Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Trabalho de Conclusão de Curso

O ENSINO DE FÍSICA COMPUTACIONAL POR MEIO DA SOLUÇÃO DE PROBLEMAS UTILIZANDO O MÉTODO DE MONTE CARLO

Acadêmico: Gustavo Meneghetti Arcolezi

Orientador: Prof. Dr. Guilherme Maia Santos

Coorientador: Prof. Dr. Breno Ferraz de Oliveira

Maringá, 2022

Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Trabalho de Conclusão de Curso

O ENSINO DE FÍSICA COMPUTACIONAL POR MEIO DA SOLUÇÃO DE PROBLEMAS UTILIZANDO O MÉTODO DE MONTE CARLO

Trabalho de Conclusão de Curso apresentado ao Departamento de Física da Universidade Estadual de Maringá, sob orientação do Prof. Dr. Guilherme Maia Santos e do Prof. Dr. Breno Ferraz de Oliveira, como parte dos requisitos para obtenção do grau de Licenciado em Física.

Acadêmico: Gustavo Meneghetti Arcolezi

Orientador: Prof. Dr. Guilherme Maia Santos

Coorientador: Prof. Dr. Breno Ferraz de Oliveira

Maringá, 2022

Sumário

Agradecimentos

Meus sinceros agradecimentos aos Professores Dr. Guilherme Maia Santos e Dr. Breno Ferraz de Oliveira pela orientação.

À Universidade Estadual de Maringá pela estrutura oferecida para realização do curso de física.

Aos meus amigos que fiz durante a graduação e levo para a vida: Eduardo, Mariana, Matheus e Vinicius.

À minha namorada, que incansavelmente me dá forças e apoio para alcançar meus objetivos. Para ela todo meu amor.

E por último, mas não menos importante, gostaria de agradecer aos meus pais pelo apoio contínuo e por serem minha base forte.

Resumo

Neste trabalho estudou-se sobre a utilização da programação no ensino de física, mais especificamente foram trabalhadas simulações de Monte Carlo utilizando a linguagem Python como forma de trabalhar conceitos físicos por meio da programação. Foi analisado como a programação pode ter efeitos benéficos no aprendizado de física desde que utilizada de maneira correta. Além disso, foram introduzidos os principais conceitos relativos à linguagem Python, que foi escolhida para trabalhar a parte da programação no decorrer deste trabalho. Foram apresentados os fundamentos dos métodos de Monte Carlo e alguns conceitos necessários para o bom entendimento dessa abordagem. Por fim, foram solucionados três problemas físico-matemáticos por meio de simulações de Monte Carlo em linguagem Python.

Palavras chave: física, programação, ensino.

Abstract

In this work we studied about the use of programming in physics teaching, more specifically Monte Carlo simulations were worked using the Python language as a way of working physical concepts through programming. It was analyzed how programming can have beneficial effects on physics learning as long as it is used correctly. In addition, the main concepts related to the Python language were introduced, which was chosen to work with the programming part during this work. The foundations of Monte Carlo methods and some concepts necessary for a good understanding of this approach were presented. Finally, three physical-mathematical problems were solved through Monte Carlo simulations in Python language.

Keywords: phisycs, programming, teaching.

Introdução

O ensino de programação promove nos alunos o desenvolvimento de habilidades como lógica, resolução de problemas, sistematização, divisão de problemas em partes menores, abstração, entre outras [1]. Tais habilidades são de especial valor para cientistas, em especial os físicos.

Além disso, a física computacional tem se mostrado um campo promissor para o futuro, e já conquistou seu espaço na pesquisa científica, abordando problemas que requerem soluções sofisticadas e muitas vezes custosas para serem abordadas em laboratórios com experimentos práticos.

Por conta desses, e de outros motivos, a programação deve ser abordada nas salas de aula ao se ensinar física. A programação pode ser uma ferramenta poderosa no processo de ensino-aprendizagem de física. Desta maneira, neste trabalho serão apresentados os conceitos básicos necessários para solucionar três problemas por meio de simulações de Monte Carlo. A escolha dos problemas que deveriam fazer parte deste trabalho foi pautada em baixa complexidade teórica e também baixa dificuldade de compreensão dos códigos produzidos.

Dessa maneira, o trabalho organiza-se da seguinte forma: no capítulo 1 será abordada uma fundamentação teórica acerca da utilização de programação no ensino de física. No capítulo 2 a linguagem de programação Python será apresentada, bem como uma introdução aos tipos de objeto que apresenta, sintaxe, funções básicas e importação de bibliotecas. No capítulo 3 serão apresentados os conceitos sobre os métodos de Monte Carlo, desde sua definição, passando por uma contextualização histórica, até os seus principais conceitos. No capítulo 4 serão solucionados alguns problemas propostos, utilizando simulações de Monte Carlo. Os problemas são a estimativa do número π, o problema do decaimento radioativo e o problema de partículas em uma caixa. Por fim, serão apresentadas algumas conclusões, discussões e perspectivas futuras.

Capítulo 1

Programação no Ensino de física

1.1 O atual modelo de ensino de física

Desde seu surgimento até sua consolidação, a área voltada ao ensino de física no ensino médio conta com a marca de notáveis docentes, porém hoje, acaba sendo marcada pela falta de professores na área "e os que existem são obrigados a treinar os alunos para as provas, para as respostas corretas, ao invés de ensinar física". Para além dos profissionais, a carga horária diminuiu em aproximadamente 3× e os conteúdos são tratados de forma tradicional. O resultado desse conjunto de fatores e desse retrocesso no meio, acaba sendo sentido pelos alunos em forma de grande dificuldade e indisposição à matéria de física [2].

> O ensino e a aprendizagem de física têm graves problemas. O modelo de aula expositiva e avaliação via prova escrita, como metodologia majoritária ou exclusiva de ensino, tem se mostrado desmotivador para os estudantes. A aula tradicional não dá ao aluno papel ativo no processo de aprendizagem. Essa situação dificulta o ensino e a aprendizagem e torna o envolvimento cognitivo dos alunos um desafio ainda maior para o professor [3].

No modelo de aula expositivo tradicional, utilizado atualmente, os assuntos tratados são desvinculados com a realidade, trabalhando a partir de conceitos, leis e equações que devem ser memorizados e exercitados através de listas a serem resolvidas com o objetivo de chegar a uma resposta certa, e não ao questionamento do que está sendo tratado [4].

A física, como a ciência que investiga as leis do universo, deve instigar e aflorar a natureza questionadora dos alunos e o desejo de conhecer o universo em que vive, tentando compreendê-lo. A forma com que a matéria é ensinada hoje está muito distante deste objetivo, sendo "ensinada através de teorias e abstrações, fugindo de modelos concretos que se baseiam em experimentos reais para desvendar tais abstrações" [5].

Uma forma de fomentar o interesse dos alunos em física e ainda complementar o processo entre ensino e aprendizagem, é a utilização de meios computacionais capazes de simular experimentos através de programas computacionais ministrados pelos próprios alunos, trabalhando como construtores [4].

Novas estratégias didáticas precisam ser estudadas e aplicadas a fim de priorizar o raciocínio crítico dos conteúdos. Com a aplicação de novos recursos, além do maior interesse na aprendizagem, os alunos poderão se dividir e compartilhar suas dúvidas e ideias, o que facilita a compreensão dos conteúdos.

1.2 Inserção de computadores no ensino

Atualmente os computadores são indispensáveis no cotidiano, na educação e nas tarefas profissionais dos cientistas e professores. Especialmente para físicos, teóricos ou experimentais, os computadores são utilizados para aquisição e visualização de dados, análises, cálculos, simulações e modelagens, entre outros. Dessa maneira, é de certa urgência que os alunos de graduação do curso de física se familiarizem formalmente com os métodos e técnicas da física computacional [6].

No âmbito do ensino de física para graduação, a física computacional tem sido utilizada de forma suplementar em disciplinas de física teórica e experimental em universidades ao redor do mundo desde meados da década de 2000 [7], entretanto seria um avanço ter a programação em primeiro plano no ensino de física, por conta dos benefícios que pode trazer ao aluno.

O pensamento computacional implica em resolver problemas, entender comportamentos e projetar sistemas com base nos fundamentos da ciência da computação, abordando os problemas por meio de divisões entre problemas menores e mais fáceis de serem resolvidos, concebendo um algoritmo mental. Além da redução, o problema também pode ser resolvido pela incorporação, transformação ou simulação, pensando na forma mais eficiente de resolvê-lo. Esta não é uma habilidade exclusiva de cientistas da computação, mas fundamental a todos. Para ler, escrever e fazer operações aritméticas, a capacidade analítica deve ser sempre acionada [8].

Na prática, há diversos exemplos na literatura em que a utilização de programação no ensino de física se mostrou muito eficaz. Em especial, do artigo de Colpo, Faria e Machado [9], é possível extrair informações valiosas.

Segundo os autores, aprender uma nova linguagem pode levar a uma diminuição da autoconfiança e motivação por levar muito tempo na correção de erros de códigos para utilização de sintaxes complexas. Então, para aplicação no ensino de física, foi escolhido a linguagem de programação em Python pela facilidade na criação de funções, sintaxes mais simples e curtas e codificação intuitiva, a fim de aumentar a autoconfiança dos alunos.

Além disso, os autores trabalharam a linguagem de programação Python em uma turma de 6 alunos de iniciação cientifica júnior na UERJ e o Colégio Pedro II - Regional Centro. Para isso, os conceitos da linguagem (estrutura de codificação, sintaxes e afins) foram ensinados alunos em 6 aulas de 1 hora cada, com base em problemas de conteúdos de física do ensino médio.

A ementa trabalhada tinha finalidade de oferecer conhecimento suficiente para que os alunos saíssem preparados para criar códigos capazes de resolver diversos problemas, mesmo que tivessem relativamente pouco conhecimento de programação.

