Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Paralelização de algoritmos de precificação de opções europeias

Lucas Civile Nagamine

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Pedro Peixoto

São Paulo
24 de Dezembro de 2021

Resumo

Lucas Civile Nagamine. **Paralelização de algoritmos de precificação de opções europeias**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Nas últimas décadas, as áreas de Finanças, Matemática e Ciência da Computação vêm se tornando cada vez mais conectadas. Um tema comum entre elas é o estudo de modelos de precificação de opções, os quais são amplamente utilizados por acadêmicos e profissionais do mercado para a obtenção do preço justo desses ativos financeiros. Este trabalho visa examinar, sob a ótica computacional, três desses métodos: o Modelo Binomial, o Método de Monte Carlo e o Método das Diferenças Finitas. Todos os modelos tiveram sua complexidade computacional analisada e foram implementados em duas versões: uma sequencial e outra paralela. Os códigos foram testados com diferentes parâmetros de entrada e tiveram seus tempos de execução medidos, com o objetivo de que se avaliasse a eficiência da paralelização nos três modelos abordados. Por fim, discutiu-se a viabilidade da utilização das implementações deste trabalho para a precificação de opções em situações do mundo real.

Palavras-chave: Opções europeias. Modelos de precificação. Paralelismo. Modelo Binomial. Método de Monte Carlo. Método das Diferenças Finitas.

Abstract

Lucas Civile Nagamine. **Parallelization of european option pricing algorithms**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

Over the past decades, the fields of Finance, Mathematics and Computer Science have become increasingly connected. A topic common to these areas is the study of option pricing models, which are widely used by academics and market practitioners who wish to obtain a fair price for those financial assets. This work aims to examine, under the computational aspects, three of these methods: the Binomial Model, the Monte Carlo Method and the Finite Difference Method. All models have had their computational complexity analyzed and have been implemented in two different versions: a sequential one and a parallel one. The codes have been tested with varied input parameters and have had their execution times measured, with the purpose of assessing the parallelization's efficiency in all three aforementioned models. Lastly, the viability of using this work's implementations for real-world option pricing situations has been discussed.

Keywords: European options. Pricing models. Parallelism. Binomial Model. Monte Carlo Method. Finite Difference Method.

Sumário

Capítulo 1

Introdução

Os contratos derivativos podem ser definidos como os instrumentos financeiros cujo valor é derivado de uma variável subjacente [9]. Essa variável, costumeiramente, é o preço (ou o comportamento do preço) de um ativo, um índice, uma taxa de juros, entre outros.

Nas últimas décadas, os mercados globais vêm presenciando o uso cada vez mais extenso de contratos derivativos por variados gêneros de instituições financeiras [2]. Atualmente, inúmeros tipos de derivativos são negociados nas bolsas e mercados de balcão do mundo todo. As principais categorias existentes são os futuros, os *forwards*, os *swaps* e as opções.

Devido à ampla gama de grupos e especificações distintas disponíveis, esses contratos são operados pelas três principais espécies de *traders* ("negociantes"): os *hedgers*, os especuladores e os arbitradores [9].

Hedgers são os participantes do mercado que buscam reduzir ou anular um determinado risco advindo de suas posições. Um exemplo simples é o do *hedge* cambial: um investidor brasileiro que possui ativos denominados em dólares americanos, mas não pretende ter risco cambial com essas posições, pode cobrir a exposição à variação cambial por meio da venda de contratos futuros de dólar contra o real. Assim, perdas incorridas com suas posições *offshore* oriundas da depreciação do dólar frente ao real serão anuladas com a respectiva valorização da venda dos futuros (e vice-versa).

Especuladores, em contraste, desejam tomar uma posição bem definida e, portanto, lucrar caso o mercado se movimente conforme suas antecipações. Tomemos como exemplo um investidor que acredita que o preço de uma ação estará maior daqui a um mês do que os valores observados hoje. Para obter exposição a essa tese, ele pode adquirir uma opção de compra da ação, com vencimento de um mês e cujo preço de exercício seja menor do que o preço que ele prevê para a ação ao fim desse período.

Por fim, os arbitradores buscam realizar operações simultâneas que, de alguma forma, propiciem lucro garantido. Um caso de fácil entendimento é o de arbitragem por discrepância de preços em mercados distintos. Se um ativo é negociado em duas bolsas de países diferentes e, em dado momento, o preço desse ativo está maior em um país do que em

outro (levando em consideração a taxa de câmbio entre as moedas de cada país), é possível comprar a ação no mercado em que ela está mais barata e vendê-la naquele em que o preço está maior, de forma a se obter um ganho garantido sem risco. Operações mais complexas de arbitragem podem ser realizadas por meio do emprego de derivativos.

Com base nos parágrafos anteriores, entende-se o porquê de os derivativos serem tão comumente negociados nos mercados globais. Cifras financeiras são um indicativo mais concreto da popularidade desses contratos ao redor do mundo: em 2017, a Autoridade Europeia de Valores e Mercados (ESMA) estimou em €660 trilhões o tamanho do mercado de derivativos somente na Europa [15].

1.1 Opções e sua precificação

O foco do presente trabalho reside sobre as opções, um dos tipos mais populares de derivativos. As opções são contratos que garantem ao seu titular (o "dono" da opção, aquele que a comprou) o direito, mas não a obrigação, de exercer a compra ou a venda de um ativo a um preço pré-estipulado e em um período pré-determinado. No próximo capítulo, discorrer-se-á mais a fundo sobre os detalhes do mercado de opções e as variáveis que influenciam o seu preço.

Por ora, é importante saber que, devido à relevância das opções no mercado financeiro, elas foram amplamente examinadas ao longo das últimas décadas. Dos inúmero estudos realizados, emergiram diversos modelos de precificação de opções, que visam determinar, dadas as condições de mercado, um preço justo para esses instrumentos financeiros. Tais modelos são bastante úteis aos *players* do mercado, para os quais é essencial estimar o valor de um ativo antes de entrar em um negócio que o envolva.

Mais do que apenas determinar o preço justo da opção, é necessário fazê-lo de forma rápida. Nos mercados atuais, pautados pela ampla liquidez e pela informatização dos processos, poucos segundos podem ser determinantes para o aproveitamento (ou a perda) de condições favoráveis nos preços. Por esse motivo, fica claro que, nos dias de hoje, a agilidade necessária torna indispensável o uso de ferramentas computacionais voltadas à precificação de ativos. Não à toa, os participantes do mercado contam com serviços digitais que unem ferramentas de precificação a plataformas de visualização de preços e envios de ordens de operação, a exemplo do *Bloomberg Terminal*.

Na seção seguinte, daremos alguns passos atrás na linha do tempo e veremos como, ao longo das últimas décadas, a área de Finanças vem se tornando cada vez mais indissociável do uso de ferramentas e métodos próprios da Computação.

1.2 Computação e finanças

O estudo de Ciência da Computação aplicada a finanças deu seus primeiros passos na década de 1950, quando o economista americano Harry Markowitz trabalhou no desenvolvimento de algoritmos que encontravam soluções aproximadas para o problema

da seleção de portfólio[1] [10]. Anos depois, métodos computacionais começaram a ser aplicados à gestão de fundos de investimento, com o uso de algoritmos voltados à detecção de oportunidades de arbitragem, tendo como pioneiro o matemático Edward Thorpe [16], também americano.

Na área de precificação de opções, o emprego de ferramentas próprias de Computação Financeira iniciou-se na década de 1970, com o surgimento de modelos de precificação mais sofisticados. Por exemplo, em 1979, quando Cox et. al [4] apresentaram o Modelo Binomial (que será abordado mais adiante neste trabalho), já se levava em consideração a adoção de procedimentos numéricos computacionalmente mais eficientes.

Nessa época, começou a migração em massa de profissionais das áreas de Matemática, Estatística e Computação (os chamados *rocket scientists*, ou "cientistas de foguete") para o mercado financeiro. Com esse movimento, pesquisas de viés quantitativo, conduzidas com o uso de ferramentas computatcionais, tornaram-se bastante comuns no campo de gestão de portfólio.

Desde então, a aplicação de Ciência da Computação a finanças vem se tornando cada vez mais disseminada entre *practitioners*. Hoje, por exemplo, existem inúmeros fundos de investimento cujas decisões são guiadas exclusivamente pelo emprego de modelos/algoritmos (os chamados fundos quantitativos). Esses fundos, somados a outras categorias sistemáticas de índices, representam aproximadamente 35% do mercado americano de renda variável [6].

Além disso, mesmo entre praticantes de gestão discricionária (cujas decisões advêm do discernimento e discricionariedade dos gestores), ferramentas computacionais tornaram-se peça fundamental do processo de análise de investimentos, atendendo a inúmeros propósitos: processamento e visualização de dados, precificação de ativos financeiros, simulação de portfólios teóricos, estudos de risco, entre outros. Por esse motivo, conforme citado na seção anterior, os últimos anos presenciaram o surgimento de diversos *softwares* voltados a profissionais do mercado financeiro.

1.3 Este trabalho

Com base nas seções anteriores, é lógico concluir que a intersecção entre as áreas de Finanças, Matemática e Computação é bastante ampla e, ao longo das últimas décadas, vem ganhando cada vez mais relevância tanto entre acadêmicos quanto *practitioners*. Com o intuito de explorar a interdisciplinariedade desse tópico, e tendo como motivação a importância da obtenção **veloz** de preços justos para opções, este trabalho visa examinar, sob a ótica computacional, os três seguintes modelos de precificação de opções:

1. Modelo Binomial;
2. Método de Monte Carlo;

[1] Este problema se resume a encontrar o conjunto de ativos que maximiza a relação retorno-risco de um portfólio. Soluções exatas para o problema exigiam poder computacional além do disponível naquela época, o que tornava necessário o desenvolvimento de algoritmos de aproximação.

3. Método das Diferenças Finitas Explícito.

O enfoque do presente estudo é a implementação de versões desses modelos que, pelo menos em alguns de seus trechos, façam uso de paralelismo, com o propósito de acelerar o tempo de execução dos programas. Dessa maneira, cada um dos métodos acima será abordado de forma que os aspectos a seguir sejam tratados:

- Conceituação teórica do modelo;
- Implementação do modelo na linguagem *Python* (versão unicamente sequencial);
- Análise da complexidade do algoritmo implementado;
- Implementação de uma versão alternativa que, ao fazer uso de paralelismo, procura acelerar o tempo de execução do algoritmo;
- Medições de ambas as implementações e comparação entre os tempos de cada versão.

A cada um desses métodos, será reservado um capítulo. Em cada capítulo, a disposição dos assuntos será feita na mesma ordem da lista acima: primeiramente, descrever-se-ão os aspectos teóricos do modelo; em sequência, a versão paralela será implementada e analisada, seguida pela alternativa paralela; depois, ambas as versões serão testadas e medidas para diferentes parâmetros de entrada e; por fim, algumas conclusões sobre os algoritmos e as medições serão apresentadas.

Antes do aprofundamento nos modelos de precificação, o capítulo "Conceitos relevantes" apresentará alguns dos tópicos mais importantes para o pleno entendimento deste trabalho - todos relacionados ao mercado de opções ou à computação paralela.

1.4 Objetivos

Os objetivos do trabalho podem ser agrupados em três frentes principais: (i) apresentação dos modelos de precificação de opções, (ii) implementação das duas versões (paralela e sequencial) de cada modelo e (iii) medições e obtenção de conclusões acerca da eficácia da paralelização do modelo. A seguir, apresentam-se os objetivos de cada frente.

1.4.1 Apresentação dos modelos

Esta frente se refere, fundamentalmente, à descrição dos três modelos de precificação abordados no trabalho, incluindo, para cada modelo, uma breve introdução histórica, uma conceituação matemática e teórica sucinta e, por fim, um detalhamento dos passos envolvidos na obtenção do preço justo da opção a partir dos parâmetros fornecidos.

É válido salientar que o propósito da apresentação dos modelos não é fornecer um rígido aprofundamento matemático, mas, sim, descrevê-los de forma que, mesmo antes da apresentação das implementações, seja possível antever o "esqueleto" dos algoritmos correspondentes. Em outras palavras, o grande objetivo desta frente é detalhar cada um dos modelos com o intuito de torná-los compreensíveis dos pontos de vista computacional e algorítmico.

1.4.2 Implementações

Para esta frente, o primeiro objetivo é a implementação da versão sequencial dos modelos de precificação, de maneira coerente com a fundamentação teórica apresentada inicialmente. Além disso, como segundo objetivo, pretende-se analisar os algoritmos implementados quanto à sua complexidade. Assim, ao fim das análises, devemos ser capazes de determinar a relação entre o tempo de execução dos algoritmos e cada um de seus parâmetros de entrada.

Finalmente, resta a implementação dos modelos em sua versão paralela. Aqui, o objetivo é desenvolver uma alternativa aos códigos sequenciais implementados anteriormente que faça uso de paralelismo. A proposta é que, em trechos do código determinantes para o tempo de execução do programa, empreguem-se múltiplos processos simultâneos, com o intuito de acelerar a precificação da opção em questão.

1.4.3 Medições e conclusões

A última frente contém os objetivos mais relevantes quanto aos resultados finais deste trabalho. Primeiramente, pretende-se estabelecer valores para os parâmetros de teste dos algoritmos que nos permitam observar com clareza o impacto de alterações no tamanho da entrada sobre o tempo final de execução dos programas. Com os diferentes conjuntos de parâmetros definidos, executar-se-ão ambas as versões de cada modelo e medir-se-ão os tempos de execução.

Os tempos coletados serão essenciais para **o principal objetivo deste estudo**: determinar em quais modelos a versão paralela se mostra mais eficiente[2] que a sequencial. Ou seja, para cada um dos modelos, avaliar-se-á se o *overhead* advindo do gerenciamento dos processos simultâneos é mais ou menos custoso que os ganhos de tempo obtidos com a paralelização dos cômputos do algoritmo.

[2] Em termos de tempo de execução.

Capítulo 2

Conceitos relevantes

Este trabalho engloba dois campos de estudo - Computação e Finanças - que, apesar de distintos, possuem inúmeros temas que podem ser abordados conjuntamente. Neste caso, os enfoques são, do lado de Finanças, o mercado de opções e, do lado de Computação, o paralelismo. Sendo assim, este capítulo discute alguns conceitos relevantes a ambos os tópicos.

2.1 Mercado de opções

Conforme definido no capítulo introdutório, derivativos são instrumentos financeiros cujo valor é dependente do valor de outro ativo (o chamado "ativo subjacente"), bem como de outras variáveis adicionais. Também explicou-se que as opções são definidas como um tipo de derivativo que garante ao seu titular (o "dono" da opção, aquele que a comprou) o direito, mas não a obrigação, de exercer a compra ou a venda de um ativo a um preço pré-estipulado e em um período pré-determinado. Ao lançador da opção (ou seja, aquele que a vendeu) cabe a obrigação de acatar a escolha do titular e vender/comprar o ativo, caso assim seja desejado.

