Instituto Tecnológico de Aeronáutica

Divisão de Engenharia Aeronáutica e Aeroespacial

Projeto de transferência Terra-Marte baseado em patched conics otimizado com algoritmo evolutivo

Autores:
Francisco Matheus[1]
Guilherme Marinot[1]
Gustavo Medeiros[1]
Rafael Mendes[1]
Luiz Sivieri[2]

Professora:
Dra. Maísa Terra

Disciplina:
MVO-41

Turma:
Aesp20

[1] Graduando em Eng. Aeroespacial no Instituto Tecnológico de Aeronáutica
[2] Pós-graduando em Física no Instituto Tecnológico de Aeronáutica

24 de junho de 2019

São José dos Campos, Brasil

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A astrodinâmica possui um problema fundamental, formulado pelo o matemático suíço Johann Heinrich Lambert (1728 - 1779), conhecido como problema de Lambert. Dentro de projetos de missões espaciais, é essencial saber o tempo de voo necessário para viajar entre determinados pontos em uma trajetória particular. De acordo com o teorema de Lambert, o tempo de transferência entre dois pontos, depende apenas da soma dos vetores posição, o semieixo maior e do comprimento que une os dois pontos. Em outras palavras, é um problema de valor de limite entre dois vetores de posição e um tempo de voo. Por meio da solução de tal problema, é possível calcular as variações de velocidades necessárias durante a transferência. No caso de uma transferência Terra-Marte, inicialmente se faz necessário identificar as posições da Terra e de Marte para determinadas datas de partida e chegada. Além disso, a solução para o problema de Lambert também é utilizada para determinar a orientação e a forma da trajetória hiperbólica de chegada, que neste caso se trata de Marte.

É importante ressaltar que o problema de Lambert não busca tempos mínimos de trajetórias, já que em suas soluções são apresentadas possíveis trajetórias cônicas (*conics-like trajectories*) entre dois pontos de interesse sobre um determinado tempo. Desse modo, as soluções de Lambert ajudam a criar um gráfico, do inglês conhecido como *porkchop plot*, que indica todas as trajetórias para um determinado intervalo de tempo, i.e., datas de partida e chegada fixas que auxiliam na análise da escolha do que pode ser a melhor trajetória tendo em base a finalidade da missão, como a redução de custo com combustível por exemplo.

2 Metodologia

2.1 Problema de Lambert

2.1.1 Teorema de Lambert

Suponha que a nave está no ponto P1 em sua órbita e o alvo estará no ponto P2 em sua órbita, e a questão é a determinação do tempo de transferência. De forma a obter uma equação descritiva do tempo de transferência, utiliza-se a equação de Kepler. Como esta descreve um problema de valor inicial e o problema de Lambert, como já dito, trata de um problema de valor de contorno orbital, é possível derivar, da equação de Kepler, a equação de Lambert que seja independente da excentricidade.

$$E_2 - E_1 - e(sinE_2 - sinE_1) = \sqrt{\frac{u}{a^3}}(t_2 - t_1) \tag{1}$$

$$\frac{a^3}{2}[\alpha - sin\alpha - (\beta - sin\beta)] = \sqrt{\mu}(t_2 - t_1) \tag{2}$$

onde α e β são descritos de tal modo que

$$\alpha = 2sin^{-1}\left(\sqrt{\frac{s}{2.a}}\right) \tag{3}$$

$$\beta = 2sin^{-1}\left(\sqrt{\frac{s-c}{2.a}}\right) \tag{4}$$

e s e c dados por

$$s = 0,5(r_{earth} + r_{mars} + c) \tag{5}$$

$$c = \sqrt{r_{earth}^2 + r_{mars}^2 - r_{earth}r_{mars}cos\theta} \tag{6}$$

A demonstração pode ser encontrada em [8]. Por meio dessa equação, é possível obter as soluções para as transferências. Tais soluções informam as velocidades, nos pontos de partida e chegada, necessárias para que uma espaçonave possa realizar a transferência entre os pontos propostos. E, ainda, essas soluções rendem não somente trajetórias elípticas, como também trajetórias hiperbólicas, dependendo de qual é o intervalo de tempo (datas de partida e chegada) que está sendo analisado.

Para resolver o problema de Lambert, inicialmente deve-se definir as datas de partidas e chegadas pretendidas para a missão espacial. Feito isso, se torna possível calcular os vetores posição de cada planeta dado os tempos iniciais e finais do objeto

de estudo. Em suma, o problema é resolvido usando o método de cônicas coladas, do inglês *patched conics*, onde a transferência de uma espaçonave se dá por meio de cônicas que vão se interligando em determinados pontos durante o percurso até o destino final. Assim sendo, ao invés de termos um problema de N corpos, reduzimos a três ou mais problemas de 2 corpos, tornando mais simples projetar a trajetória da missão. Dessa forma, pode-se utilizar a mesma abordagem feita em [9], onde a transferência interplanetária pode ser dividida como:

- A saída interplanetária, considerando que a espaçonave está inicialmente em uma orbita estacionária, e deseja-se aplicar um incremento de velocidade para que a mesma possa entrar em uma trajetória hiperbólica de transferência.

- A entrada interplanetária, i.e., a espaçonave realiza uma aproximação do planeta de destino e, se faz necessário uma redução de velocidade para que a espaçonave possa entrar em uma trajetória que irá orbitar o planeta de destino.

Com isso, se torna perceptível o método de *patched conics*, onde a primeira cônica é a orbita estacionária em que a espaçonave se encontra. A segunda cônica é a trajetória hiperbólica de escape do planeta de origem, a terceira é a cônica de transferência considerando, agora, uma órbita em torno do Sol em direção ao planeta alvo. A quarta consagra-se como uma cônica na qual a espaçonave irá de aproximar e, a ultima, a que irá orbitar depois de se aproximar do planeta pela trajetória hiperbólica. Nesse contexto, um aspecto que deve ser levado em consideração é a esfera de influência, do inglês *sphere of influence* (SOI), que nada mais é do que o lugar geométricos dos pontos primariamente afetos pelo campo gravitacional do corpo massivo ao qual se orbita, como, por exemplo, um planeta exercendo influência sobre objetos relativamente próximos, tais como espaçonaves e satélites naturais. É necessário considerar a SOI nos seguintes casos:

- Quando a espaçonave se encontra próxima ao planeta inicial e ela deve possuir um incremento de velocidade para que consiga escapar do campo gravitacional e entrar numa trajetória de transferência;

- Quando a espaçonave realiza uma aproximação a um planeta para usar seu campo gravitacional para mudar de direção;

- Novamente quando existe a aproximação do planeta e da espaçonave, e a mesma deseja-se colocar em uma trajetória que orbitará o planeta.

A expressão que descreve o raio da esfera de influência, em torno do planeta a que se refere é [1]:

$$\overline{r_{SOI}} = 0.9431a\left(\frac{m}{M}\right)^{2/5} \tag{7}$$

que foi a equação usada para todos os cálculos pertinentes deste trabalho.

2.1.2 Abordagem

Foi-se utilizado uma rotina em MATLAB que resolve de maneira robusta o problema de Lambert [2]. Essa rotina tem como *inputs* o tempo de trajeto e as posições de chegada e de partida do corpo e retorna, por sua vez, as velocidades de partida e de chegada nos dados pontos da órbita kleperiana que satisfaz as condições de contorno impostas.

Sabe-se, no entanto, que o problema de Lambert não possui, dadas as condições de contorno especificadas acimas, necessariamente apenas uma solução, e então, como parâmetro adicional, pode ser requisitado ao programa que retorne as outras soluções que não o arco da cônica no sentido anti-horário (a qual é retornada por padrão). No entanto, para esse projeto, foi considerada apenas a resposta padrão do problema. Além disso, o programa também devolve as distâncias mínima e máxima entre a espaçonave e o corpo considerado como centro de massa do sistema, durante a sua trajetória. Essas informações, contudo, não foram utilizadas no escopo desse projeto. Para mais informações, ver [2]. O código utilizado, retirado desta referência, está exposto no Apêndice A.

A partir dessa implementação foi possível simular e obter as velocidades inicial e final de cada uma das três órbitas utilizadas para compor a missão interplanetária da Terra até Marte, valendo-se da técnica de *patched conics*. A primeira (Manobra 1) consiste em sair de uma órbita de espera em torno da Terra e chegar na borda da esfera de influência da Terra (SOI_T); a segunda (Manobra 2), em ir da borda da SOI_T até a borda da SOI_{Marte}; e a terceira (Manobra 3), em ir da borda da SOI_M até uma órbita de espera em Marte, como melhor descrito a seguir.

2.2 Patched conics

A fim de se utilizar a abordagem de *patched conics*, foram feitas algumas considerações que possibilitaram unir os resultados das três manobras discernidas na seção anterior.

