_{x,t}(t, x_{t-}^u, \alpha_{t-}) dB_t^d + \sum_{d=1}^{D} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \sigma^{ad}(t, x_{t-}^u, u_t, \alpha_{t-}) dB_t^{a} \\
+ \sum_{l=1}^{L} \int_{E_0} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \gamma^{al}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) \tilde{N}_t^{l}(dt, d\zeta) \\
+ \sum_{m=1}^{M} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \eta^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}) d\tilde{Y}_t^{m} \\
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_1=1}^{N} \sum_{n_2=1}^{N} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) (\sigma^{n_1 d} \sigma^{n_2 d})(t, x_{t-}^u, u_t, \alpha_{t-}) dt \]

\[ + \sum_{l=1}^{L} \int_{E_0} \left\{ V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \gamma^{al}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) - V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \right\} \tilde{N}_t^{l}(dt, d\zeta) \\
+ \sum_{m=1}^{M} \left\{ V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \eta^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}) - V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \right\} d\tilde{Y}_t^{m} \]

(2.20)

Use (1.1) and (1.4) to obtain

\[ dX_t^j = V_{x,t}(t, x_{t-}^u, \alpha_{t-}) dt + V_{x,t}(t, x_{t-}^u, \alpha_{t-}) b(t, x_{t-}^u, u_t, \alpha_{t-}) dt \\
+ \sum_{d=1}^{D} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \sigma^{ad}(t, x_{t-}^u, u_t, \alpha_{t-}) dB_t^d \\
+ \sum_{l=1}^{L} \int_{E_0} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \gamma^{al}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) \tilde{N}_t^{l}(dt, d\zeta) \\
+ \sum_{m=1}^{M} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \eta^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}) d\tilde{Y}_t^{m} \\
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_1=1}^{N} \sum_{n_2=1}^{N} V_{x,t}(t, x_{t-}^u, \alpha_{t-}) (\sigma^{n_1 d} \sigma^{n_2 d})(t, x_{t-}^u, u_t, \alpha_{t-}) dt \]

\[ + \sum_{l=1}^{L} \int_{E_0} \left\{ V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \gamma^{al}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) - V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \right\} \tilde{N}_t^{l}(dt, d\zeta) \\
+ \sum_{m=1}^{M} \left\{ V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \eta^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}) - V_{x,t}(t, x_{t-}^u, \alpha_{t-}) \right\} d\tilde{Y}_t^{m} \]

\[ + \lambda(t, x_{t-}^u, \alpha_{t-}) \eta^{sm}(t, x_{t-}^u, u_t, \alpha_{t-}, \zeta) d\tilde{Y}_t^{m} \]

58
(Step 2.) Apply Theorem 12 to reason

\[ V_{t}(t, x_{t_{-}}^{u}, a_{t_{-}}) + h(t, x_{t_{-}}^{u}, a_{t_{-}}) + (\mathcal{L}_{t} V)(t, x_{t_{-}}^{u}, a_{t_{-}}) = 0 \]

Take the partial derivative with respect to \( x_{j} \) on both sides to get

\[
0 = V_{t}x_{j}(t, x_{t_{-}}^{u}, a_{t_{-}}) + h_{j}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}) + V_{xx_{j}}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}})b(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}) + V_{x}(t, x_{t_{-}}^{u}, a_{t_{-}})b_{x}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}})
\]

\[
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_{1}=1}^{N} \sum_{n_{2}=1}^{N} V_{x_{n_{1}}x_{n_{2}}x_{j}}(t, x_{t_{-}}^{u}, a_{t_{-}})(\sigma_{n_{1}d}^{m} \sigma_{n_{2}d}^{m})_{t}V_{x}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}})
\]

\[
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_{1}=1}^{N} \sum_{n_{2}=1}^{N} V_{x_{n_{1}}x_{n_{2}}x_{j}}(t, x_{t_{-}}^{u}, a_{t_{-}})(\sigma_{n_{1}d}^{m} \sigma_{n_{2}d}^{m})_{t}V_{x}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}})
\]

\[
+ \sum_{l=1}^{L} \int_{E_{0}} \left\{ V_{x}(t, x_{t_{-}}^{u} + \gamma^{\mu l}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}, \zeta), a_{t_{-}})
\]

\[
\left(e_{j} + \gamma_{x_{j}}^{\mu l}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}, \zeta)\right)
\]

\[
- V_{x_{j}}(t, x_{t_{-}}^{u}, a_{t_{-}})
\]

\[
- V_{xx_{j}}(t, x_{t_{-}}^{u}, a_{t_{-}})\gamma_{x_{j}}^{\mu l}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}, \zeta)
\]

\[
- V_{x}(t, x_{t_{-}}^{u}, a_{t_{-}})\gamma_{x_{j}}^{\mu l}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}, \zeta)
\]

\[
\lambda^{l}(d\zeta)
\}

\[
+ \sum_{m=1}^{M} \left\{ V_{x}(t, x_{t_{-}}^{u} + \eta^{m}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}), m)(e_{j} + \eta_{x_{j}}^{m}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}}))
\]

\[
- V_{x_{j}}(t, x_{t_{-}}^{u}, a_{t_{-}})
\]

\[
- V_{xx_{j}}(t, x_{t_{-}}^{u}, a_{t_{-}})\eta^{m}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}})
\]

\[
- V_{x}(t, x_{t_{-}}^{u}, a_{t_{-}})\eta_{x_{j}}^{m}(t, x_{t_{-}}^{u}, u_{t}, a_{t_{-}})
\]

\[
\} G^{a_{t_{-}}^{m} 1_{a_{t_{-}} = m}} d t
\]
Solve for the first term on the right-hand side and substitute into (2.20) to obtain

\[
\begin{align*}
\mathrm{d}X^j_t &= -h_{x_j}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t - V_{x_j}(t, x^u_{t^-}, \alpha_{t^-}) b_{x_j}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t \\
&\quad + \sum_{d=1}^D V_{x_j}(t, x^u_{t^-}, \alpha_{t^-}) \sigma^{x_d}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}B^d_t \\
&- \frac{1}{2} \sum_{d=1}^D \sum_{n_1=1}^N \sum_{n_2=1}^N V_{x_{n_1}x_{n_2}}(t, x^u_{t^-}, \alpha_{t^-}) (\sigma^{n_1d}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t \\
&\quad + \sum_{l=1}^L \int_{E_0} \left\{ V_{x_j}(t, x^u_{t^-} + \gamma^{sl}(t, x^u_{t^-}, u_t, \alpha_{t^-}, \zeta), \alpha_{t^-}) - V_{x_j}(t, x^u_{t^-}, \alpha_{t^-}) \right\} \\
&\quad \tilde{N}^l(\mathrm{d}t, \mathrm{d}\zeta) \\
&\quad - \sum_{l=1}^L \int_{E_0} \left\{ V_{x_j}(t, x^u_{t^-} + \gamma^{sl}(t, x^u_{t^-}, u_t, \alpha_{t^-}, \zeta), \alpha_{t^-}) - V_{x_j}(t, x^u_{t^-}, \alpha_{t^-}) \right\} \\
&\quad \lambda^l(\mathrm{d}\zeta) \mathrm{d}t \\
&\quad + \sum_{m=1}^M \left\{ V_{x_j}(t, x^u_{t^-} + \eta^{sm}(t, x^u_{t^-}, u_t, \alpha_{t^-}, m) - V_{x_j}(t, x^u_{t^-}, \alpha_{t^-}) \right\} \mathrm{d}\tilde{N}^m_t \\
&\quad - \sum_{m=1}^M \left\{ V_{x}(t, x^u_{t^-} + \eta^{sm}(t, x^u_{t^-}, u_t, \alpha_{t^-}, m) - V_{x}(t, x^u_{t^-}, \alpha_{t^-}) \right\} \\
&\quad \eta^{sm}_{x_j}(t, x^u_{t^-}, u_t, \alpha_{t^-}) G^{s_{m-1}(a_{t^-}) \neq m} \mathrm{d}t \\
\end{align*}
\]

(Step 3.) Add the following equation

\[
0 = \frac{1}{2} \sum_{d=1}^D \sum_{n_1=1}^N \sum_{n_2=1}^N V_{x_{n_1}x_{n_2}}(t, x^u_{t^-}, \alpha_{t^-}) (\sigma^{n_1d}, \sigma^{n_2d})_{x_j}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t \\
- \sum_{d=1}^D \sum_{n_1=1}^N \sum_{n_2=1}^N V_{x_{n_1}x_{n_2}}(t, x^u_{t^-}, \alpha_{t^-}) (\sigma^{n_2d}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t
\]

and rearrange terms to get

\[
\begin{align*}
\mathrm{d}X^j_t &= -h_{x_j}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t - V_{x_j}(t, x^u_{t^-}, \alpha_{t^-}) b_{x_j}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t \\
&\quad - \sum_{d=1}^D \sum_{n_1=1}^N \sum_{n_2=1}^N V_{x_{n_1}x_{n_2}}(t, x^u_{t^-}, \alpha_{t^-}) (\sigma^{n_2d}(t, x^u_{t^-}, u_t, \alpha_{t^-}) \mathrm{d}t \\
&\quad - \sum_{l=1}^L \int_{E_0} \left\{ V_{x}(t, x^u_{t^-} + \gamma^{sl}(t, x^u_{t^-}, u_t, \alpha_{t^-}, \zeta), \alpha_{t^-}) - V_{x}(t, x^u_{t^-}, \alpha_{t^-}) \right\} \\
&\quad \lambda^l(\mathrm{d}\zeta) \mathrm{d}t
\end{align*}
\]
\[- \sum_{m=1}^{M} \{ V_x(t, x_t^u + \eta^m(t, x_t^u, u_t, \alpha_t), m) - V_x(t, x_t^u, \alpha_t) \}\]

\[ \eta^m_{x_j}(t, x_t^u, u_t, \alpha_t) G_{a_{t,-}}^m 1_{a_t \neq m} dt \]

\[ + \sum_{d=1}^{D} V_{x_j}(t, x_t^u, \alpha_t) G_{a_{t,-}}^d(t, x_t^u, u_t, \alpha_t) dB_t^d \]

\[ + \sum_{l=1}^{L} \int_{E_0} \left\{ V_{x_j}(t, x_t^u + \eta^l(t, x_t^u, u_t, \alpha_t, \zeta), \alpha_t) - V_{x_j}(t, x_t^u, \alpha_t) \right\} \tilde{N}^l(dt, d\zeta) \]

\[ + \sum_{m=1}^{M} \{ V_{x_j}(t, x_t^u + \eta^m(t, x_t^u, u_t, \alpha_t), m) - V_{x_j}(t, x_t^u, \alpha_t) \} \tilde{Y}_t^m \]

Apply Theorem 12 again to produce

\[ X_j^T = V_{x_j}(T, x_T^u, \alpha_T) = g_{x_j}(x_T^u, \alpha_T) \]

and compare to (2.19) to complete the proof. \( \square \)
In chapter 1 we introduced the optimal-control problem, and proved a couple of existence theorems. In chapter 2 we enriched our the theory of optimal control by extending the two standard necessary conditions for optimality, that is, we proved the Pontryagin Minimum Principle and the Bellman Dynamic Programming Principle held for our optimal-control problem. We should now like to see our work in action. To that end, in this final chapter we provide an alternative approach to the problem of valuing and risk-managing financial derivatives.

Our plan for the present chapter is as follows. In section 3.1 we briefly sketch a view of financial markets. Our aim is to provide the reader not already familiar with financial markets enough background knowledge to appreciate our application. In section 3.2 we give an intuitive and informal derivation of the Black-Scholes pricing equation. In section 3.3 we state our practical problem. Our intention is not to develop a new theory of quantitative finance, nor is it to establish and study a new kind of option. Rather, our intention is to show that given a slightly different context the problem of valuing and risk-managing options can be cast as an optimal-control problem. In section 3.4 we give a computational procedure for solving the Hamilton-Jacobi-Bellman equation associated with our practical problem. To this end, we discuss briefly cubic splines, the transformation of partial differential equations into systems of ordinary differential equations, and the Runge-Kutta algorithm. Finally, in section 3.5 we provide a few results from our implementation.
Note that while we give the details of our computational procedure herein, we relegate to the appendices all listings of computer code and Jupyter (formerly IPython) notebooks. In appendix A we provide all code written by us, that is, Python modules for building cubic splines and for solving the Hamilton-Jacobi-Bellman equation associated with our practical problem. In appendix B we provide the Jupyter notebooks used in the configuration and validation of our computer code. (The notebooks are given in Markdown format, so require some minor editing to run in Jupyter.)

The literature for quantitative finance is quite extensive. As the task at hand requires only a basic knowledge of financial markets and the mathematical tools used therein, we mention only a few key references. We used Etheridge (2002) as our foundation for quantitative finance. As a sometime supplement, we also used Shreve (2004). Both books cover the same material, but differ in style and emphasis. We note now that our derivation in section 3.2 follows Bergomi (2016), which, though more advanced and focused specifically on the problem of stochastic volatility, nevertheless provides a very concise introduction to the Black-Scholes pricing equation.

Finally, we remind the reader that all computations were performed on an Apple iMac. The machine contained a 3.1 GHz 6-Core (12 logical cores) Intel Core i5 CPU with 12 MB L3 cache, and 16 GB 2667 MHz DDR4 of RAM. The software stack was as follows: Apple macOS Ventura (version 13.0.1), Python 3.9.13, Intel's Math Kernel Library (MKL) 2021.4.0, NumPy 1.21.5, SciPy 1.9.1, and JupyterLab 3.4.4.

3.1 A Précis of Financial Markets

Our practical problem is to value and risk manage an option using the tools developed in the previous chapters. What does that mean? What exactly is our job? To answer those questions, we must begin with a slew of definitions.

An asset is any resource controlled by a party as the result of past transactions and from which economic benefit is expected to flow. A financial asset is any nonphysical asset with value derived from a contractual claim. A security is a tradable financial asset. The basic securities in the market are stocks, that is, equity shares in corporations; and bonds, that is, tradable loans issued by individuals, corporations, governments, and supranationals. A financial instrument is any arrangement between parties that gives rise to combinations of financial assets, liabilities, and ownership.

A financial derivative is a financial instrument with value determined by market observable referents such basic securities prices, exchange prices on currencies (for example Japanese Yen, Eurozone Euro, British pound, and U.S. dollar), purchase prices on commodities (for example coffee, wheat, copper, and zinc), or printed indices (for example Term SOFR for U.S. dollar interest rates, and the Bureau of Labor Statistics's Consumer Price Index for U.S. inflation). Financial derivatives are classified as either linear or nonlinear. The former covers financial derivatives with payoffs that are linear in the underlyings, that is, linear in the market observable referents. The canonical
example of a linear financial derivative is a stock forward: an agreement today to buy or sell at a future date a stock for a price agreed to today. This case also covers financial derivatives with payoffs that are nearly linear, which often arise in cases with collateral or nonstandard discounting factors, such as exchange-traded futures.

Nonlinear financial derivatives are financial derivatives with payoffs that are clearly nonlinear. For our purposes, nonlinear financial derivatives are synonymous with options, or at the very least are financial derivatives that include some optionality, which is to say, asymmetry in the payoff. An option gives its holder the right, but not the obligation, to buy or sell a financial asset (or possibly another financial derivative) on a date and at a price determined at option inception. The standard approach to valuing and risk-managing options is to derive a pricing partial differential equation. The Black-Scholes pricing equation, which we discuss briefly in section 3.2, is the canonical pricing equation. Options are classified as either linear or nonlinear, depending on whether the corresponding pricing equations are linear or nonlinear. As we shall see, the Black-Scholes pricing equation is a linear partial differential equation. On the other hand, the Hamilton-Jacobi-Bellman equation is in the main nonlinear. That is, given a nonlinear option, its (necessarily) nonlinear pricing equation can often be identified with a nonlinear Hamilton-Jacobi-Bellman equation from some optimal-control problem. Guyon and Henry-Labordere (2014) explores this approach at length. (This is similar in spirit to how in certain cases a partial differential equation can be identified, via the Feynman–Kac formula, with the conditional expectations of an Itô diffusion given by a stochastic differential equation. In many of these cases there are, per Bellman's Curse of Dimensionality, efficiencies in computation when using Monte Carlo simulation.)