Durante a aplicação da linguagem, houveram alguns erros de escrita de códigos, mas nada que desmotivassem os alunos, acredita-se que isso ocorreu devido a simplicidade da sintaxe da linguagem escolhida, demonstrando que Python tenha sido a escolha correta para esse primeiro contato.

Em relação ao conteúdo de física, foi "abordado utilizando a proposição de problemas e a construção coletiva de soluções, elaborados em nível crescente de dificuldade" com intuito de fortificar a autoconfiança do aluno. Sobre a resolução de problemas, os alunos foram estimulados a "pensar em voz alta" sobre cada etapa para a construção coletiva de soluções.

Com a aplicação da programação no ensino, alguns pontos foram percebidos, como a baixa habilidade dos alunos em resolver problemas em conjunto com "equívocos na for-

mulação de modelos mentais adequados", dentre outras diversas dificuldades que acabam marcando o processo de aprendizado de uma nova linguagem de programação.

Apesar disso, segundo os autores, o resultado entre os alunos foi muito satisfatório, pois além de aumentar a compreensão dos conteúdos propostos, tanto em física como em computação, os alunos ainda demonstraram uma melhora na qualidade e dissolução de problemas, sendo possível dizer então que esse conhecimento será útil na vida acadêmica dos alunos.

Tendo em vista os resultados apresentados pelos autores, pode-se dizer que a aplicação da programação no ensino de física pode ser benéfica tanto no ensino médio quanto no ensino superior.

Com o conhecimento na linguagem, professores terão ferramentas para dar significado ao conteúdo, incentivar o raciocínio lógico, pensamento crítico e ainda demonstrar através de experimentos computacionais o conceito físico teórico.

Capítulo 2

Python

Python é uma das linguagens de programação mais difundidas atualmente, tendo alcançado a terceira posição no ranking de linguagens de programação mais utilizadas em uma pesquisa realizada entre desenvolvedores de *software* no ano de 2021 pelo site Stack OverFlow [10], além de aparecer nas posições mais altas de diversas outras pesquisas de popularidade de linguagens de programação [11, 12].

Esta linguagem têm sido cada vez mais popular por diversos fatores, com destaque para a linguagem ser grátis e *open source*, ou seja, é de livre distribuição e aberta para contribuições da comunidade. Ainda, Python é portável, ou seja, pode ser escrita e compilada basicamente em todas as maiores plataformas atualmente em uso, como *Linux, Windows, Mac OS, Android*, entre outros. Uma outra característica da linguagem que fortalece sua popularidade é que é uma linguagem de alto nível, ou seja, sua sintaxe se aproxima mais das linguagens humanas do que da linguagem de máquina, de forma que seu uso é relativamente fácil comparada a alguns pares como Java ou C++.

Todas estas características apresentadas pela linguagem — e mais algumas outras especificidades técnicas que não cabem ser abordadas no escopo deste trabalho — a levam a ter uma vasta aplicação desde programação de sistemas, passando por rotinas de automação de tarefas repetitivas, até programação científica, entre outras aplicações [13].

Tendo em vista os objetivos do presente trabalho, a linguagem Python se alinha de forma ideal pois por ter custo zero e de relativa facilidade de aprendizado, além dos benefícios supracitados, tanto professores quanto alunos de qualquer instituição podem ter acesso à essa tecnologia sem grandes dificuldades para reproduzir as soluções dos problemas que serão abordados no Capítulo 4 deste trabalho.

2.1 Tipos de objetos em Python

A linguagem Python comporta nativamente diversos tipos de objetos diferentes. Os principais tipos são números, caracteres (*strings*), e listas.

Objetos numéricos ainda podem ser divididos em três subtipos, denominados *int*, *float* e *complex*. Os números do tipo *int* são inteiros, enquanto que os *float* são números reais. Os números do tipo *complex* têm uma parte real e uma parte imaginária, em que ambas as partes são do tipo *float* [14].

As principais operações que os objetos numéricos do tipo *int* e *float* suportam são as seguintes [14]:

- Soma → `x + y`

- Subtração → `x - y`
- Multiplicação → `x * y`
- Divisão → `x / y`
- Resto da divisão → `x % y`
- Potenciação → `x ** y`

Os objetos do tipo caracteres, também chamados de *string*, podem ser entendidos como sequências do tipo texto. Podem ser representados com aspas simples ou duplas. *strings* podem ser concatenadas, gerando uma combinação das *strings* concatenadas. Exemplo:

`"Fí" + "si" + "ca"` retorna `"Física"`.

Além disso, objetos de outros tipos podem ser convertidos em *string* por meio do método `str(objeto)`. Por exemplo:

O numérico `225` convertido para *string* por meio de `str(225)` retorna a *string* `"225"`.

É importante notar que com a *string* `"225"` não é possível realizar operações matemáticas, pois para a linguagem isso são caracteres. Assim, se for preciso realizar operações matemáticas com este objeto, será necessário transformá-lo em um objeto do tipo numérico antes utilizando a função `int(argumento)` quando `argumento` representar um número inteiro ou `float(argumento)` quando `argumento` representar um número real.

Por fim, as listas são sequências mutáveis, ou seja, podem ser alteradas. As listas apresentam muitas formas de manipulação, para mais detalhes consulte a documentação da linguagem [14].

Na linguagem Python vetores e listas são diferentes porém similares. Vetores não são nativos, e podem ser utilizados com auxílio da biblioteca Numpy, que será abordada mais a frente neste trabalho.

2.2 Sintaxe geral e operações básicas

Linguagens de programação exigem uma sintaxe correta para o bom funcionamento do código. Cada linguagem pode ter sua sintaxe, que pode ser entendida como uma estrutura de escrita de código específica que é aceita pelo compilador da linguagem como sendo a maneira correta de programar na linguagem em questão. Assim como a língua portuguesa — e outras — exigem uma estrutura específica, as linguagens de programação também.

Nesse sentido, será apresentada a seguir a sintaxe correta da linguagem Python para as principais linhas de comando que serão utilizadas neste trabalho.

Declaração	Função	Exemplo

Atribuição	Atribui valor à variáveis	`a = 10`
		`f = "Física"`
		`x, y = "abscissas", "ordenadas"`

`print`	Exibe objetos em tela	`print("Física legal", 10)`

`if/elif/else`	Condicional	`if x < 10:`
		`  print("x vale menos que 10!")`
		`elif x < 20:`
		`  print("x vale menos que 20!")`
		`else:`
		`  print("x vale mais que 20")`

`for`	Iteração	`for x in lista:` `  print(x)`
`while`	Loops em geral	`while x > 20:` `  print("x vale mais que 20")`
`import`	Acesso a módulos	`import numpy`
`from`	Acesso a atributos	`from numpy.random import rand`

Tabela 2.1: Apresentação da sintaxe da linguagem Python para as suas principais funções nativas.

Perceba que na linguagem Python declarações compostas, que são declarações que têm outras declarações aninhadas dentro delas, seguem um padrão de uma linha de cabeçalho finalizada com dois pontos, seguido de uma nova declaração indentada e aninhada à primeira declaração, logo abaixo da linha de cabeçalho. Veja o exemplo ilustrativo a seguir:

```
Linha de Cabeçalho:
   Declaração indentada aninhada
```

Outro ponto importante da sintaxe da linguagem Python é que o fim da linha é o fim da declaração. Isso difere de outras linguagens que exigem um ponto e vírgula para definir o

fim da declaração. Além disso, o fim da indentação é o fim de um bloco. Não é necessário abrir e fechar chaves, por exemplo, para indicar início e fim de um bloco de código. A sintaxe natural da linguagem é que ao iniciar uma indentação será iniciado um bloco, e ao finalizar a indentação será finalizado o bloco. Vale pontuar que no caso do Python, indentação significa deixar espaço em branco à esquerda de cada linha do bloco, não importando quanto espaço e nem se foi utilizado a tecla espaço ou tabulação para fazer o espaço em branco, desde que todas as linhas do bloco tenham o mesmo espaço em branco à esquerda.

2.3 Funções básicas

Funções podem ser entendidas como dispositivos de estruturação de programas. Elas têm dois principais papéis: o de maximizar a reutilização de códigos e minimizar redundâncias, e o de encapsular blocos de códigos que tem realizam ações específicas e bem definidas, permitindo a separação de um sistema em subsistemas que podem ser reutilizados em diversas partes desse sistema. É possível criar funções em Python para a tarefa especifica desejada, mas neste trabalho serão abordadas somente as funções nativas e das principais bibliotecas da linguagem Python utilizadas para execução dos algoritmos que serão propostos. Funções nativas, também denomiandas funções embutidas, são funções que sempre estão disponíveis na linguagem Python, elas já vêm, como o nome sugere, embutidas no interpretador da linguagem e para utilizá-las não é necessário importar nenhuma biblioteca [15].

Primeiro serão abordadas as duas funções nativas da linguagem que serão utilizadas neste trabalho. A função `print(argumento)` é utilizada para imprimir o valor associado ao objeto `argumento` que se deseja imprimir na tela [16]. Um exemplo seria a utilização da seguinte linha de comando:

```
print("Eu gosto muito de ciência!")
```

Em que as aspas indicam que o argumento utilizado é do tipo texto (*string*). O resultado dessa linha de comando seria:

```
Eu gosto muito de ciência!
```

Vale pontuar que o argumento, ou argumentos, da função `print` podem ser quaisquer objetos, que serão convertidos em texto para serem imprimidos em tela.

