UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE MATEMÁTICA

KALLEL VINICIUS ROCHA FIORI

MANIM COMO FERRAMENTA DIDÁTICA PARA POTENCIALIZAR O ENSINO DE CÁLCULO

SÃO CARLOS – SP
2024

KALLEL VINICIUS ROCHA FIORI

MANIM COMO FERRAMENTA DIDÁTICA PARA POTENCIALIZAR O ENSINO DE CÁLCULO

Monografia do Trabalho de Conclusão de Curso apresentada ao Departamento de Matemática da Universidade Federal de São Carlos para obtenção do título de Licenciatura em Matemática.

Orientador: Prof. Dr. Jean Piton Gonçalves

SÃO CARLOS – SP
2024

AGRADECIMENTOS

A realização deste trabalho de conclusão de curso não teria sido possível sem o apoio e a contribuição de muitas pessoas.

Primeiramente, agradeço aos meus pais, por seu apoio incondicional, incentivo constante e suporte em todas as etapas da minha vida acadêmica.

Agradeço imensamente ao meu orientador, Jean Piton, por sua orientação paciente, valiosas sugestões e inestimável apoio ao longo de todo o processo. Sua dedicação e conhecimento foram fundamentais para a concretização deste trabalho.

Meus sinceros agradecimentos também vão para meus amigos Inácio Neves, Pedro Blaya e Paulo César, pelo apoio, companheirismo e por sempre acreditarem em mim. Vocês tornaram essa jornada mais leve e significativa com suas palavras de encorajamento e amizade sincera.

Finalmente, agradeço a todos os meus amigos, que de alguma forma contribuíram para a realização deste trabalho. Cada um de vocês, com seu apoio e incentivo, teve um papel essencial na minha trajetória acadêmica.

A todos, o meu mais profundo agradecimento.

Progress is noticeable when the question that tortured us has lost its meaning.

RESUMO

A Visualização na Matemática é uma área de pesquisa que estuda os elementos visuais em ferramentas computacionais, tais como gráficos, diagramas e representações geométricas para comunicar determinados conteúdos matemáticos que podem auxiliar no Ensino. No contexto computacional, o Manim é uma biblioteca Python projetada para a criação de animações e objetos matemáticos de alta qualidade gráfica. Com o objetivo de explorar o uso do Manim considerando aspetos da Visualização Matemática em contextos de Ensino, este Trabalho de Conclusão de Curso (i) revisou a literatura sobre Visualização em Computação sob a ótica de Diehl (2005), (ii) comparou aspectos da Visualização Matemática entre Manim e Geogebra e (iii) desenvolveu material didático que pode auxiliar o professor no Ensino do Cálculo Diferencial. Resultados mostram que o Manim destacou-se a como uma ferramenta eficaz no contexto educacional, com alto nível de granularidade e capacidade de reuso, sendo adaptável para diferentes perspectivas e necessidades educacionais.

Palavras-chave: Python. Manim. Ensino de Cálculo. Visualização Matemática.

ABSTRACT

Visualization in Mathematics is an area of research that studies visual elements in computational tools, such as graphs, diagrams and geometric representations to communicate certain mathematical content that can assist in Teaching. In the computational context, Manim is a Python library designed for creating high-quality graphical animations and mathematical objects. With the aim of exploring the use of Manim considering aspects of Mathematical Visualization in Teaching contexts, this Course Completion Work (i) reviewed the literature on Computer Visualization from the perspective of Diehl (2005), (ii) compared aspects of Mathematical Visualization between Manim and Geogebra and (iii) developed teaching material that can assist the teacher in Teaching Differential Calculus. Results show that Manim stood out as an effective tool in the educational context, with a high level of granularity and reuse capacity, being adaptable to different perspectives and educational needs.

Keywords: Python. Manim. Teaching Calculus. Mathematical Visualization.

LISTA DE FIGURAS

LISTA DE SIGLAS

UFSCar	Universidade Federal de São Carlos
OA	Objeto de Aprendizagem
OAM	Objeto de Aprendizagem Manim
VM	Visualização Matemática
IDLE	Integrated Development and Learning Environment

SUMÁRIO

1 INTRODUÇÃO

A visualização matemática é uma ferramenta poderosa no ensino de matemática, permitindo a tradução de conceitos abstratos em representações visuais compreensíveis. No contexto da Educação Superior, a visualização pode enriquecer o ensino de cálculo, tornando conceitos complexos mais acessíveis aos alunos.

Este trabalho de conclusão de curso centra-se na criação de materiais didáticos inovadores para o ensino de cálculo, utilizando o *Manim* (*Mathematical Animation Engine*), uma biblioteca *Python* para a criação de visualizações matemáticas. A motivação principal é investigar como essa ferramenta pode ser utilizada para desenvolver Objetos de Aprendizagem (OA) que facilitem a compreensão de tópicos como derivadas, reta tangente e problemas de otimização.

Inicialmente, abordamos a importância da visualização no ensino de cálculo e analisamos o *GeoGebra*, destacando seus paralelos com o *Manim*. Em seguida, propomos a sigla OAM (Objeto de Aprendizagem Manim) para descrever esses materiais com elevado grau de granularidade, e apresentamos uma série de OAMs desenvolvidos com o *Manim*, demonstrando seu potencial para criar materiais educacionais dinâmicos e eficazes.

Os Objetos de Aprendizagem Manim (OAMs) desenvolvidos neste trabalho seguem uma sequência lógica de conteúdos, começando com a introdução às derivadas e avançando até a visualização de um problema de otimização relacionado ao volume de uma caixa. Cada OAM foi cuidadosamente projetado para facilitar a compreensão dos conceitos, permitindo uma progressão natural no aprendizado. Durante a construção desses materiais, também foram apresentadas sugestões de uso e de modificações que possibilitam a reutilização dos OAMs em diferentes contextos educacionais, garantindo sua flexibilidade e aplicabilidade em diversas situações de ensino.

Essa análise visa não apenas explorar a aplicabilidade do *Manim* como ferramenta educacional, mas também contribuir diretamente para o desenvolvimento de novos recursos didáticos que possam impactar positivamente o ensino e a aprendizagem de cálculo na Educação Superior.

2 VISUALIZAÇÃO DE SOFTWARE E A VISUALIZAÇÃO MATEMÁTICA

A visualização é uma linguagem universal que é capaz de transcender até mesmo barreiras linguísticas. É uma forma natural de transmitir informações, conceitos e compartilhar conhecimento.

> Visualização é o processo de transformar informações em uma forma visual, permitindo que os usuários observem as informações. A exibição visual resultante permite que o cientista ou engenheiro perceba visualmente características que estão ocultas nos dados, mas que são necessárias para exploração e análise de dados (DIEHL, 2005). [1]

A visualização pode desempenhar um papel importante no contexto educacional, uma vez que, o uso de imagens pode ajudar no processo de ensino de uma forma geral, assim, a visualização torna-se uma ferramenta importante para professores e alunos, apoiando os processos de ensino e aprendizagem.

Neste capítulo, abordaremos a visualização em computação (Seção 2.1) e matemática (Seção 2.2), explorando suas implicações e algumas possíveis aplicações.

2.1 VISUALIZAÇÃO EM COMPUTAÇÃO

A visualização de *software* é uma área de estudo fundamental em Ciências da Computação, desempenhando um papel essencial na compreensão e na análise de *softwares*. A visualização, nesse contexto, proporciona uma perspectiva visual que pode permitir uma análise mais detalhada de certas características e relações que podem ser difíceis de identificar apenas através da análise textual de código-fonte.

Os elementos visuais, como gráficos, imagens, diagramas e representações visuais de dados, desempenham um papel crucial na simplificação de conceitos abstratos e na facilitação do entendimento. A capacidade de traduzir informações em formas visuais permite que as pessoas percebam relações, padrões e detalhes que muitas vezes não são perceptíveis apenas através de texto ou explicações verbais. Através da combinação de cores, formas e estruturas, os elementos visuais oferecem uma representação mais acessível e eficaz de conceitos, tornando o aprendizado e a comunicação mais envolvente e clara.

A visualização de *software* é uma área abrangente que combina princípios de computação gráfica, engenharia de *software* e design de interfaces. Stephan Diehl, em seu trabalho, explora algumas técnicas e aspectos da visualização de *software*, e como elas podem ser aplicadas de forma eficaz no contexto da computação.

[1] Tradução nossa: *Visualization is the process of transforming information into a visual form, enabling users to observe the information. The resulting visual display enables the scientist or engineer to perceive visually features which are hidden in the data but nevertheless are needed for data exploration and analysis*(DIEHL, 2005).

> O objetivo da visualização é transmitir informações através do sistema visual humano para o cérebro humano, desenhando imagens na tela do computador (DIEHL, 2005). [2]

Neste contexto, Diehl, um pesquisador na área de visualização de *software*, tem contribuído significativamente para o desenvolvimento de técnicas que auxiliam na representação visual de sistemas de software. Seu trabalho traz uma análise sobre a visualização de *software*, discutindo sobre o uso de certas ferramentas e apresenta alguns conceitos mais gerais da visualização de software.

> Percepção é o processamento de informações sensoriais e, portanto, parte da cognição humana, que também inclui consciência, raciocínio e aprendizagem. 75% de todas as informações do mundo real são percebidas visualmente; apenas 13% são percebidos através do sentido auditivo e os 12% restantes através de outros sentidos (DIEHL, 2005). [3]

Uma aplicação possível é a visualização do algoritmo de Dijkstra[4] criada pela universidade de São Francisco[5]. Essa visualização oferece uma representação interativa e envolvente desse importante algoritmo na área de computação.

---

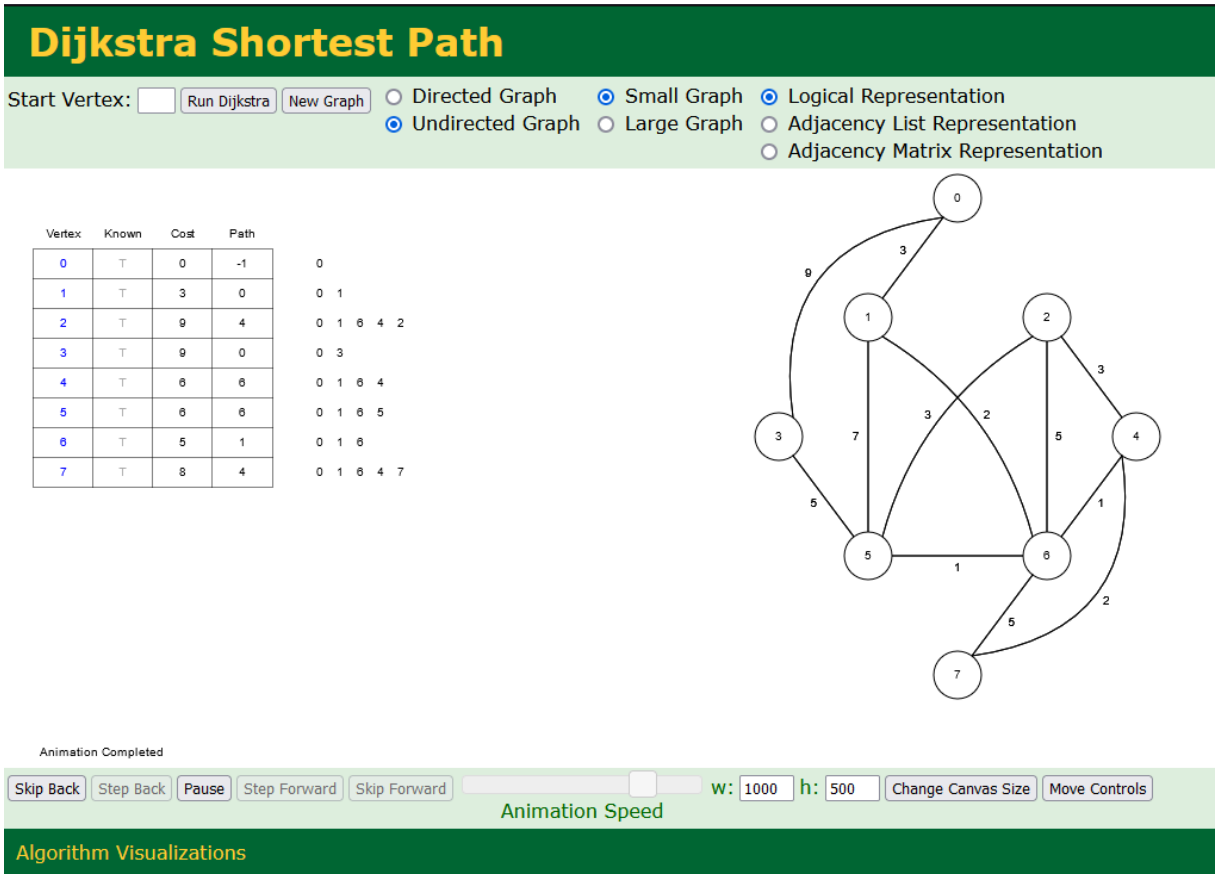
[2] Tradução nossa: *The goal of visualization is to convey information through the human visual system into the human brain by drawing images on the computer screen* (DIEHL, 2005).

[3] Tradução nossa: *Perception is the processing of sensory information and thus part of human cognition, which also includes awareness, reasoning, and learning. 75% of all information from the real world is visually perceived; only 13% is perceived through the auditory sense and the remaining 12% through other sense* (DIEHL, 2005).

[4] O algorítimo de Dijkstra trata-se de um algorítimo que visa encontrar o caminho de menor custo entre dois vértices em um grafo ponderado.

[5] Disponível em: https://www.cs.usfca.edu/ galles/visualization/Dijkstra.html Acesso em: 19/10/2023

Figura 1 – Visualização Criada pela universidade de São Francisco que mostra o algoritmo de Dijkstra



Fonte: Universidade de São Francisco (2023)[6]

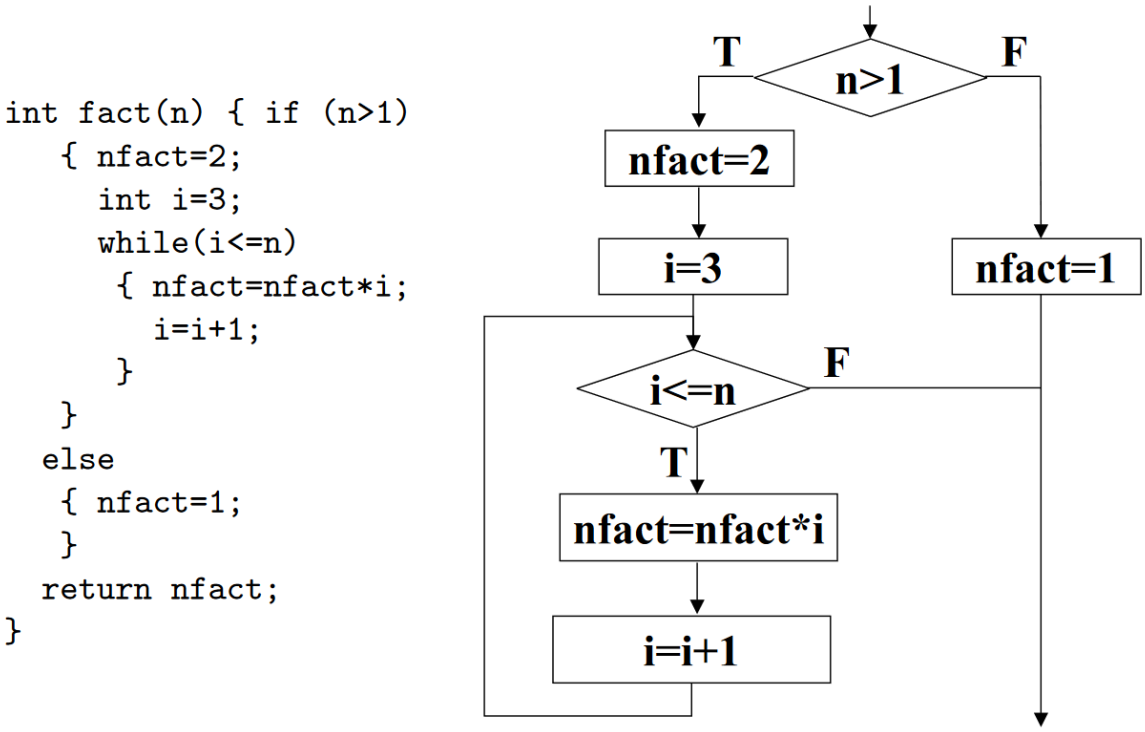
Ao explorar essa visualização, os usuários podem observar o funcionamento do algoritmo de Dijkstra em ação, acompanhando como ele encontra o caminho mais curto entre um ponto de origem e todos os outros pontos em um grafo ponderado gerado aleatoriamente.

A visualização permite que os usuários observem o processo passo a passo, destacando as etapas de relaxamento das arestas e a seleção dos vértices apropriados. As tabelas e animações tornam o entendimento do algoritmo de Dijkstra mais acessível, ao mesmo tempo em que ilustram a eficácia desse algoritmo na resolução de problemas de caminho mais curto. Essa visualização é um exemplo claro de como a representação visual pode simplificar conceitos complexos e torná-los mais acessíveis, auxiliando estudantes, profissionais e entusiastas da computação na compreensão desse algoritmo importante.

Outra aplicação de visualização que podemos encontrar são os fluxogramas, esses instrumentos permitem que os programadores possam ver de uma forma mais clara o fluxo de seus programas.

[6] Disponível em: https://www.cs.usfca.edu/ galles/visualization/Dijkstra.html Acesso em: 19/10/2023

Figura 2 – Exemplo de fluxograma para uma função que retorna o fatorial de um numero n



Fonte: DIEHL (2005, p. 40).

Na Figura 2, representa um fluxograma para uma função que retorna o fatorial de um numero *n*. Através desse fluxograma é possível observar as condicionais e operações que são feitas de acordo com o valor inicial de *n*. Dessa forma, se torna muito mais fácil, por exemplo, identificar *bugs* ou problemas do código.

Com isso, podemos ver que a visualização desempenha um papel importante em Ciências da Computação, sendo uma ferramenta de suporte para o entendimento dos assuntos e problemas gerais que podemos encontrar. De uma forma semelhante ao que vimos até agora, podemos traçar um paralelo para as visualizações dentro da matemática, afim de proporcionar novas perspectivas de problemas e exemplos que podem ser de difícil entendimento se enunciados apenas de maneira escrita e/ou verbal. Dessa forma, podemos usar a visualização como uma ferramenta educacional.

2.2 VISUALIZAÇÃO EM MATEMÁTICA

A visualização na matemática utiliza elementos visuais, como gráficos, diagramas e representações geométricas para comunicar diversos conceitos matemáticos.

Imaginar, tocar, manipular são fatores que influenciam no desenvolvimento cognitivo

> dos estudantes, dando estrutura para o entendimento de determinados conceitos. E quando o manipular não está ao alcance, a visualização pode conduzir a uma tentativa de dar concretude ao pensamento, construindo uma imagem mental, um significado ao significante (SANTOS, 2014).

Abordou-se que na Seção 2.1 a visualização desempenha um papel vital em Ciências da Computação, fornecendo uma abordagem mais concreta e acessível para a compreensão de conceitos abstratos. De forma semelhante, a Visualização Matemática (VM) se revela uma aliada no estudo e na aplicação da matemática, transformando equações, teoremas e cálculos complexos em representações visuais que podem auxiliar o seu entendimento.

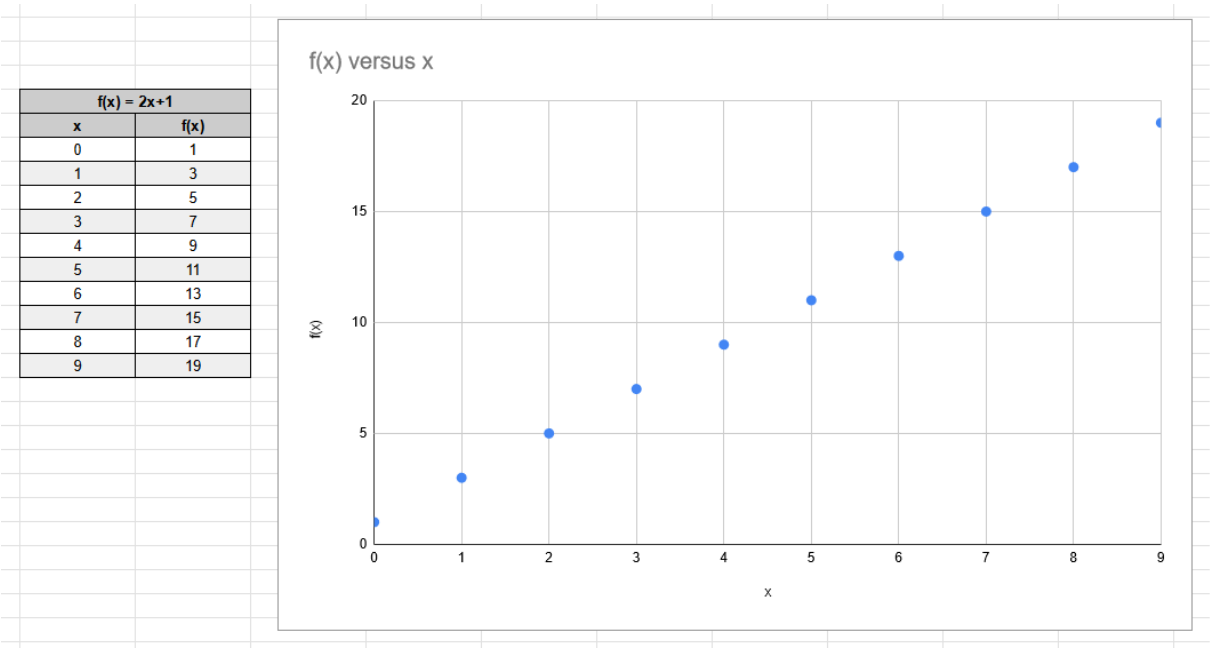
Uma abordagem possível dentro da VM é o uso de tabelas e planilhas, que permitem que estudantes, professores e pesquisadores organizem dados, modele relações matemáticas e explorem conceitos numéricos com clareza e precisão. As tabelas e planilhas são ferramentas amplamente utilizadas para visualização e análise de dados em diversas disciplinas, incluindo a matemática. Elas oferecem uma maneira eficaz de organizar informações numéricas, representar equações, demonstrar relações e realizar cálculos extensos.

Tabelas de cálculos, como aquelas criadas no *Microsoft Excel*, *Google Sheets* ou *LibreOffice Calc*, são frequentemente empregadas para representar conjuntos de dados, gerar gráficos a partir de pontos, realizar operações de cálculo e modelar funções matemáticas. Elas são especialmente úteis para visualizar relações entre variáveis, criar sequências numéricas e realizar análises estatísticas. O uso desse tipo de tecnologia como visualização de problemas traz diversas vantagens:

> Enquanto vantagens, a pesquisadora destaca que o *Microsoft Excel* pode favorecer o trabalho cooperativo e colaborativo, evidenciado na ajuda mútua durante a resolução de problemas; desenvolver a autonomia na realização das tarefas, permitindo a exploração de forma abrangente de comandos e funções, o que proporciona novas descobertas; contribuir para a persistência dos estudantes diante de eventuais dificuldades; motivar os estudantes para a aprendizagem; e desenvolver a capacidade de aprender coisas novas de forma autônoma. (ESTEVAM; KALINKE, 2013).

Um exemplo possível para o uso de tabelas na matemática é a criação de planilhas de valores para funções matemáticas, permitindo que os usuários examinem o comportamento dessas funções em diferentes pontos. A partir das tabelas, é possível criar gráficos que ilustram visualmente o comportamento da função em questão, tornando conceitos matemáticos mais tangíveis e compreensíveis.

Figura 3 – Exemplo de pontos de uma função com um gráfico dos pontos



f(x) = 2x+1	
x	f(x)
0	1
1	3
2	5
3	7
4	9
5	11
6	13
7	15
8	17
9	19

Fonte: Elaborada pelo autor (2023)[7].

O exemplo da Figura 3 traz um exemplo do uso de tabelas para criar visualizações, nesse caso, criando uma tabela com os valores de x e os resultados de $f(x)$ sendo essa função $f(x) = 2x + 1$. Esse pode parecer um exemplo simples, mas essa visualização cumpre bem o proposito de mostrar como os valores de x estão relacionados com o valor resultante de uma $f(x)$.