The parties to a financial instrument may be individuals, corporations, governments, and supranationals. As a general rule the parties do not find each other. Financial institutions are corporations that provide services as intermediaries in financial markets. There are several kinds of financial institutions: depository, contractual, and investment. Examples of depository financial institutions are banks and mortgage companies, that is, financial institutions that accept deposits and create credit. Examples of contractual financial institutions are insurance companies and pension funds. Examples of investment financial institutions are brokers and investment banks.

Some clarification is in order regarding banks and investment banks. Banks are classified according to the services that are provided. A retail bank accepts deposits and creates credit for the general public. Private banks provide additional services such as asset management, brokerages, and tax planning, for high net-worth individuals. Commercial and corporate banks provide additional services to small and medium, and large corporations, governments, and supranationals, respectively. Such services include cash management, treasury management, payment processing, and electronic transfers.

Investment banks are investment financial institutions that engage in advisory-based financial transactions on behalf of individuals, corporations, governments, and supranationals. The activities
of an investment bank are divided between buy-side services and sell-side services. In the former, the investment bank advises in mergers and acquisitions, underwrites new issuances of stock and bonds, and sells buy-sell-hold research. In the latter, the investment bank supports clients of the buy side through sales and trading of currencies, commodities, and various financial instruments such as stocks and bonds in the secondary markets, and, importantly for our purposes, financial derivatives.

Clearly, banks and investment banks are distinct financial institutions. However, some commercial banks and many corporate banks have investment-bank divisions internally. In the sequel, we will assume the perspective of a trading desk in the investment-bank division of a corporate bank. Our job will be to develop pricing equations for options traded by the desk; pricing equations that shed light on both valuation and risk-management.

3.2 A Brief Introduction to the Black-Scholes Pricing Equation

This section is self-contained, and is not essential to what follows. All terms and notation are specific to it. The reader already familiar with the Black-Scholes pricing equation may skip it.

Consider an option on stock $S$ created at time 0 that gives the holder the payoff $f(S)$ at time $T$. To be clear, in this example the payoff is a function of the stock’s price at $T$ only. It is also clear that $f$ should be nonnegative; that is, we do not consider noneconomic exercise decisions, such as those that arise when taking capital requirements or taxes into consideration. For convenience, we take $T$ as the payment date, so that option expiry and option maturity are the same date.

ASSUMPTION 1: The price of the option at time $0 \leq t \leq T$ is a function of $t$ and the prevailing (at $t$) market price of the underlying stock $S$ only. We denote this price by $P(t, S)$. We state this explicitly because this is a modeling choice! We could have included other independent variables such as inflation, or vols from the options market on one of the stock market indices, or the credit-default rate of the counterparty. As we shall see shortly, each of these would lead to a sensitivity (essentially the partial derivative of $P$ with respect to that independent variable) that would then have to be risk managed, that is, hedged. What informs our choice of $t$ and $S$ is a trading-desk choice to risk manage the option with shares of stock only. Given the payoff, we have $P(T, S) = f(S)$. Finally, both $P$ and $f$ are sufficiently regular to apply some basic theorems of calculus and to ensure that certain differential equations are well posed.

In order to determine $P$, consider how it changes over small changes in $t$ and $S$, and how the desk has chosen to risk manage the option. Let $\delta t$ denote a small change in time, say a day, and $\delta S$ the corresponding change in stock price. Let $\Delta$ denote the quantity of stock in the hedge.

The profit-and-loss (P&L) realized by the desk over $\delta t$ is then given as follows. At $t$ the desk (1) sells the option for $P(t, S)$, (2) opens a deposit account with $P(t, S)$ earning interest $r$, (3) borrows
\( \Delta S \) paying interest \( r \), and (4) buys \( \Delta \) shares of stock at \( \Delta S \). The net cashflow at \( t \) is

\[ +P(t,S) - P(t,S) + \Delta S - \Delta S \]

Notice it is 0; you don’t make money or lose money just entering into the trade. Having taken a hedged position in the option at \( t \), the desk then unwinds the hedged position at time \( t + \delta t \). Specifically, the desk (5) buys the option at \( P(t+\delta t, S + \delta S) \), (6) closes the deposit account with \( (1 + r\delta t)P(t,S) \), (7) pays off the loan at \( (1 + r\delta t)\Delta S \), and (8) sells the \( \Delta \) shares of stock at \( \Delta(S + \delta S) \). The net cashflow at \( t + \delta t \) is

\[ -P(t + \delta t, S + \delta S) + (1 + r\delta t)P(t, S) - (1 + r\delta t)\Delta S + \Delta(S + \delta S) \]

Rearranging and simplifying terms gives

\[ -[P(t + \delta t, S + \delta S) - P(t, S)] + rP(t, S)\delta t + \Delta(\delta S - r\delta t) \]  \hspace{1cm} (3.1)

As the net cashflow at \( t \) is 0, the P&L realized by the desk is entirely determined by the cashflow at \( t + \delta t \). Notice it is comprised of two pieces: the P&L of the option (which includes the interest earned on the deposit) and the P&L of the hedge.

**ASSUMPTION 2**: The market for the option, market for the hedge (that is, the stock), and the money market are all sufficiently liquid to ensure that the strategy leading to (3.1) is viable. That is, the desk can always be sure that it can enter and exit positions in the option and stock at will, and that it can make deposits and borrow funds at will.

Consider the Taylor expansion of \( P \) out to second order

\[ P(t + \delta t, S + \delta S) = P(t, S) + P_t(t, S)(\delta t) + P_S(t, S)(\delta S) + \frac{1}{2} P_{tt}(t, S)(\delta t)^2 + \frac{1}{2} P_{SS}(t, S)(\delta S)^2 + P_{tS}(t, S)(\delta t)(\delta S) \]

We should like to express \( \delta S \) and \( (\delta S)^2 \) in terms of \( \delta t \) These of course are random variables, but we can say something about the expectations over time.

**ASSUMPTION 3**: The variance of the returns of \( S \) are proportional to the time horizon. That is, \( \mathbb{E}[(\delta S)^2] \) is proportional to \( \delta t \). Indeed, let \( \sigma \) denote the constant of proportionality such that \( \mathbb{E}[(\delta S)^2] = \sigma^2 \delta t \). Note that the probability used is the actuarial probability (also known as the real-world probability or the physical probability) and not any kind of risk neutral or equivalent martingale probability.

**ASSUMPTION 4**: The desk should not systematically make money or lose money in trading this option. That is, the expectations of the P&L should be zero. This is equivalent to requirement that this be a fair game. One justification for this is to recall that there was nothing special about the
fact that the desk is selling the option; the same valuation and risk-management logic should apply regardless of whether the desk is long or short.

Thus (3.1) becomes

\[
0 = \left[ P_t(t, S)[\delta_t] + P_S(t, S)[\alpha_t] + \frac{1}{2} \sigma^2 S^2 P_{SS}(t, S)[\delta_t] \right] + r P(t, S)\delta_t + \Delta(\mathbb{E}[\delta S] - r S\delta_t)
\]

We should like to eliminate the presence of \( \mathbb{E}[\delta S] \) in the above. Clearly, setting \( \Delta = P_S(t, S) \) does this. Dividing through by \( \delta_t \), rearranging terms, and recalling the payoff, we arrive at the famed Black-Scholes pricing equation

\[
P_t(t, S) - r P(t, S) + r S P_S(t, S) = -\frac{1}{2} \sigma^2 S^2 P_{SS}(t, S)
\]

where the solution \( P(t, S) \) gives the valuation of the option and \( \Delta = P_S(t, S) \) gives the risk management of the option.

### 3.3 An Alternative Formulation Using Optimal Control

Let \( N \geq 2; \{v_t\} \) be an ordinary-control process; \( \{x_t^v\} \) be an \( \mathbb{R}^{N-1} \)-valued process; and \( \{y_t^v\} \) be an \( \mathbb{R} \)-valued process. The state process \([x_t^v, y_t^v]^T\) satisfies \( \mathbb{P} \)-a.s.

\[
d\begin{bmatrix} x_t^v \\ y_t^v \end{bmatrix} = b(t, x_t^v, y_t^v, v_t, \alpha_{t-})dt + \sum_{d=1}^{D} \sigma^d(t, x_t^v, y_t^v, v_t, \alpha_{t-})dB^d_t
\]

\[
+ \sum_{l=1}^{L} \int_{E_0} y^{il}(t, x_t^v, y_t^v, v_t, \alpha_{t-}, \zeta)\tilde{N}^l(dt, d\zeta)
\]

\[
+ \sum_{m=1}^{M} \eta^{lm}(t, x_t^v, y_t^v, v_t, \alpha_{t-})d\tilde{Y}^m_t
\]

\[
\begin{bmatrix} x_0^v \\ y_0^v \end{bmatrix} = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}
\]

where \( \hat{x} \) is fixed and \( \hat{y} \) is to be determined. We remark that for convenience we have written the state process such that it distinguishes the first \( N - 1 \) components from the last component, otherwise the functions on the right-hand side of (3.2) are exactly as given in section 1.2. In what follows, \( \{x_t^v\} \) shall denote the price process of the stocks and \( \{y_t^v\} \) the wealth process of the super-replicating portfolio (defined below).
Consider the $\mathbb{R}$-valued process given by $\mathbb{P} - a.s.$

$$h(t, x_t^v, y_t^v, v_t, \alpha_t)$$

and the random variable given by $\mathbb{P} - a.s.$

$$g(x_T^v, y_T^v, \alpha_T)$$

where again except for a small change in notation the functions $h(\cdot)$ and $g(\cdot)$ are exactly as given in section 1.2. In what follows, $h(\cdot)$ shall denote the path-dependent piece of the option’s payoff and $g(\cdot)$ shall denote the at-expiry piece of the option’s payoff. Let $\Pi(\cdot)$ denote the positive part of its real-number argument, that is

$$\Pi(z) = \max\{z, 0\}$$

for $z \in \mathbb{R}$. Then the wealth process $\{y_t^v\}$ is the wealth process of a super-replicating portfolio whenever $\mathbb{P} - a.s.$

\[
\Pi(h(t, x_t^v, y_t^v, v_t, \alpha_t) - y_t^v) = 0 \\
\Pi(g(x_T^v, y_T^v, \alpha_T) - y_T^v) = 0
\]

which in turn is equivalent to the expression

\[
\mathbb{E} \left[ \int_0^T \Pi(h(t, x_t^v, y_t^v, v_t, \alpha_t) - y_t^v)dt + \Pi(g(x_T^v, y_T^v, \alpha_T) - y_T^v) \right]
\]

being equal to zero.

For each $y \in [0, C]$ we have an optimal-control problem with state equation (3.2) and objective (3.4). The problem of determining the value and risk management of the stock option described by $h(\cdot)$ and $g(\cdot)$ is the problem of determining the least $y$ such that the optimal control produces an objective of zero. We remark that we assume if $y = C$ the objective is indeed zero; this condition will need to be checked in each application.

We should like to apply the results of chapter 2 to our practical problem. To this end we need to compute the derivative of (3.3), which does not exist in the classical sense at the origin. Our approach is to regularize (3.3), compute the classical derivative of the regularization, and then construct the generalized derivative as the limit of the classical derivative of the regularization. Unfortunately a rigorous approach would take us too far afield, so we proceed informally and direct the interested reader to Duistermaat and Kolk (2010) for more details.
Let \( \phi(\cdot) \) be defined as

\[
\phi(z) := \begin{cases} 
\frac{1}{\kappa} \exp\left\{-\frac{1}{1-z^2}\right\} & \text{if } z \in (-1, 1) \\
0 & \text{elsewise}
\end{cases}
\]

with \( \kappa > 0 \) a constant such that \( \int_{-1}^{1} \phi(z) dz = 1 \). Let \( 0 < \epsilon \ll 1 \) and \( \phi^\epsilon(\cdot) \) be defined as

\[
\phi^\epsilon(z) := \frac{1}{\epsilon} \phi\left(\frac{z}{\epsilon}\right) = \begin{cases} 
\frac{1}{\epsilon \kappa} \exp\left\{-\frac{\epsilon^2}{\epsilon^2-z^2}\right\} & \text{if } z \in (-\epsilon, \epsilon) \\
0 & \text{elsewise}
\end{cases}
\]

By repeated differentiation and taking of limit as \( z \) both decreases to \(-1\) and increases to \(1\) it is easy to deduce that both \( \phi(\cdot) \) and \( \phi^\epsilon(\cdot) \) are \( C^\infty_0(\mathbb{R}; \mathbb{R}) \). It is also clear that by using the substitution \( z = \epsilon w \) we have \( \int_{-\epsilon}^{\epsilon} \phi^\epsilon(z) dz = 1 \).

By \( (\phi^\epsilon * \Pi)(\cdot) \) we mean the convolution of \( \phi^\epsilon(\cdot) \) and \( \Pi(\cdot) \), that is, the operation defined as

\[
(\phi^\epsilon * \Pi)(z) := \int_{-\epsilon}^{\epsilon} \phi^\epsilon(w) \Pi(z-w) dw
\]

We remark that the result is \( C^\infty(\mathbb{R}; \mathbb{R}) \) and converges uniformly to \( \Pi(\cdot) \) as \( \epsilon \) decreases to zero. If \( z < -\epsilon < 0 \) we have

\[
(\phi^\epsilon * \Pi)'(z) = \int_{-\epsilon}^{\epsilon} \phi^\epsilon(w) \Pi'(z-w) dw = 0
\]

while if \( z > \epsilon > 0 \) we get

\[
(\phi^\epsilon * \Pi)'(z) = \int_{-\epsilon}^{\epsilon} \phi^\epsilon(w) \Pi'(z-w) dw = 1
\]

If \( z = 0 \) we obtain

\[
(\phi^\epsilon * \Pi)'(0) = \int_{-\epsilon}^{\epsilon} \phi^\epsilon(w) \Pi'(0-w) dw
\]

\[
= \int_{-\epsilon}^{0} \phi^\epsilon(w) \Pi'(0-w) dw + \int_{0}^{\epsilon} \phi^\epsilon(w) \Pi'(0-w) dw
\]

\[
= \frac{1}{2}
\]

where we have exploited the fact that the singularity in \( \Pi'(\cdot) \) occurs at but a single point and is
well-behaved. Let $\hat{\Pi}(\cdot)$ be defined as

$$\hat{\Pi}(z) := \begin{cases} 0 & \text{if } z < 0 \\ \frac{1}{2} & \text{if } z = 0 \\ 1 & \text{if } z > 0 \end{cases}$$

It is clear that $(\phi^\varepsilon * \Pi)'(\cdot)$ converges point-wise to $\hat{\Pi}(\cdot)$ as $\varepsilon$ decreases to zero.

Let $\mathcal{D}(\mathbb{R}; \mathbb{R})$ denote the vector space of functions $C^\infty_0(\mathbb{R}; \mathbb{R})$ endowed with the topology such that if $\psi_n(\cdot)$ is considered to converge to $\psi(\cdot)$ then there exists a compact set such that the supports of $\psi_n(\cdot)$ and $\psi(\cdot)$ are contained therein and for each $k = 0, 1, 2, \ldots$; $\psi^{(k)}_n(\cdot)$ converges uniformly to $\psi(\cdot)$. By $\mathcal{D}'(\mathbb{R}; \mathbb{R})$ we mean the continuous dual of $\mathcal{D}(\mathbb{R}; \mathbb{R})$, that is, the space of all continuous linear functionals on $\mathcal{D}(\mathbb{R}; \mathbb{R})$. The elements of the primary topological vector space will be called test functions and the elements of the continuous dual will be called generalized functions. For a generalized function $\hat{f}$ and test function $\psi(\cdot)$ we denote the pairing, that is, evaluation of $\hat{f}$ at $\psi(\cdot)$, by $\langle \hat{f}, \psi(\cdot) \rangle$.

We remark that generalized functions are not functions in the classical sense; in particular, they are not defined point-wise, but rather on how they evaluate test functions. A nonsingular generalized function is any generalized function for which there does in fact exist a classical-function representation, that is, if $\hat{f}$ is nonsingular then there exists a classical function $f(\cdot)$ such that

$$\langle \hat{f}, \psi(\cdot) \rangle = \int_{-\infty}^{\infty} f(z)\psi(z)\,dz$$

This implies there corresponds to every locally integrable function $f(\cdot)$ a nonsingular generalized function $\hat{f}$. By $\Pi$, $\phi^\varepsilon * \Pi$, and $\hat{\Pi}$ we mean the nonsingular generalized functions corresponding to $\Pi(\cdot)$, $(\phi^\varepsilon * \Pi)(\cdot)$, and $\hat{\Pi}(\cdot)$, respectively.

