MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

PROPAGAÇÃO NUMÉRICA E SEMI-ANALÍTICA DE UMA DISTRIBUIÇÃO DE DETRITOS ESPACIAIS

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA (PIBIC/CNPq/INPE)

Bolsista 1- João Paulo Marques Reginato (ITA, Bolsista PIBIC/CNPq)

E-mail: **jplg@zaz.com.br**

Bolsista 2- Sandro Felgueiras Castro (ITA, Bolsista PIBIC/CNPq)

E-mail: **sandro-ita05@bol.com.br**

Dr. Marcelo Lopes de Oliveira e Souza (DMC/ETE/INPE, Orientador)

E-mail: **marcelo@dem.inpe.br**

28-30 de Julho de 2003

RESUMO

Este trabalho tem como objetivo resumir e relatar o Projeto de Iniciação Científica de mesmo título realizado de 01/08/2002 a 31/07/2003, para simular a geração e a propagação de "Detritos Espaciais", para posterior estudo das propriedades básicas do processo. Com estas, pode-se estudar os problemas de colisão e interferência entre os "Detritos Espaciais" e outros objetos encontrados no espaço como satélites, ônibus espaciais, e estações espaciais.

Para tanto, **o 1° autor**: fez um estudo inicial em Mecânica Orbital e em erros numéricos e de integração de equações diferenciais pelo método de Runge-Kutta.A seguir, começou a simular detritos espaciais, adaptando um programa KK em linguagem C, originalmente feito para o sistema operacional UNIX, como base para essa simulação. Para melhor entendimento do programa KK, fez primeiramente uma leitura deste e a construção e posterior aperfeiçoa mento de um fluxograma do mesmo. A seguir, adaptou o programa KK para rodar no sistema operacional Windows 2000 com auxílio do programa MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0. Além disso, adaptou a saída de dados do programa KK para torná-lo compatível com os programas/ambientes MATLAB, para utilizarmos sua capacidade de análise.

A partir de 01/02/2003 **o 2°. autor** assumiu o projeto fazendo um estudo inicial de Mecânica Orbital e Teoria de Probabilidades. A seguir, leu o 1° artigo do **orientador** sobre o tema para melhorar a compreensão sobre este e iniciou um estudo aprofundado do programa KK, já adaptado pelo 1° autor. Este programa gera as posições cartesianas de um número de detritos, ao longo do tempo, além das componentes x e y da velocidade de cada detrito. Essas coordenadas e as componentes de velocidade foram impressas num documento formato .txt. Para comparar com tal descrição numérica do movimento de"Detritos Espaciais", ele iniciou a programação e os testes de uma primeira descrição analítica daqueles proposta pelo orientador.

Para tanto, fez um programa em C com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0, capaz de ler as coordenadas impressas pelo programa KK e parametrizar a propagação desses detritos. Os parâmetros utilizados nesse processo foram o tempo e as coordenadas do centro de atração gravitacional. Ele assumiu que cada detrito tinha a mesma velocidade angular constante. Depois disso, fez um estudo sobre o MATLAB, para utilizá-lo nesse processo.Todos estes resultados foram satisfatórios e serão melhorados com a renovação deste Projeto de Iniciação Científica objetivando: 1) Calcular a estatística da distribuição de "Detritos Espaciais" e estudar a sua evolução no tempo, iniciando com a posição do Centro de Massa-CM; 2) Observar e interpretar as propriedades básicas de tal processo; 3) Melhorar e testar um modelo analítico simples (geométrico, cinemático, etc.) para a distribuição de detritos espaciais e sua evolução; compará-lo com as simulações; e aperfeiçoá-lo; 4) Se houver tempo, introduzir o arrasto atmosférico e refazer os itens anteriores; 5) Documentar o trabalho realizado e publicá-lo no SICINPE 2004.

SUMÁRIO

Página

CAPÍTULO 1 – INTRODUÇÃO.....06

CAPÍTULO 2 – OBJETIVOS DO TRABALHO.....08

CAPÍTULO 3 –METODOLOGIA.....09

CAPÍTULO 4 – RESULTADOS E ANÁLISES.....10

CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS.....14

REFERÊNCIAS.....15

APÊNDICE 1 – O PROGRAMA KK.C ORIGINAL FEITO POR DANTON NUNES EM C PARA ESTAÇÕES DE TRABALHO COM UNIX E POSTSCRIPT PARA PROPAGAR E PLOTAR OS DETRITOS ESPACIAIS.....16

APÊNDICE 2 – O PROGRAMA KK2TEXTO.C FEITO POR JOÃO PAULO MARQUES REGINATO EM C PARA PCS COM WINDOWS 2000 E VISUAL STUDIO 6.0 PARA PROPAGAR E PLOTAR OS DETRITOS ESPACIAIS.....23

APÊNDICE 3 – O PROGRAMA KK2MAT.C FEITO POR JOÃO PAULO MARQUES REGINATO EM C PARA PCS COM WINDOWS 2000 E MATLAB PARA PROPAGAR E PLOTAR OS DETRITOS ESPACIAIS.....28

APÊNDICE 4 - ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA PLOTAGEM DAS FIGURAS.....34

APÊNDICE 5 - ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA CÁLCULO E PLOTAGEM DO CM DAS FIGURAS 35

APÊNDICE 6 - ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA DESTACAR OS PONTOS DA BORDA DA DISTRIBUI ÇÃO DE DETRITOS ESPACIAIS.....36

APÊNDICE 7 - PROGRAMA FEITO POR SANDRO FELGUEIRAS CASTRO PARA GERAR A FORMA ANALÍTICA DA PROPAGAÇÃO DOS DETRITOS ESPACIAIS.....37

APÊNDICE 8 - 1° DESENVOLVIMENTO ANALÍTICO DO PROBLEMA.....39

LISTA DE FIGURAS

Página

Figura 1 - Ambiente próximo da Terra com todos os satélites catalogados até julho de 1987.....06

Figura 2: População de detritos observáveis em dezembro de 1996, conforme dados da NASA e do US Space Command.....07

Figura 3 – Simulações geradas pelo programa KK2MAT.C.....10

Figura 4 – Comparação entre o método analítico e o método numérico (t=0,6T).....11

Figura 5 - Comparação entre o método analítico e o método numérico (t=0,8T).....12

Figura 6 - Comparação dos centros de massa pelos métodos analítico e numérico.....12

Figura 7 - Transformação de coordenadas.....39

CAPÍTULO 1 – INTRODUÇÃO

A necessidade de descartar um dispositivo espacial usado após ele ter desempenhado sua missão foi levantada antes mesmo de o primeiro satélite ser lançado. Porém, até 1970, este assunto não recebeu muita atenção por parte da comunidade científica. Somente com a série de explosões de segundos estágios do foguete Delta desde fins dos 1960s até dezembro de 1973, é que as atividades de pesquisa nesta área foram impulsionadas. Em 1978 um satélite de defesa soviético caiu sem controle e deixou detritos radioativos no Ártico canadense. Em 1979 o laboratório espacial SkyLab deveria cair no Oceano Pacífico mas partes dele atingiram a costa da Austrália. Em 1991 a estação espacial Salyut 7, antecessora da estação espacial MIR, caiu nas montanhas do Andes. Ninguém foi atingido em nenhum desses casos. Em julho de 1996 o satélite operacional francês CERISI foi danificado quando um fragmento do corpo de um foguete Ariane colidiu com a haste de controle de atitude por gradiente de gravidade do satélite. Há evidências de impactos de detritos espaciais sobre painéis solares de satélites; e sobre o nariz , asas e janelas dos "Space Shuttle". Algumas quase colisões entre espaçonaves operacionais, incluindo os "Space Shuttle", e grandes detritos espaciais também já aconteceram.

À vista desses e de outros casos, em 1981, a NASA anunciou um plano de 10 anos para tratar temas-chave como medições, definição do ambiente, e controle de detritos espaciais. Desde outras agências espaciais e o COPUOS da ONU têm iniciado estudos similares, sobretudo considerando que o número dos satélites e dos detritos espaciais só tem aumentado. A Figura 1 mostra o ambiente próximo da Terra com todos os satélites catalogados em Julho/1987. A Figura 2 mostra a população de detritos observáveis em dezembro de 1996, conforme dados da NASA e "US Space Command".

No Brasil, desde 1977 a Divisão de Mecânica Espacial e Controle - DMC do Instituto Nacional de Pesquisas Espaciais - INPE realiza estudos sobre os movimentos (e o eventual controle) de **um** objeto artificial ou até natural em torno da Terra ou de outros corpos celestes. Desde pelo menos 1991 vários estudos vêm investigando tais movimentos iniciando com posições e/ou velocidades sujeitas a **erros/incertezas**; e prosseguindo sob a ação de forças e/ou torques sujeitos a erros/incertezas. Desde 1998 estes estudos vêm sendo estendidos para o movimento de **vários** objetos com uma origem comum e formando uma "nuvem" que se difunde e se deforma ao longo do movimento. Estes objetos são chamados **"detritos espaciais" ("space debris")** e podem ser de origem artificial ou natural (p.ex: a fragmentação de um satélite ou de um asteróide). Quando eles ocupam/interceptam órbitas/trajetórias de interesse (p.ex: a órbita geoestacionária), eles podem oferecer risco aos veículos que as utilizem. O número e o tamanho crescente dos "detritos espaciais" e dos veículos de interesse tem gerado a necessidade crescente de estudar os "detritos espaciais" visando reduzir a sua geração, apressar a sua remoção ou decaimento, e evitar colisões e manobras com os veículos de interesse. Parte destes estudos estão se transformando em **regulamentações restritivas** à construção, lançamento, operação, e descarte de veículos espaciais por todas as agências espaciais dos países mais experientes, e se tornarão em breve uma formidável barreira técnica e comercial para as agências espaciais dos países menos experientes, como o Brasil.

