Observatório Nacional

TESE DE DOUTORADO

PARALELIZAÇÃO EM GPU DE ALGORITMOS DE N CORPOS E APLICAÇÕES A SISTEMAS PLANETÁRIOS EXTRASSOLARES

ALAN COSTA DE SOUZA

RIO DE JANEIRO
2020

Ministério da Ciência, Tecnologia, Inovações e Comunicações
Observatório Nacional
Programa de Pós-Graduação

Tese de Doutorado

PARALELIZAÇÃO EM GPU DE ALGORITMOS DE N CORPOS E APLICAÇÕES A SISTEMAS PLANETÁRIOS EXTRASSOLARES

por

Alan Costa de Souza

Tese submetida ao Corpo Docente do Programa de Pós-graduação em Astronomia do Observatório Nacional, como parte dos requisitos necessários para a obtenção do Grau de Doutor em Astronomia.

Orientador: Dr. Fernando Virgilio Roig

Rio de Janeiro, RJ – Brasil
Março de 2020

C837 Costa de Souza, Alan
Paralelização em GPU de algoritmos de N corpos e aplicações a sistemas planetários extrassolares [Rio de Janeiro] 2020.
xx, 106 p. 29,7 cm: graf. il. tab.

Tese (doutorado) - Observatório Nacional - Rio de Janeiro, 2020.

1. problema de N corpos. 2. planetas extrassolares. 3. caos. 4. aceleração em GPU. I. Observatório Nacional. II. Título.

CDU 523.4

"PARALELIZAÇÃO EM GPU DE ALGORITMOS DE N CORPOS E APLICAÇÕES A SISTEMAS PLANETÁRIOS EXTRASSOLARES"

ALAN COSTA DE SOUZA

TESE SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ASTRONOMIA DO OBSERVATÓRIO NACIONAL COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM ASTRONOMIA.

Aprovada por:

Dr. Fernando Virgilio Roig – ON
(Orientador)

Prof. Dr. Adrián Rodríguez Colucci – OV/UFRJ

Prof. Dr. Daniel Gregorio Alfaro Vigo – DCC/UFRJ

Dr. Jorge Márcio Ferreira Carvano – ON

Dr. Roberto Vieira Martins – ON

RIO DE JANEIRO, RJ – BRASIL
30 DE MARÇO DE 2020

Aos meus pais e a Deus

Agradecimentos

- Em primeiro lugar ao meu orientador Dr. Fernando Roig, que confiou em mim para levar a cabo este projeto e pelos ensinamentos ao longo destes quatro anos.
- Aos meus pais por terem me dado a libertade e terem me impulsionado a atingir meus sonhos, pelo seu apoio e amor incondicional.
- À minha amiga Patrícia Zudio Lima, por ter me ajudado nos momentos difíceis durante a caminhada do doutorado.
- À minha orientadora de Mestrado Juliana Vianna Valério, por ter aceito o desafio de me orientar no mestrado e por ter me auxiliado nos contatos que me levaram a conseguir ingressar no programa de doutorado do Observatório Nacional.
- À minha orientadora de projeto final de graduação Luziane F. de Mendonça, pelos ensinamentos que me auxiliaram até este momento.
- Aos professores do programa de pós-graduação pelos ensinamentos e orientações.
- As laboratório LC3 da UFRJ pelo apoio ao conceder acesso as máquinas contendo as placas gráficas 1080 e k80 utilziadas no trabalho.
- Ao Observatório Nacional;
- À Capes pelo financiamento;

Alan Costa de Souza

PARALELIZAÇÃO EM GPU DE ALGORITMOS DE N CORPOS E APLICAÇÕES A SISTEMAS PLANETÁRIOS EXTRASSOLARES

RESUMO

Nesta tese, analisamos a viabilidade e implementamos a paralelização de algoritmos para resolver as equações planetárias de movimento em placas GPU. Com base na versão seqüencial do algoritmo `Helio`, que faz parte do software `Swifter`, fizemos a tradução do código de `Fortran` para `C` e, em seguida, sua paralelização em `CUDA`. Testamos três abordagens diferentes para paralelização: (i) o algoritmo simplético de Ruth para um sistema de N corpos, com N grande, (ii) o algoritmo `Helio` também para um sistema de muitos corpos, e (iii) o algoritmo `Helio` aplicado a uma grade de condições iniciais para um problema de poucos corpos. Concluímos que as versões paralelizadas do algoritmo de Ruth e da grade de condições iniciais são muito mais eficientes em termos computacionais do que suas contrapartes seriais. Por outro lado, o algoritmo `Helio` aplicado a um sistema de muitos corpos mostra um desempenho, em relação à versão serial, que depende das placas de GPU e CPUs usadas. Juntamente com a paralelização, implementamos o cálculo de alguns indicadores de caos, em particular o indicador MEGNO, e validamos nosso código reproduzindo resultados conhecidos da literatura. Por fim, aplicamos o código que paraleliza uma grade de condições iniciais ao estudo da estabilidade de três sistemas planetários extrassolares: Kepler-419, Kepler-59 e Kepler-46. O interesse nestes sistemas é porque seus parâmetros dinâmicos foram obtidos através da análise de variações de tempo de trânsito e queremos verificar se os sistemas são estáveis ou não dentro dos intervalos de incerteza de seus parâmetros.

Alan Costa de Souza

GPU PARALLELIZATION OF N-BODY ALGORITHMS AND APPLICATIONS TO EXTRASOLAR PLANETARY SYSTEMS

ABSTRACT

In this thesis, we analyzed the feasibility and implemented the parallelization of algorithms to solve the planetary equations of motion on GPU cards. Based on the sequential version of the `Helio` algorithm, which is part of the `Swifter` software, we made the translation of the code from `Fortran` to `C` and then its parallelization in `CUDA`. We tested three different approaches to parallelization: (i) the symplectic algorithm of Ruth for a system of N bodies, with N large, (ii) the `Helio` algorithm also for a system of many bodies, and (iii) the `Helio` algorithm applied to a grid of initial conditions for a few-body problem. We conclude that the parallelized versions of both the Ruth algorithm and the grid of initial conditions are much more computationally efficient than their serial counterparts. On the other hand, the `Helio` algorithm applied to a many body system shows a performance with respect to the serial version that depends on the GPU cards and CPUs used. Together with the parallelization, we implemented the calculation of some chaos indicators, in particular the MEGNO indicator, and we validate our code by reproducing well known results from the literature. Finally, we applied the code that parallelizes a grid of initial conditions to the study of the stability of three extra-solar planetary systems: Kepler-419, Kepler-59 and Kepler-46. The interest on these systems is because their dynamical parameters have been obtained through the analysis of transit timing variations, and we want to verify whether the systems are stable or not within the uncertainties of their parameters.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Na modelagem da formação e evolução dinâmica de sistemas planetários são aplicadas, principalmente, duas ferramentas numéricas distintas e complementares: algoritmos de N corpos e algoritmos hidrodinâmicos. Os primeiros procuram simular as interações binárias num sistema de corpos que são tratados, em geral, como entidades individuais (AARSETH, 2003). Essas interações são fundamentalmente forças advindas de diferentes processos físicos que ocorrem no sistema. Os segundos, por sua vez, procuram simular as propriedades físicas locais (densidade, pressão, velocidade, etc.) de um meio contínuo utilizando algum tipo de discretização do meio, seja uma aproximação por partículas (MONAGHAN, 1993) ou a aplicação de uma malha no espaço de fases (TEYSSIER, 2015). Enquanto os algoritmos de N corpos são utilizados para a modelagem dos processos de acreção de sólidos (poeira, *peebles*, planetesimais, etc.) e da subsequente evolução dinâmica a longo prazo dos corpos formados (planetas, satélites, asteroides, etc.), os algoritmos hidrodinâmicos são aplicados para modelar a evolução do gás na nebulosa protoplanetária e também, em outro contexto, para simular colisões binárias de corpos sólidos (MICHEL, 2015).

Ambos algoritmos apresentam dois tipos principais de limitações. A primeira se refere ao custo computacional envolvido, que depende, basicamente, da resolução espaço-temporal do sistema. Isto está vinculado, em particular, ao número N de corpos individuais que são considerados ou ao grau de discretização do meio contínuo (número de partículas ou resolução da malha). Por exemplo, em algoritmos de N corpos o número de operações cresce aproximadamente como N^2, o que impõe um limite ao número total de corpos que pode ser simulado dependendo da capacidade computacional disponível. Além disso, a resolução temporal, que depende do tamanho do passo da simulação e que é determinante na precisão do resultado, também é um fator de peso no custo computacional. Achar um compromisso entre precisão na representação do sistema, acurácia no resultado da simulação e custo computacional razoável pode representar um grande desafio. A necessidade de mitigar estas limitações tem levado à introdução de diferentes aproximações, usualmente ao preço de degradar a precisão quantitativa do resultado da

simulação. Entre as alternativas propostas, podemos mencionar: a utilização de passos de integração diferentes para cada corpo, a aplicação de interações de longo alcance médias, como no caso de métodos de árvore (BARNES e HUT, 1986) ou métodos de partícula-malha (EASTWOOD, 1975) que permitem tipicamente reduzir o número de operações para $N \log N$, a aplicação de algoritmos simplécticos de baixa ordem que preservam aproximadamente as integrais de movimento do sistema, como a energia (ver seção 2.1), ou a segregação do sistema em subsistemas autogravitantes e não autogravitantes. Ainda que com estas aproximações não se obtenham soluções quantitativamente precisas, elas se justificam na medida em que produzem resultados qualitativamente corretos.

O segundo tipo de limitação destes algoritmos é inerente a qualquer modelo numérico é diz respeito ao fato de que o resultado de cada simulação é específico do conjunto de parâmetros e condições de contorno utilizado, carecendo assim do carácter universal dos modelos analíticos. Isto significa que para estudar de forma completa as propriedades de um dado sistema é necessário repetir a mesma simulação um grande número de vezes, considerando diferentes parâmetros de entrada e condições de contorno a cada vez. Dependendo dos recursos computacionais disponíveis, a exploração de todo o espaço de parâmetros, mesmo de um sistema simples como um problema de poucos corpos, pode se tornar inviável.

A estas limitações se soma o fato de que, em muitos casos, os processos físicos que cada tipo de algoritmo visa simular não podem ser dissociados ou tratados de forma isolada. Assim, por exemplo, num cenário de formação planetária, o material sólido do disco protoplanetário interage com o gás durante boa parte do tempo de vida do disco. De forma semelhante, numa simulação de colisão binária entre asteroides, por exemplo, o resultado final não apenas depende da propagação de fraturas no material de cada corpo mas também da interação gravitacional entre os fragmentos formados. Isto leva à necessidade de combinar ou de integrar algoritmos de N corpos e hidrodinâmicos numa mesma simulação, o que demanda um custo computacional ainda maior, além das dificuldades inerentes à junção de esquemas de cômputo diversos (LEGA *et al.*, 2013).

1.1 Paralelização

Uma alternativa que não evita as limitações mencionadas acima, mas que permite contorná-las, consiste na aplicação de estratégias de paralelização dos algoritmos. A ideia é fazer com que as partes do código que são independentes entre si sejam executadas em forma simultânea ao invés de consecutiva. Um exemplo típico em que a paralelização pode ser aplicada é o caso de laços em que cada iteração não depende das outras. Outro exemplo é quando diferentes partes do código dependem de um único bloco de código mestre mas são independentes entre si. Não existe uma receita única para paralelizar um algoritmo e a estratégia a ser aplicada depende fortemente das propriedades do al-

goritmo, do problema sob estudo e dos recursos computacionais disponíveis (FOSTER, 1995). Em geral, paralelizar um código escrito em forma serial costuma ser mais difícil do que reescrever o código paralelizado do zero.

A paralelização tornou-se possível, inicialmente, a partir da disponibilidade de computadores configurados em *clusters* e, mais recentemente, a partir da disponibilidade de processadores com múltiplos núcleos e de computadores na nuvem. Existem diferentes paradigmas de paralelização, sendo os mais populares a paralelização por troca de mensagens entre computadores de uma mesma rede e a paralelização com uso de memória compartilhada. O primeiro é implementado através diferentes bibliotecas, entre as quais se destacam `MPI` (*Message Passing Interface*) e `PVM` (*Parallel Virtual Machine*). O segundo é implementado através de interfaces de programação de aplicativo (API), com é o caso de `OpenMP` (*Open Multi-Processing*). Estes paradigmas são compatíveis com as principais linguagens compiladas, o que facilita a sua utilização em `C`, `C++` ou `Fortran`, por exemplo.

Cabe aqui destacar a diferença entre uma simulação paralelizada e uma simulação distribuída. No primeiro caso, a ideia é que um único código execute diversas tarefas em forma simultânea, sendo cada tarefa atribuída a um processador ou núcleo diferente pelo próprio código. No segundo caso, várias réplicas de um mesmo código, normalmente serial, são executadas em forma simultânea, sendo cada réplica atribuída a um processador ou núcleo pelo usuário. Este último tipo de simulação pode ser utilizado, por exemplo, quando se deseja realizar a mesma simulação com parâmetros de entrada diferentes. Distribuir uma simulação não representa uma estratégia de paralelização, estritamente falando, ainda que na prática o resultado em termos de eficiência possa ser semelhante. Notemos, entretanto, que uma simulação distribuída poderia ser controlada por um código paralelizado: neste caso, as réplicas do código serial são atribuídas aos diferentes processadores ou núcleos não pelo usuário mas pelo próprio código. Este tipo de abordagem apresenta vantagens quando se trabalha em ambientes de *clusters* multiusuários, onde o número de processos individuais que cada usuário pode executar simultaneamente costuma ser limitado.

Um paradigma de paralelização que vem se tornando cada vez mais popular é a utilização de unidades de processamento gráfico, ou GPU. A GPU, popularmente conhecida como placa de vídeo, é a componente responsável pela renderização de imagens na tela do computador. Ao contrário da unidade central de processamento, ou CPU, que hoje em dia pode chegar a ter umas poucas dezenas de núcleos de processamento, uma GPU possui, tipicamente, centenas ou milhares de núcleos, o que a torna atrativa para a aplicação de estratégias de paralelização, particularmente tendo em vista o baixo preço de uma placa de vídeo se comparado ao custo de um conjunto de computadores com poder de paralelização semelhante. As primeiras GPU, lançadas pela empresa NVIDIA no final da década de 90, operavam apenas com aritmética de inteiros, que resultava suficiente para

o processamento dos pixeis de uma imagem, mas limitava a sua aplicação para operações mais complexas de cálculo. A partir de 2002, a empresa AMD/ATI, primeiro, e posteriormente NVIDIA, introduziram GPUs capazes de processar aritmética de ponto flotante em 16 bits, e atualmente em 32 e 64 bits.

O paradigma de programação em GPUs é o que se conhece como GPGPU (*General-Purpose computing on Graphics Processing Units*) e é realizado através de pipelines ou pequenos blocos de programa denominados de *shaders*, que originariamente eram utilizados apenas para renderizar o nível apropriado de brilho, sombras e cores de uma imagem, mas que atualmente foram estendidos para executar qualquer tipo de cálculo. Cada fabricante de GPU implementa seu próprio *shader*, específico para o tipo de formato de dados que a GPU suporta, mas que deve cumprir com certos requisitos estabelecidos pelos padrões `OpenGL` e `DirectX`. A princípio, qualquer linguagem de programação que permita que o código em execução na CPU pesquise valores de retorno em um *shader* da GPU, possibilita criar um ambiente de GPGPU. Atualmente, duas linguagens de programação para GPGPU dominam o mercado. Uma delas é `OpenCL` (*Open Computing Language*) que é um padrão de programação aberto baseado em `C` e `C++`, desenvolvido e mantido pelo Khronos Group. `OpenCL` especifica uma linguagem e um conjunto de APIs para computação paralela e permite a execução de programas em plataformas heterogêneas que incluam CPUs, GPUs e outros processadores ou aceleradores de hardware. A outra linguagem é `CUDA` (*Compute Unified Device Architecture*), uma plataforma de computação paralela proprietária, desenvolvida e mantida pela NVIDIA especificamente para a sua linha de GPUs, e que pode interagir com `C`, `C++` e `Fortran`. A principal vantagem de `OpenCL` é a sua portabilidade, podendo ser utilizado inclusive nas GPU da NVIDIA. Entretanto, `CUDA` roda de forma mais rápida e eficiente nas GPUs dessa marca (DU *et al.*, 2012). Outras linguagens de programação GPGPU proprietárias são `FireStream`, para placas ATI Radeon da AMD, e `DirectCompute` da Microsoft.

É importante destacar que a frequência de trabalho de um núcleo de GPU é, tipicamente, menor do que a frequência de uma CPU ($\sim$ 500 MHz vs. $\sim$ 2 GHz, respetivamente). As CPUs mais rápidas do mercado conseguem hoje atingir $5,5$ GHz (8.6 GHz usando *overclocking*), enquanto que os núcleos de GPU mais rápidos não superam $1,5$ GHz, como é o caso da arquitetura Pascal da NVIDIA. Assim, as GPUs acabam obtendo vantagem acima das CPUs apenas por conta do número muito maior de núcleos disponíveis. Isso significa que a paralelização em GPU deve ser planejada de forma a maximizar os uso dos recursos disponíveis na placa para resultar realmente eficiente.

1.2 Contexto científico

A determinação dos parâmetros orbitais e físicos dos exoplanetas é uma das tarefas mais desafiadoras da astronomia moderna. A grande maioria desses mundos é descoberto

por meios indiretos, principalmente a análise da curva de velocidade radial (LOVIS e FISCHER, 2010) e/ou a observação de trânsitos na curva de luz (SEAGER e MALLÉN-ORNELAS, 2003) da estrela hospedeira. Isso significa que a caracterização dos sistemas fica limitada por três problemas principais: o tipo de parâmetros que as observações podem fornecer, a qualidade dos dados observacionais, e a limitação dos métodos de inversão das observações (FISCHER *et al.*, 2016; CAMERON, 2016). Assim, para a maioria dos exoplanetas, não é possível determinar um conjunto completo de parâmetros e alguns deles ficam definidos apenas em termos de limites superiores ou inferiores, ou ainda devem ser assumidos. No caso de sistemas multiplanetários, essas limitações restringem o desenvolvimento de estudos detalhados da dinâmica global que poderiam ajudar a abordar melhor o problema da origem e posterior evolução desses sistemas, bem como entender suas diferenças e semelhanças. Atualmente, existem aproximadamente 4302 exoplanetas catalogados em 3177 sistemas diferentes, sendo cerca de 703 deles sistemas multiplanetários (fonte Extrasolar Planets Encyclopaedia, 2020) A combinação de diferentes técnicas de observação ajuda a definir melhor alguns parâmetros estelares e planetários (CHARBONNEAU *et al.*, 2000), mas na maioria dos casos os sistemas não estão totalmente caracterizados. Além disso, em alguns casos, a caracterização não é viável porque os melhores parâmetros de ajuste do problema de inversão produzem uma configuração dinamicamente instável (FERRAZ-MELLO *et al.*, 2005). Isto torna os estudos dinâmicos uma ferramenta fundamental para complementar as observações e conseguir uma melhor caracterização dos sistemas. Um dos parâmetros mais críticos é a massa dos planetas, que hoje em dia só pode ser estimada com precisão razoável combinando velocidades radiais e trânsitos, ou pelo método de variações de tempo de trânsito (MIRALDA-ESCUDÉ, 2002; AGOL *et al.*, 2005; HOLMAN e MURRAY, 2005). Assim, dispor de um código de N corpos que possibilite explorar em forma sistemática, detalhada e eficiente o espaço de parâmetros de sistemas de exoplanetas, analisando a estabilidade dos mesmos através de ferramentas específicas, torna-se uma necessidade que motiva o desenvolvimento deste trabalho de tese.

1.3 Objetivos deste trabalho

O nosso objetivo é a paralelização de um algoritmo de N corpos em GPU, utilizando `CUDA`, e a sua aplicação para o estudo da estabilidade dinâmica de sistemas planetários extrassolares. A nossa escolha pela linguagem `CUDA` dá-se em função da nossa facilidade de acesso a placas de video da marca NVIDIA, bem como a sua facilidade de implementação a partir da linguagem `C`, que dominamos amplamente.

A ideia de paralelizar algoritmos de N corpos com `CUDA` não é nova e podemos encontrar facilmente na internet diversos exemplos deste tipo de códigos. De particular interesse para nós é o código `GENGA` (*Gravitational Encounters with GPU Acceleration*; GRIMM

e STADEL, 2014; https://bitbucket.org/sigrimm/genga/src/default/), que é uma versão paralelizada para GPU do popular código de dinâmica planetária `Mercury` (CHAMBERS, 1991, 2012; https://gemelli.spacescience.org/~hahnjm/software.html; reporte de *bug* em https://www.astro.keele.ac.uk/~dra/mercury/). Este código é, basicamente, um algoritmo de N corpos simpléctico de ordem dois, do tipo *leap-frog*, em que as órbitas dos corpos são avançadas a cada passo de integração ao longo de uma trajetória de referência kepleriana e as perturbações gravitacionais são adicionadas na forma de impulsos nas velocidades no início e no final de cada passo. Assim como `Mercury`, `GENGA` é capaz de manipular de forma eficiente encontros próximos entre os corpos massivos, requisito fundamental para o estudo de, por exemplo, processos de acreção em discos protoplanetários. A paralelização em GPU possibilita que `GENGA` simule sistemas de milhares de corpos totalmente autogravitantes corpos que sofrem a ação gravitacional uns dos outros), o que demandaria um custo computacional proibitivo com um código puramente serial como `Mercury`. `GENGA` também incorpora um modo de multisimulação, que permite executar em forma simultânea a mesma simulação com diferentes parâmetros de entrada. `GENGA` permite ainda incorporar nas simulações diversos processos físicos, como arrasto gasoso, migração planetária, efeito Yarkovsky (VOKROUHLICKÝ *et al.*, 2015), etc.

Também de interesse para o nosso trabalho é o código `cuSwift`, desenvolvido por HELLMICH (2017), que é uma versão paralelizada para GPU do clássico código de dinâmica planetária `Swift` (LEVISON e DUNCAN, 1994; https://www.boulder.swri.edu/~hal/swift.html). Esta versão paralelizada inclui três *drivers* que implementam diferentes esquemas de integração numérica: (i) o algoritmo simpléctico de N corpos desenvolvido por WISDOM e HOLMAN (1990), conhecido como `MVS` (*Mixed Variable Symplectic*) ou `WHM` (*Wisdom & Holman Mapping*), (ii) uma versão regularizada do algoritmo anterior que permite manipular encontros próximos entre partículas de teste e corpos massivos, desenvolvida por LEVISON e DUNCAN (1994) e conhecida como `RMVS` (*Regularized Mixed Variable Symplectic*), e (iii) o integrador `RA15` (EVERHART, 1985), que é um método de Runge-Kutta implícito de ordem 15 que usa espaçamentos de Gauss-Radau, com passo adaptativo. O foco de `cuSwift` está na simulação de órbitas de asteroides e cometas, com ênfase no cálculo do efeito Yarkovsky e sua aplicação para populações de pequenos corpos como os troianos de Júpiter (HELLMICH *et al.*, 2019).

Entretanto, nenhum destes códigos implementa o cálculo de estimadores de caos ou de estabilidade, que são grandezas fundamentais para mapear a estrutura dinâmica do espaço de fases de um sistema planetário. Além disso e até onde é do nosso conhecimento, `cuSwift`, ao contrário de `GENGA`, não está disponível publicamente. Nossa meta, portanto, é produzir um código simpléctico semelhante, porém independente, que inclua o cômputo de estimadores de caos, particularmente aqueles derivados do conceito de exponentes característicos de Lyapunov (BENETTIN *et al.*, 1980).

No capítulo 2, apresentamos as bases teóricas para o nosso trabalho, discutimos os algoritmos simplécticos para o problema de N corpos e a determinação de estimadores de caos. No capítulo 3 explanamos a metodologia utilizada no trabalho, com ênfase nos detalhes sobre a programação em CUDA e os diferentes testes de desempenho e validação de código realizados. No capítulo 4 apresentamos diferentes aplicações do nosso código ao estudo de alguns sistemas extrassolares. Os resultados dessas aplicações foram publicados em SAAD-OLIVERA *et al.* (2019, 2020) e COSTA DE SOUZA *et al.* (submetido). Finalmente, o capítulo 5 é dedicado às conclusões e perspectivas futuras.

Capítulo 2

Fundamentos

Neste capítulo, apresentamos os fundamentos e as bases necessárias para atingir o objetivo proposto. Para tal, partimos de cinco preceptos que irão nortear o desenvolvimento do nosso trabalho:

1. O algoritmo de integração numérica deve ser simpléctico, de baixa ordem, para permitir simular em forma rápida a evolução do sistema por intervalos de tempo longos, preservando a energia e garantindo que a solução será qualitativamente bem comportada.

2. O modelo de N corpos deve assumir uma estrutura de sistema planetário, com uma massa central dominante, e deve permitir simular diversas configurações orbitais, desde órbitas quase circulares e quase coplanares até órbitas muito excêntricas e inclinadas.

3. O código deve calcular estimadores de caos baseados tanto na ideia de coeficiente de difusão caótica quanto na ideia de exponente característico de Lyapunov, em particular o MEGNO (*Mean Exponential Growth of Nearby Orbits*). Isto último implica que o algoritmo deve resolver as equações variacionais do problema em forma concomitante com as equações de movimento.

4. O código deve permitir a simulação simultânea de um grande número de condições iniciais, que permitam mapear o espaço de fases do sistema de forma detalhada, valendo-se para isto da paralelização em `CUDA`.

5. O código deve ser validado em diferentes aplicações a partir de resultados conhecidos da literatura.

Abordaremos a seguir os primeiros três pontos, deixando os dois últimos para o capítulo seguinte.

2.1 Algoritmos simplécticos

Um integrador simpléctico é um algoritmo desenvolvido especificamente para resolver as equações de movimento de um sistema hamiltoniano, isto é

$$\frac{dr_i}{dt} = \frac{\partial H}{\partial p_i}, \qquad \frac{dp_i}{dt} = -\frac{\partial H}{\partial r_i}, \qquad i = 1, \ldots, n$$

onde o hamiltoniano $H(\mathbf{r}, \mathbf{p})$ é uma função escalar e $\mathbf{r}, \mathbf{p} \in \mathbb{R}^n$ são os vetores coordenada e momento canônicos, respectivamente. A dimensão destes vetores, n, determina o número de graus de liberdade do sistema. Para entender melhor como um algoritmo simpléctico funciona, vamos relembrar o conceito de transformação canônica. Um desenvolvimento mais detalhado deste tópico pode ser encontrado no livro de LANCZOS (1986).

Uma transformação de variáveis

$$(\mathbf{r}, \mathbf{p}) \to (\mathbf{R}, \mathbf{P})$$

se diz canônica ou simpléctica se a sua matriz Jacobiana

$$\mathcal{M} = \begin{pmatrix} \frac{\partial R_1}{\partial r_1} & \cdots & \frac{\partial R_n}{\partial r_1} & \frac{\partial P_1}{\partial r_1} & \cdots & \frac{\partial P_n}{\partial r_1} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ \frac{\partial R_1}{\partial r_n} & \cdots & \frac{\partial R_n}{\partial r_n} & \frac{\partial P_1}{\partial r_n} & \cdots & \frac{\partial P_n}{\partial r_n} \\ \frac{\partial R_1}{\partial p_1} & \cdots & \frac{\partial R_n}{\partial p_1} & \frac{\partial P_1}{\partial p_1} & \cdots & \frac{\partial P_n}{\partial p_1} \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial R_1}{\partial p_n} & \cdots & \frac{\partial R_n}{\partial p_n} & \frac{\partial P_1}{\partial p_n} & \cdots & \frac{\partial P_n}{\partial p_n} \end{pmatrix}$$

cumpre com a propriedade

$$\mathcal{M}^{\mathrm{T}} \mathcal{J} \mathcal{M} = \mathcal{J}$$

onde $\mathcal{J}$ é a matriz simplética antissimétrica

$$\mathcal{J} = \begin{pmatrix} \mathbf{0} & \mathcal{I} \\ -\mathcal{I} & \mathbf{0} \end{pmatrix}$$

sendo $\mathbf{0}, \mathcal{I}$ são as matrizes $n \times n$ nula e identidade, respectivamente. Do anterior se conclui que

$$(\det \mathcal{M})^2 = 1$$

e é possível mostrar que, mais precisamente, $\det \mathcal{M} = 1$. Isto significa que a transformação canônica preserva o volume no espaço de fases definido pelo produto externo $d\mathbf{r} \wedge d\mathbf{p}$. A transformação canônica é inversível e preserva a forma das equações de movimento para

a nova Hamiltoniana $H'(\mathbf{R},\mathbf{P}) = H(\mathbf{r},\mathbf{p})$

$$\frac{dR_i}{dt} = \frac{\partial H'}{\partial P_i}, \qquad \frac{dP_i}{dt} = -\frac{\partial H'}{\partial R_i}, \qquad i = 1,\ldots,n$$

É possível mostrar que a transformação canônica é determinada pela relação

$$\sum_{i=1}^{n} p_i dr_i = \sum_{i=1}^{n} P_i dR_i + dS \tag{2.1}$$

onde dS é uma diferencial exata. A função $S(\mathbf{r},\mathbf{R})$ é chamada de função geratriz da transformação e a transformação fica definida em forma implícita através das equações

$$p_i = \frac{\partial S}{\partial r_i}, \qquad P_i = -\frac{\partial S}{\partial R_i}.$$

Vamos supor agora que a função geratriz depende de um parâmetro γ tal que a transformação

$$(\mathbf{R},\mathbf{P}) \rightarrow (\mathbf{r},\mathbf{p})$$

é gerada pela função $S(\mathbf{R},\mathbf{r},\gamma)$, enquanto que a transformação

$$(\mathbf{R},\mathbf{P}) \rightarrow (\mathbf{r}+d\mathbf{r},\ \mathbf{p}+d\mathbf{p})$$

é gerada pela função $S(\mathbf{R},\ \mathbf{r}+d\mathbf{r},\ \gamma+d\gamma)$. Então, a transformação infinitesimal

$$(\mathbf{r},\mathbf{p}) \rightarrow (\mathbf{r}+d\mathbf{r},\ \mathbf{p}+d\mathbf{p})$$

também é canônica, e é possível mostrar que esta transformação está definida em forma explícita pelas equações

$$dr_i = \frac{\partial B}{\partial p_i} d\gamma, \qquad dp_i = -\frac{\partial B}{\partial r_i} d\gamma$$

onde a função geratriz B é $B(\mathbf{r},\mathbf{p},\gamma) = -\partial S/\partial\gamma$. Identificando o parâmetro $\gamma \equiv t$, resulta evidente que o fluxo $\mathbf{r}(t),\mathbf{p}(t)$ de um sistema hamiltoniano é representado por uma sucessão de transformações canônicas infinitesimais, que tem a hamiltoniana H como função geratriz.

Consideremos agora duas funções escalares $f(\mathbf{r},\mathbf{p}), g(\mathbf{r},\mathbf{p})$ e definamos a derivada de Lie (ŚLEBODZIŃSKI, 1931; WILLMORE, 1960) da função f gerada pela função g, $\mathcal{D}_g f$, como o parêntesis de Poisson

$$\mathcal{D}_g f = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial r_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial r_i} \right) \equiv \{f,g\}$$

É imediato verificar que as equações de Hamilton podem ser escritas em termos do ope-

rador derivada de Lie, $\mathcal{D}_H$, como

$$\frac{d\mathbf{z}}{dt} = \mathcal{D}_H \mathbf{z} \tag{2.2}$$

sendo $\mathbf{z} = (\mathbf{r}, \mathbf{p}) \in \mathbb{R}^{2n}$. Esta equação é semelhante à equação linear de primeira ordem

$$\frac{dy}{dx} = Ay$$

cuja solução é $y(x) = \exp((x - x_0)A)\, y_0$. Por analogia, a solução da Equação (2.2) pode ser representada como

$$\mathbf{z}(t) = \exp(t - t_0)\mathcal{D}_H\, \mathbf{z}_0$$

onde $\exp \tau\mathcal{D}_H$ é o operador série de Lie gerado por H

$$\exp \tau\mathcal{D}_H = \sum_{j=0}^{\infty} \frac{1}{k!}\tau^j \mathcal{D}_H^j$$

sendo

$$\mathcal{D}_H^j\, \mathbf{z} = \underbrace{\mathcal{D}_H\left(\mathcal{D}_H\left(\cdots\left(\mathcal{D}_H\, \mathbf{z}\right)\right)\cdots\right)}_{k \text{ vezes}} \equiv \left.\frac{d^j \mathbf{z}}{dt^j}\right|_{t_0}$$

e $\mathcal{D}_H^0\, \mathbf{z} = \mathbf{z}_0$ (STERNBERG, 2009). Logo, este operador série de Lie é, por construção, uma transformação canônica ou simpléctica, que mapeia as coordenadas e momentos num instante t_0 nas coordenadas e momentos num instante $t = t_0 + \tau$, preservando a hamiltoniana do sistema.

A limitação óbvia do operador série de Lie é que, na prática, resulta impossível calcular os infinitos termos da série. O truncamento da mesma em qualquer ordem, produz uma solução aproximada, que pode ser tão precisa quanto for desejado dependendo da ordem de truncamento, mas que não é simpléctica e, portanto, não preserva a hamiltoniana. A maioria dos algoritmos clássicos de integração de equações diferenciais ordinárias, como os métodos de Runge-Kutta e suas variantes, não são simplécticos pois se baseiam em encontrar uma fórmula que aproxime a série de Lie da solução até uma certa ordem k, que define a ordem do método (HAIRER *et al.* 1993; BUTCHER, 2008). Como consequência, estes métodos em geral não preservam a hamiltoniana e introduzem variações sistemáticas na energia do sistema, que acaba se comportando no longo prazo como um sistema dissipativo. A ideia para a construção de um algoritmo simpléctico consiste em achar uma aproximação da série de Lie da solução também até uma certa ordem, mas que seja ela mesma uma transformação canônica. Dessa maneira, a solução estará truncada efetivamente em alguma ordem, mas pelo fato de representar uma transformação canônica deverá preservar uma certa grandeza, $\overline{H}$. O ponto chave, então, é construir esta solução de forma que a transformação canônica possa ser calculada de forma explícita e a grandeza $\overline{H}$ seja tão próxima quanto desejarmos do hamiltoniano original H.

Para abordar esta questão, vamos separar o problema em duas partes. Em primeiro

lugar, vamos considerar o caso em que o hamiltoniano $H(\mathbf{r},\mathbf{p})$ é integrável. Isto significa que a solução $\mathbf{z}(t)$ pode ser exprimida explicitamente em termos de funções analíticas e esta solução deve ser, por construção, idêntica à série de Lie $\exp\tau\mathcal{D}_H\mathbf{z}$. Logo, o cômputo da série de Lie pode ser substituído diretamente pela solução analítica. Mais ainda, a integrabilidade do sistema garante que existe uma transformação canônica $(\mathbf{r},\mathbf{p})\rightarrow(\boldsymbol{\theta},\mathbf{J})$ tal que $H(\mathbf{r},\mathbf{p})=H'(\mathbf{J})$ (GOLDSTEIN, 1980). Neste caso, a aplicação às variáveis $(\boldsymbol{\theta},\mathbf{J})$ do operador série de Lie gerado por H' se reduz a

$$\begin{aligned}\exp\tau\mathcal{D}_{H'}\theta_i &= \theta_{i,0}+\tau\left\{\theta_i,H'\right\}+\frac{\tau^2}{2}\left\{\left\{\theta_i,H'\right\},H'\right\}+\cdots\\ &= \theta_{i,0}+\tau\frac{\partial H'}{\partial J_i}\\ \exp\tau\mathcal{D}_{H'}J_i &= J_{i,0}+\tau\left\{J_i,H'\right\}+\frac{\tau^2}{2}\left\{\left\{J_i,H'\right\},H'\right\}+\cdots\\ &= J_{i,0}\end{aligned}$$

e um resultado semelhante pode ser obtido se consideramos um hamiltoniano $H''(\boldsymbol{\theta})$. Em outras palavras, quando o hamiltoniano depende apenas dos momentos ou apenas das coordenadas, a série de Lie correspondente possui termos somente até $\mathcal{O}(\tau)$ e pode ser calculada explicitamente, pois

$$\exp\tau\mathcal{D}_{H'}=1+\tau\mathcal{D}_{H'},\qquad\qquad \mathcal{D}^j_{H'}=0,\qquad j\geqslant 2$$

Em segundo lugar, vamos considerar o caso em que o hamiltoniano do sistema pode ser separado em dois termos, $H=H_0+H_1$. A partir das propriedades do operador derivada de Lie e de suas regras de comutação, definidas pela identidade de Baker-Campbell-Hausdorff (ROSSMANN, 2002; HALL, 2015), o operador série de Lie para este hamiltoniano pode ser escrito em forma aproximada como

$$\begin{aligned}\exp\tau\mathcal{D}_H &= \exp\tau\mathcal{D}_{(H_0+H_1)}\\ &= \exp\tau\left(\mathcal{D}_{H_0}+\mathcal{D}_{H_1}\right)\\ &= \prod_{i=1}^{k}\exp\alpha_i\tau\mathcal{D}_{H_0}\exp\beta_i\tau\mathcal{D}_{H_1}+\mathcal{O}\left(\tau^{k+1}\right)\\ &= \exp\tau\mathcal{D}_{\overline{H}}+\mathcal{O}\left(\tau^{k+1}\right)\end{aligned}\tag{2.3}$$

(FOREST e RUTH, 1990; YOSHIDA 1990; HAIRER *et al.* 2006), onde k define a ordem de truncamento da aproximação, α_i,β_i são coeficientes constantes a serem determinados e

$$\overline{H}=H_0+H_1+\sum_{j=k}^{\infty}\tau^j\Phi_j\left(H_0,H_1\right)\tag{2.4}$$

sendo cada coeficiente desta série uma combinação linear de parêntesis de Poisson da forma

$$\Phi_j\left(H_0, H_1\right) \propto \{\cdots\{\{H_0, H_1\} \underbrace{\cdots H_0 \cdots\}\cdots H_1 \cdots\}}_{j-1 \text{ vezes}}$$

O operador

$$\exp \tau \mathcal{D}_{\overline{H}} = \prod_{i=1}^{k} \exp \alpha_i \tau \mathcal{D}_{H_0} \exp \beta_i \tau \mathcal{D}_{H_1} \tag{2.5}$$

constitui um integrador simpléctico de ordem k, representado pela aplicação consecutiva de transformações canônicas geradas alternativamente por H_0 e H_1, que após um passo de integração τ produz uma solução com erro de truncamento da ordem de τ^{k+1} e que preserva uma grandeza $\overline{H}$ que pode ser tão próxima de H quanto menor for τ. Esta grandeza costuma ser chamada de hamiltoniano sub-rogante.