Será assumido que todas as órbitas envolvidas no problema são coplanares: as órbitas de espera na Terra e em Marte e as órbitas da Terra, de Marte em torno do

Sol e, por consequência, as órbitas de transferência. Além disso, foram utilizadas apenas circunferências, e não potenciais esferas para trajetórias tridimensionais, de influências como possíveis destinos quando utilizando como ponto final a esfera de influência da Terra ou de Marte. Tais circunferências, também coplanares às órbitas, foram tomadas a fim de fazer com que o problema se resumisse a uma análise em apenas um plano.

Assim, foi-se possível definir o ponto de chegada e de partida em cada órbita, ou seja, em cada circunferência de influência, por meio de apenas um ângulo, medido sempre a partir da direção tangente à órbita da Terra em torno do Sol, considerando-a como uma circunferência, com sentido positivo o sentido da velocidade da Terra em torno do Sol. Em outras palavras, o módulo da distância dos pontos tomados como de interesse, a saber, A, B, C e D, a serem descritos a seguir, e que se farão como parâmetros livres da análise de otimização, estão fixos, permitindo-se apenas a variação das suas coordenadas angulares para caracterização dos mesmos. Tal representação dos pontos está ilustrada de forma simplificada na Figura 1 para o caso de *patched conics* simplificada. Nesta, as respectivas coordenadas angulares são $\theta_A = \pi/2$, $\theta_B = -\pi/2$, $\theta_C = \pi/2$ e $\theta_D = -\pi/2$, conforme análise desenvolvida, para o qual já temos metodologia e resultados analíticos (ver [3]) para usarmos como referência de otimização e base de análise.

Por conseguinte, foi-se possível, a partir de uma análise das diferenças vetoriais de velocidade, estimar a alteração de velocidade total imposta à espaçonave, a qual se relaciona diretamente com o custo total da missão. As diferenças vetoriais de velocidade que precisaram ser consideradas estão listadas a seguir.

1. Entre a velocidade de órbita de espera ao redor da Terra numa altitude de $0,1R_T$, onde R_T é o raio da Terra (a partir do ponto A), e a velocidade de partida da Manobra 1;

2. Entre a velocidade de chegada da Manobra 1 (chegando na SOI_T) e a velocidade de saída da Manobra 2 (saindo da SOI_T) (ponto B);

3. Entre a velocidade de chegada da Manobra 2 (chegando na SOI_M) e a velocidade de saída da Manobra 3 (saindo da SOI_M) (ponto C);

4. Entre a velocidade de chegada da Manobra 3 e a velocidade de órbita de espera ao redor de Marte, numa altitude de $0,3R_M$ (ponto D), onde R_M é o raio de Marte.

As diferenças de velocidade também precisaram contabilizar o fato de que as velocidades de chegada e partida devolvidas pelo programa nem sempre estavam no

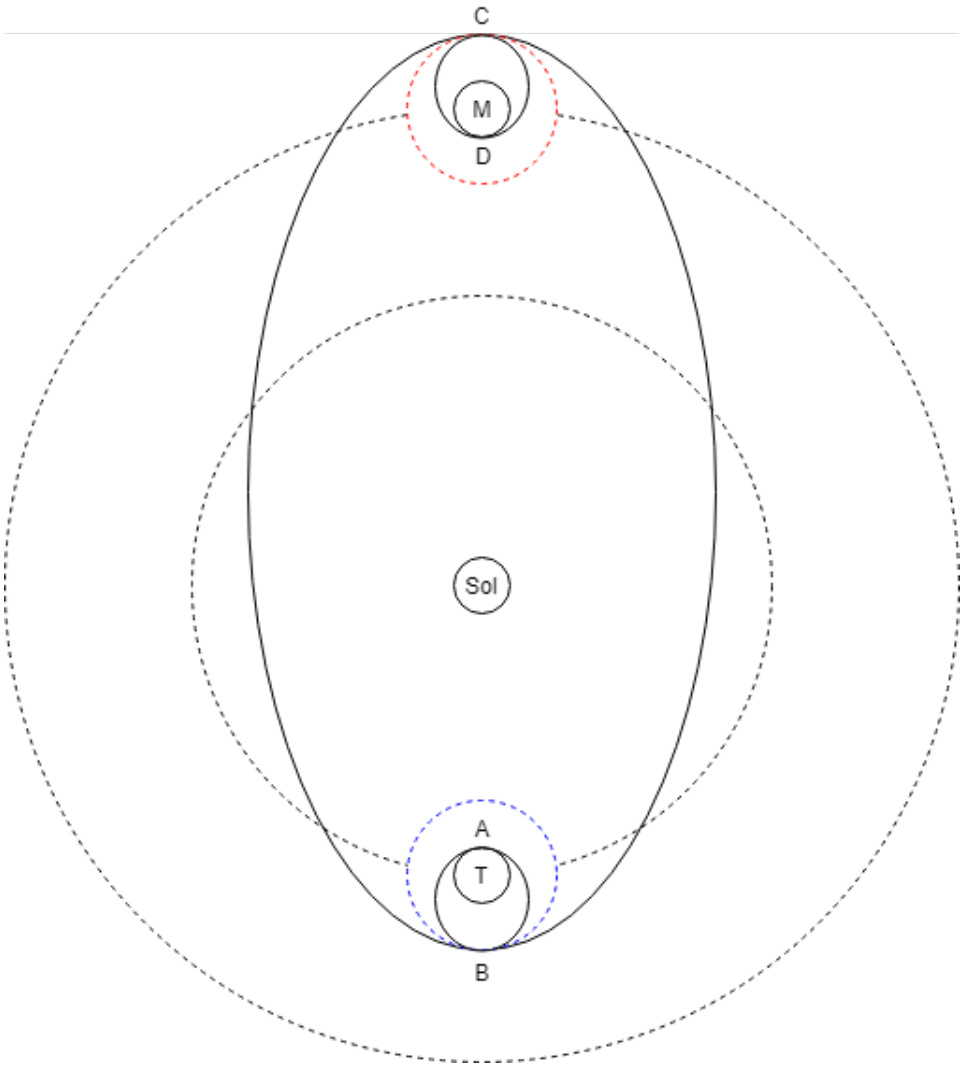


Figura 1: Órbita simplificada para ilustração dos parâmetros livres envolvidos na análise por *patched conics*.

mesmo referencial. Então, para a Manobra 1, as velocidades são apresentadas em relação à Terra, para a Manobra 2, em relação ao Sol, e para a Manobra 3, em relação à Marte. As devidas correções foram feitas numericamente.

O problema de custo total da missão, portanto, está limitado a 7 incógnitas principais, como descrito abaixo:

1. **Âng A** - Angulação entre 0 e 2π que representa a posição da espaçonave na órbita de espera em que ela irá acionar os motores para realizar a primeira manobra que comporá o movimento de *patched conics* - da órbita de espera até a borda esfera de influência da Terra (SOI_T);

2. **Âng B** - Angulação entre 0 e 2π que representa a posição de chegada da espaçonave na borda da SOI_T, assumindo movimento planar;

3. **Âng C** - Angulação entre 0 e 2π que representa a posição de chegada da espaçonave na borda da SOI_M (esfera de influência de Marte), assumindo movimento planar. Essa angulação representa a chegada da segunda manobra que compõe o *patched conics* - da SOI_T até a SOI_M;

4. **Âng D** - Angulação entre 0 e 2π que representa a posição de chegada da espaçonave na órbita de espera de Marte - o ponto final da última manobra que compõe o *patched conics* - da SOI_M até a órbita de espera em Marte;

5. **Nro Dias 1** - Número de dias que serão necessários para realizar a primeira manobra da *patched conics* - a ida da órbita de espera da Terra até a SOI_T;

6. **Nro Dias 2** - Número de dias que serão necessários para realizar a segunda manobra da *patched conics* - a ida da SOI_T até a SOI_M;

7. **Nro Dias 3** - Número de dias que serão necessários para realizar a terceira manobra da *patched conics* - a ida da SOI_M até a órbita de espera em Marte.

O código que modela este problema está exibido no Apêndice A.

2.3 Otimização

A abordagem utilizada pelo grupo, portanto, foi a análise do menor custo que a missão interplanetária pode possuir. O menor custo está relacionado linearmente com a variação de velocidade imposta à espaçonave.

Para cumprir essa abordagem, portanto, foi implementado um algoritmo de otimização computacional. Esse algoritmo foi implementado para otimizar o resultado da rotina do MATLAB utilizada anteriormente para resolução do problema de Lambert (problema de valor de contorno), que nos dava a variação de velocidade necessária para mudar de órbita a partir da resolução e consequente aplicação do algoritmo para vários pontos que descreveriam as órbitas desejadas. Em outras palavras, os algoritmos foram integrados de tal forma que a resolução de consecutivos problemas de Lambert que compartilhavam pontos referente à colagem das cônicas foi utilizada para determinar o custo da transferência a ser minimizado. Dentre os algoritmos de otimização existentes, foi decidido pela utilização do chamado *Particle Swarm Optimization (PSO).*

2.3.1 Particle Swarm Optimization (PSO)

O algoritmo de otimização *PSO* baseia-se em dinâmica populacional e utiliza-se de uma técnica de espalhamento para encontrar o mínimo global que uma função

pode assumir [5]. Para que o leitor entenda o trabalho realizado de forma completa, segue abaixo uma explicação sucinta do algoritmo.