Todo esse contexto motivou este trabalho. Seu principal objetivo é estudar, modelar e simular a propagação numérica e semi-analítica de uma distribuição de detritos espaciais, que se movimentem ao redor da Terra. Com base nesses dados, objetiva-se posteriormente se estudar as propriedades básicas desse processo. Assim, poder-se-á analisar os problemas de colisão e interferência entre os detritos espaciais e outros objetos encontrados no espaço como satélites, ônibus espaciais, e estações espaciais.

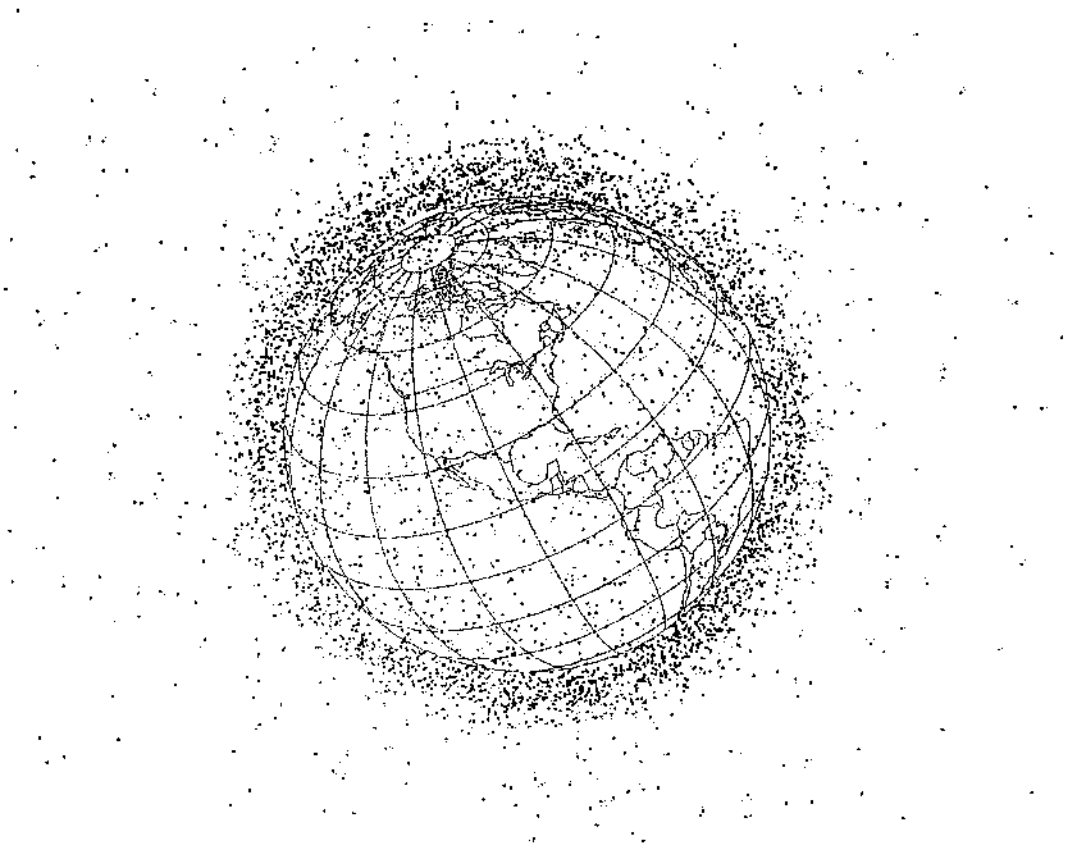

Figura 1: Ambiente próximo da Terra com todos os satélites catalogados até julho de 1987.

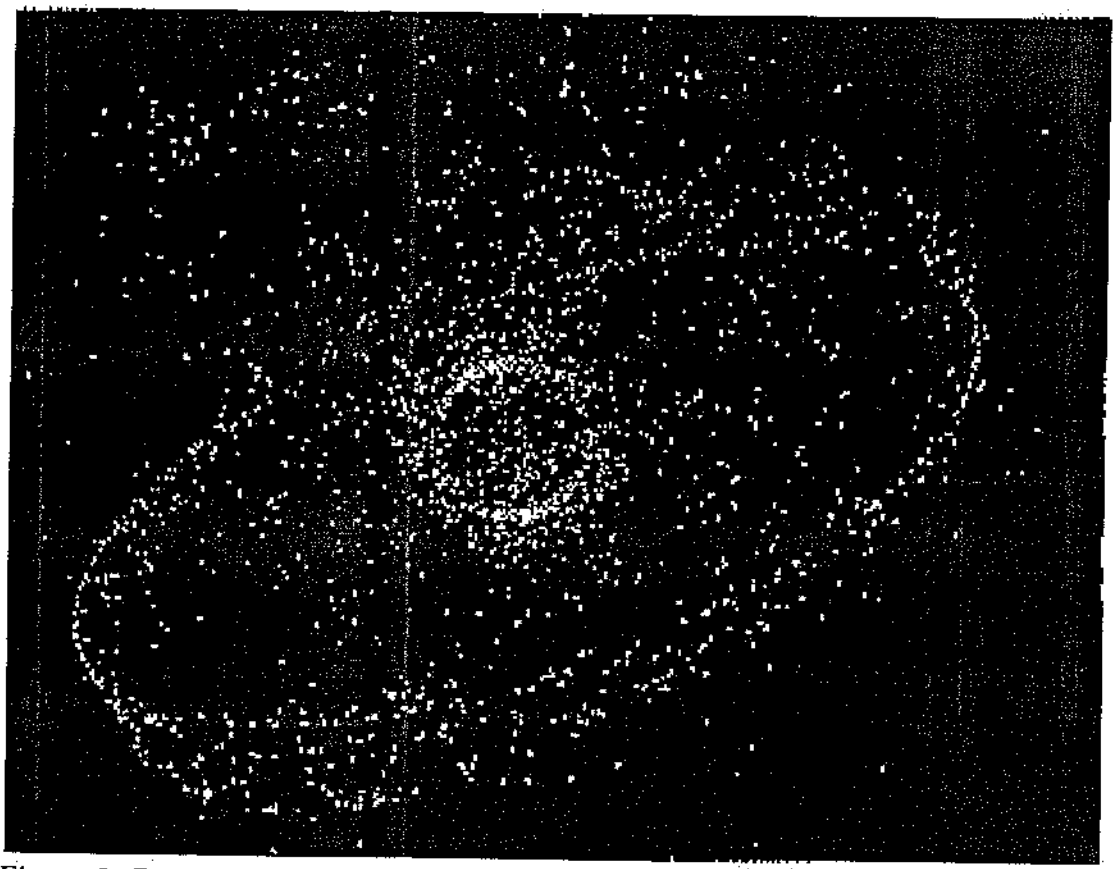

Figura 2: População de detritos observáveis em dezembro de 1996, conforme dados da NASA e do US Space Command.

CAPÍTULO 2 – OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo resumir e relatar o Projeto de Iniciação Científica de mesmo título realizado de 01/08/2002 a 31/07/2003, para simular a geração e a propagação de "Detritos Espaciais", para posterior estudo das propriedades básicas do processo. Com estas, pode-se estudar os problemas de colisão e interferência entre os "Detritos Espaciais" e outros objetos encontrados no espaço como satélites, ônibus espaciais, e estações espaciais.

Para tanto, **o 1° autor**: fez um estudo inicial em Mecânica Orbital e em erros numéricos e de integração de equações diferenciais pelo método de Runge-Kutta. A seguir, começou a simular detritos espaciais, adaptando um programa KK em linguagem C, originalmente feito para o sistema operacional UNIX, como base para essa simulação. Para melhor entendimento do programa KK, fez primeiramente uma leitura deste e a construção e posterior aperfeiçoamento de um fluxograma do mesmo. A seguir, adaptou o programa KK para rodar no sistema operacional Windows 2000 com auxílio do programa MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0. Além disso, adaptou a saída de dados do programa KK para torná-lo compatível com os programas/ambientes MATLAB, para utilizarmos sua capacidade de análise gráfica e estatística.

A partir de 01/02/2003 **o 2°. autor** assumiu o projeto fazendo um estudo inicial de Mecânica Orbital e Teoria de Probabilidades. A seguir, leu o 1° artigo do **orientador** (Souza e Nunes, 2000) sobre o tema para melhorar a compreensão sobre este e iniciou um estudo aprofundado do programa KK, já adaptado pelo 1° autor. Este programa gera as posições cartesianas de um número de detritos, ao longo do tempo, além das componentes x e y da velocidade de cada detrito. Essas coordenadas e as componentes de velocidade foram impressas num documento formato .txt. Para comparar com tal descrição numérica do movimento de "Detritos Espaciais", ele iniciou a programação e os testes de uma primeira descrição analítica daqueles proposta pelo orientador. Para tanto, fez um programa em C com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0, capaz de ler as coordenadas impressas pelo programa KK e parametrizar a propagação desses detritos. Os parâmetros utilizados nesse processo foram o tempo e as coordenadas do centro de atração gravitacional. Ele assumiu que cada detrito tinha a mesma velocidade angular constante. Depois disso, fez um estudo sobre o MATLAB, para utilizá-lo nesse processo.

Todos estes resultados foram satisfatórios e serão melhorados com a renovação deste Projeto de Iniciação Científica objetivando: 1) Calcular a estatística da distribuição de "Detritos Espaciais" e estudar a sua evolução no tempo, iniciando com a posição do Centro de Massa-CM; 2) Observar e interpretar as propriedades básicas de tal processo; 3) Melhorar e testar um modelo analítico simples (geométrico, cinemático, etc.) para a distribuição de detritos espaciais e sua evolução; compará-lo com as simulações; e aperfeiçoá-lo; 4) Se houver tempo, introduzir o arrasto atmosférico e refazer os itens anteriores; 5) Documentar o trabalho realizado e publicá-lo no SICINPE 2004.

