

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

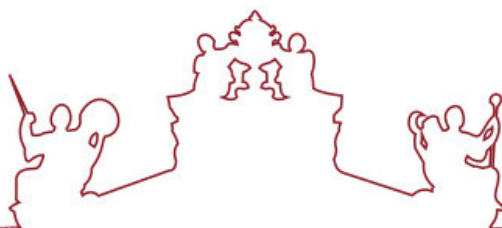
Dissertação

Análise e Visualização de dados em Astrofísica

Tomás da Costa Luís Lima

Orientador(es) | Francisco Manuel Coelho
Miguel de Avillez

Évora 2026



Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

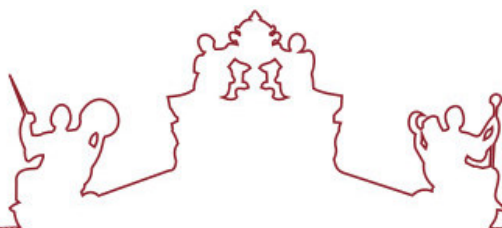
Dissertação

Análise e Visualização de dados em Astrofísica

Tomás da Costa Luís Lima

Orientador(es) | Francisco Manuel Coelho
Miguel de Avillez

Évora 2026



A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente | Lígia Maria Ferreira (Universidade de Évora)

Vogais | Francisco Manuel Coelho (Universidade de Évora) (Orientador)
Rui André Correia Oliveira (Universidade de Évora) (Arguente)

Dedico este trabalho a todos que caminharam comigo, em especial à minha família, cujo apoio foi a base para que eu pudesse perseverar. E dedico também a mim, pelas noites sem sono, pela paciência para começar de novo e pela fé inabalável de que eu conseguiria chegar até o fim

Prefácio

O trabalho desenvolvido nesta dissertação representa a convergência natural do meu percurso acadêmico e dos meus interesses pessoais. A minha formação inicial em Engenharia Eletrônica e Telecomunicações despertou cedo o gosto pelo processamento de sinal, filtragem e análise de dados, áreas nas quais realizei vários projetos aplicados. Mais tarde, na licenciatura em Engenharia Informática, aprofundei esta ligação através do projeto final, dedicado ao processamento de sinais de áudio para a deteção de batimentos cardíacos. Ao longo deste percurso, fui constatando que a programação e a análise de dados, independentemente do domínio científico, constituem um eixo central do meu entusiasmo e da minha motivação profissional.

Foi precisamente essa afinidade que me conduziu ao desafio que deu origem a este trabalho: a modernização de uma ferramenta de análise astrofísica desenvolvida em FORTRAN 2008 e recorrendo à biblioteca PGPLOT¹. Este trabalho baseia-se especificamente na versão do software AstroPGPLOT (de Avillez et al., 2009, 2020), onde em 2020 foi introduzida a extensão das grelhas através de condições de fronteira periódicas.

A transposição para a linguagem Julia e para bibliotecas gráficas modernas como o Makie visa modernizar a infraestrutura de análise de dados, adequando-a aos fluxos científicos atuais que lidam com volumes e complexidade crescentes, enquanto introduz benefícios como interatividade e melhor integração. Assim, o objetivo desta dissertação foi transpor toda a lógica científica existente para a linguagem Julia e para a biblioteca Makie de forma modular e sustentável.

O resultado desse esforço materializou-se no desenvolvimento do pacote AstroTLPlot, que integra a leitura e processamento de dados nos formatos utilizados em astrofísica computacional, a implementação de um novo núcleo de cálculos estatísticos e a criação de mecanismos de visualização adequados a grandes volumes de dados. Esta transição não apenas preservou a eficiência da solução original, como introduziu novas capacidades de interatividade, flexibilidade e integração num fluxo de trabalho científico contemporâneo.

Este projeto permitiu-me, assim, unir o meu interesse pelo processamento e análise de dados à exploração de um domínio científico exigente e inspirador como a astrofísica.

¹PGPLOT Graphics Subroutine Library (T. J. Pearson, 2001). Disponível em: <http://www.astro.caltech.edu/~tjp/pgplot/>

Agradecimentos

À minha família, pelo apoio e compreensão durante todas as fases deste percurso, especialmente nos momentos mais desafiadores.

Aos professores que, ao longo da minha formação académica, me transmitiram conhecimentos e despertaram o interesse pela investigação científica.

Um agradecimento muito especial aos meus orientadores. Agradeço a sua disponibilidade para me orientar, por me terem introduzido a esta temática de investigação e pelos preciosos detalhes técnicos relativos ao Jupyter e PGLOT. Sou particularmente grato ao Professor Miguel Avillez pela cedência do código fonte do AstroPGLOT e dos dados de simulações numéricas (Avillez et al., 2020), que foram fundamentais para a realização deste trabalho. Gostaria igualmente de destacar o Professor Francisco Coelho pelo acompanhamento próximo e orientação permanente ao longo de todo o projecto.

A todos, o meu e profundo agradecimento.

Conteúdo

Lista de Acrónimos	xvii
Sumário	xix
Abstract	xxi
1 Introdução	1
1.1 Contextualização da Pesquisa	1
1.2 Motivação e Justificação	2
1.3 Objetivos	3
1.4 Metodologia	3
1.5 Estrutura da Tese	4
2 Fundamentação Teórica e Ferramentas	5
2.1 Enquadramento Teórico	5
2.2 Ferramentas e Tecnologias Existentes	6
2.3 Evolução e Estrutura dos Formatos HDF4 e HDF5	8
2.4 O Formato YAML em Contexto Científico	9
2.5 VTK (Visualization Toolkit)	9
2.6 Bibliotecas Gráficas para Visualização de Dados Científicos	9
2.6.1 Alternativas Modernas para Visualização Científica	10
2.7 Comparação de Abordagens: FORTRAN/PGPLOT e Julia/Makie	11
2.7.1 FORTRAN vs. Julia	11
2.7.2 PGPLOT vs. Makie.jl	11
3 Transposição e Desenvolvimento do Pacote AstroTLPlot	13
3.1 Linguagens, Bibliotecas e Ambientes de Desenvolvimento	13
3.1.1 Análise do Código-Fonte Original – AstroPGPLOT (FORTRAN 2008 + PGPLOT)	14
3.1.2 Julia – Linguagem Base e Ecossistema de Bibliotecas	15
3.1.3 Jupyter Notebooks/Lab — Exploração e Desenvolvimento Interativo	16
3.1.4 Visual Studio Code – Programação, Documentação e Gestão de Projeto	17
3.2 Modelo de Dados e Formatos de Armazenamento	17
3.2.1 Estrutura e Pré-processamento dos Dados	18
3.2.2 Pré-processamento dos Dados	18
3.2.3 Exemplos de Ficheiros e Formato de Dados	18
3.3 Modernização do Sistema de Configuração: de ASCII para YAML	20
3.3.1 Adoção do Formato YAML e Vantagens Técnicas	20
3.3.2 Estrutura dos Dados de Configuração	21
3.3.3 Conclusão Metodológica	21
3.4 Desenvolvimento do Pacote AstroTLPlot	22

3.4.1	Arquitetura e Organização do Pacote AstroTLPlot	22
3.4.2	Organização de Ficheiros e Diretórios do Pacote	24
3.4.3	Justificação Técnica do Design Modular	24
3.5	Estruturas de Dados Implementadas e Organização	25
3.5.1	Constantes Físicas e Códigos de Estado	26
3.5.2	Estrutura de Configuração de Simulação (ConfigData)	27
3.5.3	Dados de Simulação (SimulationData)	27
3.5.4	Dados Estatísticos (StatisticsData2D)	28
3.5.5	Vantagens da Utilização de Estruturas de Dados no Desenvolvimento do Projeto	29
3.5.6	Estruturas AstroPGPLOT (FORTRAN 2008) vs Estruturas AstroTLPlot (Julia)	29
3.5.7	Comparação e Benefícios da Nova Abordagem	31
3.6	Transposição e Reimplementação de Algoritmos FORTRAN em Julia	31
3.6.1	Implementação em Julia e Principais Desafios	32
3.6.2	Visualização Científica em Julia com Makie	33
3.7	Transposição de Rotinas: Exemplos e Análise Comparativa	33
3.7.1	Comparação entre a leitura de configuração no AstroPGPLOT (FORTRAN) e no AstroTLPlot (Julia)	33
3.7.2	Comparação entre readdata_hdf4 (FORTRAN 2008) e readdata_hdf5 (Julia)	38
3.8	Estratégia de Transposição de Visualização: PGPLOT → Makie	41
3.8.1	Exemplo Ilustrativo da Transposição e Adaptação de Visualização	41
3.9	Geração de Mapas: AstroPGPLOT vs. AstroTLPlot	43
3.10	Novas Funcionalidades na Transposição para Julia/Makie	44
3.10.1	Sistema Unificado de Anotação e Marcação Gráfica	45
3.10.2	Visualização Combinada: Mapas de Calor com Sobreposição de Contornos	47
3.10.3	Visualização Integrada de Múltiplas Representações	49
3.10.4	Visualização Integrada de Estados Iónicos	50
3.11	Expansão do Módulo Estatístico: Funcionalidades Derivadas de statistics_data	51
3.11.1	Principais Características e Benefícios da Função: statistics_data	51
3.11.2	Identificação e Anotação de Extremos	52
3.11.3	Persistência dos Resultados Estatísticos	53
3.12	Novo Sistema de Organização de Saída e Nomenclatura Automática no AstroTLPlot	53
3.12.1	Sistema de Nomenclatura Automática de Ficheiros	54
3.13	Avaliação de Desempenho do AstroTLPlot	56
3.13.1	Validação e Benchmark de Leitura de Ficheiros	57
3.13.2	Verificação de Estatísticas de Matrizes	57
3.13.3	Testes de Visualização Científica	60
3.13.4	Robustez e Tratamento de Exceções	61
3.13.5	Gestão de Erros Estruturada no Pacote AstroTLPlot	61
3.13.6	Vantagens Observadas	61
4	AstroTLPlot: Configurações e Casos de Estudo	63
4.1	Execução e Aplicação Prática do Pacote AstroTLPlot	63
4.1.1	Instalação e Configuração do Pacote	63
4.2	Leitura e Validação de Ficheiros de Configuração e Dados	64
4.2.1	Leitura de Ficheiro de Configuração — Função load_simulation_config	64
4.2.2	Leitura de Ficheiros de Simulação — Função readdata_hdf5	66
4.2.3	Leitura de Ficheiros de Configurações Iónicas — Função ions_read	67
4.3	Análise Estatística de Campos Astrofísicos	68
4.3.1	Situação 1: Cálculo de Estatísticas Globais em Matriz Tridimensional	69

4.3.2	Situação 2: Análise Estatística em Corte Bidimensional	69
4.3.3	Situação 3: Estatísticas em Região Delimitada (Janela Espacial)	70
4.3.4	Situação 4: Aplicação de Transformação Logarítmica aos Dados	71
4.3.5	Integração com a Visualização e Persistência de Dados	72
4.3.6	Considerações Finais	73
4.4	Visualização de Campos com Cortes Planos	73
4.4.1	Situação 1: Mapa de Calor (Heatmap)	73
4.4.2	Situação 2: Mapa de Contorno (Contour Map)	74
4.4.3	Situação 3: Mapa Combinado (Heatmap + Contour)	75
4.4.4	Situação 4: Função de Densidade de Probabilidade (PDF)	76
4.4.5	Análise Comparativa: Visualização de PDFs no AstroPGPLOT vs AstroTLPlot	78
4.4.6	Visualização Integrada de Múltiplas Representações da Densidade	79
4.4.7	Visualização de Estados Iônicos do Hidrogénio	79
4.4.8	Análise de Gradientes Iônicos através de Mapas de Contorno	80
4.4.9	Análise Integrada de Estados Iônicos do Azoto - $N00^+$ a $N07^+$	81
4.4.10	Visualização com Estatísticas Integradas	82
4.4.11	Integração entre Visualização e Análise Quantitativa	83
4.5	Demonstração do Sistema de Nomes e Estrutura de Saída	83
4.6	Documentação e Repositório GitHub	85
4.6.1	Documentação Técnica e Integração com Julia	85
4.6.2	Repositório GitHub e Estrutura do Projeto	86
4.7	Avaliação Comparativa: AstroTLPlot vs. AstroPGPLOT	86
4.8	Considerações Finais	87
5	Conclusões e Trabalhos Futuros	89
5.1	Conclusões e Contribuições	89
5.2	Limitações e Desafios	90
5.3	Trabalhos Futuros	90

Lista de Figuras

2.1	Estrutura do formato de ficheiro HDF5 (Fonte: Adaptado de [NEO24]).	8
3.1	Exemplo da estrutura hierárquica de um ficheiro HDF5 utilizado no projeto.	19
3.2	Arquitetura modular do pacote AstroTLPlot, evidenciando a separação funcional entre os módulos de leitura, processamento, estatística e visualização.	23
3.3	Estrutura modular de diretórios do pacote AstroTLPlot.	24
3.4	Arquitetura das estruturas de dados do pacote AstroTLPlot.	25
3.5	Comparação de desempenho entre as versões nativa e ciclo único da função <code>statistics_data</code>	59
4.1	Exemplo de saída parcial gerada pela função <code>load_simulation_config</code>	65
4.2	Saída parcial formatada obtida com a leitura agregada pela função <code>load_simulation_config</code>	66
4.3	Exemplo de saída parcial gerada pela função <code>readdata_hdf5!</code>	67

4.4	Saída parcial da função <code>print_vect</code> , do dataset <code>simulations_data.den</code> : os primeiros 3 valores.	67
4.5	Exemplo de saída parcial gerada pela função <code>ions_read</code> , e da função <code>print_struct_info(tps)</code>	68
4.6	Mapa de calor de densidade no plano XY, gerado com a função <code>plot_heatmap</code> do <code>AstroTLPlot</code>	74
4.7	Mapa de contorno de densidade no plano XY, obtido com a função <code>plot_cont</code> do <code>AstroTLPlot</code>	75
4.8	Mapa combinado de calor e contorno da densidade no plano XY.	76
4.9	Função de distribuição de probabilidade (PDF) da densidade, gerada com a função <code>plot_pdf_test</code> do pacote <code>AstroTLPlot</code>	77
4.10	Análise comparativa das funções de densidade de probabilidade (PDF) geradas pelo <code>AstroPGPLOT</code> (esquerda) e pelo <code>AstroTLPlot</code> (direita).	78
4.11	Painel integrado reunindo as quatro representações complementares do campo de densidade: mapa de calor (superior direito), mapa de contorno (superior esquerdo), combinação calor+contorno (inferior esquerdo) e função de densidade de probabilidade - PDF (inferior direito).	79
4.12	Distribuição espacial dos estados iónicos do hidrogénio no plano XY: (a) estado neutro (H^0) e (b) estado ionizado (H^{1+}).	80
4.13	Mapas de contorno de densidade dos estados iónicos do hidrogénio: (a) neutro (H^0) e (b) ionizado (H^{1+}).	81
4.14	Painel completo dos estados iónicos do azoto (N^0 a N^{7+}).	82
4.15	Mapa de calor de densidade com estatísticas integradas, gerado pelo pacote <code>AstroTLPlot</code>	83

Lista de Tabelas

2.1	Comparação entre linguagens de programação para computação científica: Julia, Python, FORTRAN e MATLAB.	7
2.2	Comparação entre as bibliotecas <code>PGPLOT</code> e <code>Makie</code>	10
2.3	Comparação entre FORTRAN + <code>PGPLOT</code> e a abordagem moderna Julia + <code>Makie.jl</code>	11
3.1	Excerto da comparação entre a estrutura dos dados de configuração nos formatos <i>ASCII</i> do (<code>AstroPGPLOT</code>) e <i>YAML</i> (<code>AstroTLPlot</code>).	21
3.2	Comparação entre as abordagens de estruturação de dados no <code>AstroPGPLOT</code> (FORTRAN) e no <code>AstroTLPlot</code> (Julia).	31
3.3	Principais desafios na transposição de FORTRAN 2008 para Julia e soluções implementadas.	32
3.4	Análise comparativa dos paradigmas de configuração: o modelo baseado em <code>namelists</code> do <code>AstroPGPLOT</code> (FORTRAN) face à abordagem baseada em <i>YAML</i> e estruturas tipadas do <code>AstroTLPlot</code> (Julia).	37
3.5	Comparação conceptual entre as implementações no <code>AstroPGPLOT</code> (FORTRAN com <code>readdata_hdf4</code>) e no <code>AstroTLPlot</code> (Julia com <code>readdata_hdf5!</code>).	40
3.6	Comparação entre o sistema gráfico do <code>AstroPGPLOT</code> (<code>PGPLOT</code> /FORTRAN) e do <code>AstroTLPlot</code> (<code>Makie</code> /Julia) na inicialização e configuração de gráficos	43
3.7	Convenção de pastas criadas automaticamente pelo pacote <code>AstroTLPlot</code>	53
3.8	Comparação de desempenho entre as versões no (<code>AstroPGPLOT</code>) e Julia (<code>AstroTLPlot</code>) na leitura de ficheiros HDF.	57
3.9	Comparação dos resultados obtidos pelas versões <i>Ciclo Único</i> e <i>Nativa Julia</i>	58

3.10	Resumo de desempenho das três abordagens para uma matriz de dimensão (2000×2000) .	59
3.11	Principais códigos e mensagens de erro definidos no módulo <code>constants.jl</code> .	61
4.1	Resumo comparativo entre AstroPGPLOT (FORTRAN/PGPLOT) e AstroTLPlot (Julia/Makie.jl).	87

List of Listings

3.5.1	Excerto do módulo <code>constants.jl</code>	26
3.5.2	Estrutura de dados <code>ConfigData</code>	27
3.5.3	Estrutura de dados <code>SimulationData</code>	28
3.5.4	Estruturas de dados estatísticos auxiliares.	28
3.5.5	Estrutura do módulo <code>config</code> do AstroPGPLOT em FORTRAN 2008.	30
3.7.1	Excerto da sub-rotina <code>namelists</code> no código-fonte AstroPGPLOT (FORTRAN 2008).	34
3.7.2	Excerto da função genérica <code>open_file</code> em Julia.	35
3.7.3	Função <code>load_simulation_config</code> em Julia.	36
3.7.4	Excerto da sub-rotina <code>readdata_hdf4</code> no código AstroPGPLOT (FORTRAN 2008).	38
3.7.5	Função <code>readdata_hdf5!</code> no pacote AstroTLPlot (Julia).	39
3.8.1	Excerto da rotina de inicialização e configuração da janela gráfica no AstroPGPLOT (FORTRAN 2008 + PGPLOT)	41
3.8.2	Função modular para inicialização e configuração da janela no AstroTLPlot (Julia com <code>Makie.jl</code>).	42
3.9.1	Excerto da rotina <code>mapas</code> em FORTRAN.	44
3.9.2	Assinatura da função <code>maps!</code> no AstroTLPlot (Julia).	44
3.10.1	Rotina <code>copyright</code> no AstroPGPLOT (FORTRAN) para escrita de direitos de autor.	45
3.10.2	Função <code>add_text_at</code> no AstroTLPlot (Julia) para anotação flexível de figuras.	46
3.10.3	Excerto da função <code>plot_heat_cont</code> no AstroTLPlot (Julia), que combina mapa de calor e contornos.	48
3.10.4	Excerto da função <code>mapas_subplot!</code> no AstroTLPlot (Julia).	49
3.10.5	Assinatura da função <code>plot_element_subplot</code> no AstroTLPlot (Julia).	50
3.11.1	Assinatura da função <code>statistics_data</code> no AstroTLPlot (Julia).	51
3.11.2	Excerto da função de <code>plot_heatmap</code> que utiliza dados transformados via <code>statistics_data2D</code> no AstroTLPlot.	52
3.11.3	Extração de valores mínimo e máximo e coordenadas a partir do objeto <code>statistics_data2D</code> no AstroTLPlot.	52
3.11.4	Utilização das coordenadas do máximo para adicionar marcadores no AstroTLPlot.	52
3.11.5	Assinatura da função <code>export_statistics</code> no AstroTLPlot.	53
3.12.1	Excerto da rotina <code>writeplot</code> no AstroPGPLOT (FORTRAN 2008), responsável pela criação de nomes de ficheiros de saída.	55
3.12.2	Assinatura da função <code>writeplot</code> no AstroTLPlot (Julia), evidenciando parâmetros nomeados, formatos de saída flexíveis e automatização da inserção de data e hora nome do ficheiro de saída	55

3.13.1 Assinatura da função <code>statistics_data</code> no AstroTLPlot (Julia).	58
4.2.1 Exemplo em Julia da utilização da função <code>load_simulation_config</code> para leitura de ficheiro de configuração.	64
4.4.1 Chamada da função <code>plot_element_subplot</code> em Julia para o elemento químico com número atómico = 7 (Azoto).	81
4.5.1 Chamada da função <code>writeplot</code> para geração de nomes de ficheiro para densidade em múltiplos formatos.	84
4.5.2 Chamada da função <code>save_or_display</code> para armazenamento físico dos ficheiros.	84
4.6.1 Exemplo de documentação em Julia para a função <code>process_simulation_list</code>	85

Lista de Acrónimos

URL	<i>Uniform Resource Locator</i>
YML/YAML	YAML Ain't Markup Language
HDF5	Hierarchical Data Format version 5
VTk	Visualization Toolkit
CSV	Comma-Separated Values
IDL	Interactive Data Language
MATLAB	MATrix LABoratory
Jupyter	Julia, Python, R
REPL	The Read Evaluate Print Loop
NaN	Not a Number
JIT	Just-In-Time
PGPLOT	Postscript Graph Plotting
GPU	Graphics Processing Unit
LaTeX	Lamport TeX
VSCode	Visual Studio Code
API	Application Programming Interface
HPC	High-Performance Computing
NCSA	National Center for Supercomputing Applications
I/O	Input/Output
FAIR	Findability Accessibility Interoperability Reusability
ECT	Escola de Ciências e Tecnologia
UE	Universidade de Évora

Sumário

Esta dissertação descreve o processo de transposição e modernização do *software* de análise e visualização AstroPGPLOT (Avillez et al., 2009, 2020), implementado em FORTRAN e baseado na biblioteca gráfica PGPLOT², para o ecossistema moderno da linguagem Julia e da biblioteca gráfica Makie.

O trabalho visou não apenas reproduzir as funcionalidades do AstroPGPLOT, mas também avaliar a capacidade da nova plataforma em igualar ou superar a qualidade gráfica e o desempenho computacional do sistema original. O resultado deste esforço é o pacote AstroTLPlot (disponível em <https://github.com/tomalima/AstroTLPlot>), uma ferramenta modular que unifica a leitura, transformação, análise e visualização de dados provenientes de simulações astrofísicas.

Além de uma fiel transposição do núcleo científico, o AstroTLPlot introduz capacidades modernas, como visualização interativa e uma integração mais eficiente com ferramentas contemporâneas de computação científica. A validação realizada demonstra que a transposição para Julia e Makie constitui uma alternativa viável e de alto desempenho, capaz de preservar o rigor científico do sistema anterior enquanto oferece melhorias significativas em usabilidade, interoperabilidade e potencial de manutenção futura.

Este trabalho contribui, assim, para a modernização do fluxo de trabalho em análise de dados astrofísicos.

Palavras chave: Análise de Dados, Visualização, Astrofísica, Julia, Jupyter notebook

²PGPLOT Graphics Subroutine Library (T. J. Pearson, 2001). Disponível em: <http://www.astro.caltech.edu/~tjp/pgplot/>

Abstract

Data Analysis and Visualisation in Astrophysics

This dissertation describes the process of porting and modernizing the analysis and visualization software *AstroPGPLOT* (Avillez et al., 2009, 2020) – originally implemented in FORTRAN and based on the PGPLOT graphics library³, to the modern ecosystem of the Julia language and the Makie graphics library.

The work aimed not only to reproduce the functionality of the legacy code but also to assess the new platform's ability to match or surpass the graphical quality and computational performance of the original system. The result of this effort is the AstroTLPlot package (available at <https://github.com/tomalima/AstroTLPlot>), a modular tool that unifies the reading, transformation, analysis, and visualization of data from astrophysical simulations.

Beyond a faithful port of the scientific core, AstroTLPlot introduces modern capabilities such as interactive visualization and more efficient integration with contemporary scientific computing tools. The validation performed demonstrates that the transition to Julia and Makie constitutes a viable, high-performance alternative, capable of preserving the scientific accuracy of the former system while offering significant improvements in usability, interoperability, and future maintainability.

This work thus contributes to modernizing the workflow in high-complexity astrophysical data analysis.

Keywords: Data Analysis, Visualization, Astrophysics, Julia, Jupyter Notebook

³PGPLOT Graphics Subroutine Library (T. J. Pearson, 2001). Available at: <http://www.astro.caltech.edu/~tjp/pgplot/>

1

Introdução

1.1 Contextualização da Pesquisa

Atualmente, a astrofísica gera volumes massivos de dados, da ordem de dezenas e centenas de terabytes, e mesmo de petabytes, em simulações numéricas e em observações realizadas com telescópios no solo e no espaço [GDDM13]. Estes conjuntos de dados, de grande dimensão, exigem métodos de processamento e visualização eficientes, ultrapassando as capacidades das soluções tradicionais [BCL14].

Desde que a linguagem C se tornou tão rápida quanto o FORTRAN, passou também a ser utilizada. O Python é atualmente a linguagem de eleição na Astronomia e Astrofísica. As linguagens FORTRAN 2003/2008, C e C++ são sobretudo usadas em trabalhos computacionalmente exigentes, como simulações numéricas ou bibliotecas de processamento de dados e imagem, devido à necessidade de elevada velocidade de processamento, capacidade de vectorização e à exploração de instruções SIMD (*Single Instruction, Multiple Data*), favorecidas por estas linguagens, em particular pelo FORTRAN 2008 e FORTRAN 2018.

Neste contexto, a linguagem de programação Julia afirma-se como uma solução promissora, ao combinar a eficiência computacional de linguagens compiladas com uma sintaxe acessível e um ecossistema abrangente de bibliotecas científicas [BEKS17].

A linguagem Julia oferece compilação just-in-time (JIT) e elevados níveis de desempenho, apresentando, de acordo com diversos benchmarks, velocidades de processamento comparáveis às do FORTRAN e do C, particularmente em aplicações de computação científica e numérica [BKSE12a, BEKS17, The26, BKSE12b].

A compilação Just-In-Time (JIT) em Julia permite uma execução eficiente do código em cenários de execução com um número limitado de cores, facilitando igualmente a interoperabilidade com outras linguagens, como Python e MATLAB, o que pode favorecer uma transição gradual para um ambiente computacional moderno [Per19]. Paralelamente, o ecossistema de visualização científica em Julia tem registado um desenvolvimento notável, com bibliotecas como `Plots.jl`¹ e `Makie.jl`² [DK21], que permitem a criação de gráficos interativos e de alta qualidade.

Neste sentido, esta tese procura não apenas explorar as capacidades de Julia, mas também contribuir para o desenvolvimento de um pacote modular, o `AstroTLPlot` [LCdA25], concebido para agilizar o trabalho dos investigadores em astrofísica. Importa referir que este pacote tem como base o software `AstroPGPLOT` (de Avillez et al., 2009, 2020) que nos foi cedido para este trabalho. O `AstroTLPlot` integra funcionalidades de leitura de dados, pré-processamento, cálculo estatístico e visualização, oferecendo uma solução integrada para os desafios computacionais da astrofísica contemporânea.

A transição para nova ferramenta e linguagem de desenvolvimento não é apenas uma questão técnica, mas também uma necessidade estratégica para a investigação científica contemporânea [SLP12, WAB⁺14]. Ferramentas capazes de lidar com a complexidade crescente dos dados são fundamentais para que a comunidade astrofísica continue a realizar descobertas e avançar no conhecimento do Universo.

1.2 Motivação e Justificação

A motivação desta tese emerge da necessidade crescente de desenvolvimento de ferramentas computacionais eficientes para lidar com volume massivo de dados gerados na astrofísica contemporânea. Com a modernização dos observatórios e o surgimento de simulações de alta resolução, impõe-se à comunidade científica a necessidade de gerir e analisar *Big Data* [WAB⁺14, BCL14], ou seja, conjuntos de dados de grande dimensão e complexidade sem precedentes.

As ferramentas tradicionais revelam limitações em termos de desempenho e na escalabilidade, especialmente perante à complexidade crescente dos conjuntos de dados e das simulações numéricas [GDDM13]. Estas limitações impulsionam a adoção de soluções modernas, como a linguagem de programação Julia [BEKS17, Per19], que oferece uma velocidade de execução comparável a linguagens como FORTRAN e C em cenários de uso geral. A par do desempenho computacional, o ecossistema de visualização em Julia tem registado um desenvolvimento significativo, com bibliotecas como `Plots.jl` e `Makie.jl` [DK21] a oferecerem capacidades avançadas para exploração e apresentação de dados científicos.

A linguagem Julia afirma-se como uma solução robusta ao combinar a expressividade e facilidade de prototipagem do MATLAB com a acessibilidade do Python e o desempenho computacional do C e do FORTRAN. Esta convergência permite resolver o "problema das duas linguagens", oferecendo ainda integração nativa com bibliotecas científicas e suporte para formatos de dados como HDF5 e VTK [Per19, BEKS17].

Esta versatilidade torna-a particularmente adequada para operações matemáticas complexas e o tratamento de dados multidimensionais exigidos em astrofísica. A criação de gráficos interativos em Julia, através de bibliotecas como o `Makie.jl`, possibilita uma visualização clara e personalizável de dados complexos, facilitando a sua interpretação e a comunicação de resultados científicos [DK21].

A tese propõe ainda o desenvolvimento de um pacote modular, o `AstroTLPlot`, concebido para ser expansível e integrável com outras ferramentas científicas. Esta iniciativa insere-se no movimento mais amplo de promoção de código aberto e de colaboração científica [SLP12], para criar soluções inovadoras, reprodutíveis e acessíveis para a análise e visualização de dados astrofísicos.

¹`Plots.jl`: <https://github.com/JuliaPlots/Plots.jl>

²`Makie.jl`: <https://github.com/JuliaPlots/Makie.jl>

1.3 Objetivos

Objetivo Geral

Transpor o software AstroPGPLOT (de Avillez et al., 2009, 2020) implementado em FORTRAN 2008 para a linguagem de programação Julia, através do desenvolvimento do pacote modular AstroTLPlot.

Objetivos Específicos

- Transpor as rotinas de processamento e análise de dados astrofísicos desenvolvidas em FORTRAN 2008, garantindo desempenho, precisão e compatibilidade com as estruturas modernas de Julia;
- Substituir todas as dependências da biblioteca gráfica PGPLOT no código do AstroPGPLOT transposto para Julia por bibliotecas nativas deste ecossistema, especificamente o `Makie.jl`;
- Desenvolver o pacote AstroTLPlot como uma ferramenta modular, reutilizável e extensível, disponibilizado em código aberto para fomentar a colaboração e a evolução da comunidade científica;
- Integrar AstroTLPlot com o ambiente Jupyter Notebook [KRKP⁺16b], permitindo interatividade, documentação eficiente e apresentação clara dos resultados;
- Demonstrar as vantagens da linguagem Julia face as linguagens tradicionais (como FORTRAN), designadamente em termos de desempenho computacional, clareza sintática e flexibilidade de integração;
- Assegurar a aplicação sistemática de boas práticas de desenvolvimento de software científico [WAB⁺14], garantindo a reprodutibilidade dos resultados, a validação rigorosa dos métodos e a transparência do código;
- Validar a ferramenta com dados reais provenientes de simulações astrofísicas, de modo a comprovar a sua fiabilidade, robustez e aplicabilidade prática.

1.4 Metodologia

A metodologia adotada nesta tese assenta numa abordagem sistemática e iterativa, que combina a transposição de código científico do AstroPGPLOT com o desenvolvimento de uma nova ferramenta computacional em Julia [WAB⁺14]. O objetivo é assegurar, em simultâneo, rigor científico, o desempenho computacional e a conformidade com as boas práticas de programação e de engenharia de software. Este trabalho estruturou-se, assim, em seis etapas interdependentes que detalham a construção do pacote AstroTLPlot:

1. Análise do Código em FORTRAN 2008

- Estudo detalhado do código fonte do AstroPGPLOT (de Avillez et al., 2005, 2020), implementado em FORTRAN 2008, com análise das suas componentes principais: estrutura de dados, leitura de ficheiros, cálculo numérico, estatística, e visualização através da biblioteca PGPLOT [Pea01].
- Identificação de dependências, (incluindo a biblioteca PGPLOT), fluxos de dados e limitações do código existente, de modo a planear uma transposição fidedigna e sustentável do AstroPGPLOT para Julia.

2. Transposição das Rotinas para Julia

- Reimplementação progressiva das rotinas em Julia, respeitando a lógica original, mas aproveitando as vantagens da linguagem, como a compilação *Just-In-Time* (JIT), a sintaxe expressiva, *multiple dispatch* e a capacidade de integração com outras linguagens [BEKS17].
- Adoção de uma abordagem modular e extensível, promovendo a reutilização de código e a escalabilidade futura [Hin19a].

3. Integração de Bibliotecas Modernas

- Substituição das rotinas gráficas baseadas em PGPLOT por bibliotecas modernas de visualização científica, o `Makie.jl` [DK21], que oferece renderização interativa e gráficos de alta qualidade.

- Integração com o ambiente interativo do Jupyter Notebook [KRKP⁺16b], que permite a documentação do processo, a análise dinâmica dos resultados e a disseminação do conhecimento em formato reproduzível (*Markdown*).

4. Construção e Implementação do Pacote AstroTLPlot

- Desenvolvimento do pacote AstroTLPlot, concebido como uma ferramenta modular e aberta, de fácil integração em fluxos de trabalho científicos, seguindo os princípios de engenharia de software aplicada à investigação [Hin19a, WAB⁺14].
- Implementação de documentação estruturada e exemplos de utilização, alinhados com as boas práticas de desenvolvimento de software científico e de gestão de versões.

5. Validação e Testes

- Aplicação das rotinas desenvolvidas em Julia a dados reais de simulações astrofísicas, com avaliação e correção dos resultados e o desempenho da ferramenta.
- Comparação sistemática dos resultados obtidos em Julia com os da versão original em FORTRAN 2008, de modo a demonstrar a fidelidade científica e eventuais ganhos de desempenho.

6. Garantia de Qualidade e Reprodutibilidade Científica

- Adoção de práticas que asseguram a reprodutibilidade e a validação rigorosa dos resultados, incluindo a utilização de código aberto e metodologias de teste sistemático [WAB⁺14, SLP12].
- Registo contínuo da evolução do projeto em ambiente de controlo de versões (*Git/GitHub*), garantindo rastreabilidade, transparência e colaboração.

1.5 Estrutura da Tese

Esta tese encontra-se organizada em cinco capítulos principais:

- **Capítulo 1 – Introdução:** Apresenta a contextualização do tema, a motivação e justificação, os objetivos, a metodologia e a estrutura do trabalho;
- **Capítulo 2 – Fundamentação Teórica e Ferramentas:** descreve os conceitos teóricos, os formatos de dados astrofísicos, as linguagens e ferramentas tradicionais (FORTRAN 2008, PGPLOT) e modernas (Julia, Makie), bem como a comparação entre abordagens;
- **Capítulo 3 – Transposição e Desenvolvimento da Ferramenta:** Detalha o processo de transposição das rotinas em FORTRAN 2008 do AstroPGPLOT para Julia, a implementação do pacote AstroTLPlot, a reescrita e adaptação das rotinas de visualização com Makie e a validação dos resultados;
- **Capítulo 4 – Funcionamento e Aplicação da Ferramenta:** Apresenta o modo de operação da ferramenta desenvolvida e as suas principais funcionalidades, ilustrando a aplicação com dados reais e as melhorias obtidas face à implementação original;
- **Capítulo 5 – Conclusões e Trabalhos Futuros:** Apresenta as conclusões, sintetiza as principais contribuições, identifica limitações e aponta perspetivas de desenvolvimento futuro para expandir e aprofundar o projeto.

2

Fundamentação Teórica e Ferramentas

2.1 Enquadramento Teórico

As simulações numéricas em astrofísica constituem, atualmente, um dos pilares fundamentais para a compreensão da dinâmica e evolução do Universo. Através delas, é possível estudar fenómenos complexos como a evolução do Universo a partir do Big Bang, a formação e evolução de galáxias, o meio interestelar e a formação de estrelas, a turbulência, a evolução das espécies químicas, etc.: muitas vezes impossíveis de reproduzir em laboratório, permitindo criar cenários controlados onde parâmetros físicos podem ser ajustados e testados. Estas simulações fornecem, assim, uma base robusta para a validação de modelos teóricos e a interpretação de dados observados.

Historicamente, ferramentas desenvolvidas em linguagens como FORTRAN, nomeadamente o AstroPGPLOT (de Avillez+ 2005, 2020), desempenharam um papel determinante na área da simulação e visualização de dados astrofísicos [Pea01].