Many facts about generalized functions follow easily from their relation to test functions and can be determined via reference to nonsingular generalized functions. Examples include the interchange of limits, differentiation, and integration by parts. Thus we easily calculate

$$\langle \Pi', \psi(\cdot) \rangle = -\langle \Pi, \psi'(\cdot) \rangle$$

$$= - \int_{-\infty}^{\infty} \Pi(z)\psi'(z)\,dz$$

$$= - \int_{-\infty}^{\infty} \lim_{\varepsilon \to 0} (\phi^\varepsilon * \Pi)(z)\psi'(z)\,dz$$

$$= - \lim_{\varepsilon \to 0} \int_{-\infty}^{\infty} (\phi^\varepsilon * \Pi)(z)\psi'(z)\,dz$$
Thus $\Pi'$ is equal in the sense of generalized functions to the nonsingular generalized function $\hat{\Pi}$. By a slight abuse of notation we write

$$\Pi'(z) = \hat{\Pi}(z)$$

and understand that the derivative is the generalized derivative.

We briskly review the main results of chapter 2 applied to our practical problem. From (2.9) it is clear that the costate process satisfies

$$d p_t = -\Pi'(h(t, x_{t-}^u, y_{t-}^u, u_t, \alpha_t) - y_{t-}^u) \begin{bmatrix} h_{x_1} \\ \vdots \\ h_{x_{N-1}} \\ h_{y-1} \end{bmatrix} (t, x_{t-}^u, y_{t-}^u, u_t, \alpha_t) dt$$

$$- \sum_{d=1}^{D} \begin{bmatrix} b_{x_1}^1 & \cdots & b_{x_1}^{N-1} & b_{x_1}^N \\ \vdots & \ddots & \vdots & \vdots \\ b_{x_{N-1}}^1 & \cdots & b_{x_{N-1}}^{N-1} & b_{x_{N-1}}^N \\ b_{y}^1 & \cdots & b_{y}^{N-1} & b_{y}^N \end{bmatrix} \begin{bmatrix} \sigma_{x_1}^{1d} \\ \vdots \\ \sigma_{x_{N-1}}^{1d} \\ \sigma_{y}^{1d} \end{bmatrix} (t, x_{t-}^u, y_{t-}^u, u_t, \alpha_t) a_t^d dt$$

$$- \sum_{l=1}^{l} \int_{E_l} \begin{bmatrix} \gamma_{x_1}^{1l} \\ \vdots \\ \gamma_{x_{N-1}}^{1l} \\ \gamma_{y}^{1l} \end{bmatrix} \begin{bmatrix} \gamma_{x_1}^{(N-1)l} \\ \vdots \\ \gamma_{x_{N-1}}^{(N-1)l} \\ \gamma_{y}^{(N-1)l} \end{bmatrix} (t, x_{t-}^u, y_{t-}^u, u_t, \alpha_t, \zeta) r^{st}(t, \zeta) \lambda^t(d\zeta) dt$$
From (2.12) we have

\[
H(t, x, y, v, i, p, q, r, s) = \Pi(h(t, x, y, v, i) - y) + p^\top b(t, x, y, v, i) \\
+ \sum_{d=1}^{D} (q^{sd})^\top \sigma^{sd}(t, x, y, v, i) \\
+ \sum_{l=1}^{L} \int_{E_0} (r^{sl}(t, \zeta))\gamma^{sl}(t, x, y, v, i, \zeta)\lambda^{l}(d\zeta) \\
+ \sum_{m=1}^{M} (s^{sm})^\top \eta^{sm}(t, x, y, v, i) \mathcal{G}^{lm} 1_{[i \neq m]}
\]

and thus the Pontryagin Minimum Principle (Theorem 10) reads as

\[
0 \leq \Pi'(h(t, x^u_{t_1}, y^u_{t_1}, u_t, \alpha_t) - y^u_{t_1})h_v(t, x^u_{t_1}, y^u_{t_1}, u_t, \alpha_t)(v - u_t) \\
+ p^\top b_v(t, x^u_{t_1}, y^u_{t_1}, u_t, \alpha_t)(v - u_t) \\
+ \sum_{d=1}^{D} (q^{sd})^\top \sigma^{sd}(t, x^u_{t_1}, y^u_{t_1}, u_t, \alpha_t)(v - u_t) \\
+ \sum_{l=1}^{L} \int_{E_0} (r^{sl}(t, \zeta))\gamma^{sl}(t, x^u_{t_1}, y^u_{t_1}, u_t, \alpha_t, \zeta)(v - u_t)\lambda^{l}(d\zeta) \\
+ \sum_{m=1}^{M} (s^{sm})^\top \eta^{sm}(t, x^u_{t_1}, y^u_{t_1}, u_t, \alpha_t)(v - u_t) \mathcal{G}^{lm} 1_{[i \neq m]}
\]

on a set of full (\(dt \otimes \mathcal{P}\))-measure. Finally, the Hamilton-Jacobi-Bellman equation (Theorem 12) reads
as

\[ V_i(s, \xi, w, i) = -\inf_{i' \in I} \left[ \Pi(h(s, \xi, w, v, i) - w) + (\mathcal{L}_s^\nu V)(s, \xi, w, i) \right] \] (3.5)

and

\[ V(T, \xi, w, i) = \Pi(g(\xi, w, i) - w) \]

with

\[
0 = -\{\mathcal{L}_s^\nu V\}(s, \xi, w, i) \\
+ \left[ V_x^1 \cdots V_{x_{N-1}} \ V_y \right](s, \xi, w, i) b(s, \xi, w, v, i) \\
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_1=1}^{N-1} \sum_{n_2=1}^{N-1} V_{x_{n_1} x_{n_2}}(s, \xi, w, i)(\sigma^{n_1 d} \sigma^{n_2 d})(s, \xi, w, v, i) \\
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_1=1}^{N-1} V_{x_{n_1} y}(s, \xi, w, i)(\sigma^{n_1 d} \sigma^{N d})(s, \xi, w, v, i) \\
+ \frac{1}{2} \sum_{d=1}^{D} \sum_{n_2=1}^{N-1} V_{y x_{n_2}}(s, \xi, w, i)(\sigma^{N d} \sigma^{n_2 d})(s, \xi, w, v, i) \\
+ \frac{1}{2} \sum_{d=1}^{D} V_{y y}(s, \xi, w, i)(\sigma^{N d} \sigma^{N d})(s, \xi, w, v, i) \\
+ \sum_{l=1}^{I} \int_{E_0} \left\{ V(s, \xi + P_{N-1} \gamma^l(s, \xi, w, v, i, \xi), w + \gamma^N(s, \xi, w, v, i, \xi), i) \\
- V(s, \xi, w, i) \\
- \left[ V_x^1 \cdots V_{x_{N-1}} \ V_y \right](s, \xi, w, i) \gamma^l(s, \xi, w, v, i, \xi) \right\} \lambda^l(\text{d}\xi) \\
+ \sum_{m=1}^{M} \left\{ V(s, \xi + P_{N-1} \eta^m(s, \xi, w, v, i), w + \eta^N(s, \xi, w, v, i), m) \\
- V(s, \xi, w, i) \\
- \left[ V_x^1 \cdots V_{x_{N-1}} \ V_y \right](s, \xi, w, i) \eta^m(s, \xi, w, v, i) \right\} G^{im}_{1 [i \neq m]} \]

where \( P_{N-1} \in \mathbb{R}^{(N-1) \times N} \) is such that \( P_{N-1}^{ij} = 1 \) if \( i = j \), elsewise 0; and we have again made a slight change in notation so as to distinguish the first \( N - 1 \) components of the state from the last component.
### 3.4 Numerical Solutions to the Hamilton-Jacobi-Bellman Equation

Without loss of generality we assume $N = 2$, that is, $\xi \in \mathbb{R}$. Furthermore we assume that an approximation of the integral over $E_0$ in (3.2) has already been chosen so that it can be represented by an ordinary sum over a sufficiently large number of independent sources of jump noise, each with but a single jump value in $E_0$. Thus we shall drop the argument $\zeta$ from $\gamma(\cdot)$ and write $dN^I_t$ and $\lambda^I_t$ rather than $dN^I_t(d\zeta, d\zeta)$ and $\lambda^I_t(d\zeta)$, respectively. As we shall see in section 3.5 these assumptions pose no limitations on our work.

Let $n$ be a positive integer and $z_{\min} = z_0 < z_1 < \cdots < z_n = z_{\max}$ the $n + 1$ nodes of a fixed but otherwise arbitrary partition of $[z_{\min}, z_{\max}]$. A cubic spline over the given partition is a function $s_3(\cdot)$ such that $s_3(\cdot)|[z_j, z_{j+1}]$ is a polynomial of degree at most 3, $j = 0, 1, \ldots, n - 1$; and $s_3(\cdot) \in C^2([z_{\min}, z_{\max}]; \mathbb{R})$. The former condition implies $(3 + 1) \times n$ degrees of freedom while the latter $3 \times (n - 1)$ constraints, thus the dimension of the space of all cubic splines over the given nodes is $n + 3$.

In the sequel we shall be interested in the $(n + 1)$-dimensional subspace of natural cubic splines, that is, cubic splines with vanishing second derivatives at the boundary. Suppose $i, j = 0, 1, \ldots, n$ and let $f_i(\cdot)$ be the cubic spline such that

$$ f_i(z_j) = \begin{cases} 
1 & \text{if } i = j \\
0 & \text{if } i \neq j 
\end{cases} $$

and

$$ f_i''(z_0) = f_i''(z_n) = 0 $$

Clearly $\{f_i(\cdot)\}_{i=0}^n$ forms a basis for the subspace; if $s_3(\cdot)$ is a natural cubic spline with $s_3(z_j) = a_j$ then we have the representation

$$ s_3(z) = \sum_{i=0}^{n} a_i f_i(z) $$

A basis of this form is sometimes called a natural cubic spline basis; the $a_j$ are the coordinates or Fourier coefficients of $s_3(\cdot)$.

By way of illustration consider the natural cubic spline with knots at $\{0, 1, \ldots, 5\}$ and corresponding values at $\{-1, 1.5, 2.5, 0, -2, -1\}$. In figure 3.1 we have the the natural cubic spline generated by these knots and values. Notice that indeed the second derivatives are 0 at the endpoints.

Again using the set of knots $\{0, 1, \ldots, 5\}$ let us now consider the natural cubic spline basis. The basis should contain 6 functions, each with value 1 at its index in the basis and 0 at the other indices, and each with second derivatives of 0 at the endpoints. In figures 3.2, 3.3, and 3.4 we have the
basis functions with first and second derivatives. Notice that indeed each basis function is 1 at its index and 0 at the other indices and that indeed its second derivatives are 0 at the endpoints. For completeness, we have included in figure 3.5 the piecewise constant, jump discontinuous, third derivatives, insofar as they exist; note that these are not used in any of our calculations.

For each $i = 1, 2, \ldots, M$; suppose there exist functions $F^i_0(\cdot) \in C^1([0, T]; \mathbb{R})$, $F_1(\cdot) \in C^2([\xi_{\min}, \xi_{\max}]; \mathbb{R})$, and $F_2(\cdot) \in C^2([w_{\min}, w_{\max}]; \mathbb{R})$ such that

$$V(s, \xi, w, i) = F^i_0(s)F_1(\xi)F_2(w)$$

Let $\{\phi_j(\cdot)\}_{j=0}^{n_1}$ and $\{\psi_k(\cdot)\}_{k=0}^{n_2}$ be, respectively, natural cubic spline bases over the $n_1 + 1$ nodes

$$\xi_{\min} = \xi_0 < \xi_1 < \cdots < \xi_{n_1} = \xi_{\max}$$

and the $n_2 + 1$ nodes

$$w_{\min} = w_0 < w_1 < \cdots < w_{n_2} = w_{\max}$$
Figure 3.2: Natural Cubic Spline Basis (Example)

Figure 3.3: Natural Cubic Spline Basis, First Derivatives (Example)
Figure 3.4: Natural Cubic Spline Basis, Second Derivatives (Example)

Figure 3.5: Natural Cubic Spline Basis, Third Derivatives (Example)
Thus we have

\[ V(s, \xi, w, i) = F_i^0(s) \left[ \sum_{j=0}^{n_1} a_j \phi_j(\xi) \right] \left[ \sum_{k=0}^{n_2} b_k \psi_k(w) \right] = \sum_{j=0}^{n_1} \sum_{k=0}^{n_2} a_j b_k F_i^0(s) \phi_j(\xi) \psi_k(w) \]

where \( a_j \) and \( b_k \) are the coordinates of \( F_1(\cdot) \) and \( F_2(\cdot) \), respectively. We introduce the following notation

\[ \mathcal{T}(s)^{i}(s) := \begin{bmatrix} a_0 b_{0} F_{0}^{i} & \cdots & a_{0} b_{n_{2}} F_{0}^{i} \\ \vdots & \ddots & \vdots \\ a_{n_{1}} b_{0} F_{0}^{i} & \cdots & a_{n_{1}} b_{n_{2}} F_{0}^{i} \end{bmatrix} \]

and

\[ \phi(\xi) := \begin{bmatrix} \phi_0 \\ \vdots \\ \phi_{n_{1}} \end{bmatrix}, \quad \psi(\xi) := \begin{bmatrix} \psi_0 \\ \vdots \\ \psi_{n_{2}} \end{bmatrix} \]

so that we have

\[ V(s, \xi, w, i) = \phi^\top(\xi) \mathcal{T}(s) \psi(w) \]

We alert the reader to the fact that we use 0-based subscripts in the notations above rather than 1-based superscripts as is normally the case for vector or matrix components.

Thus the scalar partial differential equation with final value (3.5) has been transformed into the \((n_1 + 1)\)-by-(\(n_2 + 1\)) system of ordinary differential equations with final value

\[ \phi^\top(\xi) \mathcal{T}(s) \psi(w) = - \inf_{\nu \in U} \left[ \Pi(h(s, \xi, w, \nu, i) - w) + (\mathcal{L}_s^\nu \phi^\top \mathcal{T} \psi)(s, \xi, w, i) \right] \]

\[ \phi^\top(\xi) \mathcal{T}(T) \psi(w) = \Pi(g(\xi, w, i) - w) \]

Let \( \tau \in [0, T] \) be such that \( s = T - \tau \); and \( \mathcal{S}^i(\cdot) \) and \( \mathcal{L}^\nu(\cdot) \) be the reversals of \( \mathcal{T}(\cdot) \) and \( \mathcal{L}^\nu(\cdot) \), respectively;
that is, such that

\[ \phi^T(\xi)(\mathcal{S}')(\tau)\psi(w) = \inf_{w \in \mathcal{B}} [\Pi(h(T - \tau, \xi, w, v, i) - w) + (\Sigma_j^\nu \phi^T \mathcal{S}^i)(\tau, \xi, w, i)] \]

\[ \phi^T(\xi)\mathcal{S}^i(0)\psi(w) = \Pi[g(\xi, w, i) - w] \]

and

\[ 0 = -((\Sigma_j^\nu \phi^T \mathcal{S}^i)(\tau, \xi, w, i)) \]

\[ + (\phi')^T(\xi)\mathcal{S}^i(\tau)\psi(w) b^1(T - \tau, \xi, w, v, i) \]

\[ + \phi^T(\xi)\mathcal{S}^i(\tau)\psi'(w) b^2(T - \tau, \xi, w, v, i) \]

\[ + \frac{1}{2} \sum_{d=1}^D (\phi^\nu)'(\xi)\mathcal{S}^i(\tau)\psi'(w)(\sigma_1^d \sigma_1^d)(T - \tau, \xi, w, v, i) \]

\[ + \sum_{d=1}^D (\phi^\nu)'(\xi)\mathcal{S}^i(\tau)\psi'(w)(\sigma_1^d \sigma_1^d)(T - \tau, \xi, w, v, i) \]

\[ + \frac{1}{2} \sum_{d=1}^D \phi^T(\xi)\mathcal{S}^i(\tau)\psi''(w)(\sigma_2^d \sigma_2^d)(T - \tau, \xi, w, v, i) \]

\[ + \sum_{l=1}^L \{ \phi^T(\xi + \gamma^l(T - \tau, \xi, w, v, i))\mathcal{S}^i(\tau)\psi(w + \gamma^l(T - \tau, \xi, w, v, i)) \]

\[- \phi^T(\xi)\mathcal{S}^i(\tau)\psi(w) \]

\[-(\phi')^T(\xi)\mathcal{S}^i(\tau)\psi(w) \gamma^1(T - \tau, \xi, w, v, i) \]

\[- \phi^T(\xi)\mathcal{S}^i(\tau)\psi'(w) \gamma^2(T - \tau, \xi, w, v, i) \]

\[ \} \lambda^l \]

\[ + \sum_{m=1}^M \{ \phi^T(\xi + \eta^1m(T - \tau, \xi, w, v, i))\mathcal{S}^i(\tau)\psi(w + \eta^2m(T - \tau, \xi, w, v, i)) \]

\[- \phi^T(\xi)\mathcal{S}^i(\tau)\psi(w) \]

\[-(\phi')^T(\xi)\mathcal{S}^i(\tau)\psi(w) \eta^1m(T - \tau, \xi, w, v, i) \]

\[- \phi^T(\xi)\mathcal{S}^i(\tau)\psi'(w) \eta^2m(T - \tau, \xi, w, v, i) \]

\[ } G^{1m}_{i \neq m} \]

