SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA
PIBIC

IMPLEMENTAÇÃO EM LINGUAGEM *PYTHON* E USO DO PERIODOGRAMA Z2n

Astrofísica Estelar Observacional: Óptico, UV e Raios X

Área do conhecimento: Ciências Exatas e da Terra
Subárea do conhecimento: Astrofísica Estelar
Especialidade do conhecimento: Astrofísica de Altas Energias

Discente: Yohan Alexander Dantas de França [1]
Orientador: Prof. Dr. Raimundo Lopes de Oliveira Filho [2]

Relatório Final
Período da bolsa: de 08/2019 a 07/2020

Este projeto é desenvolvido com bolsa de iniciação científica

PIBIC/CNPq

Sumário

1. Graduando em Ciência da Computação pela Universidade Federal de Sergipe (UFS) - São Cristovão - Brasil. *e-mail:* <yohan.franca@dcomp.ufs.br>

2. Professor Associado no Departamento de Física da Universidade Federal de Sergipe (UFS) - São Cristovão - Brasil. *e-mail:* <rlopes@ufs.br>

1 Introdução

Informações podem ser codificadas em sinais. No que diz respeito a astrofísica observacional, a codificação representa informações coletadas por telescópios e registradas em seus detectores. Considerando o caso de observações de fótons, estão embutidas nesses sinais três informações distintas referentes aos fótons registrados: uma característica espacial (as regiões de registro de fótons no detector), uma característica temporal (os momentos nos quais os fótons foram registrados) e uma característica de energia (a faixa de energia na qual estão espalhados os fótons registrados).

Um exemplo clássico de interesse em astrofísica é o estudo de variabilidade temporal da intensidade luminosa de um astro – por exemplo, uma estrela. Nesse tipo de estudo são considerados fótons associados a uma dada região do céu correspondente ao astro sob investigação e que correspondem a uma dada faixa de energia, e investiga-se como a quantidade de fótons registrados variou em função do tempo. Em certas condições, como em observações em raios X, é normalmente possível identificar fóton a fóton. Em outras, como em observações no óptico, os fótons chegam em quantidade que supera a capacidade do instrumento de detectá-los individualmente, e o registro acontece em grupos de fótons. De um modo ou de outro, é possível construir um produto científico chamado curva de luz e que representa a evolução temporal do brilho da fonte em função do tempo. Os registros, quando apresentados na forma de curva de luz, são agrupados em intervalos de tempo discretos. O agrupamento se dá em um compromisso entre o intervalo de tempo ser pequeno suficiente para que seja possível identificar detalhes na evolução do brilho, e grande o suficiente para que os sinais que representa tenham significado estatístico.

A variabilidade temporal de um sinal pode ser periódica ou aperiódica. Quando periódica, existe um período T associado e que representa de quanto em quanto tempo o comportamento investigado se repete. Outra forma de caracterizar esta variabilidade é através da frequência f, que quantifica quantas dessas repetições acontecem em uma dada unidade de tempo, e que se relaciona com o período como $f = \frac{1}{T}$. Se o período é medido em segundos, tem-se a frequência em Hertz (Hz) – equivalente portanto a ciclos por segundo.

A verificação da existência de uma componente periódica em um sinal é comum em análises temporais. Considerando o caso de análise de curvas de luz, há duas possibilidades. A primeira é de análise gráfica de recorrência, que pode fazer uso de ajustes de funções periódicas (por exemplo, uma senoide). A segunda é de aplicação de ferramentas matemáticas sofisticadas para a construção dos ditos *periodogramas*, como as técnicas de Fourier (MARKS, 2009).

Os periodogramas implementam técnicas matemáticas em ferramentas computacionais. Entre eles estão o Lomb-Scargle (LOMB, 1976); (SCARGLE, 1982) e o Z_n^2 (Buccheri et al., 1983). O Lomb-Scargle é amplamente usado em astrofísica e se destinada a análise de curvas de luz. O periodograma Z_n^2 (Buccheri et al., 1983) é uma alternativa para exploração direta dos eventos (fótons), individualmente – o que pode ser feito e é adequado ao caso de observações em raios X de fontes de baixo brilho.

Este projeto teve como fim implementar o periodograma Z_n^2 em linguagem *Python* e oferecê-lo gratuitamente para a comunidade científica através de uma

interface “amigável”. Nessa implementação foram aplicadas técnicas que permitem otimizar o potencial de processamento do computador e técnicas eficientes na manipulação de dados, que juntas minimizam o tempo de obtenção do resultado final.

1.1 Processamento de sinais com taxa de amostragem finita

O registro de fótons por um detector passa uma informação instrumental analógica, no sentido de um conjunto de entrada que a priori é contínuo no espaço de seu domínio, para um conjunto de informações digitais, discretizadas, que são os sinais digitais que representam o que está sendo medido. Desse modo, o sinal é armazenado como um conjunto de amostras discretas, conforme Figura 1.

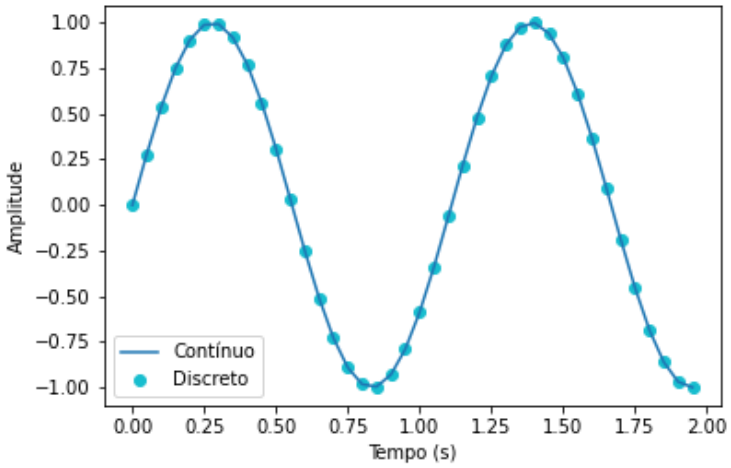


Figura 1: Sinal periódico $y = sin(2\pi t * 0.9)$ amostrado a um passo de 0,01 s.

O processo de discretização da informação original se dá basicamente por dois motivos: porque a coleta de informação é discretizada e porque o registro da informação é discretizado. A amostragem é discretizada e comumente periódica. Três variáveis importante nas observações astrofísicas são (i) a duração da observação (T_{obs}), que é o intervalo de tempo integral durante o qual a observação está sendo realizada e portanto durante quanto tempo um corpo celeste foi observado, (ii) o intervalo de tempo de cadência no acúmulo das informações (T_{exp}) e (iii) o intervalo de tempo necessário para a leitura/registro das informações e preparação do detector para reinício da exposição ($T_{leitura}$). Assim, em uma observação sem interrupção de um alvo astronômico, tem-se que: T_{obs} = $n(T_{exp}$ + $T_{leitura})$, com n sendo representativo da cadência temporal da observação. Para cada exposição (enésima exposição, n_i) do detector, existe um tempo associado à leitura do detector. T_{obs} é definido pelo astrônomo ou grupo de astrônomos que solicitaram a observação, de modo que a cobertura temporal seja suficientemente longa para que o objetivo científico seja alcançado. Porém o tempo de uso de um telescópio é limitado, tem custo, e a aprovação para realização da observação passa pelo crivo de um comitê que avalia, por exemplo, a viabilidade técnica da observação, a relevância científica do projeto proposto e a experiência dos astrônomos que solicitaram a observação. T_{exp} por vezes pode

ser definido pelo astrônomo. Uma vez tendo assumido um valor, e a observação tendo sido realizada, os fótons que foram coletados durante um intervalo de tempo T_{exp} serão contabilizados em conjunto que não poderá mais ser decomposto em subconjuntos. $T_{leitura}$ por vezes pode ser definido pelo usuário entre opções (limitadas) oferecidas para o uso de um certo detector. A definição de T_{exp} e $T_{leitura}$, quando possíveis, acontecem em um compromisso entre as exposições serem curtas para maximizar o potencial de detalhamento da variação temporal em estudo, e longas o bastante para que a quantidade de fótons observados/registrados tenha significado estatístico – notadamente, frente a contribuições que são associadas a ruídos do detector, mas também de maximizar a contribuição por fótons do corpo celeste em estudo frente a contribuições por fótons que não são da fonte em estudo (essas contribuições em conjunto são usualmente chamadas de "contribuições de background"). Juntos, T_{exp} e $T_{leitura}$ determinam a resolução temporal associada a uma dada observação astronômica.

Em análises temporais é comum agrupar exposições sucessivas, que daqui em diante será denominado por "binagem", com os novos elementos sendo chamados de "bins". A binagem degrada a resolução temporal e, equivalente às exposições, as informações que são agrupadas não podem ser mais desagrupadas. O objetivo maior da binagem é o de acentuar a informação em relação ao ruído, dando significância estatística, ou mesmo evidenciar variações que ocorrem em um intervalo de tempo substancialmente maior do que o de resolução instrumental.

Suponha, por exemplo, que uma observação foi realizada em 10 exposições sucessivas com Texp de 10 segundos, e $T_{leitura}$ de 5 segundos. Assim, $n = 10$, e T_{obs} = 10$\times$(10 + 5) s = 150 s. As informações foram agrupadas a cada Texp + $T_{leitura}$ = 15 s, que é o bin original e que está associado à resolução temporal da observação. O astrônomo pode, por exemplo, agrupar os bins de dois em dois: o primeiro e o segundo, o terceiro e o quarto, e assim sucessivamente. O novo bin passou a ser de 2$\times$15 s = 30 s e, a partir dele, não é possível reconstruir a binagem original.

É possível estimar o sinal contínuo por interpolação de sinais discretos. Porém, caso a taxa de amostragem seja baixa, a codificação do sinal discreto não irá representar de forma realista o sinal contínuo. Uma pergunta comumente respondida antes de iniciada uma análise temporal é: qual a taxa mínima de amostragem necessária para determinar como a variação real do comportamento no sinal contínuo acontece? Tome como exemplo um sinal contínuo, conforme Figura 2, que representa uma informação com frequência de 0,9 Hz registrada ao longo de uma observação de 20 s.

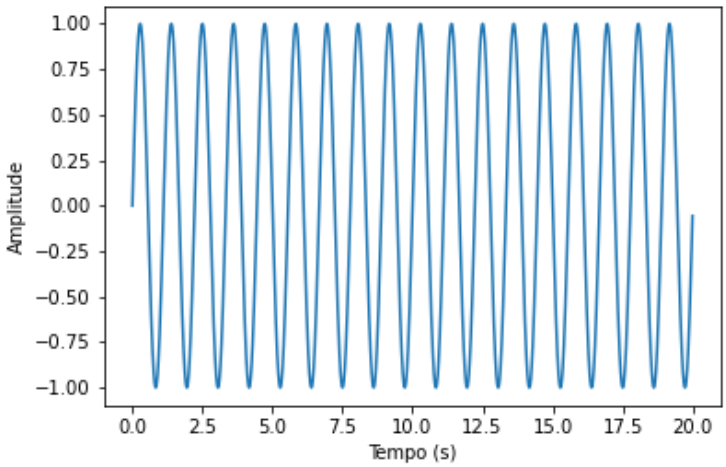


Figura 2: Sinal periódico $y = sin(2\pi t * 0.9)$ observado ao longo de 20 s.