Estes códigos, desenvolvidos em FORTRAN 2008, foram concebidos para maximizar a eficiência computacional em sistemas de elevado desempenho, respondendo às exigências de simulações de larga escala. Contudo, à medida que a complexidade dos modelos aumentou e o volume de dados cresceu exponencialmente, a comunidade científica tem procurado complementar ferramentas estabelecidas como o AstroPGPLOT, que mantém eficiência através da sua arquitetura modular e de compiladores FORTRAN 2008 modernos, com ecossistemas de programação que oferecem novas capacidades de interatividade, visualização avançada e integração fluida com fluxos de trabalho científicos contemporâneos.

Este contexto impulsionou a transição para linguagens de programação mais recentes, como o Julia, que combina desempenho elevado com maior acessibilidade e extensibilidade [BEKS17], mantendo a utilização de formatos de dados robustos e estabelecidos como o HDF5 [FHK⁺11]: um formato hierárquico inspirado em sistemas de ficheiros UNIX que evoluiu do HDF4 e se mantém como padrão na comunidade científica há mais de duas décadas.

A motivação para este trabalho prende-se com a oportunidade de transpor para Julia um software ativo e eficiente, o AstroPGPLOT, que processa grandes volumes de dados de simulações astrofísicas de forma rápida e robusta. A natureza e diversidade dos dados gerados por estas simulações, que variam conforme o tipo de problema estudado e o software de simulação utilizado, exigem ferramentas de análise e visualização capazes de lidar com diferentes formatos e escalas, mantendo o desempenho computacional.

Entre os formatos suportados pelo AstroPGPLOT e relevantes para este trabalho destacam-se:

- **HDF4 e HDF5 (Hierarchical Data Format):** formatos hierárquicos amplamente utilizados em simulações científicas, para armazenamento estruturado de grandes volumes de dados;
- **VTK (Visualization Toolkit):** formato orientado para aplicações de visualização científica tridimensional e representação volumétrica;
- **Ficheiros ASCII .DAT estruturados:** contendo dados 3D em formato matricial ou dados 1D em colunas, comum em implementações científicas mais diretas.

O manuseamento destes dados coloca desafios significativos. A sua dimensão pode atingir dezenas ou centenas de gigabytes, exigindo estratégias eficientes de armazenamento, leitura e pré-processamento. Além disso, a diversidade de formatos obriga à criação de mecanismos de conversão e normalização que assegurem a compatibilidade entre as diversas fases do fluxo de trabalho científico, desde a simulação até à análise e visualização final.

Neste contexto, a revisão de literatura evidencia não só a relevância das simulações numéricas na investigação astrofísica, mas também a necessidade urgente de infraestruturas de software modernas. Estas devem ser capazes de processar grandes volumes de dados de forma eficiente e flexível, garantindo a robustez científica que este campo exige. Para além disso, é fundamental que estas novas infraestruturas possuam uma capacidade de integração perfeita com as ferramentas contemporâneas de visualização e análise. Só assim se poderá potenciar plenamente o conhecimento gerado pelas simulações, assegurando que a comunidade científica pode extrair e interpretar os resultados de maneira eficaz.

2.2 Ferramentas e Tecnologias Existentes

O desenvolvimento de software para simulações e análise de dados em astrofísica tem historicamente recorrido a uma variedade de linguagens de programação e bibliotecas científicas. Entre as linguagens mais comuns encontram-se o FORTRAN, amplamente utilizado em programas desenvolvidos no passado devido à sua eficiência em cálculos numéricos; o C++, valorizado pelo controlo de baixo nível e desempenho; linguagens de alto nível como Python e o Matlab, apreciadas pela facilidade de prototipagem e pelo vasto ecossistema de bibliotecas científicas [Pyt23, Mat23]; e o IDL, tradicionalmente adotado em contextos de visualização e análise de dados astronómicos [Har22]. Para a visualização de dados, ferramentas como o PGPLOT [Pea01] (tradicionalmente associado a FORTRAN e C) e bibliotecas modernas como o Makie.jl [DK21], integradas no ecossistema Julia, têm permitido representar de forma eficiente resultados de simulações tridimensionais e dados volumosos. Estas soluções oferecem funcionalidades que incluem gráficos estáticos e interativos, suporte a mapas de cores avançados e integração com fluxos de trabalho de análise científica.

Um dos problemas clássicos enfrentados pelos investigadores nesta área é o chamado “problema das duas linguagens” [BEKS17]: utiliza-se frequentemente uma linguagem de alto nível (como Python ou Matlab) para prototipagem e desenvolvimento rápido de algoritmos e uma linguagem de baixo nível (como FORTRAN ou C++) para alcançar o desempenho computacional exigido por simulações de grande escala. Este dualismo aumenta a complexidade do desenvolvimento, dificulta a manutenção do código e exige

esforços de integração entre diferentes ambientes.

A linguagem Julia surge como uma alternativa moderna que combina a facilidade de prototipagem com o desempenho de linguagens compiladas [BEKS17]. O seu modelo de *compilação Just-In-Time (JIT)* e o mecanismo de *multiple dispatch* permitem desenvolver código genérico, reutilizável e altamente otimizado, mantendo simultaneamente uma sintaxe simples e expressiva. Além disso, o seu ecossistema científico inclui bibliotecas para álgebra linear, estatística, simulações numéricas, processamento de dados, visualização interativa (Makie.jl) e integração com formatos como HDF5 e VTK. Estas características permitem desenvolver *pipelines* completos de simulação e análise num único ambiente, reduzindo a fragmentação entre etapas de desenvolvimento e execução. A Tabela 2.1 apresenta uma comparação resumida entre linguagens de programação científicas amplamente utilizadas, evidenciando o posicionamento de Julia em termos de desempenho, usabilidade e integração.

Tabela 2.1: Comparação entre linguagens de programação para computação científica: Julia, Python, FORTRAN e MATLAB.

Critério	Julia	Python	FORTTRAN	MATLAB
Velocidade de Execução	Muito alta, quase igual a C/ FORTRAN	Moderada, mais lenta que Julia/ FORTRAN	Muito alta, ideal para cálculos numéricos intensivos	Alta, mas geralmente inferior a Julia/ FORTRAN
Facilidade de Utilização	Simples, sintaxe semelhante a Python	Muito simples, extensa documentação	Complexa, devido à sintaxe tradicional	Simples, com ambiente integrado intuitivo
Suporte a Bibliotecas	Crescente, com foco em computação científica	Extenso, especialmente em ciência de dados	Limitado em bibliotecas modernas	Extenso, mas dependente de ambiente proprietário
Paralelismo	Nativo, com excelente suporte	Disponível via bibliotecas (e.g., <i>multiprocessing</i> , <i>Dask</i>)	Forte, suportado desde versões iniciais	Suporte disponível, mas não tão nativo quanto Julia ou FORTRAN
Integração com Outras Linguagens	Excelente (pode chamar Python, C, FORTRAN diretamente)	Boa, integração via <i>FFI</i> e <i>wrappers</i>	Limitada, difícil integração com linguagens modernas	Boa, mas com menos flexibilidade que Julia
Tempo de Compilação	Ligeiro atraso inicial devido à compilação JIT, compensado por execuções subsequentes rápidas	Não aplicável (interpretado)	Muito rápido	Não aplicável (interpretado)
Popularidade e Comunidade	Em crescimento acelerado, mas ainda menor que Python	Enorme e estabelecida, especialmente em ciência de dados	Em declínio, prevalente em projetos legados	Estável, mas menor comparada a Python
Custo	Gratuito e open-source	Gratuito e open-source	Gratuito, embora com curva de aprendizagem mais acentuada	Licenciado, especialmente em ambientes corporativos

Julia apresenta-se como uma linguagem de programação promissora para a computação científica mo-

derna, oferecendo um excelente equilíbrio entre desempenho e facilidade de utilização. Embora ainda esteja em desenvolvimento, o seu potencial torna-a particularmente atrativa para a astrofísica computacional, onde a manipulação de grandes volumes de dados e a necessidade de execução eficiente são fatores críticos.

2.3 Evolução e Estrutura dos Formatos HDF4 e HDF5

O *Hierarchical Data Format* (HDF) é uma família de formatos concebida para o armazenamento e a organização de grandes volumes de dados científicos. Desenvolvido originalmente pelo *National Center for Supercomputing Applications* (NCSA), este formato tornou-se um padrão amplamente adotado em áreas como a astrofísica, a meteorologia, a física computacional e as ciências da Terra, devido à sua estrutura hierárquica e à elevada eficiência de entrada/saída (I/O) de dados[FHK⁺11]. O formato HDF evoluiu ao longo do tempo em duas principais gerações, HDF4 e HDF5, desenvolvidas para responder às necessidades crescentes de armazenamento estruturado e acesso eficiente a grandes conjuntos de dados científicos.

As versões mais relevantes desta família são o HDF4 e o HDF5, cada uma representando uma etapa distinta de maturidade tecnológica no tratamento de grandes volumes de dados científicos:

- **HDF4**[The22a], desenvolvido no início da década de 1990, foi amplamente utilizado em missões de satélite, meteorologia e astrofísica. No entanto, apresentava limitações ao nível da escalabilidade e da gestão de conjuntos de dados multidimensionais de grande dimensão.
- **HDF5**[The22b], introduzido como sucessor direto, implementa uma estrutura mais robusta e modular. O HDF5 organiza os dados de forma hierárquica, de modo análogo a um sistema de ficheiros, contendo grupos, conjunto de dados e atributos. Esta abordagem permite representar dados complexos e heterogêneos de forma eficiente e coerente.

Principais Características do Formato HDF5

- **Portabilidade:** compatibilidade entre diferentes sistemas operativos e arquiteturas;
- **Escalabilidade:** suporte para *datasets* de dimensão superior a terabytes;
- **Eficiência de I/O:** acesso seletivo a partes específicas dos dados e suporte a compressão nativa;
- **Metadados integrados:** possibilidade de descrever o contexto científico dos dados (ex.: unidades, coordenadas, variáveis, dimensões).

A estrutura de um ficheiro HDF5 é hierárquica, organizada em grupos, *datasets* e atributos, permitindo armazenar dados complexos e multidimensionais juntamente com metadados descritivos, como unidades, dimensões e sistemas de coordenadas. Esta organização modular facilita a portabilidade e a interoperabilidade entre diferentes plataformas e linguagens de programação. A Figura 2.1 ilustra esta estrutura hierárquica típica de um ficheiro HDF5.

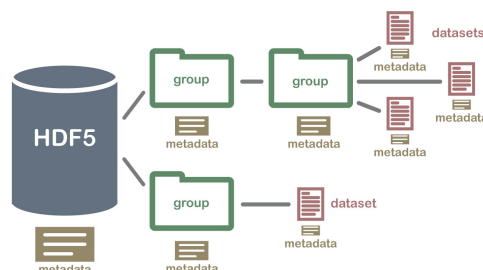


Figura 2.1: Estrutura do formato de ficheiro HDF5 (Fonte: Adaptado de [NEO24]).

Graças à sua versatilidade e desempenho, o HDF5 é atualmente um padrão em diversas áreas científicas,

incluindo a astrofísica, as ciências da Terra, a física de partículas e a biologia computacional [FHK⁺11, The23a].

2.4 O Formato YAML em Contexto Científico

O *YAML* (*YAML Ain't Markup Language*) é um formato de serialização de dados legível por humanos, amplamente utilizado para ficheiros de configuração e parametrização de simulações científicas [EO06]. A sua simplicidade e clareza tornam-no uma escolha ideal para descrever modelos e *pipelines* em software científico moderno. O *YAML* baseia-se numa estrutura hierárquica, organizada através de pares chave-valor, listas e mapas aninhados. A sua sintaxe minimalista, assente na indentação, torna a leitura e edição acessíveis mesmo para utilizadores com pouca experiência em programação. Em comparação com formatos como *JSON* ou *XML*, o *YAML* oferece maior legibilidade e flexibilidade, o que explica a sua adoção crescente em contextos científicos e de engenharia, nomeadamente para:

- Definição de parâmetros de execução em software científico;
- Configuração de *pipelines* de dados;
- Descrição de modelos experimentais ou simulações.

Esta combinação de simplicidade, legibilidade e integração com linguagens modernas de programação faz do *YAML* um formato de referência na configuração de aplicações científicas e sistemas complexos.

2.5 VTK (Visualization Toolkit)

O *Visualization Toolkit* (*VTK*) é uma biblioteca e formato de dados projetado para visualização científica tridimensional, amplamente utilizado em simulações numéricas [SML06]. A sua estrutura modular e suporte a dados volumétricos complexos tornam-no adequado para representar resultados astrofísicos de elevada resolução. A principal vantagem do *VTK* reside na sua capacidade de representar dados espaciais e escalares de forma eficiente. Um ficheiro *VTK* organiza os dados em três componentes principais:

- **Cabeçalho (*header*):** contém informações gerais, como versão do ficheiro, descrição do *dataset* e tipo de dados;
- **Dados geométricos (*geometry*):** define os pontos, células, malhas ou superfícies que descrevem o domínio espacial da simulação;
- **Atributos (*attributes*):** valores associados a cada ponto ou célula, como densidade, temperatura, pressão ou velocidades.

2.6 Bibliotecas Gráficas para Visualização de Dados Científicos

A visualização de dados científicos é um elemento central na análise de simulações astrofísicas, permitindo transformar grandes volumes de informação numérica em representações compreensíveis e interpretáveis.

Ao longo de várias décadas, o *PGPLOT* estabeleceu-se como uma biblioteca amplamente adotada para visualização científica, particularmente em domínios como a astronomia e astrofísica [Pea01], coexistindo com outras bibliotecas gráficas como o *PLplot*. Desenvolvido nos anos 1983, era especialmente utilizado em programas escritos em *FORTRAN* e *C*, fornecendo suporte a gráficos 2D e, de forma limitada, a representações 3D.

Embora o *PGPLOT* permaneça uma biblioteca eficiente e amplamente utilizada para visualização científica estática com controlo gráfico preciso [Pea01], a transposição para o ecossistema *Julia* através de bibliotecas como *Makie.jl* [DK21] oferece oportunidades para:

- Integrar visualização interativa diretamente em ambientes como *Jupyter Notebooks*;
- Explorar representações gráficas avançadas (3D, volumétricas) com APIs modernas;
- Beneficiar de um ecossistema coeso para análise, processamento e visualização de dados;
- Facilitar a prototipagem e desenvolvimento com ferramentas contemporâneas.

A Tabela 2.2 resume as principais diferenças entre as duas bibliotecas.

Tabela 2.2: Comparação entre as bibliotecas PGPLOT e Makie.jl.

Critério	PGPLOT	Makie.jl
Ano de desenvolvimento	1993 (T. J. Pearson)	Ativo desde 2018
Linguagem base	É baseado em FORTRAN 77 (com interfaces C)	Nativo de Julia (com múltiplos backends especializados)
Interatividade	Limitada; cursor gráfico básico e interatividade simples se integrada por software externo	Interatividade nativa em 2D/3D com eventos, transformações e sliders em tempo real
Suporte gráfico	Predominantemente 2D	Capacidades gráficas modernas, incluindo 2D, 3D, animações e interatividade rica.
Desempenho	Eficiente, compilável com GCC, Intel One API, Absoft, etc., com vetorização automática em FORTRAN 2003/2008	Otimizado para Computação de Alto Desempenho (HPC) e grandes <i>datasets</i>
Integração moderna	Restrita	Total, com ecossistema Julia e formatos científicos
Estado atual	Biblioteca madura, sem desenvolvimento ativo oficial recente	Em desenvolvimento contínuo e em expansão

2.6.1 Alternativas Modernas para Visualização Científica

Para contextualizar o papel do PGPLOT e das alternativas modernas, é importante distinguir entre diferentes categorias de ferramentas de visualização:

- **PGPLOT (FORTRAN/C):** Permite visualização integrada em software em execução, gerando gráficos "on-the-run" durante o processamento de dados, eficiente para milhares de cubos de dados;
- **Matplotlib (Python):** Amplamente utilizada para gráficos estáticos 2D, com integração em scripts e notebooks;
- **Plots.jl e Makie.jl (Julia):** Bibliotecas nativas do ecossistema Julia para visualização científica, com capacidades modernas de interatividade e integração.

Em síntese, a comparação evidencia que tanto o PGPLOT como o Makie.jl oferecem capacidades valiosas para visualização científica, cada uma com os seus pontos fortes. O PGPLOT mantém-se como uma solução robusta e eficiente para visualização integrada em software em execução, com controlo preciso sobre elementos gráficos e desempenho comprovada em processamento de grandes volumes de dados. Por sua vez, o Makie.jl representa uma alternativa moderna que beneficia do ecossistema Julia, oferecendo interatividade nativa, visualizações avançadas (2D/3D, animações) e integração fluida com ferramentas contemporâneas de processamento científico. A transição para soluções como o Makie.jl não invalida as capacidades do PGPLOT, mas antes oferece novas possibilidades que respondem a necessidades emergentes no fluxo de trabalho científico atual.

2.7 Comparação de Abordagens: FORTRAN/PGPLOT e Julia/Makie

A evolução das linguagens de programação e das bibliotecas gráficas utilizadas em astrofísica ilustra a transição do paradigma FORTRAN + PGPLOT [Pea01], para soluções modernas como Julia e Makie.jl [BEKS17, DK21]. Esta comparação permite compreender as razões pelas quais linguagens e bibliotecas atuais se revelam mais adequadas aos desafios contemporâneos da investigação científica.

2.7.1 FORTRAN vs. Julia

FORTRAN consolidou-se ao longo do século XX como a linguagem padrão para cálculos científicos de alto desempenho, sendo amplamente utilizada em simulações numéricas e em ambientes de computação de alto desempenho (HPC). A sua eficiência e proximidade ao hardware permitiram a execução de simulações de grande escala em recursos computacionais limitados. Contudo, a sua rigidez sintática, a curva de aprendizagem e a fraca integração com bibliotecas modernas dificultam o desenvolvimento ágil e a manutenção de grandes bases de código.

Neste contexto, surge a linguagem Julia como uma alternativa moderna, que combina facilidade de utilização de linguagens de alto nível (como Python ou Matlab) com um desempenho próximo de linguagens compiladas, como FORTRAN e C. A compilação *Just-In-Time (JIT)* garante rapidez na execução, enquanto o *multiple dispatch* e a tipagem dinâmica tornam a linguagem extremamente flexível para aplicações científicas. Acresce ainda a existência de um ecossistema em rápida expansão, com pacotes especializados em álgebra linear, estatística, análise de dados e visualização, facilitando a criação de *pipelines* integrados num único ambiente.

2.7.2 PGPLOT vs. Makie.jl

O PGPLOT desempenhou um papel pioneiro na visualização de dados científicos, sendo amplamente utilizado em linguagens como FORTRAN e C. Contudo, as suas capacidades estão limitadas a gráficos essencialmente estáticos e bidimensionais. A ausência de interatividade e a falta de suporte a visualizações complexas tornam-no pouco adequado para as necessidades atuais, em que se privilegia a exploração dinâmica e interativa de dados.

O Makie.jl, por outro lado, representa uma solução moderna, concebida de raiz para tirar partido das capacidades da linguagem Julia. Permite a criação de gráficos 2D, 3D, volumétricos e animados, com suporte a interatividade e visualização em tempo real. Além disso, integra-se de forma nativa com formatos comuns em astrofísica, como HDF5 e VTK, tornando-se uma ferramenta flexível e adaptável a diferentes cenários de análise e simulação científica. A **Tabela 2.3** apresenta uma comparação detalhada entre os ambientes clássicos FORTRAN + PGPLOT e a abordagem moderna Julia + Makie.jl, evidenciando as principais diferenças em desempenho, facilidade de utilização, manutenção e capacidades de visualização.

Tabela 2.3: Comparação entre FORTRAN + PGPLOT e a abordagem moderna Julia + Makie.jl.

Critério	FORTRAN + PGPLOT	Julia + Makie.jl
Desempenho	Elevada (HPC otimizado)	Elevada (JIT + paralelismo)
Facilidade	Sintaxe rígida, menos flexível	Sintaxe moderna, alto nível
Manutenção	Complexa em grandes projetos	Simplificada com pacotes integrados
Visualização	Estática, 2D limitado	Interativa, 2D/3D/volumes/animações
Integração	Baixa, ecossistema legado	Elevada, ecossistema moderno
Estado atual	Estável, mas em declínio	Ativo, em expansão

Em síntese, a literatura e a análise comparativa demonstram que Julia e Makie.jl representam a evolução natural face ao paradigma clássico FORTRAN + PGPLOT, conciliando desempenho com flexibilidade, interatividade e integração num único ambiente de desenvolvimento.

3

Transposição e Desenvolvimento do Pacote AstroTLPlot

Este capítulo descreve o processo de modernização e transposição do pacote AstroPGPLOT, originalmente implementado em FORTRAN 2008 com dependência da biblioteca gráfica PGPLOT, para o novo ecossistema AstroTLPlot desenvolvido em Julia com a biblioteca de visualização Makie. O trabalho culminou na concepção e desenvolvimento do pacote AstroTLPlot, que implementa um sistema integrado e moderno para leitura, processamento e visualização científica de dados em astrofísica, eliminando as dependências legadas e incorporando práticas de desenvolvimento contemporâneas. O pacote encontra-se disponível publicamente no repositório GitHub [LCdA25]. Serão detalhadas as etapas metodológicas seguidas, as decisões arquiteturais que sustentam o sistema, a estratégia de substituição do PGPLOT pelo Makie, e as melhorias de desempenho, funcionalidade e usabilidade implementadas, evidenciando os benefícios do novo ecossistema computacional para a comunidade astrofísica.

3.1 Linguagens, Bibliotecas e Ambientes de Desenvolvimento

O processo de transposição e desenvolvimento do novo sistema baseou-se na utilização da linguagem de programação Julia [BEKS17], em conjunto com bibliotecas científicas especializadas e ambientes de programação modernos. Esta combinação assegurou robustez, flexibilidade e desempenho computacional,

elementos essenciais para a modernização do código original implementado em FORTRAN 2008 (um padrão moderno da linguagem) e para a sua adaptação às exigências atuais de análise e visualização de dados em astrofísica.

Nesta secção apresentam-se as principais linguagens, bibliotecas e ambientes que sustentaram o desenvolvimento e execução do projeto, justificando as suas escolhas com base na literatura técnica e nas necessidades de investigação científica contemporânea. A transição do AstroPGPLOT (baseado em FORTRAN 2008 e na biblioteca gráfica PGPLOT) para o AstroTLPlot (implementado em Julia com Makie) representa não apenas uma mudança de linguagem, mas uma reestruturação arquitetónica completa do ecossistema de visualização científica.

3.1.1 Análise do Código-Fonte Original – AstroPGPLOT (FORTRAN 2008 + PGPLOT)

O ponto de partida para este trabalho foi o código-fonte do software AstroPGPLOT (Avillez+ 2005, 2020), uma ferramenta científica consolidada e de alto desempenho, desenvolvida em FORTRAN 2008 e que utiliza a biblioteca gráfica PGPLOT para a geração de visualizações. A combinação FORTRAN 2008 e PGPLOT demonstra eficiência comprovada para computação numérica intensiva. Uma característica distintiva do PGPLOT é a sua capacidade de renderização gráfica on-the-fly, permitindo a visualização em tempo real à medida que os dados são processados, uma funcionalidade crucial para a monitorização de simulações longas e para a inspeção imediata de grandes volumes de dados, como milhares de cubos de dados.

Contudo, o contexto tecnológico atual exige capacidades que vão além do paradigma gráfico do PGPLOT. A biblioteca, apesar da sua robustez e eficiência para plotting estático e renderização em tempo de execução, foi concebida num paradigma de gráficos essencialmente 2D e não oferece suporte nativo para funcionalidades modernas como visualizações tridimensionais interativas, composição dinâmica de painéis gráficos complexos, ou uma integração profunda com ecossistemas de análise de dados. Estas capacidades tornaram-se fundamentais para a análise exploratória de dados contemporânea.

A fase inicial do trabalho consistiu numa análise exploratória e engenharia reversa detalhada do código-fonte do AstroPGPLOT. Este processo teve como objetivos: (1) compreender a estrutura modular e a hierarquia das sub-rotinas FORTRAN 2008; (2) mapear o fluxo de execução e as dependências entre procedimentos; (3) identificar e analisar as estruturas de dados principais, incluindo variáveis em módulos e blocos COMMON; e (4) isolar a lógica de cálculo científico da lógica de renderização gráfica, que se encontrava intimamente acoplada através de chamadas à PGPLOT. Para tal, executaram-se rotinas específicas do código original, instrumentando-se pontos críticos para registar o estado de variáveis intermédias e validar os cálculos numéricos.

Dada a natureza intensiva do processamento de dados, os resultados intermédios foram redirecionados para ficheiros de texto, permitindo uma análise quantitativa precisa e estabelecendo um benchmark para a validação posterior da reimplementação em Julia. Esta etapa de análise e documentação foi crucial para garantir que a transposição mantivesse a fidelidade aos algoritmos científicos originais e aos seus resultados, cuja correção e desempenho estão comprovados.

O conhecimento profundo do código original, assim adquirido, serviu de alicerce para a reestruturação arquitetónica subsequente. O principal desafio de desenho consistiu em replicar a eficiência e a capacidade de renderização em tempo real do PGPLOT dentro do ecossistema Julia. Em vez de uma tradução literal, optou-se por uma reengenharia que desagregou a base de código monolítica em componentes funcionais distintos. Esta desagregação permitiu separar claramente a lógica de cálculo científico (os algoritmos astrofísicos) do motor de renderização gráfica. Esta separação foi fundamental para planejar a arquitetura modular do novo pacote AstroTLPlot, cujo objetivo é preservar a precisão científica e a eficiência do software original, ao mesmo tempo que introduz uma estrutura mais flexível, interoperável, e que incorpora capacidades de visualização moderna do ecossistema Makie/Julia.

3.1.2 Julia – Linguagem Base e Ecossistema de Bibliotecas

Para o desenvolvimento e execução das rotinas do novo sistema, adotou-se a linguagem de programação Julia (disponível em julialang.org), em conjunto com o ambiente Visual Studio Code (code.visualstudio.com) e o JupyterLab (jupyter.org). A seleção de Julia como núcleo do projeto fundamenta-se no seu manifesto fundacional de 2012, que propôs resolver o "trilema das linguagens de programação científica" [BKSE12a]: a busca por uma linguagem que simultaneamente ofereça (1) a velocidade de execução de linguagens compiladas como C e FORTRAN, (2) a dinâmica e a produtividade de linguagens interpretadas como Python e MATLAB, e (3) a capacidade de expressão e abstração de linguagens de propósito geral. A solução proposta, materializada na linguagem Julia, baseia-se num design inovador que combina uma compilação Just-In-Time (JIT) através do LLVM, uma tipagem dinâmica com inferência de tipos agressiva, e o paradigma de múltiplo dispatch para especialização de funções. Um aspeto igualmente decisivo para a sua adoção no contexto científico é o seu carácter de open source, com uma licença permissiva (MIT) que garante transparência, auditabilidade e livre adaptação do código-fonte, tanto da linguagem como da grande maioria do seu ecossistema de pacotes, eliminando barreiras de custo e promovendo a colaboração aberta.

Esta combinação faz de Julia uma escolha estratégica para a modernização do AstroPGPLOT. A linguagem oferece desempenho comparável ao FORTRAN 2008 para cálculos numéricos intensivos, um requisito não negociável para software astrofísico, enquanto introduz uma sintaxe expressiva de alto nível, um sistema de módulos robusto e um ecossistema de pacotes integrado. Adicionalmente, a sua capacidade de chamar diretamente bibliotecas em C, Python e, de forma crítica para este projeto, FORTRAN, permite uma transição gradual e a reutilização de rotinas otimizadas.

O seu gestor de pacotes integrado, Pkg, simplifica a instalação, atualização e manutenção de dependências, garantindo reprodutibilidade e isolamento de ambientes de execução, uma vantagem significativa em relação à gestão manual típica em projetos FORTRAN. Para o processamento e manipulação de dados, foram utilizados diversos pacotes especializados:

- CSV.jl: leitura e escrita de ficheiros tabulares, utilizada para manipulação de frações iónicas e dados estatísticos;
- HDF5.jl: gestão de ficheiros no formato HDF5, amplamente adotado em ciência de dados pela sua eficiência e capacidade de armazenar grandes volumes de informação hierárquica [FHK⁺11];
- YAML.jl: leitura de ficheiros de configuração no formato *YAML*, permitindo uma parametrização flexível e legível por humanos [EO06];
- LaTeXStrings.jl: integração de expressões matemáticas em legendas, rótulos e títulos de gráficos, garantindo consistência com a notação científica.

O aspeto central da modernização do ecossistema gráfico foi a adoção da biblioteca Makie.jl [DK21] como sucessora da PGPLOT. Esta escolha representa uma mudança de paradigma: enquanto o PGPLOT é uma biblioteca de renderização on-the-fly extremamente eficiente para a geração de gráficos 2D estáticos diretamente durante a execução de simulações, o Makie oferece um framework de visualização moderna, programável e de alto desempenho. O Makie permite a criação de gráficos interativos, tridimensionais e animados, incluindo mapas de calor e superfícies volumétricas, com suporte nativo a aceleração por GPU. Para replicar a capacidade de visualização incremental e em tempo real do PGPLOT, foi necessário desenvolver no AstroTLPlot um motor de renderização que gere e atualize a cena gráfica do Makie à medida que os dados são calculados, garantindo a funcionalidade crítica de monitorização do progresso.

Para além das bibliotecas externas, desenvolveu-se o pacote AstroTLPlot disponível publicamente em GitHub [LCdA25], que agrega os módulos de leitura, processamento, cálculo estatístico e visualização de dados astrofísicos numa arquitetura coesa. A gestão de versões e a colaboração foram asseguradas através do *Git* e da plataforma *GitHub*, amplamente reconhecidas como padrões de referência para controlo de versões e desenvolvimento colaborativo em projetos científicos e de código aberto [CS14].

A adoção deste ecossistema permite concretizar a visão do manifesto de Julia, modernizando o workflow de desenvolvimento face ao modelo tradicional do FORTRAN 2008. A integração entre as ferramentas pro-

porciona uma infraestrutura unificada para desenvolvimento, execução, análise exploratória e visualização de resultados, oferecendo simultaneamente produtividade, desempenho, reprodutibilidade e transparência, qualidades amplamente reconhecidas pela comunidade científica [BEKS17, KRKP⁺16a, Mic23b]. O objetivo do AstroTLPlot é, assim, preservar a robustez computacional do AstroPGPLOT enquanto expande radicalmente as suas capacidades analíticas e de visualização dentro de um ecossistema moderno, sustentável e aberto.

3.1.3 Jupyter Notebooks/Lab — Exploração e Desenvolvimento Interativo

O *Jupyter* é uma aplicação web de código aberto que permite a criação e partilha de documentos interativos que combinam código, visualizações e texto narrativo [KRKP⁺16a]. Inicialmente desenvolvido para as linguagens de programação Julia, Python e R — origem do acrónimo (“Ju–Py–R”) , o *Jupyter* evoluiu para suportar diversas linguagens através de *kernels* específicos. Os documentos criados neste ambiente, denominados *notebooks*, são compostos por células que podem conter código executável, texto formatado em *Markdown*, equações LaTeX, gráficos e widgets interativos. Essa combinação de elementos torna o Jupyter um recurso amplamente adotado em ciência de dados, modelação numérica e comunicação científica interativa [PG15].

Formas de Execução

Os *notebooks* permitem várias modalidades de execução:

- **Células individuais:** úteis para testes incrementais;
- **Execução sequencial completa:** através da opção `Run All`;
- **Execução interativa:** com reavaliação dinâmica de células modificadas;
- **Kernel Restart:** reinicialização do ambiente de execução, eliminando dependências residuais.

Ambientes e Variantes

O *Jupyter* é atualmente disponibilizado em diversas interfaces: *Jupyter Notebook*, *JupyterLab*, *Visual Studio Code* (através da extensão oficial) e *Google Colab*. O desenvolvimento desta tese utilizou principalmente o *JupyterLab*, pela sua interface modular e integração com Julia.

Vantagens da Utilização do Jupyter Notebooks com Julia

A utilização de *Jupyter Notebooks* com Julia oferece múltiplas vantagens práticas, especialmente nas fases de exploração e prototipagem:

- **Interatividade:** execução célula a célula, permitindo experimentação rápida e iteração sobre os resultados;
- **Integração de texto e código:** junção de explicações, código, equações e resultados num único documento;
- **Visualização imediata:** integração com bibliotecas como `Makie.jl` [DK21], facilitando a análise visual de resultados científicos;
- **Colaboração e partilha:** ficheiros de fácil disseminação e reprodução de experiências;
- **Reprodutibilidade:** encapsulação completa do processo de cálculo e visualização num único notebook [RBZ⁺19];
- **Extensibilidade:** possibilidade de integrar widgets interativos, extensões personalizadas e suporte a outras linguagens através de *kernels*;
- **Facilidade de utilização:** instalação simples via gestores de pacotes (`pip`, `conda`) e curva de aprendizagem acessível, amplamente adotada pela comunidade científica.

De acordo com as boas práticas propostas por Rule et al. (2019) [RBZ⁺19], a utilização de *Jupyter Notebooks* favorece a reprodutibilidade, a documentação e a partilha de análises computacionais, constituindo um dos pilares do ecossistema científico interativo moderno.

Limitações da Utilização do Jupyter Notebooks com Julia

Apesar das suas vantagens, a utilização do *Jupyter* com Julia apresenta também algumas limitações:

- **Desempenho:** execução ligeiramente mais lenta devido à sobrecarga da interface interativa;
- **Gestão de estado:** dependência do *kernel* persistente, podendo gerar inconsistências quando as células são executadas fora de ordem;
- **Depuração limitada:** ferramentas de depuração menos avançadas face a *IDEs* especializadas;
- **Controlo de versão:** integração mais difícil com o *Git* devido à natureza *JSON* dos ficheiros;
- **Complexidade de manutenção:** tendência à criação de células extensas e pouco modulares [KRKP⁺16b].

Papel no Desenvolvimento do Projeto

Durante as fases iniciais do desenvolvimento, o *Jupyter* foi utilizado como ambiente exploratório para validação de funções, testes rápidos e documentação dinâmica. A sua natureza interativa revelou-se essencial para avaliar a integração entre os módulos do sistema e para validar a consistência numérica da transposição. Contudo, à medida que o projeto evoluiu para uma fase de consolidação estrutural, tornou-se necessária a adoção de um ambiente mais robusto e orientado à manutenção, função assumida pelo *Visual Studio Code*.

3.1.4 Visual Studio Code – Programação, Documentação e Gestão de Projeto

O *Visual Studio Code* (VS Code) foi o ambiente de desenvolvimento integrado (*IDE*) principal utilizado para a organização e desenvolvimento do código-fonte. Através da extensão oficial para Julia, foram disponibilizadas funcionalidades avançadas de depuração, sugestão automática de código, gestão de pacotes e integração com o *Git*, garantindo uma maior produtividade e rastreabilidade do trabalho desenvolvido [Mic23b, Jul23].

Enquanto o *Jupyter* foi utilizado para prototipagem e análise exploratória, o *VS Code* proporcionou um ambiente de desenvolvimento estruturado e escalável, adequado à manutenção do sistema, à integração de módulos e à gestão de dependências. A extensão *Jupyter* permitiu ainda executar *notebooks* diretamente no mesmo espaço de trabalho, unificando as fases de experimentação e de produção [Mic23a].

Adicionalmente, a extensão *LaTeX Workshop* possibilitou a escrita e edição da dissertação no próprio ambiente de desenvolvimento [LaT23], enquanto o suporte integrado para o *Git* e o *GitHub* assegurou o controlo de versões, a revisão colaborativa e a publicação do código-fonte [CS14, Git23].

Integração e Sinergia do Ecossistema de Desenvolvimento

Em conjunto, as ferramentas analisadas desde a execução exploratória em FORTRAN 2008, passando pela adoção da linguagem **Julia** e das suas bibliotecas, até à utilização de ambientes de apoio como *Jupyter Notebooks/Lab* e *Visual Studio Code* formaram um ecossistema de desenvolvimento coeso.

Destaca-se, em particular, o papel do *Visual Studio Code*, que integrou num único ambiente a programação em Julia, a execução de notebooks, a escrita do relatório em *LaTeX* e a gestão colaborativa do projeto através do *Git* e *GitHub*.

3.2 Modelo de Dados e Formatos de Armazenamento

A qualidade e a fiabilidade dos resultados obtidos neste trabalho dependem fortemente dos dados de entrada utilizados. Nesta secção apresentam-se os principais tipos e formatos de dados utilizados, exemplos de

conteúdos e os procedimentos de preparação e pré-processamento que asseguraram a sua correta utilização e integridade científica.

