Universidade Federal do Piauí
Centro de Ciências da Natureza
Curso de Pós-graduação - Mestrado Profissional em Matemática

Sérgio Augusto Dias Castro

Algoritmos e Pensamento Computacional como Ferramenta no Processo de Ensino-Aprendizagem

Teresina
2020

Sérgio Augusto Dias Castro

Algoritmos e Pensamento Computacional como Ferramenta no Processo de Ensino-Aprendizagem

Dissertação apresentada ao Curso de Pós Graduação - Mestrado Profissional em Matemática da UFPI como requisito parcial para a obtenção do grau de MESTRE em Matemática.

Orientador: Antonio Kelson Vieira da Silva

Doutor em Matemática - UFPI

Teresina

2020

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca Setorial de Ciências da Natureza - CCN

C355a Castro, Sérgio Augusto Dias.
Algorítmos e pensamento computacional como ferramenta no processo de ensino-aprendizagem / Sérgio Augusto Dias Castro. – Teresina: 2020.
79 f. il.

Dissertação (Mestrado Profissional) – Universidade Federal do Piauí, Centro de Ciências da Natureza, Pós-Graduação em Matemática - PROFMAT, 2020.
Orientador: Prof. Dr. Antonio Kelson Vieira da Silva.

1. Lógica Matemática. 2. Programação. 3. Matemática – Ensino Médio. I. Titulo.

CDD 511.3

Bibliotecária: Caryne Maria da Silva Gomes – CRB3/1461

Sérgio Augusto Dias Castro

Algoritmos e Pensamento Computacional como Ferramenta no Processo de Ensino-Aprendizagem

Dissertação apresentada ao Curso de Pós Graduação - Mestrado Profissional em Matemática da UFPI como requisito parcial para a obtenção do grau de MESTRE em Matemática.

Aprovado em 06 de julho de 2020

BANCA EXAMINADORA

Antonio Kelson Vieira da Silva

Doutor em Matemática - UFPI

Manoel Vieira de Matos Neto

Doutor em Matemática - UFPI

Marcos Antonio Tavares Lira

Doutor em Desenvolvimento e Meio Ambiente - UFPI

À minha família, pelo amor e paciência ao longo da caminhada.

Mãe, seu cuidado e dedicação foram cruciais para seguir firme em busca do objetivo.

Pai, sua presença me deu a certeza de que nunca estou sozinho.

Irmão, teu apoio me deu forças para chegar aqui.

Agradecimentos

Gostaria de agradecer à todos aqueles que contribuíram na minha vida e me tornaram capaz de realizar essa conquista.

A minha mãe, Maria do Carmo Ribeiro Dias Castro, por todo amor, carinho e apoio incondicional em todas as decisões que venho a tomar na minha vida.

Ao meu pai, Manoel Oliveira Castro, por sempre me incentivar a alçar os maiores voos possíveis e fazer com que eu tentasse ser sempre melhor.

Ao meu irmão, Augusto Everton Dias Castro, pelo grande incentivo aos estudos e entrega à vida acadêmica, e as inúmeras ajudas nos momentos difíceis.

A minha namorada, Lais Ferraz Reis Barroso, por ser meu porto seguro quando as situações pareciam adversas, sempre com as palavras certas a serem ditas.

Ao meu orientador, Antonio Kelson Vieira da Silva, por todos os ensinamentos e dedicação à árdua tarefa que é ser professor.

Aos meus amigos da turma do PROFMAT, por todos os momentos de estudo e de descontração, sendo sempre uma injeção de ânimo para seguir em busca dos nossos sonhos.

A Universidade Federal do Piauí, por ser minha segunda casa (ou até primeira) durante todo esse período, sempre acolhedora e recheada de conhecimento.

Enfim, a todos que de forma direta e indiretamente colaboraram para que este grande dia acontecesse.

"Não há ramo da Matemática, por mais abstrato que seja, que não possa um dia vir a ser aplicado aos fenômenos do mundo real".

Nikolai Lobachevsky

Resumo

Neste trabalho analisamos alguns dos principais conteúdos apresentados aos alunos do ensino médio no estudo de matemática, sob a perspectiva da construção de algoritmos que modelem passo a passo a resolução de problemas, evidenciando e estimulando o estudo da programação como ferramenta de aprendizagem matemática. Foi feita analogia a uma gama de situações em que pode ser aplicado o pensamento computacional na forma de uma sequência de passos na resolução de problemas cotidianos, correlacionando a matemática com nosso dia a dia. Estudamos também como contornar ou excluir restrições a determinados tipos de problemas, estabelecendo uma sequência de ações viável para a resolução dos mesmos, com uma heurística que busque solução ótima na maioria dos casos possíveis. O objetivo principal do trabalho foi buscar uma ferramenta que atraia o interesse dos alunos por meio de desafios e da utilização prática dos conhecimentos matemáticos. A implementação e compilação dos códigos apresentados foram desenvolvidas com o uso do *software* livre GNU Octave. Uma sequência didática apropriada foi desenvolvida, com foco na consolidação dos conhecimentos teóricos adquiridos previamente e aplicações diversificadas.

Palavras-chaves: Lógica, matemática, programação.

Abstract

In this work we analyzed some of the main contents presented to high school students in the study of mathematics, from the perspective of building algorithms that model step by step of problem solving, showing and stimulating the study of programming as a mathematical learning tool. We made an analogy to a range of situations in which computational thinking can be applied in the form of a sequence of steps in solving everyday problems, correlating mathematics with our daily lives. We also studied how to overcome or exclude restrictions to certain types of problems, establishing a viable sequence of actions to solve them, with a heuristic that seeks an optimal solution in most possible cases. The main objective of the work was to find a tool that attracts students' interest through challenges and the practical use of mathematical knowledge. The implementation and compilation of the presented codes were developed using the free software GNU Octave. An appropriate didactic sequence was developed, focusing on consolidating previously acquired theoretical knowledge and diversified applications.

Keywords: Logic, mathematics, programming.

Sumário

1 Introdução

A matemática como conhecemos nos dias de hoje é uma ciência presente em quase tudo que nos rodeia, desde os exemplos mais simples do nosso dia a dia, como fazer contagem de elementos ou calcular o valor da conta do supermercado, até situações mais complexas como a criptografia dos aplicativos em nossos *smartphones*. O próprio termo matemática, etimologicamente, tem origem grega e em suma, significa apenas conhecimento. Segundo Mol (2017), os estudiosos da Grécia Antiga logo trataram de racionalizar os pensamentos a respeito da compreensão dos números e das formas e iniciaram a estruturar um pouco da ciência que conhecemos hoje. Até hoje a matemática rege grande parte de como assimilamos as ciências exatas como um todo, e isso faz parte de todo um apanhado histórico de como a mesma foi introduzida desde nossos antepassados.

É dever do educador matemático contemporâneo, assim como era feito em eras passadas, correlacionar os elementos cotidianos com conhecimentos teóricos, com o objetivo de incentivar e mostrar a importância da ciência para seus alunos. Como isso pode ser feito? As respostas são as mais variadas. Um exemplo claro de como isso é introduzido comumente é na ideia de situações problema, ou seja, a modelagem matemática de situações que podem ser traduzidas por meio de expressões previamente conhecidas no estudo matemático.

Segundo Biembengut e Hein (2003):

> A Modelagem Matemática é o processo que envolve a obtenção de um modelo. Este, sob certa ótica, pode ser considerado um processo artístico, visto que, para se elaborar um modelo, além de conhecimento de matemática, o modelador precisa ter uma dose significativa de intuição e criatividade para interpretar o contexto, saber discernir que conteúdo matemático melhor se adapta e também ter senso lúdico para jogar com as variáveis envolvidas (BIEMBENGUT; HEIN, 2003).

Podemos observar, então, que o processo para uma modelagem matemática não deve ser algo estático, visto que depende muito de conhecimento teórico, mas também de imaginação e criatividade para melhor representar a situação desejada. Além disso, cientificamente falando, Kaiser (2005) assegura que os alunos irão compreender a

relevância científica da Matemática se ela for trabalhada com parâmetros modelados de eventos cotidianos, pois é o ambiente na qual fazem parte. Evidentemente, no desenvolvimento de um modelo matemático grande parte das vezes, por simplificação, podemos perder parte das informações, entretanto isso deve ser feito de modo que não prejudique o entendimento e a aplicação do modelo. De acordo com Bassanezi (2012):

> Um problema real não pode ser representado de maneira exata em toda sua complexidade por uma equação matemática ou um sistema de equações. Um modelo deve ser considerado apenas como um retrato ou uma simulação de um fenômeno e sua validação depende muito da escolha das variáveis e das hipóteses formuladas (BASSANEZI, 2012).

A modelagem é um processo dinâmico, e existem diversas vertentes a se seguir. Esse trabalho traz a aplicação de algoritmos como método de ensino lúdico. Nesse sentido, um dos focos é levar o pensamento computacional para dentro da sala de aula, fazendo com que o computador seja um instrumento de aumento do poder cognitivo e operacional humano – em outras palavras, usar computadores, e redes de computadores, para aumentar nossa produtividade, inventividade e criatividade (BLIKSTEIN, 2008).

Os objetivos específicos desta pesquisa se destacam pelos seguintes pontos: levar em consideração os conteúdos abordados no ensino médio de nossas escolas com base na Base Nacional Curricular Comum (BNCC); trazer a proposta da construção de algoritmos computacionais para descrever as situações vistas em sala de aula e a participação ativa do alunado nas aulas, sendo o elo principal do processo de aprendizagem. Foi levado em consideração, ainda, os possíveis erros mais recorrentes que podem vir a acontecer no desenvolvimento do pensamento computacional, trazendo metodologias que podem ser utilizadas para contornar as situações de acordo com a linguagem computacional escolhida. Realizando tais tarefas, é culminado o objetivo principal do trabalho: motivar o uso da tecnologia em sala de aula de forma mais interativa, fazendo do aluno o ponto principal da aprendizagem, no qual ele não deverá apenas utilizar algum *software* para resolver um problema, o aluno deverá ser a mente por trás do código, descrevendo o passo a passo da operação e estimulando as mais diferentes formas de resolver uma situação problema.

O presente trabalho objetiva construir uma base sólida para servir como material teórico e prático sobre a ciência dos computadores, com foco na lógica e no pensamento

computacional como ferramenta na aprendizagem de matemática, atraindo o interesse dos alunos por meio de desafios e da utilização prática dos conhecimentos previamente adquiridos. O público alvo desse estudo se refere aos estudantes do ensino básico (fundamental e médio), entretanto, também pode se estender para os discentes de Licenciatura em Matemática, como também para aqueles professores de matemática com outra área de formação inicial.

2 Algoritmos

2.1 Conceito e Histórico

Como dito, o enfoque do nosso trabalho é apresentar dentro do vasto campo da Matemática onde possamos aplicar algoritmos para responder problemas bastante recorrentes aos alunos de ensino médio. Para tal, devemos evidenciar precisamente o que vêm a ser tais algoritmos.

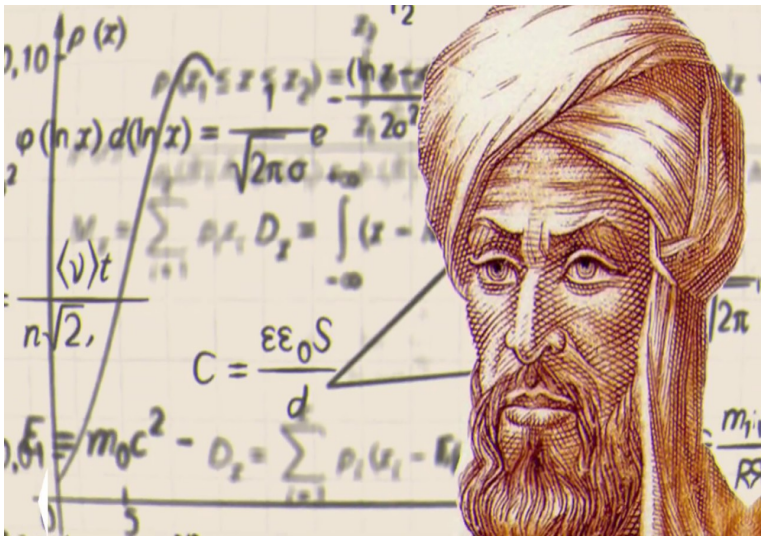
Segundo Cormen et al. (2002):

> Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. Portanto, um algoritmo é uma sequência de passos computacionais que transformam a entrada na saída. Também podemos visualizar um algoritmo como uma ferramenta para resolver um problema computacional bem especificado. O enunciado do problema especifica em termos gerais o relacionamento entre a entrada e a saída desejada. O algoritmo descreve um procedimento computacional específico para se alcançar esse relacionamento da entrada com a saída (CORMEN et al., 2002).

A definição apresentada acima é bastante assertiva do ponto de vista computacional, a ideia de construir uma sequência de passos computacionais bem definida que, por meio de alguma transformação, façam com que os valores de entrada se tornem valores de saída em um processo. Entretanto, antes da ideia computacional o estudo de algoritmos (de certo modo similar ao que conhecemos) já era estudado há muito tempo.

Para se ter noção, etimologicamente falando, o termo algoritmo por si só tem bastante ligação com a matemática. Tanto a palavra "algoritmo"quanto a palavra "algarismo"são palavras derivadas do mesmo radical *algoritmi*, que são heranças relacionadas ao nome do célebre matemático, astrônomo e geógrafo muçulmano Abu-Abdullah Muhammad ibn Musa al-Khwarizmi (738-850 d.C) (*algoritmi* seria a forma latina do nome al-Khwarizmi), sendo esse considerado um dos pais do algoritmo.

Figura 2.1: Muhammad ibn Musa Al-Khwarizmi



Fonte: Citizens Online[1]

Acredita-se que a obra *Al-kitab al-jabr wa'l muqabalah* teria uma tradução latina, que se encontra na Universidade de Cambridge, publicada pelo príncipe Boncompagni em Roma com o nome de *Algoritmi de numero Indorum*, com autoria inicial de al-Khwarizmi. De acordo com Garbi (2006), tal escrito seria carente de simbolismos e retórica, contudo teria influenciado de maneira bastante contundente os matemáticos ocidentais.

Tal obra mostrou, entre outros problemas, como resolver cada equação de 1º e 2º grau que se quisesse a um dos 6 tipos a seguir:

1. Quadrados iguais a raízes ($ax^2 = bx$)
2. Quadrados iguais a números ($ax^2 = c$)
3. Raízes iguais a números ($bx = c$)
4. Quadrados e raízes iguais a números ($ax^2 + bx = c$)
5. Quadrados e números iguais a raízes ($ax^2 + c = bx$)
6. Raízes e números iguais a quadrados ($bx + c = ax^2$)

Em tal grau, poderíamos definir de maneira bastante assertiva um passo a passo para cada um dos tipos de equação apresentada acima. Um dos exemplos mais

[1]Disponível em: https://www.citizensonline.org.uk/estimating-digital-champion-activity/

conhecidos seria o procedimento para se resolver a equação $x^2 + 10x = 39$. Observemos abaixo. Inicialmente temos a equação em questão:

$$x^2 + 10x = 39 \tag{2.1}$$

Para resolver esse problema al-Khwarizmi propõe somar 25 a ambos os lados da equação, pois, à vista disso, no lado esquerdo da igualdade teríamos um produto notável expandido, o quadrado da soma de dois termos:

$$x^2 + 10x + 25 = 39 + 25 \tag{2.2}$$

Por consequência, reduzindo do lado esquerdo e efetuando a soma no lado direito da igualdade temos como equação equivalente a exibida a seguir:

$$(x + 5)^2 = 64 \tag{2.3}$$

Extraindo a raiz quadrada em ambos os lados da expressão nos deparamos com:

$$x + 5 = \pm 8 \tag{2.4}$$

Com isso, culminamos no resultado $x = 3$ ou $x = -13$ como soluções da equação evidenciada. Da maneira exposta, temos que o passo crucial para o desenvolvimento desse tipo de problema seria adicionar em ambos os lados um valor que tornasse o lado esquerdo da igualdade em um produto notável. Isso é possível de ser feito observando os coeficientes dos termos, ou seja, o pensamento concatenado de passos se faz necessário para a resolução do problema.

