MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ANÁLISE E SIMULAÇÃO DE DETRITOS ESPACIAIS

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

Bolsista – Andreza da Costa Batista (ETEP Bolsista PIBIC/CNPq)

E-mail: andreza999@itelefonica.com.

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

E-mail: marcelo@dem.inpe.br

Julho de 2006

SUMÁRIO **página**

LISTA DE FIGURAS **página**

CAPÍTULO 1 – INTRODUÇÃO E MOTIVAÇÃO

1.1– INTRODUÇÃO

Atualmente estima-se que existam cerca de 200 mil fragmentos artificiais, dos mais variados tamanhos, girando em torno da Terra. Esses fragmentos integraram foguetes, satélites ou plataformas espaciais, que permanecem em órbita, como lixo espacial, depois de se tornarem inativos, até perderem velocidade e serem capturados pela gravidade da Terra. Eles podem ser desde uma chave de fenda perdida durante consertos espaciais até a própria estrutura de um satélite.

Em média há uma queda de três meteoritos por mês medindo de 1 a 10 metros de diâmetro, e pesando até 1000 toneladas. A maior parte dos detritos espaciais, seja artificial ou natural, geralmente cai em oceanos, por estes ocuparem a maior parte da superfície da Terra.

A probabilidade de qualquer um desses detritos vir a colidir com outros satélites em atividade ou atingir a superfície terrestre e causar um acidente é remota. Mas, mesmo sendo pequena esta probabilidade, torna-se importante o estudo da propagação desses detritos em órbitas úteis devido ao crescente número de satélites e de detritos nestas órbitas úteis.

1.2 – CRONOLOGIA DE CASOS RECENTES

A seguir estão alguns casos de detritos artificiais que atingiram a superfície terrestre:

Em 1966 caíram dois detritos grandes no Brasil, sendo eles: 1)Um tanque esférico de combustível (Hélio) de um foguete Saturno (Figura 1.1), com 1 metro de diâmetro, que caiu na costa norte e foi recuperada por pescadores. 2) Placas de metal entre 10 a 12 cm de comprimento, atingiram o Rio Negro, no Amazonas.

Em Julho de 1969 um fragmento de aproximadamente 30 cm atingiu um navio alemão no Oceano Atlântico e outros fragmentos caíram na água perto do navio. Estes foram identificados como sendo restos do primeiro estágio do foguete lançador Saturno, usado para lançar a espaçonave Apollo 11 em Julho deste mesmo ano.

Figura 1.1- Foguete Lançador Saturno V

Em 1979 o laboratório espacial SkyLab (Figura 1.2) deveria cair no Oceano Pacífico mas partes dele atingiram a costa da Austrália.

Figura 1.2- Laboratório Espacial SkyLab

Em 1991 a estação espacial Salyut 7 (Figura 1.3), antecessora da estação espacial MIR, caiu nas montanhas do Andes.

Figura 1.3- Estação Espacial Salyut 7

Em julho de 1996 o satélite operacional francês CERISE foi danificado quando um fragmento do corpo de um foguete Ariane (Figura 1.4) colidiu com a haste de controle de atitude por gradiente de gravidade do satélite.

Figura 1.4-Foguete Ariane

Em 1997 pedaços do Foguete Lançador Delta II (Figura 1.5) se espalharam entre o Texas e Oklahoma, nos Estados Unidos.

Figura 1.5- Foguete Lançador Delta II

1.3–MOTIVAÇÃO

O que motivou este trabalho foi a necessidade do estudo da análise e simulação da propagação de detritos espaciais pela fragmentação de um satélite, visando modelar e simular a propagação numérica e semi-analítica de uma distribuição de detritos espaciais que se movimentem ao redor da Terra. Poderemos assim 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–OBJETIVOS E HISTÓRICO DO TRABALHO

2.1–OBJETIVOS

Este trabalho tem como objetivos: 1) analisar e simular a geração e a propagação de detritos espaciais pela fragmentação de um satélite artificial em órbita e estudar sua evolução no tempo; 2) construir o 2º modelo analítico da propagação dos detritos, obtido pela expansão e curvatura dos eixos de uma elipse para que esta se aproxime da forma de um "bananóide", e rotação desta em torno de um centro; 3) comparar o 2º modelo analítico com o modelo numérico da propagação dos detritos, obtido pelas 3 Leis de Kepler e pela Equação de Kepler para cada detrito após a fragmentação, para posteriormente ajustar os parâmetros do 2º modelo analítico ao modelo numérico (por iterações, e depois pelo método dos mínimos quadrados, etc.) para que este simule da melhor maneira possível a propagação de detritos espaciais.

2.2–HISTÓRICO

No período entre 01 de agosto de 2002 a 31 janeiro de 2003, o 1° bolsista João Paulo Marques Reginato 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. 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. 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.