3.2.1 Estrutura e Pré-processamento dos Dados

Os resultados das simulações numéricas foram armazenados em ficheiros estruturados nos formatos (HDF4/HDF5) [The22b, The22a], VTK [SML06] sob a forma de *datasets* multidimensionais, organizados em matrizes tridimensionais ($\text{Rank} = 3$). Cada matriz representa uma grandeza física distinta, como a densidade, a energia, ou as componentes da velocidade nos eixos x , y e z . Embora em algumas simulações uma das dimensões possa ser unitária (por exemplo, $2000 \times 2000 \times 1$), a estrutura de dados mantém a lógica tridimensional.

Essa coerência estrutural garante consistência na representação e facilita a extensão futura para simulações volumétricas completas.

A organização espacial baseia-se numa grelha cartesiana (*grid*), cujos vetores armazenam as coordenadas físicas correspondentes a cada índice da matriz. Deste modo, cada elemento $A(i, j, k)$ do *dataset* está associado não apenas ao seu valor numérico, mas também às coordenadas reais (x_i, y_j, z_k) do sistema físico. De forma esquemática:

```
Grid X → [0, dx, 2dx, ..., Lx]
Grid Y → [0, dy, 2dy, ..., Ly]
Grid Z → [0, dz, ..., Lz]
Dataset(i,j,k) → valor físico (ex. densidade)
```

Para além dos valores armazenados, cada *dataset* contém ainda um conjunto de metadados fundamentais para a sua correta interpretação [The22b], tais como:

- **Rank** — número de dimensões da matriz;
- **Size** — tamanho em cada eixo (n_x, n_y, n_z) ;
- **Escalas físicas** — fatores de conversão de índices para coordenadas reais;
- **Label** — identificação da variável física, unidade de medida e instante temporal da simulação;
- **Sistema de coordenadas** — tipicamente cartesiano.

Esta camada de metadados assegura que os ficheiros sejam independentes do código de origem, promovendo a reprodutibilidade, a portabilidade e a integração com diferentes ferramentas de análise e visualização [WBC⁺17, Hin19a].

3.2.2 Pré-processamento dos Dados

Antes de iniciar a análise, foi necessário aplicar um conjunto de etapas de pré-processamento, com o objetivo de uniformizar, validar e adaptar os dados para as rotinas de cálculo e visualização. Entre as operações mais relevantes destacam-se seguintes:

- Conversão de precisão numérica ($\text{float32} \rightarrow \text{float64}$);
- Aplicação de escalas logarítmicas em variáveis com ampla gama dinâmica, como densidade e energia;
- Tratamento de valores negativos em grandezas vetoriais;
- Filtragem de *datasets* artificiais ou inválidos, incluindo matrizes uniformes ou contendo valores NaN;
- Aplicação de fatores de escala obtidos no ficheiro de configurações.

Estas operações seguem as boas práticas de preparação e validação de dados científicos [WBC⁺17], garantindo que apenas os dados válidos e fisicamente consistentes fossem utilizados nas etapas subsequentes de processamento, cálculo estatístico, visualização e interpretação.

3.2.3 Exemplos de Ficheiros e Formato de Dados

Um exemplo concreto da estrutura de um ficheiro HDF5 (.h5), tal como utilizado nas simulações astrofísicas, é ilustrado na Figura 3.1. Esta estrutura é caracterizada pelos *datasets* multidimensionais de Velocidade, Densidade, Energia, e por matrizes auxiliares (*fakes*).

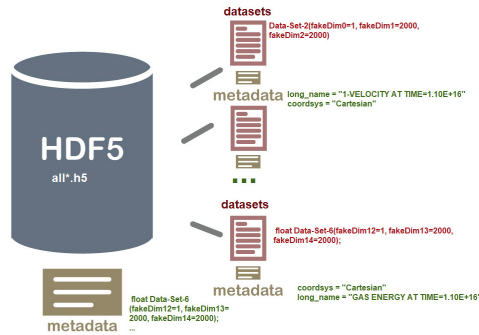


Figura 3.1: Exemplo da estrutura hierárquica de um ficheiro HDF5 utilizado no projeto.

A lista seguinte, extraída do processo de leitura, exemplifica a informação associada a um *dataset* em particular:

```
A Abrir ficheiro: data/input/dados-disk-2000x2000x1/all003000.h5
=== Informações do Dataset ===
Nome longo: DENSITY AT TIME=9.46E+15
Tipo de Dado: Float32
Dimensão X: 2000
Dimensão Y: 2000
Dimensão Z: 1
Dim Total: 4000000
Dims: (2000, 2000, 1) = 16000000 bytes
Sistema de coordenadas: Cartesian
size(3,)
size: 3
Cubo de Dados (3D)
Atributos
coordsys: Cartesian
Atributos
...
=== Informações do Dataset ===
Tipo de Dado: Float32
Dimensão X: 1
Dimensão Y: 1
Dimensão Z: 1
Dim Total: 1
Dims: (1,) = 4 bytes
Escalar (valor único)
Atributos
CLASS: DIMENSION_SCALE
Atributos
NAME: fakeDim0
...
```

A inclusão destes metadados é fundamental para a correta interpretação dos dados, uma vez que descrevem o significado físico, o contexto espacial e o instante temporal de cada matriz armazenada. Deste modo, os ficheiros HDF5 não se limita a armazenar os valores numéricos das simulações, mas encapsulam também

toda a informação necessária à sua interpretação, permitindo que o processamento e a visualização sejam feitos de forma consistente e independente do código original de simulação.

3.3 Modernização do Sistema de Configuração: de ASCII para YAML

Na versão original do AstroPGPLOT em FORTRAN, os parâmetros de configuração eram lidos a partir de ficheiros de texto simples em formato ASCII, caracterizados por delimitadores inconsistentes, falta de validação e sem qualquer estrutura hierárquica explícita. Cada ficheiro seguia uma sintaxe ad-hoc, muitas vezes definida por posições fixas de colunas ou por palavras-chave seguidas de valores em linhas específicas.

Esta abordagem, embora funcional e de implementação direta, apresentava várias limitações que se agravavam com a complexidade crescente do software: dificultava a manutenção e a extensão do código, aumentava a probabilidade de erros de sintaxe ou de interpretação por parte do utilizador, e tornava-se particularmente crítica em simulações de grande dimensão que exigiam a gestão de centenas de parâmetros interdependentes. A ausência de uma especificação formal e de um parser robusto transformava a simples tarefa de configurar uma execução numa fonte potencial de erros e de perda de tempo.

3.3.1 Adoção do Formato YAML e Vantagens Técnicas

Para ultrapassar as limitações impostas pelo formato ASCII ad-hoc, o sistema de configuração foi integralmente redesenhado e modernizado, substituindo o esquema de leitura original pela adoção do padrão YAML (YAML Ain't Markup Language). Esta transição não constitui uma simples transposição do formato, mas uma modernização arquitetónica que redefine a interface entre o utilizador e o software. O formato YAML, amplamente adotado em engenharia de software moderna para ficheiros de configuração e serialização de dados, foi selecionado pela sua ênfase na legibilidade humana, na expressividade e numa estrutura hierárquica explícita e consistente [YAM21]. A modernização permitiu uma integração direta e eficiente com o ecossistema Julia através da biblioteca especializada `YAML.jl`, que efetua a ponte entre a especificação declarativa em texto e estruturas de dados nativas da linguagem.

As principais vantagens técnicas e metodológicas desta modernização incluem:

- **Estrutura hierárquica clara e validável:** A organização dos parâmetros em secções aninhadas (por exemplo, simulação, física, saída) reflete a lógica do domínio e permite uma validação semântica automática, impossível no formato plano e posicional do ASCII.
- **Maior legibilidade e redução de erros:** A sintaxe minimalista, baseada em indentação e em pares chave-valor, é intuitiva e próxima de uma linguagem natural, reduzindo drasticamente erros de sintaxe e facilitando a revisão por parte dos investigadores.
- **Facilidade de extensão e manutenção:** A adição, remoção ou modificação de parâmetros torna-se uma operação local e segura, sem risco de quebrar a leitura de outros campos devido a desalinhamentos de colunas ou delimitadores inconsistentes.
- **Integração programática robusta:** A biblioteca `YAML.jl` converte automaticamente o ficheiro de configuração num dicionário ou estrutura tipada em Julia, permitindo acesso seguro aos parâmetros via código (e.g., `config["simulação"]["passo_temporal"]`) e eliminando a necessidade de escrever e manter rotinas complexas de parsing.
- **Portabilidade e interoperabilidade:** O YAML é um padrão independente de linguagem, o que torna os ficheiros de configuração do AstroTLPlot utilizáveis e compreensíveis noutros contextos (e.g., em scripts Python de pós-processamento) e facilita a documentação dos parâmetros.

Deste modo, o YAML atua como uma camada de abstração bem definida que desacopla totalmente a especificação da execução da simulação do seu código-fonte. Esta separação promove uma configuração flexível, repetível e auto-documentada, alinhando o fluxo de trabalho com as melhores práticas de engenharia de software para investigação reprodutível. A configuração deixa de ser um conjunto obscuro de números num ficheiro de texto para se tornar um documento estruturado que descreve explicitamente os parâmetros do experimento numérico.

3.3.2 Estrutura dos Dados de Configuração

A Tabela 3.1 ilustra a transição estrutural, apresentando um excerto da comparação entre o esquema do AstroPGPLOT em *ASCII* e o novo formato estruturado em *YAML* do AstroTLPlot.

Esta comparação evidencia não apenas uma mudança de sintaxe, mas uma transformação fundamental na organização lógica e na acessibilidade dos parâmetros de configuração. Enquanto o formato *ASCII* original exigia um conhecimento prévio da ordem fixa dos parâmetros e das suas posições no ficheiro, o formato *YAML* introduz uma organização semântica baseada em chaves, tornando cada parâmetro auto-descritivo e independente da sua posição.

ASCII (FORTRAN)	YAML (Julia)
\$debug ldebug=.t./;	Debug: ldebug: true
\$dbase directory='./dados-disk-2000x2000x1/', diratomic='./Atomic/' /	Directories: directory: './dados-disk-2000x2000x1/' diratomic: './Atomic'
\$typesim lhdrun=.true., lmhdrun=.false. /	SimulationType: lhdrun: true lmhdrun: false
\$fileformat lhdf=.true., lascii=.false., lvtk=.false. /	FileFormat: lhdf: true lascii: false lvtk: false
\$gridsize in=2000, jn=2000, kn=1 /	GridSize: in: 2000 jn: 2000 kn: 1
\$labels xlabel='X (pc)', ylabel='Y (pc)', writetime=.true., localizar=.false./	Labels: xlabel: 'X (pc)' ylabel: 'Y (pc)' writetime: true localizar: false

Tabela 3.1: Excerto da comparação entre a estrutura dos dados de configuração nos formatos *ASCII* do (AstroPGPLOT) e *YAML* (AstroTLPlot).

Os blocos de parâmetros definidos no ficheiro de configuração *YAML* incluem:

- **Debug** — ativa ou desativa o modo de depuração;
- **Directories** — define os diretórios de entrada e dados atômicos;
- **SimulationType** — determina o tipo de simulação a executar;
- **FileFormat** — especifica o formato dos ficheiros de dados (HDF5, ASCII, VTK);
- **GridSize** — define as dimensões da grelha de computação tridimensional;
- **Labels** — configura rótulos dos eixos e opções de escrita temporal.

3.3.3 Conclusão Metodológica

A substituição do esquema em *ASCII* do AstroPGPLOT pelo padrão estruturado *YAML* do AstroTLPlot representou, assim, um passo decisivo na modernização e robustificação do fluxo de trabalho de configuração. Enquanto na implementação do AstroPGPLOT a leitura dos parâmetros dependia de um parsing manual de strings com lógica procedimental complexa e sujeita a erros de formatação, no ecossistema Julia

o ficheiro YAML é carregado de forma declarativa, diretamente para uma estrutura de dados nativa (um dicionário ou um tipo composto), através da biblioteca `YAML.jl`.

Esta mudança paradigmática trouxe ganhos tangíveis em várias dimensões: transparência, pois o ficheiro de configuração tornou-se um documento legível e autoexplicativo; robustez, ao eliminar fontes de erros de interpretação e ao permitir validação de esquema; e escalabilidade, ao facilitar a gestão de conjuntos de parâmetros cada vez mais complexos e hierárquicos. Esta evolução técnica e metodológica não apenas melhora a experiência do utilizador, como também estabelece as bases arquitetónicas para o desenvolvimento de um pacote verdadeiramente modular, testável e adaptável a necessidades futuras, alinhando o AstroTLPlot com as melhores práticas de engenharia de software científico contemporâneo.

3.4 Desenvolvimento do Pacote AstroTLPlot

O desenvolvimento do pacote AstroTLPlot, disponível publicamente no repositório GitHub [LCdA25], constituiu o núcleo da implementação prática deste trabalho.

A designação do pacote AstroTLPlot, possui um duplo significado conceptualmente alinhado. A sigla **TL** simboliza, em primeiro lugar, o processo técnico central do projeto: **T**ransition & **L**egacy, ou seja, a transição de um ecossistema de software consolidado (FORTRAN 2008 / PGPlot) para uma plataforma moderna Julia/Makie). Por uma coincidência feliz, as mesmas iniciais correspondem ao nome do autor, Tomás Lima, acrescentando uma dimensão pessoal à identidade da ferramenta. O nome completa-se com **Astro** (Astrofísica) e **Plot** (Visualização Gráfica), especificando assim o seu domínio de aplicação e a sua função primária.

Este pacote agrega num projeto coeso e modular as rotinas fundamentais de leitura, processamento, análise estatística e visualização de dados astrofísicos. A sua conceção foi orientada por princípios de engenharia de software, nomeadamente reutilização, modularidade [WBC⁺17, BEKS17], extensibilidade e facilidade de utilização, assegurando que a ferramenta não se limita a satisfazer as necessidades imediatas da transposição de código, mas serve igualmente como base sólida para futuros desenvolvimentos na área. O pacote integra suporte nativo para múltiplos formatos de dados, incluindo HDF4/5, VTK, CSV e YAML, garantindo ampla compatibilidade com diferentes ferramentas de simulação e fluxos de trabalho existentes no domínio da astrofísica computacional.

3.4.1 Arquitetura e Organização do Pacote AstroTLPlot

O pacote AstroTLPlot foi concebido segundo uma arquitetura modular, alinhada com as boas práticas de engenharia de software científico e de reprodutibilidade [WBC⁺17, SNTH13]. Cada componente do código desempenha uma função claramente definida, o que reduz dependências circulares, simplifica a manutenção e facilita a integração de novas funcionalidades [BEKS17].

A filosofia de *design* assenta num núcleo central, rodeado por módulos especializados, cada um responsável por uma função específica e bem delimitada, refletindo princípios clássicos de modularidade e encapsulamento de software [WBC⁺17].

1. Módulos Funcionais Especializados

A separação funcional do código é ilustrada na Figura 3.2 e materializa-se nos seguintes módulos especializados:

- **Leitura de Dados:** responsável pela gestão de diferentes formatos de ficheiros (HDF4, HDF5, VTK, ASCII, YAML), promovendo interoperabilidade e flexibilidade;
- **Estruturas de Dados:** define as estruturas de dados que encapsulam informação física, parâmetros de execução, configuração de visualização e constantes globais, seguindo práticas de *data encapsulation*;
- **Processamento:** implementa a transformação de dados brutos em resultados científicos, incluindo operações de transformação matricial e cálculo de variáveis físicas;

- **Estatística:** inclui funções de transformação logarítmica básica e executa cálculo de soma, média, desvio padrão, variância e extremos, em consonância com bibliotecas científicas modernas [BEKS17, Per19];
- **Visualização:** responsável pela criação de mapas de calor e PDF com `Makie.jl` [DK21], assegurando desempenho e interatividade em ambiente científico;
- **Suporte:** fornece funções auxiliares e de configuração, reforçando a reutilização e consistência interna.

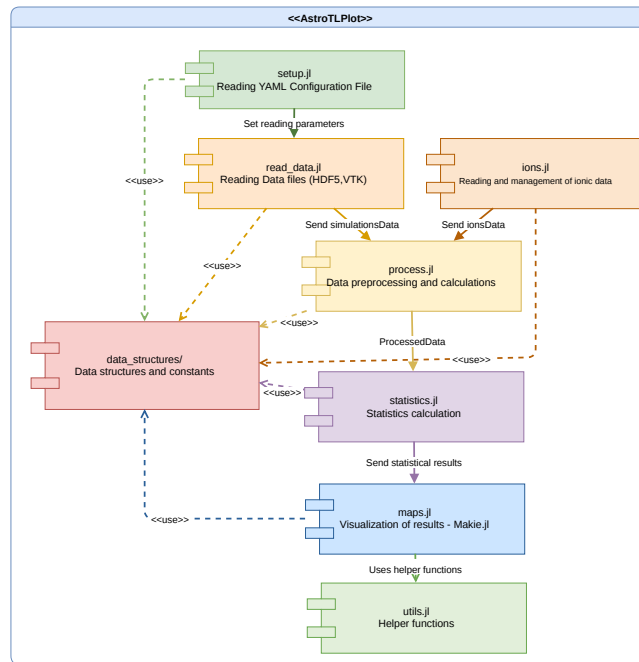


Figura 3.2: Arquitetura modular do pacote AstroTLPlot, evidenciando a separação funcional entre os módulos de leitura, processamento, estatística e visualização.

2. Interface Pública e Encapsulamento

O módulo principal `AstroTLPlot.jl` atua como ponto de integração e de interface pública do pacote, controlando a exposição das funcionalidades. Este modelo garante a modularidade e assegura uma clara separação entre a lógica interna e a API pública [BEKS17]:

- **Exposição de Funções Públicas:** O módulo central reexporta funções e estruturas de dados relevantes dos submódulos (`read_data`, `process`, `maps`, `statistics`, `ions`, `electron`, etc.), permitindo que o utilizador aceda diretamente às funcionalidades essenciais do pacote sem necessidade de conhecer a estrutura interna;
- **Encapsulamento de Funções Privadas:** Funções auxiliares ou de implementação interna permanecem confinadas aos submódulos, não sendo exportadas para o utilizador final. Esta abordagem garante que a interface pública se mantém limpa, intuitiva e estável, enquanto a lógica interna pode evoluir sem comprometer a compatibilidade;
- **Importação e Organização:** Através de declarações `using .Submodule` ou `import .Submodule: function_name`, o módulo principal controla quais funções e tipos ficam visíveis, garantindo a coesão da API.

3. Coerência e Reprodutibilidade

Ao contrário da arquitetura monolítica do software original em FORTRAN 2008, que dependia de variáveis globais e forte acoplamento funcional, o design em Julia adota um modelo orientado a módulos e estruturas tipadas. Esta mudança é metodológica e estabelece as bases para um pacote modular, escalável e de fácil manutenção. A arquitetura do AstroTLPlot promove, assim, a reutilização de código e reforça a reprodutibilidade científica do fluxo de trabalho [WBC⁺17, SNTH13, Per19].

3.4.2 Organização de Ficheiros e Diretórios do Pacote

O pacote AstroTLPlot foi concebido com uma organização modular, promovendo uma separação clara de responsabilidades entre código-fonte, dados, documentação, testes e *notebooks* interativos. Esta estrutura de diretórios segue as boas práticas de engenharia de software científica [WBC⁺17] e foi guiada pelos princípios de Clareza, Escalabilidade e Integração, o que favorece a manutenção, a colaboração e a expansão futura do projeto.

A Figura 3.3 apresenta a visão geral desta arquitetura, evidenciando a independência entre os seus componentes principais e o fluxo lógico entre os diferentes níveis do pacote:

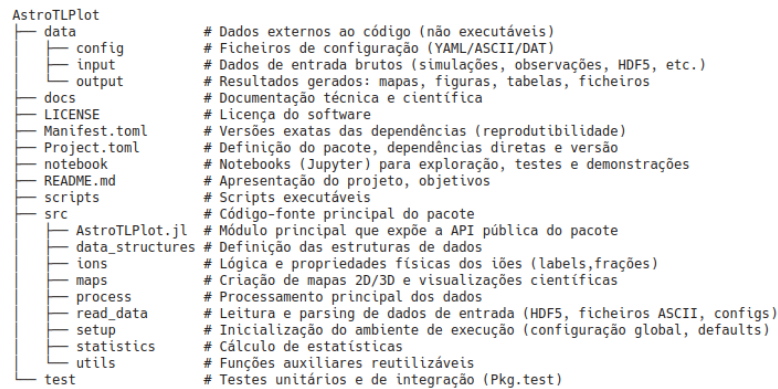


Figura 3.3: Estrutura modular de diretórios do pacote AstroTLPlot.

3.4.3 Justificação Técnica do Design Modular

A adoção de um design modular representou uma evolução técnica fundamental em relação ao AstroPGPLOT, permitindo a separação explícita de responsabilidades e a eliminação da dependência de variáveis globais. Para mitigar esta limitação, foram implementadas estruturas de dados tipadas, como `config_data` e `simulation_data`. Estas estruturas centralizam parâmetros e resultados, garantindo a consistência dos dados e promovendo a reutilização de código, em conformidade com as recomendações atuais para o desenvolvimento sustentável de software científico [Per19]. O módulo `read_config` reforça esta abordagem, fornecendo flexibilidade na parametrização de simulações e permitindo a adaptação transparente a diferentes cenários.

Graças a este modelo orientado a dados, as funções públicas recebem os *datasets* como parâmetros explícitos, tornando-as independentes do estado global e aplicáveis em múltiplos contextos (ex.: otimizando a análise através de chamadas como `statistics(simulation_data.den)`). O design modular e a eliminação da utilização excessiva de variáveis globais resultam em ganhos significativos de robustez, extensibilidade e clareza, superando as fragilidades da abordagem monolítica e altamente acoplada da implementação do AstroPGPLOT.

Deste modo, o AstroTLPlot não se limita a uma simples transposição de código FORTRAN 2008 para Julia, evoluindo para uma ferramenta moderna, modular e escalável, concebida para a comunidade científica

e preparada para futuras extensões.

3.5 Estruturas de Dados Implementadas e Organização

O desenvolvimento do pacote AstroTLPlot assenta num conjunto de estruturas de dados que centralizam os parâmetros de execução, a configuração da visualização e os resultados numéricos das simulações, seguindo as boas práticas de engenharia de software científico [WBC⁺17, BEKS17]. A definição clara e rigorosa destas estruturas em Julia revelou-se crucial, uma vez que toda a transposição das rotinas de FORTRAN 2008 e o desenvolvimento subsequente do pacote dependem diretamente da sua consistência e organização.

As estruturas foram concebidas com tipagem explícita e modularidade, princípios fundamentais no *design* moderno de software técnico-científico [SNTH13, Per19]. Esta abordagem promove maior clareza conceptual, facilita a reutilização do código, simplifica a manutenção e assegura a robustez das operações sobre os dados, para além de favorecer a rastreabilidade e transparência dos resultados. De forma a representar graficamente esta organização, a Figura 3.4 ilustra a arquitetura geral das estruturas de dados do pacote AstroTLPlot, evidenciando as relações hierárquicas e de dependência entre os seus principais módulos.

O diagrama, elaborado com base no modelo implementado em Julia, mostra as relações de dependência entre os módulos centrais: os módulos Runtime, ModIons, PGP, Config e Statistics utilizam o DataStructures, sendo que os quatro primeiros dependem adicionalmente das constantes fundamentais definidas no núcleo do sistema. A estrutura ConfigData do módulo read_config.jl integra todos os módulos Runtime, ModIons, PGP e Config. Esta representação gráfica complementa a descrição textual apresentada nas listagens, permitindo uma compreensão global da arquitetura interna do pacote e da integração entre os diferentes módulos de dados [SNTH13, Per19].

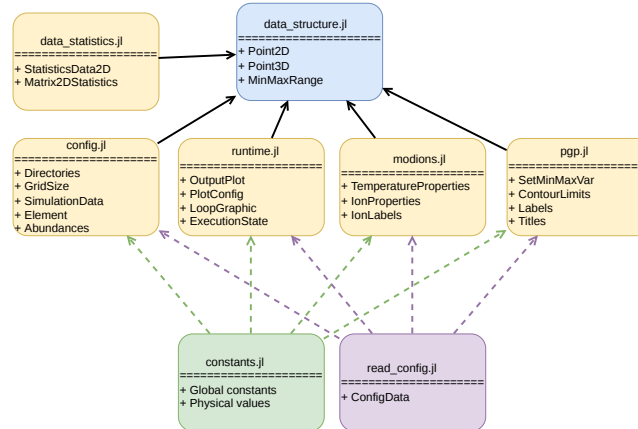


Figura 3.4: Arquitetura das estruturas de dados do pacote AstroTLPlot.

A organização das estruturas de dados no pacote segue uma separação clara de responsabilidades, distribuída pelos seguintes módulos funcionais:

- `runtime.jl`: agrega os parâmetros de execução da simulação, incluindo opções de configuração de gráficos, estado da simulação, transformações de dados, rastreamento, volumes, ficheiros temporais e dados adicionais. Esta estrutura atua como núcleo do estado da execução;
- `pgp.jl`: gere a configuração de escalas de cor, limites de contorno, visualizações, metadados gráficos e utilizados na definição de legendas e normalização dos mapas;
- `modions.jl`: concentra as propriedades iónicas, frações de iões, propriedades de temperatura, e limites de visualização, sendo fundamental para a análise química das simulações;

- `data_structures.jl`: contém definições gerais de pontos em 2D e 3D, limites de eixos, coordenadas e estatísticas de matrizes multidimensionais, servindo como base para o cálculo e visualização (adição nesta versão);
- `data_statistic.jl`: contém definições gerais para estatísticas de matrizes 1D, 2D e 3D (adição nesta versão);
- `config.jl`: fornece suporte para as configurações gerais e constantes referentes a mensagens de erros e mensagens de estado;
- `plotting_config.jl`: fornece suporte para configurações de visualização (adição nesta versão);
- `read_config.jl`: suporte para leitura de dados de configurações lidas a partir do ficheiro de entrada (adição nesta versão).

3.5.1 Constantes Físicas e Códigos de Estado

A definição destas constantes em estruturas dedicadas permite garantir a consistência científica dos cálculos, assegurando que todos os módulos partilham valores unificados e compatíveis entre si, uma exigência fundamental em simulações computacionais de elevada precisão [Hig02].

Listagem 3.5.1 Excerto do módulo `constants.jl`.

```

1  const GAMMA = 5.0 / 3.0           # Adiabatic index
2  const GAMMA1 = GAMMA - 1.0       # GAMMA minus one
3  const BOLTZ = 1.38065e-16        # Boltzmann constant (erg/K)
4  const CLIGHT = 2.99792458e+10    # Speed of light in vacuum (cm/s)
5  const HPL = 6.62606896e-27       # Planck's constant (erg·s)
6  # =====
7  # Particle Masses
8  # =====
9  const ELMA = 9.10938e-28         # Electron mass (g)
10 const MP = 1.67262e-24           # Proton mass (g)
11 const MH = 1.67353e-24          # Hydrogen atom mass (g)
12 const ME = 9.1093837015e-28
13 # =====
14 # Other Constants
15 # =====
16 const SMALLP = 1.0e-30           # Minimum pressure (g/cm·s²)
17 const CMPC = 3.086e18            # Megaparsec to cm conversion factor
18 ...
19 # Error Codes
20 const ERROR_UNKNOWN = 1          # 1 = Unmapped/Generic Error (Fallback)
21 const ERROR_FILE_NOT_FOUND = 2   # 2 = File Not Found
22 const ERROR_NO_PERMISSIONS = 3   # 3 = No Permissions
23 ...
24 # Dictionary of status messages (code => message)
25 const ERROR_MESSAGES_DICT = Dict{Int, String}({
26 # --- Status Code ---
27 STATUS_SUCCESS      => "File loaded successfully.",
28 ...
29 # --- Format-Specific Read Failures (Codes 6-8) ---
30 ERROR_HDF5_READ_FAIL => "Code 6: Failed to read data structure within the opened HDF5 file.",
31 ...

```

A existência de uma camada explícita de verificação e diagnóstico contribui diretamente para a robustez, portabilidade e reprodutibilidade do processo de execução, aspetos centrais na credibilidade dos *workflows* científicos modernos [War13]. Assim, tanto as rotinas de baixo nível como os módulos de visualização e análise estatística operam sobre um conjunto coeso de definições globais, reduzindo a propagação de erros

e melhorando a rastreabilidade das operações. A Listagem 3.5.1 apresenta um excerto representativo do módulo `constants.jl`.

Nela observa-se a organização hierárquica das constantes físicas, agrupadas por categorias como massas, constantes fundamentais ou valores mínimos admissíveis, bem como a estrutura centralizada dos códigos de erros. Estes códigos são acompanhados por um dicionário (`ERROR_MESSAGES_DICT`) que mapeia cada identificador numérico para a respetiva mensagem descritiva, permitindo a sua utilização direta nas rotinas de leitura, validação e registo de eventos.

3.5.2 Estrutura de Configuração de Simulação (ConfigData)

A estrutura `ConfigData` reúne todos os parâmetros de configuração da simulação, extraídos diretamente dos ficheiros YAML utilizados para preparar o fluxo computacional. A opção por formatos legíveis e estruturados, como YAML, segue práticas modernas de definição declarativa de parâmetros em projetos científicos complexos [HBO10]. Esta estrutura é instanciada a partir do ficheiro `indatpg.yaml` e funciona como o núcleo de configuração do pacote, sendo passada como argumento a praticamente todas as funções de leitura, processamento e visualização.

Este modelo promove um elevado grau de encapsulamento, garantindo que cada função acede apenas à informação necessária, uma abordagem recomendada em sistemas de alto desempenho e *pipelines* bem modularizados [War13]. Ao centralizar os parâmetros, minimizam-se efeitos colaterais, aumenta-se a rastreabilidade e reforça-se a reprodutibilidade do processo [War13]. A Listagem 3.5.2 apresenta um excerto representativo da estrutura `ConfigData`.

Listagem 3.5.2 Estrutura de dados `ConfigData`.

```

1  mutable struct ConfigData
2      debug::Debug           # Debugging configuration and runtime flags
3      directories::Directories # Paths for simulation input/output data
4      simulation_type::SimulationType # Defines the type of simulation (HD, MHD, etc.)
5      file_format::FileFormat  # Selected file format(s) for data I/O
6      grid_size::GridSize      # Grid resolution
7      real_dims::RealDims      # Physical dimensions mapped to the computational domain
8      maps_dims::MapsDims      # Dimensions used for map visualization and slicing
9      number_of_plots::NumberOfPlots # Number and configuration of plots to generate
10     variable_params::VariableParams # Variables to compute, store, or visualize
11     interpolation::Interpolation  # Interpolation settings for post-processing
12     abundances::Abundances      # Elemental or molecular abundances considered
13     scales::Scales              # Scaling and normalization parameters
14     ions_plot::IonsPlot         # Ion-specific plotting configuration
15     ions_type::IonsType         # Types of ions involved in the simulation
16     atomic_ionic_fraction::AtomicIonicFraction # Fractions of atomic and ionic species
17 end

```

3.5.3 Dados de Simulação (SimulationData)

A estrutura `SimulationData` armazena os campos físicos tridimensionais resultantes das simulações, densidade, energia, temperatura, velocidades e campo magnético, além dos vetores de grelha correspondentes. Estes dados são tipicamente importados de ficheiros HDF5, um formato amplamente utilizado para armazenamento estruturado e de elevado desempenho em contextos científicos [The23a]. A organização interna da estrutura facilita a indexação espacial, a manipulação eficiente dos campos e o acesso rápido às variáveis físicas, servindo como base para a análise estatística, visualização com *Makie* ou exportação em formatos derivados.

Esta forma de encapsulação segue recomendações clássicas na modelação numérica e análise de dados multidimensionais [PTVF07]. A Listagem 3.5.3 mostra um excerto da estrutura `SimulationData`.

Listagem 3.5.3 Estrutura de dados SimulationData.

```

1 mutable struct SimulationData
2     den::Array{Float64,3} # Density field
3     ene::Array{Float64,3} # Energy field
4     pre::Array{Float64,3} # Pressure field
5     tem::Array{Float64,3} # Temperature field
6     vxx::Array{Float64,3} # Velocity X-component
7     vyy::Array{Float64,3} # Velocity Y-component
8     vzz::Array{Float64,3} # Velocity Z-component
9     bxx::Array{Float64,3} # Magnetic field X-component
10    byy::Array{Float64,3} # Magnetic field Y-component
11    bzz::Array{Float64,3} # Magnetic field Z-component
12    X_grid::Vector{Float64} # Grid coordinates for X-axis
13    Y_grid::Vector{Float64} # Grid coordinates for Y-axis
14    Z_grid::Vector{Float64} # Grid coordinates for Z-axis
15 end

```

3.5.4 Dados Estatísticos (StatisticsData2D)

As estruturas StatisticsData2D, Coordinate2DResult, Matrix2DStatistics e Axis2DLimits organizam resultados estatísticos derivados de matrizes bidimensionais. Permitem calcular métricas fundamentais, soma, média, variância, desvio padrão, valores extremos e limites de eixos, seguindo abordagens clássicas da análise estatística computacional [Hig02].

Estas estruturas servem de base para operações como cortes 2D, projeções, normalizações e construção de mapas científicos de elevada precisão. A Listagem 3.5.4 apresenta um excerto representativo.

Listagem 3.5.4 Estruturas de dados estatísticos auxiliares.

```

1 # Represents a 2D coordinate and its value
2 mutable struct Coordinate2DResult{T}
3     index2D::Point2D{Int} # x,y index
4     value2D::T # value at that index
5 end
6 # Represents axis limits in 2D space
7 mutable struct Axis2DLimits
8     p2min::Point2D{Int} # min x,y indices
9     p2max::Point2D{Int} # max x,y indices
10 end
11 # Represents statistical data from a 2D matrix
12 mutable struct StatisticsData2D{T<:Real}
13     sum::T # The sum of all elements in the matrix
14     mean::T # The average value of the elements
15     variance::T # The variance of the elements
16     std_dev::T # The standard deviation
17     min_value::Coordinate2DResult{T} # Minimum value with its position
18     max_value::Coordinate2DResult{T} # Maximum value with its position
19     range_value::MinMaxRange{T} # Range (min, max) of values
20     n_elements::Int # The total number of elements
21     n_rows::Int # The number of rows
22     n_columns::Int # The number of columns
23     axis_limits::Axis2DLimits # Axis boundaries for the matrix
24 end
25 mutable struct Matrix2DStatistics{T<:Real}
26     statistics_data::StatisticsData2D{T} # Statistical descriptorst
27     matrix_result::Matrix{T} # Matrix after applying a function
28 end

```

3.5.5 Vantagens da Utilização de Estruturas de Dados no Desenvolvimento do Projeto

A adoção de estruturas de dados no desenvolvimento do pacote permitiu reorganizar de forma sistemática os elementos da simulação, assegurando que cada componente do código possui uma responsabilidade clara e independente. Esta abordagem foi fundamental para transpor a lógica monolítica do software AstroPGPLOT para uma implementação modular e idiomática em Julia, facilitando a manutenção, a extensibilidade e a consistência dos dados ao longo de todo o fluxo de execução. Neste contexto, podem ser destacadas as vantagens decorrentes da utilização de estruturas de dados:

- Redução de utilização de variáveis globais, com funções explicitamente dependentes de argumentos de entrada.
- Encapsulamento de dados relacionados, facilitando a manutenção e expansão do pacote.
- Validação automática de tipos e consistência dos dados durante a execução.
- Integração direta com funções de leitura, processamento e visualização, respeitando a filosofia modular do pacote.

O processo de transposição estrutural do código, exigiu uma revisão cuidadosa da arquitetura de módulos e das estruturas de dados, de forma a adaptar o design em FORTRAN 2008 aos paradigmas modernos da linguagem Julia [BEKS17]. Esta reformulação seguiu os princípios da engenharia de software científica, com enfoque na clareza, modularidade e capacidade de manutenção. Durante esta fase, algumas estruturas foram simplificadas, integradas ou removidas, quando a sua continuidade em Julia não se justificava. Um exemplo é o módulo `kind`, utilizado AstroPGPLOT para definir tipos numéricos (precisão simples, dupla, etc.). Em Julia, este controlo de precisão é intrínseco à linguagem, eliminando a necessidade de um módulo específico.

De forma semelhante, estruturas concebidas para contornar limitações da linguagem FORTRAN 2008, como a gestão manual de arrays com dimensões fixas ou o agrupamento de variáveis dispersas, foram substituídas por módulos mais genéricos e abstratos, assegurando maior coesão e reutilização de código. Paralelamente, foram introduzidas novas estruturas de dados, como a `read_config`, destinadas a otimizar a configuração e o controlo do fluxo de entrada e saída de informação. Estas estruturas promovem uma separação clara entre a lógica de processamento e os detalhes de *I/O*, garantindo maior flexibilidade e extensibilidade.

Esta abordagem resulta num código mais idiomático em Julia, aderente às boas práticas da linguagem, e às normas de programação científica reproduzível [SNTH13], reduzindo redundâncias e aumentando a legibilidade e manutenção do projeto. Além disso, estabelece uma base sólida para futuras evoluções do pacote AstroTLPlot, assegurando a sua robustez e escalabilidade no contexto de aplicações científicas complexas.