O modo com que al-Khwarizmi demonstrou a resolução de tais problemas lhe rendeu notabilidade, inclusive fazendo com que alguns estudiosos o considerem como o "o pai da álgebra", em detrimento de Diophantus (que muitos consideram ser o detentor desse título). Inclusive, Boyer e Merzbach (2011) citam em sua obra:

> A exposição de al-Khwarizmi era tão sistemática e completa que seus leitores não devem ter tido dificuldade para aprender as soluções. Neste sentido, pois,

> al-Khwarizmi merece ser conhecido como o pai da Álgebra (BOYER; MERZBACH, 2011).

Podemos então perceber que al-Khwarizmi foi um introdutor do algoritmo como conhecemos hoje. Mesmo com a forte correlação do mesmo com a computação, percebemos que ele fora apresentado muito anteriormente ao advento da eletrônica, e está presente fortemente na matemática e em nosso cotidiano. Sendo o algoritmo uma sequência bem definida e finita de passos, sem ambiguidade e organizada de forma ordenada podemos facilmente apresentar onde percebemos a presença deste em nosso dia a dia, como enunciaremos a seguir.

2.2 Representações de Algoritmos

Nos deparamos anteriormente com a história do algoritmo, e podemos perceber que tal definição decorre, de certa forma, do desenvolvimento de um pensamento matemático para a resolução de determinados tipos de problema. O que não nos pode fugir da mente é que a utilização de algoritmos está diretamente ligada com a matemática, todavia, diariamente fazemos usos dos mesmos sem nem ao menos nos darmos conta.

Ora, fora visto que o algoritmo é uma fórmula lógica utilizada para resolvermos situações da maneira mais automática possível, ou seja, com menos esforço, a partir de uma sequência de passos pré-estabelecidos. E a partir de que definimos como proceder tal sequência? Como definimos qual será a resposta para uma determinada situação?

Como detalhado em Ferrari e Cechinel (2008):

> Para o desenvolvimento de um algoritmo eficiente é necessário obedecermos algumas premissas básicas no momento de sua construção:
>
> - Definir ações simples e sem ambiguidade;
> - Organizar as ações de forma ordenada
> - Estabelecer as ações dentro de uma sequência finita de passos.
>
> (FERRARI; CECHINEL, 2008).

O primeiro passo para esses questionamentos é conhecer o problema em questão que deve ser tratado. Qual a incógnita? Quais as informações que tenho para proceder? A informação que possuo é suficiente para a tomada de decisão? Então um passo primordial deve ser a elaboração de um plano a ser seguido para determinadas situações. Para isso, normalmente consideramos problemas correlatos que possamos ter encontrado, e se o resultado na resolução do problema foi o esperado, então podemos tomar como base a ação que o solucionou.

Finalmente, a execução do plano e a verificação do resultado final. Ao examinar a solução obtida veremos se poderia haver outra maneira de chegar naquela resposta ou se podemos usar o mesmo método para questionamentos ligeiramente diferentes.

Para mostrar como o algoritmo funciona no nosso dia a dia devemos conhecer algumas ferramentas bastante interessantes, entre elas a **descrição narrativa**, o **fluxograma** e o **pseudocódigo**. Para a demonstração das três ferramentas enunciadas iremos

construir o algoritmo da média aritmética de duas notas de um aluno para aprovação em uma disciplina.

2.2.1 Descrição Narrativa

A descrição narrativa nada mais é do que expressar na linguagem usual os passos necessários para o algoritmo. Dentro do exemplo proposto, vamos mostrar como seria a descrição narrativa do algoritmo da média aritmética de duas notas de um aluno:

Algoritmo 1: Descrição narrativa da média aritmética de duas notas de um aluno

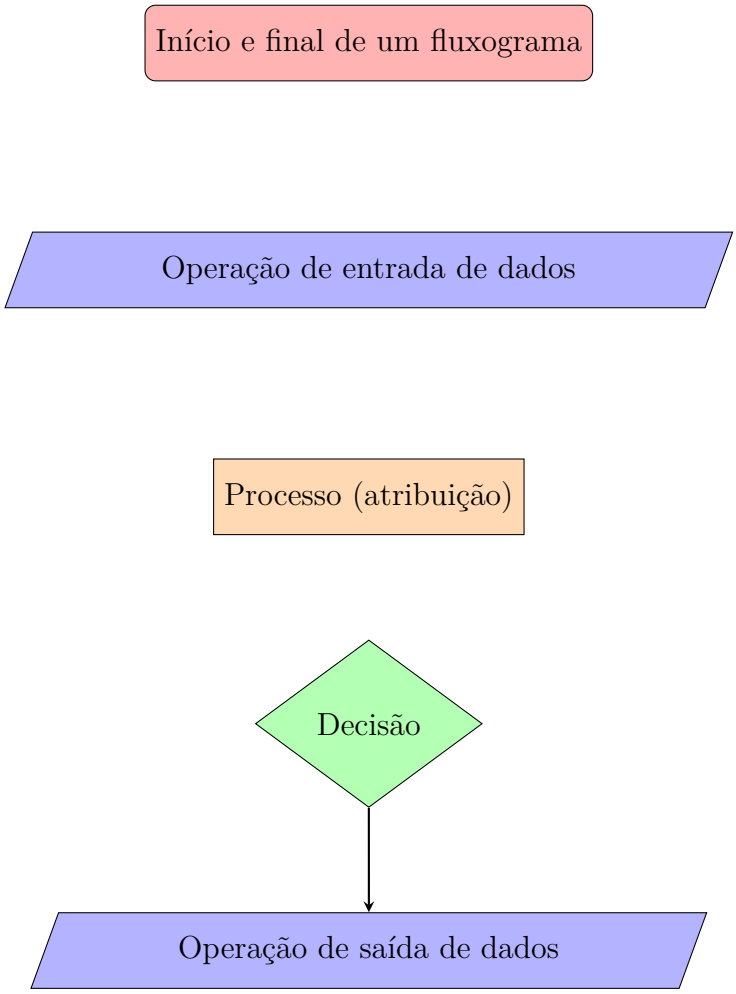
1 Devemos obter a primeira nota do aluno
2 Em seguida, obter a segunda nota
3 Somar os dois valores e dividir por dois
4 Se o resultado for maior ou igual a 7 o aluno será aprovado
5 Caso contrário, o aluno será reprovado

Podemos perceber que o tipo de representação descrito pode parecer bastante simples para o exemplo proposto, entretanto, uma das principais desvantagens do mesmo é que em exemplos mais complexos ela pode trazer brechas para más interpretações ou imprecisões. Além do que a linguagem usual em muito pouco se assemelha à linguagem de máquina (em se tratando da conversão para um algoritmo computacional) e também pois a linguagem usual varia de país para país, fazendo com que para um mesmo código traduções seriam necessárias para que fossem entendidos ao redor do mundo.

2.2.2 Fluxograma

O fluxograma se trata da representação gráfica de um algoritmo. A partir de diferentes formas geométricas temos significados distintos, como veremos adiante. Por a simbologia utilizada ser padronizada mundialmente isso facilita a compreensão independente da linguagem usual da pessoa que está visualizando a ferramenta. Alguns problemas relacionados ao uso do fluxograma é a falta de detalhamento em quais variáveis são analisadas, além do que dependendo do tamanho ou complexidade do algoritmo o fluxograma crescer bastante, dificultando tanto a aplicação como até mesmo a construção. Vamos agora conhecer os principais símbolos na construção e compreensão de fluxogramas:

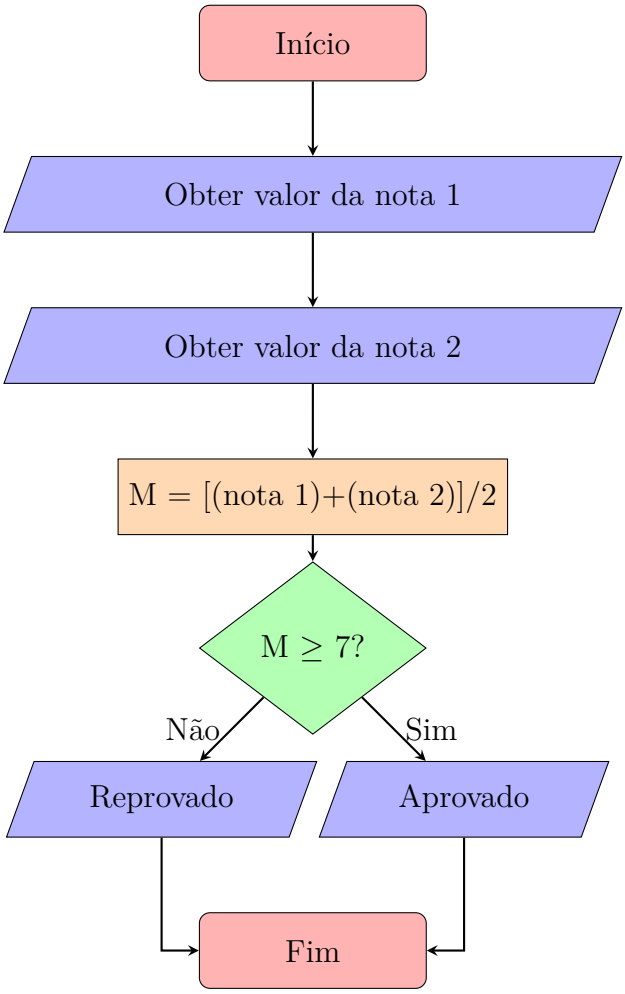
Fluxograma 1: Representação das formas geométricas presentes em um fluxograma



Fonte: O autor

De acordo com a estrutura apresentada, podemos observar como se dá a apresentação do passo a passo em meio aos algoritmos. Visto isso, iremos utilizar o nosso exemplo da média aritmética fazendo o uso de um fluxograma, observe a seguir:

Fluxograma 2: Média aritmética



Fonte: O autor

Podemos observar que o fluxograma apresentado traz uma compreensão bastante simplificada para o algoritmo proposto. Uma desvantagem acentuada se dá pelo fato de que por impor padrões para sua utilização de acordo com as representações geométricas citadas, há uma limitação no seu poder de expressão, se comparado com a descrição narrativa apresentada anteriormente.

2.2.3 Pseudocódigo

O pseudocódigo (também conhecido como linguagem algorítmica ou linguagem estruturada) é, comparado às outras duas representações algorítmicas apresentadas, aquela que possui maior detalhamento na representação de seus objetos. Tal representação se assemelha fortemente com a maneira com que códigos são de fato descritos em computador. A partir dele, é possível basicamente traduzir o algoritmo desejado para qualquer linguagem de programação que se queira, desde que se conheça os operadores necessários para tal.

Segundo Guedes (2004):

A forma geral da representação de um algoritmo na forma de pseudocódigo é a seguinte:

Algoritmo 2: Estrutura de um pseudocódigo

```
1 Algoritmo <nome do algoritmo>
2           <declaração de variaveis>
3           <subalgoritmos>
4 Início
5 <corpo do algoritmo>
6 Fim
```

Algoritmo é uma palavra que indica o início da definição de um algoritmo em forma de pseudocódigo.

<nome do algoritmo> é um nome simbólico dado ao algoritmo com a finalidade de distingui-los dos demais.

<declaração de variaveis> consiste em uma porção opcional onde são declaradas as variáveis globais usadas no algoritmo principal e, eventualmente, nos subalgoritmos.

<subalgoritmos> consiste de uma porção opcional do pseudocódigo onde são definidos os subalgoritmos.

Início e **Fim** são respectivamente as palavras que delimitam o início e o término do conjunto de instruções do corpo do algoritmo (GUEDES, 2004).

A partir da definição apresentada, iremos construir, enfim, o pseudocódigo referente ao exemplo da média aritmética das duas notas de um aluno, a seguir:

Algoritmo 3: Média aritmética de duas notas de um aluno

Dados: Nota1, Nota2 | Variáveis Numéricas

Resultado: Media | Variável Numérica

```
1 Início;
2 Ler Nota1;
3 Ler Nota2;
4 Media ← [Nota1 + Nota2]/2;
5 se Media ≥ 7 então
6 |   Escreva Aluno Aprovado!;
7 senão
8 |   Escreva Aluno Reprovado!;
9 fim
```

O que podemos observar analisando as três diferentes maneiras de representações algorítmicas apresentadas é que cada uma delas possui diferentes graus de detalhamento, e consequentemente diferentes graus de abstração. Portanto, quanto mais simples de entender uma representação, ela também tende a fornecer menos detalhes no que se refere a implementação do algoritmo em código propriamente dito. Logo, cada uma dessas é válida dependendo da intenção do aluno ou do professor.

2.3 Presença no Cotidiano

Em um primeiro momento, dadas as circunstâncias apresentadas até agora (com o exemplo dado, da média aritmética), podemos imaginar algumas situações em que o conhecimento a respeito de algoritmos pode ser utilizado, principalmente no âmbito de uma sequência de passos para, por exemplo, resolver uma equação (assim como no exemplo apresentado de al-Khwarizmi). Entretanto, nossa relação com os algoritmos é bem mais íntima, pois todos os dias estamos executando algoritmos relacionados à nossa própria rotina, nosso cotidiano.

Muitas das vezes, nossas atividades diárias são tão intuitivas e automáticas quando realizadas por nós mesmos que nos damos o direito de dispensar o tempo de pensar nas instruções que devem ser seguidas para completarmos uma tarefa, e dessa maneira deixamos passar despercebido o que de fatos estamos fazendo naquele momento. Tais ações como essa são denominadas como **algoritmos não computacionais**. O algoritmo

não computacional seria nada menos do que uma sequência de passos, como uma "receita", elaborada de forma a resolver problemas cotidianos. Aqui iremos aprofundar mais sobre isso na forma de exemplos.