To solve this system of ordinary differential equations with initial value we exploit the structure of the natural cubic spline basis and apply a 4-stage Runge-Kutta algorithm. Observe that for \( j = 0, 1, \ldots, n_1 \); and \( k = 0, 1, \ldots, n_2 \); we have

\[ (\mathcal{S}^i)'_{jk}(\tau) = \phi^T(\xi_j)(\mathcal{S}^i)'(\tau)\psi(w_k) \]
We remark that the 4-stage Runge-Kutta algorithm is explicit and single-step. It achieves greater accuracy relative to Euler’s method but at the expense of multiple function evaluations; in particular the methods uses a convex combination of 4 increment matrices to estimate successive values of the solution.

and thus

\[
(\Sigma^i)_{jk}(\tau) = \inf_{\psi \in \mathcal{U}} \left[ \Pi(h(T - \tau, \xi, j, w_k, v, i) - w_k) + (\Omega^i \phi^T \Sigma^i \psi)(\tau, \xi, j, w_k, i) \right]
\]

\[
\Sigma^i_{jk}(0) = \Pi(g(\xi, j, w_k, i) - w_k)
\]

Consider the \(n_0 + 1\) nodes

\[0 = \tau_0 < \tau_1 < \cdots < \tau_{n_0} = T\]

For \(p = 0, 1, \ldots, n_0-1\); let \(\Delta_p\) denote the time step, that is, \(\Delta_p = \tau_{p+1} - \tau_p\), and \(\Omega^i_{p+1}\) the approximation of \(\Sigma^i(\tau_{p+1})\) given recursively by

\[
\Omega^i_{p+1} = \Omega^i_p + \Delta_p \left\{ \frac{K^1}{6} + \frac{K^2}{3} + \frac{K^3}{3} + \frac{K^4}{6} \right\}
\]

\[
\Omega^i_0 = \Sigma^i(0)
\]

with the \((n_1+1)\)-by-\((n_2+1)\) increment matrices \(K^1, K^2, K^3, \text{and} \ K^4\) given by

\[
K^1_{jk} = \inf_{\psi \in \mathcal{U}} \left[ \Pi(h(T - \tau_p, \xi, j, w_k, v, i) - w_k) + (\Omega^i \phi^T \Omega^i \psi)(\tau_p, \xi, j, w_k, i) \right]
\]

\[
K^2_{jk} = \inf_{\psi \in \mathcal{U}} \left[ \Pi(h(T - (\tau_p + \Delta_p/2), \xi, j, w_k, v, i) - w_k) + (\Omega^i \phi^T (\Omega^i + (\Delta_p/2)K^1)) \psi(\tau_p + \Delta_p/2, \xi, j, w_k, i) \right]
\]

\[
K^3_{jk} = \inf_{\psi \in \mathcal{U}} \left[ \Pi(h(T - (\tau_p + \Delta_p), \xi, j, w_k, v, i) - w_k) + (\Omega^i \phi^T (\Omega^i + (\Delta_p/2)K^2)) \psi(\tau_p + \Delta_p, \xi, j, w_k, i) \right]
\]

\[
K^4_{jk} = \inf_{\psi \in \mathcal{U}} \left[ \Pi(h(T - (\tau_p + \Delta_p), \xi, j, w_k, v, i) - w_k) + (\Omega^i \phi^T (\Omega^i + \Delta_p K^3)) \psi(\tau_p + \Delta_p, \xi, j, w_k, i) \right]
\]

We remark that the 4-stage Runge-Kutta algorithm is explicit and single-step. It achieves greater accuracy relative to Euler’s method but at the expense of multiple function evaluations; in particular the methods uses a convex combination of 4 increment matrices to estimate successive values of the solution.

### 3.5 Our Alternative in Action

The alternative formulation given by (3.2) and (3.4), and the computational procedure for solving the related Hamilton-Jacobi-Bellman equation described in section 3.4, provide a powerful approach to pricing options. It avoids the use of equivalent martingale probabilities, allows for the modeling of
random jump-discontinuities, and allows for the modeling of multiple market-modes (with both friction and frictionless transitions). Importantly, this approach also allows for a tighter specification of exactly how the option is hedged. This, for example, may be used to remove any requirement that the market for the option be liquid, or that the desk be able to obtain a near arbitrary amount of financing. This may also be used to implement the risk-management policy requiring an option with only an at-expiry term to be hedged throughout its life.

Let \( \{x_t^v\} \) denote the price process of a stock and \( \{y_t^v\} \) the wealth process of a super-replicating portfolio. The control \( \{v_t\} \) determines the composition of the portfolio, which may contain only shares of the stock and a deposit account. The option is entered into at time \( t = 0 \) and expires at \( T \). We assume payment is made at \( T \), that is, that the option expiry and option maturity are the same. The option may have path-dependent term and an at-expiry term. The former is computed by \( h(\cdot) \) and the latter is computed by \( g(\cdot) \).

For illustration purposes, assume that the price process of the stock is a natural extension of the geometric Brownian motion, that is, it satisfies \( \mathbb{P} - a.s. \)

\[
\frac{dx_t}{x_t} = \mu_{a_{t^-}} dt + \sum_{d=1}^{2} \sigma^d_{a_{t^-}} dB^d_t + \sum_{l=1}^{2} \gamma^l_{a_{t^-}} d\tilde{N}_t^l + \sum_{m=1}^{2} \eta^m_{a_{t^-}} d\tilde{\Upsilon}_t^m \\
x_0 = \hat{x}
\]

where \( \mu_i, \sigma^d_i, \gamma^l_i, \eta^m_i, \) for \( i = 1, 2; \) are constant, but specific to mode; they are model parameters marked by the desk. As the price process of the stock is not controlled we drop the superscript \( v \). As \{x_t\} is nonnegative-valued, that is, \( \mathbb{R} \)-valued, we suppress the matrix notation on the coefficients. Finally, as there are only finitely many jumps we write \( d\tilde{N}_t^l \) for \( \tilde{N}_t^l(dt, d\zeta) \). (We give an interpretation of \( D = L = M = 2 \) below.)

Take \( K = 1 \) and \( U \) the set \([0, 1]\). The control process \( \{v_t\} \) then represents the fraction of the wealth process of the super-replicating portfolio invested in the stock, and \( \{1 - v_t\} \) represents the amount kept on deposit. The deposit is assumed to grow at a fixed rate of interest \( r \in \mathbb{R} \). The wealth process of the super-replicating portfolio satisfies \( \mathbb{P} - a.s. \)

\[
\frac{dy_t^v}{y_t^v} = v_t \frac{dx_t}{x_t} + (1 - v_t) r \, dt \\
y_0^v = y
\]

Crucially, notice that the wealth process of the super-replicating portfolio gains or losses value based solely on the gains and losses of the stock and on accrued interest.
In the form of (3.2) our state equation reads as $P - a.s.$

$$
\begin{align*}
\frac{d}{dt} \begin{bmatrix} x_t \\ y_t^v \end{bmatrix} &= \begin{bmatrix} \mu_{a_t} - \sigma_{a_t}^d \\ \nu_t \sigma_{a_t}^d + (1 - \nu_t) r y_t^v \end{bmatrix} dt + \sum_{d=1}^{2} \begin{bmatrix} \sigma_{a_t}^d x_t - \\ \nu_t \sigma_{a_t}^d y_t^v \end{bmatrix} dB_t^d \\
&+ \sum_{l=1}^{2} \begin{bmatrix} \gamma_{a_t}^l x_t - \\ \nu_t \gamma_{a_t}^l y_t^v \end{bmatrix} d\tilde{N}_t^l + \sum_{m=1}^{2} \begin{bmatrix} \eta_{a_t}^m x_t - \\ \nu_t \eta_{a_t}^m y_t^v \end{bmatrix} d\tilde{Y}_t^m \\
\end{align*}
$$

(3.7)

The market interpretation of (3.7) is as follows. The $dt$-term quantifies the net drift of the market. The $dB_t$-terms represents market volatility caused by the multitude of transactions (each of which is for relatively small numbers of shares of the stock), and by the flow of ordinary business news. The $d\tilde{N}_t^l$-terms also represent market volatility, but of a concentrated sort. This term quantifies the impact of large transactions (made external to our portfolio), and of extraordinary business news. Finally, the $d\tilde{Y}_t^m$-terms capture what, if any, impact changes in general market sentiment have on the stock, where the market sentiments are bullish and bearish.

Again, for illustration purposes, let the strike $S > 0$ and expiry $T$ be fixed, and consider the payoff of the form given $P - a.s.$ by

$$
\int_0^T \Pi(x_t - S) dt + \Pi(x_T - S)
$$

In the form of (3.4) our cost functional then reads as

$$
\mathbb{E} \left[ \int_0^T \Pi \left( x_t - S - y_t^v \right) dt + \Pi \left( x_T - S - y_T^v \right) \right]
$$

(3.8)

Thus our practical problem (3.7) and (3.8) is now ready for the computational procedure described in section 3.4.

Enough set-up, let’s compute! Consider the following particulars

$$
\hat{x} = 20.00 \\
r = 0.02 \\
T = 0.5 \\
S = 20.00
$$

Recall that there are 2 regimes in this practical problem. When $\alpha_{t-} = 1$ we are in a market with
bullish sentiment, and when \( \alpha_t = 2 \) we are in a market with bearish sentiment. The drift is

\[
\mu_1 = 0.08 \\
\mu_2 = -0.05
\]

As there are two sources of continuous volatility, the diffusion is

\[
D = 2 \\
\sigma_1^1 = 0.05 \\
\sigma_1^2 = 0.15 \\
\sigma_2^1 = 0.03 \\
\sigma_2^2 = 0.10
\]

Similarly, there are two sources of discontinuous volatility. Recall that lambda is the per-unit-time jump intensity of the source of discontinuous volatility. Thus the jump is

\[
L = 2 \\
\gamma_1^1 = 0.05 \\
\gamma_1^2 = 0.10 \\
\gamma_2^1 = -0.05 \\
\gamma_2^2 = -0.15 \\
\lambda_1 = 3 \\
\lambda_2 = 2
\]

The regime switching is

\[
M = 2 \\
\eta_1^1 = 0.00 \\
\eta_1^2 = -0.02 \\
\eta_2^1 = 0.01 \\
\eta_2^2 = 0.00 \\
G = \begin{bmatrix} -0.2 & 0.2 \\ 4.0 & -4.0 \end{bmatrix}
\]

Note that \( \eta_1^1 = \eta_2^2 = 0 \) as those are not actual state transitions. Notice also that the diagonal of the
The infinitesimal generator is negative and the rows do indeed sum to 0.

The results are shown in figures 3.6 and 3.7. Figure 3.6 gives the value function at $t = 0$ for a range of values of the initial underlying and the initial investment. The figure is read starting on the vertical axis: find the $\hat{x}$, which is 20.00 in the present case, and scan right until the numerical threshold for the boundary for 0 value, which in this case is set to 0.05, is hit. Sanity check the figure by noting that for fixed initial investment (that is, $y$) the value function increases as the initial underlying (that is, $\hat{x}$) increases, and that for fixed initial underlying (that is, $\hat{x}$) the value function decreases as the initial investment (that is, $y$) increases.

The particular values used in this example were chosen for reasonableness and ease of illustration. In the appendices there are a wide range of configuration cases and validation cases that

![Figure 3.6: Value Function at t=0](image)

To aid in reading off the results, figure 3.7 gives the value function at $t = 0$ for fixed initial underlying (that is, $\hat{x}$). The figure is read by looking for the vertical line, which marks out the the least initial investment (that is, $y$) such that the value function is less than the numerical threshold for the boundary for 0 value, which in this case is set to 0.05. Sanity check the figure by noting that the value function decreases as the initial investment (that is, $y$) increases.
support the appropriateness of the choices. As a general rule, the computational procedure for our practical problem (3.7) and (3.8) is robust for any positive values of $\hat{x}$, $r$, $T$, and $S$; and for $\mu$, $\sigma$, $\gamma$, and $\eta$ of magnitude no greater than 0.20 (unless the numerical configuration is changed, for example, the step size in the time dimension is increased, which of course increases compute times).
REFERENCES


APPENDIX

A

PYTHON CODE FOR SOLVING THE HAMILTON-JACOBI-BELLMAN EQUATION

The purpose of this appendix is to provide the reader with all of the computer code used to produce the results contained herein. To that end, there are two listings in this appendix. The first listing gives the computer code for building cubic splines. The second listing gives the computer code for building solutions to the Hamilton-Jacobi-Bellman Equation. In both cases, there is a test harness provided.

We remind the reader that all computations were performed on an Apple iMac. The machine contained a 3.1 GHz 6-Core (12 logical cores) Intel Core i5 CPU with 12 MB L3 cache, and 16 GB 2667 MHz DDR4 of RAM. The software stack was as follows: Apple macOS Ventura (version 13.0.1), Python 3.9.13, Intel’s Math Kernel Library (MKL) 2021.4.0, NumPy 1.21.5, SciPy 1.9.1, and JupyterLab 3.4.4.

The solvers are implemented in the functional programming paradigm, and make heavy use of closures and inner functions. This allows us to keep the computer code as close to the mathematical expressions as possible. Wherever there is a conflict, we choose the clarity of the computer code over the efficiency of its execution: this is research computer code, not production computer code. Both modules are extensively documented using the usual approach of docstrings, and there are extensive
comments for especially opaque sections of computer code. The test harnesses are implemented as Jupyter (formerly IPython) notebooks, and are included here in Markdown format. (Note that to convert the Markdown format back to executable notebooks, some slight editing might be required.)