A função `len(argumento)` retorna o comprimento do objeto passado como argumento. O comprimento pode ser entendido como o número de itens contidos nesse objeto. Se o objeto for do tipo texto, a função retornará quantos caracteres formam esse texto incluindo os espaços, se for uma sequência retornará quantos itens estão contidos nesta sequência [17]. Segue um exemplo:

```
len("Eu gosto muito de Física!")
```

O resultado da linha de comando acima seria:

```
25
```

Além das funções nativas da linguagem, existem conjuntos de funções desenvolvidas por terceiros que são encapsuladas no que se chama de bibliotecas. Para abordar essas funções de bibliotecas será necessário antes introduzir como importar bibliotecas.

2.4 Importação de bibliotecas

Uma biblioteca Python pode ser definida de forma geral como um arquivo de código que define funções com intuito de remover a necessidade de reescrever códigos que serão reutilizados muitas vezes. É possível criar novas bibliotecas, o no escopo deste trabalho serão somente importadas (acessadas) bibliotecas produzidas por terceiros, disponíveis gratuitamente.

A importação de uma biblioteca é necessária para utilização de suas funções no arquivo em que se deseja escrever um código. Acompanhe o exemplo: suponha que seja necessário tomar a média de valores. Não existe uma função nativa em Python que faça esse procedimento, entretanto para não ter a necessidade de criar uma função que calcula a média de valores toda vez que for necessário, foi criado um pacote chamado Numpy, em que há definições de diversas funções matemáticas [18], incluindo uma função que toma a média de valores passados como argumento. Assim, pode-se simplesmente importar essa biblioteca para o projeto desejado e a partir daí será possível obter a média de valores com a utilização de uma simples função já pronta.

Observe o exemplo a seguir em que são importadas várias bibliotecas:

```
from matplotlib import pyplot
from math       import exp

import numpy
import random
```

Perceba que nas duas primeiras importações, foram especificadas quais funções seriam importadas da biblioteca. Supostamente só se desejava utilizar as funções em específico e não seria necessário importar uma biblioteca inteira para isso, assim importa-se só a função necessária. Nas duas últimas linhas do exemplo acima foram importadas as bibliotecas inteiras.

A biblioteca Numpy fornece funções matemáticas, como citado anteriormente. Uma de suas funções é a `numpy.exp(argumento)`. O argumento deve ser um vetor preenchido por numéricos. A função irá calcular a exponencial de cada elemento do vetor [19]. Segue um exemplo:

```
numpy.exp([1, 2, 3])
```

O resultado da linha de código acima será equivalente a um vetor que contém a exponencial dos elementos do vetor passado como argumento para a função, ou seja, `[2.71828183, 7.3890561, 20.08553692]`.

Há também a função `numpy.sqrt(argumento)`, em que é esperado que o argumento seja um vetor preenchido por numéricos. A função irá calcular a raiz quadrada de cada elemento contido no vetor [20]. Veja o exemplo a seguir:

```
numpy.sqrt([4, 16, 25])
```

O resultado da linha de código acima será equivalente a um vetor que contém a raiz quadrada dos elementos do argumento, ou seja, `[2, 4, 5]`.

Uma outra função é a `numpy.mean(argumento)`, em que também é esperado que o argumento seja um vetor contendo numéricos. A função retornará um numérico representando a média dos valores contidos no vetor [21]. Observe o exemplo:

```
numpy.mean([10, 20])
```

O resultado da linha de código acima será `15`.

A biblioteca Numpy ainda comporta a função `numpy.array(argumento)`, que cria um vetor do formato especificado no argumento [22]. Exemplo:

```
numpy.array([1, 2])
```

O retorno da linha de código acima será um vetor com duas posições equivalente a `[1, 2]`.

Já a função `numpy.zeros(argumento)`, espera que o argumento seja um inteiro, que determina a forma de um vetor que será criado e preenchido com o número 0 em todas as posições. Segue o exemplo:

```
numpy.zeros(2)
```

O resultado da linha de código seria a criação de um objeto do tipo array (vetor) com duas posições preenchidas com zero, equivalente à `[0, 0]` [23].

Também será utilizada a função `numpy.append(vetor, valores)` que adiciona valores ao final da cópia de um vetor. É esperado que o argumento seja um vetor já existente seguido dos valores que se deseja adicionar ao final do vetor especificado [24]. Veja o exemplo:

```
vetor = numpy.array([1, 2])
vetor = numpy.append(vetor, 3)
```

Ao final das linhas de código acima, o vetor `vetor` será equivalente a `[1, 2, 3]`.

Por fim também será utilizada a função `numpy.random.uniform(a, b, c)`. Esta função constrói um vetor de número aleatórios gerados no intervalo [`a`, `b`), e o vetor assume a forma de `c`. Não é obrigatório especificar `a` e `b`, nesse caso a função assume o intervalo padrão [0, 1) para a geração dos valores aleatórios. A especificação de `c` também é opcional, nesse caso será gerado um numérico tipo *float* somente [25]. Exemplos:

```
numpy.random.uniform()
numpy.random.uniform(2, 3, 3)
```

O retorno das linhas de código acima poderiam ser:

```
0.5829239836347324
[2.11861037, 2.14700914, 2.0887218 ].
```

A biblioteca `random` também será utilizada. Seu intuito se resume à geração de números aleatórios. Nesta biblioteca são fornecidas diversas funções, porém nos algoritmos que serão apresentados neste trabalho serão utilizadas duas funções. A função `random.random()` gera um número aleatório dentro do intervalo [0, 1) [26].

A função `random.uniform(a, b)` retorna um número N do tipo floating onde $a \leq N \leq b$ se $a \leq b$ e $b \leq N \leq a$ se $b \leq a$ [26].

A última biblioteca que será utilizada é a `matplotlib`, mais especificamente uma coleção de funções dentro desta a biblioteca, chamada de `pyplot`. Para fazer sua importação é necessário executar a linha de código `import matplotlib.pyplot`. Para facilitar, podemos atribuir um nome mais curto para chamar esta biblioteca em código executando o seguindo comando `import matplotlib.pyplot as plt`.

Desta coleção de funções, serão utilizadas algumas funções que configuram gráfico. A primeira é a `plt.plot(x, y)`, em que `x` e `y` devem ser vetores contendo as coordenadas em x e y que serão configuradas para o plot respectivamente. O plot será executado na presença do comando `plt.show()`. Exemplo:

```
plt.plot([1,2], [2,3])
plt.show()
```

A linha de código acima terá como retorno o seguinte gráfico:

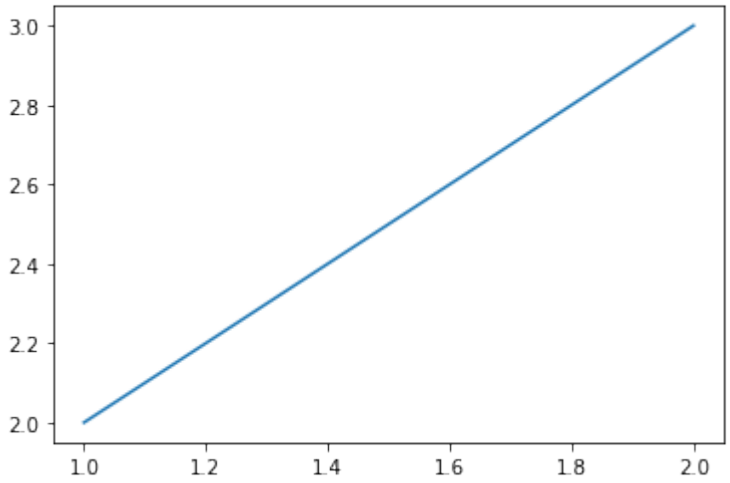

Figura 2.1: Exemplo de plot usando maplotlib.pyplot.

Há a possibilidade de customizar o gráfico mudando cor da curva, adicionando legenda à curva, espessura da curva entre outras propriedades. Para tal é necessário adicionar argumentos à chamada da função `plot(x, y)`. Para verificar as possibilidades consulte a documentação [27].

Para controlar o intervalo mostrado nos eixos x e y é usada a função:

```
plt.axis([xmin, xmax, ymin, ymax])
```

Exemplo:

```
plt.plot([1,2], [2,3])
plt.axis([0, 3, 1, 4])
plt.show()
```

O retorno das linhas de código acima será:

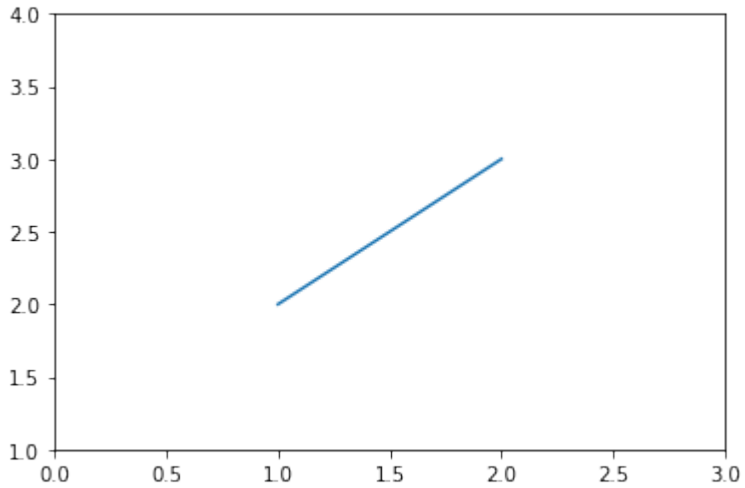

Figura 2.2: Alteração nos eixos do gráfico com plt.axis.