Quanto maior a taxa de amostragem, melhor e mais realística será a descrição da evolução temporal do sinal contínuo. A Figura 2 exemplifica uma informação de frequência lenta (<20 Hz) a uma taxa de amostragem (100 Hz) maior que a frequência da recorrência no seu comportamento. Mas o que acontece se a amostragem ocorrer em uma taxa semelhante ou menor que a frequência da informação? Nesse caso, podem surgir também artefatos em frequência que representam falsos positivos, o fenômeno conhecido como *aliasing*.

A analogia "um relógio quebrado está certo duas vezes por dia"descreve o comportamento do fenômeno. Por estar sem funcionar, a leitura no relógio em intervalos de 12 horas sempre estará correta ou adiantada/atrasada por um valor fixo. Não existe o conhecimento do comportamento do relógio entre as leituras. A impressão incorreta da hora é um exemplo do fenômeno de *alias*, neste caso ocorrido devido à uma baixa taxa de amostragem. Para o relógio quebrado, amostrar uma vez por dia ou duas vezes por dia não dá indicação de que há um problema na contagem do tempo (exceto, talvez, uma hora atrasada/adiantada por um valor fixo). Contudo, a amostragem em qualquer taxa maior que a mínima para estimar a variação real da observação de forma realista revelará que a hora no relógio não muda em função do tempo da leitura.

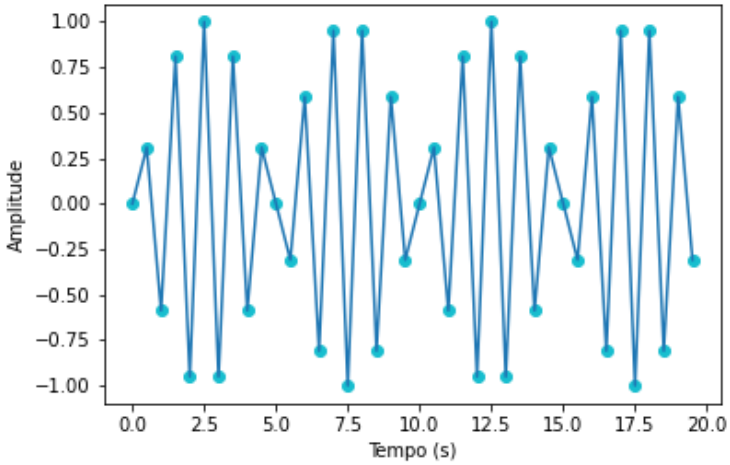


Figura 3: Sinal periódico $y = sin(2\pi t * 0.9)$ amostrado a um passo de 0,5 s.

Neste sentido, a Figura 3 representa o mesmo sinal subamostrado a uma taxa de 2 Hz. A representação estimada do sinal contínuo não é realista porque a taxa de amostragem está muito abaixo da recorrência da informação. Além disso, é possível observar um comportamento adicional e inesperado, um *alias*. Para exemplificar o surgimento deste artefato em frequência tomemos como exemplo a Figura 4, que representa a amostragem de uma informação com recorrência de 0,9 Hz a uma taxa de 1 Hz.

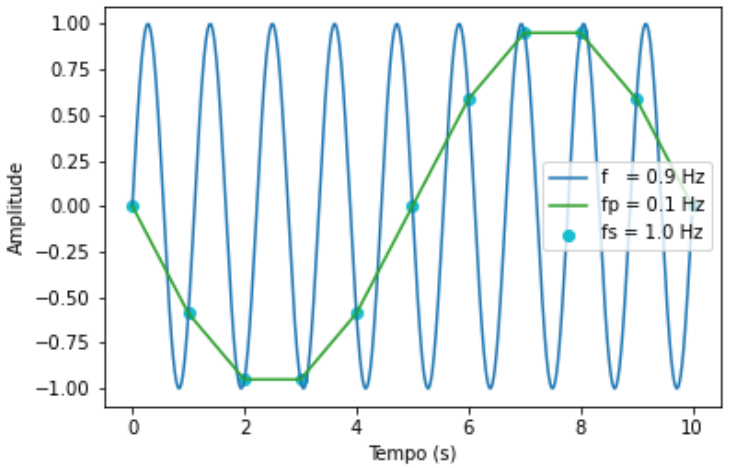


Figura 4: Sinal periódico $y = sin(2\pi t * 0.9)$ amostrado a um passo de 1 s.

Não é coincidência que a diferença entre a informação (0,9 Hz) e a amostragem (1 Hz) seja de exatamente 0,1 Hz. Foi criado um artefato em frequência de 0,1 Hz que representa um falso positivo (*alias*). Então o sinal estimado aparenta representar um sinal contínuo com comportamento recorrente da frequência do falso positivo (0,1 Hz). Para calcular a frequência percebida fp de qualquer frequência de sinal f, que é amostrada em qualquer taxa de amostragem f_{obs}, temos a Equação 2.

$$f = \frac{1}{T} \tag{1}$$

$$fp = \left| f - f_{obs} \cdot \left\lceil \frac{f}{f_{obs}} \right\rceil \right| \tag{2}$$

Então qual é a taxa mínima de amostragem para estimar a observação de forma realista? Tendo T_{obs} como o tempo de observação, e o recíproco f_{obs} como a taxa de amostragem, o teorema da amostragem de Shannon (1949) estabelece que a taxa mínima de amostragem f_n para a representação realista de uma informação deve ser o dobro da taxa de amostragem f_{obs}, conforme Equação 3.

$$f_n = 2 \cdot f_{obs} \tag{3}$$

$$T_{obs} = \frac{1}{f_{obs}} \tag{4}$$

1.2 Elementos matemáticos envolvidos na Transformada de Fourier

Observações de um comportamento periódico são melhor caracterizadas por uma análise de recorrência. A determinação desta possível recorrência geralmente é alcançada pela aplicação de um método matemático que converte os dados do domínio em tempo para o domínio em frequência. Dentre esses métodos está a Transformada Discreta de Fourier (MARKS, 2009), que é essencialmente uma transformação linear associada ao mapeamento de um conjunto de valores no domínio em tempo $x_n := x_0, x_1, x_2, ..., x_{N-1}$, para um conjunto de valores no domínio em frequência $X_k := x_0, x_1, x_2, ..., x_{N-1}$, conforme Figura 5.

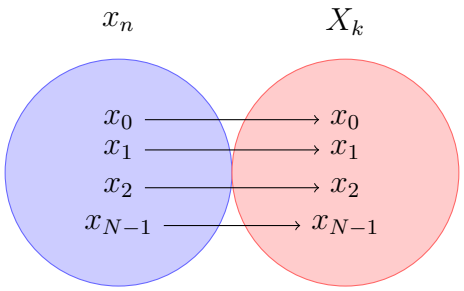


Figura 5: Diagrama da Transformada Discreta de Fourier.

A média estatística de um determinado sinal (incluindo ruído), analisada em termos de seu conteúdo de frequência, é chamada espectro. Com isso, o espectro de potência de uma observação descreve a decomposição em componentes de frequência desse sinal. No espectro de potências é possível visualizar as componentes em frequência que representam a informação em análise, destacando assim os efeitos por trás de um comportamento supostamente recorrente. Como exemplo, o espectro de potências da Figura 6 indica claramente a presença de duas componentes em frequência, 4 Hz e 7 Hz.

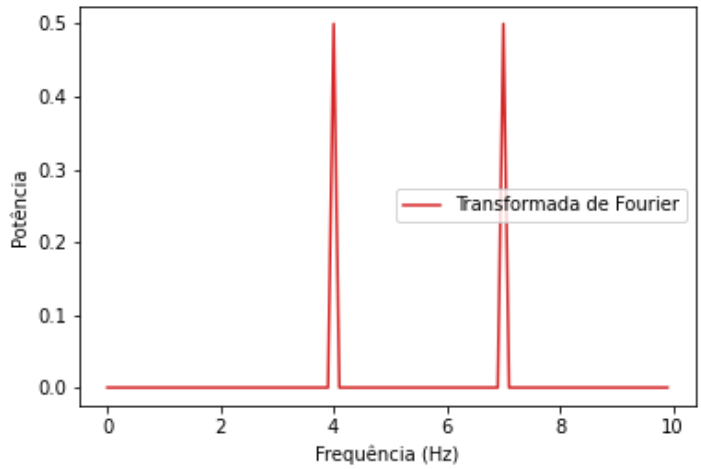


Figura 6: Espectro de potências do sinal periódico $y = sin(2\pi t * 4) + sin(2\pi t * 7)$.

Quando expressos em termos de Séries de Fourier, funções ou sinais são representados como uma combinação linear de senos e cossenos, conforme Equação 5. Cada elemento da série representa uma possível componente em frequência cujo peso é mensurável pelo seu coeficiente linear. Em conjunto, é possível determinar uma quantidade matemática que representa, tendo como base tal série: uma potência associada a dada frequência utilizada na série de Fourier, tendo ao final um espectro de potência associado a frequências. Um exemplo da representação de um sinal periódico por Séries de Fourier é apresentado nas Figuras 7 e 8.

$$f(x) = a_0 + \sum_{n=1}^{\infty} a_n \cdot cos(nx\pi/L) + \sum_{n=1}^{\infty} b_n \cdot sin(nx\pi/L)) \tag{5}$$

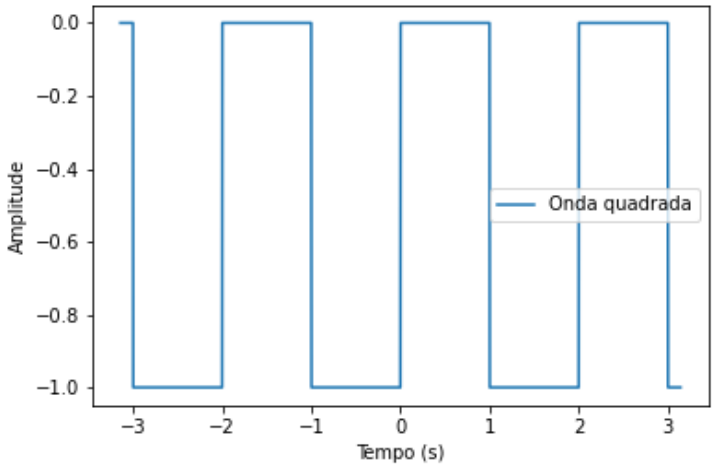


Figura 7: Sinal periódico $y = \lfloor sin(2\pi t * 4) \rfloor$ no domínio em tempo.

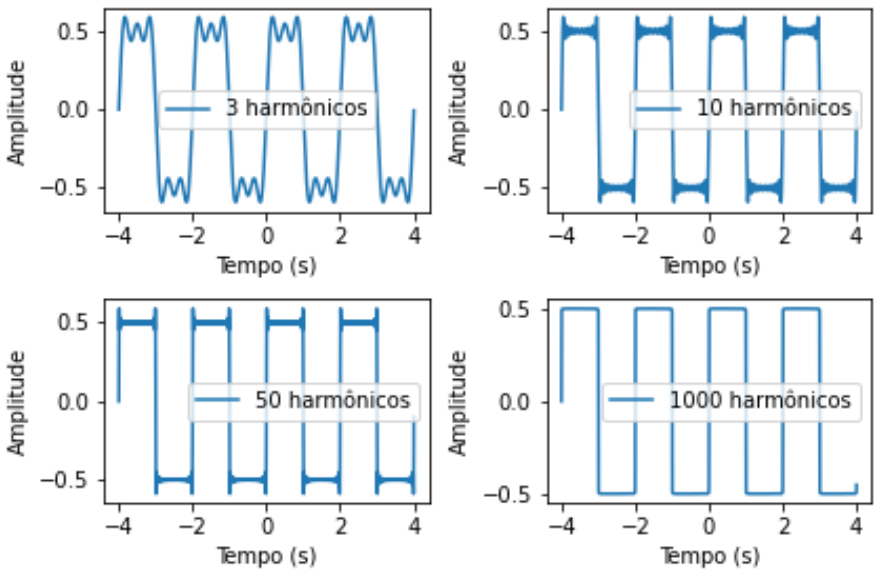


Figura 8: Série de Fourier para uma onda quadrada.