Existem ainda uma série de outras ferramentas que auxiliam na criação de VM, essas ferramentas podem ser das mais diversas, podemos encontrar mais exemplos na Seção 2.2.1. Podemos criar uma distinção entre dois tipos fundamentais de conteúdos matemáticos: objetos concretos e objetos abstratos. Essa diferenciação pode desempenhar um papel crucial na escolha das ferramentas e abordagens que empregamos para representar conceitos matemáticos de forma clara e compreensível.

Objetos concretos são objetos matemáticos, têm uma representação física tangível ou podem ser visualizados em contextos reais. Exemplos incluem figuras geométricas, gráficos de funções, sólidos tridimensionais e representações de problemas de matemática aplicada. A visualização de objetos concretos muitas vezes envolve a criação de representações visuais diretas que se assemelham a modelos físicos ou situações da realidade. Ferramentas como o GeoGebra (explorado no Capítulo 2.3) são especialmente adequadas para representar objetos concretos, permitindo a exploração interativa de formas e relações matemáticas.

[7] Imagem criada usando *Google Sheets*

Por outro lado, objetos matemáticos abstratos não têm uma correspondência direta com entidades físicas. Exemplos incluem equações, operadores, teoremas e conceitos matemáticos puramente simbólicos. A visualização de objetos abstratos requer uma abordagem diferente, muitas vezes recorrendo a representações gráficas, esquemas, diagramas ou animações para tornar esses conceitos mais concretos e acessíveis. Nesse contexto, ferramentas como o Manim (explorado no Capítulo 3) e outras bibliotecas similares podem desempenhar um papel fundamental na tradução de ideias matemáticas abstratas em representações visuais dinâmicas.

2.2.1 Ferramentas para o desenvolvimento de VM

O *Python*, uma linguagem de programação versátil e acessível, desempenha um papel significativo na educação, e sua aplicação vai muito além da programação pura. Essa linguagem oferece uma abordagem prática para a resolução de problemas matemáticos, a modelagem de equações e a análise de dados numéricos. Sua sintaxe clara e sua comunidade ativa de desenvolvedores criaram um ambiente acolhedor para estudantes e educadores que desejam explorar a matemática de maneira interativa.

> Rossum (1991), criador da linguagem de programação *Python*, a desenvolveu visando ser uma linguagem voltada para os leigos em computação. Por possuir uma sintaxe intuitiva, os alunos passam a desenvolver sua capacidade lógica e consequentemente podem vir a melhorar o seu aprendizado em matemática. (PESENTE, 2019).

No recente trabalho de Guilherme Moraes Pesente(2019), o *Python* foi explorado como um recurso para o desenvolvimento de exemplos práticos que demonstram a aplicação da linguagem na educação matemática. O trabalho demonstra como o *Python* pode ser utilizado para criar programas que sintetizam certos conceitos matemáticos.

Ao usar linguagens como o *Python*, de forma geral, os problemas precisam ser analisados de uma forma muito mais minuciosa, assim os alunos podem ver o problema de uma forma muito mais detalhada e geral, tornando o aprendizado mais envolvente e compreensível.

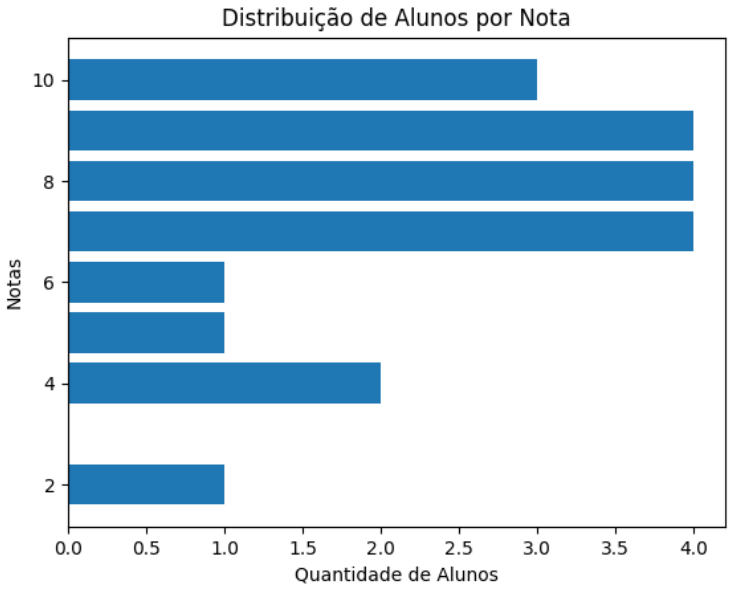
> Os códigos computacionais permitiram que os alunos respondessem aos exercícios com maior número de detalhes, separando valor por valor em variáveis, para que o computador processasse todas as informações e desse o resultado correto. Isto os ajudou na elaboração e resolução das atividades em sala de aula, pois, agora, eles não só respondiam o proposto, mas viam como poderia separar cada informação, desenvolvendo um passo a passo de respostas (PESENTE, 2019).

O referido trabalho, faz o uso do *Python* como uma ferramenta geral dentro da educação, usando a programação em si como ferramenta. Porém, dentro da visualização matemática, o *Python* também mostra-se útil, uma vez que certas bibliotecas são usadas para gerar imagens. Uma das bibliotecas mais utilizadas para essa finalidade é o *matplotlib*[8]. Essa biblioteca oferece recursos robustos para criar gráficos, tabelas e visualizações que auxiliam na compreensão de conceitos matemáticos. Vamos explorar o seu uso com um exemplo simples.

[8] Disponível em: https://matplotlib.org/ Acesso em 01/11/2023

Considere um problema no qual você deseja representar visualmente a distribuição de notas de uma turma. Com o *Python* e a biblioteca *matplotlib*, podemos criar um gráfico de barras que ilustra essa distribuição de maneira clara e eficaz. Note que para esse problema estamos trabalhando com um objeto matemático concreto, uma vez que essa lista de distribuição de notas representa algo concreto, como mostra a Figura 4.

Figura 4 – Imagem criada com matplotlib



Fonte: Elaborada pelo autor[9].

Nesse exemplo, utilizamos o *Python* e o matplotlib para criar um gráfico de barras que representa as notas dos alunos. Essa é apenas uma amostra do que é possível fazer com o *Python* na visualização matemática. Agora, é importante mencionar que, além do matplotlib, existem outras bibliotecas disponíveis, cada uma com suas próprias vantagens e aplicações específicas. Uma dessas bibliotecas é o Manim, uma biblioteca que já possui uma gama de funções prontas para se gerar VM, que exploraremos no Capítulo 3.

Um outro exemplo são os fractais que são figuras geométricas com estruturas autossimilares em infinitas escalas. Por mais que, dentro da categorização proposta, esses objetos possam ser considerados objetos concretos é relativamente complicado criar visualizações desses objetos.

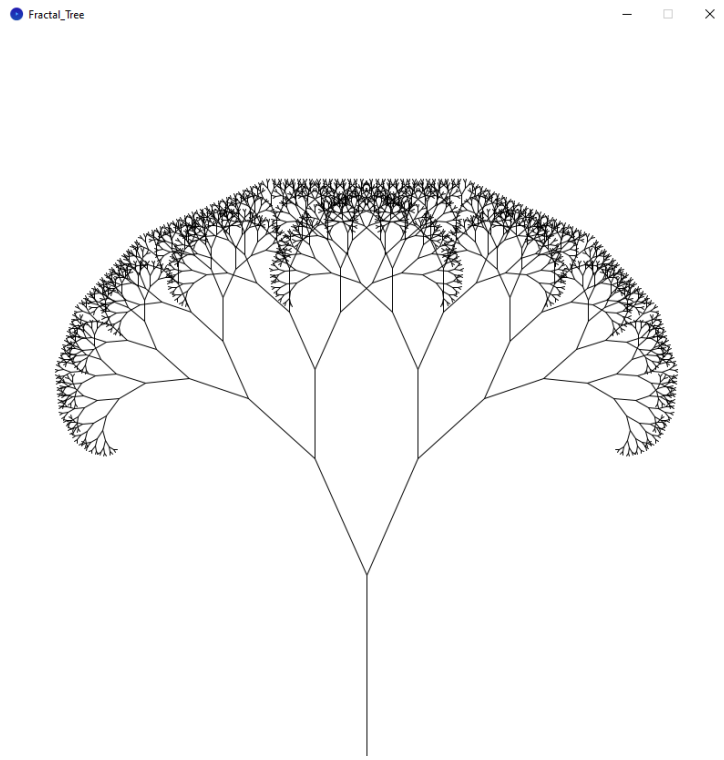
Para gerar uma visualização desses objetos pode ser mais fácil através de um ambiente programático, nesse caso, estarei usando *Processing*[10] uma plataforma baseada em *JAVA* usada para criar visualizações diversas. Nesse caso estamos interessados em criar uma visualização

[9] Imagem criada com *matplotlib* versão 3.8.0 e *Python* versão 3.12.0

[10] Disponivel em: https://processing.org/ Acesso em: 30/10/2023

para um fractal em formato de árvore, como podemos ver na Figura 5.

Figura 5 – Imagem criada com Processing mostrando um fractal em formato de arvore



Fonte: Código elaborado pelo canal The Coding Train, disponível no *Youtube*[11].

Note que não podemos representar um fractal que é de fato infinito em escala, mas visualizações como essa são uteis para visualizarmos o comportamento fractal, assim como uma possível representação geométrica. Além disso, ao usarmos ferramentas como o *Processing* podemos colocar alguns elementos de interatividade dentro de nossas visualizações. Na aplicação discutida acima, foi implementada uma interação com o mouse, onde podemos mudar o ângulo de abertura entre o galhos da árvore de fractal de forma dinâmica.

Outra ferramenta conhecida que pode auxiliar na criação de visualizações em matemática é o GeoGebra, um software matemático dinâmico que combina álgebra e geometrias. O GeoGebra fornece um ambiente interativo onde conceitos matemáticos podem ser explorados e visualizados em tempo real, permitindo que estudantes e professores compreendam de forma mais eficaz as relações matemáticas e os resultados de cálculos. Dessa forma, na categorização proposta, GeoGebra é uma ferramenta útil para a criação de visualizações de objetos concretos. No Capítulo 2.3 detalharemos o GeoGebra, com foco na visualização matemática.

[11] Imagem criada com Processing versão 4.3. Código Disponível em: https://github.com/CodingTrain/website-archive/blob/main/CodingChallenges/CC_014_FractalTree/Processing/CC_014_FractalTree/CC_014_FractalTree.pde Acesso em: 30/10/2023
Video Disponível em: https://www.youtube.com/watch?v=0jjeOYMjmDU Acesso em: 30/10/2023.

2.3 GEOGEBRA PARA O DESENVOLVIMENTO DE VM

O GeoGebra é um software capaz de gerar visualizações interativas e de simulação. Essa aplicação combina elementos de geometria, álgebra e gráficos, oferecendo uma plataforma integrada para explorar conceitos matemáticos e criar visualizações interativas para tais.

> O software GeoGebra, é programa configurado a partir de propriedades matemáticas, constituído coma finalidade da universalização do conhecimento no ambiente escolar.É um aplicativo dinâmico que fez a junção de conceitos de geometria e de álgebra em uma interface gráfica, que promove a construção de vários conceitos no campo matemático. Portanto, comprometidos com esta modalidade, ensino de matemática e tecnologia, temos como ponto forte a viabilidade de visualização, neste caso a visualização gráfica das funções e da geometria plana, uma forma de representação que contribui fortemente para a compreensão e incorporação dos conceitos matemáticos (SILVEIRA MOURA; SILVA DOS SANTOS; SILVA, 2016).

O GeoGebra é uma ferramenta acessível para educadores, aprendizes e entusiastas em matemática que pretendem explorar diversos tópicos matemáticos, desde a geometria a até álgebra. Sua abordagem dinâmica permite manipular variáveis e outros objetos matemáticos[12]dentro do ambiente e observar como as mudanças afetam os resultados apresentado dentro da interface do programa, tornando o aprendizado mais envolvente e intuitivo.

O GeoGebra tem experimentado um notável crescimento em sua popularidade no contexto educacional. À medida que a necessidade de ferramentas versáteis para a visualização matemática cresce, o GeoGebra tem se destacado como uma solução eficaz para a criação de representações dinâmicas e interativas. Sua crescente aceitação e comunidade de usuários ativos demonstram a importância e o impacto dessa ferramenta no cenário educacional, contribuindo para tornar a matemática mais envolvente e acessível.

> Além do aplicativo independente, o GeoGebra também permite a criação de páginas *web* interativas com miniaplicativos incorporados. Estes ambientes de aprendizagem e demonstração direcionados são partilhados gratuitamente por educadores matemáticos em plataformas online colaborativas como o GeoGebraWiki (www.geogebra.org/wiki). O número de visitantes do site GeoGebra aumentou de cerca de 50.000 em 2004 para mais de 5 milhões em 2010 (ver Figura 1), provenientes de mais de 180 países[13] (HOHENWATER; LAVICZA, 2011)

O GeoGebra apresenta quatro abordagens principais:

– **Gráficos no plano cartesiano**: esta versão é um recurso essencial para representações gráficas de funções e equações. Ela permite que os usuários criem gráficos, explorem propriedades de funções e analisem relações matemáticas de forma visual.

[12] Convencionou-se nesse capitulo que a palavra **objetos** possui significado computacional, ou seja, por objeto entenda como um sinônimo de classe. Quando a expressão **objetos matemáticos** for usada entenda-a como pontos, retas, polígonos, gráficos de funções, ou seja, quais quer que sejam as figuras geradas pelo GeoGebra (ou pelo Manim, como será abordado mais a frente) que podemos interagir e visualizar através da interface.

[13] Tradução nossa: *Apart from the standalone application, GeoGebra also allows the creation of interactive web pages with embedded applets. These targeted learning and demonstration environments are freely shared by mathematics educators on collaborative online platforms like the GeoGebraWiki (www.geogebra.org/wiki). The number of visitors to the GeoGebra website has increased from about 50,000 during 2004 to more than 5 million during 2010 (see Figure 1) coming from over 180 countries (HOHENWATER; LAVICZA, 2011)*

- **Geometria euclidiana plana**: a versão de geometria do GeoGebra oferece um ambiente dinâmico para construções geométricas. Usuários podem criar figuras, experimentar com transformações e explorar conceitos de geometria em uma plataforma interativa.

- **Geometria espacial**: esta versão permite a exploração de objetos matemáticos tridimensionais. Usuários podem criar e manipular sólidos, superfícies e curvas tridimensionais, explorando a matemática em um espaço tridimensional.

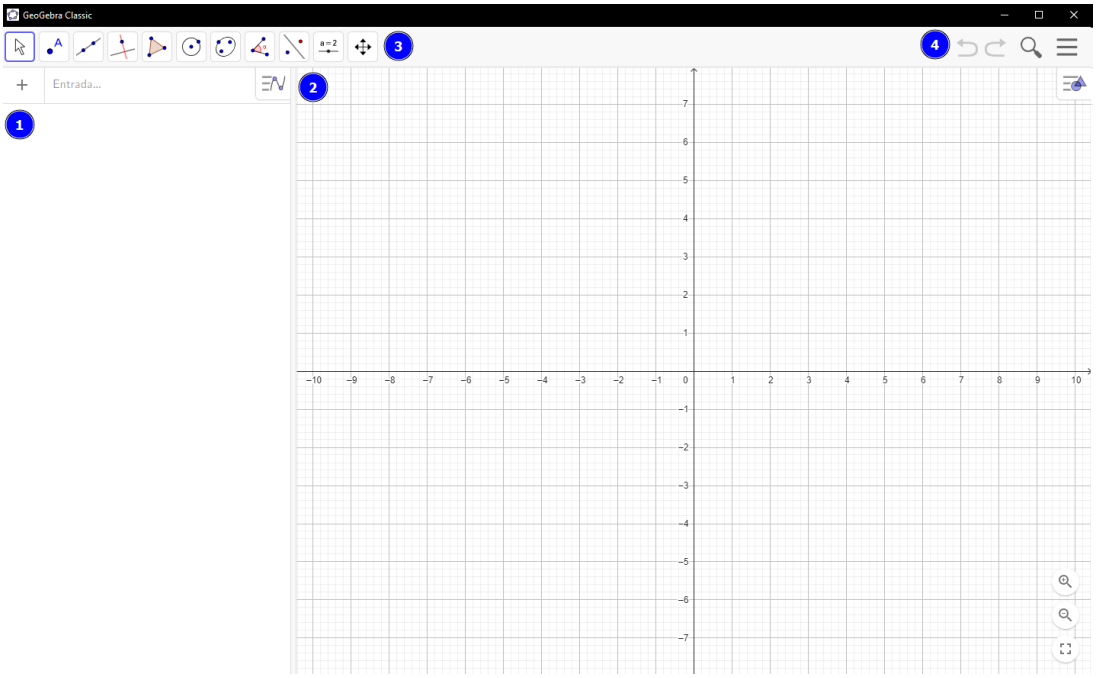
- **Planilhas**: o modo de planilhas do GeoGebra oferece uma interface dinâmica para organizar dados numéricos, permitindo cálculos automáticos e análise visual de padrões numéricos, uma ferramenta flexível e intuitiva para explorar relações e realizar experimentos numéricos.

As principais ferramentas do GeoGebra são, Calculadora gráfica, Geometria, Calculadora 3D Cada uma dessas versões tem sua própria gama de ferramentas e funcionalidades dedicadas a atender às necessidades específicas dos usuários e se adaptando a cada um dos cenários descrito acima.

2.3.1 Interface do GeoGebra

O GeoGebra, reconhecido por sua versatilidade, apresenta diferentes formas de acesso, incluindo uma interface web acessível a partir de navegadores e aplicativos offline denominados *GeoGebra Classic*. Essa dualidade de opções proporciona flexibilidade aos usuários, permitindo o uso tanto em ambientes online quanto offline, adaptando-se às diversas necessidades. Na Figura 6, podemos ver a interface do *GeoGebra Classic* versão 6.

Figura 6 – Tela inicial do GeoGebra Classic



Fonte: Elaborado pelo autor (2023)[14].

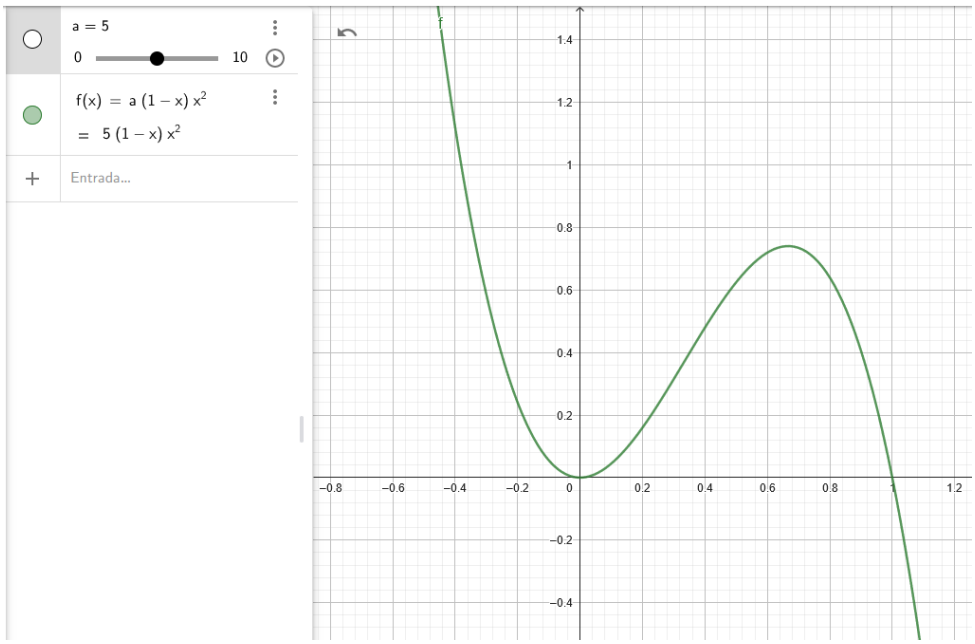
Em **(1)**, a "Paleta de Comandos"oferece uma ampla gama de funções e comandos, permitindo a inserção de expressões matemáticas para exibição e construção de gráficos e execução de comandos específicos do GeoGebra. A manipulação de objetos é centralizada na área de trabalho principal, destacada em **(2)**, localizada à direita da paleta. Nessa área, os usuários podem modificar e explorar objetos matemáticos, como pontos, retas e curvas.

A barra de ferramentas, destacada por **(3)** e **(4)** fornece acesso rápido a diversas funcionalidades, simplificando o processo de criação. Com botões específicos **(3)**, os usuários podem adicionar pontos, retas, círculos e outras construções matemáticas com apenas alguns cliques. Mais a direita , em **(4)**, há outras opções de configuração do GeoGebra.

A experiência do usuário no GeoGebra Classic, assim como o GeoGebra disponível pela aplicação *web*, é marcada pela interface simples para criar e interagir com objetos matemáticos de forma visual e dinâmica. Um exemplo de como o software pode ser usado é mostrado na Figura 7.

[14] GeoGebra Classic 6, versão 6.0.811.0, Disponível em: https://www.geogebra.org/download Acesso em: 15/11/2023

Figura 7 – GeoGebra mostrando uma função usando um seletor



Fonte: Elaborada pelo autor (2023)

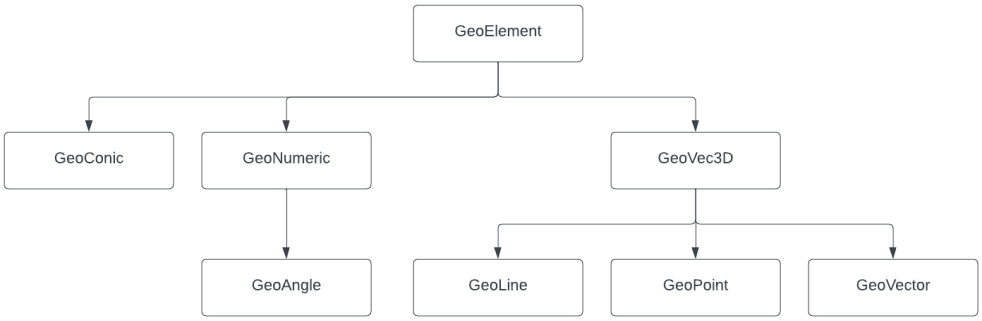
Na Figura 7 podemos ver a função $f : \mathbb{R} \to \mathbb{R}$ definida como $f(x) = a(1-x)x^2$, em que a é um parâmetro que podemos controlar com o seletor. O seletor é uma ferramenta de seleção no GeoGebra que oferece uma abordagem interativa, permitindo que os usuários ajustem dinamicamente valores de parâmetros. Ao movimentar o controle, é possível explorar e modificar diferentes aspectos de modelos matemáticos, proporcionando uma experiência prática e visual na manipulação de variáveis. No exemplo acima podemos mudar a posição do seletor mostrado à esquerda da tela, assim, mudando o valor do parâmetro a, e como discutido na Seção 2.3.2, quando mudamos uma dependência de um objeto o mesmo irá se alterar de acordo com as mudanças, nesse exemplo ao mudarmos o valor de a teremos uma alteração no gráfico de nossa $f(x)$.

No exemplo da Figura 7 podemos usar o seletor para mostrar que o ponto crítico $(0,0)$ continua sendo um mínimo local conforme variamos o valor de a, mantendo $a > 0$, porem o mesmo não vale para o outro ponto critico, que irá variar de acordo com o valor de a. Podemos observar esse comportamento conforme variamos o valor de a usando o seletor dentro do GeoGebra. No campo educacional, destacamos os trabalhos de Moura et al. (2016), na qual aborda uma serie de aplicações possíveis para o GeoGebra no ambiente educacional, dentre elas, o uso enquanto ferramenta para visualização de conceitos matemáticos contidos em problemas e exercícios disponibilizados pelos autores da pesquisa.

2.3.2 A dinâmica entre objetos no GeoGebra

Desde o início do seu ciclo de desenvolvimento, Markus Hohenwarter levou como prioridade a precisão matemática dos objetos criados pelo GeoGebra e a interatividade entre esses objetos. No seu trabalho *GeoGebra Ein Softwaresystem für dynamische Geometrie und Algebra der Ebene* (2002) podemos ver, em detalhes, todas as funcionalidades e particularidades que o GeoGebra possui. Em especial, na Seção 8.1.3, Markus traz uma discussão sobre um funcionalidade que podemos observar até hoje dentro do GeoGebra que é a dependência entre os objetos no GeoGebra. Como discutimos antes, a palavra objeto é colocada aqui no sentido computacional, ou seja, por objetos queremos dizer que cada elemento do GeoGebra é representado por uma classe dentro do código-fonte, a Figura 8 mostra a relação entre essas classes.