Assim, uma possível estratégia para construir um integrador simpléctico consiste em separar o hamiltoniano do problema em uma série finita de termos $H = \sum_{i=0}^{m} H_i$ de maneira que cada um desses termos tratado de forma isolada seja integrável analiticamente ou dependa apenas das coordenadas ou apenas dos momentos (WISDOM, 1982; CANDY e ROZMUS, 1991). Desta forma, as séries de Lie que aparecem na Equação (2.5) podem ser calculadas em forma explícita e a solução do sistema ao longo de um passo τ é aproximada pela aplicação consecutiva das soluções geradas individualmente por cada uma das partes do hamiltoniano. Um exemplo deste tipo de estratégia é o caso em que o hamiltoniano do sistema pode ser representado como a soma de um termo de energia cinética, que só depende das velocidades, mais um termo de energia potencial que só depende das posições, isto é, $H(\mathbf{r}, \mathbf{p}) = T(\mathbf{p}) + U(\mathbf{r})$. Outro exemplo, que é do nosso particular interesse, é o caso do problema de N corpos planetário, em que há uma massa central muito maior que as restantes. Veremos na próxima seção que nesse caso o hamiltoniano pode ser usualmente escrito como $H(\mathbf{r}, \mathbf{p}) = K(\mathbf{r}, \mathbf{p}) + P(\mathbf{r})$, onde K representa um problema de 2 corpos cuja solução são órbitas keplerianas ao redor da massa central e P representa as perturbações gravitacionais mútuas, que dependem somente das posições dos corpos.

2.1.1 Integrador simpléctico de primeira ordem

Para determinar os coeficientes α_1, β_1 do integrador simpléctico no caso $k = 1$, procede-se de forma que a solução do integrador seja comparável à série de Lie original truncada até ordem 1, ou seja

$$\begin{aligned}
\exp \tau \mathcal{D}_H &= \exp \alpha_1 \tau \mathcal{D}_{H_0} \exp \beta_1 \tau \mathcal{D}_{H_1} + \mathcal{O}(\tau^2) \\
1 + \tau \mathcal{D}_H + \mathcal{O}(\tau^2) &= \left(1 + \alpha_1 \tau \mathcal{D}_{H_0} + \mathcal{O}(\tau^2)\right)\left(1 + \beta_1 \tau \mathcal{D}_{H_1} + \mathcal{O}(\tau^2)\right) + \mathcal{O}(\tau^2) \\
1 + \tau \mathcal{D}_{H_0} + \tau \mathcal{D}_{H_1} + \mathcal{O}(\tau^2) &= 1 + \alpha_1 \tau \mathcal{D}_{H_0} + \beta_1 \tau \mathcal{D}_{H_1} + \mathcal{O}(\tau^2)
\end{aligned}$$

e comparando termos da mesma ordem resulta imediato que $\alpha_1 = \beta_1 = 1$. O integrador de primeira ordem

$$\exp \tau \mathcal{D}_{\overline{H}} = \exp \tau \mathcal{D}_{H_0} \exp \tau \mathcal{D}_{H_1}$$

preserva o hamiltoniano sub-rogante

$$\overline{H} = H_0 + H_1 + \frac{\tau}{2} \{H_0, H_1\} + \mathcal{O}(\tau^2) \tag{2.6}$$

No caso em que $H = H_0(\mathbf{p}) + H_1(\mathbf{r})$ temos que

$$\exp \tau \mathcal{D}_{H_0} \exp \tau \mathcal{D}_{H_1} = (1 + \tau \mathcal{D}_{H_0}) (1 + \tau \mathcal{D}_{H_1})$$

que constitui o que se conhece como integrador de Euler semi-implícito.

2.1.2 Integrador simpléctico de segunda ordem

Tomando agora $k = 2$ e procedendo de forma semelhante, temos por um lado

$$\begin{aligned}
\exp \tau \mathcal{D}_H &= 1 + \tau \mathcal{D}_H + \frac{\tau^2}{2} \mathcal{D}_H^2 + \mathcal{O}(\tau^3) \\
&= 1 + \tau \mathcal{D}_{H_0} + \tau \mathcal{D}_{H_1} + \frac{\tau^2}{2} \mathcal{D}_{H_0}^2 + \frac{\tau^2}{2} \mathcal{D}_{H_0} \mathcal{D}_{H_1} + \frac{\tau^2}{2} \mathcal{D}_{H_1} \mathcal{D}_{H_0} + \frac{\tau^2}{2} \mathcal{D}_{H_1}^2 \\
&\quad + \mathcal{O}(\tau^3)
\end{aligned}$$

e por outro lado

$$\begin{aligned}
\prod_{i=1}^{2} \exp \alpha_i \tau \mathcal{D}_{H_0} \exp \beta_i \tau \mathcal{D}_{H_1} + \mathcal{O}(\tau^3) &= \prod_{i=1}^{2} \left(1 + \alpha_i \tau \mathcal{D}_{H_0} + \frac{\alpha_i^2 \tau^2}{2} \mathcal{D}_{H_0}^2 + \mathcal{O}(\tau^3) \right) \\
&\quad \times \left(1 + \beta_i \tau \mathcal{D}_{H_1} + \frac{\beta_i^2 \tau^2}{2} \mathcal{D}_{H_1}^2 + \mathcal{O}(\tau^3) \right) \\
&\quad + \mathcal{O}(\tau^3) \\
&= 1 + (\alpha_1 + \alpha_2) \tau \mathcal{D}_{H_0} + (\beta_1 + \beta_2) \tau \mathcal{D}_{H_1} \\
&\quad + (\alpha_1 + \alpha_2)^2 \frac{\tau^2}{2} \mathcal{D}_{H_0}^2 + (\beta_1 + \beta_2)^2 \frac{\tau^2}{2} \mathcal{D}_{H_1}^2 \\
&\quad + (\alpha_1 \beta_1 + \alpha_2 \beta_2 + \beta_1 \alpha_2) \tau^2 \mathcal{D}_{H_0} \mathcal{D}_{H_1} \\
&\quad + \alpha_1 \beta_2 \tau^2 \mathcal{D}_{H_1} \mathcal{D}_{H_0} \\
&\quad + \mathcal{O}(\tau^3)
\end{aligned}$$

Comparando termos de igual ordem, obtemos

$$\begin{aligned}
\alpha_1 + \alpha_2 &= 1 \\
(\alpha_1 + \alpha_2)^2 &= 1 \\
\beta_1 + \beta_2 &= 1 \\
(\beta_1 + \beta_2)^2 &= 1 \\
\alpha_1\beta_1 + \alpha_2\beta_2 + \alpha_2\beta_1 &= \frac{1}{2} \\
\alpha_1\beta_2 &= \frac{1}{2}
\end{aligned}$$

onde a segunda e quarta equações são redundantes e se reduzem à primeira e terceira, respectivamente. Uma das possíveis soluções para este sistema é

$$\alpha_1 = 1, \quad \alpha_2 = 0, \quad \beta_1 = \beta_2 = \frac{1}{2}$$

que resulta no integrador simpléctico de segunda ordem

$$\exp \tau\mathcal{D}_{\overline{H}} = \exp \frac{\tau}{2}\mathcal{D}_{H_1} \exp \tau\mathcal{D}_{H_0} \exp \frac{\tau}{2}\mathcal{D}_{H_1} \tag{2.7}$$

e que preserva o hamiltoniano sub-rogante

$$\overline{H} = H_0 + H_1 - \frac{\tau^2}{12}\left\{\left\{H_0, H_1\right\}, H_0\right\} - \frac{\tau^2}{24}\left\{\left\{H_0, H_1\right\}, H_1\right\} + \mathcal{O}(\tau^4) \tag{2.8}$$

No caso em que $H = H_0(\mathbf{p}) + H_1(\mathbf{r})$ temos que

$$\exp \frac{\tau}{2}\mathcal{D}_{H_1} \exp \tau\mathcal{D}_{H_0} \exp \frac{\tau}{2}\mathcal{D}_{H_1} = \left(1 + \frac{\tau}{2}\mathcal{D}_{H_1}\right)\left(1 + \tau\mathcal{D}_{H_0}\right)\left(1 + \frac{\tau}{2}\mathcal{D}_{H_1}\right)$$

que constitui o que se conhece como integrador *leap-frog*. Notemos que outra possível solução para os coeficientes é

$$\alpha_1 = \alpha_2 = \frac{1}{2}, \quad \beta_1 = 0, \quad \beta_2 = 1$$

que resulta no integrador simpléctico

$$\exp \tau\mathcal{D}_{\overline{H}} = \exp \frac{\tau}{2}\mathcal{D}_{H_0} \exp \tau\mathcal{D}_{H_1} \exp \frac{\tau}{2}\mathcal{D}_{H_0} \tag{2.9}$$

A preferência pelo integrador (2.7) sobre o integrador (2.9) dependerá, fundamentalmente, do grau de dificuldade envolvido na determinação da solução que cada série de Lie representa. Voltaremos sobre isto na seção 2.2.1.

2.1.3 Vantagens e limitações

A vantagem óbvia dos algoritmos simplécticos reside precisamente no fato de preservarem em forma aproximada a energia do sistema, mesmo no caso de integradores de baixa ordem como os apresentados acima. Isto permite realizar simulações numéricas por longos intervalos de tempo (bilhões de anos, por exemplo, no caso do Sistema Solar), a um custo computacional baixo quando comparado ao uso de algoritmos não simplécticos, que precisariam recorrer a soluções de alta ordem para atingir um nível de conservação da energia comparável (KINOSHITA *et al.* 1991). A partir da análise das Equações (2.4), (2.6) e (2.8) podemos inferir outras vantagens e limitações relevantes dos algoritmos simplécticos.

Uma vantagem torna-se evidente quando se considera um sistema hamiltoniano em que $H_1 \ll H_0$, ou seja, quando H_1 pode ser interpretado como uma pequena perturbação sobre H_0. Assumindo que $H_1 \approx \mathcal{O}(\varepsilon)$, $\varepsilon \ll 1$, a diferença entre o hamiltoniano original e o hamiltoniano sub-rogante para um integrador simpléctico de ordem k, com passo τ, resultará ser $\left|\overline{H} - H\right| \approx \mathcal{O}(\tau^k \varepsilon)$. Desta forma, é possível utilizar um tamanho de passo $\varepsilon^{-1/k}$ vezes maior, mantendo o mesmo nível conservação da energia que no caso em que $H_1 \approx \mathcal{O}(1)$ (CHAMBERS e MURISON, 2000). Como veremos mais adiante, este é um ponto chave na aplicação de integradores simplécticos ao problema de N corpos planetário, que permite a utilização de passos de integração relativamente grandes quando comparados aos passos requeridos por integradores não simplécticos de mesma ordem.

Uma outra vantagem está no fato de que, para integradores simplécticos de ordem par, a diferença entre o hamiltoniano original e o sub-rogante depende apenas de potências pares de τ. Isto implica que $\overline{H}$ fica inalterado se substituir $\tau \to -\tau$, de forma que a solução de um integrador simpléctico de ordem par é reversível no tempo. Já integradores de ordem ímpar não são reversíveis.

Por outro lado, como o hamiltoniano $\overline{H}$ depende do tamanho do passo τ, isto faz com que o passo não possa ser mudado ao longo da integração, sob pena de mudar a física do sistema sub-rogante e produzir soluções espúrias. Esta é uma séria limitação dos algoritmos simplécticos, que resulta crucial quando alguma das partes do hamiltoniano, por exemplo H_1, se torna divergente ao longo da integração. Em algoritmos não simplécticos, essa divergência costuma ser compensada diretamente com uma diminuição do tamanho do passo, de forma a manter o erro de truncamento da solução limitado (PRESS *et al.*, 1992). Mas em algoritmos simplécticos esta estratégia não pode ser aplicada. O problema é particularmente relevante em simulações de N corpos, em que os termos de interação $\to \infty$ quando dois corpos sofrem um encontro próximo ou colisão, ou quando uma órbita se torna muito excêntrica e a velocidade aumenta significativamente durante a passagem pelo pericentro.

Existem na literatura algumas alternativas para contornar esta limitação, entre as

quais cabem destacar: o uso de passos de integração individuais para cada corpo (SAHA e TREMAINE, 1994), a regularização do tempo (MIKKOLA e WIEGERT, 2002; PETIT *et al.*, 2019), e a partição do potencial (CHAMBERS, 1991; DUNCAN *et al.*, 1998). Este último método, em particular, baseia-se na ideia de dividir e reagrupar os termos do hamiltoniano da seguinte forma

$$\begin{aligned} H &= K + P \\ &= K + \underbrace{(1-w)P} + \underbrace{wP} \\ &= \underbrace{K + \quad P_1} \quad + \underbrace{\quad P_0} \\ &= \quad H_0 \quad\quad + \; H_1 \end{aligned} \tag{2.10}$$

onde K, P_0, P_1 são integráveis individualmente e w é uma função peso, tipo degrau, que depende da distância mútua entre os pares de corpos (ou da velocidade no pericentro), de maneira que por exemplo

$$w = \begin{cases} 1 & , \quad x \geqslant x_0 \\ p_n(x) & , \quad x_1 \leqslant x < x_0 \\ 0 & , \quad x < x_1 \end{cases}$$

sendo $p_n(x)$ um polinômio de grau n ímpar. O algoritmo simpléctico então é aplicado ao sistema $H_0 + H_1$, mantendo o passo τ constante, e uma eventual divergência do termo P quando $x \to x_1$ é compensada pelo peso $w \to 0$, de forma que P_0 se mantém sempre limitado. Por outro lado, a divergência do termo P_1 nas mesmas circunstâncias pode ser contornada fazendo-se uma nova partição deste termo, usando agora uma outra função peso w'. Essa nova partição gera um subsistema $H_0' + H_1'$ que pode ser integrado em forma simpléctica também com um passo τ' constante, porém menor, $\tau' < \tau$. Este procedimento de partição pode ser aplicado recursivamente até onde for necessário, gerando a cada recursão um novo subsistema que é resolvido com um passo de integração fixo cada vez menor. Esta é a estratégia utilizada pelo integrador simpléctico `SyMBA` (*Symplectic Massive Bodies Algorithm*; DUNCAN *et al.*, 1998; https://www.boulder.swri.edu/~hal/swift.html), que é bastante popular entre a comunidade de dinâmica planetária.

Um detalhe importante na Equação (2.10) é que, ainda que os termos K, P_1 sejam integráveis em forma individual, a sua soma H_0 não necessariamente é integrável. O integrador `Mercury` (CHAMBERS, 1991, 2012), também bastante popular entre a comunidade de dinâmica planetária, se vale neste caso de um algoritmo não simpléctico de alta precisão, o método de Bulirsh-Stoer (STOER e BULIRSH, 2002), para encontrar numericamente a solução de H_0. Ao contrário de `SyMBA`, `Mercury` não aplica uma recursão e deixa que o próprio método de Bulirsh-Stoer ajuste o tamanho do passo quando o termo P_1 diverge. A rigor, `Mercury` é um algoritmo híbrido e não pode ser considerado

estritamente simpléctico.

Finalmente, uma outra limitação dos integradores simplécticos que deve ser mencionada é o fato de que o hamiltoniano sub-rogante vem representado por uma série infinita de potências de τ. Se esta série não for convergente para todo τ, o hamiltoniano $\overline{H}$ resulta ser não integrável, mesmo que o hamiltoniano original seja integrável. Uma forma alternativa de entender isto é que o hamiltoniano sub-rogante, na prática, adiciona ao hamiltoniano original termos que dependem do tempo, o que equivale a adicionar $1/2$ grau de liberdade ao problema. Logo, se o sistema original possui um número de integrais de movimento suficientes para ser integrável, a adição de $1/2$ grau de liberdade destrói essa integrabilidade (CHIRIKOV, 1979). Um exemplo clássico em que isto acontece é o caso do mapa padrão, que resulta de aplicar um integrador simpléctico de primeira ordem para resolver o hamiltoniano do pêndulo simples. Enquanto o pêndulo simples é um sistema integrável, o mapa padrão pode apresentar comportamento caótico para certas condições iniciais próximas da separatriz.

Esta limitação, entretanto, não é relevante quando o hamiltoniano original não é integrável. Além disso, as perturbações dependentes de τ que o sistema sub-rogante adiciona ao sistema original, podem ser removidas da solução pela aplicação de um método perturbativo, o que é conhecido como corretor simpléctico (SAHA e TREMAINE, 1992; WISDOM *et al.* 1996).

2.2 O problema de N corpos na forma hamiltoniana

Para formular o problema que queremos resolver, vamos começar considerando um sistema isolado de $N+1$ corpos, cada um de massa m_i, com $i = 0, \ldots, N$, que interagem entre si gravitacionalmente. Cada corpo é descrito por sua posição $\vec{r}_i = \left(r_i^{(1)}, r_i^{(2)}, r_i^{(3)}\right)$ e velocidade $\vec{v}_i = \left(v_i^{(1)}, v_i^{(2)}, v_i^{(3)}\right)$, respeito a um sistema cartesiano inercial, e o seu momento é dado por $\vec{p}_i = m_i\vec{v}_i$. Segundo a notação usada na seção 2.1, as coordenadas e momentos canônicos do sistema são dados por $\mathbf{r} = (\vec{r}_0, \vec{r}_1 \ldots, \vec{r}_N)$ e $\mathbf{p} = (\vec{p}_0, \vec{p}_1 \ldots, \vec{p}_N)$ e o número de graus de liberdade do sistema é $n = 3N + 3$. A relação entre a componente k do vetor $\mathbf{r}$ e a componente j do vetor $\vec{r}_i$, por exemplo, vem dada por $k = 3i + j$.

O hamiltoniano resulta então da soma da energia cinética mais a energia potencial do sistema, isto é

$$\begin{aligned} H(\mathbf{r}, \mathbf{p}) &= \sum_{i=0}^{N} \frac{|\vec{p}_i|^2}{2m_i} - \sum_{i=0}^{N-1} \sum_{j=i+1}^{N} \frac{Gm_im_j}{|\vec{r}_i - \vec{r}_j|} \\ &= T(\mathbf{p}) + U(\mathbf{r}) \end{aligned} \tag{2.11}$$

sendo G a constante de gravitação. Como o baricentro do sistema, em particular, constitui um sistema inercial, esta formulação do hamiltoniano de N corpos costuma ser chamada

de baricêntrica.

O sistema que nos interessa, entretanto, é aquele em que $m_0 \gg m_i$, para $i \geqslant 1$. Neste caso, ambos termos T e U são da $\mathcal{O}(m_0)$ e a aplicação de um integrador simpléctico gera um sistema sub-rogante cuja diferença com o sistema original será da $\mathcal{O}(\tau^k m_0)$, tornando-se necessário aplicar um passo de integração muito pequeno ou utilizar um algoritmo de alta ordem.

Isto resulta evidente ao considerar que, quando um dos corpos é muito mais massivo que os restantes, estes últimos acabam orbitando o primeiro seguindo trajetórias aproximadamente elípticas. Entretanto, a solução gerada por $T(\mathbf{p})$ representa um movimento retilíneo uniforme durante um intervalo de tempo τ. Logo seria preciso usar um tamanho de passo bem pequeno para que os segmentos retilíneos consecutivos da trajetória fornecessem uma boa aproximação da elipse.

Para evitar esta limitação, a solução consiste, basicamente, em descrever o movimento relativo dos corpos m_i em relação a m_0, o que é conhecido como problema de N corpos planetário. Porém, é possível mostrar que a simples translação do sistema de referência para o corpo m_0 não fornece, em geral, um forma hamiltoniana para o problema. Isto se deve ao fato de que as posições e velocidades (ou momentos) em relação a m_0, chamadas de heliocêntricas ou astrocêntricas, não constituem um conjunto de coordenadas canonicamente conjugadas. Em outras palavras, a transformação de coordenadas baricêntricas para astrocêntricas não é uma transformação canônica[1]. Veremos a seguir dois conjuntos específicos de coordenadas canônicas que podem ser utilizados para representar o problema de N corpos planetário. Para uma revisão mais completa sobre outros conjuntos de coordenadas canônicas que podem ser utilizados, dependendo das especificidades do sistema, ver CHAMBERS *et al.* (2002) e CHAMBERS (2010).

2.2.1 Coordenadas de Jacobi

Este sistema de coordenadas foi introduzido por Carl Gustav Jacobi na formulação do problema restrito de três corpos e posteriormente utilizado por Charles-Eugène Delaunay e George William Hill na descrição do movimento da Lua ao redor da Terra, perturbada pelo Sol (BROUWER e CLEMENCE, 1961).

As coordenadas de Jacobi estabelecem uma hierarquia entre os corpos. que são ordenados em sentido crescente, desde o que orbita mais próximo de m_0 até o que orbita mais distante. As coordenadas são então definidas em forma tal que a posição e velocidade do corpo i são referidas ao baricentro dos $i-1$ corpos internos a ele. Isto implica que cada corpo "enxerga" o seu próprio sistema de referência. A transformação é definida da

[1] Existem duas exceções em que as coordenadas astrocêntricas preservam a forma hamiltoniana do sistema. A primeira é o problema de dois corpos. A segunda é o problema restrito de N corpos, isto é, quando existem no sistema M corpos massivos e $N-M$ partículas de teste. Neste último caso, cada partícula de teste é representada por um hamiltoniano que depende do tempo (sistema não autônomo).

seguinte forma

$$\begin{aligned}
\vec{R}_0 &= \vec{r}_{\rm c}^{(N)} \\
\vec{V}_0 &= \vec{v}_{\rm c}^{(N)} \\
\vec{R}_i &= \vec{r}_i - \vec{r}_{\rm c}^{(i-1)} \\
\vec{V}_i &= \vec{v}_i - \vec{v}_{\rm c}^{(i-1)}, \qquad i = 1, ..., N
\end{aligned}$$

onde

$$\vec{r}_{\rm c}^{(i)} = \frac{1}{M_i}\sum_{k=0}^{i} m_k \vec{r}_k, \qquad \vec{v}_{\rm c}^{(i)} = \frac{1}{M_i}\sum_{k=0}^{i} m_k \vec{v}_k$$

e $M_i = \sum_{k=0}^{i} m_k$. Nesta transformação, $\vec{R}_0, \vec{V}_0$ determinam a posição e velocidade do baricentro de todo o sistema, enquanto que $\vec{R}_1, \vec{V}_1$ constituem as coordenadas astrocêntricas do corpo mais interno, m_1.

A partir das nova velocidades, definem-se os novos momentos

$$\vec{P}_i = \mu_i \vec{V}_i$$

onde

$$\mu_0 = M_N, \qquad \mu_i = m_i \frac{M_{i-1}}{M_i}, \quad i \geqslant 1$$

e o hamiltoniano resulta ser

$$\begin{aligned}
H(\mathbf{R}, \mathbf{P}) &= \frac{\left|\vec{P}_0\right|^2}{2\mu_0} + \sum_{i=1}^{N} \frac{\left|\vec{P}_i\right|^2}{2\mu_i} - \sum_{i=0}^{N-1}\sum_{j=i+1}^{N} \frac{G m_i m_j}{|\vec{r}_j - \vec{r}_i|} \\
&= \frac{\left|\vec{P}_0\right|^2}{2\mu_0} + \sum_{i=1}^{N} \frac{\left|\vec{P}_i\right|^2}{2\mu_i} - \sum_{i=1}^{N} \frac{G m_0 m_i}{|\vec{r}_i - \vec{r}_0|} - \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \frac{G m_i m_j}{|\vec{r}_j - \vec{r}_i|} \\
&\quad - \sum_{i=1}^{N} \frac{G m_0 m_i}{\left|\vec{R}_i\right|} + \sum_{i=1}^{N} \frac{G m_0 m_i}{\left|\vec{R}_i\right|} \\
&= \frac{\left|\vec{P}_0\right|^2}{2\mu_0} + \sum_{i=1}^{N} \left(\frac{\left|\vec{P}_i\right|^2}{2\mu_i} - \sum_{i=1}^{N} \frac{G\mathcal{M}_i \mu_i}{\left|\vec{R}_i\right|} \right) - \sum_{i=1}^{N} G m_0 m_i \left(\frac{1}{\left|\vec{R}'_i\right|} - \frac{1}{\left|\vec{R}_i\right|} \right) \\
&\quad - \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \frac{G m_i m_j}{\left|\vec{R}'_j - \vec{R}'_i\right|}
\end{aligned}$$

onde

$$\mathcal{M}_i = \frac{m_0 M_i}{M_{i-1}}$$

e

$$\begin{aligned}\vec{R}'_i &= \vec{r}_i - \vec{r}_0 \\ &= \vec{R}_i + \sum_{k=1}^{i-1} \frac{m_k}{M_k} \vec{R}_k\end{aligned}$$

representa a posição astrocêntrica do corpo i.

Notemos que $\vec{P}_0 = \sum_{i=0}^{N} \vec{p}_i$ constitui o momento linear total do sistema e, como o hamiltoniano não depende de $\vec{R}_0$, segue-se que $\vec{P}_0$ é constante e pode ser desconsiderado. O hamiltoniano então pode ser escrito como

$$H(\mathbf{R}, \mathbf{P}) = \sum_{i=1}^{N} H_{\text{Kep}}(\vec{R}_i, \vec{P}_i) + H_{\text{Pert}}(\mathbf{R}) \tag{2.12}$$

onde

$$H_{\text{Kep}} = \frac{\left|\vec{P}_i\right|^2}{2\mu_i} - \sum_{i=1}^{N} \frac{G\mathcal{M}_i\mu_i}{\left|\vec{R}_i\right|}$$

é um hamiltoniano do problema de dois corpos que descreve o movimento kepleriano de um corpo de massa μ_i ao redor de um corpo central de massa $\mathcal{M}_i$, enquanto que

$$H_{\text{Pert}} = -\sum_{i=1}^{N} Gm_0m_i \left(\frac{1}{\left|\vec{R}'_i\right|} - \frac{1}{\left|\vec{R}_i\right|} \right) - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{Gm_im_j}{\left|\vec{R}'_j - \vec{R}'_i\right|}$$

representa as perturbações gravitacionais mútuas. Notemos que, ainda que H_{Pert} seja proporcional a m_0, ele depende das diferenças entre $1/R'_i$ e $1/R_i$ que são da $\mathcal{O}(m_k/M_k) \approx \mathcal{O}(1/m_0)$. Portanto, H_{Pert} resulta ser, em geral, um termo muito pequeno comparado a H_{Kep}.

Uma das desvantagens das coordenadas de Jacobi é que uma vez que os corpos foram ordenados numa dada a hierarquia, ela não pode ser alterada, isto é, as órbitas não podem se intersectar entre si. Além disso, no caso de órbitas muito excêntricas, a diferença entre $\vec{R}_i$ e $\vec{R}'_i$, para $i > 1$, torna-se significativa perto do pericentro, fazendo com que o primeiro somatório em H_{Pert} deixe de ser pequeno comparado a H_{Kep}. Estas desvantagens limitam a aplicação do hamiltoniano (2.12) a sistemas em que as órbitas são pouco excêntricas e se encontram bem separadas.

2.2.2 O método simpléctico de Wisdom e Holman

Aplicando ao hamiltoniano (2.12) o integrador simpléctico de segunda ordem dado pela Equação (2.7), obtém-se o conhecido integrador de Wisdom e Holman. A solução neste

caso é dada pela sequência de operadores

$$\left(\vec{R}_i, \vec{P}_i\right)_{t+\tau} = \left(1 + \frac{\tau}{2}\mathcal{D}_{H_{\rm Pert}}\right) \exp \tau\mathcal{D}_{H_{\rm Kep}} \left(1 + \frac{\tau}{2}\mathcal{D}_{H_{\rm Pert}}\right) \left(\vec{R}_i, \vec{P}_i\right)_t$$

em que o operador série de Lie do meio é substituído pela solução kepleriana do problema de dois corpos. O integrador se reduz à seguinte receita:

1. Se é a primeira vez, calcular as forças $\vec{F}_{i,\rm Pert} = -\dfrac{\partial H_{\rm Pert}(\mathbf{R})}{\partial \vec{R}_i}$
2. Aplicar um impulso nos momentos $\vec{P}'_i = \vec{P}_i + \dfrac{\tau}{2}\vec{F}_{i,\rm Pert}$ deixando as posições fixas, $\vec{R}'_i = \vec{R}_i$
3. Avançar $\vec{R}'_i$ e $\vec{P}'_i$ ao longo de uma órbita kepleriana por um intervalo de tempo τ, obtendo $\vec{R}''_i$ e $\vec{P}''_i$
4. Recalcular as forças $\vec{F}''_{i,\rm Pert} = -\dfrac{\partial H_{\rm Pert}(\mathbf{R}'')}{\partial \vec{R}''_i}$
5. Aplicar um impulso nos momentos $\vec{P}'''_i = \vec{P}''_i + \dfrac{\tau}{2}\vec{F}''_{i,\rm Pert}$ deixando as posições fixas, $\vec{R}'''_i = \vec{R}''_i$
6. Voltar ao ponto #2

Notemos que, com exceção do primeiro passo, as forças de perturbação são calculadas apenas uma vez a cada passo, já que no ponto #5 as posições ficam inalteradas.

Para avançar as posições e velocidades sobre a trajetória kepleriana, o método se vale das funções f, g de Gauss, como descrito em DANBY (1988). Esquematicamente, dados $\vec{R}, \vec{V}$ no instante t, os valores de $\vec{R}', \vec{V}'$ para um instante qualquer t' são dados por

$$\vec{R}' = f(t', t)\ \vec{R} + g(t', t)\ \vec{V}, \qquad \vec{V}' = \frac{\partial f(t', t)}{\partial t'}\ \vec{R} + \frac{\partial g(t', t)}{\partial t'}\ \vec{V} \tag{2.13}$$

onde f e g são funções escalares que independem do sistema de coordenadas utilizado. A forma das funções f, g e diferente para as órbitas elípticas, parabólicas e hiperbólicas, mas elas dependem basicamente de $\left|\vec{R}\right|, \left|\vec{V}\right|$ e das diferenças $\Delta t = t' - t$ e $\Delta u = u' - u$, sendo u a anomalia excêntrica. Em particular, para determinar Δu é necessário resolver a equação de Kepler, e este cálculo é optimizado através de um algoritmo descrito por STUMPFF (1988), que torna o método eficiente e rápido. Neste ponto, resulta evidente a vantagem de aplicar o integrador (2.7) ao invés do integrador (2.9), já que com este último as forças continuam sendo calculadas uma vez a cada passo de integração, mas a equação de Kepler precisaria ser resolvida duas vezes a cada passo, ao invés de uma.

Este método foi introduzido por WISDOM e HOLMAN (1991) e está incluído em vários pacotes de integração numérica para sistemas planetários, como `Swift` (LEVISON

e DUNCAN, 1994), `Swifter` (KAUFMANN e LEVISON; http://www.boulder.swri.edu/swifter/), `Mercury` (CHAMBERS, 1991, 2012), `Rebound` (REIN e LIU, 2012; REIN e TAMAYO, 2015; https://rebound.readthedocs.io/en/latest/) e `EVORB` (FERNANDEZ *et al.*, 2002; http://www.fisica.edu.uy/~gallardo/evorb.html).

2.2.3 Coordenadas astrocêntricas canônicas

Este sistema de coordenadas foi introduzido por Henri Poincaré e, na literatura recente, é às vezes referido como coordenadas democráticas heliocêntricas (ou astrocêntricas), pois a massa central é a mesma para todos os corpos.

A transformação parte de definir

$$\begin{aligned} \vec{R}_0 &= \vec{r}_0 \\ \vec{R}_i &= \vec{r}_i - \vec{r}_0, \qquad i = 1, ..., N \end{aligned}$$

e aplicar a definição de transformação canônica dada pela Equação (2.1) para achar os momentos $\vec{P}_i$, isto é

$$\sum_{i=0}^{N} \vec{P}_i \cdot d\vec{R}_i = \sum_{i=0}^{N} \vec{p}_i \cdot d\vec{r}_i = \sum_{i=1}^{N} \vec{p}_i \cdot d\vec{R}_i + \left(\vec{p}_0 + \sum_{i=1}^{N} \vec{p}_i \right) \cdot d\vec{R}_0$$

portanto

$$\begin{aligned} \vec{P}_0 &= \vec{p}_0 + \sum_{i=1}^{N} \vec{p}_i \\ \vec{P}_i &= \vec{p}_i, \qquad i = 1, ..., N \end{aligned}$$

O conjunto de coordenadas $\vec{R}_i, \vec{P}_i$ representam um sistema no qual as posições são astrocêntricas enquanto que os momentos (ou velocidades) são baricêntricos (BROUWER e CLEMENCE, 1961). O hamiltoniano adota a forma

$$\begin{aligned} H(\mathbf{R}, \mathbf{P}) = & \frac{\left|\vec{P}_0\right|^2}{2m_0} - \sum_{i=1}^{N} \frac{\vec{P}_0 \cdot \vec{P}_i}{m_0} + \sum_{i=1}^{N} \left(\frac{\left|\vec{P}_i\right|^2}{2m_i} - \sum_{i=1}^{N} \frac{G m_0 m_i}{\left|\vec{R}_i\right|} \right) \\ & - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{G m_i m_j}{\left|\vec{R}_j - \vec{R}_i\right|} + \frac{1}{2m_0} \left| \sum_{i=1}^{N} \vec{P}_i \right|^2 \end{aligned} \tag{2.14}$$

Mais uma vez, $\vec{P}_0$ constitui o momento linear total do sistema e, como o hamiltoniano não depende de $\vec{R}_0$, segue-se que $\vec{P}_0$ é constante e, em particular, $\vec{P}_0 = 0$ no baricentro.

O hamiltoniano então pode ser escrito como

$$H(\mathbf{R},\mathbf{P}) = \sum_{i=1}^{N} H_{\text{Kep}}(\vec{R}_i,\vec{P}_i) + H_{\text{Pert}}(\mathbf{R}) + H_{\text{Star}}(\mathbf{P}) \tag{2.15}$$

onde novamente

$$H_{\text{Kep}} = \frac{\left|\vec{P}_i\right|^2}{2m_i} - \frac{Gm_0m_i}{\left|\vec{R}_i\right|}$$

é um hamiltoniano do problema de dois corpos que descreve o movimento kepleriano de um corpo de massa m_i ao redor de um corpo central de massa m_0,

$$H_{\text{Pert}} = -\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \frac{Gm_im_j}{\left|\vec{R}_j - \vec{R}_i\right|}$$

representa as perturbações gravitacionais mútuas e

$$H_{\text{Star}} = \frac{1}{2m_0}\left|\sum_{i=1}^{N}\vec{P}_i\right|^2$$

é um termo vinculado ao movimento do corpo central ao redor do baricentro do sistema. Resulta claro que se $m_0 \gg m_i$, tanto H_{Pert} como H_{Star} são termos pequenos comparados a H_{Kep}.

A principal vantagem deste conjunto de coordenadas, em comparação às de Jacobi, é que não há qualquer hierarquia entre os corpos, logo as órbitas podem se intersectar. Por outro lado, uma característica indesejável é que elas não fornecem uma interpretação física clara do movimento, já que os momentos $\vec{P}_i$ não são proporcionais às velocidades $\vec{V}_i = d\vec{R}_i/dt$ e portanto não são tangentes às trajetórias. De fato,

$$\vec{P}_i = m_i\vec{V}_i - \frac{m_i}{m_0}\sum_{j=1}^{N}\vec{P}_j$$

e isto é, provavelmente, o principal motivo pelo qual este sistema foi negligenciado no passado em favor do sistema de Jacobi.

Uma desvantagem destas coordenadas é que, ao permitir o cruzamento de órbitas, existe a possibilidade de ocorrerem encontros próximos entre os corpos que fazem com que H_{Pert} deixe de ser pequeno comparado a H_{Kep}. Analogamente, a ocorrência de órbitas muito excêntricas faz com que durante a passagem pelo pericentro a velocidade orbital aumente significativamente e o termo H_{Star} também deixe de ser pequeno comparado a H_{Kep}. Estas duas situações limitam a aplicação do hamiltoniano (2.14), mas podem ser contornadas com esquemas de partição como mencionamos na seção 2.1.3.

Finalmente, cabe destacar que no caso de um problema de dois corpos ($N = 1$), a Equação (2.14) se reduz à forma

$$H = H_{\text{Kep}}(\vec{R}_1, \vec{P}_1) + H_{\text{Star}}(\vec{P}_1)$$

de modo que o problema de dois corpos não fica representado, a princípio, por um hamiltoniano puramente kepleriano. Em uma integração simpléctica, o efeito da perturbação $H_{\text{Star}} = \left|\vec{P}_1\right|^2 / 2m_0$ irá se traduzir numa precessão lenta do pericentro da órbita, que resulta mais proeminente quanto maior for o tamanho do passo. Pode ser argumentar que, num problema de N corpos, esta precessão espúria não é relevante, pois a evolução do pericentro é dominada pelas perturbações gravitacionais. Entretanto, uma forma simples de evitar este problema consiste em incorporar a parte puramente quadrática do termo H_{Star} no termo H_{Kep}. Nesse caso, o hamiltoniano (2.14) é reagrupado da forma

$$H(\mathbf{R}, \mathbf{P}) = \sum_{i=1}^{N} \mu_i \left(\frac{\left|\vec{P}_i\right|^2}{2m_i} - \frac{G\mathcal{M}_i m_i}{\left|\vec{R}_i\right|} \right) - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{G m_i m_j}{\left|\vec{R}_j - \vec{R}_i\right|} + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\vec{P}_i \cdot \vec{P}_j}{m_0} \tag{2.16}$$

onde

$$\mu_i = \frac{m_0 + m_i}{m_0}$$
$$\mathcal{M}_i = \frac{m_0}{\mu_i} = \frac{m_0^2}{m_0 + m_i}$$

Notemos que agora o primeiro termo é uma soma de hamiltonianos de 2 corpos, cada um multiplicado por um fator μ_i. Neste caso, as soluções respectivas são órbitas keplerianas nas variáveis $\vec{R}_i, \vec{P}_i$ mas num tempo reescalado $dt' = \mu_i \, dt$ (note-se também a mudança na massa central).