Há a possibilidade de configurar a legenda dos eixos x e y com as funções `plt.xlabel(argumento)` e `plt.ylabel(argumento)`, respectivamente. O argumento esperando é do tipo texto (*string*) [28, 29]. Veja o exemplo:

```
plt.plot([1,2], [2,3])
plt.axis([0, 3, 1, 4])
plt.xlabel("Eixo x")
plt.ylabel("Eixo y")
plt.show()
```

O retorno das linhas de código acima será:

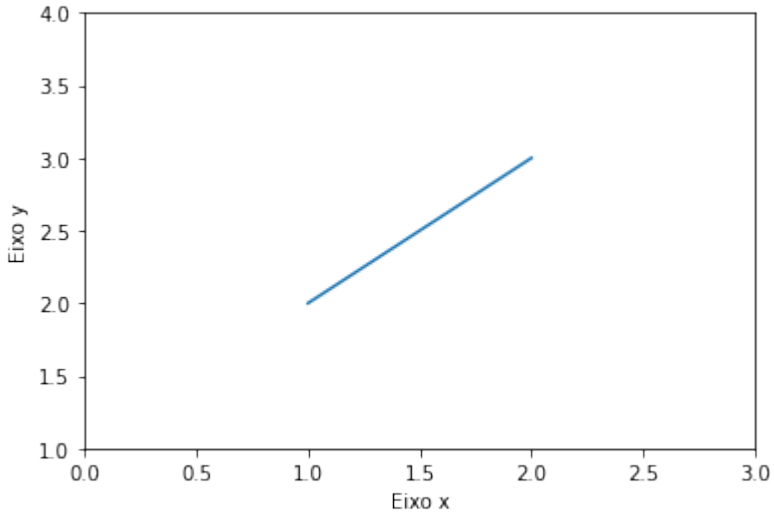

Figura 2.3: Alteração das legendas dos eixos do gráfico com plt.xlabel e plt.ylabel.

Há a opção de adicionar uma legenda para a curva com a função `plt.legend()`. Esta função pode ser usada sem argumentos, porém é necessário configurar uma `label` para o plot correspondente [30]. Veja o exemplo:

```
plt.plot([1,2], [2,3], label="Curva Exemplo")
plt.axis([0, 3, 1, 4])
plt.xlabel("Eixo x")
plt.ylabel("Eixo y")
plt.legend()
plt.show()
```

O retorno das linhas de código acima é mostrado na figura abaixo:

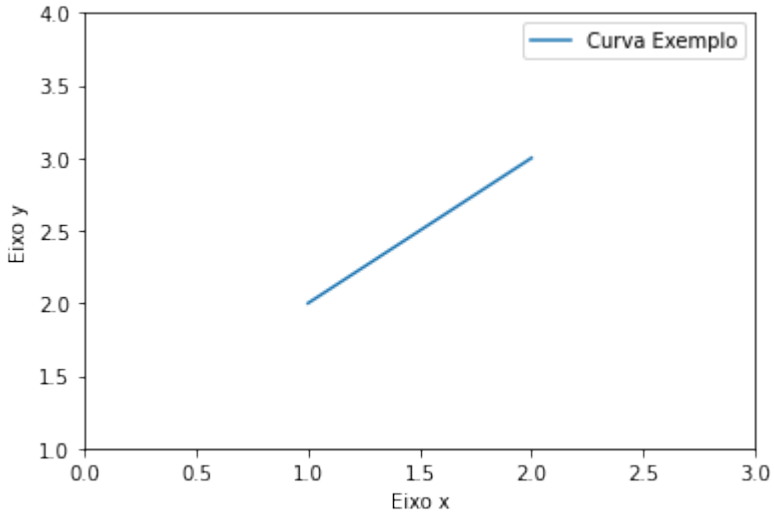

Figura 2.4: Alteração da legenda da curva do gráfico com plt.legend.

Por fim, para exibir o gráfico (ou gráficos) configurados, usa-se a função `plt.show()`, que não necessita de argumentos em sua chamada [31].

Capítulo 3

Métodos de Monte Carlo

3.1 O que é

Métodos de Monte Carlo são empregados em diversas áreas da ciência, desde a solução de integrais multidimensionais até a solução de problemas físicos e químicos, ou ainda em previsões de comportamentos de bolsas de valores.

Pode-se descrever os métodos de Monte Carlo como métodos de simulação estatística, em que entende-se simulação estatística como qualquer método que utilize sequências de números aleatórios para executar a simulação [32].

Em muitas aplicações de Monte Carlo, o fenômeno físico é simulado diretamente, de forma que às vezes nem é necessário se utilizar das equações diferenciais que descrevem o comportamento do sistema. O único requisito é que o sistema físico possa ser descrito por funções distribuição de probabilidade. Uma vez que essas funções são conhecidas, a simulação pode ser executada criando amostragens randômicas por meio dessas funções. Dessa maneira, muitas simulações são executadas e o resultado desejado é determinado como uma média sobre o número de observações.

3.2 Uma breve história da simulação de Monte Carlo

A utilização da probabilidade de certos eventos ocorrerem para obtenção de resultados concretos parece não ser muito intuitivo. Entretanto, é desta forma que simulações de Monte Carlo dos mais variados tipos vêm sendo utilizadas para prever clima, recessões econômicas, dentre outras aplicações. A ideia de usar a aleatoriedade de uma forma definitiva — no sentido de resultado definido — foi revolucionária. No século XVIII, o cientista francês Georges Louis LeClerc, conde de Buffon (1707-1788), utilizou métodos randômicos em diversos estudos, sendo o mais famoso o experimento denominado de agulha de Buffon. Neste experimento ele lançava agulhas repetidamente em um tabuleiro com linhas desenhadas para estimar o valor de π. LeClerc mostrou que para uma agulha de mesmo comprimento que a distância entre as linhas, a probabilidade de a agulha intersectar uma linha seria de $2/\pi$. Alguns consideram esse experimento como uma primeira instância de uma simulação de Monte Carlo [33].

No século XIX e começo do século XX, as simulações começaram a ser cada vez mais utilizadas como meio de confirmar teorias, analizar dados e em matemática estatística. Exemplos disso são a utilização um baralho com cartas marcadas para tratar tabelas de dados, ou a utilização de um dispositivo giratório para um algoritmo de suavizar curvas,

ou ainda a descrição de um método geral para simulação usando um dado modificado para gerar uma distribuição normal, entre outros [34].

Atualmente há uma grande diferença entre esses estudos iniciais e as simulações típicas de Monte Carlo modernas, que abordam assuntos que em muitos casos seriam de difícil investigação por outras metodologias, como a modelagem da formação de galáxias. O foco hoje em dia é encontrar um análogo probabilístico do problema, e o resolver probabilisticamente [33].

Esta forma de simulação foi inicialmente desenvolvida e utilizada sistematicamente durante o Projeto Manhattan, um investimento americano no período da Segunda Guerra Mundial a fim de desenvolver armas nucleares. John von Neumann e Stanislaw Ulam sugeriram utilizar estas simulações para estudar propriedades de viagens de nêutrons através de blindagem de radiação, e batizaram o método em homenagem ao Casino Monte Carlo, em Mônaco. Posteriormente, Neumann, Ulam e outros pesquisadores utilizaram simulações para muitos outros problemas nucleares e estabeleceram as bases para os métodos de simulações de Monte Carlo [33].

Atualmente, simulações de Monte Carlo são muito utilizadas como ferramenta científica para problemas que não têm soluções analíticas e que demandam experimentações custosas, impraticáveis ou que demandem muito tempo. Pesquisadores estudam sistemas complexos, repetem e/ou modificam experimentos e examinam quantidades ocultas em experimentações.

Há de se considerar as desvantagens do uso de simulações também: requerem grandes recursos computacionais, não geram soluções exatas e a qualidade dos resultados é proporcional a qualidade do modelo e dos dados de entrada utilizados. Soluções analíticas e alternativas experimentais devem ser consideradas antes de recorrer diretamente à simulações [33].

3.3 Principais conceitos

Há pelo menos quatro fatores cruciais para entender a estratégia básica de Monte Carlo. São eles:

- Variáveis aleatórias;
- Funções de distribuição de probabilidade (PDFs [1]);
- Momentos de uma PDF;
- A variância σ^2.

Números aleatórios, no contexto que foi apresentado neste trabalho, são aproximações numéricas ao conceito estatístico de variáveis estocásticas, também denominadas variáveis aleatórias [32].

Uma variável estocástica pode ser discreta ou contínua. Neste trabalho será definido uma letra maiúscula para denotar uma variável estocástica, como X, Y e Z.

Dois conceitos fundamentais são associados às variáveis estocásticas, o domínio e a PDF (funções de distribuição de probabilidade). O domínio é o conjunto $\mathbb{D} = \{x\}$ de todos os valores acessíveis que a variável pode assumir, de forma que $X \in \mathbb{D}$. Num

[1] Sigla do inglês, que significa *probability distribution function*.

lançamento de dado pode-se obter seis diferentes números, neste caso o domínio discreto para o número obtido é $x \in \{1, 2, 3, 4, 5, 6\}$.