Figura 8 – Organograma das relações de herança entre os objetos do GeoGebra



Fonte: Elaborada pelo Autor, referência em Hohenwarter (2002, p. 160)

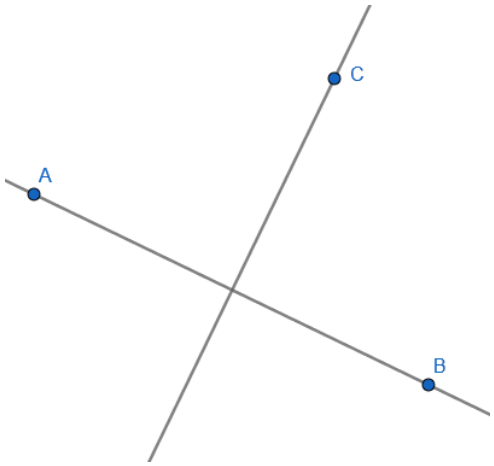
Na Figura 8 podemos ver que cada objeto matemático que criamos dentro do GeoGebra possui sua respectiva classe, em termos de linguagem de programação. Essas classes armazenam propriedades relevantes para o determinado objeto matemático, como posição, cor, entre outras propriedades que irão variar de acordo com a natureza do objeto matemático em questão. Além disso, o organograma nos mostra como as propriedades são herdadas de objeto a objeto, por exemplo, o objeto *GeoLine* herda características do objeto *GeoVec3D*, que por sua vez herda características do objeto *GeoElement*, que é mostrado como um objeto pai.

A dinâmica entre objetos no GeoGebra nada mais é do que a dependência entre os objetos. Por exemplo, usando o GeoGebra para criar uma reta passando por dois pontos A e B, temos que se qualquer um desses pontos tiverem suas posições alteradas, então a reta também mudará de acordo. A Figura 9 ilustra a afirmação a seguir:

> As dependências dos objetos são representadas por meio de algoritmos. A reta g que passa por dois pontos A e B depende desses dois pontos. Se um dos pontos A ou B mudar, a reta g deverá ser recalculada. Como isso funciona com nossas estruturas de

> dados e algoritmos? Cada objeto geométrico (GeoElement) possui uma lista de seus algoritmos dependentes. Quando um GeoElement muda, todos os seus algoritmos dependentes são atualizados. Esses algoritmos recalculam todas as suas saídas. Cada uma dessas saídas é um GeoElement e pode, por sua vez, ter algoritmos dependentes que agora também precisam ser atualizados, etc [15] (HOHENWARTER, 2002)

Figura 9 – Reta passando por dois pontos A e B e uma reta perpendicular a ela passando por um ponto C

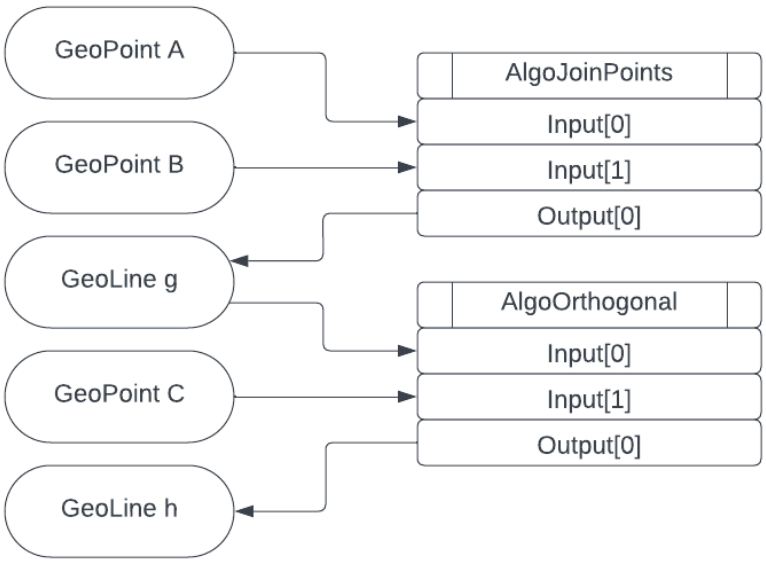


Fonte: Elaborada pelo Autor (2023)

É possível ver uma reta que passa por dois pontos: A e B e uma reta perpendicular a essa reta passando por um ponto C. Essa mesma situação é analisada por Hohenwarter(2002), como podemos ver na Figura 10.

[15] Tradução nossa: *Die Abhängigkeiten der Objekte werden mit Hilfe von Algorithmen dargestellt. Die Gerade g durch zwei unkte A und B hängt von diesen beiden Punkten ab. Ändert sich einer der Punkte A oder B, so muss die Gerade g neu berechnet werden. Wie funktioniert dies mitunseren Datenstrukturen und Algorithmen? Jedes geometrische Objekt (GeoElement) besitzt dazu eine Liste seiner abhängigen Algorithmen. Wenn sich ein GeoElement ändert, werden alle seine abhängigen Algorithmen aktualisiert. Diese Algorithmen berechnen dabei alle ihre Ausgaben neu. Jede dieser Ausgaben ist ein GeoElement und kann ihrerseits wieder abhängige Algorithmen besitzen, die nun auch aktualisiert werden müssen, usw (HOHENWARTER, 2002)*

Figura 10 – Análise dos GeoElement



Fonte:Elaborada pelo Autor, referência em Hohenwarter (2002, p. 162)

Nessa visualização, Hohenwarter traz uma explicação de como o sistema do GeoGebra gerencia os elementos para manter o dinamismo entre eles. Nesse exemplo temos dois pontos *A* e *B* que são descritos como *GeoPoint*. No GeoGebra podemos criar pontos de uma forma isolada, porem, como podemos ver na Figura 9 os pontos *A* e *B* estão sendo usados para gerar uma reta, na Figura 10 podemos ver que seus objetos correspondentes são usado como *input* para a uma função chamada *AlgoJoinPoints*, que retorna um novo objeto, o *GeoLine*, essa é justamente a reta que passa por *A* e *B* na Figura 9.

Como os pontos *A* e *B* são parâmetros necessários para a existência da reta gerada por eles, o GeoGebra age de forma retroativa, se algum dos parâmetros da função *AlgoJoinPoints*, nesse caso, ou o ponto *A* ou o ponto *B*, forem alterados a reta também será alterada. Da mesma forma, a função *AlgoOrthogonal* recebe a reta gerada pela função *AlgoJoinPoints* como parâmetro, ou seja, se a reta gerada por *A* e *B* for alterada, a reta ortogonal também será alterada.

Dessa forma, temos que as alterações entre os objetos matemáticos do GeoGebra são cadenciadas, ou seja, quando o usuário fizer qualquer alteração em qualquer objeto matemático seus respectivos objetos dentro do GeoGebra irão desencadear alterações em todos os objetos em cadeia, assim, podemos observar como o GeoGebra ganha uma de suas principais características, a dinâmica entre objetos.

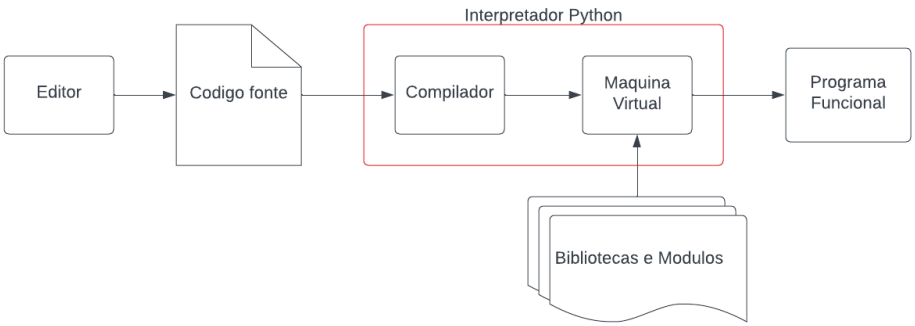
3 MANIM

O *Manim*, acrônimo para *Mathematical Animation Engine*[16], é uma biblioteca *Python* desenvolvida com o intuito de criar visualizações matemáticas precisas e dinâmicas. *Python* é uma linguagem de programação de alto nível, de propósito geral, conhecida por sua sintaxe clara e legível. O desenvolvimento do *Python* começou no final dos anos 1980, quando Guido van Rossum, um programador holandês, decidiu criar uma linguagem de *script* interpretada que fosse fácil de ler e usar. A primeira versão, *Python* 0.9.0, foi lançada em 1991. Uma das características notáveis é a comunidade ativa de desenvolvedores que contribuem para seu aprimoramento contínuo.

Com seu design intuitivo e ampla gama de bibliotecas, *Python* tornou-se uma ferramenta versátil em várias aplicações, desde desenvolvimento web até Inteligência Artificial e Ciência de Dados, por exemplo.

O *Python* é uma linguagem interpretada, o que significa que os *scripts Python* não precisam ser compilados antes de serem executados, assim como ocorre em linguagens como C ou C++, que precisam que o código-fonte seja convertido em algo que a maquina possa entender. Em vez disso, um interpretador *Python* lê e executa o código-fonte diretamente. Isso proporciona uma abordagem interativa para o desenvolvimento, permitindo que os desenvolvedores experimentem e testem trechos de código imediatamente.

Figura 11 – Diagrama do ciclo de execução de um *script Python*



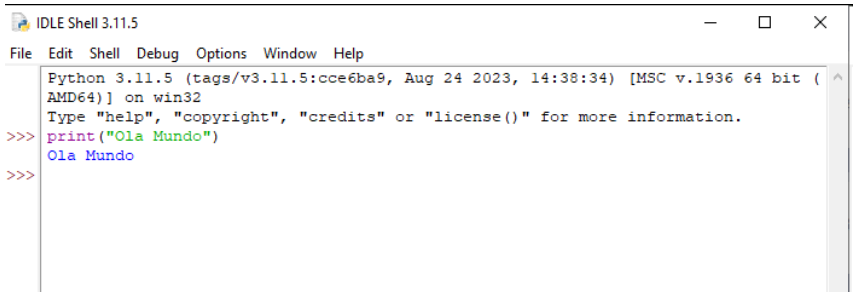
Fonte: Elaborada pelo Autor (2023)

A Figura 11 mostra o diagrama do ciclo de execução de um *script Python* que se apoia em Bibliotecas e Módulos. O interpretador *Python* pode ser acessado no terminal ou *prompt* de comando, digitando *python* seguido do nome do arquivo ou diretamente interagindo com o *prompt* interativo.

[16] Disponível em: https://www.manim.community/ Acesso em: 02/11/2023

Como a maioria das linguagens de quarta geração, a implementação de *scripts Python* é apoiada por *IDEs* (*Integrated Development Environment*) ou ambiente de desenvolvimento integrado, que pode ser tanto a básica quanto a Visual *Studio Code*, sendo a IDE principal. IDEs são editores de texto preparados para programação, dessa forma, esses editores de texto possuem ferramentas uteis para as linguagens suportadas, por exemplo, IDEs *Python* reconhecem algumas funções especificas do *Python*, dessa forma essas IDEs podem auxiliar o programador com funcionalidade uteis como por exemplo *autocomplete*. A Figura 12 mostra um exemplo de implementação do código clássico *"Hello World"* na *IDLE Python*.

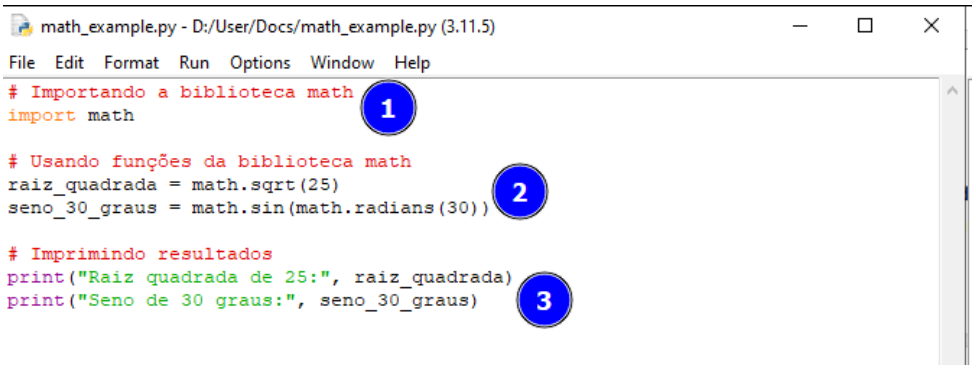
Figura 12 – IDLE *Python* executando o comando *print*



Fonte: Elaborada pelo Autor (2023)

A *IDLE* (*Integrated Development and Learning Environment*) é uma *IDE* básica incluída na instalação padrão do *Python*, um ambiente simples para escrever e executar *scripts Python*. Do ponto de vista de funções matemáticas, o *Python* requer o uso da biblioteca *math*. A Figura 13 traz um exemplo de sua aplicação para o cálculo de raiz quadrada e seno.

Figura 13 – Exemplo de código usando a biblioteca *math*



Fonte: Elaborada pelo Autor (2023)[17]

3.1 USABILIDADE DO MANIM

O *Manim* é uma biblioteca *Python* desenvolvida originalmente por Grant Sanderson, o criador do popular canal do *YouTube*, "*3Blue1Brown*"[18]. Especificamente projetado para criar animações matemáticas, o *Manim* facilita à criação de videos e de imagens para a visualização de conceitos em matemática, fornecendo uma ampla gama de ferramentas para a criação de objetos matemáticos.

Grant Sanderson introduziu o *Manim* para preencher uma lacuna na criação de animações matemáticas educativas de alta qualidade. O projeto inicial, hoje conhecido como *ManimCE* (*Community Edition*), impulsionada pela comunidade de desenvolvedores, evoluiu para a versão mais recente,*ManimGL*.

Atualmente, o *ManimCE*[19] é a versão mais amplamente adotada. Ela oferece uma série de melhorias, correções de *bugs* e recursos adicionais em comparação com as primeiras versões do *Manim*. Uma distinção notável entre o *ManimCE* e o *ManimGL* é a escolha das bibliotecas gráficas. Enquanto o *ManimCE* utiliza o Cairo para manipulação de gráficos, o *ManimGL* opta pelo *OpenGL*. Essa diferenciação impacta diretamente a forma como as animações são renderizadas e processadas, contribuindo para as características específicas de cada versão.

Ao criar *scripts* com o *Manim*, é crucial entender a diferença fundamental na execução. Enquanto *scripts Python* convencionais são executados através do interpretador, como vimos anteriormente na Figura 11, os *scripts Manim* passam por um processo diferente. Na Figura 14 podemos ver como o *Manim* gerencia os arquivos *Python* para criar as animações.

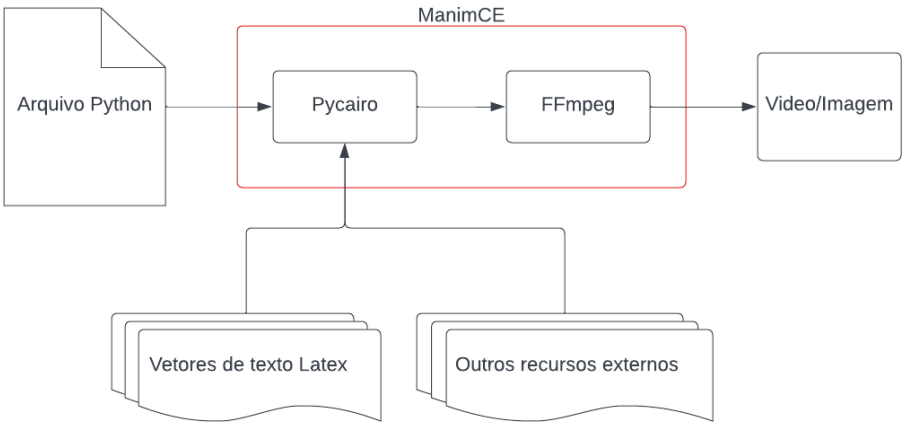
---

17 Código criado e testado no Python versão 3.11.5

18 Disponivel em: https://www.youtube.com/@3blue1brown Acesso em: 07/12/2023

19 Em nosso capítulo, nos concentraremos principalmente no *ManimCE*, explorando suas capacidades e recursos.

Figura 14 – Diagrama de funcionamento do ManimCE



Fonte: Elaborada pelo Autor (2023)

Nosso arquivo *Python* é inicialmente convertido em uma sequência de imagens usando *Pycairo*, que nada mais é do que uma outra biblioteca *Python* que possui inúmeras ferramentas para interagir com o Cairo[20]. Em seguida, as imagens são transformadas em outros formatos de arquivos, como vídeos, por exemplo, usando o *FFmpeg*[21].

O processo de a criação das animações pode incluir recursos externos diversos, ou seja, é possível inserir imagens dentro das animações. Textos LaTeXpodem ser escritos diretamente nas animações. Durante as interações do *Pycairo*, em que será convertido para um arquivo do tipo *Scalable Vector Graphic*, ou SVG[22] e então serão adicionados a animação final[23].

3.2 ENTENDENDO A ESTRUTURA DE UMA ANIMAÇÃO MANIM

Assim como o *Geogebra*, o *Manim* é usado para criar objetos matemáticos e gerar interações entre eles. Diferentemente do *Geogebra*, o Manim não possui uma interface gráfica para interagirmos com esses objetos. *Manim* é um biblioteca *Python*, com isso, criar animações usando *Manim* requer um pouco de conhecimento sobre desenvolvimento *Python*. Para entender como o *Manim* usa a estrutura *Python* para gerar animações vamos analisar linha a linha o

[20] Mais detalhes sobre a biblioteca Cairo podem ser encontrados em: https://www.cairographics.org/. Acesso em: 24/11/2023

[21] O *FFmpeg* nada mais é do que um *software* de código aberto capaz de gerenciar e converter arquivos em diversos tipos de formatos. Mais detalhes sobre o *FFmpeg* podem ser encontrados em: https://ffmpeg.org/about.html. Acesso em: 24/11/2023

[22] Esses arquivos são baseados em XML, e descrevem de forma vetorial desenhos e gráficos bidimensionais

[23] Para mais informações sobre esse processo, acesse a documentação do *Manim*. Disponível em: https://docs.manim.community/en/stable/index.html Acesso em: 29/11/2023

funcionamento de cada função presente no exemplo do Algoritimo 24.

Algoritmo 1: Código *Manim* que apresenta 3 figuras na tela

```
1  from manim import *
2
3  class Figuras(Scene):
4      def construct(self):
5          circulo = Circle(color=RED)
6          self.play(Create(circulo))
7          self.wait(1)
8
9          quadrado = Square()
10         quadrado.set_color(BLUE)
11         quadrado.set_fill(color=BLUE,opacity=0.2)
12         quadrado.move_to([0,-2.5,0])
13         self.play(Create(quadrado))
14         self.wait(1)
15
16         tri = Triangle(color=GREEN)
17         tri.set_fill(color=GREEN,opacity=0.7)
18         tri.move_to([0,2.5,0])
19         self.play(FadeIn(tri))
20         self.wait(1)
```

Na primeira linha do código temos um `import` da biblioteca *Manim*, assim como muitas outras bibliotecas *Python*, *Manim* tem seus objetos e funções predefinidas que precisam ser importados caso queiramos usa-los.

Mais a baixo, na linha 3, temos a definição de uma classe do *Python*, nesse caso, *Manim* usa isso para identificar as cenas de uma certa animação, no exemplo, temos uma cena chamada `Figuras`. Na linha 4, definimos o método `construct`, essa é uma função padrão do ambiente *Manim* e é dentro dela que definimos todos os objetos e animações.

Note que, o código do Algoritimo 24 está separado convenientemente em três partes, cada uma dessas partes é responsável por definir uma figura dentro da animação. Na primeira parte, da linha 5 á 7, definimos uma variável `circulo`, na linha 5, nela atribuímos o valor da função `Circle()`, essa é uma das muitas funções no *Manim* que são usadas para criar objetos matemáticos, nesse caso a função é responsável por criar um circulo.

Existem umas serie de formas para modificar, os objetos matemáticos dentro do *Manim*, nesse caso, passamos um parâmetro para a função, nesse caso, o parâmetro `color`, isso faz com que a cor das linhas que formam a figura mudem para a cor indicada.

Na linha 6, indicamos o método de animação, dentro do `self.play()`, nesse caso temos o método `Create()`, existem muitos outros métodos, durante o Capitulo 4.2 veremos outros métodos em mais detalhes. Por ultimo, na linha 7, temos outra função do *Manim* que apenas adiciona uma pausa dentro da animação. Note que, nessas linhas criamos um objeto matemático, nesse caso um circulo, assim como no *Geogebra*, podemos definir as características do objeto matemático, nesse caso alteramos apenas a cor, mas poderíamos alterar também o raio,

[23] Animação pode ser visualizada em: https://kallelfiori.com/index.php/2024/03/14/introducao-ao-manim/. Acesso em: 25/04/2024 e o código pode ser encontrado na pagina do GitHub: https://github.com/Kallel181/Manim-Calculo/tree/main/examples Acesso em: 25/04/2024

a transparência do objeto, a posição e outras propriedades.

No segundo bloco, da linha 9 à 14, definimos uma segunda figura. Nesse caso, na linha 9, temos um `quadrado`, usando a função `Square()`, dessa vez não alteramos a cor da figura diretamente pelos parâmetros da função `Square()`, ao invés disso usamos algumas outras funções que fazem essas alterações. Na linha 10, temos o uso da função `set_color()` que tem o mesmo efeito de usarmos o parâmetro `color` dentro da função `Square()`. Mais a frente, na linha 11 temos o uso da função `set_fill()`, essa função é usada para mudar o cor e a opacidade da região interna das figura, nesse caso estamos alterando o cor interna do nosso quadrado para a cor azul, mais a frente usamos também o parâmetro `opacity`, nesse caso colocamos o valor `0.2`, assim indicamos que a opacidade da parte interna do quadrado precisa ser de 20%.

Dessa vez, diferente do exemplo da figura anterior, temos que alterar a posição do nosso quadrado. Por padrão todos os objetos dentro do *Manim* são colocados no centro da tela, podemos imaginar a tela do *Manim* como um grande plano cartesiano, com a origem no centro da tela, com isso, se quisermos deixar nosso quadrado mais a baixo precisamos alterar sua coordenada *y* para algo negativo, para isso usamos a função `move_to()` que move o objeto de acordo com as coordenada inseridas. No `Manim` as coordenadas precisam ser usadas em formato de *Array*, nesse caso temos três coordenadas dentro do *Array*, a primeira referente a coordenada *x*, a segunda referente a coordenada *y* e a terceira referente a coordenada *z*. No caso, usamos a função `move_to()` passando um *Array* `[0,-2.5,0]`, efetivamente movendo o objeto mais abaixo.

Uma observação muito importante que temos que ressaltar é, mudar o posicionamento dos objetos na tela não necessariamente irá criar uma animação do mesmo se movendo na tela, caso o objetivo seja animar um objeto se movendo precisamos usar métodos dentro da função `self.play()`, uma vez que a mesma é responsável por gerenciar animações e movimentos.

Nas linhas 13 e 14, fazemos assim como no bloco de código anterior, quando finalizamos a construção da figura desejada e todos os seus parâmetros foram configurados de acordo com nossas preferencias, podemos usar um método de animação para inserir a figura na tela, com a função `self.play()` e realizamos uma pausa com a função `self.wait()`.

No terceiro bloco, da linha 16 à 20, temos a definição e animação de uma terceira figura, dessa vez, na linha 16 definimos um triângulo com o nome de `tri` com a função `Triangle()`, assim como no circulo, usamos o parâmetro `color` dentro da função `Triangle()` para definir a cor da borda da figura. Na linha 17, temos a função `set_fill()`, assim como no exemplo do quadrado, definimos os parâmetros `color` como `GREEN` e `opacity` como `0.7`, assim deixando o

[23] Dentro do *Manim* algumas cores já são predefinidas, para ver a lista de cores já configuradas dentro do ambiente acesse: https://docs.manim.community/en/stable/reference/manim.utils.color.manim_colors.html Acesso: 28/04/2024

[23] O posicionamento dos objetos vai ser discutido em mais detalhes durante o Capitulo 4.2, mas a forma ideal de posicionar esses objetos nesse caso seria usando a função `next_to()`, mas, para manter a simplicidade em um exemplo inicial escolhemos fazer da forma apresentada.

[23] Mesmo trabalhando em uma visualização no plano é importante informa a coordena *z* de um objeto já que isso é usado para gerenciar seu posicionamento nas "camadas", ou seja, objetos com *z* maior são desenhados acima de objetos com *z* menor

interior do triângulo verde e com uma opacidade de 70%.

Assim como no quadrado, precisamos mover o triângulo mais acima, nesse casos fazemos um processo semelhante com a função `move_to()`, na linha 18, mas desta vez, movemos o objetos para as coordenadas `[0,2.5,0]`, efetivamente deixando-o mais acima do circulo.