Neste capítulo, serão discutidos alguns dos pontos principais no tocante a opções, como as classificações/categorias existentes e o impacto de diferentes variáveis no preço de uma opção.

2.1.1 Classificações

Opções podem ser classificadas de diferentes maneiras. Atualmente, os três aspectos distintos que mais são utilizados quando da classificação de uma opção são (*i*) o direito do titular, (*ii*) a classe do ativo subjacente e (*iii*) o estilo da opção.

Direito do titular

Esta é a classificação mais relevante sobre a natureza da opção. São duas as nomenclaturas utilizadas: a *call*, opção que garante ao titular o direito de **comprar** o ativo subjacente a um determinado preço, e a *put*, que garante o direito de **vender** o ativo a certo preço. É

bastante comum a utilização dos termos em inglês, uma vez que não existe tradução em uma única palavra equivalente em português. Para evitar anglicismos, podem-se empregar as expressões *opção de compra* e *opção de venda*.

Neste estudo, trabalhar-se-á com os dois tipos de opção: as *calls* serão abordadas nos capítulos relativos à precificação por Simulação de Monte Carlo e pelo Modelo Binomial, enquanto se utilizarão as *puts* no capítulo de precificação pelo Método das Diferenças Finitas.

Classe do ativo subjacente

Para esta classificação, o aspecto de importância é a classe do ativo cujo direito de compra/venda está sendo negociado. Existem inúmeros rótulos: opção de ação, opção de futuro, opção de índice, opção de *swap*, opção de moeda, entre outros. Apesar da vasta gama de classes de ativo subjacente, o princípio por trás de todas essas classificações é o mesmo. Nos capítulos seguintes, será tratado apenas o caso mais comum, as opções de ação[1].

Estilo da opção

Esta é, possivelmente, a classificação mais complexa sobre o tipo da opção. Aqui, diferenciam-se as opções com relação ao(s) período(s) em que o titular pode exercê-las (ou seja, quando ele pode optar por utilizar seu direito de negociação do ativo subjacente). Existem diversos estilos distintos, mas os mais comuns são:

- Opções europeias: aquelas em que o exercício só pode ocorrer no vencimento da opção;
- Opções americanas: aquelas em que o exercício pode ocorrer em qualquer período anterior ou igual à data de vencimento.

Apesar de os nomes dos estilos envolverem regiões geográficas, ambos os tipos mencionados são negociados em mercados de todo o mundo. Neste trabalho, apenas as opções europeias serão abordadas.

2.1.2 Posições em uma opção

Conforme explicado acima, existem dois "lados" em uma opção: o titular, que comprou a opção e possui o direito de negociação sobre o ativo subjacente, e o lançador, que vendeu a opção e a quem cabe ser a contraparte do negócio caso o titular deseje efetuá-lo.

Considerando a classificação das opções em *call* e *put*, vê-se que há quatro posições possíveis em uma opção:

- *Long call*: posição comprada em uma opção de compra - ou seja, a posição que possui o direito de comprar o ativo subjacente;
- *Short call*: posição vendida em uma opção de compra - ou seja, a posição do lançador da opção;

[1] Além disso, neste trabalho, assumem-se ações não pagadoras de dividendos.

- *Long put*: posição comprada em uma opção de venda - ou seja, a posição que possui o direito de vender o ativo subjacente;
- *Short put*: posição vendida em uma opção de venda - ou seja, a posição do lançador da opção;

No jargão financeiro, são bastante usuais os termos *long*, para se referir a uma posição comprada, e *short*, relativo a uma posição vendida.

2.1.3 Exemplos ilustrativos

Para facilitar o entendimento sobre a mecânica do mercado de opções, vejamos exemplos envolvendo a negociação de uma opção de compra (*call*) e uma de venda (*put*), sendo ambas europeias.

Call

Bob vende a Alice uma opção de compra por R$ 1,50. Bob, portanto, é o lançador (*short*), ao passo que Alice, por ser a titular (*long*), tem o direito de comprar a ação pelo preço de exercício no dia do vencimento. No contrato da opção, está estipulado que o preço de exercício é de R$ 12,00 e que o tempo até o vencimento é de 2 meses. Atualmente, a ação subjacente à opção é negociada a R$ 11,00.

Ao fim dos dois meses, os cenários possíveis de lucro/prejuízo para Bob e Alice são:

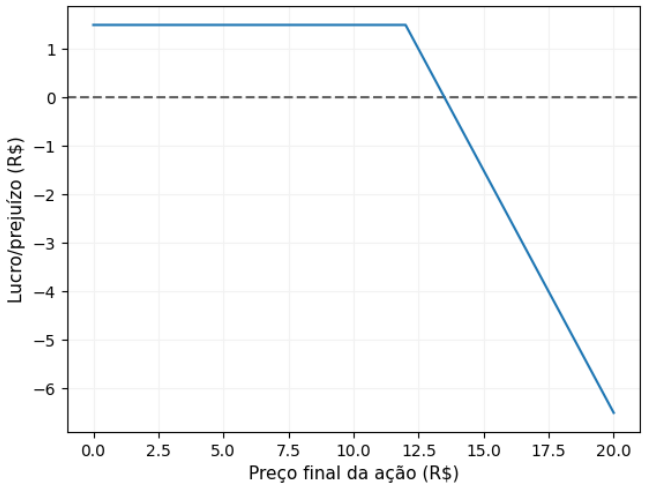

Figura 2.1: *Bob (short call)*

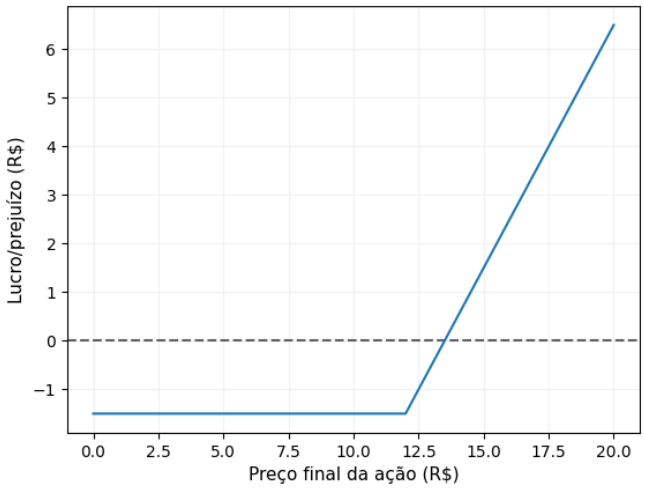

Figura 2.2: *Alice (long call)*

Para Bob, o lançador da opção, o lucro máximo de R$1,50 ocorre quando o preço da ação é menor que R$12,00 (ou seja, nos casos em que não há exercício da opção). Aos R$13,50, o resultado líquido de Bob é zero e, em preços maiores, ele incorre prejuízo. Para Alice, a situação é oposta. Sua perda máxima ocorre quando a ação vale menos que R$12,00 e não há exercício. Depois disso, seu resultado aumenta gradativamente, sendo que se obtém lucro quando o preço é maior que R$13,50.

Put

Nesta subseção, Bob vende a Alice uma opção de venda, com as mesmas especificações descritas no item acima. Agora, a relação dos resultados de Alice e Bob com o preço final da ação subjacente é:

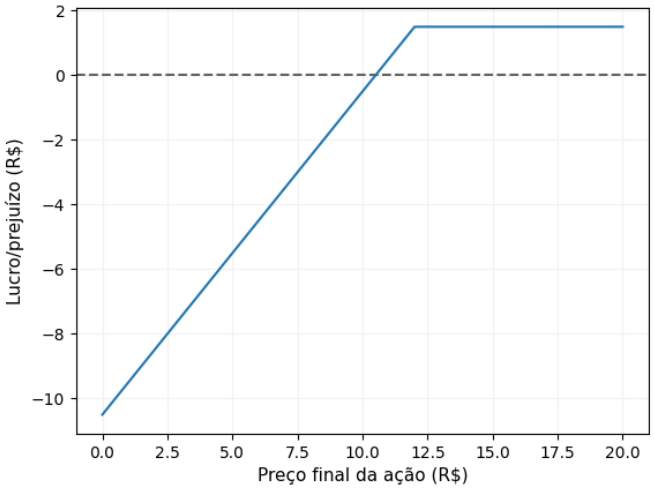

Figura 2.3: *Bob (short put)*

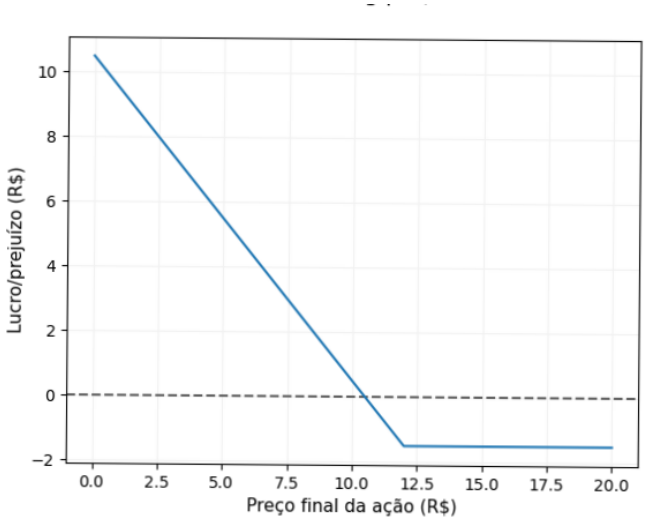

Figura 2.4: *Alice (long put)*

Assim como no caso da *call*, o lucro máximo de Bob continua a ser de R$1,50, mas seu resultado piora conforme o preço da ação **decresce**, sendo que não há lucro nem prejuízo quando a ação tem o preço de R$10,50. Para Alice, a perda máxima é de R$1,50 e ocorre quando a ação custa R$12,00 ou mais. Abaixo disso, quanto menor o preço da ação, melhor o resultado de Alice, sendo que, dos R$10,50 para baixo, ela obtém lucro com a operação.

2.1.4 Variáveis de interesse

Não é apenas o preço do ativo subjacente que determina o valor de suas opções. Há diversas outras variáveis a serem levadas em consideração, que podem afetar positiva ou negativamente o preço da opção a ser precificada. Nesta seção, abordar-se-ão em mais detalhes as variáveis que influenciam o preço de opções de ações que não pagam dividendos[2]:

Preço atual da ação

No caso de uma *call*, conforme o preço da ação aumenta, a opção ganha valor - uma vez que, quanto maior for a diferença (positiva) entre o preço da ação e o preço de exercício, maior será o lucro do titular da opção.

Para as *puts*, o efeito é inverso: dado que não é do interesse do titular vender uma ação por um preço menor que o praticado no mercado, a opção perde valor quando o preço da ação aumenta.

[2] Grande parte das variáveis abordadas é utilizada pelos modelos de precificação de opções tratados nos capítulos a seguir. Aos leitores do presente estudo, sugere-se observar como, de fato, as relações desenhadas neste capítulo entre as variáveis e o preço da opção são válidas nos modelos de precificação.

Preço de exercício

Dos comentários do item anterior, pode-se depreender que a relação entre a opção e o preço de exercício funciona de forma oposta à relação entre a opção e o preço da ação. As *calls* decrescem e as *puts* crescem conforme o preço de exercício aumenta.

Tempo até o vencimento

Via de regra, quanto maior o tempo de vencimento, maior é o valor das opções. Uma forma intuitiva de entender essa relação é que, em períodos mais longos, a incerteza sobre o preço futuro da ação é maior e, como a opção garante ao titular um preço "travado" para operar, é mais interessante ter o preço travado em um cenário maior de incertezas.

Volatilidade

A posse de uma opção garante riscos assimétricos para o titular. Ao comprá-la, ele assume um risco de perda máxima igual ao preço que pagou pela opção, mas os ganhos potenciais são teoricamente infinitos[3].

Dessa maneira, uma volatilidade maior, ao indicar possíveis movimentos de maior magnitude (para cima e para baixo) no preço da ação, aumentam os potenciais de ganho, mas não os de perda, para o titular. Sendo assim, aumentos na volatilidade de uma ação resultam em preços maiores para a opção correspondente.

Taxa livre de risco

Um acréscimo na taxa livre de risco resulta em aumento da taxa de retorno esperada das ações (por serem estas ativos de risco). Dessa forma, as expectativas quanto aos preços futuros tendem a ser maiores, o que beneficia os titulares de opções de compra e prejudica os titulares de opções de venda. Sendo assim, um aumento na taxa livre de risco eleva o preço das *calls* e reduz o preço das *puts*.

2.2 Paralelismo

Paralelismo e concorrência são dois dos recursos computacionais mais utilizados para a redução do tempo de execução de programas. Apesar de similares, os termos representam conceitos ligeiramente distintos: concorrência é o estado em que existe sobreposição dos períodos entre o começo e o fim de duas tarefas distintas - não sendo necessário que elas sejam executadas concomitantemente -; em paralelismo, por outro lado, a execução simultânea é condição *sine qua non*.

Neste trabalho, visa-se acelerar os códigos relativos aos algoritmos de precificação de opções por meio de paralelismo. Ou seja, nas versões paralelas dos modelos, os problemas computacionais são particionados de forma que as partes possam ser processadas individualmente e ao mesmo tempo por núcleos de processamento da máquina distintos. Por fim,

[3] O ganho, em teoria, pode ser infinito para o titular de uma opção de compra. No caso de uma *put*, o ganho máximo do titular é igual ao preço de exercício menos o valor pago pela opção.

os resultados de cada cálculo paralelo são agregados, o que nos permite chegar à "resposta final" do problema.

Na subseção a seguir, discute-se brevemente como ocorre o paralelismo na linguagem *Python*, adotada nos códigos dos próximos capítulos.

2.2.1 Paralelismo em *Python*

Antes de focarmos nas especificidades do *Python*, é necessário introduzir dois novos conceitos: processos e *threads*. Um processo é, basicamente, a execução de uma aplicação ou programa, cujos dados e memória próprios são, via de regra, inacessíveis para outros processos. A *thread* é um conjunto de instruções a serem executadas pela máquina. Cada processo possui múltiplas *threads*, as quais, em teoria, podem compartilhar memória e informação entre si.

Em *Python*, todavia, o compartilhamento de memória entre diferentes *threads* de um mesmo processo é bloqueado por meio do *GIL* (*Global Interpreter Lock*, ou Trava Global do Interpretador). Esse bloqueio serve para evitar que múltiplas *threads* atualizem um mesmo objeto de memória ao mesmo tempo, o que poderia corrompê-la. Na prática, a consequência gerada pelo bloqueio do *GIL* é que, em *Python*, *threads* de um mesmo processo não podem ser executadas de forma paralela, mas apenas concorrente.

Uma vez que, na linguagem em questão, cada processo possui o seu próprio interpretador e, portanto, o seu próprio *GIL*, o paralelismo pode ser obtido por meio da execução simultânea de processos distintos pelos diferentes núcleos da máquina. Ou seja, a execução de operações da *CPU* pode ser acelerada pelo uso paralelo de múltiplos processos, em detrimento da concorrência de múltiplas *threads* pertencentes a um mesmo processo.