3.5.6 Estruturas AstroPGPLOT (FORTRAN 2008) vs Estruturas AstroTLPlot (Julia)

No software AstroPGPLOT, implementado em FORTRAN 2008, os dados são organizados numa arquitetura baseada em módulos (por exemplo, *module config*, *module kinds*) que funcionam como repositórios globais de variáveis e arrays alocados. Esta opção de desenho facilita a passagem de variáveis e arrays na assinatura das rotinas, e evita deliberadamente o uso de *Derived Types (equivalentes a structs)*, com o objetivo de não comprometer a vetorização automática do código e, em particular, o *unroll* de ciclos. Esta abordagem prioriza a otimização de baixo nível e o desempenho de cálculo puro, centralizando toda a configuração e os dados físicos em unidades acessíveis globalmente.

A Listagem 3.5.5 apresenta um excerto representativo do módulo `config` original em FORTRAN 2008, que agrega parâmetros de execução, escalas físicas, variáveis de densidade, temperatura, campos magnéticos e configurações iónicas. Esta estrutura evidencia uma arquitetura monolítica e fortemente acoplada, onde a lógica de domínio, os parâmetros de controlo e os dados de simulação coexistem no mesmo espaço de nomes global.

Este paradigma, embora eficiente em tempo de execução para cálculos vetoriais, introduz desafios signi-

ficativos em termos de modularidade, manutenibilidade e rastreabilidade de dependências. Foi precisamente esta característica que motivou a reformulação completa na arquitetura do AstroTLPlot, que adota uma filosofia oposta e complementar: estruturas de dados tipadas (structs) e módulos com estado encapsulado. Esta transição representa uma mudança de paradigma de um modelo orientado à otimização de ciclos para um modelo orientado à engenharia de software, mantendo o desempenho através dos mecanismos de especialização e inferência de tipos do Julia.

Listagem 3.5.5 Estrutura do módulo config do AstroPGPLOT em FORTRAN 2008.

```

1  module config
2  ! debug flag
3  logical :: ldebug
4  ! database location
5  character(len=90):: directory,diratomic
6  logical :: lhdf, lascii,lvtk
7  logical :: lhdrun,lmhdrun
8  logical :: ldens,ltemp,lpres,lpram,lmagn,lentr,lmach,lions,lratios,lele,lele_kern
9  integer :: in, jn, kn
10 integer :: imin, imax, jmin, jmax, kmin, kmax
11 real :: x1, x2, y1, y2, z1, z2, dx, dy, dz
12 real :: xmin, xmax, ymin, ymax, zmin, zmax
13 character(2):: units
14 real, dimension(:,:,:), allocatable, save :: den, ene, pre, pok, tem
15 real, dimension(:,:,:), allocatable, save :: vxx, vyy, vzz, vel2
16 real, dimension(:,:,:), allocatable, save :: bxx, byy, bzz
17 real, dimension(:,:,:), allocatable, save :: sentrop,ramp,rampok
18 real, dimension(:,:), allocatable, save :: dentot
19 integer:: nfiles,nfile_start,nfile_end,jump,nl,numx,numy
20 ! Scale the data
21 real :: timescale,bscale,denscale,temscale,velscale,elescale
22 logical:: logs
23 ! Ions
24 ! We use maxkern to set variables which have to be defined
25 ! for all elements aet up initially with lhyd,lhe,lcar,lnit,...
26 integer,parameter::max_atomic_number=26
27 logical ,dimension(1:max_atomic_number):: zelem
28 real ,dimension(:),allocatable,save:: abund
29 ! Set the variables that will be defined according the maximum Z and
30 ! number of elements in use. That is, kernmax e nelem
31 integer:: nelem
32 integer:: kernmax
33 ! Elements ids... 1=H, 2=He, 3=C,...
34 integer,dimension(:),allocatable,save:: idk,idkmin,idkmax
35 logical:: lhyd, lhel,lcar,lnit,loxy,lne,lmg,lsil,lsul,lar,lfe
36 ! Define the elements to plot images
37 logical:: plhyd, plhel,plcar,plnit,ploxy,plne,plmg,plsil,plsul,plar,plfe
38 logical,dimension(1:26)::celem,plelem
39 ! Abundances
40 logical :: lallen,lag89,lasplund,lgas07,lagss09
41 ! Metallicity and depletion
42 real:: zmetal,deplt
43 ! Atomic and ionic fractions
44 integer:: id,ncool,ntemp_spline
45 real:: dlog_temp
46 ! Type of interpolation. Note, ntemp_spline=ncool-1 and starts in 0
47 logical::lspline,lrloss
48 end module config

```

3.5.7 Comparação e Benefícios da Nova Abordagem

A transição do software AstroPGPLOT (FORTRAN 2008) para a implementação modular do AstroTLPlot (Julia) trouxe melhorias significativas na organização e na gestão de dados. Enquanto no AstroPGPLOT as estruturas eram monolíticas e dependiam fortemente de variáveis globais compartilhadas, uma escolha deliberada para otimização de vetorização, mas um fator de risco para dependências ocultas, em Julia cada struct encapsula de forma explícita apenas os dados do seu domínio, promovendo a clareza e a independência funcional. Como benefício direto, as funções passam a receber instâncias de estruturas tipadas (como `SimulationData` ou `ConfigData`) como parâmetros explícitos, permitindo que operem de forma independente, rastreável e testável, sem dependerem de estado global implícito.

A Tabela 3.2 sintetiza as diferenças estruturais e metodológicas entre o código original em FORTRAN 2008 e a reimplementação em Julia, evidenciando as vantagens da adoção de estruturas modulares e da eliminação de variáveis globais. Esta comparação destaca as melhorias alcançadas ao nível da modularidade, reutilização, manutenção e clareza computacional, fundamentais para a evolução futura do pacote.

Tabela 3.2: Comparação entre as abordagens de estruturação de dados no AstroPGPLOT (FORTRAN) e no AstroTLPlot (Julia).

Característica	AstroPGPLOT (FORTRAN 2008)	AstroTLPlot (Julia)
Paradigma de dados	Módulos globais com variáveis compartilhadas, evitando Derived Types para vetorização	Structs tipadas com estado encapsulado, promovendo composição
Acesso a dados	Global e implícito (via USE)	Explícito, via instâncias passadas como parâmetros
Modularidade	Baixa (acoplamento forte a dados globais)	Elevada (módulos e structs independentes)
Reutilização de código	Limitada pelo contexto global	Alta (funções aplicáveis a qualquer instância compatível)
Manutenção e testes	Complexa devido a dependências ocultas	Facilitada por testes unitários e interfaces bem definidas

A estrutura modular do AstroPGPLOT (FORTRAN 2008), concebida para otimizar a vetorização através da partilha global de arrays, foi reavaliada e reformulada na transposição para Julia. A sua natureza monolítica e a centralização excessiva de variáveis motivaram uma reestruturação profunda. As variáveis dispersas foram reagrupadas em structs tipadas e coerentes, como `SimulationData`, `ConfigData` e `StatisticsData2D`, cada uma responsável por um domínio específico (configuração, dados físicos, visualização, etc.). Esta nova organização, que substitui a lógica baseada em módulos globais por uma arquitetura de componentes encapsulados, estabeleceu uma base consistente para o pacote AstroTLPlot. Os resultados são ganhos tangíveis em modularidade, reutilização, manutenção e clareza computacional. Enquanto o design original privilegiava o desempenho de baixo nível, o novo paradigma privilegia a composição, a testabilidade e a sustentabilidade a longo prazo, sem comprometer o desempenho graças ao sistema de tipos e à compilação JIT *Just-In-Time* (JIT) da linguagem Julia.

3.6 Transposição e Reimplementação de Algoritmos FORTRAN em Julia

A transposição dos algoritmos e rotinas desenvolvidos no AstroPGPLOT (FORTRAN 2008) constituiu o núcleo central deste trabalho. O processo não se limitou a uma simples tradução de código, mas implicou uma análise abrangente, adaptação e modernização, de forma a garantir que a versão em Julia do AstroTLPlot preservasse a fidelidade científica e a precisão numérica do software original, ao mesmo tempo que explorava as potencialidades oferecidas pela nova linguagem e pelo seu ecossistema moderno.

[BEKS17, WBC⁺17].

3.6.1 Implementação em Julia e Principais Desafios

A transposição das rotinas do AstroPGPLOT (FORTRAN 2008) para Julia implicou a superação de diversos desafios relacionados com as diferenças de paradigma, estrutura e modelo de execução entre as duas linguagens. Cada situação identificada exigiu decisões de *design* orientadas por critérios de clareza, desempenho e fiabilidade científica, garantindo a consistência dos resultados e aproveitando o ecossistema de Julia [BEKS17]. A análise dos principais desafios técnicos e das correspondentes soluções de implementação está sintetizada na Tabela 3.3.

Tabela 3.3: Principais desafios na transposição de FORTRAN 2008 para Julia e soluções implementadas.

Desafio	Abordagem em FORTRAN (AstroPGPLOT)	Solução em Julia (AstroTLPlot)
Indexação de arrays	Indexação base 1, com possível redefinição do limite inferior (e.g., para 0).	Manutenção da indexação base 1, com transformações explícitas para compatibilidade quando necessário.
Tipos de dados	Declaração explícita com dimensões fixas e precisão definida (e.g., REAL*8).	Uso de Float32/Float64 e tipos paramétricos (T) para flexibilidade e reutilização.
Estruturação de dados	Arrays multidimensionais e variáveis globais partilhadas em módulos.	Structs tipadas que encapsulam parâmetros, dados e resultados, promovendo modularidade.
Fluxo de execução e dependências	Rotinas dependentes de variáveis globais, com acoplamento forte.	Funções modulares com argumentos explícitos, explorando <i>multiple dispatch</i> .
Ciclos e operações sobre arrays	Laços aninhados explícitos para iterar sobre arrays.	Substituição por <i>broadcasting</i> e operações vetorizadas, quando possível, para clareza e eficiência.
Leitura de ficheiros (HDF4/5)	Dependência de bibliotecas externas específicas.	Utilização de pacotes modernos do ecossistema Julia (HDF5.jl, YAML.jl).
Robustez numérica	Cálculos sensíveis (e.g., logarítmicos) sem verificação sistemática.	Introdução de verificações preventivas e funções auxiliares para evitar erros numéricos.
Tratamento de erros	Falhas abruptas sem mecanismos de recuperação ou diagnóstico.	Uso de <i>exception handling</i> (try...catch) para capturar erros e permitir execução controlada.

A superação destes desafios não só assegurou a transposição rigorosa das rotinas em FORTRAN 2008, como também resultou numa implementação mais robusta, modular e sustentável, em conformidade com as boas práticas de engenharia de software científico [SNTH13, WBC⁺17]. Esta abordagem estabeleceu uma base sólida para a evolução futura do pacote AstroTLPlot e para a integração de novas funcionalidades.

3.6.2 Visualização Científica em Julia com Makie

A visualização científica constitui um pilar fundamental do pacote AstroTLPlot, representando uma evolução paradigmática em relação ao sistema gráfico estático utilizado no software anterior. A adoção da biblioteca Makie.jl [DK21] como motor de renderização permitiu transformar a visualização de um mero produto final num instrumento dinâmico e integrado para a exploração e análise de dados astrofísicos.

O Makie foi selecionado pelo seu desempenho, flexibilidade e integração nativa com o ecossistema Julia. A sua arquitetura, que suporta aceleração por GPU e renderização interativa, permite construir visualizações complexas diretamente a partir das estruturas de dados tipadas do AstroTLPlot (e.g., *SimulationData*, *StatisticsData2D*), assegurando uma correspondência direta e consistente entre os dados numéricos e a sua representação gráfica.

As principais capacidades e vantagens desta integração incluem:

- **Integração Tipo-Segura com Dados do Pacote:** As visualizações são geradas programaticamente a partir das structs do AstroTLPlot, eliminando a necessidade de conversões manuais e garantindo que a figura reflecte o estado preciso dos dados calculados.
- **Automatização e Composição de Visualizações Complexas:** Através de funções modulares, é possível gerar automaticamente painéis multi-figura, séries temporais a partir de pipelines de dados reproduzíveis.
- **Interatividade Nativa para Análise Exploratória:** A capacidade de interagir com as figuras (zoom, rotação, seleção dinâmica de cortes, uso de sliders) transforma a visualização numa ferramenta ativa de investigação, permitindo uma compreensão mais intuitiva de estruturas complexas em dados multidimensionais.
- **Reprodutibilidade e Exportação para Publicação:** As cenas gráficas podem ser renderizadas de forma determinística e exportadas em formatos vetoriais (PDF, SVG) ou de raster (PNG) de alta qualidade, facilitando diretamente a produção de figuras para artigos científicos.

A implementação com Makie transcende, portanto, a funcionalidade básica de *plotting*. Ela estabelece um pipeline de visualização moderna, interativa e integrada no fluxo de trabalho computacional, alinhando o AstroTLPlot com as melhores práticas contemporâneas de ciência aberta, reproduzível e exploratória [DK21, SNTH13].

3.7 Transposição de Rotinas: Exemplos e Análise Comparativa

Esta secção apresenta exemplos ilustrativos de rotinas transpostas do software AstroPGPLOT (FORTRAN 2008) para o pacote AstroTLPlot (Julia), evidenciando as alterações de implementação, os ganhos em modularidade e legibilidade, e a integração com o novo sistema de tipos e estruturas de dados. Os exemplos permitem compreender de forma concreta o impacto das decisões de transposição e a evolução das rotinas na nova linguagem.

Entre as rotinas mais relevantes transpostas destacam-se as associadas à leitura e validação de parâmetros de configuração, ao processamento numérico de dados astrofísicos e à geração da pipeline de visualizações científicas com Makie. Esta transição permitiu substituir a lógica puramente procedimental do AstroPGPLOT por uma abordagem funcional e orientada a dados, onde o sistema de *multiple dispatch* do Julia otimiza a execução de acordo com o tipo de estrutura processada.

3.7.1 Comparação entre a leitura de configuração no AstroPGPLOT (FORTRAN) e no AstroTLPlot (Julia)

A leitura e interpretação dos parâmetros de execução constituem um dos primeiros elementos de contacto entre o utilizador e o sistema de simulação. No software AstroPGPLOT, esta operação era realizada a partir de um ficheiro de configuração no formato ASCII (.dat), denominado *indatpgp3.dat*. A leitura de parâmetros de configuração era efetuada através de *namelists*, uma funcionalidade nativa do FORTRAN

que permite para agrupar variáveis relacionadas em blocos nominados. O ficheiro de configuração incluía secções distintas, como /debug/, /dbase/ e /typesim/, que definiam parâmetros de depuração, caminhos para dados e tipo de simulação a executar.

A Listagem 3.7.1 apresenta um excerto da subrotina `namelists`, ilustrando a estrutura sequencial de leitura e a forte dependência de variáveis globais previamente declaradas nos módulos, característica típica do paradigma procedimental e de estado global do FORTRAN 2008.

Listagem 3.7.1 Excerto da sub-rotina `namelists` no código-fonte AstroPGPLOT (FORTRAN 2008).

```

1  subroutine namelists
2      use config
3      use pgp
4      use runtime
5      use modions
6      implicit none
7      ! Debug outputs
8      namelist/debug/ldebug
9      ! Directories where the data cubes are and where the atomic data is
10     namelist/dbase/directory,diratomic
11     ! Run type
12     namelist/typesim/lhdrun,lmhdrun
13     ! Type of datafile hdf, ASCII or VTK
14     namelist/fileformat/lhdf,lascii,lvtk
15     ! Grid definitions
16     namelist/gridsz/in, jn, kn
17     ! Domain dimensions
18     namelist/realdims/xmin, xmax, ymin, ymax, zmin, zmax,units
19     ! Plotting domain dimensions
20     namelist/mapsdims/x1, x2, y1, y2, z1, z2
21     ! (... vários outros blocos ...)
22     open(1,file='indatpgp',status='old') ! Open namelists input file
23     read (1,debug)
24     write(2,debug)
25     ! (... leitura sequencial de cada NAMELIST ...)
26     close(1)
27 end subroutine namelists

```

Com a transposição para Julia, procedeu-se a uma modernização estrutural deste mecanismo. O ficheiro de configuração passou a utilizar o formato YAML (*YAML Ain't Markup Language*), amplamente adotado em software científico e aplicações de engenharia de dados pela sua legibilidade, hierarquia explícita e suporte nativo a diferentes tipos de dados [WBC⁺17]. Esta alteração não só simplificou a edição manual dos ficheiros, como permitiu uma integração mais direta com as estruturas de dados tipadas utilizadas no pacote AstroTLPlot.

A função `load_simulation_config` foi reimplementada para ler e interpretar diretamente o conteúdo do ficheiro `.yaml`, povoando de forma automática as estruturas de dados `ConfigData`, `PGPData`, `RuntimeData` e `ModionsData`. A rotina auxiliar, `open_file` implementa um sistema robusto de abertura e validação de ficheiros em múltiplos formatos: (`.yaml`, `.hdf5`, `.csv`, `.vtk`), assegurando verificações rigorosas de existência, permissões e integridade antes da leitura dos dados. A Listagem 3.7.2 apresenta um excerto representativo desta implementação.

Listagem 3.7.2 Excerto da função genérica `open_file` em Julia.

```

1 function open_file(file_path::String)
2     # 1. Check if the file exists (Maps to ERROR_FILE_NOT_FOUND = 2)
3     if !isfile(file_path)
4         return ERROR_FILE_NOT_FOUND
5     end
6     # 2. Check read permissions (Maps to ERROR_NO_PERMISSIONS = 3)
7     # Using try/catch for stat() to handle complex I/O failures before HDF5/CSV tries
8     file_stat = try
9         stat(file_path)
10    catch
11        return ERROR_NO_PERMISSIONS
12    end
13
14    if !(file_stat.mode & 0o444 != 0) # Check if the file has read permissions
15        return ERROR_NO_PERMISSIONS
16    end
17
18    # 3. Check file extension and determine format
19    file_ext = splitext(file_path)[2]
20
21    # --- Data Formats ---
22
23    # Check for HDF4 file (.hdf)
24    if file_ext == ".hdf"
25        try
26            raw_data = HDF4.hdfopen(file_path, "r")
27            return raw_data
28        catch
29            return ERROR_HDF4_READ_FAIL # New Error Code 7
30        end
31
32    # Check for HDF5 file (.h5 or .hdf5)
33    elseif file_ext == ".h5" || file_ext == ".hdf5"
34        try
35            raw_data = HDF5.h5open(file_path, "r")
36            return raw_data
37        catch
38            return ERROR_HDF5_READ_FAIL # New Error Code 6
39        end
40
41    # Check for YAML file (.yaml or .yml)
42    elseif file_ext == ".yaml" || file_ext == ".yml"
43        try
44            parsed_data = YAML.load_file(file_path)
45            return parsed_data
46        catch
47            return ERROR_YAML_PARSING # Error Code 4
48        end
49    ...
50    # Check for VTK file (.vtk)
51    elseif file_ext == ".vtk"
52        # VTK file handling might require a specific library and logic,
53        # but for now, we can just return a placeholder or handle the error.
54        return ERROR_UNSUPPORTED_TYPE # Or custom VTK failure code
55
56    # Unsupported file type (Maps to ERROR_UNSUPPORTED_TYPE = 9)
57    else
58        return ERROR_UNSUPPORTED_TYPE
59    end
60 end

```

Esta abordagem modular separa claramente as responsabilidades de validação, leitura e interpretação dos dados, melhorando a manutenibilidade e a clareza do código, princípios fundamentais de engenharia de software científico [WBC⁺17, SNTH13]. Além disso, o tratamento explícito de códigos de erros (por exemplo, `ERROR_FILE_NOT_FOUND`, `ERROR_YAML_PARSING`) contribui para uma execução mais segura e previsível, reduzindo falhas silenciosas comuns em linguagens de baixo nível.

A Listagem 3.7.3 apresenta a nova função `load_simulation_config`, que implementa a leitura modular e validada de parâmetros a partir de ficheiros YAML, eliminando a dependência de variáveis globais e reforçando a integridade dos dados. Cada conjunto de parâmetros é lido e mapeado para a estrutura de dados tipada (`ConfigData`), eliminando a necessidade de variáveis globais e reforçando a integridade dos dados e a rastreabilidade da execução.

Listagem 3.7.3 Função `load_simulation_config` em Julia.

```

1 function load_simulation_config(filename)
2 # =====
3 # Function: load_simulation_config
4 # Purpose: Reads simulation configuration and plotting parameters from a YAML file.
5 # =====
6 """
7 load_simulation_config(filename::String) -> Tuple
8 Reads and parses the simulation configuration, plotting parameters, and runtime settings
9 from a specified YAML file. It populates several primary configuration structures
10 (ConfigData, PGPDData, RuntimeData, ModionsData) from the file content.
11 # Arguments
12 - `filename::String`: The path to the input YAML file (e.g., "indatpgp3.yaml").
13 # Returns
14 - `Tuple`: A tuple containing the initialized data structures on success:
15   (`config_data::ConfigData`, `pgp_data::PGPDData`, `runtime_data::RuntimeData`, `modions_data::ModionsData`)
16 - `Int`: An integer error code (1, 2, 3, or 4) if file loading/parsing fails.
17 (Error Codes: 1=File Not Found, 2=No Permissions, 3=YAML Parsing Error, 4=Wrong Extension)
18 """
19 function load_simulation_config(filename)
20
21 # 1. Use the robust file opener to check for errors (I/O, permissions, extension)
22 # and load the YAML data.
23 # 'data' will be either the parsed dictionary (on success) or an Int error code.
24 data = open_file(filename)
25 # 2. Check for error code return. If 'data' is an integer, it means an error occurred.
26 if isa(data, Int)
27     return data # Return the error code directly (1, 2, 3, or 4)
28 end
29 # 3. If successful, 'data' is the parsed dictionary. Proceed with original struct construction.
30 # Configuration Data (Settings related to simulation runs and basic I/O)
31 config_data = ConfigData(
32     Debug(data["Debug"]["ldebug"]),
33     Directories(data["Directories"]["directory"], data["Directories"]["diratomic"]),
34     SimulationType(data["SimulationType"]["lhdrun"], data["SimulationType"]["lmhdrun"]), ...
35     ...
36     # (... restantes parâmetros omitidos para brevidade ...)
37     # Retorna os dados agrupados em structs
38     return (config_data, pgp_data, runtime_data, modions_data)
39 end

```

A comparação entre a subrotina `namelists` em FORTRAN 2008 e a função `load_simulation_config` em Julia, evidencia uma evolução substancial na forma como o sistema de configuração é estruturado e interpretado. Enquanto a primeira abordagem depende de leitura sequencial e variáveis globais, a segunda utiliza um ficheiro YAML hierárquico, associado a estruturas de dados tipadas e mecanismos de validação

robustos. A abordagem em Julia introduz várias melhorias significativas no âmbito da engenharia de software.

A Tabela 3.4 sintetiza as principais diferenças estruturais e metodológicas, destacando os ganhos em legibilidade, modularidade, extensibilidade e as boas práticas de software introduzidas.

Tabela 3.4: Análise comparativa dos paradigmas de configuração: o modelo baseado em namelists do AstroPGPLOT (FORTRAN) face à abordagem baseada em YAML e estruturas tipadas do AstroTLPlot (Julia).

Aspeto	AstroPGPLOT (FORTRAN com namelists)	AstroTLPlot (Julia com YAML + Estruturas Tipadas)
Formato de ficheiro de configuração	Ficheiro ASCII com blocos /namelist/, de estrutura rígida e dependente do compilador.	Ficheiro YAML hierárquico, legível e independente da linguagem, permitindo organização modular e validação automática.
Leitura e parsing	Rotina sequencial de leitura com formatação fixa e sem verificação de consistência.	Leitura dinâmica através de <code>YAML.jl</code> , com suporte a tipos compostos e carregamento automático em estruturas de dados.
Mapeamento de variáveis	Variáveis globais atribuídas diretamente no código principal.	Parâmetros encapsulados em estruturas compostas (<code>ConfigData</code> , <code>Debug</code> , <code>Directories</code> , etc.), assegurando coerência e isolamento.
Tratamento e validação de erros	Sem mecanismos robustos de deteção de erros de leitura ou parâmetros ausentes.	Utilização de blocos <code>try/catch</code> e validação automática de campos obrigatórios, com mensagens descritivas e logs detalhados.
Legibilidade e manutenção	Sintaxe pouco intuitiva e restrita a listas planas de parâmetros.	Estrutura hierárquica, semanticamente clara e facilmente documentável; ideal para revisão colaborativa.
Extensibilidade e modularidade	Adição de novos parâmetros exige alterações diretas no código FORTRAN e recompilação.	Novos campos podem ser adicionados apenas no ficheiro YAML, com suporte automático no carregamento de estruturas.
Integração com visualização e análise	Sem ligação direta entre parâmetros e rotinas gráficas.	Os parâmetros de visualização (como limites de eixos, cores, modos 3D) são lidos e aplicados automaticamente nas rotinas <code>Makie</code> .
Interoperabilidade	Formato proprietário, difícil de integrar com outras linguagens.	Compatível com ecossistemas modernos (Python, C, JavaScript), facilitando automação e reprodutibilidade.
Encapsulamento e segurança	As variáveis são globais, sem controlo de acesso ou validação de tipo.	Estruturas tipadas em Julia controlam o acesso e asseguram a integridade dos dados, promovendo as boas práticas de encapsulamento.
Boas práticas de software	Paradigma procedimental e centrado em sub-rotinas estáticas.	Arquitetura orientada a dados e modular, promovendo separação de responsabilidades, reprodutibilidade e manutenção sustentável.

Em síntese, a reformulação do sistema de leitura do ficheiro de configuração representou muito mais do que uma simples tradução de código: constituiu uma reengenharia conceptual que transformou um processo rígido e dependente de variáveis globais num mecanismo dinâmico, seguro e expansível. A adoção do formato YAML, combinada com estruturas tipadas, reforçou robustez, reprodutibilidade e interoperabilidade,

consolidando a ligação entre configuração, execução e visualização num ecossistema coerente e sustentável [SNTH13, WBC⁺17].

3.7.2 Comparação entre `readdata_hdf4` (FORTRAN 2008) e `readdata_hdf5` (Julia)

A leitura dos dados científicos provenientes das simulações constitui uma etapa crítica no fluxo de processamento do código. No software AstroPGPLOT, esta tarefa era executada pela subrotina `readdata_hdf4`, responsável por carregar sequencialmente os *datasets* contidos em ficheiros no formato HDF4. Cada conjunto de dados era armazenado em matrizes tridimensionais associadas a variáveis físicas (velocidade, densidade, energia), com base em variáveis globais e estruturas de dados fixas, refletindo o paradigma procedimental do FORTRAN 2008.

A Listagem 3.7.4 apresenta um excerto desta subrotina, ilustrando o fluxo sequencial de leitura e a dependência direta de variáveis globais partilhadas.

Listagem 3.7.4 Excerto da sub-rotina `readdata_hdf4` no código AstroPGPLOT (FORTRAN 2008).

```

1  subroutine readdata_hdf4
2  use config
3  use pgp
4  use runtime
5  implicit none
6  integer:: i,j,k,indx
7  integer:: dsldata,dsgdims,dsgdast,dsgdisc,status
8  integer,dimension(3):: dimsizes
9  integer:: maxrank, rank
10 allocate(scratch(1:in*jn*kn,1:nmaxfunc))
11 status = dsgdisc(1,dimsizes(1),xscale)
12 status = dsgdisc(2,dimsizes(2),yscale)
13 status = dsgdisc(3,dimsizes(3),zscale)
14 do nfunc=1,nmaxfunc
15 status = dsgdast(datalabel, dataunit, datafmt, coordsys)
16 status = dsldata(trim(datafile),rank,dimsizes,data)
17 scratch(:,nfunc)=data(:)
18 enddo
19 do k=1,kn
20 do j=1,jn
21 do i=1,in
22 indx = (k-1)*jn*in + (j-1)*in + (i-1) + 1
23 vxx(i,j,k)=scratch(indx,1)*1e5
24 ...
25 if(lmhdrun) then
26 ..
27 den(i,j,k)=scratch(indx,7)
28 ene(i,j,k)=scratch(indx,8)
29 else
30 den(i,j,k)=scratch(indx,4)
31 ene(i,j,k)=scratch(indx,5)
32 ...
33 deallocate(scratch)
34 end subroutine readdata_hdf4

```

Na implementação do AstroTLPlot, a função `readdata_hdf5!` preserva integralmente a lógica científica da leitura, mas introduz uma arquitetura moderna, modular e extensível. Esta versão substitui o formato HDF4 [The22a] pelo padrão HDF5 [The22b], que oferece maior estabilidade, compatibilidade e desempenho.

A função integra-se com a rotina genérica `open_file`, responsável por verificar a existência e integridade dos ficheiros antes da leitura. Os dados são armazenados na estrutura de dados tipada, `SimulationData`, a estrutura `ConfigData` e `RuntimeData` para informação de configuração, garantindo encapsulamento, coerência e acesso seguro.

A Listagem 3.7.5 apresenta um excerto da implementação.

Listagem 3.7.5 Função `readdata_hdf5!` no pacote `AstroTLPlot` (Julia).

```

1 function readdata_hdf5!(simulations_data::SimulationData, config_data::ConfigData,
2   runtime_data::RuntimeData)
3   # Extract grid dimensions from configuration
4   in_grid = trunc{Int, config_data.grid_size.grid_point.x}
5   jn_grid = trunc{Int, config_data.grid_size.grid_point.y}
6   kn_grid = trunc{Int, config_data.grid_size.grid_point.z}
7   # Get file directory and starting file number
8   directory = config_data.directories.directory
9   filenum = config_data.number_of_plots.nfile_start
10  lmhdrun = config_data.simulation_type.lmhdrun # Flag for MHD simulation
11  # Construct HDF5 filename
12  datafile = "$(directory)all100$(filenum).h5"
13  # 1. Use the universal opener and check the result
14  h5file = open_file(datafile)
15  # 2. Check for error code return
16  if isa(h5file, Int)
17  # If an error code is returned (e.g., 2, 3, 6), return it to the caller.
18    return h5file
19  end
20  try
21    simulations_data.Z_grid = read(h5file["fakeDim0"])
22    simulations_data.X_grid = read(h5file["fakeDim1"])
23    simulations_data.Y_grid = read(h5file["fakeDim2"])
24  catch e
25    println("Error reading grid scales from HDF5 file: ", e)
26  end
27  # Read main simulation datasets with error handling
28  try
29    simulations_data.vxx = read(h5file["Data-Set-2"]) * 1e5
30    simulations_data.vyy = read(h5file["Data-Set-3"]) * 1e5
31    simulations_data.vzz = read(h5file["Data-Set-4"]) * 1e5
32    simulations_data.den = read(h5file["Data-Set-5"])
33    simulations_data.ene = read(h5file["Data-Set-6"])
34    if lmhdrun
35      simulations_data.bxx = read(h5file["Data-Set-7"])
36      simulations_data.byy = read(h5file["Data-Set-8"])
37      simulations_data.bzz = read(h5file["Data-Set-9"])
38    end
39    println("Datasets read successfully - HDF5.")
40    catch e
41      println("Error reading datasets: ", e)
42    end
43    close(h5file)
44  end

```

A função `readdata_hdf5!` representa uma evolução significativa em termos de engenharia de software e arquitetura científica. Ao substituir variáveis globais por estruturas compostas, o código tornou-se mais legível, seguro e modular, promovendo uma clara separação entre dados, configuração e execução em linha com as práticas modernas de programação. Além disso, a adoção do formato HDF5 modernizou o fluxo de leitura e armazenamento, garantindo compatibilidade com ferramentas externas e favorecendo a

interoperabilidade entre diferentes ambientes de análise.

Esta evolução é sintetizada na Tabela 3.5, que compara as principais diferenças conceituais e técnicas entre as versões `readdata_hdf4` (FORTRAN 2008) e `readdata_hdf5!` (Julia). A análise evidencia ganhos claros ao nível da modularidade, gestão de memória, tratamento de erros e integração com o ecossistema científico moderno, refletindo uma mudança estrutural profunda e orientada para a sustentabilidade do código.

Tabela 3.5: Comparação conceptual entre as implementações no AstroPGPLOT (FORTRAN com `readdata_hdf4`) e no AstroTLPlot (Julia com `readdata_hdf5!`).

Aspeto	FORTRAN <code>readdata_hdf4</code>	Julia (<code>readdata_hdf5!</code>)
Formato de dados	formato com suporte limitado e dependente de bibliotecas antigas	Adota HDF5, padrão moderno, eficiente e amplamente suportado pela comunidade científica.
Organização de dados	Leitura direta para <i>arrays</i> globais (<code>vxx</code> , <code>den</code> , etc.).	Leitura encapsulada em estruturas tipadas (<code>SimulationData</code>), garantindo isolamento e segurança dos dados.
Gestão de memória	Requer alocação e desalocação manuais com <code>allocate/deallocate</code> .	Gerida automaticamente pelo runtime de Julia, prevenindo fugas de memória.
Tratamento de erros	Ausência de mecanismos de captura de exceções; qualquer falha interrompe a execução.	Implementação de blocos <code>try/catch</code> , com mensagens de diagnóstico e tolerância a falhas.
Dependências e acoplamento	Depende de múltiplos módulos e variáveis globais (<code>config</code> , <code>runtime</code> , <code>pgp</code>).	Parâmetros recebidos como argumentos estruturados, reduzindo acoplamento e aumentando a reutilização.
Escalabilidade e extensibilidade	Estrutura rígida; novos <i>datasets</i> exigem alterações manuais no código.	Extensível; novos campos podem ser adicionados em <code>SimulationData</code> sem alterar a função principal.
Desempenho e pipeline	Processamento iterativo e leitura linear dos dados.	Leitura vetorizada e eficiente via <code>HDF5.jl</code> , com suporte a paralelização futura.
Boas práticas de software	Paradigma procedimental e baixa separação de responsabilidades.	Arquitetura modular, orientada a dados e integrada no ecossistema científico de Julia.

A transposição da rotina representa uma evolução técnica significativa, refletindo a passagem de um paradigma procedimental e centrado em variáveis globais para uma arquitetura modular, orientada a dados e sustentada por estruturas tipadas. Esta reformulação melhorou substancialmente a legibilidade, a fiabilidade e a manutenção do código, eliminando a gestão manual de memória e introduzindo mecanismos explícitos de tratamento de erros. Em conjunto, estas melhorias traduzem-se num sistema mais robusto, reprodutível e sustentável, consolidando o AstroTLPlot como uma ferramenta alinhada com as boas práticas de desenvolvimento de software científico contemporâneo [SNTH13, WBC⁺17].

3.8 Estratégia de Transposição de Visualização: PGPLOT → Makie

A transposição das rotinas de visualização do AstroPGPLOT para o AstroTLPlot representou um desafio significativo, devido as diferenças fundamentais entre o paradigma e modelo de interação com o utilizador entre as duas bibliotecas, PGPLOT e Makie. Enquanto o PGPLOT segue um modelo estritamente procedimental [Pea01], numa sequência linear de chamadas para abrir janelas, definir escalas e desenhar elementos, o Makie adota um paradigma declarativo e orientado e orientado a cenas (scenes), mais alinhado com o ecossistema moderno da linguagem Julia. Neste modelo, as propriedades dos gráficos são definidas de forma flexível e parametrizável [DK21].

Dada esta diferença estrutural, uma transposição direta, linha a linha, revelou-se impraticável. Optou-se, portanto, por uma estratégia de transposição adaptativa, guiada por princípios de *design* modular e declarativo [DK21], que incluiu as seguintes etapas:

- **Reaproveitamento da lógica científica:** Preservou-se a lógica funcional original das rotinas implementadas em FORTRAN 2008 (definição de escalas, legendas, mapas de calor, contornos e etiquetas);
- **Substituição de funções gráficas:** As chamadas às primitivas do PGPLOT foram substituídas por equivalentes no Makie, utilizando funções como `heatmap`, `contour`, `text!` e `Colorbar` [DK21];
- **Implementação de anotações e rótulos:** Foram introduzidos mecanismos de anotação de eixos, títulos e rótulos, assegurando a continuidade visual em relação à versão original;
- **Adoção de boas práticas de engenharia de software:** A transposição seguiu as boas práticas do ecossistema Julia [BEKS17], incluindo a utilização de estruturas tipadas, modularização e parametrização explícita, o que aumenta a extensibilidade, a legibilidade e a manutenção do código.

A transposição para o Makie ultrapassou a mera reprodução das funcionalidades pré-existentes, permitindo ainda a introdução de recursos gráficos mais avançados e interativos, adequados às exigências da visualização científica contemporânea [HBO10].

Deste modo, a transposição representa uma evolução conceptual, técnica e estética do processo de visualização, reforçando a integração entre o código científico e a exploração visual dos resultados numéricos.

