UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CÂMPUS DE GUARATINGUETÁ - SP

PATRÍCIA BUZZATTO SIQUEIRA

Adaptação do integrador Rebound para o estudo de anéis planetários

Guaratinguetá - SP
2016

PATRÍCIA BUZZATTO SIQUEIRA

Adaptação do integrador Rebound para o estudo de anéis planetários

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Física da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Bacharelado em Física.

Orientador: Prof. Dr. Rafael Sfair

Guaratinguetá - SP
2016

Siqueira, Patrícia Buzzatto
S618a Adaptação do integrador Rebound para o estudo de anéis planetários / Patrícia Buzzatto Siqueira - Guaratinguetá, 2016
88 f.:il.
Bibliografia: f. 87-88

Trabalho de Graduação em Licenciatura em Física - Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2016
Orientador: Rafael Sfair

1. Planetas - Órbitas 2. Métodos de simulação 3. Radiação solar 4. Anéis de armazenamento I. Título

CDU 523.4

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

PATRÍCIA BUZZATTO SIQUEIRA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO EM BACHARELADO EM FÍSICA"**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM NOME DO CURSO

Prof. Dr. Marco Aurélio Alvarenga Monteiro
Coordenador

BANCA EXAMINADORA:

Prof. Dr. Rafael Sfair
Orientador/UNESP-FEG

Profª. Drª. Silvia Maria Giuliatti Winter
UNESP-FEG

Prof. Dr. Ernesto Vieira Neto
UNESP-FEG

Dezembro de 2016

DADOS CURRICULARES

PATRÍCIA BUZZATTO SIQUEIRA

NASCIMENTO	30-03-1992 - Cachoeira Paulista / SP
FILIAÇÃO	Fernando Antonio Rosa de Siqueira Alcineia Aparecida Buzzatto
2008-2010	Técnico em Informática Industrial Colégio Técnico Industrial de Guaratinguetá - UNESP
2011-2016	Graduando em bacharelado em Física Faculdade de Engenharia de Guaratinguetá - UNESP

AGRADECIMENTOS

Meus agradecimentos a todos os familiares, amigos, funcionários e professores da FEG-UNESP, que contribuíram para a realização deste trabalho. Em especial, dedico meus agradecimentos:

- A toda a minha família que sempre me estimulou a crescer e é o meu porto seguro;
- Aos meus pais Alcineia e Fernando por todo o carinho e incentivo;
- Ao meu irmão Gustavo por sempre me apoiar, ajudar e por todas as nossas conversas sobre ciência e tecnologia que sempre me ajudaram a amadurecer;
- A todos os meus amigos, mas em especial Larissa e Ana Luiza que mesmo com toda a distância sempre estiveram e estarão ao meu lado, nos momentos bons e ruins;
- Ao Luca por todo o carinho e amor, e por sempre me apoiar, acalmar e me fazer acreditar que sou capaz de realizar todas minhas metas;
- Aos meus antigos colegas de trabalho do DSA/CPTEC-INPE, em especial o Diego, Mario, Denis, Ladylaine, Simone, Nathália e Helder, que ensinaram muito ao longo dos anos, por todo amadurecimento que adquiri e principalmente pela amizade;
- A equipe de FEG - Robótica por todo ensinamento, companheirismo e por me deixarem fazer parte não só de uma equipe, mas também de uma família;
- Ao Prof. Dr. Rafael Sfair, por todo ensinamento, incentivo, confiança, paciência e orientação;
- A banca examinadora pelas sugestões e incentivo;
- Aos meus amigos e colegas de classe que de forma direta ou indiretamente me ajudaram muito em toda a minha graduação.

“Happiness can be found even in the darkest of times,

if one only remembers to turn on the light.”

Alvo Dumbledore

RESUMO

Estudos de dinâmica orbital podem ser abordados através de integradores numéricos, como Rebound, escrito em C, que oferece diversos algoritmos de integração, que facilita a adaptação de condições iniciais e possui velocidade de execução. O estudo de anéis planetários serve como laboratório para o entendimento do processo de formação planetária e a dinâmica de galáxias e o Rebound é uma ótima ferramenta de integração para sistemas de N-corpos que interagem gravitacionalmente, podendo lidar com forças conservativas e não conservativas, também com encontros próximos entre partículas e órbitas de alta excentricidade mantendo os erros sistemáticos abaixo da precisão de máquina. Neste trabalho temos o objetivo de incluir ao pacote de integração do Rebound forças perturbativas em anéis planetários, como a força de radiação solar, achatamento planetário e perturbações dos satélites próximos aos anéis. Para validar os métodos desenvolvidos iremos incluir no pacote efeitos que alteram a órbita das partículas nos anéis μ e ν de Urano, tais como o achatamento do planeta e as forças de radiação solar. Também consideramos os efeitos devido à presença de satélites próximos aos anéis. Por fim, fizemos um estudo extra de um sistema triplo asteroidal 87 Sylvia. Todos os resultados obtidos aqui são corroborados pelas teorias atuais, assim como as equações propostas neste trabalho.

Palavras-chave: Anéis planetários. Simulações numéricas. Força de radiação solar. Achatamento planetário. Integrador de N-corpos. Rebound.

ABSTRACT

The studies about orbital dynamics can be approached through numerical integrators such as Rebound, written in C, that offers several algorithms of integration and facilitates the adaptation of initial conditions and has speed of execution. The study of planetary rings serves as a laboratory for understanding the planetary formation process and the dynamics of galaxies, and Rebound is a great integration tool for gravitationally interacting N-body systems, which can deal with conservative and non-conservative forces, also with close encounters between particles and high eccentricity orbits while keeping systematic errors below machine accuracy. In this work we aim to include in the integration package of the Rebound perturbative forces in planetary rings, such as the solar radiation force, planetary oblateness and disturbances of the satellites near the rings. To validate the developed methods we will include in the package effects that alter the orbit of the particles in the rings μ and ν of Uranus, such as the planetary oblateness and the forces of solar radiation. We also consider the effects due to the presence of satellites near the rings. Finally, we did an extra study of a triple asteroidal system 87 Sylvia. All the results obtained here are corroborated by the current theories, as well as the equations proposed in this work

Keywords: Planetary rings. Numerical simutations. Radiation force. Planetary oblateness. N-bodies integrator. Rebound.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A comunidade científica tem ajudado a impulsionar o avanço contínuo de computadores cada vez mais rápidos e, juntamente com este avanço, pacotes de software mais robustos e sofisticados estão sendo desenvolvidos. Avanços nestas áreas têm expandido a importância da computação na ciência, que proporciona ferramentas de estudo que vão desde uma simples ferramenta de análise numérica até grandes ambientes para realização de experimentos virtuais complexos.

Os *laboratórios virtuais* são ambientes para realizações de experimentos virtuais que permitem a reprodução de fenômenos naturais complexos. Estes tipos de laboratórios fazem parte de uma área da ciência esta cada vez mais em crescimento, criando simulações que permitem a investigação de fenômenos muito difíceis e até mesmo impossíveis de serem reproduzidos em um laboratório real.

O desenvolvimento de pacotes de software, a modelagem de fenômenos naturais, juntamente com a tarefa de produzir e analisar os dados de simulações, caracterizam uma área da ciência extremamente importante e que esta cada vez mais em ascensão.

As simulações astrofísicas têm evoluído em conjunto com o desenvolvimento de sistemas físicos computacionais, tornando-se uma ferramenta indispensável para o entendimento de grandes bases de dados observacionais e para os estudos sobre a formação e evolução de sistemas astrofísicos. A demanda por tais simulações tem crescido de acordo com o aumento da quantidade e da qualidade dos dados observacionais, especialmente com a introdução de arquiteturas multi-core tais como em Graphics Processing Units (GPUs) e Central Processing Units (CPUs), que atuam como aceleradores de desempenho de propósito geral (Ferrari, 2015).

Particularmente, estamos interessados em simulações gravitacionais de N-corpos, as quais requerem uma maneira rápida para calcular as forças e um método de integração suficientemente preciso para evoluir as partículas no tempo, e uma série de metodologias estão sendo desenvolvidas para este tipo de estudo.

Simulações astrofísicas realísticas foram praticamente impossíveis no passado, tanto pela falta de poder computacional suficiente, quanto pela ausência de códigos que implementam diversos domínios astrofísicos. Com a introdução de arquiteturas paralelas, placas dedicadas GRAvit PipE e os GPUs, um grande aumento em performance computacional foi alcançado e

os algoritmos atuais tem melhorado gradualmente em termos de aplicação dinâmica e precisão numérica.

As simulações que empregam múltiplos domínios astrofísicos podem ser divididas em duas vertentes. A primeira vertente é baseada na construção de códigos seguindo uma estrutura monolítica, que consiste em basear-se em um código gravitacional bem fundamentado e gradativamente adicionar à estrutura existente mais fenômenos físicos. A segunda vertente baseia-se na construção de códigos seguindo uma estrutura modular, onde cada domínio astrofísico é implementado de maneira autônoma e independente dos demais (Ferrari, 2015). Neste caso, o código adicional deve ser escrito para cada aplicação em específico, no qual os diferentes domínios necessários são acoplados da maneira desejada. Um exemplo é o pacote Rebound (Rein & Liu, 2012) que é o nosso objeto de estudo deste presente trabalho.

Existe uma dificuldade no contínuo desenvolvimento de códigos monolíticos, devido a tendência na modelagem numérica de cenários realísticos que envolvem interações físicas complexas, pois a maior parte dos códigos computacionais foram desenvolvidos para um problema, e muitas vezes não pode ser utilizável fora de seu domínio de aplicação. A vantagem de utilizar códigos modulares é que cada desenvolvedor pode focar na construção de códigos cada vez mais robustos e eficientes com base na sua própria área. A ideia por trás da modulação é bastante poderosa, pois permite o acoplamento de diferentes códigos especializados sem a necessidade de modificar cada um deles individualmente.

No verão de 2011, Shang-Fei e Hanno Rein começaram a escrever o Rebound durante uma escola de verão. É um pacote de software com um código altamente modular que pode trabalhar com uma variedade de problemas. O Rebound é paralelizável e pode ser executado em clusters usando milhares de núcleos. Isto permite executar simulações extremamente grandes. A primeira versão do Rebound foi projetada para resolver dinâmicas colisionais, como em anéis planetários, mas, além disto, também podendo resolver o problema clássico de N-corpos. Inicialmente era composto apenas com três integradores simpléticos (leap-frog, o integrador de epiciclo simplético (SEI)(Rein & Tremaine 2011) e o mapa de Wisdom-Holman (WH)(Rein & Tamayo 2015). Nesta primeira versão já estavam presentes os módulos de condições de contorno e os algoritmos baseados em árvores. Tendo todos os módulos extremamente paralelizados com MPI e OpenMP.

Em 2015, foi introduzido ao pacote do Rebound mais um módulo de integração, o IAS15 que é um versátil integrador de 15^a ordem (Rein & Spiegel, 2015). É um integrador baseado na quadratura de Gauß-Radau que pode lidar com forças conservativas e não conservativas, com encontros próximos e tem como a principal vantagem um controle de passo de integração

adaptativo. As propriedades adaptativas deste integrador são tão boas, ou ainda melhores, do que os integradores simpléticos tradicionalmente utilizados. Isto ocorre devido ao fato de que, além de erros resultantes do esquema numérico, existem erros associados com a precisão finita de números de ponto flutuante em um computador. Todos os integradores sofrem destes erros. Newcomb (1899) foi o primeiro a estudar sistematicamente a propagação destes tipos de erros. Se cada operação em um algoritmo é imparcial (o resultado real é arredondado para o número de ponto flutuante representável mais próximo), então o erro cresce como uma caminhada aleatória , i.e. $\propto n^{1/2}$, onde n é o número de passos totais de uma integração. As quantidades angulares, como a fase de uma órbita, crescem ainda mais rapidamente,$\propto n^{3/2}$. Esta é conhecida com a lei de Brouwer. O integrador IAS15 é ótimo no sentido de que seu erro de energia segue a lei de Brouwer. Outros integradores não possuem alguma garantia sobre erros associados à precisão nos ponto flutuantes. O integrador IAS15 possui uma alta ordem de precisão que permite o desempenho de simulações de longo prazo com precisão de máquina mantida ao longo de pelo menos 10^9 escalas orbitais utilizando apenas 100 passos de tempo por órbita.

Integradores simpléticos são aqueles que dependem de que os sistemas que estão atuando sejam Hamiltonianos. Estes integradores conservam todos os invariantes de Poincaré, tais como a densidade de fase-espaço, e têm uma quantidade conservada que seja considerada como uma versão ligeiramente perturbada da Hamiltoniana original. Em muitas situações isso se traduz em um limite superior no erro de energia total. Entretanto, existem diversas complicações ao trabalhar com integradores simpléticos, uma das grandes dificuldades é trabalhar com forças não conservativas, forças que não podem ser descritas por um potencial, como a força de radiação solar que depende da velocidade da partícula e não somente de sua posição. Partículas de poeira em anéis planetários estão sujeitas a estes tipos de forças. Quando forças não conservativas são incluídas na equação do movimento, o conceito de integrador simplético, que depende do sistema ser Hamiltoniano, é perdida. O integrador IAS15 por não ser um integrador simplético consegue lidar com estes tipos de forças.

O integrador IAS15 serve como uma ótima ferramenta de integração, pois preserva a simpleticidade de um sistema hamiltoniano melhor que os integradores simpléticos normalmente utilizados e não exige a necessidade, como integradores simpléticos, de que os sistemas sejam hamiltonianos, como em sistemas com forças perturbativas. Este integrador do Rebound promete que o desempenho dos erros não serão afetados por quaisquer fatores. Isto faz com que este integrador aparente ser ideal para o estudo de anéis planetários, cujas partículas de tamanhos micrométricos sofrem a ação de diversas forças.

Além disto, o fato de trabalhar com um integrador no sistema do Rebound é vantajosa, pois

ele é extremamente modular, o que faz com que exista a possibilidade de criar diversos módulos para as forças não conservativas que são aplicadas em anéis. Ao estudar um caso específico da astronomia a modulação permite que os códigos sejam reaproveitados e introduzidos em diversos sistemas, sem a necessidade de reescrever o código e fazendo apenas um acoplamento dos módulos. Um exemplo disto será feito com o estudo do sistema triplo de asteroides na seção 5 onde aplicamos alguns dos módulos desenvolvidos no estudo de anéis planetários.

Portanto, estes foram os grandes estímulos para este trabalho, trabalhar com integrações numéricos em um sistema prático, com linguagem de programação fácil e eficiente, utilizar a modulação que otimiza, organiza e generaliza os códigos possibilitando que sejam trabalhados em diversos sistema e principalmente trabalhar com integradores práticos e precisos que são ideais para o estudo de anéis planetários. O estudo de anéis planetários serve como laboratório para o entendimento do processo de formação planetária e a dinâmica de galáxias e o Rebound demonstrou ser uma ótima ferramenta para este tipo de estudos.

O Rebound não provém nenhum algoritmo para resolver forças não gravitacionais e perturbativas, o que faremos será criar um novo módulo para resolver estas forças perturbativas, de maneira que possa ser aplicada em qualquer sistema. Para isto, é necessário que exista um estudo prévio de como estas forças atuam alterando as órbitas das partículas.

Na primeira seção deste trabalho apresentaremos ao leitor o software, e sua estrutura, explicaremos os módulos físicos disponíveis e proveremos alguns exemplos para demonstrar a praticidade e a precisão do Rebound. A primeira seção será essencial para o desenvolvimento e entendimento das seções seguintes, no qual dividimos cada umas das forças dissipativas em diferentes seções, sempre apresentando a teoria e o algoritmo de uma maneira geral. No último capítulo aplicaremos estas forças para a validação dos códigos desenvolvidos aqui, também faremos uma introdução teórica sobre o sistema que será aplicado nossos algoritmos e então serão apresentado os resultados obtidos.

2 REBOUND

O Rebound é um software que pode integrar o movimento de partículas sob a influência da gravidade, podendo ser estas partículas representadas como estrelas, planetas, satélites, cometas ou partículas de poeira. É uma ótima ferramenta no estudo de anéis planetários, pois pode lidar com forças conservativas e não conservativas, também com encontros próximos entre partículas e órbitas de alta excentricidade mantendo os erros sistemáticos abaixo da precisão de máquina.

O Rebound é escrito em C e oferece diversos algoritmos de integração o que facilita a adaptação de condições iniciais e possui um bom desempenho. Rebound é bastante flexível e pode ser customizado para ser mais acurado e eficiente para resolver diversos problemas da astrofísica. O software possui versões disponíveis em C ou Python, porém todos os códigos computacionais que necessitam bastante processamento são escritos em C, de forma que mesmo que o usuário decida utilizar a versão Python ele sempre terá um processamento bem rápido, pois quando o código em C é compilado é gerado um programa em linguagem de máquina (instruções compreendidas pelo CPU) e é esse programa que é depois executado. Enquanto que o Python é uma linguagem de script cujo código é interpretado em tempo de execução e portanto seu processamento é bem mais lento.

A linguagem de programação que se deseja utilizar sempre irá depender da preferência ou para algumas aplicações específicas, por exemplo: se você deseja integrar apenas algumas partículas em um sistema planetário com um integrador de alta precisão ou com o um integrador simplético então utilize a versão em Python. Mas se você deseja executar uma larga simulação com milhões de partículas, em qualquer integrador, com visualizações no OpenGL ou então utilizar algoritmos de interação entre partículas em árvore então utilize a versão em C.

2.1 Estrutura do Rebound

O código fonte do Rebound esta hospedado na plataforma do github através do link: http://github.com/hannorein/rebound/. O código pode ser obtido através de um simples download do repositório inteiro fornecido com um arquivo do tipo tar ou zip, entretanto a própria plataforma do github fornece um controle de versão, assim se o usuário clonar o projeto ele ficará informado de todas as atualizações e facilitando o processo sem ser necessário efetuar o download de todo o repositório.

Todo o código fonte pode ser encontrado na pasta *src/* e é dividido em módulos, nos quais explicaremos na seção 2.2.

O Rebound também disponibiliza um arquivo chamado “Makefile” que define as regras de compilação, o texto contido dentro deste arquivo é responsável por fazer a ligação (*linking*) dos módulos utilizados, montagem de arquivos de projeto, limpeza de arquivos temporários, execução de comandos e facilita a compilação do código. Para compilar basta digitar o comando “make” na pasta que contém o “Makefile” e o programa em C com a rotina principal e a compilação será executada seguindo os seguintes passos:

1. Define as variáveis do ambiente que controlam as opções do usuário, como otimização do código, visualização em tempo real e paralelização.
2. Define os caminhos simbólicos para especificar os módulos escolhidos no problema do usuário.
3. Executa o “makefile” dentro do diretório *src/* compilando e agrupando os módulos definidos pelo usuário.
4. Um arquivo binário, chamado “rebound” é copiado dentro do diretório do programa principal, no qual poderá ser executado.

O uso de arquivos “makefile“ em projetos é vantajoso pois evitam a compilação de arquivos desnecessários. Por exemplo, se o programa utiliza diversas bibliotecas e é alterada apenas uma, o *make* descobre, fazendo uma comparação das datas de alteração dos arquivos fontes com as dos arquivos anteriormente compilados, qual arquivo foi alterado e compila apenas a biblioteca necessária. Também automatiza tarefas rotineiras como limpeza de vários arquivos criados temporariamente na compilação e pode ser usado como linguagem geral de script, embora seja mais usado para compilação.

Uma documentação do código fonte também é disponibilizada e pode ser gerada através do *doxygen* dentro do diretório *doc/* executando o comando “make doc” em qualquer diretório que contenha um arquivo “makefile”. Não existe nenhuma documentação estática, uma vez que a estrutura do código depende crucialmente dos módulos escolhidos.

2.2 Módulos e parâmetros

O Rebound é escrito em C e extremamente modular, o que significa dizer que o código é divido em partes que realizam tarefas específicas. Em programas estruturados como o Rebound

é sempre preferível dividir as grandes tarefas de computação em tarefas menores e utilizar seus resultados parciais para compor o resultado final desejado. A modularização é extremamente vantajosa, pois facilita na manutenção, ajuda na organização do código e permite o reuso do código, otimizando os projetos e diminuindo a alocação de memória para o processamento.

Na linguagem C cada módulo é chamado de função, sendo que cada função pode receber e/ou fornecer informações a outras funções. Estas informações ou valores podem ser enviados as funções através de duas maneiras: passagem de parâmetros por valor e passagem de parâmetro por referência.

Na chamada de função por parâmetros, o conteúdo de uma variável é copiado para o parâmetro da função. Desta maneira, qualquer modificação dentro da função chamada não altera o valor das variáveis passadas nos argumentos.

Na chamada de função por referência, é enviado o endereço da memória onde o argumento está armazenado. De maneira que a função deve estar preparada para receber o endereço de memória e seus parâmetros serão variáveis do tipo ponteiro. Dentro da função este endereço será utilizado para alterar o valor do argumento, acessando a posição de memória onde ele está.

O Rebound utiliza a chamada de função por referência, desta forma todas as variáveis do sistema do Rebound devem ser definidas através de ponteiros. Esta é uma maneira bastante eficaz para este tipo de problema, pois as maiorias das variáveis são de configuração e não serão alteradas durante a execução e mesmo nas variáveis que ocorrem mudança de estado são alteradas dentro do código fonte, como as variáveis das partículas no sistema, podem ser acessadas em qualquer função do programa, não é necessário que seja a função onde estas variáveis são utilizadas e nem mesmo a função principal, basta acessar o endereço de memória que esta variável está armazenada.

Todas as variáveis do Rebound estão incluídas dentro da estrutura chamada "reb_simulation" que é estrutura principal encapsulando uma simulação do Rebound inteira. Essa estrutura contém todas as variáveis, sinalizadores de status e ponteiros de uma simulação do Rebound. Assim, ao enviar para uma função esta estrutura, garantimos que a função terá acesso à todos os dados da simulação. Para criar uma simulação Rebound é utilizada a função *reb_create_simulation()*. Isso garantirá que todas as variáveis e ponteiros são inicializados corretamente. Por exemplo, para declarar esta estrutura usamos o comando:

```
1 struct reb_simulation* r = reb_create_simulation();
```

Como pode ser observado a variável *r* é do tipo ponteiro da estrutura, portanto é necessário utilizar a sintaxe de flechas $r->$ para definir as variáveis. Todos os significados das variáveis

de uma simulação do Rebound podem ser consultados através da documentação da Interface de Programação de Aplicativos (API) disponibilizados no link:

http://rebound.readthedocs.io/en/latest/c_api.html

ou podem ser acessados através do próprio código fonte, onde o começo do programa é através do arquivo "rebound.h" dentro do diretório *src/*.

Os módulos do Rebound são definidos utilizando os parâmetros da estrutura da simulação que irão criar alguns links simbólicos. Desta forma não é necessário executar nenhum script de configuração para compilar o código. Os link simbólicos são criados pelo "makefile". Quando diferentes módulos são implementados somente o "makefile" será alterando. Esta configuração permite ao usuário trabalhe com diversos projetos ao mesmo tempo usando diferentes módulos.