As linha 19 e 20 são como nos dois exemplos anteriores das duas figuras, realizamos um método de animação com o `self.play()` para adicionar o triângulo na tela, porem, dessa vez, usamos um método de animação diferente, `FadeIn()`. Na linha 20, temos um ultimo `self.wait()` esperando 1 segundo, e isso define o fim da nossa animação.

Durante o processo de criação de uma animação *Manim* é importante ter em mente de forma clara o que se deseja criar e sempre visitar a documentação[24] da ferramenta para verificar o funcionamento de partes especificas da ferramenta. De forma geral, todas as animações feitas com *Manim* seguem essa estrutura básica, muito embora existam outros conceitos e funções que precisam ser usados para animações mais complexas, essa estrutura se mantém a mesma.

O exemplo apresentado no Algoritimo 24 por mais que seja algo simples, mostra com precisão boa parte da estrutura de uma animação *Manim*, muito embora existam muitos outros conceitos, funções e métodos diferentes, essa estrutura de mantém relativamente intacta independentemente da animação construída.

3.3 APLICANDO O MANIM

O *Manim* oferece uma ampla gama de recursos que permitem aos usuários expressar conceitos matemáticos de maneira clara e visualmente claras. Desde a representação de equações até a animação de gráficos, o Algoritimo 2 traz a aplicação de um código que gera o gráfico das

[24] A documentação pode ser encontrada no seguinte link: https://docs.manim.community/en/stable/ Acesso: 25/04/2024

funções seno e cosseno[25].

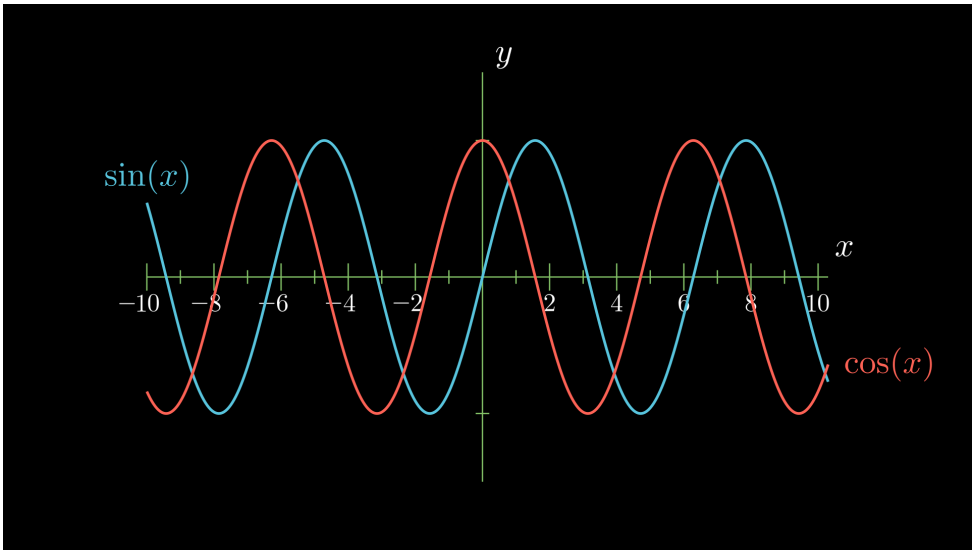
Algoritmo 2: Código que gera um gráfico da função seno e cosseno em Manim.

```
14  from manim import *
15
16  class graph(Scene):
17    def construct(self):
18      axes = Axes(
19        x_range=[-10, 10.3, 1],
20        y_range=[-1.5, 1.5, 1],
21        x_length=10,
22        axis_config={"color": GREEN},
23        x_axis_config={
24          "numbers_to_include": np.arange(-10, 10.01, 2),
25          "numbers_with_elongated_ticks": np.arange(-10, 10.01, 2),
26         },
27          tips=False,
28      )
29
30      axes_labels = axes.get_axis_labels()
31      sin_graph = axes.plot(lambda x: np.sin(x), color=BLUE)
32      cos_graph = axes.plot(lambda x: np.cos(x), color=RED)
33
34      sin_label = axes.get_graph_label(
35      sin_graph, "\\sin(x)", x_val=-10, direction=UP / 2
36      )
37      cos_label = axes.get_graph_label(cos_graph, label="\\cos(x)")
38
39      self.play(Write(axes),Write(axes_labels))
40      self.wait(1)
41      self.play(Write(sin_graph),Write(sin_label))
42      self.wait(1)
43      self.play(Write(cos_graph),Write(cos_label))
44      self.wait(2)
```

Neste exemplo, utilizaremos o *Manim* para criar uma animação da representação gráfica das funções seno e cosseno. O *Manim* proporciona um controle detalhado sobre a aparência de objetos visuais, permitindo uma personalização minuciosa de cada elemento na cena. No Algoritimo 2 definimos o objeto `axes` usando a classe `Axes` sendo possível construir o gráfico da maneira desejada, mudando os *parâmetros* da classe. Dessa forma o *Manim* alterará a figura final. Um dos parâmetros alterados é o `color`, dentro de `axis_config`. Nesse caso alteramos o código para que esse parâmetro seja igual a `GREEN`, conforme mostrado na Figura 15 em que é possível perceber que os eixos do gráfico gerado pelo *Manim* estarão na cor verde.

[25] Animação pode ser visualizada em: https://www.youtube.com/watch?v=L_MYNpRIQ0s. Acesso em: 29/11/2023

Figura 15 – Gráfico gerado pelo *Manim* usando o código da Figura 2



Fonte: Elaborada pelo Autor (2023), Disponível no link[26]

Os eixos do gráfico possuem uma série de outros parâmetros, como por exemplo `x_range` e `y_range`, esses parâmetros receberam uma lista de três valores. Nesse caso, precisamos indicar o valor mínimo para o eixo, o valor máximo para o eixo e o incremento. Na Figura 2 o `x_range` possui a lista de valores `[-10,10.1,1]`, dessa forma, instruímos o *Manim* que nosso eixo x deve começar em -10 e ir até 10.1 e criando uma marcação a cada incremento de 1.

Mais a frente podemos configurar aspectos ainda mais específicos do nosso eixos modificando o parâmetro `x_axis_config`, para esse parâmetro temos uma lista de valores e propriedade que precisam ser inseridas, nessa lista modificamos os valores de `numbers_to_include` e `numbers_with_elongated_ticks`, o primeiro parâmetro altera quais números devem ser incluídos no eixo, nesse caso, como inserimos o valor `np.arange(-10, 10.01, 2)`, o *Manim* irá colocar todos os números de -10 a 10.01 em um incremento de 2.

Já `numbers_with_elongated_ticks` usamos para indicar quais números devem ter as barras indicativas maiores, como inserimos o valor `np.arange(-10, 10.01, 2)`, os mesmos numeros que foram incluídos na figura também terão as barras indicativas maiores.

Mais adiante são criados os gráficos das funções seno e cosseno, `sin_graph` e `cos_graph` respectivamente. Nesse caso é necessário usar o método `plot()`, que é oferecida dentro o objeto `axes`. Esse método precisa receber no mínimo o método que irá definir o gráfico a ser plotado. Na Figura 2 podemos ver que para os objetos `sin_graph` e `cos_graph` adicionalmente foi usado o parâmetro `color`, que assim como nos eixos, é usado para mudar o cor do objeto em questão, nesse caso, estamos mudando as cores dos gráficos do seno e cosseno para azul e vermelho,

[26] https://www.youtube.com/watch?v=L_MYNpRIQ0s

respectivamente.

Podemos ver na Figura 15 que todas as configurações escritas no arquivo *Python* da Figura 2 são consideradas para gerar a imagem que queremos. Dessa forma, podemos ver que o *Manim* oferece uma grande quantidade de customizações, dessa forma as visualizações podem ser modificadas detalhadamente de acordo com a necessidade o usuário.

Diferentemente de ferramentas convencionais, o *Manim* concede ao usuário a capacidade de ajustar não apenas o conteúdo, mas também a estética da animação. Nas últimas linhas da Figura 2 podemos ver o uso do método `self.play()`, com esse método podemos controlar quais serão as animações de entrada para os objetos, além de podemos modificar alguns aspectos gerais dos objetos em cena. Nesse caso, usamos o método `self.play()` juntamente com o método `Write()` para adicionar os objetos a nossa animação[27].

Com o *Manim* também é possível gerar visualizações em 3D. O Algoritimo 3 traz a aplicação de um código[28].

Algoritmo 3: Código simples para gerar uma superfície $z = 2sen(x) + 2cos(y)$.

```
14  from manim import *
15
16  class PlotSurfaceExample(ThreeDScene):
17    def construct(self):
18      resolution_fa = 32
19      self.set_camera_orientation(phi=75 * DEGREES, theta=-60 * DEGREES)
20
21      axes=ThreeDAxes(x_range=(-3,3,1),y_range=(-3,3,1),z_range=(-5,5,1))
22
23      def param_trig(u, v):
24        x = u
25        y = v
26        z = 2 * np.sin(x) + 2 * np.cos(y)
27        return z
28
29      trig_plane = axes.plot_surface(
30        param_trig,
31        resolution=(resolution_fa, resolution_fa),
32        u_range = (-3, 3),
33        v_range = (-3, 3),
34        colorscale = [BLUE, GREEN, YELLOW, ORANGE, RED],
35      )
36
37      self.wait(1)
38      self.play(Create(axes))
39      self.wait(1)
40      self.play(Write(trig_plane))
41      self.wait(2)
42      self.move_camera(phi=75*DEGREES,theta=60*DEGREES,run_time=4)
43      self.wait(2)
```

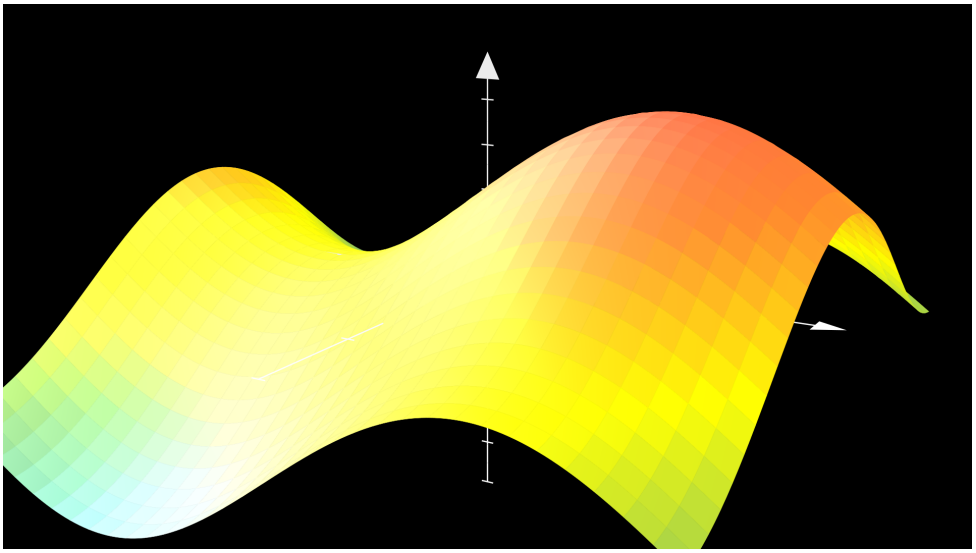
Neste exemplo, utilizaremos o *Manim* para criar uma animação da superfície $z = 2sen(x) + 2cos(y)$. O *Manim* oferece uma experiência tridimensional, permitindo-nos explorar visualmente

[27] Video da animação disponível em: https://www.youtube.com/watch?v=L_MYNpRIQ0s Acesso em: 29/11/2023

[28] Animação pode ser visualizada em: https://www.youtube.com/watch?v=-pPEZnELZ8c Acesso em: 29/11/2023

a geometria dessa superfície.

Figura 16 – Superfície gerada pelo *Manim*



Fonte: Elaborada pelo Autor (2023), Disponível no link[29]

O destaque dessa animação é a movimentação suave pela cena tridimensional, com opções de alterar a perspectiva, a elevação e a distância, proporcionando uma visualização dinâmica da superfície.

Ao trabalhar com cenas em três dimensões o *Manim* irá exigir algumas configurações adicionais, nesse caso, no começo do código mostrado na Figura 3 podemos ver a presença de duas linhas que não criam nenhum objeto, temos a variável `resolution_fa` que será usada posteriormente para definir a resolução da malhar que será usada para fazer a superfície, e por último, outra configuração que remos que fazer é posicionar a posição da câmera da cena, usando o método `set_camera_orientation()`.

Entenda a câmera da cena como uma câmera de fato, ela está sempre apontada para o centro da cena[30] mostrando quais quer objetos que estejam em seu campo de visão, nesse caso para alterar sua posição usamos coordenadas esféricas, no exemplo da Figura 3 configuramos a câmera com os seguintes parâmetros `phi = 75 * DEGREES` e
texttttheta = -60 * DEGREES, esses valores ajustam o posicionamento da câmera de forma a obtermos o posicionamento que pode ser observado na Figura 16.

Feita essa configuração inicial, é definido mais abaixo os eixos de referência para nossa superfície, usando a classe `ThreeDAxes`, semelhante ao exemplo anterior, o Manim oferece uma

[29] https://www.youtube.com/watch?v=-pPEZnELZ8c

[30] Na documentação podemos ver que o método `set_camera_orientation()` pode receber um parâmetro chamado `frame_center`, nesse parâmetro podemos inserir algum outro objeto da cena, dessa forma, mudamos o centro da câmera para o centro desse objeto.

série de parâmetros configuráveis, nesse exemplo vamos modificar apenas o comprimento desses eixos, usando os parâmetros `x_range`, `y_range` e `z_range`.

Para se criar uma superfície, primeiro, precisamos definir um método no *Python* que retorne os valores de z de acordo com os valores de x e y, para isso é criado o método `param_trig`, nesse exemplo, a superfície que vamos criar é dada por $z = 2sen(x) + 2cos(y)$. Com esse método criado, usa-se em seguida o método `plot_surface` que é oferecido na classe `ThreeDAxes`, nesse método precisamos fornecer o método criado anteriormente que define nossa superfície, nesse caso, `param_trig`, qual o tamanho da superfície, nesse caso, usamos os parâmetros `u_range` e `v_range`. Os outros parâmetros dessa classe definem outros aspectos da superfície, `colorscale` faz uma transição dentre as cores inseridas na superfície e o parâmetro resolutivo define quantos vértices a malha da superfície deve ter, dessa forma, quanto maior for a variável definida anteriormente como `resolutio_fa` mais subdivisões terá a superfície.

Por último, são apresentados os objetos matemáticos criados, nesse caso, criamos os eixos, depois usamos o método `Write` para apresentar a superfície e no fim, a câmera é deslocada para uma nova posição.

Um último exemplo[31], vamos criar uma dependência dinâmica entre objetos através do gráfico de uma função exponencial, o Algoritimo 4 traz aplicação de um código que gera um gráfico para a função $f(x) = e^x$ e atualiza as coordenadas de um ponto que caminha sobre o gráfico.

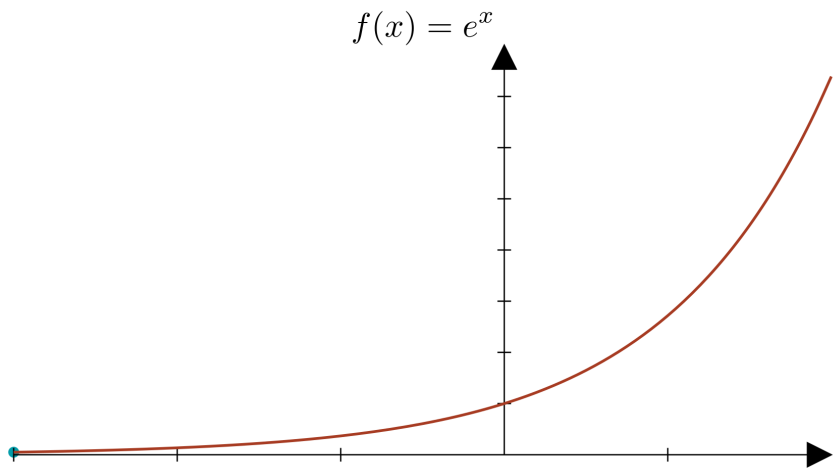
[31] Animação pode ser visualizada em: https://www.youtube.com/watch?v=NAQCvr9Df4Y Acesso em: 05/11/2023

Algoritmo 4: Código simples para gerar o gráfico da função $f(x) = e^x$ e atualizar as coordenadas de um ponto que caminha sobre o gráfico.

```
28  from manim import *
29
30  class Updater(Scene):
31    def construct(self):
32      axes = Axes(
33        x_range=[-3, 2],
34        y_range=[0, 8],
35        axis_config={"color": WHITE},
36       )
37      t = ValueTracker(-3)
38
39      exp_graph = axes.plot(lambda x: np.exp(x), color=BLUE)
40      exp_label = MathTex("f(x)=e^{x}").to_edge(UP)
41      dot = Dot(color=RED).move_to(axes.coords_to_point(-3, np.exp(-3)))
42
43      self.play(
44         Create(axes),
45         Write(exp_label),
46         Create(exp_graph),
47         Create(dot)
48         )
49
50      def update_dot(dot):
51        dot.move_to(axes.coords_to_point(
52          t.get_value(),
53          np.exp(t.get_value()))
54          )
55      dot.add_updater(update_dot)
56
57      self.play(
58        t.animate.set_value(2),
59        run_time = 3
60      )
61      self.wait(1)
```

Na Figura 17 encontra-se uma imagem do gráfico gerado pelo *Manim*.

Figura 17 – Imagem gerada a partir do código apresentado na Figura 4



Fonte: Elaborada pelo Autor (2023), Disponível no link[32]

Uma funcionalidade essencial do *Manim* que merece destaque é a capacidade de utilizar *updaters*. Através do método `add_updater()`, o *Manim* possibilita a vinculação dinâmica entre dois objetos, similar ao Geogebra, como mostrado na Seção 2.3.2, permitindo que as modificações realizadas em um objeto sejam refletidas automaticamente em outros objetos conectados. Um dos objetos que é comumente usado junto aos *updaters* é o objeto `ValueTracker`.

O código começa criando um sistema de coordenadas `Axes` com intervalos específicos para os eixos x e y. Em seguida, é usado um `ValueTracker` para criar uma variável `t` que será rastreada ao longo do tempo. Esse objeto apenas serve para armazenar um valor especifico e mudar ele de acordo com comandos futuros.

Essa sincronização dinâmica é particularmente valiosa ao criar animações complexas, onde diversos elementos precisam evoluir de maneira coordenada. Ao utilizar `updaters`, torna-se possível atrelar propriedades específicas de objetos, como posição, escala ou orientação, de modo que qualquer alteração em um desses elementos resulta em uma atualização automática nos demais objetos vinculados. Esse comportamento segue um comportamento semelhante ao que foi mostrado na Seção 2.3.2 na Figura 10, porem com o *Manim* nós temos total controle de como e quais objetos estão vinculados.

Após isso, o código cria o gráfico de uma função exponencial e exibe a expressão matemática correspondente no topo da cena. Um ponto vermelho `dot` é colocado inicialmente no gráfico na coordenada correspondente a $x = -3$.

A função `update_dot` é definida para mover o ponto ao longo do gráfico conforme

[32] https://www.youtube.com/watch?v=NAQCvr9Df4Y

`t` muda. A função `add_updater` é usada para vincular a função `update_dot` ao ponto `dot`, permitindo que ele seja atualizado automaticamente durante a animação.

A animação começa criando os objetos definidos, nesse caso, o texto LaTeX usado para indicar a função que corresponde ao gráfico, os eixos e o próprio gráfico. Mais abaixo o método `self.play` é usado de novo, mas dessa vez, mudando `t` para o valor 2 ao longo de 3 segundos. Durante essa animação, o ponto `dot` é atualizado automaticamente ao longo do gráfico da função exponencial.

3.4 CRIANDO E EXPORTANDO ANIMAÇÕES COM MANIM

Após a instalação da biblioteca e suas dependências[33], já é possível começar a criar as animações com o *Manim*, para isso, é necessário abrir um arquivo *Python*, extensão `.py`. Para verificar se a instalação foi bem sucedida é possível usar dois pequenos códigos de exemplo. O primeiro exemplo[34], mostrado no Algoritimo 5 ajuda a verificar se a biblioteca está instalada corretamente.

Algoritmo 5: Código *Manim* simples para Gerar um circulo na tela

```
1  from manim import *
2
3  class Circulo(Scene):
4      def construct(self):
5          circle = Circle()
6          circle.set_fill(ORANGE, opacity=0.5)
7          self.play(Create(circle))
```

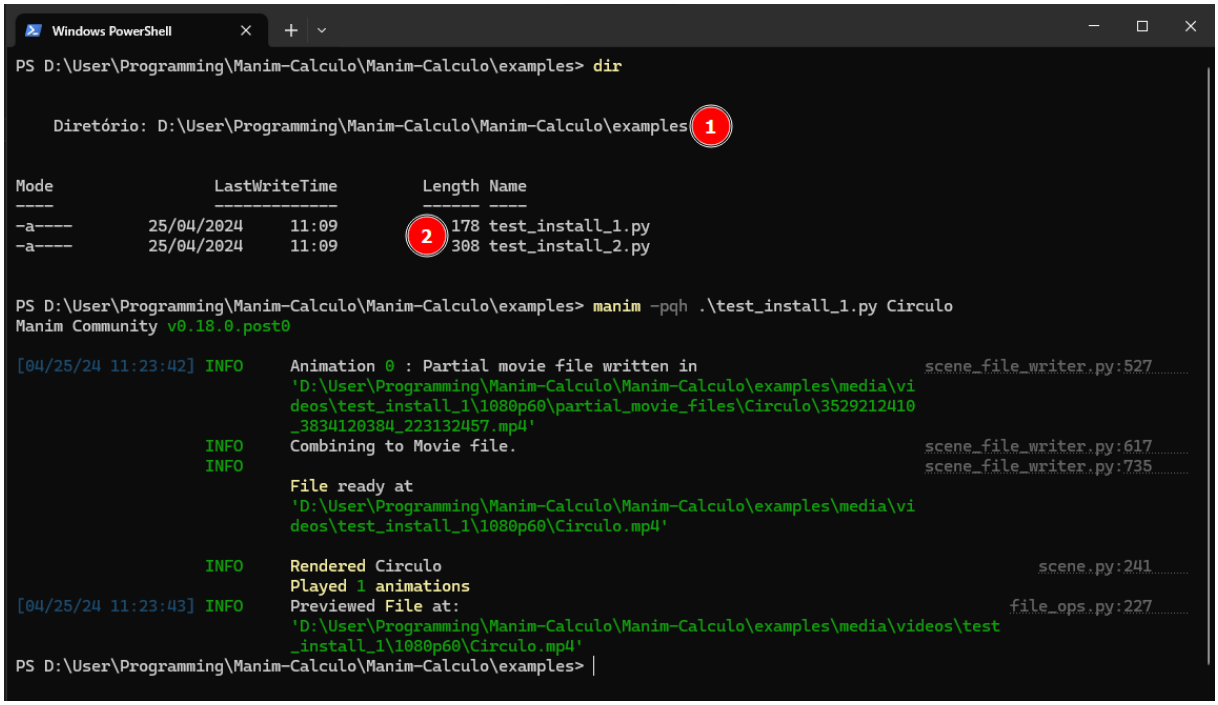
Para prosseguir com o teste de instalação é necessário localizar o arquivo criado através do terminal [35], para isso, em sistemas *Windows* podemos apenas mudar de diretório até o diretório onde esta salvo o arquivo `.py`, após navegado até o diretório do arquivo é possível usar alguns comandos do Manim para gerar um arquivo de video da animação escrita no arquivo. Para isso, é usado o comando: `manim -pqh test_install1.py Circulo`, como mostrado na Figura 18.

[33] Na documentação Oficial do *Manim* é possível encontrar um guia para a instalação da ferramenta em diferentes sistemas operacionais: https://docs.manim.community/en/stable/installation.html. Alternativamente, é possível encontrar um guia detalhado de instalação para *Windows* no site do Autor: https://kallelfiori.com/index.php/2024/03/14/introducao-ao-manim/

[34] Animação pode ser visualizada em: https://kallelfiori.com/index.php/2024/03/14/introducao-ao-manim/. Acesso em: 25/04/2024 e o código pode ser encontrado na pagina do GitHub: https://github.com/Kallel181/Manim-Calculo/tree/main/examples Acesso em: 25/04/2024

[35] O uso do terminal pode variar de acordo com o sistema operacional em questão, para esse trabalho estaremos mostrando apenas comandos para sistemas *Windows*, podem os passos podem ser replicados de forma muito semelhante em outros sintemas operacionais, com pequenas ressalvas de comando de navegação no terminal que pode ser diferentes.

