MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-13032-PRE/8309

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

Vanessa de Lima Takaoka*

*Bolsista UNIVAP

Relatório Final de Projeto de Iniciação Científica (PIBIC/CNPq/INPE), orientado pelo Dr. Marcelo Lopes de Oliveira e Souza

INPE
São José dos Campos
2005

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 – Vanessa de Lima Takaoka (UNIVAP, Bolsista PIBIC/CNPq)

E-mail: vanessatakaoka@gmail.com

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

E-mail: marcelo@dem.inpe.br

Junho de 2005

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

Vanessa de Lima Takaoka[1] (UNIVAP, Bolsista PIBIC/CNPq)

Dr. Marcelo Lopes de Oliveira e Souza[2] (DMC/ETE/INPE)

RESUMO

Este Relatório Final reporta e resume o trabalho da bolsista Vanessa de Lima Takaoka retomando, reformulando a execução e continuando o projeto, iniciado em 01 de Agosto de 2002, que tem como objetivo simular a geração e propagação de "Detritos Espaciais", para fomentar o estudo posterior de suas propriedades básicas.

Entre 01 de Agosto de 2002 e 31 de Janeiro de 2003, o 1° bolsista, João Paulo Marques Reginato, desenvolveu um estudo de Mecânica Orbital através da apostila de Kuga e Rao, e iniciou a simulação de detritos espaciais com um programa em linguagem C para PCs com sistema operacional Windows 2000 e linguagem gráfica OpenGL, por ele migrado a partir de um programa em C para estações com UNIX e PostScript feito pelo Eng. Danton Nunes.

Entre 01 de Fevereiro de 2003 e 01 de Dezembro de 2003 o 2° bolsista, Sandro Felgueiras Castro, deu continuidade ao projeto de Iniciação Científica em andamento desde 2002, para simular a geração e propagação de "Detritos Espaciais", visando fomentar o estudo posterior de suas propriedades básicas. Ele também desenvolveu um 1° modelo analítico do problema de propagação de detritos espaciais, e o cálculo e a plotagem dos Centros de Massa numérico e analítico.

Entre 01 de Dezembro de 2003 e 31 de Julho de 2004 o 3° bolsista, Anderson Patrick Alves Pereira, iniciou a elaboração e o estudo de um 2° modelo analítico que representaria melhor a propagação dos detritos espaciais. Esse modelo foi idealizado a partir das observações dos resultados do projeto de pesquisa precedente, no qual notou-se que a propagação dos detritos espaciais ocorria segundo a forma de uma elipse deformada ("bananóide"). Os passos desse modelo foram sendo simulados em ambiente MATLAB durante a sua construção chegando-se, até o presente momento, a uma elipse cujos eixos são girados enquanto o seu centro gira, segundo uma circunferência, em torno de um ponto (centro de massa).

Entre 01de março de 2005 e 31 de Julho de 2005 a 4ª bolsista, Vanessa de Lima Takaoka, assumiu esse projeto de pesquisa. Seguindo a orientação do Dr. Marcelo Lopes de Oliveira e Souza, Vanessa iniciou um estudo em Mecânica Orbital, também através da apostila de Kuga e Rao. A seguir, ela iniciou o estudo de tudo o que havia sido feito pelos 3 bolsistas anteriores visando retomar, reformular a execução e continuar o projeto anterior. Isto incluiu e enfatizou a compreeensão, a execução e o teste dos programas feitos, a elaboração de interfaces amigáveis para facilitar o seu uso e o prosseguimento do estudo e da plotagem do passos do 2° modelo analítico visando a sua comparação com o modelo numérico dos detritos espaciais. Esse modelo foi idealizado a partir das observações dos resultados do projeto de pesquisa precedente, no qual notou-se que a propagação dos detritos espaciais ocorria segundo a forma de uma elipse deformada ("bananóide"), cujos eixos cresciam segundo alguma taxa, ao mesmo tempo em que a elipse era girada e o seu centro girava em torno de um ponto (provavelmente o CM da Terra) segundo uma circunferência.

Se a continuidade do projeto for aprovada, será realizada uma curvatura dos eixos da elipse para que esta se aproxime da forma de um "bananóide". Os parâmetros do modelo serão ajustados para que este simule da melhor forma possível a propagação de detritos espaciais. Se houver tempo, tudo será repetido com arrasto atmosférico. Os resultados finais serão publicados em artigos nos Seminários de Iniciação Científica do INPE.

1 Aluna do Curso de Ciência da Computação da UNIVAP. E-mail: vanessatakaoka@gmail.com
2 Pesquisador Titular da Divisão de Mecânica Espacial e Controle do INPE. E-mail: marcelo@dem.inpe.br.

SUMÁRIO

Página

CAPÍTULO 1 – INTRODUÇÃO E MOTIVAÇÃO.....05
1.1 – INTRODUÇÃO..... 05
1.2 – MOTIVAÇÃO..... 06
CAPÍTULO 2 – OBJETIVOS E HISTÓRICO DO TRABALHO..... 07
2.1 – OBJETIVOS..... 07
2.2 – HISTÓRICO.....07
CAPÍTULO 3 –METODOLOGIA.....10
3.1 – INTRODUÇÃO.....10
3.2 – FORMULAÇÃO MATEMÁTICA E SISTEMA DE COORDENADAS.....10
3.3 - SIMULAÇÕES NUMÉRICAS.....11
3.4 REFORMULAÇÃO E FACILITAÇÃO DOS PROGRAMAS.....12
CAPÍTULO 4 – RESULTADOS E ANÁLISES.....13
CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS.....20
REFERÊNCIAS BIBLIOGRÁFICAS.....21
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.....22
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.....28
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.....32
APÊNDICE 4 - ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA PLOTAGEM DAS FIGURAS.....37
APÊNDICE 5 - ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA CÁLCULO E PLOTAGEM DO CM DAS FIGURAS 38
APÊNDICE 6 - ROTINA FEITA POR JOÃO PAULO MARQUES REGINATO EM MATLAB PARA DESTACAR OS PONTOS DA BORDA DA DISTRIBUIÇÃO DE DETRITOS ESPACIAIS.....39
APÊNDICE 7 - PROGRAMA FEITO POR SANDRO FELGUEIRAS CASTRO PARA GERAR A FORMA ANALÍTICA DA PROPAGAÇÃO DOS DETRITOS ESPACIAIS.....40
APÊNDICE 8 - 1° DESENVOLVIMENTO ANALÍTICO DO PROBLEMA.....42
APÊNDICE 9 - RELATÓRIO DO PROGRAMA ELIPSE DESENVOLVIDO POR PATRICK ALVES PEREIRA.....43

