DETERMINAÇÃO DE PARÂMETROS DE EXOPLANETAS POR AJUSTE DO TRÂNSITO DA CURVA DE LUZ PELO ALGORITMO MCMC

Felipe Pereira Pinho, Beatriz Duque Estrada Teixeira da Silva e Adriana Benetti Marques Valio

Apoio:PIBIC Mackenzie

RESUMO

Neste trabalho realizamos a implementação do algoritmo *Monte Carlo via Cadeias de Markov* (MCMC) ao código ECLIPSE para ajuste de curvas de luz obtidas pela missão espacial Kepler (NASA). As curvas de luz com trânsitos são utilizadas para determinação de parâmetros de exoplanetas e são ajustadas por MCMC dado um chute inicial. Para tanto, utilizamos a orientação a objetos, organização de código em blocos, paralelização e otimização de processos através de scripts na linguagem de programação C. O algoritmo foi implementado com sucesso, tendo obtido uma melhoria no tempo de execução de 50 até mais de 80 vezes.

Palavras-chave: Algoritmo MCMC, curva de luz, método de trânsito.

ABSTRACT

In this work we implemented the Markov Chain Monte Carlo (MCMC) algorithm to the ECLIPSE code to fit light curves obtained by the Kepler (NASA) space mission. Transit light curves are used to determine exoplanet parameters and are fit by MCMC given an initial guess. For that, we used object-oriented programming, organized code in blocks, parallelization, and process optimization through scripts in the C programming language. The algorithm was successfully implemented, improving the runtime by 50 to more than 80 times.

Keywords: MCMC algorithm, light curve, transit method.

1. INTRODUÇÃO

Na década de 1990 foram encontrados os primeiros exoplanetas: planetas que orbitam estrelas fora do nosso Sistema Solar. Com esta descoberta, diversos questionamentos foram levantados expandindo nosso conhecimento acerca do Universo, como por exemplo se há vida fora da Terra ou como ocorre o processo de formação de planetas. Para que estas questões pudessem ser respondidas, foram desenvolvidos diversos métodos para a identificação, estudo e compreensão destes astros.

Os métodos observacionais utilizados nos permitem detectar exoplanetas e obter seus diversos parâmetros, mas apenas até uma certa precisão. Por conta disso, tais métodos estão sempre em constante aprimoramento para que os resultados obtidos nas observações sejam cada vez mais precisos e condizentes com os parâmetros reais. Entre estes métodos, estão o método de trânsito, de velocidade radial, de cronometragem de pulsos, de geração de imagens diretas, de astrometria e de microlentes gravitacionais (FEITOSA FILHO, 2019). Nesta pesquisa, será utilizado o método de trânsito para a estimação dos parâmetros de exoplanetas.

Para encontrar exoplanetas cada vez menores e mais distantes do nosso Sistema Solar são utilizados satélites espaciais. Entre estes, podemos destacar a missão Kepler e seu sucessor, o satélite TESS (*Transiting Exoplanet Survey Satellite*). Estas missões possuem como objetivo principal a descoberta de novos exoplanetas através do método de trânsito.

O método de trânsito permite que sejam identificadas quedas de menos de 1% no brilho da estrela, podendo indicar a presença de exoplanetas. A partir da curva de luz, podem-se estimar parâmetros planetários, como o raio, semi-eixo e inclinação orbital da órbita do planeta. Existem, porém, alguns fatores que podem dificultar a determinação dos parâmetros do planeta por interferirem na curva de luz da estrela, como a presença de luas orbitando o exoplaneta (ZOLNERKEVIC, 2012) ou manchas na superfície da estrela (SILVA, 2003).

Por meio do uso do modelo ECLIPSE, programa desenvolvido em Python, serão geradas as curvas de luz de estrelas dados os parâmetros de entrada estelares e planetários para o estudo e estimação dos parâmetros planetários.

1.1. Objetivos

Implementar o algoritmo MCMC (*Monte Carlo via Cadeias de Markov*) no modelo ECLIPSE para ajuste dos dados da curva de luz de estrelas com trânsitos de exoplanetas. Este ajuste permite determinar com maior precisão os parâmetros reais do exoplaneta e sua órbita. Como o modelo também possibilita a simulação de manchas na superfície da estrela e exoluas, os parâmetros destes também podem ser obtidos.

Para que essa análise fosse possível, fez-se necessário o uso de programas em Python para o tratamento dos dados. Para tal, foram necessárias algumas melhorias para tornar o programa mais eficiente, como por exemplo:

- Divisão do código em blocos;
- Adaptação para orientação a objetos;
- Tratamentos de erros;
- Paralelização dos diversos cálculos realizados para aumento da velocidade de processamento.

Por fim, com o programa otimizado, foram utilizados dados observacionais (curvas de luz estelares) obtidos pelo método de trânsito por satélites como TESS e Kepler para adquirir os parâmetros necessários e desejados do planeta.

1.2. Justificativa

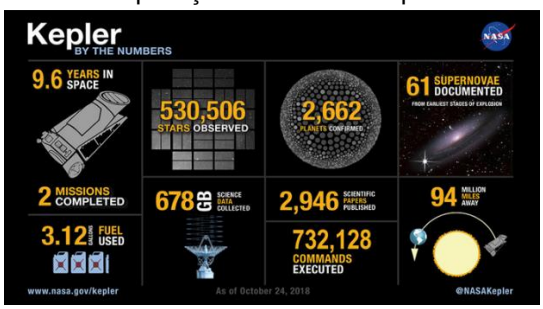
A linguagem de programação Python torna mais simples e acessível a forma de analisar exoplanetas pelo método de trânsito com ajuste de dados pelo modelo MCMC, enquanto a linguagem C permite com que todo o processamento do software seja otimizado, diminuindo seu tempo de execução em uma quantidade considerável de tempo. Todo o programa gerado foi disponibilizado publicamente em um repositório no Github para que este seja utilizado por qualquer usuário interessado, seja ele um cientista perito na área, um estudante ou alguém que desconheça do assunto, contribuindo assim, para a difusão de assuntos envolvendo ciências e astronomia.

2. REFERENCIAL TEÓRICO

2.1. Principais Satélites

O satélite Kepler, lançado ao espaço em 2009, foi responsável por monitorar mais de 500 mil estrelas e pelo método de detecção de trânsito, confirmar a existência de mais de 2500 planetas e encontrar mais de 2000 possíveis candidatos (KEPLER'S..., 2020).

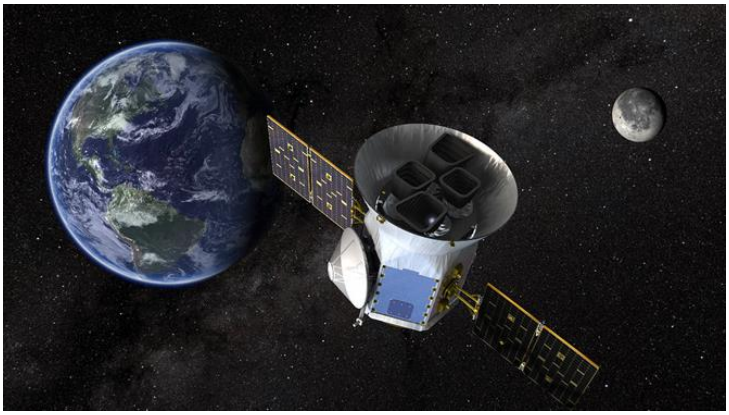
Figura 1 – Dados estatísticos relacionados às descobertas de exoplanetas durante o tempo de operação do satélite Kepler.



