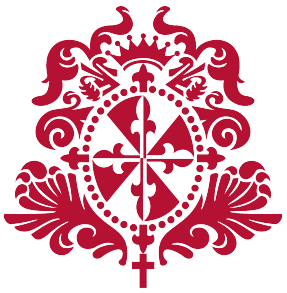


UNIVERSIDAD DEL ROSARIO

Soluciones Numéricas al Problema de Selección de Portafolio de Merton

Autor: Wilson Fernando Moreno Pirachican

Trabajo presentado como requisito para optar por el título de:
Mágister en Finanzas Cuantitativas

Director: Hugo E. Ramirez

Facultad de Economía
Maestría en Finanzas Cuantitativas
Universidad del Rosario

Bogotá, Colombia
2021

A Ingrid y mi familia.

Resumen. Supongamos que un agente con riqueza positiva desea invertir una proporción en un activo de riesgo y el resto en un bono. El problema consiste en escoger el porcentaje de riqueza óptimo que maximíce su utilidad al final del periodo de inversión. Este problema ya tiene solución analítica conseguida por Merton en las décadas de 1960 y 1970.
El propósito de este trabajo es presentar aportes de tipo númérico en la aproximación del portafolio óptimo que resuelve el problema de Merton. Para conseguir dicho propósito se plantean los siguientes objetivos:

- Proponer y comparar un esquema numérico similar al que presentó el autor Kafash en su artículo *Approximating the Solution of Stochastic Control Problems and the Merton's Portfolio Selection Model* en [11].
- Presentar un algoritmo basado en redes neuronales que prediga el valor del portafolio óptimo con datos simulados.

Por la versatilidad de una red neuronal, se elige este método para la predicción de portafolios óptimos con datos empíricos, en donde se mide su comportamiento y la posterior corrección mediante calibración. Esta idea se deja como trabajo posterior a lo presentado aquí.
Palabras Clave:Portafolio de Merton; Cadenas de Markov; Método Monte Carlo; Diferencias Finitas; Redes Neuronales.

Abstract. Suppose an agent with positive wealth wishes to invest a proportion in a risky asset and the rest in a bond. The problem is to choose the optimal wealth percentage that maximizes your utility at the end of the investment period. This problem already has an analytical solution achieved by Merton in the 1960s and 1970s.
The purpose of this work is to present numerical contributions in the approximation of the optimal portfolio that solves the Merton problem. To achieve this purpose, the following objectives are proposed:

- Propose and compare a numerical scheme similar to the one presented by the author Kafash in his article *Approximating the Solution of Stochastic Control Problems and the Merton's Portfolio Selection Model* in [11].
- Present an algorithm based on neural networks that predicts the value of the optimal portfolio with simulated data.

Due to the versatility of a neural network, this method is chosen for the prediction of optimal portfolios with empirical data, where its behavior and the subsequent correction through calibration are measured. This idea is left as work after what is presented here.
Keywords: Merton Portfolio; Markov Chains; Monte Carlo method; Finite differences; Neural Networks.

Contenido

Capítulo 1

Introducción

En este trabajo consideramos el problema de selección de portafolio de Merton. Este problema se plantea desde la pregunta, ¿cuáles deberían ser las proporciones de riqueza que un agente debe invertir entre un activo riesgoso y un bono para que su utilidad final sea máxima?, una exposición teórica se muestra en el capítulo 2, donde se puede encontrar el planteamiento y solución analítica. Debido a que bajo los supuestos trabajados en el planteamiento se permite una solución analítica, no siempre es el caso cuando modificamos dichos supuestos; se pueden asumir consideraciones que podrían hacer del problema algo más complejo y posiblemente, no se pueda obtener una solución con métodos algebraicos. Esto hace que sea necesario contar con herramientas numéricas para aproximar la solución.
En el capítulo 3, se plantea una situación específica y se exponen dos soluciones numéricas conocidas como *aproximaciones mediante cadenas de Markov*, donde se replican algunos resultados del artículo de Kafash [11] y además se propone un método adicional que tiene mejor aproximación. En el capítulo 4, formulamos el método de Monte Carlo y describimos dos algoritmos de optimización. Consideramos el método de diferencias finitas en el capítulo 5, aquí hacemos una transformación de la ecuación HJB que para este problema es tipo *backward*, a una tipo *forward*; y en una grilla espaciada uniformemente discretizamos las derivadas. En el capítulo 6 usamos dos algoritmos de optimización cuyo resultados denotarán los *outputs* de los cuales una red neuronal planteada aprenderá, todo con el fin de hacer predicciones del portafolio que maximiza la utilidad del agente.

Desde la publicación del problema propuesto y solucionado por Robert Merton en 1971, [17], varios autores han trabajado sobre variaciones del planteamiento inicial que suponen la resolución de dicho problema mediante técnicas numéricas. Nosotros consideramos enfoques computacionales encaminados a la solución numérica del problema girando en un contexto específico de un activo riesgoso y otro libre de riesgo, y bajo la naturaleza de ser un problema de tiempo continuo. Autores como Kushner y otros en 1991 [16], consideraron problemas de control óptimo estocástico más generales mediante la aproximación de la función de valor mediante cadenas de Markov; se trata de realizar un proceso iterativo hacia atrás de operaciones vectoriales. Bajo esta técnica, Yin y otros en 2009 [23], desarrollaron un procedimiento para la aproximación del problema con restricciones acotadas. También Krawczyk (2001) [15] examinó un método alternativo de derivar soluciones numéricas a una clase de problemas de control óptimo estocástico, entre ellos aproximación con cadenas de Markov al problema de Merton; Kafash (2019) en [11], mostró un algoritmo de aproximación de la función de valor mediante cadenas de Markov para un activo riesgoso y otro libre de riesgo.
En cuanto a las simulaciones de Monte Carlo, Detemple J. y otros en [6] en 2003, propusieron una forma de asignación de capitales basadas en simulaciones en entornos realistas y complejos de inversión con variables de estado dinámicas y un gran número de factores y activos. En el mismo año, Desai y otros en [5], implementan un algoritmo para la selección óptima de un portafolio que consta de un activo riesgoso y otro libre de riesgo bajo el modelo de volatilidad estocástica con observaciones discretas.
En el método de diferencias finitas, Kafash en [11] en 2019, usa en las aproximaciones por cadenas de Markov, usa un esquema explícito para la discretización de la ecuación de Hamilton Jacobi y Bellman (HJB) asociada al problema de selección de portafolio. En 2008 Min y Zhong [4] , presentaron un método de solución numérica que usa un esquema de diferencias finitas combinado con un método de penalidad con costos de transacción.

Por otro lado, en 1997 Steiner y Wittkemper en [22] usaron redes neuronales para estimar las distribuciones de los retornos para predecir básicamente el comportamiento de un grupo de activos en específico. Freitas y otros en [7], presentaron un modelo basado en la predicción de optimización de portafolios que pueden capturar oportunidades de inversión a corto plazo; usaron redes neuronales para predecir retornos derivando medidas de riesgo, constituido también en la predicción de errores que tienen un fundamento estadístico análogo al modelo de media varianza.

Capítulo 2

Problema de selección de portafolio

2.1 Introducción a control óptimo estocástico

En esta sección describimos una clase especial de problemas que involucran encontrar un valor máximo o mínimo de ciertas funciones que modelan fenómenos, ya sean determinísticos (cuya ocurrencia se puede predecir con un alto grado de probabilidad) o estocásticos (aquellos que están gobernados por el "azar"). Nos interesa plantear formalmente un problema que describe el porcentaje de riqueza óptimo que debe invertir un agente en un activo riesgoso (que conlleva en sí mismo una variable aleatoria que pueda hacer que gane o pierda) y un activo libre de riesgo (aquel que en teoría, significa una ganancia fija) con el fin de maximizar la utilidad en el tiempo final T. Para esto, inicialmente introducimos terminología relacionada con la teoría de cálculo estocástico y control óptimo estocástico en una dimensión, junto con el método de **programación dinámica** introducido por Bellman en 1957, [2].

2.1.1 Terminología y conceptos fundamentales de cálculo estocástico

En esta sección mencionamos, sin ir a profundidad de los detalles algebraicos, conceptos básicos relacionados con el cálculo estocástico. Es un breve recordatorio de conceptos que manejaremos en las siguientes secciones. El lector que desea un estudio más detallado de las fórmulas y definiciones a continuación, puede consultar el libro de Björk [3], el cual fue considerado para presentar el material de este capítulo.

Definición 2.1.1. *Un proceso estocástico W se llama **movimiento Browniano** si las siguientes condiciones se satisfacen:*

1. $W(0) = 0$.
2. *El proceso W tiene incrementos independientes, es decir, si $r < s \leq t < u$ entonces $W(u) - W(t)$ y $W(s) - W(r)$ son variables estocásticas independientes.*
3. *Para $s < t$ la variable estocástica $W(t) - W(s)$ tiene distribución normal $\mathcal{N}(0, \sqrt{t-s})$.*
4. *W tiene trayectorias continuas*[1].

El movimiento Browniano es un proceso usado para modelar cierto tipo de eventos que suceden aleatoreamente y que pueden ser considerados como normales.
Un concepto importante en cuanto a la información que se conozca desde un tiempo t_0 a un tiempo parcial t es el de filtración.

Definición 2.1.2. *Sea $\{X : t \geq 0\}$ un proceso estocástico definido sobre un espacio de probabilidad $(\Omega, \mathcal{F}, \mathbb{P})$. Se define la **sigma álgebra generada por X sobre el intervalo** $[0, t]$ como*

$$\mathcal{F}_t^X = \sigma\{X(s) : s \leq t\}.$$

[1] Es decir, la aplicación $t \longmapsto W(t)$ es continua.

Definición 2.1.3. *Una **filtración** $\{\mathcal{F}_t\}_{t\geq 0}$ (o $\mathcal{F}_t$) sobre un espacio de probabilidad $(\Omega, \mathcal{F}, \mathbb{P})$ es una familia indexada de sigma álgebras sobre Ω, tal que*

$$\mathcal{F}_t \subseteq \mathcal{F}, \text{ para todo } t \geq 0,$$
$$s \leq t \implies \mathcal{F}_s \subseteq \mathcal{F}_t.$$

Definición 2.1.4. *Consideremos un espacio de probabilidad $(\Omega, \mathcal{F}, \mathbb{R})$ y un proceso estocástico X sobre el mismo espacio. Decimos que X es **adaptado** a la filtración $\{\mathcal{F}_t\}_{t\geq 0}$ si*

$$X(t) \in \mathcal{F}_t, \qquad \text{para todo } t \geq 0.$$

En términos informales, el hecho de que X sea adaptado a $\mathcal{F}_t$ significa que para cada instante de tiempo t, el proceso $X(t)$ está completamente determinado por la información en $\mathcal{F}_t$ a la cual tenemos acceso en este instante.

2.1.2 Procesos de difusión

Definición 2.1.5. *Un proceso estocástico X es una **difusión** si su dinámica local puede ser aproximada a una ecuación diferencial estocástica del tipo*

$$\begin{aligned} dX(t) &= \mu(X(t), t)dt + \sigma(X(t), t)dW(t) \\ X(0) &= x, \end{aligned} \tag{2.1}$$

donde $W(t)$ es un movimiento Browniano en una dimensión, μ es velocidad determinística (o drift) y σ es el término de perturbación Gaussiano (o el término de difusión).

La ecuación (2.1) tiene la siguiente forma integral:

$$X(t) = x + \int_0^t \mu(X(s), s)ds + \int_0^t \sigma(X(s), s)dW(s), \tag{2.2}$$

donde la primera integral del lado derecho de la igualdad denota una integral de Lebesgue sobre la variable $s \in [0, t]$, mientras que la segunda integral denota una integral *estocástica* en el sentido que contiene el diferencial del movimiento Browniano $dW(s)$.

Finalizamos esta sección enunciando dos resultados importantes del cálculo estocástico: el proceso de Itô y la fórmula integral de Itô, los cuales nos servirán en nuestra exposición de control estocástico.

Definición 2.1.6. *Sea X un proceso estocástico sobre el espacio de probabilidad filtrado $(\Omega, \mathcal{F}, \{\mathcal{F}_t^W\}_{t\geq 0}, \mathbb{P})$ y suponga que existe un número real x y dos procesos μ y σ adaptados a $\mathcal{F}_t^W$ tal que es válida la siguiente igualdad*

$$X(t) = x + \int_0^t \mu(s)ds + \int_0^t \sigma(s)dW(s) \tag{2.3}$$

donde W es un movimiento Browniano. En forma diferencial, (2.3) tiene la siguiente expresión

$$\begin{aligned} dX(t) &= \mu(t)dt + \sigma(t)dW(t) \\ X(0) &= x. \end{aligned} \tag{2.4}$$

*En este caso decimos que X tiene una **ecuación diferencial estocástica** dada por (2.4) con condición inicial $X(0) = x$. Al proceso estocástico X lo llamamos **proceso de Itô**.*

Teorema 2.1.1. *(**Fórmula de Itô**) Asuma que X es un proceso de Itô dado por (2.4), donde μ, σ son procesos adaptados a la filtración $\mathcal{F}_t^X$, y sea f una función[2] $(C^{2,1})$. Entonces*

$$f(X(t), t) = f(x, 0) + \int_0^t \left(\frac{\partial f}{\partial t} + \mu \frac{\partial f}{\partial x} + \frac{1}{2}\sigma^2 \frac{\partial^2 f}{\partial x^2} \right) ds + \int_0^t \sigma \frac{\partial f}{\partial x} dW(s), \tag{2.5}$$

donde $X(0) = x$.

[2]Una función $C^{2,1}$ es una función de dos variables que es dos veces continuamente diferenciable en la primera variable y continuamente diferenciable en la segunda.

Ejemplo 2.1.1. *(**Movimiento Browniano Geométrico**). Un proceso de difusión de gran utilidad es el movimiento Browniano Geométrico, el cual se usa para modelar procesos de precios de activos riesgosos entre otras aplicaciones.*
*Un proceso $S(t)$ sigue un **Movimiento Browniano Geométrico** si es solución de la ecuación diferencial estocástica*

$$\begin{aligned} dS(t) &= \mu S(t)dt + \sigma S(t)dW(t), \\ S(0) &= S_0, \end{aligned} \tag{2.6}$$

donde $W(t)$ es un movimiento Browniano estándar en t, μ (porcentage de drift) y σ (porcentage de volatilidad) son constantes.
La solución de la ecuación diferencial estocástica (2.6) es dada por

$$S(t) = S_0 \exp\left\{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)\right\}.$$

Nótese que si $S_0 > 0$ entonces $S(t) > 0$ para todo $t \geq 0$.

□

2.1.3 Descripción del problema de control estocástico

Introducimos a continuación conceptos claves para entender la formulación de un problema de control óptimo estocástico en una dimensión[3]. Supondremos que $X(t)$ es un proceso estocástico conocido como **espacio de estados** (al cual nos referiremos más adelante como proceso de riqueza) sobre un espacio filtrado de probabilidad $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$. Supondremos que para un punto fijo $x_0 \in \mathbb{R}$, $X(t)$ tiene una representación en forma de ecuación diferencial estocástica **controlada**.

$$\begin{aligned} dX(t) &= \mu(X(t), t, u(t))dt + \sigma(X(t), t, u(t))dW(t), \\ X(0) &= x_0, \end{aligned} \tag{2.7}$$

donde W es un movimiento Browniano en una dimensión; y en principio, el drift μ y la perturbación σ se pueden considerar como funciones

$$\begin{aligned} \mu &: \mathbb{R} \times \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}, \\ \sigma &: \mathbb{R} \times \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}. \end{aligned}$$

El término "controlado" se refiere a la presencia de un nuevo proceso u. Esto significa que podemos controlar parcialmente la dinámica de $X(t)$ eligiendo un proceso de **control** u de forma adecuada. Ahora intentamos dar una definición matemática más precisa para μ y σ.
Para la formulación del problema requerimos que el control $u = u(t)$ sea adaptado a la filtración generada por el proceso $X(t)$; es decir, en el tiempo t se permite que el proceso de control u dependa de los valores que $X(t)$ obtuvo en el pasado. De forma natural se puede definir un proceso de control adaptado u como si fuera una función determinística u:

$$\begin{aligned} u &: \mathbb{R} \times \mathbb{R}_+ \to \mathbb{R} \\ u &= u(X(t), t) \end{aligned}$$

tal función se denomina un **control de retroalimentación** y en adelante, sólo consideramos procesos de control de este tipo y nos referiremos a ellos simplemente como controles. Usamos letra negrita para indicar que **u** es una función. Por otro lado, usamos usamos la notación u cursiva, para denotar el control en cierto tiempo.
Supóngase ahora que hemos encontrado el control $u(X(t), t)$, entonces podemos sustituirlo en la ecuación (2.7) para obtener la ecuación diferencial estocástica

$$dX(t) = \mu(X(t), t, \mathbf{u}(X(t), t))dt + \sigma(X(t), t, \mathbf{u}(X(t), t))dW(t) \tag{2.8}$$

En la mayoría de los casos, se deben satisfacer algunas **restricciones de control**, y por tanto, condicionamos a nuestro control u a tomar valores en cierto conjunto $U \subseteq \mathbb{R}$ tal que $u(X(t), t) \in U$ para cada t. Por tanto, definimos la clase de **controles admisibles**.

[3]Desde luego, hay una exposición multivariable en [3].

Definición 2.1.7. *Un control* $\boldsymbol{u}$ *es llamado* ***admisible*** *si*

- $\boldsymbol{u}(x,t) \in U$ *para todo* $t \in \mathbb{R}_+$ *y todo* $x \in \mathbb{R}$.
- *Para cualquier punto inicial* (x,t) *la ecuación diferencial estocástica*

$$\begin{aligned} dX(s) &= \mu(X(s),s,\boldsymbol{u}(X(s),s))ds + \sigma(X(s),s,\boldsymbol{u}(X(s),s))dW(s), \\ X(s) &= x \end{aligned}$$

  tiene una única solución.

- *Para todo* $k \in \mathbb{N}$ *se satisface que*

$$\mathbb{E}\left(\int_0^\infty |\boldsymbol{u}(X(s),s)|^k ds\right) < \infty$$

  y además[4]

$$\mathbb{E}^{x,t}(||X(\cdot)||^k) := \mathbb{E}^{x,t}\left(\sup_{s\in[0,\infty)} |X(s)|^k\right) < \infty.$$

  La clase de todos los controles admisibles se denota con $\mathcal{U}$.

Sea una ley de control $\mathbf{u}$ con un proceso controlado correspondiente X^u, entonces frecuentemente usaremos la notación u_t en lugar de usar $\mathbf{u}(X^u(t),t)$.
Para un control dado $\mathbf{u}$, el proceso solución X por supuesto dependerá del valor inicial x, también del control u. Para ser precisos, deberíamos denotar X por $X^{x,\mathbf{u}}$, pero a veces suprimimos x, $\mathbf{u}$ o ambos.
A modo de notación, consideremos la ecuación (2.8), entonces

- Para cualquier valor fijo $u \in \mathbb{R}$, las funciones μ^u, σ^u, C^u y F^u están definidas por

$$\begin{aligned} \mu^u(x,t) &= \mu(x,t,u(x,t)), \\ \sigma^u(x,t) &= \sigma(x,t,u(x,t)), \\ C^u(x,t) &= (\sigma(x,t,u(x,t)))^2. \end{aligned}$$

- Para cualquier control $\mathbf{u}$, las funciones $\mu^{\mathbf{u}}$, $\sigma^{\mathbf{u}}$, $C^{\mathbf{u}}(t.x)$ y $F^{\mathbf{u}}(x,t)$ están definidas por

$$\begin{aligned} \mu^{\mathbf{u}}(x,t) &= \mu(x,t,\mathbf{u}(x,t)), \\ \sigma^{\mathbf{u}}(x,t) &= \sigma(x,t,\mathbf{u}(x,t)), \\ C^{\mathbf{u}}(x,t) &= \sigma(x,t,\mathbf{u}(x,t))^2, \\ F^{\mathbf{u}}(x,t) &= F(x,t,\mathbf{u}(x,t)). \end{aligned}$$

- Para cualquier valor fijo u, el operador de derivadas parciales $\mathcal{A}^u$ está definido como

$$\mathcal{A}^u = \mu^u(x,t)\frac{\partial}{\partial x} + \frac{1}{2}C^u(x,t)\frac{\partial^2}{\partial x^2}.$$

- Para cualquier control $\mathbf{u}$, el operador diferencial parcial $\mathcal{A}^{\mathbf{u}}$ es definido por

$$\mathcal{A}^{\mathbf{u}} = \mu^{\mathbf{u}}(x,t)\frac{\partial}{\partial x} + \frac{1}{2}C^{\mathbf{u}}(x,t)\frac{\partial^2}{\partial x^2}$$

La idea ahora, es establecer la función objetivo del problema de control, y por lo tanto consideramos como dadas un par de funciones

$$\begin{aligned} F &: \mathbb{R}\times\mathbb{R}_+\times\mathbb{R}\to\mathbb{R}, \\ \Phi &: \mathbb{R}\to\mathbb{R}, \end{aligned}$$

[4]El símbolo $\mathbb{E}^{x,t}(Y)$ denota la esperanza condicional de que el evento Y suceda dado que en el momento t, el espacio de estados se encuentra en la posición x.

donde F y Φ las interpretamos como las funciones que representan la utilidad instantánea y la final respectivamente. Estas funciones satisfacen las condiciones de crecimiento polinomial:

$$|F(x,t,u)| \leq C(1+|x|^k+|u|^k), \tag{2.9}$$

$$|\Phi(x,t)| \leq C(1+|x|^k), \tag{2.10}$$

para algún $k \in \mathbb{N}$ y C una constante positiva.
Luego, definimos la **función de ganancias** J_0 de nuestro problema, tal que

$$J_0 : \mathcal{U} \to \mathbb{R},$$

definida por

$$J_0(\mathbf{u}) \quad = \quad \mathbb{E}\left[\int_0^T F(X^{\mathbf{u}}(t),t,\mathbf{u}_t)dt + \Phi(X^{\mathbf{u}}(T))\right]$$

donde $X^{\mathbf{u}}(t)$ es la solución de (2.8) con la condición inicial $X(0)=x_0$.
Nuestro objetivo es establecer la maximización de J sobre todos los $\mathbf{u} \in \mathcal{U}$, es decir, queremos encontrar $\hat{J}_0$, tal que

$$\hat{J}_0 = \sup_{\mathbf{u}\in\mathcal{U}} J_0(\mathbf{u}), \tag{2.11}$$

De igual forma nos interesa encontrar el argumento óptimo u^* de J, si tal control tiene la propiedad de

$$J_0(\mathbf{u}^*) = \hat{J}_0.$$

decimos que $\mathbf{u}^*$ es un **control óptimo** para el problema dado. Nótese que posiblemente para cualquier problema de optimización puede que el control óptimo no exista.[5]

2.1.4 La ecuación de Hamilton-Jacobi-Bellman

En la búsqueda de algún funcional $\hat{J}_0$ como en (2.11), es natural hacerse algunas preguntas que son de fundamental importancia. ¿Existe un control óptimo?, ¿si un control óptimo existe, cómo lo encontramos?. La respuesta a la primera pregunta requiere de una clase de condiciones sobre las funciones μ y σ, C y F (incluso a mayor dimensión), que nosotros no los mostramos en este documento pero los asumiremos, sin embargo, el lector interesado puede referirse al texto de Pham [19] página 38 o también al texto de Korn [13] páginas 224-226. Para contestar la segunda pregunta, haremos uso del método de **programación dinámica**, que toma como referencia al principio de optimalidad de Bellman: "una política óptima tiene la propiedad de que cualquiera que sea el estado inicial y la decisión inicial, las decisiones restantes deben constituir una política óptima con respecto al estado resultante de la primera decisión". La idea consiste en establecer una ecuación diferencial conocida como la ecuación de Hamilton-Jacobi-Bellman (que para abreviar nos referimos a ella como HJB en adelante) que describe el comportamiento local de la función de valor la cual definimos a continuación.
Procedemos como sigue: fijemos un punto $t \in [0,T]$ (aquí T es el tiempo final o madurez); tambien fijamos un punto x en el espacio de estados, es decir, en $\mathbb{R}$. Por tanto, para nuestro par fijo (x,t) definimos el siguiente problema de control

Definición 2.1.8. *El problema de control $\mathcal{P}(x,t)$ es definido como el problema de maximizar*

$$\mathbb{E}^{x,t}\left[\int_t^T F(X^{\boldsymbol{u}}(s),s,\boldsymbol{u}_s)ds + \Phi(X^{\boldsymbol{u}}(T))\right], \tag{2.12}$$

dada la dinámica

$$dX^u(s) \quad = \quad \mu(X^{\boldsymbol{u}}(s),s,\boldsymbol{u}_s)ds + \sigma(X^{\boldsymbol{u}}(s),s,\boldsymbol{u}_s)dW(s) \tag{2.13}$$

$$X(t) \quad = \quad x \tag{2.14}$$

y las restricciones

$$u(y,s) \in U, \qquad \textit{para todos } (y,s) \in \mathbb{R}\times[t,T]. \tag{2.15}$$

[5] Hay ejemplos con horizonte tiempo infinito que se podrían consultar en [19].