LISTA DE FIGURAS

Página

Figura 1 – Tela Principal do Programa kk2mat.c com os parâmetros de entrada.....14

Figura 2 – Resultado dos pontos gerados no arquivo texto (interface gráfica de saída).....15

Figura 3 – Interface gráfica do programa "propagacao.cpp".....16

Figura 4 – Elipse Simples centrada em (6,1).....17

Figura 5 – Giro da Elipse (sem crescimento dos eixos).....17

Figura 6 – Giro da elipse (com crescimento dos eixos).....,.17

Figura 7 – Rotação da elipse simples em (8,4).....,.18

Figura 8 – Giro da elipse em torno do CM associado à rotação da mesma.....18

Figura 9 – Giro de duas voltas da elipse associado à rotação da mesma.....18

Figura 10 – Forma "bananóide" a ser construída pelo modelo.....19

CAPÍTULO 1 – INTRODUÇÃO E MOTIVAÇÃO

1.1 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 o “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.

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.

1.2 MOTIVAÇÃO

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.

CAPÍTULO 2 – OBJETIVO E HISTÓRICO DO TRABALHO

2.1 OBJETIVO

Esse trabalho visou retomar, atualizar e continuar o projeto anterior, cujo objetivo principal era estudar, modelar, simular e comparar as propagações numérica e semi-analítica de uma distribuição de detritos espaciais que se movimentem ao redor da Terra.

Com base nesses dados, objetiva-se, posteriormente, estudar as propriedades básicas desse processo. Assim, será possível analisar os problemas de colisão e interferência dos detritos espaciais com outros objetos encontrados no espaço como satélites, ônibus espaciais, e estações espaciais.

2.2 HISTÓRICO

O projeto anterior, iniciado em agosto de 2002, teve como objetivo simular em PCs com Windows 2000 a geração e propagação de "Detritos Espaciais", para fomentar o estudo posterior de suas propriedades básicas.

No período entre 01 de agosto de 2002 a 31 janeiro de 2003, o 1° bolsista João Paulo Marques Reginato inicialmente desenvolveu um estudo de Mecânica Orbital através da apostila de Kuga e Rao (1995)., e iniciou a simulação de detritos espaciais com a utilização do programa KK2TEXTO.C ("Kepler-Kolmogorov 2 em texto") em linguagem C para PCs com sistema operacional Windows 2000 e linguagem gráfica OpenGL, que se encontra no Apêndice 2. Este programa KK2TEXTO.C foi adaptado do programa KK.C ("Kepler-Kolmogorov") em linguagem C para estações de trabalho Sun com sistema operacional UNIX e linguagem gráfica PostScript, que se encontra no Apêndice 1. Este programa KK.C foi desenvolvido pelo Eng. Danton Nunes (2000), e é responsável por gerar as posições sucessivas das partículas em estudo, ao longo do tempo. A adaptação KK2TEXTO.C foi feita com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0. Posteriormente, João Paulo fez uma outra adaptação, esta com saídas para o ambiente MATLAB através do programa KK2MAT.C ("Kepler-Kolmogorov 2 em Matlab") em linguagem C para PCs com sistema operacional Windows 2000 e MATLAB 5.x ou superior, que se encontra no Apêndice 3. E fez outros programas auxiliares, que se encontram nos Apêndice 4 a 6.

A partir de 01 de fevereiro de 2003, João Paulo não pode mais continuar seu trabalho. Então, o 2° bolsista Sandro Felgueiras Castro assumiu esse projeto de pesquisa. Ele iniciou um estudo em Mecânica Orbital, também através 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).que lhe permitiu fazer um estudo conjunto 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 KK2TEXTO.C, 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 em formato .txt. Assim, em maio foi proposto e testado um 1º modelo analítico do movimento dos detritos. Para tanto, foi feito um programa POSIÇÕESCARTESIANAS em C, que se encontra no Apêndice 7, com auxílio do MS Visual C++ 6.0 do ambiente MS Visual Studio 6.0, capaz de ler as coordenadas impressas pelo programa KK2TEXTO.C 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 por trechos, conforme equacionamento que se encontra no Apêndice 8. Depois disso, foi feito um estudo sobre o MATLAB, para utilizá-lo nesse processo de comparação com os resultados numéricos. Ainda neste projeto, Sandro calculou a estatística da distribuição de "Detritos Espaciais" e estudou a sua evolução no tempo, iniciando com a posição do Centro de Massa – CM.

Em 01 de dezembro de 2003 o 3° bolsista, Anderson Patrick Alves Pereira, assumiu esse projeto de pesquisa. Seguindo a orientação do Dr. Marcelo Lopes de Oliveira e Souza, Anderson iniciou um estudo em Mecânica Orbital, também através da apostila de Kuga e Rao (1995). A seguir, ele iniciou a elaboração e o estudo de um 2° modelo analítico que representaria a propagação dos detritos espaciais. Esse modelo foi idealizado a partir das observações dos resultados do projeto de pesquisa precedente, no qual notou-se que a propagação dos detritos espaciais ocorria segundo a forma de uma elipse deformada ("bananóide"), cujos eixos cresciam segundo alguma taxa, ao mesmo tempo em que a elipse era curvada e rotacionada e o seu centro girava em torno de um

ponto (provavelmente o CM da Terra) segundo uma circunferência. Para tanto, ele elaborou o programa ELIPSE.C que se encontra no Apêndice 9, e desenvolveu até 31/Julho/2004.

Em 01de março de 2005, a 4ª bolsista, Vanessa de Lima Takaoka, assumiu esse projeto de pesquisa e procedeu conforme a metodologia descrita a seguir.