The following listing gives the computer code for building cubic splines. The main functions are `cubic_spline_factory` (line 6) and `natural_cubic_spline_basis_factory` (line 218).

```python
import numpy as np

def cubic_spline_factory(knots,
                         values,
                         lh_deriv_type='s',
                         lh_deriv_val=0.0,
                         rh_deriv_type='s',
                         rh_deriv_val=0.0):
    
    Cubic splines are functions in a single variable that are piece-wise cubic and globally twice continuously differentiable.

    Parameters
    ----------
    knots, values : ndarray
        1D arrays containing the abscissae and ordinates, respectively.
    lh_deriv_type, rh_deriv_type : {'f', 's'}, optional
        Indication of whether the spline's remaining degrees of freedom are saturated by 1st or 2nd derivatives at the endpoints.
    lh_deriv_value, rh_deriv_value : float, optional
        The numerical value of the endpoint derivatives.

    Returns
    -------
    out : tuple
        4 functions that are, in order, the spline and its derivatives.

    Notes
    -----.
    The algorithm used is taken from [1].
```
References
----------

```python
knots = np.array(knots, copy=True)
values = np.array(values, copy=True)

# The step sizes between knots. Note that the algorithm's h_i is here stored in h[i-1].
h = knots[1:] - knots[:-1]

# The upper diagonal of the M-continuity system.
lam = np.zeros(knots.size - 1)
lam[1:] = h[1:] / (h[:-1]+h[1:]).

# The lower diagonal of the M-continuity system.
mu = np.zeros(knots.size - 1)
mu[:-1] = h[:-1] / (h[:-1]+h[1:]).

# The right-hand side of the M-continuity system.
d = np.zeros(knots.size)
d[1:-1] = (6.0 / (h[:-1] + h[1:]),
* ((values[2:]-values[1:-1]) / h[1:]
- (values[1:-1]-values[:-2]) / h[:-1]))

# The derivative at the left endpoint.
if lh_deriv_type == 'f':
    lam[0] = 1.0
d[0] = (6.0 / h[0],
* ((values[1]-values[0])/h[0] - lh_deriv_val))
elif lh_deriv_type == 's':
    lam[0] = 0.0
d[0] = 2.0 * lh_deriv_val

# The derivative at the right endpoint.
if rh_deriv_type == 'f':
```
mu[-1] = 1.0
d[-1] = (6.0
    / h[-1]
    * (rh_deriv_val - (values[-1]-values[-2])/h[-1]))

elif rh_deriv_type == 's':
    mu[-1] = 0.0
d[-1] = 2.0 * rh_deriv_val

# Now solve the M-continuity system.
M = np.linalg.solve((2.0 * np.eye(knots.size)
    + np.diag(lam, 1)
    + np.diag(mu, -1)),
    d)
C1 = ((values[1:]-values[:-1]) / h[:]
    - (h[:] / 6.0 * (M[1:]-M[:-1])))
C2 = values[:-1] - h[:]**2/6.0*M[:-1]

# Nested helper-function that identifies which M, C1, and C2 are used in
# the evaluation of the spline at some x. It does this by finding the
# index in the ndarrays created in the solving of the M-continuity
# system above. It also computes in advance some values used in all of
# the closures created below.
def segment(x):
    if x <= knots[0]:
        seg = 1
    elif x > knots[-2]:
        seg = knots.size - 1
    else:
        seg = np.searchsorted(knots, x)
    num1 = knots[seg] - x
    num2 = x - knots[seg-1]
    denom = h[seg-1]
    return seg, num1, num2, denom

def cubic_spline(x):
    """A 1D cubic spline.

Parameters
---------
    x : float
        The point at which to evaluate the spline.
Returns
-------
out : float
    The value of the spline at 'x'.

See Also
--------
cubic_spline_factory

""

seg, num1, num2, denom = segment(x)
return (num1**3 / (6.0*denom) * M[seg-1]
    + num2**3 / (6.0*denom) * M[seg]
    + num2*C1[seg-1]
    + C2[seg-1])

def cubic_spline_1_deriv(x):
    """The 1st derivative of a 1D cubic spline.

Parameters
----------
x : float
    The point at which to evaluate the 1st derivative.

Returns
-------
out : float
    The value of the 1st derivative at 'x'.

See Also
--------
cubic_spline_factory

""

seg, num1, num2, denom = segment(x)
return (-1.0 * num1**2 / (2.0*denom) * M[seg-1]
    + num2**2 / (2.0*denom) * M[seg]
    + C1[seg-1])
def cubic_spline_2_deriv(x):
    """The 2nd derivative of a 1D cubic spline.

    Parameters
    ----------
    x : float
        The point at which to evaluate the 2nd derivative.

    Returns
    -------
    out : float
        The value of the 2nd derivative at 'x'.

    See Also
    --------
    cubic_spline_factory
    """
    seg, num1, num2, denom = segment(x)
    return (num1 / denom * M[seg-1]
            + num2 / denom * M[seg])

def cubic_spline_3_deriv(x):
    """The 3rd derivative of a 1D cubic spline.

    Parameters
    ----------
    x : float
        The point at which to evaluate the 3rd derivative.

    Returns
    -------
    out : float
        The value of the 3rd derivative at 'x'.

    See Also
    --------
    cubic_spline_factory
    Notes
    -----
By 3rd derivative we really mean the piece-wise constant function
defined, at all abscissae except knots, by the 3rd derivative.

```python
seg, num1, num2, denom = segment(x)
return (-1.0 / denom * M[seg-1]
       + 1.0 / denom * M[seg])
```

```python
def natural_cubic_spline_basis_factory(knots):
    """A factory that makes natural cubic-spline bases.

    Natural cubic-spline bases are collections of natural cubic splines such
    that each member is 1 at only a single knot, 0 at all other knots, and
    each member in the collection distinct. Recall that natural cubic
    splines necessarily have second derivatives of 0 at both the left- and
    right-hand endpoints.

    Parameters
    ----------
    knots : ndarray
        1D array containing the abscissae.

    Returns
    -------
    out : tuple
        4 functions that are, in order, the spline basis and its derivative
        bases.

    ""
    basis = [cubic_spline_factory(knots,row) for row in np.eye(knots.size)]
```

```python
def natural_cubic_spline_basis(x):
    """A 1D natural cubic-spline basis.
```
Parameters
----------
x : float
    The point at which to evaluate the spline.

Returns
-------
out : ndarray
    A 1D array containing the value of each basis spline at 'x'.

See Also
--------
cubic_spline_factory
natural_cubic_spline_basis_factory

```
return np.array([cubic_spline[0](x) for cubic_spline in basis])
```

def natural_cubic_spline_basis_1_deriv(x):
    """The 1st derivative of a 1D natural cubic-spline basis.

Parameters
----------
x : float
    The point at which to evaluate the 1st derivative.

Returns
-------
out : ndarray
    A 1D array containing the 1st derivative of each basis spline at 'x'.

See Also
--------
cubic_spline_factory
natural_cubic_spline_basis_factory

```
return np.array([cubic_spline[1](x) for cubic_spline in basis])
```
def natural_cubic_spline_basis_2_deriv(x):
    """The 2nd derivative of a 1D natural cubic-spline basis.

    Parameters
    ----------
    x : float
        The point at which to evaluate the 2nd derivative.

    Returns
    -------
    out : ndarray
        A 1D array containing the 2nd derivative of each basis spline at 'x'.

    See Also
    --------
    cubic_spline_factory
    natural_cubic_spline_basis_factory

    """

    return np.array([cubic_spline[2](x) for cubic_spline in basis])

def natural_cubic_spline_basis_3_deriv(x):
    """The 3rd derivative of a 1D natural cubic-spline basis.

    Parameters
    ----------
    x : float
        The point at which to evaluate the 3rd derivative.

    Returns
    -------
    out : ndarray
        A 1D array containing the 3rd derivative of each basis spline at 'x'.

    See Also
    --------
    cubic_spline_factory
    natural_cubic_spline_basis_factory
return np.array([cubic_spline[3](x) for cubic_spline in basis])

return (natural_cubic_spline_basis,
        natural_cubic_spline_basis_1_deriv,
        natural_cubic_spline_basis_2_deriv,
        natural_cubic_spline_basis_3_deriv)

The following listing gives the Jupyter notebook (in Markdown format) for testing the spline module. Indeed, all of the spline figures in section 3.4 are generated from it.

```python
# spline-test-harness.md

import matplotlib.pyplot as plt
import numpy as np
import spline

%matplotlib inline

### ‘cubic_spline_factory’ and the Natural Cubic Spline

knots = np.array([0, 1, 2, 3, 4, 5])
values = np.array([-1, 1.5, 2.5, 0, -2, -1])

spl, spl_1, spl_2, spl_3 = spline.cubic_spline_factory(knots, values)

A picture is worth a thousand words. Plot this spline and its derivatives. Notice that the second derivative is 0 at each endpoint.

X = np.linspace(knots[0], knots[-1], 100*(knots[-1]-knots[0]))
Y = np.array([spl(x) for x in X])
Y_1 = np.array([spl_1(x) for x in X])
Y_2 = np.array([spl_2(x) for x in X])
```
Y_3 = np.array([spl_3(x) for x in X])
plt.figure()
plt.subplot(221)
plt.plot(X, Y)
plt.title(r'$s_3(\cdot)$ (natural)');
plt.subplot(222)
plt.plot(X, Y_1)
plt.title(r'$s_3^{\prime}(\cdot)$ (natural)');
plt.subplot(223)
plt.plot(X, Y_2)
plt.title(r'$s_3^{\prime\prime}(\cdot)$ (natural)');
plt.subplot(224)
plt.plot(X, Y_3)
plt.title(r'$s_3^{\prime\prime\prime}(\cdot)$ (natural)');
plt.tight_layout()
plt.savefig('figures/spline-natural-cubic-with-derivatives')

```python
### 'cubic_spline_factory' and the General Cubic Spline
```
```
gspl, gspl_1, gspl_2, gspl_3 = spline.cubic_spline_factory(knots, values, 'f', -2.5, 's', 5.0)
```
```
A picture is worth a thousand words. Plot this spline and its derivatives. Notice that the first derivative at the left endpoint is -2.5 and that the second derivative at the right endpoint is 5.
```
```
```
```
```python
Y_g = np.array([gspl(x) for x in X])
Y_g_1 = np.array([gspl_1(x) for x in X])
Y_g_2 = np.array([gspl_2(x) for x in X])
Y_g_3 = np.array([gspl_3(x) for x in X])
plt.figure()
plt.subplot(221)
plt.plot(X, Y_g)
plt.title(r'$s_3(\cdot)$ (general)');
```
```
plt.subplot(222)
plt.plot(X, Y_g_1)
plt.title(r'$s_3^\prime(\cdot)$ (general)');
plt.subplot(223)
plt.plot(X, Y_g_2)
plt.title(r'$s_3^\prime\prime(\cdot)$ (general)');
plt.subplot(224)
plt.plot(X, Y_g_3)
plt.title(r'$s_3^\prime\prime\prime(\cdot)$ (general)');
plt.tight_layout()
plt.savefig('figures/spline-general-cubic-with-derivatives')

```python
splb, splb_1, splb_2, splb_3 = spline.natural_cubic_spline_basis_factory(knots)
```

A picture is worth a thousand words. Plot the first spline in the basis and its derivatives. Notice that the spline is 1 at the first knot and 0 at the other knots, and that the second derivative is 0 at each endpoint.

```python
X = np.linspace(knots[0], knots[-1], 100*(knots[-1]-knots[0]))
Y = np.array([splb(x)[0] for x in X])
Y_1 = np.array([splb_1(x)[0] for x in X])
Y_2 = np.array([splb_2(x)[0] for x in X])
Y_3 = np.array([splb_3(x)[0] for x in X])
plt.figure()
plt.subplot(221)
plt.plot(X, Y)
plt.title(r'$s_3(\cdot)$ (natural)');
plt.subplot(222)
plt.plot(X, Y_1)
plt.title(r'$s_3^\prime(\cdot)$ (natural)');
plt.subplot(223)
plt.plot(X, Y_2)
plt.title(r'$s_3^\prime\prime(\cdot)$ (natural)');
plt.subplot(224)
plt.plot(X, Y_3)
plt.title(r'$s_3^\prime\prime\prime(\cdot)$ (natural)');
```
A picture is worth a thousand words. Plot all of the splines in the spline basis. Notice that each spline is 1 at a single knot and 0 at the other knots.

```
plt.tight_layout()
plt.savefig('figures/spline-cubic-first-basis-with-derivatives')
```

A picture is worth a thousand words. Plot the first derivative of all of the splines in the spline basis.
A picture is worth a thousand words. Plot the second derivative of all of the splines in the spline basis. Notice that each spline's second derivative is 0 at each endpoint.
A picture is worth a thousand words. Plot the third derivative of all of the splines in the spline basis.

```python
X = np.linspace(knots[0], knots[-1], 100*(knots[-1]-knots[0]))
Y_0 = np.array([splb_3(x)[0] for x in X])
Y_1 = np.array([splb_3(x)[1] for x in X])
Y_2 = np.array([splb_3(x)[2] for x in X])
Y_3 = np.array([splb_3(x)[3] for x in X])
Y_4 = np.array([splb_3(x)[4] for x in X])
Y_5 = np.array([splb_3(x)[5] for x in X])
```
The following listing gives the computer code for building solutions to the Hamilton-Jacobi-Bellman Equation. The main function is value_function_factory (line 8).

```
import numpy as np
import spline

def value_function_factory(h,
g, b, D, sig, L, gam, lam, M, eta, G, tt, xx, yy,
```

```
A factory that makes optimal-control value-functions.

Value functions are solutions to Hamilton-Jacobi-Bellman partial differential equations. HJB PDEs are usually nonlinear, and model stochastic optimal control problems.

This function solves an optimal-control problem in 2 state variables, each of which may be driven by multiple Brownian motions, compensated Poisson-jumps, and compensated Poisson regime-switches.

Parameters
----------

h : func
    The running cost in the optimal-control problem. Returns a scalar, given points in ‘tt’, ‘xx’, ‘yy’, ‘vv’, and a nonnegative integer less than M.

g : func
    The shutdown cost in the optimal-control problem. Returns a scalar, given points in ‘xx’, ‘yy’, and a nonnegative integer less than M.

b : func
    The drift in the dynamics of the optimal-control problem. Returns a 1D array of shape (2,), given points in ‘tt’, ‘xx’, ‘yy’, ‘vv’, and a nonnegative integer less than M.

D : int
    The number of Brownian motions in the dynamics of the optimal-control problem.

sig : func
    The diffusion in the dynamics of the optimal-control problem. Returns a 2D array of shape (2,‘D’), given points in ‘tt’, ‘xx’, ‘yy’, ‘vv’, and a nonnegative integer less than M.

L : int
    The number of compensated Poisson-jumps in the dynamics of the optimal-control problem. L >= 1. If there are no jumps, simply zero-out ‘gam’ and ‘lam’.

gam : func
    The jump in the dynamics of the optimal-control problem. Returns a 2D array of shape (2,‘L’), given points in ‘tt’, ‘xx’, ‘yy’, ‘vv’, and a nonnegative integer less than M.

lam : ndarray
    1D array of shape (L,) containing the jump intensities of the compensated Poisson-jumps.
M : int
The number of compensated Poisson regime-switches in the dynamics of
the optimal-control problem. M >= 1. If there are no regime
switches, simply zero-out 'eta' and 'G'.
eta : func
The regime switch in the dynamics of the optimal-control problem.
Returns a 2D array of shape (2,'M'), given points in ‘tt’, ‘xx’,
‘yy’, ‘vv’, and a nonnegative integer less than M.
G : ndarray
2D array of shape (M,M) containing the regime-switching intensities
of the compensated Poisson regime-switches.

Returns
------
out : func
A solved value function.

Notes
-----
The current implementation mirrors the natural mathematical statement of
the HJB PDE, and prioritizes form over speed: this is research code, not
production code.