2.2.4 O método simpléctico `Helio`

A construção de um integrador simpléctico de ordem dois para resolver o hamiltoniano (2.15) parte do mesmo princípio apresentado na seção 2.1.2 só que agora o hamiltoniano possui três termos, logo

$$\exp \tau \mathcal{D}_H = \prod_{i=1}^{2} \exp \alpha_i \tau \mathcal{D}_{H_0} \exp \beta_i \tau \mathcal{D}_{H_1} \exp \gamma_i \tau \mathcal{D}_{H_2} + \mathcal{O}(\tau^3)$$

Expandindo as séries de Lie em ambos lados da equação até ordem dois e comparando termos de mesma ordem, se chega a um conjunto de equações para os coeficientes que

tem como possível solução

$$\alpha_1 = 1, \quad \alpha_2 = 0, \quad \beta_1 = \beta_2 = \gamma_1 = \gamma_2 = \frac{1}{2}$$

O integrador, então, é representado pela sequência de operadores

$$\begin{aligned}\left(\vec{R}_i, \vec{P}_i\right)_{t+\tau} = &\left(1 + \frac{\tau}{2}\mathcal{D}_{H_{\rm Pert}}\right)\left(1 + \frac{\tau}{2}\mathcal{D}_{H_{\rm Star}}\right)\exp\tau\mathcal{D}_{H_{\rm Kep}} \\ &\times \left(1 + \frac{\tau}{2}\mathcal{D}_{H_{\rm Star}}\right)\left(1 + \frac{\tau}{2}\mathcal{D}_{H_{\rm Pert}}\right)\left(\vec{R}_i, \vec{P}_i\right)_t\end{aligned}$$

e, novamente, o operador série de Lie do meio é substituído pela solução kepleriana do problema de dois corpos. O integrador se reduz à seguinte receita:

1. Se é a primeira vez, calcular as forças $\vec{F}_{i,\rm Pert} = -\dfrac{\partial H_{\rm Pert}(\mathbf{R})}{\partial \vec{R}_i}$
2. Aplicar um impulso nos momentos $\vec{P}_i' = \vec{P}_i + \dfrac{\tau}{2}\vec{F}_{i,\rm Pert}$ deixando as posições fixas, $\vec{R}_i' = \vec{R}_i$
3. Aplicar uma translação nas posições $\vec{R}_i'' = \vec{R}_i' + \dfrac{\tau}{2}\displaystyle\sum_{k=1}^{N}\frac{\vec{P}_k'}{m_0}$ deixando os momentos fixos $\vec{P}_i'' = \vec{P}_i'$
4. Avançar $\vec{R}_i''$ e $\vec{P}_i''$ ao longo de uma órbita kepleriana por um intervalo de tempo τ, obtendo $\vec{R}_i'''$ e $\vec{P}_i'''$
5. Aplicar uma translação nas posições $\vec{R}_i^{\rm IV} = \vec{R}_i''' + \dfrac{\tau}{2}\displaystyle\sum_{k=1}^{N}\frac{\vec{P}_k'''}{m_0}$ deixando os momentos fixos $\vec{P}_i^{\rm IV} = \vec{P}_i'''$
6. Recalcular as forças $\vec{F}_{i,\rm Pert}^{\rm IV} = -\dfrac{\partial H_{\rm Pert}(\mathbf{R}^{\rm IV})}{\partial \vec{R}_i^{\rm IV}}$
7. Aplicar um impulso nos momentos $\vec{P}_i^{\rm V} = \vec{P}_i^{\rm IV} + \dfrac{\tau}{2}\vec{F}_{i,\rm Pert}^{\rm IV}$ deixando as posições fixas, $\vec{R}_i^{\rm V} = \vec{R}_i^{\rm IV}$
8. Voltar ao ponto #2

Notemos que, com exceção do primeiro passo, as forças de perturbação são calculadas apenas uma vez a cada passo, já que no ponto #7 as posições ficam inalteradas. A princípio, é possível inverter a ordem de aplicação dos passos #2 e #3, ou dos passos #5 e #7, mas nesse caso as forças acabam tendo que ser calculadas duas vezes a cada passo de integração.

Este método é referido às vezes como método DH, pelo seu uso de coordenadas democráticas heliocêntricas. Foi introduzido por CHAMBERS (1991) e constitui a base dos

integradores `Mercury` e `SyMBA`, além de estar incluído em outros pacotes de integração como `Swift` e `Swifter`. Curiosamente, todas estas implementações estão baseadas no hamiltoniano (2.14) e, portanto, podem introduzir uma precessão artificial do pericentro das órbitas, como já discutimos.

2.3 Estimadores de caos

Caos é um fenômeno caracterizado por um comportamento irregular e imprevisível das soluções de um sistema dinâmico. O conceito de caos está vinculado ao conceito de divergência exponencial de soluções vizinhas (LICHTENBERG e LIEBERMAN, 1992). Dado um sistema de equações diferenciais ordinárias autônomo

$$\frac{d\mathbf{z}}{dt} = \mathbf{f}(\mathbf{z}), \qquad \mathbf{z} \in \mathbb{R}^n \tag{2.17}$$

e duas condições iniciais em t_0 vizinhas, $\mathbf{z}_0$ e $\mathbf{y}_0 = \mathbf{z}_0 + \boldsymbol{\epsilon}_0$, a distância entre as duas soluções varia aproximadamente como

$$\boldsymbol{\epsilon}(t) = \boldsymbol{\epsilon}_0 \left[1 + (t - t_0)^\gamma\right] e^{\lambda(t-t_0)} \tag{2.18}$$

Assim, num sistema regular, $\lambda = 0$ e a distância cresce segundo uma lei de potências com o tempo. Por outro lado, num sistema caótico $\lambda > 0$ e, após um certo tempo, o termo exponencial domina o crescimento. Este comportamento pode ser interpretado em termos de perda de informação, já que num sistema regular, um erro na solução irá se propagar em forma linear (progressão aritmética), mas num sistema caótico o erro se propaga em forma exponencial (progressão geométrica). Logo, o comportamento de um sistema caótico acaba sendo previsível apenas no curto prazo.

Uma condição necessária e suficiente para que um sistema manifeste caos é que o mesmo seja não integrável. Se o sistema possui $n - 1$ integrais primeiras da forma

$$I_i(t, \mathbf{z}) = C_i, \qquad i = 1, \ldots, n-1$$

onde as constantes C_i são fixadas pelas condições iniciais do problema, ele pode ser reduzido a uma única equação

$$\frac{dz_n}{dt} = f_n(z_n, t, C_1, \ldots, C_{n-1})$$

que pode ser resolvida por quadraturas, resultando assim integrável[2]. Geometricamente, as integrais primeiras limitam o domínio acessível à solução a um subespaço de $\mathbb{R}^n$, que se

[2]Um sistema de EDOs se diz completamente integrável se possui n integrais primeiras. Sistemas de EDOs lineares, por exemplo, são completamente integráveis.

reduz a $\mathbb{R}$ quando o sistema é integrável, sendo possível prever onde a solução se encontrará a qualquer tempo (ARNOLD, 1992). Se existem apenas $m < n-1$ integrais primeiras, a solução pode, a princípio, ter acesso a qualquer parte do subespaço $\mathbb{R}^{n-m}$, não sendo possível, a priori, prever onde a solução se encontrará a qualquer tempo. Neste subespaço, uma solução caótica apresentará um comportamento ergódico[3], podendo preencher todo o volume de espaço disponível (WALTERS, 1982). Sistemas integráveis podem se tornar não integráveis pela adição de termos não lineares ou de termos de acoplamento entre as variáveis. No caso específico de sistemas mecânicos, isto é, aqueles que seguem a segunda lei de Newton, a adição de graus de liberdade ao problema pode destruir a integrabilidade.

O conceito de caos também pode ser vinculado, dentro de um contexto específico, ao conceito de difusão anômala. Como mencionamos na seção 2.1, em sistemas hamiltonianos a integrabilidade garante que existe um conjunto de coordenadas canônicas $\boldsymbol{\theta}, \mathbf{J}$ tais que o hamiltoniano depende apenas dos momentos, $H = H_0(\mathbf{J})$, que são portanto constantes (os momentos estão relacionados às integrais primeiras do sistema). Se perturbamos o hamiltoniano com um termo H_1 da forma

$$H = H_0(\mathbf{J}) + \varepsilon H_1(\boldsymbol{\theta}, \mathbf{J})$$

onde $\varepsilon \ll 1$, em geral se quebra a integrabilidade do sistema. Os momentos apresentarão uma variação $\Delta\mathbf{J} \sim \mathcal{O}(\varepsilon)$ e a maioria das soluções do sistema perturbado permanecerão próximas às soluções do sistema não perturbado durante todo o tempo. Entretanto, poderão haver soluções que exibam uma mudança arbitrariamente grande nos momentos, manifestando um comportamento caótico. Para uma dada condição inicial $\boldsymbol{\theta}_0, \mathbf{J}_0$ em $t = 0$, a série temporal $\mathbf{J}(t)$ pode ser caracterizada pela sua média $\boldsymbol{\mu}$ e variância $\boldsymbol{\sigma}^2$

$$\mu_i(t) = \frac{1}{t}\int_0^t J_i(t')\,dt', \qquad \sigma_i^2(t) = \frac{1}{t}\int_0^t \left[J_i(t') - \mu_i(t)\right]^2\,dt'$$

e se cumpre que

$$\left\langle \sigma_i^2(t) \right\rangle = D_i\, t^{2H_e}$$

onde D_i constitui o coeficiente de difusão, H_e é o denominado exponente de Hurst, e o símbolo $\langle \cdots \rangle$ representa a média sobre um ensemble de condições iniciais (METZLER *et al.*, 2014). Num processo difusivo clássico, $H_e = 1/2$. Valores de $H_e > 1/2$ ou $< 1/2$ caracterizam difusão anômala: super-difusão ou sub-difusão, respectivamente. O coeficiente de difusão D_i fornece uma medida de quanto as soluções do problema perturbado vão se afastar das soluções do problema não perturbado.

O conceito de difusão caótica que é aplicado aos momentos canônicos $\mathbf{J}$ pode também

[3] Um sistema dinâmico ergódico é aquele em que a média de uma solução sobre o tempo apresenta o mesmo comportamento que a média da solução sobre as variáveis do espaço de fases.

ser estendido às frequências fundamentais do problema, $\boldsymbol{\nu}$, definidas a partir de:

$$\nu_i = \frac{\partial H_0}{\partial J_i}$$

A análise frequencial é uma técnica que engloba diferentes métodos que são baseados nesta ideia e que podem ser utilizados como estimadores de caos. Entretanto, a implementação destes métodos requer o uso de algoritmos numéricos de análise de Fourier discreta, que apresentam uma certa complexidade na sua aplicação e que consideramos estarem além do escopo deste trabalho de tese.

Nas seções a seguir, veremos como implementar estes conceitos no cálculo numérico da solução de um problema de N corpos e como podemos definir indicadores que quantifiquem o comportamento caótico.

2.3.1 As equações variacionais

Consideremos novamente a Equação (2.17) e duas soluções próximas $\mathbf{z}(t)$ e $\mathbf{y}(t) = \mathbf{z}(t) + \boldsymbol{\epsilon}(t)$, assumindo que $\boldsymbol{\epsilon}(t) \ll 1$ pelo menos num dado intervalo de tempo. Podemos então escrever

$$\begin{aligned} \frac{d\mathbf{y}}{dt} &= \mathbf{f}(\mathbf{y}) \\ \frac{d\mathbf{z}}{dt} + \frac{d\boldsymbol{\epsilon}}{dt} &= \mathbf{f}(\mathbf{z} + \boldsymbol{\epsilon}) \\ &= \mathbf{f}(\mathbf{z}) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right|_{\mathbf{z}(t)} \boldsymbol{\epsilon} + \mathcal{O}(\boldsymbol{\epsilon}^2) \end{aligned}$$

onde $\partial\mathbf{f}/\partial\mathbf{z} = \mathbf{M}$ representa a matriz Jacobiana $n \times n$ de $\mathbf{f}$ ($M_{ij} = \partial f_i/\partial z_j$), avaliada sobre a solução de referência. Se descartamos os termos de ordem superior, podemos escrever

$$\frac{d\boldsymbol{\delta}}{dt} = \mathbf{M}(t)\,\boldsymbol{\delta} \tag{2.19}$$

onde utilizamos $\boldsymbol{\delta}$ no lugar de $\boldsymbol{\epsilon}$ para deixar claro que estamos linearizando o sistema[4].

O sistema de Equações (2.19) constituem as denominadas equações variacionais do sistema (2.17), Trata-se de um sistema de EDOs linear, homogêneo, não autônomo cuja solução pode ser determinada por aproximações sucessivas na forma de uma série infinita (MAGNUS, 1954; PARS, 1979), mas com exceção de alguns casos particulares, não possui uma forma analítica fechada. Assim, para a maioria das aplicações práticas, as equações variacionais são integradas numericamente junto com as equações do sistema.

[4] $\boldsymbol{\epsilon}$ representa a distância euclídea entre as soluções no espaço de fases, enquanto que $\boldsymbol{\delta}$ constitui a projeção da distância no subespaço tangente à solução. Em forma análoga à secante entre dois pontos vizinhos numa curva que pode ser aproximada pela tangente à curva quando os pontos ficam infinitamente próximos.

Num algoritmo de integração numérica padrão, como Runge-Kutta, a solução simultânea das Equações (2.17) e (2.19) não representa qualquer dificuldade, sendo apenas necessário levar em consideração que as segundas dependem da solução das primeiras. Entretanto, num algoritmo simpléctico, a resolução das equações variacionais não pode ser incorporada diretamente ao método, pois o sistema (2.19) não possui forma hamiltoniana.

Uma alternativa seria aninhar dentro do algoritmo simpléctico um método de Runge-Kutta para resolver somente as equações variacionais a partir da solução fornecida pelo integrador simpléctico. Outra alternativa consiste em calcular diretamente a cada passo de integração as variações $\boldsymbol{\delta}$ no espaço tangente a partir da diferenciação direta da solução simpléctica, não sendo necessário resolver o sistema (2.19). Esta última opção foi sugerida por MIKKOLA e INNANEN (1999), que a denominou de mapa tangente simpléctico, e foi aplicada por primeira vez ao integrador de Wisdom e Holman. Em nosso caso, decidimos aplicar a mesma ideia ao integrador `Helio`.

Consideremos o hamiltoniano da Equação (2.15)

$$H(\mathbf{r},\mathbf{p}) = \sum_{i=1}^{N} H_{\mathrm{K}}(\vec{r}_i,\vec{p}_i) + H_{\mathrm{P}}(\mathbf{r}) + H_{\mathrm{S}}(\mathbf{p})$$

e definamos o vetor de variação tangente à solução como $\boldsymbol{\delta} = (\delta\mathbf{r},\delta\mathbf{p}) \equiv (\delta\mathbf{r}_1,\ldots,\delta\mathbf{r}_N,\delta\mathbf{p}_1,\ldots,\delta\mathbf{p}_N)$. A aplicação do operador série de Lie $\exp\frac{\tau}{2}\mathcal{D}_{H_{\mathrm{P}}}$ implica uma mudança nos momentos tal que

$$\vec{p}_i' = \vec{p}_i - \frac{\tau}{2}\sum_{\substack{j=1\\ j\neq i}}^{N} G m_i m_j \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3} \tag{2.20}$$

e diferenciando esta equação temos

$$\delta\vec{p}_i' = \delta\vec{p}_i - \frac{\tau}{2}\sum_{\substack{j=1\\ j\neq i}}^{N} G m_i m_j \left(\frac{\delta\vec{r}_i - \delta\vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3} - 3\,(\vec{r}_i - \vec{r}_j)\cdot(\delta\vec{r}_i - \delta\vec{r}_j)\,\frac{(\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j|^5} \right) \tag{2.21}$$

Por sua vez, a aplicação do operador série de Lie $\exp\frac{\tau}{2}\mathcal{D}_{H_{\mathrm{S}}}$ implica uma mudança nas coordenadas

$$\vec{r}_i' = \vec{r}_i + \frac{\tau}{2}\sum_{j=1}^{N}\frac{\vec{p}_j}{m_0} \tag{2.22}$$

de forma que diferenciando esta equação temos

$$\delta\vec{r}_i' = \delta\vec{r}_i + \frac{\tau}{2}\sum_{j=1}^{N}\frac{\delta\vec{p}_j}{m_0} \tag{2.23}$$

Finalmente, a aplicação do operador série de Lie $\exp\tau\mathcal{D}_{H_{\rm K}}$ avança a solução ao longo da órbita kepleriana com as funções f, g de Gauss (Equações 2.13). Diferenciando esta solução temos

$$\begin{aligned}\delta\vec{r}_i' &= f\,\delta\vec{r}_i + g\,\delta\vec{v}_i + \vec{r}_i\,\delta f + \vec{v}_i\,\delta g\\ \delta\vec{v}_i' &= \dot{f}\,\delta\vec{r}_i + \dot{g}\,\delta\vec{v}_i + \vec{r}_i\,\delta\dot{f} + \vec{v}_i\,\delta\dot{g}\end{aligned} \tag{2.24}$$

As expressões completas para $\delta f, \delta g, \delta\dot{f}, \delta\dot{g}$ são fornecidas por MIKKOLA e INNANEN (1999).

A combinação do integrador simpléctico com o mapa tangente se reduz então aos seguintes passos:

1. Se é a primeira vez, calcular as forças
2. Aplicar um impulso em $\mathbf{p}$ de acordo com a Equação (2.20) deixando $\mathbf{r}$ fixo
3. Aplicar um impulso em $\delta\mathbf{p}$ de acordo com a Equação (2.21) deixando $\delta\mathbf{r}$ fixo
4. Aplicar uma translação em $\mathbf{r}$ de acordo com a Equação (2.22) deixando $\mathbf{p}$ fixo
5. Aplicar uma translação em $\delta\mathbf{r}$ de acordo com a Equação (2.23) deixando $\delta\mathbf{p}$ fixo
6. Avançar $\mathbf{r}, \mathbf{p}$ ao longo da órbita kepleriana por um intervalo de tempo τ
7. Avançar $\delta\mathbf{r}, \delta\mathbf{p}$ com as Equações (2.24) por um intervalo de tempo τ
8. Aplicar uma translação em $\mathbf{r}$ de acordo com a Equação (2.22) deixando $\mathbf{p}$ fixo
9. Aplicar uma translação em $\delta\mathbf{r}$ de acordo com a Equação (2.23) deixando $\delta\mathbf{p}$ fixo
10. Recalcular as forças
11. Aplicar um impulso em $\mathbf{p}$ de acordo com a Equação (2.20) deixando $\mathbf{r}$ fixo
12. Aplicar um impulso em $\delta\mathbf{p}$ de acordo com a Equação (2.21) deixando $\delta\mathbf{r}$ fixo
13. Voltar ao ponto #2

Uma forma de checar a precisão na solução das equações variacionais fornecida por este método consiste em perceber que a variação de qualquer integral de movimento do sistema

(2.17) é uma integral de movimento do sistema (2.19). No caso particular de um sistema hamiltoniano, temos que

$$\delta H = \frac{\partial H}{\partial \mathbf{r}} \cdot \delta\mathbf{r} + \frac{\partial H}{\partial \mathbf{p}} \cdot \delta\mathbf{p} = \text{cte.}$$

2.3.2 Exponentes de Lyapunov

A solução das equações variacionais determina como se comporta o fluxo do sistema num entorno da solução de referência. Isto, por exemplo, fornece informação direta acerca de como o erro da solução se propaga com o tempo. Assim, conforme vimos no início da seção 2.3, podemos usar a solução das equações variacionais não apenas para determinar se a solução de referência é caótica ou não, mas também para quantificar o quão caótica ela pode ser.

Para entender melhor este ponto, vamos considerar primeiro alguns casos particulares das Equações (2.19) (MARCHAL, 1988):

2.3.2.1 Matriz M constante

Neste caso, é possível determinar os n autovalores, μ_i, e autovetores, $\mathbf{u}_i$, da matriz. Se os autovetores são todos linearmente independentes (isto é, a matriz tem forma diagonal pura), a solução geral é dada por

$$\boldsymbol{\delta}(t) = \boldsymbol{\delta}_0 \exp(t\mathbf{M}) = \sum_{k=1}^{n} \delta_{0,k}\mathbf{u}_k \exp \mu_k t$$

Se os autovetores não são todos independentes, a matriz pode ser reduzida à forma de Jordan e, nesse caso, aparecem na solução termos mistos da forma $t^m \exp \mu_k t$, onde a potência m dependerá da multiplicidade do autovalor μ_k. Notemos a semelhança desta solução com a evolução proposta pela Equação (2.18). Define-se então os exponentes característicos de Lyapunov, λ_i, do sistema como

$$\lambda_i = \text{Re}(\mu_i)$$

2.3.2.2 Matriz M periódica,

No caso em que $\mathbf{M}(t+T) = \mathbf{M}(t)$, a solução das equações variacionais também é periódica com mesmo período. É possível então mostrar que existe uma matriz $\boldsymbol{\Sigma}$ constante e não singular, tal que para qualquer vetor de condições iniciais $\boldsymbol{\delta}_0 = \boldsymbol{\delta}(0)$ se cumpre que

$$\boldsymbol{\delta}(T) = \boldsymbol{\Sigma}\boldsymbol{\delta}(0)$$
$$\boldsymbol{\delta}(kT) = \boldsymbol{\Sigma}^k\boldsymbol{\delta}(0)$$

A matriz $\mathbf{\Sigma}$ é denominada de matriz monodrômica (WINTNER, 2014) ou de Floquet e os seus autovalores, σ_i, são denominados de números de Floquet. Neste caso, se definem os exponentes característicos de Lyapunov como

$$\lambda_i = \frac{1}{T} \ln |\sigma_i|$$

2.3.2.3 Caso geral

Quando $\mathbf{M} = \mathbf{M}(t)$, a solução para qualquer instante de tempo é dada através da matriz de transição de estado $\mathbf{S}(t)$, de forma que

$$\boldsymbol{\delta}(t) = \mathbf{S}(t)\boldsymbol{\delta}(0)$$

Esta é uma matriz $n \times n$ em que as colunas são as soluções das equações variacionais geradas por n condições iniciais $\boldsymbol{\delta}_{k,0}$ linearmente independentes, por exemplo $\boldsymbol{\delta}_{k,0} = \mathbf{e}_k$, sendo $\mathbf{e}_k$ o versor unitário na direção k do espaço de fases. Se $s_i(t)$ são os autovalores da matriz $\mathbf{S}$ para um dado t, define-se os exponentes característicos de Lyapunov como

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \ln |s_i(t)|$$

Na prática, os exponentes de Lyapunov são definidos diretamente a partir da solução das equações variacionais através da fórmula

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \ln \frac{\|\boldsymbol{\delta}_i(t)\|}{\|\boldsymbol{\delta}_{i,0}\|} \tag{2.25}$$

onde $\|\cdots\|$ representa a norma euclídea do vetor.

Os exponentes de Lyapunov permitem quantificar a divergência exponencial num entorno da solução de referência. Dadas n variações iniciais linearmente independentes, cada uma delas vai definir um valor de λ_k, não todos distintos, que determina o comportamento da variação na direção k do espaço de fases. O conjunto dos n exponentes de Lyapunov, $\lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_n$, assim gerados é denominado de espectro de Lyapunov do sistema (OSELEDETCH, 1968; BENETTIN *et al.*, 1980).

Uma propriedade importante dos exponentes de Lyapunov é que para sistemas conservativos se cumpre que

$$\sum_{i=1}^{n} \lambda_i = 0$$

sendo que pelo menos um dos $\lambda_i = 0$ (HAKEN, 1983)[5]. Isto implica em duas possibilidades:

[5]Este exponente está associado à direção do espaço de fases em que a solução se propaga, pois não pode haver divergência exponencial na direção de propagação da solução.

1. Todos os $\lambda_i = 0$, que constitui o que se conhece como estabilidade de primeira ordem não assintótica e é o caso que ocorre em pontos de equilíbrio estáveis ou em órbitas periódicas ou quase periódicas estáveis.

2. Existe algum $\lambda_k \neq 0$, em cujo caso deve existir algum $\lambda_i > 0$, de forma que haverá divergência exponencial de soluções vizinhas numa dada direção do espaço de fases e o sistema, a princípio, será caótico.

Neste segundo caso, é possível mostrar que, partindo de qualquer variação inicial $\boldsymbol{\delta}_0$ arbitrária, a fórmula (2.25) sempre vai fornecer o valor do máximo exponente de Lyapunov, λ_m. Isto se deve a que com o transcurso do tempo, todos os vetores $\boldsymbol{\delta}_i(t)$ tendem a se orientar na direção de máxima divergência exponencial. Assim, para obter todo o espectro de Lyapunov a partir de n variações iniciais $\boldsymbol{\delta}_{i,0}$ linearmente independentes é necessário aplicar periodicamente um processo de ortonormalização dos vetores $\boldsymbol{\delta}_i(t)$ (WOLF *et al.*, 1985).

Cabe destacar que mesmo no caso em que esteja sendo determinado λ_m apenas, ou seja, quando só é propagado um único vetor de variação $\boldsymbol{\delta}(t)$, a norma do vetor $\|\boldsymbol{\delta}\|$ pode crescer indefinidamente atingindo valores além da precisão de punto flotante da máquina (valores maiores que 10^{308} no caso de variáveis de dupla precisão). Em função disto, e para evitar o acúmulo de erros de arredondamento, torna-se necessário executar periodicamente uma renormalização do vetor variação, retornando ele ao seu estado inicial

$$\delta_k(t) \longleftarrow \delta_k(t)\frac{\|\boldsymbol{\delta}_0\|}{\|\boldsymbol{\delta}(t)\|}$$

Se esta normalização é aplicada a intervalos regulares $\mathcal{T}$, a definição do exponente de Lyapunov passa a ser

$$\lambda_\mathrm{m} = \lim_{s\to\infty} \frac{1}{s\mathcal{T}} \sum_{i=1}^{s} \ln \frac{\|\boldsymbol{\delta}(i\mathcal{T})\|}{\|\boldsymbol{\delta}_0\|}$$

onde $\boldsymbol{\delta}(i\mathcal{T})$ representa o vetor variação imediatamente antes da i-ésima renormalização (CONTOPOULUS e BARBANIS, 1989).

O tempo de Lyapunov de um sistema dinâmico é definido com $1/\lambda_\mathrm{m}$ e serve para especificar a escala de tempo característico no qual o comportamento caótico se manifesta.

2.3.3 MEGNO

Uma das limitações evidentes no cálculo de exponentes de Lyapunov reside no $\lim_{t\to\infty}$ na fórmula (2.25). Numa simulação numérica o tempo é limitado e, portanto, só podemos aspirar a obter uma valor "parcial" do exponente máximo

$$\lambda_\mathrm{m}(t) = \frac{1}{t} \ln \frac{\|\boldsymbol{\delta}(t)\|}{\|\boldsymbol{\delta}_0\|} \tag{2.26}$$

às vezes chamado de exponente de Lyapunov a tempo finito. Em sistemas fortemente caóticos, isto é, com tempos de Lyapunov curtos, a fórmula (2.26) converge rapidamente para um valor > 0. Porém, em sistemas fracamente caóticos, pode ser necessário integrar por um longo intervalo de tempo até obter alguma convergência. Mais ainda, o fato de que $\lambda(t)$ tenda sistematicamente a 0 até um dado t, não garante que ele vai manter essa tendência se o tempo aumentar. Logo, o máximo exponente de Lyapunov resulta ser extremamente ineficiente para caracterizar sistemas pouco caóticos e praticamente inútil para caracterizar sistemas estáveis.

Diversas alternativas têm sido propostas para contornar esta limitação entre as quais cabem mencionar: FLI (*Fast Lyapunov Indicator*; FROESCHLÉ *et al.*, 1997), OFLI (*Orthogonal Fast Lyapunov Indicator*; FOUCHARD *et al.*, 2002), RLI (*Relative Lyapunov Indicator*; SÁNDOR *et al.*, 2004), SALI (*Smaller ALignment Index*; SKOKOS, 2001), GALI (*Generalized ALignment Index*; SKOKOS *et al.*, 2007), SD (*Spectral Distance*; VOGLIS *et al.*, 1999), SSNs (*Spectra of Stretching Numbers*; CONTOPOULOS e VOGLIS, 1996), entre outros. Para uma revisão detalhada destes métodos ver MAFFIONE *et al.* (2011) e MAFFIONE (2012).

Nesta seção vamos considerar a alternativa conhecida como MEGNO: *Mean Exponential Growth factor of Nearby Orbits* (CINCOTTA e SIMÓ, 2000, CINCOTTA *et al.*, 2003). Este indicador se caracteriza porque permite determinar de forma muito rápida se o sistema é estável ou caótico. A ideia parte de reescrever a Equação (2.26) na forma

$$\lambda_{\mathrm{m}}(t) = \frac{1}{t}\int_0^t \frac{\dot{\delta}(t')}{\delta(t')}dt'$$

onde $\delta = \|\boldsymbol{\delta}\|$, $\dot{\delta} = \dot{\boldsymbol{\delta}} \cdot \boldsymbol{\delta} / \|\boldsymbol{\delta}\|$. Com esta definição, fica evidente que λ_{m} representa a média temporal, $\left\langle \dot{\delta}/\delta \right\rangle$, da taxa de divergência exponencial de soluções vizinhas. O conceito de MEGNO parte de uma definição semelhante:

$$Y(t) = \frac{2}{t}\int_0^t \frac{\dot{\delta}(t')}{\delta(t')}\, t'\, dt' \tag{2.27}$$

Neste caso, ao invés de calcular a média da taxa de divergência, λ_{m}, estamos calculando a média do fator de divergência, $\lambda_{\mathrm{m}} t$. É possível mostrar que, tanto no caso de órbitas periódicas ou quase periódicas estáveis, ou instáveis, ou órbitas caóticas, a grandeza $Y(t)$ apresentará um comportamento específico acrescido de termos oscilatórios com média zero. Estes termos oscilatórios fazem com que $\lim_{t\to\infty} Y(t)$ não exista e, em função disto, define-se o MEGNO como a média

$$\overline{Y}(t) = \frac{1}{t}\int_0^t Y(t')\, dt' \tag{2.28}$$

de forma a eliminar as oscilações. O MEGNO assim definido caracteriza diferentes tipos

de comportamento da seguinte forma:

- Se a solução é quase periódica estável, $\overline{Y}(t) = \lim_{t\to\infty} \overline{Y}(t) = 2$
- Se a solução é regular e próxima de uma solução periódica estável, $\overline{Y}(t) \lesssim 2$, $\lim_{t\to\infty} \overline{Y}(t) = 0$
- Se a solução é regular e próxima de uma solução periódica instável, $\overline{Y}(t) \gtrsim 2$,$\lim_{t\to\infty} \overline{Y}(t) \gg 2$
- Se a solução é caótica, $\overline{Y}(t) = \lim_{t\to\infty} \overline{Y}(t) = \dfrac{\lambda_{\mathrm{m}}}{2} t$

Notemos que nestas definições ainda é tomado o $\lim_{t\to\infty}$. Entretanto, o MEGNO apresenta duas vantagens significativas em relação ao máximo exponente de Lyapunov.

A primeira é que para uma órbita regular, $Y(t)/t \approx 2/t$ enquanto que $\lambda_{\mathrm{m}}(t) \approx \ln t/t$, logo o MEGNO converge comparativamente muito mais rápido que o exponente Lyapunov. Isto possibilita caracterizar sem ambiguidades o movimento regular em intervalos de tempo curtos, algo que, como vimos, não é possível com o exponente de Lyapunov.

A segunda vantagem é que para uma órbita caótica, $Y(t)/t \approx \lambda_{\mathrm{m}}(t)$, logo o MEGNO converge com a mesma velocidade que o exponente de Lyapunov. Entretanto, como $\overline{Y}(t) \approx \lambda_{\mathrm{m}} t/2$, não é necessário esperar a que o MEGNO convirja: basta analisar o comportamento de $\overline{Y}(t)$ sobre um intervalo de tempo curto e obter o valor de λ_{m} ajustando uma reta por mínimos quadrados. No caso mais geral, $\overline{Y}(t) = \alpha t + \beta$, e os parâmetros α, β ajustados caracterizam totalmente o tipo de movimento.

2.3.3.1 Cálculo do MEGNO

Para determinar numericamente as integrais (2.27) e (2.28), GOŹDZIEWSKI *et al.* (2001) propõe um método em que além das equações de movimento (2.17) e as variacionais (2.19), são integradas simultaneamente mais duas equações dadas por:

$$\frac{d\varphi}{dt} = \frac{\dot{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}}{\boldsymbol{\delta} \cdot \boldsymbol{\delta}} t, \qquad \frac{d\psi}{dt} = \frac{2\varphi}{t}$$

Então, conhecendo $\varphi(t), \psi(t)$, calcula-se

$$Y(t) = \frac{2\varphi}{t}, \qquad \overline{Y}(t) = \frac{\psi}{t}$$

Este método pode ser facilmente utilizado com integradores não simplécticos, em que a incorporação de mais EDOs não oferece dificuldades, porém não se adapta ao esquema simpléctico apresentado na seção 2.3.1. Além disso, este método requer a determinação em cada passo de integração de $\dot{\boldsymbol{\delta}}$, que no esquema simpléctico não é calculado.

Uma alternativa de cálculo para o caso de integradores simplécticos foi apresentada por BREITER *et al.* (2005), com base na definição de MEGNO para sistemas discretos (CINCOTTA *et al.*, 2003), e consiste na aplicação de duas fórmulas recursivas:

$$\begin{aligned} Y(t_n) &= \frac{n-1}{n}\, Y(t_{n-1}) + 2 \ln \frac{\delta(t_n)}{\delta(t_{n-1})} \\ \overline{Y}(t_n) &= \frac{(n-1)\,\overline{Y}(t_{n-1}) + Y(t_n)}{n} \end{aligned} \tag{2.29}$$

onde n é o número do passo, isto é, $t_n = t_0 + n\tau$, e as condições iniciais são $\delta(t_0) = \delta_0$, $Y(t_0) = \overline{Y}(t_0) = 0$. Esta é a estratégia aplicada no nosso código.

Cabe destacar que, independentemente do formato adotado para o cálculo do MEGNO, continua sendo necessário renormalizar periodicamente o vetor variação para evitar o acúmulo de erros de arredondamento. A renormalização é introduzida a intervalos regulares na Equação (2.29) de forma que $\delta(t_{n-1}) \longrightarrow \delta_0$.

2.3.4 Estimadores de difusão

Como vimos no início da seção 2.3, é possível utilizar o conceito de difusão para quantificar comportamentos caóticos. A principal vantagem destes indicadores, quando comparados ao exponente de Lyapunov ou o MEGNO, é que eles dependem apenas da evolução temporal dos momentos do problema não perturbado, logo não é necessário resolver as equações variacionais do sistema.

Em nosso código decidimos implementar os seguintes estimadores:

- Variância em semieixo maior e excentricidade orbital, com respeito à condição inicial:

$$\begin{aligned} VA_i(t_n) &= \frac{1}{s-1} \sum_{k=0}^{s} \left[a_i(t_k) - a_i(t_0)\right]^2 \\ VE_i(t_n) &= \frac{1}{s-1} \sum_{k=0}^{s} \left[e_i(t_k) - e_i(t_0)\right]^2 \end{aligned} \tag{2.30}$$

  onde de acordo com o termo kepleriano da Equação (2.15), o semieixo maior da órbita do corpo i é

$$a_i = -\frac{G m_0 m_i}{2\, H_{\mathrm{Kep}}(\vec{R}_i, \vec{P}_i)}$$

  e a sua excentricidade é

$$e_i = \sqrt{1 - \frac{\left|\vec{R}_i \times \vec{P}_i\right|^2}{G m_0 m_i^2 a_i}}$$

  O semieixo e a excentricidade não são momentos canônicos, mas possuem análogos

canônicos que constituem as denominadas variáveis de Delaunay-Poincaré. Eles estão vinculados às integrais de movimento do problema kepleriano, em particular a energia e o módulo do momento angular.
A variância é um estimador muito útil, pois nos fornece uma ideia de quão espalhada está a solução em torno do seu valor médio. Se a solução é regular, espera-se que os momentos do problema perturbado permaneçam próximos dos valores constantes do problema não perturbado e, nesse caso, a variância dos momentos deve ser pequena. Por outro lado, se a solução é caótica, os momentos do problema perturbado podem exibir grandes variações, que se traduzem numa variância alta.
Notemos que se consideramos uma grade uniforme de condições iniciais no espaço (a, e), os valores médios dessas soluções não se distribuem numa grade uniforme no espaço $(\overline{a}, \overline{e})$. Como estamos interessados em analisar a estabilidade do sistema sobre uma grade uniforme de órbitas, resulta mais adequado computar a variância não em relação à média, mas em relação ao valor inicial.
Como vimos, estes valores de VA e VE poderiam ser vinculados a coeficientes de difusão e exponentes de Hurst.

- Máxima excentricidade orbital:

$$ME_i(t_n) = \max_{k \leqslant n} e_i(t_k) \tag{2.31}$$

Na dinâmica planetária, a máxima excentricidade é um parâmetro relevante para avaliar a estabilidade das órbitas. Este parâmetro não tem a ver com difusão, nem com divergência de trajetórias vizinhas. Ele fornece informação sobre a possibilidade de uma órbita se tornar muito excêntrica e começar a cruzar as órbitas dos outros corpos, favorecendo eventuais encontros próximos que podem desestabilizar o sistema planetário.

- Máxima variação em excentricidade:

$$MVE_i(t_n) = \max_{k \leqslant n} |e_i(t_k) - e_i(t_0)| \tag{2.32}$$

O MVE substitui com vantagens ao estimador ME, porque este último sempre fornece valores altos quando a excentricidade inicial é alta.

Uma limitação das variâncias e o MVE é que variações seculares da órbita com grande amplitude podem gerar valores altos dos indicadores, levando a identificar erroneamente um comportamento regular como caótico. Entretanto, isto pode ser desambiguado notando que nesse caso a variância terá um valor alto, porém constante (o que equivale a que o coeficiente de difusão associado é zero). Em qualquer caso, é sempre recomendável utilizar vários indicadores de caos simultaneamente para conseguir caracterizar em forma

adequada o comportamento do sistema. Por último cabe mencionar que, diferentemente do MEGNO, não há valores absolutos de cada indicador que definam se uma dada solução é regular ou cáotica. Os valores dos indicadores devem ser sempre analisados de forma comparativa em relação aos valores das soluções vizinhas na grade.

2.3.5 Mapas dinâmicos

O nosso principal objetivo na determinação de estimadores de caos é o de construir mapas dinâmicos, isto é, mapear o valor de um dado indicador de caos em cima de uma grade de condições iniciais, permitindo assim identificar quais regiões do espaço de fases são regulares e quais são caóticas. Como veremos no capítulo 4, nas nossas aplicações a sistemas extrassolares, em geral, escolhe-se um planeta do sistema sob estudo para fazer este mapeamento, de maneira que dois dos parâmetros dinâmicos iniciais deste planeta são variados ao longo da grade, enquanto que os restantes (tanto deste planeta como dos restantes do sistema) são mantidos inicialmente fixos. A escolha de quais parâmetros definem a grade e dos limites da grade dependerá do tipo de análise que se deseja fazer, sendo que em nosso caso os limites das grades, em geral, estão vinculados às incertezas nos valores dos parâmetros que as definem. Assim por exemplo, se um planeta tem o seu semieixo e a sua excentricidade determinados com certos erros $\Delta a, \Delta e$, a grade de condições iniciais costuma ser escolhida nos intervalos $a \pm \Delta a$ e $e \pm \Delta e$.