CAPÍTULO 3 – METODOLOGIA

3.1 INTRODUÇÃO

Em 01 de março de 2005, a 4ª bolsista, Vanessa de Lima Takaoka, assumiu esse projeto de pesquisa. Seguindo a orientação do Dr. Marcelo Lopes de Oliveira e Souza, Vanessa iniciou um estudo em Mecânica Orbital, também através da apostila de Kuga e Rao (1995). A seguir, ela iniciou o estudo de tudo o que havia sido feito pelos 3 bolsistas anteriores visando retomar, atualizar a execução e continuar o projeto anterior. Isto incluiu e enfatizou a compreeensão, a execução e o teste dos programas feitos, a elaboração de interfaces amigáveis para facilitar o seu uso e o prosseguimento do estudo e da plotagem do 2° modelo analítico visando a sua comparação com o modelo numérico dos detritos espaciais. Esse modelo foi idealizado a partir das observações dos resultados do projeto de pesquisa precedente, no qual notou-se que a propagação dos detritos espaciais ocorria segundo a forma de uma elipse deformada ("bananóide"), cujos eixos cresciam segundo alguma taxa, ao mesmo tempo em que a elipse era rotacionada e o seu centro girava em torno de um ponto (provavelmente o CM da Terra) segundo uma circunferência. Para tanto ,seguiram-se o equacionamento e os passos a seguir.

3.2 - FORMULAÇÃO MATEMÁTICA E SISTEMA DE COOR DENADAS

No caso tridimensional podemos modelar uma nuvem de detritos espaciais como um conjunto de N partículas/objetos sem interação movendo-se num campo gravitacional central, cada uma delas sujeita à dinâmica em coordenadas inerciais O X_i, Y_i, Z_i dadas pelas leis de Newton, como feito por Contopoulos(1964). No caso bidimensional ($Z_i \equiv 0$, dropping i) estudado aqui, tal dinâmica se torna:$\forall t \in [t_0,t_f]$,

$$m.\dot{v}_t(t) = T.\cos\alpha(t) - m.v_r(t).\dot{f}(t) \quad (1)$$

$$\dot{f}(t) = \frac{v_t(t)}{r(t)} \quad (2)$$

$$\dot{r}(t) = v_r(t) \quad (3)$$

$$m.\dot{v}_r(t) = T.\text{sen}\,\alpha(t) - \frac{\mu.m}{r^2(t)} + m.v_t(t).\dot{f}(t) \quad (4)$$

que são as componentes transversal e radial da força resultante atuando em cada partícula, com condições iniciais dadas abaixo. As outras variáveis são: T= magnitude da

força aplicada; $\dot{v}_t(t), \dot{v}_r(t)$ = acelerações transversal e radial; $v_t(t), v_r(t)$ = velocidades transversal e radial; $\dot{f}(t)$ = velocidade angular; r(t) = vetor de posição do centro O à partícula com massa m.

Para modelar uma explosão radial $T(0) = \beta.R.\omega(t)$, modelaremos o corpo original como um disco homogêneo e girante no plano X_i, Y_i que é fragmentado em N partículas iguais. Cada uma destas partículas tem: a) vetor posição inicial (logo após a explosão) igual à sua posição final no corpo (logo antes da explosão); b) velocidade linear relativa (ao centro de massa = CM) inicial na direção transversal devida à rotação do disco $\omega(t)$ e proporcional ao raio relativo R; e na direção radial devida à explosão com gradiente β em relação ao raio relativo R; outra partícula simétrica para preservar o momento linear durante a explosão. As simulações foram realizadas inicialmente com N = 500 particles de modo a nos permitir estimar esperanças pelas médias aritméticas sobre 500 realizações (média sobre o ensemble) mais tarde. Este valor de N produziu convergência de todas as médias aritméticas testadas para os seus valores de regime permanente. Isto nos permitiu aplicar a equação de Kepler e as coordenadas Keplerianas a cada uma das partículas.

3.3 - SIMULAÇÕES NUMÉRICAS

Nas simulações, o corpo original descreve uma órbita circular com raio normalizado R=1 (adimensional) e período normalizado $T=2\pi$. Com esta normalização, o corpo original se torna um pequeno círculo com diâmetro 10^{-6}. Ele está girando com a velocidade angular orbital. Isto modela muito bem um satélite de 2 metros apontado para a Terra numa órbita baixa da Terra. Contudo, ele poderia representar qualquer coisa numa órbita circular. Além disto, a discussão a seguir enfatiza mais os aspectos qualitativos do movimento da nuvem de detritos; e números reais são de pouco ou nenhum interesse.

O programa KK e suas adaptações simula o movimento da fronteira inicial (**=envelope)** de um ensemble de N partículas partindo do corpo original com velocidades resultantes das velocidades de translação e rotação do satélite, e de uma explosão radial com gradiente β. Ela usa uma órbita circular de referência com raio r, período T, para um disco homogêneo e girante com raio R tal que, após uma mudança de variáveis conveniente, eles se tornam: r = 1 unidade de comprimento, $T = 2\pi$ unidades de tempo, e $R \cong 1 \times 10^{-6}$ r. As condições iniciais são: fronteira inicial = circunferência do disco,

velocidade radial inicial da borda do satélite = 10^{-2} ou 1% da velocidade orbital linear. O Programa KK.C e suas adaptações propaga o movimento do envelope inicial e o plota com período T/10.

O programa KK e suas adaptações simula também o movimento do interior inicial do mesmo ensemble de partículas, partindo do mesmo corpo original, com as mesmas condições iniciais, e com o mesmo gradiente de explosão.

3.4 - REFORMULAÇÃO E FACILITAÇÃO DOS PROGRAMAS

Os programas elaborados anteriormente pelos outros bolsistas – kk2.c, kk2mat.c, propagacao.cpp - foram reformulados para um ambiente mais "amigável" com o usuário, utilizando uma nova interface gráfica, já que os mesmos tinham de forma simples esta aplicação.

Alguns modelos foram simulados no ambiente MATLAB para estudo, ajustando os parâmetros necessários nos testes, com a meta de obtenção de aproximação dos dados anteriores.

Os vários modelos testados proporcionaram à bolsista um grau de conhecimento maior quanto ao assunto tratado. Sendo assim, os estudos realizados a partir da apostila de Kuga e Rao tornaram-se mais claros ao aplicá-los nessa fase de teste, que é por sinal importantíssima para os próximos objetivos dos estudos.

Enfim, objetivou-se com os novos modelos a inclusão de novas interfaces, buscando a facilitação da ferramenta na utilização das simulações dos debris, e simultâneamente a interação do assunto tratado no projeto.

CAPÍTULO 4 – RESULTADOS E ANÁLISES

De acordo com os objetivos específicos do projeto, obtivemos os seguintes resultados:

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 foram realizados os exercícios referentes aos capítulos da mesma. 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.

3) Simular a geração e propagação de Detritos Espaciais pela fragmentação de um satélite.

Com base nos programas desenvolvidos pelos bolsistas anteriores, foram compostas novas interfaces de entrada e saída para os mesmos, com a finalidade de aprimoramento e simultaneamente um profundo estudo da propagação dos detritos espaciais, resultando em uma melhor compreensão dos processos.