Figura 18 – Gerando video com Manim



Fonte: Elaborada pelo Autor (2024)

Note que na Figura 18, precisamos inserir dois parâmetros no comando que podem variar de acordo com a aplicação, uma deles é o nome dado ao arquivo *Python*, e o outro é o nome da cena inserido na definição da classe no arquivo *Python*. Além disso, é importante navegar até o diretório onde estão salvos os arquivos, como mostrado em **(1)** na Figura 18, para se certificar que os arquivos estão no diretório atual, é necessário usar o comando `dir` no *Windows* (ou `ls` no Linux), o resultado será todos os arquivos presentes no diretório, como mostrado em **(2)** na Figura 18, dessa forma *Manim* será capaz de encontrar os arquivos e gerar os vídeos.

O processo de criação dos vídeos pelo *Manim* é altamente dependente do *pycairo* e *ffmpeg*, assim como discutido no trabalho anterior, se tudo estiver instalado corretamente e apos executado o comando, a animação construída será mostrada.

Outro exemplo importante para testar se o *Manim* foi corretamente instalado é uma animação simples que use LaTeX. Dessa forma é possível checar se o *Manim* identifica que os pacotes LaTeX foram corretamente instalados. Assim como no exemplo anterior, é possível usar o código como mostrado no Algoritimo 6

Algoritmo 6: Código *Manim* simples para Gerar um texto LaTeX

```
from manim import *

class Latex(Scene):
        def construct(self):
                text = MathTex(
                "\\frac{d}{dx}f(x)g(x)=","f(x)\\frac{d}{dx}g(x)","+",
                "g(x)\\frac{d}{dx}f(x)"
                )
                self.play(Write(text))
                self.wait(2)
```

Assim como anteriormente, é necessário navegar até o diretório desse arquivo *Python* e executar o comando: `manim -pqh test_install2.py Latex`. Se o LaTeX for encontrado corretamente e apos executado o comando, a animação construída será mostrada.

3.5 POTENCIALIDADES

As potencialidades do *Manim* quanto a visualização matemática são:

a) Com uma comunidade ativa existe uma série de outras bibliotecas que podem ser usadas para complementar as funcionalidades do *Manim*, como, por exemplo, *Manim Slides*[36] que permite a criação de *slides* dinâmicos e *Manim voiceover*[37] que permite a sincronização de faixas de áudio diretamente dentro do ambiente do *Manim*.

b) *Manim* possui uma integração direta com o ambiente LaTeX, dessa forma, qualquer tipo de documento ou escrita LaTeX pode ser usada dentro do Manim. Além disso, o ambiente também possui uma série de métodos específicos para interagir com esses textos, como formas de destacar partes do texto e animações exclusivas para esse tipo de escrita.

c) Por estar inserido no ambiente *Python*, temos acesso a uma vasta gama de bibliotecas que podem ser usadas em conjunto com *Manim*.

d) Tanto *Manim Community* quanto *ManimGL* possuem motores gráficos[38] robustos (*Pycairo* e *OpenGL*, respectivamente). Assim, a qualidade das animações, imagens ou slides gerados possuem uma alta qualidade.

[35] Animação pode ser visualizada em: https://kallelfiori.com/index.php/2024/03/14/introducao-ao-manim/. Acesso em: 25/04/2024 e o código pode ser encontrado na pagina do GitHub: https://github.com/Kallel181/Manim-Calculo/tree/main/examples Acesso em: 25/04/2024

[36] Disponível em: https://www.manim.community/plugin/manim-slides/ Acesso em: 06/11/2023.

[37] Disponível em: https://www.manim.community/plugin/manim-voiceover/ Acesso em: 06/11/2023.

[38] Motores gráficos são softwares que convertem dados geométricos em imagens visíveis, fundamentais para renderização eficiente e interação em jogos, simulações e ambientes virtuais, proporcionando experiências visuais envolventes em diversas plataformas.

4 OBJETOS DE APRENDIZAGEM

Em sua essência, um Objeto de Aprendizagem(OA) é qualquer recurso projetado com o propósito de facilitar o aprendizado. Isso pode incluir desde simples vídeos educativos até simulações interativas complexas e apresentações multimídia, ou seja, os OAs também podem ser entendidos como uma ferramenta de VM. O principal diferencial dos OAs é sua capacidade de serem reutilizáveis, adaptáveis e interoperáveis, o que significa que podem ser facilmente integrados em diferentes contextos educacionais e plataformas tecnológicas.

> Os OAs podem ser criados em qualquer mídia ou formato, podendo ser simples como uma animação ou uma apresentação de slides, ou complexos como uma simulação. Normalmente, eles são criados em módulos que podem ser reusados em diferentes contextos (AGUIAR; FLÔRES, 2014).

Os OAs são concebidos para promover uma aprendizagem mais eficaz, atendendo a uma ampla variedade de estilos de aprendizagem e necessidades dos alunos. Ao oferecerem recursos interativos dotados de esquemas visualmente atrativos e contextualizados, os OAs podem potencializar a aprendizagem da matemática em cenários em que o lápis e o papel são insuficientes.

> Para auxiliar os alunos na compreensão de conceitos mais complexos é conveniente optar por uma animação ou simulação que permita a manipulação de parâmetros e a observação de relações de causa e efeito dos fenômenos (AGUIAR; FLÔRES, 2014).

No entanto, entender como identificar e classificar OAs é fundamental para sua eficácia pedagógica. Mendes (2004) propõe uma lista de características para os OAs, que são as seguintes:

- Reusabilidade. Reutilizável diversas vezes em diversos ambientes de aprendizagem;
- Adaptabilidade. Adaptável a qualquer ambiente de ensino;
- Granularidade. Conteúdo em pedaços, para facilitar sua reusabilidade;
- Acessibilidade. Acessível facilmente via Internet para ser usado em diversos locais;
- Durabilidade. Possibilidade de continuar a ser usado, independente da mudança de tecnologia;
- Interoperabilidade. Habilidade de operar através de uma variedade de hardware, sistemas operacionais e *browsers*, intercâmbio efetivo entre diferentes sistemas.

> Os objetos com essas características são normalmente armazenados em grandes bases de dados disponíveis na Internet, chamados de repositórios. Entretanto, para que os objetos de aprendizagem possam ser localizados nos repositórios e reutilizados em diversos ambientes de aprendizagem, é necessário que eles e seu conteúdo sejam descritos de uma forma padronizada, que permita o intercâmbio de informações. Para isso são necessários padrões comuns que possibilitem o intercâmbio entre sistemas de aprendizagem na web e que facilitem o compartilhamento de recursos (MENDES; SOUZA; CAREGNATO, 2004).

Algo importante a se notar sobre essas características dos OAs é que elas podem variar por três fatores, escolhas de *design* do autor, limitações da ferramenta/tecnologia escolhida e escolhas de compartilhamento do autor, por exemplo, a acessibilidade pode variar de acordo com as formas de se exportar o OA fornecida por uma certa ferramenta e pela escolha feita para sua divulgação por parte do autor do OA. As características elencadas por Mendes (2004), nada mais são do que as partes que compõem um OA. Além dessa segmentação, os OAs podem ser classificados em diferentes tipos, conforme proposto por Barbosa (2014), que incluem:

- Apresentação. Instrução direta, usando recursos com a intenção de transmitir um conteúdo específico.
- Prática. Exercício e prática, jogo educacional ou representação que permita a prática e a aprendizagem de certos procedimentos.
- Simulação. Representação de algum processo ou sistema da vida cotidiana.
- Conceitual. Representação de conceitos chave ou conceitos relacionados ao conteúdo de uma disciplina.
- Informação. Expõe a informação organizada.
- Representação contextual. Apresentam-se os dados da maneira que emerge de um cenário autêntico.

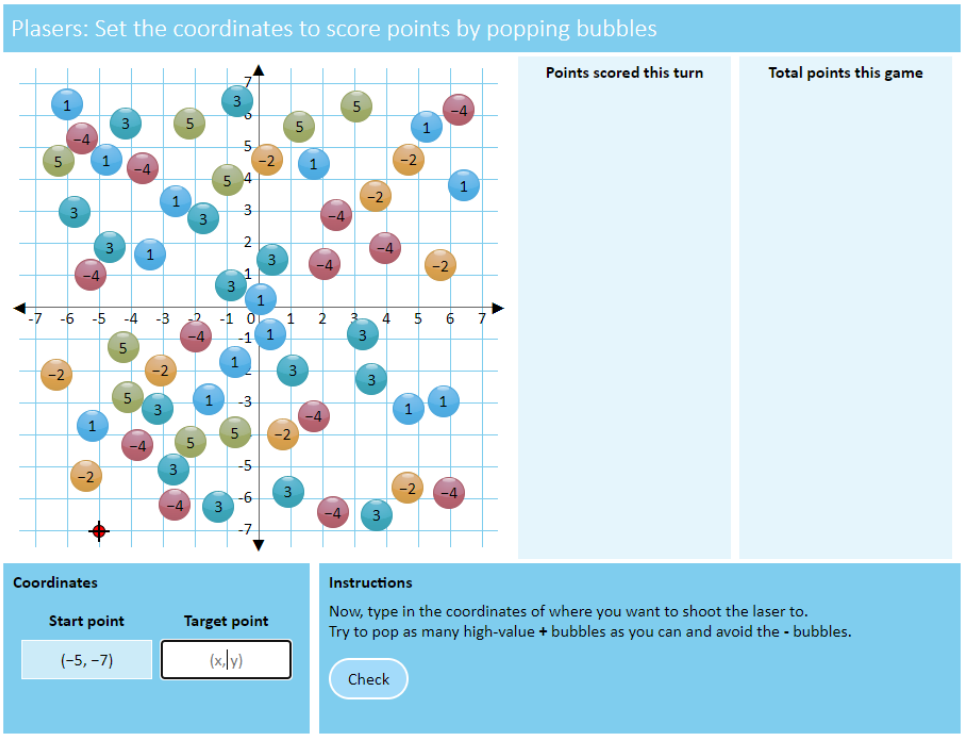
Essa classificação leva em consideração as características intrínsecas de um OA, suas formas de usos e seus objetivos para assim definir o tipo de OA.

4.1 EXEMPLOS DE OBJETOS DE APRENDIZAGEM OBJETOS DE APRENDIZAGEM E O ENSINO À DISTÂNCIA

Um exemplo de Objeto de Aprendizagem pode ser encontrado no repositório do Ministério da Educação da Nova Zelândia [39]. Este exemplo apresenta uma atividade interativa que visa fortalecer a compreensão dos alunos sobre coordenadas cartesianas, além de promover habilidades de resolução de problemas e raciocínio espacial.

[39] Disponível em: https://nzmaths.co.nz Acesso em: 09/05/2024

Figura 19 – Exemplo de OA usando coordenadas cartesianas



Fonte: Ministério da Educação da Nova Zelândia[40]

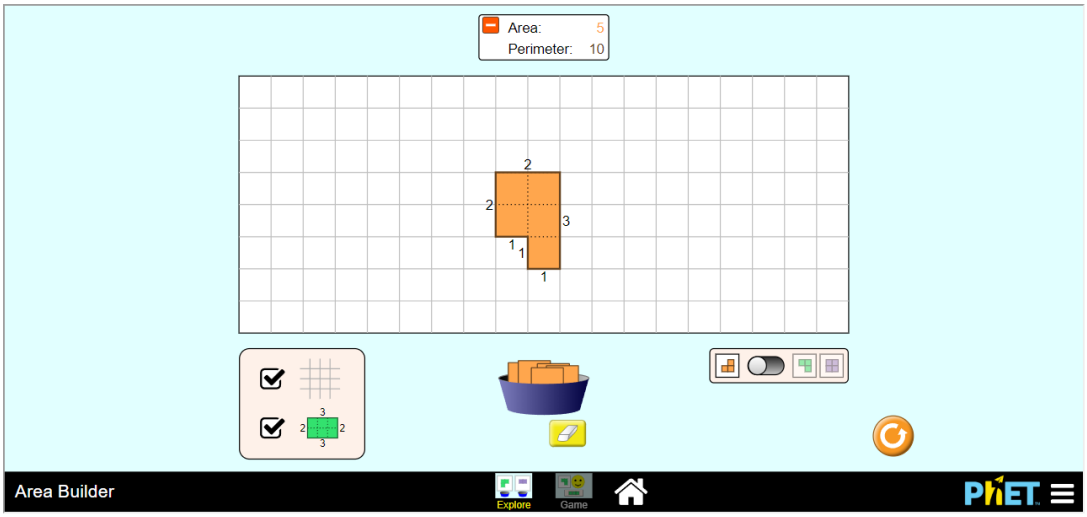
A atividade mostrada na Figura 19 consiste em uma interface na qual os alunos são desafiados a encontrar pontos em um plano cartesiano e, em seguida, selecionar um segundo ponto para traçar uma reta entre eles. Durante o percurso da reta, há bolinhas com pontos posicionadas estrategicamente. Cada vez que a reta atravessa uma dessas bolinhas, o aluno é recompensado com uma certa quantidade de pontos, incentivando assim a precisão e a compreensão do conceito.

Outro exemplo pode ser encontrado no repositório *LibreTexts Mathematics*[41], no exemplo da Figura 20, temos uma interface onde o aluno pode interagir para colocar quadrados dentro da tela para formar figuras.

40 Disponível em: https://nzmaths.co.nz/sites/default/files/static/LearningObjects/plasers/coordinates+-.html Acesso em: 08/05/2024

41 Disponível em: *https://math.libretexts.org* Acesso em: 09/05/2024

Figura 20 – Exemplo de OA usando áreas e perímetro



Fonte: LibreTexts Mathematics[42]

O ferramenta automaticamente já denota a área da figura contraída e seu perímetro, e, dessa forma, o aluno pode explorar diversas configurações de figuras e observar a sua mudança de perímetro de área.

Como mostrado anteriormente, podemos usar as classificações de Barbosa (2014) para identificarmos os tipos de OAs apresentados nos exemplos acima. Ambos os exemplos apresentados podem ser classificados como OA de tipo Prática, uma vez que eles podem ser enquadrados como jogos educacionais.

Estes exemplos ilustram como os OAs podem ser ferramentas enriquecedoras do processo educacional, fornecendo experiências de aprendizagem envolventes e eficazes. Ao utilizar atividades interativas e recursos dinâmicos, os educadores podem transformar conceitos abstratos em experiências tangíveis.

No contexto contemporâneo da educação, o ensino à distância (EaD) tem se tornado cada vez mais relevante, especialmente diante de desafios como a pandemia de COVID-19 e a necessidade de adaptação a novas modalidades de aprendizagem. Nesse cenário, os Objetos de Aprendizagem (OA) desempenham um papel fundamental, oferecendo uma série de benefícios que podem potencializar a eficácia da EaD.

> No Brasil, os cursos a distância utilizam cada vez mais o suporte da internet e os ambientes virtuais de aprendizagem, o que traz junto o desafio da estruturação de materiais didáticos adequados para apoiar as ações pedagógicas nesses cursos (CARNEIRO; SILVEIRA, 2014).

[42] Disponível em: https://math.libretexts.org/Learning_Objects/PhET_Simulations/PhET%3A_Area_Builder Acesso em: 08/05/2024

No ambiente da EaD, onde os estudantes podem estar geograficamente dispersos e ainda enfrentarem diferentes desafios de acesso à tecnologia, a capacidade de adaptar os recursos educacionais é fundamental para garantir uma experiência de aprendizagem inclusiva e eficaz.

Um dos principais benefícios dos OA é sua acessibilidade, uma vez que transformam o ensino intragável em algo tangível.

> É claro que as tecnologias da informação e comunicação, como já foi mencionado anteriormente, estão transformando o ensino de bem intangível, como a "fala" das aulas expositivas, em objetos tangíveis, como sites da web e vídeos, entre outros (MENDES; SOUZA; CAREGNATO, 2004).

4.2 OBJETOS DE APRENDIZAGEM MANIM

O *Manim*, do ponto de vista computacional, é uma biblioteca de *Python* que permite gerar animações de gráficos, objetos matemáticos, fórmulas matemáticas, dentre outros. Dessa forma, é possível que o *Manim* seja uma ferramenta que possui um grande potencial na criação de OAs para o ensino da Matemática. A proposta aqui é que o responsável por ensinar algo dentro da area de matemática possa compreender e utilizar OAs para o ensino de Cálculo Diferencial e Integral, de forma que se concretiza a instrumentalização de suas aulas.

Ao trabalhar com *Manim* é importante observar que os OAs gerados pela ferramenta são do tipo Apresentação, segundo a classificação de Barbosa (2014), ou seja, os arquivos gerados pelo Manim, por padrão, são arquivos de vídeo, porém, podemos utilizar bibliotecas adicionais para mudar esse comportamento padrão, como por exemplo *Manim Slides*[43] e o *Manim Presentation*[44]. Estas são bibliotecas desenvolvidas pela comunidade, com o objetivo de fornecer aos educadores uma ferramenta de criação de slides com as animações geradas pelo *Manim*. Assim, ampliando ainda mais a adaptabilidade dos OAs produzidos pela ferramenta, e mantendo seu tipo de OA dentro da classificação de Barbosa (2014), que nesse caso são OAs do tipo Apresentação.

Objetos de Aprendizagem criados usando Manim possuem uma série de vantagens sobre outros OAs. A hipótese deste trabalho é de que, uma vez que o *Manim* pode apoiar o ensino de matemática e, ao mesmo tempo, é uma ferramenta computacional extremamente eficiente na animação Matemática, então é possível adaptá-lo para um OA Matemático. Sendo assim, estaremos denominando, de agora em diante, um OA Matemático desenvolvido no *Mamin* como Objeto de Aprendizagem *Manim* (OAM). Nesse contexto, determinou-se neste trabalho que as principais características de um OAM são:

- **Flexibilidade e Personalização**: *Manim* oferece uma ampla gama de funcionalidades e personalizações. Educadores podem adaptar os OAMs para diferentes níveis de ensino e estilos de aprendizagem, criando materiais didáticos que atendem às necessidades

[43] https://www.manim.community/plugin/manim-slides/
[44] https://github.com/galatolofederico/manim-presentation

específicas de seus alunos. A flexibilidade da biblioteca permite aos OAs contraídos nela uma granularidade que nao é facilmente encontrada em outras ferramentas.

- **Integração nativa com LaTeX**: Uma das grandes vantagens do *Manim* é a integração com LaTeX, permitindo a inserção de equações matemáticas formatadas de forma profissional. Isso é particularmente útil em disciplinas como matemática, física e engenharia, onde a apresentação correta de fórmulas e notações é crucial.

- **Reprodutibilidade e Compartilhamento**: O código usado para criar as animações em *Manim* pode ser facilmente compartilhado e reproduzido. Isso permite que educadores colaborem e troquem materiais, além de possibilitar a revisão e atualização contínua dos OAMs.

- **Recursos *Open Source***: Sendo uma ferramenta de código aberto, *Manim* é acessível a qualquer educador ou instituição sem custo. A comunidade de usuários é ativa e colaborativa, oferecendo suporte, tutoriais e recursos adicionais que facilitam e estendem o uso da biblioteca.

Seguindo a classificação de Mendes (2004), analisa-se os OAMs deste trabalho da seguinte forma:

- Reusabilidade: pode ser utilizado em diversos contextos educacionais, desde aulas introdutórias de cálculo até cursos mais avançados. A natureza modular da animação permite que ela seja integrada em diferentes partes de um currículo de matemática.

- Adaptabilidade: é adaptável, muito embora sua temática esteja alocada dentro de conteúdos de cálculo, dessa forma, ela é adaptável em contextos onde o assunto a ser abordado é compatível com o do OAM.

- Granularidade: possui uma granularidade alta. Isso advém de uma característica do *Manim*, uma vez que os objetos são feitos de forma programática a granularidade do OAM é a maior possível.

- Acessibilidade: essa é uma característica que varia de acordo com as escolhas de compartilhamento do autor. Para esse objeto tanto versões em vídeo quanto código estão disponíveis em plataformas online.

- Durabilidade: como um recurso digital, este OAM tem uma alta durabilidade. Pode ser atualizado e revisado conforme necessário sem perda de qualidade, e a tecnologia subjacente ao *Manim* (nesse caso os formatos de video mp4) assegura que o conteúdo permaneça relevante e funcional ao longo do tempo.

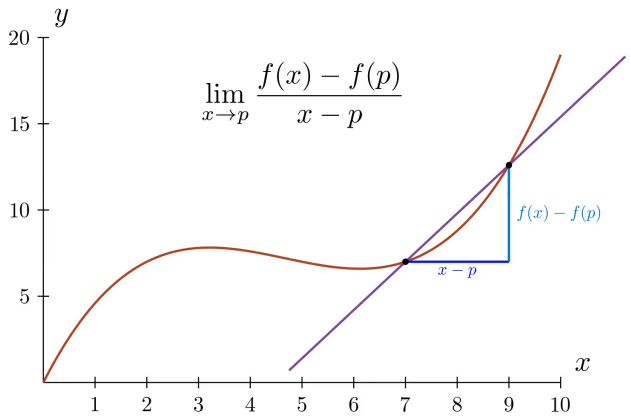
– Interoperabilidade: a animação pode ser exportada em formatos de vídeo amplamente suportados, permitindo sua integração em diversas plataformas educacionais e sistemas de gerenciamento de aprendizagem (LMS). Além disso, o código-fonte pode ser reutilizado em diferentes ambientes de programação e sistemas operacionais.

Ao adotar o OAM, estamos ampliando as possibilidades pedagógicas e contribuindo para um ambiente de ensino onde o professor é o criador dos materiais aos quais necessita. Mais abaixo vamos analisar alguns exemplos de OAMs.

4.3 OAM-1: DERIVADA E RETA TANGENTE

O OAM-1[45] aborda a ideia geométrica de derivadas através do conceito de limite visualizado graficamente pela reta tangente a uma curva. O objeto mostra, de maneira visual e dinâmica, como se define a derivada de uma função em um ponto específico utilizando o conceito de limite, conforme descrito por Guidorizzi (2013, V. 1, p. 214). A animação inicia com a representação de uma função contínua $f(x)$ e um ponto p na curva. Em seguida, demonstra o processo de aproximação de um ponto x a p, ilustrando como a secante que passa pelos pontos $(x, f(x))$ e $(p, f(p))$ gradualmente se transforma na reta tangente à curva no ponto p.

Figura 21 – Tela intermediaria da animação do OAM-1



Fonte: Elaborada pelo autor (2024), Disponível no link[46]

[45] O código completo para o OAM está presente nos Apêndices A , e também pode ser encontrados no repositório do GitHub. Disponível em: https://github.com/Kallel181/Manim-Calculo/tree/main Acesso em: 15/05/2024. Uma versão em video da animação pode ser visualizada em: https://drive.google.com/file/d/1zPgo6pB7528bcW9dcPgktRmCv-Sl8bjm/view?usp=drive_link Acesso em: 15/05/2024.

[46] https://drive.google.com/file/d/1zPgo6pB7528bcW9dcPgktRmCv-Sl8bjm/view?usp=drive_link

Conforme x se aproxima de p, o OAM destaca o comportamento do quociente incremental $\frac{f(x)-f(p)}{x-p}$ e como ele tende ao valor da derivada $f'(p)$. Esse processo é mostrado graficamente para ajudar os alunos a visualizar o conceito abstrato de limite e a compreensão intuitiva da derivada como a inclinação da reta tangente no ponto p.

4.3.1 Uso educacional

O OAM-1 pode ser utilizado em disciplinas de Cálculo no que se refere à introdução do conceito de derivada e da reta tangente. Ele oferece uma representação visual dinâmica que pode aumentar o engajamento dos alunos e melhorar a retenção do conteúdo.

4.3.2 Sugestão de uso

Pode-se utilizar o OAM-1 para mostrar a definição de derivada como o limite de um quociente. A animação pode ser exibida para a turma enquanto uma expelição dos conceitos é feita de forma simultânea, facilitando a visualização do processo de aproximação da reta tangente à curva. A animação é construída para mostrar geometricamente a reta tangente através do limite quociente:

$$f'(p) = \lim_{x \to p} \frac{f(x)-f(p)}{x-p}$$

Esse mesmo limite mais a frente é mostrador em Guidorizzi (2013, V. 1, p. 216) como a derivada de uma função no ponto p.