CAPÍTULO 3 – METODOLOGIA

João Paulo Marques Reginato, em agosto de 2002, iniciou um estudo em Mecânica Orbital através da apostila de Kuga e Rao (1995). Como esta é uma excelente texto sobre Introdução à Mecânica Orbital, longo mas fácilmente disponível, suporemos que o leitor a tenha estudado completamente seu Capítulo sobre o Movimento Orbital; e não reproduziremos suas equações neste trabalho. Foi feito também um estudo de erros numéricos e de integração de equações diferenciais pelo método de Runge-Kutta através do livro de Carnahan, Luther e Wilkes (1976). Como este é uma excelente texto sobre Análise Numérica Aplicada, longo mas fácilmente disponível, suporemos que o leitor tenha estudado completamente seu Capítulo sobre erros numéricos e de integração de equações diferenciais pelo método de Runge-Kutta; e não reproduziremos suas equações neste trabalho.

Após essa etapa introdutória, ele começou a simulação de detritos espaciais, com a utilização de um programa chamado KK em linguagem C, feito por Nunes (2000) para o sistema operacional UNIX, como base para essa simulação. Para melhor entendimento do programa KK, foi feita primeiramente uma leitura e a construção de um fluxograma do mesmo e posterior aperfeiçoamento. A seguir, adaptou-se o programa KK para rodar no sistema operacional Windows 2000 com auxílio do programa MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0. Além disso, a saída de dados do programa KK foi adaptada para tornar-se compatível com os programas/ambientes MATLAB, para utilizar-se sua capacidade de análise gráfica e estatística.

Em fevereiro de 2003, Sandro Felgueiras Castro assumiu esse projeto de pesquisa. Ele iniciou um estudo em Mecânica Orbital, através também da apostila de Kuga e Rao (1995). Foram feitos todos os exercícios referentes aos capítulos 1 a 6. Após isso, foi iniciado um estudo de probabilidades e processos estocásticos, através do Livro do Meyer (1970). A seguir, leu o 1° artigo do **orientador** (Souza e Nunes, 2000) sobre o tema para melhorar a compreensão sobre este e iniciou um estudo aprofundado do programa KK, já adaptado pelo 1° autor. Este programa gera as posições cartesianas de um número de detritos, ao longo do tempo, além das componentes x e y da velocidade de cada detrito. Essas coordenadas e as componentes de velocidade foram impressas num documento formato .txt. Para comparar com tal descrição numérica do movimento de "Detritos Espaciais", ele iniciou a programação e os testes de uma primeira descrição analítica daqueles proposta pelo orientador. Para tanto, fez um programa em C com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0, capaz de ler as coordenadas impressas pelo programa KK e parametrizar a propagação desses detritos. Os parâmetros utilizados nesse processo foram o tempo e as coordenadas do centro de atração gravitacional. Ele assumiu que cada detrito tinha a mesma velocidade angular constante. Depois disso, fez um estudo sobre o MATLAB, para utilizá-lo nesse processo.

CAPÍTULO 4 – RESULTADOS E ANÁLISES

De acordo com os objetivos específicos do projeto, obtivemos os seguintes resultados **com o 1° bolsista**:

1) Familiarizar-se com o tema e com a literatura inicial sobre detritos espaciais.
Inicialmente foi feita a leitura e compreensão da apostila "Introdução à Mecânica Orbital" e posteriormente foi feito o gabarito de todos os exercícios referentes aos capítulos 1 a 6, concluído no final de Agosto. Além disso, foi utilizado o paper de Souza e Nunes (2000) para a melhor compreensão do tema.

2) Conhecer métodos básicos de análise de Detritos Espaciais.
Com o auxílio da literatura inicial, foi possível compreender algumas ferramentas básicas para a análise de Detritos Espaciais, como as três leis de Kepler, a equação de Kepler, as coordenadas cartesianas, polares e keplerianas, dentre outras. Através do livro "Apllied Numerical Methods" foi feito o estudo de erros numéricos, bem como de integração pelo método Runge-Kutta.

3) Simular a geração e propagação de Detritos Espaciais pela fragmentação de um satélite.
A partir de setembro o programa KK.C em linguagem C para simulação de detritos espaciais começou a ser utilizado. O programa, que simula a explosão de um satélite, foi inicialmente adaptado para rodar em sistema operacional Windows 2000, tarefa concluída em outubro com o KK2TEXTo.C. Com isso, o resultado da simulação pode ser visto através do programa Corel Draw® ou em formato PDF, utilizando o programa Adobe Acrobat Reader®.Outras adaptações do programa visam torná-lo mais confortável para o usuário, como a possibilidade do mesmo escolher os parâmetros da explosão, tais como número de pontos (fragmentos) do satélite ou o gradiente de explosão.

Na fase atual, o programa está adaptado para que a sua saída de dados seja compatível com o programa MATLAB, pois o mesmo possui um ferramental de análises muito grande, que facilita a compreensão do fenômeno estudado.
A Figura 3 mostra exemplos de simulações geradas pelo programa KK2MAT.C:

Figura 3.a – tempo = 0,0s

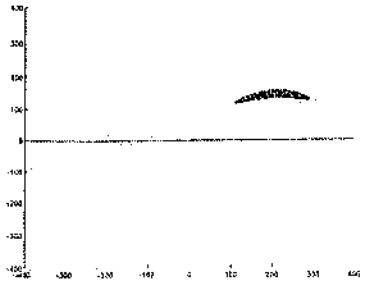

Figura 3.b – tempo = T/10 s

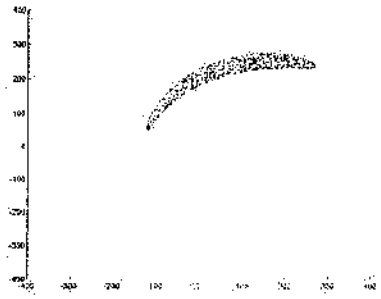

Figura 3.c – tempo = 2T/10 s

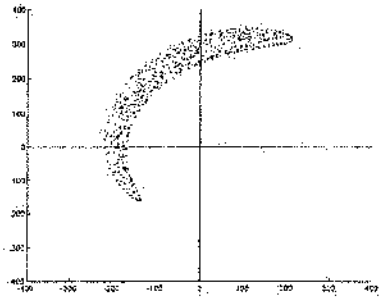

Figura 3.d – tempo = 3T/10 s

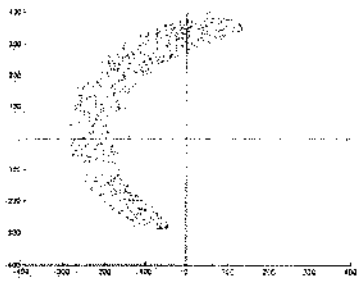

Figura 3.e – tempo = 4T/10 s

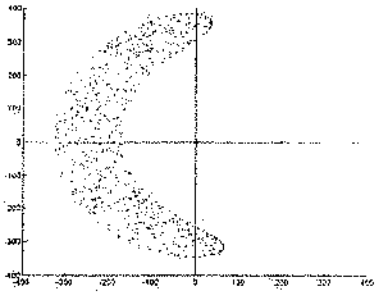

Figura 3.f – tempo = 5T/10 s

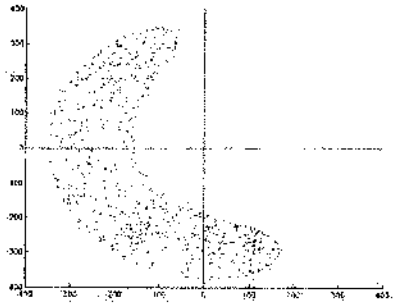

Figura 3.g – tempo = 6T/10 s

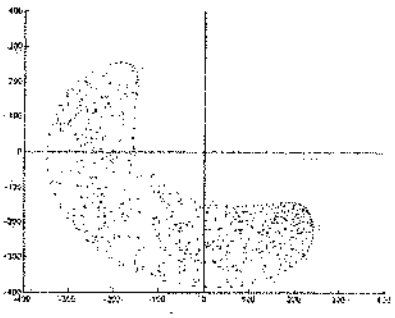

Figura 3.h – tempo = 7T/10 s

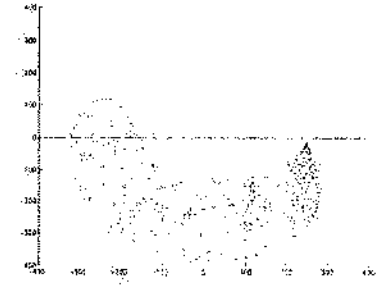

Figura 3.i – tempo = 8T/10 s

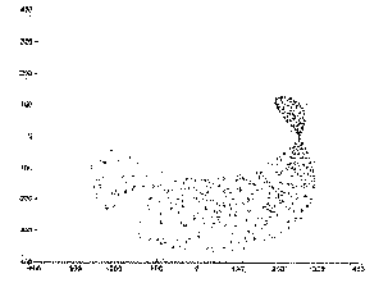

Figura 3.j – tempo = 9T/10 s

De acordo com os objetivos específicos do projeto, obtivemos os seguintes resultados **com o 2° bolsista**:

Este trabalho, iniciado em agosto de 2002, tem como objetivo simular a geração e propagação de "Detritos Espaciais", para fomentar o estudo posterior de suas propriedades básicas.

No período entre 1 de agosto de 2002 a 1 dezembro de 2002, o 1° bolsista João Paulo inicialmente desenvolveu um estudo de Mecânica Orbital, e iniciou à simulação de detritos espaciais com a utilização de um programa em linguagem C, para melhorar a interface de um programa em UNIX denominado KK. Este programa fora desenvolvido por Nunes (2000), e é responsável por gerar as posições sucessivas das partículas em estudo, ao longo do tempo. Esta adaptação foi feita com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0.