Para implementar um novo módulo basta criá-lo no diretório do problema e modificar o link no "makefile". O idealizadores do Rebound aconselham a não adicioná-los no diretório *src/* porque eles acreditam que nenhum arquivo neste diretório deve ser alterado e porque desta forma fica mais fácil de sincronizar a pasta com as novas versões do Rebound.

Neste trabalho seguimos os padrões sugeridos pelo Rebound e adicionamos novos módulos para tratar de forças perturbativas em anéis planetários. Os códigos deste módulo serão discutidos na seção 3.2 e 4.2.

2.2.1 Módulos físicos disponíveis

Por ser um código extremamente modular o usuário necessita escolher entre diferentes módulos de gravidade, colisão, condições de contorno e integrador. Ele também permite que seja implementado diferentes módulos complementares com o mínimo de esforço. Os módulos são escolhidos em tempo de execução que são definidos na estrutura de simulação do Rebound.

As seções a seguir listam alguns dos módulos disponíveis.

2.2.1.1 Soluções Gravitacionais

O Rebound possui alguns módulos de (auto)-gravidade. O módulo de gravidade calcula de maneira exata ou aproximada a aceleração experimentada pelas partículas. Para uma partícula com o índice i a aceleração a é dada por:

$$a_i = \sum_{j=0}^{N_{active}-1} \frac{Gm_j}{(r_{ji}^2 + b^2)^{3/2}} \hat{r}_{ji} \tag{1}$$

onde G é a constante gravitacional, m_j é a massa da partícula j e r_{ji} é a distância relativa entre as partículas i e j. O parâmetro de amortecimento gravitacional é dado por b que por padrão é zero, mas este valor pode ser alterado dependendo do problema. A variável N_{active} especifica o número de partículas massivas na simulação, sendo que as partículas com o índice igual ou maior que esta variável são tratados como partículas testes. Por padrão todas as partículas possuem massa e são consideradas na soma da equação (1). Mais informações sobre como definir as propriedades das partículas serão apresentadas na seção 2.2.2.

O módulo padrão de gravidade do Rebound é o chamado *REB_GRAVITY_COMPENSATED* que utilizada dois tipos de somatório, o direto e o compensado. O somatório direto computa diretamente a equação 1. Se existem N_{active} partículas massivas e N partículas no total, a escala de performance é de $O(N.N_{active})$. A soma direta é eficiente apenas com poucas partículas ativas; tipicamente $N_{active} \leq 10^2$.

O somatório compensado (Kahan 1965;Higham 2002; Hairer et al. 2006) busca melhorar a precisão nas adições que envolvem um pequeno e um grande número com pontos flutuantes. Este cenário ocorre em todos os integradores, por exemplo, quando atualizamos a posição:

$$x_{n+1} = x_n + \delta x \tag{2}$$

Se for comparado δx é tipicamente pequeno comparado com x e portanto muitos dígitos são perdidos se for implementado da maneira mais direta. A soma compensada permite acompanhar a precisão perdida e reduzir o acúmulo de erros aleatórios, trabalhando efetivamente com a precisão estendida e um mínimo de trabalho adicional para a CPU. Isto é implementado em cada atualização do valor de posição e velocidade de uma partícula no final e durante cada passo de tempo da integração.

O Rebound também possui outros módulos gravitacionais disponíveis que estão listados abaixo mas que não serão abordados, pois não foram foco de estudo e uso deste trabalho.

REB_GRAVITY_COMPENSATED:

Somatório direto com somatório compensado, $O(N^2)$, padrão

REB_GRAVITY_NONE:

Sem auto-gravidade

REB_GRAVITY_TREE:

Algoritmo em árvore, Barnes & Hut 1986, $O(Nlog(N))$

REB_GRAVITY_OPENCL:

Somatório direto,$O(N^2)$, mas acelerado utilizando OpenCL (placa de vídeo).

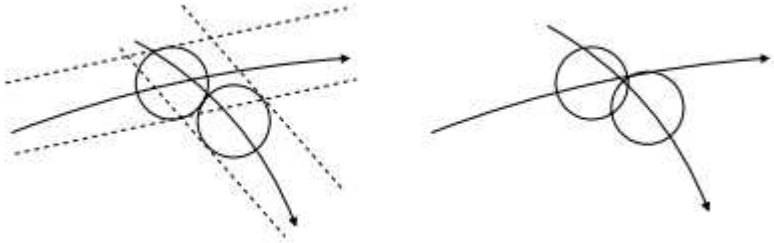
REB_GRAVITY_FFT:

Solução bidimensional usando FFTW, funciona em uma caixa periódica e em um plano compartilhado(*shearing sheet*).

2.2.1.2 Algoritmo de detecção de colisões

O Rebound suporta diferentes módulos de detecção de colisões. Todos os métodos de colisões fazem a procura por colisões apenas com aproximações, então algumas colisões podem ser perdidas ou pode ocorrer de detectar uma colisão inexistente. Isto acontece porque algumas trajetórias das partículas que são curvas são aproximadas em linhas retas ou ocorre porque algumas partículas precisam se sobrepor para que ocorra uma colisão (fig. 1).

Figura 1 - Diferentes algoritmos de detecção de colisões. A imagem à esquerda mostra trajetórias curvas que são aproximadas em linhas retas. No da direita as trajetórias não são aproximadas, as partículas irão colidir apenas quando houver sobreposição.



Fonte: (Rein & Liu, 2012)

Em todos os módulos, a ordem das colisões é aleatória. Isto assegura que não haja uma preferência na ordenação que possa conduzir a falsas correlações quando uma das partículas colide com múltiplas partículas durante um período de tempo.

Um detalhe importante é que deve haver um cuidado ao escolher um integrador, pois alguns possuem o passo de integração fixo que é definido pelo usuário e é preciso garantir que o passo é suficientemente pequeno para que uma partícula colida com outra em apenas um passo de tempo, pelo menos em média.

Utiliza-se um modelo de colisão em esfera dura e livre. As colisões individuais são resolvidas usando a conservação de momento e energia. No modelo de colisões inelásticas a velocidade, sendo constante ou arbitrária, depende do coeficiente normal de restituição ε. A velocidade relativa após uma colisão é então dada por:

$$v_n' = -\varepsilon v_n \tag{3}$$

$$v_t' = v_t \tag{4}$$

onde v_n e v_t são as velocidades relativa normal e tangencial antes da colisão.

O módulo mais simples de busca por colisões disponibilizado pelo Rebound é o de busca direta que procura por colisões com os vizinhos mais próximos. A colisão é detectada quando qualquer par de partículas se sobrepõem. O tratamento das colisões deve ser realizado individualmente.

Cada par de partículas é verificado a cada passo de tempo, fazendo com que a escala do método seja $O(N^2)$. Semelhante ao método de soma direta para a gravidade, isso só é útil para um pequeno número de partículas. O Rebound disponibiliza módulos para verificar colisões bem mais rápidos, que estão listados a seguir, mas que não serão abordados neste trabalho.

REB_COLLISION_NONE:

Sem detectar colisões, padrão

REB_COLLISION_DIRECT:

Pesquisa direta com os corpos próximos, $O(N^2)$

REB_COLLISION_TREE:

Algoritmo de busca em árvore, $O(Nlog(N))$

REB_COLLISION_SWEPP :

Algoritmo de plano de varredura, para problemas de baixa dimensionalidade, $O(N)$ ou $O(N^{1.5})$ depende da geometria

2.2.1.3 Condições de contorno

O Rebound também disponibiliza algumas condições de contorno que podem ser utilizadas em diferentes aplicações. Por exemplo: se uma partícula do sistema se afasta a uma certa distância que não será mais relevante na integração, para diminuir o processamento o Rebound remove esta partícula da simulação. Ou se o projeto possui condições periódicas, no qual em certo momento uma partícula se afasta passando a influenciar pouco, mas ela retorna voltando a ser importante, o Rebound possui módulos que criam “caixas", em até três dimensões, que verificam quais partículas estão próximas ou não umas das outras para poder descartar, naquele momento as menos relevantes, isto diminui muito o processamento.

Neste projeto, não iremos tratar de nenhum sistema que seja necessário a aplicação das condições de contorno, mas para efeito de demonstrar a eficiência deste software listamos resumidamente todas as

opções abaixo.

REB_BOUNDARY_NONE:

As partículas não são afetadas pelas condições de contorno, padrão.

REB_BOUNDARY_OPEN:

As partículas são removidas da simulação se deixarem a caixa.

REB_BOUNDARY_PERIODIC:

Condições de contorno periódicas. As partículas são inseridas do outro lado, se elas cruzam os limites da caixa. Pode-se utilizar um número arbitrário de caixas fantasmas com este módulo.

REB_BOUNDARY_SHEAR:

Condições de contorno periódicas de cisalhamento. Semelhante às condições de contorno periódicas, mas as caixas fantasmas estão se movendo com velocidade constante, definida pelo cisalhamento.

2.2.1.4 Integradores

Vários integradores diferentes foram implementados no Rebound. A maioria dos integradores são simpléticos e de segunda ordem em precisão. Os integradores simpléticos possuem diversas vantagens sobre os integradores não-simpléticos: conservam todos os invariantes de Poincaré tais como a densidade do espaço de fase e têm uma quantidade conservada, que pode ser considerada como uma versão ligeiramente perturbada da Hamiltoniana original. Em muitas situações isso se traduz em um limite superior no erro de energia total. No contexto da mecânica celeste, se os objetos de interesse estão em órbitas quase Keplerianas, então um integrador simplético de variável mista é particularmente útil, permitindo uma alta precisão, utilizando passos de tempo de integração relativamente grandes. Porém a natureza simplética é formalmente perdida assim que a (auto)-gravidade ou as colisões são aproximadas ou quando as forças dependentes da velocidade são incluídas.

A maioria dos integradores seguem o esquema de Drift-Kick-Drift(DKD) que significa dizer que cada passo de tempo é divido em 3 sub-passos. Os sub-passos nos integradores simpléticos do Rebound foram implementados de maneira diferente dos integradores comumente utilizados. Primeiro é descrito a evolução das partículas em termos da Hamiltoniana H que frequentemente pode ser descrita como a soma de duas Hamiltonianas:

$$H = H_1 + H_2 \tag{5}$$

A maneira como a Hamiltoniana será tratada depende do integrados, mas normalmente é dividida em duas partes. Onde usualmente primeiro identifica a energia cinética $H_1(p)$ e o segundo a energia

potencial $H_2(q)$, onde p e q são o momento canônico e as coordenadas, no primeiro sub-passo, o *drift*, as partículas evoluem sob a Hamiltoniana H_1 durante metade do passo de tempo $dt/2$. Então, durante o próximo sub-passo, o *kick*, as partículas evoluem sob a Hamiltoniana H_2 para um passo de tempo inteiro dt. Por fim, as partículas evoluem novamente sob a Hamiltonia H_1 durante meio passo de tempo. Observe que as posições e velocidades são sincronizadas no tempo apenas no final dos passos de tempo DKD.

O integrador simplético Wisdom-Holman (WH, Wisdom & Holman 1991) calcula o movimento kepleriano de dois corpos orbitando um ao outro com exatidão e precisão da máquina durante o sub-passo *drift*. Assim, é muito preciso para problemas em que o movimento de partículas é dominado por um potencial central $1/r$ e as perturbações adicionadas no sub-passo *kick* são pequenas. No entanto, o integrador WH é substancialmente mais lento do que outros integradores porque a equação de Kepler é resolvida iterativamente a cada passo de tempo para cada partícula.

O integrador assume que o objeto central é a partícula com o índice 0, que está na origem e que não se move. As coordenadas de todas as partículas são assumidas como heliocêntricas. Durante os sub-passos as coordenadas são convertidas para as coordenadas de Jacobi, sendo de acordo com seu índice. A partícula com índice 1 tem o primeiro índice de Jacobi, e assim por diante. Isso funciona melhor se as partículas são classificadas de acordo com seu semi- eixo maior. Observe que isso não é feito automaticamente, o usuário que irá definir quais serão os índices das partículas conforme a ordem que ele vai adicionando-as na simulação.

O Rebound utiliza o mesmo passo de tempo para todas as partículas. O passo de tempo pode ser fixo e deve ser definido pelo usuário como nos integradores *Leap-frog*, *Wisdom-Holman Mapping*, *Symplectic Epicycle Integrator*. Ou o passo pode ser definido automaticamente como no integrador IAS15.

O integrador com controle de passo adaptativo que o Rebound disponibiliza é o chamado IAS15, que é de 15^a ordem de precisão. Ele é baseado no integrador na quadratura de Gauß-Radau que suporta forças conservativas e não conservativas, encontros próximos e órbitas de alta excentricidade, mantendo os erros sistemáticos abaixo da precisão de máquina. Testes feitos em Rein & Spiegel 2015 mostraram que este integrador é superior a outros integradores simpléticos em velocidade e precisão. De fato o IAS15 preserva a simpleticidade dos sistemas Hamiltonianos melhor do que os integradores normalmente utilizados.

Existem algumas complicações ao utilizar integradores simpléticos. Uma das dificuldades é manter o passo de tempo adaptativo mantendo a simpleticidade. A maioria das variáveis simpléticas trabalham em sistemas heliocêntricos, necessitando que um objeto trabalhe com uma “estrela” no sistema, ou seja, com um corpo central. Isto pode ser um grande problema para sistemas em que não tem um corpo central bem definido, como o problema de estrelas binárias. Outro problema são forças não conservativas, forças que não podem ser descritas por um potencial, como as forças de radiação porque elas dependem da velocidade da partícula, não apenas de sua posição. Quando forças não conservativas são incluídas na

equação de movimento, o conceito de um integrador simplético, que depende do sistema Hamiltoniano, é quebrado.

O principal objetivo do integrado IAS15 é ter um controle de passo adaptativo mantendo simpleticidade do sistema e atingir uma alta precisão. A principal equação que este integrador tenta resolver é da forma:

$$y'' = F[y', y, t] \tag{6}$$

onde y'' é a aceleração da partícula e F é a função que descreve uma força específica, que pode depender da posição da partícula y, da velocidade y' e do tempo t. Esta equação é geral o suficiente para permitir a dependência de velocidades arbitrárias e de forças não conservativas. Assim, o sistema pode não corresponder a um sistema Hamiltoniano.

O IAS15 realiza uma expansão da equação (6) e algumas aproximações para poder estimar a posição e velocidade das partículas. Estes valores são dados em função da posição e velocidades correspondentes ao início do passo de tempo, que é um valor exato, também são dados em função de valores arbitrários no tempo e de valores correspondentes ao final do passo de integração. O truque deste integrador para realizar aproximações extremamente precisas é que ele faz uso de sub-passos com espaçamentos do tipo Gauß-Radau (espaçamentos equidistantes). O espaçamento de Gauß-Radau está intimamente relacionado com a quadratura Gaußian padrão que pode ser usada para aproximar uma integral, mas que faz uso dos estados iniciais. A utilização disto é vantajosa, pois nós já sabemos as posições e as velocidades das partículas no início do passo de integração.

Para obter o valor final das posições e velocidades através do IAS15 é realizada uma estimativa das forças durante o passo de integração, que pode ser considerado como sub-passo que realiza um esquema de previsão e correção da seguinte maneira: primeiro, realiza uma estimativa aproximada para as posições e velocidades para calcular as forças. A primeira interação é realizada considerando uma partícula se movendo com aceleração constante (previsão). Então, são utilizadas as forças para calcular melhor a estimativa das posições e das velocidades (correção). Este processo é realizado até que as posições e velocidades convirjam para a precisão da máquina. Apenas algumas (aproximadamente 2) interações são necessárias para que se atinja a precisão da máquina.

Esquemas de ordem inferior exigirão muito mais passos de tempo por órbita para obter a precisão necessária. Então por que utilizar um integrador como IAS15? Primeiramente todo integrador depende do problema abordado, mas o principal é manter o passo de tempo como fração do tempo dinâmico, em torno de 1%, e manter os erros gerados em cada passo de tempo bem pequenos.

Para aumentar a velocidade de processamento deste tipo de integrador, o Rebound também disponibiliza um integrador simplético híbrido chamado de Hermes que usa WHFast para integrações de longo

prazo, mas muda para IAS15 para encontros próximos. Ele utiliza dois critérios para fazer a troca entre os integradores, uma é feita através do raio da esfera de Hill (por padrão é 3) e a outra é em termos do raio do corpo central (por padrão é 15). Mas ele já realiza um controle a cada interação para saber qual é o fator de troca, em termo da esfera de Hills é mais apropriado.

Outros integradores disponíveis pelo Rebound estão listados abaixo mas alguns não foram alvo de estudo e nem foram utilizados na validação do software.

REB_INTEGRATOR_IAS15:

IAS15 é um integrador de 15^a ordem com controle adaptativo do tamanho de passo. É um integrador com ordem muito elevada e não simplético que pode trabalhar com forças arbitrárias (dependentes ou não da velocidade) e é na maioria dos casos com precisam a baixo da precisão da máquina. IAS15 pode integrar equações variacionais. Rein & Spiege 2015.

REB_INTEGRATOR_WHFAST:

WHFast é o integrador descrito em Rein & Tamayo 2015, é um integrador de segunda ordem Wisdom Holman com corretores simpléticos de 11^a ordem. É extremamente rápido e preciso, usa as funções de Gauss f e g para resolver movimento Kepleriano e pode integrar equações variacionais.

REB_INTEGRATOR_WHFASTHELIO:

Integrador simplético de segunda ordem usando as soluções de Kepler para WHFast (Rein & Tamayo 2015) mas funciona em coordenadas heliocêntrica democráticas. Este sistema de coordenadas é melhor se os planetas têm órbitas de passagem ou posições de *swap* durante as integrações.

REB_INTEGRATOR_EULERR:

Esquema de Euler, primeira ordem

REB_INTEGRATOR_SEI:

Integrador de epiciclo simplético (SEI), integrador simplético variável misto para um plano compartilhado (shearing sheet), de segunda ordem, Rein & Tremaine 2011.

REB_INTEGRATOR_HERMES:

Integrador simplético híbrido que usa WHFast para integrações de longo prazo, mas muda para IAS15 para encontros próximos.

2.2.2 Estruturas e variáveis

Na documentação da API do Rebound são disponibilizadas todas as estruturas que o software contém. A seguir iremos fazer uma breve apresentação destas estruturas para poder exemplificar o quão

prático é o Rebound para trabalhar com as partículas de um sistema. Como já havíamos mencionado anteriormente, a estrutura *reb_simulation* é a mais importante para uma simulação do Rebound, pois contém todas as variáveis da simulação, como:

```
1 struct reb_simulation {
2 /*
3 * Variaveis temporais:
4 */
5     double  t;                      > Tempo atual da simulação
6     double  dt;                     > Passo de tempo atual.
7     double  dt_last_done;           > Último dt utilizado pelo integrador
8     int exact_finish_time;          > Define quando termina a integração.
```

Estas variáveis tratam do tempo de integração da simulação e através delas podemos definir o passo de tempo inicial ou fixo, mas também podemos acessá-las para poder acompanhar a simulação. Podemos definir duas maneiras para finalizar o tempo de integração, quando definimos a variável *exact_finish_time* igual a 1 a integração será finalizada no tempo máximo definido pelo usuário (padrão), quando o valor é 0 ele irá finalizar no próximo passo de integração.

```
1 /*
2 * Numero e tipo de particulas:
3 */
4   int N;                          > Número total das atuais partículas
5   int N_active;                   > Número total de partículas ativas
6   int testparticle_type;          > Tipo da partícula teste.
7   struct reb_particle *particles> Vetor principal que contém todas partículas.
```

Observe que N é o número atual, o que significa dizer que caso alguma partícula seja removida do sistema, seja por condições de contorno ou colisões este número será atualizado.

O número de partículas define quantas não serão tratadas como partículas teste. Se a variável *test-particles_type* for definida com o valor 0, as partículas teste não irão influênciar nenhuma outra partícula. Mas se for definida como 1, isto significa que as partículas com o índice menor do que *N_active* não sentem as partículas testes (similar a pequenas partículas do *Mercury*), mas as partículas teste sentem umas as outras.

A estrutura de partículas é umas das estruturas mais importantes pois é através delas que iremos inserir corpos na simulação. Esta estrutura é formada da seguinte maneira:

```
1  struct reb_particle {
2      double x;                    > Posição cartesiana x da partícula
3      double y;                    > Posição cartesiana y da partícula
4      double z;                    > Posição cartesiana z da partícula
5      double vx;                   > Componente x da velocidade
6      double vy;                   > Componente y da velocidade
7      double vz;                   > Componente z da velocidade
8      double ax;                   > Componente x da aceleração
9      double ay;                   > Componente y da aceleração
10     double az;                   > Componente z da aceleração
11     double m;                    > Massa
12     double r;                    > Raio
13     double lastcollision;        > Última vez que houve uma colisão física
14     struct reb_treecell* c;      > Ponteiro para a célula que contem a partícula
15     uint32_t hash;               > Identificador da partícula
16     void* ap;                    > Para adição externa de novas propriedades
17     struct reb_simulation* sim; > Ponteiro para a simulação que pertence
18 };
```

Além desta estrutura existe uma outra estrutura auxiliar para as partícula, que serve para calcular e armazenar órbitas keplerianas a partir das coordenadas cartesianas, como a estrutura *reb_orbit* com os seguintes argumentos:

```
struct reb_orbit {
    double d;                       > Distância radial do objeto central
    double v;                       > Velocidade relativa ao objeto central
    double h;                       > Momento angular
    double P;                       > Periodo orbital
    double n;                       > Movimento médio
    double a;                       > Semi-eixo maior
    double e;                       > Excentricidade
    double inc;                     > Inclinação
    double Omega;                   > Longitude do nodo ascendente
    double omega;                   > Argumento do pericentro
    double pomega;                  > Longitude do pericentro
    double f;                       > Anomalia verdadeira
    double M;                       > Anomalia média
    double l;                       > Longitude média
    double theta;                   > Longitude verdadeira
    double T;                       > Tempo de passagem do pericentro
};
```

```
/*
* Variaveis fisicas:
*/
 double  G                          > Constante gravitacional.
 double  softening                  > Parâmetro de amortecimento gravitacional.
 unsigned int gravity_ignore_terms  > Ignora a gravidade do corpo central.
 struct reb_vec3d *gravity_cs       > Vetor que contém a informação do somatório
                                      compensado da gravidade.
```

A constante gravitacional por padrão é 1, então caso as outras constantes físicas não estejam normalizadas é necessário alterar o valor da constante gravitacional na unidade dos valores utilizados no sistema. Por exemplo: caso utilize o sistema internacional de unidades o valor deve ser $G = 6.672 \times 10^{-11} m^3 kg^{-1} s^{-2}$.