O *PSO* trabalha com um número de "partículas" n definido pelo usuário. Cada partícula representa um único estado dentre todos os possíveis inputs do programa - no caso do projeto, isso representa um vetor de 7 incógnitas, onde cada uma está entre os valores de *upper bound (UB)* e de *lower bound (LB)*, também definidos pelo usuário. As sete incógnitas são os parâmetros descritos na seção anterior.

Tendo definido o número de partículas n, o UB e o LB, o *PSO* cria as partículas aleatoriamente. Assim, existirão n vetores de 7 incógnitas, e essas incógnitas estarão definidas randomicamente.

Então o programa aplica a função para cada uma dessas partículas, encontrando um valor de custo para cada uma delas. A partir daí, começa a etapa iterativa do programa. O usuário define o número de iterações i que serão realizadas.

O algoritmo tem como premissa convergir as partículas para mínimos locais, ainda diversificando o resultado o bastante para que cada partícula consiga explorar novas regiões e encontrar, ainda, mínimos elegíveis ao mínimo global da função de custo.

> Ele [o algoritmo] o faz a partir de uma função velocidade, a qual possui um valor para cada uma das partículas criadas. Implementada apenas internamente, ela define o quanto a partícula se afasta da sua posição anterior a partir de uma série de parâmetros que definem o quanto ela está próxima ou distante de um possível mínimo. A função velocidade possui valor randômico na primeira iteração e é dada pela equação 8.

$$v_{i+1} = \omega v_i + \phi_p r_p (b_i - x_i) + \phi_g r_g (b_g - x_i) \tag{8}$$

onde

- x_i representa o estado atual (vetor com sete elementos) da partícula.
- b_i representa o melhor estado histórico já encontrado *pela partícula*.
- b_g representa o melhor estado histórico já encontrado dentre *todas as partículas*.

Assim, a equação 8 pode ser analisada em cada um de seus termos para o entendimento de como os valores convergem. O primeiro termo da equação, ωv_i representa a inércia da velocidade - não permitir que a velocidade seja alterada de maneira muito acentuada de uma iteração para a outra. O segundo termo, $\phi_p r_p (b_i - x_i)$, representa que quanto mais próxima a partícula estiver do seu mínimo histórico, menor a sua velocidade, ou seja, ela tende a permanecer nesse mínimo local. O terceiro termo,

$\phi_g r_g(b_g - x_i)$, representa a convergência para o mínimo global, ou seja, quanto mais perto a partícula estiver do mínimo global (mínimo histórico encontrado dentre todas as partículas), menor a sua velocidade. Note que ambas as diminuições entre b e x são vetoriais, de forma que, quando a partícula se distancia de um mínimo (seja ele local ou global), ela tende a voltar para lá devido a esses termos.

A respeito dos outros parâmetros, segue descrição detalhada.

- ω - Peso de Inércia - Não permitir que a velocidade seja alterada de maneira muito acentuada de uma iteração para a outra;
- ϕ_p - Parâmetro Cognitivo - Valor entre 0 e 1 escolhido pelo usuário, representa o peso atribuído à variação da velocidade relativo à proximidade com um mínimo local;
- ϕ_g - Parâmetro Social - Valor entre 0 e 1 escolhido pelo usuário, representa o peso atribuído à variação da velocidade relativo à proximidade com um mínimo global;
- r_p - Parâmetro Randômico Cognitivo - Valor aleatório entre 0 e 1 atrelado ao parâmetro cognitivo.
- r_g - Parâmetro Randômico Social - Valor aleatório entre 0 e 1 atrelado ao parâmetro social.

Os parâmetros r_p e r_g adicionam um fator aleatório ao sistema, permitindo que sejam encontrados outros mínimos que não limitados àqueles que já foram descobertos.

Após definido o vetor velocidade, a partícula é atualizada (iterada) de acordo com a equação 9. O processo é repetido i vezes, de acordo com o que foi definido pelo usuário.

$$x_{i+1} = x_i + v_i \tag{9}$$

Para melhor visualização da convergência das partículas, a Figura 2 a representa quando utilizado o algoritmo de PSO.

2.3.2 Escolha dos Parâmetros

Implementou-se, portanto, o algoritmo de PSO para explorar os custos associados à resolução de uma transferência feita com cônica coladas através da função de

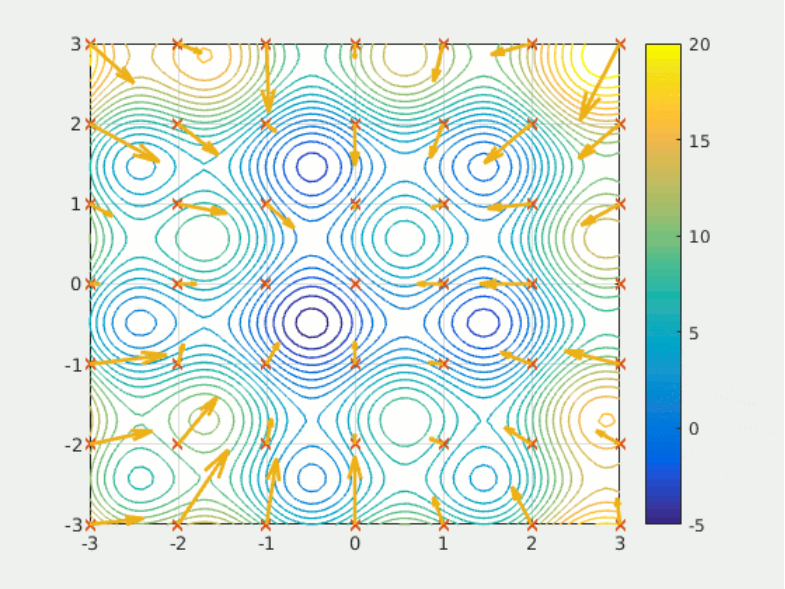


Figura 2: Convergência de partículas num exemplo ilustrativo de utilização do algoritmo PSO.

Lambert, como descrito na seção anterior. Então, foi preciso decidir os parâmetros de execução do código.

A primeira tarefa foi escolher os *UBs* e os *LBs* de cada incógnita utilizada. Para fazê-lo, foi-se utilizada uma estimativa de missão interplanetária entre a Terra e Marte [3], a partir da técnica de *patched conics*, valendo-se de hipóteses simplificadoras semelhantes às utilizadas nesse projeto, impondo transferências de Hohmann em todas as manobras que foram descritas na seção anterior. Essa estimativa retorna valores de tempo de voo de 0,55, 8,67 e 0,84 meses para as manobras 1, 2 e 3, respectivamente. Assim, foi decidido utilizar os seguintes *upper* e *lower bounds* como os demonstrados na Tabela 1.

Tabela 1: Upper e Lower Bounds de cada uma das incógnitas do PSO.

	Âng A (rad)	**Âng B (rad)**	**Âng C (rad)**	**Âng D (rad)**	**Nro Dias 1**	**Nro Dias 2**	**Nro Dias 3**
Lower Bound	0	0	0	0	0	0	0
Upper Bound	2π	2π	2π	2π	60	360	60

Dessa forma, foi definida a dimensão do problema.

De acordo com [4], o algoritmo PSO e seus parâmetros devem ser escolhidos de maneira a balancear adequadamente a exploração do espaço de estados e afluência a

um mínimo encontrado, a fim de evitar convergência prematura a um mínimo local e ainda assim assegurar uma boa taxa de convergência para o mínimo global.

Assim, o próximo passo foi definir os parâmetros ω, ϕ_p e ϕ_g, como descritos na seção anterior. Tomamos como base resultados empíricos que, conforme [7], resultam em boa convergência e exploração do espaço de soluções, resultando nos seguintes valores: $\omega = 0,9$, $\phi_p = 0,6$ e $\phi_g = 0,8$.

O tempo de execução da tarefa foi um limitante importante para definir os parâmetros de execução do software. Quanto maior a quantidade de partículas n que serão utilizadas, maior a chance de cobrir de maneira efetiva todo o espaço de estados que são abordados pelo problema, no entanto maior o poder computacional necessário. De maneira semelhante, o número de iterações i é relevante para que a partícula consiga explorar o espaço de estados suficientemente e que consiga convergir para o mínimo global.

Iterações iniciais do problema foram feitas com relativamente poucas partículas e poucas iterações, a saber $n = 50$ e $i = 100$. O tempo de execução foi pequeno (aproximadamente 10 segundos), no entanto, devido aos resultados imensamente díspares obtidos[1], foi-se decidido aumentar ambos os parâmetros para se obter resultados mais consistentes. Esses resultados foram descartados.