A partir de 01 de fevereiro de 2003 o 2° bolsista, Sandro Felgueiras Castro, assumiu esse projeto de pesquisa. Ele 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. Assim, em maio foi proposto e testado um 1º modelo analítico do movimento dos detritos. Para tanto, foi feito um programa POSIÇÕESCARTESIANAS 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 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. 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. 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.

Em 01 de março de 2005, a 4ª bolsista, Vanessa de Lima Takaoka, assumiu esse projeto de pesquisa. 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 compreensão, 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.

Em 17 de Fevereiro de 2006, a 5ª bolsista, Andreza da Costa Batista assumiu esse projeto de pesquisa e procedeu conforme a metodologia descrita a seguir.

CAPÍTULO 3 –METODOLOGIA

3.1 – INTRODUÇÃO

Em 17 de fevereiro de 2005, foi iniciado este projeto de pesquisa seguindo a orientação do Dr. Marcelo Lopes de Oliveira e Souza. Foi realizado um estudo em Mecânica Orbital, através da apostila de Kuga e Rao (1995) e a seguir, iniciou-se o estudo de tudo o que havia sido feito pelos bolsistas anteriores visando compreender, atualizar e continuar o projeto anterior.

3.2 – FORMULAÇÃO MATEMÁTICA: MÉTODO NUMÉRICO

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 (vide Apêndice 1) simula o movimento da fronteira inicial (**=envelope)** de um conjunto 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 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 conjunto de partículas, partindo do mesmo corpo original, com as mesmas condições iniciais, e com o mesmo gradiente de explosão.

Com base nos programas desenvolvidos pelos bolsistas anteriores, 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-os nos gráficos do mesmo programa.

As Figuras 3.1-3.2 abaixo demonstram as aplicações:

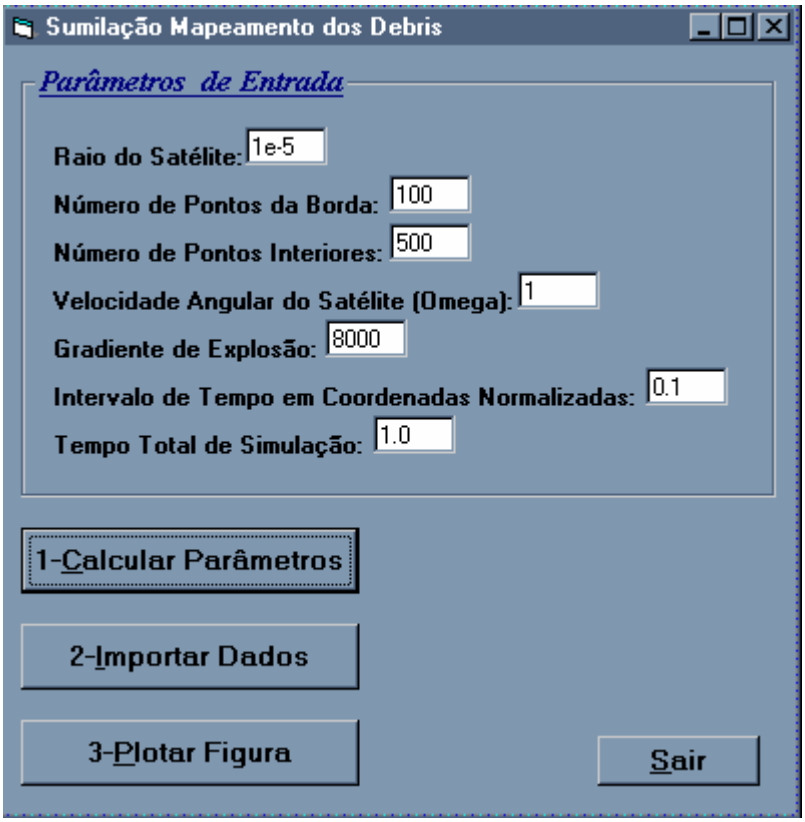

Figura 3.1 – Tela Principal do Programa kk2mat.c com os parâmetros de entrada

A Figura 3.1 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 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 para ser possível a sua visualização em forma gráfica. Finalmente as telas são impressas e demonstradas com os instantes na Figura 3.2 a seguir:

