Universidade da Beira Interior
Departamento de Informática

Nº 1 – 2025: *Trabalho Prático do Grupo #1*

Elaborado por:

Filênia Fernandes
Lara Fernandes Gabry
Redinho Marques Zumbo
Sebastian Vallejo Jimenez

Orientador:

Professor Doutor Tiago Filipe Dias dos Santos Roxo

11 de dezembro de 2025

Agradecimentos

A elaboração deste trabalho contou com a orientação do Professor Tiago Filipe Dias dos Santos Roxo, no âmbito da unidade curricular Laboratórios de Informática. Os esclarecimentos prestados, bem como o acompanhamento contínuo ao longo do semestre, foram determinantes para a adequada compreensão dos conceitos abordados e para o desenvolvimento rigoroso das funcionalidades implementadas no projeto.

De salientar também o contributo dos colegas, cuja partilha de ideias e colaboração em momentos de discussão técnica permitiram aprofundar o entendimento dos temas tratados e melhorar a qualidade do trabalho realizado.

Agradece-se igualmente o acesso aos recursos e ferramentas disponibilizados pela instituição, que facilitaram tanto a implementação do programa como a preparação do presente relatório, e a todos os que, direta ou indiretamente, contribuíram para a concretização deste projeto.

Resumo

Este relatório descreve o desenvolvimento de um programa interativo em *Python* para processamento de vetores e matrizes, no contexto da unidade curricular Laboratórios de Informática. O trabalho teve como principal objetivo consolidar conceitos introdutórios de programação, nomeadamente leitura e validação de dados, decomposição funcional e manipulação de estruturas de dados simples em ambiente de linha de comandos.

A resolução do problema passou pela implementação de um programa baseado em menu executado em *Command Line Interface* (Ambiente de Linha de Comandos) (CLI), que permite ao utilizador introduzir um vetor de dez valores inteiros e aplicar diversas operações sobre esse vetor e sobre matrizes dele derivadas. O código foi organizado em funções independentes, documentadas com *docstrings* e anotações em *Doxygen*, e foi desenvolvido num ambiente controlado com recurso a *Git*, *GitHub* e LaTeX para a produção do relatório técnico.

Os resultados obtidos mostram que o programa cumpre os requisitos funcionais e não funcionais definidos, apresentando um comportamento robusto face a entradas inválidas e produzindo resultados numéricos consistentes. O projeto contribuiu, assim, para o desenvolvimento de competências fundamentais em programação, validação de software e documentação técnica.

Palavras-Chave e Keywords

Palavras-chave: Análise de dados, Documentação de código, Interfaces em linha de comandos, LaTeX, Matrizes, Programação em Python, Testes de software, Vetores.

Keywords: Command-line interfaces, Data analysis, LaTeX, Matrices, Python programming, Software testing, Source code documentation, Vectors.

Acrónimos

BVA	*Boundary Value Analysis* (Análise de Valores Limite)
CD	Ciência de Dados
CLI	*Command Line Interface* (Ambiente de Linha de Comandos)
E2E	*End-to-End* (Testes de Ponta-a-Ponta)
HCI	*Human-Computer Interaction* (Interação Humano–Computador)
IA	Inteligência Artificial
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
QA	*Quality Assurance* (Garantia de Qualidade)
SE	*Software Engineering* (Engenharia de Software)
UBI	Universidade da Beira Interior

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo

1

Introdução

1.1 Enquadramento

O presente documento constitui um relatório técnico elaborado no âmbito do trabalho prático da unidade curricular Laboratórios de Informática, integrada no primeiro ano, primeiro semestre da licenciatura em Inteligência Artificial (IA) e Ciência de Dados (CD) da Universidade da Beira Interior (UBI). O relatório descreve de forma estruturada o problema proposto, a solução implementada e o processo seguido na concretização do projeto, adotando uma escrita impessoal e orientada para a documentação rigorosa do trabalho desenvolvido.

O projeto enquadra-se no contexto da formação inicial em informática aplicada às áreas de IA e CD, tendo como principal objetivo consolidar conhecimentos introdutórios de programação e manipulação de dados. No âmbito desta unidade curricular, o trabalho prático assume um papel integrador, estabelecendo a ligação entre os conteúdos teóricos apresentados em aula e a sua aplicação prática na resolução de problemas concretos.

Em termos científicos e tecnológicos, o projeto insere-se na área da Computação, com particular incidência em duas subáreas fundamentais: a programação imperativa em *Python* e a manipulação de estruturas de dados numéricas, nomeadamente vetores e matrizes. Paralelamente, o desenvolvimento do programa e a elaboração do relatório em *LaTeX* permitem introduzir e consolidar boas práticas de engenharia de software a nível introdutório, como a organização modular do código, a validação sistemática de entradas e a documentação cuidada das funcionalidades implementadas.

1.2 Motivação UBI

A abordagem do problema proposto neste trabalho reveste-se de especial importância no contexto da formação em IA e CD, uma vez que o processamento estruturado de dados numéricos, a validação de entradas e a correta organização de algoritmos constituem competências basilares em qualquer aplicação computacional. A capacidade de manipular vetores e matrizes de forma rigorosa é um requisito transversal a inúmeras áreas, desde a análise de dados até ao desenvolvimento de sistemas inteligentes.

Para além do aspeto estritamente técnico, a motivação do projeto reside na necessidade de fomentar, desde uma fase inicial do percurso académico, práticas de trabalho alinhadas com a realidade profissional. A implementação de um programa interativo com múltiplas funcionalidades, aliada à sua documentação formal, promove o desenvolvimento de metodologias de trabalho estruturadas, clareza na comunicação técnica e atenção ao rigor na escrita de relatórios científicos.

Deste modo, o trabalho prático contribui não apenas para a consolidação de conhecimentos de programação introdutória, mas também para a aquisição de competências metodológicas essenciais em IA e CD, nomeadamente a capacidade de planear, desenvolver, documentar e analisar soluções computacionais de forma sistemática e fundamentada.

1.3 Problema e Objetivos do Trabalho

O problema específico abordado neste projeto consiste na necessidade de desenvolver um programa capaz de recolher, validar e processar dados numéricos introduzidos pelo utilizador, assegurando simultaneamente correção lógica, robustez na validação de entradas e clareza na interação. Em contextos introdutórios de programação, a ausência de estruturas adequadas de validação e organização conduz frequentemente a soluções pouco fiáveis ou difíceis de manter, o que compromete tanto a qualidade do software como a aprendizagem dos conceitos fundamentais.

Neste sentido, o projeto procura responder ao desafio de integrar, num único sistema coerente, múltiplas operações sobre estruturas de dados numéricas, garantindo que todas as funcionalidades são acessíveis através de um mecanismo de interação controlado e intuitivo, baseado num menu textual e compatível com a execução em ambiente de *Command Line Interface* (Ambiente de Linha de Comandos) (CLI).