A partir de fevereiro de 2003, João Paulo não pode mais continuar seu trabalho. A partir de então, o 2° bolsista fez um estudo de Mecânica Orbital e Probabilidades. Em seguida, ele estudou o trabalho de Souza e Nunes (2000), para melhorar a compreensão sobre o tema abordado. Em março de 2003, iniciou a manipulação dos softwares STK4.3 e MASTER99, que possuem um banco de dados de "space debris", capazes de simular detritos em condições reais. Em abril de 2003, ele iniciou um estudo aprofundado do programa KK, adaptado por João Paulo. Este programa gera as posições cartesianas de um número de detritos, ao longo do tempo, além das componentes x e y da velocidade de cada detrito. Essas coordenadas e as componentes de velocidade foram impressas num documento formato .txt. Assim, em maio foi iniciada uma descrição analítica do movimento dos detritos. Para tanto, foi feito um programa em C com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0, capaz de ler as coordenadas impressas pelo programa KK e parametrizar a propagação desses detritos. Os parâmetros utilizados nesse processo foram o tempo e as coordenadas do centro de atração gravitacional. Assumiu-se que cada detrito tinha a mesma velocidade angular constante. Depois disso, foi feito um estudo sobre o MATLAB, para utilizá-lo nesse processo.

Os resultados dessa primeira parametrização foram satisfatórios. Na Figura 4 temos um pequeno exemplo da utilização dos recursos descritos anteriormente:

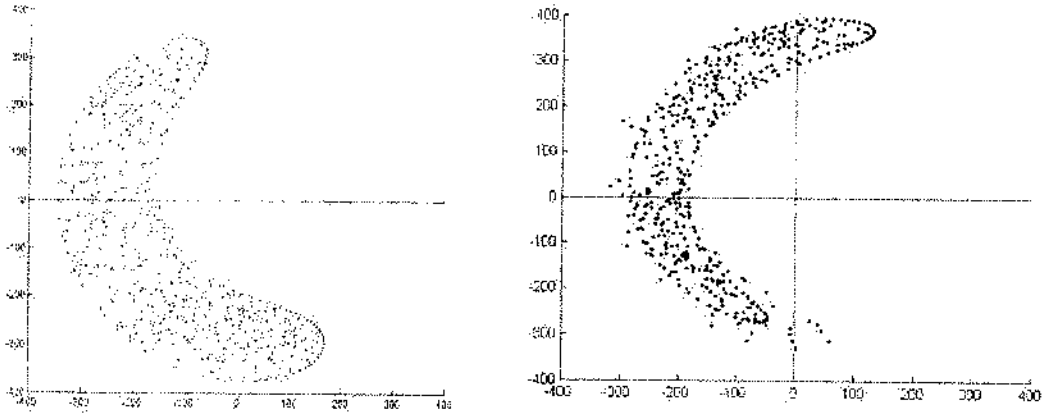

Fig.4.a – Método analítico (t=0,6T) Fig.4.b – Método numérico (t=0,6T)

Figura 4 – Comparação entre o método analítico e o método numérico (t=0,6T)

Nas Figuras 4.a e 4.b, a explosão foi feita considerando-se um satélite como um disco, contendo 100 pontos na borda e 500 pontos no interior. O tempo é 6T/10 para os dois casos, onde T é o período. Os parâmetros adimensionais utilizados na Fig.4.a foram: o centro de atração gravitacional = $(0;10.5R_t)$; a velocidade angular do sistema = 0.0001; o Raio da Terra $R_t = 1$; e o período da órbita $T = 2\pi$. O exemplo continua na Figura 5.

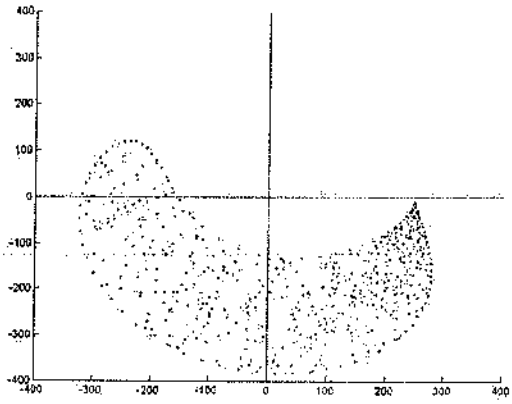

Fig.5.a – Método analítico (t=0,8T)

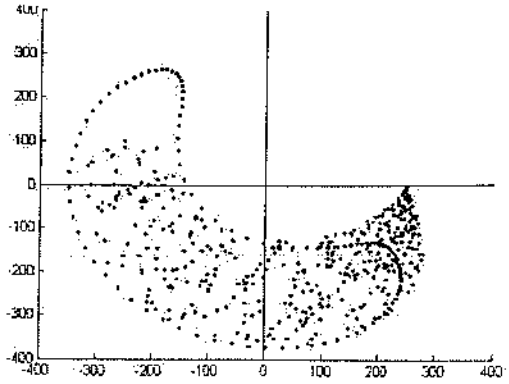

Fig.5.b – Método numérico (t=0,8T)

Figura 5 - Comparação entre o método analítico e o método numérico (t=0,8T)

Nas Figuras 5.a e 5.b, a explosão também foi idealizada para um satélite em forma de disco, com 100 pontos na borda e 500 pontos no interior. O tempo, nos dois casos, é 8T/10, T é o período. Os parâmetros utilizados na Fig.5.a foram: o centro de atração gravitacional = $(0;10.5R_t)$; velocidade angular do sistema = 0.0001 ; R_t = Raio da Terra.
Um estudo preliminar sobre a evolução do centro de massa dos detritos espaciais, considerando-se o método numérico e o método analítico, revelou bastante semelhança entre ambos os métodos. Isso pode ser comprovado na Figura 6. Nela, o símbolo '+' é relativo ao centro de massa utilizando-se o método numérico. O símbolo 'x' é referente ao método analítico.

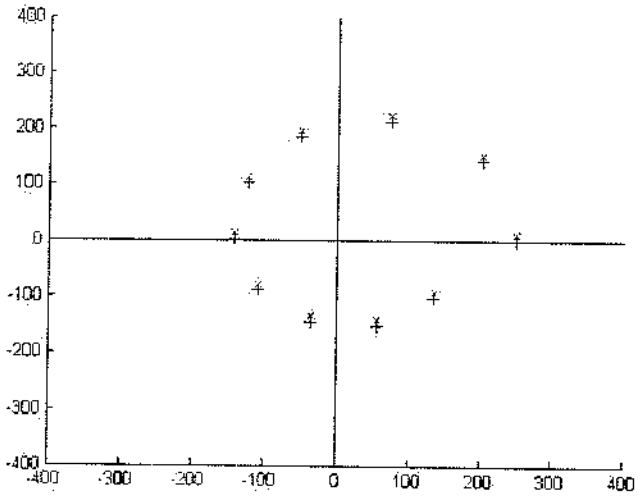

Figura 6: Comparação dos CM's - métodos analítico e o numérico

A análise das Figuras 4-6 revela que a parametrização utilizada, apesar provisória, já é capaz de reproduzir várias propriedades da propagação (forma "bananóide" com concavidade voltada para dentro, os pontos retornam à posição inicial da explosão após um período completo, etc). Assim, os próximos passos são refinar mais a parametrização utilizada, caracterizando-se a relação do centro de atração gravitacional com o tempo. Após isso, utilizar-se-á o método dos mínimos quadrados para se ajustar cada distribuição a seu respectivo centro de atração gravitacional.

CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve como objetivo resumir e relatar o Projeto de Iniciação Científica de mesmo título realizado de 01/08/2002 a 31/07/2003, para simular a geração e a propagação de "Detritos Espaciais", para posterior estudo das propriedades básicas do processo. Com estas, pode-se estudar os problemas de colisão e interferência entre os "Detritos Espaciais" e outros objetos encontrados no espaço como satélites, ônibus espaciais, e estações espaciais. Para tanto, **o 1° autor**: fez um estudo inicial em Mecânica Orbital e em erros numéricos e de integração de equações diferenciais pelo método de Runge-Kutta. A seguir, começou a simular detritos espaciais, adaptando um programa KK em linguagem C, originalmente feito para o sistema operacional UNIX, como base para essa simulação. Para melhor entendimento do programa KK, fez primeiramente uma leitura deste e a construção e posterior aperfeiçoamento de um fluxograma do mesmo. A seguir, adaptou o programa KK para rodar no sistema operacional Windows 2000 com auxílio do programa MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0. Além disso, adaptou a saída de dados do programa KK para torná-lo compatível com os programas/ambientes MATLAB, para utilizarmos sua capacidade de análise. A partir de 01/02/2003 **o 2°. autor** assumiu o projeto fazendo um estudo inicial de Mecânica Orbital e Teoria de Probabilidades. A seguir, leu o 1° artigo do **orientador** sobre o tema para melhorar a compreensão sobre este e iniciou um estudo aprofundado do programa KK, já adaptado pelo 1° autor. Este programa gera as posições cartesianas de um número de detritos, ao longo do tempo, além das componentes x e y da velocidade de cada detrito. Essas coordenadas e as componentes de velocidade foram impressas num documento formato .txt. Para comparar com tal descrição numérica do movimento de "Detritos Espaciais", ele iniciou a programação e os testes de uma primeira descrição analítica daquéles proposta pelo orientador. Para tanto, fez um programa em C com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0, capaz de ler as coordenadas impressas pelo programa KK e parametrizar a propagação desses detritos. Os parâmetros utilizados nesse processo foram o tempo e as coordenadas do centro de atração gravitacional. Ele assumiu que cada detrito tinha a mesma velocidade angular constante. Depois disso, fez um estudo sobre o MATLAB, para utilizá-lo nesse processo. Todos estes resultados foram satisfatórios e serão melhorados com a renovação deste Projeto de Iniciação Científica objetivando: 1) Calcular a estatística da distribuição de "Detritos Espaciais" e estudar a sua evolução no tempo, iniciando com a posição do Centro de Massa-CM; 2) Observar e interpretar as propriedades básicas de tal processo; 3) Melhorar e testar um modelo analítico simples (geométrico, cinemático, etc.) para a distribuição de detritos espaciais e sua evolução; compará-lo com as simulações; e a perfeiçoá-lo; 4) Se houver tempo, introduzir o arrasto atmosférico e refazer os itens anteriores; 5) Documentar o trabalho realizado e publicá-lo no SICINPE 2004.