Fonte: Kepler's... (2020).

Como sucessor do Kepler, o satélite TESS, lançado ao espaço em 2018, também teve a missão de observar e analisar estrelas próximas ao nosso Sistema Solar. Também utilizando o método de detecção de trânsito, até o presente momento, já confirmou a existência de 120 planetas e encontrou mais de 2500 possíveis candidatos (TRANSITING..., 2021).

Figura 2 – Satélite TESS no espaço. Imagem ilustrativa.



Fonte: Transiting… (2021).

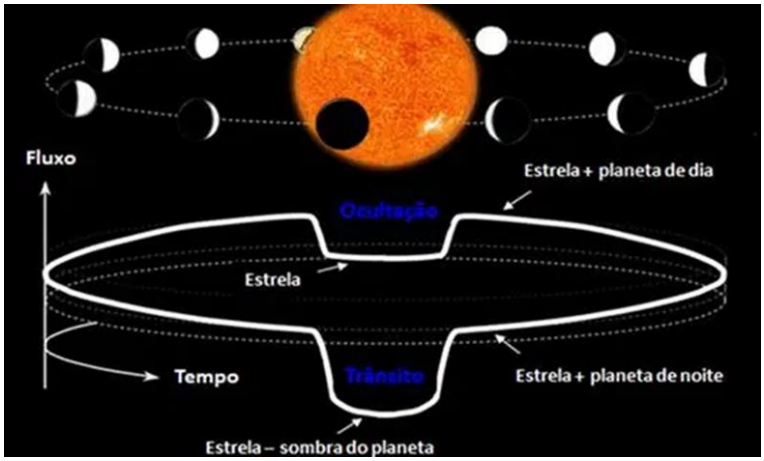
2.2. O método de trânsito

Quando um planeta, durante sua órbita, se encontra entre um observador e a estrela a qual orbita, este causa uma diminuição da intensidade do brilho desta estrela do ponto de vista do observador. Este decréscimo do brilho estelar, por ser proporcional à razão entre as áreas da estrela e do planeta, ocorre em porcentagens muito baixas, mas o suficiente para causar um efeito detectável na curva de luz de uma estrela. Por meio dessas variações da curva de luz, podemos encontrar novos planetas. Este método de detecção é chamado de método de trânsito (BARBOSA, 2017).

Para que seja possível identificar o planeta, é necessário que sua órbita possua um ângulo de inclinação favorável para o observador, o mais próximo de 90º para que o planeta eclipse a estrela. É também favorável que o período orbital do planeta seja curto, ou seja, que o raio orbital seja pequeno favorecendo a ocorrência do trânsito, ajudando na confirmação de que realmente é um planeta orbitando uma estrela.

Por meio do método observacional de trânsito é possível estimar diversos parâmetros do planeta, como o tempo de trânsito, ângulo de inclinação da órbita planetária, raio planetário e distância entre a estrela e o planeta, ou semieixo orbital.

Figura 3 – Representação esquemática do método de trânsito.



Fonte: Astropt.

2.3. Manchas, luas e planetas

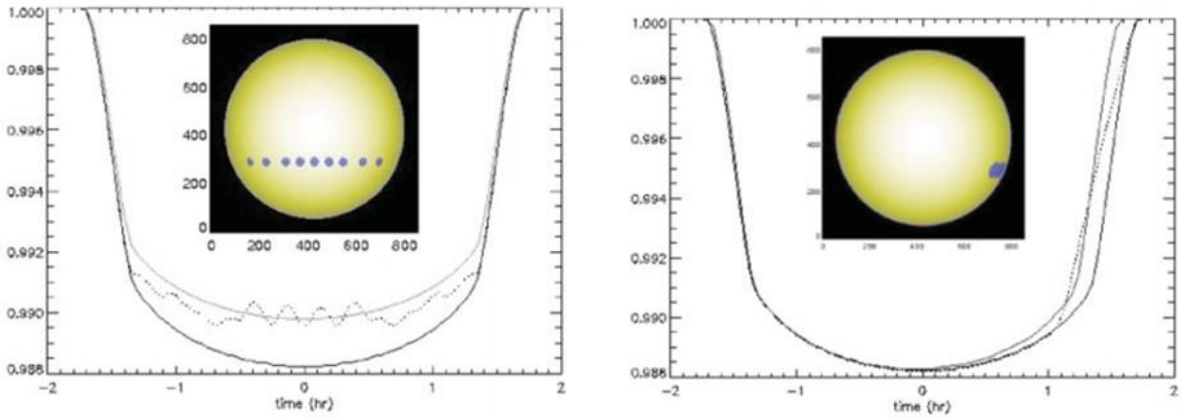
Existem alguns fatores que podem causar variações na curva de luz de uma estrela. Estes podem ser tanto a passagem de um objeto como um planeta ou uma lua eclipsando a estrela em relação a um observador ou a presença de manchas na superfície da estrela.

Os exoplanetas, alvo deste estudo, são os principais causadores de variações nas curvas de luz que devemos analisar. É por meio do eclipse da estrela que podemos analisar a curva de luz e descobrir alguns dos parâmetros planetários. Mas existem outros fatores que podem influenciar a curva de luz, dificultando a obtenção dos parâmetros desejados.

As manchas estelares são regiões de alta concentração de campos magnéticos na superfície das estrelas que interferem na temperatura dessas regiões, tornando a região das manchas mais frias em relação à superfície da estrela. Por conta deste decréscimo na temperatura, forma-se uma região escura na superfície da estrela (SIMPLICIO NETTO, 2019).

Estas manchas podem dificultar a obtenção precisa de parâmetros planetários durante a análise da curva de luz. É possível notar de acordo com a Figura 4, painel esquerdo, que a profundidade da curva de luz é afetada pela presença das manchas, influenciando o cálculo preciso do raio do planeta. Já no painel direito da Figura 4, é possível notar uma redução da duração do trânsito da curva de luz, afetando diretamente a obtenção do parâmetro semieixo orbital.

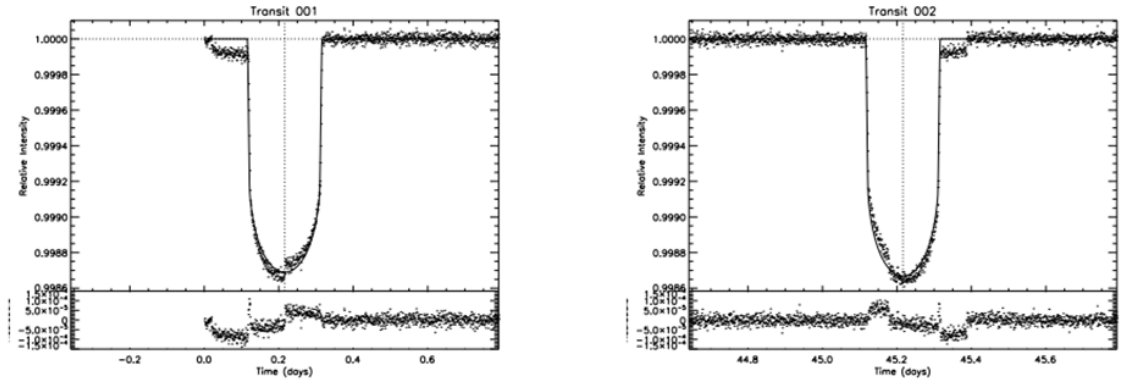
Figura 4 – Esquerda: Influência de múltiplas manchas na superfície da estrela, com os dados representados pela linha pontilhada e o ajuste pela linha cinza, em comparação a uma estrela sem manchas (curva preta). Direita: Influência de mancha na região próxima ao limbo, com os dados representados pela linha pontilhada e o ajuste pela linha cinza, em comparação a uma estrela sem manchas (curva preta).