O objetivo principal do trabalho consiste no desenvolvimento de um programa interativo, implementado em *Python*, capaz de ler um conjunto fixo

de dez números inteiros, validar cada entrada de acordo com um intervalo previamente definido e disponibilizar um conjunto de operações de processamento e análise sobre esses dados. No final do projeto, pretende-se obter uma aplicação funcional e robusta, capaz de responder corretamente às opções selecionadas pelo utilizador e de manter um ciclo de execução consistente até à sua finalização explícita.

De forma complementar, o trabalho tem como objetivo garantir que a solução desenvolvida respeita boas práticas de programação introdutória, nomeadamente no que diz respeito à modularização de funções, à clareza do fluxo de execução e à correta gestão de erros de entrada, promovendo uma base sólida para o desenvolvimento de aplicações mais complexas nas áreas de IA e CD.

Para além do objetivo principal, foram definidos vários objetivos secundários que suportam e complementam a solução proposta. Em particular, o projeto visa alcançar os seguintes objetivos técnicos:

- assegurar uma validação rigorosa das entradas do utilizador, garantindo que todos os valores respeitam o intervalo permitido e o tipo de dados esperado;
- implementar operações fundamentais sobre vetores, incluindo ordenação, filtragem, cálculo de funções matemáticas e determinação de medidas estatísticas;
- construir e manipular estruturas matriciais, nomeadamente uma matriz 2×10 derivada do vetor inicial, uma matriz 10×10 resultante de produtos entre vetores e a respetiva transposta;
- disponibilizar funcionalidades adicionais, como a identificação de números compostos, a leitura de um segundo vetor e a apresentação de informação de ajuda através do menu e da *flag* `--help`.

Para além dos aspetos técnicos, o projeto tem ainda como objetivos secundários o desenvolvimento de competências metodológicas relevantes no contexto de Laboratórios de Informática, incluindo a documentação estruturada do código, a elaboração de um relatório técnico em *LaTeX* e a consolidação de práticas de trabalho adequadas à utilização de ferramentas e ambientes típicos do desenvolvimento de software.

1.4 Abordagem

A abordagem adotada para a resolução do problema baseou-se no desenvolvimento incremental e estruturado de uma aplicação interativa, alinhada com os princípios da programação introdutória e com as boas práticas apresentadas na unidade curricular. Antes de iniciar a codificação, foi realizada uma análise detalhada do enunciado, de forma a compreender o comportamento esperado de cada funcionalidade e a garantir que todos os requisitos fossem corretamente interpretados e integrados na solução final.

Com base nessa análise, optou-se por uma abordagem modular, em que cada funcionalidade do programa foi implementada sob a forma de uma função independente, devidamente documentada. Esta estratégia permitiu isolar responsabilidades, facilitar a manutenção do código e assegurar que a interação com o utilizador fosse gerida de forma centralizada através de um menu textual executado em ambiente de CLI. A implementação foi realizada de forma progressiva, iniciando pelas operações mais simples e evoluindo gradualmente para funcionalidades de maior complexidade.

De forma a concretizar a abordagem definida, o desenvolvimento do trabalho seguiu o seguinte conjunto de passos ordenados:

1. análise dos requisitos do enunciado e identificação das funcionalidades obrigatórias e adicionais;
2. definição da estrutura geral do programa e do ciclo principal de interação com o utilizador;
3. implementação das funções de leitura e validação do vetor inicial;
4. desenvolvimento das operações fundamentais sobre vetores;
5. implementação das funcionalidades associadas à manipulação de matrizes;
6. integração de funcionalidades adicionais, incluindo leitura de um segundo vetor e mecanismos de ajuda;
7. validação global do funcionamento do programa e consolidação da versão final.

1.5 Organização do Documento

De forma a apresentar de modo claro e coerente o trabalho desenvolvido, o presente documento encontra-se organizado em várias partes e capítulos, es-

truturados para conduzir o leitor desde o enquadramento geral do problema até às conclusões finais e perspetivas de evolução futura.

Os elementos introdutórios, incluídos no *FrontMatter*, compreendem a capa, os agradecimentos, o resumo, as palavras-chave, a lista de acrónimos e os restantes componentes preliminares necessários à contextualização do trabalho. Estes elementos fornecem uma visão global do projeto antes da apresentação do conteúdo técnico.

O Capítulo 1 corresponde à introdução, onde são apresentados o contexto da unidade curricular, a motivação do trabalho, o problema considerado, os objetivos definidos e a abordagem geral adotada para a sua resolução.

No Capítulo 2 é apresentado o enquadramento teórico do projeto, com foco no estado da arte e nos trabalhos relacionados. Neste capítulo são discutidos os principais conceitos e práticas relevantes para a resolução do problema, bem como analisadas abordagens semelhantes aplicadas no contexto do ensino introdutório da programação, permitindo enquadrar e contextualizar a solução desenvolvida.

O Capítulo 3 descreve o planeamento e as opções de engenharia de software adotadas. Neste capítulo são detalhados os requisitos funcionais e não funcionais, as tecnologias e ferramentas utilizadas e a forma como estas foram articuladas no desenvolvimento do projeto.

No Capítulo 4, referente à execução e desenvolvimento, são apresentados os aspetos práticos da utilização do programa, incluindo o ambiente de execução, as dependências, a estrutura geral do sistema e o procedimento necessário para a sua correta execução em linha de comandos.

O Capítulo 5 é dedicado aos testes e à validação do sistema. São descritos os critérios de teste definidos, os cenários de execução considerados, os resultados obtidos e os ajustes realizados a partir da verificação do comportamento do programa.

Por fim, o Capítulo 6 apresenta as conclusões principais do trabalho e identifica possíveis direções para trabalho futuro, refletindo sobre os objetivos alcançados, as limitações encontradas e o potencial de evolução da solução desenvolvida. O documento termina com a bibliografia, onde são listadas as fontes consultadas ao longo do trabalho.

Capítulo

2

Estado da Arte

2.1 Introdução

Este capítulo apresenta o enquadramento teórico do projeto, abordando o estado da arte e trabalhos relacionados relevantes para a resolução do problema. São discutidos os principais conceitos associados à programação introdutória, nomeadamente a manipulação de vetores e matrizes e a interação com o utilizador em ambientes de linha de comandos, que fundamentam a implementação do sistema desenvolvido.

Adicionalmente, são analisadas abordagens semelhantes existentes no contexto educativo, permitindo enquadrar o projeto num panorama académico mais amplo, identificar as suas particularidades e justificar as principais opções técnicas adotadas.

2.2 Estado-da-Arte

A utilização da linguagem de programação *Python* no ensino introdutório da computação representa atualmente o estado-da-arte em numerosos cursos de ciência e engenharia. A sua adoção generalizada deve-se, em grande medida, à combinação entre uma sintaxe simples, uma semântica expressiva e uma vasta biblioteca *standard*, que permite desenvolver programas funcionais sem uma carga excessiva de complexidade técnica. Segundo [7], estas características tornam *Python* particularmente adequada para a introdução de conceitos fundamentais como controlo de fluxo, modularização através de funções e manipulação de estruturas de dados básicas, promovendo uma aprendizagem progressiva e orientada à resolução de problemas.