A PDF é uma função $p(x)$ no domínio que, no caso discreto, descreve a probabilidade ou frequência relativa com que esses valores de X ocorrem:

$$p(x) = Prob(X = x).$$

No caso contínuo, a PDF não descreve diretamente a probabilidade, ao invés disso, a probabilidade para a variável estocástica — que pode assumir qualquer valor num intervalo infinitesimal ao redor de x — é definida como $p(x)dx$. Nesse caso, a função contínua $p(x)$ representa uma densidade de probabilidade. A probabilidade de uma variável estocástica assumir qualquer valor num intervalo não infinitesimal $[a, b]$ é dada pela integral:

$$Prob(a \leq X \leq b) = \int_a^b p(x)dx.$$

Um outro conceito de interesse nesse contexto é a função de distribuição de probabilidade cumulativa (CDF [2]) $P(x)$, que é a probabilidade de uma variável estocástica X assumir qualquer valor menor que x:

$$P(x) = Prob(X \leq x) = \int_{-\infty}^{x} p(x')dx'.$$

A relação entre uma CFD e a PDF correspondente é dada por:

$$p(x) = \frac{d}{dx}P(x).$$

Há duas condições que todas as PDFs precisam satisfazer. A primeira é que a PDF precisa estar entre 0 e 1, ou seja, $0 \leq p(x) \leq 1$. É de certa forma intuitivo que algum valor do domínio tenha probabilidade de ocorrer menor ou igual a um (100%), e maior ou igual a zero (0%). Além disso, a PDF precisa ser normalizada, ou seja, todas as probabilidades somadas resultam em uma unidade. Para PDFs discretas e contínuas essa condição é representada respectivamente por:

$$\sum_{x_i \in \mathbb{D}} p(x_i) = 1$$
$$\int_{x \in \mathbb{D}} p(x)dx = 1$$

Outro conceito importante é o de valor esperado de uma função. Suponha $h(x)$ sendo uma função arbitrária no domínio da variável estocástica X cuja PDF é dada por $p(x)$. O valor esperado de h em relação a p é definido como:

$$\langle h \rangle_X \equiv \int h(x)p(x)dx$$

[2]Sigla do inglês, que significa *cumulative probability distribution function.*

A partir deste momento, o X será omitido quando for implicitamente conhecida a PDF.

Em especial, uma classe de valores esperados é muito útil, denominada como momentos. O n-ésimo momento da PDF p é definido como:

$$\langle x^n \rangle \equiv \int x^n p(x) dx$$

O momento de ordem zero é a condição de normalização de p. Observe:

$$\begin{aligned} \langle x^0 \rangle &= \int x^0 p(x) dx \\ 1 &= \int p(x) dx. \end{aligned}$$

O momento de ordem um é chamado de a média de p:

$$\begin{aligned} \langle x^1 \rangle &= \int x^1 p(x) dx \\ \langle x \rangle &= \int x p(x) dx. \end{aligned}$$

De modo qualitativo pode-se entender o primeiro momento como o centroide, ou o valor médio da PDF, e por isso geralmente é denominado simplesmente o valor esperado de p. Uma PDF pode ser expandida em termos de seus momentos. Para duas PDFs serem consideradas iguais, cada um de seus momentos devem ser iguais [32].

Há ainda uma família de momentos denominados os momentos centrais. O n-ésimo momento central é definido como:

$$\langle (x - \langle x \rangle)^n \rangle \equiv \int (x - \langle x \rangle)^n p(x) dx$$

O momento central de ordem zero e de primeira ordem são triviais, igual a 1 e 0 respectivamente. Entretanto o momento central de ordem dois é de especial importância, e é denominado como a variância de p. Para a variável estocástica X, a variância é representada como:

$$\begin{aligned} \sigma_X^2 = \langle (x - \langle x \rangle)^2 \rangle &\equiv \int (x - \langle x \rangle)^2 p(x) dx \\ &= \int \left(x^2 - 2x\langle x \rangle + \langle x \rangle^2 \right) p(x) dx \\ &= \langle x^2 \rangle - 2\langle x \rangle \langle x \rangle + \langle x \rangle^2 \\ &= \langle x^2 \rangle - \langle x \rangle^2 \end{aligned}$$

A raiz quadrada da variância, $\sigma = \sqrt{\langle (x - \langle x \rangle)^2 \rangle}$, é denominada como o desvio padrão de p, que pode ser interpretado qualitativamente como o espalhamento de p ao redor de sua média.

Tendo em vista os conceitos abordados acima, é possível ilustrar a utilização das simulações de Monte Carlo para resolver problemas, que serão apresentadas no capítulo 4.

Capítulo 4

Problemas Propostos

Como forma de deixar mais palpáveis os conceitos até aqui apresentados, serão apresentados e resolvidos um problema matemático e dois problemas físicos por meio de simulações de Monte Carlo. Estas soluções podem ser utilizadas como exemplos para o ensino do básico de programação em python para a graduação em física, ou simplesmente como guia de estudo para o aluno que tenha interesse em aprender sobre este assunto.

4.1 Estimativa do número π

Existem algumas maneiras de estimar o número π utilizando métodos de Monte Carlo. Aqui serão apresentadas duas delas, o Método da Razão das Áreas e o Método do Valor Médio.

4.1.1 Método da Razão das Áreas

Para estimar π por meio deste método deve-se iniciar assumindo um círculo unitário inscrito num quadrado de lado também unitário. Veja a figura 4.1.

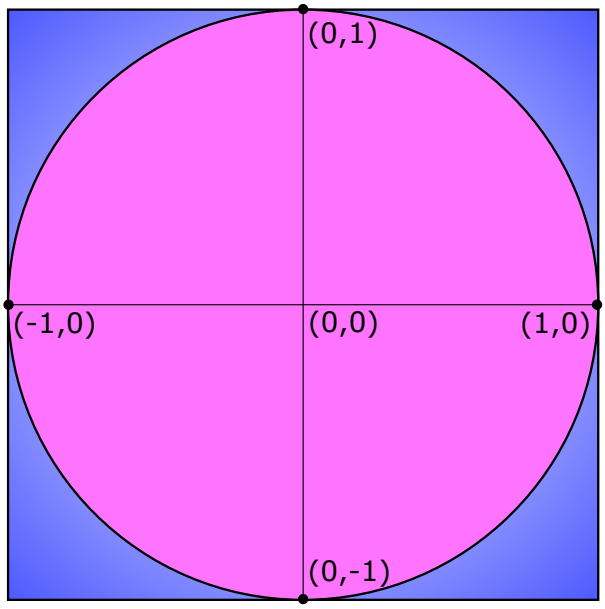

Figura 4.1: Círculo unitário inscrito num quadrado também unitário.

O conceito geral a partir daí é gerar pontos randômicos uniformemente contidos nesse plano cujo domínio é delimitado pelas bordas do quadrado unitário. Dessa forma, pode-se calcular a probabilidade de um dado ponto ser gerado dentro do círculo, simplesmente calculando a razão entre a área do cículo e a área total do quadrado. Isso é válido pois os pontos serão distribuído uniformemente por todo o domínio. Esta probabilidade P é dada por:

$$P = \frac{A_{círculo}}{A_{quadrado}} = \frac{\pi \ . \ r^2}{l^2}$$

Como $r = \dfrac{1}{2}$ e $l = 1$, ficamos com:

$$P = \frac{\pi \ . \ \dfrac{1}{4}}{1}$$

$$P = \frac{\pi}{4}.$$

Se por um lado foi encontrado esse valor para P analiticamente, por outro lado pode-se calcular a probabilidade P simplesmente calculando a razão entre o número de pontos que efetivamente foram gerados dentro do domínio do círculo e o número de pontos totais gerados, desde que o número de pontos gerados seja muito grande.

Pode-se fazer isso utilizando um algoritmo que gera pontos randomicamente e verifica se eles estão dentro ou fora do círculo e faz a contagem do número de pontos gerados dentro do círculo. Em seguida o algoritmo calcula a razão entre este valor e o número total de pontos gerados, resultando na mesma probabilidade P.

Deste modo, tem-se um valor numérico simulado para P, que pode-se comparar diretamente ao valor encontrado anteriormente, de $\frac{\pi}{4}$, estimando assim o valor de π por meio de simulação de Monte Carlo.

Finalmente, a estimativa de π será dada da seguinte maneira:

$$\frac{n^\circ \text{ de pontos dentro do círculo}}{n^\circ \text{ total de pontos}} = \frac{\pi}{4}$$

$$4 \ \frac{n^\circ \text{ de pontos dentro do círculo}}{n^\circ \text{ total de pontos}} = \pi,$$

em que tanto o n° de pontos dentro do círculo quanto o n° total de pontos serão conhecidos após a simulação.

A seguir será apresentado o código do simples algoritmo descrito acima. Note que com poucos recursos da linguagem, já é possível resolver problemas interessantes com poucas linhas de código.

```
1  import random
2
3  quantidade_pontos = 500000
4  pontos_circulo    = 0
5  pontos_quadrado   = 0
6
7  for i in range(quantidade_pontos):
8
9      rand_x = random.uniform(-1, 1)
10     rand_y = random.uniform(-1, 1)
11
12     dist_origem = rand_x ** 2 + rand_y ** 2
13
14     if dist_origem <= 1:
15         pontos_circulo += 1
16
17     pontos_quadrado += 1
18
19     pi = 4 * pontos_circulo / pontos_quadrado
20
21 print("Estimativa final de Pi =", pi)
```

O código acima resulta em:

```
Estimativa final de Pi = 3.140416
```

Observe que na linha 1 do código, foi importada a biblioteca random. Ela fornecerá funções que utilizaremos para gerar as posições aleatórias dos pontos. Da linha 3 à 5, foram inicializadas três variáveis que utilizaremos no decorrer do algoritmo. O valor inicial da variável `quantidade_pontos` é totalmente arbitrária, entretanto quanto maior seu valor, mais o resultado convergirá para o valor esperado. Na linha 7 há a chamada da função `for` que vai repetir um ciclo iterando sobre `i`, iniciando em `i = 0` e incrementando até `i = 499999`. Para cada ciclo desse, será gerado um valor aleatório para a posição x e y do ponto, como é possível observar nas linhas 9 e 10. Logo após, na linha 12, é calculada a distância do ponto em relação à origem do sistema. É verificado, na linha 14, se essa distância é menor que uma unidade, se sim o ponto encontra-se dentro do círculo e portanto a variável `pontos_circulo` é incrementada em uma unidade (linha 15). A seguir, na linha 17, a variável `pontos_quadrado` também é incrementada em uma unidade. Note que todos os pontos estarão dentro do quadrado, então esta variável deverá ser incrementada em todos os ciclos. A seguir, na linha 19, é feito o cálculo que resulta na estimativa para o número π. A cada ciclo o valor de π será diferente e será substituído pelo novo valor, já que depende do número de pontos dentro do círculo e do número de pontos dentro do quadrado, que são variáveis que são modificadas a cada ciclo. Logo em seguida, na linha 21, é chamada a função `print` para mostrar o resultado obtido.