A Transformada Discreta de Fourier, conforme Equação 6, passa o sinal da perspectiva em tempo para a perspectiva em frequências. Então a Transformada descreve a decomposição em componentes de frequência do sinal (espectro de potência), e cada componente pode ser representada por uma Série de Fourier. Para exemplificar o comportamento da Transformada tomemos como exemplo prático um sinal A com 7 amostras, conforme Tabela 1.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot [cos(2\pi kn/N) - i \cdot sin(2\pi kn/N)] \tag{6}$$

$A =$	-1	1	-1	1	-1	1	-1

Tabela 1: Sinal amostrado de uma senoide A.

A intensidade do sinal varia entre 1 e -1 para cada frequência e ao somar todas as amostras obtém-se -1. Os fatores positivos anulam os fatores negativos ao longo do caminho, e caso fosse um sinal mais extenso, continuariam a fazê-lo. Independentemente da extensão do sinal, a soma de todas as amostras nunca seria igual a nada maior que 1 ou menor que -1. Seja o caso de multiplicação, amostra a amostra, por um outro sinal B de frequência diferente. Conforme Tabela 2, a soma de AB agora é 0. Os valores positivos anulam os valores negativos mesmo após a multiplicação.

$A =$	-1	1	-1	1	-1	1	-1
$B =$	-1	0	1	0	-1	0	1
$A * B =$	1	0	-1	0	1	0	-1

Tabela 2: Sinal amostrado da multiplicação das senoides $A * B$.

$A =$	-1	1	-1	1	-1	1	-1
$C =$	-1	1	-1	1	-1	1	-1
$A * C =$	1	1	1	1	1	1	1

Tabela 3: Sinal amostrado da multiplicação das senoides $A * C$.

Mas o que acontece no caso de uma multiplicação de A por um sinal com a mesma frequência C? Neste caso a soma de $AC = 7$ porque um negativo vezes um negativo é positivo. A amplitude desse número nos diz a quantidade de C em A, vezes o número total de amostras. Caso o valor obtido fosse -7, significaria que A é exatamente o oposto de C. Desta forma, ao aplicar a Transformada Discreta de Fourier sobre um sinal amostrado (em azul na Figura 9), as componentes em frequência do sinal se sobressaem no espectro de potências gerado pela Transformada (em vermelho na Figura 9).

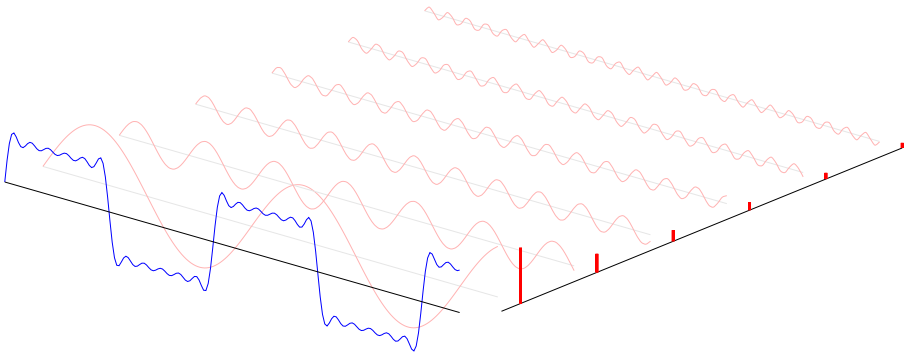


Figura 9: Diagrama em perspectiva dos domínios de um sinal.

1.3 Aplicação de técnicas de Fourier a periodogramas

Z_n^2 é um periodograma baseado no método Rayleigh, que aplica o tempo de chegada dos fótons detectados em uma observação feita por um telescópio a uma análise de Fourier (Buccheri et al., 1983). Essa característica lhe dá a vantagem de explorar a observação em seu limite de resolução temporal, tornando-o útil na detecção de sinais periódicos em altas frequências. Esse pode ser o caso, por exemplo, de rotação de objetos compactos como anãs brancas e estrelas de nêutrons. Ao contrário do Z_n^2, o método clássico do Lomb-Scargle é aplicado sobre curvas de luz, que por construção partem do processo de binagem das exposições (ou de binagem original com tempo de exposição longo). Uma área da astrofísica que tem proveito com o uso do Z_n^2 é a astrofísica de raios X, na qual é comum identificar individualmente fóton a fóton – dada a baixa taxa em que chegam ao detector (fótons ópticos chegam a um fator de pelo menos $\sim 10^3$ por unidade de tempo quando comparados ao que é classicamente observado em raios X).

A aplicação do método Z_n^2 é iniciada reduzindo o tempo de chegada de um dado fóton a um valor de fase ϕ_j, no intervalo de 0 a 1, conforme Equação 8:

$$frac(x) = x - \lfloor x \rfloor \tag{7}$$

$$\phi_j[0,1] = frac(v_i \Delta t_{ij} + \dot{v}_i \frac{\Delta t_{ij}^2}{2} + \dot{v}_i \frac{\Delta t_{ij}^3}{6}) \tag{8}$$

onde Δt_{ij} é o tempo do evento em análise t_j menos o tempo do primeiro evento detectado T_0.

O espectro de potência Z_n^2 (resultado do programa implementado) é dado pela Equação 9:

$$Z_n^2 = \frac{2}{N} \cdot \sum_{k=1}^{n} [(\sum_{j=1}^{N} cos(k\phi_j))^2 + (\sum_{j=1}^{N} sin(k\phi_j))^2] \tag{9}$$

2 Objetivos

- **Acadêmico:** Introduzir o aluno à Ciência, com exploração de conteúdos matemáticos, físicos e de ciências da computação.
- **Técnico-Científico:** Implementar o periodograma Z_n^2 (Buccheri et al., 1983) em linguagem Python e oferecê-lo para uso da comunidade científica.

3 Metodologia

Este trabalho pode ser dividido em duas vertentes: teórica, com foco na compreensão dos elementos matemáticos envolvidos na técnica da Transformada de Fourier, e prática, de implementação do periodograma na linguagem de programação *Python*.

A etapa teórica explorou uma literatura que é vasta e bem estabelecida, e demonstrações interativas. Apesar de seu significado estar oculto em equações matemáticas densas, a Transformada de Fourier está intrinsecamente ligada a vários conceitos de aplicação cotidiana.

Na etapa prática foram usadas técnicas de engenharia de *software* no processo de desenvolvimento, como o auxílio de sistemas de versionamento de código (GIT / GITHUB). O sistema de controle de versão registra mudanças feitas em um grupo de arquivos e tem como função principal gerenciar o desenvolvimento do programa e gerar diferentes versões do mesmo. Com o versionamento foi construído um histórico de todas as modificações do projeto, o que tornou possível rastreá-las, criar novas versões e recuperar versões antigas. Essa técnica foi associada a escrita de testes automatizados que garantem a integridade do produto final.

3.1 Implementação de um periodograma em linguagem *Python*

A escolha da linguagem de programação *Python* (OLIPHANT, 2007) para a implementação de um periodograma se deve ao fato da mesma ser amplamente utilizada pela comunidade científica. O ecossistema de código aberto da linguagem cria praticidade pelo seu caráter de fácil usabilidade e instalação de pacotes.

Além disso, existe um suporte técnico tanto em documentação como em fóruns apoiados por membros da comunidade científica. A exemplo disto, podemos citar o projeto Astropy (PRICE-WHELAN et al., 2018). Trata-se de um esforço da comunidade para desenvolver um pacote único e de domínio público com o *Python*, com algumas ferramentas que são comuns na execução de ciência em astronomia e astrofísica. O pacote também promove a interoperabilidade entre outros pacotes de astronomia que foram escritos a partir da linguagem. Assim, a escolha do Python permite que este trabalho siga a filosofia de "ciência aberta", com o benefício de ser validado, reproduzido e aprimorado.

É importante ressaltar que os periodogramas já são implementados em diversas linguagens de programação. As mesmas técnicas de Fourier utilizadas na criação de espectros de potência dos periodogramas são a base de diversos algoritmos modernos. No contexto astronômico, uma implementação do periodograma Lomb-Scargle (VANDERPLAS, 2018) está disponível na biblioteca Astropy. Está disponível também, no pacote de computação científica NumPy (WALT; COLBERT; VAROQUAUX, 2011), uma versão mais otimizada da Transformada Discreta de Fourier (Transforma Rápida de Stone (1966)), que utiliza o método de "diagramas borboleta" para diminuir o custo computacional.

Nas buscas pela literatura técnica foi identificada apenas uma oferta de aplicativo do método Z_n^2, que foi o *software* aberto Stingray. Porém, diferentemente da versão implementada neste trabalho, os autores Huppenkothen et al. (2019) optaram por utilizar uma versão adaptada com binagem do perfil de pulso usando as fases do perfil dobrado como indica a Equação 10. Essa implementação está em contraste com a principal vantagem do método Z_n^2, que é a aplicação direta aos eventos (uma lista temporal de registro de fótons).

$$Z_n^2 = \frac{2}{\sum_j w_j} \cdot \sum_{k=1}^{n} [(\sum_{j=1}^{N} w_j cos(k\phi_j))^2 + (\sum_{j=1}^{N} w_j sin(k\phi_j))^2] \tag{10}$$

O método Z_n^2 clássico (Buccheri et al., 1983), conforme Equações 8 e 9, é potencialmente muito custoso computacionalmente. Isso porque há duas variáveis relacionadas a *loop*: **k** que representa a faixa de frequências analisadas, e **j** que são os índices dos eventos. O *loop* da variável **j** é interno ao *loop* da variável **k**. Então para cada novo **k**, tem-se a gama completa de **j**, e por isso o periodograma pode se tornar muito lento.

Expressando o método Z_n^2 na forma de pseudocódigo, conforme Algoritmo 1, é possível estimar que a complexidade do algoritmo é da classe exponencial $O(n^2)$ (HOPCROFT; MOTWANI; ULLMAN, 2003). Dada uma entrada de tamanho n, o tempo de execução dos algoritmos exponenciais tem um limite superior dado pela função $O(n^2)$, pois o número de passos necessários para a sua conclusão cresce de forma exponencial com o aumento de n.

Algorithm 1 Periodograma Z_n^2

function PERIODOGRAMA($T[N]$, $F[M]$)
    Input: $T[N]$: *array* com os tempos de chegada dos fótons
    Input: $F[M]$: *array* com a faixa de frequências analisadas
    for i = 0 até $M - 1$ **do**
        - calcula a potência Z_n^2 para cada frequência
        for j = 0 até $N - 1$ **do**
            - calcula a fase para cada tempo
            ϕ_j = $T[j] \cdot F[i]$
            ϕ_j = ϕ_j - $\lfloor \phi_j \rfloor$
            $senos[j]$ = SIN($2\pi \cdot \phi_j$)
            $cossenos[j]$ = COS($2\pi \cdot \phi_j$)
        end for
        sen = SUM($senos$)
        cos = SUM($cossenos$)
        $Z_n^2[j]$ = $sen^2 + cos^2$
    end for
    $Z_n^2[j]$ = $Z_n^2 \cdot (2/\text{LEN}(T))$
    return Z_n^2
end function

O limite superior para o caso do periodograma Z_n^2 é dado pela função $O(n \times m)$. A quantidade de fótons n é diretamente proporcional ao custo computacional: quanto maior o número total de fótons n, maior será o tempo de execução do algoritmo. Por outro lado, o tamanho da entrada m diz respeito a faixa de frequências investigada, associada ao número de frequências analisadas em tal faixa. Então quanto maior for o número total de frequências m, maior será o tempo de execução. Um fator importante na definição desta faixa m é o passo em frequência δ_ν, que ao contrário da faixa, é inversamente proporcional ao custo computacional. A faixa de frequências m representada é de característica contínua, e o passo em frequência δ_ν representa a taxa de amostragem para a discretização desta faixa. Então quanto menor o passo em frequência δ_ν, maior o tempo de execução.