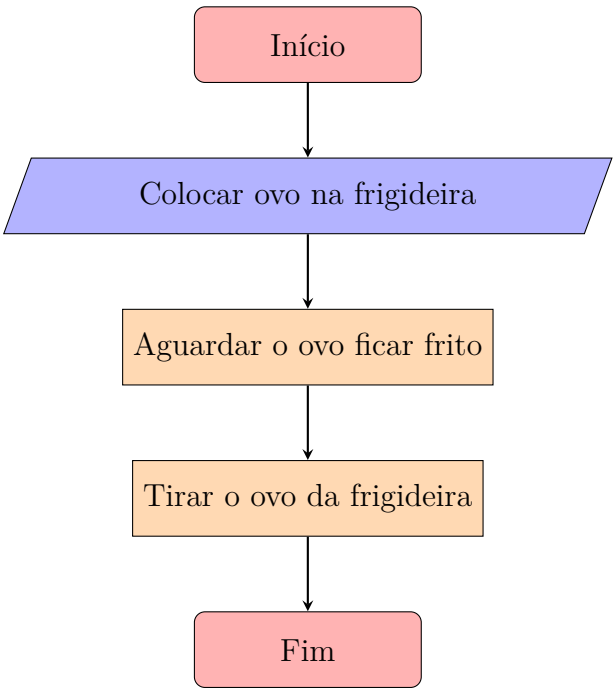
2.3.1 Fritando um Ovo

Esse é um exemplo simples e clássico de um algoritmo que não nos damos o trabalho de pensar cautelosamente passo a passo o que devemos fazer, por sua simplicidade de execução. Observe, a partir de algumas das representações de algoritmo apresentadas previamente, como podemos definir essa tarefa trivial, ou seja, quais os passos sequenciados necessários para fritar um ovo:

Algoritmo 4: Descrição narrativa de como fritar um ovo

1. Em primeiro lugar, devemos colocar um ovo em uma frigideira;
2. Em seguida, aguardar por alguns segundos o ovo ficar frito;
3. Por fim, tirar o ovo da frigideira;

Fluxograma 3: Como fritar um ovo



Fonte: O autor

A partir da descrição narrativa e do fluxograma apresentado podemos ter uma ideia simples de como devemos fritar um ovo. Todavia, podemos inclusive pensar que o desenvolvimento apresentado vem a suprimir vários passos intermediários que deveriam

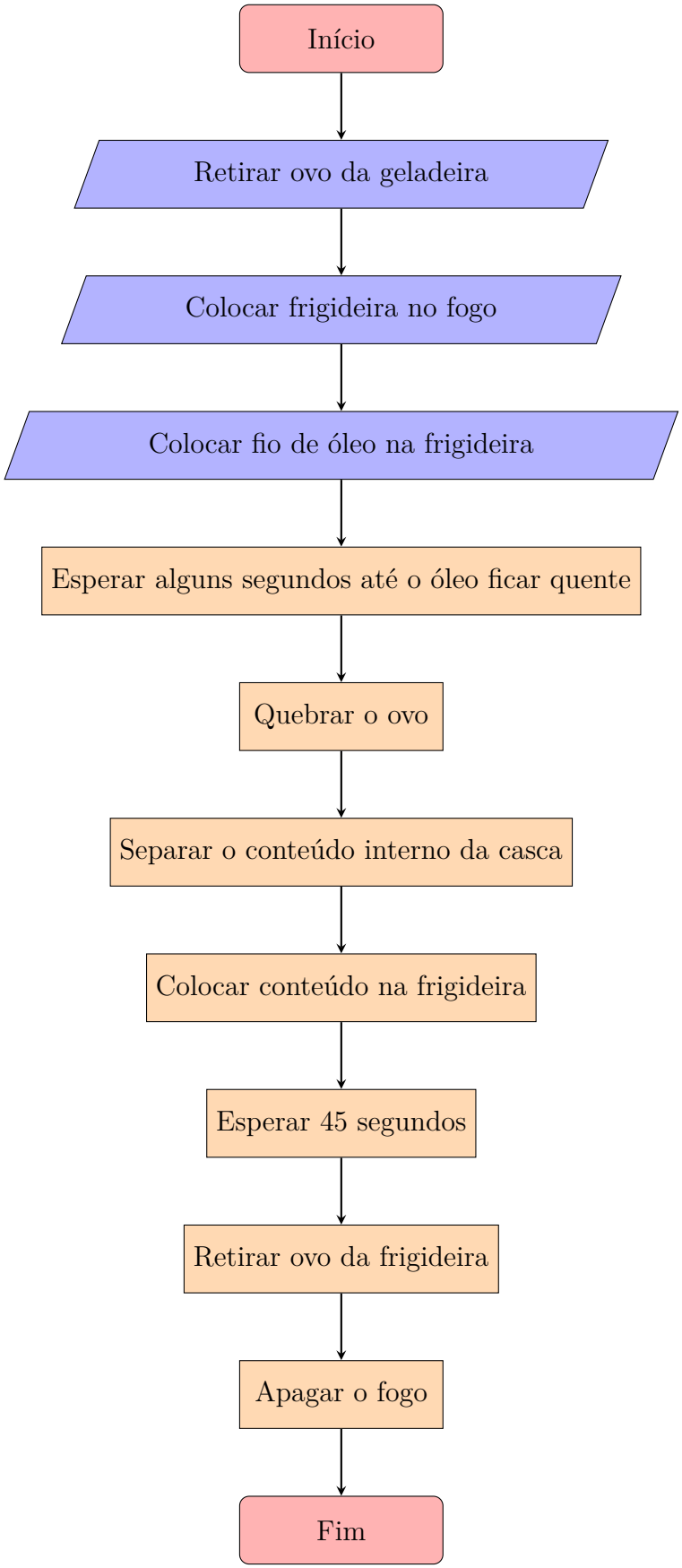
ser apresentados no código. Tais passos intermediários, dentro do estudo dos algoritmos, não devem ser negligenciados, pois uma sequência mais completa nos leva a resultados com mais exatidão, o que traz mais eficiência ao processo. Sendo assim, podemos apresentar uma nova maneira do algoritmo de como fritar um ovo:

Algoritmo 5: Descrição narrativa mais detalhada de como fritar um ovo

1 Em primeiro lugar, devemos retirar um ovo da geladeira;
2 Em seguida, colocar uma frigideira no fogo;
3 Colocar um fio de óleo na frigideira;
4 Esperar alguns segundos até o óleo ficar quente;
5 Quebrar o ovo, separando o conteúdo interno da casca;
6 Colocar o conteúdo dentro da frigideira;
7 Esperar 45 segundos;
8 Retirar o ovo da frigideira;
9 Apagar o fogo;

O próximo passo é construir o fluxograma desse algoritmo. Podemos perceber que por acrescentar uma maior variedade de instruções o código fica mais detalhado, e consequentemente com menos probabilidade de erros de interpretação, o que dentro da lógica do algoritmo facilita ainda mais para que as instruções possam ser executadas por cada um. Veremos a seguir, porém, que com praticamente o triplo de instruções o fluxograma também irá aumentar consideravelmente, o que já fora discutido previamente como uma característica negativa do uso de fluxogramas. Observe a seguir:

Fluxograma 4: Como fritar um ovo, detalhadamente



Fonte: O autor

2.3.2 Fazendo uma Ligação Telefônica

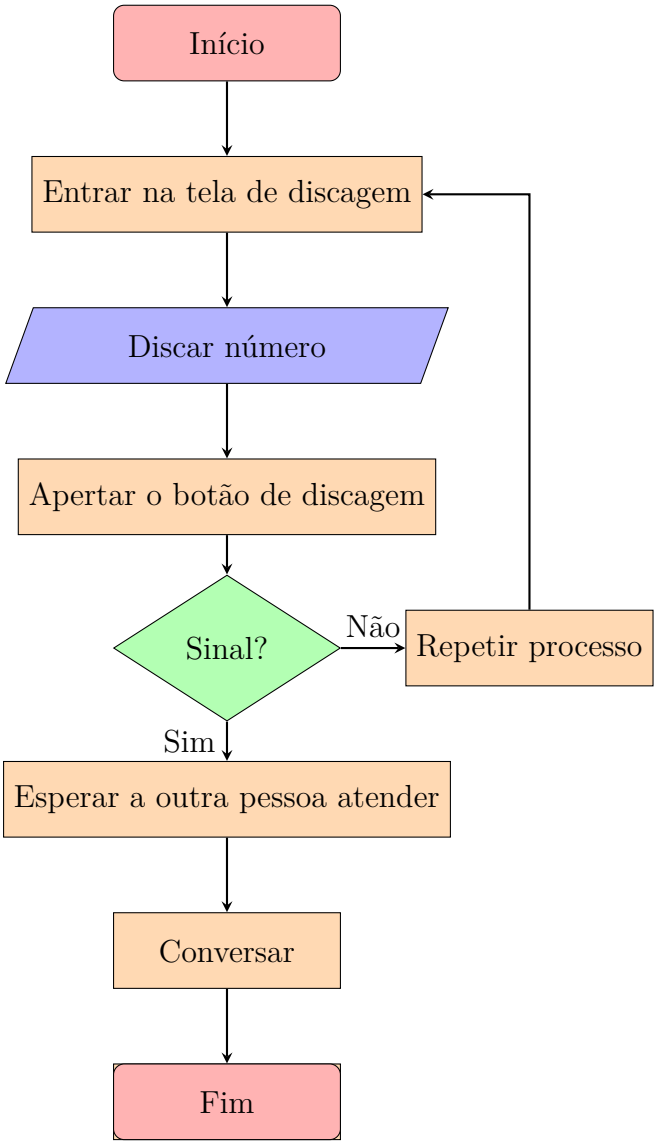
Nos exemplos apresentados anteriormente percebemos que o fluxograma, em ambos os casos, seguia uma linearidade. Isso significa que os passos deveriam ser feitos estritamente da maneira apresentada, sem nenhuma condição especial que pudesse se opor ao fluxo apresentado. Todavia, sabemos que em vários momentos de nosso dia a dia dependemos de condições para saber quando realizar determinada ação. No exemplo apresentado abaixo, numa adaptação de um código apresentado por Tonet e Koliver (2013), iremos mostrar o algoritmo que usualmente utilizamos ao tentar efetuar uma ligação telefônica de um aparelho celular.

Algoritmo 6: Descrição narrativa de como fazer uma ligação

1 Em primeiro lugar, devemos entrar na tela de discagem;
2 Em seguida, discar o número do telefone da pessoa que queremos conversar;
3 Então, aperte o botão de discagem;
4 Se ouvir o sinal de chamada, então espere a pessoa atender para então conversar, e ao final desligar a ligação;
5 Senão, repita o processo desde o início;

É possível observar por meio dos conectivos empregados na descrição narrativa que existem condições relacionadas ao fluxo do algoritmo. No exemplo em questão, tais conectivos são **SE** e **SENÃO**. Tal alteração do fluxo se reflete no comportamento das setas no fluxograma, como apresentado a seguir.

Fluxograma 5: Como fazer uma ligação



Fonte: O autor

Podemos perceber que os exemplos reais do nosso cotiano muito se assemelham ao processo apresentado acima. Nossas ações, desde a hora que acordamos até quando vamos dormir, são executadas a partir de decisões que derivam de condições, que muitas vezes passam desapercebidas pelos nossos olhos, por estarmos tão acostumados a pensar de maneira bastante direta. Ao refletir sobre isso, é possível perceber que cada dia de nossas vidas é um grande algoritmo, que pode ser narrado ou descrito por um fluxograma, o que torna o estudo desse conteúdo bastante aplicável.

3 Codificação e Programação

3.1 Aplicação da Computação

A utilização de algoritmos é crucial no que diz respeito ao ensino da matemática. Se temos uma sequência finita e bem definida de passos a ser cumprida, quando trazemos informações em forma de rotinas, o computador é a ferramenta mais poderosa para ser nossa aliada nos estudos. O computador foi criado de modo a ser uma ferramenta que proporcionaria uma substituição dos aspectos mecânicos do modo de pensar do homem. Em outras palavras, foi um mecanismo criado de modo a ajudar na diminuição de esforços repetitivos, como por exemplo na resolução de cálculos matemáticos de modo geral.

Conhecido como *"o pai da computação"*, o britânico Alan Turing[1](1912-1954) construiu um dos primeiros modelos de máquina abstrata, batizada de *Máquina de Turing*.

Figura 3.1: Alan Turing

Fonte: Head Topics[2]

[1]Matemático inglês pioneiro na computação e considerado o pai da Ciência dos Computadores. Ficou conhecido por ter decifrado o principal código NAZI durante a Segunda Guerra Mundial. Devido à decifração dos códigos das mensagens de comunicação dos NAZIS salvou milhares de pessoas. Desenvolveu trabalhos na área da matemática, lógica, computação, biologia e filosofia, foi o primeiro a abordar de forma estruturada a inteligência artificial (BRANDÃO, 2017).

[2]Disponível em: https://headtopics.com/br/h-65-anos-morria-alan-turing-o-pai-da-computac-o-e-da-ia-6283404

Como evidenciado em Fonseca (2007), a descrição que Turing dá a esse dispositivo é a seguinte:

> *"Computar é normalmente escrever símbolos em um papel. Suponha que o papel é quadriculado, podendo ser ignorada a bidimensionalidade, que não é essencial. Suponha que o número de símbolos é finito. [...]. O comportamento do(a) computador(a) é determinado pelos símbolos que ele(a) observa num dado momento, e seu estado mental nesse momento. Suponha que exista um número máximo de símbolos ou quadrículas que ele(a) possa observar a cada momento. Para observar mais serão necessárias operações sucessivas. Admitamos um número finito de estados mentais [...]. Vamos imaginar que as ações feitas pelo(a) computador(a) serão divididas em operações tão elementares que são indivisíveis. Cada ação consiste na mudança do sistema computador(a) e papel. O estado do sistema é dado pelos símbolos no papel, os símbolos observados pelo(a) computador(a) e seu estado mental. A cada operação, não mais de um símbolo é alterado, e apenas os observados são alterados. Além de mudar símbolos, as operações devem mudar o foco da observação, e é razoável que esta mudança deva ser feita para símbolos localizados a uma distância fixa dos anteriores. [...] Algumas destas operações implicam mudanças de estado mental do computador(a) e portanto determinam qual será a próxima ação".*

Temos em tal descrição o princípio de funcionamento dos computadores que utilizamos hoje. Poderosas ferramentas de cálculo que podem fazer milhões de operações em frações de segundo, graças ao advento do transistor, e consigo a microeletrônica. Não obstante, de modo a utilizar de toda a magnitude que o computador pode trazer para o âmbito da matemática, é necessário saber "conversar" com o mesmo. Para isso, é indispensável uma linguagem comum à máquina e ao usuário, pois é necessário haver comunicação para que o computador entenda o que é desejado que se faça. Com tal intuito é que devemos nos familiarizar a uma *linguagem de programação*.

A linguagem de programação será nossa forma de conversação com a máquina. O computador, em suma, trabalha com a representação binário (apenas os dígitos 0 e 1),

representando cada um deles a menor unidade de informação do computador, o *bit*. Os dígitos 0 e 1 estão relacionados com valores de tensão elétrica na máquina, sendo 0 um valor idealmente igual a 0 *volts* e 1 representando poucos *volts*. É possível perceber que se dependêssemos da representação binária para tratarmos da comunicação com o computador teríamos uma árdua tarefa pela frente, então a linguagem de programação é a responsável por simplificar o serviço. Segundo Sousa, Dias e Formiga (2014):

> Com o intuito de tornar menos complicada, mais eficiente e menos sujeita a erros a tarefa de programar computadores, foram criadas linguagens de programação mais próximas às linguagens naturais. Elas são compostas de um conjunto de palavras-chave, normalmente em inglês, e símbolos que estabelecem os comandos e instruções que podem ser utilizados pelo programador na construção de seus programas (SOUSA; DIAS; FORMIGA, 2014).

As linguagens de programação que mais se assemelham à descrição apresentada são chamadas de *linguagens de alto nível*. Nas linguagens de alto nível temos um grau de abstração maior, ou seja, não temos que falar diretamente na representação binário, o que torna mais fácil nosso entendimento dos códigos. Entre alguns exemplos de linguagens que preenchem tais requisitos estão *Pascal, C, C++, Python e GNU Octave*. Existem também linguagens com um grau de abstração menor, que com isso o usuário deve se comunicar de maneira mais próxima com o entendimento da máquina. As linguagens que contém essa característica são chamadas de *linguagens de baixo nível*. O exemplo mais famoso é o *Assembly*. Para que quaisquer linguagens venham a ser compreendidas pela máquina se faz necessário o uso de programas que sejam capazes de transformar a informação dada pelo usuário em uma informação que o computador possa ser capaz de interpretar. Dessa maneira, existem diversos *tradutores* e *compiladores* para fazer essa tarefa, de acordo com a linguagem escolhida. De acordo com Evaristo e Crespo (2000):

> Um programa fonte deve ser traduzido para a linguagem de máquina. Há dois tipos de programas que fazem isto: os interpretadores que traduzem os comandos para a linguagem de máquina um a um e os compiladores que traduzem todo o programa para a linguagem de máquina. Um compilador ao receber como entrada um programa fonte fornece como saída um programa escrito em linguagem de máquina, chamado programa objeto. A compilação do programa, portanto, gera um programa que pode então ser executado. É comum nos referirmos à execução do programa fonte quando se está executando o programa objeto. Já um interpretador traduz para a linguagem de máquina os comandos

> do programa um a um, executando-os em seguida. Assim a interpretação de um programa não gera um programa objeto (EVARISTO; CRESPO, 2000).