Caso deseje ignorar a gravidade do corpo central, isso só é possível com dois integradores WHFast e WHFastHelio. Caso utilize WHFast, defina a variável que ignora o corpo central como 1 e para o WHFastHelio como 2.

```
/*
* Variaveis status:
*/
     REB_STATUS status;          > Para sair da simulação no final do próximo passo.
     double exit_max_distance    > Sai da simulação se a distância da origem é maior
                                   que este valor
     double exit_min_distance    > Sai da simulação se a distância da origem é menor
                                   que este valor
     double usleep               > Espera microssegundos ao final do passo de tempo
```

Estas variáveis ajudam a controlar a simulação, podendo finalizar a integração caso ocorra algum evento. A variável *usleep* espera o tempo de microssegundos com o valor que foi definido a esta variável, esta espera é especialmente útil quando utiliza-se as visualizações pelo OPENGL.

```
/*
* Variaveis de colisoes:
*/
 int collision_resolve_keep_sorted> Mantém a ordenação ao remover uma partícula
 struct reb_collision *collisions > Contém todas as colisões
 int collisions_allocatedN        > Quantidade de colisões alocadas
 double collisions_plog           > Acompanha a troca de momento
 long collisions_Nlog             > Acompanha o número de colisões
 double minimum_collision_velocity> Usado para o modelo de colisão de esfera rígida.
```

Estas variáveis ajudam a controlar e a acompanhar as colisões da simulação que ficam salvas na estrutura de colisões, cujo algoritmo é:

```
struct reb_collision{
        int p1;                  > Uma das partículas que colidiram
        int p2;                  > Uma das partículas que colidiram
        int ri;                  > Índice da célula de root(Usado apenas com MPI)
        struct reb_ghostbox gb;  > Ghostbox (da partícula p1, usada com condições
}                                  de contorno)
```

O algoritmo do Rebound possui diversas rotinas, sendo que as principais delas estão ligadas aos módulos de colisão, integração, condições de contorno e gravidade. Para definir qual módulo será utilizado é necessário enviar o valor de cada um deles para estas rotinas. O Rebound já possui alguns variáveis criadas para ajudar na definição de cada um destes módulos. A seguir mostraremos a rotina e as variáveis

para cada um dos módulos disponíveis.

Rotina de busca por colisões disponíveis:

```
reb_simulation::@0 reb_simulation::collision
```

`REB_COLLISION_NONE = 0`	Não procura por colisões
`REB_COLLISION_DIRECT = 1`	Busca direta
`REB_COLLISION_TREE = 2`	Busca por árvore

Rotina das condições de contorno disponíveis:

```
reb_simulation::@2 reb_simulation::boundary
```

`REB_BOUNDARY_NONE = 0`	Não busca por colisões (padrão)
`REB_BOUNDARY_OPEN = 1`	Condições de contorno abertas.
`REB_BOUNDARY_PERIODIC = 2`	Condições de contorno periódicas
`REB_BOUNDARY_SHEAR = 3`	Condições de contorno periódico de cisalhamento

Rotina das condições de contorno disponíveis:

```
reb_simulation::@3 reb_simulation::gravity
```

`REB_GRAVITY_NONE = 0`	Não calcula as forças gravitacionais
`REB_GRAVITY_BASIC = 1`	Soma direta
`REB_GRAVITY_COMPENSATED = 2`	Soma compensada
`REB_GRAVITY_TREE = 3`	Cálculo por algoritmo de árvore

Rotina dos integradores disponíveis:

```
reb_simulation::@1 reb_simulation::integrator
```

`REB_INTEGRATOR_IAS15 = 0`	Integrador IAS15 (padrão)
`REB_INTEGRATOR_WHFAST = 1`	Integrador Wisdom-Holman (WH)
`REB_INTEGRATOR_SEI = 2`	Integrador Simplético de Epiciclos (SEI)
`REB_INTEGRATOR_LEAPFROG = 4`	Integrador Leap frog
`REB_INTEGRATOR_HERMES = 5`	Integrador Hermes (experimental)
`REB_INTEGRATOR_WHFASTHELIO = 6`	Integrador WHFastHelio coordenadas heliocêntricas
`REB_INTEGRATOR_NONE = 7`	Não realiza integração

Cada um destes integradores possui uma estrutura própria, com diversas variáveis que podemos alterar para modificar a integração devido aos interesses da aplicação. A seguir mostraremos a estrutura dos integradores que foram estudados neste trabalho que são o WH, Hermes e IAS15.

```
struct reb_simulation_integrator_whfast {
unsigned int corrector
unsigned int recalculate_jacobi_this_timestep
unsigned int safe_mode
struct reb_particle* restrict reb_simulation_integrator_whfast::p_j
unsigned int keep_unsynchronized
}
```

A variável do *corrector* altera o corretor simplético. Com o valor 0 é desativado o corretor. 3 indica um corretor de terceira ordem (2 estágios). 5: corretor de quinta ordem (4 estágios). 7: corretor de sétima ordem (6 estágios). 11: corretor de décima primeira ordem (10 estágios).

O valor de *recalculate_jacobi_this_timestep* recalcula as coordenadas de Jacobi no final do próximo passo, depois disto ele define o valor novamente como 0. Então se deseja que o valor seja recalculado em cada passo, é necessário que este valor seja definido em cada passo de integração.

O *safe_mode* é uma variável que já é definida por padrão, serve para que seja recalculado as coordenadas de Jacobi e sincronize em cada passo de integração. Para desativá-la é necessário definir como 0. Isto resultará em uma aceleração no processamento, porém deve haver um cuidado para sincronizar e recalcular coordenadas Jacobi quando necessário.

A estrutura *p_j* contém as coordenadas de Jacobi para todas as partículas.

A variável inteira *keep_unsynchronized* gera as coordenadas iniciais no final do passo de integração, sem alterar as coordenadas de Jacobi.

```
struct
reb_simulation_integrator_hermes{
 struct reb_simulation *mini     > Cria uma mini simulação com IAS15.
 struct reb_simulation *global   > Cria uma simulação global com WHFast
 double hill_switch_factor       > Fator de troca dos integradores em termos do raio de
                                   Hill(padrão: 3)
 double solar_switch_factor      > Fator de troca dos integradores em termos do raio do
                                   corpo central(padrão:15)
 int adaptive_hill_switch_factor> Calcular automaticamente o fator de troca da esféra
                                   de hills
  int mini_active                > Ativar mini simulação
}
```

Para entender melhor o que é uma mini ou global simulação veja Silburt et al 2016.

```
struct reb_simulation_integrator_ias15 {
  double epsilon                          > Precisão(padrão: 1e-9)
  double min_dt                           > Define um passo mínimo.
  unsigned int epsilon_global             > Determina como os erros relativos da ace-
}                                           leração são estimados.
```

2.2.3 Código básico

O Rebound possui uma estrutura bem simples. Mostraremos os comandos básicos da versão em C do Rebound para fazer as configurações, definir os módulos, incluir as partículas, integrar o sistema e aplicar as forças dissipativas.

Todos os códigos devem incluir a biblioteca do Rebound, que contém todas as declarações das estruturas e funções que iremos utilizar.

```
#include "rebound.h"
```

Primeiramente sempre deve ser criado a estrutura *reb_simulation*. Esta é a principal estrutura que contém todas as variáveis, os ponteiros e partículas de uma simulação do Rebound. Podem ser criadas várias estruturas *reb_simulation* ao mesmo tempo, porque ele pode ter o programa paralelizado com segurança (*thread-safe*).

```
struct reb_simulation* r = reb_create_simulation();
```

A variável *r* é um ponteiro da estrutura, com ele utilizado a sintaxe de flechas para definir as variáveis do Rebound. Através deste ponteiro podemos definir os módulos, por exemplo o módulo de

Integração:

```
r->integrator = REB_INTEGRATOR_WHFAST;
```

Gravitação:

```
r->gravity      = REB_GRAVITY_BASIC;
```

Condições de contorno:

```
r->boundary     = REB_BOUNDARY_OPEN;
```

Colisões

```
r->collision    = REB_COLLISION_TREE;
```

Dependendo do módulo escolhido ele exige algumas configurações se você deseja somente alterar algumas configurações padrões, é necessário recorrer a API do rebound. Neste caso contrário, basta definir os argumentos do integrador, por exemplo:

```
r->ri_ias15.epsilon = 1e-6;
```

Para inserir as partículas é necessário utilizar a estrutura *reb_particle* que possui os argumentos de posição, velocidade e aceleração um para cada eixo cartesiano, também a massa, raio, o tempo que a partícula teve a sua ultima colisão e a *hash* para identificar as partículas.

```
struct reb_particle star = {0};
```

Para adicionar as partículas na simulação usamos a função *reb_add()*, que possui dois parâmetros de entrada, o primeiro é a simulação que adicionaremos a partícula e o segundo é a partícula que deseja adicionar.

Antes de finalmente começar a integração devemos chamar a função *reb_move_to_com()*, ela move as partícula para o sistema de referência do centro de massa, prevenindo que as partículas se afastem da origem.

```
 reb_move_to_com(r);
```

Então para começar a integração é necessário chamar a função *reb_integrate()* no qual o primeiro parâmetro é a simulação que você deseja executar e o segundo é tempo de integração.

```
 reb_integrate(r,100.);
```

Agora vamos ver as funções *call-back*, estas funções são chamadas a cada passo de integração e podemos utilizá-las para gravar os dados da simulação e aplicar as forças nas partículas. O ponteiro de

função mais relevante é chamado *heartbeat* na estrutura *reb_simulation*. Primeiramente, nós declaramos e implementamos a função e, em seguida, definimos o ponteiro na rotina principal:

```
1    void heartbeat(struct reb_simulation* r){
2       printf("%f\n",r->t);
3    }
4    int main(int argc, char* argv[]) {
5       ...
6       r->heartbeat = heartbeat;
7       ...
8    }
```

Com esta função podemos acessar todas as variáveis e partículas durante a simulação. Não é necessário nenhuma variável global para isto. Por exemplo, se necessitamos imprimir a coordenada x da segunda partícula podemos utilizar esta função.

E também a função *reb_tools_particle_to_orbit* que calcula os elementos orbitais de uma determinada partícula referente a outro corpo como:

```
1 struct reb_particle p = r->particles[2];
2 struct reb_particle star = r->particles[0];
3 struct reb_orbit o = reb_tools_particle_to_orbit(r->G,p,star);
```

que retorna os elementos orbitais da partícula *p* em relação a partícula *star*.

Agora para implementar as forças adicionais definimos a função que será chamada também a cada passo de integração na rotina principal do programa:

```
1     r->additional_forces = radiation_forces;
```

Para inicializar esta função e aplicar a força na partícula que desejamos é feita seguinte forma:

```
1   void radiation_forces(struct reb_simulation* r){
2     for (int i=1;i<N;i++){
3        ...
4        r->particles[i].ax += F(x);
5        r->particles[i].ay += F(y);
6        r->particles[i].az += F(z);
7     }
8  }
```

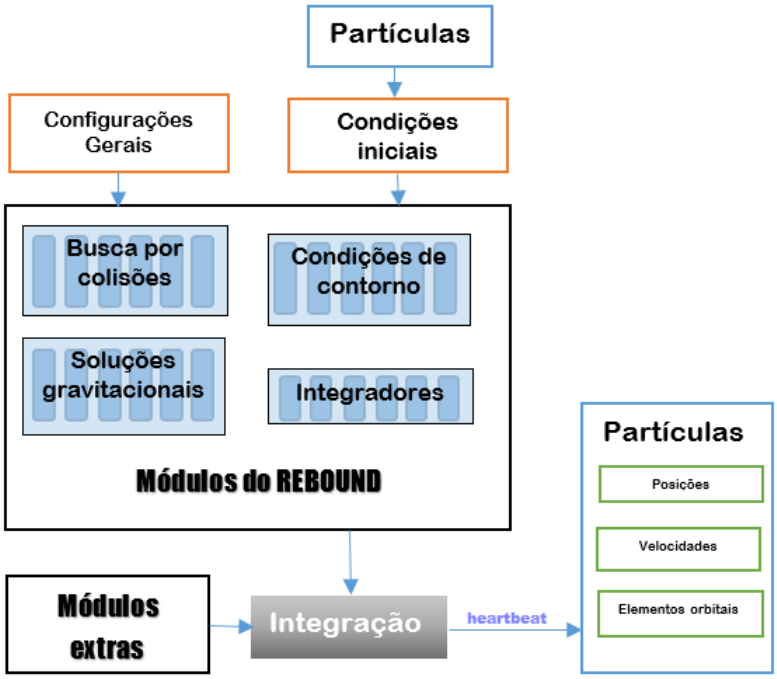
Desta maneira é utilizado um *laço* para percorrer o vetor de partículas que desejamos e aplicar a força através de operadores atribuição aritmética, como +=, em cada uma das coordenadas da aceleração de

partícula. Portanto, todas as forças aplicadas nas partículas em Rebound devem ser decompostas nas coordenadas cartesianas.

Utilizamos esta função para adicionar forças não gravitacionais. Para criar funções diferentes das que o Rebound disponibiliza basta criar as funções e defini-las na variável *additional_forces*, isto é o que faremos nas próximas seções para adicionar as forças dissipativas para o estudo de anéis planetários.

Por fim, podemos concluir que uma simulação do Rebound pode ser entendida a partir do fluxograma da figura 2. Primeiramente devemos fazer as configurações gerais do sistema, definindo quais módulos serão utilizados, tempo de integração, parâmetros externos, bibliotecas e qualquer outra configuração. Então adicionamos as partículas com suas condições iniciais e inicializamos a simulação do Rebound que, a partir dos módulos definidos e dos módulos externos, fará a integração que com auxílio da função *heartbeat* podemos monitorar a evolução da partícula a cada passo de tempo.

Figura 2 - Fluxograma de uma simulação do Rebound.



Fonte: Autoria própria

2.2.3.1 Diferentes maneiras de inicializar as partículas

Na seção 2.2.2 mostramos os argumentos da estrutura da partícula, no qual podemos definir alguns destes argumentos para inserir a partícula na simulação, como por exemplo:

```
1  struct reb_particle star = {0};
2  star.m  = 1.;
```

```
3    reb_add(r, star);
4
5    struct reb_particle p = {0};
6     p.m  = 0;                                          // massless
7    double a = 1.;                                     // a = 1 AU
8    double v = sqrt(r->G*(star.m)/a);
9    double phi = reb_random_uniform(0,2.*M_PI);  // random phase
10   p.x  = a*sin(phi);  p.y  = a*cos(phi);
11   p.vx = -v*cos(phi); p.vy = v*sin(phi);
12   reb_add(r, p);
```

Desta maneira adicionamos uma partícula central de massa 1 e em seguida adicionamos uma partícula sem massa com as posições e velocidades referentes ao corpo central.

Existe uma outra maneira de inserir uma partícula fazendo referência a outra, que é utilizando a função que o Rebound disponibiliza chamada *reb_tools_orbit_to_particle*. Com auxílio da estrutura *reb_orbit* podemos adicionar uma partícula através de seus elementos orbitais:

```
1    struct reb_particle star = {0};
2    star.hash = reb_hash("Sun");
3    star.m  = 1.;
4    reb_add(r, star);
5
6    struct reb_particle p = {0};
7    struct reb_orbit o = {0};
8
9    p.m  = 0;                                          // massless
10   o.a = 1.;                                         // a = 1 AU
11   o.f = reb_random_uniform(0,2.*M_PI);      // random phase
12
13  p =  satellite = reb_tools_orbit_to_particle(r->G,star,p.m,o.a,o.
       e,o.inc,o.Omega,o.omega,o.f);
14  p.hash = r->N;
15  reb_add(r, p);
```

Observe que os elementos orbitais diferentes do semi-seixo maior e da anomalia verdadeira não tiveram nenhum valor explicitamente declarado, mas ao declaramos a estrutura *reb_orbit* utilizamos a sintaxe {0} que definiu inicialmente todos os argumentos da estrutura como zero. Mesmo que não tenhamos definido todos os elementos orbitais, a função *reb_tools_particle_to_orbit* calculará todos os outros elementos possíveis a partir dos argumentos já definidos.

Estes foram exemplos simples que mostram duas maneiras diferentes para inserir as partículas com as mesmas condições iniciais.

2.2.3.2 Como salvar a evolução das partículas

Como apresentado na seção 2.2.3 o Rebound disponibiliza a chamada de uma função a cada passo de integração, no caso a função *heartbeat*. Esta função pode ser utilizada para acompanhar a evolução das partículas durante a integração e salvar em um arquivo cada um destes valores.

Vamos mostrar um exemplo que exibe na tela o tempo de processamento da simulação e que salva em dois arquivos diferentes a evolução das posições, velocidades e acelerações das partículas e o outro arquivo que salva a evolução orbital referente ao corpo inicial.

```
1
2  double tOutput  =       3.154e+7;       //print per year [s]
3
4  int main(int argc, char* argv[]){
5          struct reb_simulation* r = reb_create_simulation();
6          r->G    = 6.672e-11;    // [m^3 kg^-1 s^-2]
7          ...
8          r->heartbeat                    = heartbeat;
9          ...
10 }
11
12 void heartbeat(struct reb_simulation* r){
13   if (reb_output_check(r, tOutput)){
14         reb_output_timing(r, tmax);
15
16         struct reb_particle planet = *reb_get_particle_by_hash(r,
              reb_hash("Sun"));
17
18         FILE* f = fopen("orbits.txt","a");
19         int N = r->N; //N_satellite
20         for (int i=0;i<N;i++){
21            struct reb_particle p = r->particles[i];
22            struct reb_orbit o = reb_tools_particle_to_orbit(r->G,r
                 ->particles[i],planet);
23            fprintf(f,"%u\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\n",p.
                 hash,r->t,o.a,o.e,o.inc,o.Omega,o.omega,o.l,o.P,o.f);
```

```
24          }
25          fclose(f);
26
27          FILE* f = fopen("positions.txt","a");
28          int N = r->N; //N_satellite
29          for (int i=0;i<N;i++){
30             struct reb_particle p = r->particles[i];
31             fprintf(f,"%u\t%e\t%e\t%e\t%e\t%e\t%e\t%e\n",p.hash,r->t
                  ,p.x,p.y,p.z,p.vx,p.vy,p.vz);
32          }
33          fclose(f);
34
35
36  }
```

Neste caso a função *reb_output_timing*, disponibilizada pelo Rebound, serve para imprimir na tela o estado da simulação, o número atual de partículas, o tempo de integração e o passo de tempo da última integração.

A cada passo de integração a função *heartbeat* será chamada, mas os arquivos só serão gravados a cada ano do tempo de integração, o que é feito com auxílio da função *reb_output_check* que verifica o espaço de tempo, no caso *tOutput*, da simulação *r*.

O Rebound também disponibiliza algumas funções de saída como:

1. ```
   void reb_output_orbits(struct reb_simulation *r, char *
      filename)
   ```

   que salva no arquivo com nome passado ao parâmetro *filename* os parâmetros orbitais calculados nas coordenadas de Jacobi. Esta função assume que o corpo central é a partícula com índice 0, ou seja, a primeira partícula inserida na simulação. Cada vez que esta função é chamada, N - 1 linhas são anexadas ao arquivo e cada linha no arquivo corresponde a uma partícula e contém as seguintes colunas (separadas por tab): tempo($r->t$), semi-eixo maior($o.a$), excentricidade($o.e$), inclinação($o.inc$), nodo ascendente($o.Omega$), argumento do pericentro($o.omega$), longitude média($o.l$), período($o.P$), anomalia verdadeira($o.f$).

2. ```
   void reb_output_ascii(struct reb_simulation *r, char *filename
      )
   ```

   Este arquivo anexa as posições e velocidade de todas as partículas em um arquivo do tipo ASCII.

Estas ultimas funções são parecidas com as utilizadas no exemplo no início desta seção, porém não fazem uso das *hash* e também depende que o corpo central seja o com índice 0.

3. ```
   void reb_output_velocity_dispersion(struct reb_simulation *r,
      char *filename)
   ```

   Esta função anexa a um arquivo do tipo ASCII a velocidade de dispersão das partículas.

   Ainda existem algumas que salvem o status da simulação, assim caso a simulação pare e precise ser reinicializada, utiliza-se estes arquivos para continuar a simulação de onde ela parou.

4. ```
   void reb_output_binary(struct reb_simulation *r, char *
      filename)
   ```

   Salva a simulação em uma estrutura binária.

5. ```
   void reb_output_binary_positions(struct reb_simulation *r,
      char *filename)
   ```

   Escreve a posição de todas as partículas em um arquivo binário.
   ```

3 FORÇA DE RADIAÇÃO SOLAR

As partículas de poeira em anéis planetários sofrem a ação de forças dissipativas. Uma destas forças é a pressão de radiação solar que pode ser dividida em duas componentes: a pressão de radiação (RP) e o arrasto de Poynting-Robertson (PR).

Nesta seção iremos analisar os efeitos da radiação na evolução orbital das partículas de poeira em anéis planetários. Para isto, primeiramente iremos apresentar um estudo da força de radiação solar para então criamos um módulo para o Rebound de forças perturbativas em anéis planetários. O software não disponibiliza nenhum algoritmo intrínseco de forças não gravitacionais. Nosso objetivo é então criar algoritmos que possam ser inseridos como módulo ao Rebound e serem utilizados em qualquer aplicação.

3.1 Introdução Teórica

Para um grão de tamanho micrométrico que está em uma órbita circular em torno de um planeta, a pressão de radiação solar pode ser uma forte fonte de perturbação. Na sua forma mais simples, a pressão de radiação aplica uma força sobre um grão no anel dada pela força expressada por Burns et. al. (1979):

$$\overrightarrow{F}_{RP} = -\beta \frac{GM_{\odot}}{R^2} \hat{s} \tag{7}$$

onde $M_{\odot}$ é a massa do Sol, R é a distância do Sol ao planeta, $\hat{s}$ é o vetor posição unitário do planeta na direção do Sol, e β é a relação adimensional da força de gravidade para a radiação solar. Para o nosso Sol, Burns et. al. (1979) expressou:

$$\beta = 5.7 \times 10^{-5} \frac{Q_{pr}}{p_g r_g} \tag{8}$$

que é valido para partículas que obedecem a óptica geométrica ($r_g \geq 0.5\mu$). Neste caso r_g e p_g são o raio e a densidade da partícula, respectivamente, e Q_{pr} é uma constante aproximadamente unitária, cujo valor exato depende das propriedades ópticas do grão. As expressões simples dadas acima ignoraram diversas outras propriedades como o arrasto Poynting-Robertson. Estes efeitos são muito pequenos em comparação com a principal força da pressão de radiação e geralmente pode ser negligenciada em uma primeira aproximação. Apesar do fato de que eles serem fracos, forças dissipativas, como o arrasto de Poynting-Robertson pode ser importante porque eles afetam o semi-eixo maior.