En términos de la sección anterior, nuestro problema es $\mathcal{P}(x_0, 0)$.
Ahora definimos la **función de valor** y la **función de valor óptimo**.

Definición 2.1.9. *La **función de valor***

$$\mathcal{J} : \mathbb{R}^n \times \mathbb{R}_+ \times \mathcal{U} \to \mathbb{R}$$

se define por la expresión

$$\mathcal{J}(x, t, \boldsymbol{u}) = \mathbb{E}\left[\int_t^T F(X_s^u, s, \boldsymbol{u}_s)ds + \Phi(X^{\boldsymbol{u}}(T))\right]$$

Definición 2.1.10. *La **función de valor óptimo***

$$V : \mathbb{R} \times \mathbb{R}_+ \to \mathbb{R}$$

es definida por

$$V(x, t) = \sup_{u \in \mathcal{U}} J(x, t, \boldsymbol{u}) \tag{2.16}$$

Nótese que $J(x, t, u)$ es la utilidad esperada usando el control óptimo u sobre el intervalo de tiempo $[t, T]$, dado el hecho de que empieza en x en el tiempo t. La función de valor óptimo nos da la utilidad óptima esperada sobre $[t, T]$ bajo las mismas condiciones iniciales.
Nos disponemos ahora a hacer una derivación de la ecuación diferencial parcial para V. Es necesario primero establecer las siguientes suposiciones[6].

Suposición 2.1.1. *Asumimos las siguientes afirmaciones:*

1. *Existe un control óptimo $\boldsymbol{u}^*$.*
2. *La función de valor óptimo V es regular en el sentido que $V \in C^{2,1}$.*
3. *Algunos procedimientos que resultan limitantes pueden ser justificados.*

Con dichas suposiciones deducirémos la ecuación HJB, para esto fijamos un $(x, t) \in \mathbb{R} \times (0, T)$. Además escogemos un número real h (interpretado como un incremento "pequeño") tal que $t + h < T$. Elegimos un control arbitraro $\mathbf{u}$ y definimos otro control $\hat{\mathbf{u}}$ por

$$\hat{\mathbf{u}}(y, s) = \begin{cases} \mathbf{u}(y, s), & \text{si} \quad (y, s) \in \mathbb{R} \times [t, t + h] \\ \mathbf{u}^*(y, s), & \text{si} \quad (y, s) \in \mathbb{R} \times (t + h, T]. \end{cases}$$

En otras palabras, si usamos $\hat{u}$, entonces usamos un control arbitrario durante el intervalo de tiempo $[t, t+h]$, y luego cambiamos al control óptimo durante el resto de periodo.
La idea de programación dinámica se podría resumir en el siguiente procedimiento.

- Dado (x, t) fijo arriba, se consideran las siguientes estrategias sobre el intervalo $[t, T]$:
  Estrategia 1. Usamos el control óptimo $\mathbf{u}^*$.
  Estrategia 2. Usamos el control $\hat{\mathbf{u}}$ de arriba.
- Luego calculamos las utilidades esperadas a cada estrategia.
- Finalmente, usando el hecho de que la estrategia 1., debe ser al menos tan buena como la estrategia 2., y haciendo tender a h a cero, obtenemos nuestra ecuación diferencial parcial fundamental.

Iniciamos obteniendo el valor esperado de la función de ganancias con el control óptimo, es decir aplicamos el valor esperado usando **la estrategia 1**.

$$J(x, t, \mathbf{u}^*) = V(x, t).$$

[6]Esta suposición es *ad hoc* en el sentido que se destina para poder plantear el teorema 2.1.2, el cual es presentado en forma de condición necesaria.

Mientras que el valor esperado aplicando la **estrategia 2.** es

$$\begin{aligned} J(x,t,\hat{\mathbf{u}}) &= \mathbb{E}^{x,t}\left[\int_t^{t+h} F(X^{\mathbf{u}}(s),s,\mathbf{u}_s)ds + \int_{t+h}^{T} F(X^{\mathbf{u}^*}(s),s,\mathbf{u}_s^*)ds + \Phi(X^{\hat{\mathbf{u}}}(T))\right] \\ &= \mathbb{E}^{x,t}\left[\int_t^{t+h} F(X^{\mathbf{u}}(s),s,\mathbf{u}_s^*)ds + \mathbb{E}^{X(t+h),t+h}\left[\int_{t+h}^{T} F(X^{\mathbf{u}^*}(s),s,\mathbf{u}^*)ds + \Phi(X^{\mathbf{u}^*}(T))\right]\right] \\ &= \mathbb{E}^{x,t}\left[\int_t^{t+h} F(X^{\mathbf{u}}(s),s,\mathbf{u}_s)ds + V(X^{\mathbf{u}^*}(t+h),t+h).\right] \end{aligned}$$

Comparando estrategias concluimos que

$$V(x,t) \geq \mathbb{E}^{x,t}\left[\int_t^{t+h} F(X^{\mathbf{u}}(s),s,\mathbf{u}_s)ds + V(X^{\mathbf{u}}(t+h),t+h).\right] \tag{2.17}$$

El símbolo de desigualdad es debido a que al tomar un control arbitrario $\mathbf{u}$ y usarlo en el intervalo $[t,t+h]$ produce una utilidad menor a la que se obtiene usando el control óptimo u^*.

Observación 2.1.1. *La desigualdad en (2.17) cambia a igualdad si y sólo si $\mathbf{u}$ es el control óptimo.*

Puesto que V es $C^{2,1}$, podemos aplicar la fórmula de Itô a $V(X^{\mathbf{u}}(t+h),t+h)$, teniendo la siguiente expresión

$$\begin{aligned} V(X^{\mathbf{u}}(t+h),t+h) = V(x,t) &+ \int_t^{t+h}\left[\frac{\partial V}{\partial t}(X^{\mathbf{u}}(s),s) + \mathcal{A}^u V(X^u(s),s)\right]ds \\ &+ \int_t^{t+h}\frac{\partial V}{\partial x}(X^u(s),s)\sigma^u dW(s) \end{aligned} \tag{2.18}$$

Bajo nuestras hipótesis aplicamos el operador de valor esperado $\mathbb{E}^{x,t}$ a esta ecuación, nótamos que la integral estocástica se anula. Al sustituir este resultado en la desigualdad (2.17), vemos que los términos $V(x,t)$ se anulan, llegando a la siguiente desigualdad

$$\mathbb{E}^{x,t}\left[\int_t^{t+h}\left(F(X^{\mathbf{u}}(s),s,\mathbf{u}_s) + \frac{\partial V}{\partial t}(X^{\mathbf{u}}(s),s) + \mathcal{A}^{\mathbf{u}}V(X^{\mathbf{u}}(s),s)\right)ds\right] \leq 0. \tag{2.19}$$

Continuamos con la división entre h y luego le hacemos tender a 0:

$$\begin{aligned} \lim_{h\to 0}\mathbb{E}^{x,t}\left[\frac{1}{h}\int_t^{t+h}\left(F(X^{\mathbf{u}}(s),s,\mathbf{u}_s) + \frac{\partial V}{\partial t}(X^{\mathbf{u}}(s),s) + \mathcal{A}^{\mathbf{u}}V(X^{\mathbf{u}}(s),s)\right)ds\right] \\ = F(x,t,u) + \frac{\partial V}{\partial t}(x,t) + \mathcal{A}^u V(x,t) \leq 0, \end{aligned}$$

donde, recordemos que $u = \mathbf{u}(x,t)$.
Como el control u era arbitrario, esta desigualdad se cumple para todas las elecciones de $u \in \mathcal{U}$, y tendremos la igualdad si y solo si $u(x,t) = u^*(x,t)$. En este caso tenemos la siguiente ecuación

$$\frac{\partial V}{\partial t}(x,t) + \sup_{u\in\mathcal{U}}\{F(x,t,u) + \mathcal{A}^u V(x,t)\} = 0. \tag{2.20}$$

Puesto que el punto (x,t) se escogió arbitrariamente en $\mathbb{R}\times(0,T)$, en este mismo dominio la ecuación (2.20) es verídica. Hemos obtenido así, una ecuación diferencial parcial la cual está sujeta a la condición de frontera $V(x,T) = \Phi(x)$ para todo $x \in \mathbb{R}$ que es un resultado casi obvio[7]. Se ha establecido de esta forma el siguiente resultado.

[7]Recordemos que por definición

$$V(x,T) = \sup_{\mathbf{u}\in\mathcal{U}}\mathbb{E}^{x,T}\left[\int_T^T F(X_s^{\mathbf{u}},s,\mathbf{u}_s)ds + \Phi(X_T^{\mathbf{u}})\right] = \Phi(X_T^{\mathbf{u}}).$$

Teorema 2.1.2. *Asumiendo 2.1.1, las siguientes afirmaciones son verdaderas:*

1. *V satisface la ecuación HJB:*

$$\begin{aligned} \frac{\partial V}{\partial t}(x,t) + \sup_{u\in U}\{F(x,t,u) + \mathcal{A}^u V(x,t)\} &= 0, \qquad \text{para todo } (x,t) \in \mathbb{R}\times(0,T), \\ V(x,T) &= \Phi(x), \qquad \text{para todo } x \in \mathbb{R}. \end{aligned}$$

2. *Para cada $(x,t) \in \mathbb{R}\times[0,T]$ el valor máximo de la ecuación HJB arriba se alcanza cuando $u = \boldsymbol{u}^*(x,t)$.*

Varias observaciones se pueden hacer hasta el momento:

- El conjunto U de restricciones puede depender del tiempo y del estado. Dada esa especificación, la ecuación HJB aun es válida con la modificación obvia en el supremo.
- El teorema de la ecuación de HJB 2.1.2, proporciona condiciones necesarias para que dicha ecuación se válida. Nos dice que si V es la función de valor óptimo y $\mathbf{u}^*$ es el control óptimo, entonces V satisface la ecuación HJB.
- En lugar de las suposiciones hechas en (2.1.1), podríamos haber impuesto condiciones sobre las funciones μ, σ, F y C, lo cual conlleva a un trabajo más técnico de análisis, es algo en lo que no vamos a profundizar[8].

Un interesante resultado que nos brinda condiciones suficientes en el teorema de la ecuación de HJB, es el **teorema de verificación**. Nos dice

Teorema 2.1.3. *Suponga que tenemos dos funciones $H(x,t)$ y $g(x,t)$, tales que*

1. *H es suficientemente integrable[9] y resuelve la ecuación HJB:*

$$\begin{aligned} \frac{\partial H}{\partial t}(x,t) + \sup_{u\in U}\{F(x,t,u) + \mathcal{A}^u H(x,t)\} &= 0, \qquad \text{para todo } (x,t) \in (0,T)\times\mathbb{R}, \\ H(x,T) &= \Phi(x), \qquad \text{para todo } x \in \mathbb{R}. \end{aligned}$$

2. *La función g es un control admisible.*
3. *Para cada (x,t) fijo, el supremo en la expresión*

$$\sup_{u\in U}\{F(x,t,u) + \mathcal{A}^u H(x,t)\}$$

es alcanzado cuando se elige $u = g(x,t)$.

Entonces lo que sigue es válido:

- *La función de valor V para el problema de control es dado por*

$$V(x,t) = H(x,t)$$

- *Existe un control óptimo $\boldsymbol{u}^*$, tal que $\boldsymbol{u}^*(x,t) = g(x,t)$.*

[8] En [13] por ejemplo, se asume que $\mu(\cdot,\cdot,u)$, $\sigma(\cdot,\cdot,u) \in C^1([t_0,t_1]\times\mathbb{R})$ para $u \in U$. Además, para una constante $C > 0$

$$\left|\frac{\partial \mu}{\partial t}\right| + \left|\frac{\partial \mu}{\partial x}\right| \le C, \qquad \left|\frac{\partial \sigma}{\partial t}\right| + \left|\frac{\partial \sigma}{\partial x}\right| \le C,$$

$$|\mu(t,x,u)| + |\sigma(t,x,u)| \le C(1+|x|+|y|).$$

Para que $X(t) < \infty$ para todo $t > 0$. Las condiciones para F y Φ están dadas en (2.9) y (2.10) respectivamente.

[9] Significa que H es suficientemente regular como para que su valor esperado sea cero. Por ejemplo, H puede satisfacer la condición $\frac{\partial H}{\partial x}(X^u(s),s)\sigma^u(X^u(s),s) \in L^2$, para todos los controles admisibles u.

Demostración. Elegimos una ley de control $\mathbf{u} \in \mathcal{U}$, y fijemos un punto (x,t). Definimos el proceso $X^{\mathbf{u}}$ sobre el intervalo de tiempo $[t,T]$ que soluciona la ecuación

$$\begin{aligned} dX^{\mathbf{u}}(s) &= \mu^{\mathbf{u}}(X^{\mathbf{u}}(s),s)ds + \sigma^{\mathbf{u}}(X^{\mathbf{u}}(s),s)dW(s) \\ X(t) &= x. \end{aligned}$$

Sustituyendo el proceso $X^{\mathbf{u}}$ en la función H y usando la fórmula de Itô obtenemos

$$\begin{aligned} H(X^{\mathbf{u}}(T),T) &= H(x,t) + \int_t^T \left[\frac{\partial H}{\partial t}(X^{\mathbf{u}}(s),s) + \mathcal{A}^{\mathbf{u}}H(X^{\mathbf{u}}(s),s)\right]ds \\ &\quad + \int_t^T \frac{\partial H}{\partial x}(X^{\mathbf{u}}(s),s)dW(s). \end{aligned}$$

Puesto que H resuelve la ecuación HJB, vemos que

$$\frac{\partial H}{\partial t}(x,t) + F(x,t,u) + \mathcal{A}^u H(x,t) \leq 0$$

para todo $u \in U$. Así, tenemos para cada s que

$$\frac{\partial H}{\partial t}(X^{\mathbf{u}}(s),s) + \mathcal{A}^{\mathbf{u}}H(X^u(s),s) \leq -F^{\mathbf{u}}(X^{\mathbf{u}}(s),s),$$

con probabildad 1.
De la condición de frontera para la ecuación HJB tenemos $H(X^u(T),T) = \Phi(X^u(T)$, tal que llegamos a la desigualdad

$$H(x,t) \geq \int_t^T F^{\mathbf{u}}(X^{\mathbf{u}}(s),s)ds + \Phi(X^{\mathbf{u}}(T)) - \int_t^T \frac{\partial H}{\partial x}(X^{\mathbf{u}}(s),s)dW(s)$$

Aplicando el operador de valor esperado , y asumiendo suficiente integrabilidad, hacemos que la integral estocástica se anule, dejandonos la siguiente desigualdad

$$H(x,t) \geq \mathbb{E}^{x,t}\left[\int_t^T F^{\mathbf{u}}(X^{\mathbf{u}}(s),s)ds + \Phi(X^{\mathbf{u}}(T))\right] = J(x,t,\mathbf{u}).$$

Puesto que el control u fue elegido de manera arbitraria, esto nos da

$$H(x,t) \geq \sup_{\mathbf{u}\in\mathcal{U}} J(x,t,\mathbf{u}) = V(x,t)$$

Para obtener la desigualdad recíproca escogemos una ley de control específico $u(x,t) = g(x,t)$. Haciendo las mismas operaciones arriba, y usando el hecho de que por hipótesis tenemos que

$$\frac{\partial H}{\partial t}(x,t) + F^{\mathbf{g}}(x,t) + \mathcal{A}^{\mathbf{g}}H(x,t) = 0,$$

obtenemos la igualdad

$$H(x,t) = \mathbb{E}^{x,t}\left[\int_t^T F^{\mathbf{g}}(X^{\mathbf{g}}(s),s)ds + \Phi(X^{\mathbf{g}}(T))\right] = J(x,t,\mathbf{g}).$$

Por otro lado tenemos la desigualdad trivial

$$V(x,t) \geq J(x,t,\mathbf{g})$$

y usando las anteriores desigualdades, obtenemos

$$H(x,t) \geq V(x,t) \geq J(x,t,\mathbf{g}) = H(x,t).$$

Esto muestra el hecho

$$H(x,t) = V(x,t) = J(x,t,\mathbf{g}),$$

lo que prueba que $H = V$, y que $\mathbf{g}$ es el control óptimo.

□

2.1.5 Procedimiento de resolución de la ecuación HJB

Consideremos nuestro problema de control óptimo estándar con la ecuación HJB correspondiente:

$$\begin{aligned} \frac{\partial V}{\partial t}(x,t) + \sup_{u\in U}\{F(x,t,u) + \mathcal{A}^u V(x,t)\} &= 0, \\ V(x,T) &= \Phi(x). \end{aligned} \tag{2.21}$$

De manera esquemática, procedemos con realizar los siguientes pasos:

1. Considere la ecuación HJB como una ecuación diferencial parcial para una función desconocida V.
2. Fije un punto arbitrario $(x,t) \in \mathbb{R} \times [0,T]$ y resuelva, para la elección fijada de (x,t), el problema estático de optimización
$$\max_{u\in U}[F(x,t,u) + \mathcal{A}^u V(x,t)].$$
Note que en este problema u es la única variable, mientras que t y x se consideran parámetros fijados. Las funciones F, μ, σ y V se consideran como dadas.
3. La elección óptima de u, denotada por u^*, por supuesto dependerán de nuestra elección de t y x, pero también de la función V y sus derivadas parciales que se encuentran en la definición de $\mathcal{A}^u V$. Para realtar estas dependencias escribimos u^* como $u^* = u^*(x,t;V)$.
4. La función $u^*(x,t;V)$ es nuestro candidato para control óptimo, pero puesto que no sabemos V, esta descripción es incompleta. Por tanto sustituimos la expresión de u^* en la ecuación diferencial parcial (2.21), lo cual nos da
$$\frac{\partial V}{\partial t}(x,t) + F(x,t,\mathbf{u}^*) + \mathcal{A}^{\mathbf{u}^*} V(x,t) = 0, \tag{2.22}$$
$$V(x,T) = \Phi(x). \tag{2.23}$$
5. Posteriormente, resolvemos la ecuación diferencial parcial de arriba. Para llevar a cabo este trabajo, no siempre podemos obtener a V con metodos analíticos, a veces debemos proponer un candidato que dependa de un número finito de parámetros y luego usar las ecuación diferencial parcial para identificar dichos parámetros. La forma que tiene la función Φ y la utilidad instantánea F, usualmente ayudan en la proposición de nuestra solución, dicha proposición se conoce como *Ansatz*[10].

Cabe mencionar que algunos problemas de control óptimo estocástico son manipulados hasta cierto punto para poder ser solucionados.

2.2 Descripción del problema de selección óptima del portafolio de Merton

En esta sección mencionamos aspectos relevantes a tener en cuenta para poder exponer un ejemplo de aplicación a la teoría de control óptimo estocástico, el cual será nuestro punto de referencia central para obtener aproximaciones numéricas con vários métodos computacionales. El ejemplo del cual hablamos es el problema de seleccionar la proporción de inversión óptima para que un agente que desea invertir en un bono y un activo riesgoso, obtenga la máxima utilidad.
Vamos a asumir los siguientes aspectos particulares:

- El precio del activo riesgoso está modelado como un Movimiento Browniano Geométrico $S(t)$ cuya dinámica está dada por
$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t),$$
donde μ es la tasa de retorno constante y $\sigma > 0$ es la volatilidad propios de dicho activo y W es un movimiento Browniano sobre un espacio de probabilidad filtrado $\{\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\in[0,T]}, \mathbb{P}\}$ para $t \in [0,T]$, (un problema de horizonte de tiempo finito).

[10] Un *Ansatz* es una suposición sobre la forma de una función desconocida que se hace para proporcionar la solución de una ecuación.

- El bono lo describimos através de un proceso determinístico $B(t)$ con dinámica

$$dB(t) = rB(t)dt,$$

donde r[11] es la tasa de interés libre de riesgo del bono y $t \in [0, T]$.

- La riqueza del agente depende de la proporción $\pi(t)$ que invierte en el activo riesgoso y lo que invierte $1 - \pi(t)$ en el bono. Al proceso π lo llamaremos portafolio y es la caracterización que empleamos de u como control de la sección anterior.

- La riqueza del agente es un proceso controlado X con dinámica

$$dX(t) = \frac{(1 - \pi(t))X(t)}{B(t)}dB(t) + \frac{\pi(t)X(t)}{S(t)}dS(t),$$

es decir

$$dX(t) = (r + (\mu - r)\pi(t))X(t)dt + \pi(t)X(t)\sigma dW(t), \tag{2.24}$$
$$X(0) = X_0$$

Donde $X_0 > 0$ es la riqueza inicial.