A partir disso, devemos analisar qual aplicação iremos trabalhar para sermos capazes de decidir qual linguagem de programação é a mais indicada para obter o resultado desejado.

3.2 GNU Octave

A escolha da linguagem de programação a ser utilizada é de crucial importância para o desenvolvimento de qualquer projeto. Vimos anteriormente que existem as mais diversas opções, porém dependendo da aplicação desejada devemos escolher a que melhor se encaixa. No objetivo de iniciar estudantes a escreverem algoritmos matemáticos, é de bom tom buscar linguagens simples (alto nível e maior abstração), pois o principal foco dos alunos não deve ser na sintaxe ou até mesmo no tempo de processamento de um programa, e sim no que é necessário para que o código siga um passo a passo com enfoque na matemática em si.

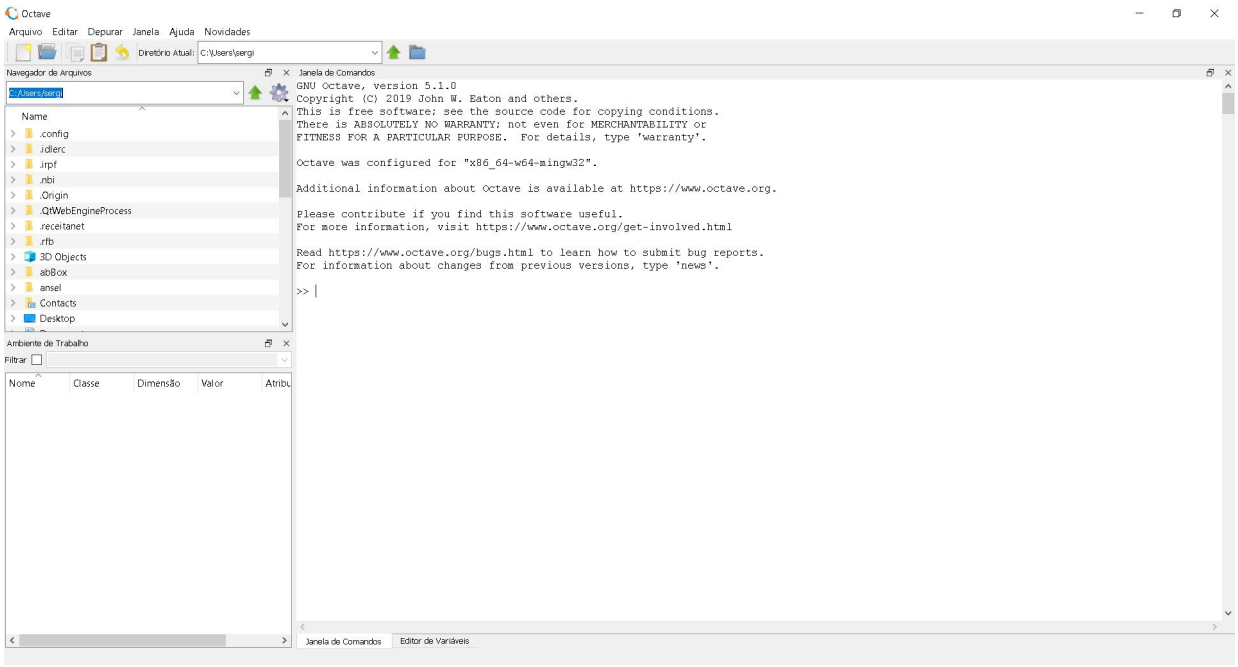
Deste modo, optou-se pelo uso do *GNU Octave*, um *software* dotado de linguagem de programação científica e livre, bastante focada em matemática. Devido a tais características, se mostra como uma ferramenta apropriada para aplicação em sala de aula. De acordo com o próprio site, em tradução livre, temos:

- Sintaxe poderosa orientada à matemática com ferramentas integradas de plotagem e visualização;
- *Software* livre, roda em *GNU / Linux, macOS, BSD e Windows*;
- *Drop-in* compatível com muitos *scripts Matlab*.

Todos os códigos que serão apresentados aqui poderão ser compilados utilizando essa ferramenta, que está disponível para *download* de maneira gratuita em WWW.GNU.ORG/SOFTWARE/OCTAVE/, como também pode ser utilizada de maneira online, sem necessidade de *download* diretamente pelo site HTTPS://OCTAVE-ONLINE.NET/.

Após instalá-lo e executá-lo em seu computador, a tela que irá aparecer será como a apresentada a seguir:

Figura 3.2: Programa *GNU Octave* em execução

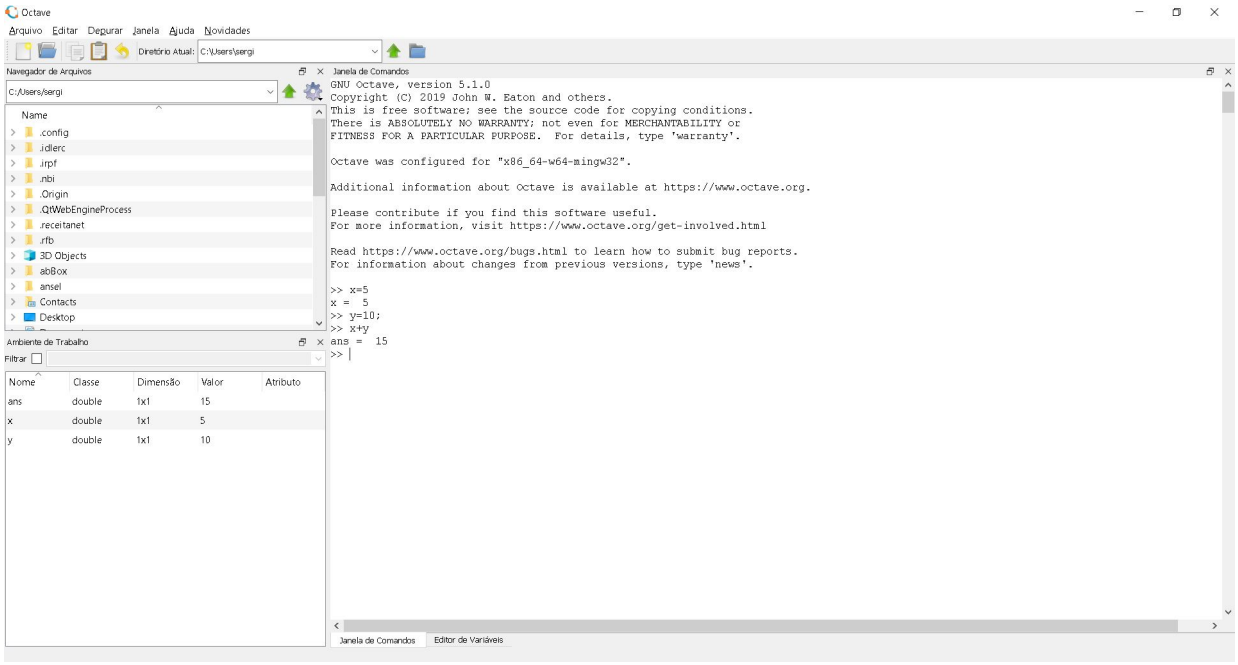


Fonte: O autor

É possível observar que temos três ambientes distintos nessa tela: o navegador de arquivos, o ambiente de trabalho e a janela de comandos. O navegador de arquivos é a ferramente que possibilita que o usuário busque com maior praticidade os arquivos com extensão ".m" (extensão dos arquivos feitos no programa) que deseja compilar, o ambiente de trabalho é onde ficam armazenadas as variáveis que estão sendo trabalhadas no momento (com seus respectivos nomes, classes, dimensões, valores e atributos), e por fim a janela de comandos, que é onde iremos digitar, linha a linha, o que queremos que seja feito, e também poderemos visualizar seus resultados.

Um dos principais motivos de utilizar essa ferramenta no ensino da programação é de que o estudante não precisará se preocupar com a declaração da classe da variável (se o número é inteiro, real, complexo, booleano). Um exemplo disso pode ser demonstrado assim, com uma simples operação de soma de variáveis, utilizando o *GNU Octave*:

Figura 3.3: Soma de duas variáveis



Fonte: O autor

É possível observar no exemplo mostrado acima que apenas escrevendo $x = 5$ pudemos atribuir o valor 5 à variável x, o que é mostrado logo em seguida. Após isso, o valor 10 foi atribuído a y, utilizando a escrita $y = 10$; mas veja que como foi usado o símbolo ";" logo após a definição não houve uma linha abaixo para mostrar que a variável recebeu o valor. Esse artifício é utilizado quando não queremos poluir a tela com muitas informações, visto que podemos visualizar os valores no ambiente de trabalho ao lado. Logo após, foi escrito $x + y$ para executar a soma das duas variáveis, e o resultado foi apresentado como $ans = 15$. Como não foi mostrado se o valor da soma deveria ser atribuído a alguma variável, o próprio compilador tratou de atribuir tal valor a uma variável *ans* (que seria equivalente ao termo em inglês *answer*, em tradução livre, o equivalente a *resposta*).

É possível perceber a praticidade na escrita dos códigos e como facilita para alunos que tiveram pouco ou nenhum contato com computação até o momento. Para ser possível implementar códigos com foco em matemática devemos conhecer os operadores necessários e funções do programa que irão nos ajudar.

3.2.1 Operadores Aritméticos

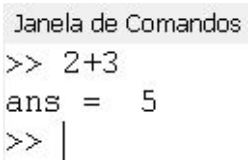
Segundo Ferrari e Cechinel (2008), os operadores aritméticos são aqueles em que os operandos são valores do tipo numérico (inteiro ou real). Esses valores numéricos podem ser acessados por meio de identificadores constantes ou por meio de variáveis. As operações aritméticas fundamentais são: adição, subtração, multiplicação, divisão, potenciação e o resto (módulo). Observe a tabela que mostra esses principais operadores aritméticos e seus respectivos símbolos, seguidos de exemplos de como utilizar:

Tabela 3.1: **Operadores Aritméticos**

Operação	**Operador**	**Exemplo**
Adição	+	2+3
Subtração	-	9-5
Multiplicação	*	-4*8
Divisão	/	6.3/10
Potenciação	** ou ^	8**3 ou 8^3
Resto (Módulo)	mod	mod(7,4)

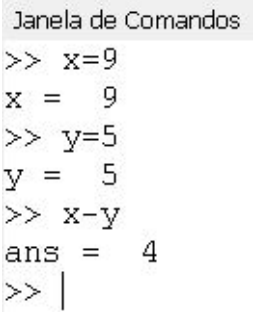
Veja como isso fica na tela do *GNU Octave*, tanto com constantes, como atribuindo previamente valores às variáveis e utilizando os operadores:

Figura 3.4: Soma de duas constantes



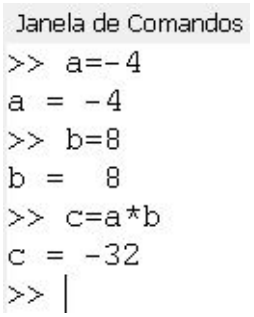
Fonte: O autor

Figura 3.5: Subtração de variáveis



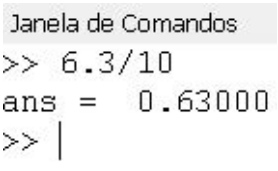
Fonte: O autor

Figura 3.6: Multiplicação de variáveis com atribuição à terceira variável



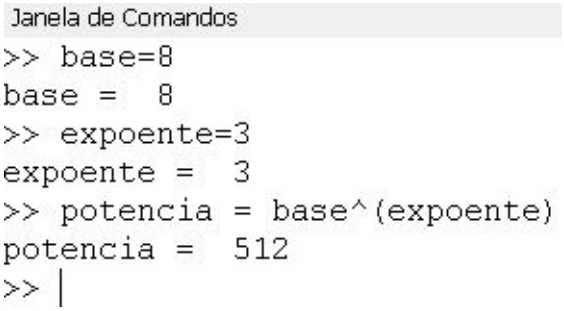
Fonte: O autor

Figura 3.7: Divisão entre números reais



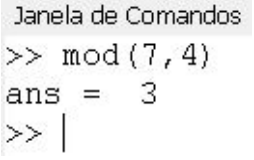
Fonte: O autor

Figura 3.8: Potenciação entre variáveis



Fonte: O autor

Figura 3.9: Resto da divisão de dois inteiros



Fonte: O autor

3.2.2 Operadores Relacionais e Lógicos

Os operadores relacionais são aqueles que irão verificar a relação entre duas constantes ou duas variáveis numéricas. Já os operadores lógicos apresentam o comportamento dos conectivos lógicos estudados na lógica proposicional. O que ambos os tipos de operadores tem em comum é o fato de que ao utilizá-los retornam como saída apenas os valores 0 (FALSO) ou 1 (VERDADEIRO). Observe a definição apresentada por Ribeiro (2018):

> Quando queremos comparar valores, usamos operadores relacionais ou também chamados operadores comparativos. Na comparação, obtemos duas possíveis respostas: True (Verdadeiro) ou False (Falso). Quando uma expressão é avaliada, só há duas possibilidades lógicas: ou ela é verdadeira ou é falsa; não se aceita uma terceira possibilidade. Essa ideia é chamada de valor lógico da expressão e constitui a base da computação: o **1 (verdadeiro)** ou **0 (falso)**, ligado ou desligado, aberto ou fechado (RIBEIRO, 2018).

Para facilitar, relembremos um pouco da lógica proposicional (onde os conectivos lógicos estão presentes).

- *Conectivo "E" (Conjunção)*: Proposições que apresentam o conectivo "E" em sua composição são chamadas de conjunções. Uma conjunção só será dada como verdade se ambas as proposições forem verdadeiras.
  Exemplo.: "Pera é uma fruta **e** o sol é quente". Como ambas as afirmações são verdadeiras, a saída dessa proposição também é verdadeira (valor lógico 1). Caso qualquer uma delas fosse falsa, o resultado seria falso (valor lógico 0).

- *Conectivo "Ou" (Disjunção)*: Proposições que apresentam o conectivo "ou" em sua composição são chamadas de disjunções. Uma disjunção só não será dada como falsa se ambas as proposições forem falsas.
  Exemplo.: "Pera não é uma fruta **e** o sol é quente". Como pelo menos uma das afirmações é verdadeira (o sol é quente), a saída dessa proposição também é verdadeira (valor lógico 1). Caso ambas fossem falsas, o resultado seria falso (valor lógico 0).

- *Negação*: Já a negação não diz respeito a conectar duas proposições, ela altera o sentido lógico de uma proposição. Se for verdadeiro ao negar se torna falso, e se for falso ao negar-se torna verdadeiro.

Exemplo: “O cachorro é uma ave”. Ao ser negado se torna: O cachorro **não** é uma ave.

Tabela 3.2: **Operadores Relacionais**

Operação	**Operador**	**Exemplo**
Maior que	>	2>3
Menor que	<	4<8
Maior ou igual que	>=	8 >= 100
Menor ou igual que	<=	100 <= 20^2
Igual a	==	0==0
Diferente de	∼= ou !=	1∼=1

Tabela 3.3: **Operadores Lógicos**

Operação	**Operador**	**Exemplo**
E	&&	(2>1)&&(5<3^2)
Ou	||	(10>41)||(6<=5)
Negação	!	!0

Para melhor compreensão, veja o que será apresentado no compilador caso você tente efetuar alguma das operações evidenciadas acima:

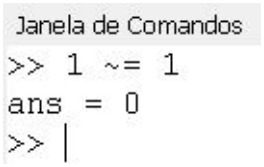
Figura 3.10: Operador relacional "menor ou igual que"

```
Janela de Comandos
>> 100 <= 20^2
ans = 1
>>
```

Fonte: O autor

Observe que o operador relacional "menor ou igual que"está fazendo uma comparação entre dois valores, 100 e 20^2, que representa 20^2, e sabemos que $20^2 = 20 * 20 = 400$. Portanto, o compilador irá processar essa informação e irá fazer a comparação 100 <= 400. Como sabemos, 100 é de fato um número menor ou igual que 400, por isto a saída será *verdadeira*, com valor lógico igual a 1.