Todas as interfaces foram desenvolvidas com o auxílio MS Visual Basic 6.0 do ambiente MS Visual Studio 6.0, capaz de ler os parâmetros informados e repassá-los ao programa original, onde por sua vez se obtém as coordenadas X e Y que são repassadas para um arquivo texto. A partir desta fonte de dados do arquivo gerado, o sistema "lê" as coordenadas, gerando os pontos e plotando-as nos gráficos do mesmo programa.

As Figuras 1-3 abaixo demonstram o resultado das aplicações:

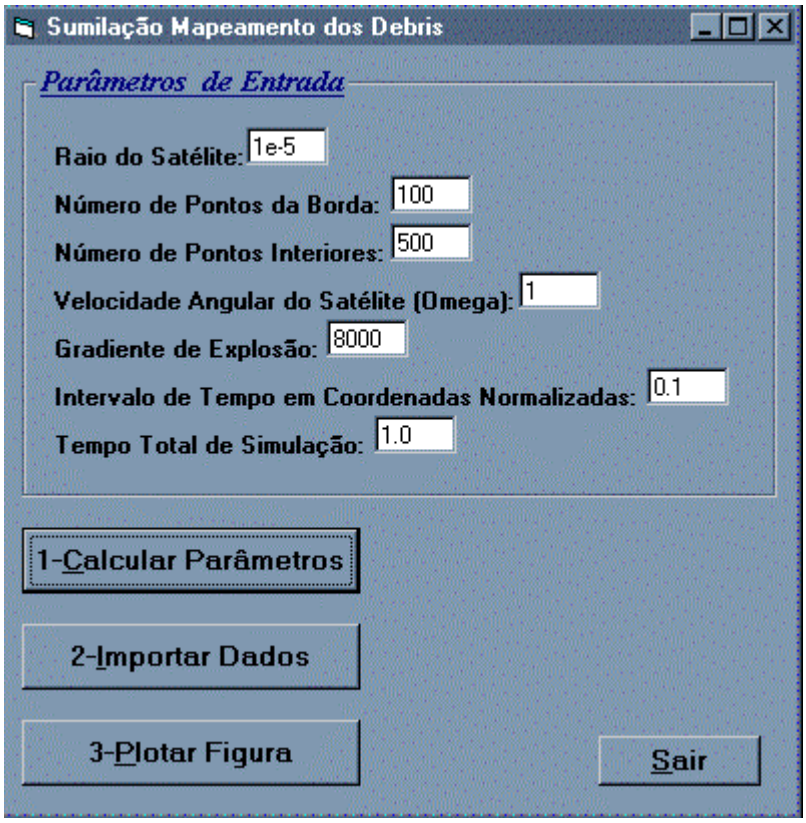

Figura 1 – Tela Principal do Programa kk2mat.c com os parâmetros de entrada

A Figura 1 representa a interface gráfica de entrada do programa kk2mat.c desenvolvido por Danton Nunes e aprimorado pelo bolsista João Paulo Marques Reginato. O programa calcula (botão de comando 1) por meio dos parâmetros informados, as coordenadas X e Y e as registram em um arquivo texto . Em seguida, o sistema importa e faz a conversão dos dados desse arquivo (botão de comando 2) para ser possível a sua visualização em forma gráfica. Finalmente as telas são impressas (botão de comando 3) e demonstradas com os instantes na Figura 2 a seguir:

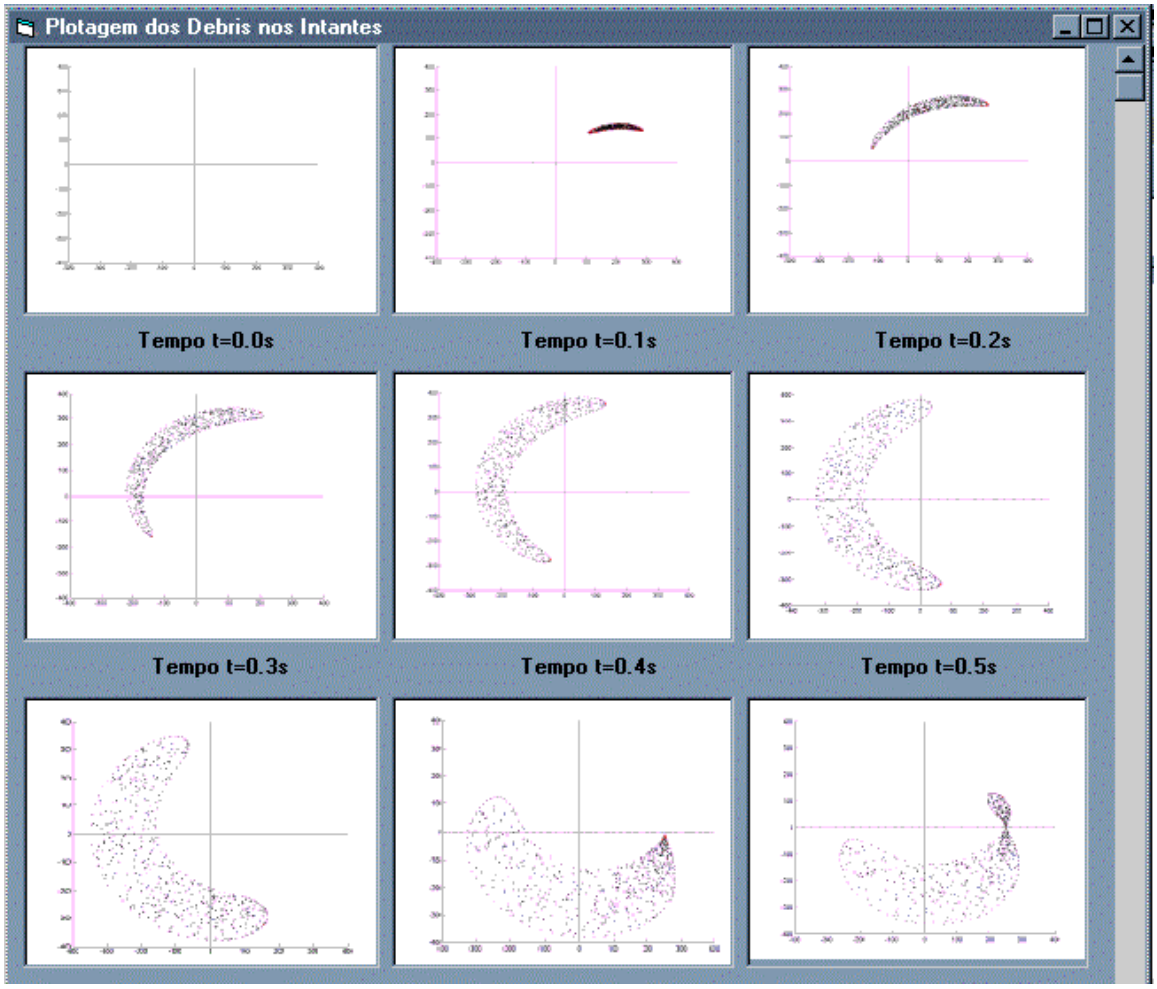

Figura 2 – Resultado dos pontos gerados no arquivo texto (interface gráfica de saída).