3.8.1 Exemplo Ilustrativo da Transposição e Adaptação de Visualização

Na implementação do AstroPGPLOT (FORTRAN 2008), a sequência de chamadas gráficas é estritamente linear e fortemente dependente da ordem de execução das subrotinas, cada função define explicitamente parâmetros de visualização, como o tamanho da janela, o enquadramento e os limites dos eixos. Esta abordagem, embora funcional, é pouco flexível e de difícil manutenção, uma vez que qualquer modificação na configuração implica alterações diretas no fluxo de chamadas.

Listagem 3.8.1 Excerto da rotina de inicialização e configuração da janela gráfica no AstroPGPLOT (FORTRAN 2008 + PGPLOT)

```

1  ! FORTRAN + PGPLOT
2  call pgbeg(0, device, numx, numy)
3  call pgpap(width, aspect)
4  call pgvsiz(0.55, width-0.8, 0.55, width*aspect-0.4)
5  call pgswin(x1, x2, y1, y2)
6  call pgbox('BCNST', 200.0, 0, 'BCNST', 200.0, 0)

```

Na transposição da rotina de visualização do AstroPGPLOT para o AstroTLPlot, exemplificada na Listagem 3.8.1, as sequências procedimentais de inicialização, para criar janelas, definir eixos, escalas e rótulos, foi implementada por uma função modular em Julia, designada `create_plot_structure()`. Esta função encapsula toda a lógica de configuração da figura e dos eixos, permitindo a criação de objetos gráficos de forma parametrizada e reutilizável [BEKS17]. Através do paradigma declarativo e da composição

funcional do Makie, é possível definir elementos visuais (título, rótulos, limites e escalas) de modo expressivo e independente da sequência de execução [DK21].

Esta abordagem promove uma clara separação entre a configuração e o conteúdo visual [HBO10], favorecendo a manutenção, a extensibilidade e a integração com outras componentes do sistema (como as rotinas de processamento de dados e de cálculo estatístico). Além disso, o Makie disponibiliza suporte nativo para renderização interativa e exportação em alta qualidade, funcionalidades inexistentes no PGPLOT.

A implementação no AstroTLPlot, ilustrada na Listagem 3.8.2, preserva a lógica da rotina original de visualização, mas adota uma abordagem mais declarativa e orientada a dados. A utilização de parâmetros nomeados (*keyword arguments*) revelou-se particularmente útil, permitindo uma interface de utilizador mais intuitiva e flexível.

Listagem 3.8.2 Função modular para inicialização e configuração da janela no AstroTLPlot (Julia com Makie.jl).

```
1 function create_plot_structure(; xsize=800, ysize=800, title="Title",
2   titlesize=16,
3   titlegap=20,
4   xlabel="X(pc)",
5   ylabel="Y(pc)",
6   valmin=0,
7   step=200,
8   valmax=1000
9 )
10 fig = Figure(size=(xsize, ysize))
11 ax = Axis(fig[1, 1], title=title, titlesize=titlesize, titlegap=titlegap,
12   xlabel=xlabel, ylabel=ylabel)
13 ticks_range = valmin:step:valmax
14 ax.xticks = ticks_range
15 ax.yticks = ticks_range
16 limits!(ax, valmin, valmax, valmin, valmax)
17 return fig, ax
18 end
```

A transposição para o ecossistema AstroTLPlot resultou assim numa solução mais expressiva e robusta, que ultrapassa em muito uma simples transposição técnica. Beneficiando de parâmetros nomeados, suporte para múltiplas janelas e integração direta com dados dinâmicos, esta abordagem demonstra as vantagens de um modelo modular, declarativo e orientado a dados [BEKS17, DK21].

Tabela 3.6: Comparação entre o sistema gráfico do AstroPGPLOT (PGPLOT/FORTRAN) e do AstroTLPlot (Makie/Julia) na inicialização e configuração de gráficos

Aspeto	AstroPGPLOT (PGPLOT/FORTRAN)	AstroTLPlot (Makie/Julia)
Paradigma gráfico	Procedimental, com sequência fixa e imperativa de chamadas.	Declarativo e orientado a dados, com composição de cenas.
Modularidade e reutilização	Limitada; o fluxo é linear e acoplado à sequência de execução.	Elevada; funções parametrizadas e independentes promovem reutilização.
Inicialização e configuração	Sequência explícita: <code>pgbegin</code> , <code>pgpap</code> , <code>pgswin</code> , <code>pgbox</code> .	Função única <code>create_plot_structure()</code> com parâmetros nomeados e valores por omissão.
Valores por omissão	Não suportados; todos os parâmetros devem ser fornecidos explicitamente.	Suporte nativo; todos os parâmetros têm valores predefinidos, facilitando uso básico.
Anotação (títulos, legendas)	Chamadas específicas: <code>pgtext</code> , <code>pgsch</code> (tamanho de caractere).	Integrada na definição do eixo via parâmetros: <code>title</code> , <code>xlabel</code> , <code>titlesize</code> .
Definição de eixos e escalas	Operações separadas: <code>pgbox</code> (caixa), <code>pgenv</code> (ambiente).	Unificada no objeto <code>Axis</code> ; escalas e limites definidos de forma coesa.
Exportação de figuras	Formato básico via <code>writeplot</code> ; opções limitadas.	Suporte nativo para múltiplos formatos (PNG, PDF, SVG) via função <code>save</code> .
Mapas de calor e contornos	Funções dedicadas: <code>colormap</code> , <code>greymaps</code> ; sem composição fácil.	Funções de alto nível: <code>heatmap</code> , <code>contour</code> ; permitem composição de camadas.
Capacidades interativas	Inexistente; gráficos são estáticos.	Suporte nativo para zoom, rotação, seleção; ideal para análise exploratória.
Integração com ecossistema	Isolado; difícil integração com outras ferramentas.	Integração natural com todo o ecossistema Julia para análise de dados.

Como resume a Tabela 3.6, enquanto o sistema gráfico do AstroPGPLOT (PGPLOT/FORTRAN) se apoiava num fluxo rígido, procedimental e dependente do estado, o AstroTLPlot (Makie/Julia) introduziu flexibilidade, modularidade e uma integração natural com o ambiente científico moderno. O resultado é uma ferramenta visual mais robusta, interativa e sustentável, capaz não apenas de reproduzir as visualizações originais, mas de incorporar novas camadas de informação e de responder às exigências atuais da análise e exploração visual de dados científicos [HBO10].

3.9 Geração de Mapas: AstroPGPLOT vs. AstroTLPlot

A função `maps` constitui uma das rotinas centrais do pacote AstroTLPlot, sendo responsável pela geração dos mapas bidimensionais de todas as variáveis presentes em `simulations_data`, e de variáveis derivadas, como `pok` e `rok`. A implementação original no AstroPGPLOT (FORTRAN 2008), apresentada na Listagem 3.9.1, ilustra de forma clara o modelo procedimental característico do software original.

Nesta abordagem, a função recebe apenas o array tridimensional `data` como argumento. Toda a informação complementar necessária à construção dos mapas, limites de eixos, escalas de cor, títulos, rótulos, opções de contorno e parâmetros gráficos, está ausente da assinatura da rotina, sendo obtida através de variáveis globais dispersas via ficheiro de configuração e de módulos partilhados (`config`, `pgp`, `runtime`, `modions`).

Listagem 3.9.1 Excerto da rotina mapas em FORTRAN.

```

1 subroutine mapas(data)
2   use config
3   use pgp
4   use runtime
5   use modions
6   implicit none
7   real,dimension(in,jn,kn),intent(in):: data
8   ...

```

Este elevado grau de acoplamento a estados globais tornava a depuração e a manutenção um processo complexo, uma vez que a alteração de um parâmetro num módulo remoto poderia gerar efeitos colaterais inesperados na renderização. No contexto da transposição, esta rotina foi identificada como crítica para a modernização arquitetural, servindo de base para a introdução de uma interface mais explícita e robusta, onde a passagem de parâmetros é centralizada em estruturas de dados bem definidas.

Em contraste, a versão no AstroTLPlot foi completamente reorganizada de acordo com princípios de *design* modular, tipagem explícita e alinhamento com o modelo de dados estruturado do pacote [BEKS17]. Este novo enquadramento é visível na assinatura da função `maps!` apresentada na Listagem 3.9.2. Ao adotar o paradigma de injeção de dependências, o estado e as configurações são representados por estruturas tipadas, `ConfigData`, `PGPData`, `RuntimeData` e `ModionsData`, passadas como argumentos explícitos.

Listagem 3.9.2 Assinatura da função `maps!` no AstroTLPlot (Julia).

```

1 function maps!(data::AbstractArray{Float64,3}, xgrid, ygrid, zgrid, var_name::String;
2   kern,ionz,is_ions,
3   cfg::ConfigData, pgp::PGPData, rt::RuntimeData, modions::ModionsData)

```

Esta abordagem não só aumenta a robustez do código, como também facilita a sua extensibilidade. A inclusão do sufixo `!` no nome da função segue as convenções idiomáticas da linguagem, sinalizando que a rotina pode modificar os objetos de visualização ou *buffers* de memória passados como argumento. Paralelamente, a tipagem estrita das estruturas de dados permite que o compilador *Just-In-Time* (JIT) otimize os ciclos de processamento dos mapas bidimensionais, assegurando que o desempenho se mantenha competitivo com a versão original em FORTRAN, enquanto a manutenção se torna significativamente mais simples e intuitiva.

A transposição para o AstroTLPlot constituiu, assim, uma reengenharia arquitetónica profunda que ultrapassou a mera reprodução funcional. A função `maps!` torna-se mais robusta, testável e preparada para evoluções futuras, beneficiando do paradigma declarativo e orientado a dados do ecossistema Julia/Makie. Este novo enquadramento traduz-se em ganhos substanciais ao nível da interatividade, qualidade gráfica e sustentabilidade computacional, mantendo simultaneamente a fidelidade científica da implementação original [HBO10]. Em última análise, estas melhorias demonstram como a aplicação de princípios modernos de engenharia de software ao código do AstroPGPLOT potencia a transparência, longevidade e reprodutibilidade, pilares essenciais do software científico contemporâneo [SNTH13, WBC⁺17].

3.10 Novas Funcionalidades na Transposição para Julia/Makie

Para além da transposição direta do código original do AstroPGPLOT, o processo de transposição constituiu uma oportunidade para introduzir um conjunto de funcionalidades inexistentes na versão FORTRAN/PGPLOT e para implementar melhorias estruturais significativas no AstroTLPlot. Estas alterações traduzem-se num salto qualitativo em termos de flexibilidade, expressividade e capacidade gráfica do sistema [Pea01, DK21]. Entre as principais extensões e melhorias introduzidas na implementação em

Julia/Makie, destacam-se:

- **Parametrização avançada:** Ajuste dinâmico e completamente programático de propriedades da figura, como resolução, dimensões, tipografia, escalas de cor, posição de legendas e limites dos eixos, permitindo uma adaptação precisa a diferentes contextos de análise e de publicação [DK21];
- **Exportação em múltiplos formatos:** Geração direta de gráficos nos formatos PNG, PDF e SVG, facilitando a integração com relatórios, artigos e outros documentos científicos [BEKS17];
- **Controlo sobre o fluxo de saída:** O Makie.jl permite renderizar visualizações no ecrã, exportar para ficheiros ou executar ambos os processos em simultâneo, ampliando a versatilidade do sistema [DK21];
- **Painéis Múltiplos e composições gráficas:** Criação de grelhas de visualização e composições lado a lado, possibilitando a comparação direta de diferentes variáveis, cortes espaciais ou instantes temporais, uma funcionalidade não implementada no AstroPGPLOT ;
- **Controlo local de regiões específicas:** Definição rigorosa de limites (xlims, ylims, zlims), ampliação (ou zoom) em áreas de interesse e sobreposição seletiva de elementos gráficos, proporcionando uma análise mais detalhada das estruturas simuladas;
- **Visualização combinada:** Integração nativa entre mapas de calor (*heatmaps*) e contornos, permitindo representar simultaneamente gradientes contínuos e fronteiras estruturais;
- **Elementos adicionais de interpretação:** Inserção automática de marcadores, rótulos e anotações em pontos de interesse, como máximos, mínimos ou regiões críticas, melhora a legibilidade e a capacidade analítica das figuras geradas.

3.10.1 Sistema Unificado de Anotação e Marcação Gráfica

A anotação de figuras com texto, marcadores e metadados representa uma funcionalidade crítica para a interpretação científica. No AstroPGPLOT, esta tarefa era realizada por rotinas específicas e pouco flexíveis, como ilustra a função `copyright` na Listagem 3.10.1.

Listagem 3.10.1 Rotina `copyright` no AstroPGPLOT (FORTRAN) para escrita de direitos de autor.

```

1  subroutine copyright
2      use pgp
3      implicit none
4      call pgsch (1.1)
5      !call pgsci (8)
6      call pgmtext('B',-1.,0.05,0.0,author)
7      call pgsci (1)
8      call pgsch (1.)
9  end subroutine copyright

```

Esta abordagem, limitada a uma posição fixa e com pouca parametrização, contrasta com a solução implementada no AstroTLPlot. A função `add_text_at` (Listagem 3.10.2) generaliza completamente o processo de anotação, permitindo inserir texto e marcadores em qualquer posição, com controlo preciso sobre cor, alinhamento, tamanho e referencial de coordenadas (absoluto ou percentual).

Listagem 3.10.2 Função `add_text_at` no AstroTLPlot (Julia) para anotação flexível de figuras..

```

1 function add_text_at(ax, x, y, text;
2     align = (:center, :center),
3     fontsize = 12,
4     color = :black,
5     is_percentage = true,
6     marker::Union{Symbol, Nothing} = nothing,
7     offset = (0.0, 0.0),
8     marker_size = 10,
9     kwargs...)
10     # Handle percentage-based positioning
11     if is_percentage
12         x_min, x_max, y_min, y_max = ax.limits[]
13         x_pos = x_min + x * (x_max - x_min)
14         y_pos = y_min + y * (y_max - y_min)
15     else
16         x_pos, y_pos = x, y
17     end
18     # Apply offset to the calculated position
19     dx, dy = offset
20     final_x, final_y = x_pos + dx, y_pos + dy
21     # Add marker at the original position (before offset) if specified
22     if !isnothing(marker)
23         scatter!(ax, [x_pos], [y_pos],
24             marker = marker,
25             markersize = marker_size,
26             color = color,
27             kwargs...)
28     # Add text at the final position (including offset)
29     text!(ax, Point2f(final_x, final_y),
30         text = string(text),
31         align = align,
32         fontsize = fontsize,
33         color = color,
34         kwargs...)
35     return (x = final_x, y = final_y)
36 end

```

Benefícios e Casos de Uso

A generalização oferecida pelo `add_text_at` traduz-se em múltiplos benefícios:

- **Generalização absoluta:** Uma única função substitui múltiplas rotinas dedicadas do PGPlot;
- **Controlo total:** Personalização de cor, alinhamento, tamanho da fonte, marcadores e *offsets*.
- **Precisão:** Suporte a coordenadas absolutas e relativas, com *offsets* ajustáveis
- **Expressividade:** Integração nativa com $\text{\LaTeX}$ para notação matemática
- **Suporte nativo a $\text{\LaTeX}$:** Permite a representação de expressões matemáticas com elevada qualidade gráfica.
- **Modularidade:** Integração imediata em qualquer tipo de visualização Makie (mapas, cortes, subplots).
- **Enriquecimento visual:** Destaque claro de valores críticos, limites, unidades e metadados da simulação.

Estes benefícios materializam-se em casos de uso concretos no AstroTLPlot:

1. **Metadados da figura** (direitos de autor, resolução):

```

1 coords2 = add_text_at(ax, 0.10, 0.05, LaTeXString(author),

```



```
2     fontsize=14, is_percentage = true, color = :black)
```

2. Anotação de características físicas (máximos, mínimos):

```
1     coords2 = add_text_at(ax, max_coords[1], max_coords[2], text,
2     is_percentage = false, offset = (0.2, 0.2),
3     marker = :circle, marker_size = 15, color = :blue)
```

3. Informação contextual (unidades, tempo da simulação):

```
1     coords2 = add_text_at(ax, 0.85, 0.90, LaTeXString(unittime),
2     fontsize=14, is_percentage = true, color = :black)
```

A transposição do sistema de anotação do AstroPGPLOT para o AstroTLPlot representa uma evolução qualitativa significativa. Enquanto a abordagem original era fragmentada, rígida e limitada a casos específicos, a nova implementação oferece uma interface unificada, parametrizável e integrada com o ecossistema gráfico moderno do Makie.

Esta generalização não apenas simplifica o código, como expande radicalmente as possibilidades de anotação científica, permitindo destacar características físicas relevantes, incluir metadados contextuais e utilizar notação matemática precisa, tudo através de uma única função coerente e bem documentada.

A comparação lado a lado das implementações (Listagens 3.10.1 e 3.10.2) evidencia esta transformação arquitetônica, demonstrando como princípios modernos de engenharia de *software*, modularidade, baixo acoplamento e alta coesão, podem ser aplicados com sucesso para modernizar ferramentas científicas consolidadas.

3.10.2 Visualização Combinada: Mapas de Calor com Sobreposição de Contornos

O sistema de visualização do AstroPGPLOT não incluía funcionalidades nativas para a geração integrada de mapas de calor com sobreposição de contornos. Esta limitação exigia que os investigadores produzissem e combinassem visualizações separadas manualmente, um processo propenso a inconsistências e difícil de reproduzir. No AstroTLPlot, esta capacidade foi implementada através da função `plot_heat_cont` (Listagem 3.10.3), que consolida num único pipeline todas as etapas necessárias: geração do mapa de calor, cálculo e sobreposição dos contornos, aplicação da escala de cores, e anotação com metadados. Esta consolidação melhora a legibilidade, reduz a duplicação de código e assegura uniformidade entre as figuras produzidas.

Listagem 3.10.3 : Excerto da função `plot_heat_cont` no AstroTLPlot (Julia), que combina mapa de calor e contornos.

```

1 function plot_heat_cont(xg::AbstractVector, yg::AbstractVector, slice::AbstractMatrix,
2   var_name::String, smin::Float64, smax::Float64;
3   xlabel::String = "", ylabel::String = "", title::String = "", author::String = "",
4   colormap = :viridis, min_color = :black, max_color = :white, colorbar_label::String = "",
5   show_stats::Bool = false, logscale::Bool = false,
6   stats::Union{Nothing, Matrix2DStatistics} = nothing )
7   ...
8   fig, ax = create_plot_structure(xlabel=xlabel, ylabel=ylabel, title=LaTeXString(title))
9   # Mapa de calor
10  hm = heatmap!(ax, xg, yg, plotdata; colormap=minha_paleta_de_cores, colorrange=crange)
11  # Sobreposição de contornos
12  contour!(ax, xg, yg, plotdata; color=:black)
13  # Escala de cor e anotações
14  escala!(fig, smin, smax, side=:right, colormap=minha_paleta_de_cores, variavel=var_name)
15  # Texto e direitos de autor
16  coords2 = add_text_at(ax, 0.10, 0.05, LaTeXString(author), fontsize=14,
17    is_percentage=true, color=:black)
18  coords2 = add_text_at(ax, 0.85, 0.90, LaTeXString(resolution), fontsize=14,
19    is_percentage=true, color=:black)
20  coords2 = add_text_at(ax, 0.85, 0.85, LaTeXString(unittime), fontsize=14,
21    is_percentage=true, color=:black)
22  ...
23  if show_stats
24    ...
25  end
26  files = writeplot(42, lpdf = true, variavel = var_name, atom = 8, filenum = 3000,
27    datetime_format = "ddmmyy_HHMM", formats = ["pdf", "png", "svg"])
28  save_or_display( fig, files; sav = true, disp = true, output_path = subdir )
29  return fig
30 end

```

Benefícios da Visualização Combinada:

A função `plot_heat_cont` implementada no AstroTLPlot representa uma inovação significativa face ao ecossistema fragmentado do AstroPGPLOT. Ao integrar num único pipeline a geração de mapas de calor e a sobreposição de contornos, esta abordagem oferece múltiplos benefícios técnicos e científicos:

- **Visualização científica unificada:** Permite a interpretação simultânea da intensidade espacial (`heatmap`) e das estruturas ou gradientes (contornos) num único painel, acelerando a análise visual e reduzindo ambiguidades na interpretação de dados astrofísicos.
- **Implementação coesa e robusta:** Substitui múltiplas chamadas dispersas e manuais por um bloco lógico integrado, eliminando pontos de falha, reduzindo erros e simplificando manutenções futuras.
- **Consistência e reprodutibilidade:** A reutilização da mesma estrutura base para diferentes visualizações assegura uniformidade gráfica e resultados reprodutíveis, fundamentais para a investigação científica.
- **Arquitetura modular:** Aproveita funções auxiliares comuns (`create_plot_structure`, `add_text_at`) que promovem a reutilização de código e facilitam evoluções futuras do sistema.
- **Qualidade de publicação pronta:** Com suporte nativo a $\text{\LaTeX}$ para anotações e exportação parametrizável para múltiplos formatos (PDF, PNG, SVG), produz figuras adequadas diretamente para comunicação científica.

Em síntese, `plot_heat_cont` encapsula uma abordagem moderna de visualização científica que não apenas adiciona uma funcionalidade ausente no sistema original, mas redefine todo o workflow de geração de figuras complexas. Esta implementação assegura consistência, robustez e qualidade visual, atributos

indispensáveis para uma comunicação eficaz de resultados na astrofísica computacional contemporânea.

3.10.3 Visualização Integrada de Múltiplas Representações

A funcionalidade de visualização integrada introduzida em AstroTLPlot assenta num modelo conceptual que organiza múltiplas representações da mesma variável física, neste caso, a densidade, num único painel coerente. Esta abordagem segue o princípio da composição gráfica estruturada, no qual diferentes camadas de informação são agregadas num arranjo espacial sistemático, permitindo ao utilizador relacionar propriedades locais, globais e estatísticas sem alternar entre figuras independentes.

A função `mapas_subplot!` apresentada na Listagem 3.10.4, materializa este conceito recorrendo a três pilares fundamentais:

- abstração das fontes de dados,
- modularização das rotinas de visualização,
- composição hierárquica das subfiguras.

Listagem 3.10.4 Excerto da função `mapas_subplot!` no AstroTLPlot (Julia).

```

1 function mapas_subplot!(data::AbstractArray{Float64,3}, xgrid, ygrid, zgrid, var_name::String;
2   cfg::ConfigData, pgp::PGPData, rt::RuntimeData, modions::ModionsData)
3   ...
4   stats = statistics_data2D(slice, log10)
5   result = setminmaxvar(var_name, pgp, modions) # External function for min/max calculation
6   ...
7   # P1: Contour plot
8   p1 = contour!(ax1, xg, yg, data_to_plot,
9     levels=LinRange(smin, smax, 10),
10    colormap=:viridis, linewidth=1.5, colorrange=(smin, smax),
11    label="Contours: $(var_name)")
12
13  # P2: Heatmap
14  p2 = heatmap!(ax2, xg, yg, data_to_plot, colormap=:viridis, colorrange=(smin, smax))
15  ...
16  end

```

A organização coordenada de quatro vistas complementares, mapa de calor, contorno, visualização combinada e PDF da distribuição, num único painel oferece vantagens substantivas:

- **Correlação analítica direta:** Permite relacionar padrões espaciais locais (identificados nos mapas) com propriedades estatísticas globais (representadas no PDF), facilitando a identificação de estruturas, gradientes e assimetrias físicas relevantes.
- **Consistência visual e semântica:** A partilha unificada de escalas, limites, paletas de cores e anotações entre todas as subfiguras elimina ambiguidades de interpretação e reforça a coerência analítica.
- **Eficiência computacional e de workflow:** Centraliza num único *pipeline* o que anteriormente exigia a geração e comparação manual de múltiplas figuras separadas, reduzindo fragmentação de código e simplificando a manutenção.
- **Flexibilidade arquitetónica:** Aproveita o sistema de *layouts* hierárquicos do Makie e funções auxiliares modulares (`create_plot_structure`, `add_text_at`) para permitir ajustes uniformes em toda a composição gráfica.
- **Prontidão para publicação:** Produz figuras complexas e informativas diretamente adequadas para análise exploratória e comunicação científica, com suporte nativo a $\text{\LaTeX}$ e exportação multi-formato.

A função `mapas_subplot!` representa mais do que uma funcionalidade gráfica adicional; ela encapsula uma filosofia de visualização científica integrada que alinha a representação dos dados com o processo de interpretação física. Ao contrário do modelo procedimental e desagregado do AstroPGPLOT, esta

implementação oferece uma estrutura composicional que transforma a visualização de uma mera ferramenta de apresentação num instrumento ativo de análise exploratória. A sincronização rigorosa entre as diferentes vistas não apenas organiza informação, como cria um contexto interpretativo enriquecido, essencial para a compreensão de sistemas astrofísicos complexos.

3.10.4 Visualização Integrada de Estados Iônicos

A função `plot_element_subplot`, implementada no `AstroTLPlot`, introduz uma capacidade analítica avançada para a visualização coordenada dos diferentes estados iônicos de um elemento químico. Esta funcionalidade, ausente no `AstroPGPLOT`, permite gerar automaticamente uma grelha de subplots que organiza comparativamente os mapas bidimensionais de todos os estados de ionização de um elemento específico.

A sua assinatura, apresentada na Listagem 3.10.5, revela uma abordagem parametrizada e tipada que encapsula as dependências necessárias de forma explícita.

Listagem 3.10.5 Assinatura da função `plot_element_subplot` no `AstroTLPlot` (Julia).

```
1 function plot_element_subplot(  
2     temp_props::TemperatureProperties,  
3     sml::SimulationData,  
4     element::Element, element_num::Int;  
5     ncols::Int=3,  
6     save_dir::String="")
```

Exemplo de Utilização e Benefícios

A invocação da função ilustra a sua flexibilidade e integração no *workflow* de análise:

```
1 plot_element_subplot(tps, simulations_data, elem, 7, ncols=2, save_dir="./subplots")
```

Neste exemplo, a função gera uma grelha com duas colunas contendo os mapas de todos os estados iônicos do azoto (elemento de índice 7, seguindo a ordem da Tabela Periódica), exportando automaticamente as figuras para o diretório especificado. Esta abordagem contrasta radicalmente com o processo manual e fragmentado exigido pelo `AstroPGPLOT`, onde cada estado iónico necessitava de ser visualizado individualmente, dificultando comparações sistemáticas.

Esta abordagem integrada oferece múltiplos benefícios técnicos e analíticos:

- **Análise comparativa eficaz:** Organiza todos os estados iónicos de um elemento numa única vista coordenada, permitindo identificar padrões de ionização espacial e transições entre estados.
- **Automatização completa:** Substitui múltiplas chamadas manuais por um *pipeline* único que gera, formata e exporta toda a sequência de visualizações.
- **Flexibilidade parametrizada:** Controlo sobre o *layout* (número de colunas) e destino de exportação, adaptando-se a diferentes necessidades de apresentação e fluxos de trabalho.
- **Consistência visual garantida:** Aplicação uniforme de escalas, paletas de cores e anotações através de todos os *subplots*, assegurando comparabilidade direta.
- **Generalização e reutilização:** Interface tipada que funciona para qualquer elemento químico e conjunto de dados de simulação, promovendo reutilização de código.

A função `plot_element_subplot` encapsula uma transformação fundamental na forma como os dados de ionização são analisados visualmente. Enquanto o `AstroPGPLOT` obrigava a uma análise estado-a-estado desconexa, esta função oferece uma visão integrada e sistemática da estrutura de ionização espacial.

Mais do que uma conveniência gráfica, representa uma ferramenta analítica que alinha a visualização com o processo físico de interpretação, permitindo correlacionar padrões espaciais de ionização com condi-

ções físicas locais de forma imediata e reproduzível. Esta capacidade constitui um avanço significativo no apoio à análise exploratória de simulações astrofísicas complexas.

3.11 Expansão do Módulo Estatístico: Funcionalidades Derivadas de statistics_data

A função `statistics_data` calcula as estatísticas essenciais de um conjunto de dados unidimensionais (1D), bidimensionais(2D) e tridimensionais(3D), constituindo o núcleo do pré-processamento numérico no pacote AstroTLPlot. Esta função fornece métricas fundamentais que suportam diversas operações gráficas, incluindo:

- Cálculo de média, variância, desvio padrão e limites automáticos para os eixos de visualização;
- Aplicação de transformações logarítmicas ou outras funções definidas pelo utilizador;
- Extração de valores mínimo e máximo e das respetivas coordenadas.

A sua assinatura da função em Julia encontra-se na Listagem 3.11.1, revela uma abordagem parametrizada e tipada:.

Listagem 3.11.1 Assinatura da função `statistics_data` no AstroTLPlot (Julia).

```
1 function statistics_data(matriz::AbstractMatrix{T},  
2   func::Union{Function, Nothing} = nothing  
3   ) where T<:Real
```

3.11.1 Principais Características e Benefícios da Função: `statistics_data`

- Cálculos integrados num ciclo único:** Durante a primeira passagem sobre a matriz, a função calcula a soma, média parcial, valores mínimo e máximo e respetivas coordenadas, realiza a conversão dos elementos de `Float32` para `Float64` e aplica, quando especificado, uma transformação personalizada (`func`). Os resultados transformados são armazenados na `matrix_result`. Uma segunda passagem, dedicada ao cálculo da variância e do desvio padrão, conclui o processo. Esta abordagem reduz significativamente o número de iterações em comparação com a implementação original no AstroPGPLOT (FORTRAN), que invocava múltiplas funções nativas (`maxval`, `minval`, `sum`, etc.), percorrendo toda a matriz em cada chamada sem reutilização de resultados intermédios.
- Suporte eficiente a transformações logarítmicas e personalizadas:** Permite aplicar transformações como $func = \log_{10}$ diretamente durante a primeira passagem. Esta integração elimina ciclos adicionais, otimizando o processamento, sobretudo para matrizes de grande dimensão, comuns em simulações astrofísicas.
- Resultados encapsulados em estruturas de dados especializadas:** Os valores calculados são armazenados em estruturas como `StatisticsData2D` e `Matrix2DStatistics`, que incluem:
 - Valores mínimo, máximo, média, soma, variância e desvio padrão;
 - Coordenadas do mínimo e máximo (`Coordinate2DResult`);
 - Limites de eixo automaticamente inferidos (`Axis2DLimits`);
 - Matriz transformada (aplicação de função) (`matrix_result`) pronta para visualização.
- Ganho significativo de desempenho:**

A redução das passagens redundantes sobre a matriz e a integração das transformações num único ciclo traduzem-se numa melhoria de desempenho face ao método original do AstroPGPLOT (FORTRAN 2008), aumentando a eficiência e evitando cálculos repetidos.
- Integração direta com funções de visualização:** Os resultados produzidos por `statistics_data` podem ser utilizados imediatamente em funções de geração de gráficos, tais como `plot_heatmap`,

`plot_cont` ou `plot_heat_cont`, incluindo o posicionamento automático de marcadores nas coordenadas dos extremos identificados.

Tipo de retorno e Utilização Prática

A função `statistics_data` devolve um objeto do tipo `Matrix2DStatisticsT`, que encapsula todas as métricas calculadas e a matriz transformada. Um exemplo típico de utilização encontra-se na Listagem 3.11.2.

Listagem 3.11.2 Excerto da função de `plot_heatmap` que utiliza dados transformados via `statistics_data2D` no `AstroTLPlot`.

```

1  ...
2  stats = statistics_data(slice, log10)
3  # no plot_heatmap verifica se já foi feito o calculado da estatística
4  stats == nothing && (stats = statistics_data2D(slice))
5  if log
6  plotdata = stats.matrix_result # já calculada com log10
7  ...

```

3.11.2 Identificação e Anotação de Extremos

A extração dos valores mínimo e máximo, bem como das respetivas coordenadas (Listagem 3.11.3), é realizada diretamente a partir do objeto retornado por `statistics_data`. Esta abordagem permite identificar rapidamente os pontos de maior interesse no *dataset*, evitando a necessidade de percorrer novamente a matriz de dados. Assim, a função estatística serve como base para operações subsequentes de anotação e visualização, assegurando consistência e eficiência no processamento.

Listagem 3.11.3 Extração de valores mínimo e máximo e coordenadas a partir do objeto `statistics_data2D` no `AstroTLPlot`.

```

1  ...
2  stats = statistics_data2D(slice, log10)
3  minval = stats.statistics_data.min_value.value
4  maxval = stats.statistics_data.max_value.value
5  max_coords = stats.statistics_data.coord_max.coord
6  min_coords = stats.statistics_data.coord_min.coord
7  ...

```

Estas coordenadas permitem anotação automática em gráficos, como ilustrado na Listagem 3.11.4:

Listagem 3.11.4 Utilização das coordenadas do máximo para adicionar marcadores no `AstroTLPlot`.

```

1  ...
2  stats = statistics_data2D(slice, log10)
3  ...
4  max_coords = stats.statistics_data.coord_max.coord
5  add_text_at(ax, max_coords[1], max_coords[2], text,
6  is_percentage=false, marker=:circle, marker_size=15)
7  ...

```

3.11.3 Persistência dos Resultados Estatísticos

Para garantir rastreabilidade e reprodutibilidade, o módulo disponibiliza a função `export_statistics` na Listagem 3.11.5, que opera sobre um objeto `Matrix2DStatistics` e suporta múltiplos formatos de saída (`.txt`, `.pdf`):

Listagem 3.11.5 Assinatura da função `export_statistics` no `AstroTLPlot`.

```
1 function export_statistics(result;
2   list_of_files::Union{Nothing,Vector{String}}=nothing,
3   sav::Bool=true,
4   disp::Bool=false,
5   output_path::String=".")::Int
```

Esta funcionalidade encerra o ciclo de processamento estatístico: após a análise e visualização, os resultados podem ser guardados de forma estruturada, preservados para inspeção futura e integrados em pipelines externos de pós-processamento ou validação. A disponibilização de múltiplos formatos de saída, aliada à verificação de permissões e ao tratamento robusto de erros, torna esta função particularmente adequada a fluxos científicos reprodutíveis e escaláveis.

A função `statistics_data` constitui uma evolução arquitetónica significativa face à abordagem fragmentada do `AstroPGPLOT`. Enquanto o sistema original recorria a múltiplas rotinas autónomas sem partilha de estados intermédios, a implementação no `AstroTLPlot` consolida todos os cálculos num *pipeline* integrado e eficiente. Esta transformação não apenas reduz o custo computacional e minimiza erros lógicos, como também estabelece uma base sólida para análise estatística reprodutível e integrada com visualização, um pilar fundamental para a investigação astrofísica computacional moderna.

3.12 Novo Sistema de Organização de Saída e Nomenclatura Automática no AstroTLPlot

A gestão consistente e reprodutível dos ficheiros gerados constitui um elemento central no desenvolvimento de software científico [WBC⁺17]. Durante a transposição do `AstroPGPLOT` (FORTRAN) para o `AstroTLPlot` (Julia), foi completamente reestruturado o mecanismo responsável pela criação de diretórios, atribuição de nomes e exportação de gráficos, unificando num conjunto coeso de funções toda a lógica de organização e escrita da saída.

Organização Hierárquica da Estrutura de Saída

O pacote `AstroTLPlot` cria automaticamente uma estrutura de diretórios modular, onde cada variável física (densidade, temperatura, espécies iónicas, PDFs ou subplots) é armazenada na sua pasta própria. Esta organização hierárquica substitui a estrutura rígida da versão original, permitindo escalabilidade, paralelização e coerência entre conjuntos de resultados [WAB⁺14]. A convenção é ilustrada na Tabela 3.7, sendo aplicada de forma automática sempre que uma visualização é produzida.

Tabela 3.7: Convenção de pastas criadas automaticamente pelo pacote `AstroTLPlot`.

Nome da Pasta	Conteúdo
den/	Mapas de densidade global
tem/	Mapas de temperatura
H0/ ou ions/H1+/	Mapas de densidade iónica
pdf/	Funções de densidade de probabilidade
subplots/	Painéis multi-gráfico

Esta estrutura mantém a coerência entre as variáveis analisadas e os resultados produzidos, permitindo, por exemplo, que múltiplas variáveis físicas sejam processadas em paralelo sem conflitos de nomes ou de diretórios.

3.12.1 Sistema de Nomenclatura Automática de Ficheiros

A função `writeln` implementa um sistema avançado de nomenclatura automática para os ficheiros de saída, assegurando consistência, rastreabilidade e organização lógica dos resultados. O sistema combina inteligentemente múltiplos elementos descritivos:

- Tipo de visualização (mapa de calor, contorno, combinado, histograma);
- Variável física e espécie iónica analisadas (ele, elenz, den, ...);
- Identificador sequencial do ficheiro (`filenum`);
- Formato de exportação (pdf, svg, png) ou combinações múltiplas;
- Carimbo temporal opcional com data e hora de execução;
- Formato temporal personalizável (DDMMAA_HHMM).