A paralelização por processos dos algoritmos deste trabalho é realizada por meio do emprego de duas bibliotecas: *multiprocessing* e *Numba*. Para encerrar este capítulo, os próximos parágrafos contêm uma apresentação sucinta de cada uma.

multiprocessing

multiprocessing é um pacote do *Python* que oferece suporte ao *spawning* de processos [11]. Isto é, permite-se gerar subprocessos a partir de um processo pai. Como os processos filhos não são limitados pelo *GIL*, eles podem ser executados em paralelo. Ademais, não há compartilhamento de memória entre eles, o que evita o evento de corrupção de memória citado anteriormente.

A classe *Pool* divide o trabalho (no caso, as múltiplas execuções de uma mesma função) entre os processos que serão rodados em paralelo. Na inicialização de um objeto dessa classe, fornece-se o argumento relativo ao número de processos simultâneos que desejamos executar - ou seja, o número de núcleos da máquina utilizados para computar o código[4]. Depois, coletam-se os resultados obtidos nas execuções da função pelos diferentes processos e agregam-se os dados.

[4] Contanto que o argumento fornecido seja menor ou igual à quantidade de núcleos disponíveis.

Numba

O pacote *Numba*, desenhado especificamente para computação científica, oferece um compilador *JIT* (*Just-In-Time*) que otimiza a transpilação de códigos em *Python* para linguagem de máquina.

Uma das funcionalidades mais interessantes desse pacote é a chamada *automatic parallelization* ("paralelização automática"), na qual o *Numba* percorre as operações definidas pelo usuário e identifica quais delas possuem semântica paralela (ou seja, quais podem ser executadas em paralelo) [1]. Ativando-se essa funcionalidade (por meio dos decoradores `@jit` e `@njit`), não é necessário fazer qualquer outra alteração no código: o próprio *Numba* gerencia o paralelismo. Além disso, o pacote é compatível com o *NumPy* e oferece suporte para algumas das suas operações matriciais.

Capítulo 3

Modelo Binomial

O Modelo Binomial de Precificação de Opções (MBPO), formalizado por Cox, Ross e Rubinstein em 1979 [4], é uma das técnicas de precificação de opções mais populares e simples [9]. Trata-se, essencialmente, de um modelo que descreve, em forma de árvore, as possíveis evoluções nos valores de uma ação e de sua respectiva opção em diferentes e igualmente intervalados instantes de tempo.

Ao adotar-se o MBPO, assume-se que o movimento do preço da ação (S) segue um "passeio aleatório": em cada um dos passos no tempo (t_i), o preço pode mover-se um certo percentual para cima ou um certo percentual para baixo (representados por u e d, respectivamente). Para fins de representação desse passeio, utiliza-se a chamada árvore binomial - um subtipo de árvore binária no qual, se dois nós irmãos adjacentes não pertencem ao último nível da árvore, então eles possuem um nó filho em comum.

Para o exemplo mostrado na figura 3.1, considere-se uma árvore binária em que, no instante de tempo t_0, uma dada ação vale R$50 (ou seja, $S_0 = 50$). De cada instante de tempo para o seguinte, a ação pode valorizar ou desvalorizar 10% ($u = 1{,}1$, $d = 0{,}9$)[1]. Os valores da opção em cada nó na árvore são representados por f. Naturalmente, f_u representa o preço da opção após uma alta no preço da ação, f_{ud} representa o preço no caso de uma alta seguida de uma queda, e assim por diante[2]:

[1] A despeito do exemplo fornecido para u e d, é perfeitamente aceitável que os valores de valorização e desvalorização da ação sejam diferentes. Por exemplo, poderíamos ter $u = 1,2$ e $d = 0,95$. Mais adiante, ver-se-á que é comum definir u e d com base na volatilidade realizada dos retornos da ação.

[2] A fim de atingir uma notação mais enxuta, adotamos que $f_{u^a d^b}$ é o preço da opção após a movimentos de alta e b de baixa. Note-se que a ordem em que esses movimentos ocorrem não interfere em f.

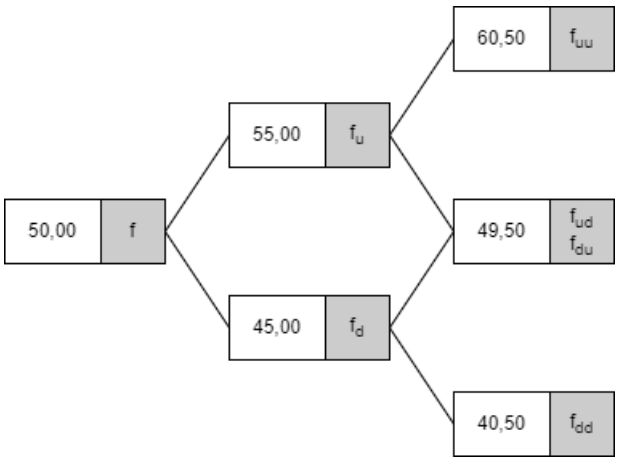

Figura 3.1: *Árvore binomial: preços das ações*

3.1 Hipóteses

Além da hipótese do "passeio aleatório" descrita acima, o modelo binomial pressupõe mais alguns pontos sobre o comportamento do mercado de ações e opções.

Ao assumirmos o passeio aleatório, fica implícita a hipótese de que a distribuição dos retornos do ativo subjacente à opção tem média e variância constantes - ou seja, o perfil de risco-retorno do ativo é o mesmo para todos os instantes de tempo.

Assume-se também que não existe possibilidade de arbitragem. Ou seja, é impossível que se monte um portfólio que **garanta** um retorno superior à taxa livre de risco[3]. A consequência disso é que um portfólio sem risco deve, obrigatoriamente, ter um retorno igual à taxa livre de risco em um dado intervalo. Esta premissa é o que nos permite identificar um "preço justo" para a opção.

Por fim, assume-se que operações de compra e venda de ativos não acarretam custos de transação[4].

3.2 Precificação das opções

Dado que são poucas as variáveis envolvidas na caracterização de uma árvore binomial, a formulação matemática também é relativamente simples. No último nível da árvore - o qual representa o instante de vencimento -, o valor da opção de compra[5] é:

$$f = \max(S - k,\ 0)$$

[3] É importante enfatizar o uso do verbo "garantir". Um portfólio pode, é claro, obter um retorno superior à taxa livre de risco. No entanto, em um cenário onde não existe arbitragem, é impossível assegurar com antecedência esse retorno excedente.

[4] O que não costuma ser verdade nos mercados reais.

[5] Neste capítulo, trataremos apenas de opções de compra (*calls*). Todavia, é simples estender o MBPO às opções de venda (*puts*), bastando substituir, nas fórmulas descritas, $S - k$ por $k - S$.

Vê-se, portanto, que, no vencimento, a opção de compra somente tem valor nos casos em que seu preço de exercício é inferior ao preço da ação - sendo o valor da opção justamente a diferença entre ambos os preços. No exemplo descrito anteriormente, esta seria a árvore binomial após a precificação do último nível:

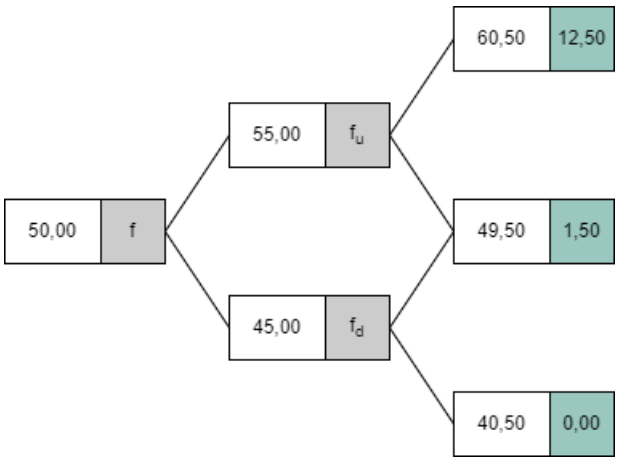

Figura 3.2: *Árvore binomial: precificação das opções do último nível*

Os preços da opção em um nó não terminal dependem dos preços de seus filhos. Para calculá-los, introduzem-se duas novas variáveis:

- r: a taxa livre de risco anual;
- Δt: o intervalo de tempo em anos decorrido entre os níveis da árvore binomial.

Assim, após a precificação das opções do último nível, obtêm-se iterativamente os preços dos níveis anteriores - até o nó raiz - por meio da equação:

$$f = \max(e^{-r\Delta t}[pf_u + (1-p)f_d],\ S - k)$$

onde:

$$p = \frac{e^{-r\Delta t} - d}{u - d}$$

Se entendermos p como a probabilidade de um movimento para cima no preço da ação, então o termo $e^{-r\Delta t}[pf_u + (1-p)f_d]$ pode ser interpretado como o preço esperado da opção (representado por $pf_u + (1-p)f_d$) trazido a valor presente (o que é feito quando o multiplicamos por $e^{-r\Delta t}$).

Além da expressão $e^{-r\Delta t}[pf_u + (1-p)f_d]$, existe a possibilidade de, assim como nos nós terminais, o valor da opção ser $S - k$, caso essa diferença seja superior ao valor presente do preço esperado. Por isso, obtém-se o máximo entre ambas as expressões.

Para o nosso exemplo, assumimos que $r = 0,08$ e $\Delta t = 1/12$. Tem-se, então, a árvore binária completamente precificada:

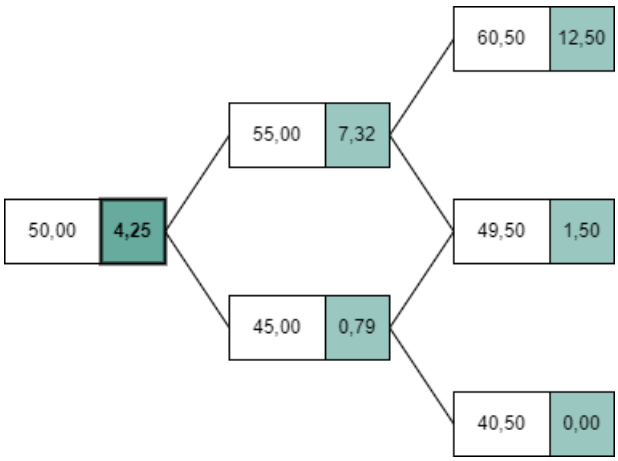

Figura 3.3: *Árvore binomial: precificação completa*

Terminando-se a construção da árvore, determina-se o preço justo da opção como aproximadamente R$4,25.

3.2.1 Estimando u e d

Para que a precificação da opção realizada pelo modelo binomial seja mais precisa, é necessário que os parâmetros do modelo sejam condizentes com as condições reais de mercado da ação subjacente à opção. Nesse sentido, é essencial que se adotem valores para u e d que reflitam a distribuição dos retornos da ação.

Cox, Ross e Rubinstein [4] propõem que se utilize a volatilidade realizada da ação (em termos simples, o desvio-padrão de seus retornos passados), representada pela letra grega σ, para se estimar u e d. Assim, é comum adotar:

$$u = e^{\sigma \sqrt{\Delta t}}$$

$$d = e^{-\sigma \sqrt{\Delta t}}$$

Uma hipótese importante dessa estimativa de u e d é que a a volatilidade da ação se mantém constante ao longo da vida da opção - isto é, para qualquer instante de tempo t, o valor de σ (e, portanto, de u e d) é o mesmo.

3.3 Implementação e Análise

É apresentada, abaixo, a implementação da classe `BinomialTree`, que abstrai a árvore binomial do modelo de precificação descrito neste capítulo. Os objetos dessa classe são inicializados com os parâmetros S_0 (preço da ação em t_0), k (preço de exercício), σ (volatilidade anual da ação), t (tempo até o vencimento da opção), n (profundidade da árvore) e r (taxa livre de risco).

```
1    import math
2
3    class BinomialTree():
4
5        def __init__(self, S0, k, sigma, t, n, r):
```

```
6           self.S0 = S0
7           self.k = k
8           self.dt = t / (n - 1)
9           self.r = r
10          self.n = n
11
12          self.d = math.exp(-sigma * math.sqrt(self.dt))
13          self.u = math.exp(sigma * math.sqrt(self.dt))
14          self.p = (math.exp(self.r * self.dt) - self.d) / (self.u - self.d)
15
16          self.grid = [[] for _ in range(self.n)]
17          self.__insert_nodes()
18
19      class BTNode():
20          def __init__(self, S, f):
21              self.S = S
22              self.f = f
23
24      def __insert_nodes(self):
25          self.grid[0].append(self.BTNode(self.S0, -1))
26
27          for i in range(1, self.n):
28              for j in range(i):
29                  if j == 0:
30                      Sd = self.grid[i-1][j].S * self.d
31                      self.grid[i].append(self.BTNode(Sd, -1))
32
33                  Su = self.grid[i-1][j].S * self.u
34                  self.grid[i].append(self.BTNode(Su, -1))
35
36      def __pricing_formula(self, fd, fu):
37          return math.exp(-self.r * self.dt) * (self.p * fu + (1-self.p) * fd)
38
39      def price_options(self):
40          for node in self.grid[-1]:
41              node.f = max(node.S - self.k, 0)
42
43          for i in range(len(self.grid)-2, -1, -1):
44              for j in range(len(self.grid[i])):
45                  self.grid[i][j].f = max(self.grid[i][j].S - self.k, self.
                        __pricing_formula(self.grid[i+1][j].f, self.grid[i+1][j+1].f)
                        )
46
47          return self.grid[0][0].f
```

A construção da árvore é composta por dois passos:

- Inserção dos nós com os preços da ação - ou seja, os caminhos relativos às possíveis evoluções no valor de S. Começa-se do nó raiz (S_0). Essa inserção é coordenada pelo método `__insert_nodes()`);
- Precificação das opções de cada nó da árvore. Os primeiros nós precificados são os do último nível, seguidos pelos dos penúltimo, e assim sucessivamente, até que se chegue no nó raiz (aquele que pretendemos precificar). O método `price_options()` realiza a precificação dos nós e retorna o preço justo do nó raiz.

À primeira vista, o fato de o modelo binomial de precificação de opções envolver a construção de uma árvore binária leva a crer que a complexidade do algoritmo é da ordem de $O(2^n)$, onde n é a profundidade da árvore - ou, em outras palavras, o número de passos percorridos no tempo. No entanto, devemos lembrar que, conforme descrito acima, a árvore binomial é um subtipo de árvore binária em que nós adjacentes compartilham de um nó filho.