Fonte: Silva-Valio (2009).

A presença de luas orbitando ao redor do exoplaneta pode também impactar na curva de luz da estrela, caso o raio e período de trânsito da lua sejam propícios. Como visto na Figura 5, a passagem de uma lua enquanto o planeta ainda está eclipsando a estrela pode alterar tanto o tempo de órbita, o que influencia na medição do semieixo orbital do exoplaneta, quanto a profundidade da curva de luz, o que influencia na medição do raio planetário.

Figura 5 – Esquerda: Trânsito planetário com uma lua transitando à frente do planeta, eclipsando a estrela antes de seu exoplaneta chegar, e saindo do disco estelar antes do planeta sair. Direita: Trânsito planetário com uma lua transitando após o planeta passar, eclipsando a estrela depois do planeta já estar eclipsando esta, e saindo do campo de visão depois do planeta já ter saído.



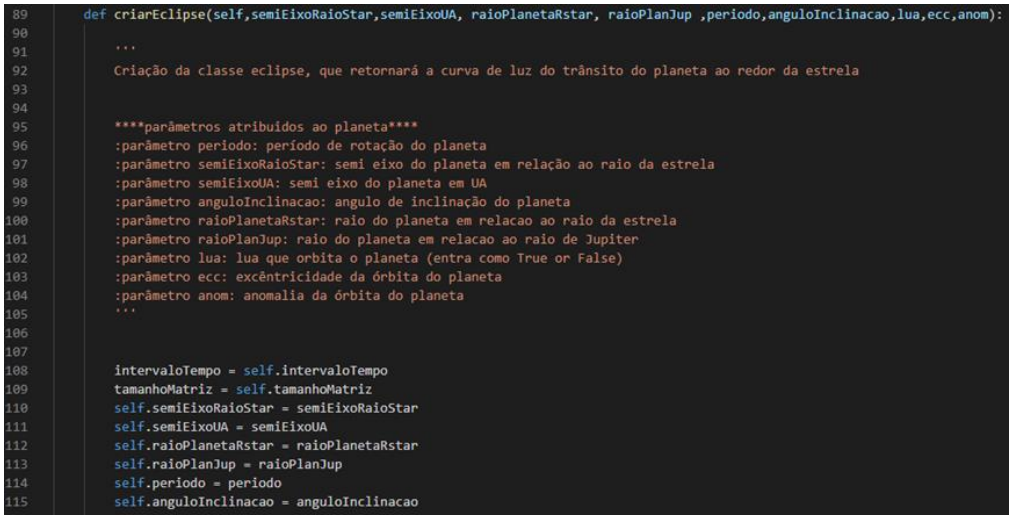
Fonte: Tusnski e Valio (2011).

2.4. Modelo ECLIPSE

Baseado no modelo de trânsitos planetários (SILVA, 2003), foi desenvolvido no Centro de Rádio Astronomia e Astrofísica Mackenzie (CRAAM) um programa orientado a objetos em Python responsável pela criação das classes **Eclipse** e **Lua** utilizadas no software (DUQUE, 2020). Neste estão contidas as funções responsáveis pelo cálculo e plotagem em gráfico da

curva de luz de uma estrela. O programa permite o cálculo da curva de luz da estrela para mais de um planeta, podendo ser adicionadas uma ou mais luas a estes. Também é possível adicionar uma ou mais manchas à estrela.

Figura 6 – Função **criarEclipse** retirada do código do programa eclipse_nv1.py.



```
def criarEclipse(self,semiEixoRaioStar,semiEixoUA, raioPlanetaRstar, raioPlanJup ,periodo,anguloInclinacao,lua,ecc,anom):

    '''
    Criação da classe eclipse, que retornará a curva de luz do trânsito do planeta ao redor da estrela


    ****parâmetros atribuídos ao planeta****
    :parâmetro periodo: período de rotação do planeta
    :parâmetro semiEixoRaioStar: semi eixo do planeta em relação ao raio da estrela
    :parâmetro semiEixoUA: semi eixo do planeta em UA
    :parâmetro anguloInclinacao: angulo de inclinação do planeta
    :parâmetro raioPlanetaRstar: raio do planeta em relacao ao raio da estrela
    :parâmetro raioPlanJup: raio do planeta em relacao ao raio de Jupiter
    :parâmetro lua: lua que orbita o planeta (entra como True or False)
    :parâmetro ecc: excêntricidade da órbita do planeta
    :parâmetro anom: anomalia da órbita do planeta
    '''


    intervaloTempo = self.intervaloTempo
    tamanhoMatriz = self.tamanhoMatriz
    self.semiEixoRaioStar = semiEixoRaioStar
    self.semiEixoUA = semiEixoUA
    self.raioPlanetaRstar = raioPlanetaRstar
    self.raioPlanJup = raioPlanJup
    self.periodo = periodo
    self.anguloInclinacao = anguloInclinacao
```

Fonte: Duque, 2020.

2.5. Algoritmo MCMC

As curvas de luz de estrelas observadas pelos satélites são os dados utilizados neste projeto. Para determinar os parâmetros planetários de maneira mais precisa, é necessário ajustar estas curvas de luz pelo modelo de trânsito utilizando o algoritmo MCMC (Monte Carlo via Cadeias de Markov).

Por meio de cálculos usando Estatística Bayesiana, o algoritmo MCMC permite com que, dada a curva de luz obtida pelo método de trânsito, seja possível estimar a curva de luz ótima. Dessa forma, são estimadas diversas combinações de parâmetros possíveis e são geradas para cada combinação uma curva de luz correspondente. Todas estas curvas de luz geradas pelo modelo são então comparadas com a curva de luz observada, analisando ponto a ponto pela regra de marginalização e obtendo uma distribuição normal ou gaussiana, à procura da que possua o menor erro entre os dados e o modelo. Deste modo, encontra-se a que é considerada o melhor ajuste e, por fim, os parâmetros do planeta são estimados com maior precisão (TUSNSKI; VALIO, 2011).

O algoritmo MCMC é utilizado através da linguagem de programação Python com o fim de analisar curvas de luz de uma estrela e determinar parâmetros do(s) planeta(s) que a orbitam. Dado um modelo de determinada estrela gerada, a amostragem dos parâmetros de determinada distribuição determinará o mais provável. É importante que a determinação dos parâmetros planetários seja feita de maneira precisa para que estudos sobre estes tenham

cada vez mais precisão, de modo a compreender quais são os efeitos da atividade estelar, como manchas, que mascaram esses parâmetros.

O método MCMC realiza uma quantidade enorme de cálculos, podendo tornar o processo de execução do programa muito lento. Para contornar essa situação, foi necessária a aplicação de técnicas de otimização de software, como o paralelismo de execução de instruções. Da mesma forma, viu-se a oportunidade do uso de linguagens auxiliares de níveis mais baixo do que Python, como C, para a execução deste método, uma vez que, por ser uma linguagem compilada, seu tempo de execução diminuirá ainda mais, tornando o programa ainda mais acessível e otimizado.