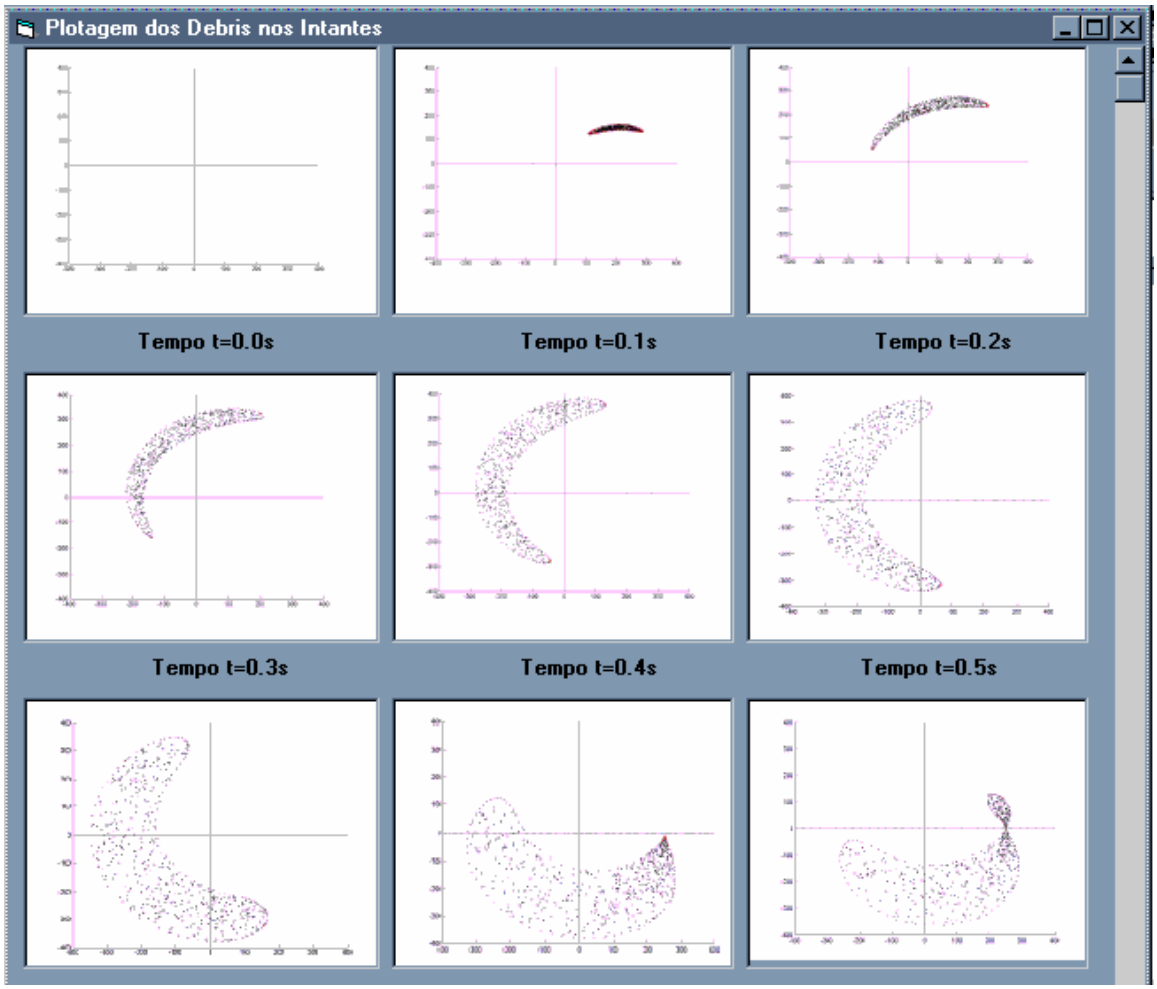

Figura 3.2 – Resultado dos pontos gerados no arquivo texto (interface gráfica de saída)

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 3.3) desenvolvido pelo segundo bolsista Sandro Felgueiras Castro.