O compartilhamento de filhos entre nós irmãos contribui sobremaneira para o tamanho reduzido da árvore binomial em relação a uma árvore binária comum. A fórmula para o número N de nós de uma árvore binomial de profundidade n é:

$$N(n) = \sum_{i=1}^{n} i = \frac{n}{2}(n+1) = \frac{n^2 + n}{2}$$

A ordem do tamanho da árvore é, portanto, $O(n^2)$. Visualmente, é fácil observar que o número de nós é limitado pelo quadrado da profundidade da árvore quando a representamos em forma de "escada":

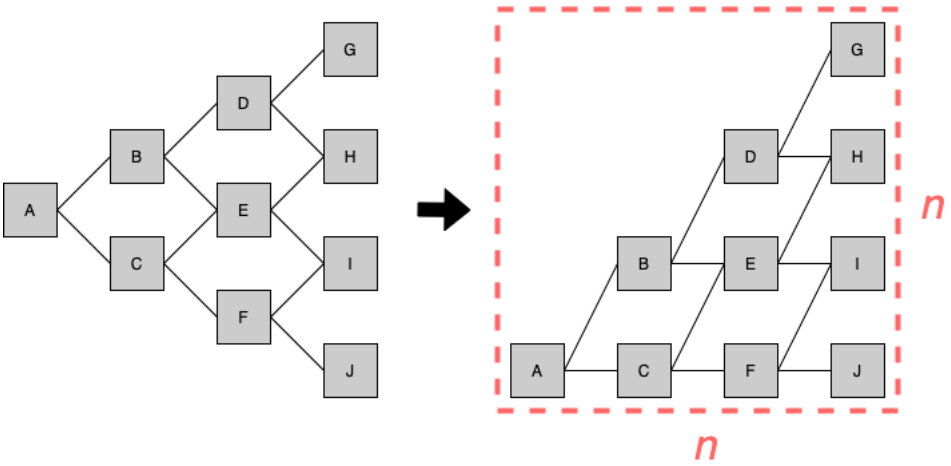

Figura 3.4: *Árvore binomial: representação visual alternativa*

Conforme descrevemos acima, o algoritmo de precificação de opções pelo Modelo Binomial envolve as etapas de construção da árvore de preços da ação e de precificação da opção correspondente em cada nó. Ambas as etapas percorrem cada nó uma vez. Dessa forma, dado que o número de nós é $O(n^2)$, sendo n a profundidade da árvore (ou o número de intervalos no tempo), conclui-se que o algoritmo de precificação de opções por meio do modelo binomial é, no todo, da ordem de $O(n^2)$.

3.4 Versão Paralela

Paralelizar o modelo binomial de precificação de opções não é uma tarefa trivial. Conforme já explicado, a precificação da opção do nó raiz envolve uma dependência iterativa do preço de seus nós descendentes - isto é, o nó raiz depende dos preços de seus filhos, os quais, por sua vez, também dependem dos preços de seus filhos, e assim sucessivamente, até os nós terminais. Dessa maneira, a dependência de dados é forte demais para que se particione a construção e precificação da árvore binomial em múltiplas

tarefas sem que uma dependa de um dado a ser calculado pela outra. Pode-se afirmar, portanto, que a natureza do algoritmo é sequencial, o que dificulta a obtenção de uma versão análoga paralela.

Sendo assim, para que se obtenha uma forma de paralelizar esse modelo, é necessário enxergá-lo sob uma ótica distinta. Nas seções acima, descrevemos o preço f de uma opção como sendo:

$$f = \max(e^{-r\Delta t}[pf_u + (1-p)f_d],\ S - k)$$

onde f_u é o preço da opção um passo adiante no tempo caso a ação suba de acordo com u e f_d é o análogo para a queda em d.

A versão abaixo, menos abrangente que a fórmula anterior, desconsidera a possibilidade de $S - k > e^{-r\Delta t}[pf_u + (1-p)f_d]$:

$$f = e^{-r\Delta t}[pf_u + (1-p)f_d] \tag{3.1}$$

Utilizando a fórmula acima, sabe-se que, se f_u e f_d não são nós terminais, então:

$$f_u = e^{-r\Delta t}[pf_{u^2} + (1-p)f_{ud}] \tag{3.2}$$

$$f_d = e^{-r\Delta t}[pf_{du} + (1-p)f_{d^2}] \tag{3.3}$$

Substituindo (3.2) e (3.3) em (3.1), obtemos a fórmula da precificação para três passos no tempo:

$$f = e^{-2r\Delta t}[p^2 f_{u^2} + 2p(1-p)f_{ud} + (1-p)^2 f_{d^2}]$$

Se f_{u^2}, f_{ud} e f_{d^2} não são terminais, então poderíamos expandi-los em termos de f_{u^3}, $f_{u^2 d}$, f_{ud^2} e f_{d^3}, e assim por diante.

Em vez de expandir a fórmula indefinidamente, pode-se generalizá-la para n passos no tempo:

$$f = e^{-(n-1)r\Delta t} \sum_{i=0}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-1-i} f_{d^{n-1-i}u^i}$$

onde $\binom{a}{b} = \frac{a!}{b!(a-b)!}$.

Assim, passa-se a enxergar o Modelo Binomial como a mera aplicação de uma fórmula, cujo cômputo, em teoria, pode ser acelerado se o cálculo dos termos do somatório envolvido for executado em paralelo. O algoritmo a seguir, que faz uso das bibliotecas *multiprocessing* e *NumPy*, realiza essa tarefa:

```
1   import math
2   import multiprocessing as mp
3   import numpy as np
4
5   def f(S0, k, d, u, nd, nu):
6       S = S0 * (d ** nd) * (u ** nu)
7       return max(S - k, 0)
8
```

```
 9    def calculate_segment(S0, k, d, u, n, start, end, p, triangle):
10       start = math.floor(start)
11       end = math.floor(end)
12
13       factors = np.vectorize(lambda i: (p ** i * (1-p) ** (n - 1 - i) * f(S0, k,
              d, u, n-1-i, i)) * triangle[n-1][i])
14
15       return np.sum(factors(np.arange(start, end)))
16
17    def price_option_parallel(S0, k, sigma, t, n, r, triangle, pools):
18       pool = mp.Pool(pools)
19
20       dt = t / (n - 1)
21
22       d = math.exp(-sigma * math.sqrt(dt))
23       u = math.exp(sigma * math.sqrt(dt))
24       p = (math.exp(r * dt) - d) / (u - d)
25
26       terms = [pool.apply_async(calculate_segment, args = (S0, k, d, u, n, n/
             pools*i, n/pools*(i+1), p, triangle)) for i in range(pools)]
27
28       return sum([t.get() for t in terms]) * (math.exp(-(n-1) * r * dt))
```

Pode-se notar que, além dos parâmetros de entrada da versão sequencial, o código paralelo recebe também o parâmetro *triangle* - um *array*[6] que representa o Triângulo de Pascal, utilizado nos coeficientes binomiais da fórmula de precificação. Pelo fato de o triângulo ser uma constante cujo cômputo leva tempo não desprezível, optou-se por excluir o seu cálculo da implementação. Assim, não precisamos realizá-lo a cada precificação executada. No entanto, caso se optasse por incluí-lo, poder-se-ia utilizar a função abaixo, cujo parâmetro n é a profundidade do Triângulo de Pascal:

```
1    def pascal_triangle(n):
2       triangle = []
3       for i in range(n):
4          l = []
5          for j in range(i+1):
6             l.append(1 if j in (0, i) else triangle[i-1][j-1] + triangle[i-1][j
                  ])
7          triangle.append(l)
8
9       return triangle
```

Conforme citamos, uma limitação do algoritmo acima refere-se ao fato de ele ignorar os casos em que o preço da opção viria de $S - k$, e não da fórmula do valor presente do preço esperado. A depender dos parâmetros de entrada, não haverá diferença no valor obtido para o nó inicial, mas, em outros casos, essa diferença pode ser significativa para o preço calculado.

Além disso, uma segunda limitação do algoritmo refere-se ao cálculo do valor de f para valores altos de n. A fórmula para se obter f depende do cômputo do coeficiente binomial, que, a depender do n utilizado, pode ser armazenado como um inteiro (que, em

[6] Ou um *NumPy array*.

Python, não é restringido pelo número de *bits*), mas não como um *float* (limitado a 64 *bits*). Dessa maneira, torna-se inviável multiplicar o coeficiente calculado pelos outros fatores da fórmula, os quais não são inteiros. Uma alternativa a essa limitação seria utilizar bibliotecas do *Python* que permitem cálculos de alta precisão, como a *Decimal*. No entanto, o emprego dessas bibliotecas retarda sobremaneira o tempo de execução do programa. Por esse motivo, optou-se pela não utilização - o que limita o valor do parâmetro n para inteiros entre 2 e 1030, em prol de um tempo de execução menor.

3.5 Medições e Comparação

Nas implementações realizadas do modelo binomial, a variável que influencia o tempo de execução do algoritmo é o número n de passos no tempo a percorrer a partir do nó raiz da árvore binomial, bem como, no caso da versão paralela, o número de processos utilizados. Dessa maneira, nos testes realizados, mantiveram-se constantes todas as outras variáveis. Adotamos:

- Uma ação cujo preço inicial é de R$ 50 ($S_0 = 50$);
- Preço de exercício da opção de R$ 55 ($k = 55$);
- Volatilidade da ação de 30% ao ano ($\sigma = 0,3$);
- Tempo até o vencimento da opção de 1 ano ($t = 1$);
- A taxa livre de risco de 12% ao ano ($r = 0,12$).

Os valores de n testados foram 10, 100, 500 e 1000 (ou seja, Δt assume os valores 1/10, 1/100, 1/500 e 1/1000). Na versão paralela, os experimentos foram realizados com 1, 2 e 4 processos.

As medições realizadas em uma máquina dotada de 16 GB de memória RAM e de um processador *Intel Core i7* de 4 núcleos foram:

n	preço (R$)	tempo (s)
10	6,340	$2,3x10^{-3}$
100	6,508	$8,0x10^{-3}$
500	6,516	0,174
1000	6,519	0,687

Tabela 3.1: Versão sequencial

n	preço (R$)	tempo (s)
10	6,340	1,053
100	6,508	1,082
500	6,516	1,098
1000	6,519	1,134

Tabela 3.2: Versão paralela - 1 processo

n	preço (R$)	tempo (s)
10	6,340	1,723
100	6,508	1,767
500	6,516	1,808
1000	6,519	2,213

Tabela 3.3: Versão paralela - 2 processos

n	preço (R$)	tempo (s)
10	6,340	2,733
100	6,508	2,845
500	6,516	2,806
1000	6,519	3,576

Tabela 3.4: Versão paralela - 4 processos

Com base nas medições realizadas, algumas conclusões podem ser obtidas:

- Conforme o esperado, as versões sequencial e paralela do modelo binomial apresentaram preços de opção iguais para o mesmo valor de n. Isso ocorre devido ao fato de que, com os parâmetros utilizados, não há casos em que $S - k > e^{-r\Delta t}[pf_u + (1-p)f_d]$. Além disso, o MBPO é essencialmente determinístico, o que pressupõe que, para os mesmos parâmetros de entrada, o modelo deve sempre retornar a mesma precificação. Quando precificamos a opção por meio de Simulação de Monte Carlo (no capítulo a seguir), os valores para o preço variam a cada experimento realizado, dado o fator de aleatoriedade presente na evolução do preço da ação naquele modelo.
- Principalmente nos testes da versão sequencial com mais passos no tempo, pode-se observar que o tempo de execução do algoritmo cresce proporcionalmente ao quadrado de n. Ao aumentarmos n de 500 para 1000 - ou seja, um aumento de 2 vezes -, o tempo vai de 0,174 segundos para 0,687 segundos (o que representa um aumento de aproximadamente 3,95 vezes).
- A observação de maior destaque a ser feita é que, para o Modelo Binomial de Precificação de Opções, a paralelização do algoritmo não se mostrou mais eficiente que a versão sequencial. Para todos os valores de n testados, a versão paralela foi executada em um tempo maior. Ademais, o tempo de execução aumenta conforme utilizamos mais processos, o que demonstra que o *overhead* incorrido com o gerenciamento de múltiplos processos, neste caso, ultrapassa o benefício advindo de seu emprego. Em outras palavras, coordernar os múltiplos processos e somar os resultados obtidos em cada um demora mais tempo do que apenas somar todos os componentes do somatório sequencialmente e de uma única vez.
- Apesar dos resultados demonstrarem a ineficiência da versão paralela apresentada, é digno de destaque que essa versão, em termos conceituais, simplifica bastante o MBPO. Ao generalizarmos a fórmula do preço da opção, nós mostramos que é possível reduzir a uma única fórmula um modelo que envolve a descrição de inúmeras evoluções possíveis nos valores de duas variáveis (a opção e sua ação subjacente).

Capítulo 4

Método de Monte Carlo

Harrison [7] define o Método de Monte Carlo[1] como o conjunto de métodos numéricos que respeitam os seguintes passos:

1. Modelagem de um sistema como uma série de funções densidade de probabilidade (FDPs);
2. Obtenção de amostras da(s) FDP(s);
3. Cálculo das estatísticas de interesse a partir das amostras obtidas.

No universo das finanças, esses métodos são bastante populares e sua utilização data de décadas atrás: em 1964, David Hertz já discutia sua utilidade na área de finanças corporativas [8]. Alguns anos depois, em 1977, Boyle abordou a precificação de opções por meio do Método de Monte Carlo [3]. Até os dias de hoje, o método ainda é bastante popular na engenharia financeira.

Quando se trata da precificação de opções europeias, a Simulação de Monte Carlo parte de um princípio bastante similar ao assumido no Modelo Binomial de Precificação de Opções (tratado no capítulo 3): o movimento no preço de uma determinada ação segue um "passeio aleatório", no qual, em cada passo do tempo, o preço move-se certa taxa para cima ou para baixo.

Abaixo, apresentamos a fundamentação teórica por trás do modelo matemático que descreve esse passeio aleatório.

4.1 Hipóteses e o processo do preço da ação

A primeira suposição a ser feita é que a evolução do preço de uma ação segue um processo estocástico markoviano. Em termos simples, isso significa assumir que, a cada intervalo no tempo, o preço se move de forma aleatória, e que apenas o preço presente importa para se definir o preço do instante seguinte - de forma que os preços passados são irrelevantes.

[1] Também conhecido como Simulação de Monte Carlo.

Além disso, supõe-se que o retorno esperado da ação em um intervalo de tempo Δt é o mesmo, independentemente de seu preço (S). Isso significa que, para o intervalo citado, a variação da ação (ΔS) seria

$$\Delta S = \mu S \Delta t$$

onde μ é uma constante igual à taxa de retorno esperado para a ação. O caso acima, no entanto, pressupõe que os retornos da ação apresentam variância zero, o que sabemos não ser verdade. Para levar em conta esse fator, assume-se que a volatilidade percentual da ação é a mesma, qualquer que seja o preço atual. Com essas suposições, obtém-se:

$$\Delta S = \mu S \Delta t + \sigma S \epsilon \sqrt{\Delta t}$$

ou

$$S(t + \Delta t) - S(t) = \mu S(t) \Delta t + \sigma S(t) \epsilon \sqrt{\Delta t}$$

onde $S(t)$ é o preço da ação no instante t, $\hat{\mu}$ é seu retorno esperado, σ, sua volatilidade e ϵ, uma variável aleatória da distribuição normal $\mathcal{N}(0, 1)$ - sendo esta a FDP vinculada à variação do preço da ação ao longo do tempo. Note-se que o último termo da expressão, $\sigma S(t) \epsilon \sqrt{\Delta t}$, tem valor esperado zero, devido à média zero da distribuição normal associada. Isso quer dizer que, ao aumentarmos o número de passos no passeio aleatório (e repetirmos mais vezes o cálculo da fórmula acima), o componente da oscilação no preço devido à volatilidade do ativo tende a se anular e, portanto, o preço final tende ao valor esperado.