No contexto do processamento numérico e da CD, a manipulação de vetores e matrizes constitui uma componente central da computação científica. Operações como ordenação, filtragem, cálculo de estatísticas descritivas, produtos matriciais e transposição são amplamente utilizadas em aplicações que lidam com grandes volumes de dados e com modelos matemáticos. Obras de referência, como [12], descrevem estas operações como blocos fundamentais sobre os quais assentam algoritmos mais avançados, salientando a importância de implementações corretas, eficientes e bem estruturadas. Em ambientes académicos, estas técnicas são frequentemente exploradas em versões simplificadas, com o objetivo de consolidar a ligação entre conceitos matemáticos e a sua concretização computacional.

A interação humano-computador em aplicações introdutórias continua, em muitos cenários, a basear-se em interfaces textuais executadas em ambiente de CLI. Este tipo de interação, embora menos sofisticado do que interfaces gráficas, apresenta vantagens significativas em termos de portabilidade, simplicidade e controlo do fluxo de execução. De acordo com princípios clássicos de *Human-Computer Interaction* (Interação Humano–Computador) (HCI), interfaces *menu-driven* permitem reduzir a carga cognitiva do utilizador, orientando a escolha de ações possíveis de forma estruturada e previsível. Estas abordagens continuam a ser amplamente utilizadas em contextos de ensino e em ferramentas de linha de comandos amplamente difundidas.

Do ponto de vista da engenharia de software, a organização modular do código e a documentação estruturada são consideradas boas práticas essenciais para garantir legibilidade, manutenção e reutilização. A utilização sistemática de comentários, *docstrings* e documentação externa responde a princípios fundamentais de *Software Engineering* (Engenharia de Software) (SE), os quais defendem que a clareza do código é tão relevante quanto a sua funcionalidade. Conforme discutido em [13], práticas adequadas de leitura e escrita de código facilitam não apenas o trabalho individual, mas também a colaboração em projetos de média e grande dimensão.

Complementarmente, a elaboração de relatórios técnicos recorrendo ao sistema tipográfico *LaTeX* constitui uma prática amplamente consolidada no meio académico. Para além de assegurar qualidade tipográfica elevada, *LaTeX* permite a gestão consistente de referências bibliográficas, figuras, tabelas e fórmulas matemáticas, promovendo rigor e uniformidade na comunicação científica. Guias clássicos, como [10], destacam a relevância desta ferramenta na produção de documentação técnica estruturada, especialmente em projetos de caráter científico e académico.

2.3 Trabalhos Relacionados

Diversos trabalhos e propostas pedagógicas abordam a introdução à programação através da manipulação de vetores, estruturas de controlo e interação com o utilizador em ambientes simples. Estas abordagens partilham objetivos educativos semelhantes ao do presente projeto, nomeadamente a consolidação de conceitos fundamentais de programação e o desenvolvimento do raciocínio algorítmico. No entanto, diferenciam-se quanto ao grau de integração das funcionalidades, à complexidade das operações implementadas e à organização global do sistema.

Um trabalho amplamente reconhecido neste contexto é apresentado por Downey em *Think Python* [3]. Nesta obra, o autor propõe uma abordagem progressiva à aprendizagem da programação, centrada na compreensão gradual de conceitos como listas, ciclos, funções e validação de entradas. À semelhança do projeto desenvolvido, é atribuída elevada importância à clareza do código e à decomposição do problema em funções bem definidas. Contudo, os exemplos apresentados por Downey são, maioritariamente, organizados sob a forma de exercícios independentes, não sendo explorada a integração sistemática de múltiplas operações num único programa orientado por um menu interativo.

Outro trabalho relevante é o de Zelle, *Python Programming: An Introduction to Computer Science* [14], que aborda o desenvolvimento de aplicações interativas simples com controlo explícito do fluxo de execução. Tal como no presente projeto, o autor valoriza a utilização de interfaces textuais em ambiente de CLI e a validação rigorosa das entradas do utilizador. A principal diferença reside no facto de os exemplos apresentados privilegiarem pequenos programas isolados, enquanto o trabalho descrito neste relatório integra operações sobre vetores e matrizes num único sistema coerente, acessível através de um menu central.

De forma a sintetizar as semelhanças e diferenças entre os trabalhos analisados e o projeto desenvolvido, a Tabela 2.1 apresenta uma comparação direta baseada em critérios relevantes, como o tipo de interação, a integração funcional e a utilização de estruturas de dados matriciais.

Tabela 2.1: Comparação entre trabalhos relacionados e o projeto desenvolvido, considerando critérios de interação, integração funcional e manipulação de estruturas de dados.

Critério	**Think Python** [3]	**Zelle** [14]	**Projeto Atual**
Programação introdutória em *Python*	Sim	Sim	Sim
Manipulação de vetores	Sim	Sim	Sim
Construção de matrizes	Não	Parcial	Sim
Menu interativo em CLI	Não	Parcial	Sim
Integração num único programa	Não	Não	Sim
Documentação estruturada	Sim	Sim	Sim

2.4 Conclusões

Neste capítulo apresentou-se o enquadramento teórico do projeto, destacando conceitos essenciais da programação introdutória: vetores, matrizes, interação em linha de comandos e documentação técnica.

A revisão do estado da arte confirmou a relevância de Python, das estruturas de dados simples e das boas práticas de SE, bem como a utilidade das interfaces textuais em CLI.

Relativamente aos trabalhos relacionados, as abordagens de Downey e Zelle influenciaram a clareza do código e a validação de entradas. O contributo distintivo deste projeto reside na integração, num único sistema interativo, de várias operações sobre vetores e matrizes através de um menu central.

Capítulo

3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

Este capítulo descreve o planeamento e as principais decisões de engenharia de software que orientaram o desenvolvimento do projeto, explicando de forma estruturada a análise e organização do problema ao longo da implementação.

Inicialmente, são abordados os princípios aplicados, incluindo a definição de requisitos, a organização modular do sistema e o fluxo de interação com o utilizador. De seguida, são apresentadas e justificadas as tecnologias utilizadas, evidenciando a forma como foram articuladas para cumprir os objetivos do trabalho com simplicidade e robustez.

3.2 Engenharia de Software

Esta secção descreve o enquadramento de SE adotado no desenvolvimento do programa de processamento de vetores e matrizes. Para além da enumeração das funcionalidades implementadas, é apresentada uma visão estruturada dos requisitos do sistema e da forma como estes se refletem na organização do código e no fluxo de execução do programa.

A abordagem seguida procura demonstrar que a solução não foi construída de forma improvisada, mas sim através de uma análise prévia dos requisitos e de uma separação lógica clara entre diferentes responsabilidades do sistema. Em particular, distingue-se a camada de interação em CLI, responsável

pela comunicação com o utilizador, da lógica de processamento numérico e da representação dos dados sob a forma de vetores e matrizes.

Requisitos Funcionais

Os requisitos funcionais definem as operações que o sistema deve disponibilizar ao utilizador. No contexto deste trabalho, estes requisitos estão diretamente relacionados com as opções disponibilizadas no menu principal do programa e com as funções correspondentes implementadas no código.