Em contraste com a implementação final deste trabalho, uma implementação simplista em linguagem *Python* consiste em aplicar o Algoritmo 1 para um arquivo de eventos n, em uma dada faixa de frequências m onde existe suspeita de periodicidade. Vale ressaltar que fica a critério do usuário analisar a significância estatística das possíveis componentes em frequência detectadas no periodograma, e também se a origem da componente é de caráter astrofísico da fonte ou de caráter instrumental.

```
1  import␣numpy␣as␣np
2  def␣periodograma(tempos,␣frequencias):
3  ␣␣␣␣for␣freq␣in␣range(len(frequencias)):
4  ␣␣␣␣␣␣␣␣for␣temp␣in␣range(len(tempos)):
5  ␣␣␣␣␣␣␣␣␣␣␣␣fase␣=␣tempos[temp]␣*␣frequencias[freq]
6  ␣␣␣␣␣␣␣␣␣␣␣␣fase␣=␣fase␣-␣np.floor(fase)
7  ␣␣␣␣␣␣␣␣␣␣␣␣fase␣=␣fase␣*␣2␣*␣np.pi
```

```
            senos[temp] = np.sin(fase)
            cossenos[temp] = np.cos(fase)
        z2n[freq] = (sin.sum() ** 2) + (cos.sum() ** 2)
        z2n = z2n * (2 / len(tempos))
    return z2n
```

Algumas características específicas desta linguagem de programação tornam elevado o custo computacional do *Python*. Por isso, foram exploradas também técnicas de paralelismo para maximizar o potencial de processamento da máquina em que o código está sendo executado, visando minimizar o tempo de processamento.

3.2 Aprimoramento da performance explorando técnicas de paralelismo

Apesar de todas as vantagens do *Python* já mencionadas, a linguagem vem com uma desvantagem que afeta a performance do algoritmo. *Python* é uma linguagem de alto nível e como tal existem várias camadas de abstração entre o código fonte e a execução do código de máquina (ABELSON; SUSSMAN; SUSSMAN, 1996). A abstração mais custosa é o fato da linguagem ser interpretada e dinamicamente tipada, com valores armazenados não em *buffers* densos, mas em objetos dispersos. Porém existem soluções modernas para contornar esse problema (que são exploradas neste trabalho), como a vetorização de operações, o paralelismo de núcleos da CPU (central de processamento) e a compilação do código fonte para código de máquina.

Mas por que estas camadas de abstração da linguagem são tão custosas? O modelo das linguagens de programação funciona em camadas de abstração que vão do baixo ao alto nível (PFENNING, 2004) (ver Figura 10). É possível afirmar que quanto mais próximo ao baixo nível e portanto da linguagem de máquina, mais performance terá um código. E por consequência, quanto mais distante do baixo nível, menos performance tem o código executado, pois existem diversos processos (abstrações) associados a tradução (compilação) do código em alto nível que é mais próximo a linguagem humana para o código de máquina.

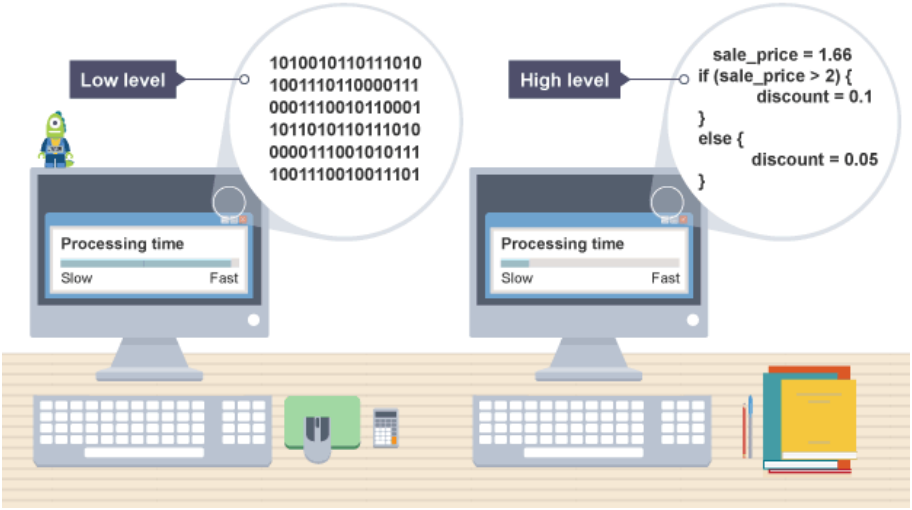


Figura 10: Digrama dos níveis de abstração das linguagens de programação.[1]

Falando mais especificamente sobre as abstrações (COLBURN; SHUTE, 2007) que tornam o *Python* relativamente ineficiente, ser dinamicamente tipada significa que o interpretador não conhece o tipo das variáveis definidas no momento em que o programa é executado. A diferença entre uma variável em linguagens compiladas (baixo nível), como C ou Fortran, e uma variável em *Python* (alto nível), é resumida na Figura 11.

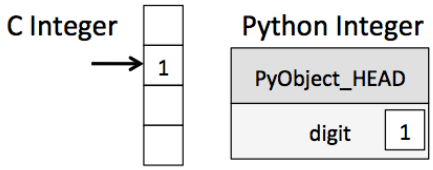


Figura 11: Diagrama das tipagens das variáveis em C e *Python*.[2]

O compilador em linguagem C conhece o tipo de uma variável desde a sua definição. Por outro lado, o interpretador em *Python* conhecerá essa informação após uma checagem que é feita no momento em que o programa for executado. Estas checagens não existem em uma linguagem compilada e, além disso, um compilador moderno pode olhar a frente na execução do código e otimizar operações repetidas ou desnecessárias, podendo resultar em um menor tempo de execução.

Dessa forma, o modelo de objetos em *Python* resulta em um acesso ineficiente da memória. A Figura 11 apresenta a camada extra de abstração ao passar de um inteiro em C para um inteiro em *Python*. O tipo inteiro em *Python* encapsula o tipo inteiro em C, e ao escalar o problema para uma operação em lote com muitos inteiros, a complexidade desta abstração resulta no acesso ineficiente da memória. Em *Python* é possível usar um objeto lista, enquanto em C é usado algum tipo de matriz baseada em *buffers*.

A solução utilizada neste trabalho é uma matriz NumPy (WALT; COLBERT; VAROQUAUX, 2011), que em sua forma mais simples é um objeto *Python* construído em torno de uma matriz em C. Assim, é atribuído um ponteiro (endereço de memória) para um *buffer* de dados contíguo de valores na memória. Uma lista em *Python*, por outro lado, possui um ponteiro (endereço de memória) para um *buffer* contíguo de ponteiros, cada um dos quais aponta para um objeto *Python* que, por sua vez, tem referências aos seus dados (neste caso, números inteiros).

[1] **Fonte:** Disponível em: <https://medium.com/@brettschules/high-level-and-low-level-languages-62776d0b89f0>

[2] **Fonte:** Disponível em: <https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/>

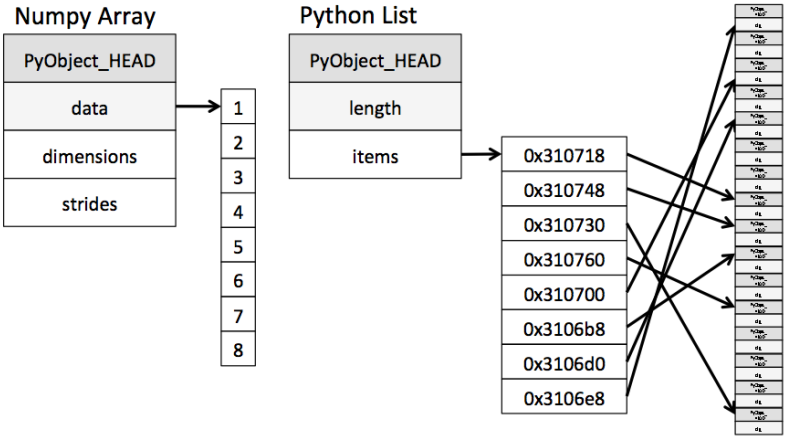


Figura 12: Digrama dos acessos a memória em NumPy e *Python*.[3]

É evidente pela Figura 12 que as abstrações do modelo *Python* adicionam uma complexidade excessiva de processamento. Então se o programa precisar executar alguma operação que percorra os dados em sequência, o modelo NumPy (WALT; COLBERT; VAROQUAUX, 2011) será muito mais eficiente. Além disso, essa estrutura de dados introduz a possibilidade de utilizar operações vetoriais. A técnica de vetorização é o processo de converter um algoritmo que opera em um único valor por vez, para operar em um conjunto de valores ao mesmo tempo, conforme Figura 13.

Os computadores modernos são equipados com processadores que permitem computação paralela rápida em vários níveis: (i) operações vetoriais ou de matriz (SIMD), que permitem executar operações semelhantes simultaneamente em vários dados, e (ii) computação paralela, que permite distribuir blocos de dados em vários núcleos da CPU e processá-los em paralelo. É importante explorar esses recursos quando se trabalha com grandes quantidades de dados, pois isso pode reduzir drasticamente o tempo de computação.

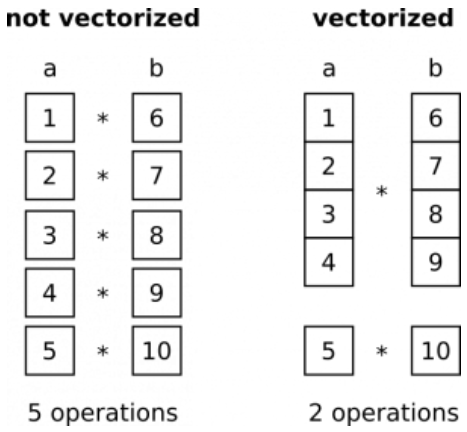


Figura 13: Exemplos de operações não vetorizada (à esquerda) e vetorizada (à direita).

É possível associar o processo de compilação do código fonte com a técnica de vetorização com o fim de diminuir (ainda mais) as abstrações, aproximando o código executado da linguagem de máquina. O Numba é um compilador JIT

[3]**Fonte:** Disponível em: <https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/>

(*just in time*) que implementa um sistema em que todo o código otimizado deve ser produzido na hora da execução. O Numba funciona em um subconjunto de código *Python* e NumPy, convertendo o código fonte em código de máquina rápido e eficiente em memória.

O Numba foi projetado para ser usado com matrizes e funções NumPy, e dessa forma ele otimiza o desempenho dos algoritmos por gerar código especializado para diferentes tipos de dados de matriz. O processo de compilação em código de máquina otimizado ocorre em tempo de execução usando a biblioteca de compiladores LLVM, e os algoritmos numéricos compilados com Numba podem se aproximar das velocidades de processamento das linguagens compiladas como C ou Fortran.