A equação acima, apesar de descrever corretamente o passeio aleatório, admite a possibilidade de que o preço da ação assuma um valor negativo, o que não é compatível com a realidade do mercado de ações. Para um modelo mais preciso, simula-se $\ln S$ em vez de S:

$$\ln S(t + \Delta t) - \ln S(t) = (\hat{\mu} - \tfrac{\sigma^2}{2}) \Delta t + \sigma \epsilon \sqrt{\Delta t}$$

Daí, segue que

$$S(t + \Delta t) = S(t) \exp\left[(\hat{\mu} - \tfrac{\sigma^2}{2}) \Delta t + \sigma \epsilon \sqrt{\Delta t}\right]$$

Esta última expressão [9] define o modelo de passeio aleatório empregado na Simulação de Monte Carlo deste capítulo.

4.2 Um exemplo

No Método de Monte Carlo, à diferença do Modelo Binomial tratado anteriormente, não são computados todos os caminhos possíveis para a ação. Em vez disso, é obtido um número suficiente de amostras que descrevem um processo completo do preço. Com todas as amostras calculadas, basta extrair a média aritmética dos preços finais e obter o *payoff* médio da opção. O valor presente desse *payoff* é o preço justo da opção.

A figura 4.1 ilustra uma simulação de 10 passeios aleatórios de uma ação sob as seguintes condições: $S_0 = 10$; $k = 10$; $t = 1$; $\mu = 0,15$; $\sigma = 0,075$; $n = 12$. A linha tracejada preta indica o caminho médio percorrido pelas amostras:

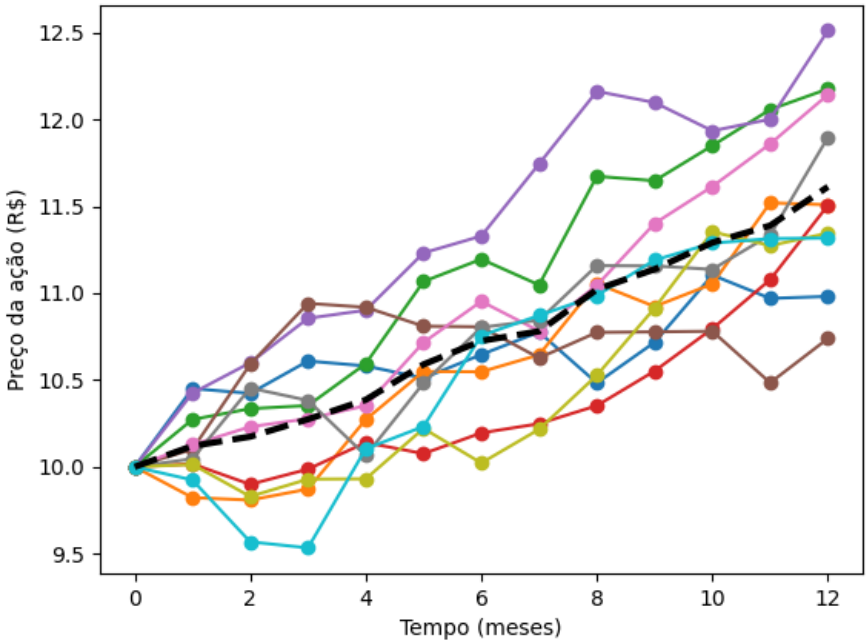

Figura 4.1: *Passeio aleatório da ação: 10 amostras*

No caso acima, o preço médio final da ação foi de R$11,61. Com esse dado, obtemos o valor presente do preço justo da opção de compra correspondente, dada uma taxa livre de risco hipotética de 3% ao ano:

$$(11,61 - 10) \times e^{(-0,03*1)} \approx 1,56$$

É válido pontuar que, mesmo com uma quantidade pequena de amostras, já se pode observar que o preço final médio calculado para as ações é próximo do valor esperado, dado por $S_0(1 + \mu T) = 10(1 + 0.15 \times 1) = 11,5$.

4.3 Implementação e Análise

A implementação da Simulação de Monte Carlo voltada à precificação de opções europeias é bastante simples. O algoritmo abaixo se resume à execução de m repetições de uma simulação do passeio aleatório percorrido pela ação, dividido em n passos. Faz-se uso do pacote *NumPy*, da linguagem *Python*, para as operações de geração da variável aleatória ϵ (descrita na seção acima), execução das simulações (de forma vetorizada) e cálculo da média dos preços finais. Além disso, assume-se que a opção europeia em questão é uma *call* (opção de compra), cujo *payoff* é igual a $f = \max(S - k, 0)$ (ou seja, a opção possui valor maior que zero quando seu preço de exercício é superior ao *spot* da ação no vencimento).

O código para a precificação de opções via Método de Monte Carlo é apresentado abaixo. O preço justo da opção, de acordo com a simulação, é retornado pela função

`monte_carlo_naive()`, que recebe como argumentos os valores de S, k, t, μ, σ, n, m e r.

```
1    import math
2    import numpy as np
3
4    def simulate(S, t, mu, sigma, n):
5       np.random.seed()
6       epsilons = np.random.normal(0, 1, n)
7       dt = t / (n - 1)
8
9       factors = np.vectorize(lambda eps: math.exp((mu - sigma * sigma / 2) * dt
             + sigma * eps * math.sqrt(dt)))
10
11      return S * np.prod(factors(epsilons))
12
13   def monte_carlo_naive(S, k, t, mu, sigma, n, m, r):
14      samples = np.array([simulate(S, t, mu, sigma, n) for _ in range(m)])
15      return max(np.mean(samples) - k, 0) * math.exp(-r * t)
```

Na Simulação de Monte Carlo para precificação de opções europeias, há duas variáveis determinantes para o tempo de execução do algoritmo: a quantidade m de amostras obtidas e o número n de passos percorridos por cada amostra. Uma vez que as outras operações matemáticas envolvidas em cada passo são constantes em relação à entrada, conclui-se, então, que o algoritmo é, no todo, $O(m \times n)$.

Para ilustrar melhor a influência de n e m no tempo de execução do algoritmo, segue um pequeno extrato das medições que serão apresentadas ao fim deste capítulo:

m	n	preço calculado (R$)	tempo (s)
10^3	2520	1,564	0,861
10^4	2520	1,562	8,650
10^4	30240	1,567	101,849

Tabela 4.1: Extrato das medições - versão sequencial

No primeiro experimento, com $n = 2520$ e $m = 10^3$, o tempo de execução é de 0,078 segundo. No caso seguinte, quando multiplicamos m por 10, o tempo medido é de 0,865 segundos - um valor aproximadamente 11 vezes maior que o do primeiro caso. Por fim, ao multiplicarmos n por 12, o tempo obtido é de 101,849 segundos. Como se pode observar, essas medições corroboram a complexidade do algoritmo apresentada acima.

4.4 Versão com Paralelismo

O conjunto dos algoritmos que envolvem Simulação de Monte Carlo é, por natureza, embaraçosamente paralelo. Em outras palavras, não há maiores dificuldades em repartir esses algoritmos em segmentos não dependentes que possam ser computados concomitantemente.

No algoritmo em questão, paraleliza-se a simulação dos m passeios aleatórios, com o intuito de tornar mais veloz a resolução do problema de precificação de opções por meio da Simulação de Monte Carlo.

Para isso, faz-se uso da biblioteca *multiprocessing* e adiciona-se o método *handler()*, que orquestra a execução paralela e retorna uma lista de todos os preços finais obtidos. O resto da implementação é idêntico à versão não-paralela.

```
1   import math
2   import numpy as np
3   import multiprocessing as mp
4
5   def simulate(S, t, mu, sigma, n):
6      np.random.seed()
7      epsilons = np.random.normal(0, 1, n)
8      dt = t / (n - 1)
9
10     factors = np.vectorize(lambda eps: math.exp((mu - sigma * sigma / 2) * dt
           + sigma * eps * math.sqrt(dt)))
11
12     return S * np.prod(factors(epsilons))
13
14  def handler(S, t, mu, sigma, n, m, pools):
15     pool = mp.Pool(pools)
16
17     samples = [pool.apply_async(simulate, args = (S, t, mu, sigma, n)) for _
           in range(m)]
18     return [s.get() for s in samples]
19
20  def monte_carlo_parallel(S, k, t, mu, sigma, n, m, r, pools):
21     samples = handler(S, t, mu, sigma, n, m, pools)
22     return max(np.mean(samples) - k, 0) * math.exp(-r * t)
```

4.5 Medições e Comparação

Nas medições realizadas, foram mantidos constantes os valores do preço inicial da ação (S), do preço de exercício da opção (k), do tempo transcorrido (t), da taxa livre de risco (r), da média de retorno anual do papel (μ) e da sua volatilidade anual (σ), uma vez que, conforme discutimos na seção anterior, esses parâmetros não influenciam o tempo de execução do algoritmo. Dessa maneira, apenas os valores do número de passos no tempo (n) e da quantidade de simulações realizadas (m) variaram nos experimentos, além do número de processos utilizados na versão paralela.

Assim, adotamos:

- Uma ação cujo preço inicial é de R$ 10 ($S = 10$);
- Uma opção de compra dessa ação com período de vencimento de um ano ($t = 1$) e preço de exercício de R$ 10 ($k = 10$);
- Taxa livre de risco de 3% ($r = 0,03$);
- Retorno esperado anual de 15% para a ação ($\mu = 0,15$);

- Volatilidade anual de 7,5% ($\sigma = 0,075$).

Os valores de m testados foram 10^2, 10^3 e 10^4, enquanto os de n foram 252 (representando os contumazes 252 dias em que há negociação de papéis nas bolsas), 2520 (ou seja, um passo no tempo a cada hora de negociação, supondo 10 horas diárias de pregão) e 30240 (um passo a cada cinco minutos). Para o número de processos do código com paralelismo, foram testados 1, 2 e 4 processos.

Seguem as medições realizadas em uma máquina dotada de 16 GB de memória RAM de um processador *Intel Core i7* de 4 núcleos:

m	n	preço (R$)	tempo (s)
10^2	252	1,545	0,016
10^3	252	1,531	0,171
10^4	252	1,582	1,810
10^2	2520	1,422	0,078
10^3	2520	1,564	0,861
10^4	2520	1,562	8,650
10^2	30240	1,567	0,972
10^3	30240	1,582	9,563
10^4	30240	1,567	101,849

Tabela 4.2: Versão sequencial

m	n	preço (R$)	tempo (s)
10^2	252	1,399	1,422
10^3	252	1,597	1,222
10^4	252	1,565	3,480
10^2	2520	1,432	1,108
10^3	2520	1,554	1,913
10^4	2520	1,568	10,306
10^2	30240	1,362	2,003
10^3	30240	1,589	10,973
10^4	30240	1,564	100,809

Tabela 4.3: Versão paralela - 1 processo

m	n	preço (R$)	tempo (s)
10^2	252	1,454	1,718
10^3	252	1,584	1,763
10^4	252	1,561	3,383
10^2	2520	1,624	1,613
10^3	2520	1,572	2,242
10^4	2520	1,565	7,611
10^2	30240	1,628	2,215
10^3	30240	1,541	7,869
10^4	30240	1,576	73,636

Tabela 4.4: Versão paralela - 2 processos

m	n	preço (R$)	tempo (s)
10^2	252	1,691	1,973
10^3	252	1,568	2,393
10^4	252	1,554	4,085
10^2	2520	1,551	2,623
10^3	2520	1,566	3,103
10^4	2520	1,574	7,156
10^2	30240	1,523	3,372
10^3	30240	1,564	6,852
10^4	30240	1,559	49,479

Tabela 4.5: Versão paralela - 4 processos

Para concluir este capítulo, algumas observações podem ser extraídas a partir dessas medições:

- De fato, conforme antecipamos na seção anterior, o tempo de execução do algoritmo parece ser linear tanto em relação a n como a m. Nas quatro condições em que foram realizadas medições, o tempo de execução, na grande maioria dos casos, aumenta conforme incrementamos essas duas variáveis;
- O uso de paralelismo é mais vantajoso para valores mais altos de m e n. Por exemplo, no caso de $m = 10^4$ e $n = 30240$ (o maior par testado), a versão sequencial levou quase 102 segundos para obter o preço justo da opção, ao passo que a versão paralela com 4 processos levou apenas 49 segundos. Para valores menores, por outro lado, a versão

sequencial é mais rápida. Isso se deve ao *overhead* (custo) gerado pelo gerenciamento dos processos paralelos.

- Os diferentes preços calculados variam pouco entre si, com uma variância de 0,00405. Boa parte dos *outliers* encontra-se nas medições cujo n foi 10^2, o que indica que esse é um valor para a quantidade de simulações que ainda não garante alta precisão.

Capítulo 5

Método das Diferenças Finitas

Os Métodos das Diferenças Finitas (MDF) compõem o conjunto de métodos numéricos voltados à resolução de equações diferenciais por meio da aproximação de derivadas por diferenças finitas. O uso desses métodos na matemática financeira foi introduzido por Eduardo Schwartz, em 1977 [13].

A equação diferencial de Black-Scholes-Merton, uma expressão matemática bastante conhecida no universo das finanças, estabelece uma igualdade que envolve o preço S de uma ação, o preço f de um derivativo qualquer dessa ação, a volatilidade da ação σ e a taxa livre de risco r [9]:

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} - rf = 0$$

Para realizar a precificação da opção, devemos discretizar essa equação diferencial, transformando-a em um conjunto de equações de diferença.

5.1 Hipóteses e precificação

Para discretizar a equação de Black-Scholes-Merton, assumimos uma malha com a seguinte configuração:

- **Eixo horizontal:** $n + 1$ pontos igualmente intervalados, representando os instantes no tempo 0, Δt, $2\Delta t$, ..., $n\Delta t = T$ (onde T é o instante de vencimento da opção);
- **Eixo vertical:** $m + 1$ pontos igualmente intervalados, representando os possíveis preços da ação subjacente 0, ΔS, $2\Delta S$, ..., $m\Delta t = S_{max}$ (onde S_{max} é um valor arbitrário para o preço da ação[1] tal que a opção, por definição, tem valor zero, qualquer que seja o instante de tempo).

[1] S_{max} deve ser escolhido de forma que o preço inicial da ação seja representado por um dos pontos do eixo vertical. Ou seja, é necessário haver um inteiro a entre 0 e m tal que $a\Delta S = S_0$, onde S_0 é o preço inicial da ação.