Por isto muitos modelos das forças de radiação desenvolvidas não são sutis o suficiente para fornecer um quadro completo. Em 1903, Poynting investigou as consequências da continua absorção e reemissão da radiação solar das partículas, ele afirmou que deveria existir um arrasto tangencial em adição à força de radiação solar. De fato ele estava correto e a análise completa considera a reemissão da luz absorvida como sendo não isotrópica no Sistema Solar. Por consequência, existe uma componente azimutal na força líquida de radiação. Então primeiramente vamos considerar partículas se movendo radialmente, como na pressão de radiação anteriormente declarada, e então adicionaremos as afirmações de Poynting. Portanto, para uma partícula estacionária movendo radialmente:

$$\vec{v}_{radial} = \dot{r}\hat{r} \tag{9}$$

Considerando o movimento relativo entre a fonte de radiação e a partícula de poeira na equação, podemos calcular a componente radial da força de radiação:

$$\vec{F}_{radial} = F_{RP}\left(1 - 2\frac{\dot{r}}{c}\right)\hat{r} \tag{10}$$

Existem duas contribuições que envolvem o termo $\dot{r}/c$, consequência do coeficiente 2. O primeiro, a energia por fóton é impulsionada por um fator de $1-\dot{r}/c$ devido ao efeito Doppler. O segundo, as partículas de poeira se movem em uma velocidade radial $\dot{r}$ relativa a fonte de radiação e consequentemente encontra fótons a um fator de $1-\dot{r}/c$. Considerando as duas contribuições juntas, a força é aumentada em $(1-\dot{r}/c)^2 \approx 1-2\dot{r}/c$ para $|\dot{r}| << c$.

A força radial não é a única força que a partícula de poeira sente e como afirmado anteriormente por Poynting, também existe uma componente azimutal, que cuja velocidade depende de:

$$\vec{v}_{azimutal} = \vec{v} - \vec{v}_{radial} \tag{11}$$

Para $v \equiv |\vec{v}| << c$, a componente azimutal é apenas a força radial multiplicada pelo raio da velocidade azimutal pela velocidade da luz.

$$\vec{F}_{azimutal} = F_{RP}\frac{-\vec{v}_{azimutal}}{c} + O[(v/c)^2] \tag{12}$$

que pode ser reescrita como:

$$\vec{F}_{azimutal} = F_{RP}\frac{-(\vec{v}-\vec{v}_{radial})}{c} \tag{13}$$

$$\vec{F}_{azimutal} = F_{RP}\frac{\dot{r}\hat{r}-\vec{v}}{c} \tag{14}$$

A força total é então dada pela soma da componente radial com a a azimutal:

$$\vec{F} = \vec{F}_{radial} + \vec{F}_{azimutal} = F_{RP}\left[\left(1-2\frac{\dot{r}}{c}\right)\hat{r} + \frac{\dot{r}\hat{r}-\vec{v}}{c}\right] \tag{15}$$

Portanto, a força de radiação solar total pode ser escrita por:

$$\vec{F} = \beta\left[\left(1-\frac{\dot{r}}{c}\right)\hat{r} - \frac{\vec{v}}{c}\right] \tag{16}$$

sendo $\dot{r} = dr/dt$ a derivada primeira em relação ao tempo do vetor posição e $\vec{v}$ é o vetor velocidade da partícula. Mas pode ser mais conveniente expressar da seguinte maneira:

$$\vec{F} = \beta\left[\left(1-\frac{2\dot{r}}{c}\right)\hat{r} - \frac{r\dot{\theta}}{c}\hat{\theta}\right] \tag{17}$$

Onde $\hat{\theta}$ é o vetor unitário ortorradial, $\hat{r}$ é o fator que representa a correta força radial de radiação e o segundo termo é o que agora consideraremos como a componente de arrasto.

Outro fator que devemos considerar é que a maioria dos sistemas com anéis planetário são coplanares com o plano do planeta, isto é, a inclinação I da partícula no anel é $I \leq \gamma$, onde γ é a obliquidade do planeta. Neste caso, o problema pode ser tratado como bidimensional, quando o Sol e partícula do anel estão no mesmo plano, caso contrário o problema deve ser tratado como tridimensional.

Para uma partícula que descreve uma órbita em torno de uma planeta com obliquidade γ, as componentes da força de radiação, em um sistema inercial cartesiano centrado no planeta, podem ser escrita pelas equações dadas por Sfair & Giuliatti Winter (2009):

$$F_x = \frac{\beta GM_s}{r_{sp}^2}Q_{pr}\left[cos(n_s t) - \left(\frac{x_s}{r_{sp}}\right)^2\left(\frac{v_{xs}}{c}+\frac{v_x}{c}\right) - \left(\frac{v_{xs}}{c}+\frac{v_x}{c}\right)\right] \tag{18}$$

$$F_y = \frac{\beta G M_s}{r_{sp}^2} Q_{pr} \left[cos(\gamma) sin(n_s t) - \left(\frac{y_s}{r_{sp}} \right)^2 \left(\frac{v_{ys}}{c} + \frac{v_y}{c} \right) - \left(\frac{v_{ys}}{c} + \frac{v_y}{c} \right) \right] \tag{19}$$

$$F_z = \frac{\beta G M_s}{r_{sp}^2} Q_{pr} \left[sin(\gamma) sin(n_s t) - \left(\frac{z_s}{r_{sp}} \right)^2 \left(\frac{v_{zs}}{c} + \frac{v_z}{c} \right) - \left(\frac{v_{zs}}{c} + \frac{v_z}{c} \right) \right] \tag{20}$$

onde r_{sp} é a posição relativa do Sol ao planeta, e (x_s, y_s, z_s) são as componentes de r_{sp}. As componentes da velocidade da partícula são (v_x, v_y, v_z) e (v_{xs}, v_{ys}, v_{zs}) são as componentes da velocidade do planeta em torno do Sol. Nesta expressão os termos que não estão relacionados a velocidade correspondem à pressão de radiação solar (RP) e o outro termo corresponde ao Poynting-Robertson (PR), que é uma força contrária à velocidade. O PR causa uma diminuição contínua na energia da partícula ocasionando uma redução no semi-eixo maior da órbita, fazendo com que ela decaia em direção ao planeta até a colisão. O tempo de decaimento σ_{PR} de um partícula em órbita planetocêntrica com $I = 0$ pode ser estimado através da expressão (Burns et al. 1979):

$$\sigma_{PR} = 9.3 \times 10^6 \frac{R^2 \rho r}{Q_{PR}} \text{ anos} \tag{21}$$

onde R é a distância heliocêntrica do planeta em AU e ρ e r estão nas unidades de cgs. Para partículas muito pequenas, pode ser mais conveniente expressar da seguinte maneira:

$$\sigma_{PR} = 530 \frac{R^2}{\beta} \text{ anos} \tag{22}$$

O arrasto de Poynting-Robertson é responsável pela oscilação da excentricidade da partícula e possui o período da variação igual ao movimento médio do planeta ao redor do Sol (Hamilton & Krivov, 1996).

3.2 Algoritmo

Neste trabalho visamos criar uma biblioteca que possa ser utilizada por qualquer usuário em diferentes aplicações, então todos os algoritmos desenvolvidos aqui são para situações gerais e depende do usuário ao utilizar o Rebound definir as partículas no programa que serão aplicadas as forças.

Como discutido na seção 2.2.3, todas as forças aplicadas no Rebound devem ser decompostas em cada uma das coordenadas cartesianas, portanto para a força de radiação solar utilizamos as equações (18) - (20) e dividimos a componente da pressão de radiação Solar (RP) e a componente de Poynting-Robertson (PR) de maneira que o usuário possa aplicar uma componente ou a outra. Isto é feito através das variáveis inteiras $valitate\_rp$ e $validate\_pr\_drag$, que são parâmetros de entrada nas nossas funções e funcionam como variáveis booleanas onde 0 não considera a componente e 1 a componente é incluída na equação da força.

O nosso código para a pressão de radiação solar é:

```
1
2 struct F_rf{double x,y,z;}F_RF={0,0,0};
3         if(valitate_rp){
4                 F_RF.x = cos(ns*r->t);
5                 F_RF.y = cos(gama)*sin(ns*r->t);
6                 F_RF.z = sin(gama)*sin(ns*r->t);
7         }
```

Onde *gama* é a obliquidade do planeta definida pelo usuário, *ns* é o movimento médio da fonte de radiação. Já a componente PR:

```
1
2 struct F_drag {double x,y,z;}F_DRAG={0,0,0};
3  if(validate_pr_drag){
4    F_DRAG.x =   ((rad_source.x/rsp)*(rad_source.x/rsp)*(rad_source.
        vx + prvx)/c) + ((rad_source.vx + prvx)/c);
5    F_DRAG.y =   ((rad_source.y/rsp)*(rad_source.y/rsp)*(rad_source.
        vy + prvy)/c) + ((rad_source.vy + prvy)/c);
6    F_DRAG.z =   ((rad_source.z/rsp)*(rad_source.z/rsp)*(rad_source.
        vz + prvz)/c) + ((rad_source.vz + prvz)/c);
7  }
```

sendo *rad_source* uma partícula da estrutura do Rebound *reb_particle* que contém as informações da fonte de radiação, as variáveis com prefixo *prv* é a velocidade relativa da partícula ao planeta, *rsp* é o modulo do vetor do planeta a fonte de radiação e *c* é a velocidade da luz.

A equação total da força de radiação é dada por:

```
1 const double F_r = betaparticles*G*rad_source.m/(rsp*rsp);
2 ...
3 for (int i=0;i<N;i++){
4         struct reb_particle* particles = r->particles;
5         if (p.m!=0.) continue;
6                 ...
7                 particles[i].ax += F_r*(F_RF.x  - F_DRAG.x);
8                 particles[i].ay += F_r*(F_RF.y  - F_DRAG.y);
9                 particles[i].az += F_r*(F_RF.z  - F_DRAG.z);
10        }
11 }
```

A variável *betaparticles* é a relação adimensional da força de gravidade para a radiação solar expressada na equação (8) e deve ser definida pelo usuário. Como pode ser observado na linha 5 do código acima que faz com que a força de radiação solar seja aplicada apenas nas partículas de poeira (sem massa), portanto mesmo que o usuário adicione a estrutura do Rebound partículas com massa, como planetas e satélites perturbativos ao sistema, a força de radiação não será aplicada nestes corpos.

Por fim, em nossa biblioteca disponibilizamos ao usuário duas opções para aplicar a força de radiação solar, uma no sistema heliocêntrico e outra no sistema planetocêntrico através das funções *force_rp_prdrag_h* e *force_rp_prdrag_p*, respectivamente. As opções de escolha dependem da aplicação do usuário, mas os sistemas planetocêntricos sempre terão seu processamento mais rápido.

A funções são disponibilizadas com os seguintes parâmetros de entrada:

```
1 void force_rp_prdrag_h(double betaparticles, double gama, struct
     reb_simulation* r, int valitate_rp, int validate_pr_drag);
2
3 void force_rp_prdrag_p(double m_source, double rsp, double
     betaparticles, double gama, struct reb_simulation* r, int
     valitate_rp, int validate_pr_drag);
```

Para utilizar estas funções no Rebound é necessário criar uma função de *callback* para atribuir as forças adicionais, como explicado na seção 2.2.3, e dentro dela chamar as funções da nossa biblioteca. Por exemplo:

```
1  double betaparticles;
2  const double gama ;
3  double rsp;
4  int main(int argc, char* argv[]){
5          struct reb_simulation* r = reb_create_simulation();
6          r->force_is_velocity_dependent = 1;
7          r->additional_forces = forces;
8          ...
9  }
10 void forces(struct reb_simulation* r){
11         force_rp_prdrag_p(m_star,rsp,betaparticles,gama,r);
12 }
```

A criação desta função é indispensável pois nossas funções possuem argumentos diferentes da *struct reb_simulation**, então não podemos atribuir diretamente em *additional_forces*, mas isto deve ser feito em qualquer sistema em que se deseja adicionar mais de uma função de força, como será realizado na seção 4.

4 OS EFEITOS DO ACHATAMENTO PLANETÁRIO

Nesta seção iremos estudar como o efeito do achatamento planetário afeta as órbitas das partículas e então quando aplicarmos as forças perturbativas, na seção 5, estudaremos como o achatamento planetário modifica a órbita das partículas que já estão sob a influência da força de radiação.

No caso de forças perturbativas como estas nós não estamos interessados em detalhar as informações de como as posições e velocidades das partículas que orbitam o corpo oblato se modificam ao longo do tempo, mas apenas como as características de suas órbitas se alteram.

O caso de uma órbita em torno de um planeta oblato, a solução da órbita média para este problema é simplesmente uma elipse em precessão (Danby, 1988). A órbita mantém o seu tamanho, forma e inclinação do plano equatorial enquanto o nodo regressa e o pericentro precessa, cada um com uma taxa constante. Quando calculamos estas taxas usando os elementos orbitais, podemos facilmente mostrar o comportamento da partícula.

Em Hamilton (1993) ele notou que os elementos osculadores diferem ligeiramente dos elementos geométricos que descrevem a verdadeira forma da órbita. Estas derivações para um corpo oblato a discrepância é da ordem de J_2.

4.1 Introdução Teórica

O tratamento das perturbações orbitais decorrentes em função do campo gravitacional de um corpo oblato pode ser encontrado em muitos textos, porém vamos fazer uma pequena discussão deles nesta seção, para poder fazer uma complementação e prover alguns exemplos.

Consideremos um planeta oblato, cujos valores são subscritos pela letra "p", orbitado por um corpo de massa m_c e coordenadas (r, ϕ, α) de acordo com o centro do planeta, onde r é a distância radial, ϕ é a longitude e α é a latitude do corpo. Pode-se mostrar através da teoria de potencial que o potencial gravitacional aplicado no corpo pode ser escrito como:

$$V_{GR} = -\frac{Gm_p}{r}\left[1 - \sum_{j=2}^{\infty} J_j \left(\frac{R_p}{r}\right)^j P_j(\text{sen}\alpha)\right] \tag{23}$$

onde $P_j(x)$ são os polinômios de Legendre e J_j são coeficientes adimensionais que caracterizam o tamanho das componentes do potencial não esférico. Se i é par então J_i são chamados de coeficientes da zona harmônica.

Note que foi assumido que V é uma função de r e α apenas, e que assumimos a prática normal tendo o potencial negativo. A ausência de ϕ em V significa que o potencial do planeta é considerado como sendo axissimétrico. Esta é uma aproximação razoável porque o fato do planeta ser não esférico é devido à seu achatamento ao longo do eixo de rotação.

Murray & Dermott obtiveram soluções que mostram que o movimento sob o ponto de massa e polinômio par $J_{(2i)}$ no potencial dá origem a três frequências: o movimento médio, n, a frequência radial, k, e a frequência vertical, ν.

$$n^2 = \frac{Gm_p}{a^3}\left[1+\frac{3}{2}J_2\left(\frac{R_p}{a}\right)^2-\frac{15}{8}J_4\left(\frac{R_p}{a}\right)^4\right] \tag{24}$$

$$k^2 = \frac{Gm_p}{a^3}\left[1+\frac{3}{2}J_2\left(\frac{R_p}{a}\right)^2+\frac{45}{8}J_4\left(\frac{R_p}{a}\right)^4\right] \tag{25}$$

$$\nu^2 = \frac{Gm_p}{a^3}\left[1+\frac{9}{2}J_2\left(\frac{R_p}{a}\right)^2-\frac{75}{8}J_4\left(\frac{R_p}{a}\right)^4\right] \tag{26}$$

Note que se $J_2 = J_4 = 0$ então $n^2 = k^2 = \nu^2 = n_0^2$, onde $n_0 = (Gm_p/a^3)^{\frac{1}{2}}$ é o movimento médio kepleriano do corpo em torno do ponto de massa do planeta. No caso do movimento médio, n, a inclusão de termos adicionas significa que para um dado semi-eixo maior o corpo se movimenta mais rápido do que a taxa esperada naquela localização se o movimento é puramente kepleriano. Portanto, uma vez que a quantidade observável para um corpo é geralmente n, o semi-eixo maior não é aquele determinado pela terceira lei de Kepler. Ao invés disto é necessário resolver a equação (24), a equação não linear em a.

A inclusão dos termos pares do polinômio $J_{(2i)}$ e as pequenas diferenças resultantes nas três frequências é que a órbita não é mais fechada e que a longitude do pericentro e do nodo ascendente já não são fixas. De fato existem algumas taxas de precessão:

$$\dot{\varpi} = n - k \tag{27}$$

$$\dot{\Omega} = n - \nu \tag{28}$$

Para $O(R_p/a)^4$ temos:

$$\dot{\varpi} = +n_0 \left[\frac{3}{2} J_2 \left(\frac{R_p}{a} \right)^2 - \frac{15}{4} J_4 \left(\frac{R_p}{a} \right)^4 \right] \tag{29}$$

$$\dot{\Omega} = -n_0 \left[\frac{3}{2} J_2 \left(\frac{R_p}{a} \right)^2 - \frac{9}{4} J_2^2 \left(\frac{R_p}{a} \right)^4 - \frac{15}{4} J_4 \left(\frac{R_p}{a} \right)^4 \right] \tag{30}$$

Podemos também usar as equações de Lagrange para derivar as expressões para $\dot{\varpi}$ e $\dot{\Omega}$. Para a segunda ordem em e e I :

$$\dot{\varpi} = +n \left[\frac{3}{2} J_2 \left(\frac{R_p}{a} \right)^2 - \frac{9}{8} J_2^2 \left(\frac{R_p}{a} \right)^4 - \frac{15}{4} J_4 \left(\frac{R_p}{a} \right)^4 \right] \tag{31}$$

$$\dot{\Omega} = -n \left[\frac{3}{2} J_2 \left(\frac{R_p}{a} \right)^2 - \frac{27}{8} J_2^2 \left(\frac{R_p}{a} \right)^4 - \frac{15}{4} J_4 \left(\frac{R_p}{a} \right)^4 \right] \tag{32}$$

Note a diferença entre as equações (30) e (31) por que foi utilizado n no lugar de n_0 fora dos colchetes.

Portanto estas frequências irão modular os elementos orbitais das partículas e também resultaram em uma força que a partícula orbitando um corpo oblato sentirál. A força perturbativa é obtida tomando a gradiente:

$$\vec{F}_{GR} = -m_g \nabla V_{GR} \tag{33}$$

A força aplicada em cada uma das direções x,y e z, primeiramente iremos calcular para J_2 e então para J_4. Portanto aplicando na equação (23) temos que:

$$-\overrightarrow{F}_{GR} = -Gm_c \nabla \left[\frac{1}{r} \left(1 - J_2 \left(\frac{R_p}{r} \right)^2 P_2(\text{sen}\alpha) \right) \right] \tag{34}$$

Vamos continuar os cálculos agora apenas para a componente x da força. Considerando $r = (x^2 + y^2 + z^2)^{1/2}$ e o polinômio de Legendre onde $P_2(x) = 1/2(3x^2 - 1)$ e sabendo que $\sin\alpha = z/r$ temos que:

$$-F_x = -Gm_c \frac{\partial}{\partial x}\left[\frac{1}{r}\left(1+\frac{1}{2}J_2\left(\frac{R_p}{r}\right)^2 - \frac{3}{2}J_2\frac{R_p^2}{r^4}z^2\right)\right] \tag{35}$$

Portanto ao realizar a derivada parcial encontramos a força em x:

$$F_x = -\frac{Gm_c}{r^3}\left[1+\frac{3}{2}J_2\frac{R_p^2}{r^2} - \frac{15}{2}J_2\frac{R_p^2}{r^4}z^2\right]x \tag{36}$$

Realizando o mesmo processo para a componente y da força, encontramos:

$$F_y = -\frac{Gm_c}{r^3}\left[1+\frac{3}{2}J_2\frac{R_p^2}{r^2} - \frac{15}{2}J_2\frac{R_p^2}{r^4}z^2\right]y \tag{37}$$

E para encontrar a componente em z apenas tomando cuidado com a derivada parcial em z, temos:

$$F_z = -\frac{Gm_c}{r^3}\left[1+\frac{9}{2}J_2\frac{R_p^2}{r^2} - \frac{15}{2}J_2\frac{R_p^2}{r^4}z^2\right]z \tag{38}$$

Agora vamos desenvolver as contas para J_4, onde a força é dada por:

$$-\overrightarrow{F}_{GR} = -Gm_c\nabla\left[\frac{1}{r}\left(1-J_2\left(\frac{R_p}{r}\right)^2 P_2(\text{sen}\alpha) - \text{J}_4\left(\frac{\text{R}_\text{p}}{\text{r}}\right)^4 \text{P}_4(\text{sen}\alpha)\right)\right] \tag{39}$$

no qual o polinômio de Legendre $P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$. Por enquanto vamos considerar os valores dependes apenas J_4, então a componente x da força é dada por:

$$F_{x_{J4}} = -Gm_c\frac{\partial}{\partial x}\left[\frac{1}{r}\left(-\frac{3}{8}J_4\left(\frac{R_p}{r}\right)^4 + \frac{30}{8}J_4\frac{R_p^4}{r^6}z^2 - \frac{35}{8}J_4\frac{R_p^4}{r^8}z^4\right)\right] \tag{40}$$

fazendo a derivada parcial :

$$F_{x_{J4}} = -\frac{Gm_c}{r^3}\left[-\frac{15}{8}J_4\left(\frac{R_p}{r}\right)^4 + \frac{210}{8}J_4\frac{R_p^4}{r^6}z^2 - \frac{315}{8}J_4\frac{R_p^4}{r^8}z^4\right]x \tag{41}$$

E para as componentes em y e z:

$$F_{y_{J4}} = -\frac{Gm_c}{r^3}\left[-\frac{15}{8}J_4\left(\frac{R_p}{r}\right)^4 + \frac{210}{8}J_4\frac{R_p^4}{r^6}z^2 - \frac{315}{8}J_4\frac{R_p^4}{r^8}z^4\right]y \tag{42}$$

$$F_{z_{J4}} = -\frac{Gm_c}{r^3}\left[-\frac{75}{8}J_4\left(\frac{R_p}{r}\right)^4 + \frac{350}{8}J_4\frac{R_p^4}{r^6}z^2 - \frac{315}{8}J_4\frac{R_p^4}{r^8}z^4\right]z \tag{43}$$

Portando a força total que uma partícula experimenta ao orbitar um corpo oblato em cada uma das componentes considerando J_2 e J_4 é dada por:

$$F_x = -\frac{Gm_c}{r^3}\left[1 + \frac{3}{2}J_2\frac{R_p^2}{r^2}\left(1 - 5\frac{z^2}{r^2}\right) - \frac{15}{8}J_4\frac{R_p^4}{r^4}\left(1 - 14\frac{z^2}{r^2} + 21\frac{z^4}{r^4}\right)\right]x \tag{44}$$

$$F_y = -\frac{Gm_c}{r^3}\left[1 + \frac{3}{2}J_2\frac{R_p^2}{r^2}\left(1 - 5\frac{z^2}{r^2}\right) - \frac{15}{8}J_4\frac{R_p^4}{r^4}\left(1 - 14\frac{z^2}{r^2} + 21\frac{z^4}{r^4}\right)\right]y \tag{45}$$

$$F_z = -\frac{Gm_c}{r^3}\left[1 + \frac{3}{2}J_2\frac{R_p^2}{r^2}\left(3 - 5\frac{z^2}{r^4}\right) - \frac{5}{8}J_4\frac{R_p^4}{r^4}\left(25 - 70\frac{z^2}{r^2} + 63\frac{z^4}{r^4}\right)\right]z \tag{46}$$