Aqui é necessário mencionar que uma diferença importante entre o MEGNO e os estimadores de difusão definidos na seção anterior, além da diferença conceitual, é que o primeiro caracteriza a estabilidade do sistema como um todo, enquanto que os segundos são calculados encima de apenas uma das órbitas do sistema. Isto pode ter vantagens ou desvantagens dependendo do problema. Por exemplo, num sistema com vários planetas em que alguns possuem um comportamento regular enquanto outros mostram um comportamento caótico, o MEGNO vai caracterizar todo o sistema como caótico mas não vai ser possível discriminar o comportamento individual dos planetas. Já no caso dos estimadores de difusão, será necessário certo cuidado ao escolher o planeta que será usado para fazer os cálculos. Em particular, na construção de um mapa dinâmico, a escolha natural para o cálculo dos índices de difusão é precisamente o planeta cujas condições iniciais variam sobre a grade, mas isto não é um critério rígido.

Capítulo 3

Metodologia

A programação imperativa é o paradigma de programação que usa declarações ou comandos que são dados para o computador executar e que alteram o estado de um programa. A maioria das linguagens de programação com as quais estamos acostumados a lidar são linguagens imperativas: `C`, `C++`, `FORTRAN`, `Python`, etc. A programação imperativa se concentra em descrever como um programa opera, em contraste com a programação declarativa, que se concentra no que o programa deve realizar sem especificar como o programa deve alcançar o resultado. Este último paradigma é o das linguagens de *markup* como `HTML`, `XML`, entre outras.

Normalmente, quando se aprende programação imperativa, trabalha-se com algoritmos sequenciais, que consistem em conjuntos de passos ou instruções que são executados um após o outro. Um algoritmo paralelo consiste em selecionar alguns dos passos de um algoritmo sequencial para serem executados ao mesmo tempo em processos distintos. Esses processos podem ser distribuídos em diferentes núcleos de um mesmo processador, ou em processadores de máquinas diferentes que se comunicam entre si, ou ainda, em processadores de uma placa de vídeo. Um passo executado em forma paralela não pode depender de resultados gerados pelos outros passos que estão sendo executados concomitantemente. Além disso, os passos que são executados ao mesmo tempo não podem interferir entre si, de forma a evitar o que se conhece como "condição de corrida" (do inglês *race condition*). Uma condição de corrida ocorre quando um programa, para funcionar corretamente, depende da sequência ou do tempo de execução dos diferentes processos incluídos no programa. Condições críticas de corrida geralmente acontecem quando vários processos dependem de algum estado compartilhado e podem causar erros de execução não determinísticos, com resultados imprevisíveis. Operações sobre estados compartilhados, como por exemplo a leitura e escritura de variáveis globais ou o acesso a setores compartilhados da memória, são seções críticas que devem ser executadas de forma mutuamente exclusiva para não correr o risco de corromper o estado compartilhado.

O principal objetivo da paralelização é diminuir o tempo de execução do código e obter assim uma melhora no desempenho. Um objetivo secundário é também fazer um

melhor aproveitamento dos recursos computacionais disponíveis em ambientes de *cluster*. É importante salientar que nem sempre as tarefas são paralelizáveis. Por exemplo, na construção de uma casa de dois andares, poderia se atribuir a construção de cada andar a um pedreiro diferente, porém o segundo andar só poderá ser construído após a finalização do primeiro. Já a pintura de cada andar poderia ser executada em forma simultânea por pintores diferentes. Por outro lado, deve-se destacar que nem sempre a paralelização de todo ou de partes de um código acarreta uma melhoria no desempenho do programa. Isto se deve a que a paralelização de tarefas envolve sempre um tempo de latência ou de sobrecarga, que às vezes pode ser maior do que o tempo de execução individual de cada tarefa. Em modelos de paralelização com memória distribuída, como `MPI`, o tempo de sobrecarga está condicionado, principalmente, pela velocidade de transmissão de mensagens entre as tarefas. Em modelos de paralelização com memória compartilhada, como `OpenMP`, a sobrecarga está vinculada ao processo de bifurcação-junção das tarefas[1]. A paralelização de um processo em n tarefas modifica o tempo de execução do processo por um fator que é sempre maior do que $1/n$.

3.1 CUDA

Como mencionemos na seção 1.1, um avanço importante na programação paralela ocorreu com o advento do uso das unidades de processamento gráfico ou GPUs. Atualmente, as GPUs possuem milhares de microprocessadores e, dependendo da aplicação, o ganho na eficiência computacional pode ser significativo. Outra vantagem já mencionada é o seu baixo custo de investimento. Mas para que se possa explorar toda a capacidade das GPUs é necessário conhecer sua arquitetura, que é bem diferente da arquitetura das CPUs. Esta seção está baseada em informação que pode ser encontrada no site da NVIDIA (`https://developer.nvidia.com/nvidia-developer-zone`) e na GUIA DE PROGRAMAÇÃO EM CUDA-C, VER. 3.2 (2010).

A arquitetura das GPUs, ou seja a forma como o hardware é organizado na placa, foi projetada para executar algoritmos com alta demanda computacional e elevada intensidade numérica, isto é, algoritmos que possuem uma grande quantidade de cálculos aritméticos em comparação com o número de acessos à memória, e nos quais uma mesma implementação pode ser usada para operar em várias partes do conjunto de dados de forma paralela. Estas características permitem explorar a chamada ocultação de latência. Quando um fluxo de execução faz um acesso à memória, que é uma operação demorada, ele é bloqueado e outro fluxo que está disponível mas não está sendo executando começa a ser executado. Isso faz com que o tempo teoricamente perdido pelo acesso à memória

[1] O modelo de bifurcar-juntar (do inglês *fork-join*) é uma maneira de configurar e executar programas paralelos, de modo que a execução se ramifique paralelamente em pontos designados do programa, para se mesclar em um ponto subsequente e retomar a execução sequencial.

por um fluxo seja compensado pela execução de outro fluxo. Outro aspecto importante da arquitetura das GPUs é a unidade de controle, que é a responsável por gerenciar o que será executado e como será executado. No caso das CPUs, a unidade de controle tem uma complexidade maior já que, para aumentar a eficiência dos algoritmos sequenciais, utilizam-se diversos mecanismos como executar passos sequenciais fora de ordem ou em paralelo sempre que possível (a denominada paralelização em nível de instrução). Além disso, as CPUs possuem uma memória cache[2] de tamanho bem maior do que as GPUs. No caso das GPUs, como um trecho do código pode ser executando várias vezes em paralelo, não é necessário uma unidade de controle tão complexa como a de uma CPU e a memória cache também é menor, porque o processo de ocultação de latência já reduz a perda de tempo pelos acessos à memória principal. Desta forma, como a unidade de controle e a memória cache são menores, sobra mais espaço físico no chip para serem adicionados mais núcleos de execução.

A linguagem `CUDA` foi implementada em 2006 como uma API de programação para placas gráficas da NVIDIA. É possível programar em `CUDA` usando diretamente instruções de máquina, mas a forma mais comum é usar extensões de linguagens como `C` ou `C++`. Em particular, no código que foi implementado no trabalho de doutorado foi usada a extensão `CUDA-C`, que consiste num conjunto de comandos incorporados à linguagem `C` padrão para que trechos do código sejam executados na placa gráfica.

Um programa em `CUDA` consiste de várias fases, algumas executadas em CPU e outras em GPU. As fases que não possuem ou possuem pouco paralelismo são executadas na CPU, enquanto que as fases que são altamente paralelizáveis são executadas na GPU. A seguir serão apresentados alguns exemplos de código em `CUDA-C`, que serão usados para explicar os comandos e os conceitos principais da programação nesta linguagem.

3.1.1 *Threads* e *kernel*

Consideremos o exemplo apresentado na Figura 3.1. Este código tem como objetivo realizar a operação vetorial $\vec{c} = \vec{a}k + \vec{b}$, onde $\vec{a}$,$\vec{b}$ são dados de entrada e k é um número escalar. Para se calcular o vetor resultante, $\vec{c}$, temos que calcular cada componente i do vetor, que requer das componentes i dos vetores $\vec{a}$ e $\vec{b}$. Como para o cálculo de cada componente de $\vec{c}$ não se precisa do valor das outras componentes, o cálculo de cada componente pode ser realizado em paralelo. O exemplo da figura mostra a implementação usando GPU.

A linha 1 define uma constante que será a dimensão dos vetores usados no código. A linha 7 define a função `main`, que é a função principal em `C`. A partir dela são chamadas todas as outras funções do programa. Na linha 8 tem a declaração das variáveis

[2]Memória cache é um tipo de memória temporária, de mais rápido acesso mas com menor capacidade. Assim, em vez da CPU ter que acessar a memória RAM, mais lenta, o acesso é feito à memória cache e apenas se o dado não estiver nela a memória RAM é acessada.

```
1  #define N 1000 //N igual ao número de elementos do vetor
2  //kernel para execução paralela na GPU
3  __global__ void somaVetoresPar(const float *a, const float *b,
       float *c, float k) {
4      int i = threadIdx.x;
5      c[i] = a[i] * k + b[i];
6  }
7  int main() {
8      float a[N], b[N], c[N];
9      //inicializa os vetores a e b
10     ...
11     // invoca o kernel com um bloco de n threads
12     somaVetoresPar<<<1, N>>>(a, b, c, k);
13     ...
14 }
```

Figura 3.1: Exemplo de *kernel* `CUDA` para a operação vetorial $\vec{a}k + \vec{b}$.

do programa, que é necessária em um programa `C`. As variáveis são do tipo `float` que representa um número real de 32 bits. Na linha 9 se faz a alocação de espaço de cada um dos vetores e se inicializam os seus valores, seja a partir do I/O padrão, seja inserindo os valores diretamente no código. Até esse momento, o código é um programa sequencial comum em `C` e a partir da linha 12 se inicia a implementação do paralelismo em `CUDA`. Na linha 12 é invocado o *kernel*, que é um tipo de função especial que só é executada na GPU. A sintaxe se assemelha com uma chamada de função simples ou de sub-rotina, mas entre o nome da função e a lista de argumentos entre parênteses existem os símbolos `<<<` e `>>>` que indicam que a chamada corresponde a um *kernel* em vez de uma função comum. Os parâmetros dentro dos símbolos `<<<` e `>>>` representam o número de blocos de *threads*, que serão explicados mais adiante, e o número de *threads* em cada bloco. No nosso caso, são usados `N` *threads* em um único bloco. Cada *thread* na GPU, que é um fluxo paralelo de execução, corresponderá a uma chamada da função *kernel* que é definida na linha 3. A função *kernel* é definida de forma similar a uma função comum em `C`, sendo a única diferença a diretiva `__global__` que indica que a função é um *kernel* e será executada na GPU.

Quando o *kernel* é invocado, são criadas a quantidade de *threads* de acordo com os parâmetros da chamada do *kernel*, no caso 1000 *threads*, e cada uma vai executar o *kernel* de forma independente e paralela. Assim, cada *thread* vai calcular uma componente do vetor $\vec{c}$ de acordo com a fórmula que está especificada no *kernel*. Aqui pode se pensar que surge, aparentemente, um problema: se temos várias *threads* e cada uma delas executa o cálculo de uma componente do vetor, como saberemos qual componente esta calculando uma dada *thread* e se duas *threads* não estão calculando a mesma componente? A resposta está na variável `threadIdx.x`. Quando se invoca o *kernel* e se criam as *threads*, é atribuído a cada uma delas um identificador único através da variável `threadIdx.x`, que varia de

`0` a `N-1`. Basta então identificar a i-ésima componente do vetor como o valor deste identificador, o que é feito na definição do próprio *kernel*, nas linhas 4 e 5.

3.1.2 Espaço de memória da GPU

Na seção anterior vimos que é necessário definir as funções *kernel* que serão executadas na GPU. Todas as variáveis que são utilizadas dentro de um *kernel* precisam ser ponteiros, ou seja, variáveis que guardam um endereço de memória ao invés de um dado, que contenham endereços de memória da GPU ou dados contidos em registradores. Quando os dados são enviados por valor, é armazenada uma cópia local nos registradores da *thread* para ser usada no cálculo da execução do *kernel* pela *thread*. Então, antes de invocarmos o *kernel*, como foi feito no exemplo da Figura 3.1, devemos transferir os dados da memória RAM da CPU para a memória da GPU. O código apresentado na Figura 3.2 mostra como isto é implementado.

O trecho entre as linhas 3 e 8 apresenta o *kernel* que será executado pelas *threads*. As variáveis `a`, `b` e c são ponteiros para endereços de memória da GPU, enquanto que o restante das variáveis usam o valor que é passado na invocação do *kernel* e que é armazenado em seus registradores respectivos, na memória local de cada *thread*. Veremos mais sobre as memórias da GPU na próxima seção.

As linhas 10 a 12 apresentam a declaração das variáveis que serão usadas no programa. Todas são ponteiros que, depois de alocados, serão os vetores usados nos cálculos. A princípio, elas podem ser ponteiros da memória RAM ou da GPU, isso só será definido quando for realizada a alocação de espaço com as funções adequadas. Por conveniência, foi usada uma variável com o nome do vetor $\vec{a}$,$\vec{b}$ ou $\vec{c}$ para representar os vetores na memória da CPU, e as variáveis correspondentes com `d_` na frente para representar os equivalentes na memória da GPU.

As linhas 15 a 18 aplicam a função `malloc`, que é usada para reservar uma área de memória na memória RAM. Após essa chamada, o ponteiro representado pela variável a, por exemplo, apontará para o primeiro elemento do vetor $\vec{a}$. Este elemento é referido como `a[0]`, e para usar o componente i do vetor basta chamar `a[i]`. De forma análoga são alocados espaços de memória para os vetores $\vec{b}$ e $\vec{c}$.

As linhas 23 a 25 fazem uma reserva de espaço de memória na GPU. A função `cudamalloc` recebe o ponteiro que depois apontará para o primeiro elemento do vetor e o número de bytes alocados. O número de bytes é igual ao número de elementos do vetor multiplicado pelo tamanho de um dado do tipo `float`, que pode ser obtido pela função `sizeof`. Após a reserva de memória, os vetores `d_a`, `d_b` e `d_c` funcionarão analogamente aos vetores alocados na memória da CPU, com a diferença que os dados estarão na GPU em vez da CPU.

Após a alocação de memória na CPU e na GPU temos que inicializar os vetores.

```
#include <stdio.h>
//kernel para execução paralela na GPU
__global__ void somaVetoresPar(const float *a, const float *b,
    float *c, float k, int n) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if(i < n) {
      c[i] = a[i] * k + b[i];
   }
}
int main() {
   float *h_a, *d_a;
   float *h_b, *d_b;
   float *h_c, *d_c;
   n_bytes = N * sizeof(float); //N é a dimensão dos vetores

   //aloca memória na CPU (host)
   h_a = (float*) malloc (n_bytes);
   h_b = (float*) malloc (n_bytes);
   h_c = (float*) malloc (n_bytes);

   //inicializa os vetores h_a e h_b
   ...
   //aloca espaço para os vetores na GPU
   cudaMalloc((void**) &d_a, n_bytes);
   cudaMalloc((void**) &d_b, n_bytes);
   cudaMalloc((void**) &d_c, n_bytes);

   //copia os vetores de entrada da CPU para a GPU (host para
       device)
   cudaMemcpy(d_a, h_a, n_bytes, cudaMemcpyHostToDevice);
   cudaMemcpy(d_b, h_b, n_bytes, cudaMemcpyHostToDevice);

   //dispara o kernel
   int n_threads = 1024; //número de threads por bloco
   int n_blocos = (N + n_threads-1) / n_threads; //número de
       blocos
   somaVetoresPar<<<n_blocos,n_threads>>>(d_a, d_b, d_c, k, N);

   //copia o resultado da GPU para a CPU (device para host)
   cudaMemcpy(h_c, d_c, n_bytes, cudaMemcpyDeviceToHost)

   //libera a memória na GPU
   cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

   //usa o vetor C
   ...
   //libera a memória na CPU
   free(h_a); free(h_b); free(h_c);

   //limpa o estado da GPU e termina
   cudaDeviceReset();
   return 0;
}
```

Figura 3.2: Exemplo de transferência de dados entre a memória RAM e a GPU.

Os vetores na CPU podem ser inicializados a partir da entrada dada pelo usuário ou diretamente no código. Mas isso não é possível para os vetores na GPU. A única forma de inicializá-los é transferindo uma cópia dos dados da área em que se encontram na memória RAM para a área da GPU. Isso é realizado pela função `cudaMemCpy`. Observando a linha 28, o primeiro argumento é o vetor destino e o segundo é o vetor origem, ou seja, de onde os dados vão ser copiados para o vetor destino. Se o primeiro vetor é o da GPU e o segundo o da CPU, ou o contrário, é definido pelo último argumento da função. Quando este último argumento for `cudaMemcpyHostToDevice`, significa que o vetor destino é um vetor na GPU e o vetor origem está na CPU. O terceiro argumento é quantidade de bytes a ser transferida, que corresponde ao número de elementos do vetor multiplicado pelo tamanho de cada dado.

A partir da linha 32, os dados já estão na GPU e prontos para ser executados. A linha 32 define o número de *threads* e a linha 33 o número de blocos. O conjunto total de *threads* é chamado de grade. A grade é subdividida em grupos de *threads* chamados blocos. Poderia ser possível usar um único bloco com as *threads*, mas por questões de recursos computacionais que serão detalhados mais adiante, existe um número máximo de *threads* em cada bloco. Então a partir de um certo número de *threads* é necessário dividir a grade em blocos de *threads*. A linha 33 determina o número de blocos necessários para guardar o número de *threads* definido na linha 32.

Na linha 34 se invoca o *kernel*, onde são criados os blocos de *threads* e cada um executa uma instância do *kernel*. O primeiro argumento da invocação é o número de blocos e o segundo é o número de *threads* em cada bloco.

A linha 37 copia o vetor $\vec{c}$ da memória da GPU para a memória RAM da CPU. Note-se que não é necessário copiar os demais vetores já que não foram alterados. A chamada da função é semelhante a chamada na linha 28. A diferença é que o vetor destino é está na memória RAM, o vetor origem está na GPU e o último argumento determina que a cópia será de uma área da GPU para a memória RAM.

As linhas finais são para liberar os espaços de memória alocados pelo programa: `cudafree` libera a área apontada por um vetor na memória da GPU, enquanto a função `free` faz o mesmo para um vetor na memória RAM.

3.1.3 Verificação de erros

Em todas chamadas de funções de `CUDA` podem ocorrer erros. Todas as funções retornam um código de erro que indica se não houve erro, em caso de sucesso, ou o tipo de erro caso eles ocorram. Portanto é recomendável verificar se as funções chamadas foram executadas com sucesso ou não. Isto pode ser feito através do código apresentado na Figura 3.3.

Na linha 2 definimos uma função que verifica se houve um erro ou não na chamada da função. Toda função de `CUDA` retorna um valor do tipo `cudaError_t`. Esta variável

```
1  //wrapper para checar erros nas chamadas de funções de CUDA
2  #define CUDA_SAFE_CALL(call) { \
3     cudaError_t err = call;     \
4     if(err != cudaSuccess) {    \
5        fprintf(stderr,"Erro no arquivo '%s', linha %i: %s.\n", \
6                __FILE__, __LINE__, cudaGetErrorString(err));   \
7        exit(EXIT_FAILURE);
8     }
9  }
10 //kernel para execução paralela na GPU
11 __global__ void somaVetoresPar(const float *a, const float *b,
       float *c, float k, int n) {
12    int i = blockIdx.x * blockDim.x + threadIdx.x;
13    if(i < n) {
14       c[i] = a[i] * k + b[i];
15    }
16 }
17 int main() {
18    float *h_a, *d_a;
19    float *h_b, *d_b;
20    float *h_c, *d_c;
21    n_bytes = N * sizeof(float); //N é a dimensão dos vetores

23    //aloca memória na CPU (host)
24    h_a = (float*) malloc(n_bytes);
25    h_b = (float*) malloc(n_bytes);
26    h_c = (float*) malloc(n_bytes);
27    if(h_a==NULL)||(h_b==NULL)||(h_c==NULL) exit(EXIT_FAILURE);

29    //inicializa os vetores h_a e h_b
30    ...
31    //aloca espaço para os vetores na GPU
32    CUDA_SAFE_CALL(cudaMalloc((void**) &d_a, n_bytes));
33    CUDA_SAFE_CALL(cudaMalloc((void**) &d_b, n_bytes));
34    CUDA_SAFE_CALL(cudaMalloc((void**) &d_c, n_bytes));

36    //copia os vetores da CPU para a GPU (host para device)
37    CUDA_SAFE_CALL(cudaMemcpy(d_a,h_a,n_bytes,
          cudaMemcpyHostToDevice));
38    CUDA_SAFE_CALL(cudaMemcpy(d_b,h_b,n_bytes,
          cudaMemcpyHostToDevice));

40    //dispara o kernel
41    int n_threads = 1024; //número de threads por bloco
42    int n_blocos = (N+n_threads-1)/n_threads;//número de blocos
43    somaVetoresPar<<<n_blocos,n_threads>>>(d_a, d_b, d_c, k, N);
44    CUDA_SAFE_CALL(cudaGetLastError());

46    //copia o resultado da GPU para a CPU (device para host)
47    CUDA_SAFE_CALL(cudaMemcpy(h_c,d_c,n_bytes,
          cudaMemcpyDeviceToHost))

49    //libera a memória na GPU
50    CUDA_SAFE_CALL(cudaFree(d_a));
51    CUDA_SAFE_CALL(cudaFree(d_b));
52    CUDA_SAFE_CALL(cudaFree(d_c));
53    ...
54 }
```

Figura 3.3: Exemplo de código para verificação de erro nas chamadas às funções `CUDA`.

tem o valor de `cudaSuccess` se a chamada foi executada com sucesso ou um código de erro caso contrário. Então a função recebe a variável de erro, verifica se foi executada com sucesso ou, se houve algum erro, imprime uma mensagem na tela. Essa mensagem é impressa usando a função `cudaGetErrorString`, que recebe o código de erro e retorna uma *string* com a sua descrição.

Nas linhas 32 a 34, 37, 38, 44, 47 e 50 a 52, as funções de `CUDA` que usamos anteriormente são chamadas como argumento da função definida na linha 2. Essas funções, então, retornam a variável que indica se houve erro e eventualmente imprimem a mensagem respectiva. Isso é útil para sabermos onde e por que ocorreram os erros e auxilia na sua solução.

3.1.4 Sincronização

Em algumas situações é necessário sincronizar as *threads* que estão sendo executadas numa GPU. Pode ser que parte do *kernel* dependa de que todas as *threads* estejam num certo ponto de execução antes de prosseguir com a execução do resto do algoritmo. A seguir comentaremos os tipos de sincronização que estão disponíveis na execução de um programa `CUDA`.

3.1.4.1 Sincronismo de bloco

Este tipo de sincronismo é realizado entre as *threads* de um mesmo bloco. Quando em algum ponto do *kernel* se executa o comando `__syncthreads`, todas as *threads* do bloco tem que alcançar essa parte do código para que o restante do *kernel* seja executado pelas *threads* do bloco. Um exemplo disso é apresentado na Figura 3.4.

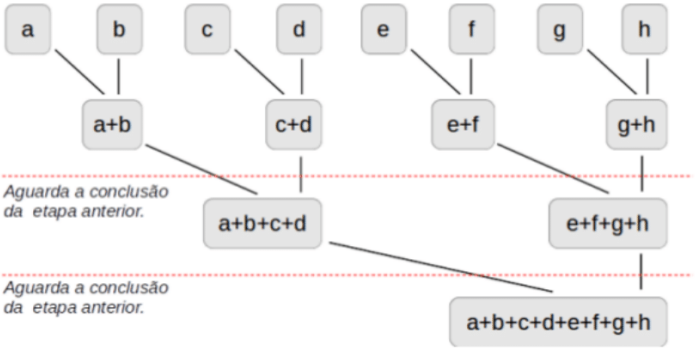

Figura 3.4: Exemplo de sincronismo de bloco para o somatório.

Vamos supor que exista um *kernel* para executar o somatório semi-paralelo dos números de `a` até `h`. Uma forma de fazer isso é somar os números consecutivos e guardar o seu resultado na posição do número à direita. Nesse caso, a posição de memória onde os valores estão sendo armazenados torna-se relevante. No exemplo da figura, o resultado

da soma de `a+b` é armazenado na posição `b`, o resultado da soma de `c+d` é armazenado na posição `d`, e assim por diante com os demais valores envolvidos. Num segundo passo, somam-se os valores das posições `b` e `d` e se guarda o resultado na posição `d`. Assim, a soma dos números de `a` até `d` estará armazenada na posição `d`.

Vale ressaltar que é necessário garantir que as somas do primeiro passo estejam todas concluídas para depois realizar as somas do segundo passo. Como as somas levam em conta apenas os valores que estão armazenados em determinadas posições, caso alguma operação do segundo passo seja executada antecipadamente, os valores envolvidos podem não estar atualizados.

3.1.4.2 Sincronismo de *kernel*

O tipo de sincronização que vimos anteriormente só pode ser realizados dentro de um mesmo bloco. Então, por exemplo, se a placa tem um limite de 1024 *threads* por bloco, e temos que realizar o somatório de 2048 números, temos que criar 2 blocos para calcular o somatório, mas não podemos sincronizar *threads* de blocos distintos da forma anterior. Uma alternativa é realizar o que se denomina de sincronização através de *kernel*. No exemplo da Figura 3.4, podemos criar um *kernel* que calcule o somatório das *threads* de um bloco e guarde esse valor em outro vetor. Logo após podemos chamar o mesmo *kernel* para somar o vetor que contém os valores da somas parciais de cada bloco. Como para um *kernel* terminar é necessário que todas as *threads* terminem, então antes da segunda chamada do *kernel* é garantido que todas as somas parciais foram efetuadas antes do somatório global. Dessa forma, é possível sincronizar *threads* de blocos diferentes.

3.1.5 Modelo de programação

A arquitetura de uma placa gráfica pode ser muito complexa de se compreender dependendo de quem está programando. Em função disso, foi desenvolvido um modelo de programação que independe de como a arquitetura da placa foi projetada. Dessa forma, o programador pode se abstrair da parte física da placa e programar apenas utilizando este modelo lógico. O modelo é constituído de um espaço de programação denominado de grade. Essa grade é dividida em blocos e cada bloco é dividido em *threads*. Voltando ao exemplo da soma vetorial, podemos dividir o problema em problemas menores, que constituem a soma de cada uma das componentes e guardar esse resultado numa componente do vetor soma. Cada soma de componentes será executada em paralelo em uma *thread*. Quando todas as *threads* tiverem executado suas tarefas, o bloco de *threads* é completado. Quando todos os blocos terminarem a execução, a grade completa sua execução e o *kernel* é terminado.

Tanto a grade quanto os blocos podem ter de uma a três dimensões. Ou seja, um bloco por exemplo é dividido em *threads* que podem estar alinhadas como um vetor (uma

dimensão), uma matriz (duas dimensões) ou um tensor (três dimensões). O número de dimensões depende do tipo de problema a ser resolvido.

Cada *thread* pode ser identificada univocamente dentro de um bloco, assim como cada bloco pode ser identificado univocamente dentro da grade, por meio das suas coordenadas. Assim por exemplo, se o domínio de dados for unidimensional, como numa operação vetorial, a grade pode ser pensada como um vetor de blocos e cada bloco como um vetor de *threads*. Dessa forma, cada *thread* terá uma coordenada em relação ao bloco, e cada um bloco terá uma coordenada em relação à grade. Supondo que tenhamos apenas um bloco, pode-se associar uma componente do vetor soma ao identificador da *thread* dentro do único bloco. Então, por exemplo, a *thread* 25 calculará a componente 25 do vetor soma. No caso de mais blocos, temos que utilizar os identificadores da *thread* no bloco e o do bloco na grade para gerar um identificador único e associá-lo à componente do vetor soma.

Os casos bidimensional e tridimensional são semelhantes ao caso unidimensional, só que cada *thread* terá um identificador para cada direção dentro do bloco, e cada bloco também terá um identificador para cada direção dentro da grade. Então, usaremos o identificador `x` da *thread* e o identificador `x` do bloco para calcular o identificador global na direção `x`. O mesmo se aplica para as direções `y` e `z`. Um caso bidimensional simples seria o cálculo da multiplicação de duas matrizes. Como o resultado é uma matriz, que é uma estrutura bidimensional, pode-se associar cada *thread* a uma posição da matriz produto. Então no caso de um único bloco, por exemplo, podemos construir um bloco bidimensional de *threads* e associar a cada *thread* com uma componente `(i,j)` da matriz produto. Assim, a posição `(2,3)` da matriz produto pode ser associada à *thread* com identificadores `x=2` e `y=3` dentro do bloco, e a mesma vai utilizar a linha 2 da primeira matriz do produto e a coluna 3 da segunda matriz.

Se o número de *threads* for menor que o limite máximo permitido num bloco, então é necessário usar apenas um bloco e o identificador da *thread* na grade é o mesmo que o identificador da *thread* no bloco. Mas se o número de *threads* for maior, é necessário dividir a grade em mais blocos e cada bloco em *threads*. Nesse caos, o identificador da *thread* na grade deverá usar uma combinação dos identificadores da *thread* no bloco e do identificador do bloco na grade para calcular a sua posição na grade. A variável `threadIdx.x` guarda o identificador local da *thread* no bloco na direção `x`. Analogamente existe um identificador nas direções `y` e `z` se o bloco for dividido de forma bidimensional ou tridimensional. A variável `blockIdx.x` guarda o identificador do bloco na grade na direção `x`, e também existem identificadores nas demais direções se for o caso. Além disso, existe mais uma variável chamada `blockDim.x` que guarda o número de *threads* que existe em cada linha do bloco, assim como existem os equivalentes para as outras duas direções. Portanto, para identificar univocamente uma *thread* na grade, devemos utilizar a seguinte

fórmula:

$$\texttt{i} = \texttt{blockIdx.x} * \texttt{blockDim.x} + \texttt{threadIdx.x}$$
$$\texttt{j} = \texttt{blockIdx.y} * \texttt{blockDim.y} + \texttt{threadIdx.y}$$
$$\texttt{k} = \texttt{blockIdx.z} * \texttt{blockDim.z} + \texttt{threadIdx.z}$$

onde `i,j,k` são os identificadores da *thread* em relação à grade nas direções `x`,`y` e `z` respectivamente.

3.1.6 Modelo de execução

Na seção anterior, discutimos o modelo de programação, que é uma abstração lógica para o funcionamento da placa gráfica. Como explicamos, o programador não precisa saber os detalhes sobre o hardware da placa gráfica para fazer um programa em `CUDA`, bastando apenas usar o modelo de programação para escrever o código a partir dele. Mas para desenvolver programas mais eficientes, é necessário entender como funciona a arquitetura da placa e como o código é executado pela arquitetura da placa. Nessa seção, abordaremos o modelo de execução das placas gráficas.

As menores unidades de processamento das placas gráficas são chamadas de *cores*, que são o equivalente a um núcleo de um processador de CPU. Vários *cores* são agrupados num multiprocessador. Quando um conjunto de *threads* é selecionado para execução, eles são executados pelos *cores* de um multiprocessador. Por esse motivo, se diz que o multiprocessador é a menor unidade de execução paralela das placas gráficas. Cada bloco de *threads* é selecionado para ser executado num multiprocessador. Dependendo do tamanho do bloco, em relação ao número de *threads* que o compõe e do limite de *threads* por multiprocessador, mais de um bloco pode ser alocado ao mesmo tempo num mesmo multiprocessador. Quando um bloco é selecionado para executar num multiprocessador, ele permanece alocado até que todas as suas *threads* terminem a execução do *kernel* e somente após isso um outro bloco, caso algum esteja disponível para execução e ainda não tiver sido alocado, é selecionado para executar nesse multiprocessador.

Quando um bloco é selecionado para ser executado num multiprocessador, suas *threads* são agrupadas em grupos chamados de *warps*. Geralmente, um *warp* é composto por 32 *threads*, todas pertencentes ao mesmo bloco. Todas as *threads* de um *warp* executam a mesma instrução ao mesmo tempo. Se existir alguma instrução de desvio condicional (`if`) e algumas *threads* se bifurcam, primeiro são executadas todas as *threads* que estão no mesmo fluxo enquanto as outras *threads* (as bifurcadas) ficam desabilitadas. Quando acaba a execução do fluxo, as *threads* que estavam desabilitadas passam a ser executadas enquanto que as outras ficam desabilitadas. Portanto, em vez das *threads* serem executadas todas em paralelo num único fluxo, foram utilizados dois fluxos executados um após

o outro devido à bifurcação. Em função disto, deve ser evitado ao máximo o uso de comandos de desvio condicional porque isso pode impactar significativamente na eficiência do código.

Quando todos os blocos acabam de serem executados nos multiprocessadores, a execução do *kernel* termina e o controle do programa é retomado pela CPU.

3.2 Swifter

Como ponto de partida para a elaboração de um algoritmo de N corpos paralelizado que sirva aos objetivos propostos nesta tese vamos a considerar o software `Swifter` (http://www.boulder.swri.edu/swifter/). Este software inclui um conjunto de diferentes algoritmos ou *drivers* para a simulação de sistemas de N corpos, com foco no problema planetário. Ele foi desenvolvido por Kaufmann e Levison com base no software `Swift` (LEVISON e DUNCAN, 1994). A principal diferença entre ambos é que este último está escrito em `Fortran 77`, enquanto que o primeiro é implementado em `Fortran 90`. Uma das vantagem de usar `Fortran 90` está no fato de poder definir variáveis dentro de estruturas de dados, que resultam mais simples de manipular e que tornam mais eficiente a alocação de memória[3]. Como nós estamos programando em C, que é uma linguagem que permite naturalmente o uso de estruturas de dados, decidimos usar o `Swifter` como base para o nosso código ao invés do `Swift`.

O `Swifter` inclui entre os seus *drivers* o algoritmo simpléctico de Wisdom e Holman (`WHM`) e o algoritmo simpléctico `Helio`, que já foram apresentados nas Seções 2.2.2 e 2.2.4. Além destes *drivers* também são incluídos:

- O método `RMVS` (*Regularized Mixed Variable Symplectic*), que como já mencionamos é similar ao algoritmo `WHM`, mas permite manipular encontros próximos entre partículas de teste e corpos massivos.

- O método `SyMBA` (*Symplectic Massive Bodies Algorithm*), que como vimos é uma extensão do algoritmo `Helio` que permite manipular encontros próximos entre corpos massivos através de uma estratégia de partição do potencial (*potential splitting*, cf. seção 2.1.3).

- Um método `TU4`, que é um algoritmo simpléctico de quarta ordem (YOSHIDA, 1990) para o hamiltoniano de N corpos em coordenadas baricêntricas (Equação 2.11).

- O integrador de Bulirsh-Stoer, não simplético

[3] Se duas variáveis, `x` e `y`, são definidas de maneira individual, elas podem ou não vir a ocupar espaços contíguos na memória. Entretanto, se elas são definidas dentro de uma estrutura, elas necessariamente vão ocupar espaços contíguos na memória. Isto torna o acesso a essas variáveis mais rápido e eficiente.

- O integrador `RA15` (EVERHART, 1985), que é um método de Runge-Kutta implícito com passo adaptativo. Este integrador é bastante popular na área da dinâmica planetária, particularmente devido à sua eficiência para lidar com órbitas muito excêntricas, como no caso de cometas.

Neste trabalho de tese, nós focamos somente na paralelização dos algoritmos `WHM` e `Helio`, que resulta suficiente para os objetivos propostos. A nossa intenção no futuro, entretanto, é implementar e paralelizar também os restantes algoritmos. Na implementação dos algoritmos `WHM` e `Helio`, nós introduzimos algumas simplificações em relação às versões do `Swifter`:

- Consideramos que todos os corpos integrados são massivos, isto é, não fizemos distinção entre corpos massivos e partículas de teste. Isto simplifica a paralelização, já que a introdução de partículas de teste requereria a alocação de *threads* e *kernels* diferentes para cada tipo de corpo.

- Consideramos todos os corpos como massas puntiformes, ou seja, não levamos em conta as perturbações devido ao achatamento ou a forma dos corpos, particularmente do corpo central.

- Eliminamos o uso de listas encadeadas. No `Swifter` algumas sequências de corpos são implementadas com uma estrutura de dados chamada de lista encadeada. Para facilitar a paralelização dos vetores estas listas foram substituídas por vetores comuns.

No restante deste capítulo, focaremos apenas na implementação do algoritmo `Helio`, destacando que o desempenho e os resultados obtidos com a implementação do algoritmo `WHM` foram semelhantes. Nas aplicações apresentadas no capítulo 4, a preferência pelo algoritmo `Helio` está fundamentada no fato de que não queremos ter que estipular a priori uma hierarquia entre as órbitas dos exoplanetas, já que não podemos garantir que as órbitas não vão se cruzar.

3.2.1 Tradução para C e validação

O passo prévio à paralelização consistiu na tradução do código do `Swifter` em `Fortran 90` para `C` e a sua posterior validação, para verificar se a implementação em `C` produzia o mesmo resultado que o código original. A simulação utilizada para validação foi um exemplo incluído no próprio `Swifter`, que considera o Sol e os planetas gigantes Júpiter, Saturno, Urano e Netuno. O tempo total de integração foi de 10^6 anos e o passo de integração utilizado foi de 1 dia.