4.1.2 Método do Valor Médio

Outra forma para estimar o valor de π por método de Monte Carlo, é usando a função $f(x) = \sqrt{1 - x^2}$. O gráfico desta função no intervalo [0,1] é mostrado na figura abaixo:

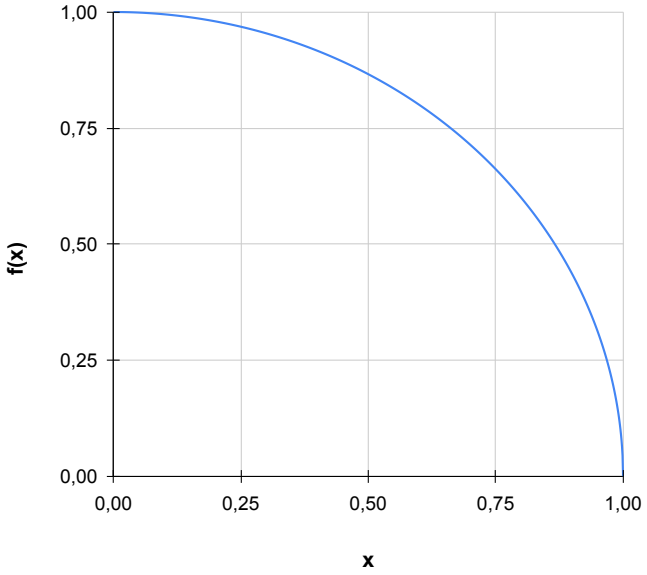

Figura 4.2: Gráfico da função $f(x) = \sqrt{1 - x^2}$ no intervalo [0,1].

Em linhas gerais, parte-se da ideia de gerar pontos randômicos no intervalo [0,1] com intuito de estimar o valor médio da função $f(x)$, que será representado por f_{avg}. Isto é feito gerando n valores para x randomicamente dentro do intervalo [0,1], calculando a $f(x) = \sqrt{1 - x^2}$ para cada x gerado e obter a média destes valores. Esse valor médio é o que equivale ao valor de f_{avg}. Colocado de outra forma, o que será feito é dado por:

$$f_{avg} = \frac{1}{n} \sum_{i=1}^{n} f(x_i),$$

onde n é a quantidade de números randômicos gerados. Evidentemente quanto maior n for, mais precisa será a estimativa. Este é o Método do Valor Médio.

Se por um lado pode-se estimar o valor médio da função num determinado intervalo gerando pontos aleatórios, por outro pode-se calcular analiticamente o valor médio da função por meio de:

$$f_{avg} = \frac{1}{b - a} \int_a^b f(x)\ dx.$$

Em particular, temos:

$$f_{avg} = \int_0^1 \sqrt{1 - x^2}\ dx.$$

É possível encontrar o valor desta integral sem calculá-la diretamente. Observe na figura 4.2 que no intervalo assumido, [0,1], a função descreve um quarto de círculo de raio unitário inscrito num quadrado de lado também unitário, como esperado. Pode-se obter o valor da integral calculando a área abaixo dessa curva.

A área de um círculo qualquer é dada por:

$$A = \pi\ r^2,$$

onde r é o raio do círculo, que nesse caso vale 1. Portanto,

$$A = \pi.$$

Dessa forma, para um quarto de círculo, a área relacionada à função será dada por:

$$A_{f(x)} = \frac{A}{4} = \frac{\pi}{4}.$$

Ou seja, o valor médio da função no intervalo dado é de

$$f_{avg} = \frac{\pi}{4}.$$

Dessa maneira, foram obtidas duas expressões para f_{avg}, que igualadas levam à:

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) = \frac{\pi}{4},$$

ou de outra forma:

$$4\ .\ \frac{1}{n}\sum_{i=1}^{n} \sqrt{1 - x_i^2} = \pi.$$

A equação acima poderá ser resolvida para π executando o simples algoritmo a seguir:

```
1 import numpy as np
2
3 n = 500000
4 x = np.random.uniform(size=n)
5
6 fx = 4 * np.sqrt(1 - x ** 2)
7 pi = np.mean(fx)
8
9 print("Estimativa final de Pi =", pi)
```

O código acima resulta em:

```
Estimativa final de Pi = 3.139322
```

Na linha 1 foi importada a biblioteca Numpy, que fornece funções que permitem manipular dados em forma de vetores, dentre outras funcionalidades. Na linha 3 e 4 foram inicializadas as variáveis `n` e `x`. `n` assume um valor inicial arbitrário que representa o número de pontos que serão utilizados para calcular o valor médio da função. `x` é inicializado

como um vetor com `n` posições preenchidas com números aleatórios, em específico como foi utilizado `n = 500000`, `x` é um vetor de 500000 posições em que cada posição foi preenchida por um número aleatório entre 0 e 1. Na linha 6, foi inicializada a variável `fx` que armazenará um vetor da mesma dimensão, entretanto aplicará a função $f(x) = \sqrt{1 - x^2}$ em cada valor armazenado no vetor `x`. Em seguida, na linha 7 é tomada a média de todos os 500000 valores que estavam armazenados em `fx`, e o resultado é atribuido à variável `pi`. Por fim, na linha 9, é feita a chamada da função `print` que imprimirá na tela o resultado obtido.

Note que ao reproduzir estes códigos, o valor resultante poderá apresentar divergências em relação ao apresentado, como esperado, pois trata-se de estimativas baseada em números aleatórios. Entretanto para intervalos grandes, a estimativa dos dois métodos apresentados deve convergir para o valor real de π.

4.2 Decaimento Radioativo

O problema do decaimento de partículas radioativas é um fenômeno físico que pode ser abordado por meio de métodos de Monte Carlo de forma razoavelmente simples e com resultados tão precisos quanto desejado.

Suponha então um conjunto de N partículas radioativas com um tempo de meia vida $t_{1/2}$. Sabe-se que no sistema restarão por volta de $\frac{1}{2}N$ partículas depois de decorrido um tempo igual a $t_{1/2}$, por volta de $\frac{1}{4}N$ depois de decorridos $2t_{1/2}$, e assim por diante. Entretanto, não é simples prever analiticamente e exatamente quantas partículas restarão em um dado tempo decorrido pois esse processo é estocástico, é regido por probabilidades.

Apesar disso, é possível simular o decaimento por Monte Carlo. Para tal, é necessário entender alguns conceitos do processo de decaimento radioativo. Para o decaimento radioativo, a probabilidade do decaimento por unidade de tempo de uma única partícula é constante e é denominada constante de decaimento. Suponha que existam N_0 partículas radioativas num dado tempo t e que a constante de decaimento é λ. A probabilidade de uma partícula qualquer decair num intervalo de tempo Δt é $P = \lambda \Delta t$. O número de partículas que decairão neste intervalo de tempo é $PN = \lambda \Delta t N$. Assim, o número total de partículas radioativas após esse intervalo de tempo terá decrescido de ΔN, como mostra a equação a seguir:

$$\Delta N = -\lambda \Delta t N.$$

Rearranjando a equação acima pode-se encontrar a taxa de decaimento:

$$\frac{\Delta N}{\Delta t} = -\lambda N$$

É possível tomar o limite de $\Delta t \to 0$, obtendo:

$$\frac{dN}{dt} = -\lambda N,$$

que é uma equação diferencial, cuja solução é dada por:

$$N(t) = N_0 e^{-\lambda t}, \tag{4.1}$$

em que N_0 é o número de partículas no tempo $t = 0$. O tempo de vida da partícula é definido como $\tau = 1/\lambda$ e o tempo de meia vida $t_{1/2} = ln(2)/\lambda$.

O algoritmo que será apresentado ao final desta seção consiste na simulação de um processo de decaimento radioativo de um conjunto de partículas nos moldes até aqui apresentados. O ponto central da simulação é determinar se uma partícula decaiu ou não num dado intervalo de tempo gerando um número aleatório entre 0 e 1. Se esse valor for maior que a probabilidade de decaimento, considera-se que a partícula decaiu ao fim do intervalo, se o valor for menor que a probabilidade de decaimento, considera-se que a partícula não decaiu ao fim do intervalo.

Essa abordagem é repetida para cada partícula contabilizando quantas partículas restaram ao final do intervalo, e depois repetido para o próximo intervalo, até que não reste mais nenhuma partícula.

Ao fim da rotina é plotado um gráfico para comparação com a solução exata apresentada na equação 4.1.