Sugere-se começar a aula apresentando a animação em um projetor, o OAM mostra o gráfico de uma função e a reta secante que se aproxima da reta tangente à medida que x se aproxima de um ponto p. Permite-se pausar a animação em momentos-chave para a explicação de cada passo da definição de derivada. Nesse caso, a animação mostra a reta secante com pontos x e p já pre-definidos, como podemos ver na Figura 21. O OAM mostra as diferenças $x-p$ e $f(x)-f(p)$,e, em seguida, os valores de x e p, e por fim, lentamente aproximando do valor de x para o valor de p.

Após a exibição, é possível pedir aos alunos que descrevam o que observaram, promovendo uma discussão sobre como a reta secante se transforma em tangente.

Utilizando a animação, e com algumas simples modificações, mostradas na Seção 4.3.3, é possível mostrar outros exemplos específicos de funções e seus pontos de tangência, variando os valores de x e p.

4.3.3 Reuso do OAM-1

No que se refere às possibilidades de mudanças no código-fonte do *Manim*, no OAM-1, é possível mudar livremente a função a ser desenhada e os pontos a serem evidenciados em x e p. Para essas modificações, é necessário primeiro mudar a função definida na linha 30.

Algoritmo 7: OAM 1 - Mudança de função

```
28  <RECORTE>
29          #Função para desenharmos o grafico
30          def f(x):
31              return 0.1 * (x - 2) * (x - 5) * (x - 7) + 7
32  <RECORTE>
```

Depois disso, mudar os eixos da linha 15 para acomodar a nova função de forma satisfatória, dessa forma, precisa-se ajustar os parâmetros `x_range`, `x_length`, `y_range` e `y_length`.

Algoritmo 8: OAM 1 - Ajuste dos eixos

```
14  <RECORTE>
15          axes = (
16              Axes(
17                  x_range = [0,10,1],
18                  x_length = 9,
19                  y_range = [0,20,5],
20                  y_length = 6,
21                  axis_config = {"include_numbers": True,
22                                 "include_tip":False},
23              )
24              .set_color(WHITE)
25          )
26  <RECORTE>
```

E por ultimo, alterar os valores do `ValueTracker` de `p` e `dx`, presentes nas linhas 49 e 50, respectivamente.

Algoritmo 9: OAM 1 - Ajuste de x e p

```
48  <RECORTE>
49          p = ValueTracker(7)
50          dx = ValueTracker(2)
51  <RECORTE>
```

Alterando essas trés pequenas partes do código teremos como resultado um OAM que funciona da mesma maneira que o original, porem, com uma função diferente.

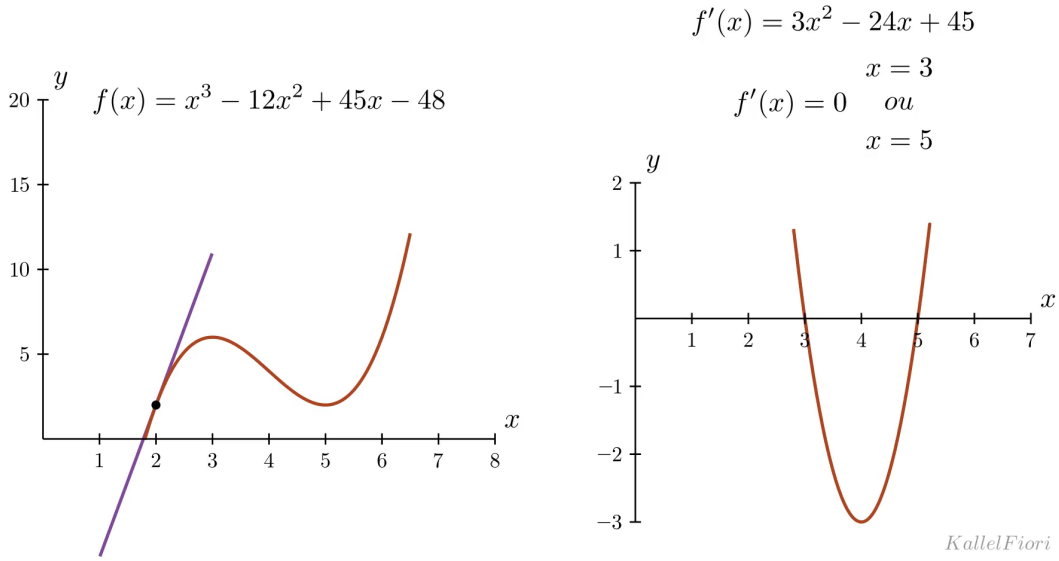
4.4 OAM-2: PONTOS CRITICOS

O OAM-2[47] aborda a conceito de ponto critico ou ponto estacionário de uma função f. O objeto mostra, de maneira visual e dinâmica, o comportamento da reta tangente, quando aproxi-

[47] O código completo para o OAM está presente nos Apêndices A , e também pode ser encontrados no repositório do GitHub. Disponível em: https://github.com/Kallel181/Manim-Calculo/tree/main Acesso em: 15/05/2024. Uma versão em video da animação pode ser visualizada em: https://drive.google.com/file/d/1bA6pokqQDyzjMoHHZwWsXx2Toaqr5MmV/view?usp=drive_link Acesso em: 30/05/2024.

mamos a valores onde temos pontos criticos da função. A animação inicia com a apresentação de uma função, no caso, temos $f : \mathbb{R} \to \mathbb{R}$ definida como $f(x) = x^3 - 12x^2 + 45x - 48$.

Figura 22 – Tela intermediaria da animação do OAM-2



Fonte: Elaborada pelo autor (2024), Disponível no link[48]

Em seguida, ao lado, como mostrado na Figura 25, apresentamos a derivada da função f, nesse caso temos $f'(x) = 3x^2 - 24x + 45$, com isso movemos a reta rangente de forma dinâmica até os pontos criticos, onde observamos o comportamento da reta tangente, nesse caso, vemos que a reta tangente estará na horizontal.

4.4.1 Uso educacional

O OAM-2 pode ser utilizado em disciplinas de Cálculo no que se refere ao conceito de derivada e da reta tangente, nesse caso, mais especificamente em uma aula ponde irá se abordar pontos criticos. Ele oferece uma representação visual dinâmica que pode aumentar o engajamento dos alunos e melhorar a retenção do conteúdo.

4.4.2 Sugestão de uso

SUgere-se utilizar o OAM-2 para mostrar o comportamento da reta tangente ao se aproximar dos pontos criticos, é possível iniciar a aula mostrando a definição apresentada por

[48] https://drive.google.com/file/d/1bA6pokqQDyzjMoHHZwWsXx2Toaqr5MmV/view?usp=drive_link

Guidorizzi (2013, V. 1, p. 417), um ponto $p \in D_f$ se diz ponto critico ou ponto estacionário de f se $f'(p) = 0$.

Em seguida professor pode apresentar a animação em um projetor, que mostra o gráfico de uma função e a reta tangente. Como a animação detalha os pontos que estamos analisando o comportamento da derivada é possível também usar este OAM como uma motivação para introduzir o conceitos de máximos e mínimos, uma vez que, na função já pre-definida, temos evidenciamos de forma clara um máximo local e um mínimo local.

Após a exibição, pode-se pedir aos alunos que descrevam o que observaram, promovendo uma discussão sobre o comportamento da reta tangente e da ideia de máximos e mínimos.

Utilizando a animação, e com algumas simples modificações, mostradas na Seção 4.4.3, o professor pode mostrar exemplos diferentes de funções, por exemplo, podemos mostrar funções que possuem pontos de inflexão.

4.4.3 Reuso do OAM-2

No que se refere às possibilidades de mudanças no código-fonte do Manim, no OAM-2, o OAM permite mudar livremente a função a ser desenhada, dessa forma, mostrando o comportamento da reta tangente em uma função diferente. Para essa modificação, é necessário mudar a função e de sua derivada definidas nas linhas 15 e 18.

Algoritmo 10: OAM 2 - Ajuste das funções LaTeX

```
12  <RECORTE>
13          #Objeto Manim que armazena o texto Latex da
14          #função as ser desenhada
15          function_tex = MathTex(r'f(x)=x^3-12x^2+45x-48')
16
17          #Objeto Manim que armazena o texto Latex da derivada da
18          #função as ser desenhada
19          derivative_tex = MathTex('f\'(x)=3x^2-24x+45')
20  <RECORTE>
```

Também é necessário alterar as funções que vão ser usadas pelo *Python* nas linhas 37 e 41. Para que as funções a serem plotadas mudem de acordo nos objetos `graph1` e `graph2`.

Algoritmo 11: OAM 2 - Ajuste das funções *Python*

```
35  <RECORTE>
36          #Função para desenharmos o grafico
37          def f(x):
38              return (x**3 - (12*(x**2)) + 45*x) - 48
39
40          #Função para desenharmos o grafico da função derivada
41          def derivative(x):
42              return ((3*x**2)-(24*x)+45)
43  <RECORTE>
```

Em seguida, mudamos os eixos `axes1`, na linha 22, e `axes2`, na linha 92, para comodar as gráficos da função e de sua derivada respectivamente.

Precisamos também mudar o texto LaTeXdos valores de x onde $f'(x) = 0$, na linha 168.

Algoritmo 12: OAM 2 - Valores das raízes LaTeX

```
167  <RECORTE>
168          solution_tex1 = MathTex('f\'(x)=0').scale(0.7)
169          solution_tex2 = MathTex('x = 3').scale(0.7)
170          solution_tex3 = MathTex('ou').scale(0.7)
171          solution_tex4 = MathTex('x = 5').scale(0.7)
172
173          solution_tex3.next_to(solution_tex2,DOWN)
174          solution_tex4.next_to(solution_tex3,DOWN)
175
176          solution_tex_x_values = VGroup(solution_tex2,
177                                         solution_tex3,
178                                         solution_tex4)
179          solution_tex_x_values.next_to(solution_tex1,RIGHT)
180
181          solution_tex = VGroup(solution_tex_x_values,solution_tex1)
182
183          solution_tex.next_to(derivative_tex,DOWN)
184          self.play(Write(solution_tex))
185  <RECORTE>
```

Em especial para esse caso, podemos acionar mais elementos caso a derivada da função possua mais de duas raízes, para isso basta adicionar um novo elemento `solution_texX` e posicionar ele de acordo.

Por ultimo também precisamos mudar e adicionar, caso necessário, algumas animações onde mudamos o valor do `ValueTracker` que armazena o valor de x, nas linhas 219 e 223.

Algoritmo 13: OAM 2 - Mudança dos valores do método *animate*

```
218  <RECORTE>
219          self.play(x.animate.set_value(3),run_time=2)
220          self.wait(WAIT_TIME)
221          self.pause()
222
223          self.play(x.animate.set_value(5),run_time=2)
224          self.wait(WAIT_TIME)
225          self.pause()
226  <RECORTE>
```

Alterando essas partes do código teremos como resultado um OAM que funciona da mesma maneira que o original, porem, com uma função diferente e evidenciando pontos criticos diferentes.

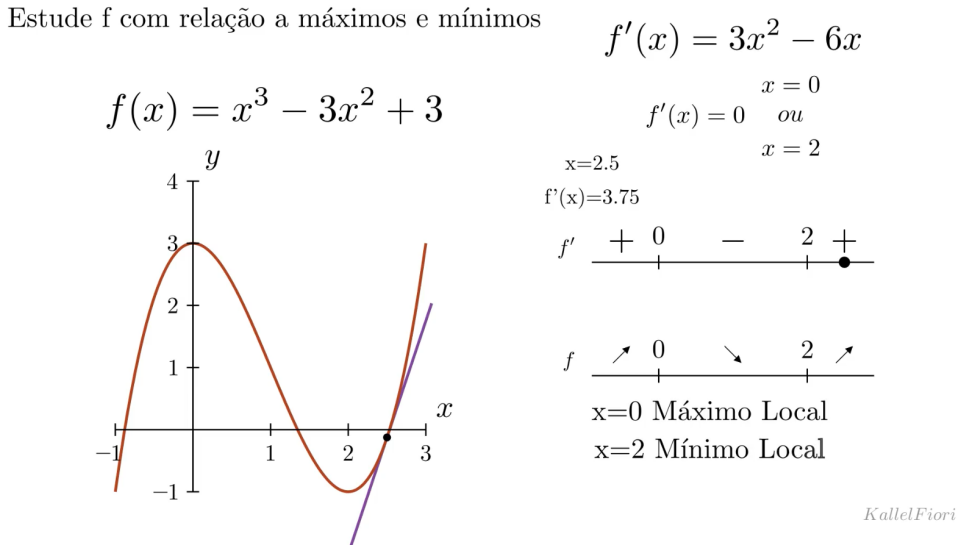
4.5 OAM-3 MÁXIMOS E MÍNIMOS

O OAM-3[49] aborda o conceito de máximos e mínimos de uma função f. O objeto mostra, de maneira visual e dinâmica, a analise dos pontos criticos de uma função, nesse caso, analisamos

[49] O código completo para o OAM está presente nos Apêndices A , e também pode ser encontrados no repositório do GitHub. Disponível em: https://github.com/Kallel181/Manim-Calculo/tree/main Acesso em: 15/05/2024. Uma versão em video da animação pode ser visualizada em: https://drive.google.com/file/d/1vbcFGaYa51HLnDAtArgyLZV5xQRx4tht/view?usp=drive_link Acesso em: 15/05/2024.

os pontos criticos da função $f : \mathbb{R} \to \mathbb{R}$ definida como $f(x) = x^3 - 3x^2 + 3$.

Figura 23 – Tela intermediaria da animação do OAM-3



Fonte: Elaborada pelo autor (2024), Disponível no link[50]

A animação inicia com a apresentação da função f para em seguida mostrar sua derivada f', usando dois segmentos de reta mais a baixo analisamos os intervalos da função entre os pontos criticos, onde os valores da derivada são positivos ou negativos. Com essa analise podemos classificas os pontos criticos da função em máximo local, mínimo local, ou ponto de inflexão.

4.5.1 Uso educacional

O OAM-2 pode ser utilizado em disciplinas de Cálculo no que se refere ao conceito de derivada e da reta tangente, nesse caso, mais especificamente em uma aula ponde irá se abordar classificação de pontos criticos, isto é, máximo local, mínimo local e pontos de inflexão. Ele oferece uma representação visual dinâmica que pode aumentar o engajamento dos alunos e melhorar a retenção do conteúdo.

4.5.2 Sugestão de uso

Sugere-se utilizar do OAM-3 para mostrar uma analise dinâmica dos pontos criticos de uma função, usando de uma notação semelhante a apresentada por Guidorizzi (2013, V. 1, p. 405). A partir de uma discussão anterior, como sugerido na Seção 4.4.2, é possivel continuar do OAM-2 a discussão de máximos e mínimos, agora analisando em mais detalhes com o presente

[50] https://drive.google.com/file/d/1vbcFGaYa51HLnDAtArgyLZV5xQRx4tht/view?usp=drive_link

OAM. Durante uma explicação, a cada intervalo analisado a partir dos pontos criticos, é possível interagir com a turma de forma a fazer perguntas direcionadoras para o conteúdo, por exemplo, *"Os valores da derivada nesse intervalo serão positivos ou negativos?"* ou *"Sabendo que os valores da derivada são positivos, isso significa que a função é crescente ou decrescente nesse intervalo?"*. Dessa forma, a animação vai se construindo com uma interação direta com os alunos.

Utilizando a animação, e com algumas simples modificações, mostradas na Seção 4.5.3, pode-se mostrar exemplos diferentes de funções, analisando intervalos diferentes e comportamentos diferentes.

4.5.3 Reuso do OAM-3

No que se refere às possibilidades de mudanças no código-fonte do *Manim*, no OAM-3, é possível mudar livremente a função a ser desenhada, as retas numéricas mostrando o sinal da derivada em cada intervalo e reta que indica se a função é crescente ou decrescente no intervalo. Para essas modificações, assim como nos casos anteriores, precisamos mudar os objetos que contem os textos LaTeXda função e da sua derivada, linha 17 e 115 respectivamente.

Algoritmo 14: OAM 3 - Mudança texto LaTeXda função

```
16  <RECORTE>
17          function_tex = MathTex(r'f(x)=x^3-3x^2+3')
18  <RECORTE>
```

Depois disso precisamos mudar as função na linha 38 que será usada para desenhar o gráfico da função, depois alteramos o parâmetro `x_range` do objeto `graph1`, na linha 42, para acomodar a nova função.

Algoritmo 15: OAM 3 - Mudança na definição da função e gráfico

```
36  <RECORTE>
37  #Função para desenharmos o grafico
38          def f(x):
39              return (x**3 - (3*(x**2)) + 3)
40
41
42          graph1 = axes1.plot(
43              f, x_range=[-1,3], color=BLUE
44          )
45  <RECORTE>
```

Além disso, é importante ajustar os eixos no objeto `axes1` presente na linha 22 para acomodar o novo gráfico.

Algoritmo 16: OAM 3 - Mudança texto LaTeXda função

```
21  <RECORTE>
22          axes1 = (
23              Axes(
24                  x_range = [-1,3,1],
25                  x_length = 5,
26                  y_range = [-1,4,1],
27                  y_length = 5,
28                  axis_config = {"include_numbers": True,
29                                 "include_tip":False}
30              )
31              .set_color(WHITE)
32          )
33  <RECORTE>
```

O valor do objeto `x` presente na linha 49 que contem o `ValueTracker` pode também ser alterado conforme necessidade, uma vez que esse objeto marca a posição inicial da reta tangente.

Algoritmo 17: OAM 3 - Mudança ValueTracker

```
48  <RECORTE>
49  x = ValueTracker(-0.5)
50  <RECORTE>
```

A função que retorna os valores da derivada, na linha 154, também precisa ser alterada para que os valores sejam mostrados corretamente.

Algoritmo 18: OAM 3 - Mudança função derivada

```
153 <RECORTE>
154         def derivative(x):
155             return (3*(x**2) - 6*x)
156 <RECORTE>
```

Apos isso é necessário alterar os objetos `LineNumber` para que eles marquem os novos intervalos que precisamos analisar, para esse OAM temos dois objetos Luncumbe idênticos, nas linhas 143 e 181 ambos precisa ter as mesmas alterações, nesse caso, iremos mudar os parâmetros `x_range`, `numbers_to_include` e `length` se necessário.

Algoritmo 19: OAM 3 - Mudança `NumberLine`

```
142 <RECORTE>
143         derivative_number_line = NumberLine(
144             x_range =[-0.9,2.9,2],
145             length = 4,
146             color = WHITE,
147             numbers_to_include = {0,2},
148             label_direction=UP
149         )
150 <RECORTE>
```

Mais a frente, a penúltima alteração que precisamos fazer é a animação da mudança dos valores de `x`, feitas com o método `animate` e os objetos que indicam se a função é crescente ou decrescente e o objeto que indica se o valor da derivada é positiva ou negativa no intervalo.

Algoritmo 20: OAM 3 - Mudança na animação dos valores de x

```
200 <RECORTE>
201         self.play(x.animate.set_value(0),run_time=2)
202         self.wait(0.1)
203         self.pause()
204 
205         plus1 = MathTex("+").move_to(derivative_number_line.n2p(-0.5))
206         plus1.shift(UP*0.3)
207         self.play(Write(plus1))
208         self.wait(0.1)
209         self.pause()
210 
211         up_arrow1 = Vector(UR*0.3).scale(0.8)
212         up_arrow1.move_to(function_number_line.n2p(-0.5))
213         up_arrow1.shift(UP*0.3)
214         self.play(Write(up_arrow1))
215         self.wait(0.1)
216         self.pause()
217 <RECORTE>
```

Nessa seção de código temos 3 partes muito semelhante a mostrada acima, isso porque temos que fazer a analise em 3 intervalos, dessa forma, caso a nova derivada tenha mais raízes, ou seja, mais pontos criticos, temos por consequência mais intervalos para serem analisados, dessa forma, essa mesma estrutura pode ser copiada para mostrar a analise desses novos intervalos. Tudo que precisamos alterar é o valor do objeto `x`, na linha 201, e alterar os objetos `plus1` e `up_arrow`, nas linhas 205 e 211, respectivamente, de forma que eles correspondam com o comportamento da derivada e função escolhida.

A ultima parte que precisamos alterar são os textos referentes aos pontos criticos e suas classificações, na linha 263.

Algoritmo 21: OAM 3 - Mudança na animação dos valores de x

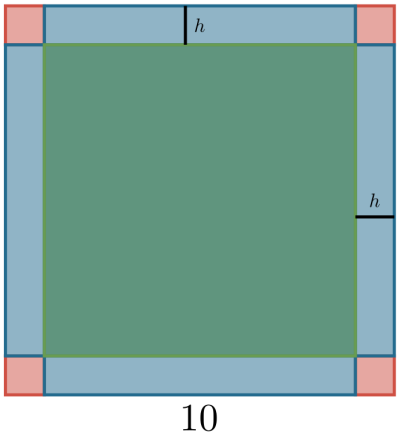
```
265 <RECORTE>
266         res1 = MathTex("x=0 Máximo Local",tex_environment="flushleft")
267         res1.scale(0.8)
268         res1.next_to(right_elements,DOWN)
269 <RECORTE>
```

4.6 OAM-4: PROBLEMA DE VOLUME MÁXIMO DE UMA CAIXA

O OAM-4[51] aborda um exercício como descrito a seguir. Uma caixa sem tampa será fabricada a partir de um pedaço de papelão de 10m por 10m cortada como aparece na figura abaixo.

[51] O código completo para o OAM está presente nos Apêndices A , e também pode ser encontrados no repositório do GitHub. Disponível em: https://github.com/Kallel181/Manim-Calculo/tree/main Acesso em: 15/05/2024. Uma versão em video da animação pode ser visualizada em: https://drive.google.com/file/d/1VLhpWRIkVe4CGY43uVPbyINwnyAIQUQE/view?usp=drive_link Acesso em: 15/05/2024.

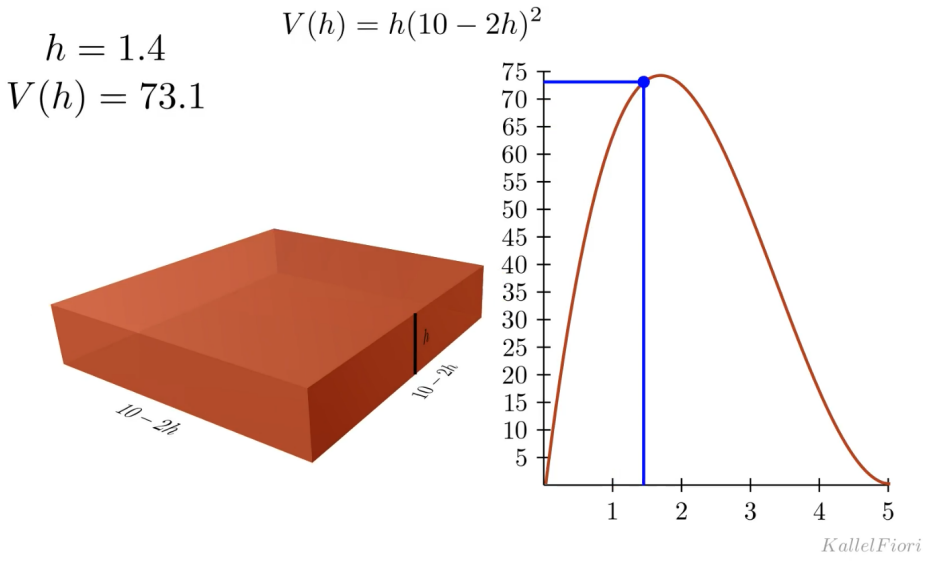
Figura 24 – Problema de volume máximo de uma caixa



Fonte: Elaborada pelo autor (2024)

Removendo as partes em vermelho e dobrando as partes em azul, como mostrado na Figura 24, deseja-se saber, quais as dimensões que produzirão uma caixa com o volume máximo?

Figura 25 – Tela intermediaria do OAM-4



Fonte: Elaborada pelo autor (2024), Disponível no link[52]

O objeto mostra em três partes, como a variação de h muda a forma com que os cortes precisam ser feitos na folha de papelão, como a variação de h muda o formato final da caixa e

[52] https://drive.google.com/file/d/1VLhpWRIkVe4CGY43uVPbyINwnyAIQUQE/view?usp=drive_link

por ultimo como a variação de h muda o volume da caixa. Tudo isso de forma dinâmica, ou seja, sempre que algum objeto dentro do OAM tenha uma dependência do valor de h, ele será alterado de acordo para representar a mudança.