Este mecanismo garante não apenas a legibilidade imediata dos ficheiros, mas também a rastreabilidade temporal completa dos resultados, facilitando a reprodução de análises e a gestão de múltiplas sessões computacionais.

A Função `writeln`: Conceito e Evolução

A rotina `writeln` desempenha o papel de camada de abstração entre o sistema de visualização e o sistema de ficheiros. Na versão do AstroPGPLOT (FORTRAN 2008), a geração de nomes de ficheiros envolvia formatação manual de *strings*, regras codificadas através de procedimentos e chamadas explícitas ao sistema operativo. A Listagem 3.12.1 evidencia esta abordagem, com lógica altamente acoplada ao ambiente de execução.

Listagem 3.12.1 Excerto da rotina `writeplot` no AstroPGPLOT (FORTRAN 2008), responsável pela criação de nomes de ficheiros de saída.

```

1  subroutine writeplot(ref,lmap,lpdf)
2  use config
3  use pgp
4  use runtime
5  use modions
6  ...
7  select case(device)
8  case('/gif')
9  ! write(command,('mv pgplot.gif",1x,a,a,a') trim(plotfile),'.',dev
10 command='mv pgplot.gif '//trim(plotfile)//'.'//dev
11 case('/png')
12 !write(command,('mv pgplot.png",1x,a,a,a') trim(plotfile),'.',dev
13 command='mv pgplot.png '//trim(plotfile)//'.'//dev
14 case('/ps','/cps','/vcps')
15 call execute_command_line('epspdf pgplot.ps',wait=.true.)
16 command='mv pgplot.pdf '//trim(plotfile)//'.pdf'
17 write(*, '(a)') command
18 end select
19 ! Execute the command for all the cases
20 call execute_command_line(command,wait=.true.,exitstat=estat,cmdstat=cstat,cmdmsg=cmsg)
21 if (cstat > 0) then
22 write(*,*) 'Command execution failed with error ', TRIM(cmsg)
23 else if (cstat < 0) then
24 write(*,*) 'Command execution not supported'
25 else
26 write(*,*) 'Command completed with status ', estat
27 endif
28 end subroutine writeplot

```

A nova implementação no AstroTLPlot (Julia) cuja assinatura é apresentada na (Listagem 3.12.2) redefine o conceito, integrando:

- Argumentos nomeados e interface autocontida;
- Suporte nativo a múltiplos formatos de exportação;
- Inclusão opcional de carimbo temporal;
- Identificação automática de espécies iónicas no nome do ficheiro e na pasta destino;
- Eliminação de dependências de comandos externos.

Assim, a função deixou de ser uma tradução literal da versão FORTRAN e tornou-se uma peça arquitetural concebida para flexibilidade, segurança e extensibilidade.

Listagem 3.12.2 Assinatura da função `writeplot` no AstroTLPlot (Julia), evidenciando parâmetros nomeados, formatos de saída flexíveis e automatização da inserção de data e hora nome do ficheiro de saída

```

1  function writeplot(ref::Int;
2  lmap::Union{Bool,Nothing}=nothing,
3  lpdf::Union{Bool,Nothing}=nothing,
4  color::Bool=false, cont::Bool=false, grey::Bool=false, side::Bool=false,
5  front::Bool=false, top::Bool=false,
6  variavel::String="", ion::String="",
7  filenum::Int=0, atom::Int=0,
8  formats::Vector{String}=["pdf"],
9  add_datetime::Bool=true,
10 datetime_format::String="yyyymmdd_HHMMSS")

```

Melhorias introduzidas na Implementação em Julia

A nova implementação da função `writelnplot` incorpora várias extensões significativas face à versão original :

1. **Suporte a múltiplos formatos de exportação:** O utilizador pode agora exportar diretamente os gráficos em diferentes formatos, PDF, PNG, SVG, através da passagem de uma lista de formatos à função. Isto simplifica a geração de material para diferentes contextos (relatórios, publicações, apresentações, etc.).
2. **Inserção opcional de data e hora do sistema nos nomes dos ficheiros:** Foi introduzida a opção de incluir a data e hora atuais no nome do ficheiro exportado. Esta funcionalidade é particularmente útil para manter versões cronológicas das figuras durante o desenvolvimento e análise das simulações, garantindo a rastreabilidade dos resultados.
3. **Identificação contextual através de prefixo de frações iónicas no nome do ficheiro e diretórios:** No caso de visualizações associadas a frações iónicas, o nome do ficheiro passa a incluir automaticamente o prefixo com o nome do elemento químico e a identificação da fração iónica correspondente. Esta melhoria elimina ambiguidades e reforça a organização dos resultados, sobretudo quando são gerados múltiplos gráficos num mesmo diretório.

Vantagens Técnicas e de Utilização

As melhorias introduzidas trazem ganhos concretos:

- **Reprodutibilidade:** a inclusão automática de metadados no nome (timestamp, ião) e a capacidade de gerar múltiplos formatos reduzem o trabalho manual e a margem para erro humano.
- **Automatização de pipelines:** o resultado previsível e normalizado facilita a integração com rotinas de pós-processamento, injeção em relatórios e repositórios de dados.
- **Portabilidade:** menos dependência de comandos do sistema operativo melhora a execução em ambientes heterogéneos.
- **Clareza para o utilizador:** a interface por argumentos nomeados é mais auto-documentada e reduz a curva de aprendizagem.
- **Aderência aos princípios FAIR:** a normalização estruturada dos metadados incorporados nos nomes dos ficheiros e a flexibilidade dos formatos de saída aumentam a *Findability* e *Reusability* dos resultados, enquanto a portabilidade e a reprodutibilidade favorecem a *Accessibility* e a *Interoperability*.
- **Gestão de dados científicos:** a estrutura promove práticas de dados abertos, facilitando o depósito em repositórios institucionais e a partilha de conjuntos de dados com metadados mínimos embutidos.

A reimplementação da rotina de atribuição de nomes e exportação de ficheiros, constitui uma nova função conceitual significativa . Em vez de uma tradução direta, a versão no AstroTLPlot define uma interface mais expressiva e parametrizável aumentando a reprodutibilidade, a automação e a interoperabilidade dos resultados, reduzindo dependências do sistema operativo e facilitando a sua integração em *pipelines* de pós-processamento e repositórios científicos.

3.13 Avaliação de Desempenho do AstroTLPlot

A implementação do pacote AstroTLPlot [LCdA25] em Julia implicou um processo sistemático de testes, validação e verificação, com o objetivo de assegurar que a transposição das rotinas desenvolvidas no AstroPGPLOT (FORTRAN 2008) preservasse a fidelidade científica, a integridade dos dados e a robustez do fluxo de execução. Esta fase foi essencial para garantir que a ferramenta desenvolvida mantivesse o comportamento físico e numérico esperado, enquanto explorava as potencialidades de um ambiente mais modular e interativo. Nesta secção, apresentam-se exemplos práticos que ilustram como cada componente do pacote foi testado, validado e verificado.

3.13.1 Validação e Benchmark de Leitura de Ficheiros

Foram utilizados ficheiros de dimensão idêntica, contendo os mesmos *datasets* de densidade e velocidade, de modo a garantir condições experimentais comparáveis. Na versão do AstroTLPlot, foram testadas duas bibliotecas distintas:

- PyHDF: interface *Python* para HDF4, acedida a partir de Julia via PyCall1;
- HDF5.jl: biblioteca nativa de Julia para leitura e escrita no formato HDF5.

Para cada abordagem, foram medidas as seguintes métricas clássicas de desempenho [B⁺20]:

- Tempo total, médio, mínimo e máximo de execução;
- Desvio padrão do tempo de execução;
- Memória total alocada e número total de alocações.

Tabela 3.8: Comparação de desempenho entre as versões no (AstroPGPLOT) e Julia (AstroTLPlot) na leitura de ficheiros HDF.

Métrica	AstroPGPLOT (FORTRAN 2008 (HDF4))	AstroTLPlot (Julia) (PyHDF - HDF4)	AstroTLPlot (Julia) (HDF5.jl - HDF5)
Tempo médio	73.2 ms	836.7 ms	623.5 ms
Tempo mínimo	—	523.7 ms	441.0 ms
Tempo máximo	—	1015 ms	966.3 ms
Desvio padrão	—	224.1 ms	198.4 ms
Memória alocada	229.13 MiB	229.13 MiB	228.94 MiB
Nº de alocações	1	7805	259

Os resultados, apresentados na Tabela 3.8, podem ser resumidos nos seguintes pontos:

- O AstroPGPLOT (FORTRAN 2008) é cerca de 8.5 vezes mais rápida do que a AstroTLPlot com HDF5.jl;
- É aproximadamente 11.4 vezes mais rápida do que a versão em Julia com PyHDF;
- A versão em Julia com HDF5.jl apresentou um desempenho 1.3 vezes superior à versão com PyHDF, refletindo menor sobrecarga (*overhead*) e melhor integração com o ecossistema nativo da linguagem.

Em termos de consistência temporal, a versão em FORTRAN apresentou um desvio padrão inferior, evidenciando maior estabilidade nas operações de leitura. O consumo de memória manteve-se praticamente equivalente entre as três versões, confirmando que as diferenças de desempenho resultam essencialmente do modelo de execução e não do volume de dados manipulados.

Conclusão

Os resultados demonstram que, embora a implementação no AstroPGPLOT (FORTRAN 2008) apresente um desempenho superior nas operações de I/O, uma consequência direta da sua compilação estática e da otimização ao nível do sistema, a adoção do AstroTLPlot representa uma escolha estratégica no contexto da evolução da aplicação. A linguagem Julia oferece uma maior produtividade, facilidade de integração com outros formatos (YAML, HDF5, VTK) e uma infraestrutura moderna de análise e visualização. Assim, o pequeno custo adicional em tempo de leitura é compensado por um aumento expressivo na flexibilidade, na clareza do código e na capacidade de expansão do sistema, assegurando um ambiente mais adequado para futuras fases de desenvolvimento e exploração científica, conforme as boas práticas em engenharia de software científico[WBC⁺17, BEKS17].

3.13.2 Verificação de Estatísticas de Matrizes

O objetivo principal foi confirmar que a implementação do AstroTLPlot preserva a precisão numérica e a equivalência científica do AstroPGPLOT, garantindo simultaneamente uma maior eficiência computacional.

Na reimplementação em Julia, foram comparadas duas abordagens:

- **Versão nativa Julia:** utilização direta das funções `minimum`, `maximum`, `mean`, `std` e `var`;
- **Versão otimizada em ciclo único (`statistics_data`):** execução de todos os cálculos num único ciclo iterativo, obtendo simultaneamente soma, média, valores extremos e coordenadas correspondentes, com suporte opcional para transformações logarítmicas. O segundo ciclo para as funções desvio padrão e variância.
- **Versão otimizada com *broadcasting*:** utilização de operações vetorizadas para aplicar funções sobre a matriz, explorando a sintaxe idiomática de Julia para reduzir a complexidade do código e melhorar a legibilidade.

A assinatura simplificada da função `statistics_data` otimizada é apresentada na Listagem 3.13.1.

Listagem 3.13.1 Assinatura da função `statistics_data` no AstroTLPlot (Julia).

```

1 function statistics_data2D(
2     matriz::AbstractMatrix{T},
3     func::Union{Function, Nothing} = nothing
4 ) where T<:Real

```

Para a avaliação das duas abordagens foram utilizados *datasets* de diferentes dimensões (500×500 e 2000×2000), seguindo o procedimento:

- execução repetida dos cálculos estatísticos para medição dos tempos mínimos, medianos e máximos;
- análise de utilização de memória com o pacote `BenchmarkTools`;
- validação dos resultados face à implementação original no AstroPGPLOT(FORTRAN 2008);
- verificação estrutural das entidades auxiliares `StatisticsData2D` e `Matrix2DStatistics`, incluindo coordenadas de extremos e limites dos eixos.

Resultados e Análise

O teste foi realizado sobre o *dataset* Energia, com dimensão 2000×2000 (4 milhões de elementos). A Tabela 3.9 apresenta a comparação direta entre as versões nativa e otimizada, demonstrando a equivalência dos resultados numéricos.

Tabela 3.9: Comparação dos resultados obtidos pelas versões Ciclo Único e Nativa Julia.

Métrica	Ciclo Único- Julia	Funções nativa- Julia	Diferença Absoluta
Soma	9.304804151295123e ⁻¹⁸	9.304804151294797e ⁻¹⁸	3.26e ⁻³¹
Média	2.3262010378237808e ⁻²⁴	2.3262010378236992e ⁻²⁴	8.16e ⁻³²
Mínimo	8.346284034928763e ⁻²⁹	8.346284034928763e ⁻²⁹	0.00e ⁰
Máximo	8.109279574473616e ⁻²²	8.109279574473616e ⁻²²	0.00e ⁰
Desvio Padrão	9.657694139643974e ⁻²⁴	9.657694139644001e ⁻²⁴	2.70e ⁻³⁵
Variância	9.327105609491358e ⁻⁴⁷	9.327105609491406e ⁻⁴⁷	4.80e ⁻⁵⁹

As diferenças numéricas observadas são da ordem de 10^{-30} , muito inferiores ao erro de arredondamento típico da aritmética de vírgula flutuante em dupla precisão ($\varepsilon_{\text{máquina}} \approx 10^{-16}$) [Gol91, Hig02]. Isto confirma a equivalência matemática das abordagens, sendo as pequenas variações atribuídas às limitações inerentes à representação `Float64`.

As coordenadas dos extremos também coincidiram:

- **Mínimo:** (1974, 471)
- **Máximo:** (99, 1514)

Desempenho Computacional

A Tabela 3.10 resume os principais indicadores de desempenho obtidos, evidenciando uma redução expressiva no tempo de execução da versão otimizada em ciclo único, sem alteração no consumo de memória.

Tabela 3.10: Resumo de desempenho das três abordagens para uma matriz de dimensão (2000×2000).

Implementação	Tempo (ms)	Alocações	Memória (MiB)	Razão (Loop/Native)
Loop-based (AstroTLPlot)	41.294	12	30.52	0.90×
Native Julia	46.054	14	61.04	1.00×
Optimized Julia (Broadcasting)	46.394	14	61.04	1.01×

Os resultados evidenciam que a implementação Loop-based apresenta melhor desempenho global, com menor tempo de execução e consumo de memória. As versões Native Julia e Broadcasting oferecem maior legibilidade e aderência ao estilo idiomático da linguagem, mas não apresentam ganhos significativos em termos de desempenho. Em particular, a técnica de *broadcasting*, apesar de ser considerada uma otimização, revelou tempos praticamente idênticos à versão nativa, mantendo o mesmo padrão de alocações e utilização de memória. Esta diferença de cerca de 10% no tempo de execução confirma a vantagem da abordagem Loop-based em cenários onde a eficiência computacional é crítica.

A Figura 3.5 apresenta a comparação gráfica entre as versões, onde se observa uma redução média de cerca de 48% no tempo de execução da abordagem em ciclo único. Esta melhoria resulta da eliminação de múltiplas chamadas a funções de agregação, concentrando os cálculos num único ciclo iterativo, o que reduz a sobrecarga de execução e aumenta a eficiência computacional.

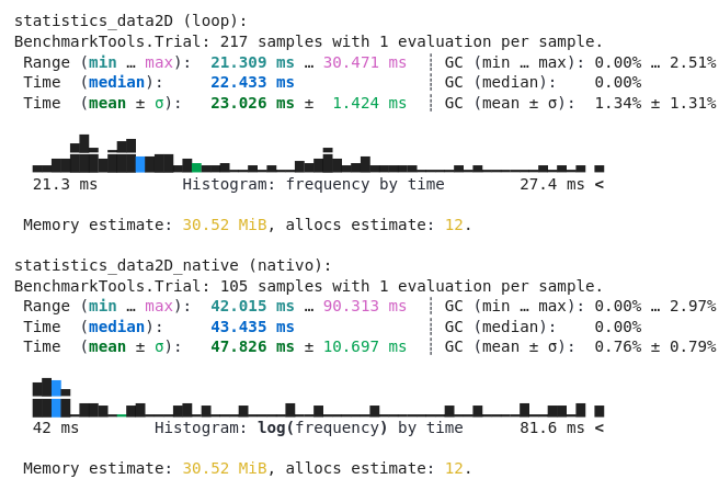


Figura 3.5: Comparação de desempenho entre as versões nativa e ciclo único da função `statistics_data`.

Comparação com a Implementação em FORTRAN

Face à versão original no AstroPGPLOT(FORTRAN 2008), a reimplementação AstroTLPlot manteve a fidelidade científica e introduziu novas métricas (variância, desvio padrão e coordenadas dos extremos), além de uma estrutura de dados mais expressiva e modular. A utilização de BenchmarkTools e de operações vetorizadas proporcionou tempos de execução competitivos e uma gestão de memória mais eficiente, reforçando a robustez da solução.

Conclusão

A função `statistics_data2D` demonstrou ser uma solução mais eficiente e expressiva, preservando a precisão e ampliando as capacidades analíticas em relação à versão original no AstroPGPLOT(FORTRAN 2008). O ganho de desempenho é particularmente relevante em contextos de grandes volumes de dados, onde a redução do tempo de processamento contribui diretamente para a eficiência global do ciclo de visualização e análise científica.

A análise comparativa das três abordagens *Loop-based* (AstroTLPlot), *Native Julia* e *Optimized Julia* (*Broadcasting*), demonstra que a escolha da técnica deve equilibrar eficiência computacional e legibilidade do código.

A implementação *Loop-based* revelou-se a solução mais eficiente, apresentando:

- **Menor tempo de execução** (10% mais rápido que as alternativas);
- **Menor consumo de memória** (quase metade das outras abordagens);
- **Menor número de alocações**, reduzindo a sobrecarga e aumentando a previsibilidade do desempenho.

Por outro lado, as versões *Native Julia* e *Broadcasting* mantêm a fidelidade científica e oferecem maior aderência ao estilo idiomático da linguagem, mas não apresentam ganhos significativos em desempenho. Em particular, a técnica de *broadcasting*, apesar de ser considerada uma otimização, mostrou resultados praticamente idênticos à versão nativa, tanto em tempo como em consumo de memória.

Recomendações:

- Para cenários onde desempenho e eficiência são prioritárias, recomenda-se a abordagem *Loop-based*.
- Quando a legibilidade, manutenção e integração com código idiomático assumem maior importância, as versões *Native Julia* ou *Broadcasting* são alternativas adequadas, embora sem vantagens significativas em termos de desempenho.

Esta flexibilidade permite ao investigador ou programador selecionar a implementação que melhor se adequa ao contexto específico do projeto, equilibrando desempenho computacional com qualidade de engenharia de software.

3.13.3 Testes de Visualização Científica

A substituição das rotinas de visualização baseadas em PGPLOT pelo pacote `Makie.jl` foi validada com sucesso, assegurando a fidelidade visual, a consistência numérica e o aumento da interatividade no processo de análise científica.

Validação e Exemplo Prático

Para validar a equivalência visual e numérica, foram gerados mapas bidimensionais e cortes nos planos XY e XZ das variáveis de densidade e energia, comparando-os diretamente com os gráficos originais produzidos no AstroPGPLOT(FORTRAN 2008). Os resultados confirmaram a correspondência pixel a pixel entre ambas as abordagens, uma vez que os valores numéricos subjacentes eram idênticos, demonstrando que a transposição preservou integralmente o conteúdo científico.

Além disso, foram testadas as principais funcionalidades interativas proporcionadas pelo `Makie.jl`, incluindo:

- Alternância entre escalas linear e logarítmica;
- Sobreposição de contornos e *colormaps* personalizados;
- Exportação para formatos de alta qualidade (PNG, PDF, SVG).

Benefícios da Nova Abordagem

A adoção do `Makie.jl` ampliou significativamente as capacidades de análise e comunicação científica do pacote, mantendo a equivalência visual com a versão no AstroPGPLOT(FORTRAN 2008), mas adicio-

nando:

- Melhor integração com o ecossistema Julia;
- Exportação vetorial de alta fidelidade;
- Possibilidade de futuras extensões para animações e visualização tridimensional.

3.13.4 Robustez e Tratamento de Exceções

Outro aspeto avaliado foi a robustez do pacote AstroTLPlot face aos erros de execução em condições inesperadas. Na versão original no AstroPGPLOT(FORTRAN 2008), certas falhas, como índices inválidos ou a ausência de ficheiros de entrada válidos, não interrompiam a execução do programa. Esta limitação podia levar à propagação de valores não inicializados e à produção de resultados incorretos sem qualquer aviso, uma situação crítica em contextos de simulação científica.

Durante a transposição para Julia e a implementação do pacote AstroTLPlot, foi desenvolvido um sistema estruturado de gestão de erros, garantindo que estas situações são detetadas e tratadas de forma controlada, aumentando significativamente a fiabilidade e a rastreabilidade da execução.

3.13.5 Gestão de Erros Estruturada no Pacote AstroTLPlot

O sistema de erros do pacote baseia-se em códigos de estado e mensagens definidas no módulo `constants.jl`, permitindo identificar com precisão a origem de cada falha. A Tabela 3.11 apresenta os principais códigos e mensagens de erro, incluindo falhas de leitura de ficheiros YAML, HDF5, CSV ou extensões incorretas.

Tabela 3.11: Principais códigos e mensagens de erro definidos no módulo `constants.jl`.

Código	Mensagem
0	Operation completed successfully.
1	Unknown or unexpected processing error.
2	File not found.
3	Read permissions denied or file access issue.
5	Incorrect file extension for expected format.
6	Failed to read data structure within the opened HDF5 file.
8	Failed to parse data structure within the opened CSV/DAT file.
9	File type is not supported by the universal open function.

Com esta abordagem, o pacote AstroTLPlot:

- Interrompe operações inválidas de forma controlada, evitando a propagação de resultados incorretos;
- Reporta explicitamente a natureza do erro ao utilizador, facilitando a depuração e rastreabilidade;
- Mantém a integridade científica e a robustez do pacote, mesmo quando os ficheiros de entrada estão ausentes ou mal formatados.

Exemplos Práticos

Durante os testes:

- O acesso a índices inválidos em estruturas como `SimulationData` gerou exceções do tipo `BoundsError`, devidamente capturadas e tratadas;
- A tentativa de leitura de ficheiros HDF5 inexistentes produziu mensagens de erro informativas, sem comprometer a execução do restante código.

3.13.6 Vantagens Observadas

Esta abordagem de gestão de exceções oferece dois benefícios principais:

- **Prevenção de execução incorreta:** os erros são tratados de forma controlada, evitando propagação de resultados inválidos;
- **Maior estabilidade e fiabilidade operacional:** o pacote torna-se mais previsível e resiliente em cenários reais, mesmo perante dados incompletos ou mal formatados.

Os diversos testes de visualização, gestão de exceções e manipulação de dados realizados durante a transposição e desenvolvimento do pacote AstroTLPlot permitiram validar cientificamente a ferramenta. A comparação com os gráficos originais no AstroPGPLOT(FORTRAN 2008) confirmou a equivalência visual e numérica, enquanto a implementação de mecanismos estruturados de tratamento de erros assegurou a execução controlada e a fiabilidade dos resultados [WBC⁺17, OR10].

Em conjunto, estes testes demonstram que o pacote AstroTLPlot:

- Garante resultados consistentes e cientificamente válidos;
- Mantém integridade numérica mesmo perante condições inesperadas ou dados incompletos;
- Permite análises interativas de alta qualidade com segurança computacional.

4

AstroTLPlot: Configurações e Casos de Estudo

4.1 Execução e Aplicação Prática do Pacote AstroTLPlot

O pacote AstroTLPlot, disponível publicamente no GitHub [LCdA25], foi concebido para fornecer um fluxo de trabalho completo em Julia, abrangendo desde a leitura de dados até ao processamento e visualização de simulações astrofísicas.

A sua conceção segue rigorosamente as boas práticas de modularização, reutilização de código e documentação integrada, recomendadas para o desenvolvimento moderno de software científico [BEKS17, Per19]. Para uma utilização correta do pacote, é necessário executar, numa fase inicial, dois passos fundamentais:

1. Instalação correta do pacote e das suas dependências;
2. Leitura e validação dos ficheiros com as configurações e dados de simulações.

O sucesso destas etapas garantem a robustez de todo o processo subsequente de análise e representação gráfica.

4.1.1 Instalação e Configuração do Pacote

A instalação do AstroTLPlot [LCdA25] segue o procedimento padrão do ecossistema Julia [The23b], recorrendo ao gestor de pacotes Pkg, que garante a gestão automática de dependências e compatibilidades

entre versões [BEKS17]. Para tal, basta executar:

```
using Pkg
Pkg.add("AstroTLPlot")
```

Este procedimento assegura que todas as bibliotecas auxiliares são instaladas e que as interfaces públicas (*API*) do pacote ficam disponíveis para utilização segura em qualquer ambiente Julia. Após a instalação, o próximo passo consiste na leitura e validação dos ficheiros de configuração e dados de simulação. Esta etapa é crucial para garantir a integridade dos dados antes de prosseguir com a análise.

4.2 Leitura e Validação de Ficheiros de Configuração e Dados

Os ficheiros de configuração e dados constituem a base de todo o processo de análise e visualização. Estes contêm os parâmetros fundamentais da simulação, as escalas físicas e espaciais, as variáveis físicas a representar, bem como as definições associadas à visualização, tais como esquemas de cores, rótulos, títulos e orientação dos mapas, ambos indispensáveis à inicialização e execução da simulação.

Durante a fase de leitura, o programa interpreta e valida o conteúdo desses ficheiros, assegurando que os valores carregados são consistentes, coerentes e compatíveis com as estruturas internas de dados. Este processo de verificação é essencial para garantir que todas as operações, desde o cálculo estatístico à representação gráfica, utilizam parâmetros físicos e numericamente corretos, preservando assim a integridade científica dos resultados.

4.2.1 Leitura de Ficheiro de Configuração — Função `load_simulation_config`

A leitura do ficheiro de configuração no formato YAML é efetuada por uma rotina dedicada `load_simulation_config`, responsável por carregar- tuplos com múltiplos conjuntos de parâmetros relativos à simulação, visualização e processamento de dados. A adoção desta abordagem modular favorece a manutenção, a extensibilidade e a rastreabilidade do código [Per19]. O carregamento dos dados pode ser realizado de duas formas equivalentes, consoante se pretenda obter as estruturas individualmente ou como uma única estrutura, permitindo uma inspeção detalhada dos dados carregados. O exemplo seguinte ilustra a utilização prática desta função:

Listagem 4.2.1 Exemplo em Julia da utilização da função `load_simulation_config` para leitura de ficheiro de configuração.

```
using AstroTLPlot
# Leitura do ficheiro de configuração
config_data, pgp_data, runtime_data, modions_data = load_simulation_config("path_to_config_file")
println(config_data)
println(pgp_data)
```

Como demonstrado na Listagem 4.2.1, a função retorna quatro estruturas distintas que contêm toda a parametrização necessária para o processamento subsequente. O `println` das estruturas, produz uma saída textual que evidencia a correta interpretação do ficheiro de configuração, conforme se pode observar na Figura 4.1.

Figura 4.1: Exemplo de saída parcial gerada pela função `load_simulation_config`.

```
ConfigData:
  Debug: Debug(true)
  Directories: Directories("data/input/dados-disk-2000x2000x1/", "./data/input/Atomic")
  Simulation Type: SimulationType(true, false)
  File Format: FileFormat(true, false, false)
PGPData:
  SetMinMaxVar: SetMinMaxVar(MinMaxRange{Float64}(-4.0, 1.5),
  MinMaxRange{Float64}(1.0, 7.0),
  MinMaxRange{Float64}(-15.0, -10.0),
  MinMaxRange{Float64}(0.0, 5.0),
  ...
```

Durante a fase de leitura, o programa interpreta e valida o conteúdo dos ficheiros, assegurando que os valores carregados são consistentes, coerentes e compatíveis com as estruturas internas de dados. Como evidenciado tanto pelo código da Listagem 4.2.1 como pela saída apresentada na Figura 4.1, este processo de verificação garante que todas as operações, desde o cálculo estatístico até à representação gráfica, utilizam parâmetros físicos e numericamente corretos, preservando assim a integridade científica dos resultados. A organização hierárquica dos dados nas estruturas bem definidas (`ConfigData`, `PGPData`, etc.) assegura uma clara separação de conceitos e facilita a manipulação dos parâmetros em fases posteriores do fluxo de trabalho.

Uma abordagem alternativa consiste na utilização de uma variável agregadora que reúne todos os conjuntos de parâmetros num único objeto de configuração, simplificando a inspeção e manipulação global dos dados. Esta metodologia é particularmente útil para uma visão consolidada de todo o ambiente de simulação:

```
all_config= load_simulation_config_struct(file_name)
println(all_config)
```

A execução deste código, produz uma saída formatada e unificada, exemplificada na Figura 4.2. Esta representação compacta facilita a verificação global da parametrização e revela-se especialmente vantajosa para depuração e validação de consistência entre os diversos módulos de simulação, visualização e processa-

Figura 4.2: Saída parcial formatada obtida com a leitura agregada pela função `load_simulation_config`.

```

SimulationSetup - Complete Configuration
=====
CONFIGURATION PARAMETERS
-----
Debug Settings:
  • Debug mode: ENABLED
Directories:
  • Main directory: data/input/dados-disk-2000x2000x1/
  • Atomic data directory: data/input/dados-disk-2000x2000x1/
Simulation Type:
  • HD run: YES
  • MHD run: NO
  ...
MODIONS PARAMETERS
-----
Ions Settings:
  • Min value: SetMinMaxIons(MinMaxRange{Float64}(-4.0, 0.0),
  MinMaxRange{Float64}(-10.0, -4.0))
=====
End of SimulationSetup configuration

```

Esta formatação de saída revela-se especialmente vantajosa para fins de depuração e verificação de consistência entre os diversos módulos de simulação, visualização e processamento. A estrutura hierárquica apresentada na Figura 4.2 permite ainda uma validação visual imediata das interdependências entre parâmetros, assegurando que todas as componentes do sistema estão corretamente configuradas e alinhadas com os requisitos da simulação.

4.2.2 Leitura de Ficheiros de Simulação — Função `readdata_hdf5`

Os ficheiros de simulação, geralmente nos formatos `*.hdf`, `*.hdf5`, `*.h5` ou `*.vtk`, armazenam as principais variáveis físicas resultantes das simulações, como densidade, energia e componentes de velocidade. A leitura destes dados é realizada através da função `readdata_hdf5!`, que carrega diretamente para memória as matrizes tridimensionais ou bidimensionais correspondentes, integrando-as nas estruturas internas do pacote, especificamente na estrutura *SimulationsData*.

Esta operação é fundamental para o cálculo de estatísticas, geração de mapas 2D e 3D, e construção de funções de distribuição de probabilidade (PDFs). A seguinte sequência de código ilustra a chamada típica à função e a respetiva inspeção dos dados carregados:

```

...
readdata_hdf5!(simulations_data, config_data, runtime_data)
print_struct_info(simulations_data)
...

```

A execução deste código produz uma saída estruturada que detalha as variáveis físicas lidas do ficheiro HDF5, tal como exemplificado na Figura 4.3. Esta saída evidencia a organização hierárquica dos dados e a consistência dimensional entre as diferentes variáveis armazenadas.

Figura 4.3: Exemplo de saída parcial gerada pela função `readdata_hdf5!`.

```

Nome: den      | Tipo: Array{Float64, 3} | Dimensões: (2000, 2000, 1)
Nome: ene      | Tipo: Array{Float64, 3} | Dimensões: (2000, 2000, 1)
...
Nome: vxx      | Tipo: Array{Float64, 3} | Dimensões: (2000, 2000, 1)
Nome: X_grid   | Tipo: Vector{Float64}   | Dimensões: (2000,)
Nome: Y_grid   | Tipo: Vector{Float64}   | Dimensões: (2000,)
Nome: Z_grid   | Tipo: Vector{Float64}   | Dimensões: (1,)

```

Para além da inspeção estrutural global, é possível examinar o conteúdo específico de variáveis individuais. A chamada da função auxiliar `print_vect(simulations_data.den, 3)` desenvolvida para verificação seguinte, demonstra a extração dos primeiros 3 valores da matriz tridimensional densidade(`simulations_data.den`): `print_vect(simulations_data.den, 3)`

A saída resultante da execução da função, apresentada na Figura 4.4, confirma a correta leitura e organização dos dados físicos, mostrando os valores numéricos específicos da variável de densidade nas primeiras posições da matriz.

Figura 4.4: Saída parcial da função `print_vect`, do dataset `simulations_data.den`: os primeiros 3 valores.

```

=== First 3 values of 3D array (dimensions (2000, 2000, 1)) ===
i      j      k      Value
---      ---      ---      -----
1      1      1      1.443324E+01
2      1      1      4.946554E+00
3      1      1      3.395136E+00

```

Esta abordagem sequencial, desde a leitura global dos dados até à inspeção detalhada de valores específicos, garante uma verificação completa da integridade dos dados carregados, assegurando que todas as operações subsequentes de análise e visualização partem de uma base de dados consistente e validada.

4.2.3 Leitura de Ficheiros de Configurações Iónicas — Função `ions_read`

Os ficheiros com as configurações iónicas contêm informações essenciais sobre as frações de ionização e variáveis associadas a cada elemento químico, sendo lidos através da função `ions_read`. Esta função carrega os dados necessários para a caracterização termodinâmica e radiativa do plasma, permitindo o cálculo de emissões e absorções em contextos astrofísicos [OF06]. A leitura é realizada através do seguinte código, onde o parâmetro `elem` especifica o elemento químico a processar:

```

tps = ions_read(config_data, pgp_data, runtime_data, modions_data, elem)
print_struct_info(tps)

```

A execução deste código produz uma saída detalhada da estrutura `tps`, conforme ilustrado na Figura 4.5. Esta saída evidencia a organização multidimensional dos dados iónicos carregados, incluindo tipos de dados e dimensões de cada matriz.

Figura 4.5: Exemplo de saída parcial gerada pela função `ions_read`, e da função `print_struct_info(tps)`.

```
Information about the fields in the struct:
Name: xion      | Type: Array{Float64, 3} | Dimensions: (10, 27, 161)
Name: xion_zs   | Type: Array{Float64, 3} | Dimensions: (10, 27, 161)
Name: xionvar   | Type: Array{Float64, 5} | Dimensions: (2000, 2000, 1, 10, 27)
Name: eleden    | Type: Array{Float64, 3} | Dimensions: (2000, 2000, 1)
Name: eledenz   | Type: Array{Float64, 4} | Dimensions: (2000, 2000, 1, 10)
Name: alogt     | Type: Vector{Float64}   | Dimensions: (161,)
...
```

A análise da saída apresentada na Figura 4.5 permite identificar a matriz `xionvar` como particularmente relevante, uma vez que armazena os diferentes estados de ionização para cada elemento. Esta estrutura multidimensional, com dimensões (2000, 2000, 1, 10, 27), constitui um componente crítico para os cálculos de emissão e absorção radiativa nas simulações astrofísicas subsequentes.

O sucesso desta fase de leitura é fundamental para todas as operações seguintes do fluxo de trabalho: cálculo de estatísticas globais e locais, geração de visualizações XY/XZ e PDFs, aplicação de transformações matemáticas e construção de subgráficos especializados.

O correto carregamento dos dados iónicos assegura que as análises termodinâmicas e radiativas posteriores partem de uma base de dados completa e validada, preservando a integridade científica de todo o processo de simulação.

4.3 Análise Estatística de Campos Astrofísicos

Para demonstrar o potencial e a aplicabilidade científica do pacote `AstroTLPlot`, foram realizados diversos casos de estudo aplicados a simulações astrofísicas. Estes estudos centravam-se em variáveis críticas, tais como tais como densidade, pressão e temperatura, fundamentais para a caracterização dos processos físicos que ocorrem em ambientes astrofísicos complexos [VMTP20].

As rotinas implementadas no pacote permitem calcular estatísticas fundamentais e distribuições de probabilidade sobre qualquer variável da simulação, utilizando o paradigma de *multiple dispatch* nativo de Julia [BEKS17]. Entre as métricas calculadas encontram-se valores médios, variâncias, desvio padrão e extremos locais, cuja combinação fornece uma visão quantitativa e interpretativa das condições médias e das flutuações no modelo.

Esta secção demonstra as capacidades de análise estatística implementadas no módulo de estatística do pacote `AstroTLPlot`, utilizando como caso de estudo o campo de densidade. Saliente-se que as mesmas rotinas são aplicáveis a qualquer variável física ou conjunto de dados multidimensional.

No `AstroPGPLOT` (implementado em FORTRAN), a análise estatística distribuía-se por várias rotinas específicas, sem um módulo unificado dedicado. O `AstroTLPlot` (desenvolvido em Julia) introduz um módulo especializado que centraliza o cálculo de estatísticas, incluindo soma, média, variância, desvio padrão e valores extremos, e assegura a sua integração direta com as funções de visualização.