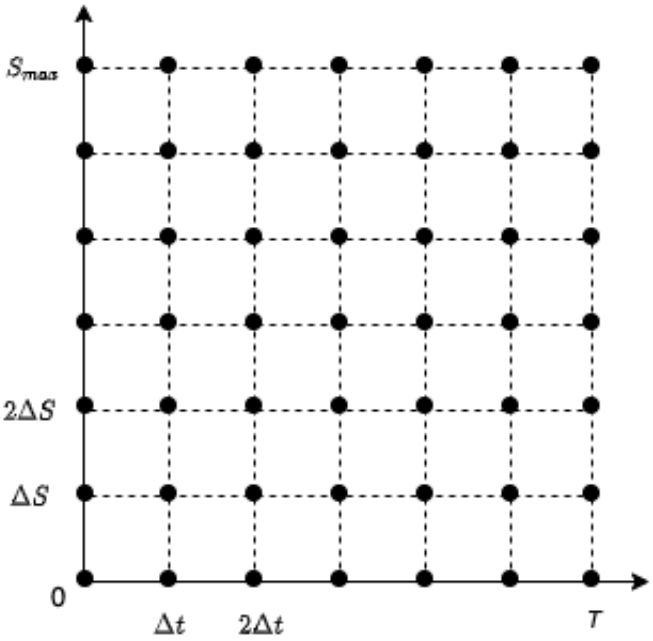

Figura 5.1: *Malha tempo-preço da ação*

Para determinar o preço justo da opção no instante $i\Delta t$ com a ação ao preço $j\Delta S$, ou seja, precificar $f_{i,j}$ em cada um dos pontos, deve-se, primeiro, estabelecer os valores de f nas bordas da malha. Para isso, definem-se algumas hipóteses compatíveis com os preços de opções **de venda** europeias[2] para as bordas:

- Quando $i = n$, a opção chegou ao vencimento e $f_{n,j} = \max(k - j\Delta S,\ 0)$, para qualquer j;
- Quando a ação tem preço 0, tem-se que $f_{i,0} = k\, e^{-r\,(n-i)}$, para qualquer i;
- Quando a ação chega ao preço máximo estabelecido (S_{max}), a opção não tem valor (por definição), ou seja, $f_{i,m} = 0$.

Com os valores das fronteiras definidos, resta calcular f nos outros pontos. Na literatura de matemática financeira, podem-se encontrar alguns métodos distintos voltados à realização desses cálculos[3]. No presente estudo, será utilizado o Método das Diferenças Finitas Explícito, em que $f_{i,j}$ é definido em termos de $f_{i+1,j-1}$, $f_{i+1,j}$ e $f_{i+1,j+1}$, de acordo com a seguinte igualdade[4] [5]:

$$f_{i,j} = \alpha_j f_{i+1,j-1} + \beta_j f_{i+1,j} + \gamma_j f_{i+1,j+1}$$

onde:

[2] Vale relembrar que, neste capítulo, serão abordadas apenas as opções de venda (*puts*). O Método de Diferenças Finitas, todavia, é aplicável também para *calls*, havendo apenas pequenas diferenças nas fórmulas de precificação, tanto para pontos de fronteira quanto para os interiores.

[3] Os mais notórios métodos de diferença voltados a esse fim são o Método Implícito, o Explícito e o de Crank-Nicolson [12].

[4] Foge do escopo deste trabalho a derivação matemática da igualdade descrita. Ela pode ser obtida por meio da aplicação das fórmulas das diferenças progressivas, regressivas e centradas, que nos permitem aproximar os valores de $\partial f/\partial S$, $\partial f/\partial t$ e $\partial^2 f/\partial S^2$.

$$\alpha_j = \frac{1}{2}\Delta t(\sigma^2 j^2 - rj)$$
$$\beta_j = 1 - \Delta t(\sigma^2 j^2 + r)$$
$$\gamma_j = \frac{1}{2}\Delta t(\sigma^2 j^2 + rj)$$

Esse sistema de equações pode ser representado matricialmente, como segue:

$$\begin{bmatrix} \beta_1 & \gamma_1 & 0 & \dots & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \dots & 0 \\ 0 & \alpha_3 & \beta_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \gamma_{m-2} \\ 0 & 0 & 0 & \alpha_{m-1} & \beta_{m-1} \end{bmatrix} \begin{bmatrix} f_{i+1,1} \\ f_{i+1,2} \\ f_{i+1,3} \\ \vdots \\ f_{i+1,m-1} \end{bmatrix} = \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ f_{i,3} \\ \vdots \\ f_{i,m-1} \end{bmatrix} - \begin{bmatrix} \alpha_1 f_{i+1,0} \\ 0 \\ 0 \\ \vdots \\ \gamma_{m-1} f_{i+1,m} \end{bmatrix}$$

Para calcular $f_{i,j}$, portanto, itera-se entre os diferentes níveis de i e, a cada nível, a precificação depende apenas de valores já conhecidos. Isso ocorre porque, devido às condições de borda pré-estabelecidas, os pontos tais que $i = n$ já têm o seu preço estabelecido. Assim, podemos precificar os pontos de $i = n - 1$, seguidos pelos de $i = n - 2$ (que dependem de $i = n - 1$), e assim por diante, até que se chegue a $i = 0$, o primeiro instante de tempo da malha.

Após a precificação da opção em todos os pares possíveis de i (tempo) e j (preço da ação), basta selecionar o par de interesse (usualmente, $i = 0$ e j tal que $j\Delta S$ equivale ao preço atual da ação). Assim, conclui-se a precificação da opção por meio do Método de Diferenças Finitas Explícito.

5.2 Implementação e Análise

O código abaixo implementa a precificação de uma opção de venda (*put*). A implementação pode ser dividida em algumas etapas:

1. Inicialização da malha de valores das opções, com o valor inicial de 0 arbitrariamente escolhido;
2. Precificação dos pontos de fronteira da malha, conforme descrito na seção anterior;
3. Precificação dos pontos interiores (mais bem descrita a seguir).

A precificação dos pontos interiores é realizada para cada instante de tempo, do maior para o menor ($n - 1$, $n - 2$, ..., 0). Ao começarmos com $i = n - 1$, conforme explicado na seção anterior, os valores dos pontos de $i = n$ já são conhecidos (por serem pontos de fronteira). Assim, aplica-se a fórmula matricial descrita acima e obtêm-se os pontos de $i = n - 1$. Depois, sabendo os valores de de $i = n - 1$, segue-se para $i = n - 2$, e assim por diante, até que se precifiquem os pontos de $i = 0$.

Os parâmetros de entrada do algoritmo são S (preço inicial da ação), $Smax$ (preço máximo da ação), t (tempo em anos até o vencimento da opção), k (preço de exercício da opção), r (taxa livre de risco anual), σ (volatilidade anual da ação), m (quantidade de preços distintos que a ação pode assumir) e n (quantidade de intervalos no tempo até o vencimento).

```
1    import numpy as np
2
3    def create_grid(m, n, Smax, k, r, t):
4       grid = np.zeros([m+1, n+1])
5       stock_prices = np.linspace(0, Smax, m + 1)
6       dt = t / n
7
8       grid[:, -1] = np.maximum(k - stock_prices, 0)
9       grid[0, 0:n] = k * np.exp(-r * dt * (n - np.arange(n)))
10
11      return grid
12
13   def get_coefficients(m, t, n, sigma, r):
14      dt = t / n
15      j_vector = np.arange(m)
16
17      alphas = 0.5 * dt * (sigma**2 * j_vector**2 - r * j_vector)
18      betas = 1 - dt * (sigma**2 * j_vector**2 + r)
19      gammas = 0.5 * dt * (sigma**2 * j_vector**2 + r * j_vector)
20
21      coefficients_matrix = np.diag(alphas[2:m], -1) + np.diag(betas[1:m]) + np.
            diag(gammas[1:m-1], 1)
22
23      return coefficients_matrix, alphas, gammas
24
25   def price_option(S, Smax, t, k, r, sigma, m, n):
26      grid = create_grid(m, n, Smax, k, r, t)
27      coefficients_matrix, alphas, gammas = get_coefficients(m, t, n, sigma, r)
28
29      factors = np.zeros(m-1)
30
31      for i in reversed(range(n)):
32         factors[0] = alphas[1] * grid[0, i+1]
33         factors[-1] = gammas[m - 1] * grid[m, i+1]
34
35         grid[1:m, i] = np.add(np.dot(coefficients_matrix, grid[1:m, i+1]),
               factors)
36
37      S_index = m * S // Smax
38      return grid[S_index, 0]
```

A análise da complexidade do algoritmo acima requer uma descrição da complexidade de alguns dos seus passos.

A inicialização da malha, representada pela função `create_grid()`, é da ordem de $O(m \times n)$, bem como a obtenção da matriz de coeficientes (`get_coefficients()`). Após a chamada dessas funções, itera-se entre os diferentes instantes de tempo (n vezes) e, a cada iteração, há uma operação de multiplicação de matriz por vetor e uma soma de vetores. Uma vez que a matriz de coeficientes tem dimensão $m \times m$, multiplicá-la por um vetor de dimensão m tem complexidade de $O(m^2)$. Isso nos diz que, no código acima, realizam-se n vezes uma operação $O(m^2)$. Portanto, pode-se concluir a complexidade do algoritmo como sendo de $O(m^2 \times n)$.

5.3 Versão Paralela

Assim como no caso do Modelo Binomial de Precificação de Opções, os Métodos de Diferenças envolvem uma forte dependência entre os dados de cada ponto de precificação, o que dificulta a implementação de versões paralelas eficientes. Assim, devemos buscar alternativas de paralelização que permitam, ainda que de forma limitada, o emprego simultâneo de múltiplos processos.

Na versão sequencial, as operações matriciais de soma e multiplicação são realizadas por meio das funções `dot()` e `add()`, da biblioteca *NumPy*. Essas funções contam com inúmeras otimizações próprias da biblioteca que as tornam bastante eficientes. No entanto, não há uso explícito de paralelismo durante o cômputo das operações - isto é, em boa parte dos casos, não se utilizam múltiplos processos simultâneos para calculá-las[5]. Pela natureza dessas operações, é evidente que elas podem ser implementadas de forma paralela. Para clarificar, vejamos o caso da multiplicação de matriz por vetor:

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 \times b_1 + a_2 \times b_2 \\ a_3 \times b_1 + a_4 \times b_2 \end{bmatrix}$$

Note-se que os elementos do vetor resultante são independentes entre si, o que permite que os cálculos sejam executados concomitantemente. Para isso, utiliza-se a biblioteca *Numba*, descrita em mais detalhes no capítulo 3. Por meio do uso do decorador `@njit`, ativa-se a funcionalidade de "paralelismo automático" - ou seja, define-se que o *Numba* deve executar as expressões do *Numpy* paralelamente.

Abaixo, há a implementação da versão paralela do Método das Diferenças Finitas Explícito para precificação de opções de venda. Essencialmente, a única mudança significativa para a versão sequencial é o uso do decorador `@njit` para as chamadas das funções `add()` e `dot()`, bem como a definição do número de processos utilizados (sendo este um parâmetro de entrada do algoritmo).

```
1    import numpy as np
2    from numba import njit, set_num_threads
3
4    def create_grid(m, n, Smax, k, r, t):
5       grid = np.zeros([m+1, n+1])
6       stock_prices = np.linspace(0, Smax, m + 1)
7       dt = t / n
8
9       grid[:, -1] = np.maximum(k - stock_prices, 0)
10      grid[0, 0:n] = k * np.exp(-r * dt * (n - np.arange(n)))
11
12      return grid
13
```

[5] A documentação da biblioteca SciPy [14] afirma que, a depender da arquitetura da máquina, é possível que os BLAS (*Basic Linear Algebra Subprograms*) façam uso de paralelismo para computar a função `dot()`. Todavia, isso não é definido pelo usuário/programador e não há qualquer tipo de controle sobre o número de processos paralelos utilizados. Assim, neste trabalho, considera-se que o código que emprega essas funções sem definição explícita de paralelismo se configura como uma implementação sequencial.

```
14    def get_coefficients(m, t, n, sigma, r):
15       dt = t / n
16       j_vector = np.arange(m)
17
18       alphas = 0.5 * dt * (sigma**2 * j_vector**2 - r * j_vector)
19       betas = 1 - dt * (sigma**2 * j_vector**2 + r)
20       gammas = 0.5 * dt * (sigma**2 * j_vector**2 + r * j_vector)
21
22       coefficients_matrix = np.diag(alphas[2:m], -1) + np.diag(betas[1:m]) + np.
             diag(gammas[1:m-1], 1)
23
24       return coefficients_matrix, alphas, gammas
25
26    @njit(parallel=True)
27    def solve(coefficients_matrix, grid, factors, m, i):
28       return np.add(np.dot(coefficients_matrix, grid[1:m, i+1]), factors)
29
30    def price_option(S, Smax, t, k, r, sigma, m, n, pools):
31       set_num_threads(pools)
32       grid = create_grid(m, n, Smax, k, r, t)
33       coefficients_matrix, alphas, gammas = get_coefficients(m, t, n, sigma, r)
34
35       factors = np.zeros(m-1)
36
37       for i in reversed(range(n)):
38          factors[0] = alphas[1] * grid[0][i+1]
39          factors[-1] = gammas[m - 1] * grid[m][i+1]
40
41          grid[1:m, i] = solve(coefficients_matrix, grid, factors, m, i)
42
43       S_index = m * S // Smax
44       return grid[S_index][0]
```

5.4 Medições e Comparação

Conforme explicado nas seções anteriores, são duas as variáveis que afetam a dimensão da malha de tempo-preços e, portanto, do tempo de execução do algoritmo do modelo de diferenças finitas: m e n - que, respectivamente, representam a quantidade de possíveis preços que a ação pode assumir e o número de passos no tempo considerados até o vencimento da opção.

Os valores fixos adotados nas medições foram:

- Preço da ação de R$ 50 ($S = 50$);
- Preço máximo da ação de R$ 100 ($S_{max} = 100$);
- Preço de exercício da opção de R$ 50 ($k = 50$);
- Tempo de vencimento da opção de um ano ($t = 1$).
- Taxa livre de risco de 12% ao ano ($r = 0,12$);
- Volatilidade anual da ação de 30% ($\sigma = 0,3$);

No mais, os valores testados para m foram 25, 50 e 100, ao passo que, para n, utilizaram-se 10^3, 10^4 e 10^5 - o que totaliza 9 combinações distintas do par (m, n).