Optou-se, então, por fazer vários testes com $n = 500$ e $i = 200$. Os resultados foram mais consistentes, ou seja, retornavam valores próximos que sugeriam grandes, no sentido de importante, mínimos locais[2]. Cada execução demorava aproximadamente 5 minutos. Assumiu-se que o maior valor encontrado para o mínimo da função custo era apenas um mínimo local e que o algoritmo eventualmente convergia para ele precipitadamente, provavelmente devido a uma falta de espalhamento de número de partículas por todo o espaço de estados ou devido a uma falta de iterações para permitir uma melhor exploração dos parâmetros.

Por fim, para averiguar a consistência do resultado de menor valor encontrado para o custo total da viagem, foi executado o programa para $n = 5000$ e $i = 2000$. O processo demorou aproximadamente 8 horas para ser completado. O resultado, no entanto, corroborou o resultado de menor custo obtido pelas execuções com menor n e menor i. Assim, foram utilizados os resultados encontrados por essa execução.

Os códigos resultantes da implementação do algoritmo PSO em Matlab feita para este trabalho pode ser conferida no Apêndice A.

[1] Tais disparidades dão indícios de que o problema apresenta vários mínimos locais que eram melhores ou piores explorados a depender da execução

[2] Grandes mínimos locais são bons candidatos para mínimo global da função. De forma geral, quanto maior for a frequência de repetição de um valor para várias execuções, maior será a probabilidade dele ser o mínimo global

3 Resultados

A Tabela 2 demonstra os resultados que foram obtidos nas diversas tentativas de minimização do custo total envolvido na trajetória estudada através do algoritmo de otimização utilizado. Nesta, temos expressa vários valores para o custo total, que apresenta-se diferente mesmo para o mesmo número de iterações. Isto deve-se ao fato de que são mínimos locais e o algoritmo da forma como é concebido não garante um mínimo global, mas sim um local para o tempo (quantidade de iterações) que tem disponível para avaliar todo o espaço amostral.

Nesse contexto, a Tabela 2 mostra as características de posicionamento de cada um dos 4 pontos, em um caráter geral, para cara uma das execuções do algoritmo, bem como o tempo de voo para cada arco de cônica colada. A soma destes nos dá o tempo total de voo.

Tabela 2: Resultados obtidos de diversas iterações do programa de otimização.

Custo Total (km/s)	**Âng A (rad)**	**Âng B (rad)**	**Âng C (rad)**	**Âng D (rad)**	**Nro Dias 1**	**Nro Dias 2**	**Nro Dias 3**	**Nro Partículas**	**Nro Iterações**
7.3540	2.6447	0.0000	4.9218	1.7212	3.11	260.97	21.45	500	200
7.3318	2.6265	0.0000	4.6822	1.5406	3.11	261.03	26.03	500	200
5.6547	2.6258	0.0000	3.0408	0.8367	3.26	260.91	2.29	500	200
5.5044	2.6292	0.0000	3.0095	0.8299	3.09	261.96	2.28	500	200
7.2284	2.6227	0.0000	0.0000	3.1418	3.09	261.29	60.00	500	200
5.5044	2.6280	0.0000	3.0095	0.8369	3.09	261.96	2.28	500	200
5.5037	2.6147	6.2832	3.0095	0.8353	3.09	261.96	2.28	5000	2000

Dado os resultados expostos acima, temos que a trajetória mais otimizada encontrada foi a de custo $\Delta v = 5.5037\ km/s$, caracterizada pelos pontos A, B, C e D conforme descrito na seção Metodologias e exemplificados pela Figura 1, mas dotados das respectivas coordenadas angulares presentes na Tabela 2 com relação à direção da trajetória da Terra em relação ao Sol, tomados no plano da órbita. Desta forma, os vetores posição dos supracitados pontos, em relação aos respectivos planetas e com suas magnitudes dadas em km, são

$$\vec{r}_{A,Terra} = (-6.06, 3.52, 0)$$

$$\vec{r}_{B,Terra} = (871.95, 12.81, 0)$$

$$\vec{r}_{C,Marte} = (-538.59, 71.56, 0)$$

$$\vec{r}_{D,Marte} = (2.96, -3.27, 0)$$

onde as velocidades referentes a cada ponto foram as apresentadas na Tabela 4.

Tendo os vetores velocidade antes e após o impulso de mudança de órbita e os respectivos vetores velocidade, podemos definir as características da órbita (elementos orbitais), que, por sua vez, estão expostos na Tabela 3.

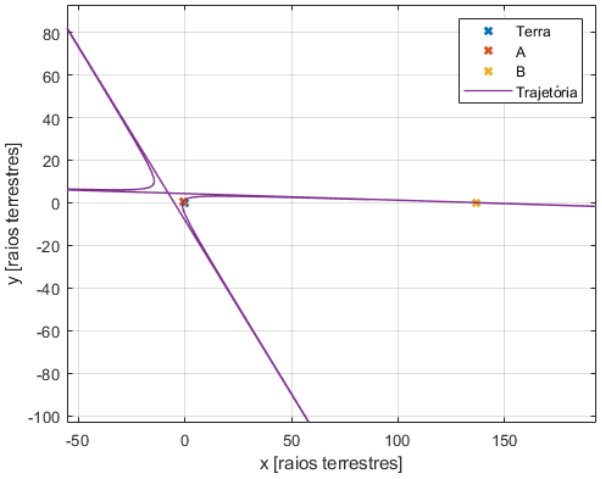
Tabela 3: Caracterização dos trechos de cônicas coladas para a trajetória ótima encontrada

Órbita	**AB**	**BC**	**CD**
Energia $[km^2/s^2]$	3.8852	-351.5353	3.4969
Momento linear $(\vec{h})$ $[km^2/s]$	(0,0,7.7254)e04	(0,0,4.8969)e09	(0,0,-2.2617)e04
Semieixo maior (a) $[km]$	-5.1294e04	1.8880e08	-6.979e03
Excentricidade (e)	1.1366	0.2078	1.725
Rotação da órbita (ϕ) [rad]	5.7569	1.6263	2.0425

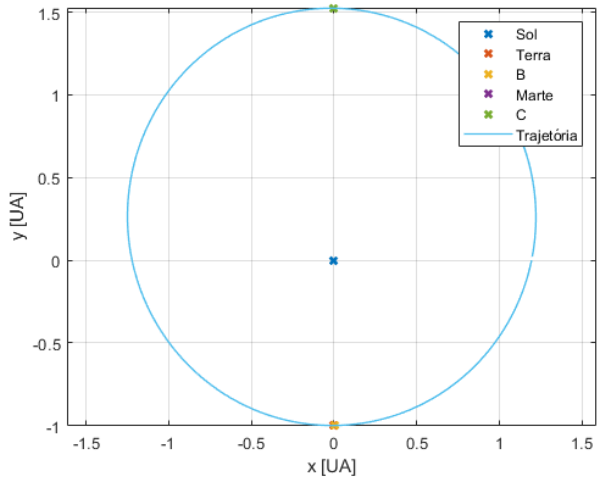
Tabela 4: Caracterização dos pontos relevantes para a trajetória ótima encontrada

Ponto	**A**	**B**	**C**	**D**
Referencial	Terra	Sol	Sol	Marte
$V_{chegada}$	7.5415	32.7336	2.6726	5.1336
V_{saida}	11.0235	32.7342	2.6741	3.1113
ΔV	3.4820	0.0007	0.0015	2.0223

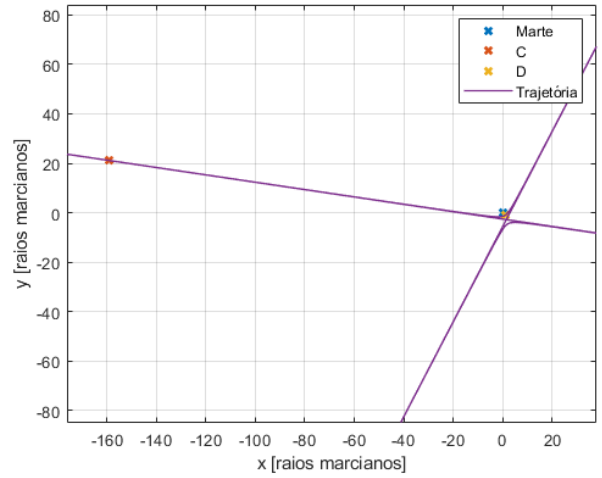
Assim, as trajetórias resultantes são as apresentadas na Figura 3 e repetidas com melhor dimensionamento na Figura 4 para melhor visualização da trajetória nas proximidades do planeta. Note a presença das assíntotas características das trajetórias hiperbólicas, que não fazem parte da órbita em si, mas foram colocadas para fins ilustrativos.