Além disso, o Numba pode executar automaticamente expressões de matriz NumPy em vários núcleos da CPU, facilitando a execução de *loops* paralelos. Sua aplicação resulta em alto desempenho sem a necessidade de escrever uma linha de código de máquina (*Assembly* x86). O código fonte permanece puro *Python* enquanto que o Numba lida com a compilação em tempo de execução para o código de máquina otimizado.

Mas o que seria a execução paralela em núcleos da CPU? As CPUs com múltiplos núcleos tornaram-se o padrão em arquiteturas modernas de computadores. Um núcleo é uma central completa para o processamento de instruções, e ter múltiplos núcleos em um processador cria a possibilidade de dividir uma tarefa em várias *threads* de processamento.

A linguagem de programação *Python* tem a desvantagem do interpretador padrão executar apenas uma instrução de cada vez (o chamado processamento serial) para evitar conflitos entre as *threads*. O interpretador faz isso por meio de um mecanismo de segurança global para *threads*, o chamado GIL (*Global Interpreter Lock*).

Duas abordagens são aplicadas na programação paralela: (i) executação de código via *threads* ou (ii) execução em vários processos. Tarefas enviadas para diferentes *threads* podem ser tratadas como subtarefas de um único processo, e essas *threads* normalmente terão acesso às mesmas áreas de memória (memória compartilhada). Essa abordagem pode facilmente levar a conflitos em caso de sincronização inadequada, por exemplo, se os processos tentarem gravar informação no mesmo local da memória ao mesmo tempo.

Por outro lado, uma abordagem mais segura é enviar vários processos para locais de memória completamente separados (memória distribuída), onde processos serão executados de modo completamente independente. Embora venha com uma sobrecarga adicional devido à comunicação entre processos separados, essa é a abordagem usada neste trabalho. A barreira da GIL foi transposta com o uso do processo de compilação do Numba, já que o código de máquina compilado retira o impedimento das camadas de abstração impostas pelo interpretador do *Python*. Com o código compilado é possível enviar vários processos, que podem ser executados independentemente um do outro. O ganho obtido com isso é a maximização de uso dos núcleos da CPU, evitando ociosidade e diminuindo o tempo de processamento.

A Figura 14 compara a execução de um processo com quatro ramos de computação em paralelo, e do mesmo processo executado de forma serial. A adoção do modelo paralelo pode resultar em diferenças drásticas no tempo de proces-

samento de computações com uma grande escala de dados.

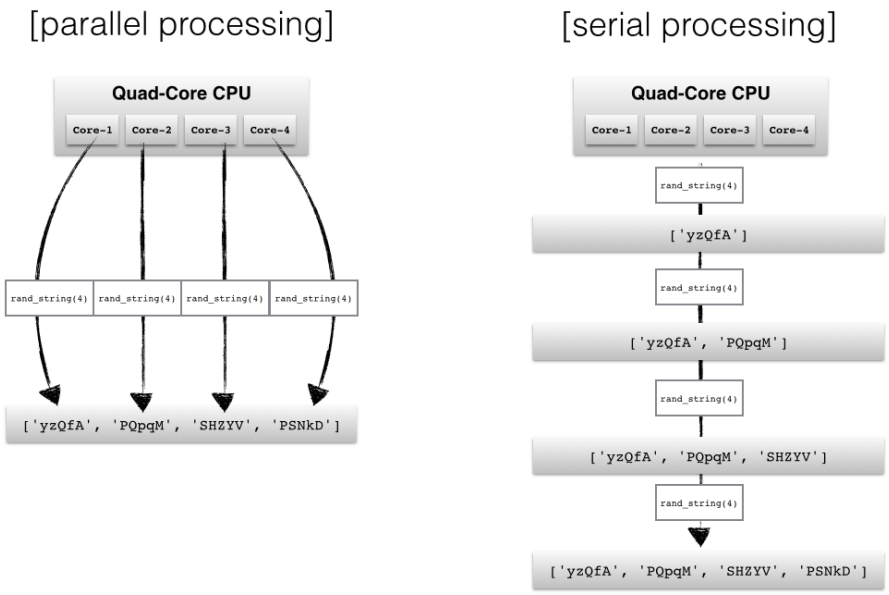


Figura 14: Diagramas ilustrativos de processos em paralelo e serial.[4]

Tendo em mente todos estes processos de otimização, o Algoritmo 1 do periodograma Z_n^2 pode ser representado pelo grafo da Figura 15. É evidente que existem dois ramos de computação que podem ser executados em paralelo. Um ramo que representa os fatores dos senos, e outro que representa os fatores dos cossenos.

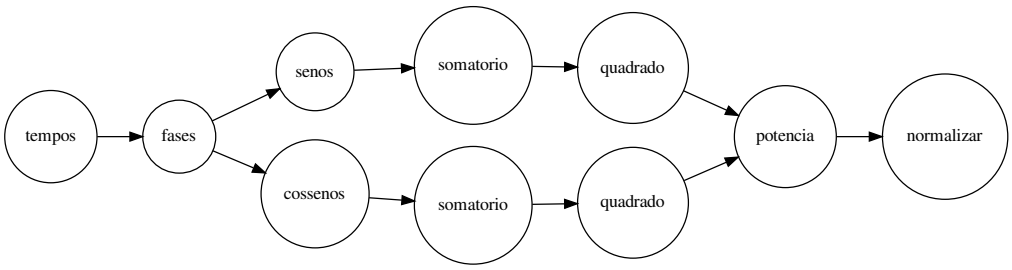


Figura 15: Grafo da execução do periodograma Z_n^2.

Além disso, os nós destacados na Figura 16 representam o conjunto inteiro de eventos analisados. Desta forma, esses valores podem ser abstraídos em uma matriz Numpy. Conforme Figura 17, essa estrutura de dados cria a possibilidade da vetorização em operações numéricas, evitando o uso desnecessário de *loops* e otimizando assim a velocidade do algoritmo e o acesso a memória.

[4] **Fonte:** Disponível em: <https://sebastianraschka.com/Articles/2014_multiprocessing.html>

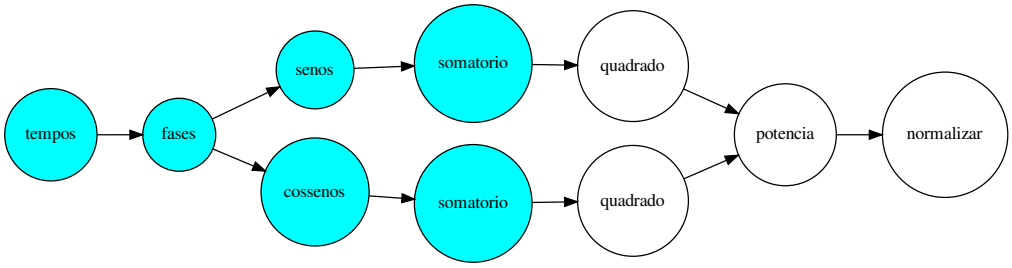


Figura 16: Nós de execução vetorial no periodograma Z_n^2.

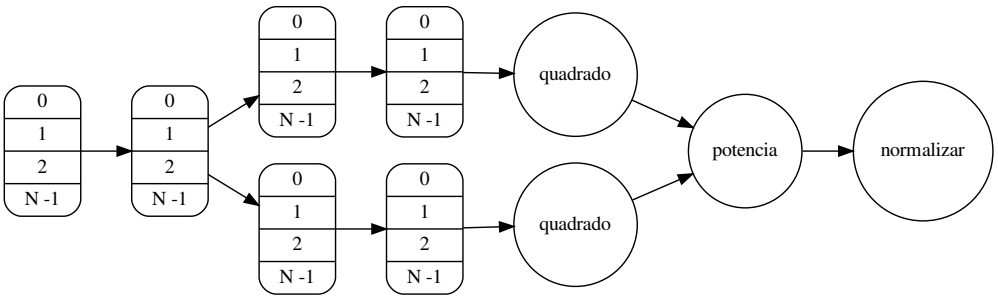


Figura 17: Grafo vetorial da execução do periodograma Z_n^2.

A Figura 17 representa a execução do algoritmo em apenas uma frequência. De fato, existe um potência para cada valor de frequência analisado. Assim, existe a possibilidade de paralelismo nos núcleos da CPU para a faixa de frequências: o espectro de potências resulta do cálculo de tarefas independentes. Considerando como ilustração o caso de execução para duas frequências, o espectro de potências é o resultado do nó destacado na Figura 18.

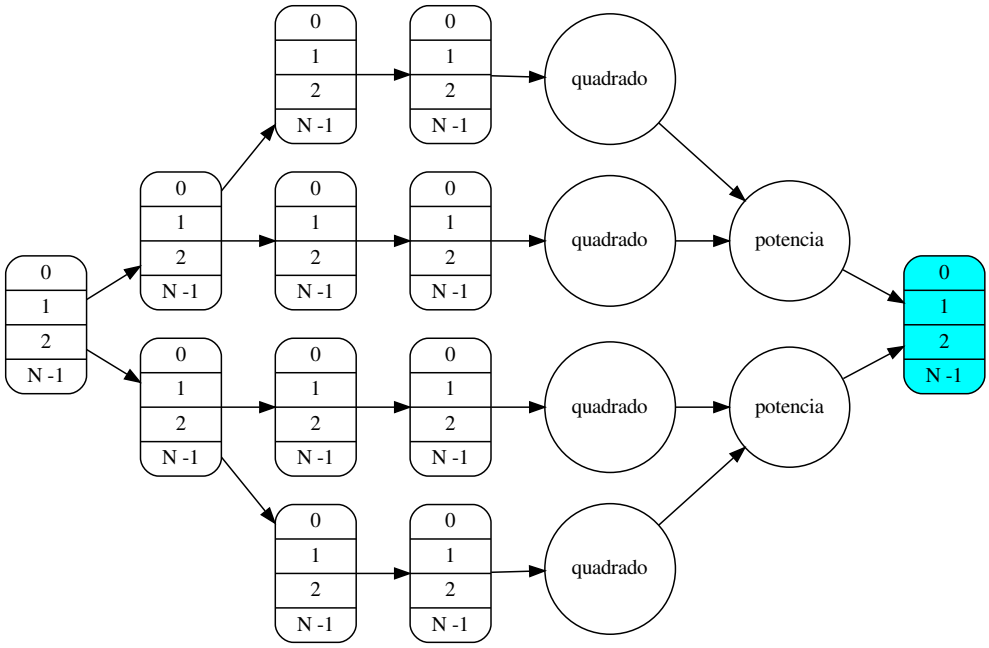


Figura 18: Grafo vetorial da execução paralela do periodograma Z_n^2.

Por fim, para comparar as performances do código fonte simplista em linguagem *Python*, como descrito na Seção 3.1, e o código fonte compilado com todas as otimizações aqui apresentadas, foi realizado um *benchmark* das diferentes funções. *Benchmark* é o ato de executar um processo com o fim de avaliar o desempenho relativo de um objeto, ou função, neste caso, com diferentes tamanhos de entrada. Os *Benchmarks* introduzem uma escala alternativa de tempo relativa ao tempo de execução de uma operação. O objetivo desta escala é prover um método de comparação da performance de vários sistemas diferentes, pois esta escala independe da arquitetura da máquina específica em que o código está sendo executado.

Considerando um mesmo arquivo de eventos, a Figura 19 apresenta o tempo de execução do código fonte em linguagem *Python* puro e seu crescimento exponencial com o aumento do número total de frequências analisadas. Por outro lado, o tempo de execução do código otimizado é praticamente linear em comparação com a alternativa simplista.