Na tentativa de se obter uma descrição analítica do movimento dos detritos, foi realizado 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.

Sendo assim, novamente com o auxílio do MS Visual Basic 6.0 é representada a interface gráfica de entradas e saídas do programa propagação.cpp (Figura 3) desenvolvido pelo segundo bolsista Sandro Felgueiras Castro.

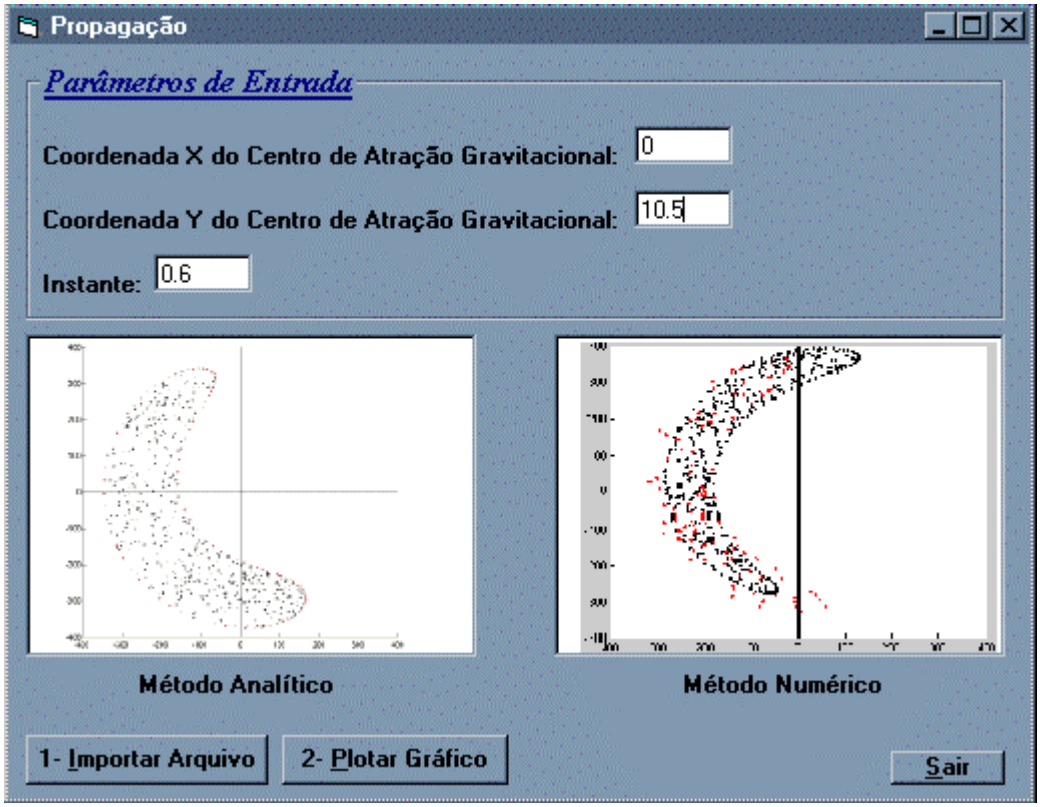

Figura 3 – Interface gráfica do programa "propagacao.cpp".

Na Figura 3, 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. 3 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 $T = 2\pi$.

Utilizando-se os conhecimentos de álgebra linear e o programa MATLAB, foi construído um modelo analítico simples para a propagação de detritos espaciais como explicado anteriormente.

No momento, está sendo estudada uma forma de se fazer com que os eixos da elipse também sejam rotacionados, para posterior implementação da próxima fase da simulação em MATLAB. As Figuras 4-10 a seguir mostram os resultados alcançados em cada fase da construção do programa:

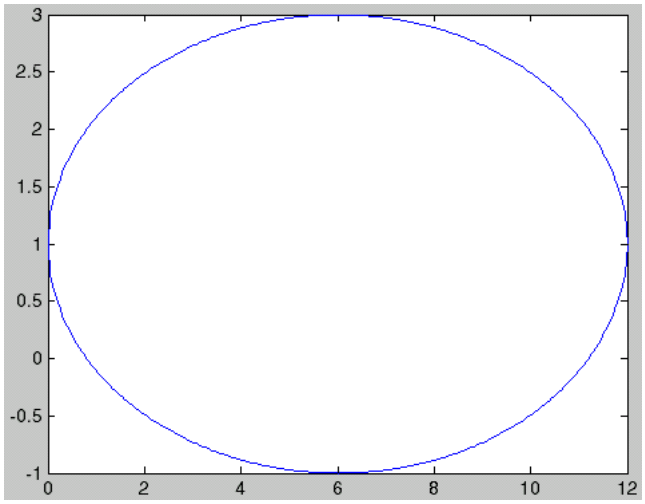

Figura 4 – Elipse simples centrada em (6,1).

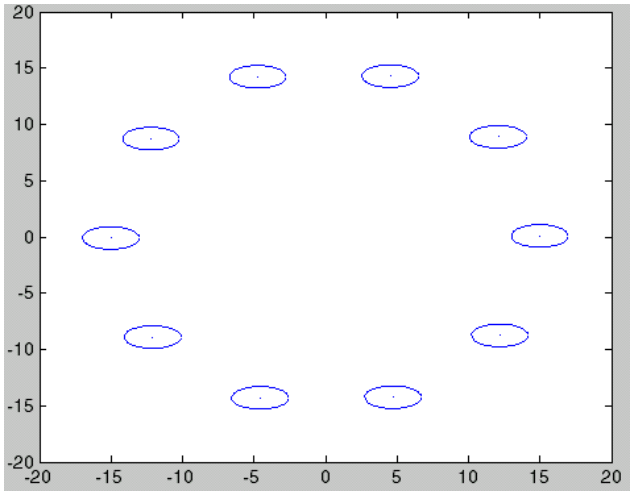

Figura 5 – Giro da elipse (sem crescimento dos eixos).