3. METODOLOGIA

A seguir serão apresentados os principais pontos que compuseram a metodologia do desenvolvimento do programa paralelizado, bem como as técnicas de otimização aplicadas. Além disso, também serão ressaltados pontos que envolvem a decisão do uso de técnicas de organização, divulgação e interação do programa com o usuário, como por exemplo o Jupyter Notebook e a plataforma GitHub.

3.1. Idealização do problema

Para a otimização do software, foram encontrados os pontos-chave onde o código possuía seu maior tempo de execução, sendo estes na geração da estrela e no cálculo da curva de luz. Ambos os trechos de código consistiam de múltiplos cálculos envolvendo matrizes. Para a melhora no desempenho, métodos de paralelização foram pesquisados para reduzir o tempo de execução do código.

Por possuir um Global Interpreter Lock (GIL), a linguagem de programação Python não permite o uso de *multithreading* paralelamente, uma vez que o GIL é uma trava que permite apenas uma *thread* ter o controle do interpretador do Python por vez (AJITSARIA, 2018). Dessa forma, para se ter uma melhora no tempo de execução de um código em Python, foi testado o impacto da chamada de scripts em C em um programa em Python uma vez que, por ser uma linguagem de mais baixo nível, funções em C conseguem fazer cálculos mais rapidamente do que funções em Python.

3.2. Pesquisa de bibliotecas de paralelização em C

Uma vez que o trecho de código definido para a otimização do programa envolvia apenas *for loops*, a biblioteca escolhida para a paralelização do processo foi a OpenMP. Esta é uma biblioteca de mais alto nível e possui mais facilidade de uso, uma vez que detecta automaticamente quantos *threads* disponíveis para uso a máquina pode oferecer, mantendo um alto ganho de desempenho.

3.3. Aplicação de scripts de C em Python

Para a aplicação da conexão do programa em Python com *scripts* em C, primeiramente foi verificada a melhora de desempenho através de programas testes, utilizando tanto programas completamente escritos em Python como programas em Python com o uso de *scripts* em C. Para o uso de *scripts* em C, a biblioteca utilizada em Python foi a “ctypes”.

Os algoritmos de teste funcionam da seguinte forma:

- Criação de um vetor com 268435456 (2 elevado a 28) posições para o programa em Python; ou 1073741824 (2 elevado a 30) posições para o programa em Python com script em C;
- Criação de um *for loop* para preencher todas as posições do vetor com 1;
- Criação de um *for loop* para somar o total de todas as posições do vetor;
- *Print* final do valor total obtido.

Figura 7 – Tempo de execução do programa em Python (268435456 recursões).

```
root@DESKTOP-9M7N84G:/mnt/c/Users/Pinho/Desktop/Testes# time -p python3 Teste.py
268435456
real 38.64
user 36.45
sys 2.15
```

Fonte: Autor.

Figura 8 – Tempo de execução do programa em Python com *script* em C não paralelizado (1073741824 recursões).

```
root@DESKTOP-9M7N84G:/mnt/c/Users/Pinho/Desktop/Testes# time -p python3 TesteC.py
1073741824
real 7.29
user 4.92
sys 2.37
root@DESKTOP-9M7N84G:/mnt/c/Users/Pinho/Desktop/Testes#
```

Fonte: Autor.

Figura 9 – Tempo de execução do programa em Python com *script* em C paralelizado (1073741824 recursões) utilizando 4 *threads*.

```
root@DESKTOP-9M7N84G:/mnt/c/Users/Pinho/Desktop/Testes# export OMP_NUM_THREADS=4
root@DESKTOP-9M7N84G:/mnt/c/Users/Pinho/Desktop/Testes# time -p python3 TesteC-OMP.py
1073741824
real 3.67
user 4.51
sys 8.01
```

Fonte: Autor.

Analisando as Figuras 7, 8 e 9, é possível verificar, comparando o tempo real de execução, uma maior eficácia quando são chamadas funções de *scripts* em C por programas em Python. Uma vez confirmada tal eficácia, foi realizada a implementação que será explicada a seguir.

3.3.1. Pré-requisitos

Para o funcionamento das chamadas de *scripts* de C em Python, os arquivos contendo as funções em C precisaram ser compilados em arquivos com a extensão ".dll", para serem utilizados no sistema operacional Windows. Para que fossem compilados para o Windows de 64 bits, modelo mais utilizado nos dias atuais, foi necessário o uso da ferramenta de compilação TDM GCC 64 (TDM-GCC..., 2021). É importante ressaltar que, por quesito de compatibilidade, a versão do Python utilizada para desenvolvimento do software foi a versão 3.8 de 64 bits, uma vez que bibliotecas importantes, como a *Lightkurve*, biblioteca de obtenção de dados dos satélites TESS e Kepler, não possui compatibilidade para instalação em versões mais atuais do Python.

3.3.2. Implementação

Para a implementação dos *scripts* em C, foi criado um programa auxiliar "func.c", contendo as funções que são utilizadas no programa em Python através da biblioteca ctypes. O *script* em C possui funções para agilizar cálculos tanto na criação da matriz da estrela como no cálculo da curva de luz gerada para uma estrela.

Para a criação dos dlls de 32 bits, os seguintes comandos foram utilizados para compilação dos arquivos:

- gcc -std=c11 -Wall -Wextra -pedantic -c -fPIC -fopenmp func.c -o func32.o -O3
- gcc -shared -fopenmp func32.o -o func32.dll -O3

Para a criação dos dlls de 64 bits, os seguintes comandos foram utilizados para compilação dos arquivos:

- gcc -std=c11 -m64 -DARCH_X86_64=1 -Wall -Wextra -pedantic -c -fPIC -fopenmp func.c -o func64.o -O3
- gcc -shared -m64 -DARCH_X86_64=1 -fopenmp func64.o -o func64.dll -O3

Figura 10 – Função para criação da estrela, em C.

```c
int* criaEstrela(int lin, int col, int tamanhoMatriz, float raio, float intensidadeMaxima, float coeficienteHum, float coeficienteDois){
    int i, j;
    int *estrela = (int*) malloc (lin * col * sizeof(int*));
    int index;

#pragma omp parallel
{
#pragma omp for collapse(2)
    for(i=0;i<lin;i++){
        for(j=0;j<col;j++){
            index = i*(lin) + j;
            estrela[index] = 0;
        }
    }
    float distanciaCentro;
    float cosTheta;

#pragma omp for collapse(2)
    for(j=0;j<col;j++){
        for(i=0;i<lin;i++){
            distanciaCentro = sqrt(pow(i-tamanhoMatriz/2,2) + pow(j-tamanhoMatriz/2,2));
            if(distanciaCentro <= raio){
                cosTheta = sqrt(1-pow(distanciaCentro/raio,2));
                index = i*(lin) + j;
                estrela[index] = (int) (intensidadeMaxima * (1 - coeficienteHum * (1 - cosTheta) - coeficienteDois * (pow(1 - cosTheta,2))));
            }
        }
    }
}
```

Fonte: Autor.

Figura 11 – Função para a criação da curva de luz, em C.

```c
double curvaLuz(double x0, double y0, int tamanhoMatriz, int raioPlanetaPixel, double *estrelaManchada, double *kk, double maxCurvaLuz){
    double valor = 0;
    int i;

#pragma omp parallel for reduction(+:valor)
    for(i=0;i<tamanhoMatriz*tamanhoMatriz;i++){
        if(pow((kk[i]/tamanhoMatriz-y0),2) + pow((kk[i]-tamanhoMatriz*floor(kk[i]/tamanhoMatriz)-x0),2) > pow(raioPlanetaPixel,2)){
            valor += estrelaManchada[i];
        }
    }

    valor = valor/maxCurvaLuz;

    return valor;
}
```

Fonte: Autor.