Assim como nos modelos tratados nos capítulos anteriores, as medições foram realizadas em uma máquina dotada de 16 GB de memória RAM e de um processador *Intel Core i7* de 4 núcleos. Os valores observados foram:

m	n	preço (R$)	tempo (s)
25	10^3	3.861	$4,3 \times 10^{-3}$
50	10^3	3.225	$6,8 \times 10^{-3}$
100	10^3	3.235	$4,1 \times 10^{-2}$
25	10^4	3.861	$2,4 \times 10^{-2}$
50	10^4	3.225	$3,2 \times 10^{-2}$
100	10^4	3.235	0,339
25	10^5	3.861	0,244
50	10^5	3.225	0,278
100	10^5	3.235	3,403

Tabela 5.1: Versão sequencial

m	n	preço (R$)	tempo (s)
25	10^3	3.861	$5,1 \times 10^{-3}$
50	10^3	3.225	$5,8 \times 10^{-3}$
100	10^3	3.235	$1,7 \times 10^{-2}$
25	10^4	3.861	$3,2 \times 10^{-2}$
50	10^4	3.225	$4,8 \times 10^{-2}$
100	10^4	3.235	0,103
25	10^5	3.861	0,340
50	10^5	3.225	0,473
100	10^5	3.235	1,138

Tabela 5.2: Versão paralela - 1 processo

m	n	preço (R$)	tempo (s)
25	10^3	3.861	$6,9 \times 10^{-3}$
50	10^3	3.225	$6,6 \times 10^{-3}$
100	10^3	3.235	$1,6 \times 10^{-2}$
25	10^4	3.861	$5,6 \times 10^{-2}$
50	10^4	3.225	$6,4 \times 10^{-2}$
100	10^4	3.235	0,121
25	10^5	3.861	0,519
50	10^5	3.225	0,623
100	10^5	3.235	1,093

Tabela 5.3: Versão paralela - 2 processos

m	n	preço (R$)	tempo (s)
25	10^3	3.861	$8,2 \times 10^{-3}$
50	10^3	3.225	$8,6 \times 10^{-3}$
100	10^3	3.235	$1,6 \times 10^{-2}$
25	10^4	3.861	0,101
50	10^4	3.225	0,132
100	10^4	3.235	0,137
25	10^5	3.861	0,914
50	10^5	3.225	1,081
100	10^5	3.235	1,380

Tabela 5.4: Versão paralela - 4 processos

As medições acima nos permitem tirar algumas conclusões:

- Assim como no caso do Modelo Binomial, tratado no capítulo anterior, as execuções do algoritmo de Diferenças Finitas respeitam a propriedade de que, para os mesmos parâmetros de entrada, o resultado obtido para o preço da opção deve ser o mesmo. Conforme comentado anteriormente, dentre os métodos numéricos e algoritmos abordados neste trabalho, somente a Simulação de Monte Carlo não é determinística e, portanto, gera resultados diferentes para a mesma entrada.
- Apenas com base nas medições da versão sequencial, não é possível inferir a complexidade de $O(m^2 \times n)$ do algoritmo. Para n, pode-se observar a proporção linear com o tempo de execução - por exemplo, do par ($m = 100$, $n = 10^4$) para ($m = 100$, $n = 10^5$), o experimento leva 10,03 vezes mais tempo para rodar -. No entanto, para m, a relação quadrática não é evidente. Isso se deve, muito provavelmente, às otimizações do *NumPy* nas operações matriciais de soma e multiplicação. Ainda que a complexidade computacional não se altere, as relevantes economias de tempo

obtidas obscurecem a relação entre o tamanho das matrizes e o tempo de execução das operações sobre elas.

- Comparando-se os tempos de execução entre as diferentes implementações, percebe-se que a versão paralela se mostrou mais eficiente nos casos em que $m = 100$ - ou seja, o paralelismo é vantajoso para m grande, independentemente do valor de n. Isso não surpreende, uma vez que a paralelização do código se dá justamente sobre as operações matriciais, que dependem apenas de m. Para os casos em que m é pequeno, o *overhead* advindo do gerenciamento de múltiplos processos excede o ganho obtido com os cômputos concomitantes sobre as matrizes. Para o maior par testado, ($m = 100$, $n = 10^5$), a execução da versão sequencial levou 3,4 segundos para rodar, ao passo que no código paralelo com dois processos (que obteve tempo menor que o de quatro), o tempo medido foi pouco superior a 1 segundo.

Capítulo 6

Conclusão

Conforme descrito no capítulo introdutório, o principal objetivo deste trabalho é determinar, para os três modelos de precificação de opções europeias implementados, em quais deles a versão paralela se mostra mais eficiente que a sequencial. Em outras palavras, queremos avaliar o *trade-off* entre o *overhead* do gerenciamento de múltiplas tarefas concomitantes e o ganho de tempo que se obtém com o cômputo simultâneo de operações do algoritmo.

Dessa forma, neste capítulo dedicado às conclusões do trabalho, o foco residirá nas comparações do tempo de execução entre as versões sequenciais e paralelas de cada modelo. Além disso, serão relembrados os principais pontos de conclusão dos capítulos anteriores, no que tange a aspectos dos métodos de precificação e de suas implementações.

Por fim, com base nos tempos medidos entre os diferentes modelos e versões, comentar-se-á sobre a viabilidade da utilização das implementações desenvolvidas em "situações do mundo real" - isto é, se seria possível que um *player* do mercado as utilizasse para precificar opções e, com base na precificação, decidisse quais operações realizar.

6.1 Modelo Binomial

Explica-se, no capítulo relativo a esse método, que, devido à forte dependência entre os preços da opção em diferentes nós da árvore binomial, seria necessário visualizar o Modelo Binomial sob uma ótica distinta para paralelizar um algoritmo que o representasse. Com isso em mente, a solução encontrada foi expandir a fórmula de precificação de um nó pai, que originalmente considera apenas os dois nós filhos, para abranger todos os seus descendentes do último nível da árvore. Dessa forma, ao precificar-se o nó raiz, consideramos os seus n descendentes terminais, onde n é, também, a profundidade da árvore binomial. O paralelismo dá-se sobre a segmentação do somatório de n fatores e o cômputo simultâneo de múltiplos segmentos.

As medições dos algoritmos do modelo binomial permitiram a extração de duas conclusões principais.

A primeira é que, conforme se havia antecipado na análise do modelo, a complexidade

de $O(n^2)$ torna-se bastante clara quando se comparam os tempos de execução com os diferentes parâmetros de entrada testados - principalmente para valores maiores de n. Isto é, mesmo se não se houvesse realizado a análise do algoritmo, poder-se-ia deduzir, com evidências empíricas, a complexidade do modelo como sendo proporcional ao quadrado de n.

A segunda conclusão, a mais relevante, é que, para as implementações realizadas, a paralelização do Modelo Binomial é claramente menos eficiente (em termos de tempo de execução) que seu análogo sequencial. Para todos os valores de entrada testados, a versão sequencial tem um tempo de execução menor que a paralela. Além disso, quando comparamos os diferentes números de processos paralelos testados, é evidente que o *overhead* gerado pelo seu gerenciamento ultrapassa os ganhos de tempo que se poderia obter com os cômputos simultâneos, uma vez que, com mais processos rodando, os tempos aumentam.

O gráfico a seguir, que sintetiza os dados das tabelas presentes no capítulo do Modelo Binomial, clarifica os pontos delineados acima:

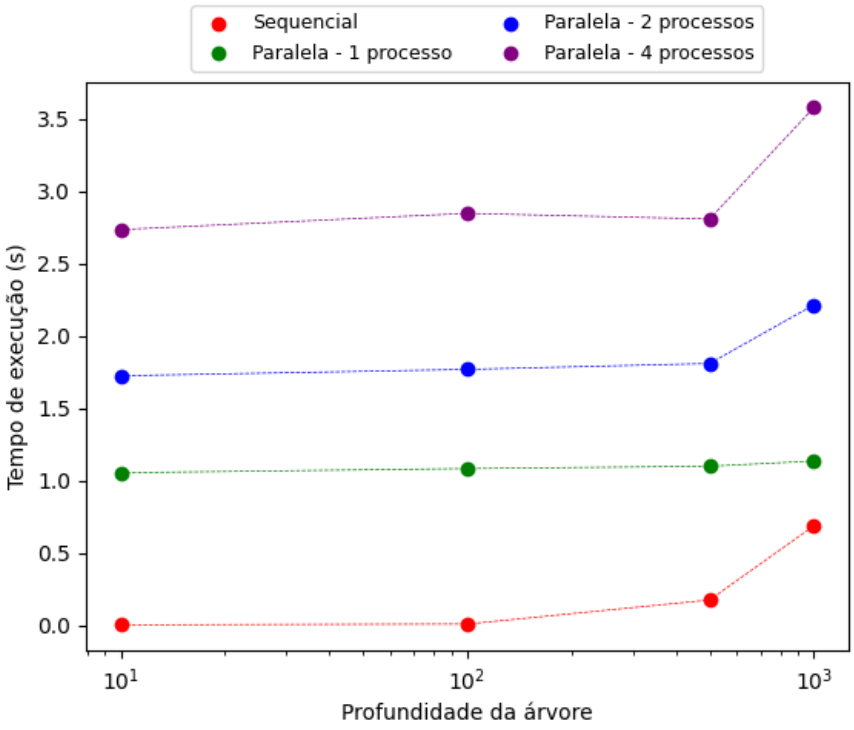

Figura 6.1: *Modelo Binomial - medições*

6.2 Método de Monte Carlo

No capítulo dedicado à Simulação de Monte Carlo, esclarece-se que esse método é "embaraçosamente paralelo", ou seja, ele pertence à classe dos algoritmos que são facilmente

paralelizados. De fato, uma vez que, neste estudo, se simula o passeio aleatório de uma ação m vezes, é inequívoco que as simulações são totalmente independentes entre si, o que garante que elas possam ser calculadas simultaneamente sem maiores dificuldades.

Sendo assim, devido à natureza paralelizável do algoritmo, a expectativa prévia era de que, no método de Monte Carlo, o paralelismo se mostrasse mais eficiente que seu análogo sequencial - provavelmente, por um fator proporcional ao número de processos adotados[1].

Após a implementação dos códigos e a realização das medições para diferentes parâmetros de entrada, puderam-se concluir, assim como no Modelo Binomial, dois pontos principais.

O primeiro refere-se à corroboração dos tempos medidos com a complexidade de $O(m \times n)$ - onde m é a quantidade de simulações e n é o número de passos no tempo percorridos - descrita na seção de implementação do modelo. De fato, acréscimos em n e m resultam em aumentos linearmente proporcionais no tempo de execução. Para evidenciar essa questão, trazemos novamente a tabela 4.1, que contém um pequeno extrato das medições:

m	n	preço calculado (R$)	tempo (s)
10^3	2520	1,564	0,861
10^4	2520	1,562	8,650
10^4	30240	1,567	101,849

Tabela 4.1: Extrato das medições - versão sequencial

Note-se que aumentos de n e m por fatores 10 e 12 levam a tempos de execução aproximadamente 10 e 12 vezes maiores, respectivamente.

O segundo ponto diz respeito à eficiência da paralelização do código. Conforme havia sido previsto, a implementação paralela obteve tempos de execução consideravelmente menores que aqueles da versão sequencial para valores "grandes" de n e m. Com valores menores, o *overhead* advindo do gerenciamento de processos paralelos supera o ganho de tempo obtido com a execução concomitante de simulações, mas, quando se aumentam n e m, a vantagem de utilização da versão paralela torna-se clara. Por outro lado, o tempo de execução não se mostra inversamente proporcional ao número de processos. Por exemplo, com $m = 10^4$ e $n = 30240$, o tempo medido para a execução paralela com quatro processos (49,5 segundos) foi aproximadamente metade - e não um quarto - do observado na medição de um processo (100,8 segundos).

Os gráficos abaixo apresentam os tempos de execução de algumas das medições realizadas. No primeiro gráfico, constam as medições em que $n = 30240$, com m variável (no eixo x). O segundo contém as execuções em que $m = 10^4$, com n variável. Ambas as figuras resumem os pontos discutidos acima.

[1] Isto é, para quatro processos, por exemplo, esperar-se-ia um tempo de execução aproximadamente quatro vezes menor.

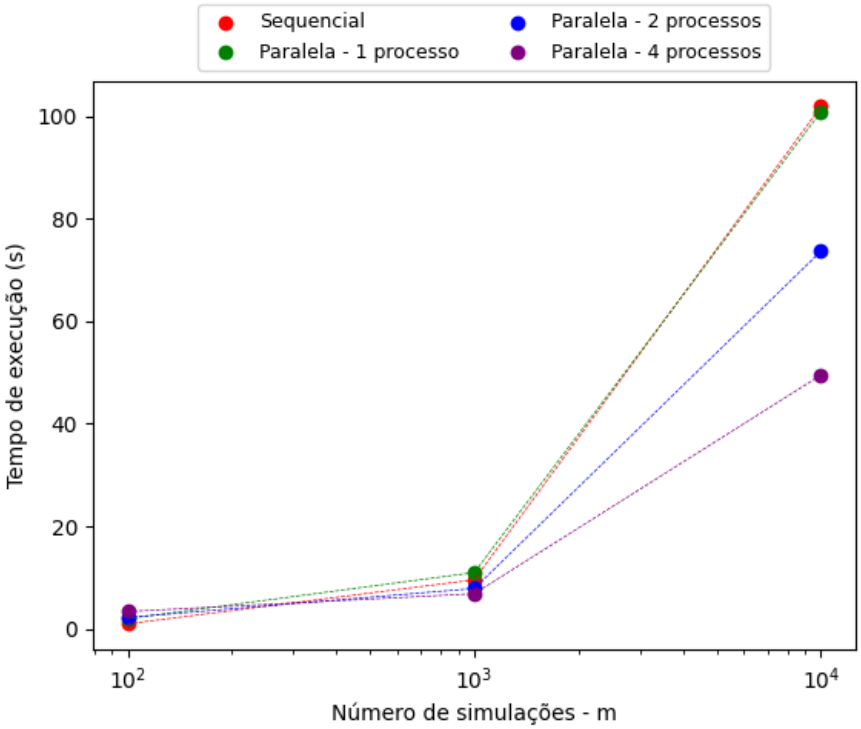

Figura 6.2: *Método de Monte Carlo - medições para* $n = 30240$

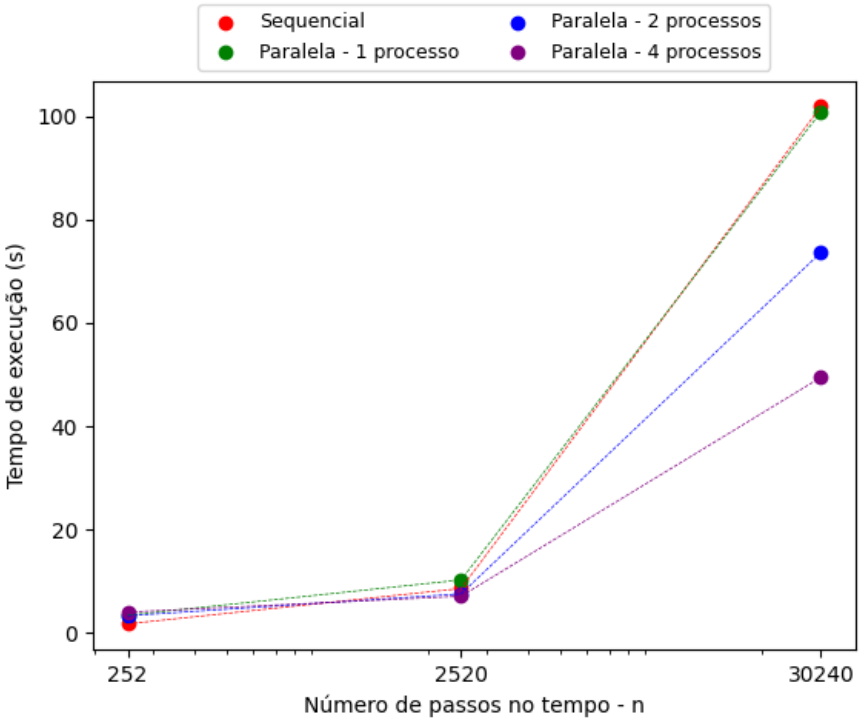

Figura 6.3: *Método de Monte Carlo - medições para* $m = 10^4$

6.3 Método das Diferenças Finitas

Conforme explicado no capítulo anterior, relativo à precificação de opções pelo Método das Diferenças Finitas Explícito, o sistema de equações a partir dos quais os pontos da malha tempo-preços são precificados pode ser representado por uma equação matricial. A implementação do método aproveita essa representação.