4.2 Algoritmo

Para implementarmos os algoritmos para o achatamento planetário devemos lembrar que o Rebound já possui módulos gravitacionais, explicados na seção **??**, onde já existe um cálculo da força gravitacional padrão na integração do Rebound. Os termo independentes de J_2 e J_4 nas equações (44) - (46) são referentes a força gravitacional de um corpo perfeitamente esférico que o Rebound já efetua os cálculos por padrão, portanto em nossos algoritmos iremos desconsiderar os termos da força gravitacional de um corpo esférico. As equações da força para cada uma das componentes cartesianas ficam da seguinte forma:

$$F_x = -\frac{3}{2}\frac{Gm_c}{r^5}R_p^2\left[J_2\left(1 - 5\frac{z^2}{r^2}\right) - \frac{5}{4}J_4\frac{R_p^2}{r^2}\left(1 - 14\frac{z^2}{r^2} + 21\frac{z^4}{r^4}\right)\right]x \tag{47}$$

$$F_y = -\frac{3}{2}\frac{Gm_c}{r^5}R_p^2\left[J_2\left(1 - 5\frac{z^2}{r^2}\right) - \frac{5}{4}J_4\frac{R_p^2}{r^2}\left(1 - 14\frac{z^2}{r^2} + 21\frac{z^4}{r^4}\right)\right]y \tag{48}$$

$$F_z = -\frac{3}{2}\frac{Gm_c}{r^5}R_p^2\left[J_2\left(1-5\frac{z^2}{r^2}\right)-\frac{5}{12}J_4\frac{R_p^2}{r^2}\left(25-70\frac{z^2}{r^2}+63\frac{z^4}{r^4}\right)\right]z \tag{49}$$

O nosso algoritmo realiza as equações acima por partes, primeiramente para J_2 e então para J_4, assim caso o usuário queira restringir a suas contas ele não gastará processamento, pois é calculado os valores apenas para as componentes que ele deseja. Isto pode ser observado através dos condicionais na função a seguir, que possui os argumentos do planeta (*struct reb_particle planet*), os valores de J2 e J4 (*double J2planet, double J4planet*) e a estrutura do Rebound (*struct reb_simulation* r*).

```
struct F_J2 {double x,y,z;}F_J2={0,0,0};
struct F_J4 {double x,y,z;}F_J4={0,0,0};

for (int i=1;i<r->N;i++){
  ...
  if(r->particles[i].hash==planet.hash){break;}
  ...
  const double fac  = (r->G*planet.m/pow(rp,5))*((1.5)*planet.r*
     planet.r);

  if(J2planet!=0){
    f_J2.x  = J2planet*fac*prx*(1 - (4.*prz*prz/pow(rp,2)));
    f_J2.y  = J2planet*fac*pry*(1 - (4.*prz*prz/pow(rp,2)));
    f_J2.z  = J2planet*fac*prz*(3 - (5.*prz*prz/pow(rp,2)));
  }

  if(J4planet!=0){
    f_J4.x  = J4planet*fac*prx*(5./4.)*(pow(planet.r/rp,2))*(1 -
       (14.*prz*prz)/pow(rp,2) + (21.*pow(prz,4))/pow(rp,4));
    f_J4.y  = J4planet*fac*pry*(5./4.)*(pow(planet.r/rp,2))*(1 -
       (14.*prz*prz)/pow(rp,2) + (21.*pow(prz,4))/pow(rp,4));
    f_J4.z  = J4planet*fac*prz*(5./12.)*(pow(planet.r/rp,2))*(25 -
       (70.*prz*prz)/pow(rp,2) + (63.*pow(prz,4))/pow(rp,4));
  }

       r->particles[i].ax += F_J2.x + F_J4.x;
       r->particles[i].ay += F_J2.y + F_J4.y;
       r->particles[i].az += F_J2.y + F_J4.z;
}
```

onde as variáveis com prefixo *pr* é a distância relativa da partícula ao planeta oblato e *rp* é o módulo do vetor do planeta à partícula.

Também adicionamos a biblioteca mais duas funções para calcular a taxa de variação do pericêntro e da longitude do nodo ascendente (equações 31 e 32).

Para $\dot{\varpi}$ a função foi escrita da seguinte maneira:

```
double feg_get_rate_pericentre(struct reb_particle p, struct
   reb_particle planet, struct reb_simulation* r, double J2planet,
   double J4planet){

   struct reb_orbit o = reb_tools_particle_to_orbit(r->G,p,planet);
   const double prefix = -o.n*(1.5)*pow(planet.r/o.a,2);
   const double omega = prefix*(J2planet - (2.5)*J4planet*pow(
      planet.r/o.a,4));

   return omega;
}
```

e para $\dot{\Omega}$:

```
double get_rate_nodo(struct reb_particle p, struct reb_particle
   planet, struct reb_simulation* r, double J2planet, double
   J4planet){

  struct reb_orbit o = reb_tools_particle_to_orbit(r->G,p,planet);
  const double prefix = -o.n*(1.5)*pow(planet.r,2)/pow(o.a,2);
  const double Omega = prefix*(J2planet - (2.25)*J2planet*J2planet*
     pow(planet.r,2)/pow(o.a,2) - (2.5)*J4planet*pow(planet.r,2)/
     pow(o.a,2));

  return Omega;
}
```

5 VALIDAÇÃO DOS MÓDULOS DE FORÇAS DISSIPATIVAS

Neste capítulo iremos validar os módulos de forças perturbativas criados anteriormente. Para isto iremos incluir no pacote do Rebound os métodos desenvolvidos e analisar os efeitos que alteram a órbita das partículas nos anéis μ e ν de Urano. Também consideramos os efeitos devido à presença de satélites próximos aos anéis e fizemos ainda alguns testes extras para um sistema diferente dos anéis planetários que é o sistema triplo asteroidal (87) Sylvia que é composto por duas pequenas luas: Romulus e Remus, no qual estudaremos a evolução orbital das pequenas luas sob a influência gravitacional de Sylvia.

Escolhemos estes sistemas porque já existem estudos nos quais poderemos comparar os nossos resultados para saber o quão eficaz são os nossos módulos criados. Em particular iremos fazer as comparações com os resultados encontrados em Sfair & Giuliatti Winter (2009), que utilizou o mesmo sistema com condições iniciais idênticas as deste trabalho, porém sua simulação numérica foi através do pacote Mercury utilizando o integrador com a variável de passo de tempo Bulirsch-Stoer (Chambers, 1999).

Para ambos os sistemas estudados faremos uma introdução, então será realizado algumas analises com o Rebound e durante a apresentação de cada resultado encontrado faremos as comparações com as bibliográficas.

5.1 Aplicação aos anéis μ e ν de Urano

Apresentaremos aqui algumas informações sobre o sistema que será estudado.

5.1.1 Introdução

Em 1781 o planeta Urano foi descoberto por Willian Herschel, entretanto o planeta já havia sido observado em 1690 por John Flamsteed, mas foi erroneamente catalogado como uma estrela, o mesmo havia sido feito o astrônomo francês Pierre Lemonnier em 1750.

Urano é o sétimo planeta do Sistema Solar, com a distância 19.19 UA do Sol. Em suas características físicas ele possui um raio de 25559 km e uma densidade de 1.318 g/cm^3 (Giorgini et. al., 1996). Primariamente sua atmosfera é composta de hidrogênio e hélio, com uma pequena quantidade de metano, que atribui coloração azulada do planeta (Fegley et. al., 1991).

Em 1986 importantes descobertas foram realizadas com a passagem da sonda Voyager II por Urano que descobriu novos satélites e anéis (Smith et. al., 1986).

5.1.1.1 Satélites

Urano possui vinte e sete satélites conhecidos (NASA & JPL, 2016). Os cinco maiores satélites são: Miranda, Ariel, Umbriel, Oberon e Titânia. Estes foram os primeiros satélites a serem descobertos. Os satélites menores foram descobertos pela passagem da Voyager II por Urano, satélites como Cordelia e Ophelia, próximos ao anel ε, os satélites da família Portia, formados por Bianca, Cressida, Desdemona, Juliet, Portia e Rosalind. Mais tarde outros pequenos satélites, como Cupid e Perdita, também da família Portia, foram descobertos fazendo uma análise mais detalhada das imagens da Voyager e por auxílio do Telescópio Espacial Hubble (HST).

Estamos especialmente interessados nos satélites menores, por estarem mais próximos aos anéis. Os satélites mais importantes para os estudos dos anéis μ e ν de Urano estão na tabela 1.

Tabela 1 - Elementos orbitais e parâmetros físicos dos satélites da simulação numérica.

	Mab	Puck	Rosalind	Portia
a(km)	97 735	86 004	69 926	66 097
e($\times 10^{-3}$)	2.54	0.39	0.58	0.51
i(^{0})	0.134	0.321	0.093	0.18
ω(^{0})	240.30	331.88	257.28	84.41
Ω(^{0})	350.74	199.48	157.64	77.71
r (km)	12	81	36	70

Fonte: Adaptado de Showalter & Lissauer (2006).

5.1.1.1.1 Mab e Puck

O satélite Puck, assim como Mab, não fazem parte da família Portia e é externo a todos os membros da família, mas ambos possuem grande influência sobre o anel μ. Ele é o maior dos satélites internos de Urano, com raio de aproximadamente 77 km.

Uma característica interessante de Puck é que seu brilho é maior em um dos lados. Este fato pode estar relacionado à interação entre Puck e as partículas do anel μ, pois o satélite está localizado na borda interna do anel (Showalter & Lissauer 2006).

5.1.1.2 Anéis

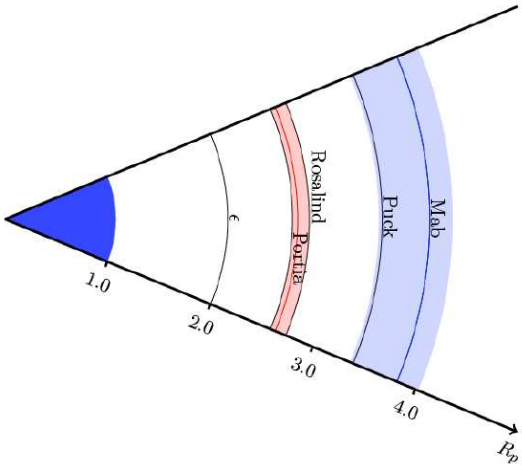
Em março de 1977, a ocultação de uma estrela por Urano revelou a existência de quatro anéis estreitos (α ,β, γ e δ) e um anel mais largo e inclinado (anel ε), o primeiro conjunto com largura de aproximadamente 10 km e o outro anel com $\sim$100km. Através de observações posteriores, foram identificados mais quatro anéis: η, 4, 5 e 6. Estes anéis em conjunto com os anéis ζ e λ descobertos pela Voyager II (Smith et al. 1986), formam o sistema principal de anéis de Urano. Com uma reobservação de todo o sistema de Urano por Showalter & Lissauer (2006) e utilizando o Telescópio Espacial Hubble

foi revelado a existência de dois novos anéis, μ e ν.

Além destes anéis, há indícios de outras estruturas similares a anéis tênues em regiões mais próximas ao planeta, mas devido ao ruído na imagem não foi possível obter nenhuma confirmação até o momento.

Tanto o anel μ quanto o ν estão situados além do anel ε, em uma região de baixa profundidade óptica. A figura (3) mostra de forma esquemática a localização dos anéis e de alguns membros da família de Portia.

Figura 3 - Representação esquemática de μ e ν. Os picos radiais dos anéis estão representados pelas linhas azul (μ) e vermelha (ν) e a extensão radial de cada um pela região sombreada. As posições dos satélites próximos também estão indicadas, assim com o anel ε. A escala é dada em unidades de raios de Urano (Rp).



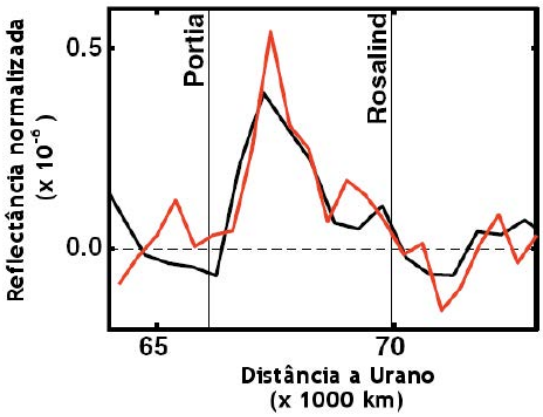
Fonte: Sfair & Giuliatti Winter(2009)

Para grandes valores do ângulo de fase os anéis parecem bastante brilhantes, indicando que eles são compostos basicamente por partículas com tamanho da ordem de micrômetros. Observações no infravermelho, utilizando a óptica adaptativa do telescópio Keck, revelaram que o anel μ é azul, enquanto ν é extremamente vermelho (de Pater. et al. 2006b).

5.1.1.2.1 Anel ν

O anel ν é acompanhado por dois satélites, Portia próximo à borda interna e Rosalind à externa. Apresenta um perfil triangular não simétrico (figura 4), similar ao anel μ, apesar de não existir nenhum satélite imerso. Existem ainda evidências de variações de acordo com a longitude: em algumas imagens são vistos aglomerados de partículas (“clumps") em diferentes regiões do anel.

Figura 4 - Perfis radiais do anel ν a partir dos dados do HST (preto) e da Voyager (vermelho). As órbitas de Portia e Rosalind estão indicadas por linhas verticais. Adaptado de Showalter & Lissauer (2006).



Fonte: Sfair & Giuliatti Winter(2009)

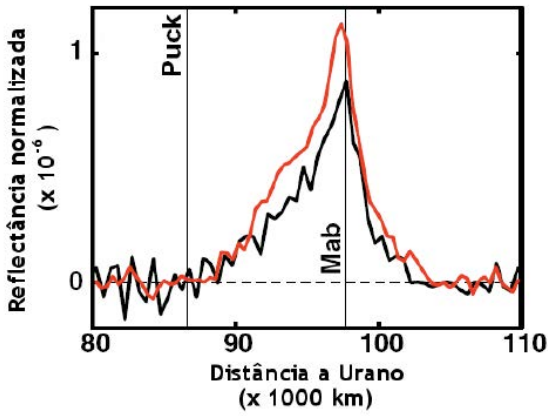
O anel apresenta espectros vermelhos, que este relacionado às propriedades das partículas de poeira, quando são pequenas tendem a espalhar a luz incidente cujo comprimento de onda é comparável ao tamanho do grão e esse espalhamento ocorre preferencialmente para grandes ângulos de fase. Esta característica foi comparada ao anel G de Saturno por Sfair & Giuliatti Winter (2009) argumentando que as partículas tem provavelmente tamanhos entre 1 e 10 μm.

Uma resolução feita por Showalter & Lissauer (2006) é de que provavelmente não há nenhum satélite de resolução limite 5km, que foi analisada, ou maior na região do anel ν. Porém isso não exclui a possibilidade da existência de inúmeros corpos menores.

5.1.1.2.2 Anel μ

O anel μ apresenta um perfil radial triangular bem definido (Showalter & Lissauer, 2006), figura (5). É possível observar que o pico do anel coincide com a órbita de Mab e o satélite Puck esta localizado na borda interna do anel.

Figura 5 - Perfis radiais do anel ν a partir dos dados do HST (preto) e da Voyager (vermelho). Os semi-eixos maiores de Puck e Mab estão indicados com linhas verticais. Adaptado de Showalter & Lissauer (2006).



Fonte: Sfair & Giuliatti Winter(2009)

Um mecanismo de impactos de micrometeoros ocorre com Mab e pode gerar o material que forma o anel, Sfair & Giuliatti Winter (2012). A taxa de produção de partículas está relacionada à velocidade de impacto, ao fluxo de impactantes e ao tamanho do satélite impactado. Para objetos esféricos a velocidade do material ejetado é comparável com a velocidade de escape e isso faz com que exista um tamanho ótimo para a eficiência na produção de partículas, que corresponde a satélites com raio de $\sim$ 10 km, valor próximo ao diâmetro de Mab.

5.1.1.2.3 Características gerais dos anéis μ e ν

Os perfis radiais dos anéis ν e μ (figuras 4 e 5) sugerem que ambos possuem uma dinâmica parecida, e sabe-se que estas partículas estão sujeitas à várias perturbações, como o achatamento do planeta, forças eletromagnéticas, arrasto atmosférico e força de radiação solar.

As perturbações citadas são resultados de forças diversas com origem diferente e cada uma provoca uma alteração diferente nas órbitas das partículas. Entretanto nenhuma dessas forças domina a evolução dos anéis μ e ν, pois neste caso o perfil radial encontrado não seria triangular. Neste caso particular, fora da distribuição radial, o comportamento é refletido em uma assimetria na distribuição radial do anel que indicaria a ação de uma força dissipativa com uma direção preferencial.

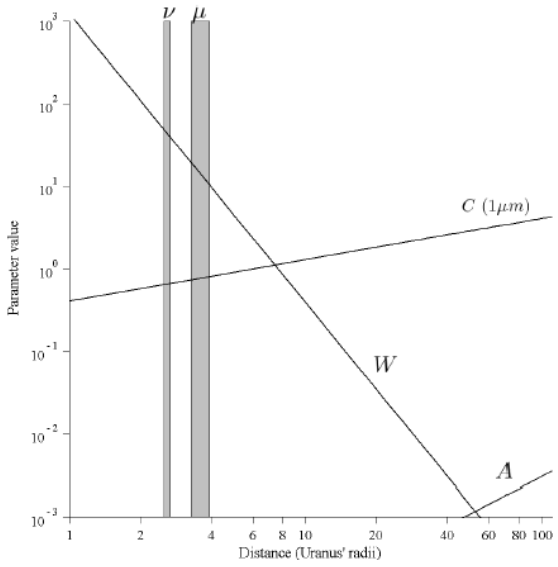
A limitação da extensão radial dos anéis μ e ν pode ser atribuída a outros três satélites que estão localizados nas bordas dos anéis. Entretanto, as bordas dos anéis por não terminarem de maneira abrupta nas órbitas dos satélites, indicando a existência de um mecanismo contínuo de remoção de partículas, mas sem a dominação de uma força dissipativa específica. O destino mais provável para o material dos anéis são estes satélites próximos.

5.1.2 Forças Perturbativas

Os anéis μ and ν são tênues, mas apresentam um grande espalhamento de luz para grandes ângulos de fase, o que pode indicar que são formados predominantemente por partículas micrométricas. Partículas deste tamanho estão sujeitas à ação de várias forças perturbativas, como o achatamento do planeta, a radiação solar e a maré solar. Estas forças serem muito menores que a força gravitacional do planeta, entretanto elas alteram a energia orbital da partícula e podem mudar significativamente a evolução ao longo do tempo.

O artigo de Hamilton & Krivov (1996) apresenta um método de comparação da intensidade de cada uma das forças perturbativas mencionadas através de parâmetros adimensionais. Em Sfair & Giuliatti Winter (2009) foi calculado os parâmetros de força de maré solar (A), pressão de radiação (C) e devido ao achatamento do planeta (W) para o primeiro coeficiente da expansão (J_2) para uma partícula esférica com densidade de 1 g/cm^3 e raio de 1 μ m ao redor de Urano, assumindo que a partícula é composta por um material ideal (figura 6).

Figura 6 - Parâmetros adimensionais das forças para uma partícula ao redor de Urano em função da distância ao planeta. As barras verticais indicam a localização e extensão radial dos anéis μ e ν.



Fonte: Sfair & Giuliatti Winter(2009)

O parâmetro maré solar é definido por:

$$A = \frac{3}{4}\frac{n_s}{n} \tag{50}$$

onde n_s é o movimento médio do planeta ao redor do Sol e n é o movimento médio da partícula.

O parâmetro da força causada pelo achatamento planetário W:

$$W = \frac{3}{2} J_2 \left(\frac{R_p}{a} \right)^2 \frac{n_s}{n} \tag{51}$$

O parâmetro da pressão de radiação C:

$$C = \frac{3}{2} \frac{n_s}{n} \sigma \tag{52}$$

onde σ é a razão entre a força da pressão de radiação e a força gravitacional do planeta para uma órbita circular. A constante σ pode ser escrita como:

O parâmetro da pressão de radiação C:

$$\sigma = \frac{3}{4} Q_{pr} \frac{Fa^2}{GM_p c \rho s} \sigma \tag{53}$$

onde Q_{pr} é uma constante relacionada à eficiência da partícula em absorver e espalhar a radiação incidente, F é o fluxo solar que chega ao planeta, ρ é densidade da partícula cujo raio é s e c é a velocidade da luz.

Através da figura (6) é possível observr que a maré solar é relevante apenas para partículas distantes do planeta (acima de 50 raios de Urano), de maneira que pode ser ignorada na região dos anéis de poeira analisados. Porém, os efeitos da radiação solar e do achatamento de Urano são apreciáveis e devem ser levados em consideração. Nas seções seguintes é apresentado o resultado de um estudo numérico sobre a evolução orbital de várias partículas perturbadas por estas duas forças e pela interação gravitacional com os satélites próximos aos anéis.

5.1.3 Aplicação da força de radiação solar

Para validar o nosso código iremos aplicar a força de radiação solar para o caso dos anéis μ e ν de Urano e iremos comparar nossos resultados com Sfair & Giuliatti Winter (2009) que utilizaram o mesmo sistema com condições iniciais idênticas as deste trabalho, porém sua simulação numérica foi através do pacote Mercury utilizando o integrador Bulirsch-Stoer (Chambers, 1999).

Em nossa simulação utilizamos o integrador numérico de 15^a ordem IAS15 e avaliamos a evolução numérica de uma amostra de partículas nos anéis μ e ν influenciadas pelo campo gravitacional de Urano e perturbadas pela força de radiação solar.

Os anéis de Urano são dominados por poeira micrométrica, então assumimos partículas esféricas de tamanho 1, 3, 5, e 10 μm com uma densidade de 1 g/cm^3 (gelo sólido puro) em uma órbita circular em torno de Urano. Primeiramente integramos o sistema utilizando a função de radiação solar para um sistema planetocêntrico com o Sol em uma órbita circular em torno do planeta e com o movimento médio

n_s constante.

A tabela (2) (adaptada de Showalter & Lissauer, 2006) resume as informações dos raios orbitais para os anéis μ e ν. Os parâmetros utilizados de Urano(raio, massa e semieixo maior) são derivadas de Murray & Dermott.

Tabela 2 - Raios dos anéis μ e ν de Urano.