Em concreto, o sistema deve permitir a leitura e validação de dados introduzidos pelo utilizador, a execução de diversos cálculos sobre o vetor inicial, a construção de matrizes derivadas e a apresentação clara dos resultados. Adicionalmente, deve oferecer mecanismos de ajuda e permitir uma terminação controlada da execução. Estes objetivos materializam-se nos seguintes requisitos funcionais:

- **RF_01** – Ler exatamente dez valores inteiros introduzidos pelo utilizador, garantindo que todos pertencem ao intervalo entre -10 e 27, com validação rigorosa de entrada;
- **RF_02** – Determinar o elemento do vetor que se encontra mais próximo de um valor de referência;
- **RF_03** – Calcular o cosseno dos elementos pertencentes à segunda metade do vetor;
- **RF_04** – Reordenar, filtrar e calcular medidas estatísticas simples sobre o vetor;
- **RF_05** – Construir matrizes derivadas do vetor inicial, incluindo matrizes duas por dez, dez por dez e a respetiva transposta;
- **RF_06** – Ler um novo vetor e calcular a soma do vetor inicial com o dobro dos seus valores;
- **RF_07** – Identificar números compostos no vetor inicial;
- **RF_08** – Disponibilizar informação de ajuda e permitir a terminação explícita do programa.

Requisitos Não Funcionais

Para além da funcionalidade, foram igualmente considerados requisitos não funcionais associados à qualidade, robustez e manutenibilidade do sistema. Estes requisitos influenciam diretamente as decisões de implementação e a forma como o programa interage com o utilizador.

Foi dada especial atenção à clareza das mensagens apresentadas, à validação de erros e à organização modular do código, de modo a facilitar tanto a utilização do programa como a sua futura manutenção ou extensão. Destacam-se os seguintes requisitos não funcionais:

- **RNF_01** – Robustez na validação das entradas introduzidas pelo utilizador, evitando falhas de execução por dados inválidos;
- **RNF_02** – Clareza e consistência das mensagens apresentadas em CLI, guiando o utilizador ao longo das diferentes operações;
- **RNF_03** – Modularidade do código, com cada funcionalidade encapsulada numa função independente;
- **RNF_04** – Documentação interna clara, através de comentários e *docstrings*, facilitando a compreensão do código;
- **RNF_05** – Portabilidade do programa para diferentes sistemas operativos, recorrendo exclusivamente à biblioteca *standard* de *Python*.

Diagrama de Casos de Uso

A Figura 3.1 apresenta o diagrama de casos de uso do sistema, no qual se evidenciam as principais interações entre o utilizador e as funcionalidades disponibilizadas pelo programa. Cada caso de uso corresponde a um conjunto de operações acessíveis através do menu principal, permitindo estabelecer uma relação direta entre as ações do utilizador e os requisitos funcionais definidos.

Este diagrama facilita a compreensão global do sistema, sintetizando de forma visual as capacidades oferecidas e o papel central do utilizador na ativação das diferentes funcionalidades. Na Figura 3.1 apresentam-se os principais casos de uso do sistema desenvolvido.

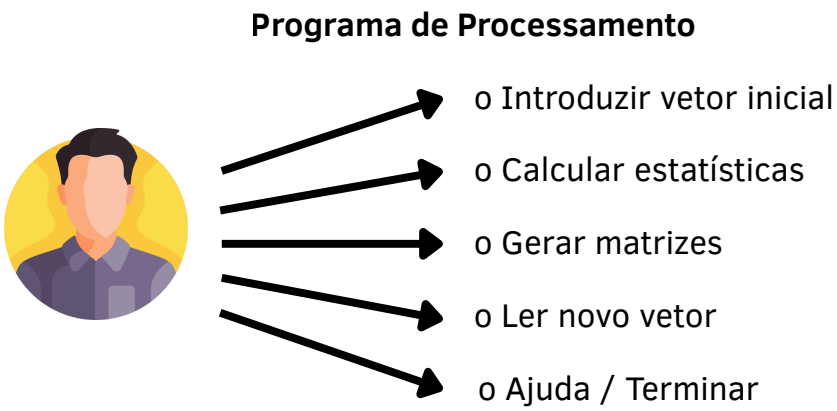

Figura 3.1: Diagrama de casos de uso do sistema desenvolvido, ilustrando as principais funcionalidades disponibilizadas ao utilizador e as respetivas interações com o programa.

Diagrama de Atividade do Ciclo Principal

O diagrama de atividade apresentado na Figura 3.2 representa o fluxo de execução do programa, desde a leitura inicial do vetor até à seleção repetida das opções do menu e à terminação da execução.

Este diagrama ilustra a natureza iterativa do programa, mostrando claramente o ciclo principal controlado pela função `init_program()`, bem como a distinção entre a execução de uma operação e a decisão de terminar o programa. A utilização deste tipo de diagrama contribui para uma melhor compreensão do comportamento dinâmico do sistema.

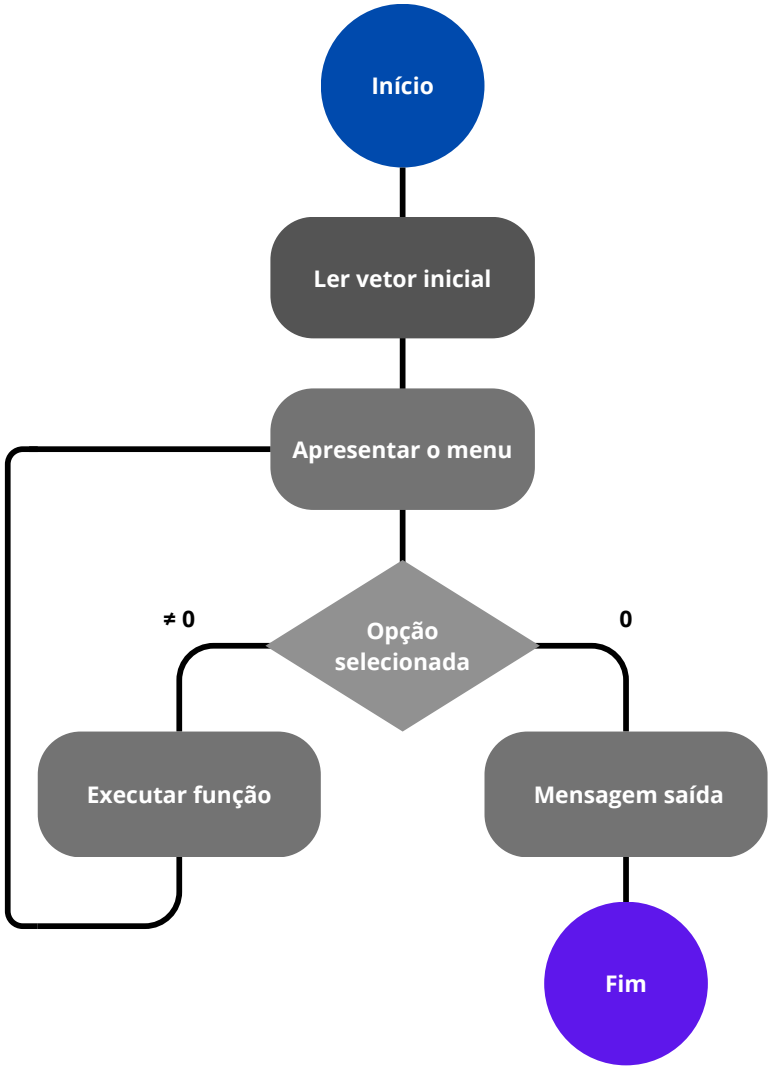

Figura 3.2: Diagrama de atividade do ciclo principal do programa, representando o fluxo de execução desde a leitura do vetor inicial até à seleção repetida das opções do menu e à terminação da execução.