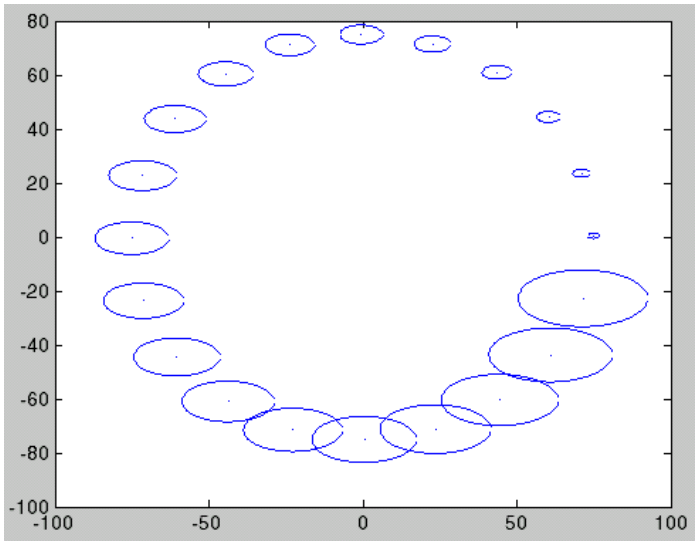

Figura 6 – Giro da elipse (com crescimento dos eixos).

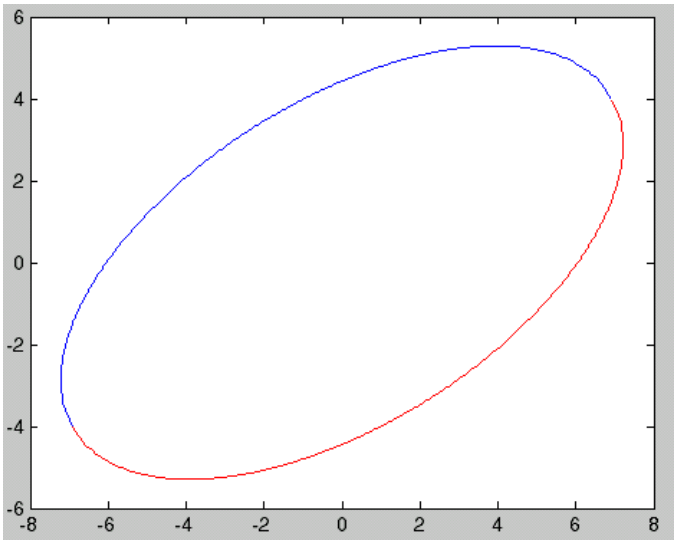

Figura 7 – Rotação da elipse simples em (8,4).

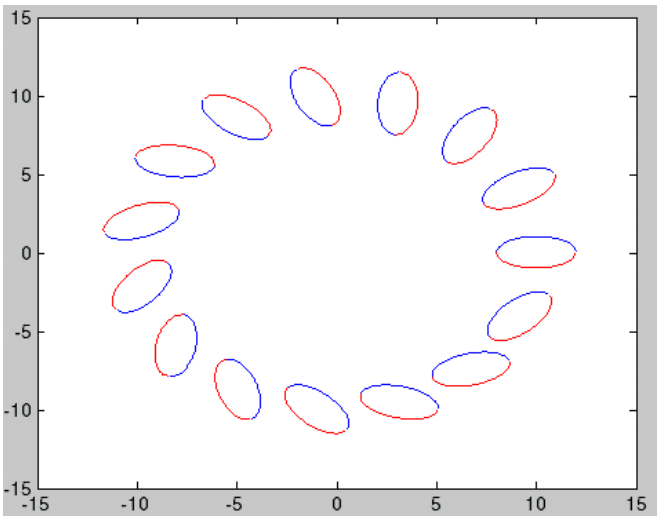

Figura 8 – Giro da elipse em torno do CM associado à rotação da mesma (15 números de pontos e 1 volta).

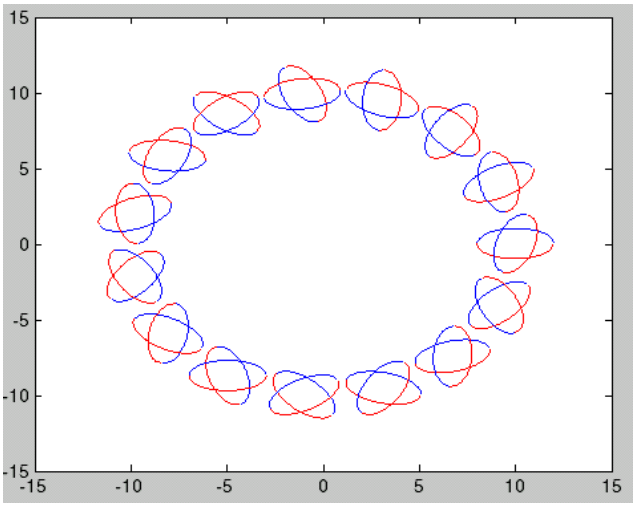

Figura 9 – Giro de duas voltas da elipse associado à rotação da mesma (15 números de pontos e 2 voltas).

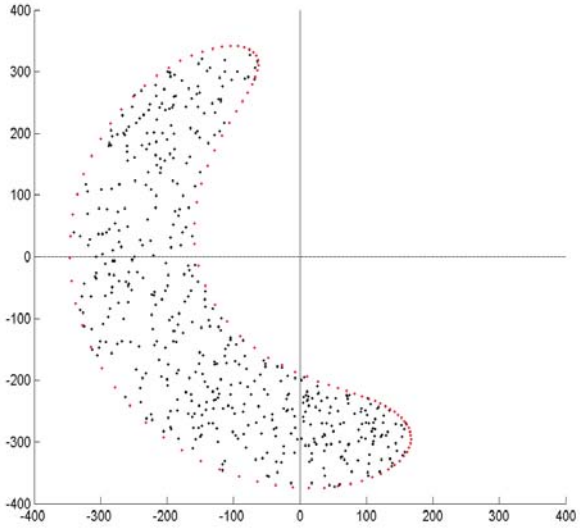

Figura 10 – Forma "bananóide" a ser construída pelo modelo

CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS

Esse trabalho visou retomar, atualizar e continuar o projeto anterior, cujo objetivo principal era estudar, modelar, simular e comparar as propagações numérica e semi-analítica de uma distribuição de detritos espaciais que se movimentem ao redor da Terra.