- Las preferencias del agente se ajustan a una función de utilidad CRRA (Constant Relative Risk Averse), dada por la expresión

$$U(x) = \frac{x^\gamma}{\gamma}, \quad \gamma \in (0, 1).$$

Este tipo especial de funciones de utilidad tienen la propiedad de que su coeficiente de aversión al riesgo relativo Arrow-Pratt es una constante, es decir, suponemos que el agente tiene una aversión al riesgo constante frente al cambio de proporción de su riqueza.

Nos enfrentamos entonces a maximizar la utilidad del agente al final del periodo $[0, T]$.

2.3 Planteamiento del problema

El objeto de esta sección es establecer formalmente el problema de la maximización de la utilidad del agente como un problema de control óptimo estocástico. De manera concreta, se pretende maximizar la utilidad esperada en el tiempo $t = T$ para esto, definimos la función de valor óptimo (o simplemente función de valor) V como

$$V(x, t) = \sup_{\pi \in [0,1]} \mathbb{E}^{x,t}[U(X(T)], \quad (x, t) \in (0, \infty) \times [0, T]. \tag{2.25}$$

Asumimos la existencia de un control óptimo π^* y que la función V es regular en el sentido que es $C^{2,1}((0, \infty) \times [0, T])$. Aplicamos ahora la fórmula de Itô a V, obtenemos la ecuación de Hamilton-Jacobi-Bellman asociada al problema (2.25):

$$\begin{aligned} V_t + \sup_{\pi \in [0,1]} \left[(r + \pi(\mu - r))xV_x + \frac{1}{2}x^2\sigma^2\pi^2 V_{xx} \right] &= 0, \\ V(x, T) &= \frac{x^\gamma}{\gamma}. \end{aligned} \tag{2.26}$$

Donde las letras en los subíndices denotan la derivada parcial de V con respecto a dicha letra.
De manera esquemática resolveremos el problema (2.26) fijando un $(t, x) \in [0, T] \times (0, \infty)$. Luego, encontramos la solución el problema de optimización estática:

$$\sup_{\pi \in [0,1]} \left[(r + \pi(\mu - r))xV_x + \frac{1}{2}x^2\sigma^2\pi^2 V_{xx} \right].$$

[11] Es preciso notar que el agente asume el riesgo de inversión en el activo riesgoso siempre que $\mu > r$, de otra manera invertiría su riqueza en el bono o también podría hacer venta corta.

Derivamos la expresión

$$rxV_x + \pi(\mu - r)xV_x + \frac{1}{2}x^2\sigma^2\pi^2V_{xx}$$

con respecto a π e igualamos a cero, para obtener un candidato de la inversión óptima del portafolio

$$\pi^* = -\frac{\mu - r}{\sigma^2}\frac{V_x}{xV_{xx}}. \tag{2.27}$$

2.4 Solución analítica

Debido a la forma del planteamiento del problema de maximización (2.26), proponemos un Ansatz de la siguiente forma

$$V(x,t) = \phi(t)U(x).$$

De acuerdo a esto, sustituimos los valores de V y π^* en (2.26):

$$\phi'(t)\frac{x^\gamma}{\gamma} + rx\phi(t)x^{\gamma-1} + \frac{(\mu-r)^2}{\sigma^2(1-\gamma)}x\phi(t)x^{\gamma-1} + \frac{1}{2}x^2\sigma^2\frac{(\mu-r)^2}{\sigma^4(1-\gamma)^2}\phi(t)(\gamma-1)x^{\gamma-2} = 0$$

Simplificando, nos encontramos con una ecuación de variables separables

$$\begin{aligned}\phi'(t) + \phi(t)\left[r\gamma + \frac{1}{2}\frac{(\mu-r)^2}{\sigma^2}\frac{\gamma}{1-\gamma}\right] &= 0\\ \phi(T) &= 1.\end{aligned}$$

Al integrar en $[t,T]$, encontramos la solución

$$\phi(t) = \exp\left[(T-t)\left(r\gamma + \frac{1}{2}\frac{(\mu-r)^2}{\sigma^2}\frac{\gamma}{1-\gamma}\right)\right].$$

Por lo tanto, la función de valor V es dada por

$$V(x,t) = \exp\left[(T-t)\left(r\gamma + \frac{1}{2}\frac{(\mu-r)^2}{\sigma^2}\frac{\gamma}{1-\gamma}\right)\right]\frac{x^\gamma}{\gamma}. \tag{2.28}$$

Recordemos ahora que por la definición de la función de utilidad $U(x) = \frac{x^\gamma}{\gamma}$, encontramos que $-\frac{U'(x)}{xU''(x)} = \frac{1}{1-\gamma}$. De esta manera, al sustituir en (2.27) encontramos que el control óptimo viene dado por la expresión

$$\pi^* = \frac{\mu - r}{\sigma^2(1-\gamma)}. \tag{2.29}$$

Capítulo 3

Aproximación mediante cadenas de Markov

En esta sección establecemos dos métodos de diferencias finitas para la discretización de la ecuación HJB (2.26). Según el texto de Kushner [16], existe una cadena de Markov $\{\xi_n^{\Delta x,\Delta t}, n < \infty\}$ que aproxima el proceso de riqueza (2.24) y cuyas probabilidades de transición son obtenidas por el esquema de diferencias finitas. Posteriormente se describe un algoritmo (backward) que consiste en conseguir distintas aproximaciones empezando en el tiempo final hasta el inicial en un intervalo de tiempo discretizado. Finalmente se muestran gráficos de las superficies del error obtenidas con la aproximación del método.

3.1 Esquemas de diferencias finitas

En este apartado, definimos dos métodos de diferencias finitas: esquema *upwind* y esquema *simétrico*. La idea es definir una cadena de Markov a partir de los coeficientes de los esquemas que al ser definidos adecuadamente, representan las probabilidades de transición de la cadena que aproxima el proceso de riqueza (2.24). La aproximación se basa en el hecho que la cadena de Markov es localmente consistente con el proceso de estados, es decir, la cadena tiene momentos (por ejemplo el valor esperado) aproximados al proceso de estados en los puntos considerados de la discretización. Una prueba de este hecho se puede encontrar en [11] pág. 768 o [23], pág. 568.

Suponga que para todo control $\pi = \pi(x^\pi(t), t)$ existe una función de valor

$$Y(x,t,\pi) = \mathbb{E}_{x,t}^{\pi}\{U(x^\pi(T))\},$$

la cual satisface

$$\begin{aligned} Y_t + (r + (\mu - r)\pi)xY_x + \frac{1}{2}x^2\pi^2\sigma^2 Y_{xx} = &\quad 0, \\ Y(x,T,\pi) = &\quad \frac{x^\gamma}{\gamma}. \end{aligned} \tag{3.1}$$

Consideramos la discretización de (3.1) en el siguiente dominio

$$D = \left\{ \begin{array}{lll} x_j = 0 + j\Delta x; & \Delta x = \frac{1-0}{N}; & j = 0, ..., 2N; \\ t_k = 0 + k\Delta t; & \Delta t = \frac{1-0}{M}; & k = 0, ..., M. \end{array} \right\},$$

donde N y M son el número de divisiónes para el eje del espacio de estados y de tiempo respectivamente. Consideramos hacer la aproximación el intervalo $[0, 2]$ (teniendo que $x_N = 1$) en lugar de $[0, 1]$. Así, después de discretizar el espacio de estados, el valor de x estará en el centro. Finalmente, usando el algoritmo de iteración hacia atrás en el tiempo, el valor $V(t, x)$ quedaría aproximado.
Para el método de diferencias finitas usamos el esquema *upwind*. Este método fue construido originalmente

para observar aproximaciones de derivadas en puntos de discontinuidad, no obstante la solución dada por (2.28) es continua en todos los puntos en el dominio D; la usamos por la idea propuesta por Kushner y Dupois en [16] capítulo 4 y 5, quienes usan este esquema para demostrar convergencia de la cadena de Markov al proceso de riqueza (2.24).
El esquema upwind se resume en realizar las siguientes discretizaciones para las derivadas parciales presentes en (3.1) en el punto $(x_j, t_{k+1}) \in D$ usamos la notación $Y(x_j, t_k, \pi) = y_{j,k}$:

$$Y_t(x_j, t_{k+1}, \pi) = \frac{y_{j,k+1} - y_{j,k}}{\Delta t},$$

$$\begin{aligned} Y_x(x_j, t_{k+1}, \pi)(r + (\mu - r)\pi) = & \frac{y_{j+1,k+1} - y_{j,k+1}}{\Delta x} x_j (r + (\mu - r)\pi)^+ \\ & - \frac{y_{j,k+1} - y_{j-1,k+1}}{\Delta x} x_j (r + (\mu - r)\pi)^-, \end{aligned} \tag{3.2}$$

$$Y_{xx}(x_j, t_{k+1}, \pi) = \frac{y_{j+1,k+1} - 2y_{j,k+1} + y_{j-1,k+1}}{(\Delta x)^2},$$

donde $f^{\pm} = \max\{\pm f, 0\}$, además recordemos que $f^+ + f^- = |f|$ y $f^+ - f^- = f$. Nótese además, que el orden de los errores para Y_t, Y_x y Y_{xx} son $O(\Delta t)$, $O(\Delta x)$ y $O(\Delta x^2)$ respectivamente; para poder verificar esto, se puede consultar en el apéndice A de este trabajo.
Ahora definimos $\rho_{1,j} = x_j(r + (\mu - r)\pi)$ y $\rho_{2,j} = x_j^2 \sigma^2 \pi^2$ y junto con los esquemas de derivadas conseguidos, sustituimos en (3.1)

$$\begin{aligned} y_{j,k} = & \; y_{j-1,k+1} \left(\rho_{1,j}^- \frac{\Delta t}{\Delta x} + \frac{1}{2} \rho_{2,j} \frac{\Delta t}{(\Delta x)^2} \right) \\ & + y_{j,k+1} \left(1 - |\rho_{1,j}| \frac{\Delta t}{\Delta x} - \rho_{2,j} \frac{\Delta t}{(\Delta x)^2} \right) \\ & + y_{j+1,k+1} \left(\rho_{1,j}^+ \frac{\Delta t}{\Delta x} + \frac{1}{2} \rho_{2,j} \frac{\Delta t}{(\Delta x)^2} \right) \end{aligned} \tag{3.3}$$

Nótese que la suma de los coeficientes es igual a la unidad.
De manera análoga, consideramos otra aproximación para la derivada parcial $Y_x(x_j, t_{k+1}, \pi)$ con la fórmula de *diferencias centrales simétricas*:

$$Y_x(x_j, t_{k+1}, \pi) = \frac{y_{j+1,k+1} - y_{j-1,k+1}}{2\Delta x} \tag{3.4}$$

la cual mejora la aproximación del método. De manera similar al esquema anterior, encontramos una expresión correspondiente al esquema de diferencia central simétrica (al que llamamos esquema *simétrico*):

$$\begin{aligned} y_{j,k} = & \; y_{j-1,k+1} \left(\frac{1}{2} \rho_{2,j} \frac{\Delta t}{(\Delta x)^2} - \frac{1}{2} \rho_{1,j} \frac{\Delta t}{\Delta x} \right) \\ & + y_{j,k+1} \left(1 - \rho_{2,j} \frac{\Delta t}{(\Delta x)^2} \right) \\ & + y_{j+1,k+1} \left(\frac{1}{2} \rho_{1,j} \frac{\Delta t}{\Delta x} + \frac{1}{2} \rho_{2,j} \frac{\Delta t}{(\Delta x)^2} \right). \end{aligned} \tag{3.5}$$

Se puede observar que los órdenes del error para este método son los mismos al del esquema upwind excepto que el término Y_x es de orden $O(\Delta x^2)$, por tal motivo la aproximación por el esquema simétrico resulta mejor que upwind.

3.2 Probabilidades de transición y algoritmo de aproximación

De acuerdo con Kushner y Dupois [16], existe una cadena de Markov $\{\xi^{\Delta x, \Delta t}, n < \infty\}$ que se puede aproximar de manera localmente consistente a (2.24), para hacer esto posible, se necesita hacer una interpolación

adecuada de tiempo continuo. Las interpolaciones que usamos son funciones constantes a trozos definidas como

$$\xi^{\Delta x,\Delta t}(t) = \xi_n^{\Delta x,\Delta t}, \quad \pi^{\Delta x,\Delta t}(t) = \pi_n^{\Delta x,\Delta t} \quad \text{para } t \in [n\Delta t, n\Delta t + \Delta t)$$

Asumimos que $\mu > r$ es decir, el rendimiento del bono no excede al del activo riesgoso, lo que hace que $\rho_{1,j} = x_j(r + (\mu - r)\pi) \geq 0$ y lo cual nos permite afirmar que[1] $|\rho_{1,j}| = \rho_{1,j}$ y $\rho_{1,j}^- = \max\{-\rho_{1,j}, 0\} = 0$. Por el esquema upwind, definimos las probabilidades de transición de la cadena de Markov como

$$\begin{aligned} P(x_j, x_{j-1}|\pi_j) &= \frac{1}{2}\rho_{2,j}\frac{\Delta t}{(\Delta x)^2}, & j = 1, ..., 2N, \\ P(x_j, x_j|\pi_j) &= 1 - \rho_{1,j}\frac{\Delta t}{\Delta x} - \rho_{2,j}\frac{\Delta t}{(\Delta x)^2}, & j = 0, ..., 2N, \\ P(x_j, x_{j+1}|\pi_j) &= \rho_{1,j}\frac{\Delta t}{\Delta x} + \frac{1}{2}\rho_{2,j}\frac{\Delta t}{(\Delta x)^2}, & j = 0, ..., 2N-1. \end{aligned} \tag{3.6}$$

Que corresponden a los coeficientes del esquema (3.2) simplificados. Puesto que $P(x_j, x_{j-1}|\pi_j)$ y $P(x_j, x_{j+1}|\pi_j)$ son positivas, debemos escoger cuidadosamente a Δt y Δx de tal manera que $P(x_j, x_j|\pi_j) > 0$ para $j = 0, ..., 2N$.
Por otro lado, las probabilidades de transición para el esquema simétrico (3.5) son dadas por:

$$\begin{aligned} P(x_j, x_{j-1}|\pi_j) &= \frac{1}{2}\rho_{2,j}\frac{\Delta t}{(\Delta x)^2} - \frac{1}{2}\rho_{1,j}\frac{\Delta t}{\Delta x}, & j = 1, ..., 2N, \\ P(x_j, x_j|\pi_j) &= 1 - \rho_{2,j}\frac{\Delta t}{(\Delta x)^2}, & j = 0, ..., 2N, \\ P(x_j, x_{j+1}|\pi_j) &= \frac{1}{2}\rho_{1,j}\frac{\Delta t}{\Delta x} + \frac{1}{2}\rho_{2,j}\frac{\Delta t}{(\Delta x)^2}, & j = 0, ..., 2N-1. \end{aligned} \tag{3.7}$$

Algoritmo

En esta sección, presentamos el algoritmo mostrado en [11], el cual nos resultará útil para la obtención de una aproximación a la función de valor óptimo V y también al control óptimo π^*. La idea es implementar un esquema iterativo hacia atrás en la grilla del tiempo: para $t = t_M = T$, calculamos $U(x_j)$ para cada $j = 0, 1, ..., 2N$, luego se calcula π_k^* con la fórmula (2.27) con V calculado en t_{k+1} y se evalúa en la matriz P, que posteriormente será multiplicada por el vector de aproximación de los valores óptimos y así, por cada t_k, con $k = M-1, M-2, .., 0$ se llegará a un valor aproximado tanto de V como de π^*.
Se escogen los parámetros M y N que determinarán los tamaños de los pasos en el tiempo Δt y en el espacio Δx. Sea ahora $V^{\Delta x,\Delta t}$ la aproximación por cada iteración en el tiempo de V. Comenzamos en el tiempo final $t_M = T$ en el cual evaluamos la función de utilidad $V_{t_M}^{\Delta x,\Delta t} = (U(x_0, ..., U(x_{2N}))^\top$.
Para $k = M-1, ..., 1, 0$, se evalúa el vector $V_{t_k}^{\Delta t,\Delta x}$ con la fórmula

$$\begin{bmatrix} V^{\Delta x,\Delta t}(x_0, k\Delta t) \\ V^{\Delta x,\Delta t}(x_1, k\Delta t) \\ \vdots \\ V^{\Delta x,\Delta t}(x_{2N}, k\Delta t) \end{bmatrix} = \max_{\vec{\pi}} \left\{ P(\vec{\pi}) \begin{bmatrix} V^{\Delta x,\Delta t}(x_0, k\Delta t + \Delta t) \\ V^{\Delta x,\Delta t}(x_1, k\Delta t + \Delta t) \\ \vdots \\ V^{\Delta x,\Delta t}(x_{2N}, k\Delta t + \Delta t) \end{bmatrix} \right\}$$

donde la matriz $P(\pi_k^{\Delta x,\Delta t})$ es la matriz de representación de las probabilidades de transición de la cadena de Markov y $\vec{\pi}^\top = (\pi_0, \pi_1, ..., \pi_{2N})$. En dicha matriz se pueden introducir cualquiera de los dos esquemas de probabilidades anteriormente descritos correspondiente a (3.6) o (3.7):

$$P(\vec{\pi}) = \begin{bmatrix} P_{0,0}(\pi_0) & P_{0,1}(\pi_0) & 0 & \cdots & 0 & 0 \\ P_{1,0}(\pi_1) & P_{1,1}(\pi_1) & P_{1,2}(\pi_1) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & P_{2N-1,2N-1}(\pi_{2N-1}) & P_{2N-1,2N}(\pi_{2N-1}) \\ 0 & 0 & 0 & \cdots & P_{2N,2N-1}(\pi_{2N}) & P_{2N,2N}(\pi_{2N}) \end{bmatrix}$$

[1] Para el problema que estamos considerando aquí, no se necesita el esquema upwind, sin embargo, a este esquema lo seguiremos llamando así para evitar más definiciones.

donde $P_{i,j}(\pi_i) = P(x_i, x_j|\pi_i)$ para $0 \leq i, j \leq 2N$. Esta matriz se podría reconocer como la matriz de transición asociada a la cadena de Markov $\{\xi_n^{\Delta x,\Delta t}\}$ que se aproxima al proceso de riqueza (2.24). Es fácil ver que el estado x_0 es un estado absorbente, sin embargo las probabilidades para el último estado x_{2N} no parecen cumplir con el requisito que dice que la suma de las probabilidades en esta fila de la matriz es igual a uno; esto se debe a que el esquema upwind en el último valor también debe considerar a V_{xx}. De hecho, por el método expuesto $\pi(x_{2N}, t_k) = 0$ para todo $k \in \{0, 1, ..., M\}$, esto hace ver que

$$P_{2N,2N-1}(\pi_{2N}) + P_{2N,2N}(\pi_{2N}) = 1 - x_{2N} r \frac{\Delta t}{\Delta x}$$

Dada esta razón, consideramos la misma matriz de arriba, pero con una posición adicional "ficticia" $2N+1$ en el espacio de estados:

$$\begin{bmatrix} P_{0,0}(\pi_0) & P_{0,1}(\pi_0) & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & P_{2N,2N-1}(\pi_{2N}) & P_{2N,2N}(\pi_{2N}) & P_{2N,2N+1}(\pi_{2N}) \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}$$

como la matriz de transición de probabilidad de $\{\xi_n^{\Delta x,\Delta t}\}$.

3.3 Resultados numéricos

A continuación presentamos los errores obtenidos para diferentes valores de Δt y Δx con el fin de comparar el valor del control óptimo (2.29) con el aproximado proporcionado por el algoritmo, también mostramos algunos resultados concernientes a los valores aproximados que alcanza cada método.
Por simplicidad, se tomarán los parámetros del artículo de Kafash [11]: $\mu = 0.11$, $r = 0.05$, $\sigma = 0.4$, $\gamma = 0.5$ con $0 \leq t \leq 1$ y $0 \leq x \leq 1$. Con el método upwind obtenemos las siguientes errores para la aproximación de π^* analítico, que corresponde a un valor de 0.75.

Δx	π^* Aprox.	Error	Error %.
0.2	0.680937	6.91e-02	9.21
0.1	0.708825	4.12e-02	5.49
0.02	0.728873	2.11e-02	2.82
0.0125	0.730698	1.93e-02	2.57
0.01	0.731315	1.87e-02	2.49

Tabla 3.1: Método upwind con $\Delta t = 10^{-4}$.

Δx	π^* Aprox.	Error	Error %.
0.2	0.680928	6.91e-02	9.21
0.1	0.708815	4.12e-02	5.49
0.02	0.728861	2.11e-02	2.82
0.0125	0.730686	1.93e-02	2.58
0.01	0.731303	1.87e-02	2.49

Tabla 3.2: Método upwind con $\Delta t = 10^{-5}$.

Por otro lado, las aproximaciones de la función de valor en $V(1, 0) \approx 2.073830085701483$: [2]

Δx	V Aprox.	Error	Error %.
0.2	2.068942	4.89e-03	0.24
0.1	2.071239	2.59e-03	0.12
0.02	2.072850	9.80e-04	0.05
0.0125	2.072808	1.02e-03	0.05
0.01	2.072720	1.12e-03	0.05

Tabla 3.3: Valores de V usando upwind y $\Delta t = 10^{-4}$.

Δx	V Aprox.	Error	Error %.
0.2	2.068973	4.86e-03	0.23
0.1	2.071304	2.53e-03	0.12
0.02	2.073183	6.47e-04	0.03
0.0125	2.073343	4.88e-04	0.02
0.01	2.073388	4.42e-04	0.02

Tabla 3.4: Valores de V usando upwind y $\Delta t = 10^{-5}$.

El tiempo de ejecución para el método upwind con la mejor aproximación en nuestras consideraciones: $\Delta t = 10^{-5}$ ($M = 100.000$) y $\Delta x = 0.01$ ($N = 100$); es de 233 segundos (3 minutos y 53 segundos aproximadamente) programado en el lenguaje `Python`; y con aproximaciones a $V(1, 0)$ y a π^* con errores porcentuales

[2] Por la forma de V en (2.28), se observa que es estritamente creciente en el eje x y estrictamente decreciente en el eje t, razón por la cual alcance su máximo en (1,0).

respectivos 0.021% y 2.5%.

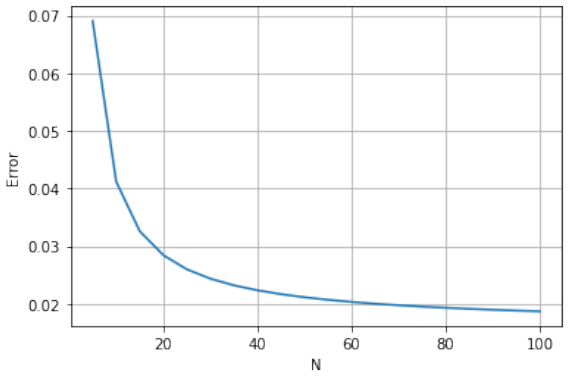


Figura 3.1: Errores en las aproximaciones a π^* con el método upwind $\Delta t = 10^{-5}$.