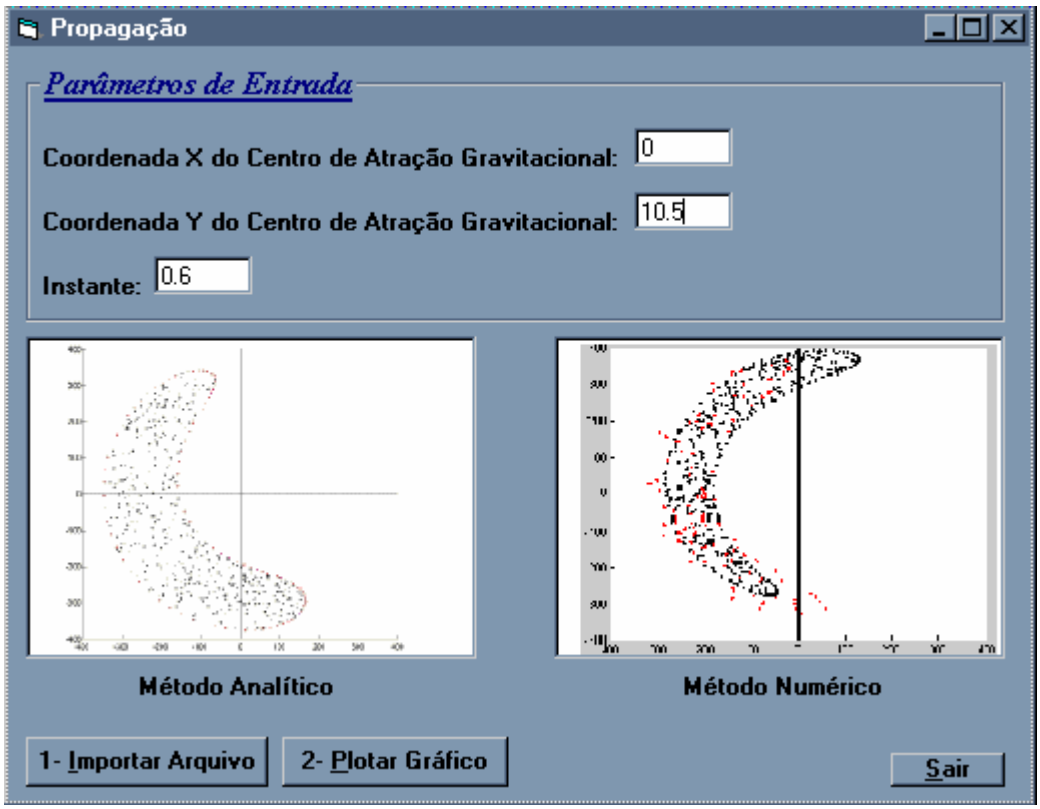

Figura 3.3 – Interface gráfica do programa "propagacao.cpp"

Na Figura 3.3, 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 Figura 3.3 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 $T = 2\pi$.

3.3 – FORMULAÇÃO MATEMÁTICA: MÉTODO ANALÍTICO (GEOMÉTRICO)

Os detritos espaciais artificiais giram em torno do centro de massa (provavelmente o CM da Terra) descrevendo uma órbita elíptica e respeitando as 3 Leis de Kepler. Para simular a propagação desses detritos e sua respectivas órbitas foram utilizadas as equações descritas a seguir:

Construção da Elipse (Figura 3.4):

Nas construções dos programas,vide Programa 1 no Apêndice 2, utilizamos a equação da elipse em coordenadas cartesianas:

$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1$$

logo,

$$y = y_0 \pm b\sqrt{1-\frac{(x-x_0)^2}{a^2}}$$

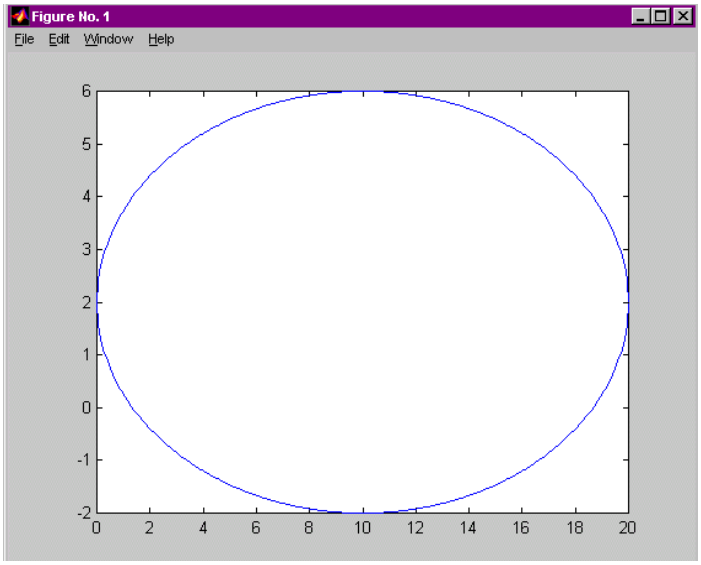

Figura 3.4: Elipse Simples centrada em (10,2)

Giro da Elipse(sem crescimento dos eixos; Figura 3.5):

Primeiramente supomos que os detritos giram em torno do centro de massa com velocidade angular constante, segundo uma órbita circular e simulamos essa propagação, vide Programa 2 no Apêndice 2, na qual o centro de cada elipse, que representa o detrito, é dada por:

$$x_0 = r\cos\theta$$

$$y_0 = r\,\mathrm{sen}\,\theta$$

onde r é o raio da órbita circular e θ o ângulo entre o raio da órbita e eixo da abscissa.