REFERÊNCIAS:

KUGA, H. K., RAO, K. R., *Introdução à Mecânica Orbital,* INPE, São José dos Campos, SP, 1995.

SOUZA, M.L.O., NUNES, D., *Forecasting Space Debris Distribution: A Measure Theory Approach,* 51th. International Astronautical Congress–IAC. Rio de Janeiro, RJ, 2-6 Out.2000, Paper IAA-00-IAA.6.4.07.

CARNAHAN B., LUTHER H. A., WILKES J.O., *Applied Numerical Methods,* John Wiley & Sons, New York, NY, 1969.

MEYER, P.L., *Probabilidade-Aplicações à Estatística*, Ao Livro Técnico S.A, Rio de Janeiro, RJ, 1970.

APÊNDICE 1 – O PROGRAMA KK.C ORIGINAL FEITO POR DANTON NUNES EM C PARA ESTAÇÕES DE TRABALHO COM UNIX E POSTSCRIPT PARA PROPAGAR E PLOTAR OS DETRITOS ESPACIAIS

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define PI 3.1415926

typedef
        struct pto
        {
                struct pto *next,*prev;         /* Chain links          */
                double x,y,             /* Cartesian coors      */
                        ux,uy,          /* Cartesian velocities         */
                        r,theta,/* Polar coordinates   */
                        ur,utheta,      /* Polar velocities     */
                        a,              /* Semi-major axis      */
                        p,              /* Semi-latus rectum    */
                        e,              /* Eccentricity         */
                        K,              /* Angular momentum             */
                        E,              /* Total energy                 */
                        theta0,         /* Arg. of periapside   */
                        t0,             /* Time at periapside   */
                        T;              /* Period               */
        } point;

double now=0.0;         /* current time */

double Kepler (double M /* mean anomaly */, double e /* eccentricity */)
/* solves the Kepler equation by Newton-Raphson */
{
        double u,v; int n=10000; /* anti loop */
        v=M;   /* starting point. excellent guess for circular orbits. */
        do {
                u=v;
                v+=(M-u+e*sin(u))/(1.0-e*cos(u));
                n--;
        } while (u!=v && n);
        return u; /* eccentric anomaly */
}

void CtoP (point *p)
/* computes Polar things from Cartesian position and velocity */
{
        p->r = sqrt(p->x*p->x+p->y*p->y);
        p->theta = atan2(p->y,p->x);
        p->ur = (p->x*p->ux+p->y*p->uy)/p->r;
        p->utheta = p->K/p->r/p->r;
```

```
}

void PtoK (point *p)
/* computes Keplerian things from polar position and velocity */
{
        double T /* Kinetic Energy */ = (p->ux*p->ux+p->uy*p->uy)/2.0;
        double u,       /* eccentric anomaly */
               f,       /* true anomaly      */
               rrna2;  /* r*ur/(n*a2) */
        p->K /* angular momentum */ = (p->x*p->uy-p->y*p->ux);
        p->E /* total energy. always <0 for closed orbits */ = T-1.0/p->r;
        p->a /* semi-major axis */ = -1.0/(2.0*p->E);
        p->T /* period from 3rd. Kepler's law */ = 2.0*PI*sqrt(pow(p->a,3.0));
        rrna2 = p->r*p->ur*p->T/p->a/p->a;
        p->e /* eccentricity */ = sqrt(rrna2*rrna2+pow(1.0-p->r/p->a,2.0));
        p->p /* semi-latus rectum*/ = p->a*(1.0-p->e*p->e);
        u /* eccentric anomaly */ = atan2(rrna2,1.0-p->r/p->a);
        f /* true anomaly */ = atan2(sqrt(1-p->e*p->e)*sin(u),cos(u)-p->e);
        p->theta0 = p->theta - f;
        p->t0 = now - p->T * (u-p->e*sin(u) /* mean anom.*/)/(2.0*PI);
}

void KtoP (point *p)
/* computes polar coordinates & velocities from Keplerian constants + time */
{
        double u,       /* eccentric anomaly */
               f,       /* true anomaly      */
               M,       /* mean anomaly */
               xxx;
        M = (now - p->t0) * 2.0*PI / p->T;
        u /* eccentric anomaly */ = Kepler(M,p->e);
        f /* true anomaly */ = atan2(sqrt(1-p->e*p->e)*sin(u),cos(u)-p->e);
        p->theta = f + p->theta0;
        p->r = p->p/(1.0+p->e*cos(f));
        p->utheta = p->K/p->r/p->r;
        p->ur = p->e*p->r*p->r*sin(f)/p->p*p->utheta;
}

void PtoC (point *p)
/* Cartesianise polar things (my comments are getting sarchastic...) */
{
        p->x = p->r*cos(p->theta);
        p->y = p->r*sin(p->theta);
        p->ux = p->ur*cos(p->theta)-p->r*p->utheta*sin(p->theta);
        p->uy = p->ur*sin(p->theta)+p->r*p->utheta*cos(p->theta);
}


/* handling of linked chain of points */

void insertp(point *p /* point to be inserted... */, point *q /* ...here */)
```

```
/* inserts the point *p after *q */
{
        p->prev=q;
        p->next=q->next;
        q->next=p->next->prev=p;
}

/* plotting stuff */

double vscale=5.0;    /* velocity scale        */
double sscale=250.0; /* coordinate scale     */
int npath=0;

FILE *plotter=stdout;/* where to spit PostScript® out */

void psputs(char *s)
{
        fprintf(plotter,s);
}

void setcolour(double r, double g, double b)
{
        fprintf(plotter,"%5.2g %5.2g %5.2g setrgbcolor\n",r,g,b);
}
#define setcolor setcolour

void newpath()
{
        npath=1;
        psputs("newpath ");
}

void sewpoint(point *p)
{
        fprintf(plotter,"%g %g %s\n", p->x*sscale, p->y*sscale,
                        npath?"moveto":"lineto");
        npath=0;
}

void startplot()
{       /* gambiarra, so far */
        fprintf(plotter,"%g %g translate\n", sscale+50.0,sscale+100.0);
        fprintf(plotter,"%g %g moveto %g %g lineto stroke\n",
                        -sscale,0.0,+sscale,0.0);
        fprintf(plotter,"%g %g moveto %g %g lineto stroke\n",
                        0.0,-sscale,0.0,+sscale);
        setcolour(0.0,0.0,0.0);
}

void label(int n)
{
```

```
        fprintf(plotter,
                "/Courier findfont 22 scalefont setfont\n"
                "0.0 %g moveto (N.orb.: %d) show\n",
                -sscale-30.0, n);
}

void stopplot()
{       /* gambiarra, so far */
        psputs("showpage\n");
}

void segplot (point *p)
/* plots a tiny segment on p's position, proportional to its velocity */
{
        fprintf(plotter,"%g %g moveto ", p->x*sscale, p->y*sscale);
        fprintf(plotter,"%g %g rlineto stroke\n",
                        p->ux*vscale, p->uy*vscale);
}

/* ---- miscellanea ---- */

void usage()
{
        fprintf(stderr,
                "Kepler-Kolmogorov Kindergarten. © D.Nunes 2000\n"
                "       Arg     default         meaning\n"
                "-------------------------------------------------\n"
                "Parent body model\n"
                "       -r      1E-6            radius of parent body\n"
                "       -n      500             number of points (M.Carlo)\n"
                "       -p      100             number of border points\n"
                "       -w      1               rotation of parent body\n"
                "       -b      0               blast gradient\n"
                "Initial condition (centre of parent body). Only one of:\n"
                "       -C      1 0 0 1         Cartesian coords & velocity\n"
                "       -P      1 0 0 1         Polar coords & velocity\n"
                "       -K      <tbd>           Keplerian elements\n"
                "Simulation (times in terms of parent body's period)\n"
                "       -t      0.1             sampling interval\n"
                "       -T      1e2             total simulation time\n"
                "Plotting\n"
                "       -o      <stdout>        output file (PostScript)\n"
                "Miscellanea\n"
                "       -h      -               print this text\n"
                "       -s      1               seed of random numbers\n"
                "-------------------------------------------------\n"
                );
}


main(int argc, char **argv)
```

```
{
        point pbody    /* centre of parent body */
                = { &pbody, &pbody,         /* Links */
                    1.0, 0.0, 0.0, 1.0,     /* Cart. elements */
                    1.0, 0.0, 0.0, 1.0,     /* Polar elements */
                    1.0, 1.0, 0.0, 1.0, -0.5,   /* Kepler et al */
                    0.0, 0.0, 2*PI } ;
        double pbr=1e-5;        /* radius */
        int npts=500;           /* # of inner points */
        int nbps=100;           /* # of border points */
        int omega=1.0;              /* rotation of parent body */
        int blastg=1000;            /* blast gradient. kaboom would be a better name */
        double dt=0.1;          /* time interval */
        double T=1e2;           /* total simulation time */

/* parse arguments */
        char *progname=*argv++;
        while (--argc && *argv)
        {
                if (0[*argv] == '-' && ! 2[*argv]) switch(1[*argv++])
                {
                case 'r': pbr=atof(*argv++); argc--; break;
                case 'n': npts=atoi(*argv++); argc--; break;
                case 'p': nbps=atoi(*argv++); argc--; break;
                case 'w': omega=atof(*argv++); argc--; break;
                case 'b': blastg=atof(*argv++); argc--; break;
                case 't': dt=atof(*argv++); argc--; break;
                case 's': srand(atoi(*argv++)); argc--; break;
                case 'T': T=atof(*argv++); argc--; break;
                case 'h': usage(); exit(0);
                case 'C': pbody.x=atof(*argv++); argc--;
                          pbody.y=atof(*argv++); argc--;
                          pbody.ux=atof(*argv++); argc--;
                          pbody.uy=atof(*argv++); argc--;
                          CtoP(&pbody); PtoK(&pbody);
                          break;
                case 'P': pbody.r=atof(*argv++); argc--;
                          pbody.theta=atof(*argv++); argc--;
                          pbody.ur=atof(*argv++); argc--;
                          pbody.utheta=atof(*argv++); argc--;
                          PtoC(&pbody); PtoK(&pbody);
                          break;
                case 'K': fprintf(stderr,"-K is not implemented so far.\n"); exit(1);
                case 'o': if (!(plotter=fopen(*argv++,"w")))
                                { fprintf(stderr,"Could not open plotter.\n"); exit(1); }
                          argc--; break;
                default: fprintf(stderr,"What kind of shit is this: %s?\n", *argv);
                                exit(2);
                }
                else { fprintf(stderr,"Try %s -h\n", progname); exit(2); }
        }
```