```
T = tt[-1]
tautau = T - tt[::-1]
ntau = tautau.size

# Create the cubic splines for the state variables x and y. Note that
# we are really creating a natural cubic spline basis for each state
# variable.
#
# Evaluation of a spline basis returns a 1D array:
#     idx 0 = which spline in basis
#
# Evaluation of a spline basis at its knots is going to happen a lot,
# so compute these values up front. Caches are 2D arrays:
#     idx 0 = which knot in the spline
#     idx 1 = which spline in the basis
xx = np.copy(xx)
xn = xx.size
```
phi, phi_1, phi_2, phi_3 = spline.natural_cubic_spline_basis_factory(xx)
phi_0_c = np.array([phi(x) for x in xx])
phi_1_c = np.array([phi_1(x) for x in xx])
phi_2_c = np.array([phi_2(x) for x in xx])

yy = np.copy(yy)
ny = yy.size

psi, psi_1, psi_2, psi_3 = spline.natural_cubic_spline_basis_factory(yy)
psi_0_c = np.array([psi(y) for y in yy])
psi_1_c = np.array([psi_1(y) for y in yy])
psi_2_c = np.array([psi_2(y) for y in yy])

# Nested helper-function that computes the Lagrangian’s drift term.
def drift(tau, grad, j, k, v, i):
    return np.dot(grad, b(tau, xx[j], yy[k], v, i))

# Nested helper-function that computes the Lagrangian’s diffusion term.
def diffusion(tau, hess, j, k, v, i):
    diff = sig(tau, xx[j], yy[k], v, i)
    bms = np.array([np.linalg.multi_dot([diff[:,d],
           hess,
           diff[:,d]])
        for d in np.arange(D)])
    return 1 / 2.0 * np.sum(bms)

# Nested helper-function that computes the Lagrangian’s jump term.
def jump(tau, base, val, grad, j, k, v, i):
    jmp = gam(tau, xx[j], yy[k], v, i)
pjs = np.array([((np.linalg.multi_dot([phi(xx[j]+jmp[0,el]),
           base[i],
           psi(yy[k]+jmp[1,el])])
      - val
      - np.dot(grad, jmp[:,el]))
* lam[el])
    for el in np.arange(L)])
    return np.sum(pjs)

# Nested helper-function that computes the Lagrangian’s
# regime-switching term.
def regime(tau, base, val, grad, j, k, v, i):

# Transitioning from regime i to regime m=i in the comprehension
# isn't really a transition, so zero-out its contribution manually
# before summing and returning.
rs = eta(tau, xx[j], yy[k], v, i)
prs = np.array([((np.linalg.multi_dot([phi(xx[j]+rs[0,m]), base[m],
               psi(yy[k]+rs[1,m]))
              - val
              - np.dot(grad, rs[:,m]))
             * G[i,m])
             for m in np.arange(M)])
prs[i] = 0.0
return np.sum(prs)

# Nested helper-function that determines the optimizing control for a
# given (i,j,k) of the solution array.
def optimum(tau, base, j, k, i):

    # Within a stage, and for a given (i,j,k), some parts of the
    # Lagrangian are independent of the control, so compute these values
    # up front.
    val = np.linalg.multi_dot([phi_0_c[j],
                               base[i],
                               psi_0_c[k]])
    grad = np.array([np.linalg.multi_dot([phi_1_c[j],
                                      base[i],
                                      psi_0_c[k])],
                      np.linalg.multi_dot([phi_0_c[j],
                                      base[i],
                                      psi_1_c[k]]))
    hess = np.array([np.linalg.multi_dot([phi_2_c[j],
                                      base[i],
                                      psi_0_c[k]]),
                     np.linalg.multi_dot([phi_1_c[j],
                                      base[i],
                                      psi_1_c[k]]),
                     np.linalg.multi_dot([phi_1_c[j],
                                      base[i],
                                      psi_1_c[k]]),
                     np.linalg.multi_dot([phi_0_c[j],
                                      base[i],
                                      psi_0_c[k]])])
psi_2_c[k]])

candidates = np.array([(h(tau, xx[j], yy[k], v, i)
    + drift(tau, grad, j, k, v, i)
    + diffusion(tau, hess, j, k, v, i)
    + jump(tau, base, val, grad, j, k, v, i)
    + regime(tau, base, val, grad, j, k, v, i))
    for v in vv])

idx_optimum = np.argmin(candidates)
return (candidates[idx_optimum], vv[idx_optimum])

# Nested helper-function that computes a stage of the Runge-Kutta method
# for all (i,j,k) of the solution array.
def runge_kutta_stage(tau, base):

    optima = np.array([[optimum(tau, base, j, k, i)
        for k in np.arange(ny)]
        for j in np.arange(nx)]
        for i in np.arange(M)])

    return (np.array(optima[:,:,:,0]), np.array(optima[:,:,:,1]))

# Build the solution array. Do this via a 4-stage Runge-Kutta algorithm
# on the system of ODEs.
#
# The solution is a 5D array.
#   idx0 = [p] the backward time
#   idx1 = [0] for the value or [1] for the optimal control
#   idx2 = [i] the regime
#   idx3 = [j] the state variable x
#   idx4 = [k] the state variable y
fU = np.zeros((ntau,2,M,nx,ny))
fU[0,0] = np.array([[[g(x, y, i) for y in yy]
            for x in xx]
            for i in np.arange(M)])

for p, (tau, delta) in enumerate(zip(tautau[:-1],
    tautau[1:] - tautau[:-1])):
    K1, u1 = runge_kutta_stage(tau, fU[p,0])
    K2, u2 = runge_kutta_stage(tau+1/2.0*delta, fU[p,0]+1/2.0*delta*K1)
    K3, u3 = runge_kutta_stage(tau+1/2.0*delta, fU[p,0]+1/2.0*delta*K2)
    K4, u4 = runge_kutta_stage(tau+delta, fU[p,0]+delta*K3)
    fU[p+1,0] = fU[p,0] + delta*(1/6.0*K1+1/3.0*K2+1/3.0*K3+1/6.0*K4)
    fU[p+1,0] = np.array([[max(fU[p+1,0,i,ix,iy], 0)]]
for iy in np.arange(ny)
    for ix in np.arange(nx)
    for i in np.arange(M))
fU[p+1,1] = u1

def value_function(p, j, k, i):
    """A solved value function.

    This closure is low level insofar that it only takes args that are
    indices of the discretizations passed in to its factory. It is
    intended to be wrapped in user-supplied application-specific code.

    Parameters
    ---------
    p : int
        An index in ‘tt‘ from its factory.
    j : int
        An index in ‘xx‘ from its factory.
    k : int
        An index in ‘yy‘ from its factory.
    i : int
        A nonnegative integer less than M.

    Returns
    -------
    out : tuple
        2 floats that are the value and optimizing control.

    See Also
    --------
    value_function_factory
    """
    return (fU[ntau-p-1,0,i,j,k], fU[ntau-p-1,1,i,j,k])

    return value_function

The following listing gives the Jupyter notebook (in Markdown format) for testing the hjb module. Indeed, all of the results and figures in section 3.5 are generated from it.
A helper function that computes the mark-to-model for a given range in variable
'y', with 't' and 'x' already fixed. Recall that 'y' is the initial
investment and that we are looking for the smallest 'y' with a sufficiently
small residual.

```
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time
component of the payoff.
expiry = 6.0/12.0
strike = underlying_initial

def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)

""
The details of the drift coefficient function. 'drift' has one row per regime.

"""python
drift = np.array([0.08,
                  -0.05])
def b(t, x, y, v, i):
    return np.array([drift[i]*x,
                     v*drift[i]*y + (1-v)*r*y])

""
The details of the diffusion coefficient function. 'diff' has one row per
regime, and one column per Brownian motion.

"""python
D = 2
diff = np.array([[0.05, 0.15],
                 [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])

""
The details of the jump coefficient function. 'jump' has one row per regime,
and one column per compensated Poisson. 'lam' also has one column per
compensated Poisson.

"""python
L = 2
jump = np.array([[0.05, 0.10],
                 [-0.05, -0.15]])
def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],
                     [v*jmp*y for jmp in jump[i]]])

lam = np.array([3.0, 2.0])

112
The details of the regime switching coefficient function. ‘switch‘ and ‘G‘ must be square, and the rows of ‘G‘ must sum to 0.

```python
M = 2
switch = np.array([[0.0, -0.02],
                    [0.01, 0.0]])
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])
G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])
```

The configuration of the numerical scheme.

```python
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5
tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width, underlying_initial+underlying_width, 2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The solution of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```python
value_function = hjb.value_function_factory(h,
                                           g,
                                           b,
                                           D,
                                           sig,
                                           L,
                                           gam,
                                           tt,
                                           xx,
                                           yy,
                                           vv)
```
A plot showing the value function at $t = 0$ (which implies regime = 0) for given ranges in ‘$y^\prime$’ and ‘$x^\prime$’. Recall that ‘$y^\prime$’ (horizontal axis) represents the range of initial investments and ‘$x^\prime$’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial’ on the vertical axis and scan right until you hit the ‘threshold’ contour, and then scan down to the horizontal axis.

```python
YY_0, XX_0 = np.meshgrid(yy, xx)
VAL_0 = np.zeros((xx.size, yy.size))
for j,k in np.ndindex(xx.size, yy.size):
    VAL_0[j,k] = value_function(0, j, k, 0)[0]
```

```python
levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
plt.clabel(C, inline=1, fontsize=10)
plt.xlabel('Initial Investment ($)')
plt.ylabel('Initial Underlying ($)')
plt.title('Value Function at $t=0$ ($)');
plt.savefig('figures/hjb-test-harness-value-function-at-t-equal-0')
```

To determine the value of the derivative programmatically, create a spline in ‘$y^\prime$’ with $x = \text{underlying_initial}$ and $t = 0$, and then use the helper function ‘mtm’.

```python
initial_values = spline.cubic_spline_factory(
    yy,
```
A plot showing the decrease in the value function as initial investment increases.

```python

X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.figure()
plt.plot(X, Y)
plt.axvline(x=quote, color='r', linestyle='-')
plt.axhline(y=0.0, color='r', linestyle='-')
plt.xlabel('Initial Investment ($)')
plt.ylabel('Value Function at t=0 ($)')
plt.title('Value Function per Initial Investment');
plt.savefig('figures/hjb-test-harness-value-function-per-initial-investment')
```

```
The purpose of this appendix is to provide some basic guidance on the configuration of the numerical scheme, as well as to provide some rigorous model validation. To that end, this appendix will present two configuration cases and eight validation cases. The two configuration cases assess the appropriateness of the discretization of the control and of the time. Essentially the cases support our choice of configuration, allowing for speedy calculations. The eight validation cases test the model's response to changes in certain inputs such as underlying dynamics and payoff. The intention is to gain confidence in the model by observing that its responses are monotonic and directionally reasonable.

We remind the reader (for the last time!) that all computations were performed on an Apple iMac. The machine contained a 3.1 GHz 6-Core (12 logical cores) Intel Core i5 CPU with 12 MB L3 cache, and 16 GB 2667 MHz DDR4 of RAM. The software stack was as follows: Apple macOS Ventura (version 13.0.1), Python 3.9.13, Intel's Math Kernel Library (MKL) 2021.4.0, NumPy 1.21.5, SciPy 1.9.1, and JupyterLab 3.4.4.

The cases are implemented as Jupyter (formerly IPython) notebooks, and are included here in Markdown format. (Note that to convert the Markdown format back to executable notebooks, some slight editing might be required.) In each case, a description and figures are given, and the results are briefly discussed.
Configuration Case: Control Discretization

The purpose of this case is to determine if a sparse discretization of the control is sufficient. To that end, we test four discretizations. The individual results are shown in figures B.1 and B.2. In figure B.3 we see that increasing the granularity of the discretization does not change the quote.

![Figure B.1: Control Discretization, Value Function at t=0](image)

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import hjb
```
Figure B.2: Control Discretization, Value Function per Initial Investment

Figure B.3: Control Discretization, Quote per Control Discretization
A helper function that computes the mark-to-model for a given range in variable ‘y’, with ‘t’ and ‘x’ already fixed. Recall that ‘y’ is the initial investment and that we are looking for the smallest ‘y’ with a sufficiently small residual.

```python
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```python
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```python
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```

The details of the drift coefficient function. ‘drift‘ has one row per regime.
```python
drift = np.array([0.08,  
                 -0.05])
def b(t, x, y, v, i):
    return np.array([drift[i]*x,  
                     v*drift[i]*y + (1-v)*r*y])
```

The details of the diffusion coefficient function. ‘diff’ has one row per regime, and one column per Brownian motion.

```python
D = 2
diff = np.array([[0.05, 0.15],  
                 [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],  
                     [v*df*y for df in diff[i]]])
```

The details of the jump coefficient function. ‘jump’ has one row per regime, and one column per compensated Poisson. ‘lam’ also has one column per compensated Poisson.

```python
L = 2
jump = np.array([[0.05, 0.10],  
                 [-0.05, -0.15]])
def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],  
                     [v*jmp*y for jmp in jump[i]]])
lam = np.array([3.0, 2.0])
```

The details of the regime switching coefficient function. ‘switch’ and ‘G’ must be square, and the rows of ‘G’ must sum to 0.

```python
M = 2
switch = np.array([[0.0, -0.02],  
                   [0.01, 0.0]])
```
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])
G = np.array([[[-0.2, 0.2],
               [4.0, -4.0]]]

The configuration of the numerical scheme.

```
# 'python
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5      # redefine below as part of the validation
ict = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                underlying_initial+underlying_width,
                2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)      # redefine below as part of the
# validation
```

The definition of the validation case, and corresponding solutions of the HJB
equation. WARNING: this is where all the math is so it will take a while to
execute.

```
# 'python
control_granularities = np.array([1, 5, 10, 15])
value_functions = []
for control_granularity in control_granularities:
    vv = np.linspace(0, 1, control_granularity+1)
    value_functions.append(hjb.value_function_factory(h,
                    g,
                    b,
                    D,
                    sig,
                    L,
                    gam,
                    lam,
                    M,
                    eta,
```
The plots showing for each validation case the value function at $t = 0$ (which implies regime = 0) for given ranges in ‘$y$‘ and ‘$x$‘. Recall that ‘$y$‘ (horizontal axis) represents the range of initial investments and ‘$x$‘ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial‘ on the vertical axis and scan right until you hit the ‘threshold‘ contour, and then scan down to the horizontal axis.

```python
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Cntrl Disc = ' + str(control_granularities[n]+1) + '(pt)')
plt.tight_layout()
plt.savefig('figures/configuration-case-control-disc-value-function-at-t-equal-0')
```

The plots showing for each validation case the increase in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(yy,

```
A plot showing the results of the validation case.

```
control_granularities
```

```
quotes
```

```
X = np.array(control_granularities+1)
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('Cntrl Disc (pt)')
plt.ylabel('Quote ($)')
plt.title('Quote per Cntrl Disc');
plt.savefig('figures/configuration-case-control-disc-quote-per-cntrl-disc')
```

Configuration Case: Time Discretization

The purpose of this case is to determine if a sparse discretization of time is sufficient. To that end, we test four discretizations. The individual results are shown in figures B.4 and B.5. In figure B.6 we see that increasing the granularity of the discretization does not change the quote.

![Figure B.4: Time Discretization, Value Function at t=0](image)

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import hjb
```

124
Figure B.5: Time Discretization, Value Function per Initial Investment

Figure B.6: Time Discretization, Quote per Time Discretization
import spline

%matplotlib inline

'''
A helper function that computes the mark-to-model for a given range in variable 'y', with 't' and 'x' already fixed. Recall that 'y' is the initial investment and that we are looking for the smallest 'y' with a sufficiently small residual.

'''

```
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```

The details of the drift coefficient function. 'drift' has one row per regime.
drift = np.array([0.08, -0.05])

def b(t, x, y, v, i):
    return np.array([drift[i]*x, v*drift[i]*y + (1-v)*r*y])

The details of the diffusion coefficient function. ‘diff’ has one row per regime, and one column per Brownian motion.

D = 2

diff = np.array([[0.05, 0.15], [0.03, 0.10]])

def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]], [v*df*y for df in diff[i]]])

The details of the jump coefficient function. ‘jump’ has one row per regime, and one column per compensated Poisson. ‘lam’ also has one column per compensated Poisson.

jump = np.array([[0.05, 0.10], [-0.05, -0.15]])

def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]], [v*jmp*y for jmp in jump[i]]])

lam = np.array([3.0, 2.0])

The details of the regime switching coefficient function. ‘switch’ and ‘G’ must be square, and the rows of ‘G’ must sum to 0.

switch = np.array([[0.0, -0.02], [0.01, 0.0]])
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])

G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])

The configuration of the numerical scheme.

```python
steps_per_annum = 100  # redefine below as part of the validation
underlying_width = 10
max_premium = 10
control_granularity = 5
tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width, underlying_initial+underlying_width, 2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```python
steps_per_annums = np.array([80, 90, 100, 110])
value_functions = []
for steps_per_annum in steps_per_annums:
    tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
    value_functions.append(hjb.value_function_factory(h,
                                                      g,
                                                      b,
                                                      D,
                                                      sig,
                                                      L,
                                                      gam,
                                                      lam,
                                                      M,
                                                      128)
The plots showing for each validation case the value function at $t = 0$ (which implies regime $= 0$) for given ranges in ‘$y$‘ and ‘$x$‘. Recall that ‘$y$‘ (horizontal axis) represents the range of initial investments and ‘$x$‘ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial‘ on the vertical axis and scan right until you hit the ‘threshold‘ contour, and then scan down to the horizontal axis.

```python
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Time Disc = ' + str(steps_per_annums[n]+1) + '(pt)')
plt.tight_layout()
plt.savefig('figures/configuration-case-time-disc-value-function-at-t-equal-0')
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(eta,
    G,
    tt,
    xx,
    yy,
    vv))
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(eta,
    G,
    tt,
    xx,
    yy,
    vv))
```
yy,
np.array([value_function(0,
    underlying_width,
    k,
    0)[0]
    for k in np.arange(yy.size)])[0]
quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='-')
plt.axhline(y=0.0, color='r', linestyle='-')
plt.title('Time Disc = ' + str(steps_per_annums[n]+1) + '(pt)')
plt.tight_layout()
plt.savefig('figures/configuration-case-time-disc-value-function-per-
initial-investment')

A plot showing the results of the validation case.