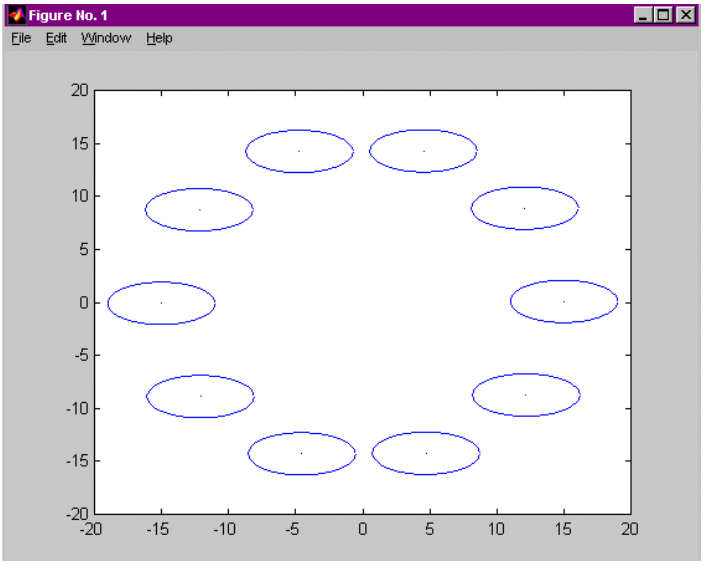

Figura 3.5: Giro da Elipse (sem crescimento dos eixos)

Rotação dos eixos da Elipse (Figura 3.6):

Para rotacionar os eixos da elipse, vide Programa 3 no Apêndice 2, foi feita uma transformação de coordenadas através do conceito de matrizes de rotação:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\varphi & sin\varphi \\ -\operatorname{sen}\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

onde x e y são as coordenadas no plano cartesiano, x' e y' são as coordenadas no plano rotacionado e φ é o ângulo de rotação.

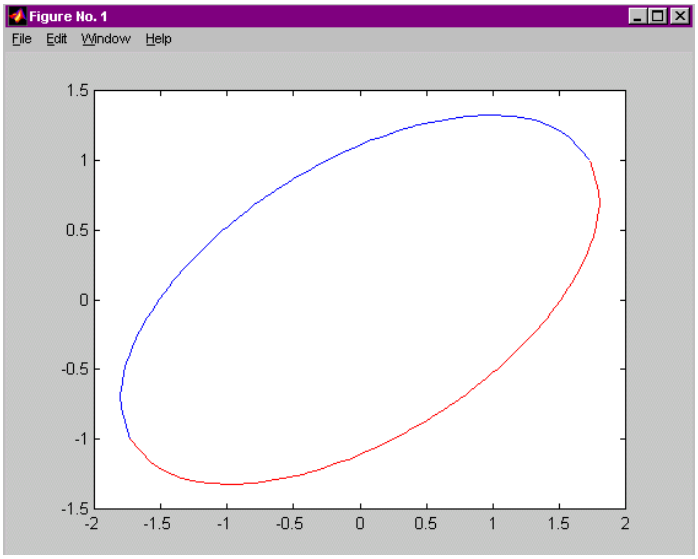

Figura 3.6: Elipse Rotacionada centrada em (0,0)

Giro da Elipse (com crescimento dos eixos; Figura 3.7):

Após rotacionar os eixos da elipse foi feita uma variação nos seus eixos para melhor simular a propagação dos detritos, ou seja, a medida que o tempo passa as elipses vão crescendo de acordo com uma taxa pré-estabelecida e girando em torno do centro de massa, mantendo a órbita circular.

Para essa simulação foram usadas as formulações citadas anteriormente e acrescentou-se a taxa de crescimento para os eixos, vide Programa 4 no Apêndice 2.

A variação do ângulo de rotação é dada por:

$$\varphi = -1.2\theta$$

onde φ é o ângulo de rotação e θ o ângulo entre o raio da órbita e eixo da abscissa.

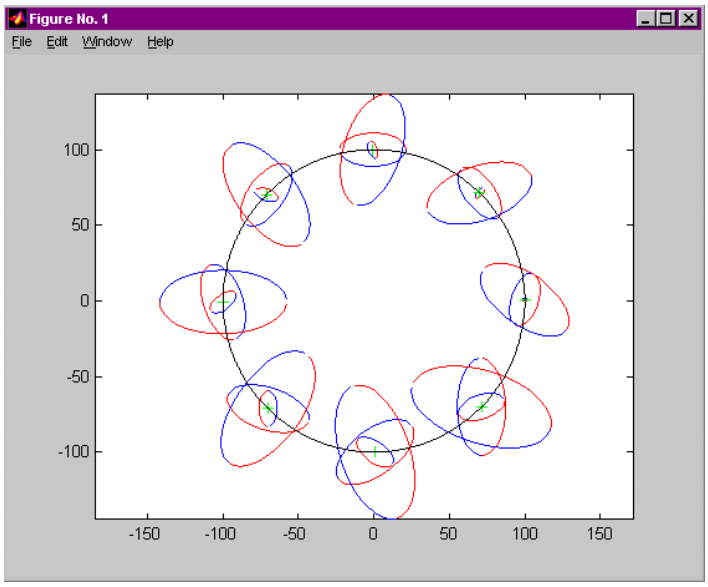

Figura 3.7: Giro da Elipse Rotacionada com crescimento dos eixos

Elipses Rotacionadas e Reposicionadas de acordo com o Centro de Massa (Figura 3.8):