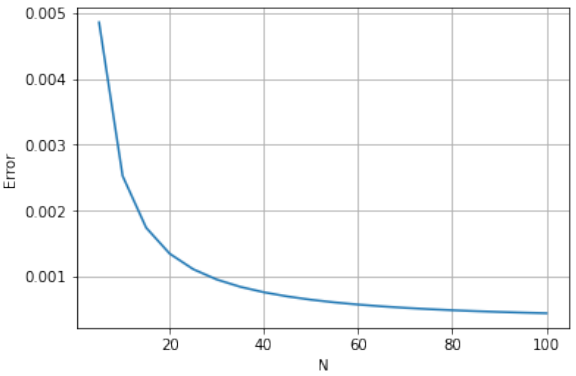


Figura 3.2: Errores en las aproximaciones a $V(1,0)$ con el método uwind $\Delta t = 10^{-5}$.

De manera análoga obtenemos los resultados correspondientes a la aproximación de π^* con el método simétrico:

Δx	π^* Aprox.	Error	Error %.
0.2	0.686791	6.32e-02	8.43
0.1	0.712334	3.77e-02	5.02
0.02	0.729969	2.00e-02	2.67
0.0125	0.731486	1.85e-02	2.47
0.01	0.731990	1.80e-02	2.40

Tabla 3.5: Método simétrico con $\Delta t = 10^{-4}$.

Δx	π^* Aprox.	Error	Error %.
0.2	0.686781	6.32e-02	8.42
0.1	0.712325	3.77e-02	5.02
0.02	0.729959	2.00e-02	2.67
0.0125	0.731475	1.85e-02	2.47
0.01	0.731979	1.80e-02	2.40

Tabla 3.6: Método simétrico con $\Delta t = 10^{-5}$.

Las aproximaciones al valor $V(1,0)$ con el método simétrico se muestran en las siguientes tablas:

Δx	V Aprox.	Error	Error %.
0.2	2.073728	1.01e-04	0.005
0.1	2.073643	1.87e-04	0.009
0.02	2.073334	4.96e-04	0.024
0.0125	2.073111	7.19e-04	0.035
0.01	2.072961	8.69e-04	0.042

Tabla 3.7: $V(1,0)$ aproximado con $\Delta t = 10^{-4}$.

Δx	V Aprox.	Error	Error %.
0.2	2.073762	6.81e-05	0.003
0.1	2.073710	1.20e-04	0.006
0.02	2.073670	1.60e-04	0.007
0.0125	2.073648	1.83e-04	0.008
0.01	2.073632	1.97e-04	0.01

Tabla 3.8: $V(1,0)$ aproximado con $\Delta t = 10^{-5}$.

El tiempo de ejecución para el método simétrico con la mejor aproximación en nuestras consideraciones: $\Delta t = 10^{-5}$ ($M = 100.000$) y $\Delta x = 0.01$ ($N = 100$); es de 453.387 segundos (7 minutos y 33 segundos aproximadamente).
Las gráficas de los errores en la aproximación a π^* y $V(1,0)$ con el método simétrico son las siguientes

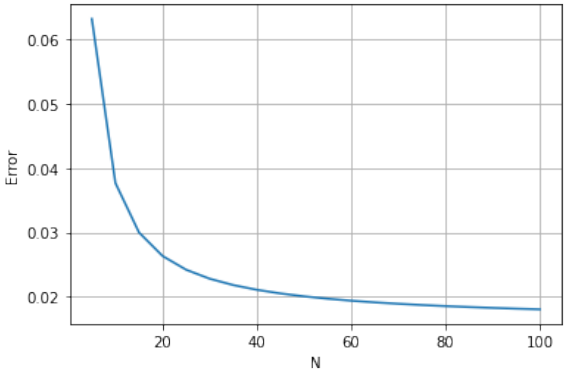


Figura 3.3: Errores en las aproximaciones a π^* con el método simétrico $\Delta t = 10^{-5}$.

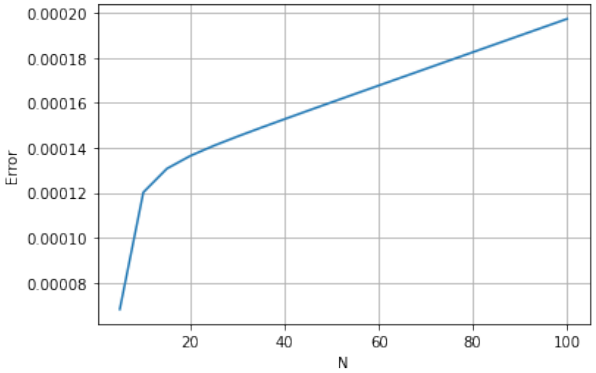


Figura 3.4: Errores en las aproximaciones a $V(1,0)$ con el método simétrico $\Delta t = 10^{-5}$.

Como se puede evidenciar en las gráficas, la mejora del método simétrico sobre el upwind no es de mayor importancia en cuanto al control óptimo π^*, los cambios más significativos están presentes en los valores obtenidos que aproximan la función de valor óptimo V. En las figura 3.1 y 3.2, podemos ver los errores que resultan al aplicar el método upwind, y se puede observar que tanto las aproximaciones a π^* como a $V(1,0)$ tienden a disminuir cuando N aumenta de valor. Lo mismo sucede para el control óptimo en el método simétrico, el cual es un poco mejor que el obtenido por el método upwind, no obstante, al ver la figura 3.4, pareciera que el esquema presenta inestabilidad en $(1,0)$, aunque el intervalo es pequeño. Sin embargo, en la figura 3.6 podemos ver que el error tiende a estabilizarse cuando escogemos $\Delta t = 10^{-6}$. Podemos decir que a medida que disminuimos el tamaño de Δt, el error se estabiliza en un número pequeño a medida que disminuimos Δx. El mejor resultado para el método simétrico con $N = 5$ y $M = 10^6$ toma 272.4 segundos (4 minutos 33 segundos) con un error porcentual de $0.00025\% = 2.5 \times 10^{-4}\%$ para la aproximación a $V(1,0)$ y de 3.4% para la aproximación a π^*.

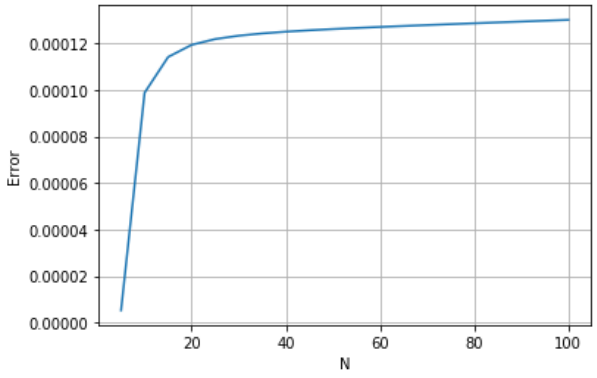


Figura 3.5: Errores en las aproximaciones a $V(1,0)$ con el método simétrico $\Delta t = 10^{-6}$.

Finalmente, presentamos gráficas del error obtenidos por cada método.

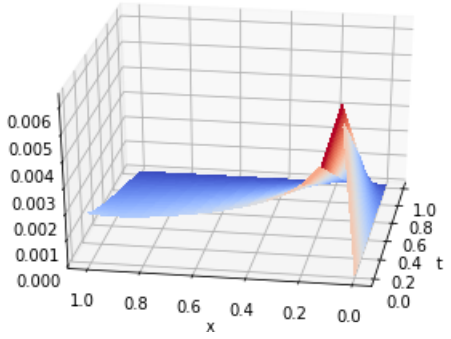


Figura 3.6: Errores producidos por el método upwind con $M = 40$ y $N = 15$

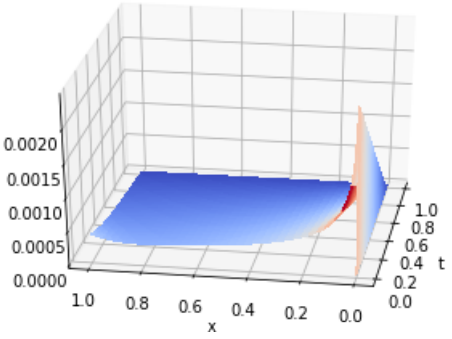


Figura 3.7: Errores producidos por el método upwind con $M = 3000$ y $N = 100$.

Al comparar las figuras 3.6 y 3.7 se aprecia una disminución de los errores producidos por el método upwind. También se visualiza un pico cercano a $(0, 0)$, esto se debe a un error que se propagó durante la ejecución del algoritmo.
Similarmente mostramos las gráficas correspondientes al método simétrico eligiendo los mismos parámetros que en el método upwind.
Se puede apreciar también la mejora de la aproximación del esquema simétrico frente al upwind con los mismos parámetros, se pueden comparar las figuras 3.6 y 3.8; o las figuras 3.7 y 3.9.

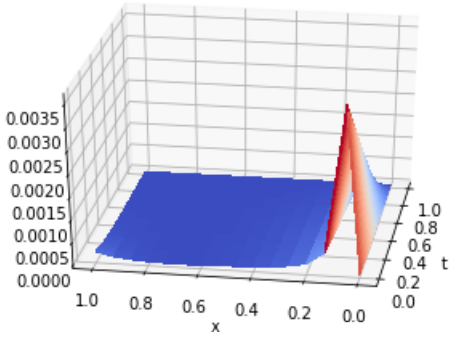


Figura 3.8: Errores producidos por el método simétrico con $M = 40$ y $N = 15$.

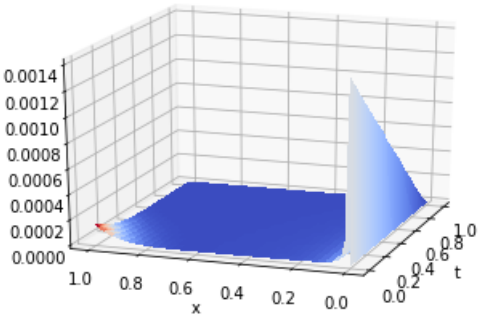


Figura 3.9: Errores producidos por el método simétrico con $M = 3000$ y $N = 100$.

Nótese que los errores en cada método tuvieron una disminución proporcional cuando aumentaron los valores de M y N.

Capítulo 4

Método Monte Carlo

El método Monte Carlo es ampliamente conocido en el campo de las ciencias computacionales. Este método fue publicado en 1949 como "The Monte Carlo Method" por los autores Metropolis y Ulam en la *Journal of the American Statistical Association*. Sin embargo, ya se había usado en la segunda guerra mundial por J. Von Neuman y S. Ulam, personajes que son conocidos como los fundadores del método Monte Carlo.
La idea del método Monte Carlo, es aproximar un valor esperado $\mathbb{E}(X)$ de una variable aleatoria X mediante el cálculo del promedio de un gran número de los resultados de experimentos independientes, los cuales tienen la misma distribución de X. El fundamento teórico sobre el cual está basado este método es el teorema fuerte de los grandes números, el cual afirma que la media aritmética de una sucesión de variables aleatorias de valor real, independientes e idénticamente distribuidas $\{X_n\}_{n\in\mathbb{N}}$, converge con probabilidad 1, al valor esperado $\mathbb{E}(X_1) = \mu$ que por supuesto es el mismo para todo X_n.
Con base en la idea del método Monte Carlo, vamos a aproximar la función de valor (2.25) mediante la simulación de un número grande K de caminos del movimiento Browniano, cada uno de ellos con M pasos en el tiempo t, los cuales nos ayudarán en la construcción del proceso de riqueza (2.24). Luego en el tiempo final $t_M = T$ evaluamos la función de utilidad $U(x_k(t_M))$ para $k = 1, ..., K$ dado un nivel de inversión π; después, calculamos la media aritmética $\frac{1}{K}\sum_{k=1}^{K} U(x_k(t_M))$ lo que nos servirá como aproximación de $\mathbb{E}\{U(x_T)\}$. Finalmente, y puesto que π es constante en el tiempo, se procede a encontrar $\sup_{\pi\in[0,1]} \mathbb{E}\{U(x_T)\}$ mediante dos procesos de optimización que llamamos el "método exhaustivo" y el "método Golden section Search". Como una observación, los valores de μ, σ, γ y r, serán los mismos que en la sección 3.3.

4.1 Implementación del método

Iniciamos simulando $K = 250.000$ caminos del movimiento Browniano estándar y $M = 100$ pasos en el tiempo; $T = 1$; cada paso mide $\Delta t = T/M$; la generación de tales caminos $W_k(t)$ se puede llevar a cabo aplicando el método de *Euler-Maruyama* a las dinámicas:

$$\begin{aligned} \Delta W_k(t_i) &= Z\sqrt{\Delta t}, & Z \sim \mathcal{N}(0,1), \\ W_k(0) &= 0, & i = 1, ..., M, \end{aligned}$$

para $k = 1, ..., K$. Luego, con el mismo procedimiento simulamos el proceso de riqueza (2.24) con un π fijo:

$$\begin{aligned} x_k^\pi(t_i) &= x_k^\pi(t_{i-1}) \exp\left[\left(r + (\mu - r)\pi - \frac{1}{2}\sigma^2\pi^2\right)\Delta t + \sigma\pi\Delta W_k(t_i)\right] \\ x_k^\pi(0) &= 1, \text{ (riqueza inicial)} \qquad i = 1, ..., M \end{aligned} \tag{4.1}$$

Además, para el instante $t_M = T$ se aplica la función de utilidad:

$$U(x) = \frac{x^\gamma}{\gamma}$$

Para concluir la implementación del método Monte Carlo, encontramos la media aritmética:

$$\frac{1}{K}\sum_{k=1}^{K} U(x_k^\pi(T))$$

para obtener una aproximación de $\mathbb{E}(U(x^\pi(T)))$.
Puesto que nuestro propósito es encontrar $\sup_{\pi\in[0,1]} \mathbb{E}(U(x^\pi(T)))$, formulamos una función $\hat{V} = \hat{V}(\pi) = \frac{1}{K}\sum_{k=1}^{K} U(x_k^\pi(T))$ nuestra función objetivo con $\pi \in [0,1]$. Exponemos dos métodos de optimización de esta función: el método exhaustivo y golden section search.

4.2 Método Exhaustivo

Supongamos que queremos encontrar los valores extremos de una función f en un intervalo cerrado $[a,b]$ junto con su argumento óptimo $q^* \in [a,b]$. El método exhaustivo consiste en definir $\{q_0 = 1, q_1, ..., q_n = b\}$ una partición de $[a,b]$ cuyos valores usualmente se toman igualmente distanciados; luego evaluamos la función f en q_j, $j = 1, ..., n$, al terminar se escoge el q_j^* para el cual $f(q_j^*)$ sea el valor más grande (o pequeño). Cabe aclarar que entre más fina se tome la partición, se espera que f se aproxime mejor a su valor extremo.
Nosotros tomamos la partición $P = \{0, 0.01, 0.02, ..., 0.98, 0.99, 1.00\}$ e implementamos la descripción anteriormente dada. Fijando una semilla en el paquete numpy de `Python` y tomando el tiempo de ejecutación, obtenemos los siguientes resultados numéricos:
La aproximación a V nos da 2.0751 con argumento máximo $\hat{\pi} = 0.77$ demorándose 209.92 segundos. La siguiente gráfica muestra el comportamiento de $\hat{V}$ contra π durante la implementación del método exhaustivo:

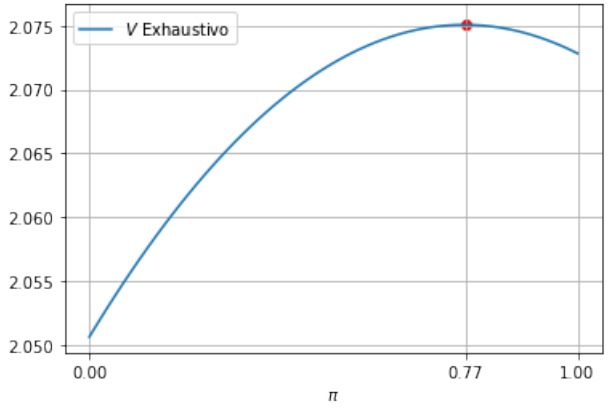


Figura 4.1: Gráfica de $\hat{V}$ vs. π, método Exhaustivo.

4.2.1 Análsis de error del método exhaustivo

Por cada elemento en la partición se mide el error absoluto con respecto al valor verdadero $V \approx 2.0738$, obteniendo la siguiente gráfica:

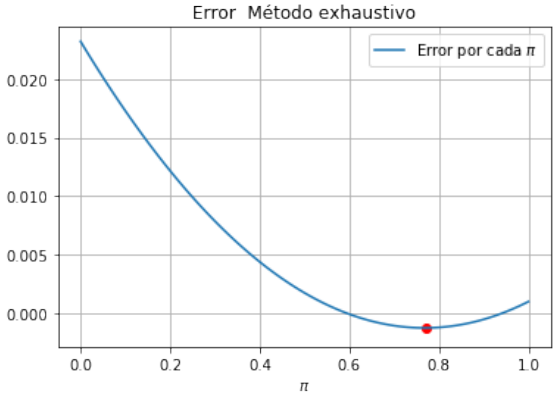


Figura 4.2: Gráfica de los errores

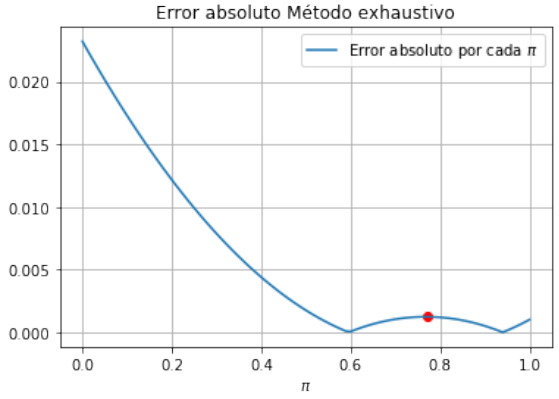


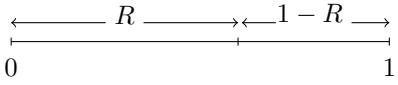
Figura 4.3: Gráfica de los errores absolutos

Observando la figura 4.1, encontramos que $\hat{V} > V(1,0)$, lo que nos indica que el método Monte Carlo con optimización exhaustiva y las especificaciones indicadas, alcanza un valor mayor al analítico . Por esta razón, en la figura 4.2 al calcular los errores producidos por el método en cada π con respecto a $V(1,0)$, se puede ver que una porción del gráfico está por debajo del eje horizontal. Nótese que la optimización no se refiere a minimizar el error producido, ya que de ser así, el método de exhaustivo presentaría una falla porque la función $\hat{V}$ tiene dos óptimo, véase la figura 4.3.
También medimos la media de errores porcentuales tanto para los valores de $\hat{V}$ como los de $\hat{\pi}$:

$$\begin{aligned} \text{Error porcentual}_V &= \frac{|V-\hat{V}|}{V}100\% = 0.06\% \\ \text{Error porcentual}_\pi &= \frac{|\pi^*-\hat{\pi}|}{\pi^*}100\% = 2.7\% \end{aligned}$$

4.3 Método "Golden Section Search"

Supongamos que dividimos el intervalo $[0,1]$ en subintervalos separados de longitudes R y $1-R$, como se vé en el siguiente gráfico



Se dice que estos subintervalos son divididos en la *razón áurea* si la longitud de todo el intervalo sobre la longitud del segmento más largo es igual a la longitud del segmento más largo sobre la longitud del segmento más pequeño. Esto puede ser escrito como $1/R = R/(1-R)$ o $R^2 + R - 1 = 0$ por ser $R > (1-R)$. Al resolver esta ecuación obtenemos las soluciones

$$R_1 = \frac{\sqrt{5}-1}{2} \qquad \text{y} \qquad R_2 = -\frac{\sqrt{5}+1}{2}$$

elegimos $R = R_1$ la raíz positiva que se encuentra en el intervalo $[0,1]$. El número R_1 es aproximadamente 0.618 y es conocido como la razón áurea.
Supongamos que f es una función unimodal en el intervalo $[a,b]$. El objetivo del método es encontrar el valor óptimo y argumento óptimo. La longitud del intervalo final puede ser controlada y puede ser arbitrariamente pequeña escogiendo un nivel de tolerancia. La longitud del intervalo final será menor que nuestro nivel de tolerancia especificado.
El procedimiento para encontrar una aproximación al valor máximo es iterativo. Requiere de las evaluaciones de f en los puntos de prueba $x_1 = b - R(b-a)$ y $x_2 = a + R(b-a)$ y luego se determina el nuevo intervalo. Si $f(x_1) < f(x_2)$, entonces el nuevo intervalo es (x_1, b); si $f(x_1) > f(x_2)$, entonces el nuevo intervalo es (a, x_2). Las iteraciones continuan hasta que la longitud del intervalo final sea menor que la tolerancia impuesta, y el intervalo final contiene el punto de la solución óptima.
Para la implementación, consideramos $[a,b] = [0,1]$, una función $f = f(\pi)$ la cual es expresada como el valor promedio de las simulaciones cuyo paso final es evaluado en la función de utilidad; además consideramos una tolerancia $\epsilon = 0.001$.
Como resultados obtenidos al aplicar este algoritmo, se obtiene la aproximación de $\hat{V} = 2.0751$, el argumento máximo $\hat{\pi} = 0.76786$ y el tiempo transcurrido: 62.94 segundos.
Podemos ver que en términos de optimización de tiempo, el método "Golden section Search" es de mejor desempeño comparado al exhaustivo. Presentamos la gráfica correspondiente de $\hat{V}$ contra π:

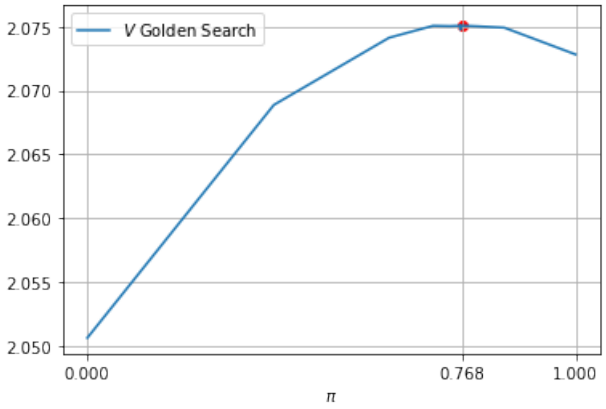


Figura 4.4: Gráfica de $\hat{V}$ vs. π, Golden Search.

En la figura 4.4, se pueden visualizar pliegues o esquinas en la gráfica, esto se debe a que no se está haciendo una revisión exhaustiva como el método anterior, los puntos en el eje horizontal donde se presentan las esquinas en la gráfica son precisamente los que se obtienen al hacer las particiones con el método Golden Section Search.