3.3 Tecnologias Importantes

O desenvolvimento deste trabalho recorreu a um conjunto reduzido, mas cuidadosamente selecionado, de tecnologias que suportaram tanto a implementação do programa em *Python* como a organização, documentação e gestão do projeto ao longo do semestre. As ferramentas adotadas são amplamente utilizadas no contexto académico e encontram-se alinhadas com os objetivos

da unidade curricular Laboratórios de Informática.

De forma geral, as escolhas efetuadas privilegiaram soluções que permitissem concentrar o esforço no raciocínio algorítmico, na correta manipulação de estruturas de dados e na clareza da interação com o utilizador. Procurou-se, assim, evitar dependências excessivas ou tecnologias desproporcionadas à dimensão e natureza do projeto.

Linguagem de Programação *Python*

A implementação do programa foi realizada em *Python*, uma das linguagens mais utilizadas no ensino introdutório da programação, particularmente nas áreas de IA e CD. A sua sintaxe simples e expressiva facilita a leitura do código e a tradução direta dos algoritmos para a implementação, aspeto especialmente relevante num contexto pedagógico [7].

Ao longo deste trabalho, todas as funcionalidades foram desenvolvidas recorrendo exclusivamente à biblioteca *standard* de *Python*, nomeadamente listas, ciclos, funções e módulos básicos. Esta opção contribuiu para uma estrutura modular clara, em que cada operação sobre vetores ou matrizes corresponde a uma função bem definida, facilitando os testes, a depuração e a manutenção futura do código.

Ambiente de Execução e Gestão de Dependências

Para garantir consistência no ambiente de execução entre os diferentes elementos do grupo, foi utilizada a distribuição *Anaconda*. Esta ferramenta simplifica a instalação e gestão do interpretador *Python*, assegurando que o programa pode ser executado sem conflitos de versões ou dependências externas.

A utilização de um ambiente controlado revelou-se particularmente útil durante as fases de teste e validação, permitindo reproduzir os mesmos resultados em diferentes sistemas operativos. Para além disso, esta opção está alinhada com boas práticas de portabilidade, uma vez que o programa não depende de bibliotecas adicionais à instalação base de *Python* [1].

Ambientes de Desenvolvimento

Durante o desenvolvimento do código foram utilizados dois *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)s (IDEs): *Visual Studio Code* e *PyCharm*. O *Visual Studio Code* destacou-se pela sua leveza e pela integração eficiente com o terminal e com sistemas de controlo de versões, permitindo ciclos rápidos de edição e execução do programa [8].

O *PyCharm*, por sua vez, disponibilizou funcionalidades adicionais de apoio ao desenvolvimento em *Python*, como inspeções automáticas, sugestões de melhoria e ferramentas de depuração. A utilização combinada destes ambientes contribuiu para a deteção precoce de erros e para o aumento progressivo da qualidade do código produzido [6].

Controlo de Versões

A gestão do código-fonte foi realizada com recurso ao sistema de controlo de versões *Git*, utilizando um repositório alojado no *GitHub*. Esta abordagem permitiu manter um histórico organizado das alterações efetuadas, identificar o contributo de cada elemento do grupo e facilitar a integração das diferentes partes do código [2, 4].

O uso regular de *commits* possibilitou acompanhar a evolução do projeto ao longo do tempo e reverter alterações sempre que necessário, reduzindo o risco de perda de trabalho e promovendo uma colaboração mais estruturada.

Documentação Técnica em LaTeX

A elaboração do relatório técnico foi realizada em LaTeX, uma ferramenta amplamente utilizada no meio académico para a produção de documentação científica e técnica [10]. Esta escolha permitiu assegurar consistência gráfica, correta gestão de referências bibliográficas e uma estrutura clara e bem organizada do documento.

A edição colaborativa foi suportada pela plataforma *Overleaf*, que possibilitou a escrita simultânea do relatório, a compilação automática e a manutenção de versões sucessivas do documento [11]. Esta solução facilitou a articulação entre o desenvolvimento do código e a sua documentação, garantindo que o relatório refletisse corretamente o estado final do projeto.

Documentação de Código com *Doxygen*

Para a documentação técnica do código-fonte foi utilizada a ferramenta *Doxygen*, amplamente reconhecida no contexto da engenharia de software para a geração automática de documentação a partir de comentários estruturados no código [5]. Esta ferramenta permitiu documentar de forma sistemática as funções implementadas, especificando o seu objetivo, parâmetros de entrada, valores de retorno e comportamento geral.

No contexto deste projeto, o uso do *Doxygen* incentivou boas práticas de escrita de código, promovendo comentários claros e consistentes diretamente

associados à implementação. A documentação gerada facilita ainda a compreensão do funcionamento interno do programa, tanto para os autores como para terceiros, reforçando os requisitos não funcionais relacionados com legibilidade, manutenção e extensibilidade futura do sistema.

Articulação entre as Tecnologias

As tecnologias descritas foram utilizadas de forma integrada ao longo de todo o trabalho. O código foi desenvolvido e testado localmente em *Python*, num ambiente gerido por *Anaconda* e editado através dos IDEs selecionados. Em paralelo, o código era versionado com *Git* e sincronizado através do *GitHub*, permitindo um trabalho colaborativo organizado e rastreável.

Em simultâneo, o relatório foi elaborado em LaTeX no *Overleaf*, acompanhando a evolução do projeto e documentando de forma progressiva as decisões tomadas. Esta articulação entre implementação, controlo de versões, geração automática de documentação com *Doxygen* e produção do relatório contribuiu para um processo de desenvolvimento coerente, alinhado com os princípios fundamentais de SE.

De forma a sintetizar as tecnologias utilizadas e o seu papel no desenvolvimento do projeto, a Tabela 3.1 apresenta uma visão comparativa entre as ferramentas adotadas, a sua finalidade principal e o respetivo contributo para o cumprimento dos requisitos definidos.

Tabela 3.1: Principais tecnologias utilizadas no desenvolvimento do projeto, indicando a sua finalidade e o respetivo papel na implementação do sistema.