(a) Trajetória de escape da Terra

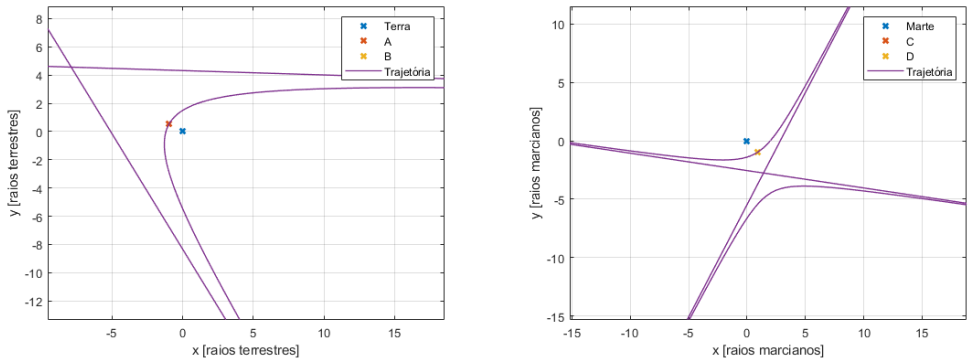


(b) Trajetória de viagem no Sistema Solar



(c) Trajetória de captura gravitacional por Marte

Figura 3: Arcos de cônica que constituem a transferência interplanetária proposta



(a) Escape da Terra. (b) Aproximação de Marte.

Figura 4: Trajetórias hiperbólicas de excesso com omissão do ponto de escape e de captura.

4 Análises

4.1 Comparação com a literatura

A Administração Nacional da Aeronáutica e Espaço (NASA) dos Estados Unidos possui um site, o qual oferece uma ferramenta que busca trajetórias espaciais. Nesta ferramenta, é possível inserir o destino e a janela de lançamento de interesse e, com base nessas informações, o programa resulta vários dados interessantes, dentre eles o tempo de duração para cada trajetória que cumpre o percurso, como também os incrementos e velocidades totais do percurso. O algoritmo envolvido utiliza o método do problema de Lambert, usando uma data de lançamento da Terra e uma duração de transferência até o objeto de destino, e assim determinando a órbita que conecta os dois corpos. Em outras palavras, as informações geradas pelo programa condizem com o incremento de velocidade necessário para que a espaçonave entre na trajetória de transferência e qual a duração dessa viagem. Tais informações se tornam bastante relevantes para verificação dos dados encontrados neste trabalho. Contudo, é importante ressaltar que tais dados obtidos no site da NASA servem apenas para um levantamento de dados preliminares ou mesmo suposições iniciais para futuros projetos, já que o programa assume que pequenos corpos podem ter massa-zero e desconsidera outras perturbações.

Dessa forma, inseriu-se no programa da NASA os seguintes parâmetros:

- Ano de lançamento 2010 - 2040;
- 300 dias de duração da trajetória de transferência;
- 8 km/s sendo a variação máxima de velocidade total.

Data de lançamento da Terra	Data de chegada em Marte	Duração da trajetória de transferência (dias)	Energia necessária para sair SOI da Terra	Incremento de velocidade para entrar na trajetória de transferência (km/s)	Variação de velocidade total requerida (km/s)
Nov-29-2018	Aug-28-2019	272	10.9	3.71	4.72
Dec-06-2013	Sep-04-2014	272	10.1	3.68	4.73
Oct-30-2011	Jul-28-2012	272	9.2	3.63	4.75
Nov-02-2024	Jul-16-2025	256	24	4.27	4.87
Oct-11-2039	Jun-23-2040	256	21.9	4.18	4.84
Nov-24-2026	Aug-07-2027	256	13.8	3.84	4.68
Dec-01-2011	Aug-13-2012	256	11.7	3.75	4.84

Tabela 5: Valores de duração trajetória próximas ao calculado neste trabalho

Data de lançamento da Terra	Data de chegada em Marte	Duração da trajetória de transferência (dias)	Energia necessária para sair SOI da Terra	Incremento de velocidade para entrar na trajetória de transferência (km/s)	Variação de velocidade total requerida (km/s)
May-19-2033	Sep-08-2033	112	32.3	4.6	7.32
Mar-11-2031	Aug-02-2031	144	28.7	4.46	7.28
Oct-12-2022	Mar-05-2023	144	42.7	5.02	7.28
Aug-19-2020	Dec-09-2020	112	31.7	4.58	7.22
Dec-17-2011	Jul-12-2012	208	19.1	4.06	5.71
May-03-2033	Sep-24-2033	144	15	3.89	5.59
Dec-31-2028	Jul-27-2029	208	12.9	3.8	5.58
Nov-02-2024	May-29-2025	208	24.8	4.3	5.58
Aug-19-2020	Jan-10-2021	144	25.6	4.33	5.55

Tabela 6: Valores de variação de velocidade total próximas ao calculado neste trabalho

O programa gerou um total de 100 trajetórias possíveis, e dentro dessas possíveis buscou-se valores próximos encontrados ao produzir esse trabalho, compiladas nas Tabelas 5 e 6.

A ferramenta do site da NASA também possibilita uma animação de como seria realizado a trajetória de cada uma das possíveis trajetórias inicialmente oferecidas. A partir dessa animação obtivemos as Figuras 5 e 6, as quais facilitam a compreensão de onde seriam os pontos de lançamento e interceptação do objeto de estudo. É interessante notar que, quanto maior a variação de velocidade durante a transferência, menor é o arco da trajetória.

4.2 Discussões e propostas de melhoria

No mais, temos que os resultados para custo encontrados neste trabalho condizem com a literatura [6] para as órbitas a que se propõem, como pode-se ver nas Tabelas 7 e 8, que indicam custos típicos, tanto para a transferência de uma órbita LEO

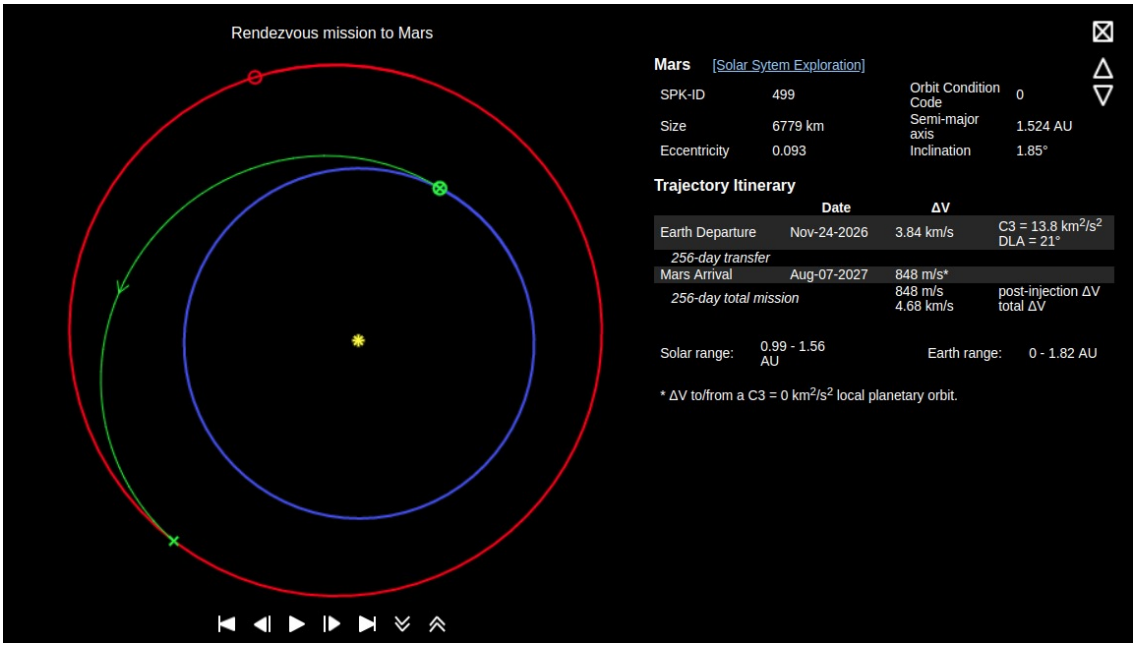


Figura 5: Trajetória possível com a duração mais próxima encontrada no trabalho.