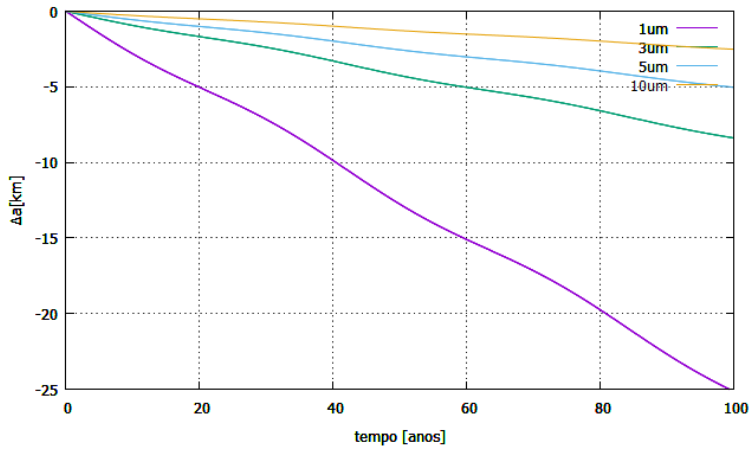
Ring	Borda Externa (km)	Borda Interna (km)	Pico do Raio (km)	Largura (km)
μ	86 000	103 000	97 700	17 000
ν	66 100	69 900	67 300	3 800

Fonte: Adaptado de Showalter & Lissauer (2006).

Nós analisamos os efeitos de cada componente da força de radiação solar nos elementos orbitais da partícula. Como esperado o estudo numérico mostrou que cada componente da força de radiação solar é responsável por um efeito distinto na órbita da partícula.

Primeiramente consideramos apenas a componente do arrasto de Poyting-Robertson para partículas de diferentes tamanhos, mas condições iniciais identicas. Nesta simulação não consideramos o Sol como uma partícula efetiva na simulação, apenas foram realizados os calculos refentes a posição relativa do Sol até Urano, como se o planeta tivesse orbitando com excentricidade zero. Portanto, Urano é o corpo central desta simulação. Para o anel μ elas estão localizadas no pico do anel e a figura (7) mostra que este efeito causa uma decaimento no semi-eixo maior causado por uma perda de energia da partícula.

Figura 7 - Variação do semi-eixo maior devido ao arrasto de Poynting-Robertson para partículas de diferentes tamanhos no anel μ de Urano com condições iniciais idênticas. $\Delta a = 0$ corresponde ao semi-eixo maior inicial da partícula.

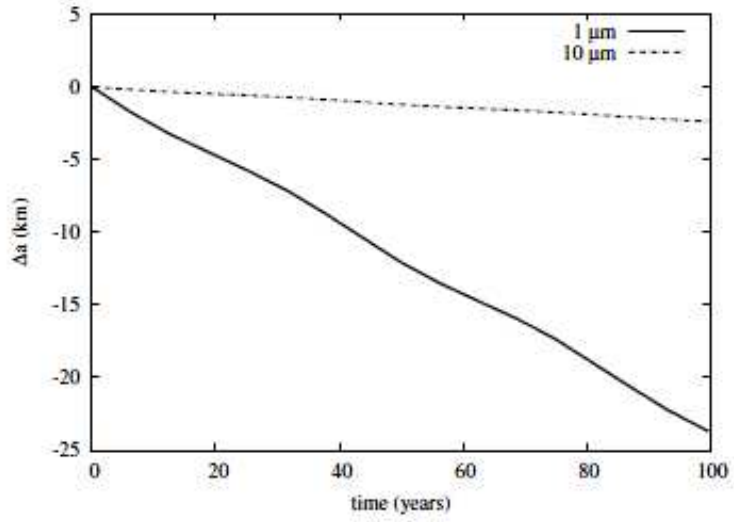


Fonte: Produção do próprio autor

A taxa de decaimento depende do tamanho da partícula, para uma partícula de $1\mu m$ localizada em $a = 97500km$ (aproximadamente no pico do anel μ) ela decai 2.5 km em aproximadamente 100 anos.

Com uma simples extrapolação é possível ver que o tempo necessário para que uma partícula deste tamanho colida com o planeta é de 3.9×10^6 anos, enquanto uma partícula de $10\mu m$ demora aproximadamente 3.9×10^5 anos para colidir com Urano. Estes dois valores são da mesma ordem de grandeza daqueles obtidos através da equação (21), porém os valores desviaram certa de 15% na simulação com Rebound enquanto os valores com a obtidos por Sfair & Giuliatti Winter (2009) desviaram apenas 5%, como pode ser observado na figura (8) retirada do artigo.

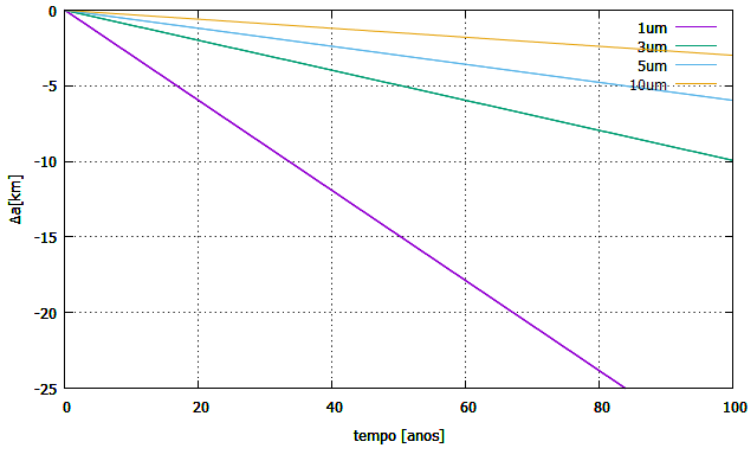
Figura 8 - Variação do semi-eixo maior devido ao arrasto de Poynting-Robertson para partículas de $1\mu m$ e $10\mu m$ nos anel μ de Urano com condições iniciais idênticas. $\Delta a = 0$ corresponde ao semi-eixo maior inicial da partícula.



Fonte: Sfair & Giuliatti Winter (2009)

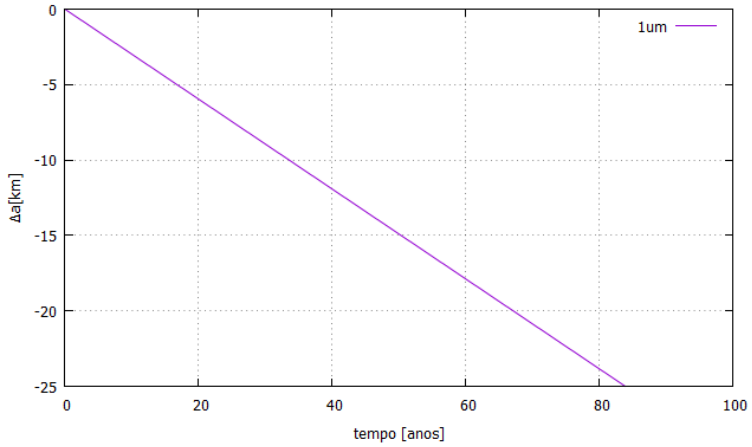
Ao analisar a partícula no anel com as mesmas condições iniciais, mas agora incluindo o Sol como uma partícula na simulação, onde ele é o corpo central, não mais o planeta, obtivemos um resultado mais satisfatório(figura 9), pois os valores desviram cerca de 1% daquelas calculado na equação (21). Para estas simulações utilizamos o integrador Hermes, porque seu processamento é muito mais rápido do que com o integrador IAS15 e a diferença entre eles foi muito sútil, como pode ser observado na figura (10) para uma partícula 1μm, o integrador Hermes gastou certa de 12.6min enquanto o IAS15 gastou de 3h para processar o mesmo sistema.

Figura 9 - Variação do semi-eixo maior devido ao arrasto de Poynting-Robertson para partículas de diferentes tamanhos no anel μ de Urano com condições iniciais idênticas. $\Delta a = 0$ corresponde ao semi-eixo maior inicial da partícula. Foi utilizado o integrador Hermes.



Fonte: Produção do próprio autor

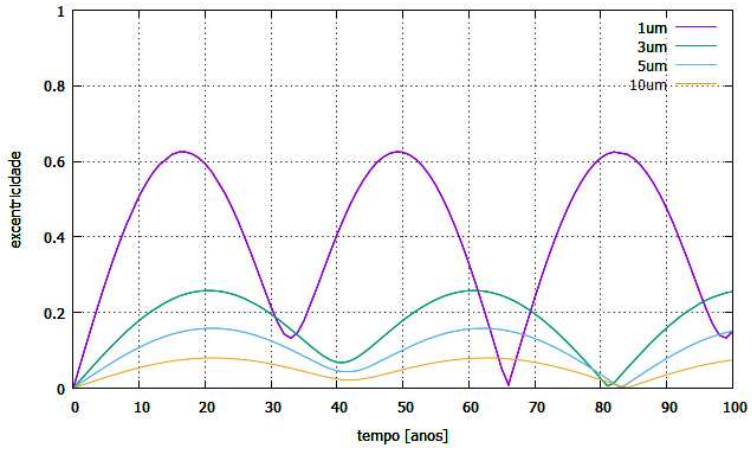
Figura 10 - Variação do semi-eixo maior devido ao arrasto de Poynting-Robertson para partículas de tamanho $1\mu m$ no anel μ de Urano. $\Delta a = 0$ corresponde ao semi-eixo maior inicial da partícula. Foi utilizado o integrador IAS15.



Fonte: Produção do próprio autor

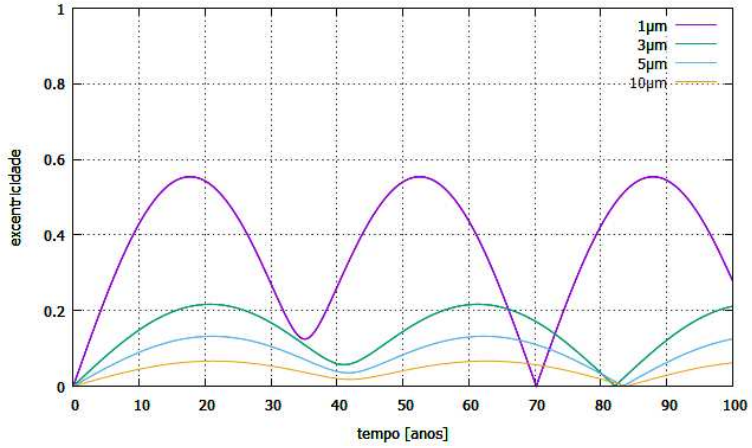
Consideramos então apenas as componentes da pressão de radiação solar (RP). Este efeito causa uma variação na excentricidade das partículas, como pode ser visto na figura (11) para o anél μ e figura (12) para ν . Para as maiores partículas o período de oscilação da excentricidade é aproximadamente igual ao período orbital do planeta ($\sim$ 84anos). A excentricidade de partículas de 1 μm atinge valores maiores que 0.5, que podem cruzar a órbita de satélites próximos que podem eventualmente colidir com algum deles. Estes resultados estão de acordo com os encontrados por Sfair & Giuliatti Winter (2009) (figura 13).

Figura 11 - Variação na excentricidade devido a pressão de radiação solar para uma partícula de diferentes tamanhos no anel μ de Urano com condições iniciais idênticas.



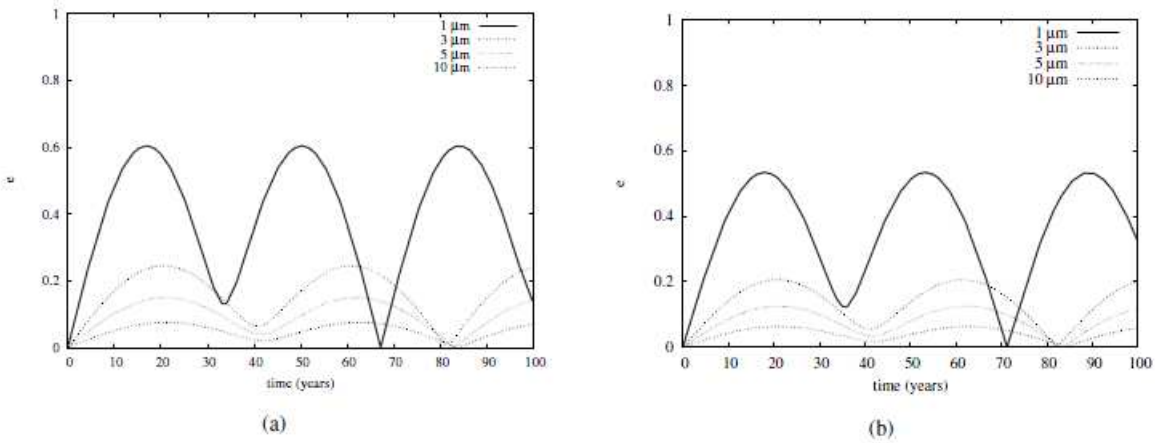
Fonte: Produção do próprio autor

Figura 12 - Variação na excentricidade devido à pressão de radiação solar para uma partícula de diferentes tamanhos no anel ν de Urano com condições iniciais idênticas.



Fonte: Produção do próprio autor

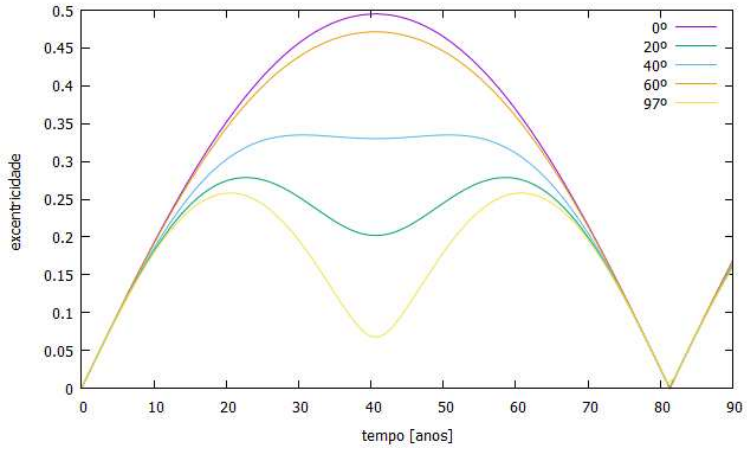
Figura 13 - Variação na excentricidade devido à pressão de radiação solar para uma partícula de diferentes tamanhos no anel μ e ν de Urano com condições iniciais idênticas para cada anel.



Fonte: Sfair & Giuliatti Winter (2009)

As concavidades vistas na figura (11) estão relacionadas à grande inclinação do plano equatorial de Urano. A figura (14) mostra a influencia de γ na evolução temporal da excentricidade.O aumento da obliquidade causa o aparecimento de um mínimo local próximo a metade do período orbital ($\sim$ 42 anos).

Figura 14 - Casos hipotéticos mantendo constantes os parâmetros de uma partícula de 3μm e também os parâmetros físicos de Urano, variando apenas o valor da obliquidade.



Fonte: Produção do próprio autor

5.1.4 Aplicação do achatamento planetário

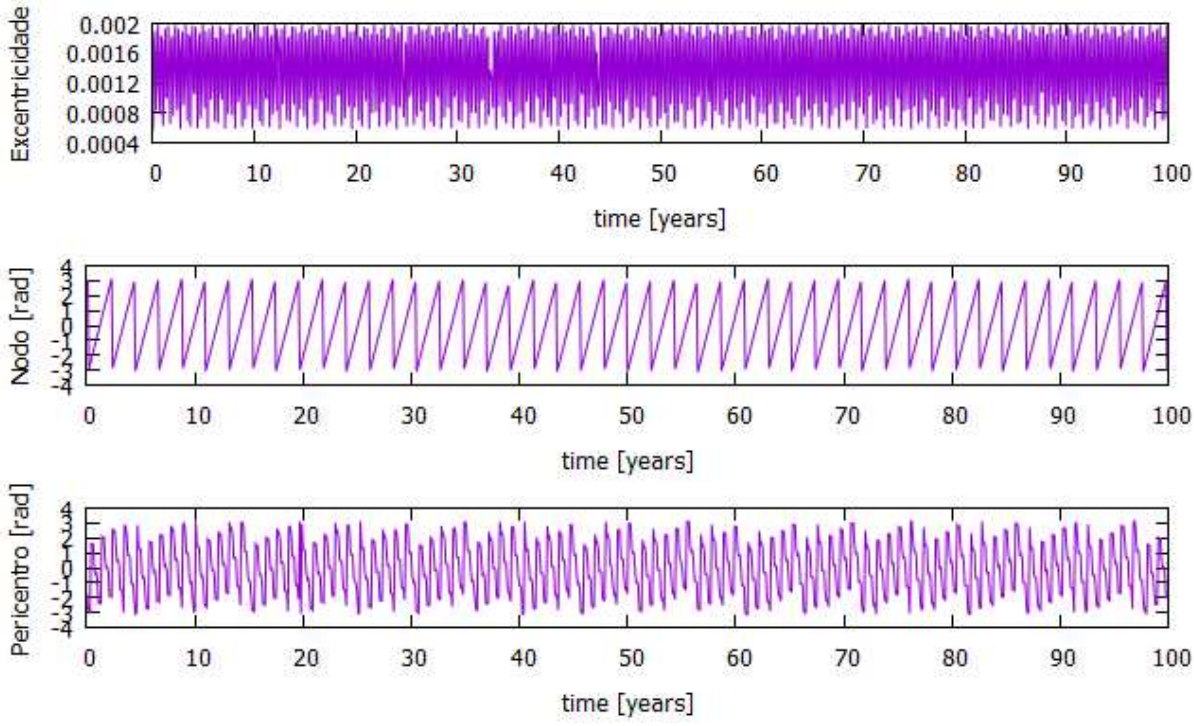
Para validar o nosso código aplicamos o achatamento em Urano e iremos analisar a evolução orbital dos anéis μ e ν e comparar nossos resultados com Sfair & Giuliatti Winter (2009). Em nossa simulação utilizamos o integrador numérico de 15^a ordem IAS15 e avaliamos a evolução numérica de uma amostra de partículas esféricas de tamanho 1, 3, 5, e 10 μm com uma densidade de 1 g/cm^3 (gelo sólido puro) nos anéis μ e ν influenciadas pelo campo gravitacional de Urano como um corpo não esférico.

A tabela (2) resume as informações dos raios orbitais para os anéis μ e ν. Os parâmetros utilizados de Urano (raio, massa, semieixo maior, J_2 e J_4) são derivadas de Murray & Dermott.

Iremos analisar os efeitos do achatamento planetário e em seguida como este efeito modifica a força de radiação solar. Primeiramente, vamos analisar como este efeito modifica a órbita de um satélite de Urano e comparar com o período de oscilação dado pelas equações (29) e (30) para avaliar o quão preciso é o Rebound.

A figura (15) mostra a influência da força de achatamento da componente J2 na evolução orbita do satélite Rosalind, cujos os valores de semi-eixo maior (a), excentricidade (e), inclinação (i), argumento do pericentro (ω), nodo ascendente (Ω) e raio são derivados de Showalter & Lissauer (2006) na tabela (1).

Figura 15 - Simulação numérica para o satélite Rosalind orbitando Urano com a componente J_2.



Fonte: Produção do próprio autor

Calculando a taxa de precessão do nodo e do pericentro de Rosalind com as equações (29) e (30) encontramos os seguintes valores:

$$\dot{\varpi} = 9.163479 \times 10^{-08} \text{ rad/seg} = 165.5940\ ^{\circ}\text{/ano} \tag{54}$$

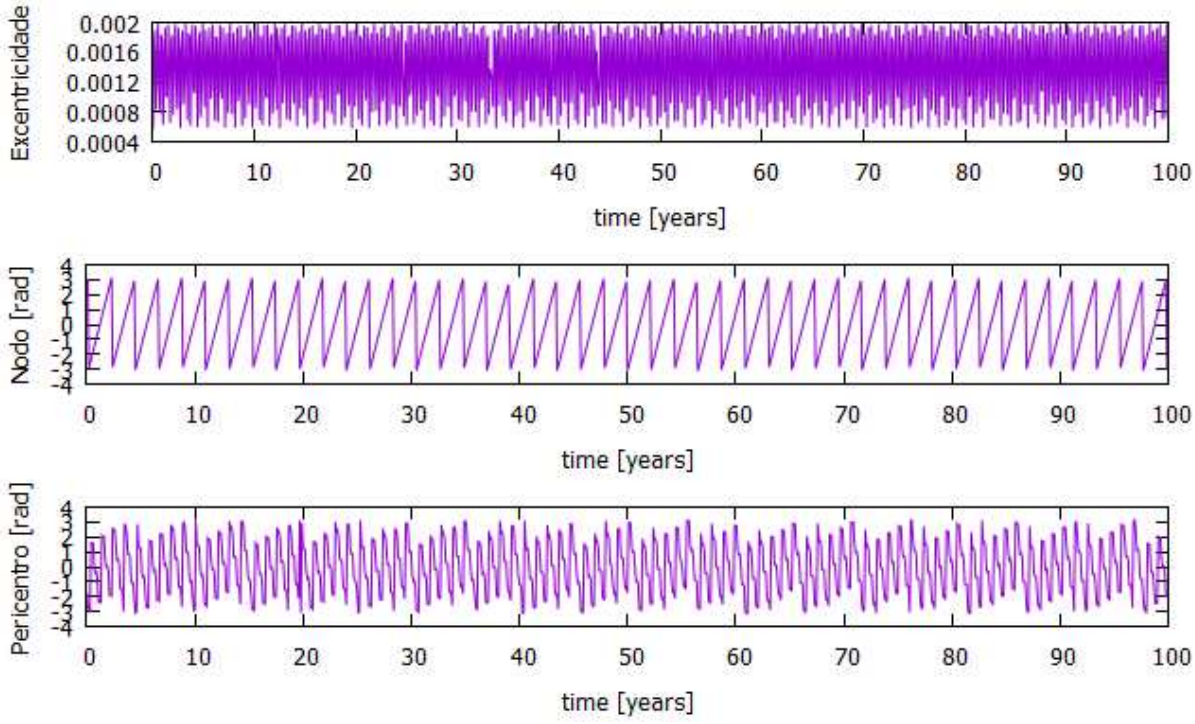
$$\dot{\Omega} = -9.157028 \times 10^{-08} \text{ rad/seg} = -165.4775\ ^{\circ}\text{/ano} \tag{55}$$

Como pode ser observado na figura (15), mostra a variação de curto perído da excentricidade e a variação de ϖ e Ω em função do tempo. O período completo é aproximadamente 2.2 anos, o valor esta de acordo com as equações. Considerando as componentes J_2 e J_4, a inclusão de J_4 diminui ligeiramente o período de oscilação, chega a ser quase imperceptível a diferença entre o gráfico da figura (15) e (16). De fato a diferença entre os valores das taxas incluindo J_4 é muito pequeno, onde os valores são:

$$\dot{\varpi} = 9.186567 \times 10^{08} \text{ rad/seg} = 166.0113\ ^{\circ}\text{/ano} \tag{56}$$

$$\dot{\Omega} = -9.180117 \times 10^{-08} \text{ rad/seg} = -165.8947\ ^{\circ}\text{/ano} \tag{57}$$

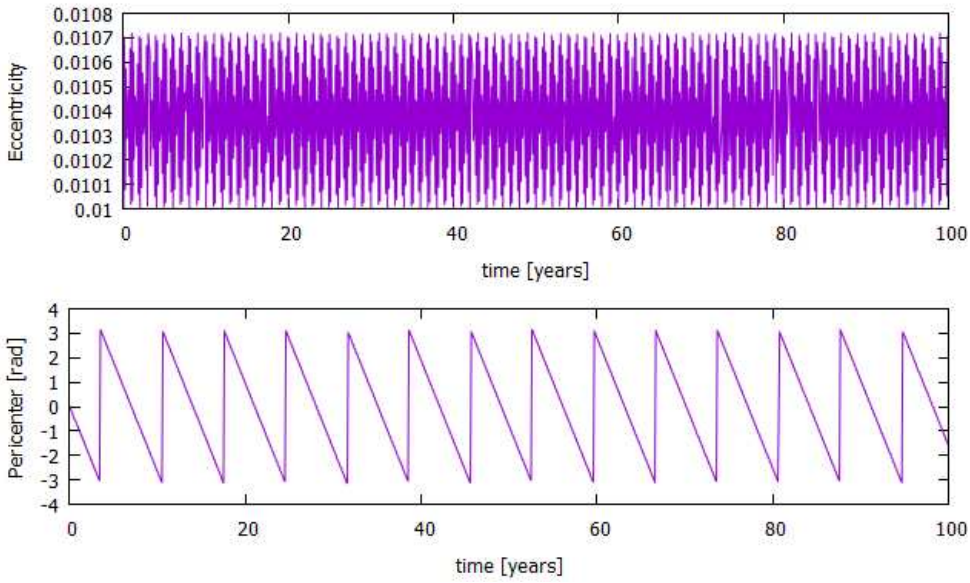
Figura 16 - Simulação numérica para o satélite Rosalind orbitando Urano com a componente J_2 e J_4.