Para tanto, iniciou-se um estudo em Mecânica Orbital. A seguir, iniciou-se o estudo de tudo o que havia sido feito pelos 3 bolsistas anteriores visando retomar, atualizar a execução e continuar o projeto anterior. Isto incluiu e enfatizou a compreeensão, a execução e o teste dos programas feitos, a elaboração de interfaces amigáveis para facilitar o seu uso, e o prosseguimento do estudo e da plotagem do 2° modelo analítico visando a sua comparação com o modelo numérico dos detritos espaciais. Esse modelo foi idealizado a partir das observações dos resultados do projeto de pesquisa precedente, no qual notou-se que a propagação dos detritos espaciais ocorria segundo a forma de uma elipse deformada ("bananóide"), cujos eixos cresciam segundo alguma taxa, ao mesmo tempo em que a elipse era rotacionada e o seu centro girava em torno de um ponto (provavelmente o CM da Terra) segundo uma circunferência.

O trabalho feito até agora permitiu concluir que: os programas foram compreendidos, executados e testados com êxito; suas interfaces amigáveis foram elaboradas e utilizadas; e vários passos para estudo e a plotagem do 2° modelo analítico foram dados visando a sua comparação com o modelo numérico dos detritos espaciais.

Com base nestes programas e neste modelo, objetiva-se, posteriormente, estudar as propriedades básicas desse processo. Assim, será possível analisar os problemas de colisão e interferência dos detritos espaciais com outros objetos encontrados no espaço como satélites, ônibus espaciais, e estações espaciais.

REFERÊNCIAS BIBLIOGRÁFICAS

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

2) JOHNSON, N.L., & MCKINIGHT, D.S. *Artificial Space Debris (Updated Edition)*. Krieger Pub. Co., Malabar, FL, USA, 1991.

3) CHOBOTOV, V.A. (ed.) *Orbital Mechanics (2 Ed.)* Reston, VA, USA, AIAA, 1996.

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

5) ROSSER, J.B. (ED.) *Space Mathematics, Part I.* American Mathematical Society, New York, NY, USA, 1996.

6) CHANDRASEKHAR, S. Principles of Stellar Dynamics. Chicago Univ. Press, Chigaco, IL, USA, 1942; e Dover Pub., New York, NY, USA, 1960.

7) SIGMON, KERMIT, *MATLAB Primer,* 4th Edition, CRC Press, 1994.

8) HANSELMAN, D., LITTLEFIELD, B. MATLAB:Versão do Estudante: Guia do Usuário. Makron Books, São Paulo, 1997.

São José dos Campos, 17 de junho de 2005.

_____
Vanessa de Lima Takaoka /Bolsista

Certifico que a aluna Vanessa de Lima Takaoka e os seus trabalhos, realizados no período de 01/02/2005 a 31/07/2005, descritos neste Relatório foram plenamente satisfatórios.

_____
Marcelo Lopes de Oliveira e Souza/Orientador

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 3.14159265358979323846264338327 95

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 */
                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;

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.

1)

```
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;
```

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

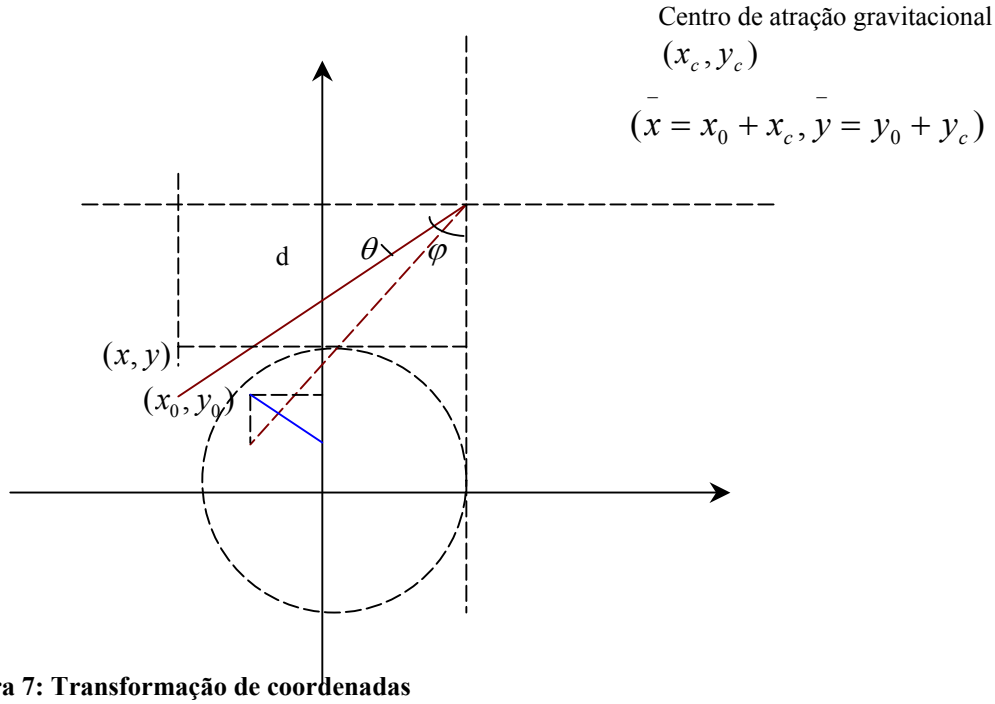

Figura 7: Transformação de coordenadas

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

Por problemas computacionais, oriundos do fato de w ser um número muito pequeno (da ordem de 10^{-370}) , tomou-se um valor fixo de w.