4.6.1 Uso educacional

O OAM-2 pode ser utilizado em disciplinas de Cálculo no que se refere ao conceito de derivada, nesse caso, aplicando um exercício de volume máximo. Este OAM proporciona uma representação visual dinâmica que pode aumentar o engajamento dos alunos e melhorar a retenção do conteúdo. A visualização das mudanças dependentes do valor de h pode permitir que os alunos compreendam de forma mais intuitiva o impacto das variações de parâmetros em problemas de otimização.

4.6.2 Sugestão de uso

Sugere-se iniciar a aula apresentando o exercício, seja em uma folha de papel, ou seja através do próprio OAM. Em seguida, antes de iniciar uma resolução, pode-se apresentar a animação em um projetor, o OAM flui de forma a mostrar como construir a caixa variando h e como as dimensões e volume da caixa variam conforme variamos h, ou seja, o OAM não apresentar uma resolução do problema, e sim uma explicação que pode ajudar a ligar o exercício com os conteúdos anteriormente vistos.

Em um primeiro momento a animação mostra apenas a folha de papelão, e como os cortes acontecem se variamos h. Apos essa parte, a animação mostra a caixa de outro angulo, mostrando como a folha de papelão que vimos anteriormente se transforma na caixa. Por ultimo a animação mostra a caixa ao lado do gráfico da função $V : \mathbb{R} \to \mathbb{R}$ definida como $V(h) = h(10-2h)^2$, dessa forma podemos ver como a variação de h altera o formata do caixa ao mesmo tempo que podemos ver como isso altera o volume final da caixa. Pode-se avançar a animação por essas partes enquanto as mudanças são narradas.

Para uma conclusão do exercício é possível alterar o OAM-2 e OAM-3, seguindo as modificações propostas nas seções 4.4.3 e 4.5.3 respectivamente, esses OAMs se encaixam na temática do exercício, e as modificações propostas cobrem exatamente as modificações que precisam ser feitas para adaptarmos os OAMs ao exercício proposto.

4.6.3 Reuso do OAM-4

Antes de comentamos sobre as modificações que podem ser feitas no OAM-4, precisamos entender como o OAM-4 funciona. diferentemente dos outros OAMs, o OAM-4 faz um uso intensivo do método `always_update`, ou seja, existe uma alta interdependência entre os elementos dentro do OAM, nesse caso, boa parte dos elementos são alterado e redesenhados de

acordo com variações feitas no objeto `ValueTracker` `h` presente na linha 15.

Algoritmo 22: OAM 4 - ValueTracker h

```
14  <RECORTE>
15          h = ValueTracker(1)
16  <RECORTE>
```

Dessa forma, alterações apenas no valor do objeto `ValueTracker`, não só na sua definição da linha 15, mas também alterações nos métodos `animate`, irão alterar toda a animação.

Como o OAM apresenta uma visualização para um exercício, uma exemplo de modificação que pode ser feita é nos valores inicias do problema, para isso, precisamos alterar as condições inicias das varáveis `h` e `sheet_value`, presentes nas linhas 15 e 16.

Algoritmo 23: OAM 4 - Varaveis h e sheet_value

```
14  <RECORTE>
15          h = ValueTracker(1)
16          sheet_value = 10
17  <RECORTE>
```

Depois disso, precisamos alterar todos os objetos `MathTex` para que eles apresentem textos condizentes com as novas variáveis do exercício, nesse caso, precisamos alterar os objetos `side_label` na linha 26, `side1_label` e `side2_label` nas linhas 213 e 218 e `volume_func` na linha 236. Com isso, também precisamos alterar o gráfico mostrando na ultima parte do OAM, assomo como nas outras alterações, precisamos alterar o objeto `Axes`, presente na linha 244, para acomodar o novo gráfico e alterar a função presente na linha 258 para os novos valores do exercício.

Algoritmo 24: OAM 4 - Função *Python* do volume do caixa

```
14  <RECORTE>
15          def vol(x):
16              return x * (10-2*x)**2
17  <RECORTE>
```

Por últimos, de acordo com as modificações feitas, o posicionamento da câmera precisa ser alterado de acordo com a nova caixa feita com os novos valores, para isso basta usar o método `self.move_camera()`. O posicionamento da câmera em cenas que usam o ambiente `ThreeDScene` usa coordenadas esféricas, com isso, com as modificações feitas, novos valores para o posicionamento da câmera precisam ser encontrados para encaixar todos os objetos na tela.

5 CONSIDERAÇÕES FINAIS

Este trabalho de conclusão de curso investigou o uso da biblioteca *Manim* no ensino de cálculo em uma variável real, focando na criação de animações que auxiliem a compreensão de conceitos matemáticos complexos através de visualizações dinâmicas e interativas.

Nas análises realizadas, o *Manim* destacou-se a como uma ferramenta eficaz no contexto educacional, especialmente na criação de Objetos de Aprendizagem (OA), a biblioteca se mostrou versátil durante as criação de todos os OAMs mostrados no trabalho, fornecendo funcionalidades que abarcavam perfeitamente as necessidades de cada um dos OAMs. Além disso, os OAMs desenvolvidos demonstraram um alto nível de granularidade e capacidade de reuso, sendo adaptáveis para diferentes perspectivas e necessidades educacionais. Os seguintes OAMs foram construídos:

- OAM-1: Aborda a ideia geométrica de derivadas através do conceito de limite visualizado graficamente pela reta tangente a uma curva. A animação mostra como se define a derivada de uma função em um ponto específico utilizando o conceito de limite.

- OAM-2: Foca no conceito de ponto crítico ou ponto estacionário de uma função. A animação demonstra o comportamento da reta tangente em pontos críticos, mostrando a reta tangente horizontal nesses pontos.

- OAM-3: Analisa os máximos e mínimos de uma função, classificando os pontos críticos como máximo local, mínimo local ou ponto de inflexão, com base na análise dos intervalos onde a derivada é positiva ou negativa.

- OAM-4: Resolve um problema de otimização, mostrando como variar as dimensões de uma caixa sem tampa feita a partir de um pedaço de papelão de 10m por 10m para obter o volume máximo. A animação ilustra as mudanças na forma e no volume da caixa de acordo com a variação da altura.

Os OAMs foram construídos seguindo uma sequência didática, onde o OAM-1 é complementado pelo OAM-2 e assim por diante. O OAM-4 utiliza os conceitos abordados nos outros três OAMs para resolver o problema de otimização.

Mesmo que o trabalho tenha focado na construção de OAMs dentro do cálculo em uma variável real, por estarmos trabalhando em um ambiente *Python*, isso permite que uma série de outras bibliotecas possam ser importadas, como pandas[53] ou numpy[54], dessa forma, os OAMs podem ser construídos para qualquer área dentro das ciências exatas.

Em resumo, a integração do *Manim* no ensino de cálculo mostrou-se promissora, permitindo a criação de materiais didáticos inovadores e dinâmicos. Os resultados apontam para

[53] Disponível em: https://pandas.pydata.org Acesso em: 01/07/2024
[54] Disponível em: https://numpy.org Acesso em: 01/07/2024

a significativa contribuição do *Manim* na criação materiais que podem contribuir para a compreensão de conceitos abstratos, promovendo um aprendizado mais intuitivo e envolvente. Esta análise espera contribuir para a melhoria do ensino e aprendizagem de matemática, fornecendo uma base sólida para futuras pesquisas e aplicações práticas na área.

REFERÊNCIAS

AGUIAR, Eliane Vigneron Barreto; FLÔRES, Maria Lucia Pozzatti. Objetos de Aprendizagem: Conceitos Basicos. In: [s.l.]: Evangraf Ltda, 2014. cap. 1. Citado na p. 44.

BARBOSA, Gisele. **MANUAL DO PROFESSOR PARA UTILIZAÇÃO DE OBJETOS DE APRENDIZAGEM**. [S.l.: s.n.], 2014. Citado nas pp. 45, 47, 48.

CARNEIRO, Mára Lúcia Fernandes; SILVEIRA, Milene Selbach. **Objetos de Aprendizagem como elementos facilitadores na Educação a Distância**. [S.l.: s.n.], 2014. Citado na p. 47.

DIEHL, Stephan. **Software visualization**. [S.l.]: 27th International Conference on Software Engineering, 2005. Disponível em: <https://www.researchgate.net/publication/221555679_Software_visualization>. Citado nas pp. 11, 12.

ESTEVAM, Everton José Goldoni; KALINKE, Marco Aurélio. **Recursos Tecnológicos e Ensino de Estatística na Educação Básica: um cenário de pesquisas brasileiras**. [S.l.]: Revista Brasileira de Informática na Educação, 2013. Disponível em: <https://www.researchgate.net/profile/Everton-Estevam/publication/272703733_Recursos_Tecnologicos_e_Ensino_de_Estatistica_na_Educacao_Basica_um_cenario_de_pesquisas_brasileiras/links/5681425e08ae1975838f704c/Recursos-Tecnologicos-e-Ensino-de-Estatistica-na-Educacao-Basica-um-cenario-de-pesquisas-brasileiras.pdf>. Citado na p. 15.

GUIDORIZZI, H. L. **Um curso de cálculo**. São Paulo: LTC, 2013. Citado nas pp. 50, 51, 54, 56.

HOHENWARTER, Markus. **GeoGebra Ein Softwaresystem f¨ur dynamische Geometrie und Algebra der Ebene**. [S.l.]: Revistal Digital FAPAM, 2002. Disponível em: <https://docplayer.org/21662951-Geogebra-ein-softwaresystem-fuer-dynamische-geometrie-und-algebra-der-ebene.html>. Citado na p. 25.

HOHENWATER, Markus; LAVICZA, Zsolt. In: **Model-Centered Learning**. Edição: Robert Schoen e Lingguo Bu. [S.l.]: Sense Publishers, 2011. THE STRENGTH OF THE COMMUNITY: HOW GEOGEBRA CAN INSPIRE TECHNOLOGY INTEGRATION IN MATHEMATICS. Citado na p. 20.

MENDES, Rozi Mara; SOUZA, Vanessa Inácio; CAREGNATO, Sônia Elisa. **A propriedade intelectual na elaboração de objetos de aprendizagem**. [S.l.: s.n.], 2004. Disponível em: <https://lume.ufrgs.br/bitstream/handle/10183/548/000502901.pdf?sequence=1&isAllowed=y>. Citado nas pp. 44, 45, 48, 49.

PESENTE, Guilherme Moraes. **O ensino de matemática por meio da linguagem de programação Python**. [S.l.]: Dissertação (Mestrado em Ensino de Ciência e Tecnologia) - Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2019. Disponível em: <https://repositorio.utfpr.edu.br/jspui/handle/1/5020>. Citado na p. 17.

SANTOS, Alessandra Dendi dos. **Um Estudo Epistemológico da Visualização Matemática: o acesso ao conhecimento matemático no ensino por intermédio dos processos de visualização**. [S.l.]: UNIVERSIDADE FEDERAL DO PARANÁ, 2014. Disponível em: <http://www.exatas.ufpr.br/portal/ppgecm/wp-content/uploads/sites/27/2016/03/045_AlessandraHendidosSantos.pdf>. Citado na p. 15.

SILVEIRA MOURA, Daniela Alves da; SILVA DOS SANTOS, Alex da; SILVA, Jhonatan Júnio da. **TECNOLOGIA A FAVOR DA EDUCAÇÃO MATEMÁTICA: GEOGEBRA E SUAS APLICAÇÕES**. [S.l.]: Revistal Digital FAPAM, 2016. v. 7, p. 333–346. Disponível em: <https://periodicos.fapam.edu.br/index.php/synthesis/article/view/146>. Citado na p. 20.

APÊNDICE A – OBJETOS DE APRENDIZAGEM MANIM

Todos os códigos dos OAMs também podem ser encontrados no repositório do GitHub[55], e estão sobre a licença CC0[56].

Algoritmo 25: OAM 1 - Reta tangente

```
from manim import *
from manim_presentation import Slide

class tangente(Slide):
    def construct(self):

        WM = MathTex('KallelFiori').scale(0.5)
        WM.set_opacity(0.4)
        WM.move_to([6,-3.5,0])
        self.add(WM)

        #Objeto Manim que armazena os eixos cartesianos
        #Com alterações na função a ser mostrada é importante mudar o valores do graficos para que
        #a função seja melhor desenhada.
        axes = (
            Axes(
                x_range = [0,10,1],
                x_length = 9,
                y_range = [0,20,5],
                y_length = 6,
                axis_config = {"include_numbers": True, "include_tip":False},
            )
            .set_color(WHITE)
        )

        #Adicionando texto para indicar os eixos
        axes_labels = axes.get_axis_labels(x_label="x",y_label="y")

        #Função para desenharmos o grafico
        def f(x):
            return 0.1 * (x - 2) * (x - 5) * (x - 7) + 7

        #Objeto Manim que armazenada o texto Latex da definição de derivada (Guidorizzi)
        derivative = MathTex('f\'(p)=','\\lim_{x\\rightarrow p}\\frac{f(x)-f(p)}{x-p}')
        derivative_target = derivative.generate_target()
        derivative_target.shift(UP*2)

        #Objeto Manim que armazena o texto Latex apenas do limite da definição anterior
        limit = MathTex('\\lim_{x\\rightarrow p}\\frac{f(x)-f(p)}{x-p}').shift(UP*2)

        #Objeto Manim que armazena o grafico da função previamente definida
        graph = axes.plot(
            f, x_range=[0,10], color=BLUE
        )

        #Value Trackers armazenam os valores de p e dx, e podem ser usados para animar a reta secante
        #posteriormente.
        #Os valores iniciais podem ser alterados conforme necessidade.
        p = ValueTracker(7)
        dx = ValueTracker(2)
```

55 Disponível em: https://github.com/Kallel181/Manim-Calculo/tree/main Acesso em: 15/05/2024

56 Disponível em: https://creativecommons.org/public-domain/cc0/ Acesso em: 15/05/2024

```
51          #Objeto Manim que armazena a reta secante ao grafico conforme p e dx.
52          #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p e dx irão ser
53          #alterados posteriomente.
54          secant = always_redraw(
55              lambda: axes.get_secant_slope_group(
56                  x = p.get_value(),
57                  graph = graph,
58                  dx = dx.get_value(),
59                  dx_line_color = YELLOW,
60                  dy_line_color = ORANGE,
61                  dy_label = "f(x)-f(p)",
62                  dx_label = "x-p",
63                  secant_line_color = GREEN,
64                  secant_line_length = 8,
65              )
66          )
67
68          #Objeto Manim que armazena o ponto no grafico correspondente ao valor de p.
69          #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p ira ser
70          #alterado posteriomente.
71          dot1 = always_redraw(
72              lambda: Dot()
73              .scale(0.7)
74              .move_to(axes.c2p(p.get_value(), f(p.get_value())))
75          )
76
77          #Objeto Manim que armazena o ponto no grafico correspondente ao valor de p + dx.
78          #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p e dx irão ser
79          #alterados posteriomente.
80          dot2 = always_redraw(
81              lambda: Dot()
82              .scale(0.7)
83              .move_to(axes.c2p(
84                  p.get_value() + dx.get_value(),
85                  f(p.get_value() + dx.get_value())
86              ))
87          )
88
89          #Movendo os pontos para a camada mais acima, para que a reta secante e o grafico não sejam
90          #desenhados acima dos pontos
91          dot1.z_index = 1
92          dot2.z_index = 1
93
94          #Objeto Manim que muda de acordo com o valor de x+dx, indicando nosso x
95          #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p e dx irão ser
96          #alterados posteriomente.
97          x_text = always_redraw(
98              lambda: Tex(r'x='+str(p.get_value()+dx.get_value())[:3]).scale(0.6).shift(UP+2*LEFT)
99          )
100
101         #Objeto Manim que muda de acordo com o valor de x, indicando nosso p
102         #O objeto precisa estar dentro da função always_redraw uma vez que o valor de p ira ser
103         #alterado posteriomente.
104         p_text = always_redraw(
105             lambda: Tex(r'p='+str(p.get_value())[:3]).scale(0.6).next_to(x_text,DOWN)
106         )
107
108         self.play(Write(derivative))
109         self.wait(0.1)
110         self.pause()
111
112         self.play(MoveToTarget(derivative))
113         self.wait(0.1)
114         self.pause()
```

```
115         #Funções de animação
116         #Agora como todos os objetos já foram definidos, podemos continuar com a animação dos mesmos.
117         self.play(Write(VGroup(axes,axes_labels,graph)))
118         self.wait(0.1)
119         self.pause()
120 
121         self.play(TransformMatchingTex(derivative,limit))
122         self.wait(0.1)
123         self.pause()
124 
125         self.play(Create(VGroup(dot1,dot2,secant)))
126         self.wait(0.1)
127         self.pause()
128 
129         #Uma vez que os objetos que dependem do valor de x e dx foram colocados dentro das funções
130         #always_redraw eles serão alterados de acordo conforme os valores de x e dx forem alterados.
131         self.play(Write(VGroup(x_text,p_text)))
132         self.wait(0.1)
133         self.pause()
134 
135         self.play(dx.animate.set_value(0.001),run_time=8)
136         self.wait(0.1)
137         self.pause()
138 
139         self.play(FadeOut(x_text))
140         self.play(p.animate.set_value(1),run_time=5)
141         self.wait(0.1)
142         self.pause()
143 
144         self.play(p.animate.set_value(7),run_time=5)
145         self.wait(0.1)
146         self.pause()
147 
148         self.play(
149             *[FadeOut(mob)for mob in self.mobjects]
150         )
151         self.wait()
```

Algoritmo 26: OAM 2 - Pontos criticos

```
from manim import *
from manim_presentation import Slide

class pontos_criticos(Slide):
    def construct(self):
        WAIT_TIME = 1

        WM = MathTex('KallelFiori').scale(0.5)
        WM.set_opacity(0.4)
        WM.move_to([6,-3.5,0])
        self.add(WM)

        #Objeto Manim que armazena o texto Latex da função as ser desenhada
        function_tex = MathTex(r'f(x)=x^3-12x^2+45x-48')

        #Objeto Manim que armazena o texto Latex da derivada da função as ser desenhada
        derivative_tex = MathTex('f\'(x)=3x^2-24x+45') #raizes: 5 e 3

        #Objeto Manim que armazena os eixos cartesianos
        #Com alterações na função a ser mostrada é importante mudar o valores do graficos para que
        #a função seja melhor desenhada.
        axes1 = (
            Axes(
                x_range = [0,8,1],
                x_length = 8,
                y_range = [0,20,5],
                y_length = 6,
                axis_config = {"include_numbers": True, "include_tip":False},
            )
            .set_color(WHITE)
        )

        #Adicionando texto para indicar os eixos
        axes1_labels = axes1.get_axis_labels(x_label="x",y_label="y")

        #Função para desenharmos o grafico
        def f(x):
            return (x**3 - (12*(x**2)) + 45*x) - 48

        #Função para desenharmos o grafico da função derivada
        def derivative(x):
            return ((3*x**2)-(24*x)+45)

        #Objeto Manim que armazena o grafico da função previamente definida por f
        graph1 = axes1.plot(
            f, x_range=[1.8,6.5], color=BLUE
        )

        #Value Tracker armazena o valor de x, e pode ser usado para animar a reta tangente
        #posteriormente.
        #Os valores iniciais podem ser alterados conforme necessidade.
        x = ValueTracker(2)

        #dx nesse exemplo não precisa ser um ValueTracker, uma vez que só estamos
        #interessados na animação da reta tangente.
        dx = 0.001
```

```
57
58          #Objeto Manim que armazena a reta secante ao grafico conforme x e dx.
59          #O objeto precisa estar dentro da função always_redraw uma vez que o valor de x vai ser
60          #alterado posteriomente.
61          tangent = always_redraw(
62              lambda: axes1.get_secant_slope_group(
63                  x = x.get_value(),
64                  graph = graph1,
65                  dx = dx,
66                  secant_line_color = GREEN,
67                  secant_line_length = 4,
68              )
69          )
70
71          #Objeto Manim que armazena o ponto no grafico correspondente ao valor de p.
72          #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p ira ser
73          #alterado posteriomente.
74          dot1 = always_redraw(
75              lambda: Dot()
76              .scale(0.7)
77              .move_to(axes1.c2p(x.get_value(),f(x.get_value())))
78          )
79
80          #Movendo o ponto para a camada mais acima, para que a reta tangente e o grafico não sejam
81          #desenhados acima do ponto
82          dot1.z_index = 1
83
84          #VGroup é usado para agrupar elementos em um unico lugar, assim modificações posteriores podem ser
85          #feitas diretamente no grupo ao invez de elemento a elemento.
86          left_elements = VGroup(tangent,graph1,axes1,axes1_labels,function_tex)
87
88          #Objeto Manim que armazena os eixos cartesianos
89          #Com alterações na função a ser mostrada é importante mudar o valores do graficos para que
90          #a função seja melhor desenhada.
91          #Esse segundo grafico vai ser usado para mostrar a função derivada
92          axes2 = (
93              Axes(
94                  x_range = [0,7,1],
95                  x_length = 7,
96                  y_range = [-3,2,1],
97                  y_length = 6,
98                  axis_config = {"include_numbers": True, "include_tip":False},
99              )
100             .set_color(WHITE)
101         )
102         #Movendo os eixos para a camada mais acima, para que o grafico não seja desenhado
103         #acima dos eixos
104         axes2.z_index = 1
105
106         #Adicionando texto para indicar os eixos
107         axes2_labels = axes2.get_axis_labels(x_label="x",y_label="y")
108
109         #Objeto Manim que armazena o grafico da função previamente definida por derivative
110         graph2 = axes2.plot(
111             derivative, x_range=[2.8,5.2], color=BLUE
112         )
113
114
115         #Nesse bloco fazemos ajustes no posicionamentos do elementos para inserir o segundo
116         #grafico posteriomente
117         derivative_tex.shift(UP*3)
118         right_elements = VGroup(derivative_tex,axes2,axes2_labels,graph2)
119         right_elements.scale(0.7)
120         right_elements.shift(RIGHT*4)
121         right_elements.shift(DOWN*0.2)
122
123
124         self.play(Write(function_tex))
125         self.wait(WAIT_TIME)
126         self.pause()
```

```
127
128
129             function_tex_target = function_tex.generate_target()
130             function_tex_target.shift(UP*3)
131             self.play(MoveToTarget(function_tex))
132             self.wait(WAIT_TIME)
133             self.pause()
134
135
136             self.play(Write(VGroup(axes1,graph1,axes1_labels,tangent,dot1)))
137             self.wait(WAIT_TIME)
138             self.pause()
139
140
141             left_elements_target = left_elements.generate_target()
142             left_elements_target.scale(0.7)
143             left_elements_target.shift(LEFT*3)
144             self.play(MoveToTarget(left_elements))
145             self.wait(WAIT_TIME)
146             self.pause()
147
148
149
150             self.play(Write(derivative_tex))
151             self.wait(WAIT_TIME)
152             self.pause()
153
154
155             self.play(Write(VGroup(axes2,axes2_labels,graph2)))
156             self.wait(WAIT_TIME)
157             self.pause()
158
159
160             derivative_tex_target = derivative_tex.generate_target()
161             derivative_tex_target.shift(UP)
162
163             axes2_elements=VGroup(axes2,graph2,axes2_labels)
164             axes2_elements_target  = axes2_elements.generate_target()
165             axes2_elements_target.shift(DOWN)
166             self.play(MoveToTarget(derivative_tex),MoveToTarget(axes2_elements))
167
168             solution_tex1 = MathTex('f\'(x)=0').scale(0.7)
169             solution_tex2 = MathTex('x = 3').scale(0.7)
170             solution_tex3 = MathTex('ou').scale(0.7)
171             solution_tex4 = MathTex('x = 5').scale(0.7)
172
173             solution_tex3.next_to(solution_tex2,DOWN)
174             solution_tex4.next_to(solution_tex3,DOWN)
175
176             solution_tex_x_values = VGroup(solution_tex2,solution_tex3,solution_tex4)
177             solution_tex_x_values.next_to(solution_tex1,RIGHT)
178
179             solution_tex = VGroup(solution_tex_x_values,solution_tex1)
180
181             solution_tex.next_to(derivative_tex,DOWN)
182             self.play(Write(solution_tex))
183
184             right_dot1 = Dot().move_to(axes2.c2p(3,0))
185             right_dot2 = Dot().move_to(axes2.c2p(5,0))
186
187             right_dot1.z_index = 1
188             right_dot2.z_index = 1
189
190             self.play(Write(VGroup(right_dot1,right_dot2)))
191             self.wait(WAIT_TIME)
192             self.pause()
193
194             #Objeto Manim que muda de acordo com o valor de x
195             #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p e dx irão ser
196             #alterados posteriomente.
197             x_value_tex = always_redraw(
198                 lambda:
199                 MathTex('x=',str(x.get_value())[:5])
200                 .scale(0.7)
201                 .shift(UP*1.3+LEFT*4)
202             )
```

```
203
204
205             #Objeto Manim que muda de acordo com o valor da função derivada.
206             #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p e dx irão ser
207             #alterados posteriomente.
208             derivative_function_value_tex = always_redraw (
209                 lambda:
210                     MathTex('f\'(',str(x.get_value())[:5],')=',str(derivative(x.get_value()))[:5])
211                     .scale(0.7)
212                     .next_to(x_value_tex,DOWN)
213             )
214
215             self.play(Write(VGroup(x_value_tex,derivative_function_value_tex)))
216             self.wait(WAIT_TIME)
217             self.pause()
218
219             self.play(x.animate.set_value(3),run_time=2)
220             self.wait(WAIT_TIME)
221             self.pause()
222
223             self.play(x.animate.set_value(5),run_time=2)
224             self.wait(WAIT_TIME)
225             self.pause()
226
227
228             self.play(
229                 *[FadeOut(mob)for mob in self.mobjects]
230             )
231             self.wait()
```