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

N0 = 500
lam = 0.01
dt = 0.01
P_decaimento = lam*dt

Nrestante = N0

N_t=np.array([Nrestante])

while Nrestante != 0:

    Nparticulas = Nrestante

    for N in range(Nparticulas):
        r = random.random()
        if (r < P_decaimento):
            Nrestante -= 1

    N_t = np.append(N_t,Nrestante)

tempos = np.arange(len(N_t))*dt

plt.plot(tempos,N_t,"r-",label="Simulação")

N_ave = N0*np.exp(-lam*tempos)
plt.plot(tempos,N_ave,label="Decaimento Exponencial")
```

```
31 plt.xlabel("Tempo")
32 plt.ylabel("Número de Partículas")
33 plt.legend()
34 plt.show()
```

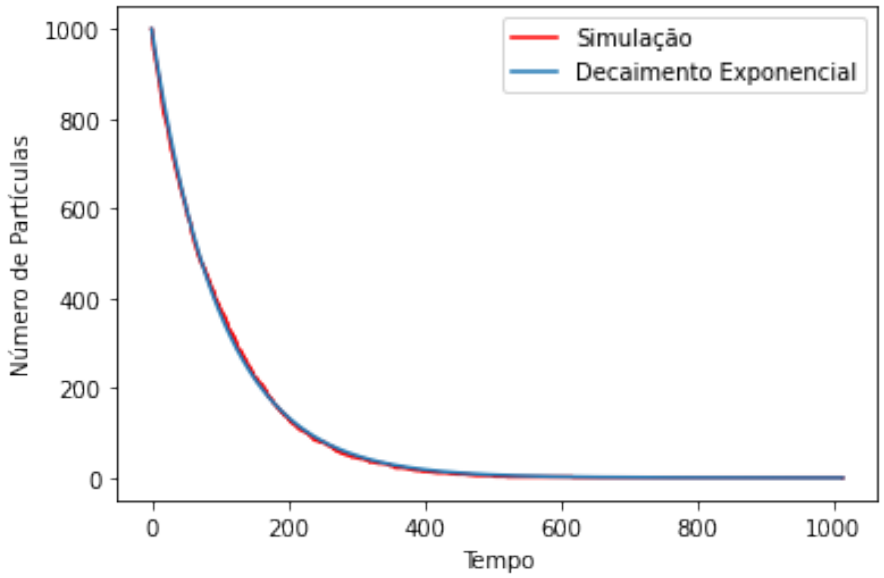

Figura 4.3: Número de partículas restantes em função do tempo.

Observe que nas três primeiras linhas do código foram importadas bibliotecas que permitirão utilizar funções matemáticas, compor gráficos e gerar números aleatórios, respectivamente. Da linha 5 à 8 foram inicializadas quatro variáveis. `N0` é o número inicial de partículas cujo valor é arbitrário, `lam` é a constante de decaimento, `dt` será utilizado para definir o intervalo de tempo e `P_decaimento` é a probabilidade de decaimento em cada intervalo de tempo. Na linha 10 é inicializada a variável `Nrestate` que representa quantas partículas ainda restam no processo de decaimento, mas como nenhuma partícula decaiu até o momento em que essa variável é inicializada, seu valor de inicialização é o valor inicial de partículas `N0`. Na linha 12 é inicializada a variável `N_t`, que será responsável por armazenar o número de partículas restantes ao final de cada intervalo de tempo. Inicialmente `N_t` será um vetor de uma posição que armazena o valor inicial `Nrestante`, que nesse caso é 1000. Mais a frente no código essa variável `N_t` será atualizada. Na linha 14 é configurado um ciclo que executará ações enquanto o número de partículas restantes for diferente de 0. Na linha 16 a variável `Nparticulas` é inicializada com o valor da variável `Nrestante`. Note que a cada ciclo, `Nrestante` poderá ter valor diferente. Na linha 18 é inicializado um ciclo que fará iteração em `N` desde 0 até `Nparticulas` - 1, em que `r` (linha 19) receberá um valor aleatório entre 0 e 1, e se for menor que `P_decaimento` (linha 20), `Nrestante` será descontada de uma unidade. Se `r` > `P_decaimento` representa o caso em que a partícula não decaiu e portanto resta o mesmo número de partículas que no ciclo anterior, não sendo necessário descontar `Nrestante`. Na linha 23 é adicionada na última posição do vetor `N_t` o valor de `Nrestante`, independente se houve mudança no valor de `Nrestante`. Na linha 25 é incializada a variável `tempos` que armazena um vetor em que cada posição guarda o tempo em cada ciclo. Esse vetor é necessário pois o valor amarzenado em sua última posição indica o tempo decorrido para o decaimento de todas as partículas, e também para que seja possível plotar o gráfico desse decaimento. Na linha

27 é configurado o plot da simulação, enquanto que da linha 29 até a 34 é configura o plot da solução analítica do problema.

4.3 Partículas em uma caixa

Assuma uma caixa dividida em duas partes iguais separadas por uma placa. No tempo $t = 0$, há n_0 partículas distribuídas na parte esquerda da caixa, quando um minúsculo orifício é feito na placa que separa as metades da caixa, de forma que somente uma partícula consegue atravessar o orifício a cada unidade de tempo decorrido.

Depois de decorrido um certo tempo o sistema atinge um estado de equilíbrio com um número igual de partículas, $\frac{n_0}{2}$, nas metades esquerda e direita da caixa. A título de simplificar o problema, que pode ter um valor de $n_0 >> 1$, é possível modelar o sistema com base em um modelo estatístico bem simples. Assume-se que todas as partículas na metade esquerda da caixa tenham a mesma probabilidade de passar para a metade direita. Além disso, o número de partículas do lado esquerdo e direito da caixa em cada tempo será denominado n_e e $n_d = n_0 - n_e$, respectivamente. A probabilidade de uma partícula passar para o lado direito da caixa em um intervalo Δt será de:

$$P = \frac{n_e}{n_0}.$$

Para solucionar esse problema utilizando Monte Carlo, é possível desenvolver um algoritmo que simula o sistema seguindo os seguintes passos:

- Escolher o número de partículas n_0;
- Montar um ciclo que incrementa sobre o tempo, em que o tempo máximo seja muito maior que n_0;
- Para cada incremento de tempo Δt há uma probabilidade P de uma partícula passar para a metade direita da caixa. Comparar essa probabilidade com um número aleatório x entre 0 e 1;
- Se $x \leq P$, decrementar o número de partículas na metade esquerda em uma unidade. Senão, incrementar o número de partículas na metade esquerda em uma unidade;
- Incrementar o tempo em uma unidade de tempo Δt e continuar o *loop*;

É possível comparar essa simulação numérica com uma solução analítica para o problema das partículas em uma caixa. Assumindo $n_e(t)$ como o número de partículas na metade esquerda da caixa depois de t unidades de tempo, ou ciclos como no algoritmo desenvolvido, a mudança em $n_e(t)$ em um intervalo de tempo Δt será dada por:

$$\Delta n = \left(\frac{n_0 - n_e(t)}{n_0} - \frac{n_e(t)}{n_0} \right) \Delta t$$

e assumindo que n_e e t são ambas variáveis contínuas, no limite de $\Delta t \to 0$ chega-se a:

$$\frac{dn_e(t)}{dt} = 1 - \frac{2n_e(t)}{n_0},$$

cuja solução é dada por:

$$n_e(t) = \frac{n_0}{2}\left(1 + e^{-2t/n_0}\right),$$

com a condição inicial $n_e(t=0) = n_0$. O seguinte código transpõe o algoritmo apresentado em linguagem Python e ao final plota um gráfico comparando o resultado da simulação com a solução analítica.

```
from matplotlib import pyplot as plt
from math       import exp

import numpy as np
import random

n0         = 500000
tmaximo    = 10*n0
valores    = np.zeros(tmaximo)
tempo      = np.zeros(tmaximo)
nesquerda = n0

for t in range (0, tmaximo, 1):
    x = random.random()

    if x <= nesquerda/n0 :
       nesquerda -= 1
    else:
       nesquerda += 1

    tempo[t]   = t
    valores[t] = nesquerda

exato = n0/2 * (1 + np.exp(-2 * tempo / n0))