Para a chamada e uso das funções no programa em Python, foram necessárias as bibliotecas ctypes e numpy. Para o reconhecimento do sistema operacional do usuário, foram necessárias as bibliotecas nativas sys e platform.

Figura 12 – Trecho de código responsável pelo reconhecimento do sistema operacional e habilitação das entradas e saídas da função de criação da estrela, no arquivo estrela.py.

```python
if(platform.architecture()[0] == "32bit"):
    my_func = WinDLL('c/func32.dll', winmode = 0x8)
elif(platform.architecture()[0] == "64bit"):
    my_func = WinDLL('c/func64.dll', winmode = 0x8)

my_func.criaEstrela.restype = ndpointer(dtype=c_int, ndim=2, shape=(self.tamanhoMatriz,self.tamanhoMatriz))
self.estrela = my_func.criaEstrela(self.tamanhoMatriz,self.tamanhoMatriz,self.tamanhoMatriz,c_float(self.raio),c_float(self.intensidadeMaxima),c_float(self.coeficienteHum),c_float(self.coeficienteDois))
```

Fonte: Autor.

Figura 13 – Trecho de código responsável pelo reconhecimento do sistema operacional e habilitação das entradas e saídas das funções de geração da curva de luz, no arquivo Eclipse.py.

```python
# Verifica se o Python é 32 ou 64 bit
if(platform.architecture()[0] == "32bit"):
    my_func = WinDLL('c/func32.dll', winmode = 0x8)
elif(platform.architecture()[0] == "64bit"):
    my_func = WinDLL('c/func64.dll', winmode = 0x8)

# Prepara os tipos de cada variável dos argumentos e do retorno da função do calculo da curva de luz
my_func.curvaLuz.restype = c_double
my_func.curvaLuz.argtypes = c_double,c_double,c_int,c_int,POINTER(c_double),POINTER(c_double),c_double
my_func.curvaLuzLua.restype = c_double
my_func.curvaLuzLua.argtypes = c_double,c_double,c_double,c_double,c_double,c_int,c_int,POINTER(c_double),POINTER(c_double),c_double
```

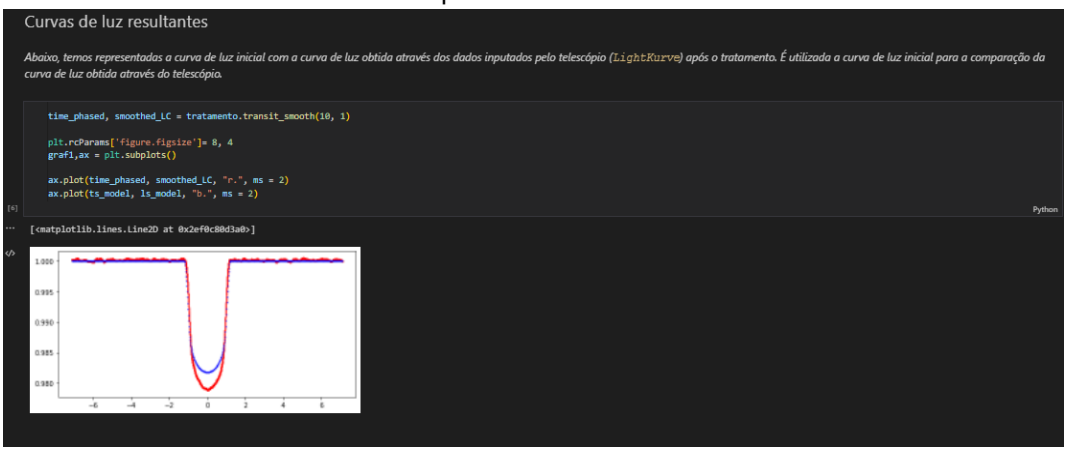
Fonte: Autor.

Dessa forma, a chamada da função em Python ocorrerá da mesma forma de uma função nativa em Python, porém com um tempo de execução muito menor, como será mostrado nos resultados.

3.4. Uso do Jupyter Notebook

O Jupyter Notebook foi escolhido como forma de desenvolvimento principal do programa pois permite ao desenvolvedor desenvolver anotações que auxiliam na compreensão e organização do código. Como pesquisas científicas envolvendo o método de trânsito fazem parte do desenvolvimento do projeto, diversos textos auxiliares foram adicionados ao decorrer do programa como forma de embasar e acompanhar textualmente o desenvolvimento de cada parte do código, como representado na Figura 13 abaixo.

Figura 13 – Exemplo de textos auxiliares para ajuda na identificação do que um bloco de código é responsável. No gráfico: Em vermelho, a curva de luz ajustada; em azul, a curva de luz de acordo com os parâmetros de entrada.



Fonte: Autor.

Além disso, em nível de usuário, o Jupyter Notebook permite que o mesmo rode o programa em blocos, facilitando sua interação com cada parte do código desenvolvida e permitindo que o mesmo interaja com os blocos que achar necessário para seu estudo no momento, como demonstrado na Figura 14 a seguir.

Figura 14 – Exemplo de blocos de código que podem ser rodados pelo usuário de forma individual (indicados através de setas vermelhas).

Fonte: Autor.

3.5. Github

A plataforma GitHub foi utilizada como forma de controlar o versionamento do programa desenvolvido, enquanto atuava como solução de divulgação dos temas científicos desenvolvidos no projeto. O projeto foi postado na plataforma do CRAAM (Centro de Rádio Astronomia e Astrofísica do Mackenzie), para dessa forma, facilitar o acesso e a busca do programa de outros estudantes da instituição ou professores interessados nos assuntos referentes ao projeto.

O programa completo pode ser visualizado e baixado através do seguinte link:

- https://github.com/Transit-Model-CRAAM/pipelineMCMC

3.6. Desenvolvimento do Algoritmo MCMC

Utilizando como base um arquivo já desenvolvido anteriormente no CRAAM, o algoritmo MCMC foi reformulado para comportar as alterações feitas nos programas base estrela.py e eclipse.py. A estrutura do programa também foi alterada para comportar a orientação a objetos, sendo criadas as novas classes Modelo, Tratamento e Ajuste.

Para a aplicação do MCMC, primeiro é necessária a adição do modelo, coletado pela biblioteca *Lightkurve*, que baixa os dados de curva de luz de trânsito. Também é possível tratar a curva de luz original, que possui ruído, fazendo com que ela fique mais suave, como pode ser visto na Figura 13. Os parâmetros de entrada são definidos pelo usuário, como o número de *walkers* e de iterações, assim como o número de recursões a serem descartadas inicialmente (*burn in*). Com estes valores definidos, o algoritmo MCMC é executado, calculando curvas de luz com diversos parâmetros diferentes a fim de descobrir qual curva é a ideal, dado o modelo, para a curva de luz observada.

4. RESULTADO E DISCUSSÃO

A seguir são apresentados os ganhos em tempo de execução do programa tanto para a criação da matriz *estrela* quanto para a geração da curva de luz, assim como algumas curvas de luz geradas pelo algoritmo MCMC presente no programa. Todos os testes foram realizados em um computador com processador AMD Ryzen 5 3600 com 6 cores e 3.59 GHz de *clock*, 32 GB de RAM 2133 MHz e sistema operacional Windows 10.