Na Figura 3.5 são apresentados os erros dos testes de validação. Em todos estas figuras, os painéis à esquerda mostram o erro absoluto entre as versões em Fortran e C

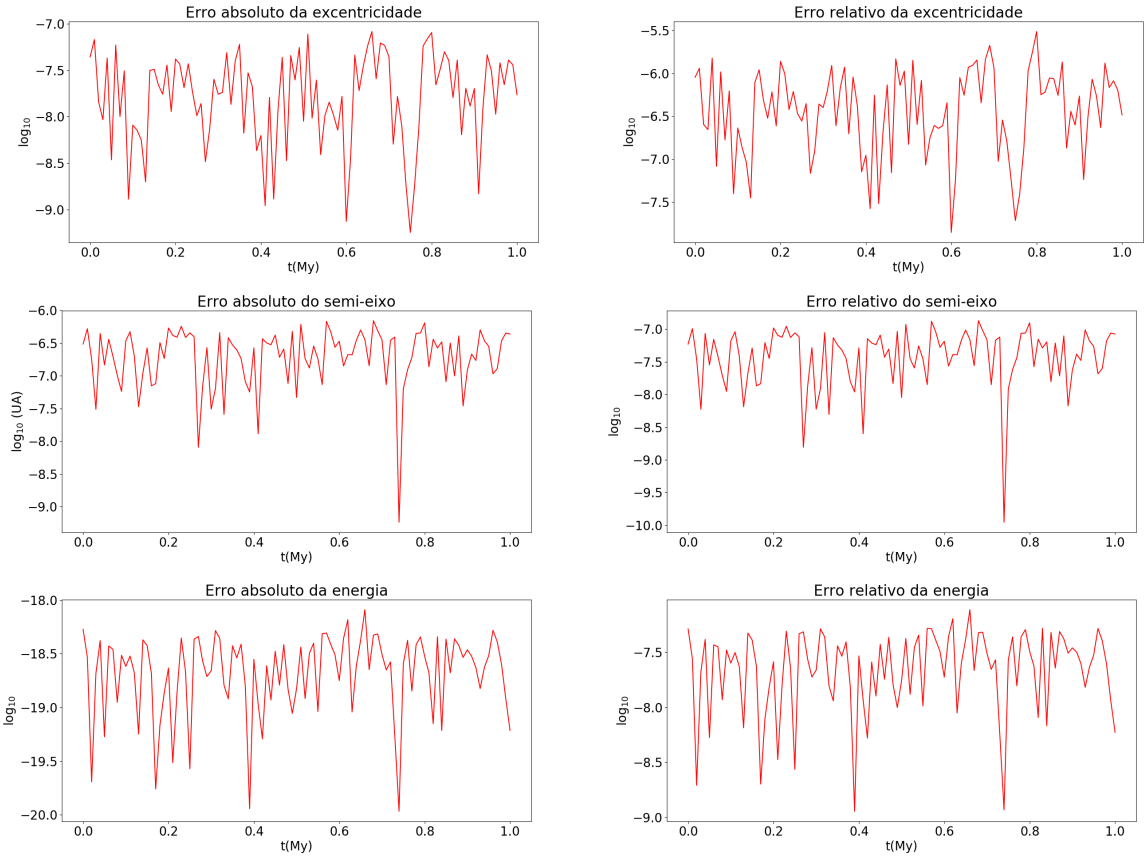

Figura 3.5: Validação da versão em `C` do algoritmo `swifter_helio`. De cima para baixo, excentricidade de Júpiter, semieixo maior de Júpiter e energia total do sistema, em função do tempo (em anos). Os painéis à esquerda mostram o erro absoluto e à direita o erro relativo entre ambas soluções.

e os painéis à direita mostram o erro relativo entre as saídas das duas implementações. Na Figura 3.5, os painéis superiores apresentam o comportamento da excentricidade de Júpiter, amostrada a cada 10^4 anos. De forma análoga, os painéis do meio apresentam a variação do semieixo de Júpiter e os painéis inferiores, o comportamento da energia mecânica total do sistema. Note-se que nas três figuras, o valores dos erros absoluto e relativo foram praticamente zero. Para a excentricidade o erro absoluto foi da ordem de 10^{-7}, para o semieixo o erro foi da ordem de 10^{-6} e para a energia, da ordem 10^{-18}. O erro relativo foi da ordem de $10^{-5,5}$ para a excentricidade , para o semieixo o erro foi da ordem de 10^{-7} e para a energia, da ordem 10^{-8}. Erros similares são obtidos quando se analisam os outros elementos orbitais e os outros planetas. Estas diferenças são muito provavelmente causadas pela forma em que as duas linguagens tratam os erros de arredondamento [4], mas como são muito pequenos consideramos que a nossa implementação em `C` está validada.

O passo seguinte consistiu em paralelizar a nossa implementação em C para ser executada em GPU. A seguir, apresentamos os detalhes do processo de paralelização.

3.3 Implementação em `CUDA` e validação

Se bem não existe uma estratégia de paralelização única para o problema de N corpos, podemos considerar dois casos extremos que envolvem estratégias diferentes: (i) a paralelização de um único problema de N corpos, com N muito grande, e (ii) a paralelização de vários problemas de N corpos distintos, com N pequeno. Para discutir o primeiro caso de paralelização, tomamos como base dois algoritmos de N corpos diferentes: um algoritmo simpléctico de quarta ordem do tipo $T + U$ (RUTH, 1983) e o já discutido algoritmo `Helio`.

3.3.1 Paralelização do algoritmo de Ruth

A paralelização do algoritmo de Ruth, neste caso, não visou obter um código completo que pudesse ser posteriormente aplicado a problemas específicos, mas foi encarada apenas como um exercício didático para analisar de que forma uma estratégia de paralelização em `CUDA` pode contribuir para otimizar simulações de N corpos. O algoritmo de Ruth foi escolhido em função da sua simplicidade e facilidade para paralelizar. O algoritmo consiste dos seguintes estágios:

$$\vec{p}_i \longleftarrow \vec{p}_i + \tau\,\beta_j \vec{F}_i$$
$$\vec{r}_i \longleftarrow \vec{r}_i + \tau\,\alpha_j \frac{\vec{p}_i}{m_i}$$

[4] os dados utilizados foram do tipo real(dp) em FORTRAN e double em C e C-CUDA, ambos representando números em representação de ponto flutuante de precisão dupla

onde m_i, $\vec{r}_i$, $\vec{p}_i$ e $\vec{F}_i$ representam, respectivamente, a massa, posição, momento e força resultante para o corpo i, τ é o tamanho do passo de integração e α_j,β_j, $j = 1, ..., 4$, são as constantes do método (cf. Equação 2.3). Note-se que o segundo estágio depende do resultado do primeiro, logo devem ser executados em forma sequencial. Para cada passo de integração, os dois estágios acima são repetidos 4 vezes, utilizando-se a cada vez os valores correspondentes de α_j,β_j. O processo de paralelização consistiu em executar os cálculos vetoriais em simultâneo. Ou seja, como cada estágio é executado para para cada coordenada x, y, z de cada corpo i, então a estratégia foi calcular em paralelo as componentes x, y, z de cada corpo. Assim, se temos 100 corpos na simulação, o número de *threads* simultâneos em cada estágio será de 100×3.

O teste de desempenho para verificar se o código paralelizado em `CUDA` é mais eficiente que o código serial executado em CPU consistiu em realizar várias simulações, variando em cada uma delas o número de corpos de 1 a 400. O tempo total de cada simulação foi de 1 uma unidade de tempo, com 5 instantes de saída intermediários. A massa de cada corpo foi igual a 1 unidade de massa. As componentes da posição inicial de cada corpo foram escolhidas iguais a $3i + k$, onde i é o número do corpo e $k = 1, 2, 3$ é o índice da componente x, y, z. As componentes do momento inicial foram escolhidas todas iguais a 1.

Os testes foram realizados usando uma versão sequencial implementada na linguagem C executando em várias CPUs e uma versão paralelizada implementada na linguagem CUDA C executando em várias GPUs e os resultados são apresentados na Figura 3.6. Como pode ser observado, em todas as placas GPU os tempos de execução são inferiores aos tempos de execução em CPU. Para o caso de poucos corpos, a diferença entre as GPUs e CPUs é bem pequena, mas a medida que o número de corpos aumenta, as GPUs tornam-se cada vez mais eficientes. Isto demonstra que vale a pena investir em um processo de paralelização em GPU, pois o ganho em tempo de execução pode ser significativo.

3.3.2 Paralelização do algoritmo `Helio`

A partir da nossa versão do algoritmo `Helio` na linguagem `C`, fizemos uma paralelização em `CUDA` constituída por três procedimentos diferentes:

1. Paralelização das operações vetoriais. Em todas as operações básicas de vetores, como somas e subtrações, realizamos as operações sobre cada coordenada x, y, z de forma paralela. Isto, em teoria, deveria reduzir o tempo de execução dessas operações por um fator $1/N$ em comparação com a operação sequencial. Dependendo do tamanho dos vetores e do número de núcleos de execução da GPU, pode não ser possível executar todas as componentes em forma paralela ao mesmo tempo. Por exemplo, supondo que somamos dois vetores de oito componentes cada um, o cálculo das oito somas em paralelo deveria, teoricamente, reduzir o tempo de exe-

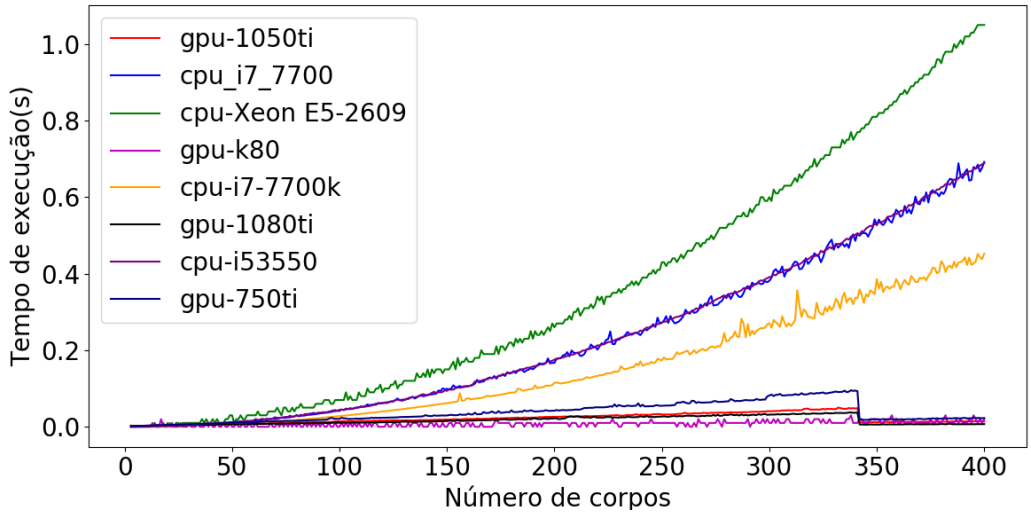

Figura 3.6: Teste de desempenho da paralelização do algoritmo de Ruth.

cução a 12,5% do tempo sequencial. Mas se, por exemplo, existirem apenas dois núcleos disponíveis para execução, só será possível executar duas somas por vez, o que reduziria o tempo de execução a apenas 50% do tempo sequencial. Em qualquer caso, a paralelização das operações vetoriais contribui para uma redução relevante do tempo de execução.

2. Semi-paralelização dos somatórios. Este foi um outro tipo de paralelização que implementamos no algoritmo `Helio`. Ela é considerada uma semi-paralelização porque não podemos executar todo o somatório paralelamente em único passo. O que se faz neste caso é, em cada passo, realizar uma certa quantidade de somas parciais em forma paralela, até que no final obtemos o somatório completo otimizado. Um exemplo desse tipo de semi-paralelismo é apresentado na Figura 3.7. Suponhamos que queremos somar todas as componentes de um vetor de dimensão 8. Num primeiro estágio, o que fazemos é calcular a soma das componentes 1 e 5, depois a das componentes 2 e 6, e assim por diante até a soma das componentes 4 e 8. Basicamente, o que fizemos foi dividir o vetor de dimensão 8 em dois subvetores de dimensão 4 e somamos as componentes destes subvetores. Como resultado desse primeiro estágio, temos agora um vetor de dimensão 4, onde cada componente contém uma soma parcial. Sobre este vetor se aplica o mesmo procedimento, ou seja, divide-se em duas partes iguais e se somam as componentes correspondentes. O algoritmo continua até que no último estágio obtemos um vetor de dimensão 1 que contém o valor da soma total. Note que as somas das componentes dos vetores em cada estágio pode ser executada em paralelo. Logo, precisamos de 3 estágios para somar 8 valores, 4 estágios para somar 16 valores e, em geral, N estágios para somar N^2 valores. Em outras palavras, para somar as componentes de um vetor de

dimensão N, precisamos de $\log_2 N$ estágios, enquanto que com a soma sequencial seriam necessários $N-1$ estágios.

3. Paralelização da parte kepleriana do algoritmo. Como vimos oportunamente, o algoritmo Helio envolve a cada passo evoluir as órbitas de cada corpo ao longo de uma solução kepleriana. A solução kepleriana em si não pode ser paralelizada aplicando as duas estratégias apresentadas anteriormente, mas o que pode ser feito é alocar N *threads* paralelas e utilizar cada *thread* para avançar a solução kepleriana de cada corpo.

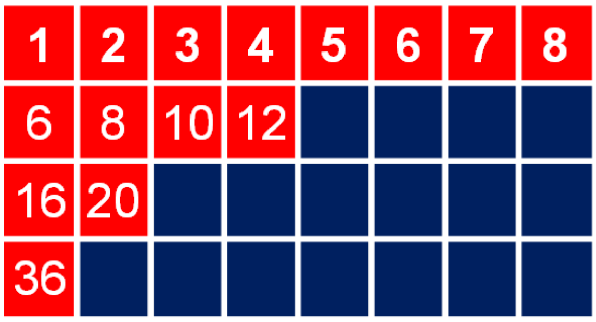

Figura 3.7: Esquema de semi-paralelização de um somatório.

Aplicando as três estratégias de paralelização descritas acima, implementamos uma versão em `CUDA-C` do algoritmo `Helio` e comparamos os tempos de execução com a versão sequencial em `C`. Na Figura 3.8 apresentamos os diferentes testes de desempenho. No eixo horizontal é indicado o número de corpos na simulação e no eixo vertical o tempo da simulação. Em cada simulação, os corpos tem posição inicial $\vec{r_i} = (i, 0, 0)$, momento inicial $\vec{p_i} = (0, i, 0)$ e massa $m_i = 10^{-30}$ [5], assumindo que a massa central vale 1. Os corpos foram configurados com massas bem pequenas de propósito para evitar eventuais encontros próximos ou colisões e as posições e momentos foram escolhidos nesse formato para poder gerar sistemas de N corpos de forma automática e rápida. Em cada simulação é executado apenas um passo de integração do algoritmo.

Cada curva na Figura 3.8 representa uma GPU ou CPU específica, sendo que nas CPUs foi rodado o algoritmo sequencial. Podemos notar que as simulações nas GPUs Tesla K80 e GeForce GTX 1080 Ti foram mais rápidas que as três CPUs testadas para valores de $N \gtrsim 4500$. Já a placa GeForce GTX 1050 Ti foi mais rápida que as três CPUs para valores de $N \gtrsim 7500$, enquanto que a placa GeForce GTX 750 Ti não ganhou de nenhuma das CPUs testadas, independentemente do valor de N. Isso mostra que o ganho de desempenho não somente depende da estratégia de paralelização como também das características do hardware utilizado.

[5] É um valor muito baixo de massa e isso poderia prejudicar a precisão do resultado, mas a intenção aqui era apenas testar os tempos de execução

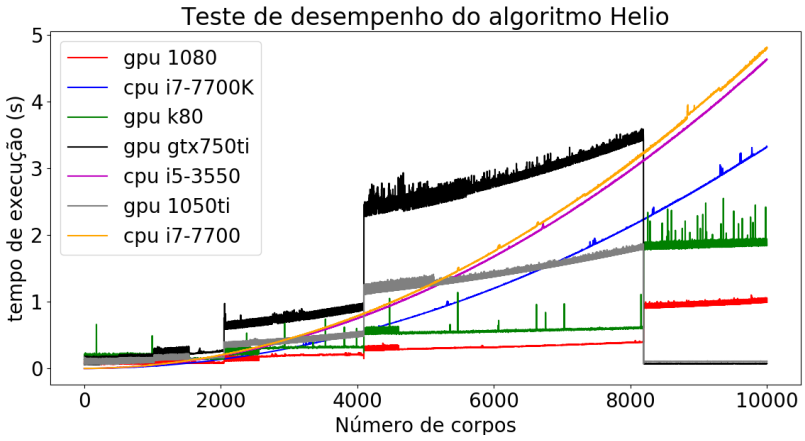

Figura 3.8: Teste de desempenho da paralelização do algoritmo `Helio`. Nas abscisas temos o número de corpos na simulação e nas ordenadas o tempo de execução em segundos.

Em relação a placa GTX 1050 Ti é necessário uma análise mais aprofundada do resultado. Podemos observar que, nos gráficos correspondentes às GPUs, ocorre de tempos em tempos um salto no desempenho. Isso está vinculado à estratégia de paralelização implementada, que faz com que a quantidade de memória alocada e o número de *threads* usadas sejam sempre múltiplos de 2. Assim, por exemplo, se $N = 1500$ serão alocados recursos considerando $N = 2048$, se $N \geqslant 2049$, alocam-se recursos para $N = 4096$, e assim por diante. Isto é o que provoca as quedas de desempenho próximas a valores de N múltiplos de 2. Portanto, analisando o gráfico para a placa GTX 1050 Ti, deveria se esperar uma queda de desempenho em $N = 8196$ que faria com que essa GPU em particular fosse superada pela CPU i7-7700k.

Por fim, cabe mencionar que para as placas GTX 750 Ti e 1050 Ti o desempenho parece ser absurdamente melhor para valores de $N > 8196$ comparado ao das outras GPUs e CPUs. Mas o que acontece neste caso é que, devido à limitação de hardware dessas placas, o programa não consegue alocar os recursos necessários e executar corretamente.

Podemos concluir que, para a nossa implementação do algoritmo `Helio` paralelizado, o desempenho das placas gráficas pode ou não ser melhor que o das CPUs dependendo das características do hardware, diferentemente do que acontecia com a paralelização do algoritmo de Ruth. O principal gargalo está no fato de que o avanço da solução kepleriana para cada corpo não é muito paralelizável e precisamente esse trecho do código consome muito tempo de cômputo. Como as GPUs em geral tem velocidade de *clock* inferior às CPUs, ou seja, executam menos cálculos por segundo, as implementações em GPU tem uma queda importante de desempenho nesse trecho do código. Em contraste, no caso do algoritmo de Ruth, como o código é altamente paralelizável, é possível aumentar o número de *threads* concorrentes e melhorar assim o desempenho das GPUs.

tipo	modelo	tempo de execução	tempo de execução normalizado
GPU	GeForce GTX 1080 Ti	1,812	1,812
GPU	Tesla K80	0	0
GPU	GeForce GTX 1050 Ti	2,447	2,447
GPU	GeForce GTX 750 Ti	3,806	3,806
CPU	i7-7700K 4,2 GHz	$1,807 \times 10^{-3}$	29,605
CPU	Xeon E5-2609 1.7 GHz	0	0
CPU	i7-7700 3,6 GHz	$1,41 \times 10^{-3}$	23,101
CPU	i5-3550 3,3 GHz	$9,59 \times 10^{-4}$	15,712

Tabela 3.1: Teste de desempenho da paralelização de uma grade de 16384 condições iniciais. Para as CPU, o tempo de execução normalizado é calculado como `tempo de execução x tamanho da grade`.

3.3.3 Paralelização de uma grade

Como mencionamos no começo da seção 3.3, um segundo caso extremo de paralelização acontece quando se tem várias simulações de N corpos distintas, cada uma com N pequeno. As diferenças entre as simulações podem ser, por exemplo, as condições iniciais, que podem estar distribuídas numa grade. A ideia aqui, então, é fazer com que as diferentes simulações para cada ponto da grade sejam executadas em forma simultânea. Este é precisamente o código que iremos aplicar mais adiante para o estudo de sistemas extrassolares.

Para testar este paradigma de paralelização, utilizaremos o algoritmo `Helio` juntamente com um subconjunto das simulações que são apresentadas na seção 4.3, quando da aplicação ao estudo do sistema extrassolar Kepler-46. A grade neste caso está constituída por 16384 condições inicias distintas e, ao invés de simularmos a grade por 10^6 dias como na aplicação, simulamos apenas 10^2 dias, o que resulta suficiente para avaliar o desempenho do código.

Para comparar os tempos de execução paralela do código em `CUDA` com o tempo de execução serial em CPU, utilizamos a implementação serial do algoritmo `Helio` na linguagem `C` [6] e introduzimos uma métrica denominada de tempo de execução normalizado. Para a simulação paralelizada, o tempo de execução normalizado é definido como o próprio tempo de execução. Para a simulação serial, o tempo normalizado é obtido executando uma vez o código sequencial, para uma dada condição inicial na grade, e multiplicando o tempo de execução resultante pelo número de pontos na grade, neste caso 16384. O teste consistiu em oito simulações, quatro em GPUs diferentes e quatro em CPUs diferentes. Os resultados são apresentados na Tabela 3.1.

Como podemos observar na última coluna, os valores de tempo de execução normalizado são sempre menores nas GPUs do que nas CPUs, indicando que as primeiras são

[6] O código foi uma tradução simples do código em FORTRAN do SWIFTER, sem otimizações.

mais eficientes. Também podemos notar que a GPU mais ineficiente tem um tempo de execução normalizado que ainda é quatro vezes mais rápido que o tempo de execução normalizado da CPU mais eficiente. De acordo com os nossos testes, as GPUs testadas acabam sendo entre 4 e 16 vezes mais rápidas do que as CPUs testadas. Cabe destacar, entretanto, que este resultado certamente depende do número de pontos na grade e que para grades de poucos pontos, o ganho em velocidade não deve ser significativo, vindo a GPU a ser inclusive mais lenta do que a CPU.

Como testes de validação do código, procuramos reproduzir resultados bem conhecidos da literatura. Apresentamos a seguir os dois testes realizados.

3.3.3.1 Mapa da ressonância asteroidal 3:1

Uma ressonância de movimentos médios k:l entre dois planetas ocorre quando a razão entre os seus períodos orbitais, P_i, é um número racional, matematicamente

$$\frac{\nu_1}{\nu_2} \simeq \frac{k}{l}$$

onde $\nu_i = 2\pi/P_i$ é a frequência orbital de cada planeta e k, l são números naturais. Por convenção, o índice 1 identifica o corpo menos massivo e o índice 2 o corpo mais massivo, de forma que quando $k > l$, o corpo mais massivo é o mais externo e quando $k < l$, o corpo mais massivo é o mais interno.

x

No cinturão de asteroides, a ressonância 3:1 com Júpiter ocorre quando o período orbital do asteroide é 1/3 do período orbital de Júpiter. Esta ressonância, localizada a 2,5 au do Sol, é a ressonância de movimentos médios mais caótica que ocorre no cinturão asteroidal. O tempo de vida médio de uma órbita capturada nesta ressonância é da ordem de apenas 10^4 anos (GLADMAN *et al.*, 1997). O comportamento caótico é causado pela sobreposição de ressonâncias seculares e secundárias que ocorrem dentro da ressonância de movimentos médios (MORBIDELLI e MOONS, 1993, 1995), e a sua consequência é o aumento brusco e intermitente da excentricidade orbital do asteroide (WISDOM, 1982, 1983; FERRAZ-MELLO *et al.*, 1996), que leva a que o corpo seja removido da ressonância por encontros próximos com os planetas terrestres. Isto faz com que a região em torno de 2,5 au esteja despovoada de asteroides, criando uma falha na distribuição destes objetos que é conhecida como falha de Kirkwood.

A ressonância asteroidal 3:1 é uma configuração dinâmica que tem sido muito estudada e a sua estrutura no espaço de fase é muito bem conhecida. Dado que o caos se manifesta em tempos relativamente curtos, esta ressonância torna-se um laboratório ideal para testar o funcionamento do nosso código paralelizado.

O que fizemos foi simular um sistema de 4 corpos constituído pelo Sol, Júpiter, Saturno e um asteroide. Cabe destacar que, nesta simulação, o asteroide não é tratado como

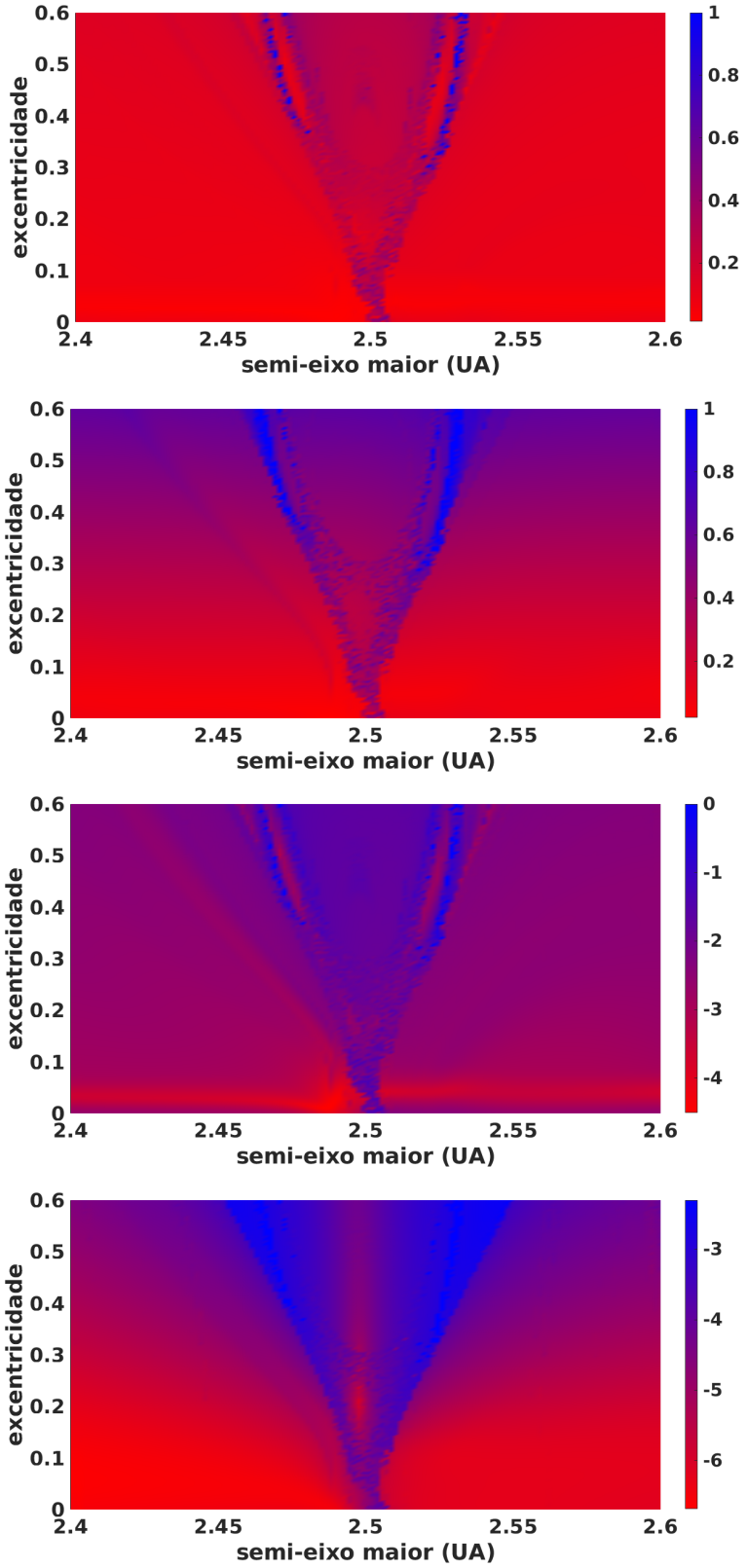

Figura 3.9: Mapas dinâmicos para uma grade de 128×128 condições iniciais do sistema Sol-Júpiter-Saturno-asteróide. De cima para baixo, os mapas apresentam, respectivamente, a máxima variação em excentricidade (Eq. 2.32), a máxima excentricidade orbital (Eq. 2.31), a variância em excentricidade e a variância em semieixo maior (Eqs. 2.30).

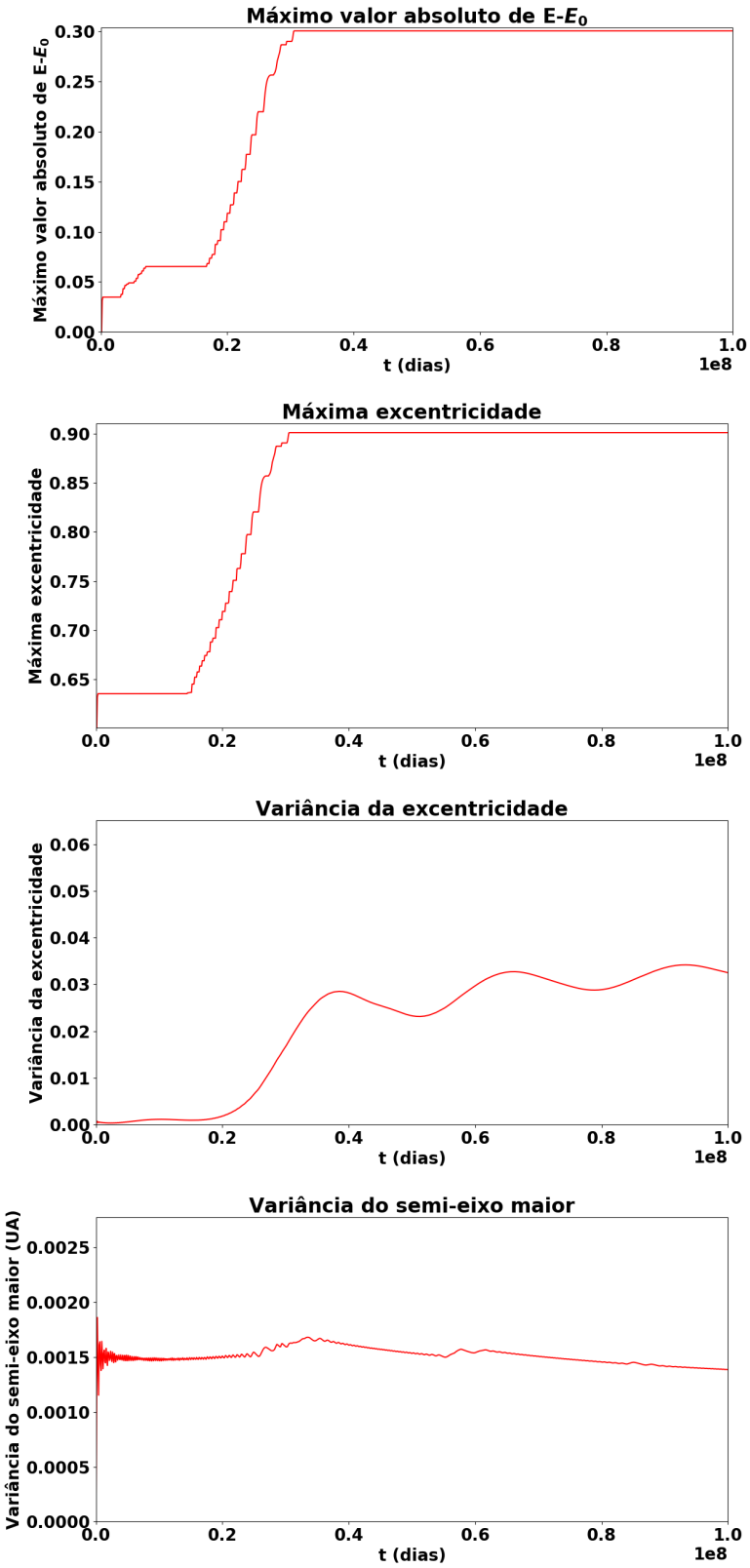

Figura 3.10: Evolução no tempo dos indicadores de estabilidade para o asteróide localizado em a= 2,532 e e=0,6.

uma partícula de teste mas como um corpo massivo com massa de 1 $M_\oplus$. As posições e velocidades iniciais heliocêntricas para as órbitas de Júpiter e Saturno foram obtidas das efemérides do JPL (*Jet Propulsion Laboratory/NASA*; https://ssd.jpl.nasa.gov/?ephemerides) para uma data arbitrária. As condições iniciais para o asteroide foram distribuídas numa grade de 128×128 no espaço de semieixo vs. excentricidade, cobrindo os intervalos $2,4 \leqslant a \leqslant 2,6$ au e $0 \leqslant e \leqslant 0,6$. Os restantes elementos orbitais do asteroide (inclinação, longitude do nodo, longitude do periélio, e longitude média) foram fixados em $i = \Omega = \varpi = \lambda = 0°$. O tempo total de integração foi de $1,269 \times 10^8$ dias (equivalente a $\sim$ 12000 revoluções de Saturno), com um passo fixo de 10 dias (equivalente a $\sim 1/150$ do período do asteroide).

Os resultados são apresentados na Figura 3.9 na forma de mapas dinâmicos, que indicam o valor de um dado estimador de caos para cada ponto da grade de condições iniciais. Neste caso, os estimadores de difusão são calculados encima da órbita do asteroide. Nestes mapas aparece bem delimitada a região ocupada pela ressonância, com o clássico formato em V centralizada em 2,5 au. As bordas desta região definem a separatriz da ressonância. As órbitas dentro da ressonância são caracterizadas pela libração em torno de 180° do ângulo crítico $\sigma_{3/1} = 3\lambda_{\rm Jup} - \lambda_{\rm Ast} - 2\varpi_{\rm Ast}$. Órbitas com amplitude de libração pequena se localizam próximas ao centro de ressonância, ao redor de 2,5 au, enquanto que aquelas com grande amplitude de libração se localizam próximas à separatriz. É precisamente na vizinhança da separatriz que ocorre a sobreposição de ressonâncias seculares, fazendo com que o movimento destas órbitas seja muito mais caótico do que nas demais regiões. Os nossos mapas dinâmicos reproduzem bem este comportamento e mostram ainda que fora da ressonância o movimento é regular, como esperado. Exibimos também na Figura 3.10, a evolução temporal dos indicadores utilizados na Figura 3.9 para um asteróide localizado em a=2,532 e e=0,6. Note que para esse exemplo caótico próximo a ressonância 3:1, os indicadores indicam aumento dos valores dos indicadores baseados na excentricidade, que é uma característica dos corpos próximos a ressonância.

Dado que o nosso código parece simular corretamente a dinâmica da ressonâcia 3:1, podemos concluir que o mesmo está validado.

3.3.3.2 Validação do MEGNO

Após o integrador numperico ter sido validado, o próximo passo foi validar a implementação das equações variacionais e do cálculo do indicador de caos MEGNO. Para essa finalidade, realizamos dois testes diferentes: (i) repetimos a simulação da ressonância 3:1 no sistema Sol-Júpiter-Saturno-Asteróide, e (ii) simulamos um sistema extrassolar hipotético que foi previamente estudado por HAGHIGHIPOUR *et al.* (2013). O resultado do teste com a ressonância 3:1 é apresentado na Figura 3.11. Note-se que, assim como nos mapas mostrados na seção anterior, o mapa do MEGNO apresenta a mesma estrutura em V centrada em a=2,5 UA, com as regiões mais caóticas nas bordas da estrutura. Ex-

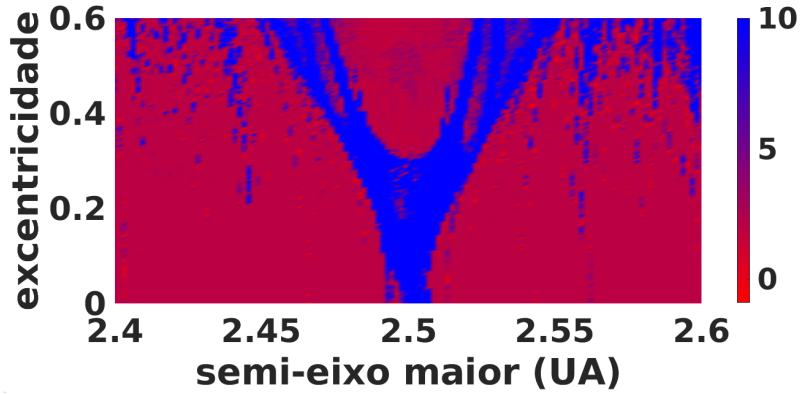

Figura 3.11: Mapa de MEGNO para a grade de condições iniciais em volta da ressonância asteroidal 3:1. Comparar com a Figura 3.9.

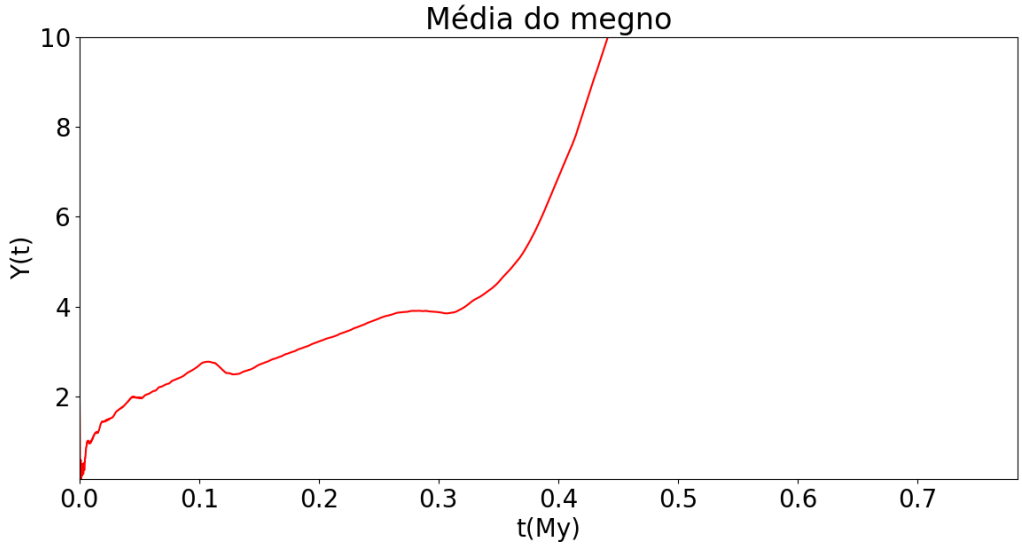

Figura 3.12: Comportamento da média do MEGNO no caso de uma órbita caótica na ressonância asteroidel 3:1 ($a = 2,5$ au, $e = 0,3$).