Na construção desse outro modelo foi modificado o ângulo de rotação, para que o prolongamento do raio da órbita coincida com o semi-eixo maior e fique ortogonal ao semi-eixo menor, vide Programa 5 no Apêndice 2. Para isso ocorrer o ângulo de rotação passou a ser:

$$\varphi = -\theta$$

onde φ é o ângulo de rotação e θ o ângulo entre o raio e a órbita e o eixo da abscissa.

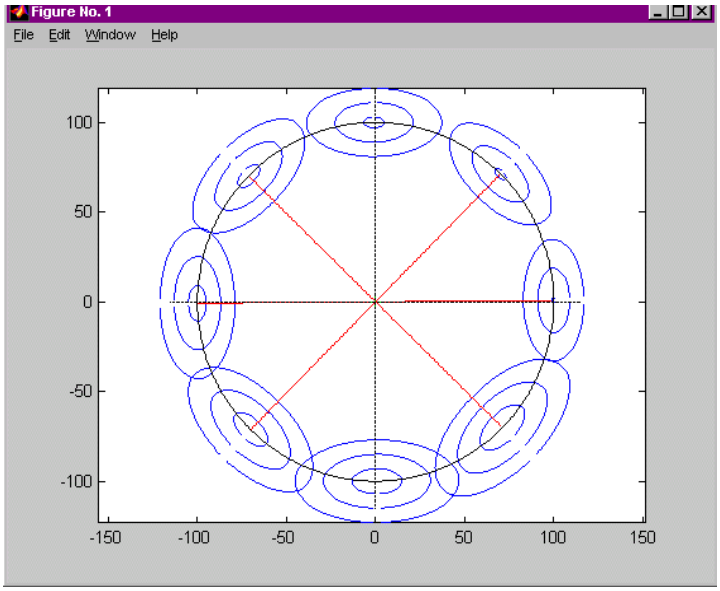

Figura 3.8: Elipses Rotacionadas e Reposicionadas de acordo com o Centro de Massa

Equações da Curvatura do Centro da Elipse (Figuras 3.9-3.10)

Está sendo feito um estudo para poder curvar os eixos da elipse. A seguir estão os resultados obtidos até então:

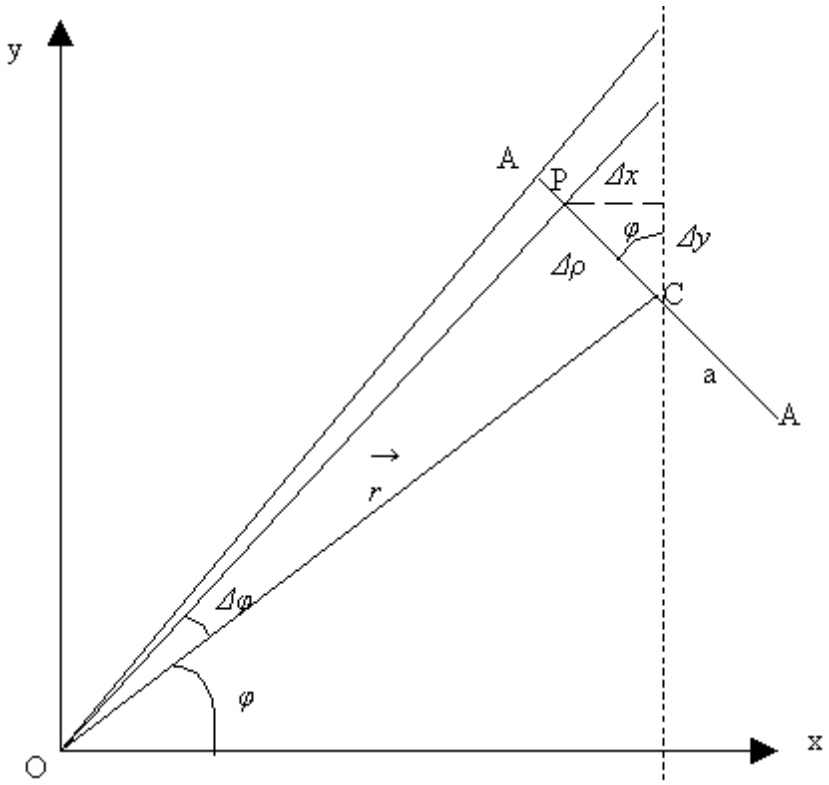

Figura 3.9: Relação entre as variáveis

Onde:

$$0 \le \Delta\varphi \le \arctan\frac{a}{\overline{OC}} \qquad 0 \le \rho \le \overline{OC}$$

$$-a \le \Delta\rho_{mín} \le \Delta\rho \le \Delta\rho_{máx} \le a \qquad 0 \le \varphi \le 2\pi\,\frac{rad}{s}$$