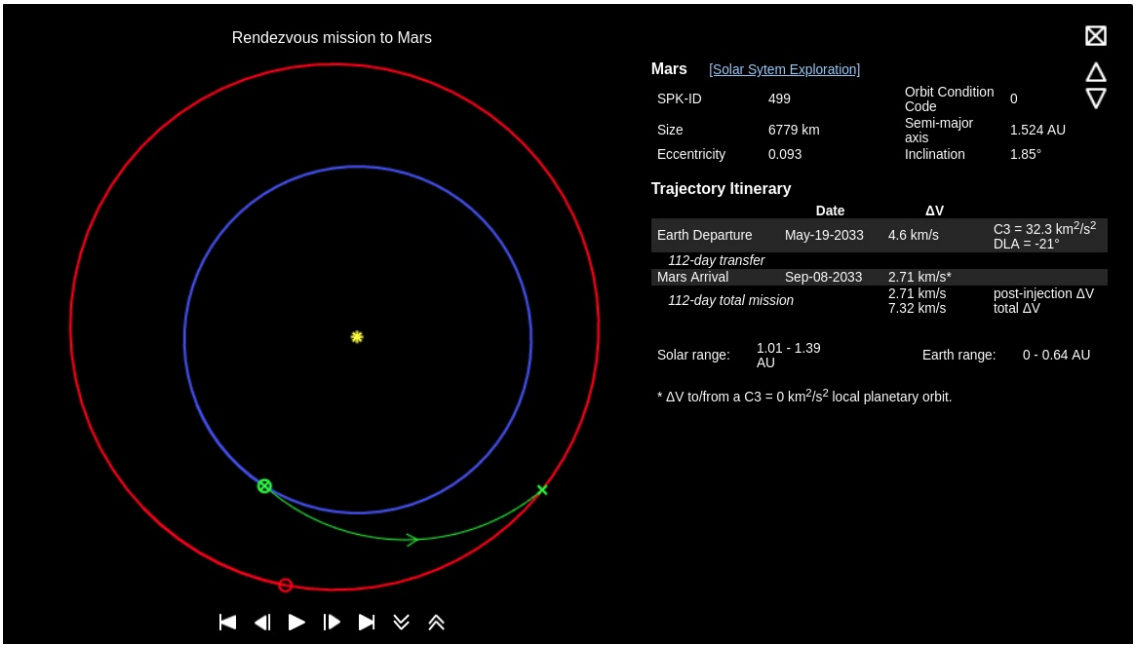


Figura 6: Possível trajetória com a maior variação de velocidade similar a encontrada neste trabalho.

From	To	Delta-v (km/s)
LEO	Mars transfer orbit	4.3[6] ("typical", not minimal)
Earth escape velocity (C3=0)	Mars transfer orbit	0.6[7]
Mars transfer orbit	Mars capture orbit	0.9[7]
Mars capture orbit	Deimos transfer orbit	0.2[7]
Deimos transfer orbit	Deimos surface	0.7[7]
Deimos transfer orbit	Phobos transfer orbit	0.3[7]
Phobos transfer orbit	Phobos surface	0.5[7]
Mars capture orbit	Low Mars orbit	1.4[7]
Low Mars orbit	Mars surface	4.1[7]
Earth–Moon Lagrange point 2	Mars transfer orbit	<1.0[6]
Mars transfer orbit	Low Mars orbit	2.7[6] (not minimal)
Earth escape velocity (C3=0)	Closest NEO[8]	0.8–2.0

Figura 7: Custos clássicos envolvidos em viagens interplanetárias.

para a órbita de transferência de Marte quando para a órbita de captura por Marte, competitivos com os encontrados, não só para a trajetória de menor custo encontrada, como para várias outras trajetórias possíveis apontadas como mínimo locais e retornadas pelo algoritmo de otimização com possíveis trajetórias. De uma maneira geral, o custo otimizado apontado para uma transferência interplanetária para Marte, partindo de uma órbita LEO, é de 3.6 km/s (Tabela 8), sendo $4.3 + 0.9 = 5.2$ km/s um valor típico para uma órbita de transferência para Marte com finalização em órbita de captura marciana (Tabela 7), mostrando possíveis otimizações e considerações a serem feitas, sem descartar a sua coerência, para reduzir o valor de aproximadamente 5.5 km/s encontrados neste estudo.

As propostas de melhorias giram em torno de considerar órbitas com inclinação entre os seus planos, tanto os característicos dos planetas envolvidos como os das órbitas de transferência, o que pode abrir novas oportunidades de otimização e consequente minimização do custo. Outro fator prático seria explorar melhor o conjunto de soluções aumentando o número de iterações, o espaço amostral e a quantidade de partículas associadas ao método de otimização utilizado.

Destination	Orbital radius (AU)	Δv to enter Hohmann orbit from Earth's orbit	v exiting LEO	Δv from LEO
Sun	0	29.8	31.7	24.0
Mercury	0.39	7.5	13.3	5.5
Venus	0.72	2.5	11.2	3.5
Mars	1.52	2.9	11.3	3.6
Jupiter	5.2	8.8	14.0	6.3
Saturn	9.54	10.3	15.0	7.3
Uranus	19.19	11.3	15.7	8.0
Neptune	30.07	11.7	16.0	8.2
Pluto	29.66 (perih.)	11.6	16.0	8.2
Infinity	∞	12.3	16.5	8.8

Figura 8: Custos generalizados envolvidos em transferências interplanetárias em função do destino.

4.3 Análise dos dados obtidos

Temos que os dados obtidos neste trabalho foram, como dito acima, condizentes com o esperado, tanto em valores típicos como em formato da trajetória, quanto às cônicas envolvidas, e ao tempo de duração da transferência. Observa-se pela Tabela 3 que as trajetórias de saída e chegada, de Terra e Marte, respectivamente, são naturalmente hipérboles, advindas da análise de minimização feitas, e não de consideração alguma no método.

O algoritmo mostrou-se tendencioso a minimizar o impulso associado ao escape da Terra, promovendo uma trajetória hiperbólica de excesso que naturalmente já o levava para fora do campo gravitacional e impulsionava a espaçonave à trajetória em torno do Sol praticamente com a velocidade remanescente do primeiro incremento de velocidade (relativo ao ponto A, para deixar a Terra).

Quanto à chegada em Marte, a mesma tendência se observa, ao ponto que o projeto de órbita faz com que a espaçonave chegue nos arredores da esfera de influência do planeta com velocidade similar à dele, mas um pouco maior, promovendo uma captura gravitacional a partir de uma trajetória hiperbólica de aproximação que surge naturalmente. Tal abordagem nasce a partir da oportunidade de aproveitar a energia da órbita de transferência em torno do Sol para se aproximar do planeta na sua esfera de influência, reduzindo assim o impulso necessário para correção de órbita e colagem da última cônica.

5 Conclusão

Dada a importância do tema transferências interplanetárias para o contexto das faculdades de mecânica celeste e para a formação de engenheiros aeroespaciais, conclui-se com proveito o presente trabalho com a sensação de ter feito análises que não corromperam pelo simplicismo, mas que mostram-se dispostas a abraçar novas correções e otimizações que podem dar a luz a novos trabalhos, principalmente para turmas vindouras. A pesquisa bibliográfica para a realização das análises e estudo das metodologias para desenvolvimento do tema foi de certo enriquecedor e intrigaram a novos questionamentos a medida que respostas à dúvidas iam sendo desvendadas.

A margem que este trabalho abre para futuros estudos, consagrando-se como uma abordagem inicial a problemas de fato pertinentes para o ramo do aeroespaço e fazendo pinceladas e retraduções em algumas áreas estudadas durante este ano na disciplina de Mecânica Orbital, pondera uma motivação para desenvolvimento de trabalhos que podem consistir iniciações científicas, trabalhos de conclusão de curso ou mesmo defesas de teses de pós-graduação.

Quanto à utilização do algoritmo de otimização conhecido como *Algoritmo da Migração dos Pássaros* ou, oficialmente *Particle Swarm Optimization* (Otimização do Enxame de Partículas, em tradução livre), obteve-se resultados interessantes e pertinentes quando comparado aos valores clássicos esperados pela literatura, promovendo, como discutido, gratificação de estar no caminho certo, mas sem deixar de lado a visão de que há novas formas de se abordar o problema para buscar resultados cada vez melhores.

O problema de Lambert, por fim, é uma boa oportunidade de propor exercícios para revalidar resultados clássicos, ao passo que também se faz louvável quanto à possibilidade de implementar as nuances e técnicas abordadas em sala de aula para ver o resultado na prática, mas sem sair da esfera tangível que o problema de 2 corpos nos proporciona. As análises feitas a partir das metodologias propostas podem ser bem vistas tanto sob a perspectiva numérica quanto a analítica, dando vazão a diversas abordagens, como pode-se ver com a recorrência da escolha deste tema nos demais trabalhos de conclusão da disciplina de MVO-41, Mecânica Orbital, pelos diversos grupos.