4.3.1 Análisis de error para el método "Golden section Search"

Al igual que en el anásis del error del método exhaustivo, obtenemos la gráfica de los errores absolutos medidos en cada partición de cada iteracion:

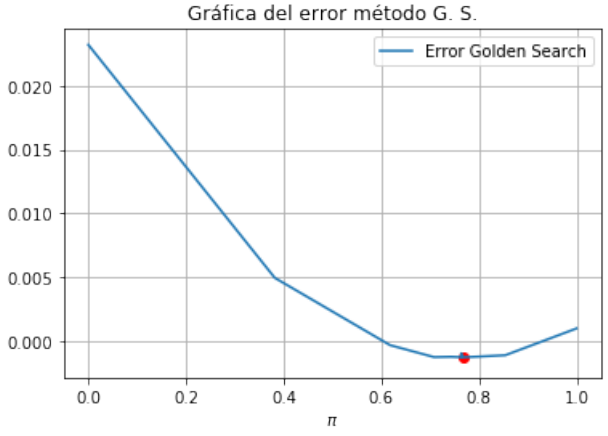


Figura 4.5: Gráfica de los errores generada por el método "Golden section Search"

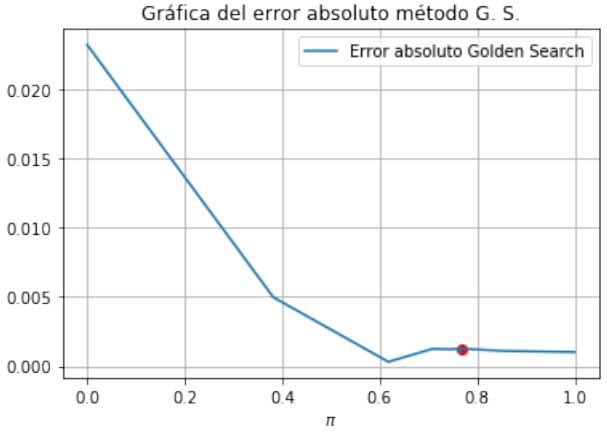


Figura 4.6: Gráfica de los errores generada por el método "Golden section Search".

Si observamos las figuras 4.5 y 4.6 que corresponden a los errores producidos por el método Golden Section Search con respecto a $V(1,0)$, obtenemos un valor mayor en la funcion objetivo que la obtenida analíticamente como con el exhaustivo. La ventaja del método Golden Section Search sobre el exhaustivo es claramente la rapidez en la obtención de los resultados cuyos tiempos de ejecución, recordemos que son 209.92 segundos para el exhaustivo y 62.94 segundos para Golden Section Search.
Ahora medimos los errores porcentuales del valor óptimo aproximado y el argumento óptimo aproximado

$$\begin{aligned} \text{Error porcentual}_V &= \frac{|V-\hat{V}|}{V}100\% = 0.06\%, \\ \text{Error porcentual}_\pi &= \frac{|\pi^*-\hat{\pi}|}{\pi^*}100\% = 2.4\%. \end{aligned}$$

Capítulo 5

Diferencias Finitas

Este método consiste en hallar una solución numérica a una ecuación diferencial parcial (EDP) dada. Un ejemplo es el caso de la ecuación de Black-Scholes la cual es una herramienta indispensable a la hora de valorar contratos derivados.
El método de diferencias finitas que empleamos está basado en la idea natural de aproximar derivadas parciales con cocientes de diferencias. Este método involucra la discretización de cada derivada en la EDP de acuerdo a la expansión de Taylor de algún orden requerido. Hay distintas formas de aproximaciones de las derivadas parciales, algunas de las cuales describimos en la primera sección de este capítulo. Por ejemplo, en el capítulo 3, usamos un tipo de aproximación conocido como diferencias hacia adelante (*forward*) para Y_t; *upwind* para el término Y_x, y la diferencia central para la segunda derivada Y_{xx}; mientras que para el método simétrico, el término Y_x se aproximó mediante la *diferencia central simétrica*.
En la sección 5.1 presentamos las condiciones que necesitamos para la solución numérica del problema de Merton en diferencias finitas. En la sección 5.2 implementamos el esquema implícito. En la sección 5.3, mostramos un método computacional para resolver el sistema de ecuaciones adquirido por el esquema implícito y finalmente, en la sección 5.4 presentamos los resultados y errores del método.

5.1 Consideraciones Iniciales

Empleamos un esquema conocido como *implícito* para resolver el problema (2.26). Como en el capítulo 3, queremos ver el desempeño del método en la aproximación de V y π^* en el punto $(1, 0)$, por tanto vamos a obtener una discretización del dominio continuo $D = \{(x, t)|0 \leq x \leq 2 \text{ y } 0 \leq t \leq 1\}$ a una grilla discreta. Se considerarán M subintervalos igualmente espaciados en τ y N subintervalos igualmente espaciados para x. Con lo cual, tenemos la siguiente grilla:

$$\bar{D} = \left\{ \begin{array}{lll} x_j = 0 + j\Delta x; & \Delta x = \frac{2-0}{N}; & j = 0, ..., N; \\ \\ t_k = 0 + k\Delta t; & \Delta t = \frac{1-0}{M} & k = 0, ..., M. \end{array} \right\}$$

Una vez conocido nuestro dominio de aproximación, establecemos el problema de tiempo continuo con valor final y condiciones de frontera:

$$V_t + \sup_{\pi \in [0,1]} \left\{ rxV_x + \pi(\mu - r)xV_x + \frac{1}{2}x^2\sigma^2\pi^2 V_{xx} \right\} = 0 \tag{5.1}$$

$$\begin{aligned} \text{Condición final} \qquad & V(x, T) = \frac{x^\gamma}{\gamma}, \\ & V(0, t) = 0 \\ \text{Condiciones de frontera} \qquad & V(2, t) = \frac{2^\gamma}{\gamma} \exp\left[(T - t) \left(r\gamma + \frac{1}{2} \frac{(\mu - r)^2}{\sigma^2} \frac{\gamma}{1 - \gamma} \right) \right] \end{aligned}$$

Esta es una ecuación tipo *backward*, no obstante, haciendo el cambio de variable $\tau = T - t$, obtenemos una ecuación tipo *forward*:

$$-\tilde{V}_\tau + \sup_{\pi\in[0,1]}\{x(r+\pi(\mu-r))\tilde{V}_x + \frac{1}{2}x^2\sigma^2\pi^2\tilde{V}_{xx}\} = 0 \tag{5.2}$$

$$\begin{aligned} &\text{Condición inicial} && \tilde{V}(x,0) = \frac{x^\gamma}{\gamma} \\ & && \tilde{V}(0,\tau) = 0 \\ &\text{Condiciones de frontera} && \tilde{V}(2,\tau) = \frac{2^\gamma}{\gamma}\exp\left[\tau\left(r\gamma + \frac{1}{2}\frac{(\mu-r)^2}{\sigma^2}\frac{\gamma}{1-\gamma}\right)\right] \end{aligned}$$

Así, tenemos que

$$\tilde{V}(x,\tau) \quad = \quad V(x,T-t).$$

Gran parte de la notación fue tomada de [10].
Nuestro objetivo en este capítulo es resolver numéricamente el problema (5.2) en el dominio $\bar{D}$ con el esquema implícito explicado en la siguiente sección.
Notaremos como $v_{j,k}$ el valor aproximado de $\tilde{V}(x = x_j, \tau = \tau_k)$ y los parámetros μ, r, σ, γ y π los consideramos constantes.

5.2 Discretización Implícita

Para la discretización implicita, en el punto (x_j, τ_{k+1}) de la grilla en el tiempo $k+1$ por el cambio de t a τ, usaremos la aproximación de diferencias hacia adelante para el término $\frac{\partial\tilde{V}}{\partial\tau}$, una aproximación de diferencias central para la derivada parcial $\frac{\partial\tilde{V}}{\partial x}$ y una aproximación de diferencias central simétrica para la segunda derivada parcial $\frac{\partial^2\tilde{V}}{\partial x^2}$. El algoritmo para llevar a cabo este esquema requiere resolver un sistema de ecuaciones, que se debe resover para conocer la solución numérica.
La aproximación hacia atrás para $\frac{\partial\tilde{V}}{\partial\tau}$ produce

$$\frac{\partial\tilde{V}}{\partial\tau} = \frac{v_{j,k+1} - v_{j,k}}{\Delta\tau} + O(\Delta\tau)$$

La aproximación central simétrica para $\frac{\partial\tilde{V}}{\partial x}$ nos da

$$\frac{\partial\tilde{V}}{\partial x} = \frac{v_{j+1,k+1} - v_{j-1,k+1}}{2\Delta x} + O((\Delta x)^2)$$

y la aproximación central en diferencias para el término $\frac{\partial^2\tilde{V}}{\partial x^2}$ nos da

$$\frac{\partial^2\tilde{V}}{\partial x^2} = \frac{v_{j-1,k+1} - 2v_{j,k+1} + v_{j+1,k+1}}{(\Delta x)^2} + O((\Delta x)^2)$$

De manera similar, despejamos Luego, al sustituir estas expresiones en la ecuación (5.2) y eliminar los términos de error y el símbolo sup obtenemos la ecuación en el punto (x_j, τ_{k+1}):

$$\begin{aligned} -\left(\frac{v_{j,k+1} - v_{j,k}}{\Delta\tau}\right) + x_j(r+(\mu-r)\pi)\left(\frac{v_{j+1,k+1} - v_{j-1,k+1}}{2\Delta x}\right) \\ +\frac{1}{2}x_j^2\sigma^2\pi^2\left(\frac{v_{j-1,k+1} - 2v_{j,k+1} + v_{j+1,k+1}}{(\Delta x)^2}\right) = 0 \end{aligned}$$

Multiplicando por $\Delta\tau$ obtenemos

$$\begin{aligned} v_{j,k} - v_{j,k+1} + x_j(r+(\mu-r)\pi)\frac{\Delta\tau}{2\Delta x}(v_{j+1,k+1} - v_{j-1,k+1}) \\ +\frac{1}{2}x_j^2\sigma^2\pi^2\frac{\Delta\tau}{(\Delta x)^2}(v_{j-1,k+1} - 2v_{j,k+1} + v_{j+1,k+1}) = 0 \end{aligned}$$

Ahora, si definimos $\alpha_j = \frac{1}{2}x_j^2\sigma^2\pi^2\frac{\Delta\tau}{(\Delta x)^2}$ y $\beta_j = x_j(r+(\mu-r)\pi)\frac{\Delta\tau}{2\Delta x}$, entonces nuestra ecuación se reduce a

$$v_{j,k} = v_{j-1,k+1}(\beta_j - \alpha_j) + v_{j,k+1}(2\alpha_j + 1) + v_{j+1,k+1}(-\beta_j - \alpha_j) \tag{5.3}$$

Sean ahora, $\hat{l}_j = \beta_j - \alpha_j$, $\hat{d}_j = 2\alpha_j + 1$ y $\hat{u}_j = -\beta_j - \alpha_j$, de tal manera que la forma matricial queda recursivamente dada por

$$A_{k+1}^{\text{Implicit}}\mathbf{v}_{k+1} = \mathbf{v}_k + \begin{bmatrix} -\hat{l}_1 v_{0,k+1} \\ 0 \\ \vdots \\ 0 \\ -\hat{u}_{N-1}v_{N,k+1} \end{bmatrix} \qquad k = 0, ..., M-1 \tag{5.4}$$

donde

$$\mathbf{v}_k = \begin{bmatrix} v_{1,k} \\ v_{2,k} \\ \vdots \\ v_{N-2,k} \\ v_{N-1,k} \end{bmatrix} \quad \text{y} \quad A_k^{\text{Implicit}} = \begin{bmatrix} \hat{d}_1 & \hat{u}_1 & & & \\ \hat{l}_2 & \hat{d}_2 & \hat{u}_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \hat{l}_{N-2} & \hat{d}_{N-2} & \hat{u}_{N-2} \\ & & & \hat{l}_{N-1} & \hat{d}_{N-1} \end{bmatrix}$$

Por la expresión (5.3), debemos despejar $\mathbf{v}_{k+1}$ en términos de $\mathbf{v}_k$, lo cual implica la inversión de la matriz $A_{k+1}^{\text{Implicit}}$. Aprovechando la forma tridiagonal de esta matriz, resolvemos el sistema (5.3) con el siguiente método.

5.3 Método para resolver un Sistema Tridiagonal

Asumimos A una matriz $(N-1)\times(N-1)$ tridiagonal de la forma

$$\begin{bmatrix} d_1 & u_1 & 0 & \cdots & 0 & 0 \\ l_2 & d_2 & u_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & l_{N-2} & d_{N-2} & u_{N-2} \\ 0 & 0 & \cdots & 0 & l_{N-1} & d_{N-1} \end{bmatrix}$$

y queremos resolver el sistema lineal $A\mathbf{x} = \mathbf{b}$ para $\mathbf{x} \in \mathbb{R}^{N-1}$ (dada la forma de A, de aquí el nombre de sistema tridiagonal), donde $\mathbf{b}$ es un vector $(N-1)\times 1$. Queremos resolver este sistema lineal haciendo los elementos de la diagonal superior cero y luego resolver el sistema o transformar en cero los elementos de la diagonal inferior y luego resolver el sistema. Aquí primero hacemos la diagonal inferior en ceros; para este objetivo procedemos como sigue:

a. Multiplicamos la primera fila por $\frac{-l_2}{d_1}$ y lo sumamos a la segunda fila para eliminar l_2. Con este resultado tenemos ahora $\tilde{d_2} = d_2 - \frac{l_2}{d_1}u_1$ y $\tilde{b_2} = b_2 - \frac{l_2}{d_1}b_1$.

b. Multiplicamos la segunda fila por $-\frac{l_3}{\tilde{d_2}}$ y la sumamos a la tercer fila para eliminar l_3. Conseguimos entonces $\tilde{d_3} = d_3 - \frac{l_3}{\tilde{d_2}}u_2$ y $\tilde{b}_3 = b_3 - \frac{l_2}{\tilde{d_2}}\tilde{b}_2$.

c. Repetimos hasta que todos los elementos en la diagonal inferior de A sean eliminados.

Ahora tenemos

$$\begin{bmatrix} d_1 & u_1 & 0 & \cdots & 0 \\ 0 & \tilde{d_2} & u_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \tilde{d}_{N-2} & u_{N-2} \\ 0 & 0 & \cdots & 0 & \tilde{d}_{N-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{b}_{N-2} \\ \tilde{b}_{N-1} \end{bmatrix}$$

Ahora podemos resolver para $\mathbf{x}$ empezando desde la $N-1$- fila:

$$\begin{aligned} \tilde{d}_{N-1}x_{N-1} &= \tilde{b}_{N-1} \\ x_{N-1} &= \frac{\tilde{b}_{N-1}}{\tilde{d}_{N-1}} \end{aligned}$$

Teniendo x_{N-1}, podemos resolver para x_{N-2} de la $(N-2)$- fila:

$$\begin{aligned} \tilde{d}_{N-2}x_{N-2} + u_{N-2}x_{N-1} &= \tilde{b}_{N-2} \\ x_{N-2} &= \frac{\tilde{b}_{N-2} - u_{N-2}x_{N-1}}{\tilde{d}_{N-2}} \end{aligned}$$

y podemos repetir este proceso hasta llegar a resolver para x_1.

5.4 Resultados y Errores

Queremos ahora investigar cómo se ajusta el esquema de diferencias finitas a la función de valor óptimo V. Asignamos los valores: $N = 10.000$ ($\Delta x = 10^{-4}$) y $M = 100$ ($\Delta\tau = 10^{-2}$), usamos los mismos valores establecidos desde el capítulo 3 a los parámetros μ, σ, r, γ y $\pi^* = \frac{\mu - r}{\sigma^2(1-\gamma)}$ (lo asumimos constante por ahora); posteriormente escogemos hacer la comparación en la grilla sobre los puntos (x_j, τ_M) para $j = 1, ..., N$ y luego medimos el error obtenido. A continuación, mostramos en las siguientes gráficas resultados correspondientes a los errores producidos por el esquema:

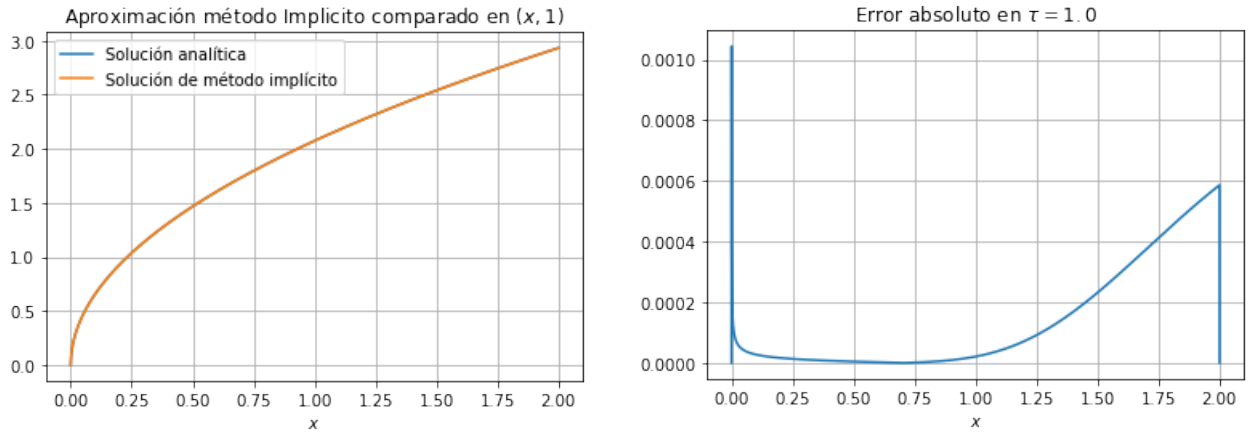


Figura 5.1: Gráficas de V con la aproximación del esquema implicito; y la gráfica del error ambas calculadas en $\tau = 1.0$.

Recordemos que la solución analítica $V(x,t)$ nos proporciona el valor máximo en $(x,t) = (1,0)$ cuando escogemos como dominio al cuadrado $[0,1]\times[0,1]$, es decir, cuando $(x,\tau) = (1,1)$ para $\tilde{V}(x,\tau)$. Denotando la aproximación a $\tilde{V}(1,1)$ como $\tilde{v}$ del método, encontramos que el error porcentual nos da $|\tilde{V}(1,1) - \tilde{v}|/\tilde{V}(1,1) \approx 0.001\%$.
Con los mismos parámetros presentamos dos gráficas que corresponden a la superficie de los errores generada por el método cuando $N = 10^4$ y $N = 10^5$:

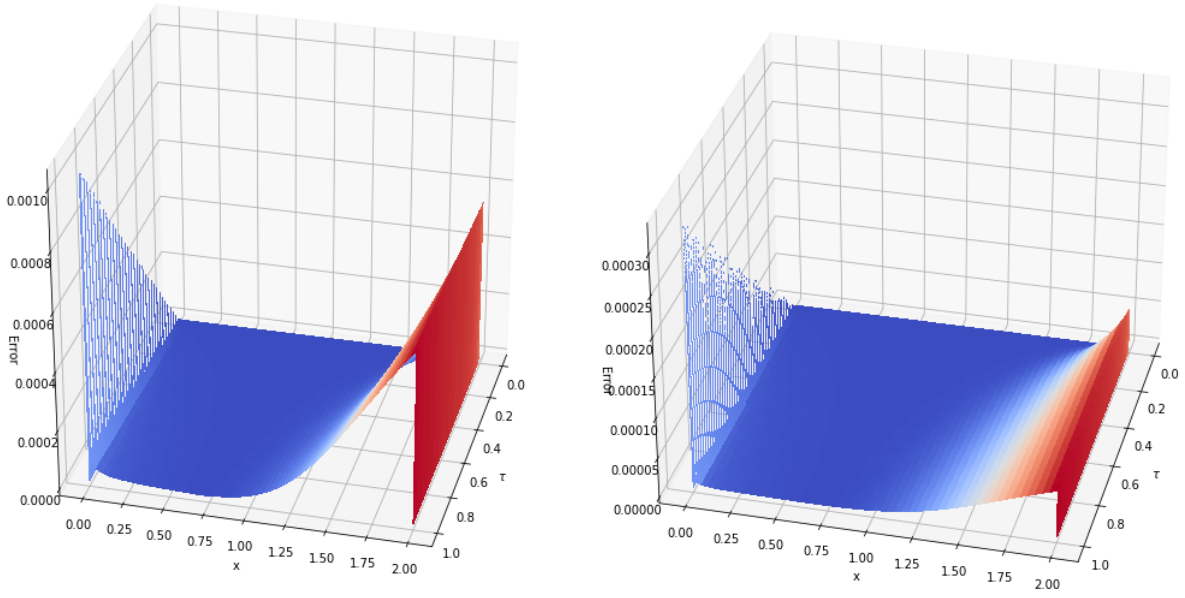


Figura 5.2: Superficie de los errores absolutos con método implícito con $N = 10^4$ (izquierda) y $N = 10^5$ (derecha).

Puesto que nuestro objetivo es encontrar aproximaciones de π^*, vamos a usar dos métodos para tal tarea: Golden Section Search (GSS) y Fibonacci. El último algoritmo es muy parecido al GSS, consiste en la reducción del intervalo de prueba en proporción de los números de la sucesión de Fibonacci $(1, 1, 2, 3, 5, 8, 13, ...)$. La idea aquí es construir una función que incorpore el método de diferencias finitas, cuyo argumento sea un parámetro π y cuyo valor es la aproximación de $\tilde{V}$ en $(1, 1)$, luego se usan los métodos con una tolerancia $\epsilon = 0.001 = 10^{-3}$ para la obtención del valor y del argumento máximos.
En la siguiente tabla se consignan los valores obtenidos en cuanto la aproximación de π^* con un $\Delta t = 10^{-2}$ fijo:

$\Delta t = 10^{-2}$ (GSS).			
N	π^* aprox.	Error abs.	Error %.
1×10^2	0.978800	2.29e-1	30.51
1×10^3	0.766218	1.62e-2	2.16
2×10^3	0.758087	8.09e-3	1.09
3×10^3	0.755435	5.43e-3	0.72
4×10^3	0.753795	3.80e-3	0.51
5×10^3	0.753062	3.06e-3	0.41
6×10^3	0.752782	2.78e-3	0.37
7×10^3	0.752329	2.33e-3	0.31
8×10^3	0.751876	1.88e-3	0.25
9×10^3	0.751596	1.60e-3	0.21
10×10^3	0.751596	1.60e-3	0.21

Tabla 5.1: Aproximaciones a π^* con $\Delta t = 10^{-2}$ método GSS.