Em OC:

$$x = \rho\cos(\omega t + \varphi_0)$$
$$y = \rho\,\mathrm{sen}(\omega t + \varphi_0)$$

Em CA:

$$\Delta x = -\Delta\rho\,\mathrm{sen}(\omega t + \varphi_0)$$
$$\Delta y = \Delta\rho\cos(\omega t + \varphi_0)$$

Em OP:

$$x_p = x + \Delta x = \rho\cos(\omega t + \varphi_0) - \Delta\rho\,\mathrm{sen}(\omega t + \varphi_0)$$
$$y_p = y + \Delta y = \rho\,\mathrm{sen}(\omega t + \varphi_0) + \Delta\rho\cos(\omega t + \varphi_0)$$

Em OA:

$$x_a = x + \Delta x = \rho\cos(\omega t + \varphi_0) - a\,\mathrm{sen}(\omega t + \varphi_0)$$
$$y_a = y + \Delta y = \rho\,\mathrm{sen}(\omega t + \varphi_0) + a\,\mathrm{sen}(\omega t + \varphi_0)$$

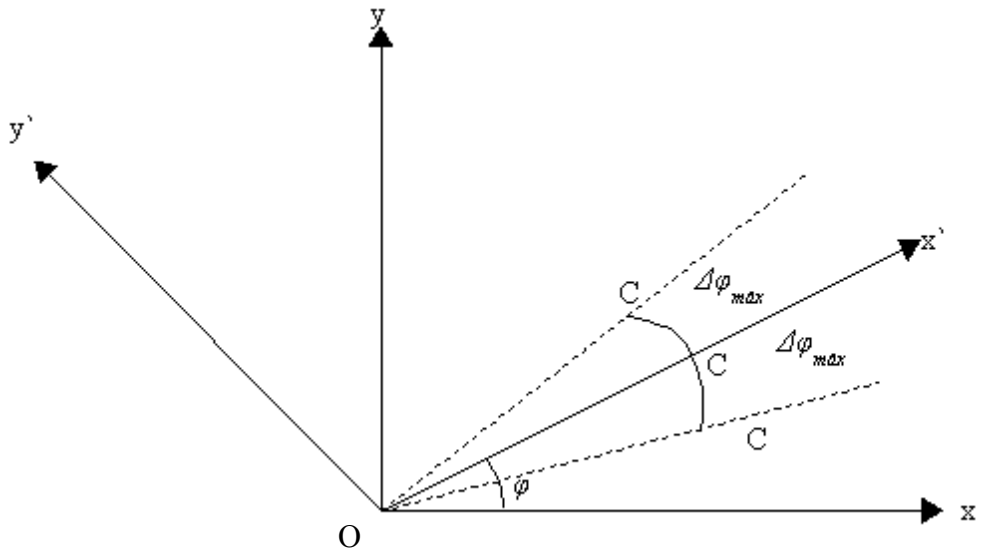

Figura 3.10: Curvamento do eixo da elipse

E de acordo com a Matriz de Rotação temos:

$$x = x'\cos\varphi - y'\,\mathrm{sen}\,\varphi$$
$$y = x'\,\mathrm{sen}\,\varphi + y'\cos\varphi$$

De acordo com a Figura 3.10 chegamos que:

$x' = \overline{OC} \cos \Delta\varphi$

$y' = \overline{OC} \operatorname{sen} \Delta\varphi$

CAPÍTULO4-CONCLUSÕES, COMENTÁRIOS E SUGESTÕES PARA O PROSSEGUIMENTO DO TRABALHO

O trabalho foi idealizado a partir das observações dos resultados dos Projetos de Pesquisa precedentes, nos quais notou-se que a propagação de Detritos Espaciais ocorria segundo a forma de uma elipse progressivamente deformada ("bananóide"), cujos eixos cresciam segundo alguma taxa, ao mesmo tempo em que a elipse girava em torno do seu Centro de Massa-CM, e este girava em torno de um ponto (provavelmente o Centro de Atração da Terra) segundo a órbita inicial.

Os programas foram compreendidos, executados e testados com êxito e foram dados vários passos para a construção do 2^o modelo analítico da propagação dos detritos e comparação do mesmo com o modelo numérico, obtido pelas 3 Leis de Kepler e pela Equação de Kepler para cada detrito após a fragmentação. Para futuramente, ajustarmos os parâmetros do 2^o modelo analítico ao modelo numérico (por iterações, e depois pelo método dos mínimos quadrados, etc.) para que este simule da melhor maneira possível a propagação de detritos espaciais.