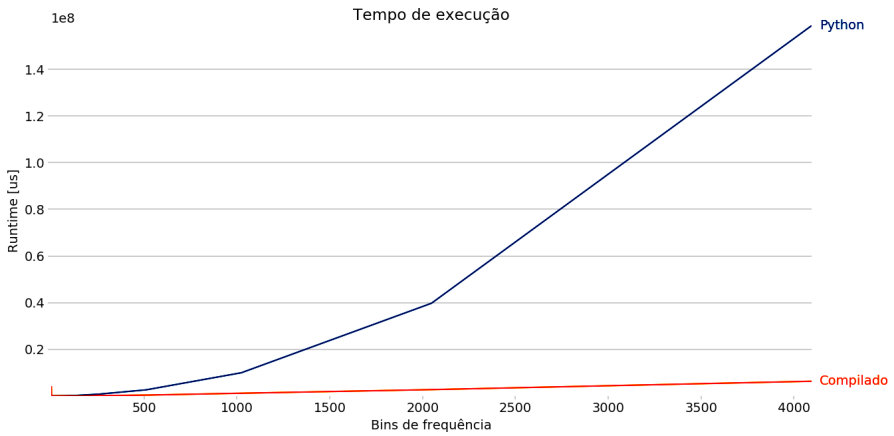


Figura 19: *Benchmark* comparativo dos tempos de execuções dos códigos.

Como a escala de tempo de execução é relativa a uma instrução de máquina, de forma a ser independente da máquina analisada, os valores apresentados não representam valores de tempo absoluto. Esses valores são apenas uma medida relativa da execução de uma instrução do programa, que devem ser multiplicados por uma constante relativa a cada máquina para uma estimativa do tempo absoluto. Existem diversas variáveis que influenciam nesta constante, como a frequência de operação da máquina, ciclos de *clock* e a quantidade de núcleos de processamento na CPU – e por isso se trata de uma medida relativa.

Além disso, outra coisa importante a ser notada é o custo inicial do processo de compilação. Como o código fonte é compilado assim que ocorre sua chamada, existe um *delay* inicial na execução do programa, conforme Figura 20.

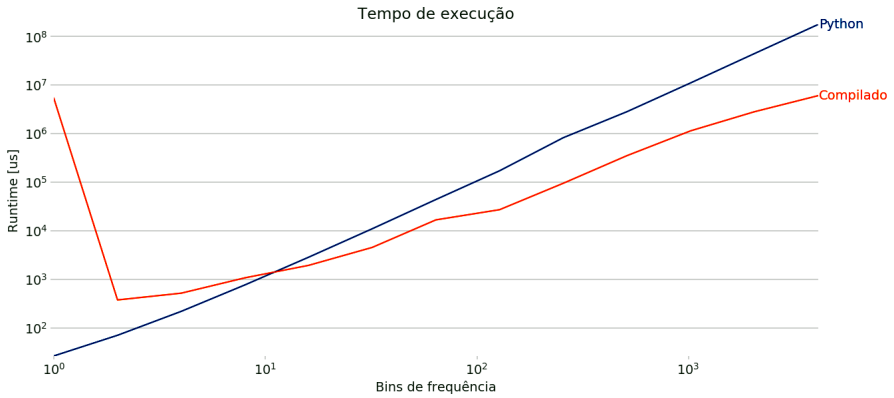


Figura 20: *Benchmark* comparativo dos tempos de execuções em escala logarítmica.

3.3 Criação de uma interface e empacotamento do programa

O programa desenvolvido está disponível na forma de pacote (z2n-periodogram). Um pacote em linguagem *Python* é uma coleção de módulos e submódulos na de arquivos e pastas. Pacotes são uma maneira de estruturar *names spaces* utilizando a sintaxe de *dotted names* do paradigma de programação orientada a objetos (ABADI; CARDELLI, 1998). A utilização de técnicas de modularização do código, além de organizar o código, promove uma abstração dos métodos que

o pacote implementa. Essa estratégia oferece a possibilidade de importar os módulos e métodos disponíveis no pacote de forma bastante simples e flexível.

Está disponível no site do *Python* (<https://www.python.org/>) um repositório de códigos oficial com pacotes construídos por terceiros e que podem ser instalados em qualquer plataforma. Este repositório de código oficial é chamado de *Python Package Index* (PyPI) e a melhor forma para acessá-lo é através de um gerenciador de pacotes. A instalação, atualização ou remoção do *Python* vem atualmente com um utilitário em linha de comando de nome PIP – acrônimo para Instalador de Pacotes *Python*. É nesse formato que o programa desenvolvido neste trabalho, de domínio público, e que funciona em ambiente Linux por linha de comando, está disponibilizado. Sua instalação é feita com o comando "pip install z2n-periodogram" e sua inicialização se dá com o comando "z2n" (<https://pypi.org/project/z2n-periodogram/>).

A abstração criada com o fim de melhor servir aos propósitos do programa desenvolvido neste trabalho foi a de dois objetos: *Series* e *Plot*. O objeto *Series* representa uma série temporal e os elementos necessários para investigá-la, como o tempo de chegada dos fótons, a faixa de frequência analisada e a amostragem em frequência adotada, assim como o seu resultado, que é o espectro de potências. Por outro lado, o objeto *Plot* permite a plotagem do espectro de potências de duas instâncias do objeto *Series*, sendo uma para a fonte observada, e outra, opcional ao usuário, para um arquivo que representa a contribuição de *background* associada a observação investigada.

Além disso, a interface é o elo de comunicação humano-computador, sendo a componente responsável por mapear ações do usuário em solicitações de processamento ao sistema, bem como apresentar os resultados obtidos. Foi escolhida a Interface via Linha de Comando (CLI) para a aplicação desenvolvida neste projeto, que é um programa que permite que o usuário digite comandos de texto dando instruções ao sistema para executar funções específicas. A interface em linha de comando implementa um fluxo dos módulos e métodos disponíveis no programa a partir de um ponto de entrada, de forma a abstrair a necessidade do usuário de entender cada bloco de funcionamento (ver Figura 21).

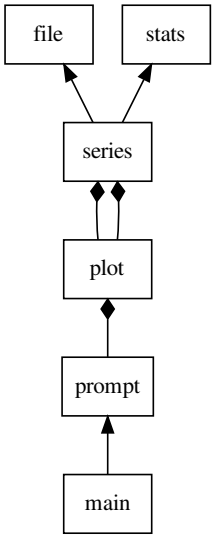


Figura 21: Digrama do fluxo da interface em linha de comando.

Foi implementada também uma janela de plotagem interativa com o uso da biblioteca Matplotlib (HUNTER, 2007), que oferece ao usuário praticidade na seleção de regiões no espectro de potências. Uma vez tendo obtido o espectro de potências, é possível ao usuário selecionar um pico de interesse de modo interativo, e essas seleções podem ser usadas para ajuste de uma função gaussiana, conforme Equação 11, ou também para mudar a faixa de frequências usada na construção do espectro de potências.

$$g(x) = A \cdot \mathrm{e}^{-\frac{(x-\overline{x})^2}{2\cdot\sigma^2}} \tag{11}$$

A prática de documentar códigos é bastante incentivada na comunidade de *software* aberto, visto que a documentação descreve o uso e funcionalidade do código para os usuários. O trabalho desenvolvido seguiu essa tendência, com a elaboração de uma documentação completa do programa e que está disponível em <https://z2n-periodogram.readthedocs.io>. A documentação fornece aos usuários um manual dos métodos disponíveis na aplicação, uma API (Interface de Programação de Aplicações) da implementação destes métodos, instruções para instalação e ajuda tanto na resolução de certos problemas encontrados no uso quanto na instalação do programa [5].

4 Resultados e discussões

O estudo de objetos astrofísicos se depara em diferentes contextos com a verificação de periodicidade associado a uma dada variável. Neste trabalho a variável é o brilho de uma fonte astrofísica, e sua dependência é temporal. O periodograma Z_n^2 foi implementado em linguagem Python, e tem como entrada uma

[5] O programa é aberto e portanto de característica dinâmica: informações sobre a versão mais recente estão disponíveis na documentação *online*.

lista de tempo associado a cada fóton registrado. Sua manipulação é feita pelo usuário em terminal Linux, com a entrada dos parâmetros necessários para a análise: definição dos limites da faixa de frequências que será varrida, o passo em frequência e a quantidade de harmônicos que serão considerados na análise. Como resultado final tem-se um espectro de potências. A ação se dá através de uma única linha de comando em terminal Linux ou através de interação do tipo passo-a-passo.

A área que se beneficia diretamente com o uso do periodograma Z_n^2 é a Astrofísica de Raios X, por lidar com baixa contagem em fótons (taxa típica inferior a poucos registros por segundo). Por isso a fonte escolhida para a validação cientifica do programa foi um pulsar em um sistema do tipo binária de raios-X, RX J0146.9+6121 (PALOMBARA; MEREGHETTI, 2006). O nome pulsar vem do fato da luz ser pulsada, no sentido de variar o brilho de modo regular no tempo. Nesse caso o pulsar é uma estrela de nêutrons, e o pulso decorre do fato dela estar girando. Assim, a validação do programa aqui desenvolvido foi feita com a determinação do período de rotação de uma fonte de luz pulsada (estrela de nêutrons) e com a comparação do valor obtido com o valor apresentado na literatura científica: aproximadamente 23 minutos, como determinado por Palombara e Mereghetti (2006).

A validação fez uso de um conjunto de dados em raios X [6] obtido com a câmera EPIC pn do satélite XMM-*Newton*, correspondendo a fótons com energia entre 0,3 e 10 keV (observação número 0201160101; realizada em 14/01/2004). Os dados foram disponibilizados em arquivos de eventos no formato FITS (Sistema de Transporte de Imagem Flexível) (Wells; Greisen; Harten, 1981). O FITS é o formato de arquivo digital mais comumente usado na astronomia, e é um padrão aberto que define um formato útil para armazenamento, transmissão e processamento de dados. Formatado em matrizes N-dimensionais, o arquivo digital pode conter informações como uma imagem 2D, ou tabelas 1D que podem armazenar, por exemplo, séries temporais como as analisadas neste trabalho. O acesso ao conteúdo do arquivo de eventos via Python foi feito por meio da biblioteca PyFITS (Barrett; Bridgman, 1999), parte do pacote Astropy, que disponibiliza métodos de leitura e escrita de arquivos no formato FITS.

O periodograma Z_n^2 foi aplicado ao arquivo de eventos apresentado acima. Porém, a título de ilustração, foi construída a partir dele a curva de luz (e portanto binada em tempo, que nesse caso foi em intervalos sucessivos de 100 s) que está apresentada na Figura 22, de modo a evidenciar o comportamento oscilatório do brilho em raios X do pulsar.

[6] A redução e o tratamento dos dados foram feitas pelo astrofísico Prof. Dr. Raimundo Lopes de Oliveira Filho. O conjunto está disponível para testes em <https://github.com/YohanAlexander/z2n-periodogram/blob/master/rxj0146_pnbaryc_0310kev.ds>.

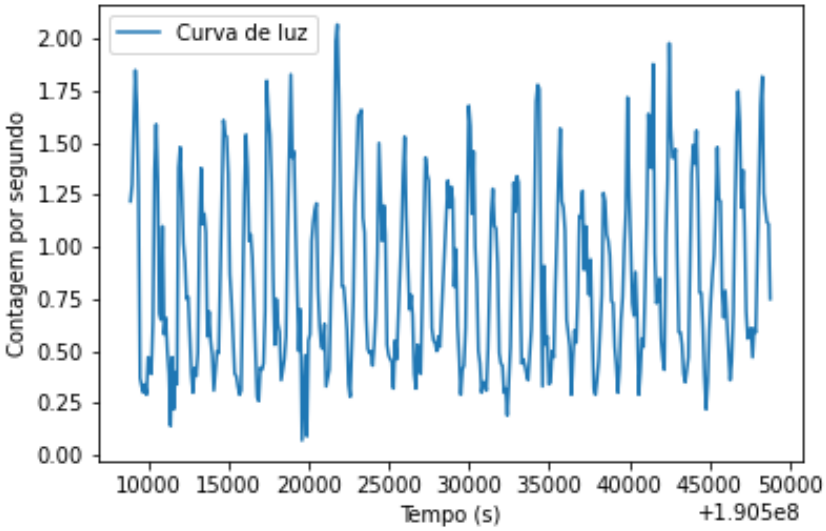


Figura 22: Curva de luz da binária de raios-x RX J0146.9+6121.