Tecnologia	**Finalidade principal**	**Integração no projeto**
Python	Linguagem de programação utilizada para implementar todas as funcionalidades de leitura, processamento numérico e geração de matrizes.	Implementação do ficheiro `vector_operations.py` e das funções de manipulação de vetores e matrizes.
Anaconda	Gestão de ambientes de execução em CLI, garantindo versões consistentes do interpretador e das bibliotecas.	Criação de um ambiente isolado para executar e testar o programa em diferentes máquinas.
Visual Studio Code / PyCharm	IDEs usados para edição, depuração e organização do código-fonte.	Suporte ao desenvolvimento diário, inspeção de erros e execução rápida do programa.
LaTeX / Overleaf	Ferramentas de produção do relatório técnico com gestão de secções, figuras, tabelas e referências.	Escrita colaborativa do relatório de projeto, seguindo o template oficial fornecido pela UBI.
Git / GitHub	Controlo de versões distribuído e alojamento remoto do repositório.	Registo histórico das alterações, coordenação do trabalho em grupo e partilha do código final.
Doxygen	Geração automática de documentação a partir de comentários estruturados no código.	Documentação sistemática das funções, parâmetros e valores de retorno, reforçando a legibilidade e a manutenção futura.

3.4 Conclusões

Este capítulo apresentou o planeamento e as principais decisões de engenharia de software que orientaram o desenvolvimento do projeto. A definição dos requisitos funcionais e não funcionais permitiu estruturar a implementação de forma clara e coerente.

A opção por uma arquitetura simples — separando interação em CLI, lógica de processamento e estruturas de dados — facilitou a leitura, depuração e evolução do código. As tecnologias utilizadas, como *Python*, IDEs e controlo de versões, asseguraram um desenvolvimento organizado, enquanto a documentação em *Doxygen* e LaTeX reforçou a clareza e manutenção do projeto.

No conjunto, este planeamento fornece a base essencial para compreender a implementação e a análise apresentada nos capítulos seguintes.

Capítulo

4

Execução e Desenvolvimento

4.1 Introdução

Depois de apresentados o enquadramento e as tecnologias utilizadas, este capítulo descreve as condições de execução do programa e a forma como o utilizador interage com ele no dia a dia. O foco está na perspetiva prática: o ambiente necessário, o modo de lançamento em linha de comandos e o comportamento geral durante a execução.

O objetivo é mostrar como o sistema foi concebido para ser simples de usar, consistente em diferentes máquinas e alinhado com os requisitos funcionais e não funcionais definidos anteriormente, sem repetir os detalhes internos da implementação.

4.2 Ambiente de Execução e Dependências

O programa foi concebido para correr em CLI, recorrendo apenas à linguagem *Python* e a bibliotecas incluídas na instalação *standard*. Desta forma, evita-se a necessidade de instalar pacotes externos e reduz-se a complexidade de configuração.

Em termos práticos, os requisitos mínimos para executar o programa são:

- um interpretador *Python* versão 3.8 ou superior;
- acesso ao módulo `math`, utilizado para cálculos trigonométricos (por exemplo, o cosseno dos elementos do vetor);
- acesso ao módulo `sys`, usado para tratar argumentos passados na linha de comandos (nomeadamente a flag `-help`).

Durante o desenvolvimento foi utilizado um ambiente gerido por *Anaconda*, o que permitiu manter as mesmas versões de *Python* entre os elementos do grupo e testar o programa em diferentes sistemas operativos (Windows, macOS e distribuições *Linux*) sem alterações no código.

4.3 Execução do Programa em Linha de Comandos

Toda a interação com o programa é feita em linha de comandos: o utilizador escreve instruções, o sistema interpreta-as e devolve o resultado no próprio terminal. Este tipo de interface é particularmente adequado ao contexto da unidade curricular, pois reforça a ligação entre o código escrito e a sua execução direta no sistema operativo.

De forma resumida, o fluxo de utilização é o seguinte:

1. o utilizador executa o ficheiro `vector_operations.py`;
2. o programa lê um vetor inicial de dez valores inteiros, validando cada entrada;
3. é apresentado um menu com as diferentes operações disponíveis;
4. a opção escolhida é executada e o resultado é mostrado no ecrã;
5. o programa regressa ao menu até o utilizador selecionar a opção para terminar.

Como suporte visual ao modelo de execução adotado neste trabalho, a Figura 4.1 apresenta um esquema conceptual que ilustra a interação típica entre o utilizador, a interface de linha de comandos e o objeto computacional. Esta representação foi extraída de *Java, Java, Java: Object-Oriented Problem Solving* [9], obra de referência no ensino introdutório da computação, e é utilizada neste contexto por descrever de forma clara e direta o fluxo de entrada e saída de dados em aplicações executadas em ambiente CLI.

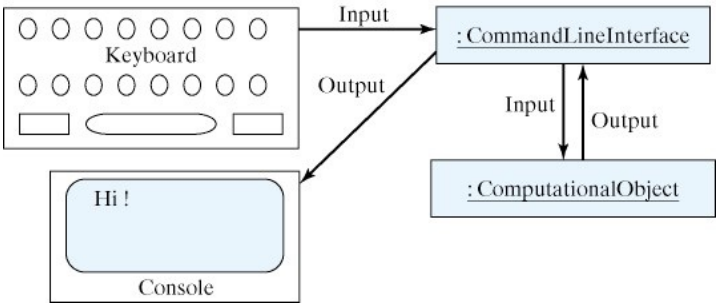

Figura 4.1: Modelo conceptual da interação entre o utilizador, a interface de linha de comandos (CLI) e o processamento interno da aplicação, ilustrando o fluxo de entrada e saída de dados (adaptado de [9]).

4.4 Procedimento de Execução

A execução do programa não exige um processo de instalação complexo. Basta que o ficheiro `vector_operations.py` esteja disponível numa pasta acessível a partir do terminal.

O lançamento normal do programa é feito com o seguinte comando:

```
python vector_operations.py
```

Excerto de Código 4.1: Execução do programa a partir da linha de comandos

Ao executar este comando, o programa inicia a leitura do vetor inicial e, de seguida, apresenta o menu com as opções numeradas de `1` a `11`, bem como a opção `0` para terminar.

Caso o utilizador queira apenas consultar a mensagem de ajuda geral, sem introduzir quaisquer dados, pode utilizar a flag `-help`:

```
python vector_operations.py --help
```

Excerto de Código 4.2: Execução do programa em modo de ajuda

Neste modo, o programa apresenta no terminal uma breve descrição do seu objetivo e das funcionalidades disponíveis, terminando de seguida. Este comportamento permite esclarecer rapidamente o utilizador antes da execução completa do sistema.

4.5 Comportamento Durante a Execução

Durante a execução normal, todos os dados são mantidos em memória, sob a forma de listas (vetores) e listas de listas (matrizes). O programa não guarda

informação em ficheiros nem utiliza bases de dados, uma vez que o objetivo é proporcionar um uso interativo e imediato.

A validação de entrada é um dos aspetos centrais do comportamento do programa:

- se o utilizador introduzir um valor fora do intervalo permitido (−10 a 27), é apresentada uma mensagem de erro e o mesmo índice do vetor é pedido novamente;
- se for introduzido um carácter não numérico, a conversão para inteiro gera uma exceção, que é tratada com um bloco `try/except`, voltando a solicitar o valor em falta;
- o menu só avança para a operação correspondente quando a opção escolhida é válida; caso contrário, é apresentada uma mensagem de aviso e o menu é repetido.