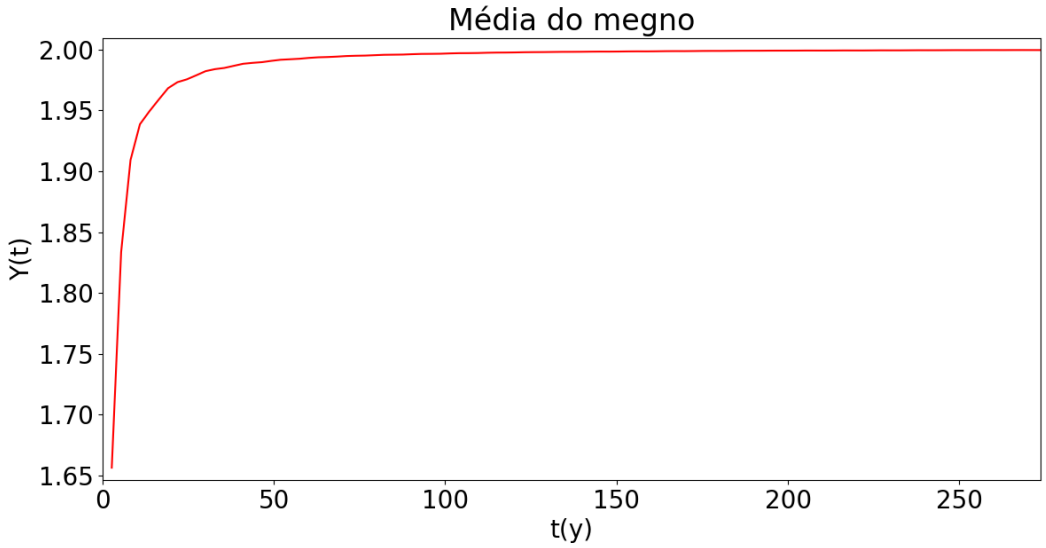

Figura 3.13: Comportamento da média do MEGNO no caso de uma órbita estável fora da ressonância asteroidel 3:1 ($a = 2,4$ au, $e = 0$).

traindo duas simulações da grade, uma localizada numa região caótica e outra em uma região estável, podemos analisar em detalhe o comportamento do MEGNO. Note-se que na Figura 3.12, a média do MEGNO cresce linearmente com o tempo, o que caracteriza um comportamento caótico, enquanto que na Figura 3.13 a média do MEGNO converge para o valor 2, o que caracteriza um comportamento regular.

Para a simulação do sistema extrassolar hipotético foi considerado um sistema de 3 corpos constituído por uma estrela de 1 massa solar, um planeta de 1 massa de Júpiter e um planeta de 1 massa da Terra em uma configuração orbital extremamente compacta, com ambos planetas muito próximos à estrela. As posições e velocidades iniciais astrocêntricas para as órbitas dos planetas foram obtidas de HAGHIGHIPOUR *et al.* (2013). Em particular, as condições iniciais para o planeta terrestre foram distribuídas numa grade de 128×128 no espaço de semieixo vs. excentricidade, cobrindo os intervalos $0,01 \leqslant a \leqslant 0,1$ au e $0 \leqslant e \leqslant 1$, mantendo os restantes elementos orbitais (inclinação, longitude do nodo, longitude do periélio, e longitude média) fixos. O tempo total de integração foi de 322 anos, com um passo fixo de 14 minutos.

Os resultados são apresentados na Figura 3.14, onde comparamos o nosso mapa (à esquerda) como o mapa publicado por HAGHIGHIPOUR *et al.* (à direita). Ambos mapas são praticamente idênticos, com exceção de uma faixa caótica presente em nosso mapa para valores muito altos de excentricidade que está vinculada a órbitas que colidem com a estrela na nossa simulação. Com base neste resultado, podemos concluir que o nosos código está validado, e pode ser aplicado a casos reais como veremos no capítulo 4.

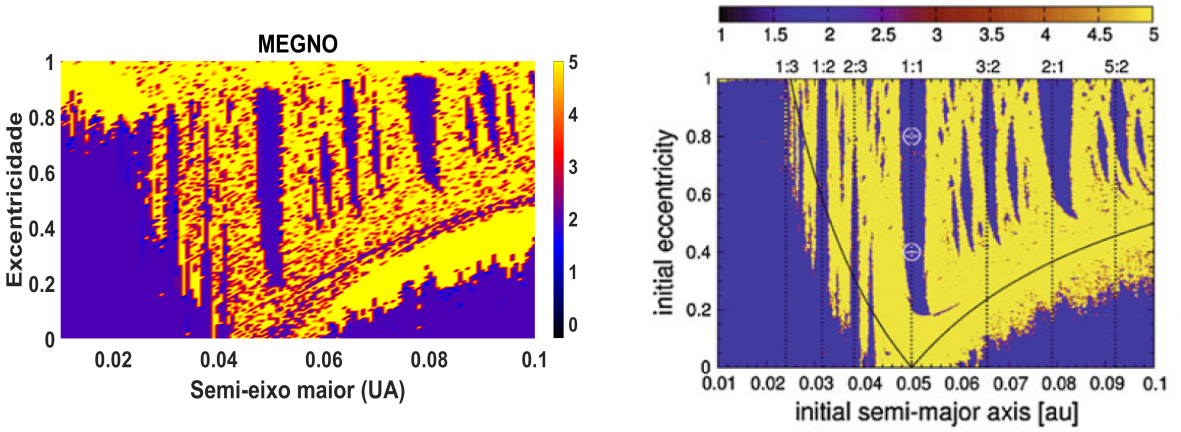

Figura 3.14: Comparação entre o mapa de MEGNO calculado com o nosso código, à esquerda, e o resultado publicado em HAGHIGHIPOUR *et al.* (2013), à direita, para um sistema extrassolar hipotético.

3.4 Manual do usuário

Nesta seção apresentamos um manual de uso para o código desenvolvido durante este trabalho, que paraleliza a execução de uma grade de condições iniciais e cujas aplicações serão apresentadas no próximo capítulo. O programa consiste de vários arquivos que contêm a rotina principal e as sub-rotinas em `C` e `CUDA-C` (um arquivo individual para cada sub-rotina), uma pasta de entrada contendo os arquivos com as condições inicias da grade, e uma pasta de saída contendo os arquivos com os resultados da simulação. A seguir detalharemos a estrutura de cada arquivo.

3.4.1 Arquivos de entrada

A pasta de entrada, deve conter um arquivo de configuração de parâmetros, chamado de `param.in`, e uma pasta com os arquivos contendo os dados de entrada da simulação: condições inicias dos corpos e condições iniciais da grade.

3.4.1.1 Arquivo de configuração `param.in`

Consiste num arquivo texto onde cada linha configura uma dada propriedade da simulação. O formato de cada linha é constituído pelo nome da variável de configuração, em letras maiúsculas, seguido de espaços em branco e do valor da variável, que pode ser um número ou uma *string*. Algumas variáveis de configuração possuem valores padrão que são utilizados quando as mesmas não são definidas no arquivo param.in. As variáveis de configuração possíveis são:

- `T0`: define o instante inicial da simulação. O valor padrão é 0. O tipo é `float`.
- `TSTOP`: define o instante final da simulação. O tipo é `float`.

- `DT`: define o tamanho do passo de integração da simulação. O tipo é `float`.

- `ISTEP_OUT`: define o número de passos de integração entre duas gravações consecutivas dos resultados nos arquivos de saída. O tipo é `int`. Este parâmetro deve ser utilizado com cuidado, porque cada vez que se escrevem dados no disco, os dados devem ser primeiro transferidos da memória da GPU para a memória RAM para depois serem gravados no disco. Como este processo é muito demorado, fazer saídas muito frequentes pode degradar significativamente o desempenho do programa.

- `INORM`: define o número de passos de integração entre duas renormalizações consecutivas da solução das equações variacionais para o cálculo do indicador MEGNO. O tipo é `int`. Recomenda-se usar o mesmo valor de `ISTEP_OUT` ou 0 para não renormalizar.

- `MEGNOMAXIMO`: define um limite superior, arbitrário, para o valor do indicador MEGNO. Quando o MEGNO de alguma das simulações ultrapassa este valor, ele é fixado neste valor máximo. O tipo é `float`.

- `SET_TXT`: define se o usuário deseja guardar as posições e velocidades dos corpos em cada instante num arquivo em formato ASCII. O valor padrão é 0, que faz com que o arquivo não seja gerado. Para o arquivo ser gravado, use o valor 1. O tipo é `int`. Este parâmetro é útil se o usuário estiver interessado apenas em obter alguns dos indicadores de caos e não todas as posições e velocidades dos corpos. Com isto se ganha em desempenho, já que a escrita em disco e o tempo total de transferência dos dados da memória da GPU para CPU são menores.

- `SET_MAXA`: define se o usuário deseja guardar o indicador máximo semieixo maior num arquivo ASCII. O valor padrão é 0, que faz o arquivo não ser gerado. Para o arquivo ser gravado, use o valor 1. O tipo é `int`.

- `SET_MAXE`, `SET_MAXABSEE0`, `SET_DA`, `SET_DE`: parâmetros análogos ao `SET_MAXA` para os restantes indicadores de caos: máxima excentricidade, máximo $|e - e_0|$, variância no semieixo maior e variância na excentricidade.

- `SET_A0`: define se o usuário quer gravar o valor inicial do semieixo maior. Isso é útil para verificar se a grade está correta e pode ser usado para testes e gráficos. O valor padrão é 0. O tipo é `int`.

- `SET_E0`, `SET_I0`: análogos ao `SET_A0` mas para a excentricidade e inclinação iniciais.

- `SET_Y`, `SET_MEGNO`, `SET_NORMA`: análogos ao `SET_A0` mas para a média do MEGNO, o MEGNO e a norma do vetor variação $\boldsymbol{\delta}$.

- `INDICE_GRADE`: define o índice do corpo que irá variar em cada *thread* da simulação. Corresponde ao índice do corpo cujas condições iniciais variam na grade. Por exemplo, se tivermos o Sol com `id=0`, a Terra com `id=1` e Júpiter com `id=2`, e este parâmetro for igual a 1, isso significa que os dados de entrada do Sol e Júpiter serão os mesmos para todas as *threads* e a Terra terá os dados de entrada diferentes em cada *thread*. O tipo é `int`.

- `PL_IN`: define o nome do diretório/arquivo que contem os dados dos corpos que são comuns a todas as *threads*. O *path* deve ser definido em forma relativa ao diretório em que será executado o programa. O tipo é `char`.

- `PL_IN2`: define o nome do diretório/arquivo que contem os dados do corpo cujos dados de entrada são diferentes em cada *thread*. O *path* deve ser definido em forma relativa ao diretório em que será executado o programa. O tipo é `char`.

- `BIN_OUT`: define o nome do diretório/arquivo que conterá os dados de saída (posições e velocidades) de todos os corpos de uma das simulações da grade para alguns instantes de tempo. O *path* deve ser definido em forma relativa ao diretório em que será executado o programa. O tipo é `char`.

- `CAOS_FILE`: define os nomes dos arquivos que contém os indicadores de caos no diretório de saída. Deve ser especificado diretório/prefixo dos nomes dos arquivos de saída. O *path* deve ser definido em forma relativa ao diretório em que será executado o programa. O tipo é `char`.

- `LINHAS`: define o número de *threads* na direção de `x` em cada bloco. O tipo é `int`.

- `COLUNAS`: define o número de *threads* na direção de `y` em cada bloco. O tipo é `int`.

- `LINHASB`: define o número de blocos na direção de `x` da grade. O tipo é `int`.

- `COLUNASB`: define o número de blocos na direção de `y` da grade. O tipo é `int`.

- `L`: define o índice `x` da *thread* cujos valores serão gravados no arquivo definido por `BIN_OUT`. O tipo é `int`.

- `C`: define o índice `y` da *thread* cujos valores serão gravados no arquivo definido por `BIN_OUT`. O tipo é `int`. Se por exemplo `L=4` e `C=5`, significa que a thread cujo `threadIdx.x=4` e `threadIdx.y=5` terá os dados de todos os corpos (posições e velocidade) gravados no arquivo de saída correspondente.

3.4.1.2 Arquivo definido na variável `PL_IN`

Este arquivo conterá os dados dos corpos cujos valores são iguais em todas as *threads*. A estrutura do arquivo é a seguinte:

```
nbod
mass_0
id_1 mass_1
x_1 y_1 z_1
vx_1 vy_1 vz_1
.
.
id_i mass_i
x_i y_i z_i
vx_i vy_i vz_i
.
.
```

Na primeira linha, `nbod` é o número total de planetas na simulação, incluindo o corpo na grade (o corpo central assume o índice 0). O tipo é `int`.

Na segunda linha, `mass_0` é a massa do corpo central, em unidades tais que façam com que a constante gravitacional seja igual a 1 (dependendo das unidades de distância e de tempo escolhidas). O tipo é `float`.

Da linha 3 em diante aparecem os dados de cada um dos corpos, havendo 3 linhas para cada corpo.

Na primeira linha, `id_i` é o identificador do corpo na simulação. O tipo é `int`. Note-se que o identificador não precisa estar ordenado de forma sequencial, podendo vir primeiro o corpo 3, depois o corpo 2, depois o corpo 5, etc. As únicas restrições são que `id_i` deve ser $\leqslant$ `nbod` e `id_i` deve ser$\neq$ `INDICE_GRADE`. `mass_i` é a massa do corpo, nas mesmas unidades da massa central. O tipo é `float`.

A segunda linha contém a posição do corpo em coordenadas x, y, z astrocêntricas (em relação ao corpo central), e a terceira as velocidades v_x, v_y, v_z também astrocêntricas. Todos estes valores são de tipo `float`. É importante lembrar que as unidades de distância, massa e tempo devem ser auto-consistentes, de forma que $G = 1$.

3.4.1.3 Arquivo definido na variável `PL_IN2`

Este arquivo conterá os dados do corpo na grade, que terá parâmetros de entrada diferentes para cada *thread*. Cada condição inicial na grade estará representada por um conjunto de três linhas da forma

```
id_i mass_i
x_i y_i z_i
```

```
vx_i vy_i vz_i
```

A ordem é tal que devem-se colocar primeiro as condições iniciais que serão simuladas nas *threads* da primeira linha do bloco e depois as que serão simuladas nas demais linhas. Em outras palavras, se a grade da simulação tiver, por exemplo, um bloco com n *threads* na direção `x` e m *threads* na direção `y`, primeiro vem a condição inicial que será simulada na *thread* da primeira linha e da primeira coluna do bloco, depois vem a condição inicial que será simulada na *thread* da primeira linha e segunda coluna do bloco, e assim por diante. Note-se que `id_i` é o mesmo para todas as condições iniciais e deve ser igual ao valor de `INDICE_GRADE`. As coordenadas e velocidades devem ser astrocêntricas.

3.4.1.4 Condições iniciais para as equações variacionais

As condições iniciais para o vetor variação $\boldsymbol{\delta}$ utilizado no calculo do MEGNO são definidas automaticamente dentro do código. As condições são tais que

$$\begin{aligned} \delta\mathbf{r}_k &= (1,0,0) \qquad \text{para } k = \texttt{INDICE\_GRADE} \\ \delta\mathbf{r}_i &= (0,0,0) \qquad \text{para todo } i \neq k \\ \delta\mathbf{v}_i &= (0,0,0) \qquad \text{para todo } i \end{aligned}$$

3.4.2 Arquivos de saída

Os arquivos de saída se localizam na pasta saída no mesmo diretório do executável. A seguir são especificados os tipos de arquivos que são gerados pela implementação.

3.4.2.1 Arquivo de dados dos corpos para uma das simulações da grade

Nesse arquivo serão gravados os dados dos corpos que são simulados em uma das *threads* do programa, escolhida através dos parâmetros `L` e `C` do arquivo `param.in`. O nome do arquivo, assim como seu diretório, são definidos pelo parâmetro `BIN_OUT`. Este arquivo consiste de vários blocos similares de linhas, cada bloco representando a configuração dos corpos num dado instante da simulação. Os instantes de tempo e o intervalo entre os mesmos são dados pelo valor de `ISTEP_OUT*DT` conforme definidos no arquivo de configurações. O primeiro instante de tempo a ser gravado é o instante inicial `T0`.

Para cada instante de tempo, o bloco de dados consiste de `n+1` linhas com a seguinte estrutura

```
time npl
id_1 mass_1 x_1 y_1 z_1 vx_1, vy_1 vz_1
.
.
id_n mass_n x_n y_n z_n vx_n vy_n vz_n
```

Na primeira linha vêm precisamente o tempo `time` e o número de planetas na simulação `npl`. A partir da segunda linha, cada linha contém a informação de cada um dos planetas pela ordem de `id`: massa, posição x, y, z e velocidade v_x, v_y, v_z astrocêntricas. Note-se que são gravados os dados de todos os planetas, inclusive o da grade.

3.4.2.2 Arquivos de indicadores de caos

Esses arquivos tem o nome composto pelo prefixo definido em `CAOS_FILE` seguido do nome do indicador de caos que está gravado. A letra `A` identifica o máximo semieixo maior, a letra `E` a máxima excentricidade, `DA` e `DE` as respectivas variâncias do semieixo e da excentricidade, `MabsEE0` a máxima diferença em excentricidade, e `Y` a média do indicador MEGNO. Todos os arquivos são compostos de uma linha inicial, indicando o instante em que os indicadores foram calculados, seguidos de várias linhas que consistem numa matriz que representa à grade. Cada linha dessa matriz corresponde a uma linha da grade de *threads* e cada coluna representa uma coluna da grade *threads*. Assim por exemplo, se quisermos o indicador da *thread* na linha 4 e na coluna 5 da grade, devemos procurar na linha 6 e na coluna 6 deste arquivo.

3.4.2.3 Arquivos auxiliares.

Possuem um formato matricial semelhante ao dos arquivos de indicadores de caos, mas armazenam o semieixo maior e a excentricidade (ou inclinação) iniciais. O nome de cada arquivo é composto pelo mesmo prefixo `CAOS_FILE` seguido de `A0` e `E0` (ou `I0`), conforme corresponda.

São gerados também outros dois arquivos auxiliares, denominados `megno` e `norma`, que armazenam, respectivamente, o valor do MEGNO em cada instante de tempo e o valor da norma do vetor variação em cada instante de tempo. Estes arquivos auxiliam na detecção de problemas com o cálculo do valor médio do MEGNO e podem ser dispensados na maioria das aplicações.

Capítulo 4

Aplicações

Como comentamos no início desta tese, as aplicações que apresentaremos a seguir visam complementar e auxiliar estudos sobre a determinação de parâmetros dinâmicos e físicos em sistemas de exoplanetas. Em todos os casos, o principal objetivo dos trabalhos foi o de aprimorar ou refinar os valores dos parâmetros orbitais e das massas dos planetas, com base na análise estatística de variações de tempo de trânsito e na análise dinâmica.

Em sistemas multi-planetários, a interação gravitacional mútua pode ser detectada sob certas condições. Em particular, os tempos centrais de trânsitos consecutivos de um mesmo planeta, t_C, podem não ocorrer exatamente em intervalos uniformes de efemérides. Em vez disso, devido às perturbações gravitacionais de outros planetas, os valores de t_C podem se adiantar ou atrasar levemente, gerando as denominadas variações de tempo de trânsito ou TTVs (do inglês *Transit Timing Variations*). A análise das TTVs pode fornecer uma melhor caracterização das órbitas e também das massas dos planetas envolvidos, ou pelo menos impor valores limites (MIRALDA-ESCUDÉ, 2002; AGOL *et al.*, 2005; HOLMAN e MURRAY, 2005). As TTVs têm sido usadas principalmente para confirmar e caracterizar sistemas em que todos os planetas apresentam trânsitos (HOLMAN *et al.*, 2010; LISSAUER *et al.*, 2011; STEFFEN *et al.*, 2012; XIE, 2014), mas elas também podem ser usadas para detectar e caracterizar companheiros planetários que não transitam (BALLARD *et al.*, 2011; NESVORNÝ *et al.*, 2012, 2013, 2014; DAWSON *et al.*, 2014; MANCINI *et al.*, 2016). Quando não há medidas de velocidades radiais disponíveis, a observação e análise das TTVs oferecem a única ferramenta atualmente disponível para determinar as massas dos planetas. O interesse pelas massas está no fato de que, combinadas com as estimativas dos raios planetários obtidas a partir dos trânsitos, permitem estimar as densidades planetárias e assim colocar restrições aos modelos de estrutura interna.

Nas aplicações apresentadas a seguir, a análise das TTVs foi feita utilizando uma ferramenta estatística de inferência bayesiana denominada `MultiNest` (FEROZ *et al.*, 2009, 2019). Os detalhes sobre esta metodologia encontram-se descritos na tese de doutorado de SAAD-OLIVERA (2019). Em resumo, o procedimento consiste em assumir um modelo

de sistema planetário com pelos menos dois planetas e tentar reproduzir, através de uma integração numérica, os sinais das TTVs observadas. O algoritmo `MultiNest` é o que se encarrega de encontrar o conjunto de parâmetros, isto é, elementos orbitais e massas dos planetas, que produzem o melhor ajuste das TTVs calculadas às observadas.

Os valores assim ajustados são posteriormente testados em termos de estabilidade dinâmica, para o qual aplicamos o algoritmo desenvolvido nesta tese. Estes testes são importantes não apenas para validar os resultados do ajuste e verificar a estabilidade do sistema dentro dos intervalos de incerteza dos parâmetros, mas também para entender a dinâmica do sistema. Um dos aspectos relevantes da dinâmica é o fato de que todos os sistemas estudados na literatura que apresentam TTVs possuem configurações quase-ressonantes entre os planetas, isto é, os planetas encontram-se muito próximos de ressonâncias mútuas de movimentos médios (mas sem chegar a estar capturados nas ressonâncias). Esta característica está vinculada a um viés observacional, já que a proximidade da ressonância contribui precisamente para amplificar o sinal dos TTVs (AGOL *et al.*, 2005; HOLMAN e MURRAY, 2005; LITHWICK *et al.*, 2012). Entretanto, a proximidade da ressonância também impõe restrições aos valores dos parâmetros do sistema em termos de estabilidade, já que em geral as ressonâncias são passíveis de apresentar caos, especialmente para órbitas que evoluem próximas da separatriz.

4.1 O sistema Kepler-419

Os resultados apresentados nesta seção foram publicados em SAAD-OLIVERA *et al.* (2019).

O sistema Kepler-419 apresenta dois planetas, um dos quais, Kepler-419b, transita a estrela e o outro, Kepler-419c, não a transita. A existência de Kepler-419c foi deduzida a partir do sinal de TTV observado nos trânsitos de Kepler-419b (DAWSON *et al.*, 2012, 2014), que indicavam que Kepler-419c estaria localizado em uma órbita mais externa. A combinação de observações de trânsitos e TTVs com medidas de velocidade radial permitiu que ALMENARA et al. (2018) conseguissem caracterizar as massas da estrela e de ambos planetas, obtendo valores de $M_\star = 1,39 \pm 0,48\ M_\odot$, $M_b = 2,71 \pm 0,66\ M_{\rm Jup}$ e $M_c = 7,4 \pm 2,6\ M_{\rm Jup}$. Estes autores também concluíram que as massas dos planetas poderiam ser determinadas com medidas de TTV apenas, desde que um número suficiente destas estivesse disponível.

Assim, o principal objetivo do trabalho foi estimar as massas dos planetas utilizando o conjunto de medidas de 20 TTVs para Kepler-419b disponibilizado por HOLCZER *et al.* (2016). Os resultados forneceram valores de $M_b = 3,3^{+1,3}_{-1,2}\ M_{\rm Jup}$ e $M_c = 6,5^{+0,6}_{-0,5}\ M_{\rm Jup}$, ou $M_b = 3,8^{+1,8}_{-1,9}\ M_{\rm Jup}$ e $M_c = 7,4^{+2,5}_{-2,6}\ M_{\rm Jup}$, dependendo do valor assumido para a massa da estrela. Estes resultados serviram para validar a metodologia, mostrando que é possível obter estimativas boas das massas sem ter que recorrer à análise conjunta de trânsitos e

velocidades radiais. Isto é especialmente relevante quando se leva em consideração que medidas de velocidade radial só estão disponíveis, por enquanto, para um número muito limitado de sistemas.

Como complemento ao trabalho, nós aplicamos o código desenvolvido nesta tese para explorar o espaço de fases do sistema e a sua estabilidade. Uma característica peculiar de Kepler-419b é a sua alta excentricidade orbital ($e_b \simeq 0,82$). Este fato, somado à ampla gama de possíveis valores de massa estimados para este planeta e à proximidade dos mesmos à ressonância mútua de movimentos médios 10:1, levanta questionamentos acerca da possível estabilidade a longo prazo do sistema. ALMENARA *et al.* (2018) mostraram que, para as órbitas finais estimadas com o seu modelo, o sistema Kepler-419 está próximo de um ponto de equilíbrio estável de alta excentricidade no plano $(e_b, \varpi_b - \varpi_c)$. Aqui, exploramos a dinâmica do sistema usando mapas dinâmicos que abrangem os principais intervalos de incerteza nos parâmetros M_b e a_c.[1]. Nós escolhemos estes parâmetros porque eles não estão tão bem restringidos quanto os outros parâmetros. Os mapas foram construídos computando vários indicadores de caos em uma grade de 128×32 condições iniciais. As grades abrangem o intervalo de ± 15 dias e $\pm 0,1$ em torno dos valores do melhor ajuste para o período P_c e excentricidade e_c, respectivamente. Os outros elementos orbitais dos planetas foram fixados em seus valores de melhor ajuste. Os parâmetros usados na simulação estão resumidos na Tabela 4.1. Para cada um dos dois possíveis conjuntos de parâmetros encontradas no trabalho, calculamos três grades diferentes, assumindo três valores diferentes da massa M_b: um correspondente ao valor do melhor ajuste e dois correspondentes aos valores definidos por $\pm 1\sigma$. A massa de M_c foi sempre fixada no valor do melhor ajuste. Cada condição inicial foi simulada por um intervalo de 5×10^6 dias (equivalente a ~ 7000 revoluções[2] do planeta mais externo), com um passo de integração de 0,2 dias (equivalente a $\sim 1/300$ do período do planeta mais interno).

A Figura 4.1 mostra os resultados fornecidos pelo índice de máxima excentricidade, que neste caso corresponde ao valor máximo da excentricidade do planeta Kepler-419c ao longo da simulação. Como vimos, este índice é muito simples de calcular e fornece bastante informação sobre a estrutura do espaço de fase, identificando bem as regiões de estabilidade e caos. Comparando os painéis da esquerda na Figura 4.1 com os da direita, concluímos que o comportamento dinâmico global do sistema é independente dos parâmetros estelares assumidos. Por outro lado, o comportamento é fortemente dependente da massa de Kepler-419b. De fato, quanto maior a massa M_b (painéis inferiores), mais

[1] Os mapas não foram construídos variando M_b e a_c, porque queríamos analisar se os parâmetros encontrados estão próximos a ressonâncias, e as estruturas V típicas de ressonâncias aparecem somente em mapas de semi-eixo por excentricidade

[2] O tempo máximo de integração foi arbitrário, e foi escolhido como um tempo suficiente longo para os indicadores convergirem mas não muito longo de forma a tornar a simulação inviável em termos de tempo de execução.

Kepler-419 (Dawson et al. 2012)		
$M_\star$ ($M_\odot$)	$1.22^{+0.12}_{-0.08}$	
$R_\star$ ($R_\odot$)	$1.4^{+0.37}_{-0.21}$	
	Kepler-419b	**Kepler-419c**
Dynamical fit		
$M_p/M_\star$ ($\times 10^{-3}$)	$2.54^{+0.95}_{-1.01}$	$5.06^{+0.08}_{-0.15}$
P_p (d)	$69.732^{+0.001}_{-0.001}$	$694.3^{+10.1}_{-8.6}$
e_p	$0.82^{+0.03}_{-0.01}$	$0.162^{+0.009}_{-0.008}$
b_p	$0.2^{+0.1}_{-0.2}$	$10.0^{+6.3}_{-5.9}$
ϖ_p (°)	348^{+21}_{-14}	165^{+19}_{-13}
λ_p (°)	–	59^{+20}_{-13}
Ω_p (°)	270	268^{+142}_{-137}
δt (d)	$0.028^{+0.002}_{-0.001}$	–
Derived parameters		
M_p (M_{Jup})	$3.3^{+1.3}_{-1.2}$	$6.5^{+0.6}_{-0.5}$
a_p (au)	$0.3545^{+0.0014}_{-0.0009}$	$1.64^{+0.14}_{-0.09}$
i_p (°)	$89.7^{+0.6}_{-0.8}$	$87.71^{+1.3}_{-1.2}$
I_p (°)	$1.5^{+0.7}_{-0.9}$	$1.9^{+1.3}_{-1.2}$
I_{mut} (°)	–	$0.5^{+1.5}_{-0.5}$
R_p (R_{Jup})	$0.8^{+0.2}_{-0.1}$	–
ρ_p (g cm^{-3})	$6.8^{+6.0}_{-3.8}$	–

Kepler-419 (Almenara et al. 2018)		
$M_\star$ ($M_\odot$)	$1.39^{+0.48}_{-0.48}$	
$R_\star$ ($R_\odot$)	$1.8^{+0.2}_{-0.2}$	
	Kepler-419b	**Kepler-419c**
Dynamical fit		
$M_p/M_\star$ ($\times 10^{-3}$)	$2.60^{+0.89}_{-1.00}$	$5.07^{+0.08}_{-0.14}$
P_p (d)	$69.7332^{+0.001}_{-0.001}$	$694.7^{+10.1}_{-8.6}$
e_p	$0.82^{+0.02}_{-0.01}$	$0.162^{+0.009}_{-0.007}$
b_p	$0.26^{+0.11}_{-0.15}$	$8.9^{+6.3}_{-5.1}$
ϖ_p (°)	346^{+24}_{-12}	163^{+22}_{-12}
λ_p (°)	–	58^{+22}_{-11}
Ω_p (°)	270	264^{+143}_{-135}
δt (d)	$0.028^{+0.001}_{-0.001}$	–
Derived parameters		
M_p (M_{Jup})	$3.8^{+1.8}_{-1.9}$	$7.4^{+2.5}_{-2.6}$
a_p (au)	$0.370^{+0.005}_{-0.005}$	$1.71^{+0.58}_{-0.58}$
i_p (°)	$88.1^{+0.8}_{-1.0}$	$87.8^{+2.1}_{-2.0}$
I_p (°)	$1.8^{+0.8}_{-1.1}$	$2.1^{+1.7}_{-1.4}$
I_{mut} (°)	–	$0.4^{+3.3}_{-0.4}$
R_p (R_{Jup})	$1.1^{+0.1}_{-0.1}$	–
ρ_p (g cm^{-3})	$3.7^{+2.3}_{-2.7}$	–

Tabela 4.1: Dados do sistema Kepler-419.

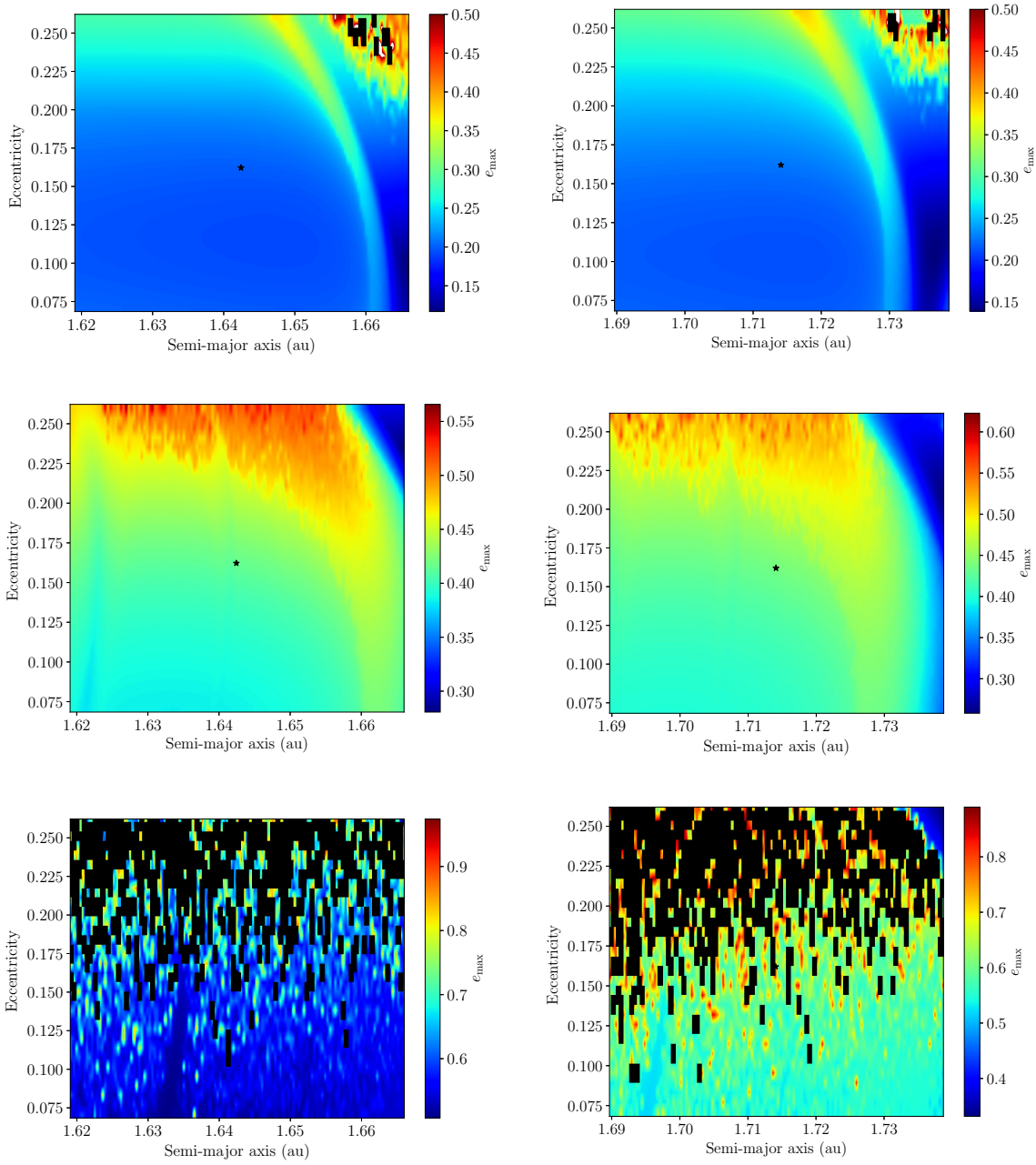

Figura 4.1: Mapas dinâmicos do índice de excentricidade máxima calculados para Kepler-419c. Cada coluna de painéis corresponde a cada uma das duas possíveis soluções de parâmetros encontradas para o sistema a partir da analise de TTVs. Cada linha de painéis corresponde a diferentes valores da massa M_b, do menor valor (em cima) ao maior valor (em baixo) dentro de 1σ de incerteza. Os valores do melhor ajuste para Kepler-419c são indicados pela estrela preta no centro da grade. As regiões azul/ciano indicam movimento estável. As regiões pretas condições iniciais instáveis que levaram a encontros próximos entre os corpos (planeta-planeta ou planeta-estrela) ou à ejeção de um planeta.

instável o sistema se torna. A cor preta na Figura 4.1 indica condições iniciais que levaram a encontros próximos entre os planetas a uma distância mútua $r < R_b^{\rm Hill} + R_c^{\rm Hill}$, ou que levaram um dos planetas a uma distância astrocêntrica $r < 2R_\star$ ou $r > 10$ au, ou a uma distância pericêntrica $q < 2R_\star$, constituindo órbitas instáveis. Resultados semelhantes são obtidos analisando-se outros índices de caos, como a variância em excentricidade ou em semieixo.

Os mapas dinâmicos mostram claramente a localização da ressonância 10:1, com o clássico formato em V que se evidencia parcialmente na margem direita dos mapas. Notamos que, para valores baixos de M_b, a separatriz da ressonância mostra uma faixa de caos estreita e que a solução nominal do planeta, representada pela estrela branca, se encontra fora desta faixa. A medida que a massa M_b aumenta, a ressonância aumenta a sua largura e a faixa de caos na separatriz se torna mais e mais proeminente, até que a solução nominal fica totalmente imersa numa região altamente caótica. Podemos concluir que, para os valores da solução nominal do sistema, representada pelos painéis da fila do meio, o sistema é marginalmente estável.

4.2 O sistema Kepler-59

Os resultados apresentados nesta seção foram publicados em SAAD-OLIVERA *et al.* (2020).

O sistema Kepler-59 está constituído por dois planetas, Kepler-59b, mais interno e Kepler-59c, mais externo, sendo que ambos transitam a estrela. Até o momento, a melhor determinação dos parâmetros deste sistema tinha sido fornecida por STEFFEN et al. (2013) quem, combinando a análise de TTVs com estudos de evolução dinâmica, conseguiram impor limites superiores para as massas dos planetas de forma a garantir que o sistemas fosse dinamicamente estável. Em particular, estes autores encontraram que $M_b < 2.05\ M_{\rm Jup}$ e $M_c < 1.37\ M_{\rm Jup}$.

O principal objetivo do trabalho foi aprimorar os valores das massas, impondo limites mais restritos. Para tal, foram analisados dois conjuntos diferentes de TTVs:

1. Um conjunto de 72 TTVs para Kepler-59c apenas, fornecido por HOLCZER et al. (2016)

2. Um conjunto de 71 TTVs para Kepler-59c e 110 TTVs para Kepler-59b fornecido por ROWE et al. (2015)

Para cada um destes conjuntos, o ajuste com `MultiNest` fornece duas possíveis soluções, tendo ao total quatro conjuntos de parâmetros possíveis para o sistema. Cabe destacar que as duas soluções obtidas com os dados de Holczer são compatíveis com as duas soluções obtidas a partir dos dados de Rowe, dentro de 1σ de incerteza. Entretanto,

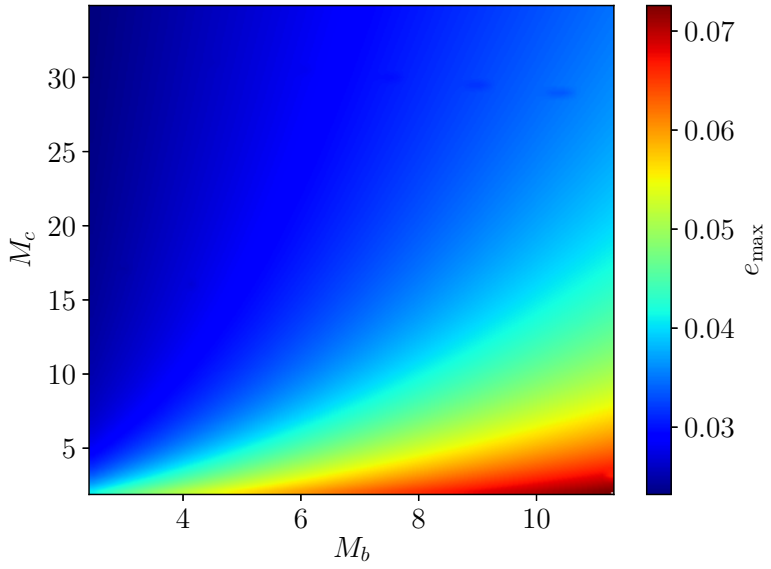

Figura 4.2: Mapa dinâmico do sistema Kepler-59 no espaço de massas. O indicador corresponde à excentricidade máxima de Kepler-59c.

estas últimas apresentam erros nos parâmetros que podem ser até 3 vezes menores que no caso das primeiras. Isto é de certa forma esperado, já que Rowe fornece dados de TTVs para ambos os planetas, fazendo com que o ajuste dos parâmetros seja mais preciso. Considerando os resultados inferidos, os valores das massas para os dois planetas podem variar nos intervalos $2,2 \leqslant M_b \leqslant 10\ M_\oplus$ e $1 \leqslant M_c \leqslant 33\ M_\oplus$, com valores mais prováveis de $M_b \simeq 4\ M_\oplus$ e $M_c \simeq 3\ M_\oplus$. Considerando que os raios estimados a partir da curva de luz dos trânsitos são $R_b = 1,5\ R_\oplus$ e $R_c = 2,2\ R_\oplus$, estes valores indicariam a presença de uma super-Terra e um mini-Netuno no sistema. Além disso, em função dos respectivos períodos orbitais, os planetas se encontrariam próximos de uma ressonância 2:3 de movimentos médios.