```
/* make nbps border points. */
        {
                point *pt;
                double a;
                int i;
                for (i=0; i<nbps; i++)
                {
                        pt=(point *)malloc(sizeof(point));
                        if (!pt) exit(3);
                        a=i*2.0*PI/nbps;
                        pt->x = pbody.x + pbr*cos(a);
                        pt->y = pbody.y + pbr*sin(a);
                        insertp (pt, &pbody);
                }
        }

/* make npts random inner points */
        {
                point *pt;
                double a,r;
                int i;
                for (i=0; i<npts; i++)
                {
                        pt=(point *)malloc(sizeof(point));
                        if (!pt) exit(3);
                        a=2.0*PI*rand()/(RAND_MAX+1.0);
                        r=rand()/(RAND_MAX+1.0);
                        r=pbr*sqrt(r);
                        pt->x = pbody.x + r*cos(a);
                        pt->y = pbody.y + r*sin(a);
                        insertp (pt, &pbody);
                }
        }

/* make them move, i.e. calculate velocities */
        {
                point *pt;
                double x,y;
                for (pt=pbody.next; pt!=&pbody; pt=pt->next)
                {
                        x=pt->x-pbody.x;
                        y=pt->y-pbody.y;
                        pt->ux=pbody.ux + blastg*x - omega*y;
                        pt->uy=pbody.uy + blastg*y + omega*x;
                        CtoP(pt); PtoK(pt);

                }
        }
```

```
/* "now, the sport begins!" (W.Shakespeare, Henry V) */
        {
                double deltat=dt*pbody.T;
                double endt=T*pbody.T;
                int i;
                for (now=0.0; now<endt; now+=deltat)
                {
                        point *pt=&pbody;
                        startplot();
                        do {
                                KtoP(pt); PtoC(pt);
                                segplot(pt);
                                pt=pt->next;
                        } while (pt!=&pbody);
                        setcolour(1.0,0.0,0.0); newpath();
                        for (i=nbps, pt=pbody.prev; i; i--, pt=pt->prev) sewpoint(pt);
                        psputs("closepath stroke\n");
                        stopplot();
                }
        }

        fclose(plotter); exit(0);
}
```

APÊNDICE 2– O PROGRAMA KK2TEXTO.C FEITO POR JOÃO PAULO MARQUES REGINATO EM C PARA PCS COM WINDOWS 2000 E VISUAL STUDIO 6.0 PARA PROPAGAR E PLOTAR OS DETRITOS ESPACIAIS

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#define PI 3.14159265358979323846264338327795

typedef
        struct pto
        {
                struct pto *next,*prev;         /* Chain links          */
                double x,y,             /* Cartesian coors      */
                        ux,uy,          /* Cartesian velocities         */
                        r,theta,/* Polar coordinates   */
                        ur,utheta,      /* Polar velocities     */
                        a,              /* Semi-major axis      */
                        p,              /* Semi-latus rectum    */
                        e,              /* Eccentricity         */
                        K,              /* Angular momentum             */
                        E,              /* Total energy                 */
                        theta0,         /* Arg. of periapside   */
                        t0,             /* Time at periapside   */
                        T;              /* Period               */
        } point;



double now=0.0;       /* current time */

double Kepler (double M /* mean anomaly */, double e /* eccentricity */)
/* solves the Kepler equation by Newton-Raphson */
{
        double u,v; int n=10000; /* anti loop */
        v=M;  /* starting point. excellent guess for circular orbits. */
        do {
                u=v;
                v+=(M-u+e*sin(u))/(1.0-e*cos(u));
                n--;
        } while (u!=v && n);
        return u; /* eccentric anomaly */
}

void CtoP (point *p)
/* computes Polar things from Cartesian position and velocity */
{
        p->r = sqrt(p->x*p->x+p->y*p->y);
```

```
void insertp(point *p /* point to be inserted... */, point *q /* ...here */)
/* inserts the point *p after *q */
{
        p->prev=q;
        p->next=q->next;
        q->next=p->next->prev=p;
}

/* plotting stuff */

double vscale=5.0;    /* velocity scale        */
double sscale=250.0;  /* coordinate scale      */
int npath=0;

main(int argc, char **argv)
{
        FILE *plot=fopen("detritos.ps","w");
        char newpath[]="newpath ";
        char closepath[]="closepath stroke";
        char showpage[]="showpage";


        point pbody    /* centre of parent body */
                = { &pbody, &pbody,          /* Links */
                    1.0, 0.0, 0.0, 1.0,      /* Cart. elements */
                    1.0, 0.0, 0.0, 1.0,      /* Polar elements */
                    1.0, 1.0, 0.0, 1.0, -0.5,    /* Kepler et al */
                    0.0, 0.0, 2*PI } ;
        double pbr=1e-5;      /* radius */
        int npts=500;         /* # of inner points */
        int nbps=100;         /* # of border points */
        float omega=1;                /* rotation of parent body */
        float blastg=0;       /* blast gradient. kaboom would be a better name */
        double dt=0.1;        /* time interval */
        double T=1e1;         /* total simulation time */

/* interface do programa*/

        printf(" Qual eh o gradiente de explosao?\n");
        scanf("%f",&blastg);
        printf(" Qual eh o numero de pontos da borda?\n");
        scanf("%d",&nbps);
        printf(" Qual eh o numero de pontos interiores?\n");
        scanf("%d",&npts);
        printf(" Qual eh a velocidade angular do satelite?\n");
        scanf("%f",&omega);

/* make nbps border points. */
        {
                point *pt;
```

```
                double a;
                int i;
                for (i=0; i<nbps; i++)
                {
                        pt=(point *)malloc(sizeof(point));
                        if (!pt) exit(3);
                        a=i*2.0*PI/nbps;
                        pt->x = pbody.x + pbr*cos(a);
                        pt->y = pbody.y + pbr*sin(a);
                        insertp (pt, &pbody);
                }
        }

/* make npts random inner points */
        {
                point *pt;
                double a,r;
                int i;
                for (i=0; i<npts; i++)
                {
                        pt=(point *)malloc(sizeof(point));
                        if (!pt) exit(3);
                        a=2.0*PI*rand()/(RAND_MAX+1.0);
                        r=rand()/(RAND_MAX+1.0);
                        r=pbr*sqrt(r);
                        pt->x = pbody.x + r*cos(a);
                        pt->y = pbody.y + r*sin(a);
                        insertp (pt, &pbody);
                }
        }

/* make them move, i.e. calculate velocities */
        {
                point *pt;
                double x,y;
                for (pt=pbody.next; pt!=&pbody; pt=pt->next)
                {
                        x=pt->x-pbody.x;
                        y=pt->y-pbody.y;
                        pt->ux=pbody.ux + blastg*x - omega*y;
                        pt->uy=pbody.uy + blastg*y + omega*x;
                        CtoP(pt); PtoK(pt);

                }
        }

/* "now, the sport begins!" (W.Shakespeare, Henry V) */
        {
                double deltat=dt*pbody.T;
                double endt=T*pbody.T;
                int i;
```

```
        for (now=0.0; now<endt; now+=deltat)
        {
                point *pt=&pbody;
                fprintf(plot,"%g %g translate\n", sscale+50.0,sscale+100.0);
                fprintf(plot,"%g %g moveto %g %g lineto stroke\n",
                        -sscale,0.0,+sscale,0.0);
                fprintf(plot,"%g %g moveto %g %g lineto stroke\n",
                        0.0,-sscale,0.0,+sscale);
                fprintf(plot,"%5.2g %5.2g %5.2g setrgbcolor\n",0.0,0.0,0.0);
                fprintf(plot,
                        "/Courier findfont 14 scalefont setfont\n"
                        "0.0 %g moveto (time: %f) show\n",
                        -sscale-30.0, now);
                do {
                        KtoP(pt); PtoC(pt);
                        fprintf(plot,"%g %g moveto ", pt->x*sscale, pt->y*sscale);
                        fprintf(plot,"%g %g rlineto stroke\n",
                        pt->ux*vscale, pt->uy*vscale);
                        pt=pt->next;
                } while (pt!=&pbody);
                fprintf(plot,"%5.2g %5.2g %5.2g setrgbcolor\n",1.0,0.0,0.0);
                npath=1;
                fprintf(plot,"%s",newpath);
                for (i=nbps, pt=pbody.prev; i; i--, pt=pt->prev)
                        {
                        fprintf(plot,"%g %g %s\n", pt->x*sscale, pt->y*sscale,
                        npath?"moveto":"lineto");
                        npath=0;
                        }
                fprintf(plot,"%s",closepath);
                fprintf(plot,"\n%s\n",showpage);
        }
}

printf("\n\n Foi criado o arquivo 'detritos.ps' com a figura desejada!");
printf("\n Aperte qualquer tecla para encerrar.\n");
getch();
fclose(plot); exit(0);
                                        }
```

APÊNDICE 3 – O PROGRAMA KK2MAT.C FEITO POR JOÃO PAULO MARQUES REGINATO PARA PCS COM WINDOWS 2000 E MATLAB PARA PROPAGAR E PLOTAR OS DETRITOS ESPACIAIS

I - INTRODUÇÃO

Esse relatório visa esclarecer o funcionamento do programa KK2MAT.C que simula a explosão de um satélite e as mudanças feitas no mesmo para atender algumas necessidades no desenvolvimento do Projeto de Iniciação Científica "Análise e Simulação de Detritos Espaciais".