Estas verificações garantem que todas as operações matemáticas são executadas sobre dados consistentes, evitando erros de execução e resultados inesperados. Além disso, o programa procura manter mensagens claras e coerentes, em linha com os requisitos de usabilidade discutidos no capítulo de planeamento.

4.6 Conclusão

Neste capítulo foi descrito o enquadramento prático da execução do programa, incluindo o ambiente necessário, o modo de lançamento em CLI e o comportamento esperado durante a utilização. A combinação de poucas dependências, comandos simples e validação rigorosa das entradas assegura um sistema fácil de instalar e utilizar.

A execução do programa reflete as decisões de planeamento tomadas nas fases anteriores, garantindo coerência entre a conceção do sistema e a experiência efetiva do utilizador.

Capítulo

5

Testes e Validação

5.1 Introdução

Depois de definido o enquadramento conceptual e descrito o processo de desenvolvimento do programa, torna-se necessário explicar como foi verificado se o sistema se comporta realmente como esperado. Este capítulo foca-se, por isso, na fase de *Quality Assurance* (Garantia de Qualidade) (QA), clarificando que tipos de testes foram realizados, como foram executados e de que forma os resultados levaram a pequenos ajustes no código.

Ao longo das secções seguintes são descritos os cenários de teste utilizados, o procedimento seguido na linha de comandos e os principais problemas detetados. O objetivo não é esgotar todas as possibilidades de teste, mas mostrar que o programa foi validado de forma sistemática e coerente com os requisitos apresentados nos capítulos anteriores.

5.2 Especificação dos Testes

A estratégia de testes foi pensada para cobrir os aspetos mais críticos do programa: validação das entradas, correção dos cálculos numéricos e comportamento global do menu interativo. Em vez de recorrer a testes automáticos, optou-se por uma bateria de testes manuais estruturados, adequada à escala reduzida do projeto, mas ainda assim inspirada em práticas básicas de QA.

Foram definidos cinco grupos principais de testes:

- **Testes de limites** — verificação do comportamento do programa quando o utilizador introduz valores no limite do intervalo permitido (-10 e 27) ou sequências de valores repetidos;

- **Testes de robustez** — introdução intencional de entradas inválidas (letras, símbolos, números fora do intervalo) para confirmar se a validação impede que sejam aceites;
- **Testes matemáticos** — comparação manual dos resultados obtidos para médias, cossenos, matrizes de produtos e identificação de números compostos;
- **Testes funcionais** — execução completa do menu, escolhendo cada opção pelo menos uma vez e verificando se o programa regressa corretamente ao menu principal;
- **Testes ponta-a-ponta (*End-to-End* (Testes de Ponta-a-Ponta) (E2E))** — simulação de uma utilização "normal" do programa, desde a leitura do vetor inicial até à terminação, sem interrupções intermédias.

Cada grupo de testes foi associado aos requisitos funcionais mais relevantes, garantindo que todas as funcionalidades descritas anteriormente eram abrangidas pelo plano de testes.

5.3 Execução dos Testes

Os testes foram realizados diretamente na CLI, executando o programa a partir do ficheiro `vector_operations.py`. Para facilitar a repetição dos cenários, foram utilizados vetores de exemplo previamente escolhidos, com valores que permitiam observar claramente os efeitos de cada operação.

Um cenário típico de teste começava com a introdução de um vetor de dez valores mistos, por exemplo:

```
# Valores introduzidos pelo utilizador na CLI:
[ -10, -3, 0, 4, 6, 9, 15, 18, 21, 27 ]
```

Excerto de Código 5.1: Exemplo de vetor utilizado durante os testes

Com este vetor foi possível:

- verificar o cálculo do valor mais próximo de 15;
- confirmar o resultado da média;
- observar o comportamento da filtragem de valores não divisíveis por três;
- analisar a construção da matriz 10×10 de produtos.

Para os testes de robustez, foram introduzidas entradas como `"abc"`, `40` ou `-999`. Em todos os casos, o programa respondeu com uma mensagem de erro clara e voltou a pedir o mesmo valor, sem avançar para a posição seguinte do vetor, tal como planeado.

Os testes E2E consistiram em executar uma sessão completa: leitura do vetor, utilização de várias opções do menu (por exemplo, 1, 3, 5 e 10) e terminação através da opção `0`. Este tipo de teste permitiu confirmar que o fluxo global de interação é estável e que o programa não fica preso em ciclos inesperados.

5.4 Análise dos Resultados e Aprimoramento

Os testes não serviram apenas para confirmar que o programa funcionava, mas também para identificar pontos a melhorar. Durante as primeiras execuções foram detetados alguns problemas:

- entradas inválidas estavam a ser contabilizadas para o tamanho total do vetor, o que foi corrigido com um bloco `try/except` que repete a leitura sem avançar o índice;
- a função de identificação de números compostos não considerava corretamente certos valores pequenos; o algoritmo foi ajustado para ignorar números inferiores a 2 e testar divisores até à raiz quadrada;
- a apresentação das matrizes era pouco legível, especialmente para valores com mais dígitos; a função `mostrar_matriz()` passou a utilizar formatação alinhada;
- algumas mensagens do menu eram pouco claras em relação ao intervalo permitido para os valores; o texto foi revisto para ser mais explícito.

Depois destas correções, os testes foram repetidos, confirmando que os problemas tinham sido resolvidos. A experiência mostrou, na prática, a importância de uma abordagem iterativa: testar, observar, corrigir e voltar a testar. Mesmo num projeto pequeno, esta rotina aproxima o processo de um ciclo simples de *Boundary Value Analysis* (Análise de Valores Limite) (BVA) e validação contínua.

Para sintetizar o plano de testes e as melhorias introduzidas, a Tabela 5.1 apresenta um resumo dos grupos de testes definidos, do objetivo de cada um e de exemplos concretos utilizados.

Tabela 5.1: Resumo dos principais tipos de testes realizados durante a fase de validação do programa, incluindo os respetivos objetivos e exemplos de cenários utilizados.

Tipo de teste	**Objetivo principal**	**Exemplo de cenário**
Limites	Verificar o comportamento para valores extremos do intervalo permitido.	Vetor contendo -10 e 27.
Robustez	Garantir rejeição de entradas inválidas.	Introdução de letras ou valores fora do intervalo.
Matemáticos	Confirmar a correção dos cálculos efetuados.	Validação manual de médias e matrizes.
Funcionais	Avaliar o correto funcionamento das opções do menu.	Execução sequencial das opções disponíveis.
E2E	Avaliar o comportamento global do sistema.	Sessão completa desde a leitura até à terminação.

5.5 Conclusão

A fase de testes e validação permitiu confirmar que o programa cumpre os requisitos definidos para leitura, processamento e apresentação de resultados sobre vetores e matrizes. Os diferentes tipos de testes mostraram que o sistema reage de forma robusta a entradas inválidas, produz resultados numéricos consistentes e mantém um fluxo de interação previsível na CLI.