Figura 3.11: Operador relacional "diferente de"

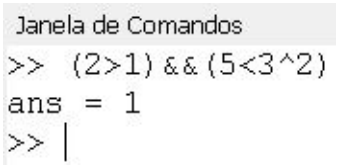


Fonte: O autor

Já nesse exemplo apresentado acima, queremos saber se o número 1 é "diferente de"1. Ora, sabemos que 1 é igual a 1, logo ao escrever que são diferentes estamos lidando com uma afirmativa falsa, e deste modo teremos saída *falsa*, com valor lógico igual a 0.

Veja a implementação dos exemplos de operadores lógicos:

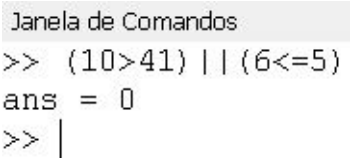
Figura 3.12: Operador lógico "E"



Fonte: O autor

No exemplo acima, como está sendo usado o operador "E"o valor lógico apenas é verdadeiro se ambas as expressões forem verdadeiras. Como no lado esquerdo temos $2 > 1$, sabemos a partir das operações relacionais que isso é verdadeiro, logo possui valor lógico 1. No lado esquerdo temos $5 < 3\hat{}2$, que o compilador irá calcular a potência $3^2 = 9$ e verificará se $5 < 9$. Como sabemos que isso também é verdade, temos o valor lógico igual a 1. Com isso, a operação lógica que sera feita é 1&&1, que tem como resultado 1.

Figura 3.13: Operador lógico "Ou"

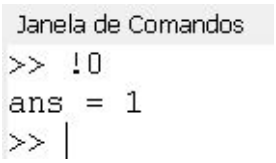


Fonte: O autor

Seguindo o mesmo raciocínio, agora com o operador "OU", o compilador analisa se $10 > 41$ e se $6 <= 5$. Como sabemos, 10 não é maior que 41 e 6 também não é menor ou igual que 5, logo ambas as expressões são falsas, e terão valor lógico igual a 0. Como

no "OU"temos valor lógico 1 quando pelo menos uma das afirmações é verdadeira, nesse caso em que nenhum é verdadeiro teremos saída igual a 0.

Figura 3.14: Operador lógico "Negação"



Fonte: O autor

Por último, a negação de uma expressão. De maneira simples, a negação do valor lógico 0 (falso) será 1 (verdadeiro).

3.2.3 Funções computacionais do *software*

Agora que já estamos familiarizados com o básico e necessário da parte matemática do *GNU Octave* devemos prosseguir com algumas funções úteis para o desenvolvimento dos algoritmos computacionais. O *software* em questão possui milhares de funções embutidas que facilitam bastante o trabalho para pessoas que usam a computação científica em alto nível, diversas delas abreviando vários cálculos matemáticos. Se o leitor tiver curiosidade de conhecer todas as funções oferecidas basta apenas digitar *help –list* e pressionar *enter* na janela de comando que irão ser listadas todas as funções que há para uso no programa.

Inclusive, a ferramenta *help* é de suma importância para o usuário do *GNU Octave*. Sempre que o leitor tiver alguma dúvida sobre alguma função, como quanto à sua funcionalidade ou como utilizá-la, poderá sanar digitando *help NOMEDAFUNÇÃO* e obterá informações detalhadas sobre a função escolhida. Um bom teste para isso que é que comece digitando *help help* (para obter ajuda sobre a própria função *help*), e também faça o mesmo para todas as funções que serão apresentadas a seguir.

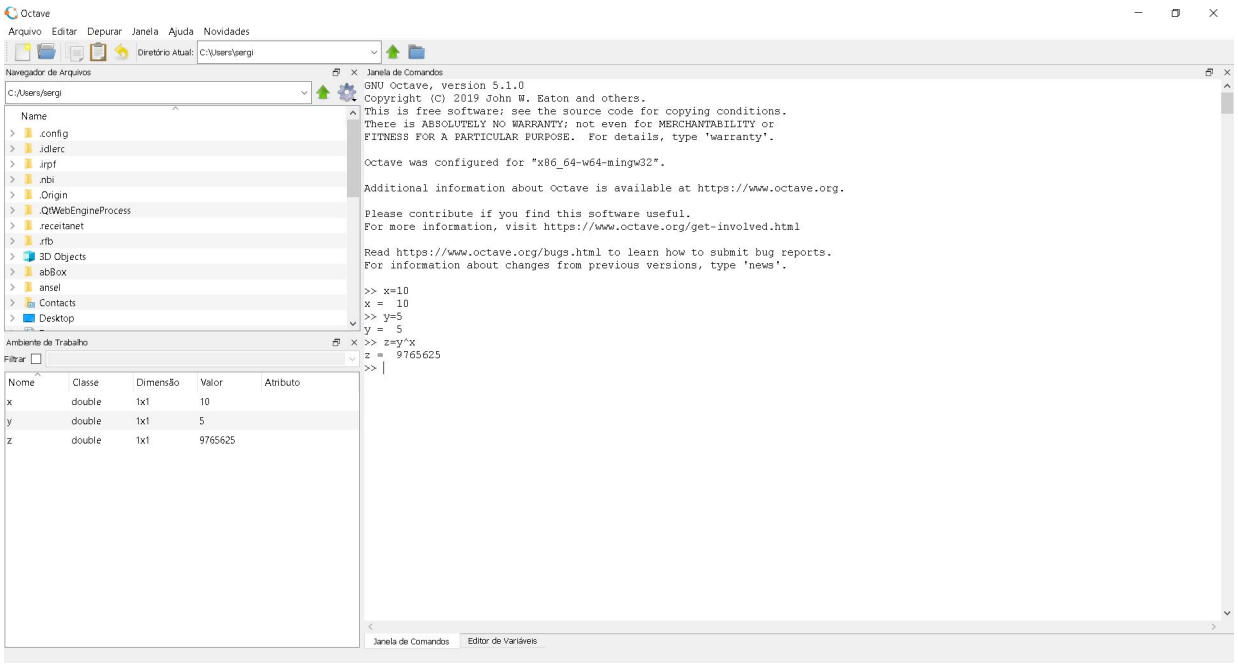
Vale ressaltar que em alguns códigos inclusive disponibilizados aqui poderão ser encontradas outras funções que não as enunciadas abaixo. Nesse caso, cabe a curiosidade do leitor ao tentar compreender seu funcionamento, lembrando que a função *help* auxilia bastante nesse trabalho (de preferência acompanhado de um site de tradução do inglês para o português). A seguir mencionaremos alguns comandos computacionais do *software*

GNU Octave.

3.2.3.1 *clc*

A funçao *clc* serve para "limpar a tela", ou seja, retira linhas de código que já tenham sido digitadas, porém sem limpar variáveis que tenham sido definidas. Observe no exemplo abaixo:

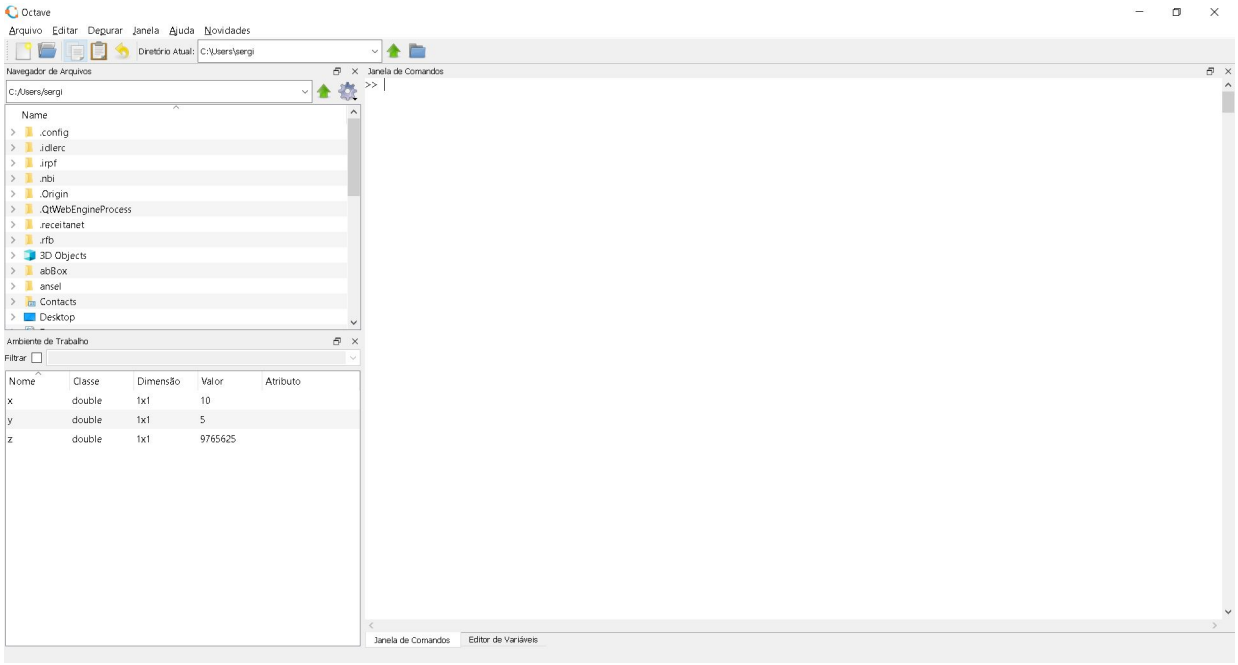
Figura 3.15: Atribuição de valores à três variáveis



Fonte: O autor

Veja que foram digitadas e confirmadas três linhas de código. Primeiramente foi feita a atribuição à variável x, com o comando $x = 10$. Em seguida, a atribuição de $y = 5$. Por último, utilizando os valores definidos de x e y, foi atribuído um valor a z decorrente de uma potenciação de base y e expoente x, com $z = y\hat{}x$. É possível observar no canto esquerdo inferior que os a classe, a dimensão e o valor de cada uma das variáveis já se encontra disponível no ambiente de trabalho do *software*. Após isso, foi digitado uma linha de código contendo apenas o comando *clc*. Pressionando *enter*, o resultado deve ser como o mostrado abaixo:

Figura 3.16: *GNU Octave* após o uso do comando *clc*



Fonte: O autor

Observe que após a utilização do comando temos uma janela de comandos completamente limpa, mais amigável aos usuários. Mesmo com a janela de comandos limpa, no ambiente de trabalho é possível perceber que o valor atribuído às variáveis continua intacto, ou seja, o usuário ainda pode usar as variáveis definidas para operações que sejam desejadas.

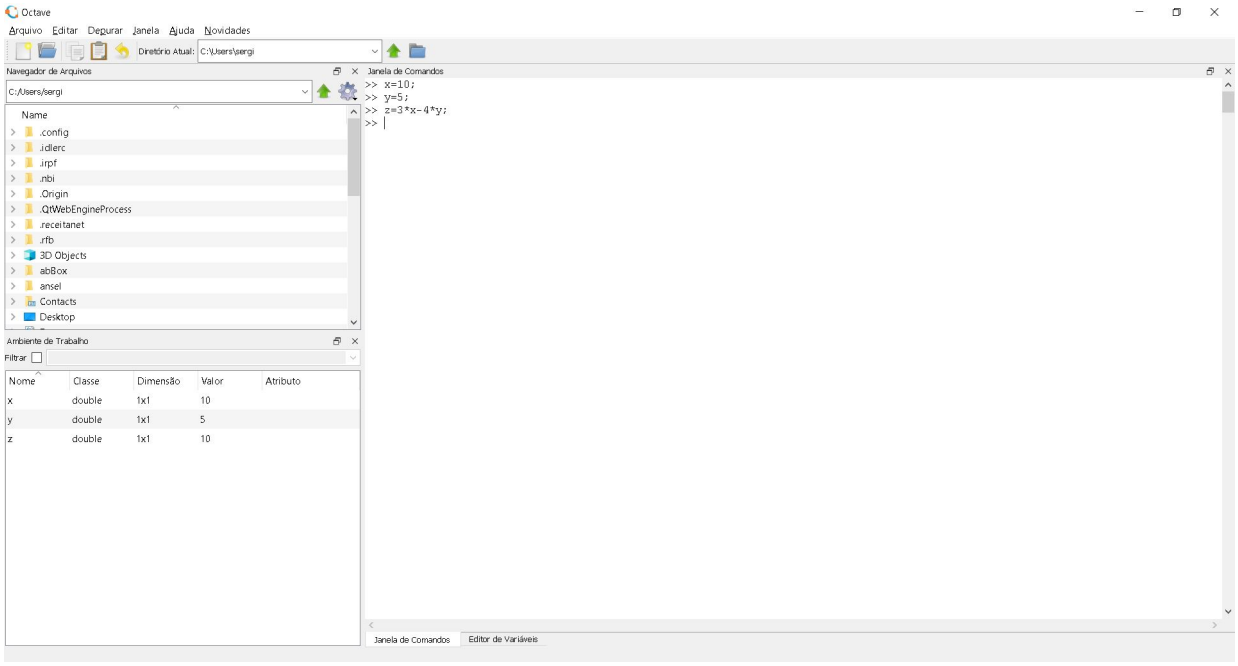
3.2.3.2 *clear*

O comando *clear* também é uma função de limpeza do espaço de trabalho. Dessa vez, utilizando esse comando podemos limpar uma ou mais variáveis do ambiente de trabalho, ou limpar todas de uma vez. Imagine que você possui uma variável x que está trabalhando e deseja agora exclui-la, pois não será mais necessária em seu algoritmo. Para executar esse comando você deve digitar *clear x*, e ao apertar *enter* essa variável não estará mais no ambiente de trabalho.

Caso você esteja trabalhando com mais de uma variável e deseja apagar elas simultaneamente apenas com um comando você deve digitar *clear all*. O parâmetro *all*, em tradução livre do inglês, significa *todas*, logo ao utilizar o comando dessa maneira você estará limpando todas as variáveis do seu ambiente de trabalho. Observe abaixo a tela

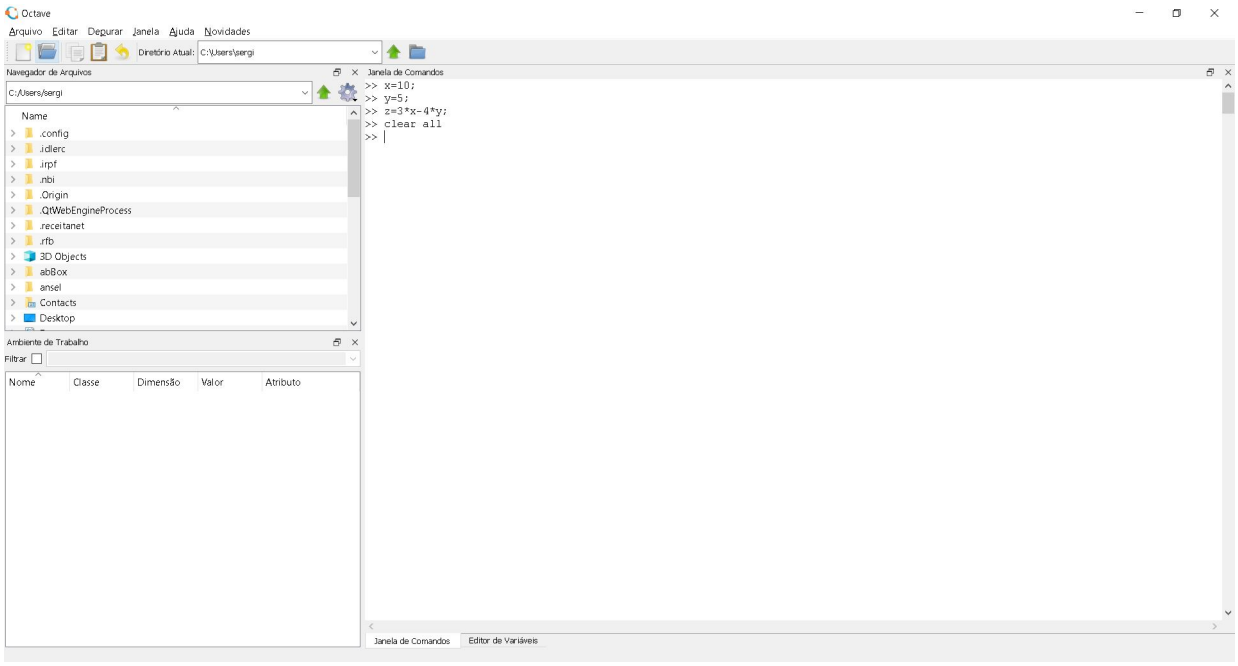
do *GNU Octave* antes e depois do uso do comando:

Figura 3.17: Tela com definição de variáveis, antes de utilizar *clear all*



Fonte: O autor

Figura 3.18: Tela após o uso de *clear all*



Fonte: O autor

Repare que as linhas de comando após o uso do comando continuam aparecendo na janela de comando, visto que o *clear* apenas apaga as variáveis do ambiente de

trabalho.