II – O PROBLEMA DA VISUALIZAÇÃO DAS FIGURAS DE SIMULAÇÃO

Primeiramente, foi feita a tentativa de simulação utilizando o programa KK.C original de Danton Nunes (Apêndice 1). Esse programa foi construído em sistema operacional UNIX e tem saída para POSTSCRIPT. Para tanto, seria necessária a utilização de algum visualizador dessa linguagem, como por exemplo o GHOSTVIEW. Um exemplo de saída em POSTSCRIPT pode ser vista na seguinte parte do programa:

```
void segplot (point *p)
{
        fprintf(plotter,"%g %g moveto ", p->x*sscale, p->y*sscale);
        fprintf(plotter,"%g %g rlineto stroke\n",
                        p->ux*vscale, p->uy*vscale);
}
```

Podemos ver a utilização de comandos como "moveto" e "rlineto", que são apropriados para essa linguagem.

O programa KK.C apresenta certo grau de dificuldade no seu entendimento por utilizar uma linguagem mais pesada do que os alunos de graduação estão acostumados a lidar, além de utilizar termos de uma nova linguagem, o POSTSCRIPT.

A utilização de um visualizador não foi concretizada e, com isso, foi feita uma nova tentativa que foi melhor sucedida. Rodando o programa KK.C no sistema operacional WINDOWS 2000 e com a utilização do MICROSOFT VISUAL STUDIO pode ser feito o seguinte procedimento: Ao ser executado, o programa gera o arquivo kk.exe que contém os códigos em POSTSCRIPT. Em ambiente DOS, utilizamos o seguinte comando:

```
kk.exe > kk.ps
```

Esse comando transforma o arquivo em figuras que podem ser visualizadas com o auxílio do programa COREL DRAW. Outra possibilidade é a utilização do ACROBAT DISTILLER, para transformar as figuras em PDF para visualização no programa ADOBE ACROBAT READER.

Nessa etapa, o programa gerava as figuras com as condições iniciais já embutidas no mesmo, só permitindo ao usuário que desejasse mudanças nessa condições fazê-las diretamente no código do programa. Para otimizar o programa eram necessárias algumas alterações. Para tanto, foram retiradas algumas linhas de comando que não eram mais necessárias e feitas algumas adaptações, visto que o sistema operacional não era mais o UNIX e sim o WINDOWS 2000. A nova versão, chamada KK2TEXTO.C encontra-se no Apêndice 2.

Essa versão já permitia ao usuário a escolha de algumas condições diretamente na tela de execução do programa. Com isso, o usuário fica mais confortável para o estudo da explosão. Além disso, o programa já tinha como saída o arquivo DETRITOS.PS, não sendo mais necessária a utilização do DOS. O próximo passo seria possibilitar ao usuário a escolha de todas as condições do programa e a introdução de "janelas" para tornar o programa mais amigável.

III – ADAPTAÇÃO PARA O MATLAB

Para uma melhor análise da explosão do satélite surgiu a necessidade da compatibilização dos dados com o programa MATLAB, que possui grande capacidade de análise gráfica e estatística, necessária para o projeto. Para tanto, foram feitas alterações no KK2TEXTO.C e a nova versão foi chamada KK2MAT.C.

Nessa versão, a saída de dados foi totalmente alterada, visto que não era mais necessário o POSTSCRIPT, e sim uma tabela de dados com as coordenadas X e Y dos pontos do satélite em cada instante depois da explosão, chamada MAT.TXT, tornando-se dado de entrada do MATLAB.

IV – ENTENDENDO O PROGRAMA KK2MAT.C (PASSO A PASSO)

Para que haja um maior entendimento do programa KK2MAT.C, vamos explicar alguns de seus comandos. A listagem do programa está completa.

1)

```
typedef
        struct pto
        {
                struct pto *next,*prev;         /* Chain links          */
                double x,y,                     /* Cartesian coors      */
                       vx,vy;                   /* Cartesian velocities */
                       r,theta,                 /* Polar coordinates    */
                       vr,vtheta,               /* Polar velocities     */
                       a,                       /* Semi-major axis      */
                       p,                       /* Semi-latus rectum    */
                       e,                       /* Eccentricity         */
                       K,                       /* Angular momentum     */
                       E,                       /* Total energy         */
                       theta0,                  /* Arg. of periapside   */
                       t0,                      /* Time at periapside   */
                       T;                       /* Period               */
        } point;
```

Nessas linhas temos apenas a inicialização do ponto, com todos os seus parâmetros. Vale ressaltar que o comando *struct pto *next, *prev* faz a ligação entre o ponto anterior e o posterior para ser feita a figura de simulação.

2)

```
double Kepler (double M /* mean anomaly */, double e /* eccentricity */)
/* solves the Kepler equation by Newton-Raphson */
{
        double u,v; int n=10000; /* anti loop */
        v=M;  /* starting point. excellent guess for circular orbits. */
        do {
```

```
                u=v;
                v+=(M-u+e*sin(u))/(1.0-e*cos(u));
                n--;
        } while (u!=v && n);
        return u; /* eccentric anomaly */
}

void CtoP (point *p)
/* computes Polar things from Cartesian position and velocity */
{
        p->r = sqrt(p->x*p->x+p->y*p->y);
        p->theta = atan2(p->y,p->x);
        p->ur = (p->x*p->ux+p->y*p->uy)/p->r;
        p->utheta = p->K/p->r/p->r;
}

void PtoK (point *p)
/* computes Keplerian things from polar position and velocity */
{
        double T /* Kinetic Energy */ = (p->ux*p->ux+p->uy*p->uy)/2.0;
        double u,       /* eccentric anomaly */
               f,       /* true anomaly      */
               rrna2;  /* r*ur/(n*a2) */
        p->K /* angular momentum */ = (p->x*p->uy-p->y*p->ux);
        p->E /* total energy. always <0 for closed orbits */ = T-1.0/p->r;
        p->a /* semi-major axis */ = -1.0/(2.0*p->E);
        p->T /* period from 3rd. Kepler's law */ = 2.0*PI*sqrt(pow(p->a,3.0));
        rrna2 = p->r*p->ur*p->T/p->a/p->a;
        p->e /* eccentricity */ = sqrt(rrna2*rrna2+pow(1.0-p->r/p->a,2.0));
        p->p /* semi-latus rectum*/ = p->a*(1.0-p->e*p->e);
        u /* eccentric anomaly */ = atan2(rrna2,1.0-p->r/p->a);
        f /* true anomaly */ = atan2(sqrt(1-p->e*p->e)*sin(u),cos(u)-p->e);
        p->theta0 = p->theta - f;
        p->t0 = now - p->T * (u-p->e*sin(u) /* mean anom.*/)/(2.0*PI);
}

void KtoP (point *p)
/* computes polar coordinates & velocities from Keplerian constants + time */
{
        double u,       /* eccentric anomaly */
               f,       /* true anomaly      */
               M;       /* mean anomaly */
               M = (now - p->t0) * 2.0*PI / p->T;
        u /* eccentric anomaly */ = Kepler(M,p->e);
        f /* true anomaly */ = atan2(sqrt(1-p->e*p->e)*sin(u),cos(u)-p->e);
        p->theta = f + p->theta0;
        p->r = p->p/(1.0+p->e*cos(f));
        p->utheta = p->K/p->r/p->r;
        p->ur = p->e*p->r*p->r*sin(f)/p->p*p->utheta;
}
```

```
void PtoC (point *p)
/* Cartesianise polar things (my comments are getting sarchastic...) */
{
        p->x = p->r*cos(p->theta);
        p->y = p->r*sin(p->theta);
        p->ux = p->ur*cos(p->theta)-p->r*p->utheta*sin(p->theta);
        p->uy = p->ur*sin(p->theta)+p->r*p->utheta*cos(p->theta);
}
```

Essas são as sub-rotinas que fazem as transformações de coordenadas no programa. A sub-rotina Kepler resolve a equação de Kepler pelo método de Newton-Raphson, enquanto que as sub-rotinas CtoP, PtoK, KtoP, PtoC, fazem as mudanças de coordenadas cartesianas para polares, polares para keplerianas, keplerianas para polares e polares para cartesianas, respectivamente.

3)

```
double vscale=5.0;    /* velocity scale        */
double sscale=250.0; /* coordinate scale     */
int npath=0;

main(int argc, char **argv)
{
        FILE *plot=fopen("mat.txt","w");

        point pbody    /* centre of parent body */
                = { &pbody, &pbody,          /* Links */
                    1.0, 0.0, 0.0, 1.0,      /* Cart. elements */
                    1.0, 0.0, 0.0, 1.0,      /* Polar elements */
                    1.0, 1.0, 0.0, 1.0, -0.5,    /* Kepler et al */
                    0.0, 0.0, 2*PI } ;
        double pbr=1e-5;       /* radius */
        int npts=500;          /* # of inner points */
        int nbps=100;          /* # of border points */
        float omega=1;                  /* rotation of parent body */
        float blastg=5000;  /* blast gradient. kaboom would be a better name */
        double dt=0.1;         /* time interval */
        double T=1.0;      /* total simulation time */
```

Essa parte do programa cria a tabela mat.txt que é utilizada como entrada de dados do MATLAB. Podemos ver as condições iniciais no código fonte do programa, tais como gradiente de explosão e numero de pontos.