Apêndice A

Códigos[3]

Solução para o Problema de Lambert

```
% LAMBERT            Lambert-targeter for ballistic flights
%                    (Izzo, and Lancaster, Blanchard & Gooding)
%
% Usage:
%    [V1, V2, extremal_distances, exitflag] = lambert(r1, r2, tf, m, GM_central)
%
% Dimensions:
%              r1, r2 ->  [1x3]
%              V1, V2 ->  [1x3]
% extremal_distances ->  [1x2]
%               tf, m ->  [1x1]
%         GM_central ->  [1x1]
%
% This function solves any Lambert problem *robustly*. It uses two separate
% solvers; the first one tried is a new and unpublished algorithm developed
% by Dr. D. Izzo from the European Space Agency [1]. This version is extremely
% fast, but especially for larger [m] it still fails quite frequently. In such
% cases, a MUCH more robust algorithm is started (the one by Lancaster &
% Blancard [2], with modifcations, initial values and other improvements by
% R.Gooding [3]), which is a lot slower partly because of its robustness.
%
% INPUT ARGUMENTS:
% ================================================================================
%    name         units     description
% ================================================================================
%   r1, r1        [km]      position vectors of the two terminal points.
%     tf         [days]     time of flight to solve for
%      m           [-]      specifies the number of complete orbits to complete
%                           (should be an integer)
% GM_central   [km3/s2]    std. grav. parameter [U+FFFD]M= mu) of the central body
%
% OUTPUT ARGUMENTS:
% ================================================================================
%   name              units    description
% ================================================================================
%  V1, V2             [km/s]   terminal velocities at the end-points
%  extremal_distances  [km]    minimum(1) and maximum(2) distance of the
%                              spacecraft to the central body.
%  exitflag             [-]    Integer containing information on why the
%                              routine terminated. A value of +1 indicates
%                              success; a normal exit. A value of -1
%                              indicates that the given problem has no
%                              solution and cannot be solved. A value of -2
%                              indicates that both algorithms failed to find
```