$\Delta t = 10^{-2}$ (Fibonacci).			
N	π^* aprox.	Error abs.	Error %.
1×10^2	0.875125	1.83e-3	16.684
1×10^3	0.749938	6.25e-5	0.0083
2×10^3	0.749938	6.25e-5	0.0083
3×10^3	0.749938	6.25e-5	0.0083
4×10^3	0.749938	6.25e-5	0.0083
5×10^3	0.749938	6.25e-5	0.0083
6×10^3	0.749938	6.25e-5	0.0083
7×10^3	0.749938	6.25e-5	0.0083
8×10^3	0.749938	6.25e-5	0.0083
9×10^3	0.749938	6.25e-5	0.0083
10×10^3	0.749938	6.25e-5	0.0083

Tabla 5.2: Aproximaciones a π^* con $\Delta t = 10^{-2}$ método Fibonacci.

Mientras los valores aproximados de la función valor se representan en las siguientes tablas

$\Delta t = 10^{-2}$ (GSS).			
N	V aprox.	Error abs.	Error %.
1×10^2	2.075987	2.16e-3	0.10
1×10^3	2.073928	9.81e-5	0.005
2×10^3	2.073884	5.34e-5	0.003
3×10^3	2.073870	3.95e-5	0.002
4×10^3	2.073863	3.29e-5	0.001
5×10^3	2.073859	2.89e-5	0.001
6×10^3	2.073856	2.62e-5	0.001
7×10^3	2.073855	2.44e-5	0.001
8×10^3	2.073853	2.31e-5	0.001
9×10^3	2.073852	2.20e-5	0.001
10×10^3	2.073851	2.11e-5	0.001

Tabla 5.3: Aproximaciones a V con $\Delta t = 10^{-2}$ método GSS.

$\Delta t = 10^{-2}$ (Fibonacci).			
N	V aprox.	Error abs.	Error %.
1×10^2	2.075662	1.83e-3	0.088
1×10^3	2.073918	8.76e-5	0.004
2×10^3	2.073881	5.08e-5	0.002
3×10^3	2.073869	3.85e-5	0.002
4×10^3	2.073862	3.22e-5	0.001
5×10^3	2.073859	2.85e-5	0.001
6×10^3	2.073856	2.60e-5	0.001
7×10^3	2.073854	2.42e-5	0.001
8×10^3	2.073853	2.29e-5	0.001
9×10^3	2.073852	2.18e-5	0.001
10×10^3	2.073851	2.10e-5	0.001

Tabla 5.4: Aproximaciones a V con $\Delta t = 10^{-2}$ con método Fibonacci.

Las gráficas correspondientes a los errores absolutos producidos por los métodos para la aproximación de π^* son las siguientes:

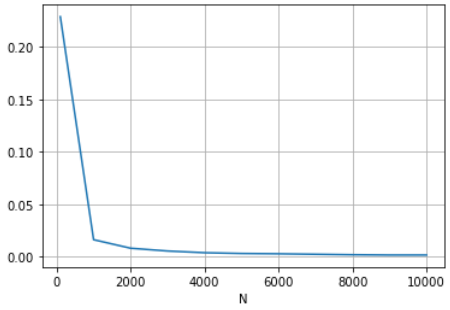


Figura 5.3: Errores absolutos de aproximación a π^* con método GSS,

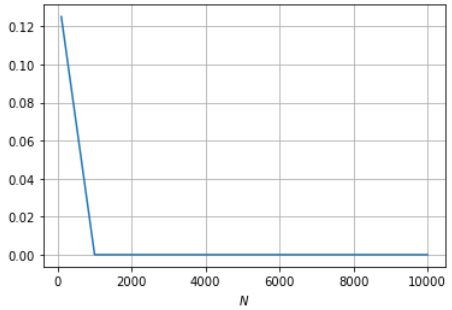


Figura 5.4: Aproximaciones a π^* con $\Delta t = 10^{-2}$ con método Fibonacci.

A continuación se presentan las gráficas de los errores absolutos producidos por los métodos para la aproximación de V:

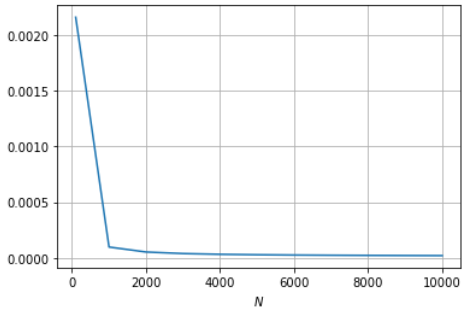


Figura 5.5: Errores absolutos de aproximación a V con método GSS ($\Delta t = 10^{-2}$),

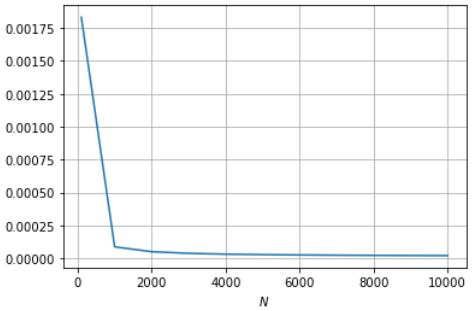


Figura 5.6: Aproximaciones a V con $\Delta t = 10^{-2}$ con método Fibonacci.

En lo que resta de sección, mostramos los resultados alcanzados en cuanto a la aproximación de π^* y V manteniendo a Δx fijo y variando Δt:

$\Delta x = 10^{-3}$ GSS.			
M	π^* Approx.	Error abs.	Error %.
100	0.766218	1.62e-2	2.16
200	0.766218	1.62e-2	2.16
400	0.766218	1.62e-2	2.16
800	0.766218	1.62e-2	2.16
1000	0.766218	1.62e-2	2.16
2000	0.766218	1.62e-2	2.16
4000	0.766218	1.62e-2	2.16
8000	0.766217	1.62e-2	2.16
10000	0.766217	1.62e-2	2.16

$\Delta x = 10^{-3}$ GSS.			
M	V Approx.	Error abs.	Error %.
100	2.073928	9.809e-5	0.0047
200	2.073920	8.985e-5	0.0043
400	2.073916	8.571e-5	0.0041
800	2.073914	8.365e-5	0.0040
1000	2.073913	8.323e-5	0.0040
2000	2.073912	8.241e-5	0.0040
4000	2.073912	8.199e-5	0.0040
8000	2.073912	8.178e-5	0.0039
10000	2.073912	8.174e-5	0.0039

Tabla 5.5: Aproximaciones de π^* y a V con GSS al esquema implícito.

Ahora presentamos tablas análogas (con los mismos parámetros elegidos) pero con el método de optimización de Fibonacci:

$\Delta x = 10^{-3}$ Fibonacci.			
M	π^* Approx.	Error abs.	Error %.
100	0.749938	6.25e-5	0.0083
200	0.749938	6.25e-5	0.0083
400	0.749938	6.25e-5	0.0083
800	0.749938	6.25e-5	0.0083
1000	0.749938	6.25e-5	0.0083
2000	0.749938	6.25e-5	0.0083
4000	0.749938	6.25e-5	0.0083
8000	0.749938	6.25e-5	0.0083
10000	0.749938	6.25e-5	0.0083

$\Delta x = 10^{-3}$ Fibonacci.			
M	V Approx.	Error abs.	Error %.
100	2.073918	8.758e-5	0.0042
200	2.073909	7.928e-5	0.0038
400	2.073905	7.513e-5	0.0036
800	2.073903	7.305e-5	0.0035
1000	2.073903	7.263e-5	0.0035
2000	2.073902	7.180e-5	0.0035
4000	2.073901	7.138e-5	0.0034
8000	2.073901	7.117e-5	0.0034
10000	2.073901	7.113e-5	0.0034

Tabla 5.6: Aproximaciones de π^* y a V con optimización Fibonacci al esquema implícito.

Como se puede ver en estas tablas, no hay variabilidad en la aproximación del argumento máximo tanto del algoritmo de golden search como el de Fibonacci, esto debido a que se trabajó con una tolerancia $\epsilon = 0.001$ en cada método; además, recordemos que el orden del error del método implícito para este caso es $O(\Delta\tau) + O((\Delta x)^2)$ el cual afecta al valor de la función de valor aproximada por el esquema. Nótese también, que el método de optimización de Fibonacci tiene una mejor aproximación que el algoritmo de Golden Search.
Nótese que las mejores aproximaciones tanto a $\tilde{V}(1,1)$ como a π^* se consiguen cuando $\Delta x = 0.0001$ ($N = 10000$), $\Delta t = 0.01$ ($M = 100$) con error porcentual 0.001% y 0.0083% respectivamente; y con una duración de ejecución de 373.6 segundos (6 minutos con 14 segundos aproximadamente) en el lenguaje `Python`.

Capítulo 6

Redes Neuronales

En este capítulo exponemos los conceptos y teoría necesarios para el entendimiento del manejo de las redes neuronales. Usamos como referencia teórica el libro de Aurélien Géron [8] y en la implementación las ideas proporcionadas por Jojo Moolayil en el texto [18].

Así como la idea que dio a luz el invento del avión fue inspirada por las aves, la idea del aprendizaje profundo e inteligencia artificial es basada en las redes neuronales biológicas. Una neurona biológica se activa mediante estimulos eléctricos externos los cuales la hacen reaccionar (o no) de forma que ella pueda transmitir una respuesta concreta dadas estas interacciones. Para el caso de inteligencia artificial, más concretamente, neuronas artificiales, los estimulos eléctricos que reciben son llamados *inputs* y las respuestas que representan la reacción de la neurona se llaman *outputs*. La suma ponderada de los inputs harán que la neurona artificial se active o no, mediante un mecanismo de evaluación de entradas, el cual se conoce como *función de activación*. Una *capa oculta* será conocida como el conjunto de neuronas que interactúan con los estimulos de entrada o con otras neuronas al mismo tiempo, cuando el modelo contempla una capa con una neurona, obtenemos la unidad básica de arquitectura neuronal, el *perceptrón*, la idea inicial es considerar más capas y más de una neurona; y mediante la minimización del error, poder determinar cuál es el número óptimo para ajustar nuestro modelo. En otros términos, una red neuronal artificial o simplemente, red neuronal, es un modelo estadístico no lineal; es implementado usualmente para hacer tareas de regresión o clasificación.
En nuestro estudio actual, ampliaremos nuestro espectro de posibilidades para los parámetros que han sido seleccionados como fijos en las secciones anteriores; la idea es que ellos nos sirvan como inputs a una red neuronal que al comienzo, será una estructura básica con una capa oculta; el objetivo será predecir el argumento máximo de (2.25) y comparar con el control óptimo del problema de Merton (2.29).

6.1 Perceptrón

Un perceptrón está compuesto de una capa oculta y una neurona conectada a los inputs. Estas conexiones son frecuentemente representadas como los *pesos*. Además, al conjunto de inputs se adiciona un término de sesgo $x_0 = 1$. Esta es una característica en el sesgo que usualmente es representada usando un tipo especial de neurona llamada *neurona de sesgo*, tiene en todo tiempo el valor de 1. El perceptrón calcula sumas ponderadas de sus inputs ($z = w_0 + w_1x_1 + w_2x_2 + \cdots w_nx_n = \mathbf{x}^\top\mathbf{w}$), luego aplica la función de activación[1] a esa suma y el resultado de la salida será $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(z)$ donde $z = \mathbf{x}^\top\mathbf{w}$, como se muestra en la figura 6.1.

[1]Para esta explicación usamos la función de activación *step* definida más adelante.

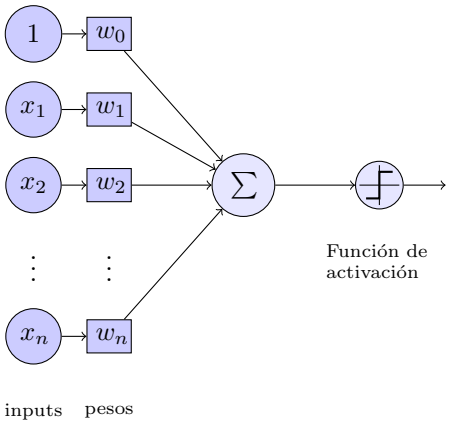


Figura 6.1: Arquitectura de un perceptrón

En términos del álgebra lineal, se pueden calcular eficientemente las salidas de una capa de neuronas artificiales para varias instancias a la vez:

$$h_{\mathbf{W},\,\mathbf{b}} = \varphi(\mathbf{XW} + \mathbf{b})$$

donde

- $\mathbf{X}$ representa la matriz de las variables independientes (*inputs*). Tiene una columna por variable y una fila para cada observación.
- La matriz de pesos $\mathbf{W}$ contiene todos los pesos que representan las conexiones excepto para los de la neurona del sesgo. Tienen una fila por cada neurona de entrada y una columna por cada neurona artificial en la capa.
- El vector de sesgo $\mathbf{b}$ contiene los pesos asociados al input x_0 entre la neurona de sesgo y las neuronas artificiales. Hay un término de sesgo por neurona artificial.
- La función φ es llamada función de activación.

¿Cómo se entrena un perceptrón?, la respuesta a esta pregunta está inspirada en la idea formulada en 1949 conocida como *la regla de Hebbs* en el libro *La organización del comportamiento*, consiste en que una neurona con capacidad de activar otra neurona, forma una conexión más fuerte entre las dos. Los perceptrones son entrenados usando una variante de esta regla teniendo en cuenta el error producido por la red al hacer una predicción. La regla se resume en la ecuación siguiente:

$$w_{i,j}^{(\text{próximo paso})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{6.1}$$

donde[2]

- $w_{i,j}$ es el peso que representa la conexión entre la neurona de entrada $i-$ésima y la neurona de salida $j-$ésima.
- x_i es el $i-$ésimo valor de los inputs.
- $\hat{y}_j$ es la salida de la $j-$ésima neurona de salida para la instancia de entrenamiento actual.

[2]Esta fórmula se deriva como un forma de minimizar a la función (*de pérdida*) objetivo:

$$\sum \eta(y - \hat{y})^2 = \sum \eta(y - \varphi(\mathbf{XW} + \mathbf{b}))^2.$$

- y_j es el resultado objetivo de la $j-$ésima neurona de la instancia de entrenamiento actual.
- η es la tasa de aprendizaje.

En la próxima sección se explica cómo está concatenada la red neuronal. Dado que las neuronas están conectadas con inputs y outputs completamente de manera *secuencial* a dicha conexión la conocemos como conexión *densa*.

6.2 Arquitectura Multi-Capa de una Red Neuronal Profunda y Propagación Hacia Atrás

Los perceptrones por sí solos no tienen una mayor ventaja frente a otros modelos de aprendizaje de máquina. Presentan limitaciones serias frente, incluso a algunos problemas de linealidad como por ejemplo el problema clasificación del O exclusivo XOR[3]. No obstante, las limitaciones de los perceptrones pueden ser eliminadas si enlazamos múltiples perceptrones ubicados en una serie secuencial de capas (agrupaciones de neuronas). Las *capas ocultas* son las capas que no incluyen los inputs y outputs visibles.
Un modelo de red neuronal con dos o más capas ocultas es llamado *red neuronal profunda* (RNP) como se vé en la figura siguiente.

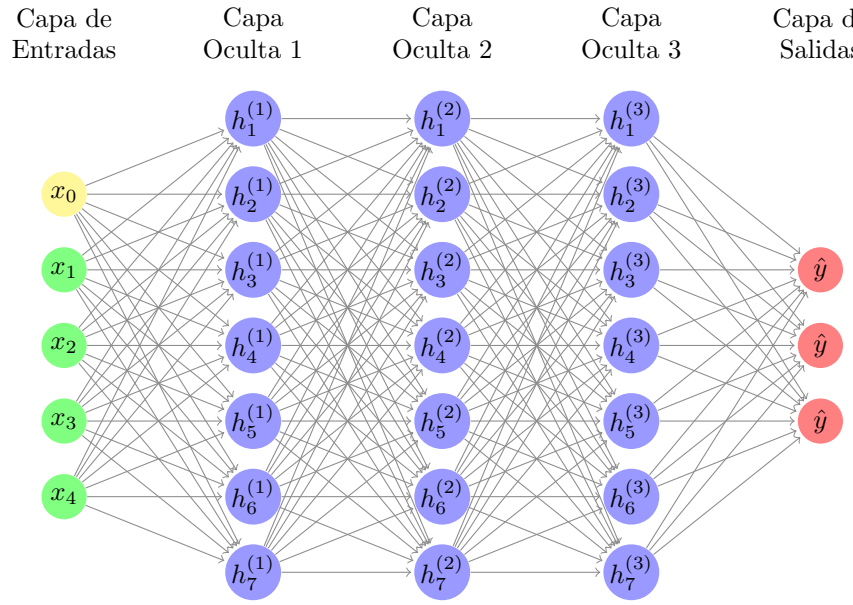


Figura 6.2: Estructura de una red Neuronal profunda con tres capas ocultas.

Se usa el algoritmo de propagación hacia atrás o *Backpropagation* en inglés. Por cada instancia de entrenamiento la propagación hacia atrás primero hace una predicción (paso adelante), mide el error, luego cruza cada capa en reversa para medir la contribución del error de cada conexión (paso en reversa), y finalmente retoca los pesos en las conexiones para reducir el error (el paso de Gradiente Descendiente). La función objetivo para nuestro caso será la *media de errores cuadráticos*:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.2}$$

El algoritmo se resume en los siguientes pasos:

- Procesa un mini lote (batch por su traducción del inglés), el cual es un número de muestras en las entradas que se propagarán en la red al tiempo (por ejemplo, se podrían tomar 50 observaciones por

[3]Véase por ejemplo [8] página 288.

lote). Cada paso considerado desde la capa de las entradas hasta la medición del error en la capa de salida, se le conoce como *época*.

- Cada mini lote se toma desde la capa visible de inputs, luego se envía a la primera capa oculta. El algoritmo luego calcula la salida de todas las neuronas en esta capa (para cada instancia en el mini lote). Luego de manera secuencial, el resultado pasa a la siguiente capa, su resultado es calculado y pasa a la siguiente capa y así hasta lleguemos a la última capa, la capa de los resultados. Este es el paso hacia adelante (o *forward phase*), exactamente como hacer predicciones, excepto que todos los resultados intermedio son preservados ya que se necesitan para la propagación hacia atrás.

- Posteriormente, se miden los errores mediante la función de pérdida en la capa de los outputs y se calcula el gradiente de esta función con respecto a los pesos en cada nodo. Luego se actualizan los pesos en las conexiones de la capa de las salidas, como en (6.1)). Continuando en la capa oculta anterior a la capa de los outputs, se calculan los gradientes por nodos nuevamente, recordando que la función de pérdida se puede expresar como una composición de los parámetros en las neuronas de esta capa (de aquí viene la importancia de la regla de la cadena) y se actualizan los pesos. De esta manera, continuamos la corrección de pesos llevando un proceso en reversa de capa tras capa hasta terminar en la capa de los inputs.

6.3 Funciones de activación

Ahora, ¿por qué necesitamos funciones de activación?, el trabajo de la función de activación es dar no linealidad a los pesos entre capas, esto ya que al trabajar componiendo sólamente transformaciones lineales el desempeño de una red neuronal profunda (de varias capas ocultas) se puede comparar con el de una red de una sola capa, lo cual resulta inconveniente sobre todo si se está trabajando con datos que no tengan un comportamiento lineal (lo cual es usual).
Cuatro de las funciones de activación más popularmente usadas están dadas por las siguientes ecuaciones:

$$f(x) = \begin{cases} -1 & \text{si} \quad x < 0 \\ 1 & si \quad x \geq 0 \end{cases} \qquad \text{función Paso,} \tag{6.3}$$

$$g(x) = \frac{1}{1+\exp(-x)} \qquad \text{función Sigmoide,} \tag{6.4}$$

$$h(x) = \tanh(x) \qquad \text{función Tangente hiperbólico} \tag{6.5}$$

$$k(x) = \max(0, x) \qquad \text{función ReLU.} \tag{6.6}$$

Inicialmente se trabajaba la función de Paso, sin embargo debido a su comportamiento "plano" el Gradiente Descendiente no tiene a dónde dirigirse. Esta función fue reemplazada por el Sigmoide que tiene una derivada bien definida no nula de tal manera que el Gradiente Descendiente pueda tener algún proceso.
Para el caso de la función Tangente hiperbólico, es continua, derivable y su rango varía entre -1 y 1, lo que la hace diferente al Sigmoide. Dicho rango hace que el resultado en cada capa se centre alrededor de 0, lo cual puede aumentar la velocidad de convergencia.
La función ReLU (Rectified Linear Unit por sus siglas en inglés), es continua pero no diferenciable en $x = 0$, su derivada es 1 para $x > 0$ y 0 para $x < 0$. Sin embargo, en la práctica funciona bien y lo más importante es el hecho que no tiene resultado máximo lo cual ayuda a que el paso del Gradiente descendiente se maneje sin obstáculos como el estancamiento. En la implementación que hacemos de nuestro modelo, usamos esta función de activación debido a sus características.

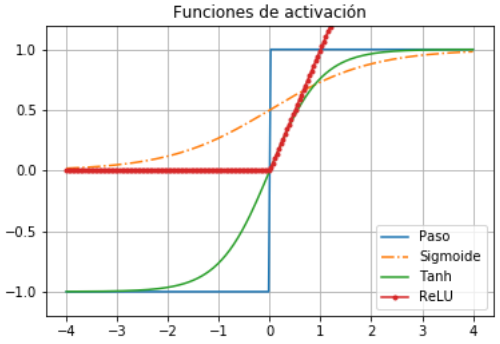


Figura 6.3: Cuatro funciones de acrivación comunmente usadas.

6.4 Algoritmos de Optimización Rápida

Un aspecto muy importante es la velocidad en el proceso de optimización durante el algoritmo de propagación hacia atrás (hay más aspectos, como la inicialización de los pesos en las conexiones, función de activación en contexto y normalización por lotes sin embargo, se hará énfasis en la asignación de pesos).
Los algoritmos optimizadores actualizan los pesos para minimizar la función de pérdida. La función de pérdida[4] J, indica al optimizador si se está moviendo en la dirección correcta para llegar al mínimo global. Cómo ya hemos visto en la sección anterior, la regla de actualización de los pesos es parecida a la que aparece en el proceso de aprendizaje de un perceptrón: la actualización de los pesos se obtiene por $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\theta} J(\boldsymbol{\theta})$, sin importar cuáles eran los valores anteriores de los gradientes.

Optimización de Momentum

La optimización Momentum tiene en cuenta lo que fueron los gradientes previos: en cada iteración, resta el gradiente local del vector *momentum* $\mathbf{m}$ (multiplicado por la tasa de aprendizaje) y actualiza los pesos simplemente sumando el vector momentum. En pocas palabras, el gradiente es usado como aceleración, no como velocidad. La idea es simular una clase de mecanismo de impulso que sea afectado por fricción, con lo que se introduce un nuevo hiperparámetro β, el cual debe tener un valor entre 0 (alta fricción) y 1 (sin fricción). Un valor típico del momentum es 0.9.
Algoritmo Momentum

1. $\mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$

Si el gradiente se mantiene constante, la velocidad terminal o el tamaño de los pesos actualizados será igual al gradiente multiplicado por la tasa de aprendizaje η y $\frac{1}{1-\beta}$ (ignorando el signo). Por ejemplo, si $\beta = 0.9$, entonces la velocidad terminal es igual a 10 veces el gradiente por la tasa de aprendizaje, así que la optimización de Momentum termina siendo 10 veces más rápida que el Gradiente Descendiente.
El inconveniente con la optimización de Momentum es que suma otro hiperparámetro. Sin embargo, el valor del momentum de 0.9 usualmente funciona bien en la práctica y casi siempre es más rápida que el Gradiente Descendiente.