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

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

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

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

APÊNDICE 9- RELATÓRIO DO PROGRAMA ELIPSE DESENVOLVIDO POR ANDERSON PATRICK ALVES PEREIRA

A construção da elipse que melhor representa o espalhamento dos detritos espaciais (forma "bananóide") passou por várias etapas. A seguir estão descritos os códigos-fontes de cada fase do trabalho:

a) ELIPSE SIMPLES

```
function elipse

%Propriedades: Elipse simples centrada em (10,2).

  %coordenadas do centro
 x0=10;
 y0=2;

 %definição dos semi-eixos
 ex=10;
 ey=4;

 %desenho
    x=(-ex+x0):0.05:(ex+x0);
    ysup=y0+ey.*sqrt( 1-((x-x0)/ex).^2 );
    yinf=y0-ey.*sqrt( 1-((x-x0)/ex).^2 );

    plot(x,ysup);
    hold on
    plot(x,yinf);
end
```

b) ELIPSE CUJO CENTRO GIRA SEGUNDA UMA CIRCUNFERÊNCIA

```
function elipse

%Propriedades: Gira em torno de (0,0).

for  t=1:100:1000
   % t = variável de tempo
   w=2*3.14159*0.001; %velocidade angular
   raio=10;
   teta=w.*t;

  % desenho de uma elipse

 %coordenadas do centro da elipse
 xe=raio.*cos(teta);
 ye=raio.*sin(teta);

 %definicao dos semi-eixos
 ex=2; %eixo x da elipse
 ey=1; %eixo y da elipse

 %desenho
   x=(-ex+xe):0.05:(ex+xe);
   ysup=ye+ey.*sqrt( 1-((x-xe)/ex).^2 );
   yinf=ye-ey.*sqrt( 1-((x-xe)/ex).^2 );

   plot(xe,ye);
   hold on;
   plot(x,ysup);
   hold on;
   plot(x,yinf);
end

end
```

c) ELIPSE CUJO CENTRO GIRA SEGUNDO UMA CIRCUNFERÊNCIA E CUJOS EIXOS TAMBÉM CRESCEM.

```
function elipse

%Propriedades: Gira em torno de (0,0) com crescimento dos eixos.

for  t=1:50:1000
   % t = variável de tempo
   w=2*3.14159*0.001; %velocidade angular
   raio=50;
   teta=w.*t;

 % desenho de uma elipse
   %coordenadas do centro da elipse
   xe=raio.*cos(teta);
   ye=raio.*sin(teta);

   %definicao dos semi-eixos
   ex=2+0.02*t; %taxa de crescimento do eixo x da elipse
   ey=1+0.01*t; %taxa de crescimento do eixo y da elipse

   %equacoes da elipse
   x=(-ex+xe):0.05:(ex+xe);
   ysup=ye+ey.*sqrt( 1-((x-xe)/ex).^2 );
   yinf=ye-ey.*sqrt( 1-((x-xe)/ex).^2 );

   %plotagem da elipse
   plot(xe,ye);
   hold on;
   plot(x,ysup);
   hold on;
   plot(x,yinf);

end

end
```

d) ELIPSE SIMPLES COM OS EIXOS ROTACIONADOS

```
function elipse

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % DESENHO DA ELIPSE

 % rotacao da elipse
     giro=-pi/6; %angulo de rotacao da elipse


 %definicao dos semi-eixos
 ex=2; %eixo x da elipse
 ey=1; %eixo y da elipse

 %desenho
    x=(-ex):0.05:(ex);
    ysup = ey.*sqrt( 1-((x)/ex).^2 );
    yinf = -ey.*sqrt( 1-((x)/ex).^2 );

 %rotacao da elipse
    x1= x.*cos(giro) + ysup.*sin(giro);
    x2= x.*cos(-giro) + ysup.*sin(-giro);
    ysup1= -x.*sin(giro) + ysup.*cos(giro);
    yinf1= -x.*sin(giro) - ysup.*cos(giro);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%elipse rotacionada
    plot(x1,ysup1);
    hold on;
    plot(x2,yinf1,'r');

end
```

e) ELIPSE COM OS EIXOS ROTACIONADOS E O CENTRO GIRANTE SEGUNDO UMA CIRCUNFERÊNCIA

```
function elipse

n=8; %numero de pontos
v=1; % numero de voltas

 for  t=1:100:(100*n*v)    % t = variavel de tempo
   w=2*pi/(100*n); %velocidade angular
                                          raio=10;
   teta=w.*t;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % DESENHO DA ELIPSE

 %coordenadas do centro da elipse
 xe=raio.*cos(teta);
 ye=raio.*sin(teta);

                                        % angulo de rotacao da elipse
                                             giro=-1.2.*teta;

 %definicao dos semi-eixos
 ex=2; %eixo x da elipse
 ey=1; %eixo y da elipse

 %desenho
   x=(-ex):0.05:(ex);
   ysup = ey.*sqrt( 1-((x)/ex).^2 );
   yinf = -ey.*sqrt( 1-((x)/ex).^2 );

   x1= xe+x.*cos(giro) + ysup.*sin(giro);
   x2= xe+x.*cos(-giro) + ysup.*sin(-giro);
   ysup1= ye-x.*sin(giro) + ysup.*cos(giro);
   yinf1= ye-x.*sin(giro) - ysup.*cos(giro);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%elipse rotacionada
   %plot(xe,ye,'g*'); %plota o centro de cada elipse
   %hold on;
   plot(x1,ysup1);
   hold on;
   plot(x2,yinf1,'r');

end
```

f) ELIPSE COM OS EIXOS ROTACIONADOS E O CENTRO GIRANTE SEGUNDO UMA CIRCUNFERÊNCIA (2 VOLTAS)

```
function elipse

n=8; %numero de pontos
v=2; % numero de voltas

 for  t=1:100:(100*n*v) % t = variavel de tempo
    w=2*pi/(100*n); %velocidade angular
                                                  raio=10;
    teta=w.*t;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % DESENHO DA ELIPSE

 %coordenadas do centro da elipse
 xe=raio.*cos(teta);
 ye=raio.*sin(teta);
                                               % angulo de rotacao da elipse
                                                      giro=-1.2.*teta;
 %definicao dos semi-eixos
 ex=2; %eixo x da elipse
 ey=1; %eixo y da elipse

 %desenho
    x=(-ex):0.05:(ex);
    ysup = ey.*sqrt( 1-((x)/ex).^2 );
    yinf = -ey.*sqrt( 1-((x)/ex).^2 );

    x1= xe+x.*cos(giro) + ysup.*sin(giro);
    x2= xe+x.*cos(-giro) + ysup.*sin(-giro);
    ysup1= ye-x.*sin(giro) + ysup.*cos(giro);
    yinf1= ye-x.*sin(giro) - ysup.*cos(giro);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%elipse rotacionada
    %plot(xe,ye,'g*'); %plota o centro de cada elipse
    %hold on;
    plot(x1,ysup1);
    hold on;
    plot(x2,yinf1,'r');

end
```