Algoritmo 27: OAM 3 - Maximos e Minimos

```
from manim import *
from manim_presentation import Slide

class maximos_minimos(Slide):
    def construct(self):
        WM = MathTex('KallelFiori').scale(0.5)
        WM.set_opacity(0.4)
        WM.move_to([6,-3.5,0])
        self.add(WM)

        #Objeto Manim que armazena o texto Latex do "enunciado" do problema que vamos analisar
        example1 = MathTex("Estude f com relação a máximos e mínimos",tex_environment="flushleft")
        example1.scale(0.8)
        example1.shift(UP*3.5 + LEFT*3)

        #Objeto Manim que armazena o texto Latex da função as ser desenhada
        function_tex = MathTex(r'f(x)=x^3-3x^2+3')

        #Objeto Manim que armazena os eixos cartesianos
        #Com alterações na função a ser mostrada é importante mudar o valores do graficos para que
        #a função seja melhor desenhada.
        axes1 = (
            Axes(
                x_range = [-1,3,1],
                x_length = 5,
                y_range = [-1,4,1],
                y_length = 5,
                axis_config = {"include_numbers": True,
                               "include_tip":False}
            )
            .set_color(WHITE)
        )
        axes1.z_index = 1 #Movendo os eixos para uma camada mais acima

        #Adicionando texto para indicar os eixos
        axes1_labels = axes1.get_axis_labels(x_label="x",y_label="y")

        #Função para desenharmos o grafico
        def f(x):
            return (x**3 - (3*(x**2)) + 3)

        #Objeto Manim que armazena o grafico da função previamente definida por f
        graph1 = axes1.plot(
            f, x_range=[-1,3], color=BLUE
        )

        #Value Tracker armazena o valor de x, e pode ser usado para animar a reta tangente
        #posteriormente.
        #Os valores iniciais podem ser alterados conforme necessidade.
        x = ValueTracker(-0.5)

        #dx nesse exemplo não precisa ser um ValueTracker, uma vez que só estamos
        #interessados na animação da reta tangente.
        dx = 0.001
```

```
55
56          #Objeto Manim que armazena a reta secante ao grafico conforme x e dx.
57          #O objeto precisa estar dentro da função always_redraw uma vez que o valor de x vai ser
58          #alterado posteriomente.
59          tangent = always_redraw(
60              lambda: axes1.get_secant_slope_group(
61                  x = x.get_value(),
62                  graph = graph1,
63                  dx = dx,
64                  secant_line_color = GREEN,
65                  secant_line_length = 4,
66              )
67          )
68
69          #Objeto Manim que armazena o ponto no grafico correspondente ao valor de p.
70          #O objeto precisa estar dentro da função always_redraw uma vez que os valores de p ira ser
71          #alterado posteriomente.
72          dot1 = always_redraw(
73              lambda: Dot()
74              .scale(0.7)
75              .move_to(axes1.c2p(x.get_value(),f(x.get_value())))
76          )
77
78          #Movendo o ponto para a camada mais acima, para que a reta tangente e o grafico não sejam desenhados
79          #acima do ponto
80          dot1.z_index = 1
81
82          #VGroup é usado para agrupar elementos em um unico lugar, assim modificações posteriores podem ser
83          #feitas diretamente no grupo ao invez de elemento a elemento.
84          graph1_group = VGroup(tangent,graph1,axes1,axes1_labels,dot1)
85          graph1_group.scale(0.8)
86
87          self.play(Write(VGroup(example1)))
88          self.wait(0.1)
89          self.pause()
90
91          self.play(Write(function_tex))
92          self.wait(0.1)
93          self.pause()
94
95          function_tex_target = function_tex.generate_target()
96          function_tex_target.shift(UP*2)
97          self.play(MoveToTarget(function_tex))
98          self.wait(0.1)
99          self.pause()
100
101         #Alinhamentos precisa ser definido apos a modunça de posição do objeto function_tex
102         graph1_group.next_to(function_tex,DOWN)
103         self.play(Write(VGroup(axes1,graph1,axes1_labels)))
104         self.wait(0.1)
105         self.pause()
106
107         #Movendo os elementos mais a esquerda para abrir espaço a direita
108         left_elements = VGroup(graph1_group,function_tex)
109         left_elements_target = left_elements.generate_target()
110         left_elements_target.scale(1.1)
111         left_elements_target.shift(LEFT*3)
112         self.play(MoveToTarget(left_elements))
113         self.wait(0.1)
114         self.pause()
115
116         derivative_tex = MathTex('f\'(x)= 3x^2 - 6x')
117         derivative_tex.shift(RIGHT*3.5)
118         derivative_tex.shift(UP*(function_tex.get_center()[1])) #alinhamento com a função anterior
119         self.play(Write(derivative_tex))
120         self.wait(0.1)
121         self.pause()
```

```
122
123             solution_tex1 = MathTex('f\'(x)=0').scale(0.7)
124             solution_tex2 = MathTex('x = 0').scale(0.7)
125             solution_tex3 = MathTex('ou').scale(0.7)
126             solution_tex4 = MathTex('x = 2').scale(0.7)
127
128             solution_tex3.next_to(solution_tex2,DOWN)
129             solution_tex4.next_to(solution_tex3,DOWN)
130
131             solution_tex_x_values = VGroup(solution_tex2,solution_tex3,solution_tex4)
132             solution_tex_x_values.next_to(solution_tex1,RIGHT)
133
134             solution_tex = VGroup(solution_tex_x_values,solution_tex1)
135
136             solution_tex.next_to(derivative_tex,DOWN)
137             self.play(Write(solution_tex))
138             self.wait(0.1)
139             self.pause()
140
141             #Objeto Manim que armazena uma linha de numeros, funcionando como o objeto Axes(), mas mostrando
142             #apenas um eixo.
143             derivative_number_line = NumberLine(
144                 x_range =[-0.9,2.9,2],
145                 length = 4,
146                 color = WHITE,
147                 numbers_to_include = {0,2},
148                 label_direction=UP
149             )
150             derivative_number_line.next_to(solution_tex,DOWN,buff=1.0)
151             derivative_number_line_label = MathTex("f\'").scale(0.6).next_to(derivative_number_line,LEFT)
152
153             #Função python que retorna os falores da função derivada
154             def derivative(x):
155                 return (3*(x**2) - 6*x)
156
157             #Objeto Manim que muda de acordo com o valor da derivada da f.
158             #O objeto precisa estar dentro da função always_redraw uma vez que o valor de x vai ser
159             #alterado posteriomente.
160             derivative_text = always_redraw(
161                 lambda: Tex('f\'(x)='+str(derivative(x.get_value()))[:4])
162                 .scale(0.6)
163                 .next_to(derivative_number_line,UP)
164                 .shift(LEFT*2)
165             )
166
167             #Objeto Manim que muda de acordo com o valor de x.
168             #O objeto precisa estar dentro da função always_redraw uma vez que o valor de x vai ser
169             #alterado posteriomente.
170             x_text = always_redraw(
171                 lambda: Tex(r'x='+str(x.get_value())[:4])
172                 .scale(0.6)
173                 .next_to(derivative_text,UP)
174             )
175
176             dot2 = always_redraw(
177                 #n2p é semelhante a função c2p do objeto Axes(
178                 lambda: Dot(derivative_number_line.n2p(x.get_value()))
179             )
180
181             function_number_line = NumberLine(
182                 x_range =[-0.9,2.9,2],
183                 length = 4,
184                 color = WHITE,
185                 numbers_to_include = {0,2},
186                 label_direction=UP
187             )
188             function_number_line.next_to(derivative_number_line,DOWN,buff=1.0)
189             function_number_line_label = MathTex("f").scale(0.6).next_to(function_number_line,LEFT)
```

```
190
191
192             self.play(Write(VGroup(derivative_number_line,
193                                    x_text,
194                                    dot2,
195                                    derivative_text,
196                                    derivative_number_line_label,
197                                    function_number_line,
198                                    function_number_line_label)))
199             self.wait(0.1)
200             self.pause()
201
202             #Nos blocos abaixo fazemos as variações de x indo de -0.5 até 2.5, fazendo pequenas pausas
203             #para mostrar os valores da derivada, assim, obtendo as informações para classificarmos
204             #os pontos em minimos e maximos locais.
205
206             #valor 0
207             self.play(x.animate.set_value(0),run_time=2)
208             self.wait(0.1)
209             self.pause()
210
211             plus1 = MathTex("+").move_to(derivative_number_line.n2p(-0.5))
212             plus1.shift(UP*0.3)
213             self.play(Write(plus1))
214             self.wait(0.1)
215             self.pause()
216
217             up_arrow1 = Vector(UR*0.3).scale(0.8)
218             up_arrow1.move_to(function_number_line.n2p(-0.5))
219             up_arrow1.shift(UP*0.3)
220             self.play(Write(up_arrow1))
221             self.wait(0.1)
222             self.pause()
223
224             #valor 2
225
226             self.play(x.animate.set_value(2),run_time=2)
227             self.wait(0.1)
228             self.pause()
229
230             minus = MathTex("-").move_to(derivative_number_line.n2p(1))
231             minus.shift(UP*0.3)
232             self.play(Write(minus))
233             self.wait(0.1)
234             self.pause()
235
236             down_arrow1 = Vector(DR*0.3).scale(0.8).move_to(function_number_line.n2p(1))
237             down_arrow1.shift(UP*0.3)
238             self.play(Write(down_arrow1))
239             self.wait(0.1)
240             self.pause()
241
242             #valor 2.5
243
244             self.play(x.animate.set_value(2.5),run_time=2)
245             self.wait(0.1)
246             self.pause()
247
248             plus2 = MathTex("+").move_to(derivative_number_line.n2p(2.5))
249             plus2.shift(UP*0.3)
250             self.play(Write(plus2))
251             self.wait(0.1)
252             self.pause()
253
254             up_arrow2 = Vector(UR*0.3).scale(0.8).move_to(function_number_line.n2p(2.5))
255             up_arrow2.shift(UP*0.3)
256             self.play(Write(up_arrow2))
257             self.wait(0.1)
258             self.pause()
259
260             #Agrupamento dos elementos a direita
261             function_number_line_elements = VGroup(plus1,
262                                                    plus2,
263                                                    minus,
264                                                    function_number_line,
265                                                    function_number_line_label)
```

```
266         derivative_number_line_elements = VGroup(up_arrow1,
267                                                  up_arrow2,
268                                                  down_arrow1,
269                                                  derivative_number_line_label,
270                                                  derivative_number_line,
271                                                  dot2)
272 
273         right_elements = VGroup(derivative_tex,
274                                 derivative_text,
275                                 solution_tex,
276                                 x_text,
277                                 function_number_line_elements,
278                                 derivative_number_line_elements)
279         right_elements_target = right_elements.generate_target()
280 
281         right_elements_target.shift(UP)
282         self.play(MoveToTarget(right_elements))
283 
284         res1 = MathTex("x=0 Máximo Local",tex_environment="flushleft").scale(0.8)
285         res1.next_to(right_elements,DOWN)
286         res2 = MathTex("x=2 Mínimo Local",tex_environment="flushleft").scale(0.8)
287         res2.next_to(res1,DOWN)
288 
289         self.play(Write(VGroup(res1,res2)))
290         self.wait(0.1)
291         self.pause()
292 
293         self.play(
294             *[FadeOut(mob)for mob in self.mobjects]
295         )
296         self.wait()
```

Algoritmo 28: OAM 4 - Problema do Volume Máximo de uma Caixa

```
from manim import *
from manim_presentation import Slide

class caixa(ThreeDScene,Slide):
    def construct(self):
        WAIT_TIME = 1

        WM = MathTex('KallelFiori').scale(0.5)
        WM.set_opacity(0.4)
        WM.move_to([6,-3.5,0])

        self.add_fixed_in_frame_mobjects(WM)
        self.add(WM)

        h = ValueTracker(1)
        sheet_value = 10

        self.set_camera_orientation(theta=270 * DEGREES,zoom=0.5)

        outer_sheet_full = Square(side_length=sheet_value,color=BLUE)
        outer_sheet_full.set_fill(color=BLUE,opacity=0.5)
        outer_sheet_full.move_to(ORIGIN)

        self.play(Write(outer_sheet_full))
        self.wait(WAIT_TIME)
        self.pause()

        side_label = MathTex('10').scale(2).next_to(outer_sheet_full,DOWN)

        self.play(Write(side_label))
        self.wait(WAIT_TIME)
        self.pause()

        self.play(FadeOut(side_label))
        self.wait(WAIT_TIME)
        self.pause()

        outer_sheet1_fixed = always_redraw(
            lambda: Rectangle(height=sheet_value,width=sheet_value-2*h.get_value(),color=BLUE)
            .set_fill(color=BLUE,opacity=0.5)
            .move_to(outer_sheet_full.get_center())
        )


        outer_sheet2_fixed = always_redraw(
            lambda: Rectangle(height=sheet_value-2*h.get_value(),width=sheet_value,color=BLUE)
            .set_fill(color=BLUE,opacity=0.5)
            .move_to(outer_sheet_full.get_center())
        )


        outer_sheet_union = always_redraw(
            lambda: Union(outer_sheet1_fixed,outer_sheet2_fixed)
            .move_to(outer_sheet_full.get_center())
        )
```

```
56
57          removed_squares = always_redraw(
58              lambda: Difference(outer_sheet_full,outer_sheet_union,color=RED)
59              .move_to(ORIGIN)
60              .set_fill(color=RED,opacity=0.5)
61          )
62
63          inner_sheet_fixed = always_redraw(
64              lambda: Square(side_length=sheet_value-2*h.get_value(),color=GREEN)
65              .set_fill(color=GREEN,opacity=0.5)
66              .move_to(outer_sheet_full.get_center())
67          )
68
69          h_label1_fixed = always_redraw(
70              lambda: MathTex('h')
71              .move_to([(sheet_value-2*h.get_value())/2 + (h.get_value()/2),0,0])
72          )
73
74          h_label2_fixed = always_redraw(
75              lambda: MathTex('h')
76              .move_to([0,(sheet_value-2*h.get_value())/2 + (h.get_value()/2),0])
77          )
78
79          line_y_fixed = always_redraw(
80              lambda: Line(start=ORIGIN,end=[h.get_value(),0,0])
81              .next_to(h_label1_fixed,DOWN)
82          )
83
84          line_x_fixed = always_redraw(
85              lambda: Line(start=ORIGIN,end=[0,h.get_value(),0])
86              .next_to(h_label2_fixed,LEFT)
87          )
88
89          h_value_label = always_redraw(
90              lambda: MathTex('h=',str(h.get_value())[:3])
91              .shift(UP*3)
92              .shift(LEFT*4)
93          )
94          self.add_fixed_in_frame_mobjects(h_value_label)
95
96          self.play(Write(removed_squares),
97                    FadeOut(outer_sheet_full),
98                    FadeIn(outer_sheet1_fixed),
99                    FadeIn(outer_sheet2_fixed),
100                   Write(inner_sheet_fixed),
101                   Write(h_label1_fixed),
102                   Write(line_x_fixed),
103                   Write(h_label2_fixed),
104                   Write(line_y_fixed),
105                   Write(h_value_label))
106         self.wait(WAIT_TIME)
107         self.pause()
108
109         self.play(FadeOut(removed_squares))
110         self.play(h.animate.set_value(3),run_time=2)
111         self.wait(WAIT_TIME)
112         self.pause()
113
114         self.play(h.animate.set_value(1),run_time=2)
115         self.wait(WAIT_TIME)
116         self.pause()
117
118         outer_sheet1 = always_redraw (
119             lambda: Rectangle(height=sheet_value,width=sheet_value-2*h.get_value(),color=BLUE)
120             .set_fill(color=BLUE,opacity=0.5)
121             .move_to(ORIGIN)
122             .shift(np.array([0,0,-1])*(h.get_value()/2))
123         )
```

```
124
125         outer_sheet2 = always_redraw (
126             lambda: Rectangle(height=sheet_value-2*h.get_value(),width=sheet_value,color=BLUE)
127             .set_fill(color=BLUE,opacity=0.5)
128             .move_to(ORIGIN)
129             .shift(np.array([0,0,-1])*(h.get_value()/2))
130         )
131
132         inner_sheet = always_redraw(
133             lambda: Square(side_length=sheet_value-2*h.get_value(),color=GREEN)
134             .set_fill(color=GREEN,opacity=0.5)
135             .move_to(outer_sheet1.get_center())
136         )
137
138         h_label1 = always_redraw(
139             lambda: MathTex('h')
140             .move_to([(sheet_value-2*h.get_value())/2 + (h.get_value()/2),0,0])
141             .shift(np.array([0,0,-1])*(h.get_value()/2))
142         )
143
144         h_label2 = always_redraw(
145             lambda: MathTex('h')
146             .move_to([0,(sheet_value-2*h.get_value())/2 + (h.get_value()/2),0])
147             .shift(np.array([0,0,-1])*(h.get_value()/2))
148         )
149
150         line_y = always_redraw(
151             lambda: Line(start=ORIGIN,end=[h.get_value(),0,0])
152             .next_to(h_label1,DOWN)
153         )
154
155         line_x = always_redraw(
156             lambda: Line(start=ORIGIN,end=[0,h.get_value(),0])
157             .next_to(h_label2,LEFT)
158         )
159
160         self.play(FadeOut(h_label1_fixed),
161                   FadeOut(line_x_fixed),
162                   FadeOut(h_label2_fixed),
163                   FadeOut(line_y_fixed))
164
165         outer_sheet_full.move_to(inner_sheet.get_center())
166         self.play(outer_sheet1_fixed.animate.move_to(inner_sheet.get_center()),
167                   outer_sheet2_fixed.animate.move_to(inner_sheet.get_center()),
168                   inner_sheet_fixed.animate.move_to(inner_sheet.get_center()))
169
170         self.play(Write(outer_sheet1),
171                   Write(outer_sheet2),
172                   FadeOut(outer_sheet1_fixed),
173                   FadeOut(outer_sheet2_fixed),
174                   FadeOut(inner_sheet_fixed),
175                   Write(h_label1),
176                   Write(h_label2),
177                   Write(line_y),
178                   Write(line_x),
179                   Write(inner_sheet))
180         self.wait(WAIT_TIME)
181         self.pause()
182
183         box = always_redraw(
184             lambda: Prism(dimensions=[(sheet_value-(2*h.get_value())),
185                                       (sheet_value-(2*h.get_value())),
186                                       h.get_value()])
187                                       .rotate(PI / 2)
188         )
```

```
189
190             self.play(Write(box))
191             self.move_camera(phi=70 * DEGREES, theta=315 * DEGREES,zoom=0.8)
192             self.wait(WAIT_TIME)
193             self.pause()
194
195             self.play(h.animate.set_value(3),run_time=2)
196             self.wait(WAIT_TIME)
197             self.pause()
198
199             self.play(h.animate.set_value(2),run_time=2)
200             self.wait(WAIT_TIME)
201             self.pause()
202
203             h_label_high = always_redraw(
204                 lambda: MathTex('h')
205                 .move_to([(sheet_value-(2*h.get_value()))/2,0.5,0])
206                 .rotate(90 * DEGREES,[1,0,0])
207                 .rotate(90 * DEGREES,[0,0,1])
208             )
209             line_high = always_redraw(
210                 lambda: Line(start=[0,0,h.get_value()/2],end=[0,0,-h.get_value()/2])
211                 .move_to([(sheet_value-(2*h.get_value()))/2,0,0])
212             )
213
214             self.play(Write(h_label_high),Write(line_high))
215             self.wait(WAIT_TIME)
216             self.pause()
217
218             side1_label = always_redraw(
219                 lambda: MathTex('10-2h')
220                 .move_to([0,(-(sheet_value-(2*h.get_value()))/2) - 0.5,-h.get_value()/2])
221             )
222
223             side2_label = always_redraw(
224                 lambda: MathTex('10-2h')
225                 .move_to([((sheet_value-(2*h.get_value()))/2) + 0.5,0,-h.get_value()/2])
226                 .rotate(angle=90 * DEGREES, axis=[0,0,1])
227             )
228
229             self.play(FadeOut(outer_sheet1),
230                       FadeOut(outer_sheet2),
231                       FadeOut(inner_sheet),
232                       FadeOut(h_label1),
233                       FadeOut(h_label2),
234                       FadeOut(line_y),
235                       FadeOut(line_x))
236             self.play(Write(side1_label),Write(side2_label))
237             self.wait(WAIT_TIME)
238             self.pause()
239
240             #=============== Part 2 ===============
241             volume_func = MathTex('V(h)=h(10-2h)^2')
242             self.add_fixed_in_frame_mobjects(volume_func)
243             volume_func.shift(UP*3)
244
245             self.play(Write(volume_func))
246             self.wait(WAIT_TIME)
247             self.pause()
248
249             axes = (
250                 Axes(
251                     x_range = [0,5,1],
252                     x_length = 5,
253                     y_range = [0,75,5],
254                     y_length = 6,
255                     x_axis_config = {"include_numbers": True, "include_tip":False},
256                     y_axis_config={
257                     "numbers_to_include": np.arange(0, 75.1, 5),
258                     "include_tip":False},
259                 )
260                 .set_color(WHITE)
261             )
262
263             def vol(x):
264                 return x * (10-2*x)**2
```

```
265
266             graph = axes.plot(
267                 vol, x_range=[0,5], color=BLUE
268             )
269
270             self.move_camera(theta=300 * DEGREES,zoom=0.4,frame_center=[0,6,-1])
271             self.add_fixed_in_frame_mobjects(axes)
272             axes.scale(0.9)
273             axes.shift(RIGHT*4)
274             self.add_fixed_in_frame_mobjects(graph)
275             graph.scale(0.9)
276             graph.shift(RIGHT*4)
277
278
279             self.play(Write(axes),Write(graph),h.animate.set_value(1),run_time=1)
280             volume_func_target = volume_func.generate_target()
281             volume_func_target.scale(0.8)
282             volume_func_target.shift(UP*0.3)
283
284             self.play(MoveToTarget(volume_func))
285             self.wait(WAIT_TIME)
286             self.pause()
287
288             line_vertical = always_redraw(
289                 lambda: Line(start = axes.c2p(h.get_value(),0),
290                              end = axes.c2p(h.get_value(),vol(h.get_value())),
291                              color = YELLOW
292                             )
293             )
294             line_horizontal = always_redraw(
295                 lambda: Line(start = axes.c2p(h.get_value(),vol(h.get_value())),
296                              end = axes.c2p(0,vol(h.get_value())),
297                              color = YELLOW
298                             )
299             )
300             self.add_fixed_in_frame_mobjects(line_vertical)
301             self.add_fixed_in_frame_mobjects(line_horizontal)
302
303             dot = always_redraw(
304                 lambda: Dot(point=axes.c2p(h.get_value(),vol(h.get_value())),color=YELLOW)
305             )
306             self.add_fixed_in_frame_mobjects(dot)
307
308             volume_label = always_redraw(
309                 lambda: MathTex('V(h)=',str(vol(h.get_value()))[:4])
310                 .next_to(h_value_label,DOWN)
311             )
312             self.add_fixed_in_frame_mobjects(volume_label)
313
314             self.play(Write(line_vertical),Write(line_horizontal),Write(dot),Write(volume_label))
315             self.wait(WAIT_TIME)
316             self.pause()
317
318             self.play(h.animate.set_value(4.5),run_time=5)
319             self.wait(WAIT_TIME)
320             self.pause()
321
322             self.play(h.animate.set_value(1),run_time=5)
323             self.wait(WAIT_TIME)
324             self.pause()
325
326             self.play(
327                 *[FadeOut(mob)for mob in self.mobjects]
328             )
329             self.wait()
```