plt.plot(tempo, exato, label="Resultado Exato")
plt.plot(tempo, valores, label="Simulação")
plt.plot()
plt.axis([0,tmaximo, n0/4, n0])
plt.xlabel("Tempo")
plt.ylabel("Número de partículas")
plt.legend()
plt.show()
```

É possível comparar o resultado exato com a simulação proposta. O algoritmo apresentado já tem como saída o seguinte gráfico:

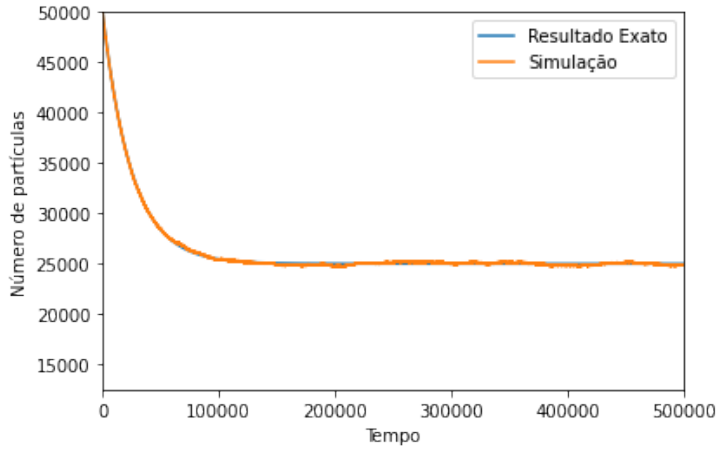

Figura 4.4: Número de partículas na metade esquerda da caixa em função do tempo.

Note que nas linhas 1 e 2 do código foram importadas funções específicas de bibliotecas, que permitirão plotar gráficos e calcular a exponencial de valores, respectivamente. Nas linhas 4 e 5 foram importadas bibliotecas que permitirão utilizar funções matemáticas e que geram números aleatórios, respectivamente. Da linha 7 à 11 foram inicializadas 5 variáveis. `n0` representa o número total de partículas e seu valor é arbitrário, `tmaximo` representa um tempo máximo, tempo esse no qual deseja-se obter informações sobre o sistema. Seu valor deve ser algumas vezes maior que o número de partículas [32]. As variáveis `valores` e `tempo` armazenam cada uma um vetor com número de posições igual a `tmaximo`, todas elas preenchidas com o valor zero. Essas posições serão modificadas posteriormente durante os ciclos da simulação de forma que `tempo` será um vetor com os tempos e `valores` guardará os valores respectivos a cada tempo. `nesquerda` é a variável que representa o número de partículas que ocupam a metade esquerda da caixa. No tempo inicial, todas as partículas estão na metade esquerda, e portanto `nesquerda` recebe inicialmente o valor de `n0`. Na linha 13 é iniciado um ciclo em que `t` é iterado desde zero até `tmaximo` - 1, sendo incrementado em uma unidade por ciclo. Dentro desse *looping* é gerado um valor `x` aleatório entre 0 e 1 (linha 14). Na linha 16 é verificado se `x` é menor que a probabilidade `P = nesquerda/n0` de passar para a metade direta da caixa. Se sim, significa que uma partícula deixou a metade esquerda da caixa e portanto `nesquerda` é descontada em uma unidade (linha 17), caso contrário (linha 18), uma partícula deixou a metade direita e entrou na metade esquerda da caixa, e portanto `nesquerda` é incrementada em uma unidade (linha 19). Nas linhas 21 e 22 é armazenado o tempo `t` no vetor `tempo` e o valor `nesquerda` é armazenado no vetor `valores`, ambos na mesma posição porém em vetores diferentes. Na linha 24 é calculado o resultado exato para o fenômeno. Da linha 26 à linha 33 é configurado o gráfico para comparação do resultado exato com a simulação.

Finalmente, tendo em vista as opções de simulações apresentadas, que podem servir como uma investigação em que se aprende os conceitos teóricos aliados a novos conhecimentos em programação, mas que também podem ser utilizadas por um professor em sala de aula como ferramenta que estimula o aprendizado do aluno, fica evidente o poder da física computacional e também como programação pode ajudar a abordar de forma mais palpável alguns conceitos que podem ser muito abstratos para os alunos. Além disso, essas abordagens podem ser sementes que permitam o aluno se questionar coisas como: aliando física e programação, que tipo de coisas seria possível simular?

Conclusões

Neste trabalho foi abordada a temática do uso da programação no ensino da física. No Capítulo 1 foi apresentada uma breve fundamentação bibliográfica do uso de programação no ensino de física em que ficaram nítidos os benefícios de sua inserção no ensino de física. Além disso, o relato de professores que já aplicaram a programação no ensino de física mostra que essa prática é possível e ainda que é acessível.

No Capítulo 2, foram discutidos os pontos que culminaram na escolha do uso da linguagem Python no decorrer do trabalho, além de especificidades técnicas, bem como uma fundamentação básica do funcionamento das principais ferramentas da linguagem que foram utilizadas no presente trabalho. Conceitos como tipos de objetos, sintaxe geral da linguagem, funções básicas e importação de bibliotecas foram discutidos.

Já no Capítulo 3, foram abordados os conceitos fundamentais acerca dos métodos e simulações de Monte Carlo, como variáveis estocásticas, funções de distribuição de probabilidade, momentos, entre outros.

Por fim, no Capítulo 4, foram apresentados problemas físico-matemáticos e suas respectivas soluções utilizando simulações de Monte Carlo, resgatando todos os conceitos apresentados nos capítulos anteriores do trabalho. Foi estimado o valor de π por meio de dois métodos distintos, em que ambos se utilizam de geração de valores aleatórios para chegar à solução do problema, foi simulado o comportamento de um sistema de decaimento radioativo e um outro sistema de partículas em uma caixa, ambos por simulação de Monte Carlo. Além disso, no caso da estimativa do valor de π, foi analisado o desvio padrão de ambos os métodos utilizados, como meio de determinar qual método gerou resultado mais acurado.

Tendo em vista todas as etapas percorridas para a realização deste trabalho, foi possível concluir que a programação definitivamente deve conquistar mais espaço dentro das salas de aulas de ensino de física, principalmente no nível superior, onde espera-se que os graduandos, ao final de seus cursos de graduação, sejam capazes de lidar com problemas modernos que requerem soluções mais sofisticadas, ponto esse no qual a programação tem se mostrado ferramenta indispensável. Foi possível concluir ainda, que com pouco conhecimento técnico e baixo custo, é possível construir abordagens poderosas para problemas físicos. Além disso, ao introduzir a programação em sala de aula, são estimuladas diversas habilidades que físicos necessitam desenvolver a fim de conduzir investigações em laboratório ou de forma teórica.

Entre as perspectivas de trabalhos futuros, pretendemos construir planos de aulas e/ou atividades de conteúdos específicos para aplicação na graduação em física, em que a programação utilizando linguagem Python tenha papel central e facilitador par ao entendimento do conteúdo físico em questão.

Referências Bibliográficas

[1] R. Scherer, F. Siddiq, and B. Sánchez Viveros, "The cognitive benefits of learning computer programming: A meta-analysis of transfer effects.," *Journal of Educational Psychology*, vol. 111, p. 764, 2019.

[2] M. A. Moreira, "Uma análise crítica do ensino de física," *Estudos Avançados*, vol. 32, pp. 73–80, 2018.

[3] M. B. dos Santos, "Uma sequência didática com os métodos instrução pelos colegas (peer instruction) e ensino sob medida (just-in-time teaching) para o estudo de ondulatória no ensino médio," Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016.

[4] M. L. Cassal, G. Orengo, and S. M. de Aguiar Isaia, "O problema do lançamento oblíquo no ensino de física com abordagem na programação de computadores," *Revista Educar Mais*, vol. 5, pp. 878–900, 2021.

[5] S. R. N. Coral and L. P. Guimarães Filho, "Monitoria de física no ensino médio: Uma experiência de aprendizagem," *CATAVENTOS-Revista de Extensão da Universidade de Cruz Altas*, vol. 3, 2012.

[6] K. R. Roos, "An incremental approach to computational physics education," *Computing in Science & Engineering*, vol. 8, pp. 44–50, 2006.

[7] A. Bäcker, "Computational physics education with python," *Computing in Science & Engineering*, vol. 9, pp. 30–33, 2007.

[8] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, pp. 33–35, 2006.

[9] R. A. Colpo, A. U. de Faria, and A. F. Machado, "O ensino de física no ensino médio intermediado por programação em linguagem python," *X ENPEC*, 2015.

[10] S. OverFlow, "2021 developer survey." <https://insights.stackoverflow.com/survey/2021#overview>, 2021. Acesso em 24 de Fev. 2022.

[11] I. Spectrum, "Top programming languages 2021." <https://spectrum.ieee.org/top-programming-languages/>, 2021. Acesso em 28 de Fev. 2022.

[12] Tiobe, "Tiobe index for february 2022." <https://www.tiobe.com/tiobe-index/>, 2022. Acesso em 28 de Fev. 2022.

[13] M. Lutz, *Learning Python, Fifth Edition*. O'Reilly Media, Inc., 2013.

[14] P. S. Foundation, "Built-in types." <https://docs.python.org/3/library/stdtypes.html>. Acesso em 06 de Mar. 2022.

[15] P. S. Foundation, "Funções embutidas." <https://docs.python.org/pt-br/3/library/functions.html>. Acesso em 07 de Abr. 2022.

[16] P. S. Foundation, "Built-in functions." <https://docs.python.org/3/library/functions.html#print>, 2022. Acesso em 05 de Mar. 2022.

[17] P. S. Foundation, "Built-in functions." <https://docs.python.org/3/library/functions.html#len>, 2022. Acesso em 05 de Mar. 2022.

[18] NumPy, "About us." <https://numpy.org/about/>, 2021. Acesso em 05 de Mar. 2022.

[19] NumPy, "numpy.exp." <https://numpy.org/doc/stable/reference/generated/numpy.exp.html?highlight=exp#numpy.exp>. Acesso em 05 de Mar. 2022.

[20] NumPy, "numpy.sqrt." <https://numpy.org/doc/stable/reference/generated/numpy.sqrt.html?highlight=sqrt#numpy.sqrt>. Acesso em 05 de Mar. 2022.

[21] NumPy, "numpy.mean." <https://numpy.org/doc/stable/reference/generated/numpy.mean.html?highlight=mean#numpy.mean>. Acesso em 05 de Mar. 2022.

[22] NumPy, "numpy.array." <https://numpy.org/doc/stable/reference/generated/numpy.array.html?highlight=array#numpy.array>. Acesso em 05 de Mar. 2022.

[23] NumPy, "numpy.zeros." <https://numpy.org/doc/stable/reference/generated/numpy.zeros.html?highlight=zeros#numpy.zeros>. Acesso em 05 de Mar. 2022.

[24] NumPy, "numpy.append." <https://numpy.org/doc/stable/reference/generated/numpy.append.html?highlight=append#numpy.append>. Acesso em 05 de Mar. 2022.

[25] NumPy, "numpy.random.uniform." <https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html>. Acesso em 06 de Mar. 2022.

[26] P. S. Foundation, "random — generate pseudo-random numbers." <https://docs.python.org/3/library/random.html>, 2022. Acesso em 06 de Mar. 2022.

[27] Matplotlib, "matplotlib.pyplot.plot." <https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot>. Acesso em 06 de Mar. 2022.

[28] Matplotlib, "matplotlib.pyplot.xlabel." <https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.xlabel.html>. Acesso em 06 de Mar. 2022.

[29] Matplotlib, "matplotlib.pyplot.ylabel." <https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.ylabel.html>. Acesso em 06 de Mar. 2022.

[30] Matplotlib, "matplotlib.pyplot.legend." <https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.legend.html>. Acesso em 06 de Mar. 2022.

[31] Matplotlib, "matplotlib.pyplot.show." <https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.show.html>. Acesso em 06 de Mar. 2022.

[32] M. Hjorth-Jensen, *Computational Physics - Lecture Notes Fall 2013*. University of Oslo, 2013. Disponível em: <http://www.sicyon.com/resources/library/compute/CompuPhysics2013.pdf>. Acesso em 02 Fev. 2022.

[33] R. L. Harrison, "Introduction to monte carlo simulation," in *AIP conference proceedings*, vol. 1204, pp. 17–21, American Institute of Physics, 2010.

[34] S. M. Stigler, *Statistics on the table: the history of statistical concepts and methods*. Harvard University Press, 2002.