O módulo estatístico do `AstroTLPlot` incorpora funcionalidades auxiliares que ampliam o rigor e a flexibilidade da análise:

- Localização automática das coordenadas espaciais associadas aos valores mínimo e máximo;
- Definição automática de limites de eixos para visualização;
- Conversão de tipos numéricos (`Float32` para `Float64`) para maior precisão;
- Aplicação opcional de transformações logarítmicas, essencial para variáveis com grande gama dinâmica.

A demonstração do cálculo estatístico é realizada em quatro contextos distintos:

1. Matriz tridimensional completa;
2. Corte bidimensional da simulação;
3. Região delimitada (janela) dentro de um corte;
4. Corte 2D com transformação logarítmica aplicada.

Estas funcionalidades constituem a base quantitativa para a análise física no AstroTLPlot, permitindo a caracterização precisa de tendências globais e a identificação de variações locais nos campos astrofísicos.

4.3.1 Situação 1: Cálculo de Estatísticas Globais em Matriz Tridimensional

A primeira situação avalia a matriz tridimensional de densidade completa, permitindo obter uma visão global das condições físicas simuladas. A função `statistics_data` calcula as estatísticas fundamentais da variável e retorna um dicionário estruturado com os resultados, enquanto `print_statistics` gera um sumário formatado para visualização no terminal. A análise é feita diretamente sobre a matriz tridimensional de densidade,, conforme ilustrado no seguinte excerto:

```
result_statistics = statistics_data(simulations_data.den)
code = export_statistics(result_statistics)
```

A execução deste código produz uma saída formatada que sintetiza as principais métricas estatísticas, conforme exemplificado abaixo:

```
1. GENERAL STATISTICS:
Sum: 5.571739012751547e6
Mean: 1.3929347531878866
Variance: 0.0
Standard Deviation: 0.0
Number of Elements: 4000000
Number of Rows: 2000
Number of Columns: 2000
...
```

Esta saída mostra de forma condensada as estatísticas descritivas fundamentais, incluindo soma, média, variância e valores extremos, bem como os limites espaciais do domínio analisado.

A estrutura de apresentação foi concebida para oferecer uma leitura rápida e organizada dos resultados, permitindo ao investigador avaliar instantaneamente a consistência e a homogeneidade dos dados tridimensionais da simulação. A precisão numérica demonstrada na saída confirma a robustez do módulo estatístico do AstroTLPlot no processamento de grandes volumes de dados astrofísicos.

4.3.2 Situação 2: Análise Estatística em Corte Bidimensional

A segunda situação analisa um corte bidimensional da matriz tridimensional de densidade, permitindo caracterizar a distribuição espacial das variáveis físicas num plano específico da simulação. Esta abordagem é particularmente relevante para a análise de secções XY ou XZ em ambientes astrofísicos tridimensionais, onde a inspeção planar facilita a identificação de estruturas e gradientes.

Para realizar a análise de um corte bidimensional, seleciona-se um índice fixo ao longo do eixo z, extraíndo assim uma secção plana da matriz tridimensional. O procedimento é implementado através do seguinte código:

```
z_index = 1
matriz_2d = simulations_data.den[:, :, z_index]
result_statistics = statistics_data(matriz_2d)
code = export_statistics(result_statistics)
```

A execução deste código gera uma saída formatada que sumariza as estatísticas descritivas do corte bidimensional, conforme demonstrado no exemplo seguinte:

1. GENERAL STATISTICS:

Sum: 5.571739012751547e6

Mean: 1.3929347531878866

Variance: 33.44367173255261

Standard Deviation: 5.783050383020418

Number of Elements: 4000000

Number of Rows: 2000

Number of Columns: 2000

2. EXTREME VALUES:

Minimum Value: 4.9977748712148275e-5

Minimum Coordinates: Point2D{Int64}(1974, 471)

Maximum Value: 485.58560326189314

Maximum Coordinates: Point2D{Int64}(99, 1514)

...

A análise do corte bidimensional evidencia flutuações locais significativas e permite a comparação sistemática entre regiões de alta e baixa densidade. A identificação das coordenadas espaciais associadas aos valores extremos, tal como demonstrado na saída, constitui informação crucial para a análise de instabilidades e gradientes de densidade no plasma simulado, facilitando a interpretação física das estruturas observadas.

4.3.3 Situação 3: Estatísticas em Região Delimitada (Janela Espacial)

A terceira situação foca a análise numa região de interesse específica da simulação, restringindo o domínio computacional para o estudo detalhado de propriedades locais. Esta abordagem permite isolar e caracterizar fenómenos astrofísicos que ocorrem em escalas espaciais reduzidas, como estruturas de choque, regiões de turbulência ou gradientes localizados.

Para delimitar a análise a uma subregião espacial, define-se uma janela retangular através das coordenadas $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$, que especifica os limites da área de estudo. A implementação desta análise é realizada através do seguinte código:

```
z_index = 1
matriz_2d = simulations_data.den[:, :, z_index]
result_statistics = statistics_data(matriz_2d, 300, 500, 300, 500)
code = export_statistics(result_statistics)
```

A execução do código produz uma saída estatística específica para a região delimitada, conforme exemplificado:

1. GENERAL STATISTICS:

Sum: 327844.6014215005

Mean: 8.114764521212358

Variance: 404.6097576013044

Standard Deviation: 20.11491381043688

Number of Elements: 40401

Number of Rows: 2000

Number of Columns: 2000

2. EXTREME VALUES:

Minimum Value: 0.0035709504359446387

Minimum Coordinates: Point2D{Int64}(439, 302)

Maximum Value: 376.35731105368404
 Maximum Coordinates: Point2D{Int64}(465, 380)

Esta análise regional revela-se particularmente relevante para o estudo de regiões onde se desenvolvem estruturas de choque ou gradientes físicos pronunciados. A comparação sistemática entre as propriedades locais, evidenciadas na janela espacial, e os valores médios globais da simulação permite quantificar desvios locais e identificar anomalias físicas. A precisão na localização das coordenadas dos valores extremos, tal como demonstrado na saída, fornece informação essencial para a interpretação física de processos astrofísicos em microescala.

4.3.4 Situação 4: Aplicação de Transformação Logarítmica aos Dados

A quarta situação demonstra a aplicação de transformações matemáticas aos dados, especificamente a transformação logarítmica, com o objetivo de realçar estruturas em diferentes escalas de densidade. Esta abordagem é particularmente útil para distribuições com grande variação de ordem de grandeza, comum em variáveis astrofísicas como densidade e temperatura.

A função `statistics_data` permite aplicar transformações aos dados antes do cálculo estatístico, mantendo a integridade dos valores originais. Neste exemplo, utiliza-se a transformação logarítmica de base 10 ($\log_{10}$) sobre os valores da matriz bidimensional:

```
z_index = 1
matriz_2d = simulations_data.den[:, :, z_index]
result_statistics = statistics_data(matriz_2d, log10)
print_vect(matriz_2d, 2, 3, 2, 3)
print_vect(result_statistics.matrix_result, 2, 3, 2, 3)
```

A execução deste código permite a comparação direta entre os valores originais e os transformados, conforme evidenciado nas saídas seguintes:

Matriz original (valores lineares):

```
=== Values of 2D array (dimensions (2000, 2000), range x[2:3], y[2:3]) ===
i    j      Value
---  ---  -
2    2      2.219104E-23
3    2      8.254305E-24
2    3      2.679023E-23
3    3      2.212574E-23
```

Matriz transformada (valores logarítmicos):

```
=== Values of 2D array (dimensions (2000, 2000), range x[2:3], y[2:3]) ===
i    j      Value
---  ---  -
2    2      -2.265382E+01
3    2      -2.308332E+01
2    3      -2.257202E+01
3    3      -2.265510E+01
```

Total elements displayed: 4

A transformação logarítmica demonstra a sua utilidade ao comprimir a ampla gama dinâmica dos valores originais (que variam em várias ordens de grandeza) para uma escala linearmente interpretável. Esta operação permite observar com maior clareza variações subtis dentro de cada ordem de grandeza,

facilitando a identificação de padrões e estruturas que permaneciam ocultas na representação linear. A capacidade de processar distribuições fortemente não lineares constitui uma funcionalidade essencial para a análise quantitativa de plasmas astrofísicos, onde variáveis como densidade e temperatura tipicamente exibem variações exponenciais.

4.3.5 Integração com a Visualização e Persistência de Dados

A função `statistics_data` foi concebida para ser perfeitamente integrada no fluxo de visualização. O cálculo estatístico é frequentemente realizado como um passo prévio à geração do gráfico, e os seus resultados são reutilizados para evitar processamento redundante. Este processo é ilustrado dentro da função `plot_heatmap`, onde as estatísticas são calculadas (aplicando-se `log10` se necessário) e a matriz já transformada é diretamente utilizada para a criação do gráfico:

```
...
stats = statistics_data2D(slice, log10)
stats == nothing && (stats = statistics_data2D(slice))
if log
    plotdata = stats.matrix_result # já calculada com log10
...
```

Esta integração direta permite que, quando é solicitada uma transformação logarítmica (`log = true`), a matriz resultante da transformação, armazenada no campo `matrix_result` da estrutura estatística, seja imediatamente utilizada para o plot, sem necessidade de recalcular ou reaplicar a função `log10` aos dados originais.

Para permitir o registo, análise externa ou reutilização dos resultados estatísticos gerados, o módulo inclui a função `export_statistics`. Esta função recebe o objeto retornado por `statistics_data` e escreve todas as métricas calculadas, incluindo média, desvio padrão, valores extremos e respetivas coordenadas, num ficheiro de texto formatado. A nomenclatura automática do ficheiro, que inclui o nome da variável analisada e um *timestamp* (ex.: `statistics_output-den-20250307_131218.txt`), garante a rastreabilidade e evita sobreposições:

Exemplo de utilização:

```
z_index = 1
matriz_2d = simulations_data.den[:, :, z_index]
result_statistics = statistics_data2D(matriz_2d)

# list generate by writeplot
list_of_files=["statistic-den-003000-042_091025_0149.txt"]

code = export_statistics(result_statistics; list_of_files=list_of_files, sav=true, disp=false)

if code == AstroTLPlot.STATUS_SAVE_SUCCESS
    println(" Success: ",AstroTLPlot.error_message(code)) #error_message(code))
else
    println(" Failure: ", error_message(code))
end
```

Saída esperada:

```
Success: Code 30: File saved successfully.
path = "./statistic-den-003000-042_091025_0149.txt"
```

Esta funcionalidade completa o ciclo de processamento, assegurando que os dados sumarizados, fundamentais para a interpretação científica, possam ser arquivados e consultados independentemente dos gráficos gerados.

4.3.6 Considerações Finais

Os resultados das quatro situações demonstradas evidenciam a versatilidade da função `statistics_data` e a capacidade do pacote `AstroTLPlot` para processar matrizes multidimensionais de grande dimensão, extraíndo propriedades estatísticas tanto globais como locais com elevado rigor numérico. A generalidade das funcionalidades demonstradas é sustentada por três características fundamentais do pacote:

- A metodologia apresentada é extensível a outros conjuntos de dados, como energia (`simulations_data.ene`) ou componentes de velocidade;
- A análise pode ser circunscrita a submatrizes específicas (janelas espaciais) para o estudo detalhado de regiões de interesse;
- O mecanismo de *multiple dispatch* inerente a Julia permite que a função estatística processe genericamente matrizes de diferentes dimensionalidades (1D, 2D ou 3D), adaptando-se automaticamente aos parâmetros fornecidos.

Esta flexibilidade assegura que o pacote `AstroTLPlot` não se limita a análises globais, sendo uma ferramenta robusta para estudos dirigidos a subdomínios espaciais e variáveis físicas distintas, adaptando-se às necessidades do investigador. Esta integração entre cálculo e visualização constitui uma abordagem moderna de análise científica interativa, alinhada com as boas práticas em visualização científica de alto desempenho [DK21]. A adoção desta metodologia assegura uma interpretação quantitativa rigorosa e uma exploração visual eficiente das simulações astrofísicas.

4.4 Visualização de Campos com Cortes Planos

A visualização dos resultados das simulações constitui um dos pilares do pacote `AstroTLPlot` desenvolvido, permitindo explorar grandezas físicas tanto numa perspetiva global como local. O módulo de visualização foi implementado em `Makie.jl`, possibilitando a criação de mapas bidimensionais obtidos a partir de cortes nos planos XY e XZ. Estes cortes fornecem uma representação intuitiva da distribuição espacial de variáveis como densidade, energia, pressão, temperatura e estados iónicos, facilitando a interpretação física dos resultados.

Esta abordagem segue as boas práticas de visualização científica moderna, beneficiando da renderização acelerada por GPU e das capacidades interativas de `Makie.jl` [DK21]. Para demonstrar estas capacidades, são apresentados diferentes representações gráficas, mapas de calor, de contorno e combinações de ambos, geradas de forma modular através das funções do pacote. Os casos demonstrativos utilizam o conjunto de dados de densidade, embora as mesmas funções possam ser aplicadas a qualquer variável da simulação.

4.4.1 Situação 1: Mapa de Calor (Heatmap)

O mapa de calor constitui uma das metodologias mais diretas para visualizar a distribuição espacial de variáveis físicas em simulações astrofísicas. Cada ponto do plano XY (ou XZ) é representado por uma cor cuja intensidade corresponde ao valor da grandeza em análise, como densidade ou energia, permitindo identificar rapidamente regiões de maior ou menor concentração e oferecendo uma perceção imediata do comportamento global do sistema simulado.

A geração do mapa de calor é realizada através da função `plot_heatmap`, que implementa escalas de cor contínuas seguindo convenções de visualização científica estabelecidas para a interpretação intuitiva de campos escalares multidimensionais [War13]. A seguinte implementação ilustra a chamada típica desta função:

```
using AstroTLPlot
```

```

...
plot_heatmap(xg, yg, slice, var_name,
             smin, smax;
             is_ion,
             kern, ionz,
             xlabel = pgp.labels.xlabel,
             title = pgp.title.title,
             author = pgp.title.auth ylabel = pgp.labels.ylabel, or,
             resolution = pgp.title.resolution,
             unittime = pgp.title.unittime,
             colormap = :viridis,
             min_color = :blue,
             max_color = :red,
             show_stats = false,
             logscale = cfg.scales.logs,
             stats = stats
             )

```

A execução deste código produz a visualização apresentada na Figura 4.6, que exemplifica a aplicação da função ao campo de densidade no plano XY. A figura demonstra a eficácia da representação por mapa de calor na identificação de estruturas espaciais e gradientes físicos, com a paleta de cores viridis a facilitar a distinção entre regiões de diferentes valores de densidade.

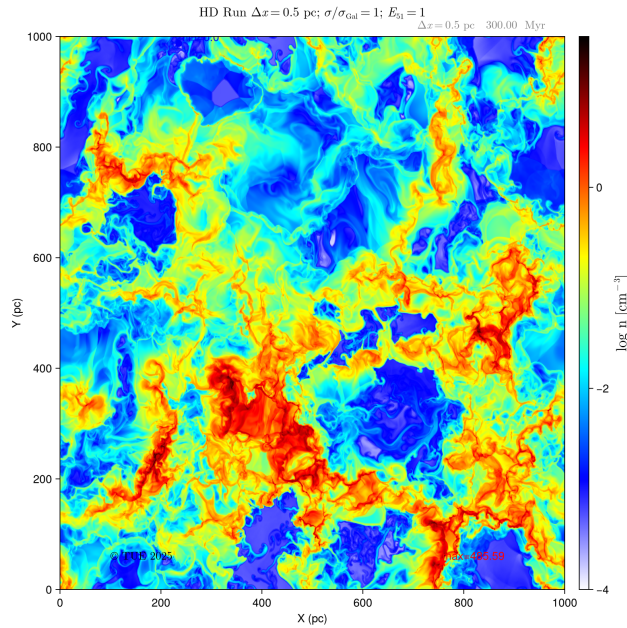


Figura 4.6: Mapa de calor de densidade no plano XY, gerado com a função `plot_heatmap` do AstroTLPlot.

4.4.2 Situação 2: Mapa de Contorno (Contour Map)

O mapa de contorno representa as linhas de nível (isolinhas) da variável física em estudo, permitindo identificar gradientes espaciais e delimitar fronteiras estruturais no domínio simulado. Esta forma de visualização é particularmente eficaz para evidenciar variações suaves e demarcar regiões com valores similares, facilitando a análise quantitativa de transições físicas.

A geração de mapas de contorno é realizada através da função `plot_heat_cont`, que calcula e representa as curvas de nível com base na matriz bidimensional de dados. A implementação seguinte demonstra

a utilização desta função:

```
using AstroTLPlot
...
plot_heat_cont(xg, yg, slice, var_name, smin, smax;
is_ion, kern, ionz,
xlabel = pgp.labels.xlabel,
title = pgp.title.title,
author = pgp.title.auth ylabel = pgp.labels.ylabel, or,
resolution = pgp.title.resolution,
unittime = pgp.title.unittime,
colormap = :viridis,
min_color = :blue,
max_color = :red,
show_stats = false,
logscale = cfg.scales.logs,
stats = stats
)
```

A execução deste código produz a visualização apresentada na Figura 4.7, que ilustra a aplicação da função ao campo de densidade no plano XY. A figura demonstra a capacidade do mapa de contorno em destacar as regiões de transição e as fronteiras principais da distribuição espacial, com as linhas de nível a representarem intervalos constantes de densidade.

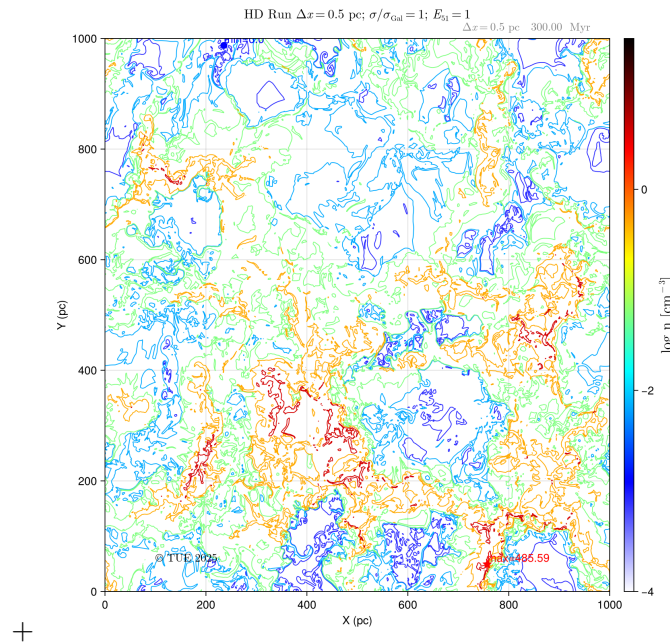


Figura 4.7: Mapa de contorno de densidade no plano XY, obtido com a função `plot_cont` do `AstroTLPlot`.

4.4.3 Situação 3: Mapa Combinado (Heatmap + Contour)

O mapa combinado integra simultaneamente a visualização contínua da densidade através do mapa de calor com a representação das linhas de contorno (isolinhas). Esta abordagem híbrida permite uma análise mais completa dos campos físicos, combinando a percepção global proporcionada pela cor com a precisão quantitativa oferecida pelas curvas de nível.

A geração desta visualização combinada é realizada através da função `plot_heat_cont`, que sobrepõe automaticamente as linhas de contorno ao mapa de calor base. A implementação seguinte ilustra a utilização desta função para criar representações visuais integradas:

```
using AstroTLPlot
...
plot_heat_cont(xg, yg, slice, var_name, smin, smax;
is_ion, kern, ionz,
xlabel = pgp.labels.xlabel,
title = pgp.title.title,
author = pgp.title.auth ylabel = pgp.labels.ylabel, or,
resolution = pgp.title.resolution,
unittime = pgp.title.unittime,
colormap = :viridis, min_color = :blue, max_color = :red,
show_stats = false,
logscale = cfg.scales.logs,
stats = stats
)
```

A execução deste código produz a visualização integrada apresentada na Figura 4.8, que demonstra a eficácia da abordagem combinada para a análise do campo de densidade no plano XY. Esta representação fornece simultaneamente uma visão global da distribuição espacial através da codificação por cores e a precisão quantitativa das isolinhas, permitindo uma análise visual detalhada das estruturas físicas.

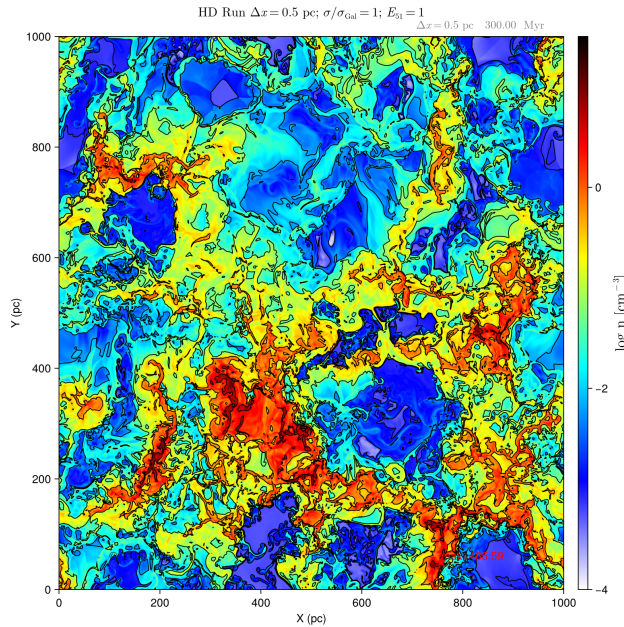


Figura 4.8: Mapa combinado de calor e contorno da densidade no plano XY.

4.4.4 Situação 4: Função de Densidade de Probabilidade (PDF)

De forma complementar à análise espacial, torna-se fundamental caracterizar a distribuição estatística dos valores das variáveis físicas simuladas. A Figura 4.9 apresenta a função de densidade de probabilidade (PDF) da variável densidade, representada sob a forma de histograma. Este tipo de análise permite visualizar a frequência relativa de ocorrência de intervalos de valores, facilitando a identificação de regimes dominantes, caudas estatísticas e padrões de dispersão característicos de fenômenos fisicamente relevantes.

A geração da PDF é realizada através da função `plot_pdf_test`, desenvolvida especificamente no âmbito do pacote `AstroTLPlot`. Esta função integra o cálculo automático de estatísticas descritivas (média, desvio padrão, valores mínimo e máximo), a possibilidade de aplicar transformações logarítmicas e a configuração dinâmica dos parâmetros de visualização definidos nos módulos da simulação. A seguinte implementação ilustra a sua utilização:

```
plot_pdf_test(xg, yg, slice, var_name;
  kern, ionz, is_ions,
  author = pgp.title.author,
  resolution = pgp.title.resolution,
  unittime = pgp.title.unittime,
  show_stats = false,
  logscale = cfg.scales.logs,
  stats = stats)
```

A execução deste código produz a análise estatística apresentada na Figura 4.9, que demonstra a distribuição de probabilidade do campo de densidade. A representação em histograma permite identificar os valores mais prováveis, a dispersão estatística e a existência de caudas na distribuição, informação essencial para a caracterização física do sistema simulado.

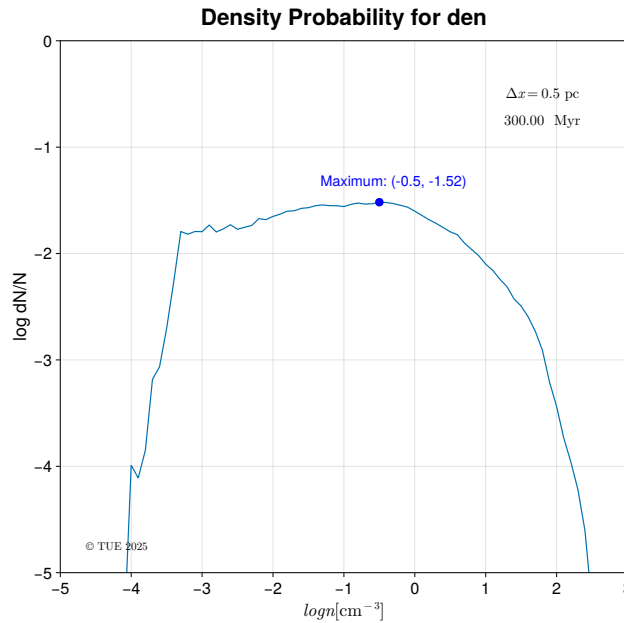


Figura 4.9: Função de distribuição de probabilidade (PDF) da densidade, gerada com a função `plot_pdf_test` do pacote `AstroTLPlot`.

Os quatro tipos de visualização apresentados complementam-se sinergicamente na análise das variáveis físicas simuladas: o *heatmap* oferece uma percepção global imediata da distribuição espacial, o mapa de contorno destaca com precisão os gradientes e fronteiras estruturais, e o mapa combinado integra ambas as perspectivas para uma análise visual mais completa. A função de densidade de probabilidade (PDF) complementa esta análise providenciando a caracterização estatística da distribuição de valores, identificando regimes dominantes e comportamentos nas caudas da distribuição.

A aplicabilidade uniforme destas funções a diferentes variáveis físicas, energia, temperatura, pressão, estados iónicos, evidencia a modularidade e flexibilidade do pacote `AstroTLPlot`. Esta abordagem unificada entre cálculo numérico e visualização científica reforça a integração conceptual do sistema, alinhando-se com as práticas contemporâneas de visualização científica de alto desempenho [DK21, War13].

4.4.5 Análise Comparativa: Visualização de PDFs no AstroPGPLOT vs AstroTLPlot

A Figura 4.10 apresenta uma análise comparativa direta entre a representação da função de densidade de probabilidade (PDF) de densidade gerada pelo AstroPGPLOT (implementado em FORTRAN/PGPLOT) e a nova implementação desenvolvida no AstroTLPlot (em Julia/Makie). Esta comparação evidencia a evolução das capacidades de visualização científica e a maior completude informativa alcançada.

A subfigura 4.10a ilustra o PDF produzido pelo AstroPGPLOT, que fornece uma representação cientificamente correta dos dados através da biblioteca PGPLOT, uma solução robusta e multiplataforma desenvolvida por T. J. Pearson em 1993, que permitiu durante décadas a geração de gráficos de qualidade de publicação diretamente integrados em software científico em execução. A versão do AstroTLPlot, apresentada na subfigura 4.10b, incorpora um conjunto abrangente de melhorias visuais, funcionais e informativas que representam uma evolução face à implementação original:

- **Anotação automática de coordenadas extremas:** Marcador e valor explícito no máximo da distribuição;
- **Metadados completos:** Título descritivo, legendas em $\text{\LaTeX}$ com notação científica correta;
- **Informação de autoria:** Direitos de autor e atribuição integrados na visualização;
- **Elementos visuais avançados:** Cor personalizada da curva do PDF, grade de referência e eixos calibrados;
- **Contexto estatístico:** Integração de estatísticas descritivas diretamente no gráfico (opcional);
- **Exportação nativa** em formatos matriciais e vetoriais de alta resolução.

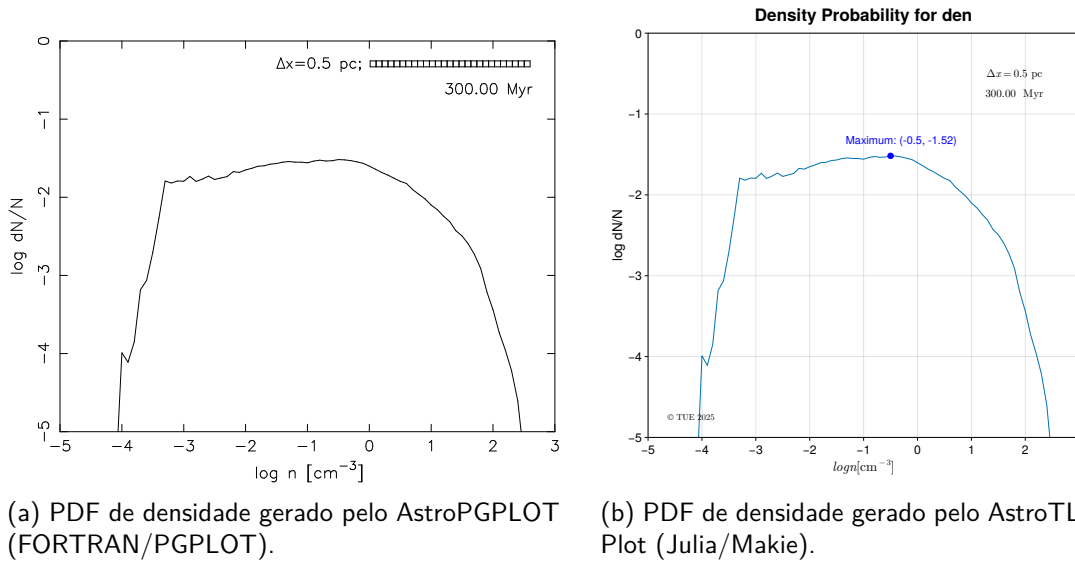


Figura 4.10: Análise comparativa das funções de densidade de probabilidade (PDF) geradas pelo AstroPGPLOT (esquerda) e pelo AstroTLPlot (direita).

O desenvolvimento do AstroTLPlot representa um avanço na completude da representação científica. A capacidade de integrar automaticamente coordenadas extremas, metadados descritivos e informação contextual diretamente na visualização elimina a necessidade de pós-processamento manual e assegura a rastreabilidade completa dos resultados. Esta evolução, suportada pelas capacidades do ecossistema Julia e do Makie.jl [DK21], alinha-se com as práticas contemporâneas de visualização científica [War13] e introduz novas funcionalidades que complementam as capacidades estabelecidas pela implementação original.

4.4.6 Visualização Integrada de Múltiplas Representações da Densidade

Uma das inovações introduzidas no AstroTLPlot é a capacidade de integrar múltiplas representações visuais num único painel gráfico coordenado. A Figura 4.11 ilustra um painel unificado que combina quatro visualizações complementares do campo de densidade: mapa de calor, o mapa de contorno, visualização combinada e função densidade de probabilidade (PDF).

Esta abordagem integrada permite uma análise comparativa e a correlação direta entre propriedades locais e globais da simulação, oferecendo uma perspectiva holística do sistema astrofísico em estudo. A função responsável por esta funcionalidade, `mapas_subplot!`, foi desenvolvida especificamente para o pacote AstroTLPlot e representa uma evolução face ao AstroPGPLOT, que gerava visualizações individuais através da biblioteca PGPLOT.

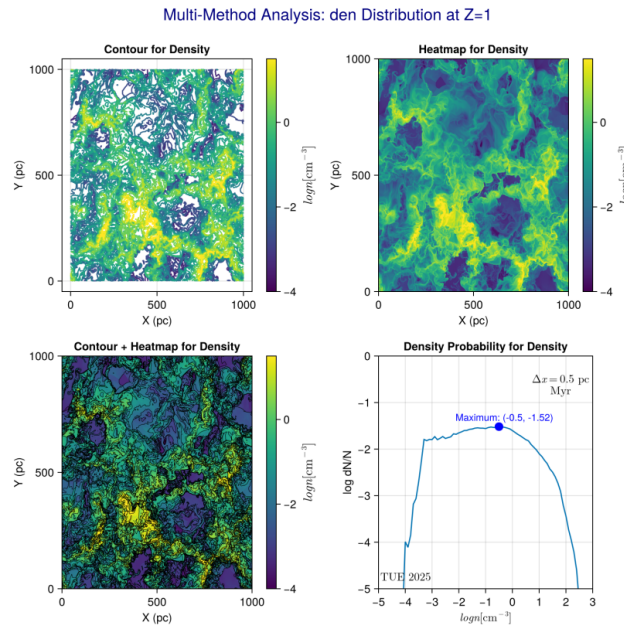


Figura 4.11: Painel integrado reunindo as quatro representações complementares do campo de densidade: mapa de calor (superior direito), mapa de contorno (superior esquerdo), combinação calor+contorno (inferior esquerdo) e função de densidade de probabilidade - PDF (inferior direito).

A implementação desta funcionalidade é realizada através da seguinte chamada da função:

```
mapas_subplot!(simulations_data.den,
simulations_data.X_grid, simulations_data.Y_grid, simulations_data.Z_grid, "den";
cfg = config_data,
pgp = pgp_data,
rt = runtime_data,
modions = modions_data)
```

A execução deste código gera a visualização integrada apresentada na Figura 4.11, demonstrando a eficácia da abordagem unificada na análise compreensiva do campo de densidade. A disposição coordenada das quatro representações permite ao investigador correlacionar diretamente características espaciais identificadas nos mapas com a distribuição estatística global evidenciada pelo PDF.

4.4.7 Visualização de Estados Iónicos do Hidrogénio

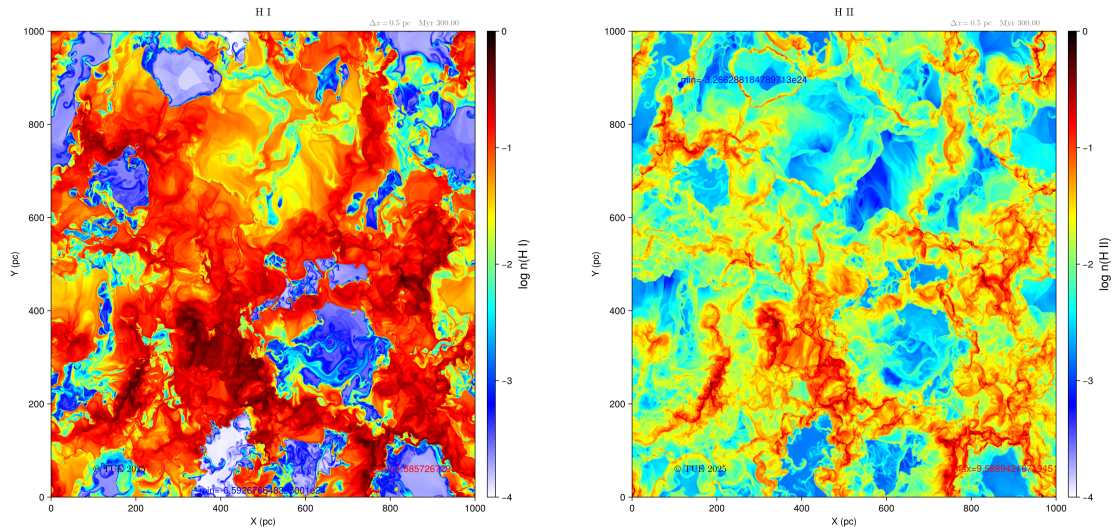
O pacote AstroTLPlot permite a representação discriminada dos diferentes estados iónicos de elementos químicos em simulações astrofísicas, funcionalidade essencial para estudar processos de ionização em plas-

mas. No caso específico do hidrogénio, as Figuras 4.12a e 4.12b ilustram as distribuições espaciais do estado neutro (H^0) e do estado ionizado (H^{1+}), respetivamente.

Esta capacidade analítica avançada permite caracterizar as regiões onde o elemento se encontra predominantemente neutro (H^0) ou ionizado (H^{1+}), fornecendo informação crucial para a compreensão dos mecanismos de ionização-radiativa e colisional. A geração destes mapas iónicos é realizada através de funções especializadas do pacote que processam as frações de ionização calculadas pelo código de simulação.

```
# Exemplo de código para geração de mapas iónicos
plot_ionic_species(simulations_data, "H", 0) # Hidrogénio neutro (H)
plot_ionic_species(simulations_data, "H", 1) # Hidrogénio ionizado (H)
```

A análise comparativa dos mapas apresentados na Figura 4.12 permite identificar gradientes de ionização, delimitar zonas de transição e estabelecer correlações espaciais com outras variáveis físicas, como temperatura e densidade eletrónica. Através da visualização no plano XY, é possível caracterizar a estrutura espacial dos processos de ionização, identificando regiões de fotoionização, frentes de ionização e zonas de equilíbrio termodinâmico.



(a) Mapa de densidade do Hidrogénio neutro (H^0).

(b) Mapa de densidade do Hidrogénio ionizado (H^{1+}).

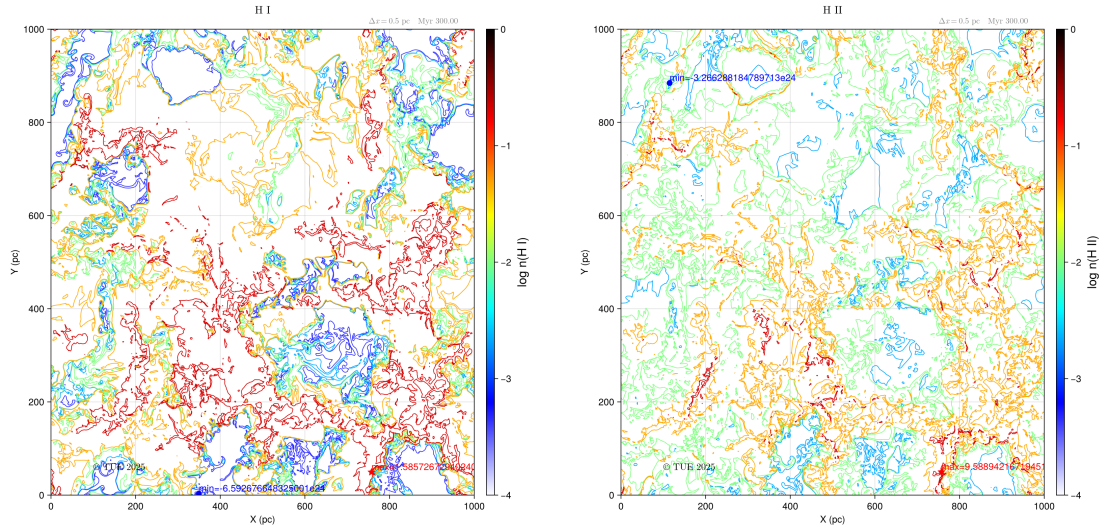
Figura 4.12: Distribuição espacial dos estados iónicos do hidrogénio no plano XY: (a) estado neutro (H^0) e (b) estado ionizado (H^{1+}).