Na primeira versão implementada, as operações de multiplicação de matriz por vetor e soma de vetores são realizadas de forma sequencial, por meio das funções `add()` e `dot()`, do *NumPy*. A versão paralela, por outro lado, emprega a funcionalidade de paralelismo automático da biblioteca *Numba*, que garante a execução das funções supracitadas de forma paralela e otimizada. Dessa maneira, o cálculo dos elementos dos vetores resultantes é realizado simultaneamente.

Sendo assim, dois fatores contribuíram para que, antes da realização das medições, a expectativa fosse de que a versão paralela apresentasse tempos melhores que a sequencial: (i) o cômputo simultâneo de múltiplos elementos do vetor resultante da multiplicação e (ii) as otimizações da operação advindas do uso da biblioteca *Numba*.

Após as medições, constatou-se que a expectativa se mostrou parcialmente correta. Os tempos obtidos com a versão paralela foram melhores que os da sequencial apenas nos casos em que $m = 100$ (sendo esse o maior valor testado para a variável). A conclusão, pois, é que o paralelismo é vantajoso para m grande, qualquer que seja o valor de n. Uma vez que a paralelização do código ocorre justamente nas operações sobre as matrizes, cujo tamanho depende unicamente de m, é compreensível que seja esta a variável determinante para a efetividade do código paralelo.

Além disso, os tempos medidos na versão sequencial não evidenciam com clareza a complexidade de $O(m^2 \times n)$ do algoritmo. Conforme se discutiu no capítulo 5, a relação linear entre n e o tempo de execução pôde ser detectada, mas o mesmo não ocorreu com a relação quadrática de m, devido às otimizações nas funções do *NumPy* relativas às operações matriciais.

Mais uma vez, os gráficos a seguir clarificam as conclusões traçadas.

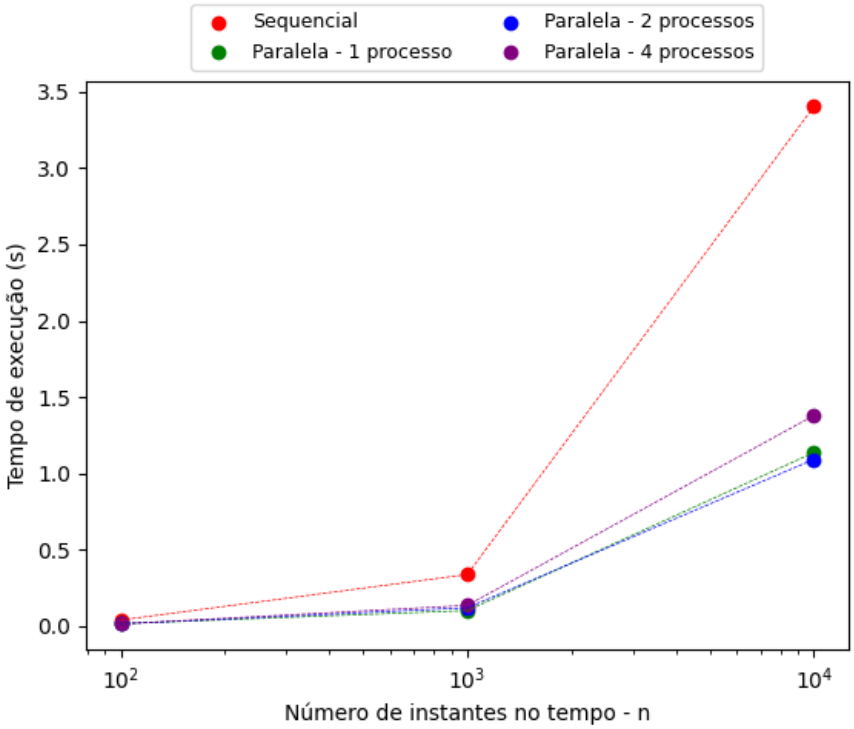

Figura 6.4: *Método das Diferenças Finitas - medições para* $m = 100$

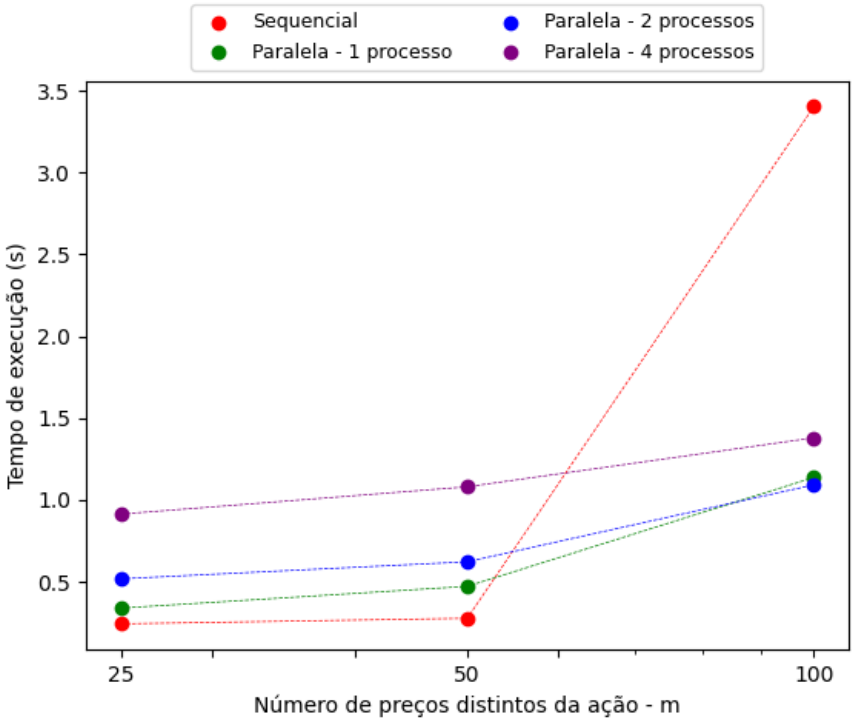

Figura 6.5: *Método de Monte Carlo - medições para* $n = 10^5$

6.4 Conclusões finais

As análises das implementações e medições dos três modelos de precificação de opções abordados permitem-nos responder à principal pergunta deste trabalho: *para quais modelos a versão paralela se mostra mais vantajosa?*

Com base nos tempos medidos, conclui-se que:

- No Modelo Binomial, a versão sequencial obteve tempos menores para todos os parâmetros testados e, portanto, o código paralelo mostrou-se ineficiente;
- No Método de Monte Carlo, principalmente para n e m grandes, os tempos medidos na versão paralela foram menores que os da sequencial, o que nos permite concluir que ela cumpriu o seu propósito de tornar a execução mais veloz;
- No Método das Diferenças Finitas, a paralelização por meio do uso da biblioteca *Numba* acelerou o tempo de execução do código para os casos em que m (que representa o número de preços distintos para ação e a dimensão das matrizes trabalhadas) era grande, mas a versão sequencial foi mais rápida para m pequeno (menor que 100).

Sendo assim, a versão paralela se mostra mais vantajosa para a Simulação de Monte Carlo e, a depender dos parâmetros adotados, no Método das Diferenças Finitas, ao passo que, no Modelo Binomial, a versão sequencial foi mais eficiente.

Por fim, foi levantada no começo deste capítulo a possibilidade da utilização dos programas implementados neste trabalho por *players* do mercado - isto é, por indivíduos e empresas que operam opções europeias e desejam precificá-las em tempo real. Para que se aborde essa questão em sua completude, é necessário levar alguns pontos distintos em consideração.

O primeiro refere-se à dificuldade da obtenção de um dos parâmetros de entrada dos modelos: a volatilidade da ação subjacente, representada pela letra grega σ. Dado que queremos precificar a opção para o futuro, utilizar a volatilidade realizada de um período recente não parece ser a abordagem mais adequada, uma vez que ela falha em capturar as expectativas futuras dos movimentos nos preços. Vejamos um exemplo simples: nas semanas seguintes aos ataques de 11 de setembro, a volatilidade de ações de empresas aéreas sofreu um forte aumento. Se, poucos dias após o evento, se precificassem as opções utilizando a volatilidade realizada dos últimos 60 dias, certamente o valor empregado seria muito menor que o real.

Por esse motivo, uma prática comum no mercado é o cálculo da volatilidade implícita dos ativos - a volatilidade que pode ser inferida a partir dos preços de suas opções. Ou seja, na prática, determina-se a volatilidade com base no preço da opção, e não o contrário. De qualquer forma, a volatilidade implícita é um dado que pode ser utilizado para, juntamente com os outros parâmetros de entrada, precificar uma opção por meio dos modelos abordados neste trabalho.

No tocante aos tempos de execução, é certo que os algoritmos implementados atenderiam às necessidades de um *player* do mercado. Nas medições realizadas, boa parte dos parâmetros testados é maior do que os valores que seriam efetivamente utilizados.

Dificilmente, negocia-se uma opção com vencimento superior a 90 dias. Além disso, a frequência dos preços da ação (no caso dos passeios aleatórios que discutimos) empregada provavelmente seria a diária. Por fim, apenas um intervalo curto de diferentes preços para a ação seria necessário, uma vez que variações superiores a, diga-se, 40% em um período de 90 dias são incomuns. Logo, pode-se afirmar que, em termos práticos, os algoritmos poderiam ser utilizados.

Por outro lado, ainda em termos de tempo de execução, os programas implementados não seriam capazes de competir com as ferramentas de precificação dos serviços e plataformas voltados ao mercado financeiro existentes nos dias de hoje. Naturalmente, as empresas por trás dessas plataformas possuem poder computacional, *expertise* e anos de otimizações em seus produtos que os tornam incomparáveis com a implementação de um trabalho de graduação. Assim, conclui-se que, apesar de possível, seria improvável que os algoritmos fossem de fato postos a uso - o que de forma alguma desmerece as implementações feitas neste trabalho, cujo propósito era mais a realização de um exercício acadêmico que a sua efetiva adoção por *practitioners*.

Referências

[1] *Automatic parallelization with @jit*. URL: https://numba.readthedocs.io/en/stable/user/parallel.html (acesso em 09/12/2021) (ver p. 14).

[2] Söhnke M. Bartram, Gregory W. Brown e Jennifer Conrad. "The Effects of Derivatives on Firm Risk and Value". Em: *Journal of Financial and Quantitative Analysis* 46.4 (fev. de 2010), pp. 967–999. URL: https://ssrn.com/abstract=1550942 (ver p. 1).

[3] Phelim P. Boyle. "Options: A Monte Carlo approach". Em: *Journal of Financial Economics* 4.3 (1977), pp. 323–338. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(77)90005-8. URL: https://www.sciencedirect.com/science/article/pii/0304405X77900058 (ver p. 25).

[4] John C. Cox, Stephen A. Ross e Mark Rubinstein. "Option pricing: A simplified approach". Em: *Journal of Financial Economics* 7.3 (1979), pp. 229–263. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(79)90015-1. URL: https://www.sciencedirect.com/science/article/pii/0304405X79900151 (ver pp. 3, 15, 18).

[5] Sargo Danho. "Pricing Financial Derivatives with the Finite Difference Method". Estocolmo, Suécia: KTH Royal Institute of Technology, 2017 (ver p. 34).

[6] The Economist. "The stockmarket is now run by computers, algorithms and passive managers". Em: *The Economist* (5 de out. de 2019). URL: https://www.economist.com/briefing/2019/10/05/the-stockmarket-is-now-run-by-computers-algorithms-and-passive-managers (acesso em 23/06/2021) (ver p. 3).

[7] Robert Harrison. "Introduction To Monte Carlo Simulation". Em: *AIP conference proceedings* 1204 (2010), pp. 17–21. DOI: 10.1063/1.3295638 (ver p. 25).

[8] David Hertz. "Risk Analysis in Capital Investment". Em: *Harvard Business Review* (out. de 1964), pp. 169–181 (ver p. 25).

[9] John C. Hull. *Options, futures, and other derivatives*. 6. ed., Pearson internat. ed. Upper Saddle River, NJ [u.a.]: Pearson Prentice Hall, 2006. XXII, 789. ISBN: 978-0-13-197705-1. URL: http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+563580607&sourceid=fbw_bibsonomy (ver pp. 1, 15, 26, 33).

[10] Harry Markowitz. *Portfolio Selection: efficient diversification of investments*. New York: John Wiley & Sons, Inc., 1959. ISBN: 0300013728 (ver p. 3).

[11] *multiprocessing — Process-based parallelism*. URL: https://docs.python.org/3/library/multiprocessing.html (acesso em 28/11/2021) (ver p. 13).

[12] Dana Rose-Anne e Monique JeanBlanc. *Financial Markets in Continuous Time*. New York: Springer, 2007. ISBN: 978-3-540-71149-0 (ver p. 34).

[13] Eduardo S. Schwartz. "The valuation of warrants: Implementing a new approach". Em: *Journal of Financial Economics* 4.1 (jan. de 1977), pp. 79–93. URL: https://ideas.repec.org/a/eee/jfinec/v4y1977i1p79-93.html (ver p. 33).

[14] Scipy. *Parallel Programming with numpy and scipy*. URL: https://scipy.github.io/old-wiki/pages/ParallelProgramming (acesso em 22/06/2021) (ver p. 37).

[15] European Securities e Markets Authority. *ESMA DATA ANALYSIS VALUES EU DERIVATIVES MARKET AT €660 TRILLION WITH CENTRAL CLEARING INCREASING SIGNIFICANTLY*. URL: https://www.esma.europa.eu/press-news/esma-news/esma-data-analysis-values-eu-derivatives-market-%E2%82%AC660-trillion-central-clearing (acesso em 19/08/2021) (ver p. 2).

[16] Edward Thorpe. "A Perspective on Quantitative Finance: Models for Beating the Market". Em: *Wilmott Magazine* (2003), pp. 33–38 (ver p. 3).