Fonte: Produção do próprio autor

Com estes resultados nós já podemos validar o nosso algoritmo para o achatamento planetário. Porém, também queremos avaliar a evolução de uma partícula de poeira no anel de Urano, isto porque estas partículas são de tamanho micrométrico e sofrem ações de mais forças. Para uma partícula localizada no pico do anel μ de Urano sob a influência da componente J_2 sua evolução orbital pode ser observada na figura (17).

Figura 17 - Simulação numérica para partícula no pico do anel μ de Urano com excentricidade igual a 0.01 mostrando a pequena variação causada na excentricidade e a precessão no pericentro.



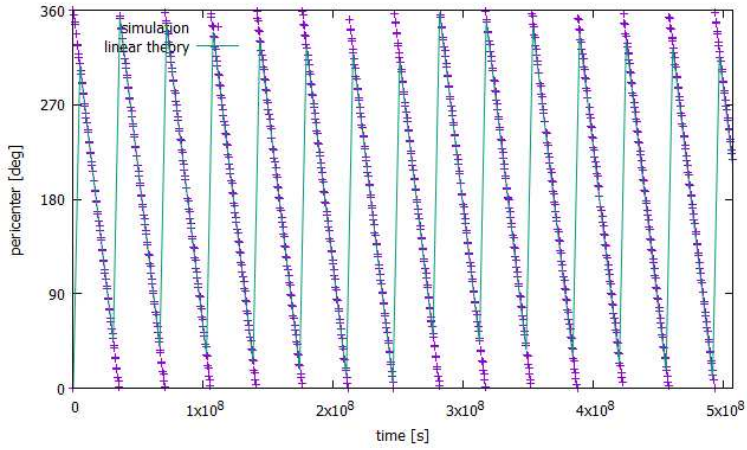
Fonte: Produção do próprio autor

Calculando a taxa de variação do pericentro encontramos o seguinte valor:

$$\dot{\varpi} = 2.84228 \times 10^{-8} \text{ rad/seg} = 51.36509 \text{ }^{\circ}\text{/ano} \tag{58}$$

Com este valor podemos gerar um gráfico teórico para comparar com os nossos resultados que podem ser observado na figura (18) provando mais uma vez que o nosso algoritmo e a equação da força que desenvolvemos esta de acordo com a teoria.

Figura 18 - Simulação numérica para partícula no pico do anel μ de Urano sob a influência da componente J_2 do achatamento planetário com excentricidade igual a 0.01 mostrando a variação no pericentro.



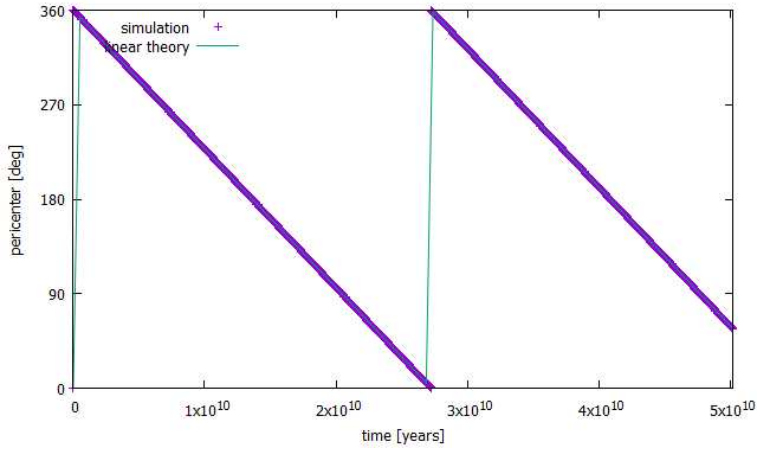
Fonte: Produção do próprio autor

Agora vamos avaliar individualmente a componente J_4, a taxa de variação do pericentro é:

$$\dot{\varpi} = 3.668559 \times 10^{-11} \text{ rad/seg} = 6.629485 \times 10^{-02} \text{ °/ano} \tag{59}$$

A figura (19) mostra o quão lenta é a variação do pericentro, ele demora certa de 556 anos para completar um período.

Figura 19 - Simulação numérica para partícula no pico do anel μ de Urano sob a influência da componente J_4 do achatamento planetário com excentricidade igual a 0.01 mostrando a variação no pericentro.



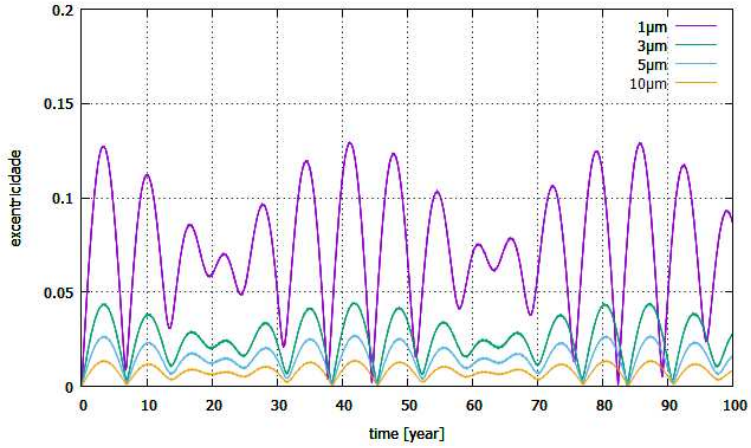
Fonte: Produção do próprio autor

Estamos interessados em avaliar a órbita de uma partícula de poeira no anel de Urano, pois estas sofrem o efeito do achatamento planetário e da força de radiação solar.

5.1.5 Efeitos combinados do achatamento e da radiação solar

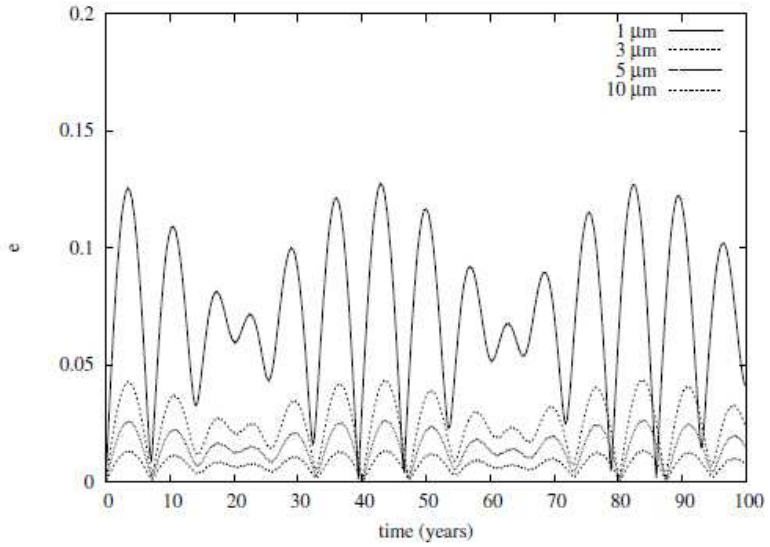
Os efeitos do achatamento são mais evidentes quando é adicionada a força de radiação solar. A figura (20) mostra a variação da excentricidade para partículas de diferentes tamanhos e condições iniciais idênticas às da figura (11). Fazendo uma comparação entre às duas figuras mostramos que o achatamento causa um amortecimento na variação da excentricidade e a diminuição na amplitude de oscilação é mais acentuada para partículas mais próximas ao planeta. Este comportamento é esperado uma vez que a intensidade da radiação solar diminui e os efeitos do achatamento de Urano aumentam quanto mais próximo do planeta. O período de oscilação da excentricidade também é modificado pela inclusão do achatamento, passando a ser menor. O comportamento da excentricidade passa a ser modulado pela combinação de três frequências: n_s, ϖ e uma frequência de curto período relacionada à inclusão do termo J_2. Estes resultados estão de acordo com Sfair & Giuliatti Winter (2009) como pode ser observado na figura (21) para uma partícula na anel μ com as mesmas condições iniciais.

Figura 20 - Variação na excentricidade devido a pressão de radiação solar e a componente J2 do achatamento planetário para partículas de diferentes tamanhos no anel μ de Urano com condições iniciais idênticas.



Fonte: Produção do próprio autor

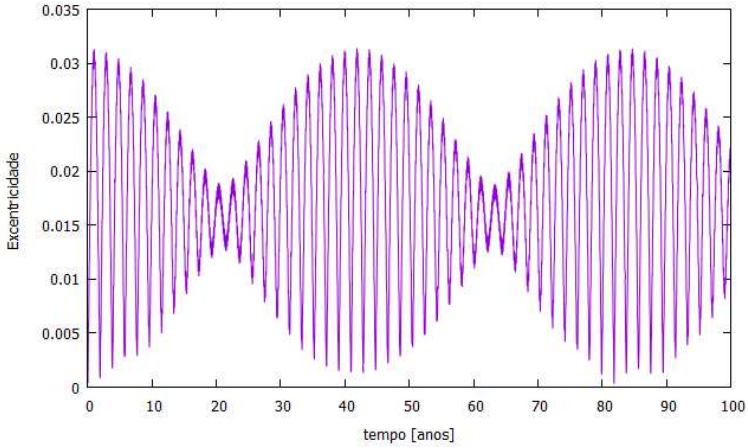
Figura 21 - Variação na excentricidade devido a pressão de radiação solar e a componente J2 do achatamento planetário para partículas de diferentes tamanhos no anel μ de Urano com condições iniciais idênticas.



Fonte: Sfair & Giuliatti Winter (2009)

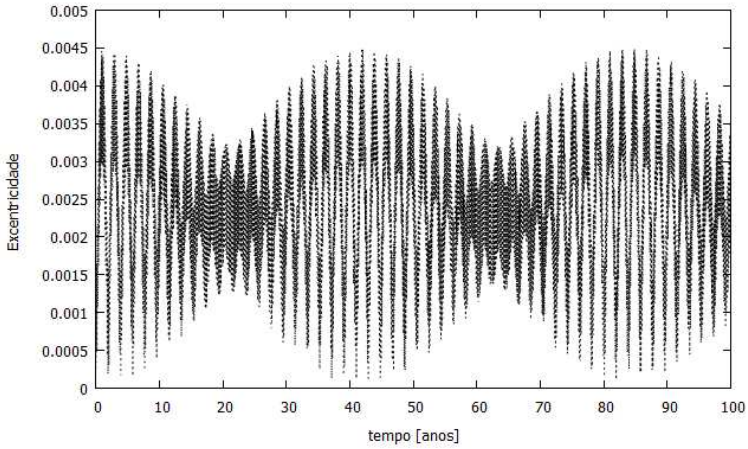
A redução da amplitude de oscilação é mais evidente para as partículas no anel ν. A figura (22) mostra a variação da excentricidade para uma partícula do tamanho de $1\mu m$ e a figura (23) para o tamanho de 10 μm localizadas no pico do anel ν de Urano com as mesmas condições iniciais da figura (12). Comparando estes resultados com os encontrados por Sfair & Giuliatti Winter (2009) (figura 24) é possível observar uma certa diferença na amplitude das imagens, para ambas as partículas a diferença foi de 0.01 no módulo amplitude.

Figura 22 - Variação na excentricidade devido a pressão de radiação solar e a componente J2 do achatamento planetário para uma partícula do tamanho de $1\mu m$ no anel ν de Urano.



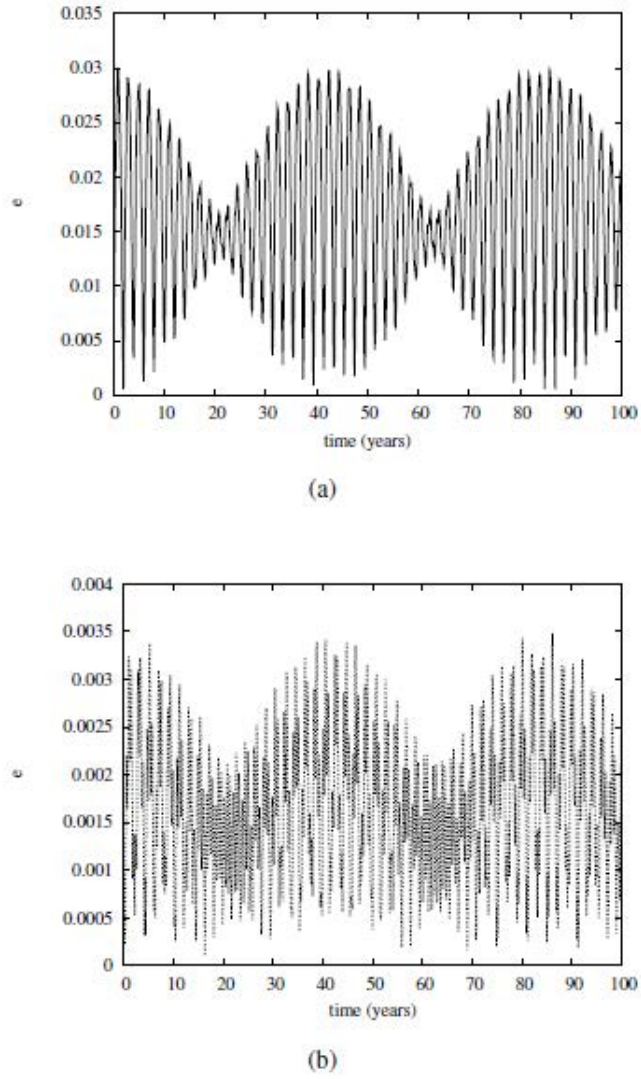
Fonte: Produção do próprio autor

Figura 23 - Variação na excentricidade devido a pressão de radiação solar e a componente J2 do achatamento planetário para uma partícula do tamanho de $10\mu m$ no anel ν de Urano.



Fonte: Produção do próprio autor

Figura 24 - Variação na excentricidade devido a pressão de radiação solar e a componente J2 do achatamento planetário para partículas de tamanhos de 1μ e $10\mu m$ no anel ν de Urano.



Fonte: Produção do próprio autor

5.1.6 Influência dos satélites nos anéis

A adição das duas forças perturbativas geram alterações nas órbitas das partículas possibilitando o cruzamento com as órbitas dos satélites próximos podendo ocasionar em colisões entre eles.

Para cada anel analisamos a evolução de partículas esféricas de tamanho 1, 3, 5, e 10 μm com uma densidade de $1 g/cm^3$ (gelo sólido puro) nos anéis μ e ν. Para cada anel, um conjunto de 1000 partículas foram radialmente e azimutalmente distribuídos dentro da região do anel e condições iniciais foram distribuídas aleatoriamente sendo integrada para um intervalo de 1000 anos. Os elementos orbitais e o raio dos satélites analisados estão listados na tabela (1). Para todos eles assumimos a densidade uniforme de 1.3 g/cm^3, igual a do satélite Miranda. As informações sobre os anéis μ e ν estão na tabela (2).

Quando a distância entre a partícula e um satélite foi menor do que o raio do satélite, foi detectada uma colisão, neste caso, a partícula é removida do sistema.

As partículas nos anéis são consideradas como partículas testes e, portanto elas não influenciam nenhuma partícula do sistema, mas sofrem influência dos corpos considerados como ativos que são o planeta (Urano) e os satélites (Mab, Puck, Rosalind e Portia).

Utilizamos o integrador IAS15 e modificamos o algoritmo de colisões diretas para fazer a pesquisa das colisões. A alteração do algoritmo colisões foi realizada para fazer a pesquisa apenas das colisões que o planeta e os satélites sofrem, portanto todas as colisões entre as partículas nos anéis foram desconsideradas, isso fez com que o tempo gasto no processamento diminuísse pela metade.

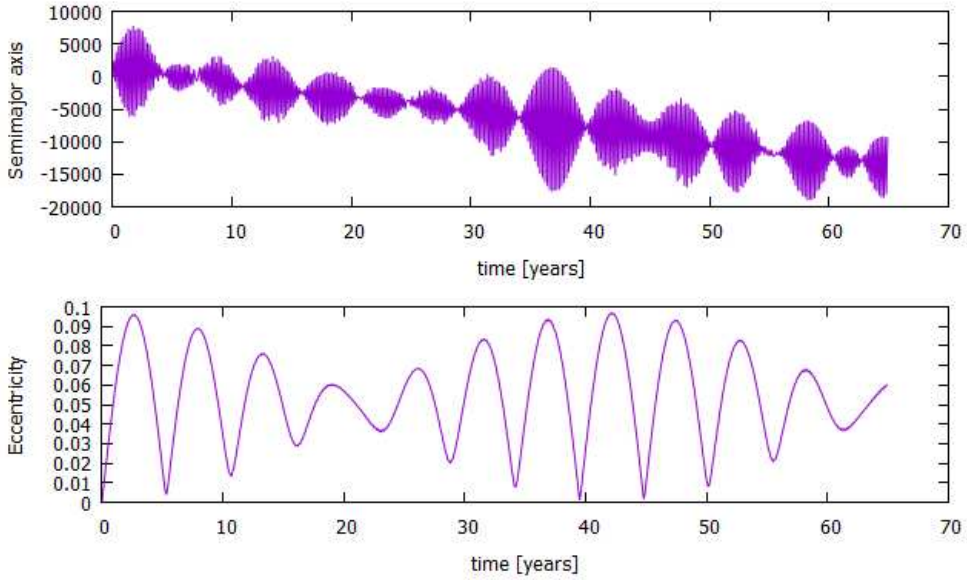
Foram realizadas duas simulações diferentes, uma para cada anel. Consideramos 1000 partículas nos anéis e para o anel μ foram integrados Urano, Mab e Puck e para o anel ν foram integrados Urano, Rosalind e Portia. As simulações foram dividas porque podem ser tratados como sistemas independentes, pois um sistema não influência o outro devido a distância, isto diminui o tempo de processamento. Também poderíamos analisar as partículas apenas com o planeta e um satélite por vez, mas o tempo de processamento gasto para cada passo de integração foi o mesmo nos dois tipos de sistemas.

O tempo gasto no processamento está intimamente ligado ao número de partículas nos anéis, ao integrador e o módulo de colisão. Os módulos de força não alteram significativamente o tempo gasto.

Utilizando estas condições foram registradas diversas colisões entre as partículas e os satélites, sendo que a proporção da quantidade de partículas que colidiram com cada satélite é compatível com Sfair & Giuliatti Winter (2009) onde 3% das partículas colidem com Mab e o resto com Puck para o anel μ e as colisões para o anel ν são cerca de 40% colidem com Rosalind e 60% com Portia. Estes resultados são bons, pois as colisões se devem especialmente por dois fatores: o tamanho dos satélites e a localização. Os satélites Puck e Portia são maiores que seus companheiros Mab e Rosalind, respectivamente. Especialmente para o caso de Puck e Mab, no qual Puck é 6.7 vezes maior, com uma área que chega a ser 45 vezes superior a de Mab, aumentando a possibilidade de impacto.

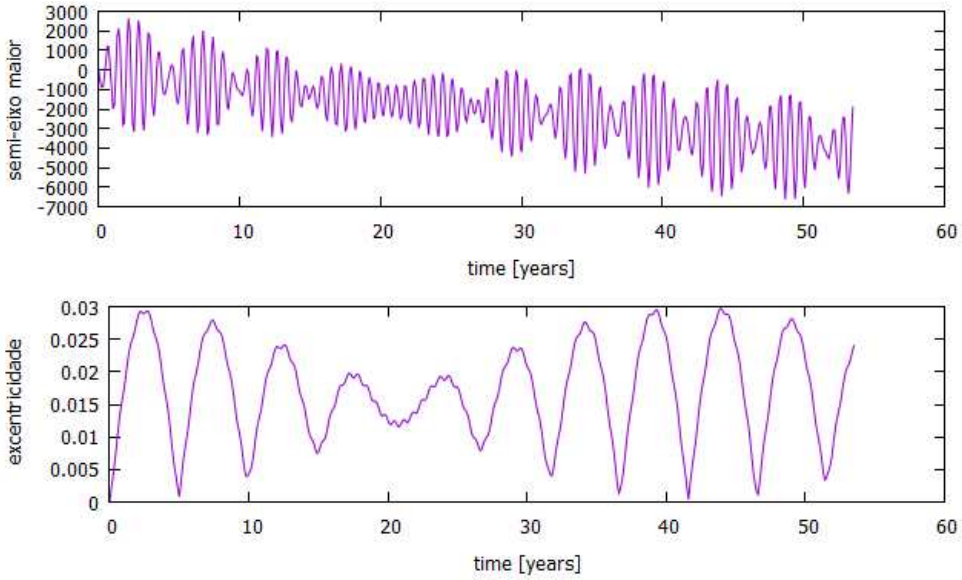
Também analisamos a evolução orbital de uma partícula no anel μ para cada tamanho utilizado na simulação que colidiram com o satélite, que podem ser vistos nas figuras (25) , (26), (27) e (28), para partículas de 1, 3, 5, e 10 μm. Analisando as figuras algumas partículas demonstraram alguns ruídos que podem ser que sejam ocasionados por aproximações com os satélites, mas todas as figuras apresentam um padrão na evolução, diferente com Sfair & Giuliatti Winter (2009) cujas imagens apresentam uma evolução caótica, registrando saltos nos encontros próximos com os satélites(figura 29).

Figura 25 - Variação do semi-eixo maior e da excentricidade devido a força de radiação solar e o achatamento planetário J_2 para partículas de 1μm anel μ de Urano com sob a influência dos satélites Mab e Puck. O tempo em que os gráficos terminam é exatamente o tempo de integração que foi registrado a colisão desta partícula com o satélite Puck.