Estas visualizações especializadas constituem ferramentas poderosas para a investigação de processos micro-físicos em meios astrofísicos, permitindo correlacionar a estrutura iónica com propriedades macroscópicas do plasma [OF06, Dra11, VMTP20]. A capacidade de analisar separadamente cada estado de ionização representa um avanço significativo na caracterização termodinâmica de simulações numéricas em astrofísica.

4.4.8 Análise de Gradientes Iónicos através de Mapas de Contorno

Os mapas de contorno permitem representar, de forma mais precisa, as variações graduais da densidade iónica. No caso do hidrogénio, o estado neutro (H^0) e ionizado (H^{1+}) exibem padrões distintos que refletem as condições físicas do meio astrofísico, nomeadamente as zonas de transição entre regiões frias e quentes, e a presença de frentes de ionização.

Através das curvas de nível, é possível observar gradientes espaciais que o mapa de calor não evidencia com a mesma clareza. Estas representações são essenciais para comparar a morfologia dos estados iônicos e identificar estruturas dinâmicas associadas à evolução da simulação. A Figura 4.13 apresenta os contornos correspondentes aos estados H^0 e H^{1+} , obtidos a partir dos mesmos cortes XY utilizados anteriormente para os mapas de calor, permitindo uma análise comparativa direta.



(a) Mapa de contorno da densidade iônica do hidrogénio neutro (H^0).

(b) Mapa de contorno da densidade iônica do hidrogénio ionizado (H^{1+}).

Figura 4.13: Mapas de contorno de densidade dos estados iônicos do hidrogénio: (a) neutro (H^0) e (b) ionizado (H^{1+}).

A análise conjunta dos mapas de contorno permite caracterizar com precisão a geometria das interfaces entre regiões neutras e ionizadas, fornecer estimativas quantitativas dos gradientes de densidade iônica e identificar assinaturas morfológicas de processos físicos específicos, como frentes de choque ou regiões de fotoionização. Esta capacidade analítica avançada constitui um valor acrescentado significativo do AstroTLPlot para o estudo de plasmas astrofísicos.

4.4.9 Análise Integrada de Estados Iônicos do Azoto - $N00^+$ a $N07^+$

A análise integrada de todos os estados de ionização de um elemento constitui uma funcionalidade avançada do pacote AstroTLPlot, permitindo uma caracterização abrangente da estrutura termodinâmica do plasma astrofísico. A Figura 4.14 apresenta o painel completo dos estados iônicos do azoto, desde o estado neutro (N) até ao estado totalmente ionizado (N^{7+}), organizado automaticamente numa grelha de visualização coordenada.

Esta representação integrada é gerada pela função `plot_element_subplot`, desenvolvida especificamente para o AstroTLPlot, que automatiza a criação de painéis comparativos para qualquer elemento químico. A implementação para o azoto é realizada através do seguinte código:

Listagem 4.4.1 Chamada da função `plot_element_subplot` em Julia para o elemento químico com número atómico = 7 (Azoto).

```
1 plot_element_subplot(tps, simulations_data, elem, 7, ncols=4, save_dir="./subplots")
```

A execução deste código produz a visualização apresentada na Figura 4.14, que dispõe sequencialmente

os oito estados de ionização do azoto. Esta disposição sistemática permite analisar a evolução espacial do grau de ionização e correlacioná-la diretamente com a estrutura térmica do meio. A progressão desde íons neutros (N^0) até íons altamente ionizados (N^7) reflete condições locais de temperatura e densidade eletrônica, seguindo padrões coerentes com o equilíbrio termodinâmico esperado.

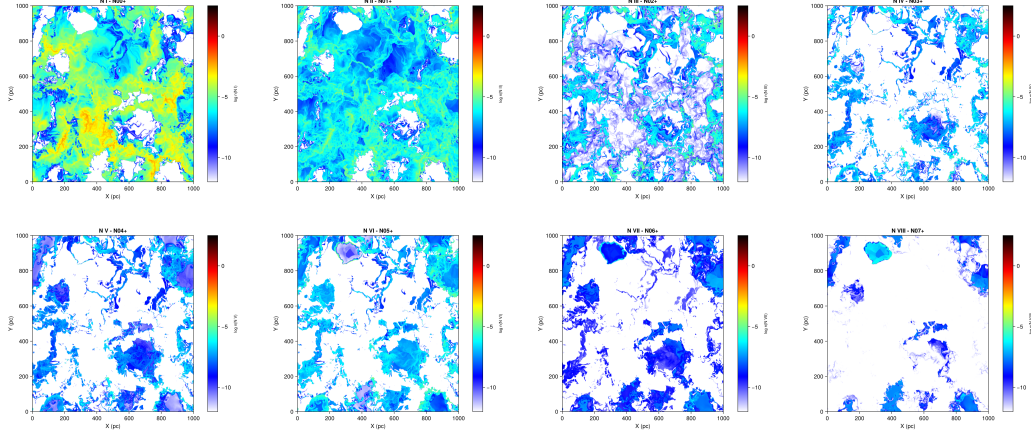


Figura 4.14: Painel completo dos estados iônicos do azoto (N^0 a N^{7+}).

A análise conjunta revela que as zonas mais energéticas exibem predominância de íons com carga elevada (N^6 , N^{7+}), enquanto as regiões mais frias e densas mantêm maior fração de íons neutros e pouco ionizados (N^0 , N^{1+}). Esta abordagem visual é particularmente valiosa para validar a consistência física da simulação, uma vez que a distribuição espacial dos diferentes íons deve seguir padrões coerentes com os mecanismos de ionização predominantes. A funcionalidade é genérica e pode ser aplicada a qualquer elemento químico, representando uma ferramenta poderosa para o estudo de plasmas astrofísicos multi-ionizados.

4.4.10 Visualização com Estatísticas Integradas

O pacote AstroTLPlot introduz uma funcionalidade inovadora: a integração direta de estatísticas descritivas nos próprios mapas de visualização. A Figura 4.15 ilustra esta abordagem, onde métricas como média, desvio padrão, valores extremos e número de elementos são apresentadas diretamente sobre o gráfico, eliminando a necessidade de consulta a relatórios externos e facilitando a interpretação imediata dos dados.

Esta capacidade, suportada pela flexibilidade do `Makie.jl` [DK21], permite a personalização avançada da apresentação das anotações estatísticas, incluindo posição, tamanho, formato e estilo tipográfico, otimizando a legibilidade sem comprometer a clareza do mapa principal. A implementação é realizada através de parâmetros configuráveis na função de visualização:

```
plot_heatmap(xg, yg, slice, var_name, smin, smax;
show_stats = true,           # Ativar estatísticas
...
)
```

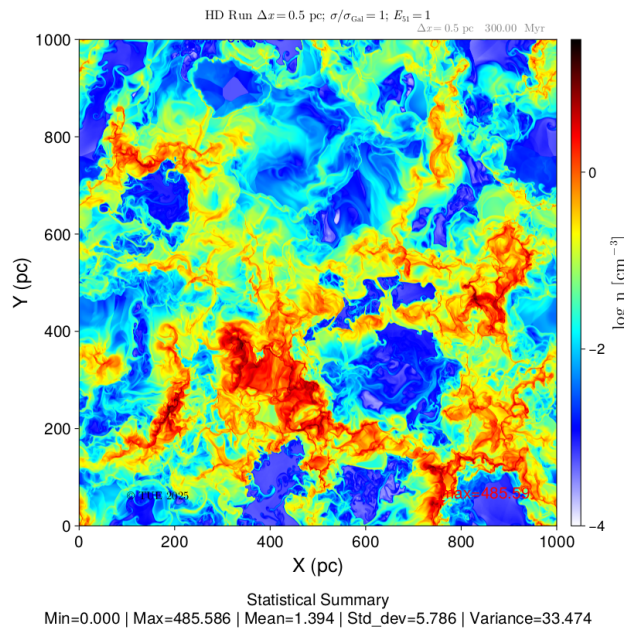



Figura 4.15: Mapa de calor de densidade com estatísticas integradas, gerado pelo pacote AstroTLPlot.

As vantagens desta abordagem incluem:

- **Contextualização imediata:** Interpretação quantitativa direta dos padrões visuais;
- **Personalização avançada:** Adaptação da apresentação estatística ao tipo de visualização;
- **Transparência analítica:** Manutenção da rastreabilidade entre dados e estatísticas;
- **Eficiência workflow:** Eliminação de processos manuais de extracção de métricas.

4.4.11 Integração entre Visualização e Análise Quantitativa

A abordagem multimodal implementada no AstroTLPlot, que integra mapas de calor, contornos, histogramas e subplots, transcende a mera ilustração gráfica, constituindo-se como uma componente analítica essencial para a interpretação de fenómenos astrofísicos complexos. A conjugação de diferentes perspetivas e níveis de detalhe potencia uma leitura robusta, verificável e cientificamente fundamentada dos resultados numéricos, promovendo a reprodutibilidade e fiabilidade da análise computacional.

Esta evolução consolida o carácter analítico do AstroTLPlot, que evolui de uma ferramenta essencialmente gráfica (na versão original em FORTRAN/PGPLOT) para um ambiente integrado de visualização e análise quantitativa, alinhado com as práticas contemporâneas de computação científica [WBC⁺17, DK21, War13].

4.5 Demonstração do Sistema de Nomes e Estrutura de Saída

Nesta secção apresentamos a aplicação prática do sistema de organização de saída implementado no AstroTLPlot. Através das funções `writeln` e `save_or_display`, o utilizador gera automaticamente os nomes de ficheiros, os diretórios correspondentes e os formatos de exportação pretendidos.

O exemplo seguinte ilustra a criação de mapas de densidade global em múltiplos formatos. A função `writeln` gera o nome do ficheiro conforme a convenção definida, enquanto `save_or_display` cria o diretório adequado (neste caso, `den/`) e armazena fisicamente as imagens produzidas:

Listagem 4.5.1 Chamada da função `writeplot` para geração de nomes de ficheiro para densidade em múltiplos formatos.

```

1  files = writeplot(42,
2  lpdf = true,
3  color = true,
4  variavel = var_name,
5  atom = 8,
6  filenum = 3000,
7  add_datetime = true,
8  datetime_format = "ddmmyy_HHMM",
9  formats = ["pdf", "png"]
10 )

```

A execução do código da Listagem 4.5.1 produz a seguinte saída, confirmando a geração dos nomes de ficheiro:

```

[SUCESSO] Ficheiro gerado: PDF-den003000-042_011025_0325.pdf
[SUCESSO] Ficheiro gerado: PDF-den003000-042_011025_0325.png

```

Posteriormente, a função `save_or_display` utiliza estes nomes gerados para criar a estrutura hierárquica de diretórios e efectuar o armazenamento físico dos ficheiros:

Listagem 4.5.2 Chamada da função `save_or_display` para armazenamento físico dos ficheiros.

```

1  save_or_display(
2  fig, files;
3  sav = true,
4  disp = true,
5  output_path = "./data/output/mapas/teste"
6  )

```

A execução da Listagem 4.5.2 resulta na criação da estrutura de diretórios e no armazenamento efectivo dos ficheiros:

```

[ Info: Figure saved: ./data/output/mapas/color/den/PDF-den-003000-042_011025_0325.pdf
[ Info: Figure saved: ./data/output/mapas/color/den/PDF-den-003000-042_011025_0325.png

```

A análise do fluxo completo demonstra que o sistema implementa correctamente a convenção de nomenclatura definida, combinando os parâmetros especificados no código: o tipo de visualização (PDF), a variável (den), o número sequencial (3000), o identificador (42) e o carimbo temporal (011025_0325). A integração entre a geração de nomes (`writeplot`) e o armazenamento físico (`save_or_display`) garante uma automatização completa do processo de exportação, promovendo as boas práticas de ciência reprodutível e gestão sistemática de dados científicos [SNTH13].

Este fluxo confirma, na prática, a robustez e consistência do sistema de organização de saída, assegurando que todas as visualizações produzidas no contexto astrofísico são arquivadas de forma sistemática e rastreável.

A implementação representa um ganho substancial face ao `AstroPGPLOT`, que armazenava todos os ficheiros numa única pasta sem metadados estruturados. Ao estruturar hierarquicamente os resultados e incorporar metadados descritivos diretamente nos nomes dos ficheiros, o novo sistema reforça significativamente a aderência aos princípios FAIR (Findable, Accessible, Interoperable, Reusable), garantindo que os dados científicos sejam não apenas preservados, mas também prontamente localizáveis, interpretáveis e reutilizáveis pela comunidade científica.

4.6 Documentação e Repositório GitHub

O pacote AstroTLPlot inclui documentação abrangente que cobre todas as funcionalidades do software, desde as rotinas de leitura de dados até à geração de visualizações e cálculo de estatísticas. A documentação encontra-se organizada em secções temáticas claras, complementadas com exemplos de utilização prática em Julia e explicações detalhadas sobre o funcionamento de cada função, facilitando a aprendizagem e integração no fluxo de trabalho do utilizador.

Esta abordagem representa uma evolução significativa face ao AstroPGPLOT (implementado em FORTRAN), cuja documentação se encontrava maioritariamente no código fonte, tornando a compreensão das rotinas dependente da leitura direta do código fonte. A versão Julia para o AstroTLPlot adota uma metodologia orientada ao utilizador, permitindo que o pacote seja facilmente compreendido, mantido e expandido pela comunidade científica, em conformidade com as boas práticas de desenvolvimento de software científico [WAB⁺14, BEKS17].

A documentação integra *docstrings* detalhadas para as funções principais, descrevendo parâmetros de entrada, saídas esperadas e comportamento de cada rotina. Complementarmente, inclui notas em *Markdown* com explicações contextuais e recomendações de utilização em cenários práticos, assegurando que os investigadores possam aceder rapidamente a informação relevante sem necessidade de consultar documentação externa.

4.6.1 Documentação Técnica e Integração com Julia

O pacote AstroTLPlot implementa documentação técnica seguindo as boas práticas do ecossistema Julia, através de *docstrings* estruturadas em formato *Markdown*. Estas descrevem detalhadamente o propósito de cada função, argumentos de entrada, valores de retorno e observações sobre o comportamento interno.

O exemplo seguinte ilustra o padrão adotado na documentação da função principal de processamento:

Listagem 4.6.1 Exemplo de documentação em Julia para a função `process_simulation_list`.

```

1  """
2  process_simulation( simdata::SimulationData; cfg::ConfigData, pgp::PGPData, rt::RuntimeData,
3  modions::ModionsData, variable_names::Vector{String} )
4
5  Main simulation processing function.
6
7  Instead of extracting variable names from `SimulationData`,
8  this version takes a pre-validated list of variable names (`variable_names`)
9  and performs preprocessing for each one.
10
11 # Arguments
12 - `simdata::SimulationData`: Main simulation data object
13 - `cfg::ConfigData`: Configuration data
14 - `pgp::PGPData`: Plot/graphics settings
15 - `rt::RuntimeData`: Runtime execution data
16 - `modions::ModionsData`: Ions configuration
17 - `variable_names::Vector{String}`: List of validated variable names to process
18 """
19 function process_simulation_list( simdata::SimulationData; cfg::ConfigData, pgp::PGPData,
20 rt::RuntimeData, modions::ModionsData, variable_names::Vector{String} )
21     # == Loop over validated variable names ==
22     for var_name in variable_names
23         println(" Processing variable: ", var_name)
24         # Preprocess each variable
25         process_variable(var_name, simdata; cfg, pgp, rt, modions)
26     end
27 end

```

Este formato permite a geração automática de páginas de ajuda no ambiente Julia (via comando `?`) e é compatível com ferramentas como `Documenter.jl`, assegurando sincronização entre código e documentação.

4.6.2 Repositório GitHub e Estrutura do Projeto

O código-fonte encontra-se disponível publicamente em <https://github.com/tomalima/AstroTLPlot> [LCdA25], garantindo transparência e acesso continuado. A estrutura do repositório inclui:

- **Código-fonte:** organizado modularmente na pasta `src`
- **Exemplos práticos:** notebooks Jupyter na pasta `notebook`
- **Dados de teste:** ficheiros de configuração e dados em `data`
- **Sistema de testes:** suites de teste na pasta `test`
- **Documentação:** gerada automaticamente em `docs`
- **Gestão de dependências:** ficheiros `Project.toml` e `Manifest.toml`

Esta organização facilita a instalação, reprodução de resultados e extensão do pacote, promovendo a adoção pela comunidade científica. A abordagem de desenvolvimento aberto alinha-se com as práticas contemporâneas de ciência computacional [WBC⁺17, Hin19b, Tea24], assegurando sustentabilidade e evolução colaborativa do projecto.

4.7 Avaliação Comparativa: AstroTLPlot vs. AstroPGPLOT

A transposição do software AstroPGPLOT (implementado em FORTRAN 2008 com a biblioteca PGPLOT) para o pacote AstroTLPlot (desenvolvido em Julia com a biblioteca Makie.jl) permitiu uma avaliação comparativa abrangente entre as duas implementações, analisando dimensões críticas como desempenho computacional, arquitetura de software e capacidades de visualização.

Do ponto de vista do desempenho, os resultados demonstraram que o AstroTLPlot, quando adequadamente otimizado, atinge tempos de execução comparáveis aos do AstroPGPLOT, que mantém eficiência através da vetorização automática em compiladores FORTRAN modernos (GCC, Intel, Absoft). Esta paridade de desempenho confirma que a transposição para Julia preserva a eficiência numérica característica da implementação original.

No âmbito da arquitetura de software, o AstroTLPlot adotou um paradigma modular baseado em estruturas de dados dedicadas e parâmetros tipados, enquanto o AstroPGPLOT segue uma abordagem otimizada para simulações numéricas de alto desempenho. Esta diferença arquitetural traduz-se em ganhos de legibilidade e capacidade de evolução do código no ecossistema Julia.

Quanto ao tratamento de erros, o AstroTLPlot implementa um sistema estruturado que desacopla a lógica principal da gestão de falhas, enquanto o AstroPGPLOT baseia-se em verificações manuais através de códigos de retorno, abordagens que refletem os paradigmas das respetivas linguagens.

Em termos de visualização científica, a transposição do PGPLOT para o Makie.jl representou uma evolução significativa. Enquanto o AstroPGPLOT utiliza a biblioteca PGPLOT, uma solução robusta e multiplataforma desenvolvida em 1993 que permite gerar gráficos estáticos de qualidade de publicação, o AstroTLPlot beneficia das capacidades modernas do Makie.jl para criar visualizações interativas, subplots coordenados e integração de estatísticas diretamente nos gráficos.

Esta transição ampliou as possibilidades de análise visual mantendo a integração com os respetivos ecossistemas computacionais. Como sintetizado na Tabela 4.1, o AstroTLPlot introduz novas capacidades em flexibilidade, interatividade e organização de outputs, enquanto mantém a eficiência computacional necessária para aplicações científicas.

Tabela 4.1: Resumo comparativo entre AstroPGPLOT (FORTRAN/PGPLOT) e AstroTLPlot (Julia/Makie.jl).

Característica	AstroPGPLOT (FORTRAN/PGPLOT)	AstroTLPlot (Julia/Makie.jl)
Desempenho	com vetorização automática em compiladores modernos (GCC, Intel, Absoft)	Comparável, com compilação JIT e otimizações específicas do ecossistema Julia
Arquitetura	Otimizada para simulações numéricas de alto desempenho	Modular, com tipos dedicados e separação clara de responsabilidades
Visualização	Estática, com gráficos de qualidade de publicação via PGPLOT	Interativa, com subplots coordenados e integração de estatísticas via Makie.jl
Interatividade	Possível via linha de comandos ou integração em software externo	Nativamente interativa, com integração em Jupyter Notebooks
Organização outputs	Estrutura fixa de ficheiros	Automática, com nomenclatura configurável e princípios FAIR
Ecossistema	Maduro e estável, com compiladores FORTRAN estabelecidos	Em desenvolvimento ativo, com comunidade científica crescente

4.8 Considerações Finais

A reengenharia do AstroTLPlot a partir do AstroPGPLOT (FORTRAN 2008 + PGPLOT) para o ecossistema moderno de Julia e Makie.jl demonstrou ser uma transposição tecnologicamente viável e cientificamente vantajosa. O novo pacote unifica a leitura de dados, análise estatística e visualização num fluxo coerente, superando limitações fundamentais da arquitetura anterior em termos de modularidade, interoperabilidade e aderência a boas práticas de ciência computacional.

As principais conquistas técnicas desta transposição incluem:

- **Desempenho computacional otimizado:** Preservação da eficiência numérica do FORTRAN 2008 – que beneficia da vetorização automática em compiladores modernos – através das operações vetorizadas nativas de Julia sobre matrizes N-dimensionais, com suporte acrescido para processamento *out-of-core* (por blocos) de conjuntos de dados massivos.
- **Visualização científica interativa e de alto nível:** Transição do paradigma gráfico estático e limitado do PGPLOT para as capacidades dinâmicas, ricas e interoperáveis do Makie.jl, permitindo a geração automática de figuras de publicação com composição avançada (*subplots*), anotações estatísticas e exploração direta dos dados.
- **Reprodutibilidade e rastreabilidade integral:** Todo o fluxo de trabalho, desde a análise até à figura final, é documentado e reconstruível a partir de scripts e ficheiros de configuração, com controlo de versão (Git) e ambientes reprodutíveis, formalizando práticas de investigação que eram de difícil implementação na abordagem original.
- **Arquitetura modular e extensível:** Redesenho do código em módulos coesos com APIs bem definidas, facilitando a manutenção, testes unitários e extensão por terceiros, em contraste com a natureza frequentemente monolítica do AstroPGPLOT
- **Organização Automática e Rastreável de Resultados:** Geração automática de uma hierarquia de pastas e ficheiros onde cada figura é guardada com um carimbo temporal preciso e o nome da variável física analisada. Isto elimina a sobreposição acidental de resultados, assegura uma localização intuitiva e fornece um registo implícito da sequência e momento de cada análise.
- **Configuração Transparente e Interoperável em YAML:** Substituição integral do ficheiro binário de configuração (.dat) pelo formato padrão YAML. Esta transposição garante que todos os parâmetros de

análise são estritamente legíveis por humanos e máquinas, permitindo versionamento eficaz, auditoria direta e facilitando a integração com ferramentas modernas de gestão de fluxos de trabalho científicos.

- **Produtividade de Desenvolvimento com Desempenho de Nível Nativo:** A implementação em Julia permite a expressão concisa de operações matemáticas complexas através de uma sintaxe vetorizada e de alto nível (e.g., `vel2 = simdata.vxx.^2 .+ simdata.vyy.^2 .+ simdata.vzz.^2`), que é compilada just-in-time para código máquina otimizado. Isto elimina a complexidade e a propensão a erros dos loops explícitos em FORTRAN, mantendo a eficiência computacional comparável à do código original compilado.
- **Ambiente Interativo e Reprodutível para Análise Exploratória (Jupyter):** Integração nativa com notebooks Jupyter, que unem código executável, visualizações dinâmicas, documentação rica e resultados num único documento persistente. Isto transforma o fluxo de trabalho de um ciclo de compilação-execução estático para um processo exploratório e iterativo, ideal para a descoberta científica e para criar narrativas reprodutíveis que comunicam análise e resultados em simultâneo.

O resultado final confirma a transposição bem-sucedida: o AstroTLPlot em Julia mantém o desempenho da base FORTRAN 2008 e acrescenta-lhe uma infraestrutura moderna para análise e visualização científica.

Esta evolução técnica assegura a sustentabilidade do código, a sua extensibilidade para novas variáveis físicas e a sua conformidade integral com os padrões de reprodutibilidade (FAIR), transformando-o numa ferramenta robusta para a investigação astrofísica atual.

5

Conclusões e Trabalhos Futuros

5.1 Conclusões e Contribuições

O presente trabalho dedicou-se à transposição e modernização do ecossistema de análise astrofísica anteriormente assente no binómio FORTRAN/PGPLOT (AstroPGPLOT), transpondo-o para a linguagem Julia com a biblioteca Makie. O objetivo central foi superar as limitações de interatividade e manutenção do AstroPGPLOT, estabelecendo um fluxo de trabalho contemporâneo para o processamento de simulações de elevada complexidade. A investigação abrangeu desde a gestão de I/O em formatos heterogéneos (HDF4, HDF5, VTK) até à implementação de algoritmos estatísticos e técnicas de visualização científica avançada.

A transição para o novo paradigma permitiu não só replicar a funcionalidade original, mas expandir as capacidades analíticas através de uma arquitetura de componentes encapsulados. De entre os principais resultados e contributos obtidos, destacam-se:

- **A reimplementação de rotinas críticas e transposição tecnológica:** mantendo a precisão e o desempenho das operações originais [BEKS17];
- **Modernização da camada de visualização com Makie.jl:** A substituição da biblioteca PGPLOT permitiu a transição de representações estáticas para gráficos interativos de alta fidelidade e a exportação multiformato para publicação científica sem recorrer comandos externos [DK21];
- **Desenvolvimento do pacote AstroTLPlot:** Baseada numa arquitetura modular e extensível. Segue as normas modernas de desenvolvimento de software científico, incluindo documentação via *docstrings*, guias em Markdown e uma estrutura que favorece a reprodutibilidade [WBC⁺17];

- **Validação e Verificação de Equivalência:** Através de uma validação sistemática, confirmou-se a convergência de resultados entre AstroPGP e o novo pacote AstroTLPlot, assegurando que a transição tecnológica não comprometeu a integridade dos dados físicos analisados;
- **Superação do "Problema das Duas Linguagens":** Demonstrou-se que Julia é uma solução eficaz para a astrofísica computacional, unindo a facilidade de prototipagem à desempenho de execução, o que simplifica drasticamente o ciclo de desenvolvimento e manutenção científica [Per19].

A principal contribuição deste projeto reside na entrega de uma ferramenta capaz de tornar a análise astrofísica mais eficiente, intuitiva e acessível à comunidade investigadora. A consolidação do binómio Julia/Makie provou ser uma escolha estratégica robusta, oferecendo uma plataforma que integra harmoniosamente o elevado desempenho computacional com a flexibilidade necessária para a inovação científica [BEKS17].

Em síntese, os objetivos propostos foram plenamente atingidos, resultando numa infraestrutura de software sólida que servirá de base para futuras integrações com novos modelos de simulação e bibliotecas do ecossistema científico global.

5.2 Limitações e Desafios

Apesar dos avanços alcançados na transição tecnológica, o desenvolvimento do pacote AstroTLPlot evidenciou limitações em dois domínios particularmente críticos para a sua plena adoção e utilidade prática: a interatividade das visualizações e a abrangência da documentação técnica. A análise destes aspetos é fundamental para contextualizar o estado atual da ferramenta e orientar o seu desenvolvimento futuro.

No domínio da interatividade visual, a principal limitação reside na implementação de funcionalidades dinâmicas avançadas. Embora a biblioteca gráfica Makie.jl forneça a base para a criação de visualizações interativas [DK21], a integração de um conjunto abrangente de controlos em tempo real, como a seleção dinâmica de variáveis físicas, o ajuste imediato de escalas (linear, logarítmica, potência) e a manipulação sincronizada de múltiplos cortes espaciais ou temporais, ainda não foi totalmente concretizada. A implementação robusta destas funcionalidades exige não apenas um desenvolvimento adicional da interface, mas também uma otimização cuidada da gestão de eventos e do pipeline de renderização, assegurando a responsividade da ferramenta mesmo quando aplicada a conjuntos de dados de grande dimensão típicos de simulações astrofísicas.

Paralelamente, subsiste o desafio de alargar e sistematizar a documentação e os exemplos de utilização. Para que a ferramenta seja plenamente adotada pela comunidade científica, é necessário desenvolver tutoriais interativos, *notebooks* pedagógicos com casos de estudo representativos e uma referência *API* exaustiva. Esta documentação deve ilustrar a aplicação do pacote a diferentes cenários de investigação, cobrindo desde tarefas básicas de visualização até análises estatísticas avançadas, facilitando assim a sua integração em fluxos de trabalho científicos variados.

As limitações identificadas, centradas na experiência do utilizador e na acessibilidade da informação, não diminuem o valor da transposição efetuada, mas definem os passos necessários para consolidar o AstroTLPlot como uma ferramenta de referência, robusta e eficiente na análise de dados astrofísicos.

5.3 Trabalhos Futuros

O desenvolvimento do pacote AstroTLPlot abre múltiplas perspetivas de evolução, tanto no plano técnico como científico. Entre as principais linhas de evolução destacam-se:

- **Introdução de interatividade avançada:** Expansão das capacidades do Makie.jl [DK21] para permitir a seleção dinâmica de variáveis, o ajuste de escalas, a visualização de cortes múltiplos em tempo real e a exploração interativa de grandes volumes de dados. Esta melhoria aumentará significativamente a utilidade da ferramenta em investigação exploratória e em contexto de ensino;
- **Integração com Inteligência Artificial e Machine Learning:** Tirar partido do ecossistema de aprendizagem automática em Julia (ex: Flux.jl) para implementar a deteção automatizada de

estruturas morfológicas e padrões em simulações de larga escala. Esta integração permitirá evoluir de uma análise puramente descritiva para uma exploração de dados assistida por IA, facilitando a identificação de fenómenos físicos complexos de forma automatizada e escalável;

- **Expansão da documentação e exemplos de utilização:** Criação de tutoriais interativos, *notebooks* explicativos [KRKP⁺16b] e exemplos de casos de estudo adicionais, garantindo que investigadores e estudantes consigam utilizar a ferramenta de forma autónoma e eficaz.

Em suma, os trabalhos futuros visam expandir a interatividade, a robustez e a inteligência analítica do pacote, consolidando o AstroTLPlot como uma plataforma versátil e de vanguarda para a astrofísica computacional. Ao integrar capacidades de análise automatizada e visualização reativa, a ferramenta não só apoiará novas fronteiras de investigação, como também facilitará a difusão do conhecimento científico de forma mais intuitiva. Este roteiro de evolução assegura que o projeto se mantenha alinhado com as tendências globais de computação científica reprodutível, aberta e escalável [WBC⁺17].

Bibliografia

- [B⁺20] P. E. Black et al. Performance analysis of scientific software in julia. *Journal of Computational Science*, 42:101098, 2020.
- [BCL14] Massimo Brescia, Stefano Cavuoti, and Giuseppe Longo. Data mining and knowledge discovery in astronomy. *The Astronomy and Astrophysics Review*, 22(1):1–35, 2014.
- [BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.
- [BKSE12a] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.
- [BKSE12b] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. The julia programming language. <https://julialang.org>, 2012.
- [CS14] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2nd edition, 2014.
- [DK21] Simon Danisch and Julius Krumbiegel. Makie.jl: Flexible high-performance data visualization for julia. *Journal of Open Source Software*, 6(65):3349, 2021.
- [Dra11] Bruce T. Draine. *Physics of the Interstellar and Intergalactic Medium*. Princeton University Press, 2011.
- [EO06] Clark Evans and Jon Orwant. Yaml: The yaml ain't markup language. In *Proceedings of the 2006 Linux Symposium*, pages 235–244, 2006.
- [FHK⁺11] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47, 2011.
- [GDDM13] Matthew Graham, S. G. Djorgovski, A. J. Drake, and A. A. Mahabal. Big data in astronomy. *Astronomy and Computing*, 3:1–10, 2013.
- [Git23] GitHub. Github documentation, 2023.
- [Gol91] David Goldberg. *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, volume 23. 1991.

- [Har22] Harris Geospatial Solutions. *IDL (Interactive Data Language) Reference Guide*, 2022. NV5 Geospatial (formerly Harris Geospatial).
- [HBO10] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *Queue*, 8(5):20–30, 2010.
- [Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.
- [Hin19a] Konrad Hinsén. Dealing with software complexity in computational science. *Computing in Science & Engineering*, 21(2):98–106, 2019.
- [Hin19b] Konrad Hinsén. The scientific software ecosystem. *Computing in Science & Engineering*, 21(2):98–103, 2019.
- [Jul23] Julia Community. *Julia Extension for Visual Studio Code*, 2023. Accessed: 2023.
- [KRKP⁺16a] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks—a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, 2016.
- [KRKP⁺16b] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks—a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, 2016.
- [LaT23] LaTeX Workshop Contributors. *Latex workshop extension for vs code*, 2023.
- [LCdA25] Tomás Lima, Francisco Coelho, and Miguel de Avillez. *Astrotlplot: A julia package for astrophysical data analysis and visualization*, 2025.
- [LP07] Bertram Ludäscher and Beth Plale. *Scientific Data Management: Challenges, Existing Technology, and Deployment*. Computing Research Association, 2007.
- [Mat23] MathWorks. *MATLAB Documentation*, 2023. Accessed: 2023.
- [Mic23a] Microsoft. *Jupyter extension for visual studio code*, 2023.
- [Mic23b] Microsoft. *Visual Studio Code Documentation*, 2023. Accessed: 2023.
- [Mun14] Tamara Munzner. *Visualization Analysis and Design*. AK Peters/CRC Press, Boca Raton, FL, 2014.
- [NEO24] NEON (National Ecological Observatory Network). *About HDF5 Files*, 2024.
- [OF06] Donald E. Osterbrock and Gary J. Ferland. *Astrophysics of Gaseous Nebulae and Active Galactic Nuclei*. University Science Books, 2nd edition, 2006.
- [OR10] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010.
- [Pea01] T. J. Pearson. *PGPLOT Graphics Subroutine Library*, 2001. California Institute of Technology.
- [Per19] Jeffrey M. Perkel. Julia: come for the syntax, stay for the speed. *Nature*, 572:141–142, 2019.

- [PG15] Fernando Pérez and Brian E. Granger. Project jupyter: Computational narratives as the engine of collaborative data science. *Computing in Science & Engineering*, 17(2):7–14, 2015.
- [PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.
- [Pyt23] Python Software Foundation. *Python Documentation*, 2023. Accessed: 2023.
- [RBZ⁺19] Adam Rule, Amanda Birmingham, Cristian Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H. Nguyen, Sara Brin Rosenthal, Fernando Pérez, et al. Ten simple rules for writing and sharing computational analyses in jupyter notebooks. *PLOS Computational Biology*, 15(7):e1007007, 2019.
- [SLP12] Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Implementing reproducible research. *Computational Statistics*, 27(4):1–19, 2012.
- [SML06] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, Inc., 4th edition, 2006.
- [SNTH13] Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLOS Computational Biology*, 9(10):e1003285, 2013.
- [Tea24] The JuliaDocs Team. Documenter.jl — a documentation generator for julia. <https://juliadocs.github.io/Documenter.jl/stable/>, 2024. A static documentation generator that integrates with Julia’s docstring system and supports reproducible examples via literate programming.
- [The22a] The HDF Group. *HDF4 Reference Manual*, 2022. Accessed: 2023.
- [The22b] The HDF Group. *HDF5 Reference Manual*, 2022. Accessed: 2023.
- [The23a] The HDF Group. *HDF5 User’s Guide*, 2023. Accessed: 2023.
- [The23b] The Julia Programming Language. *Julia Documentation*, 2023. Accessed: 2023.
- [The26] The Julia Project. Julia micro-benchmarks. Julia Language Website, 2026.
- [VMTP20] Mark Vogelsberger, Federico Marinacci, Paul Torrey, and Ewald Puchwein. Cosmological simulations of galaxy formation. *Nature Reviews Physics*, 2:42–66, 2020.
- [WAB⁺14] Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *PLOS Biology*, 12(1):e1001745, 2014.
- [War13] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 3rd edition, 2013.
- [WBC⁺17] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6):e1005510, 2017.
- [YAM21] YAML Language Development Team. *YAML Ain’t Markup Language (YAML) Version 1.2.2*, 2021. Accessed: 2023.



UNIVERSIDADE DE ÉVORA
INSTITUTO DE INVESTIGAÇÃO
E FORMAÇÃO AVANÇADA

Contactos:

Universidade de Évora

Instituto de Investigação e Formação Avançada — IIFA

Palácio do Vimioso | Largo Marquês de Marialva, Apart. 94

7002 - 554 Évora | Portugal

Tel: (+351) 266 706 581

Fax: (+351) 266 744 677

email: iifa@uevora.pt