Mais do que um passo final, os testes funcionaram como uma extensão natural do processo de desenvolvimento: cada problema encontrado levou a pequenas melhorias no código e na interface, reforçando a qualidade global do projeto. Este capítulo completa, assim, a visão sobre o funcionamento prático do sistema e prepara o enquadramento para a discussão global de resultados e trabalho futuro.

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

A realização deste projeto permitiu atingir integralmente os objetivos definidos no enunciado da unidade curricular. A implementação do programa em *Python*, com suporte a operações sobre vetores e matrizes, validação rigorosa de entradas e um menu interativo em CLI, demonstra de forma concreta que os requisitos funcionais e não funcionais foram cumpridos com sucesso. A execução consistente do programa em diferentes cenários de teste, bem como a ausência de erros de execução após a fase de validação, constituem uma evidência clara desse cumprimento.

Uma das principais conclusões retiradas deste trabalho é a importância de planear a estrutura do programa antes da sua implementação. A divisão do código em funções independentes, cada uma com uma responsabilidade bem definida, revelou-se fundamental para manter a clareza, facilitar a depuração de erros e permitir uma evolução incremental do sistema. Esta abordagem evidenciou, na prática, conceitos básicos de engenharia de software que serão essenciais ao longo do percurso académico e profissional.

Do ponto de vista técnico, o trabalho com vetores e matrizes contribuiu para consolidar competências de raciocínio lógico e matemático. A construção de matrizes derivadas do vetor inicial, bem como o cálculo de produtos, transpostas e medidas simples, exigiu um controlo rigoroso de índices e uma atenção constante à correção dos algoritmos. Estes aspetos reforçaram a compreensão de como estruturas de dados simples podem dar origem a operações mais complexas quando bem combinadas.

Outro aspeto relevante foi o papel da validação de entradas e do tratamento de erros. A necessidade de impedir que valores inválidos comprome-

tessem a execução do programa mostrou claramente que a robustez não é um detalhe opcional, mas uma condição essencial para a fiabilidade do software. A utilização de mecanismos como `try/except` permitiu consolidar conceitos fundamentais da linguagem *Python* e compreender melhor a interação entre o sistema e o utilizador.

É também importante destacar a experiência adquirida na documentação do código com *Doxygen*. A escrita de comentários estruturados, associados diretamente às funções implementadas, reforçou a disciplina na programação e promoveu uma maior consciência sobre a necessidade de tornar o código compreensível para terceiros. A documentação automática gerada demonstrou que um esforço relativamente pequeno durante a implementação pode resultar num recurso valioso para manutenção futura, reutilização do código e trabalho colaborativo.

De forma global, este projeto constituiu uma primeira experiência completa de desenvolvimento de software, desde a conceção de uma solução até à sua validação e documentação. Para além dos conhecimentos técnicos adquiridos, permitiu desenvolver uma visão mais clara sobre a importância da organização, do rigor e do método no processo de programação.

6.2 Trabalho Futuro

Apesar de o programa desenvolvido ser funcional e cumprir todos os objetivos propostos, existem vários aspetos que ficaram por fazer, sobretudo devido ao âmbito e às limitações naturais do projeto. A interface de interação, por exemplo, é exclusivamente textual e baseada em menus na CLI. Esta opção foi deliberada, uma vez que o foco do trabalho estava na lógica de programação e na manipulação de estruturas de dados, e não no desenvolvimento de interfaces gráficas, que exigiria tecnologias e conhecimentos adicionais.

Uma evolução natural do trabalho seria a implementação de uma interface gráfica, recorrendo a bibliotecas como *Tkinter* ou *PyQt*. Esta melhoria permitiria tornar a interação mais intuitiva e apresentar os resultados — particularmente as matrizes — de forma mais visual e organizada, facilitando a interpretação por utilizadores menos experientes.

Seria igualmente interessante alargar o conjunto de operações matemáticas disponíveis. No contexto dos vetores, poderiam ser integradas medidas estatísticas mais avançadas, como variância, desvio padrão, normalização ou deteção de valores extremos. Relativamente às matrizes, poderiam ser adicionadas operações como cálculo de determinantes, inversão ou decomposições simples. No entanto, estas extensões não foram incluídas por irem além dos objetivos definidos para este trabalho introdutório.

Outra possibilidade de desenvolvimento futuro passa pela generalização do programa para vetores de dimensão variável ou pela leitura de dados a partir de ficheiros externos. A exportação dos resultados para formatos como `.csv` ou `.json` permitiria integrar o programa com outras ferramentas de análise ou visualização de dados, aproximando-o de cenários reais encontrados nas áreas de IA e CD.

Por fim, o trabalho aqui apresentado pode ter aplicações interessantes noutros contextos educativos, nomeadamente como ferramenta de apoio ao ensino introdutório de programação, álgebra linear ou análise de dados. A sua estrutura simples e modular torna-o adequado como base para exercícios práticos, projetos incrementais ou demonstrações de conceitos fundamentais, servindo como ponto de partida para desenvolvimentos mais avançados em unidades curriculares futuras.

Bibliografia

[1] Anaconda, Inc. *Anaconda Documentation*. [Online] https://docs.anaconda.com/. 2025.

[2] Scott Chacon e Ben Straub. *Pro Git*. 2nd. Apress, 2014.

[3] Allen B. Downey. *Think Python: How to Think Like a Computer Scientist*. 2nd. O'Reilly Media, 2015. ISBN: 978-1491939369.

[4] GitHub. *GitHub Documentation*. [Online] https://docs.github.com/. 2025.

[5] Dimitri van Heesch. *Doxygen: Source Code Documentation Generator*. Versão 1.9.x. Disponível em https://www.doxygen.nl/manual/. 2025.

[6] JetBrains. *PyCharm Documentation*. [Online] https://www.jetbrains.com/pycharm/documentation/. 2025.

[7] Mark Lutz. *Learning Python*. 5th. O'Reilly Media, 2013. ISBN: 978-1449355739.

[8] Microsoft. *Visual Studio Code Documentation*. [Online] https://code.visualstudio.com/docs. 2025.

[9] Ralph Morelli et al. *Java, Java, Java: Object-Oriented Problem Solving*. 4th. Recurso online. Runestone Academy, 2024. URL: https://runestone.academy/ns/books/published/javajavajava/index.html.

[10] Tobias Oetiker et al. *The Not So Short Introduction to LaTeX*. [Online] https://tobi.oetiker.ch/lshort/lshort.pdf. Último acesso a 12 de Março de 2019. 2018.

[11] Overleaf. *Overleaf Documentation*. [Online] https://www.overleaf.com/learn. 2025.

[12] William H. Press et al. *Numerical Recipes: The Art of Scientific Computing*. 3rd. Cambridge University Press, 2007. ISBN: 978-0521880688.

[13] Diomidis Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, 2003. ISBN: 978-0201799408.

[14] John M. Zelle. *Python Programming: An Introduction to Computer Science*. 3rd. Franklin, Beedle & Associates, 2016. ISBN: 978-1590282755.