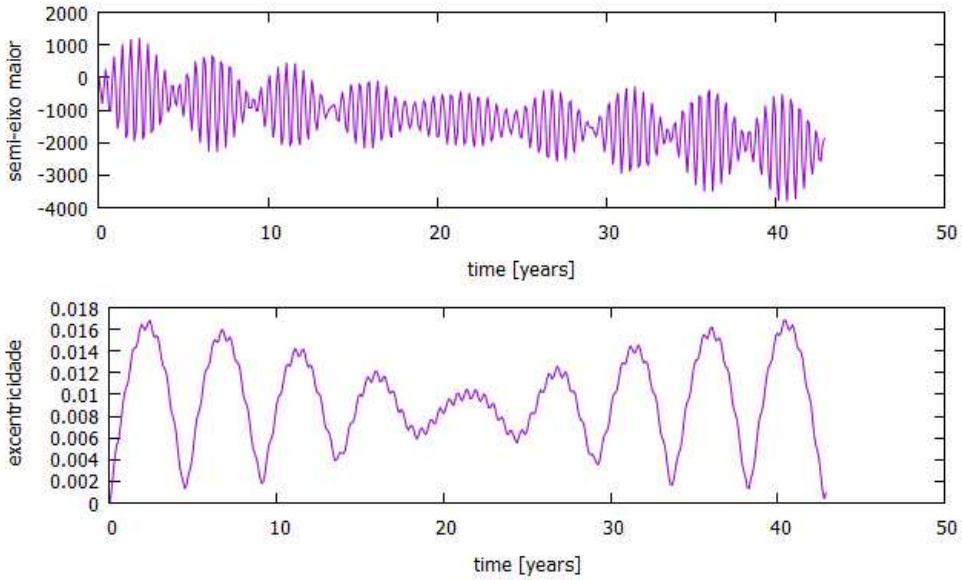
Fonte: Produção do próprio autor

Figura 26 - Variação do semi-eixo maior e da excentricidade devido a força de radiação solar para partículas de 3μm anel μ de Urano com sob a influência dos satélites Mab e Puck. O tempo em que os gráficos terminam é exatamente o tempo de integração que foi registrado a colisão desta partícula com o satélite Puck.



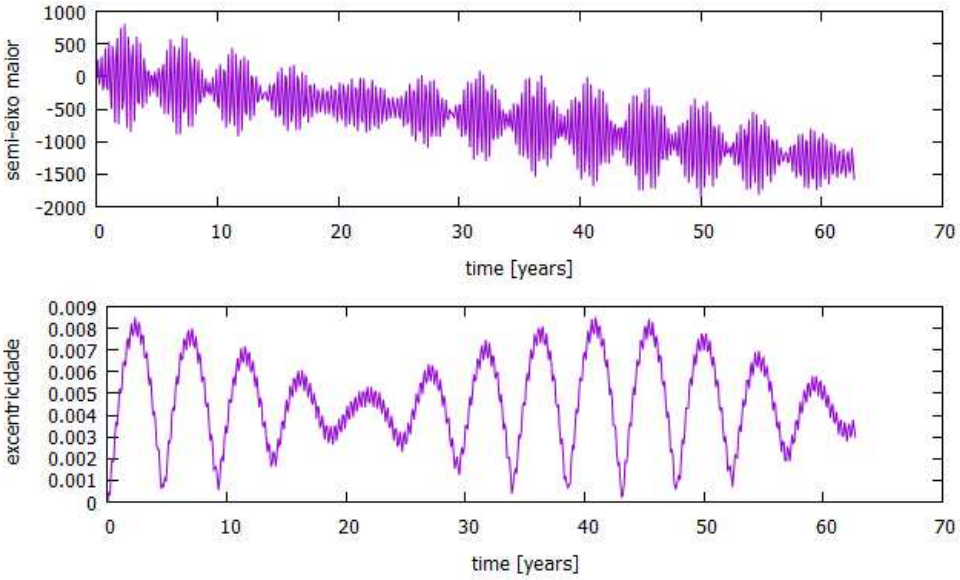
Fonte: Produção do próprio autor

Figura 27 - Variação do semi-eixo maior e da excentricidade devido a força de radiação solar e o achatamento planetário J_2 para partículas de 5μm anel μ de Urano com sob a influência dos satélites Mab e Puck. O tempo em que os gráficos terminam é exatamente o tempo de integração que foi registrado a colisão desta partícula com o satélite Puck.



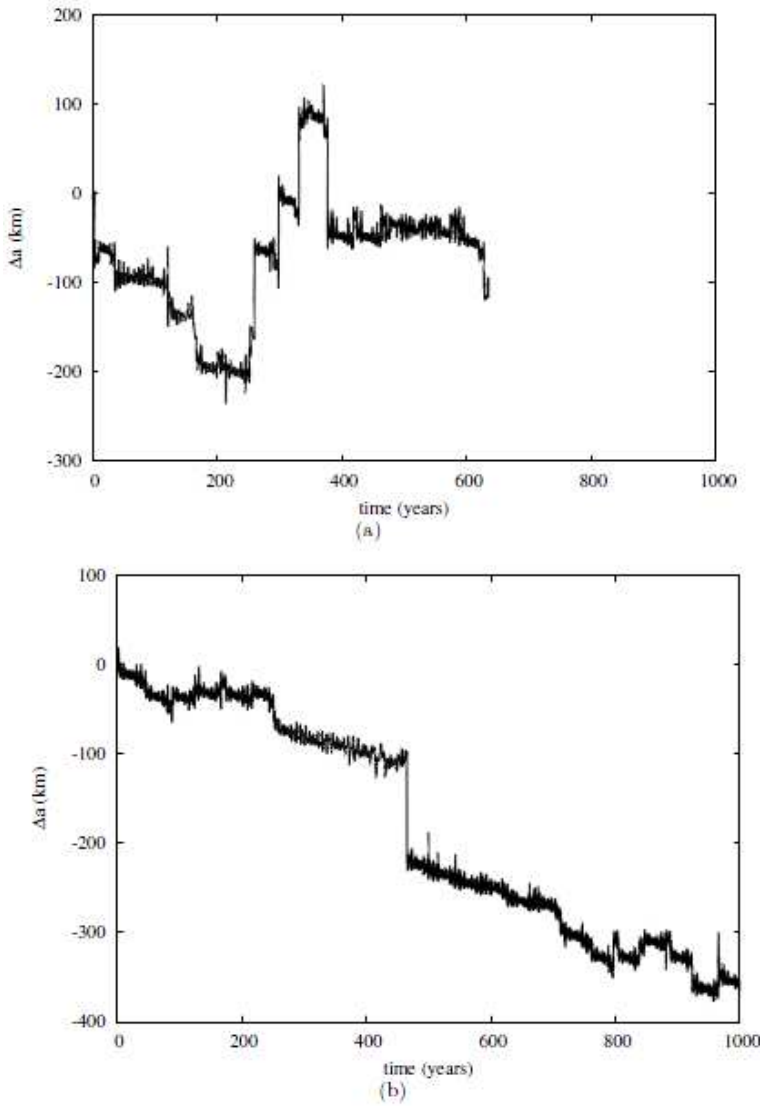
Fonte: Produção do próprio autor

Figura 28 - Variação do semi-eixo maior e da excentricidade devido a força de radiação solar e o achatamento planetário J_2 para partículas de 10μm anel μ de Urano com sob a influência dos satélites Mab e Puck. O tempo em que os gráficos terminam é exatamente o tempo de integração que foi registrado a colisão desta partícula com o satélite Puck.



Fonte: Produção do próprio autor

Figura 29 - Evolução do semi-eixo maior de duas partículas de 1 μm do anel μ perturbadas pela força de radiação solar, pelo achatamento de Urano e pela interação gravitacional com os satélites Puck and Mab. Em (a) a partícula colide com Puck após 635 anos e a partícula representada em (b) permanece na região do anel durante todo o período analisado. Em cada gráfico $\Delta a = 0$ corresponde ao semi-eixo maior inicial da partícula. A largura do anel é 17000 km



Fonte: Sfair & Giuliatti Winter(2009)

Por fim, apesar da proporção da quantidade de colisões estar de acordo com os fatores físicos e orbitais dos satélites e também com outras bibliografias, a conclusão que podemos chegar é que os módulos utilizados em nossas simulações com Rebound talvez não sejam os mais indicados, apesar de se encaixar nas especificações. É uma simulação extremamente lenta que inviabilizou o processamento e analise a longo prazo além de não gerar as devidas interações com os satélites.

5.2 Testes extras com asteroide 87 Sylvia

Nesta seção iremos analisar um sistema asteroidal triplo chamado 87 Sylvia. É um sistema localizado no cinturão de asteroides, composto por um asteroide primário (Sylvia) e duas luas (Romulus e Remus). É um sistema interessante de ser analisado, pois podemos analisar a interação dos satélites com a perturbação do Sol e identificar a ressonância secular que afeta a dinâmica do sistema.

5.2.1 O Sistema

O primeiro sistema triplo asteroidal descoberto foi o 87 Sylvia (Marchis et al., 2005b). É composto por um corpo primário, que chamaremos apenas de Sylvia, com cerca de 280km, localizado no principal cinturão de asteroides, com semi-eixo maior de 3.29AU, excentricidade de 0.08 e de inclinação igual a 11°. Este sistema possui duas luas pequenas chamadas de Romulus e Remus, com tamanhos em torno de 18 ± 4 km e 7 ± 2 km, ambos possuem orbitas praticamente circulares e com os raios de 1360km e 710km, respectivamente. Sendo que iremos estudar a estabilidade de Romulus e Remus, explorando a dinâmica do sistema, como é a interação entre os satélites, o que acontece com a adição do Sol e investigaremos a contribuição da adição do achatamento planetário de Sylvia.

5.2.2 Simulação

Utilizamos o integrador IAS15 e incluímos as partículas através de seus elementos orbitais disponíveis na tabela (3). Avaliamos a evolução orbital no intervalo de 5×10^4 anos, que corresponde a 5×10^6 períodos orbitais de Romulus.

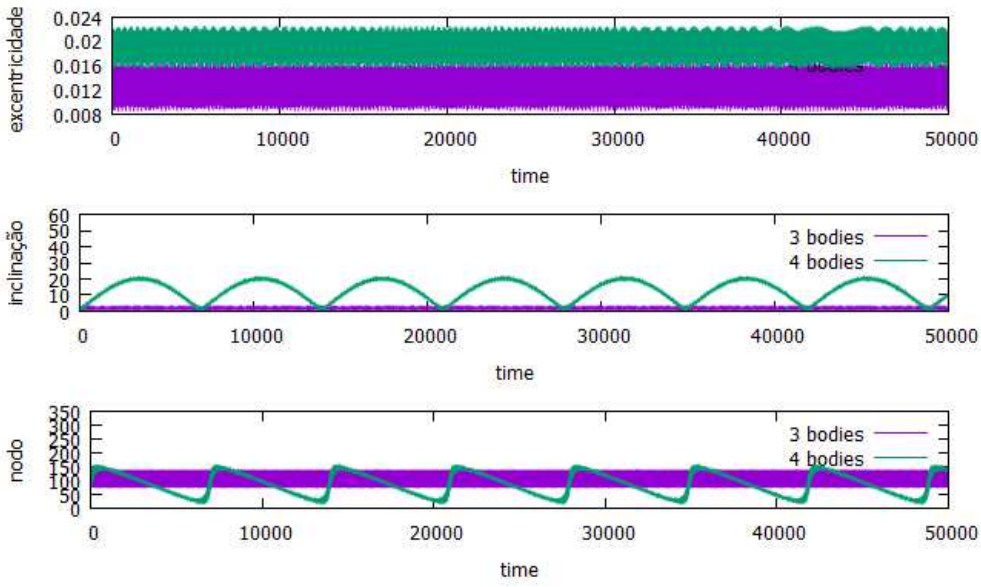
Tabela 3 - Os dados físicos e orbitais do sistema de Sylvia.

	Massas(kg)	a	e	I°	$\omega(^{\circ})$	$\Omega(^{\circ})$	f($^{\circ}$)	Período
Sylvia	$1.4780 \times 10^{1}9$	3.49 AU	0.08	10.855	26 6195	73 195	8 .51412	6.52 anos
Romulus	$3.6625 \times 10^{1}5$	1356 km	0.001	1.7	0.58	0.51	81.88	3.65 dias
Remus	$2.1540 \times 10^{1}4$	706 km	0.016	2.0	0.093	0.18	12.695	1.38 dias

Fonte: Marchis et al., (2005b)

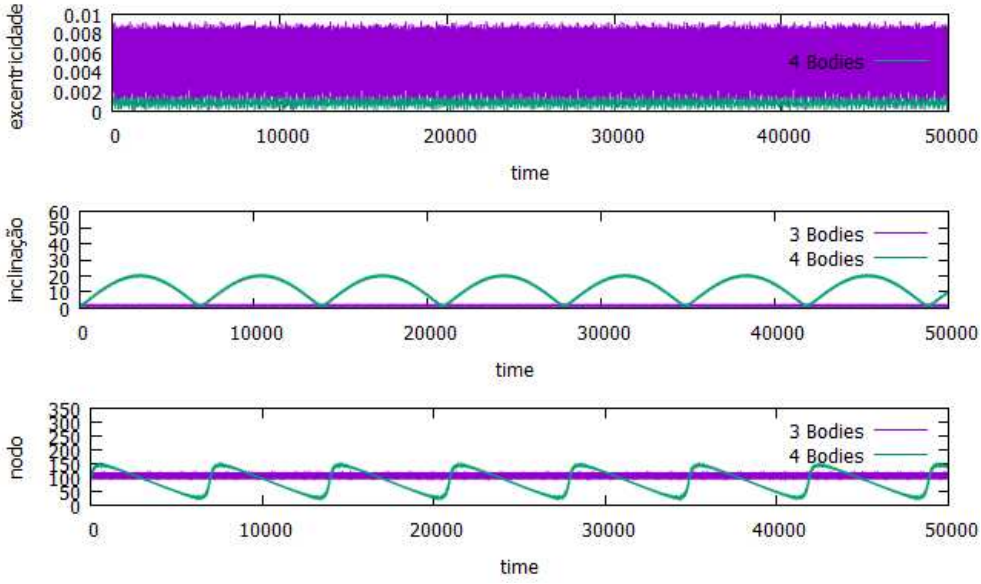
Primeiramente realizamos a simulação para o problema de 3 corpos, Sylvia-Romulus-Remus e então de 4 corpos Sylvia-Romulus-Remus-Sol. As figuras (30) e (31) mostram a evolução temporal da excentricidade (e), inclinação (I) e longitude do nodo ascendente (Ω) de Remus e Romulus, respectivamente.

Figura 30 - Evolução da excentricidade, da inclinação e da longitude do nodo ascendente para Remus. O problema de 3 corpos é composto por Sylvia-Romulus-Remus e o problema de 4 corpos Sylvia-Romulus-Remus-Sol.



Fonte: Produção do próprio autor

Figura 31 - Evolução da excentricidade, da inclinação e da longitude do nodo ascendente para Romulus. O problema de 3 corpos é composto por Sylvia-Romulus-Remus e o problema de 4 corpos Sylvia-Romulus-Remus-Sol.



Fonte: Produção do próprio autor

Os resultados da simulação de 3 corpos não mostram nenhum variação significativa dos elementos orbitais dos satélites. Já a inclusão do Sol na simulação causa uma mudança significativa na inclinação

com uma amplitude de aproximadamente 20° e com o período 6×10^3 anos. Também causa uma variação na longitude do nodo ascendente chegando aproximadamente 120° com o mesmo período orbital da inclinação. Este resultado esta de acordo com Winter et al.(2009) cujos os valores e comportamentos da inclinação e longitude do nodo ascendente foram os mesmos.

A excentricidade também teve o seu valor alterado, para Remus houve um aumento do valor da excentricidade passando a oscilar em torno de 0.015 à 0.023 e Romulus teve uma diminuição passando a oscilar em torno de 0.0002 à 0.0016. Esta foi a principal discrepância dos resultados obtidos em Winter et al. (2009). Isto pode ser devido a diferença dos integradores utilizados, mas será necessária uma análise mais cuidadosa, pois os valores que a excentricidade bem diferentes. Entretanto, a inclinação e a longitude do nodo ascendente tiveram a evolução orbital igual ao resultado da literatura e assim como neste trabalho verificamos que os satélites possuem as frequências e amplitudes similares, que é um fato dos satélites estarem conectados, como foi explicado em Winter et al. (2009).

6 CONCLUSÃO

O estudo de anéis planetários serve como laboratório para o entendimento do processo de formação planetária e a dinâmica de galáxias e o Rebound é uma ótima ferramenta de integração para sistemas de N-corpos que interagem gravitacionalmente, podendo lidar com forças conservativas e não conservativas também com encontros próximos entre partículas e órbitas de alta excentricidade mantendo os erros sistemáticos abaixo da precisão de máquina.

O fato do Rebound ser escrito em C melhora o desempenho do processamento dos códigos e auxilia na modularização, que divide as grandes tarefas de computação em tarefas menores e utiliza seus resultados parciais para compor o resultado final desejado. A modularização é extremamente vantajosa, pois facilita na manutenção, ajuda na organização do código e permite o reuso do código, otimizando os projetos e diminuindo a alocação de memória para o processamento. Outra vantagem de utilizar códigos modulares é que cada desenvolvedor pode focar na construção de códigos cada vez mais robustos e eficientes com base na sua própria área. Isto nos permitiu o acoplamento de diferentes códigos especializados sem a necessidade de modificar cada um deles individualmente.

O Rebound mostrou-se eficiente para os tipos de estudos abordados, pois permite uma maneira rápida para calcular as forças e um método de integração suficientemente preciso para evoluir as partículas no tempo. Sua praticidade facilita o entendido de como criar simulações, disponibilizando diversos módulos que podem ser alterados sem o mínimo de esforço para a aplicação desejada, além de auxiliar o usuário nas condições iniciais das partículas ao prover funções que realizam os cálculos das coordenadas cartesianas e dos elementos orbitais.

O foco de estudo deste trabalho foi incluir módulos de forças perturbativas ao pacote de software do Rebound. As equações das forças para cada perturbação precisou ser incluída manualmente, pois o pacote originalmente resolve apenas o problema considerando as forças gravitacionais.

Para a vadiação das equações e algoritmos, desenvolvidos neste trabalho, optamos pelo integrador IAS15 pelo fato de possibilitar integrações com forças não conservativas, sem perder a precisão sobre os erros sistemáticos. Este integrador mostrou um bom desempenho diante das forças perturbativas aplicadas ao comparar com as literaturas. Entretanto, ele pode não ser o integrador mais indicado para trabalhar com milhares de partículas em uma simulação, mesmo que sejam partículas testes.

Por fim ao comparar o desempenho e a praticidade com outros pacotes geralmente utilizados (e.g. Mercury) indicam que o Rebound é mais eficiente e pode ser utilizado em futuros estudos.

REFERÊNCIAS

H. Rein; S.-F. Liu. **REBOUND: an open-source multi-purpose N-bodycode for collisional dynamics.** , 537:A128, January 2012.

J. A. Burns; P. L. Lamy; S. Soter. **Radiation forces on small particlesin the Solar System: A re-consideration.** , 232:263–265, April 2014.

Douglas P. Hamilton; Alexander V. Krivov. **Circumplanetary dust dy-namics: Effects of solar gravity, radiation pressure, planetary oblateness,and electromagnetism**.Icarus, 123(2):503 – 523, 1996.

Danby, J. M. A. 1988. **Fundamentals of Celestial Mechanics** (2nd ed.) Willmann-Bell, Richmond, VA.

Douglas P. Hamilton. **Motion of dust in a planetary magnetosphere: Orbit-averaged equations for oblateness, electromagnetic, and radiation forces withapplication to saturn's e ring**.Icarus, 101(2):244 – 264, 1993.

Carl D. Murray and Stanley F. Dermott.**Solar System Dynamics**. Cambridge University Press, 1998.

Showalter, M. R.; Lissauer, J. J. 2006, **Sci.**, 311, 973

Sfair; Giuliatti Winter 2009. **Orbital evolution of the μ and ν dust ring particles of Uranus**, Astronomy & Astrophysics 505 (2), 845-852

Sfair; Giuliatti Winter. **The role of Mab as a source for the μ ring of Uranus**. 2012.

Giorgini, J. D. et al. **In Bulletin of the American Astronomical Society**, volume 28, pag. 1158. 1996.

Fegley, B. J. et al. **Spectroscopy and chemistry of the atmosphere of Uranus**, pag. 147-203. Uranus, 1991.

Smith, B. A. et al. **D. Voyager 2 in the uranian system - imaging science results**. Science, 233:43-64, 1986.

de Pater, I., Hammel; H. B.; Gibbard, S. G.; Showalter, M. R. **New dust belts of uranus: One ring, two ring, red ring, blue ring**. Science, 312:92-94, 2006b.

NASA & JPL. **Solar System Exploration**. 2016

Chambers, J. E. **A hybrid symplectic integrator that permits close encounters between massive bodies**. MNRAS, 304:793-799, 1999.

W. Kahan. **Pracniques: Further remarks on reducing truncation errors**.Commun. ACM, 8(1):40–, January 1965.

O. C. Winter et al. **On the sta-bility of the satellites of asteroid 87 sylvia**.Monthly Notices of the RoyalAstronomical Society, 395(1):218, 2009

Ferrari, Guilherme Gonçalves. **Novos mapas simpléticos para integração de sistemas hamiltonianos com múltiplas escalas de tempo : enfoque em sistemas gravitacionais de N-corpos**. Disponível em:

<http://hdl.handle.net/10183/127985>

G. Efstathiou, M. Davis; S. D. M. White; C. S. Frenk. **Numerical techniques for large cosmological N-body simulations. Astrophysical Journal Supplement Series**, 57:241–260, February 1985. doi: 10.1086/191003.

Hanno Rein; Scott Tremaine. **Symplectic integrators in the shearingsheet**.Monthly Notices of the Royal Astronomical Society, 415(4):3168–3176, 2011.

H. Rein; D. Tamayo. **WHFAST: a fast and unbiased implementation ofa symplectic Wisdom-Holman integrator for long-term gravitational simu-lations**. , 452:376–388, September 2015.

H. Rein; D. S. Spiegel. **IAS15: a fast, adaptive, high-order integrator forgravitational dynamics, accurate to machine precision over a billion orbits**., 446:1424–1437, January 2015.

S. Newcomb. **Note on the relation of the photographic and visual magni-tudes of the stars**.Astronomische Nachrichten, 148(18):285–286, 1899

D. Brouwer. **On the accumulation of errors in numerical integration**. ,46:149–153, October 1937.

Silburt, A. Rein; H., Tamayo, D., **HERMES: A Hybrid Integrator for Simulating Close Encounters and Planetesimal Migration**, 2016, MNRAS (submitted), Disponível em: <http://astro.utoronto.ca/ silburt/HERMES.pdf>