Gradiente Acelerado de Nesterov

En 1983 Yurii Nesterov propuso una variante de la optimización de Momentum. La idea de la optimización de Nesterov fue la de medir el gradiente de la función de costo no en la posición local si no en la dirección del momento.

[4]Un ejemplo de función de pérdida la tenemos en (6.2). Hay varios tipos de función de pérdida que pueden ser consultados en [8] o [1].

Algoritmo NAG (Nesterov Accelerated Gradient)

1. $\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta} + \beta\mathbf{m})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$

Esta pequeña modificación funciona en el sentido que el vector momentum estará apuntando en la dirección correcta, hacia el óptimo, esto ayudará a reducir las oscilaciones y converger más rápido.

AdaGrad

En el proceso de optimización del gradiente, durante cada iteración, el gradiente no apunta directamente hacia el óptimo global y esto hace dicho proceso muy lento, el algoritmo *AdaGrad* corrige su dirección en un punto inicial un poco más apuntando al óptimo; el algoritmo escala el vector gradiente a lo largo de las direcciones más empinadas. En el siguiente algoritmo, la operación binaria entre vectores con el símbolo $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$ significa un nuevo vector $\mathbf{v}^\top = (v_1, ..., v_k)^\top$ donde $v_i = \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_i}\right)^2$ para $i = 1, ..., k$; la operación $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \epsilon}$ es un nuevo vector $\mathbf{w}^\top = (w_1, ..., w_k)$ donde $w_i = \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_i \sqrt{s_i + \epsilon}}$ para $i = 1, ..., k$.

Algoritmo de AdaGrad

1. $\mathbf{s} \leftarrow \mathbf{s} + \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \epsilon}$

En el primer paso se asignan componente a componente las derivadas parciales de la función de costo con respecto a cada una de las componentes de $\boldsymbol{\theta}$ a un vector $\mathbf{s}$. Si la función de costo es más pronunciada a lo largo de la $i-$ésima dimensión, entonces la componente $i-$ésima del vector $\mathbf{s}$ será más grande en cada iteración.
El segundo paso es casi idéntico al del Gradiente Descendiente, pero con una gran diferencia: el vector gradiente es multiplicado por un factor $\sqrt{\mathbf{s} + \epsilon}$ que lo disminuye, donde ϵ es un número muy pequeño que evita la división por cero, usualmente es el número 10^{-10}.
Debido a que para problemas cuadráticos sencillos AdaGrad funciona muy bien, no sucede lo mismo para otro tipo de problema como es el entrenamiento de las redes neuronales, el algoritmo suele terminar demasiado pronto (no alcanza el global) porque la tasa de aprendizaje se vuelve muy pequeña impidiendo la optimización completa.

RMSProp

El algoritmo *RMSProp* es una modificación del anterior AdaGrad, la idea es trabajar con los gradientes más recientes (lo que es opuesto a trabajar con todos los gradientes desde el principio del entrenamiento). Por esta razón, en el primer paso, se usa decaimiento exponencial.
Algoritmo RMSProp

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1 - \beta)\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s} + \epsilon}$

La tasa de decaimiento β se toma como el valor 0.9 por defecto. Aunque es un nuevo hiperparámetro, este valor predeterminado a menudo funciona bien, por lo que es posible que no necesite ajustarlo en absoluto.

Adam

Adam significa *Adaptive momentum estimation* por sus siglas en inglés. Este algoritmo es una combinación de las ideas de optimización de momentum y RMSProp, conlleva un promedio de dacaimiento exponencial de los gradientes pasados. El algoritmo se describe a continuación.
Algoritmo Adam

1. $\mathbf{m} \leftarrow \beta_1\mathbf{m} - (1 - \beta_1)\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
2. $\mathbf{s} \leftarrow \beta_2\mathbf{s} + (1 - \beta_2)\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$

3. $\hat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1-\beta_1^k}$, donde k es la $k-$ésima iteración.

4. $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\beta_2^k}$, donde k es la $k-$ésima iteración.

5. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \hat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \epsilon}$

Los pasos 1,2 y 5 hacen ver la relación con los algoritmos de momentum y RMSProp, la diferencia es que en el paso 1 se calcula un promedio decaimientos exponenciales en lugar de una suma de decaimientos exponenciales, pero de hecho estos son equivalentes excepto por un factor constante (el promedio de decaimientos exponenciales es solo $1 - \beta_1$ por la suma de los decaimientos exponenciales). Los pasos 3 y 4 son detalles técnicos, puesto que $\mathbf{m}$ y $\mathbf{s}$ se inicializan en 0, ellos se sesgan al inicio del entrenamiento hacia 0, luego estos dos pasos ayudarán a impulsar a $\mathbf{m}$ y $\mathbf{s}$ al principio del entrenamiento.
Los hiperparámetros β_1 y β_2 tendrán valores por defecto de 0.9 y 0.999. El término ϵ es usualmente inicializado con el valor 10^{-7}. Estos valores están establecidos por defecto en la clase de Adam programada en la librería *Keras* implementada en el lenguaje de programación *Python*.

6.5 Implementación

En esta sección se explica en detalle el procedimiento fundamentado en el método Monte Carlo por secciones en un data-frame (un conjunto de variables y observaciones) para hacer que una red neuronal profunda de dos capas ocultas, pueda predecir argumento máximo de la función de valor. Debido al impacto actual del *Deep Learning*, mostramos los códigos que implementamos en este capítulo, con el objetivo que el lector pueda fijarse en cada detalle en la programación. Además porque, hasta donde llega nuestro conocimiento, nadie ha hecho este entrenamiento.
Comenzamos importando librerías necesarias: *Numpy* y *Pandas* principalmente para las simulaciones y para la manipulación de datos en el data-frame (o diagrama de datos).
Se programa una función "BM" cuyos parámetros son N un número entero correspondiente a el número de caminos (u observaciones en el data-frame) representando movimientos Brownianos; n es el paso o lo que mide Δt, y T el tiempo final. Observemos que se establece una semilla para poder contar con los mismos datos.

```
import numpy as np
import pandas as pd
from math import sqrt

def BM(N,n,T):
    np.random.seed(123)
    dt = float(T/n)
    B = np.zeros((n+1,N))
    for i in range(1,n+1):
        z = np.random.standard_normal(N)
        B[i] = B[i-1] + z*sqrt(dt)
    return B
```

Establecemos los parámetros que hemos venido manejando $T = 1.0$, $r = 0.05$, $\gamma = 0.5$, $\mu = 0.11$, $\sigma = 0.4$,$n = 100$ $x_0 = 1.0$, variaremos el valor de N que corresponde al número de movimientos Brownianos.
Con los siguientes comandos generamos nuestro data-frame, el cual contiene como variables los valores obtenidos de la simulación de los movimientos Brownianos en el tiempo t_i para $i = 0, ..., n$.

```
W = BM(N,n,T)
enc = []                                    # Encabezado
for i in range(n+1):
    enc.append('$W_{t_{%d}}$'%i)
df_1 = pd.data-frame(np.transpose(W))
df_1.columns = enc
```

Lo que nos muestra el siguiente data-frame:

W_{t_0}	W_{t_1}	W_{t_2}	W_{t_3}	W_{t_4}	W_{t_5}	W_{t_6}	W_{t_7}	W_{t_8}	W_{t_9}	...	$W_{t_{91}}$	$W_{t_{92}}$	$W_{t_{93}}$	$W_{t_{94}}$	$W_{t_{95}}$	$W_{t_{96}}$	
0.0	-0.108563	-0.075533	-0.033299	-0.037716	-0.069210	0.104476	0.036808	-0.081645	-0.113967	...	0.997725	0.970239	0.849593	0.812449	0.907126	0.798271	
0.0	0.099735	-0.042500	-0.089556	-0.104541	-0.060618	-0.125738	-0.194119	-0.300527	-0.212463	...	-0.411130	-0.510574	-0.362963	-0.401947	-0.532986	-0.602357	-
0.0	0.028298	-0.046202	0.069230	0.231266	0.301019	0.327424	0.366146	0.330047	0.359494	...	1.519247	1.518074	1.375111	1.314485	1.264598	1.249464	
0.0	-0.150629	0.026726	-0.287333	-0.419752	-0.436515	-0.432726	-0.466927	-0.511320	-0.610743	...	-0.115751	0.005885	0.115900	0.238274	0.230083	0.218905	
0.0	-0.057860	0.022735	0.152542	0.129721	-0.179352	-0.183161	-0.077374	0.065835	0.006490	...	-0.722714	-0.949190	-0.945608	-1.107581	-1.045723	-1.180265	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
0.0	0.006954	0.205306	0.146904	0.104374	0.004672	0.077165	0.247129	0.346359	0.397645	...	-0.156745	-0.158160	0.026587	-0.074343	-0.031846	0.090379	-
0.0	-0.056538	0.002518	0.094663	0.043026	0.127006	0.319741	0.216855	0.322904	0.382619	...	-0.234890	-0.212085	-0.077706	-0.022000	-0.004887	-0.020390	-
0.0	0.035746	-0.113289	-0.001642	0.092534	0.007766	0.134402	0.216181	0.435845	0.504958	...	0.868269	0.767838	0.867518	0.776454	0.796699	0.785568	
0.0	-0.036686	-0.082773	-0.170693	-0.228626	-0.302445	-0.347251	-0.426097	-0.297545	-0.154463	...	-1.137148	-1.298767	-1.423326	-1.382970	-1.468863	-1.573428	-
0.0	0.007193	-0.067560	-0.166608	-0.240534	-0.369811	-0.217334	-0.233751	-0.234849	-0.408897	...	-1.597019	-1.615308	-1.622035	-1.695140	-1.710735	-1.829486	-

Figura 6.4: Data-frame obtenido por el procedimiento

A continuación vamos a combinar el método Monte Carlo y el Golden Section Search (ya usados en una sección anterior), para calcular el argumento óptimo. Para esto, se programan tres rutinas de funciones: para el proceso de optimización usamos la función "GS" con las iniciales de Golden Search; otra con el nombre "final-mean-wealth" cuyos argumentos son B un bloque de data-frame, μ, σ, r, γ y n; esta función usa el método de Euler sobre las filas del data-frame que representan los movimientos Brownianos para la simulación del proceso de riqueza y posteriormente se calcula el promedio final como resultado. Una última función llamada "my func", es una función auxiliar basada en final-mean-wealth que solo depende del parámetro pi sobre el cual debemos hacer la optimización.

```
def GS(f,a,b,tol):
    from math import sqrt
    ai,bi, invphi = a,b,(sqrt(5)-1)/2
    while(abs(bi-ai)>tol):
        ci = bi - (bi-ai)*invphi
        di = ai + (bi-ai)*invphi
        if(f(ci)>f(di)):
            bi = di
        else:
            ai = ci
    return [f((ai+bi)/2),(ai+bi)/2]

def final_mean_wealth(B,mu,sigma,gamma,pi,n):
    dt = float(1/n)
    v = np.zeros((len(B),n+1))
    v[:,0] = x0
    for j in range(1,n+1):
        v[:,j] = v[:,j-1]*np.exp((r+(mu-r)*pi-0.5*sigma**2*pi**2)*dt + sigma*pi*(B.iloc
            [:,j]-B.iloc[:,j-1]))
    v[:,-1] = v[:,-1]**gamma/gamma
    return np.mean(v[:,-1])

def my_func(pi):
    return final_mean_wealth(B,mu,sigma,gamma,pi,n)
```

Luego dividimos el data-frame por segmentos de filas y llevamos a cabo la optimización sobre cada bloque, a dicho bloque le asignamos el pi óptimo obtenido con las rutinas mencionadas arriba. Posteriormente, re-ordenamos las filas del data-frame de tal manera que se pueda ver variación de los argumentos obtenidos. Además establecemos columnas (variables) con los parámetros ya usados.

```
seg = 2000        #Longitud del segmento del data-frame
m = int(N/seg)
pi_star = np.zeros(N)
for i in range(m):
    B = df_1.iloc[i*seg:(i+1)*seg]
    pi = GS(my_func,0.0,1.0,1e-2)[1]
    pi_star[i*seg:(i+1)*seg] = pi
df_1['$\pi^*$'] = pi_star
```

```
df_1[['$r$','$\mu$','$\sigma$']] = pd.data-frame({
    'r': r*np.ones(N),
     '$\mu$': mu*np.ones(N),
     '$\sigma$': sigma*np.ones(N)
})

df_2 = df_1.copy().sample(frac=1,random_state=123)
df_2.reset_index(drop=True)
```

Con dicho código llegamos a un data-frame con la siguiente apariencia.

W_{t_3}	W_{t_4}	W_{t_5}	W_{t_6}	W_{t_7}	W_{t_8}	W_{t_9}	...	$W_{t_{95}}$	$W_{t_{96}}$	$W_{t_{97}}$	$W_{t_{98}}$	$W_{t_{99}}$	$W_{t_{100}}$	π^*	r	μ	σ
0.151835	0.266412	0.269462	0.365284	0.217149	0.083438	0.104789	...	0.847252	0.821301	0.821443	0.751256	0.685117	0.676101	0.730450	0.05	0.11	0.4
-0.034786	-0.101891	-0.118629	-0.117366	-0.074594	0.111054	-0.168899	...	-0.731775	-0.676680	-0.533079	-0.578630	-0.647717	-0.715926	0.696008	0.05	0.11	0.4
-0.054253	-0.225427	-0.251698	-0.137643	-0.186252	-0.205949	-0.233463	...	-1.200792	-1.217306	-1.112867	-1.004418	-1.054988	-1.157760	0.664672	0.05	0.11	0.4
0.176744	0.100325	0.122095	0.087755	0.112924	0.143276	-0.044898	...	0.120800	-0.052620	0.020860	0.082550	-0.000662	0.000419	0.802439	0.05	0.11	0.4
-0.013409	-0.005954	0.091380	0.111758	0.305426	0.350352	0.355667	...	0.801557	0.807606	0.795157	0.775024	0.666173	0.500771	0.561346	0.05	0.11	0.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
-0.191348	-0.196414	-0.122923	-0.251689	-0.274513	-0.433596	-0.564738	...	-0.964581	-1.067500	-1.159355	-0.871565	-0.751827	-0.844227	0.664672	0.05	0.11	0.4
0.142522	0.155215	0.040625	-0.009601	0.010881	-0.042458	-0.101087	...	-0.224429	-0.408748	-0.455467	-0.531812	-0.472188	-0.446105	0.751736	0.05	0.11	0.4
-0.117242	-0.169911	-0.198481	-0.312809	-0.258846	-0.164745	-0.107768	...	-1.792435	-1.863132	-1.999092	-2.200961	-2.357171	-2.428109	0.717294	0.05	0.11	0.4
0.120281	0.062706	0.105561	0.081766	0.243390	0.305620	0.207686	...	-0.088273	-0.042085	-0.013679	-0.087839	-0.002560	0.034874	0.773022	0.05	0.11	0.4
0.228654	0.219117	0.155393	0.064508	0.033690	-0.016693	-0.094220	...	-0.280907	-0.274969	-0.271703	-0.271489	-0.264259	-0.220646	0.579527	0.05	0.11	0.4

Figura 6.5: Data-frame final para el entrenamiento de la red neuronal.

Obsérvese que el valor el data-frame de la figura 6.5 contiene las filas distintas al de la figura 6.5, ya que se re-organizaron las observaciones.
Es importante notar que se hacen las subdivisiones en bloques para hacer la optimización Monte-Carlo, esto debido a que $\mathbb{E}[f(X)] \neq f(\mathbb{E}[X])$, por lo que no es necesariamente igual el valor esperado de un máximo a un máximo de un valor esperado.

6.6 Entrenamiento de la Red Neuronal

En esta sección explicaremos los comandos usados para el entrenamiento de la red neuronal. Comenzamos con la asignación de los inputs a una variable X y el output a una variable Y, recordemos que este problema es de regresión, lo que nos llevará a resultados aproximados numéricamente al valor obtenido analíticamente (2.27).
Hacemos la división de los datos simulados en un conjunto de entrenamiento para X y Y, y un conjunto de validación para poder probar qué tan acertada ha sido la predicción de la red neuronal que proponemos. Para esta tarea importamos las librerías necesarias que nos ayudan en dicho objetivo.

```
X = df_2.drop(['$\pi^*$'],axis=1)
Y = df_2[['$\pi^*$']]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,random_state=123)
x_train, x_val, y_train, y_val = train_test_split(x_train,y_train,test_size=0.1,
    random_state=123)
```

Posteriormente, procedemos a importar de la librería *Keras*, las funciones básicas para el entrenamiento de la red neuronal. Invocamos en la variable "model" el tipo de orden en el algoritmo de aprendizaje de la red neuronal, en este caso es Sequential. Luego construimos la red neuronal con dos capas ocultas, cada una con la misma función de activación relu, el número de inputs corresponde a 104 (101 de los pasos del movimiento Browniano más 3 parámetros ya establecidos r, mu y sigma); cada capa oculta contiene 200 neuronas. Después, escogemos el optimizador adam para la función objetivo de los errores medios cuadráticos; y compilamos con un número de épocas de 20 y un tamaño de lote de 50.

```
model = Sequential()
model.add(Dense(200,input_dim = 104,activation='relu'))
```

```
model.add(Dense(200,activation='relu'))
model.add(Dense(1,activation = "relu"))
model.compile(optimizer='adam',loss="mean_absolute_error",metrics=["mean_squared_error"])
model.fit(x_train.values,y_train.values, validation_data=(x_val,y_val),epochs=20,
    batch_size=50)
```

Las predicciones junto con los valores de los argumentos óptimos se muestran en el siguiente gráfico.

	$\hat{\pi}$	Predicted
5444	0.87625	0.752604
5500	0.87625	0.774880
9054	0.74875	0.795687
567	0.74875	0.777444
146	0.74875	0.787779
...	...	...
6206	0.74875	0.780456
8358	0.74875	0.772447
9611	0.74875	0.788819
8481	0.74875	0.792371
7974	0.74875	0.782260

El promedio de $\hat{\pi}$ es de 0.77425 y el de Predicted es 0.77452.

Algunos Resultados Gráficos

En esta sección presentamos resultados de acuerdo a los procedimientos explicados anteriormente. El mismo procedimiento para predecir el valor de $\hat{\pi}$ el cual fue obtenido por el procedimiento de optimización, fue usado de igual manera para hacer predicciones sobre la función de valor $\tilde{V}$ obtenida en el proceso de optimización.

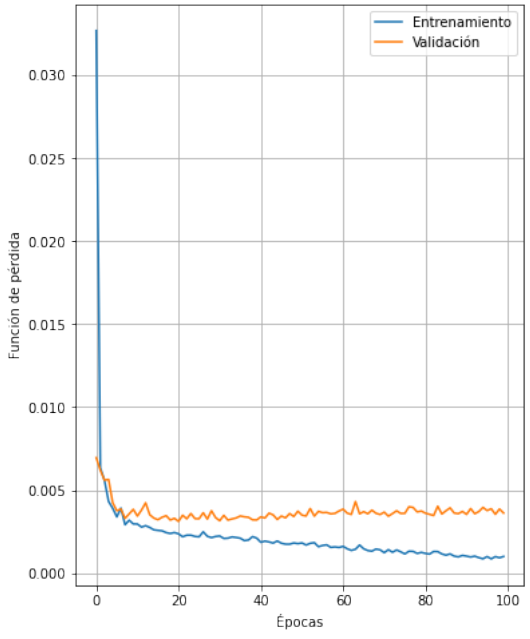


Figura 6.6: Evolución de la función de pérdida por épocas en la predicción de $\hat{\pi}$

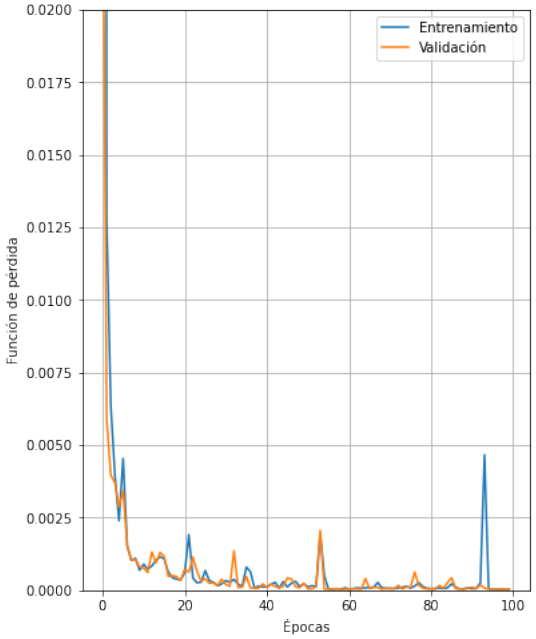


Figura 6.7: Evolución de la función de pérdida por épocas en la predicción de $\tilde{V}$

En la figura 6.6 se muestran los resultados del aprendizaje y la precisión con la cual la red neuronal predijo el valor del portafolio óptimo usando como output de referencia el valor $\hat{\pi}$ através de un número

de épocas de 100. Las predicciones obtenidas muestran que a medida que aumenta el número de épocas, la función de pérdida cae para los datos de entrenamiento y se mantiene en una franja pequeña para los datos de validación. Además, el tiempo de ejecución del modelo es de 33.5 segundos. El error porcentual en la predicción promedio de $\hat{\pi}$ (el cual es el conseguido por el método de optimización) es 0.84 %, mientras la predicción promedio de $\hat{V}(1,0)$ (igualmente conseguida por el método de optimización) tiene un error porcentual de 0.03 % aproximadamente.
Lo anteriormente mostrado se realizó para una muestra de 10.000 caminos. En la siguiente gráfica observamos un comportamiento típico del error porcentual entre la predicción y el portafolio óptimo de Merton, a medida que aumentamos el número de simulaciones N:

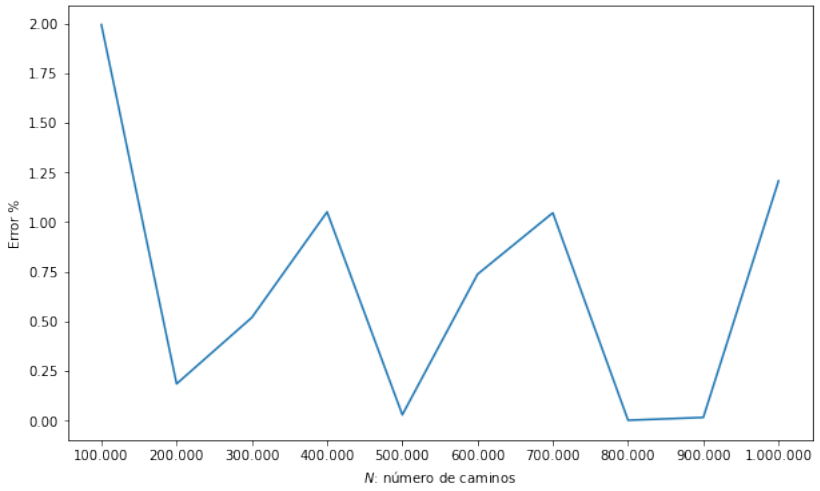


Figura 6.8: Errores porcentuales para la red neuronal hasta con 10^6 caminos simulados.