3.2.3.3 *disp*

Esta função tem papel de saída, ou seja, ela irá exibir na tela o que o usuário desejar. Se o programador deseja imprimir um número na tela, por exemplo, o número 10, deve apenas inserir *disp(10)* e apertar *enter*. Se o parâmetro desejado for de texto, o usuário deverá colocar o texto entre aspas simples (’). Repare que nessa função o parâmetro desejado vai dentro de parênteses. Veja na imagem abaixo dois exemplos da função *disp* na mesma tela, um com número e um com texto:

Figura 3.19: Utilização da função *disp*

```
Janela de Comandos
>> disp(25)
 25
>> disp('Adorando o GNU Octave!')
Adorando o GNU Octave!
>>
```

Fonte: O autor

3.2.3.4 *input*

Esta função tem papel de entrada e saída. Ela exibe um texto, assim como a função *disp* e espera uma entrada do usuário, que pode ser atribuída a uma variável e ser utilizada posteriormente. Assim como na função anterior, o parâmetro de saída deve estar dentro de parênteses, e sendo utilizado texto deve estar entre aspas simples (’). Observe a atribuição de um valor a uma variável utilizando a função *input*:

Figura 3.20: Utilização da função *input*

```
Janela de Comandos
>> x=input('Qual o valor de x?:  ')
Qual o valor de x?:  2^5
x =  32
>>
```

Fonte: O autor

Veja que ao inserir o valor de 2^5 para x o *software* se encarrega de atribuir o valor já feita a operação de potenciação, que tem como resultado 32.

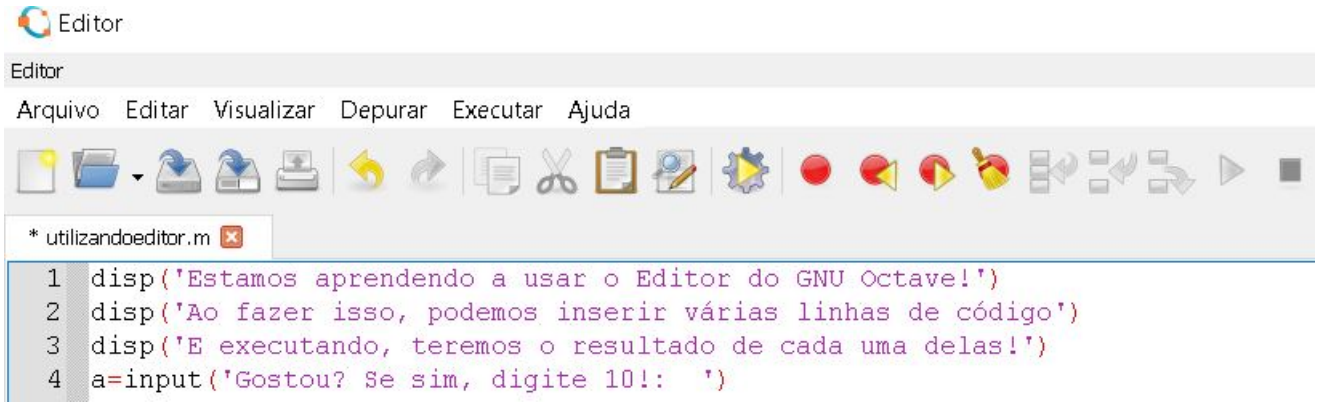
3.2.3.5 *edit*

Essa função é crucial para a construção algorítmica no *GNU Octave*. Ao utilizar *edit* abrirá uma nova janela do programa nomeado *Editor*. Nesse editor de texto poderemos escrever *scripts* que poderão ser salvos no computador, na extensão *.m*, os *arquivos M*. Na utilização do Editor o usuário poderá digitar várias linhas de código sequenciadamente, construindo um algoritmo com vários comandos. Ao terminar seu código, você poderá buscar o ícone *Salvar Arquivo e Executá-lo* (vide imagem a seguir) ou apertar a tecla *F5* do seu computador. O nome do arquivo fica à sua escolha, porém não esqueça de salvar como um arquivo M. Essa ferramenta é bastante poderosa e evita que tenhamos que digitar as mesmas linhas de código várias e várias vezes, além de poder compartilhar códigos entre amigos sem precisar copiar todo o texto para isso. A seguir temos também um exemplo da utilização do Editor.

Figura 3.21: Ícone de "Salvar Arquivo e Executá-lo"

Fonte: O autor

Figura 3.22: Tela do Editor em uso



Fonte: O autor

Figura 3.23: Tela da janela de comandos após executar *script* do editor

```
Janela de Comandos
>> utilizandoeditor

Estamos aprendendo a usar o Editor do GNU Octave!
Ao fazer isso, podemos inserir várias linhas de código
E executando, teremos o resultado de cada uma delas!
Gostou? Se sim, digite 10!:  10
a =  10
>>
```

Fonte: O autor

3.2.3.6 *if*

Essa estrutura é uma das mais famosas na maioria das linguagens de programação. Pode ser chamada de uma estrutura de *seleção* ou *condicional*, de modo que irá realizar determinado comando *SE* (tradução do inglês para português de *if*) algo tiver valor lógico verdadeiro (utilizando os operadores relacionais e lógicos estudados previamente).

Além de poder utilizar o *if* isoladamente apenas para um caso desejado, também é possível utilizá-lo com o comando que realizará determinado comando caso o valor lógico da expressão seja contrário ao que se interessa, conhecido como *else* (em tradução livre, *SE NÃO*). Veja abaixo exemplos do *if* quando usado isoladamente e em conjunto com o *else*.

Figura 3.24: Programa utilizando função *if*

```
funcaoIF.m
1  x=input('Insira um valor para x: ');
2  if(x>0)
3    disp('x é positivo')
4  end
```

Fonte: O autor

Figura 3.25: Programa utilizando função *if* e *else*

funcaoIFeELSE.m

```
1 x=input('Insira um valor para x: ');
2 if(x>0)
3   disp('x é positivo')
4 else
5   disp('x é igual a zero ou negativo')
6 end
```

Fonte: O autor

Tente escrever cada um dos exemplos acima no seu Editor no *GNU Octave* para observar como funciona a função em cada caso!

É possível perceber que a estrutura em ambos os códigos é similar: ao usar *if* temos uma condição entre parênteses que será analisada como verdadeira ou falsa, e de acordo com seu valor lógico teremos algum comando a ser executado. Observe que no primeiro exemplo o usuário irá digitar um valor para x, e após tal valor ser atribuído à variável temos uma condição no nosso *if* para mostrar um texto na tela: o texto só será mostrado caso tenhamos $x > 0$. Ao colocar um número maior que zero teremos a resposta na tela que x é positivo, e caso você insira para x um valor menor ou igual a zero não terá resposta nenhuma, pois nenhuma linha de código foi escrita para fazer algo além disso.

Todavia, no código subsequente temos uma estrutura similar adicionada de mais duas linhas, sendo essas a função *else* e o comando que lhe diz respeito. Nesse caso teremos que novamente inserir um valor para x e que será analisado dentro do parênteses do *if*, e assim como no primeiro código se x for maior que zero a mesma mensagem anteriormente dita será impressa em tela. Mas agora também temos um *senão* adicionado ao nosso problema, ou seja, caso o número não seja maior que zero outra mensagem irá aparecer, concluindo que com essa estrutura cobrimos os dois casos possíveis.

Quanto à sintaxe da escrita da função veja que nos dois casos ela possui no final o comando *end*. Tal comando especifica que as condições analisadas terminam ali, finalizando o uso da função *if* ou *ifelse*.

3.2.3.7 *switch*

Essa estrutura, em pequena escala, tem características similares à previamente apresentada. Essa função irá analisar uma expressão e poderá executar determinado

comando para cada um dos vários casos possíveis como resultado de tal expressão.

Um código utilizando *switch* acaba sendo de entendimento mais fácil, pois os casos são mais explícitos em sua sintaxe. A função é muito útil para a criação de "menus"nos programas, por evidenciar com clareza cada uma de suas opções.

Um *switch* pode utilizar como parâmetro uma variável, e dependendo do valor que for atribuído previamente a essa variável ela irá chamar um *case* (em tradução livre, *caso*) diferente. Você poderá observar que podemos especificar os comandos que queremos para cada caso, como também ao final criar um caso para aquela condição que não atender a nenhum dos casos anteriores, chamado de *otherwise* (que significaria *de outra forma*, ou seja, de maneira diferente dos casos evidenciados antes).

A depender da edição que você estiver utilizando do *GNU Octave*, a função poderá ser encerrada apenas com o comando *end*, assim como *if*, ou para que num programa mais complexo fique mais claro o que está sendo encerrado poderá utilizar um *endswitch*.

Observe abaixo um código utilizando a estrutura *switch* para a criação de uma pequena calculadora de dois números:

Figura 3.26: Programa utilizando função *switch*

funcaoSWITCH.m

```
1  x=input('Digite um numero x: ');
2  y=input('Digite um numero y: ');
3  disp('Digite 1 se deseja uma SOMA')
4  disp('Digite 2 se deseja uma SUBTRACAO')
5  disp('Digite 3 se deseja uma MULTIPLICACAO')
6  opcao=input('Digite 4 se deseja uma DIVISAO: ');
7  switch (opcao)
8
9  case 1
10 disp(x+y)
11
12 case 2
13 disp(x-y)
14
15 case 3
16 disp(x*y)
17
18 case 4
19 disp(x/y)
20
21 otherwise
22 disp('Nao foi inserida opcao válida!')
23
24 endswitch
```

Fonte: O autor

3.2.3.8 *for*

Outro tipo de estrutura presente nas linguagens de programação são as estruturas de *repetição*. Entre elas, uma das mais famosas é o *for* (em tradução, *para*), que indica que uma determinada função irá repetir comando de indice **para** (ou **até**) outro valor.

No *GNU Octave* é possível colocar mais de um valor numérico em uma variável, criando matrizes da dimensão desejada. Os *vetores* na programação são bastante famosos, e são matrizes normalmente formadas por uma coluna e várias linhas ou por uma linha e várias colunas. A partir do comando aqui apresentado é possível criar um programa que o usuário determina valores para cada uma das posições de uma matriz ou vetor.

Assim como na função *if* e *switch*, é importante que o aluno que elabora o algoritmo indique com *end* onde termina a iteração do *for*, ou um *endfor* para maior detalhamento de qual função deve ser encerrada naquela linha de comando. Cada uma das linhas de comando implementadas dentro dessa função será repetida cada vez que se

avançar na iteração da função.

Iremos apresentar um dos conceitos mais simples utilizando o *for*, um contador. Esse contador irá apenas imprimir valores na tela, dentro do intervalo escolhido pelo usuário ao executar o programa, um exemplo de como mostrar os números inteiros dentro de determinado intervalo. Veja o código e tente fazê-lo em seu computador para melhor entendimento:

Figura 3.27: Programa utilizando função *for*

funcaoFOR.m

```
1  a=input('Indique o primeiro valor do intervalo: ');
2  b=input('Indique o ultimo valor do intervalo: ');
3  for (i=a:b)
4    disp(i)
5  endfor
```

Fonte: O autor

Observe a sintaxe dentro do parênteses do *for*. Foi definida uma variável auxiliar "i"de modo a variar de "a"até "b", com o símbolo ":"indicando o "até". Por padrão, o "passo"do *for* é unitário (implemente o código e veja o resultado!), por isso nesse algoritmo os números vão aparecendo com incremento sempre de uma unidade. Experimente aumentar (ou diminuir) o passo da função colocando o tamanho o número que você desejar, como por exemplo: $for(i = a : 3 : b)$. Você verá que nesse caso os incrementos irão ser de três em três unidades. Você também pode alterar para números decimais ou negativos, desde que colocando os extremos do intervalo na ordem correta.

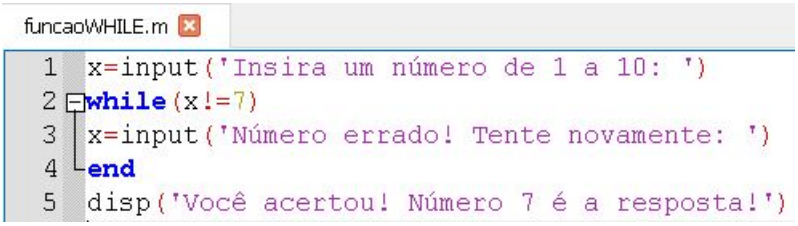
3.2.3.9 *while*

Assim como o *for*, essa também se trata de uma estrutura de *repetição*. O *while* (significa *enquanto*, em inglês) irá continuar operando suas linhas de comando enquanto a condição de seus parênteses tiver valor lógico verdadeiro, ou seja, apenas irá sair dessa função quando o valor lógico for falso.

Esse comando será um dos mais cruciais quando tivermos que trabalhar com **restrições** nos nossos algoritmos matemáticos, pois para alguns problemas não podemos continuar com o procedimento do algoritmo **enquanto** certas condições não forem preenchidas. Entretanto, aqui para exemplificar de forma simples, iremos construir um joguinho de adivinhação no qual o programa não encerra **enquanto** o usuário não acerta

o número que previamente colocamos no código. Observe a seguir:

Figura 3.28: Programa utilizando função *while*



Fonte: O autor

É possível perceber que nesse programa você pode tentar qualquer valor diferente de 7, porém você só observará o texto inserido no comando da última linha quando inserir 7 como valor de x. Isso se dá porque a condição colocada no parênteses do *while* foi que a estrutura de repetição iria continuar funcionando **enquanto** tivessemos x diferente de 7, ou seja, $x!=7$. Logo, quando tivermos $x == 7$ (não confundir o comando "="de atribuição de valores à uma variável com o "==", que verifica se a igualdade é válida ou não) teremos o contrário do que a função busca, portanto a função já pode ser encerrada.

Observe que assim como nas demais últimas funções aqui apresentadas temos o comando *end* para garantir o final da estrutura. Um *endwhile* também pode fazer esse papel, e deixa um código mais complexo com uma melhor organização.

Com o conhecimento dessas principais funções básicas do *GNU Octave* é possível construir diversos algoritmos matemáticos, basta usar a imaginação.