Utilizando o nosso código, construímos mapas dinâmicos para avaliar a estabilidade do sistema no espaço de parâmetros M_b, M_c (Figura 4.2). Consideramos uma grade uniforme de 64×64 condições iniciais abrangendo aproximadamente os intervalos de massa mencionados acima, e propagamos as órbitas por 2×10^5 dias (equivalente a ~ 12000 revoluções do planeta mais externo), com um passo de integração de 0,04 dias (equivalente a $\sim 1/300$ do período do planeta mais interno). Os dados do sistema estão resumidos na Tabela 4.2. Constatamos assim que o sistema é estável ao longo de todo o intervalo de massas possíveis.

	S_1^H		S_2^H	
	Kepler-59b	Kepler-59c	Kepler-59b	Kepler-59c
Dynamical fit				
$M_p/M_\star$ ($\times 10^{-3}$)	$0.15^{+0.06}_{-0.04}$	$0.4^{+0.3}_{-0.2}$	$0.07^{+0.01}_{-0.01}$	$0.05^{+0.03}_{-0.02}$
P_p (d)	$11.879^{+0.009}_{-0.007}$	$17.970^{+0.001}_{-0.002}$	$11.869^{+0.003}_{-0.002}$	$17.969^{+0.002}_{-0.002}$
e_p	$0.03^{+0.03}_{-0.02}$	$0.02^{+0.02}_{-0.01}$	$0.07^{+0.02}_{-0.02}$	$0.04^{+0.02}_{-0.02}$
b_p	$0.4^{+0.3}_{-0.2}$	$0.5^{+0.3}_{-0.3}$	$0.4^{+0.3}_{-0.2}$	$0.5^{+0.3}_{-0.3}$
ϖ_p (°)	209^{+109}_{-172}	74^{+65}_{-150}	296^{+115}_{-73}	65^{+70}_{-124}
λ_p (°)	113^{+177}_{-47}	–	155^{+163}_{-76}	–
Ω_p (°)	272^{+117}_{-121}	270	271^{+117}_{-115}	270
δt (d)	–	$0.0206^{+0.007}_{-0.007}$	–	$0.027^{+0.007}_{-0.007}$
Derived parameters				
M_p ($M_\oplus$)	$7.1^{+3.0}_{-2.1}$	$20.9^{+13.6}_{-12.5}$	$3.5^{+0.6}_{-0.7}$	$2.1^{+1.4}_{-1.2}$
a_p (au)	$0.112^{+0.002}_{-0.002}$	$0.148^{+0.002}_{-0.002}$	$0.112^{+0.002}_{-0.002}$	$0.148^{+0.002}_{-0.002}$
i_p (°)	$88.6^{+1.0}_{-0.8}$	$88.7^{+0.7}_{-0.7}$	$88.9^{+0.9}_{-0.8}$	$89.1^{+0.7}_{-0.7}$
I_p (°)	$1.3^{+1.0}_{-0.8}$	$1.2^{+0.7}_{-0.7}$	$1.01^{+0.97}_{-0.83}$	$0.85^{+0.74}_{-0.73}$
I_{mut} (°)	–	$0.12^{+1.27}_{-0.12}$	–	$0.15^{+1.21}_{-0.15}$
R_p ($R_\oplus$)	$1.5^{+0.1}_{-0.1}$	$2.2^{+0.1}_{-0.1}$	$1.5^{+0.1}_{-0.1}$	$2.2^{+0.1}_{-0.1}$
ρ_p (g cm^{-3})	11^{+4}_{-5}	11^{+7}_{-9}	$5.6^{+1.0}_{-1.7}$	$1.1^{+0.7}_{-0.9}$

	S_1^R		S_2^R	
	Kepler-59b	Kepler-59c	Kepler-59b	Kepler-59c
Dynamical fit				
$M_p/M_\star$ ($\times 10^{-3}$)	$0.11^{+0.08}_{-0.04}$	$0.10^{+0.08}_{-0.04}$	$0.07^{+0.01}_{-0.01}$	$0.06^{+0.02}_{-0.02}$
P_p (d)	$11.8715^{+0.0005}_{-0.0005}$	$17.9742^{+0.0013}_{-0.0009}$	$11.8714^{+0.0004}_{-0.0004}$	$17.9737^{+0.0008}_{-0.0008}$
e_p	$0.05^{+0.09}_{-0.03}$	$0.05^{+0.08}_{-0.03}$	$0.09^{+0.09}_{-0.05}$	$0.09^{+0.08}_{-0.05}$
b_p	$0.5^{+0.3}_{-0.3}$	$0.5^{+0.3}_{-0.3}$	$0.5^{+0.3}_{-0.3}$	$0.5^{+0.3}_{-0.3}$
ϖ_p (°)	309^{+148}_{-86}	21^{+57}_{-62}	301^{+150}_{-69}	29^{+51}_{-89}
Ω_p (°)	273^{+109}_{-106}	270	271^{+110}_{-112}	270
δt (d)	$1.079^{+0.008}_{-0.008}$	$0.004^{+0.004}_{-0.002}$	$1.079^{+0.008}_{-0.008}$	$0.004^{+0.005}_{-0.003}$
Derived parameters				
M_p ($M_\oplus$)	$5.3^{+4.0}_{-2.1}$	$4.6^{+3.6}_{-2.0}$	$3.0^{+0.8}_{-0.8}$	$2.6^{+0.9}_{-0.8}$
a_p (au)	$0.112^{+0.002}_{-0.002}$	$0.148^{+0.002}_{-0.002}$	$0.112^{+0.002}_{-0.002}$	$0.148^{+0.002}_{-0.002}$
i_p (°)	$88.38^{+1.07}_{-1.08}$	$88.8^{+0.8}_{-0.8}$	$88.4^{+1.1}_{-1.0}$	$88.8^{+0.8}_{-0.8}$
I_p (°)	$1.66^{+1.07}_{-1.09}$	$1.6^{+0.8}_{-0.8}$	$1.6^{+1.2}_{-1.1}$	$1.6^{+0.8}_{-0.8}$
I_{mut} (°)	–	$0.1^{+1.3}_{-0.1}$	–	$0.04^{+1.40}_{-0.04}$
R_p ($R_\oplus$)	$1.5^{+0.1}_{-0.1}$	$2.2^{+0.1}_{-0.1}$	$1.5^{+0.1}_{-0.1}$	$2.2^{+0.1}_{-0.1}$
ρ_p (g cm^{-3})	8^{+6}_{-4}	$2.4^{+2}_{-1.5}$	$4.9^{+1.3}_{-1.9}$	$1.4^{+0.5}_{-0.6}$

Tabela 4.2: Dados do sistema Kepler-59.

4.3 O sistema Kepler-46

Os resultados apresentados nesta seção foram submetidos para publicação na revista *Astronomy and Computing* (COSTA DE SOUZA *et al.*, 2020).

O sistema Kepler-46 é constituído por dois planetas, sendo que um deles, Kepler-46b, transita a estrela central enquanto que o outro, Kepler-46c, não transita. Assim como no caso do sistema Kepler-419, a presença de Kepler-46c foi inferida a partir dos TTVs observados em Kepler-46b. Esse sistema foi inicialmente estudado por NESVORNÝ *et al.* (2012), quem analisando o sinal de TTVs encontraram duas possíveis soluções para os parâmetros orbitais de Kepler-46c: uma em que os planetas ficam próximos de uma ressonância de movimentos médios 3:5 (períodos $P_b \simeq 33,6$ dias e $P_c \simeq 57,3$ dias), e outra em que ficam próximos de uma ressonância 2:5 (períodos $P_b \simeq 33,6$ dias e $P_c \simeq 81,5$ dias). Entretanto, NESVORNÝ *et al.* (2012) descartam esta segunda solução, já que a mesma não consegue reproduzir os tempos de duração de trânsito observados (TDV, do inglês *Transit Duration Variation*). Por outro lado, estes autores conseguem uma boa estimativa da massa $M_c \simeq 0,38\ M_{\rm Jup}$, mas apenas um limite superior para a massa $M_b < 6\ M_{\rm Jup}$, esta última determinada a partir de análise da estabilidade dinâmica.

Mais recentemente, SAAD-OLIVERA *et al.* (2017) refazem a análise dos TTVs deste sistema utilizando uma amostra maior de trânsitos e aplicando `MultiNest`. Estes autores encontram novamente as duas soluções mencionadas anteriormente, porém a segunda delas apresenta uma baixa significância estatística e poderia ser descartada. Além disso, estes autores conseguem restringir os valores das massas de ambos planetas a partir da análise dos TTVs, obtendo valores de $M_b \simeq 0,88\ M_{\rm Jup}$ e $M_c \simeq 0,36\ M_{\rm Jup}$. Isto significa que o sistema está constituído por um planeta tipo Júpiter e outro tipo Saturno em uma configuração de quase ressonância 2:5, constituindo assim uma espécie de Sistema Solar "encolhido" que, por algum motivo, evoluiu de maneira muito diferente do nosso sistema[3].

Por outro lado, NESVORNÝ *et al.* (2012) tinham levantado a possibilidade de que poderia haver no sistema Kepler-46 um terceiro planeta, Kepler-46d, de tipo terrestre, que apresentaria trânsitos e teria um período orbital de 6,8 dias, porém a sua existência não foi confirmada até o momento.

Neste trabalho, a nossa abordagem é explorar o espaço de fase do sistema Kepler-46 através de mapas dinâmicos, visando identificar as regiões caóticas e estáveis e correlacionar os resultados com a configuração atual do sistema, sua possível evolução no passado e as similaridades que possa apresentar com o nosso Sistema Solar. Para isto, analisamos dois modelos diferentes. Num primeiro modelo, consideramos apenas dois planetas no sistema, mantendo os parâmetros orbitais de um deles (Kepler-46b) fixos e variando os parâmetros do outro (Kepler-46c). Posteriormente, consideramos um modelo com

[3]No Sistema Solar, Júpiter e Saturno se encontram próximos de uma ressonância de movimentos médios 2:5. Os modelos mais recentes de formação planetária mostram que estes dois planetas tiveram um papel decisivo na evolução dinâmica primordial de todo o sistema (e.g. WALSH, 2011).

três planetas no sistema, e mapeamos os espaço de condições iniciais do terceiro planeta (Kepler-46d), mantendo os parâmetros dos outros dois (Kepler-46b,c) fixos. Os dados do sistema estão resumidos na Tabela 4.3.

No modelo com dois planetas, variamos as condições iniciais numa grade é uniforme de 1024×32 [4] no espaço de período vs. excentricidade, cobrindo os seguintes intervalos: $35 \leqslant P_c \leqslant 90$ dias e $0 \leqslant e_c \leqslant 0,4$. Os restantes parâmetros iniciais de ambos planetas foram fixados nos valores ajustados por SAAD-OLIVERA *et al.* (2017). No modelo com três planetas, variamos as condições iniciais numa grade uniforme de 512×32 cobrindo os intervalos $5 \leqslant P_d \leqslant 32$ dias e $0 \leqslant e_d \leqslant 0,6$. Neste caso, os restantes parâmetros iniciais de Kepler-46d foram fixados em valores arbitrários e, em particular, analisamos nove grades diferentes considerando as combinações de valores de inclinação $i_d = 0°, 10°$ e massa $M_d = 0,5\,,\ 1,0\,,\ 5,0\,,\ 10,0\,,\ 15,0\ M_\oplus$. Em ambos modelos, as condições iniciais foram simuladas por um intervalo de tempo de 10^6 dias (equivalente a $\sim$ 10000-20000 órbitas do planeta mais externo). No modelo com dois planetas, o tamanho do passo de integração foi de 0,1 dias e no modelo com três planetas foi de 0,02 dias (equivalentes a $\sim 1/300$ do período do planeta mais interno).

4.3.1 Resultados

As Figuras 4.3 a 4.7 exibem os valores dos diversos indicadores de caos no final da simulação. Os gráficos são qualitativamente semelhantes e, portanto, para os quatro indicadores a análise é equivalente. As regiões na cor preta/verde escuro correspondem às configurações nas quais Kepler-46c é instável e eventualmente colide com o Kepler-46b. Nos mapas é possível identificar várias estruturas em forma de V, correspondentes às principais ressonâncias de movimentos médios. A ressonância 1:2 aparece claramente no meio do mapa ($P \sim 66$ dias), mas o movimento dentro dela para baixas excentricidades ($e \lesssim 0,2$) se apresenta bastante instável. Por outro lado, a ressonância 2:5 em $P \sim 85$ dias apresenta uma ilha de estabilidade para órbitas de alta excentricidade ($e \gtrsim 0,2$), e é possível ainda resolver algumas estruturas internas que provavelmente estão associadas a ressonâncias secundárias. A solução do melhor ajuste para o Kepler-46c se encontra à direita da ressonância 3:5 ($P \sim 55$ dias), em uma região bastante estável. A princípio, o par planetário poderia ter migrado primordialmente ao longo de um intervalo significativo de distâncias, atravessando várias ressonâncias de movimentos médios, porém nunca poderiam ter migrado mais próximo do que a localização da ressonância 2:3 em $P \sim 50$ dias.

As Figuras 4.8 a 4.13 apresentam os mapas dinâmicos para diferentes índices de caos no modelo com três planetas, assumindo dois valores extremos de massa para Kepler-46d: uma massa tipo Marte ($0,5\ M_\oplus$) e uma massa tipo mini-Netuno ($15\ M_\oplus$). Os mapas apresentam uma estrutura que se assemelha à observada no cinturão de asteroides no nosso

[4] A grade é retangular porque o intervalo de valores de períodos é maior que o intervalo de valores das excentricidades

	Kepler-46b	*Kepler-46c*
Transit Fit		
$R_P/R_\star$	$0.0887^{+0.0010}_{-0.0012}$	...
b_P	$0.757^{+0.022}_{-0.027}$	...
Dynamical Fit		
$M_P/M_\star$ ($\times 10^{-4}$)	$9.372^{+3.941}_{-3.618}$	$3.835^{+0.057}_{-0.055}$
P_P (days)	$33.648^{+0.004}_{-0.005}$	$57.325^{+0.116}_{-0.098}$
e_P	$0.0321^{+0.0069}_{-0.0078}$	$0.0354^{+0.0057}_{-0.0059}$
b_P	$0.757^{+0.022}_{-0.027}$	$1.483^{+0.418}_{-0.322}$
ϖ_P (°)	$264.2^{+8.2}_{-8.9}$	$294.16^{+8.70}_{-6.42}$
λ_P (°)	...	$338.0^{+0.3}_{-0.3}$
Ω_P (°)	270	$261.4^{+22.7}_{-24.3}$
δt (days)	$0.0130^{+0.0006}_{-0.0006}$	...
Secondary Parameters		
M_P (M_J)	$0.885^{+0.374}_{-0.343}$	$0.362^{+0.016}_{-0.016}$
a_P (au)	$0.1971^{+0.0001}_{-0.0001}$	$0.2811^{+0.0003}_{-0.0003}$
i_P (°)	$89.04^{+0.14}_{-0.14}$	$88.66^{+0.26}_{-0.27}$
I_P (°)	$0.957^{+0.028}_{-0.034}$	$1.35^{+0.38}_{-0.29}$
$I_{\rm mut}$ (°)	...	$0.43^{+0.40}_{-0.26}$
R_P (R_J)	$0.810^{+0.035}_{-0.036}$	...
ρ (g cm^{-3})	$2.069^{+0.913}_{-1.136}$	...
	Kepler-46	
Stellar Parameters		
$M_\star$ ($M_\odot$)	$0.902^{+0.040}_{-0.038}$	
$R_\star$ ($R_\odot$)	$0.938^{+0.038}_{-0.039}$	
$\rho_\star$ (g cm^{-3})	$1.54^{+0.22}_{-0.17}$	
$\log g_\star$ [a]	$4.447^{+0.040}_{-0.035}$	
$T_{\rm eff}$ (K)	5155^{+150}_{-150}	
$L_\star$ ($L_\odot$)	$0.556^{+0.078}_{-0.070}$	
M_V	$5.60^{+0.17}_{-0.17}$	
Age (Gyr)	$9.7^{+3.7}_{-3.5}$	
Distance (pc)	855^{+68}_{-65}	
$[M/H]$	$0.41^{+0.10}_{-0.10}$	

Tabela 4.3: Dados do sistema Kepler-46.

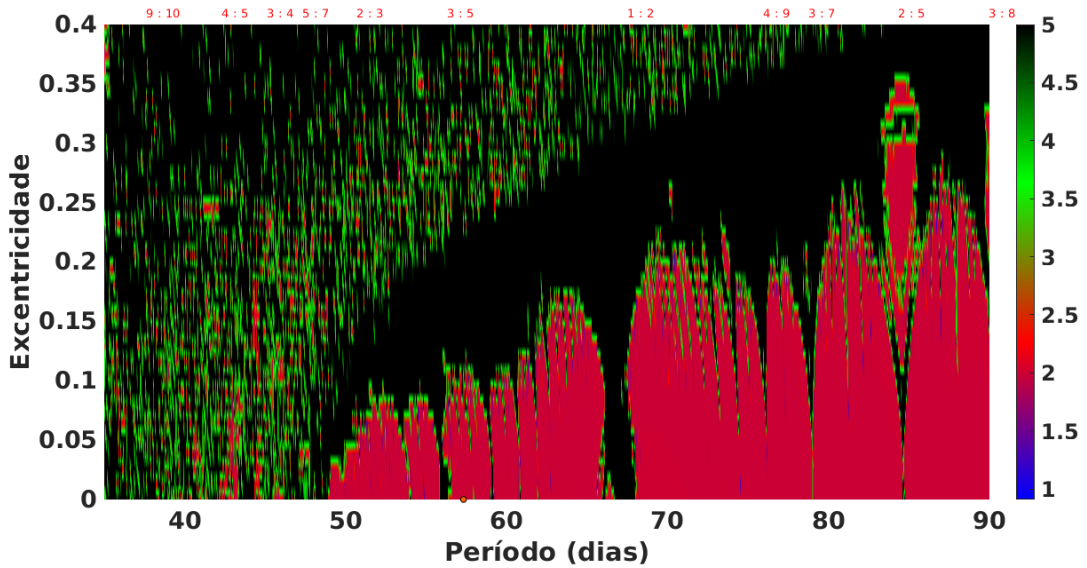

Figura 4.3: Mapa dinâmico do MEGNO no modelo de dois planetas. Note-se a diferença na escala horizontal em relação às figuras seguintes.

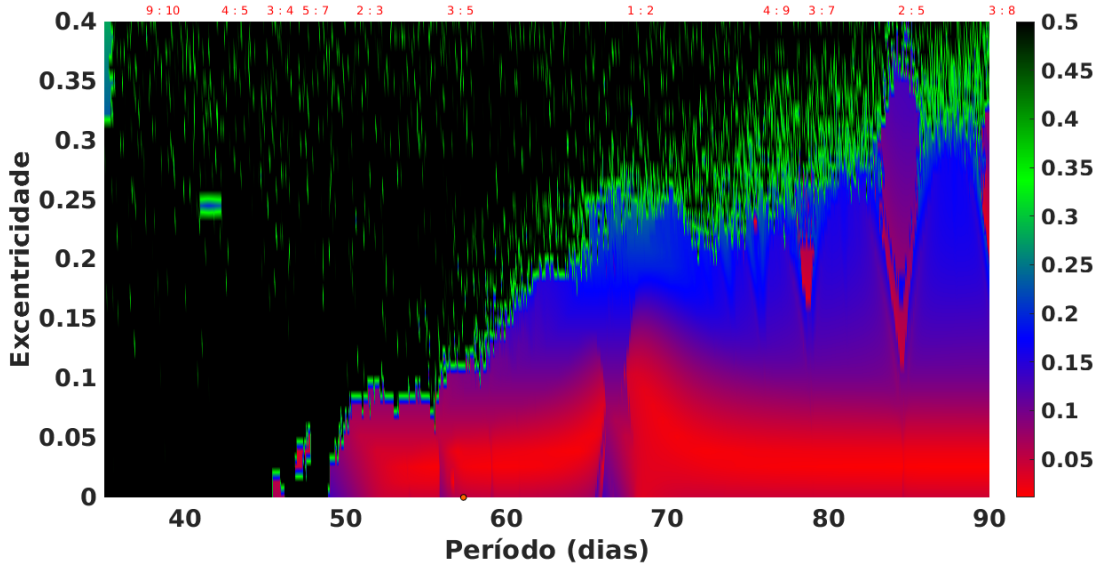

Figura 4.4: Mapa dinâmico da máxima variação em excentricidade no modelo de dois planetas. As condições iniciais da grade e o índice de caos correspondem a Kepler-46c. O período é em dias. Na parte superior do gráfico são indicadas as localizações de diferentes ressonâncias de movimentos médios.

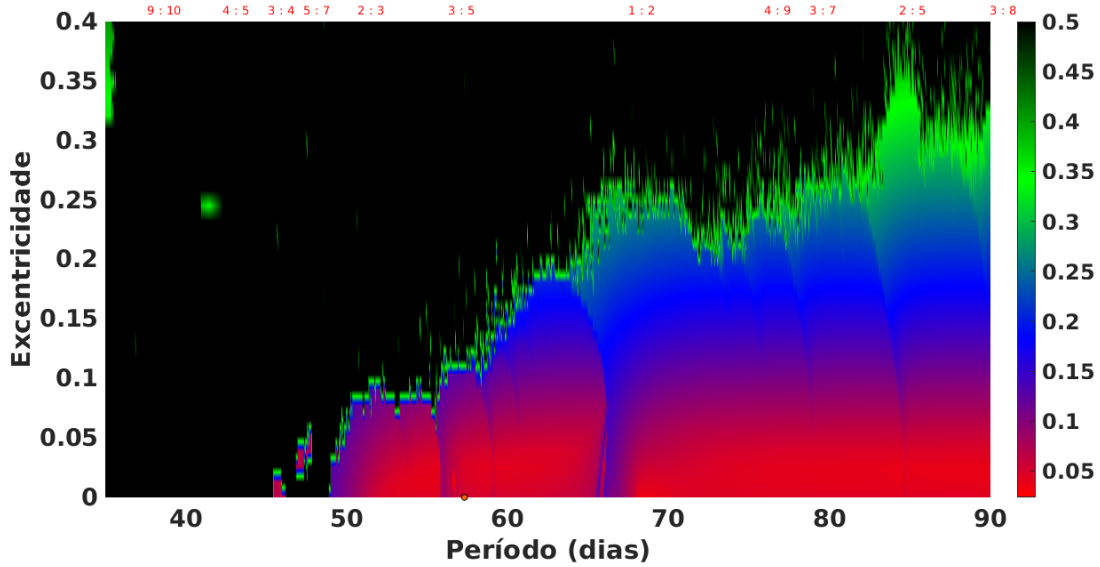

Figura 4.5: Mapa dinâmico da máxima excentricidade no modelo de dois planetas. As condições iniciais da grade e o índice de caos correspondem a Kepler-46c. O período é em dias.

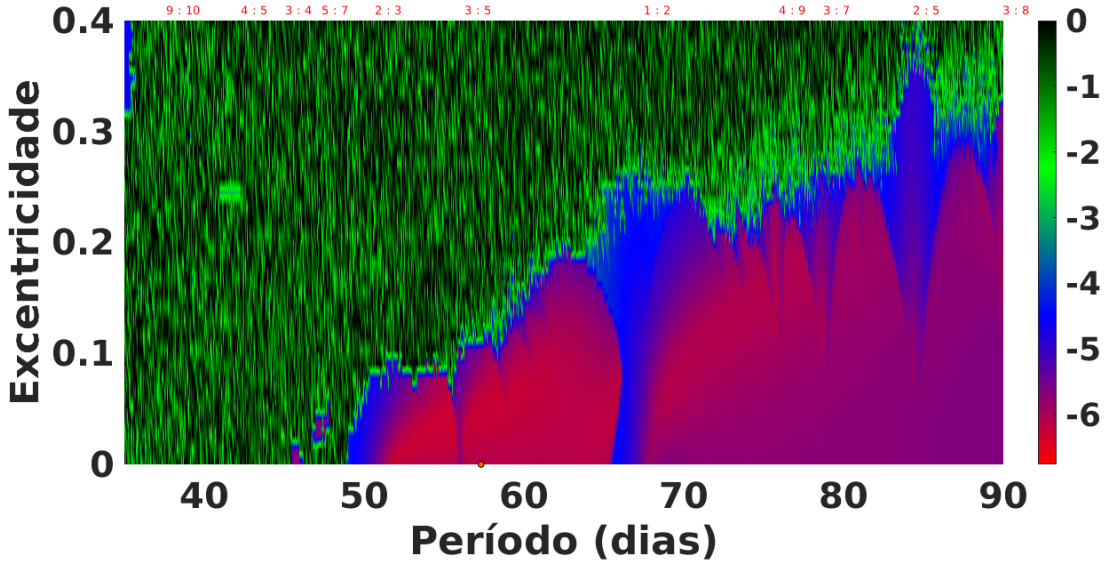

Figura 4.6: Mapa dinâmico da variância em semieixo maior no modelo de dois planetas. As condições iniciais da grade e o índice de caos correspondem a Kepler-46c. O período é em dias.

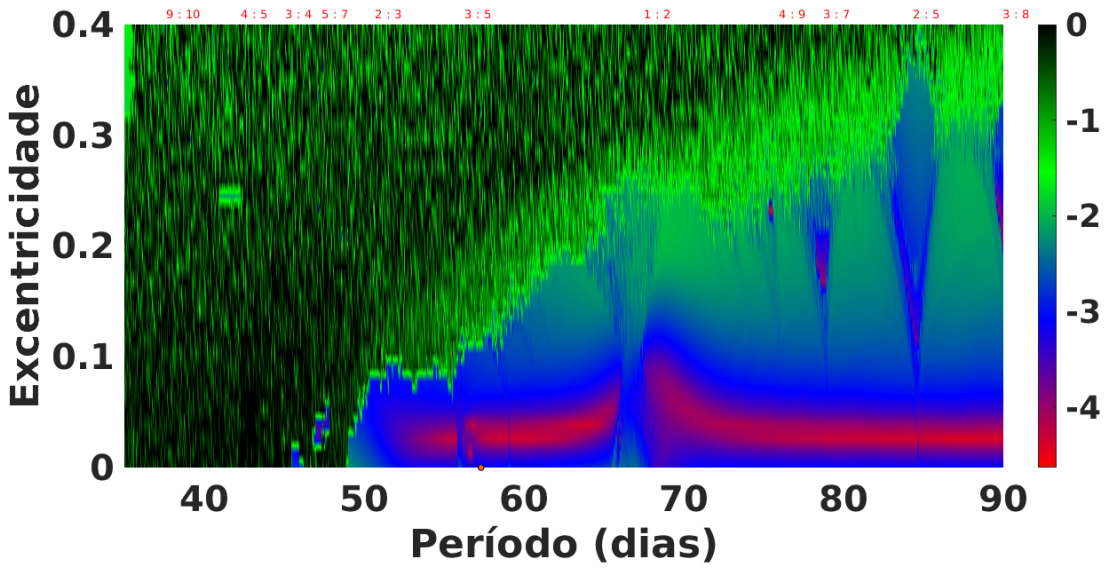

Figura 4.7: Mapa dinâmico da variância em excentricidade no modelo de dois planetas. As condições iniciais da grade e o índice de caos correspondem a Kepler-46c. O período é em dias.

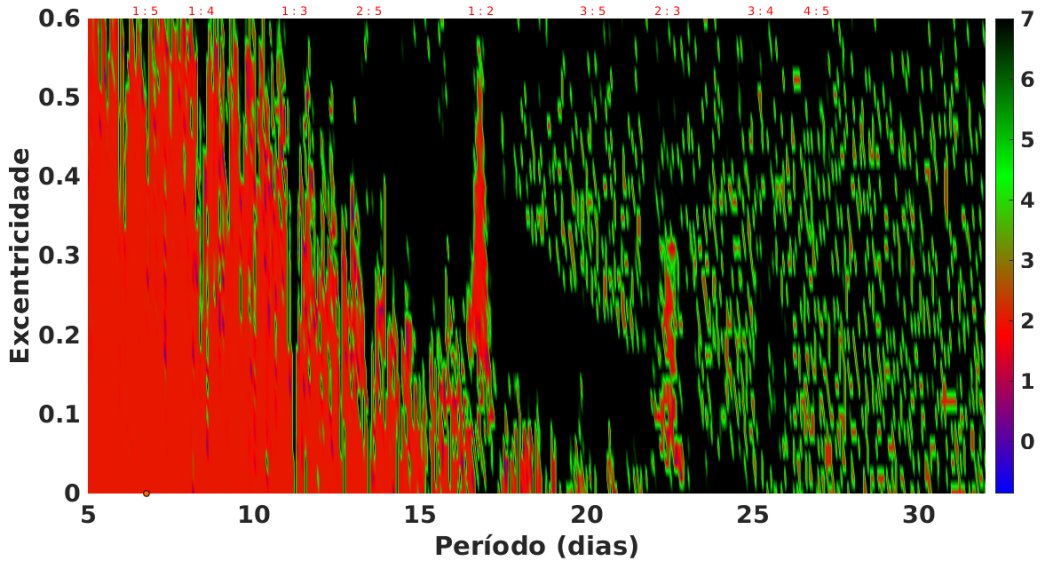

Figura 4.8: Mapa dinâmico do MEGNO no modelo de três planetas. As condições iniciais da grade correspondem a um Kepler-46d fictício com massa de 0,5 $M_{\oplus}$. O MEGNO caracteriza a estabilidade do sistema como um todo. O período é em dias.

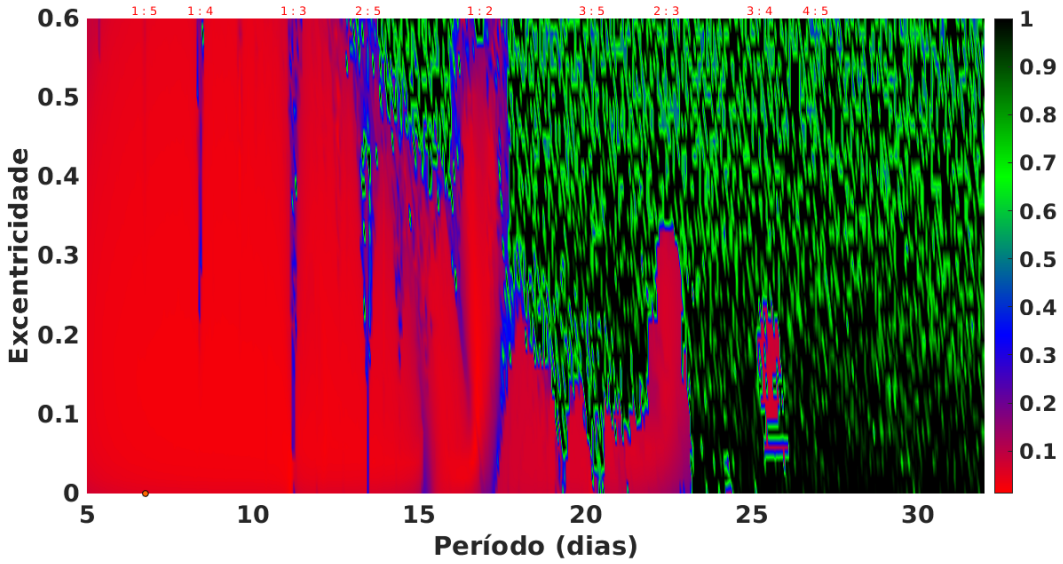

Figura 4.9: Mapa dinâmico da máxima variação em excentricidade no modelo de três planetas. As condições iniciais da grade e o índice de caos correspondem a um Kepler-46d fictício com massa de 0,5 $M_{\oplus}$. O período é em dias.

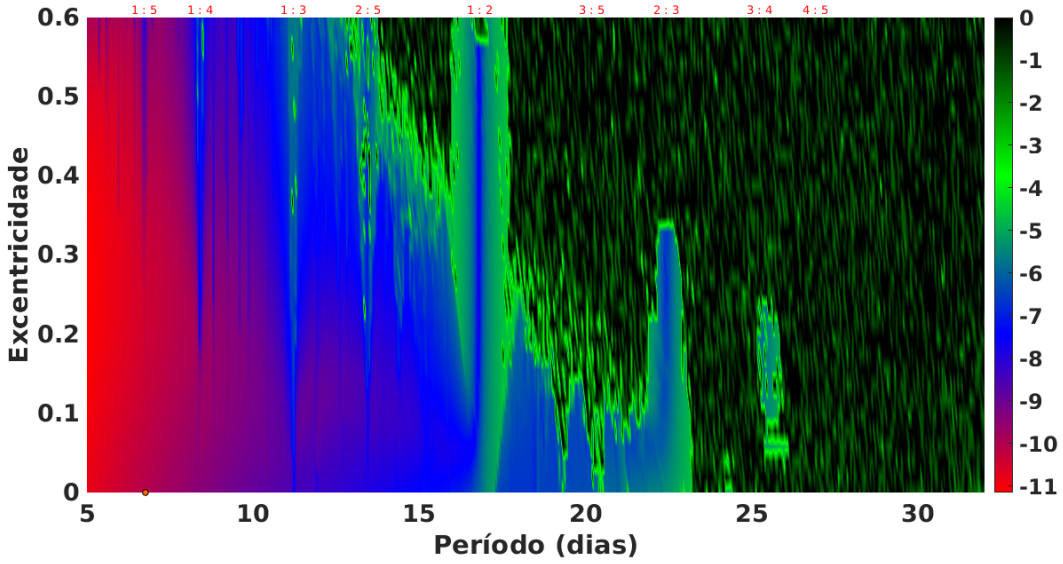

Figura 4.10: Mapa dinâmico da variância em semieixo maior no modelo de três planetas. As condições iniciais da grade e o índice de caos correspondem a um Kepler-46d fictício com massa de 0,5 $M_{\oplus}$. O período é em dias.

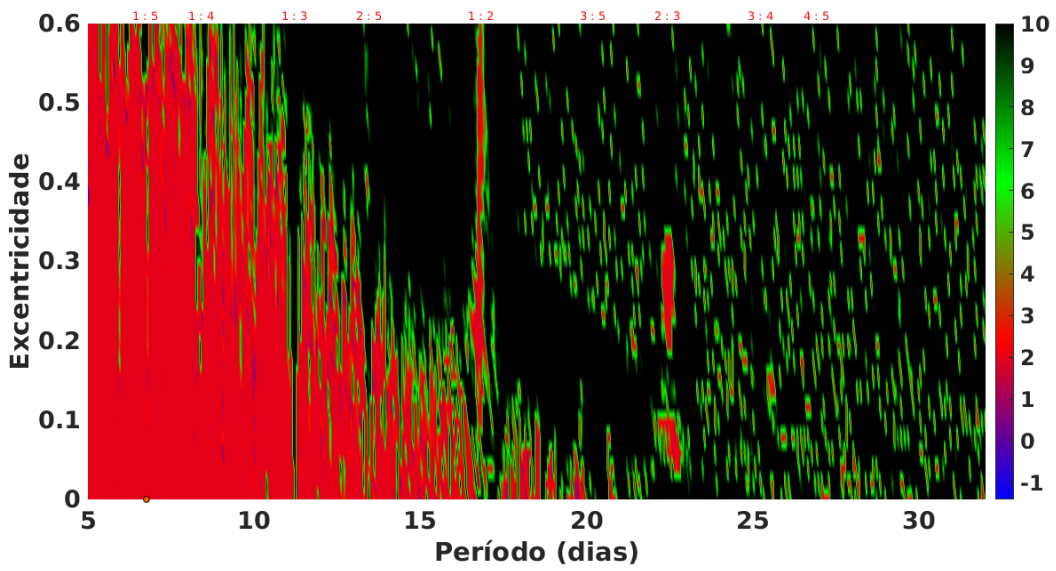

Figura 4.11: Mapa dinâmico do MEGNO no modelo de três planetas. As condições iniciais da grade correspondem a um Kepler-46d fictício com massa de 15 $M_{\oplus}$. O período é em dias. O MEGNO caracteriza a estabilidade do sistema como um todo.

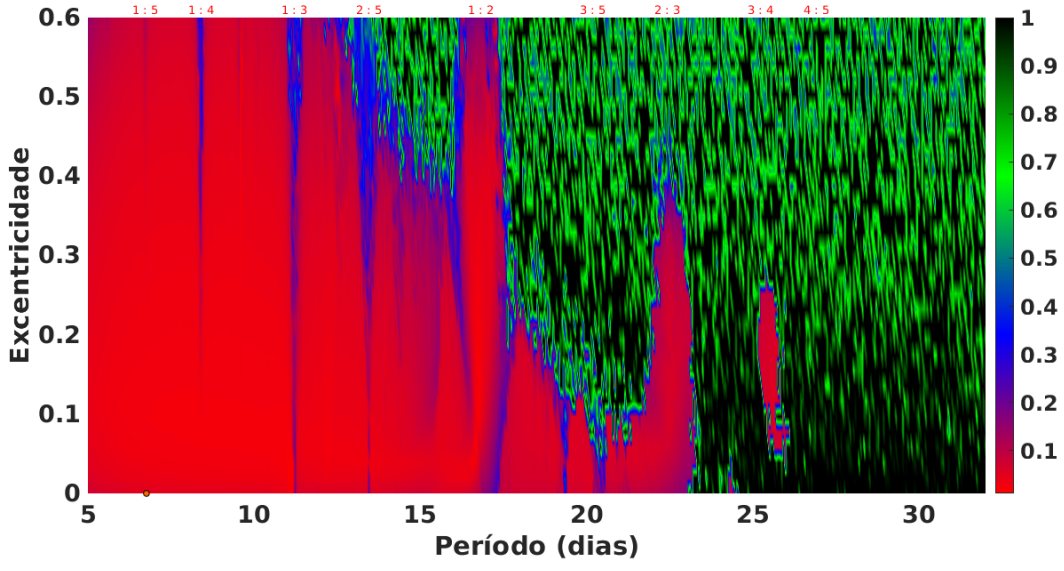

Figura 4.12: Mapa dinâmico da máxima variação em excentricidade no modelo de três planetas. As condições iniciais da grade e o índice de caos correspondem a um Kepler-46d fictício com massa de 15 $M_{\oplus}$. O período é em dias.