4.1. Tempos de execução

Após a implementação dos *scripts* em C, análises de desempenho foram feitas para verificar o ganho em tempo de execução do programa em cada parte que foi alterado. Como é possível verificar na Tabela 1, ao utilizar *scripts* em C não paralelizados o ganho de tempo é de 10 vezes, enquanto utilizando *scripts* de C paralelizados o ganho de tempo chega a 34 vezes.

Tabela 1 – Comparação dos tempos de execução para a criação da matriz estrela de 856 X 856 pixels em Python, Python com *script* em C e Python com *script* em C paralelizado. O ganho é medido em relação ao tempo de execução em Python.

Tabela de Tempos			
Tamanho Matriz = 856			
Linguagem	*Python*	*C*	*C-Paralelo*
Tempo 1 (s)	1,203272	0,121027	0,02992034
Tempo 2 (s)	1,19927	0,122027	0,039009571
Tempo 3 (s)	1,197269	0,122028	0,037007809
Tempo 4 (s)	1,199269	0,122026	0,036008358
Tempo 5 (s)	1,225214	0,121308	0,034007788
Média	1,204859	0,121683	0,035190773
Ganho	1	9,901607	34,23792147

Fonte: Autor.

Também foram realizados testes aumentando o tamanho de pixels da matriz para 4280x4280 e 8560x8560 pixels, com a finalidade de verificar o ganho de tempo para casos de uso maiores.

Tabela 2 – Comparação dos tempos de execução para a criação da matriz estrela em Python, Python com *script* em C e Python com *script* em C paralelizado. O ganho é medido em relação ao tempo de execução em Python. Esquerda: Matriz de 4280x4280 pixels; Direita: Matriz de 8560x8560 pixels.

Tabela de Tempos				***Tabela de Tempos***			
Tamanho Matriz = 4280				**Tamanho Matriz = 8560**			
Linguagem	*Python*	*C*	*C-Paralelo*	Linguagem	*Python*	*C*	*C-Para*
Tempo 1 (s)	17,87931	1,55652	0,34707856	Tempo 1 (s)	68,65416	6,05529	1,291533
Tempo 2 (s)	18,0556	1,555792	0,3543973	Tempo 2 (s)	69,66869	6,034745	1,379547
Tempo 3 (s)	18,17764	1,564352	0,37609148	Tempo 3 (s)	*	6,06352	1,27929
Tempo 4 (s)	17,64724	1,559293	0,3470788	Tempo 4 (s)	*	6,088376	1,295971
Tempo 5 (s)	17,615	1,555917	0,34907794	Tempo 5 (s)	*	6,057378	1,28729
Média	17,87496	1,558375	0,35474482	***Média***	69,16142	6,059862	1,306726
Ganho	1	11,47026	50,3882088	***Ganho***	1	11,41304	52,92726

Fonte: Autor.

Como é possível verificar na Tabela 2 à esquerda, para 4280x4280 pixels, o ganho de C não paralelizado cresceu 11 vezes, assim como o ganho de C paralelizado cresceu 50 vezes.

Aumentando ainda mais o tamanho da matriz para 8560x8560 pixels, como é visto na Tabela 2 à direita, o tempo de execução em Python fica muito grande e impraticável, com uma média de 69,16 segundos, enquanto o tempo de execução com C paralelizado é de 1,31 segundo. O programa em Python também começou a apresentar resultados inesperados, uma vez que teve dificuldades de alocação de memória para grandes cálculos devido à estrutura da linguagem de programação. Para os *scripts* em C, tal problema não aconteceu.

Tabela 3 – Comparação dos tempos de execução para a criação da matriz estrela de 8560x8560 pixels em Python, Python com *script* em C, Python com *script* em C paralelizado e Python com *script* em C paralelizado e compilado com o parâmetro O3 de otimização. O ganho é medido em relação ao tempo de execução em Python.

Tabela de Tempos				
Tamanho Matriz = 856				
Linguagem	*Python*	*C*	*C-Paralelo*	*C-Paralelo-O3*
Tempo 1 (s)	1,203272	0,121027	0,02992034	0,019003153
Tempo 2 (s)	1,19927	0,122027	0,039009571	0,023005486
Tempo 3 (s)	1,197269	0,122028	0,037007809	0,023005009
Tempo 4 (s)	1,199269	0,122026	0,036008358	0,023005486
Tempo 5 (s)	1,225214	0,121308	0,034007788	0,024006844
Média	1,204859	0,121683	0,035190773	***0,022405195***
Ganho	1	9,901607	34,23792147	***53,77587253***

Fonte: Autor.

Também foi testada a implementação do parâmetro O3 de otimização do próprio compilador do *script* em C, como pode ser visto na Tabela 3. Dessa forma, é possível verificar um ganho ainda maior, de 54 vezes no tempo de execução, com a implementação de tal

parâmetro para uma matriz de 856x856 pixels. Em um cenário onde o MCMC pode criar o modelo de uma estrela milhares de vezes, os ganhos percentuais apresentados são de grande importância para a otimização no tempo de execução dos programas.

Foram feitos testes para o cálculo da geração da curva de luz da estrela TRAPPIST-1, com parâmetros utilizados de acordo com a Tabela 4. Como é possível verificar na Tabela 5, os tempos de execução apenas em Python são longos tanto em 32 como 64 bits, tendo em média 54,5 segundos para o primeiro e 44,3 segundos para o segundo. Em compensação, ao utilizar *scripts* de C paralelizado, o ganho para 32 bits já é significativo, melhorando o tempo de execução em 6 vezes. Ao utilizar os scripts de C em 64 bits, a melhora no tempo de execução é de 84 vezes, o que torna extremamente interessante a aplicação de tais *scripts* no cenário do MCMC, onde a curva de luz será calculada milhares de vezes durante o processo.

Tabela 4 – Parâmetros da estrela, planeta e lua utilizados no teste.

Parâmetros	**valores**
Raio Estrela (Rsun)	0,117
u1	0,65
u2	0,28
Período Planeta	6,099
Ângulo de inclinação	89,86
Raio Planeta (Rjup)	0,0819
Semi-eixo (UA)	0,028
Ecentricidade	0
Anomalia	0
Raio Lua (Rterra)	0,5
Massa Lua (Mterra)	0,001
Massa Planeta (Mjup)	0,002
Período Lua	0,1

Fonte: Autor.

Tabela 5 – Tempos de execução para a criação da curva de luz de um planeta com lua em Python 32 bits, Python com *script* em C paralelo de 32 bits, Python 64 bits e Python com *script* em C paralelo de 64 bits.

	Tabela de tempo de execução sem animação (segundos)			
	Python 32	C 32	Python 64	C 64
	54,1168845	8,7470274	44,13836551	0,51999879
	55,1397576	8,89556575	43,7697444	0,52800059
	55,2963262	8,58399868	44,27376771	0,51900053
	53,9476991	8,74651742	44,86950064	0,52400064
	53,9779978	8,56399727	44,29877162	0,5509994
Media	54,495733	8,7074213	44,27002997	0,52839999
Ganho	1	6,25853868	1,230984778	83,7812843

Fonte: Autor.

4.2. Parâmetros obtidos

A seguir serão apresentados os parâmetros planetários e curva de luz de duas estrelas obtidos por meio do MCMC para dois planetas distintos: Kepler-17 b e Kepler-45 b. Para ambos, foram utilizados como parâmetros de entrada para o ajuste:

- Nwalkers = 70;
- Niter = 50;
- Burn in = 30.