4 Pensamento Computacional Aplicado ao Ensino de Matemática

Dentro do escopo do que fora apresentado, buscou-se introduzir uma proposta pedagógica que fosse capaz de proporcionar ao alunado uma maneira de aliar os conhecimentos prévios matemáticos (ou até mesmo na introdução de novos conceitos) com os conhecimentos aplicados do estudo da lógica e pensamento computacional, trazendo a linguagem de programação, e consequentemente o computador, como ferramenta na aprendizagem matemática.

4.1 Proposta Pedagógica

O foco para a metodologia aqui apresentada foram alunos do ensino médio, além de discentes de Licenciatura em Matemática e professores de Matemática no ensino básico em geral, pois estes possuem uma vivência com uma maior gama de conteúdos trazidos desde o ensino fundamental, que devem ser aliados com o ensino da lógica computacional, que pode ser abordado a partir da primeira série do ensino médio.

Dessa maneira, se fez necessária a definição de uma *sequência didática* que trouxesse consigo os conceitos aqui previamente apresentados, de uma maneira que fosse aprazível ao público alvo e que seja capaz de fazer com que se sintam à vontade com a nova maneira de ver matemática.

Uma proposta de sequência didática é estabelecida a seguir:

Tabela 4.1: **Sequência Didática**

Aula 1	*Estudo de Algoritmos*
Aula 2	*Operadores e Lógica Computacional*
Aula 3	*GNU Octave*
Aula 4	*Comandos Básicos e Primeiras Impressões*
Aula 5	*Aplicações na Matemática*

É recomendado que cada uma dessas aulas tenha tempo suficiente para que os

alunos tentem esgotar suas dúvidas sobre cada um dos conteúdos apresentados, mesmo condensando apenas o mais importante a ser visto em cada tópico. Podem ser 5 (cinco) aulas com duração de 2 (duas) horas cada, ou fracionar cada tópico em duas aulas, a depender da carga horária escolar. Importante ressaltar que as aulas devem integrar teoria e prática, ou seja, é indicado que as mesmas sejam ministradas em laboratório de informática dotado de aparato necessário (computadores, *notebooks*) e se possível, também, quadro branco, pincéis e projetor multimídia. Claramente, algumas das aulas introdutórias não precisarão dispor de todos esses materiais, podendo ser ministradas apenas com quadro e pincel (ou giz). O ensino do pensamento computacional pode também ser apresentado sem necessariamente utilizar os computadores, apresentando os algoritmos no quadro e analisando a linearidade de cada ação a ser tomada, caso a escola não disponha dos equipamentos necessários, entretanto, para um maior engajamento do aluno, é fortemente recomendado o uso dos computadores. Após esse período de apresentação da ferramenta, atividades subsequentes podem ser integradas ao currículo, como forma de mostrar problemas e fórmulas matemáticas com o auxílio do computador. As utilizações podem ser as mais diversas, sempre possibilitando muitos exemplos para serem mostrados.

As atividades a serem propostas à classe devem preferencialmente ser discutidas em grupos de alunos (entre 3 e 5), para que juntos possam buscar alternativas de resolução e compartilhar ideias e dúvidas com todos os outros. Dessa maneira, eles perceberão que não existe apenas uma maneira correta de resolver um problema.

Na próxima seção serão mostradas propostas de como proceder nas aulas, com algumas atividades para resolução junto ao alunos (como também desafios aos mesmos), apresentando objetivos, metodologia e verificação da aprendizagem.

4.2 Sequência Didática

Para a realização das atividades propostas, se faz necessário que os alunos tenham no momento da aula acesso à computadores ou *notebooks*, com a instalação prévia do *software* aqui sugerido, o *GNU Octave*.

4.2.1 Aula 1 - Estudo de Algoritmos

4.2.1.1 Objetivos

- Introdução do primeiro contato dos alunos com o conceito de *algoritmo*.

4.2.1.2 Conteúdos Programáticos

- O que é algoritmo?
- Algoritmos cotidianos e aplicação de algoritmos na matemática.
- Representações algorítmicas (descrição narrativa, fluxograma e pseudocódigo).

4.2.1.3 Metodologia

Nesse primeiro encontro, o professor irá introduzir o conceito de algoritmos à turma e mostrar como estes estão presentes tanto no dia a dia como também no mundo das ciências de forma geral. A partir daí, irá buscar o contato com os alunos para que os mesmos citem exemplos de onde esse conceito é encontrado no estudo da matemática.

Após os alunos citarem alguns exemplos, o professor deve mostrar as diferentes maneiras de como representar um algoritmo, propondo e representando um mesmo código das três diferentes maneiras (descrição narrativa, fluxograma e pseudocódigo).

4.2.1.4 Verificação da Aprendizagem

Nessa etapa deve-se questionar ao aluno quais atividades do dia a dia dele que podem ser descritas como algoritmo, e a partir disso ele deve mostrá-la por meio de uma ou mais das representações apresentadas. Após isso, deve buscar também pensar alguns algoritmos matemáticos diferentes dos que foram apresentados pelo professor e discuti-los com os colegas de classe.

4.2.2 Aula 2 - Operadores e Lógica Computacional

4.2.2.1 Objetivos

- Revisão básica dos operadores aritméticos comuns e introdução à logica proposicional e sua significância na computação.

4.2.2.2 Conteúdos Programáticos

- Operadores aritméticos, relacionais e lógicos.
- Como eles nos auxiliam no estudo dos algoritmos?
- Como utilizá-los na programação de computadores?

4.2.2.3 Metodologia

O professor irá fazer uma breve revisão dos operadores aritméticos comuns (soma, subtração, multiplicação, divisão e etc) e deverá dar atenção especial ao conteúdo de lógica, este que muitas vezes não é apresentado de maneira assertiva no ensino básico.

Deverão ser apresentados os principais conectivos da lógica proposicional e fazer a análise de suas tabelas verdade para que o aluno entenda quando uma relação terá valor lógico verdadeiro ou falso. Após isso, deve-se mostrar como em vários algoritmos discutidos na aula anterior era necessário a análise de alguma "pergunta"para saber qual o próximo passo a seguir dentro dos comandos dos algoritmos.

4.2.2.4 Verificação da Aprendizagem

O aluno deve buscar trazer para sala de aula exemplos de como utilizar os operadores aritméticos nos algoritmos, e também buscar construir fluxogramas de algoritmos que observam um valor lógico para saber onde seguir em um ponto de decisão. Deve-se construir o pseudocódigo desse fluxograma como complemento.

4.2.3 Aula 3 - *GNU Octave*

4.2.3.1 Objetivos

- Primeiro contato dos estudantes com o *software* escolhido, fazendo uso das operações matemáticas triviais apresentadas previamente.

4.2.3.2 Conteúdos Programáticos

- Como funciona uma linguagem de programação?
- O que é o *GNU Octave* e por que utiliza-lo?
- Utilizando os operadores matemáticos numa plataforma computacional.

4.2.3.3 Metodologia

Nessa etapa o professor deve dar uma breve explanação sobre o que é uma linguagem de programação e qual sua importância na história da computação. Feito isso, deve apresentar o *GNU Octave* e dizer o que motivou à trazer aquele ambiente de programação para sala de aula, explicar sobre suas aplicações práticas e motivar a turma sobre a ferramenta.

Em seguida deve ser apresentada na janela de comandos do *software* como proceder para utilizar os operadores matemáticos. É recomendado começar com operações aritméticas simples, apenas com números, e em seguida seguir para atribuição de valores com variáveis, soma de variáveis, até operações relacionais e lógicas. É importante trabalhar o conceito de valor lógico verdadeiro ou falso, de acordo com a saída de uma operação relacional ou lógica.

Podem ser propostas as seguintes atividades:

Atividade 1: Crie uma variável x e atribua uma valor numérico a ela. Em seguida crie uma variável y com valor numérico distinto de x. Utilize os operadores relacionais que você conhece (ex.: $x \leq y$, entre outros) e analise a variável *ans* após apertar *enter*. O que cada um dos resultados significa? Discuta com seus colegas.

Atividade 2: Após analisar o resultado de cada uma das operações relacionais, adicione um operador lógico entre algumas das expressões tratadas no exercício

anterior (ex.: $(x > y)\&\&(x! = y)$). O que o resultado dessa expressão quer dizer? Discuta com seus colegas.

4.2.3.4 Verificação da Aprendizagem

Nessa aula, a verificação será dada a partir das atividades trabalhadas em sala, com a exposição do resultado que cada grupo encontrou, e com as respostas dadas por eles a respeito das perguntas dos comandos finais das questões. Concomitantemente, o professor deve esclarecer e ratificar o funcionamento de cada um dos elementos para mitigar possíveis questionamentos sobressalentes da parte teórica.

4.2.4 Aula 4 - Comandos Básicos

4.2.4.1 Objetivos

- Apresentar aos alunos as funções nativas do *software* e que muitas vezes são comuns a outros ambientes de programação.

4.2.4.2 Conteúdos Programáticos

- Principais comandos e funções do *GNU Octave*.
- Sintaxe de cada função do *GNU Octave*.
- Editor e escrita de *scripts*.

4.2.4.3 Metodologia

Esse encontro deve ser iniciado apresentando as funções mais úteis para os algoritmos matemáticos que serão trabalhados posteriormente. É importante apresentar os elementos relevantes na sintaxe para que não haja erros de execução nos programas. Os alunos também devem ser instruídos a observarem os erros que o compilador aponta, para que assim possam perceber quando apenas houve um erro na escrita do comando.

É importante ressaltar a função *help*, pois a partir desta o aluno curioso poderá conhecer várias outras funções, além de aprimorar as que foram apresentadas anterior-

mente, fazendo melhor uso e criando programas mais concisos.
Apresentaremos aqui propostas de atividades para esta etapa:

Atividade 1: Crie seu primeiro programa de computador. Imprima na tela a mensagem "Olá, Mundo!"(Essa é uma mensagem clássica para os iniciantes de qualquer linguagem de programação).

Atividade 2: Crie algum programa com curiosidade matemática.
Dica 1: Peça ao usuário digitar um número, e em seguida imprima na tela o sucessor, o antecessor, a metade e o dobro.
Dica 2: Peça ao usuário para digitar um primeiro número, em seguida um segundo número, e caso o segundo número seja menor que o primeiro, o programa deve ser encerrado; caso o segundo número seja maior que o primeiro, imprima todos os números entre o primeiro e o último (sem incluí-los).

Atividade 3: Crie um jogo com números.
Dica: Crie um jogo de adivinhação entre duas pessoas. A primeira pessoa deve inserir um número, após isso a tela deve ser limpa (para que o segundo usuário não veja o número inserido), e então o segundo usuário tenta adivinhar o número inserido pelo primeiro usuário. Caso ele acerte, uma mensagem de parabéns deve aparecer na tela, caso contrário, deve aparecer a opção para ele tentar novamente e indicando se o número a ser acertado é maior ou menor do que o que ele escolheu anteriormente, e assim até que haja o acerto.

4.2.4.4 Verificação da Aprendizagem

O professor deve acompanhar o desenvolvimento de cada uma das atividades propostas, analisando o método que cada grupo de alunos utilizou, seus acertos e seus erros. Caso tenham problemas, devem adicionar *comentários* (por meio do caractere "%") nas suas linhas de código para tentar ajudá-los. No final da aula deve expor como cada grupo escreveu o código e discutir entre todos eles.

4.2.5 Aula 5 - Aplicações na Matemática

4.2.5.1 Objetivos

- Evidenciar aos alunos como a computação é uma poderosa ferramenta para resolução de problemas matemáticos, e fazer com que eles sejam capazes de construir seus próprios algoritmos matemáticos computacionais.

4.2.5.2 Conteúdos Programáticos

- Conteúdos matemáticos na forma de algoritmos.
- Restrições em códigos computacionais.
- Editor na construção de códigos matemáticos.

4.2.5.3 Metodologia

O professor deverá iniciar a aula apresentando algoritmos matemáticos que podem ser facilmente colocados como códigos numa linguagem de programação. Feito isso, deve evidenciar o papel das restrições em cada um destes códigos e como elas podem ser implementadas e contornadas com as funções estudadas previamente.

Alguns dos códigos que podem ser aplicados no ensino médio e que possuem tais características são:

- O algoritmo resolutivo para uma equação do segundo grau, conhecido como a Fórmula de Bháskara;
- Termo geral da progressão aritmética (P.A);
- Programa para calcular médias do Exame Nacional do Ensino Médio (ENEM) de uma universidade (média ponderada).

Essas são apenas algumas sugestões de algoritmos eficazes para mostrar a utilização de restrições em pontos cruciais. Mais alguns códigos encontram-se disponíveis no Anexo deste trabalho. Observe como os códigos podem ser evidenciados em sala de aula:

Código 4.1: Fórmula de Bháskara

```
% Formula de Bhaskara
clear all; clc;
display('Considere uma equacao do segundo grau
        do tipo Ax^2 + Bx + C')
A=input('Insira o valor do coeficiente A: ');
while(A==0)
    A=input('Insira o valor do coeficiente A, que seja
       diferente de 0: ');
end
B=input('Insira o valor do coeficiente B: ');
C=input('Insira o valor do coeficiente C: ');
clc;
delta=B^2-4*A*C;
if(delta<0)
    display('Nao possui raizes reais, serao imaginarias!')
end
x1=(-B+sqrt(delta))/(2*A);
x2=(-B-sqrt(delta))/(2*A);
if(delta == 0)
    display('Como o Delta=0, so existe uma raiz real!')
end
display('As raizes para a equacao dada sao:')
x1
x2
```

De modo que fique claro aos alunos como proceder para criar um código dessa maneira, peça aos mesmos que resolvam uma equação do segundo grau qualquer aplicando a Fórmula de Bháskara com a equação na forma $ax^2 + bx + c = 0$ (ex.: $x^2 - 5x + 6 = 0$). Após todos os grupos resolverem e encontrarem as raízes, o professor deve perguntar aos mesmos quais os passos que eles tiveram que cumprir para chegar ao resultado. Feito isso, eles devem ordenar os passos e escrever na forma de um pseudocódigo.

O professor deve agir como um facilitador, mostrando algumas restrições ao

problema: o valor do coeficiente a deve ser um valor diferente de zero (caso contrário não será equação do segundo grau), para que haja resultados reais o valor do "delta"(Δ) deve ser maior ou igual a zero, entre outras restrições possíveis.

Feito isso, deve ser analisada linha por linha o código apresentado pelo professor, pedindo para que cada grupo diga, sequencialmente, o que uma ou duas linhas de comando significam para o código em questão. O professor deve dar o enfoque necessário às restrições e como elas podem ser implementadas (por meio das funções vistas anteriormente) e o seu papel no desempenho correto do programa.

Código 4.2: Termo Geral da P.A

```
clear all; clc;
display('Calculo do termo geral An de uma P.A com termo
   inicial A1 e razao r')
opcao=1;
while(opcao==1)
a_1=input('Insira o valor de A1: ');
r=input('Insira o valor de r: ');
n=input('Digite a posicao do termo desejado: ');
while(n<=0||n~=floor(n))
    n=input('Digite um valor positivo e inteiro para n: ');
endwhile
a_n = a_1 + (n-1)*r
opcao=input('Usar novamente? 1 -> Sim // Qualquer outro
   numero -> Nao:  ');
clc;
endwhile
display('Obrigado por usar nosso programa!')
```

Para o código acima temos a ideia similar à apresentada anteriormente. Primeiro, o professor deve solicitar que os alunos reflitam sobre quais informações são necessárias absorver de uma questão que trata de progressão aritmética quando desejamos descobrir um termo geral da mesma. Dessa maneira, mostre-os que a partir do termo inicial, razão e posição que se deseja, podemos encontrar qualquer termo regido por essa sequência.