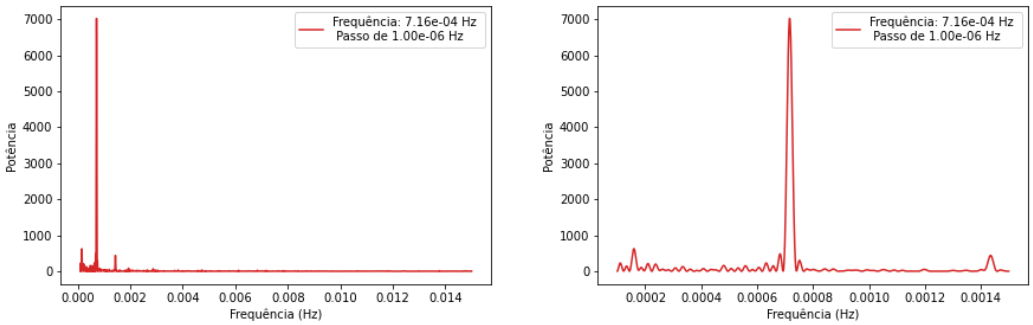
O tempo de chegada dos fótons relativo a fonte RX J0146.9+6121 foi utilizado como entrada do programa desenvolvido, e uma busca pela periodicidade indicada na literatura foi realizada. Como ponto de partida, foi considerada a faixa em frequência de 1×10^{-4} Hz a 1.5×10^{-2} Hz, e passos frequência (δ_ν) de 1×10^{-6} Hz. A janela de execução passo-a-passo do programa usando o conjunto de dados mencionado, e os parâmetros acima, é apresentada na Figura 23. O espectro de potências obtido pelo programa é apresentado nas Figura 24(a) e uma ampliação é apresentada na Figura 24(b), de modo a evidenciar a maior componente em frequência presente no espectro. Na ampliação é evidente que a potência associada à frequência 7×10^{-4} Hz se sobressai ao resto do espectro de potências: esse destaque é exatamente o que se espera como detecção positiva a partir de um periodograma. Vale registrar que foge do escopo deste trabalho avaliar a significância dos picos: uma vez tendo o periodograma, o usuário deve adotar, se for de seu interesse, a análise estatística que desejar para tal fim. Também foge do escopo deste trabalho apresentar uma incerteza associada a um dado período; porém, como uma estimativa conservadora, é apresentado ao usuário o valor do σ obtido como ajuste de uma gaussiana, conforme Equação 11, à distribuição do pico que o usuário escolher.

O fim maior desse ajuste é determinar a frequência associada a sua frequência máxima. Por outro lado, aproveita-se desse ajuste para inferir o σ associado à Gaussiana, que aqui é considerado como uma inferência conservativa da incerteza em frequência associada à frequência na qual ocorre o pico. Considerando o caso apresentado na Figura 24(b), na qual tem-se claro um pico em potência associado a frequência de 7×10^{-4} Hz, o ajuste da gaussiana indica uma frequência que é consistente com o equivalente em período de 23 min apontado para RX J0146.9+6121 por Palombara e Mereghetti (2006). Da equação 12, e mantendo o resultado numérico tal como obtido pelo programa, temos:

$$T = \frac{1}{f} = \frac{1}{0,0007(Hz)} = \frac{1396,648044(s)}{60} = 23,277467(min) \qquad (12)$$

```
z2n

        Z2n Software (2.0.0), a python package for periodograms analysis.
        Copyright (C) 2020, and MIT License, by Yohan Alexander [UFS].
        Type "help" for more information or "docs" for documentation.

The event file is needed.

Filename []: rxj0146_pnbaryc_0310kev.ds
Column TIME in EVENTS.
Event file loaded.
33888 events.
Observation time (Tobs): 40026.7 s
Sampling rate (1/Tobs): 2.5e-05 Hz
Nyquist 2*(1/Tobs): 5.0e-05 Hz
The frequency range is needed (Hz).

Nyquist as the minimum frequency [Y/n]? n

Minimum frequency (Hz) []: 1e-4

Maximum frequency (Hz) []: 1.5e-3

Use oversampling factor [Y/n]? n

Frequency steps (Hz) []: 1e-6

Minimum frequency: 1e-04 Hz
Maximum frequency: 1.5e-03 Hz
Frequency steps: 1.0e-06 Hz

Number of harmonics [1]: 1

Computation memory 0.11600 MB

Run with these values [Y/n]? y
1451 computation steps.
Calculating the periodogram: 100%|##################################| 1451/1451
Periodogram calculated.
Select the peak region to estimate uncertainty.
Is the peak region selected [y/N]? y
|                 | Z2N POWER                | GAUSSIAN FIT                 |
|-----------------|--------------------------|------------------------------|
| Power           | 6982.300396754459        | 7160.393492094508            |
| Frequency       | 0.0007170209617720351 Hz | 0.0007161472157063099 Hz     |
| Frequency error | _                        | +/- 8.692109549862768e-06 Hz |
| Period          | 1394.6593660645772 s     | 1396.3609409745961 s         |
| Period error    | _                        | +/- 16.744845163872924 s     |
| Pulsed Fraction | 64.19353519096036 %      | 65.00705222554575 %          |
```

Figura 23: Janela de execução passo-a-passo do programa (z2n-periodogram).



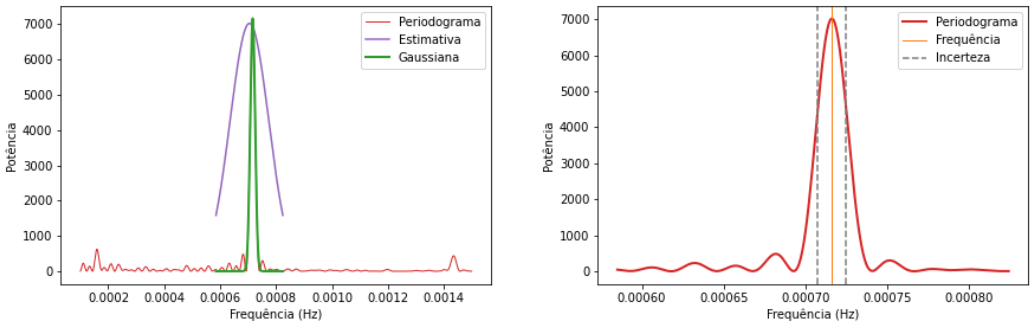
(a) (b)

Figura 24: Espectro de potências da binária de raios-x RX J0146.9+6121.

O ajuste da função gaussiana ao pico do periodograma selecionado pelo usuário é feito por meio do método dos mínimos quadrados ou regressão (SCHOMAKER et al., 2007). O método dos mínimos quadrados consiste em uma otimização matemática que encontra o melhor ajuste para um conjunto de dados, buscando minimizar a soma dos quadrados das diferenças entre o modelo e os dados observados (tais diferenças são chamadas resíduos), e está disponível no pacote de computação científica SciPy (VIRTANEN et al., 2020). Assim, o método busca minimizar o somatório na Equação 13, que quantifica a dispersão entre os valores observados (expressos por y_i^0) e os valores calculados correspondentes (y_i) de uma variável y.

$$S = \sum_{i=1}^{N} (y_i^0 - y_i)^2 \tag{13}$$

A Figura 25(a) ilustra a aplicação do método, tomando como exemplo o espectro de potências da fonte RX J0146.9+6121. É calculada uma estimativa inicial (curva em roxo), baseada nos valores de média e desvio padrão dos dados observados (curva em vermelho); em seguida o ajuste do modelo aos dados é otimizado com o método dos mínimos quadrados, como mostrado na curva em verde. A estimativa da incerteza é apresentada na Figura 25(b).



(a) (b)

Figura 25: Estimativa de incerteza pelo ajuste de uma gaussiana.

A "fração pulsada" (f_p) de um conjunto de dados em uma dada frequência é uma medida da amplitude da variação em tal frequência. Considerando que tal variação é senoidal, o método Z_n^2 permite estimar o equivalente ao percentual da fração pulsada no conjunto de dados com a Equação 14, onde N é o número total de eventos em tal conjunto (Zavlin et al., 2000).

$$f_p(\%) = \left(2 \cdot \frac{Z_n^2}{N}\right)^{\frac{1}{2}} \tag{14}$$

Além disso, o programa desenvolvido também permite ilustrar conceitos fundamentais relacionados ao processamento de sinais com taxa de amostragem finita, discutidos na seção 1.1. Na Figura 26, por exemplo, é possível visualizar o conceito de subamostragem em passos de frequência δ_ν, que implica na representação não realista do espectro de potências da fonte. Nessa figura

fica evidente a relevância da determinação do passo em frequência δ_ν ao traçar estratégias de buscas por periodicidade: uma subamostragem excessiva pode resultar em uma representação errada do espectro de potências da fonte.

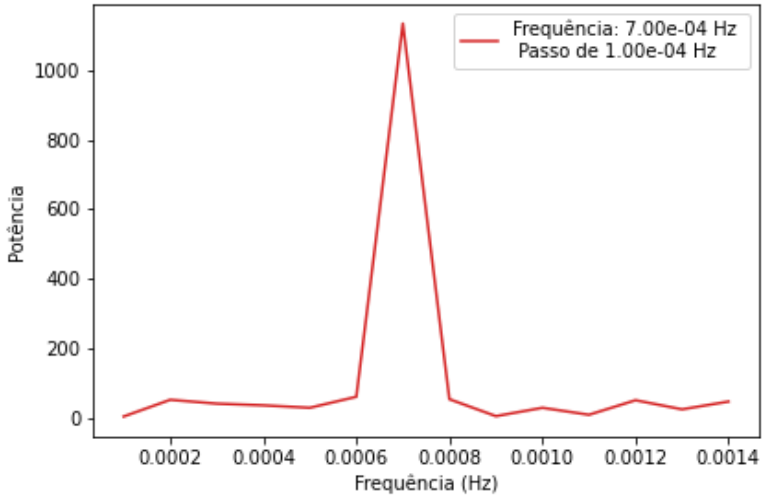


Figura 26: Espectro de potências da binária de raios-x RX J0146.9+6121 subamostrado em frequência.

No espectro de potência obtido torna-se evidente também o resultado de *alias* em frequência, cujo conceito foi apresentado anteriormente na seção 1.1. Essa característica por fim representa um falso positivo de uma componente em frequência, que neste caso esta relacionada a exploração de frequências menores que a resolução temporal da observação, conforme Figura 27.