4)

```
/* make nbps border points. */
        {
                point *pt;
                double a;
                int i;
                for (i=0; i<nbps; i++)
                {
                        pt=(point *)malloc(sizeof(point));
```

```
                    if (!pt) exit(3);
                    a=i*2.0*PI/nbps;
                    pt->x = pbody.x + pbr*cos(a);
                    pt->y = pbody.y + pbr*sin(a);
                    insertp (pt, &pbody);
            }
        }

/* make npts random inner points */
        {
            point *pt;
            double a,r;
            int i;
            for (i=0; i<npts; i++)
            {
                pt=(point *)malloc(sizeof(point));
                if (!pt) exit(3);
                a=2.0*PI*rand()/(RAND_MAX+1.0);
                r=rand()/(RAND_MAX+1.0);
                r=pbr*sqrt(r);
                pt->x = pbody.x + r*cos(a);
                pt->y = pbody.y + r*sin(a);
                insertp (pt, &pbody);
            }
        }

/* make them move, i.e. calculate velocities */
        {
            point *pt;
            double x,y;
            for (pt=pbody.next; pt!=&pbody; pt=pt->next)
            {
                x=pt->x-pbody.x;
                y=pt->y-pbody.y;
                pt->ux=pbody.ux + blastg*x - omega*y;
                pt->uy=pbody.uy + blastg*y + omega*x;
                CtoP(pt); PtoK(pt);

            }
        }
```

Nesta etapa, o programa calcula a posição e a velocidade dos pontos, tanto de borda como interiores.

```
5)
/* "now, the sport begins!" (W.Shakespeare, Henry V) */
        {
            double deltat=dt*pbody.T;
            double endt=T*pbody.T;
            int i;
            for (now=0.0; now<endt; now+=deltat)
```

```
        {
                point *pt=&pbody;
                do {
                        KtoP(pt); PtoC(pt);
                        fprintf(plot,"%g\t%g\n", pt->x*sscale, pt->y*sscale);
                        pt=pt->next;
                } while (pt!=&pbody);
                for (i=nbps, pt=pbody.prev; i; i--, pt=pt->prev)
                        {
                        fprintf(plot,"%g\t%g\n", pt->x*sscale, pt->y*sscale);
                        npath=0;
                        }
        }
}
```

Finalmente, essa parte do programa imprime a tabela mat.txt. Essa tabela tem duas colunas, sendo a primeira as coordenadas X do ponto e a segunda as coordenadas Y.

APÊNDICE 4: ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA PLOTAGEM DAS FIGURAS

```
function quebra2

load t1T.txt;
N=6010;
i=1;
while i<=N
  hold on;
  plot(t1T(i:i+500,1),t1T(i:i+500,2),'k.');
  plot(t1T(i+501:i+600,1),t1T(i+501:i+600,2),'r.');
  axis([-400 400 -400 400]);
  plot(-400:400,0,'k');
  plot(0,-400:400,'k');
  i=i+601;
  if i<N
    figure;
  end
end
```

APÊNDICE 5: ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA CÁLCULO E PLOTAGEM DO CM DAS FIGURAS

```
function cmassa

load mat.txt;
N=14020;
i=1;
while i<=N;
   hold on;
   xcm=sum(mat(i:i+600,1))/601;
   ycm=sum(mat(i:i+600,2))/601;
   cm=[xcm ycm];
   plot(cm(:,1),cm(:,2),'r.');
   axis([-400 400 -400 400]);
   plot(-400:400,0,'k');
   plot(0,-400:400,'k');
   i=i+701;
end
```

APÊNDICE 6: ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA DESTACAR OS PONTOS DA BORDA DA DISTRIBUIÇÃO DE DETRITOS ESPACIAIS

```
function cor

load zuera.c;
N=60100;
i=1;
while i<=N
   x=zuera(i:i+600,1);
   y=zuera(i:i+600,2);
   resp=corrcoef(x,y);
   resp
   i=i+601;
                                    end
```

APÊNDICE 7- PROGRAMA FEITO POR SANDRO FELGUEIRAS CASTRO PARA GERAR A FORMA ANALÍTICA DA PROPAGAÇÃO DOS DETRITOS ESPACIAIS

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

void main ( )
{

        int k ;
        int i;
        float t=0;
        long double positionx [10000];
        long double positiony [10000];
        long double centrox = 0;
        long double centroy = 0;
        long double theta [10000];
        long double distancia [10000];
        long double auxiliar [10000];

        FILE*dados;                    //dados.txt contain the cartesians cordinates from
numeric method
        FILE*saida;
        dados = fopen("Posicoes Cartesianas.txt","r");
        saida= fopen("saida.txt", "w");   //saida prints in .txt coordinates from parametric
method


        for(k=0;k<10000;k++)
        {
                positionx [k] = 0;
                positiony [k] = 0;
                distancia [k] = 0;
                auxiliar [k]= 0;
        }
        k = 0;
        for ( ;fscanf(dados,"%Lf",&positionx[k])!= -1; )
        {
                fscanf(dados,"%Lf",&positiony[k]);
                k ++;
        }
        i = k;
        printf("\nDigite a cordenada x do centro de atracao gravitacional \n(separador
decimal = ponto): ");
        scanf("%Lf",&centrox);
```

```
        printf("\nDigite a cordenada y  do centro de atracao gravitacional: ");
        scanf("%Lf",&centroy);

        for(k=0;k<i;k++)      //transfers the geometric center to gravitacional center
        {
                positionx[k]=positionx[k]+ centrox;
                positiony[k]=positiony[k]+ centroy;
        }

    for(k=0;k<i;k++)          //it calculetes the positional vector
        {
        auxiliar[k]= powl(positionx[k],2)+ powl(positiony[k],2);
        distancia[k]= sqrtl(auxiliar[k]);
        }
    for(k=0;k<i;k++)          //it calculates the initial angular position
        {
                if(positiony[k]!=0)
                        theta[k]=atan(positionx[k]/positiony[k]);
                else
                theta[k]=0;
    }
    printf("\nDigite qual e o instante desejado: ");
    scanf("%f", &t);
    for(k=0;k<i;k++)
    {
                if(sinl(theta[k])!=0)
                positionx[k]=positionx[k]*(sinl((0.0001)*t+ theta[k])/sinl(theta[k]));
                else
                {
                positionx[k]= distancia[k]*(sinl((0.0001)*t));
                        positiony[k]= distancia[k]*(cosl((0.0001)*t));
                }
                if(cosl(theta[k])!=0)
                positiony[k]=positiony[k]*(cosl((0.0001)*t+ theta[k])/cosl(theta[k]));
                else
                {
                positiony[k]= distancia[k]*(sinl((0.0001)*t));
                        positionx[k]= distancia[k]*(cosl((0.0001)*t));
                }

                fprintf(saida,"%Lf",positionx[k]);
                fprintf(saida,"\t%Lf\n",positiony[k]);
  }
        printf("\nFoi impresso um documento cujo (saida.txt) contendo as novas
cordenadas dos pontos");
        fclose(dados);
        fclose(saida);
        getch();

}
```

APÊNDICE 8 – 1" DESENVOLVIMENTO ANALÍTICO DO PROBLEMA

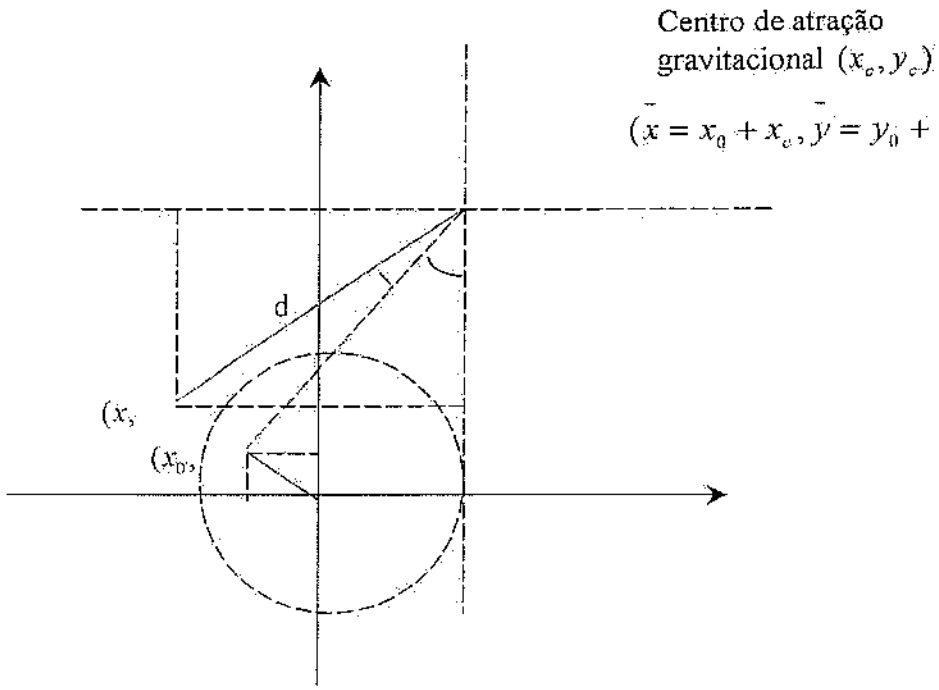

Figura 7: Transformação de coordenadas

Onde $\varphi = w$, w é a velocidade angular de capa partícula

Por problemas computacionais, oriundos do fato de w ser um número muito pequeno (da ordem de 1^{-37}) , tomou-se um valor fixo de w.

$$\bar{x} = R\,\mathrm{sen}(\varphi) \Rightarrow x = R\,\mathrm{sen}(\varphi + \theta)$$

$$\bar{y} = R\cos(\varphi) \Rightarrow y = R\cos(\varphi + \theta)$$

$$\frac{x}{\bar{x}} = \frac{\mathrm{sen}(\varphi + \theta)}{\mathrm{sen}(\varphi)} \Rightarrow x = \bar{x}\frac{\mathrm{sen}(\varphi + \theta)}{\mathrm{sen}(\varphi)}$$

$$\frac{y}{\bar{x}} = \frac{\cos(\varphi + \theta)}{\cos(\varphi)} \Rightarrow y = \bar{x}\frac{\cos(\varphi + \theta)}{\cos(\varphi)}$$