```
```
```
```
```
```
Validation Case: Diffusion

The purpose of this case is to determine if the model responds appropriately to different diffusions. To that end, we test four diffusions. The individual results are shown in figures B.7 and B.8. In figure B.9 we see that increasing the diffusion increases the quote, as expected.

![Diffusion, Value Function at t=0](image)

Figure B.7: Diffusion, Value Function at t=0

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import hjb
```


Figure B.8: Diffusion, Value Function per Initial Investment

Figure B.9: Diffusion, Quote per Diffusion
import spline

import matplotlib

'''
A helper function that computes the mark-to-model for a given range in variable 'y', with 't' and 'x' already fixed. Recall that 'y' is the initial investment and that we are looking for the smallest 'y' with a sufficiently small residual.
'''

def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop

'''
The details of the market.
'''

'''
underlying_initial = 20
r = 0.02
'''

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

'''
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
'''

The details of the drift coefficient function. 'drift' has one row per regime.
```
```
```
```
```
```
```
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])

G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])

The configuration of the numerical scheme.

```
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5

tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                underlying_initial+underlying_width,
                2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```
diffs = np.array([[0.03, 0.03],
                  [0.03, 0.03],
                  [0.06, 0.06],
                  [0.06, 0.06],
                  [0.09, 0.09],
                  [0.09, 0.09],
                  [0.12, 0.12],
                  [0.12, 0.12]])

value_functions = []
for diff in diffs:
    value_functions.append(hjb.value_function_factory(h,
                                                        g,
                                                        b,
                                                        D,
                                                        sig,
                                                        ...)}
The plots showing for each validation case the value function at $t = 0$ (which implies regime = 0) for given ranges in ‘$y$’ and ‘$x$’. Recall that ‘$y$’ (horizontal axis) represents the range of initial investments and ‘$x$’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial’ on the vertical axis and scan right until you hit the ‘threshold’ contour, and then scan down to the horizontal axis.

```python
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Diff = ' + str(diffs[n][0][0]*100) + '%')
plt.tight_layout()
plt.savefig('figures/validation-case-diffusion-value-function-at-t-equal-0')
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
```

The plots showing for each validation case the decrease in the value function as initial investment increases.
```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
        yy,
        np.array([value_function(0,
            underlying_width,
            k,
            0)[0]
            for k in np.arange(yy.size)]))[0]
    quotes.append(mtm(yy[0], yy[-1], initial_values))
    plt.subplot(221+n)
    X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
    Y = np.array([initial_values(x) for x in X])
    plt.plot(X, Y)
    plt.axvline(x=quotes[n], color='r', linestyle='--')
    plt.axhline(y=0.0, color='r', linestyle='--')
    plt.title('Diff = ' + str(diffs[n][0][0]*100) + '%')
    plt.tight_layout()
    plt.savefig('figures/validation-case-diffusion-value-function-per-initial-investment')

A plot showing the results of the validation case.

````
```python
[diff[0][0]*100 for diff in diffs]
````
```
```
````
```python
quotes
````
```
```
````
```python
X = np.array([diff[0][0]*100 for diff in diffs])
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('Diff (%)')
plt.ylabel('Quote ($')
plt.title('Quote per Diff');`````
Validation Case: Expiry

The purpose of this case is to determine if the model responds appropriately to different expiries. To that end, we test four expiries. The individual results are shown in figures B.10 and B.11. In figure B.12 we see that increasing the expiry increases the quote, as expected.

Figure B.10: Expiry, Value Function at t=0

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
```

```
```
Figure B.11: Expiry, Value Function per Initial Investment

Figure B.12: Expiry, Quote per Expiry
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

import hjb
import spline

%matplotlib inline

A helper function that computes the mark-to-model for a given range in variable ‘y’, with ‘t’ and ‘x’ already fixed. Recall that ‘y’ is the initial investment and that we are looking for the smallest ‘y’ with a sufficiently small residual.

```
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop

The details of the market.

```

```
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```
expiry = 6.0/12.0  # redefine below as part of the validation
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
```
return max(x-strike-y, 0)

The details of the drift coefficient function. ‘drift‘ has one row per regime.

```python
drift = np.array([0.08,
                  -0.05])
def b(t, x, y, v, i):
    return np.array([drift[i]*x,
                     v*drift[i]*y + (1-v)*r*y])
```

The details of the diffusion coefficient function. ‘diff‘ has one row per regime, and one column per Brownian motion.

```python
D = 2
diff = np.array([[0.05, 0.15],
                 [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])
```

The details of the jump coefficient function. ‘jump‘ has one row per regime, and one column per compensated Poisson. ‘lam‘ also has one column per compensated Poisson.

```python
L = 2
jump = np.array([[0.05, 0.10],
                 [-0.05, -0.15]])
def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],
                     [v*jmp*y for jmp in jump[i]]])
lam = np.array([3.0, 2.0])
```

The details of the regime switching coefficient function. ‘switch‘ and ‘G‘ must be square, and the rows of ‘G‘ must sum to 0.
\begin{verbatim}
\texttt{python}
M = 2
switch = np.array([[0.0, -0.02],
                   [0.01, 0.0]])
def eta(t, x, y, v, i):
    return np.array([sw*x for sw in switch[i]],
                    [v*sw*y for sw in switch[i]])
G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])
\texttt{``}

The configuration of the numerical scheme.

\begin{verbatim}
\texttt{python}
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5

# redefine below as part of the validation

tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                underlying_initial+underlying_width,
                2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
\texttt{``}

The definition of the validation case, and corresponding solutions of the HJB

equation. WARNING: this is where all the math is so it will take a while to
execute.

\begin{verbatim}
\texttt{python}
expiries = np.array([2.0/12.0, 4.0/12.0, 6.0/12.0, 8.0/12.0])
value_functions = []
for expiry in expiries:
    tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
    value_functions.append(hjb.value_function_factory(h,
                                                      g,
                                                      b,
                                                      D,
                                                      sig,
\end{verbatim}
\end{verbatim}
The plots showing for each validation case the value function at $t = 0$ (which implies regime = 0) for given ranges in ‘y’ and ‘x’. Recall that ‘y’ (horizontal axis) represents the range of initial investments and ‘x’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial’ on the vertical axis and scan right until you hit the ‘threshold‘ contour, and then scan down to the horizontal axis.

```
plt.figure()
for n, value_function in enumerate(value_functions):
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Expiry = ' + str(round(expiries[n],2)) + ' (Y)')
plt.tight_layout()
plt.savefig('figures/validation-case-expiry-value-function-at-t-equal-0')
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```
plt.figure()
```
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
        yy,
        np.array([value_function(0,
                     underlying_width,
                     k,
                     0)[0]
               for k in np.arange(yy.size)]))[0]
    quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='--')
plt.axhline(y=0.0, color='r', linestyle='--')
plt.title('Expiry = ' + str(round(expiries[n],2)) + ' (Y)')
plt.tight_layout()
plt.savefig('figures/validation-case-expiry-value-function-per-initial-investment')
'''
A plot showing the results of the validation case.

'''

'python
expiries
'''

'python
quotes
'''

'python
X = np.array(expiries)
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('Expiry (Y)')
plt.ylabel('Quote ($)')
plt.title('Quote per Expiry');
plt.savefig('figures/validation-case-expiry-quote-per-expiry')
Validation Case: Interest Rate

The purpose of this case is to determine if the model responds appropriately to different interest rates. To that end, we test four interest rates. The individual results are shown in figures B.13 and B.14. In figure B.15 we see that increasing the interest rates does not change the quote, as expected.

![Figure B.13: Interest Rate, Value Function at t=0](image)

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```
Figure B.14: Interest Rate, Value Function per Initial Investment

Figure B.15: Interest Rate, Quote per Interest Rate
import numpy as np
import hjb
import spline

%matplotlib inline

A helper function that computes the mark-to-model for a given range in variable ‘y’, with ‘t’ and ‘x’ already fixed. Recall that ‘y’ is the initial investment and that we are looking for the smallest ‘y’ with a sufficiently small residual.

```python
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```python
underlying_initial = 20
r = 0.02   # redefine below as part of the validation
```

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```python
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```
""

The details of the drift coefficient function. 'drift' has one row per regime.

```
python
drift = np.array([[0.08, 
                  -0.05]])
def b(t, x, y, v, i):
    return np.array([[drift[i]*x, 
                      v*drift[i]*y + (1-v)*r*y]])
```

""

The details of the diffusion coefficient function. 'diff' has one row per regime, and one column per Brownian motion.

```
python
D = 2
diff = np.array([[0.05, 0.15],
                  [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])
```

""

The details of the jump coefficient function. 'jump' has one row per regime, and one column per compensated Poisson. 'lam' also has one column per compensated Poisson.

```
python
L = 2
jump = np.array([[0.05, 0.10],
                  [-0.05, -0.15]])
def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],
                     [v*jmp*y for jmp in jump[i]]])

lam = np.array([3.0, 2.0])
```

""

The details of the regime switching coefficient function. 'switch' and 'G' must be square, and the rows of 'G' must sum to 0.

```
python

```

148
M = 2
switch = np.array([[0.0, -0.02],
                   [0.01, 0.0]])
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])
G = np.array([[[-0.2, 0.2],
               [4.0, -4.0]])

The configuration of the numerical scheme.

```
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5
tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                 underlying_initial+underlying_width,
                 2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```
rs = np.array([0.01, 0.02, 0.03, 0.04])
value_functions = []
for r in rs:
    value_functions.append(hjb.value_function_factory(h,
                                                      g,
                                                      b,
                                                      D,
                                                      sig,
                                                      L,
                                                      gam,
                                                      lam,
                                                      M,
                                                      r))
```
The plots showing for each validation case the value function at $t = 0$ (which implies regime = 0) for given ranges in ‘y’ and ‘x’. Recall that ‘y’ (horizontal axis) represents the range of initial investments and ‘x’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying\_initial’ on the vertical axis and scan right until you hit the ‘threshold’ contour, and then scan down to the horizontal axis.

```python
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('IR = ' + str(rs[n]*100*100) + 'bp')
plt.tight_layout()
plt.savefig('figures/validation-case-interest-rate-value-function-at-t-equal-0')
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
```
yy,
np.array([value_function(0,
    underlying_width,
    k,
    0)[0]
    for k in np.arange(yy.size)])[0]
quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='-')
plt.axhline(y=0.0, color='r', linestyle='-')
plt.title('IR = ' + str(rs[n]*100*100) + 'bp')
plt.tight_layout()
plt.savefig('figures/validation-case-interest-rate-value-function-per-initial-investment')

A plot showing the results of the validation case.

"""python
rs
"""

"""python
quotes
"""

"""python
X = np.array(rs*100*100)
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('RFR (bp)')
plt.ylabel('Quote ($)')
plt.title('Quote per RFR');
plt.savefig('figures/validation-case-interest-rate-quote-per-rfr')
"""
Validation Case: Jump Frequency

The purpose of this case is to determine if the model responds appropriately to different jump frequencies. To that end, we test four jump frequencies. The individual results are shown in figures B.16 and B.17. In figure B.18 we see that increasing the jump frequencies increases the quote, as expected.

Figure B.16: Jump Frequency, Value Function at t=0

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```

---

validation-case-jump-frequency.md

---

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```
Figure B.17: Jump Frequency, Value Function per Initial Investment

Figure B.18: Jump Frequency, Quote per Jump Frequency
import hjb
import spline

%matplotlib inline

A helper function that computes the mark-to-model for a given range in variable ‘y’, with ‘t’ and ‘x’ already fixed. Recall that ‘y’ is the initial investment and that we are looking for the smallest ‘y’ with a sufficiently small residual.

```python
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```python
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```python
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```
The details of the drift coefficient function. ‘drift’ has one row per regime.

```python
drift = np.array([[0.08,
                   -0.05]])
def b(t, x, y, v, i):
    return np.array([drift[i]*x,
                     v*drift[i]*y + (1-v)*r*y])
```

The details of the diffusion coefficient function. ‘diff’ has one row per regime, and one column per Brownian motion.

```python
D = 2
diff = np.array([[0.05, 0.15],
                 [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])
```

The details of the jump coefficient function. ‘jump’ has one row per regime, and one column per compensated Poisson. ‘lam’ also has one column per compensated Poisson.

```python
L = 2
jump = np.array([[0.05, 0.10],
                 [-0.05, -0.15]])
def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],
                     [v*jmp*y for jmp in jump[i]]])
lam = np.array([3.0, 2.0])  # redefine below as part of the validation
```

The details of the regime switching coefficient function. ‘switch’ and ‘G’ must be square, and the rows of ‘G’ must sum to 0.

```python
M = 2
switch = np.array([[0.0, -0.02],
                   [-0.02, 0.0]])
```
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])

G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])

The configuration of the numerical scheme.

```python
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5
```

```python
tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                 underlying_initial+underlying_width,
                 2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```python
lams = np.array([[1.0, 1.0],
                 [2.0, 2.0],
                 [3.0, 3.0],
                 [4.0, 4.0]])
```

```python
value_functions = []
for lam in lams:
    value_functions.append(hjb.value_function_factory(h,
                                                        g,
                                                        b,
                                                        D,
                                                        sig,
                                                        L,
                                                        gam,
                                                        lam,
                                                        )
```
The plots showing for each validation case the value function at $t = 0$ (which implies regime = 0) for given ranges in ‘y‘ and ‘x‘. Recall that ‘y‘ (horizontal axis) represents the range of initial investments and ‘x‘ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial‘ on the vertical axis and scan right until you hit the ‘threshold‘ contour, and then scan down to the horizontal axis.

```python
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Jmp Freq = ' + str(lams[n][0]) + 'per Y')
plt.tight_layout()
plt.savefig('figures/validation-case-jump-frequency-value-function-at-t-equal-0')
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
```
initial_values = spline.cubic_spline_factory(
    yy,
    np.array([value_function(0,
               underlying_width,
               k,
               0)[0]
               for k in np.arange(yy.size)])[0]
) 
quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='--')
plt.axhline(y=0.0, color='r', linestyle='--')
plt.title('Jmp Freq = ' + str(lams[n][0]) + 'per Y')
plt.tight_layout()
plt.savefig('figures/validation-case-jump-frequency-value-function-per-initial-investment')

A plot showing the results of the validation case.

[lam[0] for lam in lams]

quotes

X = np.array([lam[0] for lam in lams])
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('Jmp Freq (per Y)')
plt.ylabel('Quote ($)')
plt.title('Quote per Jmp Freq');
plt.savefig('figures/validation-case-jump-frequency-quote-per-jmp-freq')

"""
Validation Case: Jump Intensity

The purpose of this case is to determine if the model responds appropriately to different jump intensities. To that end, we test four jump intensities. The individual results are shown in figures B.19 and B.20. In figure B.21 we see that increasing the jump intensities increases the quote, as expected.

Figure B.19: Jump Intensity, Value Function at t=0

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import hjb
```
Figure B.20: Jump Intensity, Value Function per Initial Investment

Figure B.21: Jump Intensity, Quote per Jump Intensity
import spline

%matplotlib inline

'''
A helper function that computes the mark-to-model for a given range in variable 'y', with 't' and 'x' already fixed. Recall that 'y' is the initial investment and that we are looking for the smallest 'y' with a sufficiently small residual.
'''

```python
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```python
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```python
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```

The details of the drift coefficient function. 'drift' has one row per regime.
The details of the diffusion coefficient function. ‘diff‘ has one row per regime, and one column per Brownian motion.

The details of the jump coefficient function. ‘jump‘ has one row per regime, and one column per compensated Poisson. ‘lam‘ also has one column per compensated Poisson.

The details of the regime switching coefficient function. ‘switch‘ and ‘G‘ must be square, and the rows of ‘G‘ must sum to 0.
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])

G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])

The configuration of the numerical scheme.