4.2.1. Kepler-45

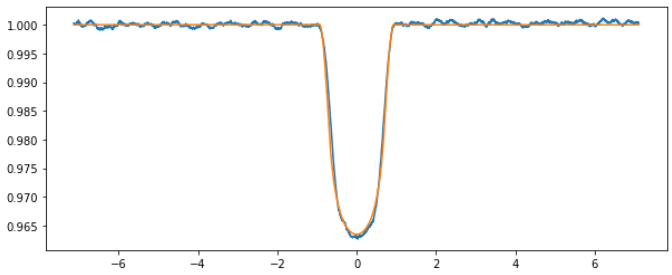
A execução do algoritmo MCMC para o cálculo da curva de luz para o trânsito do planeta Kepler-45 b levou 63 minutos e 51 segundos, consumindo 9,727 GB de memória RAM. Como visto na Figura 17, a curva de luz obtida através do programa é bem fiel à curva de luz obtida pelo método de trânsito. Os parâmetros obtidos (Tabela 7) tiveram uma leve discrepância dos parâmetros encontrados no site exoplanet.eu. Isso se deve ao fato da curva de luz obtida dos parâmetros do site não condizerem fielmente com a curva de luz de trânsito observada, como mostrado na Figura 18. O programa também nos permite verificar a correlação das variáveis através do *corner plot* disponível na Figura 19.

Tabela 7 – Comparação dos valores medidos pelo programa e valores do planeta disponíveis no site

Parâmetro	u1	u2	semiEixoUA	anguloInclinacao	raioPlanJup
Valor medido	0.582078873	0.800399793	0.021520635	87.3172047	1.5798488
Valor input	0.353	0.255	0.027	87	0.96

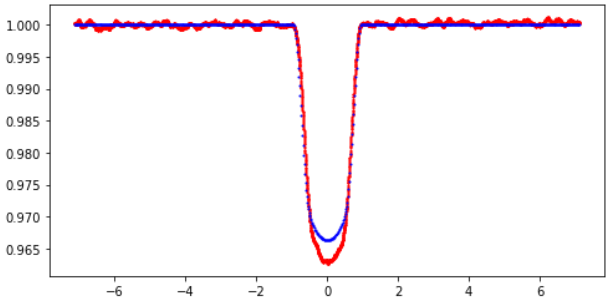
Fonte: Autor.

Figura 17 –Curva de luz calculada via MCMC (em laranja) e curva de luz de trânsito de Kepler-45 b observada pelo satélite Kepler (em azul).



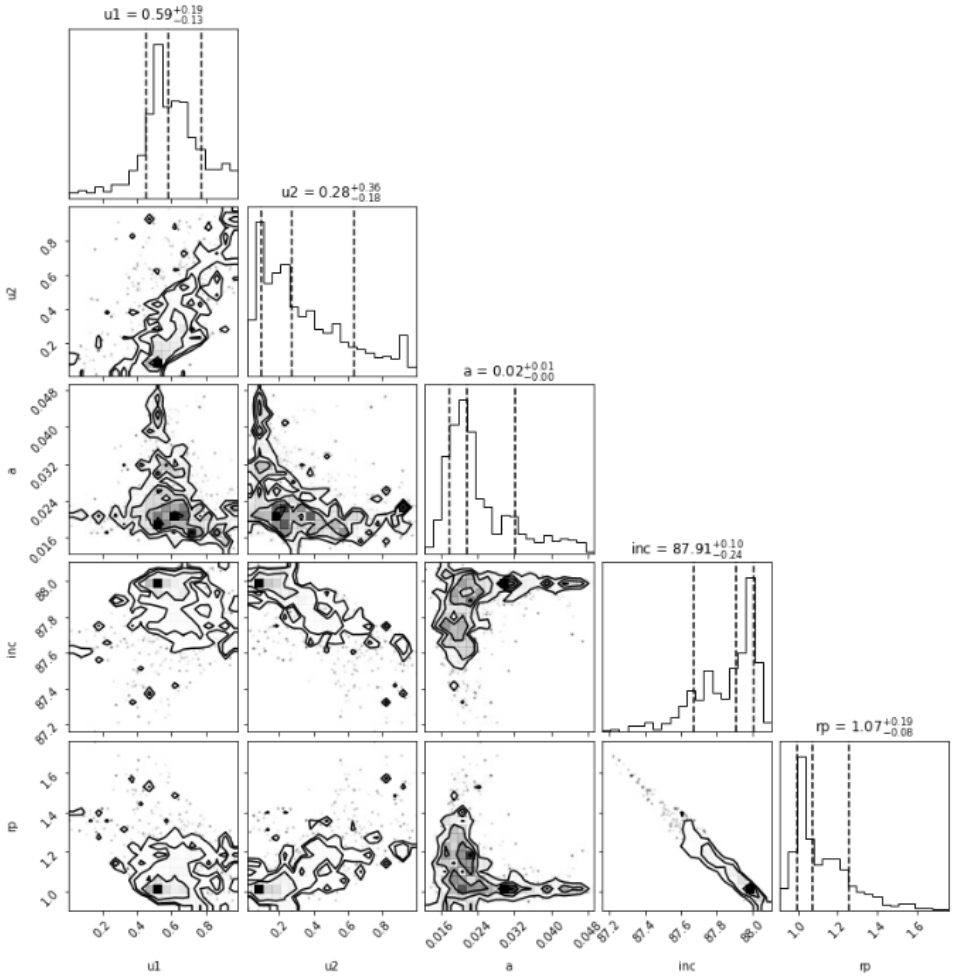
Fonte: Autor.

Figura 18 – Curva de luz baseada nos parâmetros retirados do site exoplanet.eu para Kepler-45 b (em azul) e curva de luz observada pelo satélite Kepler (em vermelho).



Fonte: Autor.

Figura 19 – *Corner plot* dos parâmetros calculados pelo MCMC para Kepler-45 b.



Fonte: Autor.

4.2.2. Kepler-17

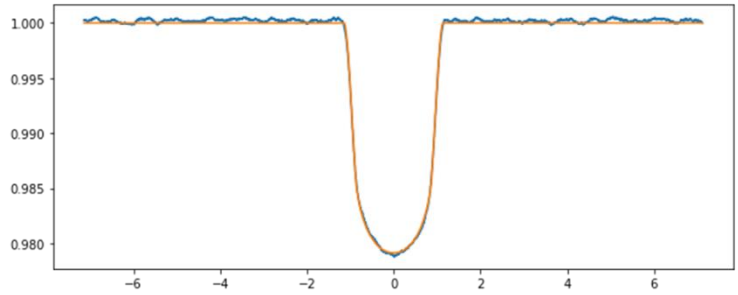
A execução do algoritmo MCMC para o cálculo da curva de luz para o trânsito do planeta Kepler-17 b levou 65 minutos e 19 segundos, consumindo 15,953 GB de memória RAM. Como visto na Figura 15, a curva de luz obtida através do programa é bem fiel à curva de luz obtida pelo método de trânsito. Os parâmetros obtidos também são bem próximos aos valores disponíveis no site exoplanet.eu (TEAM, 1995), conforme exemplificado na Tabela 6. O programa também nos permite verificar a correlação das variáveis através do *corner plot* disponível na Figura 16.

Tabela 6 – Comparação dos valores medidos pelo programa e valores do planeta disponíveis no site exoplanet.eu.