Tomando una semilla para fijar la predicción con el comando *tensorflow.random.set_seed(2)*, encontramos que en la Figura 6.8, las mejores aproximaciones se encuentran al tomar $N = 500.000$ o $N = 800.000$ o $N = 900.000$. Podemos suponer que la falta de convergencia en el método, es debido a que la asignación inicial de pesos podría causar algún tipo de ruido, lo cual haría que eventualmente no se establezca una aproximación deseada en la predicción. Adicionalmente, como se nota en la Figura 6.6, la predicción para los datos de validación parece estabilizarse en un valor cercano a cero, sin llegar a cero, lo cual supone impedir convergencia, sin embargo el error porcentual se mueve en una franja con un grosor no mayor al 2%.

Capítulo 7

Conclusiones

En este trabajo se presentó la formulación analítica del problema y solución de selección de portafolio de Merton para el caso de un activo riesgoso y otro libre de riesgo. Se quería ver varios enfoques computacionales y algoritmos que aproximaban la función de valor y el argumento que la maximizaba. Através del documento y los temas considerados se pueden establecer las siguientes observaciones y conclusiones.

1. Para la aproximación de la función de valor mediante cadenas de Markov, se replicaron algunos resultados obtenidos en el artículo de [11]. Nuestra contribución fue considerar un esquema de diferencia central simétrica (3.5), la cual permitió conseguir aproximaciones notables comparándolas con los resultados replicados. Es preciso afirmar también que aunque las aproximaciones conseguidas por el método simétrico son buenas en cuanto a la predicción del argumento óptimo π^* al igual que las sugeridas por el artículo de Kafash, los errores aumentan en un rango estrecho.

2. Para el método Monte Carlo se consideraron dos métodos de optimización: Exhausión y Golden Section Search. Aquí observamos que los métodos se aproximan muy bien tanto a los valores de la función de valor óptimo (2.25) como a su argumento maximizador (2.29). Sin embargo Golden Search produce un resultado muy próximo a los valores verdaderos en menos iteraciones, lo cual hace de este método más eficiente que el exhaustivo.

3. En el tema de diferencias finitas se consideró un esquema implícito para la ecuación (5.2). Para la optimización usamos los métodos de Golden Section Search y uno al que llamamos "Fibonacci". Los resultados muestran que el esquema implícito no muestra mayor variación cuando se modifica el número de pasos en el eje t, cosa que sí sucede al aumentar los pasos en el eje x. El método que da mayor aproximación en el argumento óptimo es el de Fibonacci, y tiene aproximaciones parecidas a los valores de V que el algoritmo Golden Section Search.

4. En redes neuronales, se consideraron métodos muy específicos para la optimización de los errores en la predicción de una red neuronal establecida con 2 capas invisibles y más de 300 neuronas. Como algoritmo de optimización se eligió el de Fibonacci. Se propuso un método Monte Carlo para el cálculo del argumento óptimo por segmentos de los data frame programados. La red muestra en el aprendizaje una buena predicción del argumento óptimo $\hat{\pi}$ a medida que aumenta el número de épocas de entrenamiento; los resultados numéricos muestran una estabilidad en la función de pérdida en la predicción de los datos de validación, mientras que en los datos de entrenamiento, decrece. Notamos que insertando una semilla en la librería de *tensorflow* podríamos examinar el comportamiento de la predicción de la red neuronal en contra del número de caminos de riqueza simulads, observando que el error porcentual se encuentra en una franja con un grosor no mayor al 2%; además se vemos que la aproximación no parece mejorar con el aumento del número de caminos esto por la asignación de pesos inicial de la red y a la estabilidad en el proceso de aprendizaje.

5. En la siguiente tabla resumimos los mejores resultados obtenidos en cuanto a errores porcentuales[1]:

[1]Es importante notar que para el método de redes neuronales, se implementó una red que aprendió de los resultados promedios obtenidos por la rutina de optimización usada, no necesariamente aprendió de la solución analítica.

Método	Error % π^*	Error % $V(1,0)$	Tiempo de ejecución (segundos)
C. de Markov, upwind	2.5	0.021	233
C de Markov, simétrico	3.9	0.00025	272.5
M. Carlo G.S.S.	2.4	0.06	63
D. Finitas Fibonacci	0.0083	0.001	374
Redes Neuronales	0.84	0.03	4 de ejecución y 34 de aprendizaje

Tabla 7.1: Tabla de los mejores resultados obtenidos en este trabajo.

Notamos en la tabla que si se desea obtener una muy buena aproximación, podremos usar el método de diferencias finitas con algoritmo de optimización el método de Fibonacci, sin embargo, es la rutina con más tiempo de ejecución. Una ventaja del método de diferencias finitas sobre el de aproximación por cadenas de Markov, es que, por ser el esquema implícito, se pueden elegir los parámetros Δx y Δt positivos arbitrarios obteniendo un resultado estable; cosa que no siempre es posible con las cadenas de Markov, ya que deben elegirse de manera que queden bien definidas las probabilidades de transición. Esto hace que el método de las cadenas de Markov, sea más rígido en comparación con las diferencias finitas.
Cabe mencionar que por la naturaleza del método Monte Carlo, se espera que sea el método con más tiempo de ejecución, pero en esta oportunidad el método de diferencias finitas es el más demorado; no obstante, el error absoluto producido por el método Monte Carlo para aproximar tanto a $V(1,0)$ como a π^* no es tan bueno como el de diferencias finitas; por tanto, para obtener una precisión parecida o mejor al método de diferencias finitas, se espera un mayor tiempo de ejecución para el método de Monte Carlo.

7.1 Trabajo Posterior

Nuestra idea es adoptar el enfoque de red neuronal y adaptarla a la predicción de portafolios óptimos calibrados a datos empíricos para distinguir cuál reacciona mejor a posibles incumplimientos en los supuestos del modelo.

Bibliografía

[1] Aggarwal C.(2018), *Neural Networks and Deep Learning*. Springer. Yorktown Heights, New York.

[2] Bellman R., (1957),*Dynamic Programming*. Princeton University Press. Princeton, New Jersey.

[3] Björk T. (2009), *Arbitrage Theory in Continuos Time 3th. Ed.*. OXFORD UNIVERSITY PRESS. Stockholm School of Economics.

[4] Dai M., Zhong Yifei (2008), *Penalty Methods for Continuos-Time Portfolio Selection with Proportional Transactions Costs*. Disponible en SSRN: https://ssrn.com/abstract=1210105 o http://dx.doi.org/10.2139/ssrn.1210105

[5] Desai R., Lele T., Viens F, (2003), *A Monte Carlo method for portfolio optimization under partially observed stochastic volatility*. IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, Proceedings (CIFEr). pág. 257-263.

[6] Detemple J., García R., Rindisbacher M. (2003), *A Monte Carlo Method for Optimal Portfolios*. The Journal of Finance Vol. 58 pág. 401-446. Wiley for the American Finance Association.

[7] Freitas F., De Souza A., De Almeida A.(2009), *Prediction-based portfolio optimization model using neural networks*. Neurocomputación, Volumen 72, números 10–12, pág. 2155-2170.

[8] Gerón A, (2019), *Hands On Machine Learning with Scikit-Learn, Keras, TensorFlow*, O'Reilly Media Inc, Canada.

[9] Giordano F., Fox W., Horton S.,2014, *A First Course in Mathematical Modeling*, CENGAGE Learning.

[10] Hirsa A. (2013) *Computational methods in Finance* CRC Press Taylor & Francis Group.

[11] Kafash B., (2019),*Approximating the Solution of Stochastic Optimal Control Problems and the Merton's Portfolio Selection Model* [Springer: Computational Economics (2019) 54:763–782].

[12] Kingma D,Lei J., (2014),*Adam: A Method for Stochastic Optimization*, published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[13] Korn R., Korn E, (2010), *Option Pricing and Portfolio Optimization* Amercan Mathematical Society. Graduate Studies in Mathematics Vol.31. Stelzenberg, Germany.

[14] Korn R., Korn E,. Kroisandt G., (2010), *Monte Carlo methods and models in finance and insurance* CRC Press Taylor & Francis Group.

[15] Krawczyk J. (2001) *A Markovian approximated solution to a portfolio management problem*. Information Technology and Management.

[16] Kushner H, Dupois P.(2001)*Numerical Methods for Stochastic Control Problems in Continuous Time* New York. Springer.

[17] Merton R., (1971), *Optimum Consumption and Portfolio Rules in a Continuous-Time Model*,J. Econom. Theory 3, 373-413.

[18] Moolayil J., (2019), *Learn Keras for Deep Neural Networks*. Vancouver, Canadá. APRESS.

[19] Pham H (2009), *Continuous-time Stochastic Control and Optimization with Financial Applications*. New York. Springer

[20] Pham H (2010), *Stochastic control and applications in finance*. University of Paris Diderot, LPMA, Paris.

[21] Ramirez H. (2017), *Numerical methods in finance course 2017*. Universidad del Rosario, Bogotá.

[22] Steiner M., Wittkemper H., *Portfolio optimization with a neural network implementation of the coherent market hypothesis*, European Journal of Operational Research, Vol. 100, issue 1, pág. 27-40.

[23] Yin G., Jin H., & Jin, Z (2009), *Numerical methods for portfolio selection with bounded constraints*. Journal of Computational and Applied Mathematics, 233(2), 564-581.

Apéndice A

Expansiones de Taylor y aproximaciones de derivadas parciales

Para la discretización de las derivadas parciales en una PDE se requiere de una herramienta frecuentemente usada, la expansión de Taylor de la solución. En esta sección clasificamos algunas formas de aproximar derivadas. Además, en cada derivación indicamos el orden del error que corresponde a cada ejemplo. Supongamos que f es una función en una variable determinística x y que tiene $k+1$ derivadas continuas, $f \in C^{k+1}$, entonces la expansión de Taylor de f centrada en $x = a$ se define como

$$f(x) = \sum_{j=0}^{k} \frac{f^{(j)}(a)}{j!}(x-a)^j + \frac{f^{(k+1)}(\xi)}{(k+1)!}(x-a)^{k+1}$$

para algún $\xi \in (x, a)$. El último término se llama el *error* y lo escribimos en notación de Landau $O((x-a)^{k+1})$. Acontinuación mostramos las aproximaciones que son frecuentemente usadas para las derivadas.

- **Aproximación hacia adelante y hacia atrás de la primera derivada.** Asumamos que $f \in C^2$, y vamos a aproximar $f^(1)(x)$. Sea $\Delta x > 0$, escribimos las expansiones de Taylor truncadas hacia adelante y hacia atrás como sigue

$$f(x+\Delta x) = f(x) + \Delta x f^{(1)}(x) + \frac{\Delta x^2}{2!}(\xi_1), \tag{A.1}$$

$$f(x-\Delta x) = f(x) - \Delta x f^{(1)}(x) + \frac{\Delta x^2}{2!}(\xi_2). \tag{A.2}$$

  Despejando $f^{(1)}(x)$ en (A.1), obtenemos la aproximación *hacia adelante* de la primera derivada

$$\begin{aligned} f^{(1)}(x) &= \frac{f(x+\Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2} f^{(2)}(\xi_1) \\ &\approx \frac{f(x+\Delta x) - f(x)}{\Delta x} \text{ con } O(\Delta x). \end{aligned}$$

  Similarmente, al despejar $f^{(1)}(x)$ de (A.2) obtenemos la aproximación *hacia atrás* de la primera derivada.

$$\begin{aligned} f^{(1)}(x) &= \frac{f(x) - f(x-\Delta x)}{\Delta x} + \frac{\Delta x}{2} f^{(2)}(\xi_2) \\ &\approx \frac{f(x) - f(x-\Delta x)}{\Delta x} \text{ con } O(\Delta x). \end{aligned}$$

- **Aproximación central simétrica de la primera derivada.** Asumiendo que $f \in C^3$, aproximamos $f^{(1)}(x)$ con un orden de error $O(\Delta x^2)$. Usamos la expansión de Taylor para f:

$$f(x+\Delta x) = f(x) + \Delta x f^{(1)}(x) + \frac{\Delta x^2}{2!} f^{(2)}(x) + \frac{\Delta x^3}{3!} f^{(3)}(\xi_3) \tag{A.3}$$

$$f(x-\Delta x) = f(x) - \Delta x f^{(1)}(x) + \frac{\Delta x^2}{2!} f^{(2)}(x) - \frac{\Delta x^3}{3!} f^{(3)}(\xi_4) \tag{A.4}$$

Realizamos la resta de (A.4) y (A.3) obteniendo

$$\begin{aligned} f(x+\Delta x) - f(x-\Delta x) &= 2\Delta x f^{(1)}(x) + \frac{\Delta x^3}{3!} f^{(3)}(\xi_3) + \frac{\Delta x^3}{3!} f^{(3)}(\xi_4) \\ &= 2\Delta x f^{(1)}(x) + O(\Delta x^3). \end{aligned}$$

De esta manera obtenemos la *aproximación central simétrica* de la segunda derivada de f:

$$\begin{aligned} f^{(1)}(x) &= \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + O(\Delta x^2) \\ &\approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \text{ con } O(\Delta x^2). \end{aligned}$$

- **Aproximación central de la segunda derivada** Asumiendo que $f \in C^4$, vamos a aproximar a $f^{(2)}(x)$. Usamos las expansiones de Taylor siguientes:

$$f(x+\Delta x) = f(x) + \Delta x f^{(1)}(x) + \frac{\Delta x^2}{2!} f^{(2)}(x) + \frac{\Delta x^3}{3!} f^{(3)}(x) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_5) \quad (A.5)$$

$$f(x-\Delta x) = f(x) - \Delta x f^{(1)}(x) + \frac{\Delta x^2}{2!} f^{(2)}(x) - \frac{\Delta x^3}{3!} f^{(3)}(x) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_6) \quad (A.6)$$

Sumando las expresiones (A.5) y (A.6), tenemos

$$f(x+\Delta x) + f(x-\Delta x) = 2f(x) + 2\frac{\Delta x^2}{2!} f^{(2)}(x) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_5) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_6).$$

Resolviendo para $f^{(2)}(x)$ encontramos la expresión para la aproximación *diferencia centrada* de la segunda derivada:

$$f^{(2)}(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} + O(\Delta x^2).$$

Apéndice B

Anexos

En esta sección se adjuntan los códigos empleados por cada método explicado.

B.1 Código de la aproximación de cadenas de Markov

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

#Parametros empleados en el paper de Kafash 2016
M = 10000
delta = float(1/M)
r = 0.05
b = 0.11
sigma = 0.4 # corregido
gamma = 0.5
T = 1.0
pi_ast = (b-r)/(sigma**2*(1-gamma))
method_1 = 'upwind'
method_2 = 'simetric'

def pi(V,x,N,b,r,sigma,h):
    '''
    Esta funcion calcula el $\pi$, donde $V_x$ y $V_{xx}$
    se aproximan mediante diferencias finitas.
    '''
    c = (r-b)/sigma**2
    res = np.zeros(2*N+2)
    for i in range(1,2*N):
        res[i] = c*(max((V[i+1]-V[i])/h,0)/(x[i]*(V[i-1]-2*V[i]+V[i+1])/h**2))
    return res

def aug_prob_matrix(x,u,b,r,delta,h,N,meth):
    '''
    Esta funcion construye la matriz P(pi_k^(Delta x,Delta t)).
    Parametros:
    x = arreglo de la particion del intervalo en el eje x.
    u = la politica optima
    b = el mu
    delta = longitud del paso en el eje t
    h = longitud del paso en el eje x
    N = numero de elementos en x
    meth = metodo usado para la construccion de la matriz, hay dos opciones:
           1. Metodo upwind
           2. Diferencias simetricas centradas.
    '''
    P = np.zeros((2*N+2,2*N+2))
    P[-1,-1] = 1.0
    if meth == 'upwind':
        for j in range(2*N+1):
```

```
            P[j,j] = 1 - x[j]*(r+(b-r)*u[j])*delta/h - (x[j]**2*u[j]**2*sigma**2)*delta/h
                **2  # d.
            P[j,j+1] = x[j]*(r+u[j]*(b-r))*delta/h + 0.5*x[j]**2*u[j]**2*sigma**2*delta/h
                **2  # d. arriba
        for j in range(1,2*N+1):
            P[j,j-1] = 0.5*x[j]**2*u[j]**2*sigma**2*delta/h**2
                                            # d. abajo
    elif meth == 'simetric':
        for j in range(2*N+1):
            P[j,j] = 1-(x[j]**2*u[j]**2*sigma**2)*delta/h**2
                                                    # d.
            P[j,j+1] = 0.5*x[j]*(r+u[j]*(b-r))*delta/h + 0.5*x[j]**2*u[j]**2*sigma**2*
                delta/h**2  # d. arriba
        for j in range(1,2*N+1):
            P[j,j-1] = 0.5*x[j]**2*u[j]**2*sigma**2*delta/h**2 - 0.5*(x[j]*(r+(b-r)*u[j])
                )*delta/h # d. abajo

    return P

def aproximacion_pi_aug(N,meth):
    '''
    Esta funcion encuentra el valor de la aproximacion mediante cadenas de Markov
    del parametro pi y de los valores de la funcion de valor optimo V.
    '''
    h = float(1/N)
    x = np.array([i*h for i in range(2*N+2)])
    psi = x**gamma/gamma
    u = pi(psi,x,N,b,r,sigma,h)
    P = aug_prob_matrix(x,u,b,r,delta,h,N,meth)
    V = np.dot(P,psi)
    A = []
    A.append(psi)
    A.append(V)
    for i in range(M-1):
        u = pi(V,x,N,b,r,sigma,h)
        P = aug_prob_matrix(x,u,b,r,delta,h,N,meth)
        V = np.dot(P,V)
        A.append(V)
    Im = [A[i][N] for i in range(M,-1,-1)]
    return u[N],Im[N]
```

B.2 Código Monte Carlo

```
pis = [i*1/M for i in range(M+1)]
def MC_Exha(pis,M,K):
    '''
    Esta funcion implementa el metodo exhaustivo para la obtencion de los valores
    y argumentos maximos de la funcion de valor.
    '''
    ind = []
    means = []
    for p in pis:
        W = WP(x0,r,mu,p,sigma,M,K,T)
        means.append(np.mean(W))
        ind.append(p)
    errors = [S-means[j] for j in range(len(means))]
    return max(means),ind[np.argmax(means)],errors

    def MC_GS(f,M,K,a,b,tol):
    '''
    Esta funcion implementa la optimizacion golden search para la obtencion
    de los valores y argumentos maximos de la funcion de valor.
    '''
    from math import sqrt
    ai,bi, invphi = a,b,(sqrt(5)-1)/2
    coord = []
    err = []
    coord.append(a)
    coord.append(b)
```

```
    err.append(f(a))
    err.append(f(b))
    while(abs(bi-ai)>tol):
        ci = bi - (bi-ai)*invphi
        di = ai + (bi-ai)*invphi
        coord.append(ci)
        err.append(f(ci))
        coord.append(di)
        err.append(f(di))
        if(f(ci)>f(di)):
            bi = di
        else:
            ai = ci
    coord.append((ai+bi)/2)
    err.append((f((ai+bi)/2)))
    cand = [f(ai),f(bi),f(ci),f(di),f((ai+bi)/2)]
    print(cand)
    ind = [ai,bi,ci,di,(ai+bi)/2]
    print(ind)
    return [max(cand),ind[np.argmax(cand)],coord,err]
```

B.3 Código de Diferencias Finitas

```
def Implicit_M(Nx,Nt,T,r,mu,sigma,gamma,pi):
    '''
    Esta funcion usa los parametros Nx, Nt para construir una grilla discretizada
    y luego se implementa el metodo implicito para encontrar la solucion numerica.

    '''
    f = lambda x,tau: np.exp(tau*(0.5*(mu-r)**2/(sigma**2)*(gamma/(1-gamma)) + r*gamma))
        *(x**gamma)/gamma
    x = np.linspace(0,2,Nx+1)
    t = np.linspace(0,T,Nt+1)
    dx = x[1] - x[0]
    dt = t[1] - t[0]
    # Construccion de matriz que representara la aproximacion
    v = np.zeros((Nx+1,Nt+1))

    # Condiciones de frontera
    v[0] = [f(0,k) for k in t]
    v[-1] = [f(2,k) for k in t]

    # Condicion inicial
    v[:,0] = [f(j,0) for j in x]

    def alpha(j):
        return 0.5*(x[j]**2)*(sigma**2)*(pi**2)*(dt/dx**2)
    def beta(j):
        return x[j]*(r + (mu-r)*pi)*(dt/(2*dx))
    # diagonal inferior
    def l(j):
        return beta(j) - alpha(j)
    # diagonal
    def d(j):
        return 2*alpha(j) + 1
    # diagonal superior
    def u(j):
        return -beta(j) - alpha(j)
    A_Imp = np.zeros((Nx-1,Nx-1))
    for i in range(1,Nx-1):
        A_Imp[i,i-1] = l(i)
    for i in range(Nx-2):
        A_Imp[i,i+1] = u(i)
    for i in range(Nx-1):
        A_Imp[i,i] = d(i)

    # Creamos una funcion que capture las condiciones de frontera
    def cap(k):
        vec = np.zeros(Nx-1)
```

```
        vec[0] = -l(0)*v[0,k]
        vec[-1] = -u(Nx-1)*v[Nx,k]
        return vec

    # Ahora una funcion que ayudara a resolver un sistema Ax = b con el metodo LU
    def Tridiag_solver(A,b):
        A = np.array(A).astype(float)
        b = np.array(b).astype(float)
        k = len(A)
        for j in range(1,k):
            A[j,j] = A[j,j] - (A[j,j-1]/A[j-1,j-1])*A[j-1,j]
            b[j] = b[j] - (A[j,j-1]/A[j-1,j-1])*b[j-1]
        x = [0]*k
        x[k-1] = b[k-1]/A[k-1,k-1]
        for i in range(k-2,-1,-1):
            x[i] = (b[i] - A[i,i+1]*x[i+1])/A[i,i]
        return x
    # Encontramos ahora los valores v
    for k in range(1,Nt+1):
        v[1:Nx,k] = Tridiag_solver(A_Imp,(v[1:Nx,k-1]+cap(k)))
    return v[int(Nx/2),-1]
```