Nesse caso, sabemos que o "a_1"(primeiro termo da progressão aritmética), assim como sua razão "r"podem ser quaisquer números reais desejados. O professor deve observar se os alunos entendem que o índice de uma sequência deve ser obrigatoriamente um número inteiro e positivo, pois essa é uma restrição crucial que deve haver no código. Um detalhe dessa estrutura é que foi adicionado um *while* a mais de forma a criar um pequeno menu ao fim do programa que pergunta ao usuário se deseja usar o programa novamente. Essa é uma opção interessante para programas que normalmente fazem vários testes. Da mesma forma, a turma deve analisar conjuntamente cada linha de código do algoritmo em questão.

Código 4.3: Média ENEM - Pesos UFPI

```
% Calculo de medias do ENEM, baseado nos pesos da UFPI
clear all; clc;
display('Calculo de medias do ENEM (pesos da UFPI)')
linguagem=input('Insira sua nota de Linguagens: ');
while(linguagem<0)
    linguagem=input('Insira sua nota de Linguagens: ');
end
matematica=input('Insira sua nota de Matematica: ');
while(matematica<0)
    matematica=input('Insira sua nota de Matematica: ');
end
humanas=input('Insira sua nota de Ciencias Humanas: ');
while(humanas<0)
    humanas=input('Insira sua nota de Ciencias Humanas: ');
end
natureza=input('Insira sua nota de Ciencias da Natureza: ');
while(natureza<0)
    natureza=input('Insira sua nota de Ciencias da Natureza:
       ');
end
redacao=input('Insira sua nota da Prova de Redacao: ');
while(redacao<0)
    redacao=input('Insira sua nota da Prova de Redacao: ');
end
```

```
24 mediaaritmetica=(linguagem+matematica+humanas+natureza+
      redacao)/5;
25 saude=(linguagem*5+matematica*4+humanas*2+natureza*5+redacao
      *3)/19;
26 exatas=(linguagem*5+matematica*5+humanas*2+natureza*4+redacao
      *3)/19;
27 sociais=(linguagem*5+matematica*2+humanas*5+natureza*4+
      redacao*3)/19;
28 clc;
29 printf('A media aritmetica das suas notas eh igual a %.2f\n',
       mediaaritmetica)
30 printf('A media para cursos da area de SAUDE %.2f\n', saude)
31 printf('A media para cursos da area de EXATAS %.2f\n', exatas
      )
32 printf('A media para cursos da area de CIENCIAS SOCIAIS %.2f\
      n', sociais)
```

O código apresentado acima calcula a média aritmética e as médias para cada eixo de ensino para ingresso na Universidade Federal do Piauí (UFPI).

Este último exemplo de código serve para propor aos alunos como atividade um programa de computador que possa ser usado pela sua comunidade. O professor deve mostrar que aplicando os conhecimentos matemáticos atrelados à situações cotidianas ele pode ajudar pessoas e servir como informação para quem está ao seu redor.

Seguindo os mesmos passos passados, o professor deve analisar linha por linha junto à classe, realçando as restrições para notas do ENEM (nenhuma nota do ENEM pode ser menor que zero, ou seja, negativa). Ao final, o aluno deverá pesquisar junto à universidade de interesse quais os pesos (média ponderada) a mesma coloca para seus cursos de graduação, assim as pessoas poderão ter noção das suas notas do Sistema de Seleção Unificada (SISU) antes que o sistema em questão esteja disponível.

Apresentamos aqui propostas de atividades para esta etapa:

Atividade 1: Construa um algoritmo que receba três números (reais e positivos) e indique se aqueles valores podem ser medidas de lados de um triângulo (use a

desigualdade triangular).

Atividade 2: Construa um algoritmo para saber se um número é primo. O usuário deverá inserir um número qualquer (inteiro positivo) e o programa deve imprimir na tela se o número é primo ou não.

Atividade 3: Construa um algoritmo que calcule a soma dos termos de uma progressão aritmética (P.A).
1º maneira: O usuário deve inserir o primeiro termo da P.A, o índice do último termo que será somado e a razão da P.A, e o programa então deve dar o valor da soma.
2ª maneira: O usuário deve inserir o primeiro termo da P.A, o índice do último termo que será somado e o último termo da P.A, e o programa então deve dar o valor da soma.

4.2.5.4 Verificação da Aprendizagem

A verificação será feita a partir das atividades propostas, no momento em que cada grupo apresenta os códigos feitos, será analisada a funcionalidade dos mesmos e em caso de mau funcionamento, os alunos irão juntamente com o professor propor mudanças para corrigir os possíveis erros.

5 Considerações Finais

Esta pesquisa procurou apresentar uma estratégia para auxiliar na assimilação de conhecimento matemático e lógico como um todo, fazendo uso da elaboração de algoritmos e suas respectivas implementações no *GNU Octave*.

Haja vista que a utilização de novas tecnologias se dá em todas as áreas do conhecimento, é preciso preparar em todos os níveis de educação tanto educandos como educadores, para que possam fazer uso das máquinas e *softwares* como aliadas no processo de ensino-aprendizagem, fazendo com que o ensino das ciências exatas como um todo possa também ajudar no exercício da criatividade.

É crucial levar à sala de aula a ideia de que o importante nesse processo não é chegar num número correto ao final de um cálculo matemático, e sim entender de que maneira devemos enxergar as informações ao nosso redor, e como podemos transformar elas de modo a alcançar nossos objetivos. Devem ser debatidas todas as possibilidades e maneiras diferentes de resolver um questionamento, visto que a interpretação e o método são os pontos marcantes para a assimilação por parte dos discentes.

É de responsabilidade do educador desmitificar a matemática para todos, tornando o processo de aprendizagem o mais prazeroso possível. É preciso haver a imersão do professor no mundo dos alunos, para que se saiba a melhor maneira de abordar um conteúdo, algum modo de chamar a atenção e mostrar que aquele conhecimento é importante na formação social do mesmo. É de incumbência do professor levar a tecnologia para a sala de aula, mas também é seu papel mostrar que a máquina mais perfeita ainda é o cérebro humano quando atiçado ao conhecimento, pois é capaz de enxergar além.

Nesse trabalho, apresentamos a lógica e o pensamento computacional como ferramenta adicional ao currículo no ensino básico, com o intuito de aliar os conhecimentos teóricos no estudo de matemática à prática, aos meios tecnológicos que já são realidade nas nossas vidas, porém, subaproveitados em sala de aula, como o computador. Nesse intuito, foi utilizado o *software GNU Octave*, ferramenta essa que permite ao estudante conhecer métodos auxiliares na resolução de problemas, além da possibilidade de engajá-lo ao interesse pela área das ciências exatas.

Com isso, foi possível trazer temas bastante abordados no ensino fundamental e médio através de exercícios que estimulam a reflexão sobre como chegar num objetivo final, buscando construir uma sequência lógica de passos para tal fim. A partir disso, o aluno também é capaz de correlacionar os algoritmos matemáticos com as atividades que realiza cotidianamente em sua vida.

É com a intenção de fugir da tradicionalidade no ensino das ciências exatas que fazemos essa proposta didática, objetivando explorar grande parte das possibilidades e potencialidades dentro do currículo escolar e trazendo mais dinâmica no que diz respeito à utilização de recursos tecnológicos por parte dos docentes.

Referências Bibliográficas

BASSANEZI, R. C. Temas e modelos. *São Paulo: Editora Unicamp*, 2012. Citado na página 10.

BIEMBENGUT, M. S.; HEIN, N. Modelagem matemática no ensino. *São Paulo*, 2003. Citado na página 9.

BLIKSTEIN, P. O pensamento computacional e a reinvenção do computador na educação. *Education & Courses*, 2008. Citado na página 10.

BOYER, C. B.; MERZBACH, U. C. *A history of mathematics*. [S.l.]: John Wiley & Sons, 2011. Citado na página 15.

BRANDÃO, P. Alan turing: da necessidade do cálculo, a máquina de turing até à computação. *Revista de Ciências da Computação*, Universidade Aberta, p. 73–88, 2017. Citado na página 27.

CORMEN, T. H. et al. Algoritmos: teoria e prática. *Editora Campus*, v. 2, p. 296, 2002. Citado na página 12.

EVARISTO, J.; CRESPO, S. *Aprendendo a Programar Programando numa Linguagem Algorítmica Executável (ILA)*. [S.l.]: Book Express, Rio de Janeiro, 2000. Citado na página 30.

FERRARI, F.; CECHINEL, C. Introdução à algoritmos e programação. *Bagé: Universidade Federal do Pampa*, 2008. Citado 2 vezes nas páginas 16 e 33.

FONSECA, C. *História da computação: O Caminho do Pensamento e da Tecnologia*. [S.l.]: EDIPUCRS, 2007. Citado na página 28.

GARBI, G. G. *A rainha das ciências*. [S.l.]: Editora Livraria da Física, 2006. Citado na página 13.

GUEDES, L. A. Algoritmo e lógica de programação. *Natal: Universidade Federal do Rio Grande do Norte*, 2004. Citado na página 20.

KAISER, G. Mathematical modelling in school–examples and experiences. *Mathematikunterricht im Spannungsfeld von Evolution und Evaluation. Festband für Werner Blum. Hildesheim: Franzbecker*, p. 99–108, 2005. Citado na página 9.

MOL, R. S. *Introdução à História da Matemática, Editora CAED-UFMG, Belo Horizonte, 2013. 138 p.* 2017. Citado na página 9.

RIBEIRO, M. R. da C. *Grafos, Algoritmos e Programação*. 152 p. Dissertação (Mestrado Profissional em Matemática) — Universidade Federal Rural de Pernambuco, Recife, 2018. Citado na página 35.

SOUSA, B.; DIAS, J. J. L.; FORMIGA, A. d. A. *Introdução a programação*. [S.l.]: UFPB, 2014. Citado na página 29.

TONET, B.; KOLIVER, C. Introdução aos algoritmos. *Apostila da Universidade de Caixias do Sul usada como manual do VisuAlg*, 2013. Citado na página 25.

A Códigos

A.1 Comprimento e área de uma circunferência

```
% Algoritmo para calculo do comprimento e area de uma
   circunferencia
clear all; clc;
disp('Calculo do comprimento e area de um circunferencia de
   raio "R"')
r=input('Insira o valor de "R": ');
while(r<0)
    r=input('Insira um valor positivo para "R": ');
end
Comprimento=2*pi*r
Area=pi*r^2
```

A.2 Distância entre dois pontos

```
1 % Algoritmo para calculo de distancia entre dois pontos
2 clear all; clc;
3 disp('Calculo da distancia entre dois pontos no plano [x,y]')
4 coordenada1=input('Insira a primeira coordenada. Ex.:[2,5]: '
    );
5 coordenada2=input('Insira a segunda coordenada. Ex.:[2,5]: ')
    ;
6 distancia=sqrt((coordenada1(1)-coordenada2(1))^2 + (
    coordenada1(2)-coordenada2(2))^2)
```

A.3 Distância entre ponto e reta

```
% Algoritmo para calculo de distancia entre ponto e reta
clear all; clc;
disp('Calculo da distancia entre ponto "[Xo,Yo]" e reta "ax + by + c =0"')
ponto=input('Insira a a coordenada do ponto Ex.:[Xo,Yo]: ');
reta=input('Insira os coeficientes "a", "b" e "c" da reta. Ex.: [a, b, c]: ');
distancia=(abs(reta(1)*ponto(1) + reta(2)*ponto(2) + reta(3)))/sqrt(reta(1)^2 + reta(2)^2)
```

A.4 Cálculo do fatorial

```
% Algoritmo para calculo do fatorial de um numero "n"
clear all; clc;
disp('Calculo do fatorial de um numero inteiro "n", ou seja, "n!"')
N=input('Insira o valor de "n": ');
while(N<0||N~=floor(N))
    N=input('Insira o valor de "n" (deve ser um numero inteiro positivo):');
end
fatorial=1;
for i=1:N
    fatorial=fatorial*i;
end
clc
disp('O fatorial de "n" tem valor igual a: ')
fatorial
```

A.5 Lei dos Cossenos

```
% Algoritmo para calculo da Lei dos Cossenos
clear all; clc;
disp('Calculo da Lei dos Cossenos')
b=input('Insira a medida do primeiro lado do triangulo: ');
while(b<=0)
    b=input('Insira a medida do primeiro lado do triangulo (
       POSITIVO): ');
end
c=input('Insira a medida do segundo lado do triangulo: ');
while(c<=0)
    c=input('Insira a medida do segundo lado do triangulo (
       POSITIVO): ');
end
angulo=input('Insira a medida do angulo entre os 2 lados
   indicados anteriormente, em graus: ');
while(angulo<=0||angulo>=180)
    angulo=input('Insira a medida do angulo entre os 2 lados
       indicados anteriormente (POSITIVO E MENOR QUE 180
       graus): ');
end
a=sqrt(b^2+c^2-2*b*c*cosd(angulo))
```

A.6 Média Aritmética

```
% Algoritmo para calculo de media aritmetica
clear all; clc;
disp('Calculo de media aritmetica')
n=input('Escolha quantos numeros serao inseridos: ');
while(n<1||n~=floor(n))
    n=input('Insira um numero inteiro positivo: ');
end
for i=1:n
    notas(i)=input('Insira a nota: ');
    while(notas(i)<0)
    notas(i)=input('A nota deve ser um numero maior ou igual
       que zero: ');
    end
end
media=sum(notas)/n
```

A.7 Quantidade de números primos em um intervalo

```
% Algoritmo para deteccaoo se o nemero eh primo ou nao
clear all; clc;
display('Numeros Primos')
n=input('Insira ate que numero devemos verificar os primos: ');
while(n<1||n~=floor(n))
    n=input('Insira ate que numero devemos verificar os primos: ');
end
primo=[];
for i=2:n
  contador=0;
    for j=1:i
        if(mod(i,j)==0)
        contador=contador+1;
        end
      if(contador>2)
      break;
      end
    end
     if(contador==2)
        primo=[primo i];
        end
end
primo
```

A.8 Sequência de Fibonacci

```
1  % Algoritmo para exibicao da Sequencia de Fibonacci, ate um
       termo n
2  clear all; clc;
3  disp('Sequencia de Fibonacci')
4  n=input('Insira ate qual termo (posicao) voce deseja exibir
       da sequencia de Fibonacci: ');
5  while(n<1||n~=floor(n))
6      n=input('Insira ate qual termo (posicao) voce deseja
           exibir da sequencia de Fibonacci: ');
7  end
8  fibonacci(1)=1; fibonacci(2)=1;
9  if(n<2)
10     fibonacci(1)
11 else
12 for i=3:n
13     fibonacci(i)=fibonacci(i-1)+fibonacci(i-2);
14 end
15 fibonacci
16 end
```

A.9 Termo geral da Progressão Geométrica (P.G)

```
% Algoritmo para calculo do termo geral de uma P.G
clear all; clc;
disp('Calculo do termo geral "An" de uma P.G com termo
   inicial "A1" e razao "q"')
a_1=input('Insira o valor de "A1": ');
q=input('Insira o valor de "q": ');
while(q==0)
    q=input('Insira um valor diferente de 0 para "q": ');
end
n=input('Digite qual a posicao do termo que se deseja saber o
    valor: ');
while(n<=0||n~=floor(n))
    n=input('Digite um valor positivo e inteiro para "n": ');
end
a_n = a_1*q^(n-1)
```

A.10 Soma dos termos da Progressão Geométrica (P.G)

```
% Algoritmo para calculo da soma de uma P.G
clear all; clc;
flag=0;
disp('Calculo da soma de uma P.G de "A1" ate "An"')
a_1=input('Insira o valor de "A1": ');
q=input('Insira o valor da razao "q": ');
if(q>=1||q<=0)
    opc=1;
end
if(q<1&&q>0)
    opc=2;
end
switch opc
    case 1
        n=input('Digite qual a posicao do termo que se deseja
            saber o valor: ');
        while(n<=0||n~=floor(n))
        n=input('Digite um valor positivo e inteiro para "n":
            ');
        end
        Soma=(a_1)*(q^n-1)/(q-1)
    case 2
        Soma=(a_1)/(1-q)
end
```