```
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5

tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                underlying_initial+underlying_width,
                2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```
jumps = np.array([[0.03, 0.03],
                  [0.03, 0.03],
                  [0.06, 0.06],
                  [0.06, 0.06],
                  [0.09, 0.09],
                  [0.09, 0.09],
                  [0.12, 0.12],
                  [0.12, 0.12]])

value_functions = []
for jump in jumps:
    value_functions.append(hjb.value_function_factory(h, g, b, D, sig, 
                                                      jump, 0))
The plots showing for each validation case the value function at \( t = 0 \) (which implies regime = 0) for given ranges in ‘y’ and ‘x’. Recall that ‘y’ (horizontal axis) represents the range of initial investments and ‘x’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial’ on the vertical axis and scan right until you hit the ‘threshold’ contour, and then scan down to the horizontal axis.

```
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Jmp Int = ' + str(jumps[n][0][0]) + '%')
plt.tight_layout()
plt.savefig('figures/validation-case-jump-intensity-value-function-at-t-equal-0')
```

The plots showing for each validation case the decrease in the value function as initial investment increases.

```
```
```
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
        yy,
        np.array([value_function(0, underlying_width, k, 0)[0]
            for k in np.arange(yy.size)]))[0]
    quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='--')
plt.axhline(y=0.0, color='r', linestyle='--')
plt.title('Jmp Int = ' + str(jumps[n][0][0]) + '%')
plt.tight_layout()
plt.savefig('figures/validation-case-jump-intensity-value-function-per-initial-investment')

A plot showing the results of the validation case.

```
[jump[0][0] for jump in jumps]
```

```
quotes
```

```
X = np.array([jump[0][0] for jump in jumps])
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('Jmp Int (%)')
plt.ylabel('Quote ($)')
plt.title('Quote per Jmp Int');

165
Validation Case: Regime Frequency

The purpose of this case is to determine if the model responds appropriately to different regime-switching frequencies. To that end, we test four regime-switching frequencies. The individual results are shown in figures B.22 and B.23. In figure B.24 we see that increasing frequency of transitioning to a regime with lower volatility decreases the quote, as expected.

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math
```

Figure B.22: Regime Frequency, Value Function at t=0
Figure B.23: Regime Frequency, Value Function per Initial Investment

Figure B.24: Regime Frequency, Quote per Regime Frequency Scenario
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import hjb
import spline

import matplotlib inline
'''

A helper function that computes the mark-to-model for a given range in variable
'y', with 't' and 'x' already fixed. Recall that 'y' is the initial
investment and that we are looking for the smallest 'y' with a sufficiently
small residual.

'''

def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop

'''

The details of the market.

'''

underlying_initial = 20
r = 0.02
'''

The details of the instrument. There is both a path-dependent and final-time
component of the payoff.

'''

expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):

```python
return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```

The details of the drift coefficient function. ‘drift’ has one row per regime.

```python
drift = np.array([[0.08,
                   -0.05]])
def b(t, x, y, v, i):
    return np.array([drift[i]*x,
                     v*drift[i]*y + (1-v)*r*y])
```

The details of the diffusion coefficient function. ‘diff’ has one row per regime, and one column per Brownian motion.

```python
D = 2
diff = np.array([[0.05, 0.15],
                 [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])
```

The details of the jump coefficient function. ‘jump’ has one row per regime, and one column per compensated Poisson. ‘lam’ also has one column per compensated Poisson.

```python
L = 2
jump = np.array([[0.05, 0.10],
                  [-0.05, -0.15]])
def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],
                     [v*jmp*y for jmp in jump[i]]])
lam = np.array([3.0, 2.0])
```

The details of the regime switching coefficient function. ‘switch‘ and ‘G’
must be square, and the rows of \(G\) must sum to 0.

```python
M = 2
switch = np.array([[0.0, -0.02],
                   [0.01, 0.0]])
def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])
G = np.array([[-0.2, 0.2],  # redefine below as part of the validation
              [4.0, -4.0]])
```

The configuration of the numerical scheme.

```python
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5
tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                 underlying_initial+underlying_width,
                 2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)
```

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

```python
Gs = np.array([[[[-1.0, 1.0],
                 [4.0, -4.0]],
                [-2.0, 2.0],
                [3.0, -3.0]],
               [[-3.0, 3.0],
                [2.0, -2.0]],
               [[-4.0, 4.0],
                [1.0, -1.0]]])
value_functions = []
```
for G in Gs:
    value_functions.append(hjb.value_function_factory(h,
        g,
        b,
        D,
        sig,
        L,
        gam,
        lam,
        M,
        eta,
        G,
        tt,
        xx,
        yy,
        vv))

""

The plots showing for each validation case the value function at t = 0 (which
implies regime = 0) for given ranges in ‘y’ and ‘x’. Recall that ‘y’
(horizontal axis) represents the range of initial investments and ‘x’
(horizontal axis) represents the range of initial underlyings. To determine
the value of the derivative visually, find ‘underlying_initial‘ on the vertical
axis and scan right until you hit the ‘threshold‘ contour, and then scan down
to the horizontal axis.

""

plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j, k in np.ndindex(xx.size, yy.size):
        VAL_0[j, k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('RS Freq Sc #' + str(n))
    plt.tight_layout()
    plt.savefig('figures/validation-case-regime-frequency-value-function-at-t-
equal-0')

171
The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
        yy,
        np.array([value_function(0,
                     underlying_width,
                     k,
                     0)[0]
                  for k in np.arange(yy.size)])[0]
    quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='-')
plt.axhline(y=0.0, color='r', linestyle='-')
plt.title('RS Freq Sc #' + str(n))
plt.tight_layout()
plt.savefig('figures/validation-case-regime-frequency-value-function-per-initial-investment')
```

A plot showing the results of the validation case.

```python
[1, 2, 3, 4]
```

```python
quotes
```

```python
X = np.array([1, 2, 3, 4])
```
Validation Case: Regime Intensity

The purpose of this case is to determine if the model responds appropriately to different regime-switching intensities. To that end, we test four regime-switching intensities. The individual results are shown in figures B.25 and B.26. In figure B.27 we see that increasing the intensity of transitioning to other regimes increases the quote, as expected.

Figure B.25: Regime Intensity, Value Function at $t=0$
Figure B.26: Regime Intensity, Value Function per Initial Investment

Figure B.27: Regime Intensity, Quote per Regime Intensity Scenario
The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```python
import math

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import hjb
import spline

%matplotlib inline

A helper function that computes the mark-to-model for a given range in variable 'y', with 't' and 'x' already fixed. Recall that 'y' is the initial investment and that we are looking for the smallest 'y' with a sufficiently small residual.

```python
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```python
underlying_initial = 20
r = 0.02
```

The details of the instrument. There is both a path-dependent and final-time
component of the payoff.

```
expiry = 6.0/12.0
strike = underlying_initial
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```

The details of the drift coefficient function. ‘drift‘ has one row per regime.

```
drift = np.array([[0.08, -0.05]])
def b(t, x, y, v, i):
    return np.array([[drift[i]*x,
                     v*drift[i]*y + (1-v)*r*y]])
```

The details of the diffusion coefficient function. ‘diff‘ has one row per regime, and one column per Brownian motion.

```
D = 2
diff = np.array([[0.05, 0.15],
                 [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])
```

The details of the jump coefficient function. ‘jump‘ has one row per regime, and one column per compensated Poisson. ‘lam‘ also has one column per compensated Poisson.

```
L = 2
jump = np.array([[0.05, 0.10],
                  [-0.05, -0.15]])
def gam(t, x, y, v, i):
\[
\text{return np.array([[jmp\times for jmp in jump[i]],}
\quad [v\times jmp\times y for jmp in jump[i]])}
\]

\[
lam = \text{np.array([[3.0, 2.0]])}
\]

""

The details of the regime switching coefficient function. ‘switch’ and ‘G’ must be square, and the rows of ‘G’ must sum to 0.

""

```
M = 2
switch = np.array([[0.0, -0.02], # redefine below as part of the validation
                   [0.01, 0.0]])

def eta(t, x, y, v, i):
    return np.array([[sw\times x for sw in switch[i]],
                     [v\times sw\times y for sw in switch[i]]])

G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])
```

""

The configuration of the numerical scheme.

""

```
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5

\[tt = \text{np.linspace}(0, \text{expiry}, \text{math.floor}(\text{steps_per_annum}\times\text{expiry}+1))\]

\[xx = \text{np.linspace}(\text{underlying_initial}-\text{underlying_width},
                         \text{underlying_initial+underlying_width},
                         2\times\text{underlying_width}+1)\]

\[yy = \text{np.linspace}(0, \text{max_premium}, \text{math.floor}(\text{max_premium}+1))\]

\[vv = \text{np.linspace}(0, 1, \text{control_granularity}+1)\]
```

""

The definition of the validation case, and corresponding solutions of the HJB equation. WARNING: this is where all the math is so it will take a while to execute.

""

```
switches = np.array([[\[0.00, -0.03],
                      \[0.03, 0.00]],
                     \[0.00, -0.06],
                     \[0.00, -0.06],
                     \[0.00, -0.06]],
                     \[0.00, -0.06]])
```

177
for switch in switches:
    value_functions.append(hjb.value_function_factory(h,
              g,
              b,
              D,
              sig,
              L,
              gam,
              lam,
              M,
              eta,
              G,
              tt,
              xx,
              yy,
              vv))

```

The plots showing for each validation case the value function at t = 0 (which implies regime = 0) for given ranges in ‘y’ and ‘x’. Recall that ‘y’ (horizontal axis) represents the range of initial investments and ‘x’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial‘ on the vertical axis and scan right until you hit the ‘threshold‘ contour, and then scan down to the horizontal axis.

```

```python
for n, value_function in enumerate(value_functions):
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
The plots showing for each validation case the decrease in the value function as initial investment increases.

```python
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
        yy,
        np.array([value_function(0,
                      underlying_width,
                      k,
                      0)[0]
                      for k in np.arange(yy.size)]))[0]
    quotes.append(mtm(yy[0], yy[-1], initial_values))

plt.subplot(221+n)
X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
Y = np.array([initial_values(x) for x in X])
plt.plot(X, Y)
plt.axvline(x=quotes[n], color='r', linestyle='--')
plt.axhline(y=0.0, color='r', linestyle='--')
plt.title('RS Int Sc #' + str(n))
plt.tight_layout()
plt.savefig('figures/validation-case-regime-intensity-value-function-per-initial-investment')
```

A plot showing the results of the validation case.

```python
[1, 2, 3, 4]
```
Validation Case: Strike

The purpose of this case is to determine if the model responds appropriately to different strikes. To that end, we test four strikes. The individual results are shown in figures B.28 and B.29. In figure B.30 we see that increasing the strike decreases the quote, as expected.

The following listing gives the Jupyter notebook (in Markdown format) for this configuration case.

```
```

A helper function that computes the mark-to-model for a given range in variable ‘y’, with ‘t’ and ‘x’ already fixed. Recall that ‘y’ is the initial investment and that we are looking for the smallest ‘y’ with a sufficiently
Figure B.28: Strike, Value Function at $t=0$

Figure B.29: Strike, Value Function per Initial Investment
small residual.

```python
threshold = 0.05
def mtm(start, stop, residuals):
    premium = start
    while premium < stop:
        if residuals(premium) < threshold:
            return premium
        else:
            premium += 0.01
    return stop
```

The details of the market.

```python
underlying_initial = 20
r = 0.02
```
The details of the instrument. There is both a path-dependent and final-time component of the payoff.

```python
expiry = 6.0/12.0
strike = underlying_initial  # redefine below as part of the validation
def h(t, x, y, v, i):
    return max(x-strike-y, 0)
def g(x, y, i):
    return max(x-strike-y, 0)
```

The details of the drift coefficient function. ‘drift’ has one row per regime.

```python
drift = np.array([0.08, -0.05])
def b(t, x, y, v, i):
    return np.array([drift[i]*x,
                     v*drift[i]*y + (1-v)*r*y])
```

The details of the diffusion coefficient function. ‘diff’ has one row per regime, and one column per Brownian motion.

```python
D = 2
diff = np.array([[0.05, 0.15],
                [0.03, 0.10]])
def sig(t, x, y, v, i):
    return np.array([[df*x for df in diff[i]],
                     [v*df*y for df in diff[i]]])
```

The details of the jump coefficient function. ‘jump’ has one row per regime, and one column per compensated Poisson. ‘lam’ also has one column per compensated Poisson.

```python
L = 2
```
jump = np.array([[0.05, 0.10],
                 [-0.05, -0.15]])

def gam(t, x, y, v, i):
    return np.array([[jmp*x for jmp in jump[i]],
                     [v*jmp*y for jmp in jump[i]]])

lam = np.array([3.0, 2.0])

"""
The details of the regime switching coefficient function. ‘switch’ and ‘G’
must be square, and the rows of ‘G’ must sum to 0.
"""

"""python
M = 2
switch = np.array([[0.0, -0.02],
                   [0.01, 0.0]])

def eta(t, x, y, v, i):
    return np.array([[sw*x for sw in switch[i]],
                     [v*sw*y for sw in switch[i]]])

G = np.array([[-0.2, 0.2],
              [4.0, -4.0]])

"""
The configuration of the numerical scheme.

"""python
steps_per_annum = 100
underlying_width = 10
max_premium = 10
control_granularity = 5

tt = np.linspace(0, expiry, math.floor(steps_per_annum*expiry)+1)
xx = np.linspace(underlying_initial-underlying_width,
                 underlying_initial+underlying_width,
                 2*underlying_width+1)
yy = np.linspace(0, max_premium, math.floor(max_premium)+1)
vv = np.linspace(0, 1, control_granularity+1)

"""
The definition of the validation case, and corresponding solutions of the HJB
equation. WARNING: this is where all the math is so it will take a while to
execute.

"""python
strikes = np.array([19, 20, 21, 22])
value_functions = []
for strike in strikes:
    value_functions.append(hjb.value_function_factory(h,
g,
b,
D,
sig,
L,
gam,
lam,
M,
eta,
G,
tt,
xx,
yy,
vv))

The plots showing for each validation case the value function at t = 0 (which implies regime = 0) for given ranges in ‘y’ and ‘x’. Recall that ‘y’ (horizontal axis) represents the range of initial investments and ‘x’ (horizontal axis) represents the range of initial underlyings. To determine the value of the derivative visually, find ‘underlying_initial’ on the vertical axis and scan right until you hit the ‘threshold’ contour, and then scan down to the horizontal axis.

```python
plt.figure()
for n, value_function in enumerate(value_functions):
    plt.subplot(221+n)
    YY_0, XX_0 = np.meshgrid(yy, xx)
    VAL_0 = np.zeros((xx.size, yy.size))
    for j,k in np.ndindex(xx.size, yy.size):
        VAL_0[j,k] = value_function(0, j, k, 0)[0]
    levels_0 = [threshold, 2, 4, 6, 8, 10, 12, 14]
    plt.contourf(YY_0, XX_0, VAL_0, levels_0, alpha=.75, cmap=plt.cm.hot)
    C = plt.contour(YY_0, XX_0, VAL_0, levels_0, colors='black')
    plt.clabel(C, inline=1, fontsize=10)
    plt.title('Strike = ' + str(strikes[n]) + '$')
plt.tight_layout()
```
plt.savefig('figures/validation-case-strike-value-function-at-t-equal-0')

The plots showing for each validation case the decrease in the value function as initial investment increases.

```
plt.figure()
quotes = []
for n, value_function in enumerate(value_functions):
    initial_values = spline.cubic_spline_factory(
        yy,
        np.array([value_function(0,
                      underlying_width,
                      k,
                      0)[0]
                      for k in np.arange(yy.size)])[0]
    quotes.append(mtm(yy[0], yy[-1], initial_values))

    plt.subplot(221+n)
    X = np.linspace(yy[0], yy[-1], math.floor(100*(yy[-1]-yy[0])))
    Y = np.array([initial_values(x) for x in X])
    plt.plot(X, Y)
    plt.axvline(x=quotes[n], color='r', linestyle='-')
    plt.axhline(y=0.0, color='r', linestyle='-')
    plt.title('Strike = ' + str(strikes[n]) + '($)')
plt.tight_layout()
plt.savefig('figures/validation-case-strike-value-function-per-initial-investment')
```

A plot showing the results of the validation case.

```
strikes
```

```
quotes
```

```
```
X = np.array(strikes)
Y = np.array(quotes)
plt.figure()
plt.plot(X, Y)
plt.xlabel('Strike ($)')
plt.ylabel('Quote ($)')
plt.title('Quote per Strike');
plt.savefig('figures/validation-case-strike-quote-per-strike')