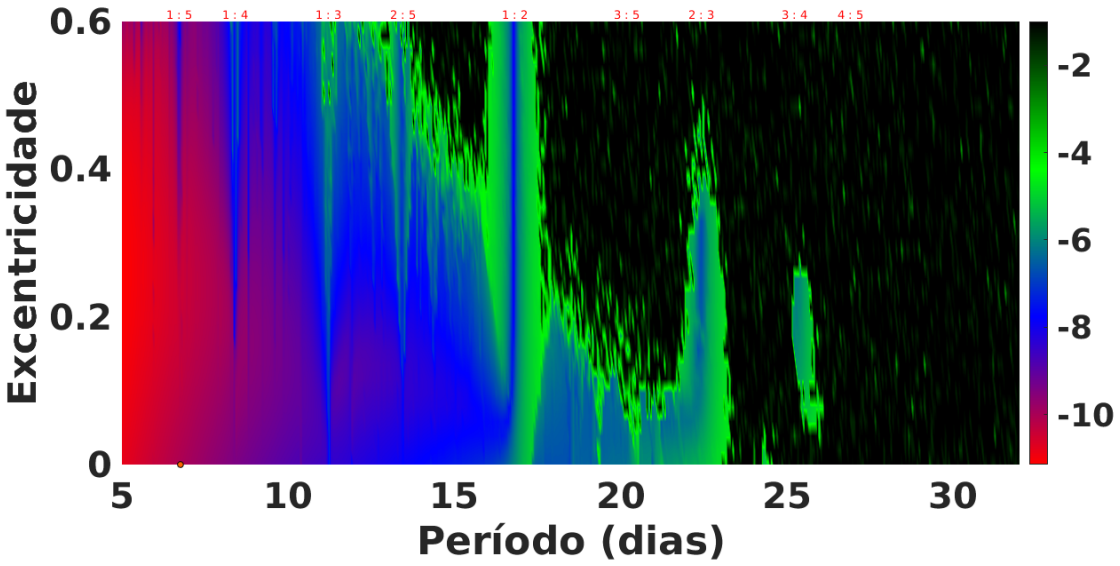

Figura 4.13: Mapa dinâmico da variância em semieixo maior no modelo de três planetas. As condições iniciais da grade e o índice de caos correspondem a um Kepler-46d fictício com massa de 15 $M_{\oplus}$. O período é em dias.

Sistema Solar. Várias ressonâncias de movimentos médios entre os planetas Kepler-46b e Kepler-46d podem ser claramente visualizadas, sendo que aquelas localizadas em $P < 16$ dias (4:1, 3:1, 5:2) se apresentam como mais caóticas, enquanto que aquelas localizadas em $P > 16$ dias (2:1, 3:2, 4:3) se apresentam como sendo mais regulares. A ressonância 2:1 em $P \sim 17$ dias aparece limitada por duas faixas caóticas, correspondentes ao movimento em volta da separatriz. Nas Figuras 4.9 e 4.10, o formato da ressonância 2:1 para valores de $e > 0,55$ é semelhante a estrutura observada na ressonância 2:1 no cinturão de asteroides (ver por exemplo NESVORNÝ e FERRAZ-MELLO, 1997; FERRAZ-MELLO *et al.*, 1998), e está relacionada à sobreposição de ressonâncias seculares dentro da ressonância de movimentos médios (MORBIDELLI e MOONS, 1993). As ilhas de estabilidade nas ressonâncias 3:2 ($P \sim 22$ dias) e 4:3 ($P \sim 26$ dias) também têm contrapartidas conhecidas no Sistema Solar: os grupos de asteroides Hildas e Thules. A região em volta de $P \sim 7$ dias onde Kepler-46d poderia estar localizado, de acordo com NESVORNÝ *et al.* (2012), se apresenta como sendo bastante regular, mesmo para órbitas com alta excentricidade.

4.3.2 Conclusões

Com base na análise dos mapas dinâmicos podemos concluir o seguinte:

- As possíveis soluções para Kepler-46c que produzem os melhores ajustes dos TTVs se encontram em regiões do espaço de fases onde o movimento é muito regular.
- A órbita sugerida para Kepler-46d também se encontra em uma região de movimento muito regular. Nossa análise, infelizmente, não permite colocar restrições nem em

sua excentricidade, nem em sua massa.

- Pelos mapas de estabilidade do modelo de dois planetas é possível observar que o planeta Kepler-46c pode ter migrado radialmente por diversas regiões que são estáveis, se afastando ou se aproximando de Kepler-46b, mas nunca pode ter se aproximado numa órbita mais próxima de kepler-46b que a definida pela posição de ressonância 2:3, já que neste caso teria entrado numa configuração instável no espaço de fase. Mesmo em alguns pontos que são ressonâncias aparentemente mais instáveis como a 1:2 e a 3:5, o planeta pode ter passado com velocidade suficiente para não ser preso nessas ressonâncias.

- Considerando os planetas Kepler-46b,c em suas órbitas atuais, o espaço de fase do sistema mostra diversas características que têm contrapartes semelhantes no nosso Sistema Solar. Estes resultados contribuem, de certa forma, para validar o nosso código paralelizado. Entre estas características, cabem destacar:

  - A região caótica ao redor da ressonância 2:1 com Kepler-46b, causada pela sobreposição de ressonâncias seculares dentro da ressonância de movimentos médios.
  - As ilhas de estabilidade dentro das ressonâncias 3:2 e 4:3 com Kepler-46b.

- Os resultados são mais ou menos independentes do indicador de caos utilizado, embora a máxima variação em excentricidade pareça ser preferível aos outros indicadores.

Capítulo 5

Conclusões e perspectivas futuras

Nesse trabalho analisamos a possibilidade de aumento de desempenho computacional através de implementações em GPU de algoritmos bastante utilizados na área de dinâmica planetária. Tentamos basicamente três abordagens: a paralelização do algoritmo Ruth de uma simulação isolada, a paralelização do algoritmo `Swifter_Helio` para uma simulação isolada, e a simulação em paralelo de uma grade de condições iniciais com o algoritmo `Helio`. Nossa conclusão é que o algoritmo de Ruth e a grade paralelizada são bem eficientes em comparação com a abordagem sequencial, independentemente do hardware utilizado, mas a paralelização de uma simulação isolada do algoritmo `Helio` apresentou ganho de desempenho apenas em algumas placas GPU.

Outra contribuição importante foi a implementação de indicadores de estabilidade que não estavam implementados na versão original do Swifter. Dessa forma, além de poder simular milhares de condições iniciais em paralelo, também é possível obter mapas de estabilidade do sistema analisado. A aplicação de nosso código a diferentes sistemas planetários extrassolares mostrou-se bastante útil para complementar os estudos sobre determinação de parâmetros orbitais destes sistemas.

Como trabalhos futuros, apresentamos a seguir algumas possibilidades:

- Implementação de outras características que estão presentes no `Swifter` e que não foram impelmentadas nesse código, como por exemplo, a inclusão das perturbações devidas ao achatamento do corpo central e a implementação de partículas de teste que possam ter encontros próximos com os corpos massivos (algoritmo `RMVS`).

- Implementação dos outros algoritmos presentes no `Swifter`, como o `Symba`.

- Aplicação do código em outros problemas de interesse em dinâmica do sistema solar e de exoplanetas.

- Melhorar o desempenho da paralelização do algoritmo `Helio` para a simulação do problema de N corpos quando o número de corpos é consideravelmente alto.

- Implementar no algoritmo correções relativísticas.
- Implementar no algoritmo efeitos de maré.

Referências Bibliográficas

AARSETH, S. J. (2003) "Gravitational N-body Simulations: Tools and Algorithms". Cambridge University Press, Cambridge, MA, ISBN 978-0-521-12153-8

AGOL, E.; STEFFEN, J.; SARI, R.; CLARKSON, W. (2005) "On detecting terrestrial planets with timing of giant planet transits". *Mon. Not. Roy. Astron. Soc.* 359, 567-579, https://www.doi.org/10.1111/j.1365-2966.2005.08922.x

ALMENARA, J. M.; DÍAZ, R. F.; HÉBRARD, G.; MARDLING, R.; DAMIANI, C.; et. al. (2018) "SOPHIE velocimetry of Kepler transit candidates. XVIII. Radial velocity confirmation, absolute masses and radii, and origin of the Kepler-419 multiplanetary system". *Astron. Astrophys.* 615, id.A90, https://www.doi.org/10.1051/0004-6361/201732500

ANGLADA-ESCUDÉ, G.; AMADO, P. J.; BARNES, J.; et al. (2016) "A terrestrial planet candidate in a temperate orbit around Proxima Centauri". *Nature* 536, 437-440, https://www.doi.org/10.1038/nature19106

ARNOLD, V. I. (1992) "Ordinary Differential Equations". Springer-Verlag, Berlin, Heidelberg, ISBN 978-3-540-34563-3

BALLARD, S.; FABRYCKY, D.; FRESSIN, F.; CHARBONNEAU, D.; DESERT, J.-M.; et al. (2011) "The Kepler-19 System: A Transiting 2.2 R⊕ Planet and a Second Planet Detected via Transit Timing Variations". *Astrophys. J.* 743, id.200, https://www.doi.org/10.1088/0004-637X/743/2/200

BARNES, J.; HUT, P. (1986) "A hierarchical $O(N \log N)$ force-calculation algorithm". *Nature* 324, 446–449, https://www.doi.org/10.1038/324446a0

BENETTIN, G.; GALGANI, L.; GIORGILLI, A.; STRELCYN, J.-M. (1980) "Lyapunov Characteristic Exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. Part 1: Theory". *Meccanica* 15, 9-20, https://www.doi.org/10.1007/BF02128236

BREITER, S.; MELENDO, B.; BARTCZAK, P.; WYTRZYSZCZAK, I. (2005) "Synchronous motion in the Kinoshita problem. Application to satellites and binary

asteroids". *Astron. Astrophys.* 437, 753–764, https://www.doi.org/10.1051/0004-6361:20053031

BROUWER, D.; CLEMENCE, G. M. (1961) "Methods of Celestial Mechanics". Academic Press, New York, ISBN 978-1-483-20075-0

BUTCHER, J. C. (2008) "Numerical Methods for Ordinary Differential Equations". John Wiley & Sons, New York, ISBN 978-0-470-72335-7

CAMERON, A. C. (2016) "Extrasolar Planetary Transits". Em: *Methods of Detecting Exoplanets*, V. Bozza, L. Mancini, A. Sozzetti (eds.). Astrophys. Space Sci. Lib. 428, Springer, Cham, pp. 89-131, https://www.doi.org/10.1007/978-3-319-27458-42

CANDY, J.; ROZMUS, W. (1991) "A Symplectic Integration Algorithm for Separable Hamiltonian Functions". *J. Comput. Phys.* 92, 230-, https://www.doi.org/10.1016/0021-9991(91)90299-Z

CHAMBERS, J. E. (1991) "A hybrid symplectic integrator that permits close encounters between massive bodies". *Mon. Not. Roy. Astron. Soc.* 304, 793-799, https://www.doi.org/10.1046/j.1365-8711.1999.02379.x

CHAMBERS, J. E.; MURISON, M. A. (2000) "Pseudo-High-Order Symplectic Integrators". *Astron. J.* 119, 425-, https://www.doi.org/10.1086/301161

CHAMBERS, J. E.; QUINTANA, E. V.; DUNCAN, M. J.; LISSAUER, J. J. (2002) "Symplectic Integrator Algorithms for Modeling Planetary Accretion in Binary Star Systems". *Astron. J.* 123, 2884-2894, https://www.doi.org/10.1086/340074

CHAMBERS, J. E. (2010) "N-Body Integrators for Planets in Binary Star Systems". Em: *Planets in Binary Star Systems*, N. Haghighipour (ed.). Astrophys. Space Sci. Lib. 366. Springer, Dordrecht, pp. 239-263, https://www.doi.org/10.1007/978-90-481-8687-79

CHAMBERS, J. E. (2012) "Mercury: A software package for orbital dynamics". *Astrophysics Source Code Library*, rec. 1201.008, http://ascl.net/1201.008

CHARBONNEAU, D.; BROWN, T. M.; LATHAM, D. W.; MAYOR, M. (2000) "Detection of Planetary Transits Across a Sun-like Star". *Astrophys. J.* 529, L45-L48, https://www.doi.org/10.1086/312457

CHIRIKOV, B. V. (1979) "A universal instability of many-dimensional oscillator systems". *Phys. Rep.* 57, 263-379, https://www.doi.org/10.1016/0370-1573(79)90023-1

CINCOTTA, P. M.; SIMÓ, C. (2000) "Simple tools to study global dynamics in non-axisymmetric galactic potentials - I". *Astron. Astrophys.* Suppl. Ser. 147, 205-228, https://www.doi.org/10.1051/aas:2000108

CINCOTTA, P. M.; GIORDANO, C. M.; SIMÓ, C. (2003) "Phase space structure of multi-dimensional systems by means of the mean exponential growth factor of nearby orbits". *Physica D: Nonlin. Phenom.* 182, 151-178, https://www.doi.org/10.1016/S0167-2789(03)00103-9

CONTOPOULUS, G.; BARBANIS, B. (1989) "Lyapunov characteristic numbers and the structure of phase-space". *Astron. Astrophys.* 222, 329-343

CONTOPOULOS, G.; VOGLIS, N. (1996) "Spectra of stretching numbers and helicity angles in dynamical systems". *Cel. Mech. Dyn. Astr.* 64, 1-20, https://www.doi.org/10.1007/BF00051601

"CUDA-C Programing Guide, ver. 3.2" (2010) http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf

DANBY, J. M. A. (1988) "Fundamentals of Celestial Mechanics, 2nd. revised edition". William-Bell Inc, Richmond, VA, ISBN 978-0-943-39620-0

DAWSON, R. I.; JOHNSON, J. A.; MORTON, T. D.; CREPP, J. R.; FABRYCKY, D. C.; MURRAY-CLAY, R. A.; HOWARD, A. W. (2012) "The Photoeccentric Effect and Proto-hot Jupiters. II. KOI-1474.01, a Candidate Eccentric Planet Perturbed by an Unseen Companion". *Astrophys. J.* 761, id.163, https://www.doi.org/10.1088/0004-637X/761/2/163

DAWSON, R. I.; JOHNSON, J. A.; FABRYCKY, D. C.; FOREMAN-MACKEY, D.; MURRAY-CLAY, R. A.; et al. (2014) "Large Eccentricity, Low Mutual Inclination: The Three-dimensional Architecture of a Hierarchical System of Giant Planets". *Astrophys. J.* 791, id.89, https://www.doi.org/10.1088/0004-637X/791/2/89

DU, P.; WEBER, R.; LUSZCZEK, P.; TOMOV, S.; PETERSON, G.; DONGARRA, J. (2012) "From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming". *Parallel Comp.* 38, 391–407, https://www.doi.org/10.1016/j.parco.2011.10.002

DUNCAN, M. J.; LEVISON, H. F.; LEE, M. H. (1998) "A Multiple Time Step Symplectic Algorithm for Integrating Close Encounters", *Astron. J.* 116, 2067-2077, https://www.doi.org/10.1086/300541

EASTWOOD, J. W. (1975) "Optimal particle-mesh algorithms". *J. Comp. Phys.* 18, 1-20, https://doi.org/10.1016/0021-9991(75)90099-6

EVERHART, E. (1985) "An efficient integrator that uses Gauss-Radau spacings". Em: *IAU Colloquium 83, Dynamics of Comets: Their Origin and Evolution*, A. Carusi, G. B. Valsecchi (eds.). Astrophys. Space Sci. Lib. 115. Reidel, Dordrecht, p. 185, https://www.doi.org/10.1007/978-94-009-5400-7_17

FEROZ, F.; HOBSON, M. P.; BRIDGES, M. (2009) "MULTINEST: an efficient and robust Bayesian inference tool for cosmology and particle physics". *Mon. Not. Roy. Astron. Soc.* 398, 1601-1614, https://www.doi.org/10.1111/j.1365-2966.2009.14548.x

FEROZ, F.; HOBSON, M. P.; CAMERON, E.; PETTITT, A. N. (2019) "Importance Nested Sampling and the MultiNest Algorithm". *Open J. Astrophys.* 2, id.10, https://www.doi.org/10.21105/astro.1306.2144

FERNÁNDEZ, J. A.; GALLARDO, T.; BRUNINI, A. (2002) "Are There Many Inactive Jupiter Family Comets Among the Near-Earth Asteroid Population?" *Icarus* 159, 358-368, https://www.doi.org/10.1006/icar.2002.6903

FERRAZ-MELLO, S.; KLAFKE, J. C.; MICHTCHENKO, T. A.; NESVORNÝ, D. (1996) "Chaotic Transitions in Resonant Asteroidal Dynamics". *Cel. Mech. Dyn. Astr.* 64, 93-105, https://www.doi.org/10.1007/BF00051608

FERRAZ-MELLO, S.; MICHTCHENKO, T. A.; NESVORNÝ, D.; ROIG, F.; SIMULA, A. (1998) "The depletion of the Hecuba gap vs the long-lasting Hilda group". *Planet. Space Sci.* 46, 1425-1432, https://www.doi.org/10.1016/S0032-0633(98)00023-3

FERRAZ-MELLO, S.; MICHTCHENKO, T. A.; BEAUGÉ, C. (2005) "The Orbits of the Extrasolar Planets HD 82943c and b". *Astrophys. J.* 621, 473-481, https://www.doi.org/10.1086/427276

FISCHER, D. A.; ANGLADA-ESCUDÉ, G.; ARRIAGADA, P.; et al. (2016) "State of the Field: Extreme Precision Radial Velocities". *Publ. Astr. Soc. Pacific* 128, 066001, https://www.doi.org/10.1088/1538-3873/128/964/066001

FOREST, E.; RUTH, R. D. (1990) "Fourth-order symplectic integration". *Physica D* 43, 105-, https://www.doi.org/10.1016/0167-2789(90)90019-L

FOSTER, I. (1995) "Designing and Building Parallel Programs". Addison-Wesley, Reading, MA, ISBN 978-0-201-57594-1, https://www.mcs.anl.gov/~itf/dbpp/

FOUCHARD, M.; LEGA, E.; FROESCHLÉ, CH.; FROESCHLÉ, CL. (2002) "On the Relationship Between Fast Lyapunov Indicator and Periodic Orbits for Continuous Flows". *Cel. Mech. Dyn. Astr.* 83, 205-, https://doi.org/10.1023/A:1020199201070

FROESCHLÉ, CL.; GONCZI, R.; LEGA, E. (1997) "The fast Lyapunov indicator: a simple tool to detect weak chaos. Application to the structure of the main asteroidal belt". *Planet. Space Sci.* 45, 881-886, https://www.doi.org/10.1016/S0032-0633(97)00058-5

GILLON, M.; TRIAUD, A. H. M. J.; DEMORY, B.-O.; et al. (2017) "Seven temperate terrestrial planets around the nearby ultracool dwarf star TRAPPIST-1". *Nature* 542, 456-460, https://www.doi.org/10.1038/nature21360

GLADMAN, B. J.; MIGLIORINI, F.; MORBIDELLI, A.; et al. (1997) "Dynamical lifetimes of objects injected into asteroid belt resonances". *Science* 277, 197-201, https://www.doi.org/10.1126/science.277.5323.197

GOLDSTEIN, H. (1980) "Classical Mechanics, 2nd. edition". Addison-Wesley, Reading, MA, ISBN 978-0-201-02918-5.

GOŹDZIEWSKI, K.; BOIS, E.; MACIEJEWSKI, A. J.; KISELEVA-EGGLETON, L. (2001) "Global dynamics of planetary systems with the MEGNO criterion". *Astron. Astrophys.* 378, 569-586, https://www.doi.org/10.1051/0004-6361:20011189

GRIMM, S. L.; STADEL, J. G. (2014) "The GENGA Code: Gravitational Encounters in N-Body Simulations with GPU Acceleration". *Astrophys. J.* 796, 23-38, https://www.doi.org/10.1088/0004-637x/796/1/23

HAGHIGHIPOUR, N.; CAPEN, S.; HINSE, T. C. (2013) "Detection of Earth-mass and super-Earth Trojan planets using transit timing variation method". *Cel. Mech. Dyn. Astr.* 117, 75-89, https://www.doi.org/10.1007/s10569-013-9510-y

HAIRER, E.; NØRSETT, S. P.; WANNER, G. (1993) "Solving ordinary differential equations I: Nonstiff problems". Springer-Verlag, Berlin, Heidelberg, ISBN 978-3-540-56670-0

HAIRER, E.; LUBICH, CH.; WANNER, G. (2006) "Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations, 2nd. edition". Springer-Verlag, Berlin, Heidelberg, ISBN 978-3-540-30663-4

HAKEN, H. (1983) "At least one Lyapunov exponent vanishes if the trajectory of an attractor does not contain a fixed point". *Phys. Lett. A* 94, 71-72, https://www.doi.org/10.1016/0375-9601(83)90209-8

HALL, B. C. (2015) "Lie Groups, Lie Algebras and Representations, Graduate Texts in Mathematics 222, 2nd. edition". Springer International Publishing, ISBN 978-3-319-13466-6

HELLMICH, S. (2017) "GPU accelerated n-Body integrators for long-term simulations of planetary systems". *Doctoral Thesis*. Universität Münster. https://repositorium.uni-muenster.de/document/miami/9737b9eb-e3af-4733-b396-31d7d0b5c44c/diss_hellmich.pdf

HELLMICH, S.; MOTTOLA, S.; HAHN, G.; KÜHRT, E.; DE NIEM, D. (2019) "Influence of the Yarkovsky force on Jupiter Trojan asteroids". *Astron. Astrophys.* 630, id.A148, https://www.doi.org/10.1051/0004-6361/201834715

HOLCZER, T.; MAZEH, T.; NACHMANI, G.; JONTOF-HUTTER, D.; FORD, E. B.; FABRYCKY, D.; RAGOZZINE, D.; KANE, M.; STEFFEN, J. H. (2016) "Transit Timing Observations from Kepler. IX. Catalog of the Full Long-cadence Data Set". *Astrophys. J. Suppl. Ser.* 225, id.9, https://www.doi.org/10.3847/0067-0049/225/1/9

HOLMAN, M. J.; MURRAY, N. W. (2005) "The Use of Transit Timing to Detect Terrestrial-Mass Extrasolar Planets". *Science* 307, 1288-1291, https://www.doi.org/10.1126/science.1107822

KINOSHITA, H.; YOSHIDA, H.; NAKAI, H. (1991) "Symplectic integrators and their application to dynamical astronomy". *Cel. Mech. Dyn. Astr.* 50, 59-71.

LANCZOS, C. (1986) "The variational principles of mechanics, 4th. revised edition". Dover Publications, Mineola, NY, ISBN 978-0-48-665067-8

LEGA, E.; MORBIDELLI, A.; NESVORNÝ, D. (2013) "Early dynamical instabilities in the giant planet systems". *Mon. Not. Roy. Astron. Soc.* 431, 3494-3500, https://www.doi.org/10.1093/mnras/stt431

LEVISON, H. F.; DUNCAN, M. J. (1994) "The Long-Term Dynamical Behavior of Short-Period Comets". *Icarus* 108, 18-36, https://www.doi.org/10.1006/icar.1994.1039

LICHTENBERG, A. J.; LIEBERMAN, M. A. (1992) "Regular and Chaotic Dynamics, Applied Mathematical Sciences 38". Springer-Verlag, New York, ISBN 978-0-3879-7745-4

LISSAUER, J. J.; FABRYCKY, D. C.; FORD, E. B.; et al. (2011) "A closely packed system of low-mass, low-density planets transiting Kepler-11". *Nature* 470, 53-58, https://www.doi.org/10.1038/nature09760

LITHWICK, Y.; XIE, J.; WU, Y. (2012) "Extracting Planet Mass and Eccentricity from TTV Data". *Astrophys. J.* 761, id.122, https://www.doi.org/10.1088/0004-637X/761/2/122

LOVIS, C.; FISCHER, D. (2010) "Radial Velocity Techniques for Exoplanets". Em: *Exoplanets*, S. Seager (ed.). University of Arizona Press, Tucson, AZ, pp. 27-53.

MAFFIONE, N. P.; DARRIBA, L. A.; CINCOTTA, P. M.; GIORDANO, C. M. (2011) "A comparison of different indicators of chaos based on the deviation vectors: application to symplectic mappings". *Cel. Mech. Dyn. Astr.* 111, 285-307, https://www.doi.org/10.1007/s10569-011-9373-z

MAFFIONE, N. P. (2012) "Comparación de Indicadores de caos en sistemas hamiltonianos". *Tesis de Doctorado*, Universidad Nacional de La Plata, http://sedici.unlp.edu.ar/handle/10915/23834

MAGNUS, W. (1954) "On the exponential solution of differential equations for a linear operator". *Comm. Pure Appl. Math.* VII (4), 649-673, https://www.doi.org/10.1002/cpa.3160070404

MANCINI, L.; LILLO-BOX, J.; SOUTHWORTH, J.; BORSATO, L.; GANDOLFI, D.; CICERI, S.; BARRADO, D.; BRAHM, R.; HENNING, TH. (2016) "Kepler-539: A young extrasolar system with two giant planets on wide orbits and in gravitational interaction". *Astron. Astrophys.* 590, id.A112, https://www.doi.org/10.1051/0004-6361/201526357

MARCHAL, C. (1988) "Round Table Discussion on Chaotic Motions". Em: *The Few Body Problem*, M. J. Valtonen (ed.), Astrophys. Space Sci. Lib. 140. Kluwer Academic Publishers, Dordrecht, pp. 91-.

METZLER, R.; JEON, J. H.; CHERSTVY, A. G.; BARKAI, E. (2014) "Anomalous diffusion models and their properties: non-stationarity, non-ergodicity, and ageing at the centenary of single particle tracking". *Phys. Chem. Chem. Phys.* 16, 24128-24164, https://www.doi.org/10.1039/C4CP03465A

MICHEL, P.; RICHARDSON, D. C.; DURDA, D. D.; JUTZI, M.; ASPHAUG, E. (2015) "Collisional Formation and Modeling of Asteroid Families". Em: *Asteroids IV*, P. Michel, F. E. DeMeo, W. F. Bottke (eds.), University of Arizona Press, Tucson, AZ, pp. 341-354, https://www.doi.org/10.2458/azu_uapress_9780816532131-ch018

MIKKOLA, S.; INNANEN, K. (1999) "Symplectic Tangent map for Planetary Motions". *Cel. Mech. Dyn. Astr.* 74, 59-, https://www.doi.org/10.1023/A:1008312912468

MIKKOLA, S.; WIEGERT, P. (2002) "Regularizing Time Transformations in Symplectic and Composite Integration". *Cel. Mech. Dyn. Astr.* 82, 375-390, https://www.doi.org/10.1023/A:1015248903174

MIRALDA-ESCUDÉ, J. (2002) "Orbital Perturbations of Transiting Planets: A Possible Method to Measure Stellar Quadrupoles and to Detect Earth-Mass Planets". *Astrophys. J.* 564, 1019-1023, https://www.doi.org/10.1086/324279

MONAGHAN, J. J. (1992) "Smoothed particle hydrodynamics". *Annual Rev. Astron. Astrophys.* 30, 543-574, https://www.doi.org/10.1146/annurev.aa.30.090192.002551

MORBIDELLI, A.; MOONS, M. (1993) "Secular Resonances in Mean Motion Commensurabilities: The 2/1 and 3/2 Cases". *Icarus* 102, 316-332, https://www.doi.org/10.1006/icar.1993.1052

MORBIDELLI, A.; MOONS, M. (1995) "Numerical evidence on the chaotic nature of the 3/1 mean motion commensurability". *Icarus* 115, 60-65, https://www.doi.org/10.1006/icar.1995.1078

NESVORNÝ, D.; FERRAZ-MELLO, S. (1997) "On the Asteroidal Population of the First-Order Jovian Resonances". *Icarus* 130, 247-258, https://www.doi.org/10.1006/icar.1997.5807

NESVORNÝ, D.; KIPPING, D. M.; BUCHHAVE, L. A.; BAKOS, G. Á.; HARTMAN, J.; SCHMITT, A. R. (2012) "The Detection and Characterization of a Nontransiting Planet by Transit Timing Variations". *Science* 336, 1133, https://www.doi.org/10.1126/science.1221141

NESVORNÝ, D.; KIPPING, D.; TERRELL, D.; HARTMAN, J.; BAKOS, G. Á.; BUCHHAVE, L. A. (2013) "KOI-142, The King of Transit Variations, is a Pair of Planets near the 2:1 Resonance". *Astrophys. J.* 777, id.3, https://www.doi.org/10.1088/0004-637X/777/1/3

NESVORNÝ, D.; KIPPING, D.; TERRELL, D.; FEROZ, F. (2014) "Photo-dynamical Analysis of Three Kepler Objects of Interest with Significant Transit Timing Variations". *Astrophys. J.* 790, id.31, https://www.doi.org/10.1088/0004-637X/790/1/31

OSELDETCH, V. I. (1968) "A multiplicative ergodic theorem. Characteristic Lyapunov exponents of dynamical systems". *Trans. Moscow Math. Soc.* 19, 197-231.

PARS, L. A. (1979) "A Treatise on Analytical Dynamics". Ox Bow Press, Woodbridge, CT, ISBN 978-0-9180-2407-7

PETIT, A. C.; LASKAR, J.; BOUÉ, G.; GASTINEAU, M. (2019) "High-order regularised symplectic integrator for collisional planetary systems". *Astron. Astrophys.* 628, A32, https://www.doi.org/10.1051/0004-6361/201935786

PRESS, W. H.; FLANNERY, B. P.; TEUKOLSKY, S. A.; VETTERLING, W. T. (1992) "Numerical Recipes in C: The Art of Scientific Computing, 2nd. edition". Cambridge Univ. Press, Cambridge, MA, ISBN 978-0-521-43108-8, http://numerical.recipes/

REIN, H.; LIU, S. F. (2012) "REBOUND: an open-source multi-purpose N-body code for collisional dynamics". *Astron. Astrophys.* 537, A128, https://www.doi.org/10.1051/0004-6361/201118085

REIN, H.; TAMAYO, D. (2015) "WHFAST: a fast and unbiased implementation of a symplectic Wisdom-Holman integrator for long-term gravitational simulations". *Mon. Not. Roy. Astron. Soc.* 452, 376-388, https://www.doi.org/10.1093/mnras/stv1257

ROSSMANN, W. (2002) "Lie Groups: An Introduction Through Linear Groups". Oxford University Press, New York, ISBN 978-0-19-859683-7

ROWE, J. F.; COUGHLIN, J. L.; ANTOCI, V.; BARCLAY, T.; BATALHA, N. M.; et al. (2015) "Planetary Candidates Observed by Kepler. V. Planet Sample from Q1-Q12 (36 Months)". *Astrophys. J. Suppl. Ser.* 217, id.16, https://www.doi.org/10.1088/0067-0049/217/1/16

RUTH, R. D. (1983) "A Canonical Integration Technique". *IEEE Trans. Nucl. Sci.* NS-30 (4), 2669, https://www.doi.org/10.1109/TNS.1983.4332919

SAAD-OLIVERA, X.; NESVORNÝ, D.; KIPPING, D. M.; ROIG, F. (2017) "Masses of Kepler-46b,c from Transit Timing Variations". *Astron. J.* 153, id.198, https://www.doi.org/10.3847/1538-3881/aa64e0

SAAD-OLIVERA, X. (2019) "Detecção de planetas nos dados do Kepler". *Tese de Doutorado*. Observatório Nacional/MCTIC, http://www.on.br/conteudo/dppg_e_iniciacao/dppg/ferramenta_teses/teses/ASTRONOMIA/[437_25-56_C]thesis_saad_olivera_2019__final.pdf

SAAD-OLIVERA, X.; COSTA DE SOUZA, A.; ROIG, F.; NESVORNÝ, D. (2019) "Masses of the Kepler-419 planets from transit timing variations analysis". *Mon. Not. Roy. Astron. Soc.* 482, 4965-4971, https://www.doi.org/10.1093/mnras/sty2990

SAAD-OLIVERA, X.; MARTINEZ, C. F.; COSTA DE SOUZA, A.; ROIG, F.; NESVORNÝ, D. (2020) "A super-Earth and a mini-Neptune around Kepler-59". *Mon. Not. Roy. Astron. Soc.* 491, 5238-5247, https://www.doi.org/10.1093/mnras/stz3369

SAHA, P.; TREMAINE, S. (1992) "Symplectic Integrators for Solar System Dynamics". *Astron. J.* 104, 1633-, https://www.doi.org/10.1086/116347

SAHA, P.; TREMAINE, S. (1994) "Long-Term Planetary Integration with Individual Time Steps". *Astron. J.* 108, 1962-, https://www.doi.org/10.1086/117210

SÁNDOR, Z.; ÉRDI, B.; SZÉLL, A.; FUNK, B. (2004) "The Relative Lyapunov Indicator: An Efficient Method of Chaos Detection". *Cel. Mech. Dyn. Astr.* 90, 127-138, https://www.doi.org/10.1007/s10569-004-8129-4

SEAGER, S.; MALLÉN-ORNELAS, G. (2003) "A Unique Solution of Planet and Star Parameters from an Extrasolar Planet Transit Light Curve". *Astrophys. J.* 585, 1038-1055, https://www.doi.org/10.1086/346105

SKOKOS, CH. (2001) "Alignment indices: a new, simple method for determining the ordered or chaotic nature of orbits". *J. Phys. A: Math. Gen.* 34, 10029-, https://www.doi.org/10.1088/0305-4470/34/47/309

SKOKOS, CH.; BOUNTIS, T. C.; ANTONOPOULOS, CH. (2007) "Geometrical properties of local dynamics in Hamiltonian systems: The Generalized Alignment Index (GALI) method". *Physica D: Nonlin. Phenom.* 231, 30-54, https://www.doi.org/10.1016/j.physd.2007.04.004

ŚLEBODZIŃSKI, W. (1931). "Sur les équations de Hamilton". *Bull. Acad. Roy. Belg.* 17 (5), 864-870.

STEFFEN, J. H.; FABRYCKY, D. C.; FORD, E. B.; CARTER, J. A.; DÉSERT, J.-M.; et al. (2012) "Transit timing observations from Kepler - III. Confirmation of four multiple planet systems by a Fourier-domain study of anticorrelated transit timing variations". *Mon. Not. Roy. Astron. Soc.* 421, 2342-2354, https://www.doi.org/10.1111/j.1365-2966.2012.20467.x

STEFFEN, J. H.; FABRYCKY, D. C.; AGOL, E.; FORD, E. B.; MOREHEAD, R. C.; et al. (2013) "Transit timing observations from Kepler - VII. Confirmation of

27 planets in 13 multiplanet systems via transit timing variations and orbital stability". Mon. Not. Roy. Astron. Soc. 428, 1077-1087, https://www.doi.org/10.1093/mnras/sts090

STERNBERG, S. (2009) "Lie Algebras". Orange Grove Books, Gainesville, FL, ISBN 978-1-61-610052-0

STOER, J.; BULIRSCH, R. (2002) "Introduction to Numerical Analysis, 3rd. edition". New York: Springer-Verlag, New York, ISBN 978-0-387-95452-3.

STUMPFF, P. (1988) "The General Kepler Equation and its Solutions". Em: *Long Term Evolution of Planetary Systems*, R. Dvorak, J. Henrard (eds.). Springer, Dordrecht, pp. 211-222, https://www.doi.org/10.1007/978-94-009-2285-318

TEYSSIER, R. (2015) "Grid-Based Hydrodynamics in Astrophysical Fluid Flows". *Annual Rev. Astron. Astrophys.* 53, 325-364, https://doi.org/10.1146/annurev-astro-082214-122309

THE EXTRASOLAR PLANETS ENCYCLOPAEDIA(2020), http://exoplanet.eu/

VOGLIS, N.; CONTOPOULOS, G.; EFTHYMIOPOULOS, C. (1999) "Detection of Ordered and Chaotic Motion Using the Dynamical Spectra". *Cel. Mech. Dyn. Astr.* 73, 211-220, https://www.doi.org/10.1023/A:1008307332442

VOKROUHLICKÝ, D.; BOTTKE, W. F.; CHESLEY, S. R.; SCHEERES, D. J.; STATLER, T. S. (2015) "The Yarkovsky and YORP Effects". Em: *Asteroids IV*, P. Michel, F. E. DeMeo, W. F. Bottke (eds.), University of Arizona Press, Tucson, AZ, pp. 509-531, https://www.doi.org/10.2458/azu_uapress_9780816532131-ch027

WALSH, K. J.; MORBIDELLI, A.; RAYMOND, S. N.; O'BRIEN, D. P.; MANDELL, A. M. (2011) "A low mass for Mars from Jupiter's early gas-driven migration". *Nature* 475, 206-209, https://www.doi.org/10.1038/nature10201

WALTERS, P. (1982) "An Introduction to Ergodic Theory". Springer-Verlag, New York, ISBN 0-387-95152-0

WILLMORE, T. J. (1960) "The definition of Lie derivative". *Proc. Edinburgh Math. Soc.* 12 (1), 27-29, https://www.doi.org/10.1017/S0013091500025013

WINTNER, A. (2014) "The Analytical Foundations of Celestial Mechanics, reprint edition". Dover Publications, Mineola, NY, ISBN 978-0-4867-8060-3

WISDOM, J. (1982) "The origin of the Kirkwood gaps - A mapping for asteroidal motion near the 3/1 commensurability". *Astron. J.* 87, 577-593, https://www.doi.org/10.1086/113132

WISDOM, J. (1983) "Chaotic behavior and the origin of the 3/1 Kirkwood gap". *Icarus* 56, 51-74, https://www.doi.org/10.1016/0019-1035(83)90127-6

WISDOM, J.; HOLMAN, M. (1991) "Symplectic maps for the N-body problem". *Astron. J.* 102, 1528-1538, https://www.doi.org/10.1086/115978

WOLF, A.; SWIFT, J. B.; SWINNEY, H. L.; VASTANO, J. A. (1985) "Determining Lyapunov exponents from a time series". *Physica D: Nonlin. Phenom.* 16, 285-317, https://www.doi.org/10.1016/0167-2789(85)90011-9

XIE, J.-W. (2014) "Transit Timing Variation of Near-resonance Planetary Pairs. II. Confirmation of 30 Planets in 15 Multiple-planet Systems". *Astrophys. J. Suppl.* 210, id.25, https://www.doi.org/10.1088/0067-0049/210/2/25

YOSHIDA, H. (1990) "Construction of higher order symplectic integrators". *Phys. Lett. A.* 150, 262-, https://www.doi.org/10.1016/0375-9601(90)90092-3