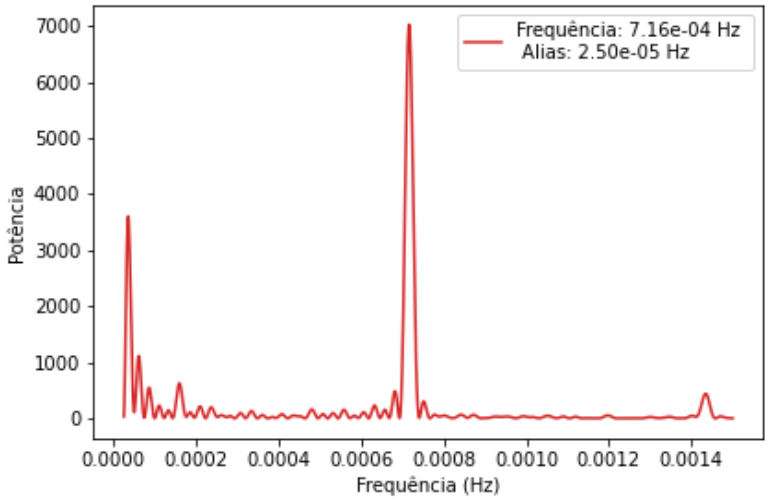


Figura 27: Espectro com *alias* da binária de raios-x RX J0146.9+6121.

Como a taxa de amostragem geralmente depende do instrumento de detecção, neste caso é possível afirmar que este artefato gerado (*alias*) é uma característica em sinal induzida por aspectos técnicos do detector. Um arquivo de *background* é um conjunto de eventos sem a contribuição da fonte em análise. Dessa forma, ao comparar os espectros de potência obtidos respectivamente

pela fonte e pelo *background*, pode ficar mais claro quais são as componentes em frequência associadas ao instrumento de detecção e quais as associadas a fonte observada, conforme Figura 28.

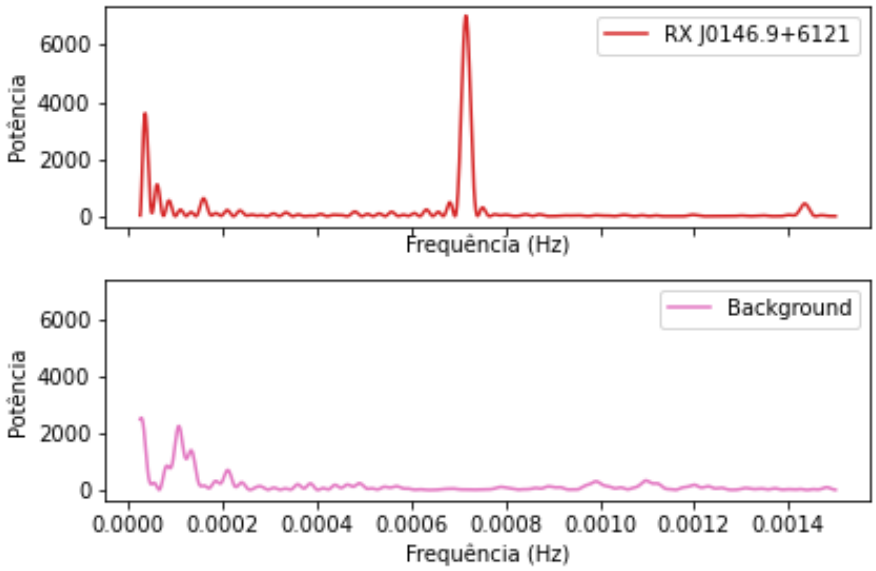


Figura 28: Comparativo do *background* da binária de raios-x RX J0146.9+6121.

5 Conclusões

Este trabalho se propôs a implementar o periodograma Z_n^2 (Buccheri et al., 1983) em linguagem *Python*, e oferecê-lo gratuitamente para uso da comunidade científica. O programa funciona em ambiente Linux por linha de comando, que pode ser de entrada única ou de forma interativa com o usuário. Foi elaborada uma documentação para acesso *online* e que está disponível em <https://z2n-periodogram.readthedocs.io>, que orienta o usuário quanto a instalação e execução do programa desenvolvido.

O programa foi validado com a exploração de dados obtidos por satélites de observações em raios X, com a comparação do resultado final (espectro de potências) de detecção positiva de oscilações periódicas de fontes astrofísicas apresentadas na literatura científica. Por fim, o programa foi explorado por pesquisadores que se mostraram satisfeitos com o processo de instalação e uso simplificados.

Referências

ABADI, M.; CARDELLI, L. *A Theory of Objects*. Springer New York, 1998. (Monographs in Computer Science). ISBN 9780387947754. Disponível em: <https://books.google.com.br/books?id=4xT3LgCPP5UC>.

ABELSON, H.; SUSSMAN, G.; SUSSMAN, J. *Structure and Interpretation of Computer Programs*. MIT Press, 1996. (Electrical engineering and computer science series). ISBN 9780262011532. Disponível em: <https://books.google.com.br/books?id=1DrQngEACAAJ>.

Barrett, P. E.; Bridgman, W. T. PyFITS, a FITS Module for Python. In: Mehringer, D. M.; Plante, R. L.; Roberts, D. A. (Ed.). *Astronomical Data Analysis Software and Systems VIII*. [s.n.], 1999. (Astronomical Society of the Pacific Conference Series, v. 172), p. 483. Disponível em: <https://ui.adsabs.harvard.edu/abs/1999ASPC..172..483B>.

Buccheri, R. et al. Search for pulsed γ-ray emission from radio pulsars in the COS-B data. *Astronomy & Astrophysics*, v. 128, p. 245–251, dec 1983. Disponível em: <https://ui.adsabs.harvard.edu/abs/1983A&A...128..245B>.

CHARNES, A.; FROME, E. L.; YU, P. L. The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *Journal of the American Statistical Association*, Informa UK Limited, v. 71, n. 353, p. 169–171, mar 1976. Disponível em: <https://doi.org/10.1080%2F01621459.1976.10481508>.

COLBURN, T.; SHUTE, G. Abstraction in computer science. *Minds and Machines*, Springer Science and Business Media LLC, v. 17, n. 2, p. 169–184, jun 2007. Disponível em: <https://doi.org/10.1007%2Fs11023-007-9061-7>.

HOPCROFT, J.; MOTWANI, R.; ULLMAN, J. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education International, 2003. (Addison-Wesley series in computer science). ISBN 9780321210296. Disponível em: <https://books.google.com.br/books?id=FQp0QgAACAAJ>.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 9, n. 3, p. 90–95, 2007. Disponível em: <https://doi.org/10.1109%2Fmcse.2007.55>.

HUPPENKOTHEN, D. et al. Stingray: A modern python library for spectral timing. *Astrophysical Journal*, American Astronomical Society, v. 881, n. 1, p. 39, aug 2019. Disponível em: <https://doi.org/10.3847%2F1538-4357%2Fab258d>.

LOMB, N. R. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science*, Springer Science and Business Media LLC, v. 39, n. 2, p. 447–462, feb 1976. Disponível em: <https://doi.org/10.1007%2Fbf00648343>.

Lopes de Oliveira, R. *A new class of X-ray emitters: the γ Cassiopeiae-like sources*. Tese (Doutorado) — Instituto de Astronomia, Geofísica e Ciências Atmosféricas, Universidade de São Paulo, R. do Matão 1226, 05508-090 São Paulo, Brazil Observatoire Astronomique, UMR 7550 CNRS, Université Louis Pasteur, 11 rue de l'Université, 67000 Strasbourg, France, ago. 2007. Disponível em: <https://ui.adsabs.harvard.edu/abs/2007PhDT.....20L>.

LUKE, H. The origins of the sampling theorem. *IEEE Communications Magazine*, Institute of Electrical and Electronics Engineers (IEEE), v. 37, n. 4, p. 106–108, apr 1999. Disponível em: <https://doi.org/10.1109%2F35.755459>.

MARKS, R. *Handbook of Fourier Analysis & Its Applications*. Oxford University Press, 2009. ISBN 9780198044307. Disponível em: <https://books.google.com.br/books?id=Sp7O4bocjPAC>.

OLIPHANT, T. E. Python for scientific computing. *Computing in Science & Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 9, n. 3, p. 10–20, 2007. Disponível em: <https://doi.org/10.1109%2Fmcse.2007.58>.

OLIVEIRA, R. L. de et al. SU lyn: Diagnosing the boundary layer with UV and hard x-ray data. *Astrophysical Journal*, American Astronomical Society, v. 864, n. 1, p. 46, aug 2018. Disponível em: <https://doi.org/10.3847%2F1538-4357%2Faad2d5>.

PALOMBARA, N. L.; MEREGHETTI, S. XMM-newton observation of the be/neutron star system RX j0146.9+6121: a soft x-ray excess in a low luminosity accreting pulsar. *Astronomy & Astrophysics*, EDP Sciences, v. 455, n. 1, p. 283–289, jul 2006. Disponível em: <https://doi.org/10.1051%2F0004-6361%3A20065107>.

PFENNING, F. Benjamin c. pierce. types and programming languages. the MIT press, cambridge, massachusetts, 2002, xxi + 623 pp. *Bulletin of Symbolic Logic*, Cambridge University Press (CUP), v. 10, n. 02, p. 213–214, jun 2004. Disponível em: <https://doi.org/10.1017%2Fs1079898600003954>.

PRICE-WHELAN and A. M. et al. The astropy project: Building an open-science project and status of the v2.0 core package. *Astronomical Journal*, American Astronomical Society, v. 156, n. 3, p. 123, aug 2018. Disponível em: <https://doi.org/10.3847%2F1538-3881%2Faabc4f>.

SCARGLE, J. D. Studies in astronomical time series analysis. II - statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal*, IOP Publishing, v. 263, p. 835, dec 1982. Disponível em: <https://doi.org/10.1086%2F160554>.

SCHOMAKER, M. et al. *Linear Models and Generalizations: Least Squares and Alternatives*. Springer Berlin Heidelberg, 2007. (Springer Series in Statistics). ISBN 9783540742272. Disponível em: <https://books.google.com.br/books?id=3LK9JoGEyN4C>.

SHANNON, C. Communication in the presence of noise. *Proceedings of the IRE*, Institute of Electrical and Electronics Engineers (IEEE), v. 37, n. 1, p. 10–21, jan 1949. Disponível em: <https://doi.org/10.1109%2Fjrproc.1949.232969>.

STONE, H. R66-50 an algorithm for the machine calculation of complex fourier series. *IEEE Transactions on Computers*, Institute of Electrical and Electronics Engineers (IEEE), EC-15, n. 4, p. 680–681, aug 1966. Disponível em: <https://doi.org/10.1109%2Fpgec.1966.264428>.

VANDERPLAS, J. T. Understanding the lomb–scargle periodogram. *Astrophysical Journal Supplement Series*, American Astronomical Society, v. 236, n. 1, p. 16, may 2018. Disponível em: <https://doi.org/10.3847%2F1538-4365%2Faab766>.

VIRTANEN, P. et al. SciPy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, Springer Science and Business Media LLC, v. 17, n. 3, p. 261–272, feb 2020. Disponível em: <https://doi.org/10.1038%2Fs41592-019-0686-2>.

WALT, S. van der; COLBERT, S. C.; VAROQUAUX, G. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 13, n. 2, p. 22–30, mar 2011. Disponível em: <https://doi.org/10.1109%2Fmcse.2011.37>.

Wells, D. C.; Greisen, E. W.; Harten, R. H. FITS - a Flexible Image Transport System. *Astronomy & Astrophysics Supplement Series*, v. 44, p. 363, jun 1981. Disponível em: <https://ui.adsabs.harvard.edu/abs/1981A&AS...44..363W>.

Zavlin, V. E. et al. Discovery of 424 Millisecond Pulsations from the Radio-quiet Neutron Star in the Supernova Remnant PKS 1209-51/52. , v. 540, n. 1, p. L25–L28, set. 2000. Disponível em: <https://ui.adsabs.harvard.edu/abs/2000ApJ...540L..25Z>.