Parâmetro	u1	u2	semiEixoUA	anguloInclinacao	raioPlanJup
Valor medido	0.412384713	0.244866318	0.018224583	87.8771807	1.31165677
Valor site	0.405	0.262	0.02591	87.2	1.312

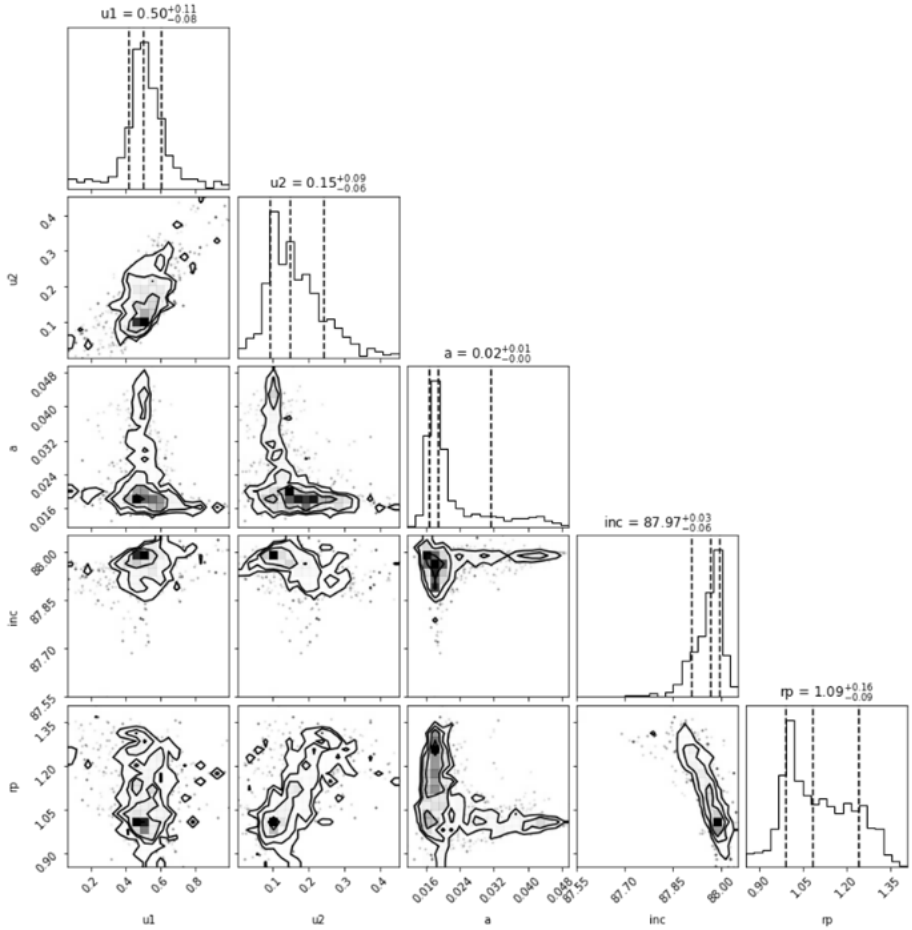
Fonte: Autor.

Figura 15 –Curva de luz calculada via MCMC (em laranja) e curva de luz observada do trânsito de Kepler-17 b pelo satélite Kepler (em azul).



Fonte: Autor.

Figura 16 – *Corner plot* dos parâmetros calculados pelo MCMC para os parâmetros de Kepler-17 b.



Fonte: Autor.

5. CONSIDERAÇÕES FINAIS

A programação em Python juntamente com *scripts* em C se mostrou muito eficiente. Enquanto a programação em Python auxilia na parte de plotagem, análise de gráficos e bibliotecas de astronomia, a linguagem C trata da parte de cálculos extensos, paralelização e otimizações do processo. Isto pode ser verificado ao analisar as tabelas comparativas de tempo de execução apresentadas, onde foi alcançada uma melhoria no tempo de execução de 50 até mais de 80 vezes.

É importante também ressaltar que, aliado às tecnologias estudadas, o algoritmo MCMC foi de grande importância para obtenção dos parâmetros do exoplaneta e sua órbita, se mostrando uma ferramenta eficaz para ajuste da curva de luz observada de uma estrela com ruído, com uma boa eficácia para a estimação de parâmetros de exoplanetas e suas incertezas.

6. REFERÊNCIAS

AJITSARIA, Abhinav. **What Is the Python Global Interpreter Lock (GIL)?**. [*S. l.*], 6 mar. 2018. Disponível em: https://realpython.com/python-gil/. Acesso em: 2 dez. 2021.

BARBOSA, Ruben. **Detecção de Exoplanetas: Introdução ao método do trânsito**. [*S. l.*]: Astropt, 2 fev. 2017. Disponível em: https://www.astropt.org/2017/02/02/detecao-de-exoplanetas-introducao-ao-metodo-do-transito/. Acesso em: 5 out. 2021.

DUQUE, B.D.E.T. **Atividade Estelar e o Raio de Exoplanetas**. [*S. l.*], 15 jul. 2020. Disponível em: https://github.com/biaduque/astronomy. Acesso em: 10 out. 2021.

FEITOSA FILHO, I.L. **Análise das Distribuições de Exoplanetas no Espaço de Parâmetros Físicos e Orbitais**. 2019. Dissertação (Mestrado em Astronomia) - Instituto de Astronomia, Geofísica e Ciências Atmosféricas da Universidade de São Paulo, São Paulo, 2019.

KEPLER'S legacy: discoveries and more. [*S. l.*]: NASA, 24 set. 2020. Disponível em: https://exoplanets.nasa.gov/keplerscience/. Acesso em: 15 ago. 2021.

SILVA, A.V.R. Method for spot detection on solar like stars. **The Astrophysical Journal**, [*S. l.*], v. 585, n. 2, p. 147-150, 11 fev. 2003.

SILVA-VALIO, Adriana. The influence of starspots activity on the determination of planetary transit parameters. **Proceedings IAU Symposium**. [*S. l.*], v 264, p. 440-442, 1 ago. 2009.

SIMPLICIO NETTO, Dirceu Yuri. **Rotação diferencial de estrelas pelos métodos de trânsito e máxima entropia**. 2019, 90 f. Tese (doutorado em Ciências e Aplicações Geoespaciais) - Universidade Presbiteriana Mackenzie, São Paulo, 2019.

TDM-GCC 10.3.0 release. [*S. l.*], 24 maio 2021. Disponível em: https://jmeubank.github.io/tdm-gcc/articles/2021-05/10.3.0-release. Acesso em: 12 dez. 2022.

TEAM, Exoplanet (coord.). The Extrasolar Planets Encyclopaedia. [*S. l.*], 1995. Disponível em: http://exoplanet.eu/. Acesso em: 4 abr. 2022.

TRANSITING Exoplanet Survey Satellite (TESS). [*S. l.*]: NASA, 22 mar. 2021. Disponível em: https://exoplanets.nasa.gov/tess/. Acesso em 15 ago. 2021.

TUSNSKI, L. R.; VALIO, A. Transit model of planets with moon and ring systems. **The Astrophysical Journal**. [*S. l.*], v. 743, n. 1, p. 97, 10 dez. 2011.

ZOLNERKEVIC, Igor. **Nas redondezas de outros mundos***.* 191. ed. [*S. l.*]: Pesquisa FAPESP, jan. 2012. Disponível em: https://revistapesquisa.fapesp.br/nas-redondezas-de-outros-mundos/. Acesso em: 14 set. 2021.

Contatos: felipe_pinho8@hotmail.com, biaduque7@hotmail.com e avalio@craam.mackenzie.br