Poderemos assim 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.

São José dos Campos, 07 de julho de 2006.

_____

Andreza da Costa Batista /Bolsista

Certifico que a aluna Andreza da Costa Batista e os seus trabalhos, realizados no período de 17/02/2006 a 31/07/2006, descritos neste Relatório foram plenamente satisfatórios.

_____

Marcelo Lopes de Oliveira e Souza/Orientador

APÊNDICE 1

Programa KK2.MAT.C feito por João Paulo Marques Reginato para PCS com Windows 2000 e Matlab para propagar e plotar os detritos espaciais

INTRODUÇÃO

Esse apêndice 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".

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 3). 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, 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 nessas 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 4.

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.

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.

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.

Programa KK.C original feito por Dalton Nunes em C para estações de trabalho com UNIX e POSTSCRIPT para propagar e plotar 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);
}
```

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 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 2

Programas Elipses desenvolvido por Anderson Patrick Alves Pereira em Matlab:

Programa 1:

```
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
```

Programa 2:

```
%function elipse1
        %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=5;
        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=4; %eixo x da elipse
        ey=2; %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
```

Programa 3:

```
% 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
```

Programas Simulações de Órbitas desenvolvido por Flávio Francesco Soares Schmidt em Matlab:

Programa 4:

```
function elipse_eixo4(n,v,raio)
    %n=8; n. de elipses em uma volta completa
    %v=2; n. de voltas
    % raio -> raio da orbita circular
    for t=1:100:(100*n*v)
    w=2*pi/(100*n);
    %raio=10;
    teta=w.*t;
    xe=raio.*cos(teta);
    ye=raio.*sin(teta);
    giro=-1.2.*teta;
    ex=2+0.02*t;
    ey=1+0.01*t;
    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);
    plot(xe,ye,'g+');
    hold on;
    plot(x1,ysup1);
    hold on;
    plot(x2,yinf1,'r');
    hold on;
    end
    %plotagem da orbita circular para referencia
    ey=raio;
    ex=raio;
    x=(-ex):0.5:(ex);
```

```
        ysup=ey.*sqrt(1-((x)/ex).^2);
        yinf=-ey.*sqrt(1-((x)/ex).^2);
        plot(x,ysup,'-k');
        hold on;
        plot(x,yinf,'-k');
        hold on;
        axis equal;
function end
```

Programa 5:

```
function elipse_eixo6(n,v,raio)
        %valores recebidos
        %n=8; n. de elipses em uma volta completa
        %v=2; n. de voltas
        % raio -> raio da orbita circular
        %inicia o laço de repetiçao para fazer o giro completo na orbita
        for t=1:100:(100*n*v)
        w=2*pi/(100*n);
        %raio=10;
        teta=w.*t;
        %define o centro de cada elipse
        xe=raio.*cos(teta);
        ye=raio.*sin(teta);
        %giro equivalente ao angulo percorrido da orbita
        giro=-teta;
        %definiçao do crescimento da elipse
        ey=2+0.02*t;
        ex=1+0.01*t;
        %elipse base
        x=(-ex):0.05:(ex);
        ysup=ey.*sqrt(1-((x)/ex).^2);
        yinf=-ey.*sqrt(1-((x)/ex).^2);
        %elipse rotacionada e reposicionada de acordo com o centro (ex,ey)
        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);
        %plotagem dafigura de cada elipse modificada
        plot(x1,ysup1);
        hold on;
        plot(x2,yinf1);
        hold on;
        %plotagem da orbita circular para referencia
        ey=raio;
        ex=raio;
        x=(-ex):0.5:(ex);
        ysup=ey.*sqrt(1-((x)/ex).^2);
        yinf=-ey.*sqrt(1-((x)/ex).^2);
        plot(x,ysup,'-k');
        hold on;
        plot(x,yinf,'-k');
        hold on;
        %plotagem do centro da orbita (Centro de atraçao gravitacional)
        plot(0,0,'+g');
        hold on;
        %plotagem das linhas de raio do centro da orbita ate o centro de cada elipse
        plot(0:1:xe,0:ye/xe:ye,'r'); %lado direito, onde xe>0
        hold on;
        plot(xe:1:0,ye:ye/xe:0,'r'); %lado esquerdo, onde xe<0
        hold on;
        %plotagem dos eixos x e y para orientaçao, respectivamente
        plot(-(raio+15):1:(raio+15),0,'k');
        plot(0,-(raio+15):1:(raio+15),'k');
        %igualando a escala dos eixos x e y
        axis equal
        end
function end
```