[3]Estão expostos aqui apenas as principais rotinas utilizadas. Contudo, todos os códigos utilizados estão integralmente disponíveis em https://github.com/chicomcastro/Earth-Mars-optimized-transfer.

```
45 %                               a solution. This should never occur since
46 %                               these problems are well-defined, and at the
47 %                               very least it can be determined that the
48 %                               problem has no solution. Nevertheless, it
49 %                               still occurs sometimes for accidental
50 %                               erroneous input, so it provides a basic
51 %                               mechanism to check any application using this
52 %                               algorithm.
53 %
54 % This routine can be compiled to increase its speed by a factor of about
55 % 10-15, which is certainly advisable when the complete application requires
56 % a great number of Lambert problems to be solved. The entire routine is
57 % written in embedded MATLAB, so it can be compiled with the emlmex()
58 % function (older MATLAB) or codegen() function (MATLAB 2011a and later).
59 %
60 % To do this using emlmex(), make sure MATLAB's current directory is equal
61 % to where this file is located. Then, copy-paste and execute the following
62 % commands to the command window:
63 %
64 %    example_input = {...
65 %         [0.0, 0.0, 0.0], ...% r1vec
66 %         [0.0, 0.0, 0.0], ...% r2vec
67 %          0.0, ...           % tf
68 %          0.0, ...           % m
69 %          0.0};              % muC
70 %    emlmex -eg example_input lambert.m
71 %
72 % This is of course assuming your compiler is configured correctly. See the
73 % documentation of emlmex() on how to do that.
74 %
75 % Using codegen(), the syntax is as follows:
76 %
77 %    example_input = {...
78 %         [0.0, 0.0, 0.0], ...% r1vec
79 %         [0.0, 0.0, 0.0], ...% r2vec
80 %          0.0, ...           % tf
81 %          0.0, ...           % m
82 %          0.0};              % muC
83 %    codegen lambert.m -args example_input
84 %
85 % Note that in newer MATLAB versions, the code analyzer will complain about
86 % the pragma "%#eml" after the main function's name, and possibly, issue
87 % subsequent warnings related to this issue. To get rid of this problem, simply
88 % replace the "%#eml" directive with "%#codegen".
89 %
90 %
91 %
92 % References:
93 %
94 % [1] Izzo, D. ESA Advanced Concepts team. Code used available in MGAM, on
95 %     http://www.esa.int/gsp/ACT/inf/op/globopt.htm. Last retreived Nov, 2009.
96 % [2] Lancaster, E.R. and Blanchard, R.C. "A unified form of Lambert's theorem."
97 %     NASA technical note TN D-5368,1969.
98 % [3] Gooding, R.H. "A procedure for the solution of Lambert's orbital ...
      boundary-value
99 %     problem. Celestial Mechanics and Dynamical Astronomy, 48:145[U+FFFD]165,1990.
```

```
100 %
101 % See also lambert_low_ExpoSins.
102
103
104 % Please report bugs and inquiries to:
105 %
106 % Name       : Rody P.S. Oldenhuis
107 % E-mail     : oldenhuis@gmail.com
108 % Licence    : 2-clause BSD (see License.txt)
109
110
111 % If you find this work useful, please consider a donation:
112 % https://www.paypal.me/RodyO/3.5
113
114 % If you want to cite this work in an academic paper, please use
115 % the following template:
116 %
117 % Rody Oldenhuis, orcid.org/0000-0002-3162-3660. "Lambert" <version>,
118 % <date you last used it>. MATLAB Robust solver for Lambert's
119 % orbital-boundary value problem.
120 % https://nl.mathworks.com/matlabcentral/fileexchange/26348
121
122
123
124 % ------------------------------------------------------------------
125 % Izzo's version:
126 % Very fast, but not very robust for more complicated cases
127 % ------------------------------------------------------------------
128 function [V1, ...
129           V2, ...
130           extremal_distances, ...
131           exitflag] = lambert(r1vec, ...
132                               r2vec, ...
133                               tf, ...
134                               m, ...
135                               muC) %#coder
136
137 % original documentation:
138 %{
139  This routine implements a new algorithm that solves Lambert's problem. The
140  algorithm has two major characteristics that makes it favorable to other
141  existing ones.
142
143  1) It describes the generic orbit solution of the boundary condition
144  problem through the variable X=log(1+cos(alpha/2)). By doing so the
145  graph of the time of flight become defined in the entire real axis and
146  resembles a straight line. Convergence is granted within few iterations
147  for all the possible geometries (except, of course, when the transfer
148  angle is zero). When multiple revolutions are considered the variable is
149  X=tan(cos(alpha/2)*pi/2).
150
151  2) Once the orbit has been determined in the plane, this routine
152  evaluates the velocity vectors at the two points in a way that is not
153  singular for the transfer angle approaching to pi (Lagrange coefficient
154  based methods are numerically not well suited for this purpose).
155
```

```
156  As a result Lambert's problem is solved (with multiple revolutions
157  being accounted for) with the same computational effort for all
158  possible geometries. The case of near 180 transfers is also solved
159  efficiently.
160
161   We note here that even when the transfer angle is exactly equal to pi
162  the algorithm does solve the problem in the plane (it finds X), but it
163  is not able to evaluate the plane in which the orbit lies. A solution
164  to this would be to provide the direction of the plane containing the
165  transfer orbit from outside. This has not been implemented in this
166  routine since such a direction would depend on which application the
167  transfer is going to be used in.
168
169  please report bugs to dario.izzo@esa.int
170 %}
171
172 % adjusted documentation:
173 %{
174  By default, the short-way solution is computed. The long way solution
175  may be requested by giving a negative value to the corresponding
176  time-of-flight [tf].
177
178  For problems with |m| > 0, there are generally two solutions. By
179  default, the right branch solution will be returned. The left branch
180  may be requested by giving a negative value to the corresponding
181  number of complete revolutions [m].
182 %}
183
184 % Authors
185 % .-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.
186 % Name       : Dr. Dario Izzo
187 % E-mail     : dario.izzo@esa.int
188 % Affiliation: ESA / Advanced Concepts Team (ACT)
189
190 % Made more readible and optimized for speed by Rody P.S. Oldenhuis
191 % Code available in MGAM on   http://www.esa.int/gsp/ACT/inf/op/globopt.htm
192
193 % last edited 12/Dec/2009
194
195 % ADJUSTED FOR EML-COMPILATION 24/Dec/2009
196
197     % initial values
198     tol = 1e-14;    bad = false;     days = 86400;
199
200     % work with non-dimensional units
201     r1 = sqrt(r1vec*r1vec.');  r1vec = r1vec/r1;
202     V = sqrt(muC/r1);          r2vec = r2vec/r1;
203     T = r1/V;                  tf    = tf*days/T; % also transform to seconds
204
205     % relevant geometry parameters (non dimensional)
206     mr2vec = sqrt(r2vec*r2vec.');
207     % make 100% sure it's in (-1 <= dth <= +1)
208     dth = acos( max(-1, min(1, (r1vec*r2vec.')/mr2vec)) );
209
210     % decide whether to use the left or right branch (for multi-revolution
211     % problems), and the long- or short way
```

```
212     leftbranch = sign(m);    longway = sign(tf);
213     m = abs(m);              tf = abs(tf);
214     if (longway < 0), dth = 2*pi - dth; end
215
216     % derived quantities
217     c       = sqrt(1 + mr2vec^2 - 2*mr2vec*cos(dth)); % non-dimensional chord
218     s       = (1 + mr2vec + c)/2;                     % non-dimensional ...
            semi-perimeter
219     a_min   = s/2;                                    % minimum energy ...
            ellipse semi major axis
220     Lambda = sqrt(mr2vec)*cos(dth/2)/s;               % lambda parameter ...
            (from BATTIN's book)
221     crossprd = [r1vec(2)*r2vec(3) - r1vec(3)*r2vec(2),...
222                 r1vec(3)*r2vec(1) - r1vec(1)*r2vec(3),...% non-dimensional ...
                        normal vectors
223                 r1vec(1)*r2vec(2) - r1vec(2)*r2vec(1)];
224     mcr        = sqrt(crossprd*crossprd.');           % magnitues thereof
225     nrmunit    = crossprd/mcr;                        % unit vector thereof
226
227     % Initial values
228     % ------------------------------------------------------------
229
230     % ELMEX requires this variable to be declared OUTSIDE the IF-statement
231     logt = log(tf); % avoid re-computing the same value
232
233     % single revolution (1 solution)
234     if (m == 0)
235
236         % initial values
237         inn1 = -0.5233;      % first initial guess
238         inn2 = +0.5233;      % second initial guess
239         x1   = log(1 + inn1);% transformed first initial guess
240         x2   = log(1 + inn2);% transformed first second guess
241
242         % multiple revolutions (0, 1 or 2 solutions)
243         % the returned soltuion depends on the sign of [m]
244     else
245         % select initial values
246         if (leftbranch < 0)
247             inn1 = -0.5234; % first initial guess, left branch
248             inn2 = -0.2234; % second initial guess, left branch
249         else
250             inn1 = +0.7234; % first initial guess, right branch
251             inn2 = +0.5234; % second initial guess, right branch
252         end
253         x1 = tan(inn1*pi/2);% transformed first initial guess
254         x2 = tan(inn2*pi/2);% transformed first second guess
255     end
256
257     % since (inn1, inn2) < 0, initial estimate is always ellipse
258     xx   = [inn1, inn2];  aa = a_min./(1 - xx.^2);
259     bbeta = longway * 2*asin(sqrt((s-c)/2./aa));
260     % make 100.4% sure it's in (-1 <= xx <= +1)
261     aalfa = 2*acos(  max(-1, min(1, xx)) );
262
263     % evaluate the time of flight via Lagrange expression
```

```
264     y12  = aa.*sqrt(aa).*((aalfa - sin(aalfa)) - (bbeta-sin(bbeta)) + 2*pi*m);
265
266     % initial estimates for y
267     if m == 0
268         y1 = log(y12(1)) - logt;
269         y2 = log(y12(2)) - logt;
270     else
271         y1 = y12(1) - tf;
272         y2 = y12(2) - tf;
273     end
274
275     % Solve for x
276     % ------------------------------------------------------------
277
278     % Newton-Raphson iterations
279     % NOTE - the number of iterations will go to infinity in case
280     % m > 0  and there is no solution. Start the other routine in
281     % that case
282     err = inf;  iterations = 0; xnew = 0;
283     while (err > tol)
284         % increment number of iterations
285         iterations = iterations + 1;
286         % new x
287         xnew = (x1*y2 - y1*x2) / (y2-y1);
288         % copy-pasted code (for performance)
289         if m == 0, x = exp(xnew) - 1; else x = atan(xnew)*2/pi; end
290         a = a_min/(1 - x^2);
291         if (x < 1) % ellipse
292             beta = longway * 2*asin(sqrt((s-c)/2/a));
293             % make 100.4% sure it's in (-1 <= xx <= +1)
294             alfa = 2*acos( max(-1, min(1, x)) );
295         else % hyperbola
296             alfa = 2*acosh(x);
297             beta = longway * 2*asinh(sqrt((s-c)/(-2*a)));
298         end
299         % evaluate the time of flight via Lagrange expression
300         if (a > 0)
301             tof = a*sqrt(a)*((alfa - sin(alfa)) - (beta-sin(beta)) + 2*pi*m);
302         else
303             tof = -a*sqrt(-a)*((sinh(alfa) - alfa) - (sinh(beta) - beta));
304         end
305         % new value of y
306         if m ==0, ynew = log(tof) - logt; else ynew = tof - tf; end
307         % save previous and current values for the next iterarion
308         % (prevents getting stuck between two values)
309         x1 = x2;  x2 = xnew;
310         y1 = y2;  y2 = ynew;
311         % update error
312         err = abs(x1 - xnew);
313         % escape clause
314         if (iterations > 15), bad = true; break; end
315     end
316
317     % If the Newton-Raphson scheme failed, try to solve the problem
318     % with the other Lambert targeter.
319     if bad
```

```
320         % NOTE: use the original, UN-normalized quantities
321         [V1, V2, extremal_distances, exitflag] = ...
322             lambert_LancasterBlanchard(r1vec*r1, r2vec*r1, longway*tf*T, ...
                leftbranch*m, muC);
323         return
324     end
325
326     % convert converged value of x
327     if m==0, x = exp(xnew) - 1; else x = atan(xnew)*2/pi; end
328
329     %{
330       The solution has been evaluated in terms of log(x+1) or tan(x*pi/2), we
331       now need the conic. As for transfer angles near to pi the Lagrange-
332       coefficients technique goes singular (dg approaches a zero/zero that is
333       numerically bad) we here use a different technique for those cases. When
334       the transfer angle is exactly equal to pi, then the ih unit vector is not
335       determined. The remaining equations, though, are still valid.
336     %}
337
338     % Solution for the semi-major axis
339     a = a_min/(1-x^2);
340
341     % Calculate psi
342     if (x < 1) % ellipse
343         beta = longway * 2*asin(sqrt((s-c)/2/a));
344         % make 100.4% sure it's in (-1 ≤ xx ≤ +1)
345         alfa = 2*acos( max(-1, min(1, x)) );
346         psi  = (alfa-beta)/2;
347         eta2 = 2*a*sin(psi)^2/s;
348         eta  = sqrt(eta2);
349     else       % hyperbola
350         beta = longway * 2*asinh(sqrt((c-s)/2/a));
351         alfa = 2*acosh(x);
352         psi  = (alfa-beta)/2;
353         eta2 = -2*a*sinh(psi)^2/s;
354         eta  = sqrt(eta2);
355     end
356
357     % unit of the normalized normal vector
358     ih = longway * nrmunit;
359
360     % unit vector for normalized [r2vec]
361     r2n = r2vec/mr2vec;
362
363     % cross-products
364     % don't use cross() (emlmex() would try to compile it, and this way it
365     % also does not create any additional overhead)
366     crsprd1 = [ih(2)*r1vec(3)-ih(3)*r1vec(2),...
367                ih(3)*r1vec(1)-ih(1)*r1vec(3),...
368                ih(1)*r1vec(2)-ih(2)*r1vec(1)];
369     crsprd2 = [ih(2)*r2n(3)-ih(3)*r2n(2),...
370                ih(3)*r2n(1)-ih(1)*r2n(3),...
371                ih(1)*r2n(2)-ih(2)*r2n(1)];
372
373     % radial and tangential directions for departure velocity
374     Vr1 = 1/eta/sqrt(a_min) * (2*Lambda*a_min - Lambda - x*eta);
```

```
375      Vt1 = sqrt(mr2vec/a_min/eta2 * sin(dth/2)^2);
376
377      % radial and tangential directions for arrival velocity
378      Vt2 = Vt1/mr2vec;
379      Vr2 = (Vt1 - Vt2)/tan(dth/2) - Vr1;
380
381      % terminal velocities
382      V1 = (Vr1*r1vec + Vt1*crsprd1)*V;
383      V2 = (Vr2*r2n + Vt2*crsprd2)*V;
384
385      % exitflag
386      exitflag = 1; % (success)
387
388      % also compute minimum distance to central body
389      % NOTE: use un-transformed vectors again!
390      extremal_distances = ...
391          minmax_distances(r1vec*r1, r1, r2vec*r1, mr2vec*r1, dth, a*r1, V1, ...
                 V2, m, muC);
392
393  end
394
395  %-------------------------------------------------------------------------
396  % Lancaster & Blanchard version, with improvements by Gooding
397  % Very reliable, moderately fast for both simple and complicated cases
398  %-------------------------------------------------------------------------
399  function [V1, ...
400            V2, ...
401            extremal_distances, ...
402            exitflag] = lambert_LancasterBlanchard(r1vec, ...
403                                                   r2vec, ...
404                                                   tf, ...
405                                                   m, ...
406                                                   muC) %#coder
407  %{
408  LAMBERT_LANCASTERBLANCHARD       High-Thrust Lambert-targeter
409
410  lambert_LancasterBlanchard() uses the method developed by
411  Lancaster & Blancard, as described in their 1969 paper. Initial values,
412  and several details of the procedure, are provided by R.H. Gooding,
413  as described in his 1990 paper.
414  %}
415
416  % Please report bugs and inquiries to:
417  %
418  % Name        : Rody P.S. Oldenhuis
419  % E-mail      : oldenhuis@gmail.com
420  % Licence     : 2-clause BSD (see License.txt)
421
422
423  % If you find this work useful, please consider a donation:
424  % https://www.paypal.me/RodyO/3.5
425
426      % ADJUSTED FOR EML-COMPILATION 29/Sep/2009
427
428      % manipulate input
```

```
429     tol     = 1e-12;                                % optimum for numerical ...
            noise v.s. actual precision
430     r1      = sqrt(r1vec*r1vec.');                  % magnitude of r1vec
431     r2      = sqrt(r2vec*r2vec.');                  % magnitude of r2vec
432     r1unit  = r1vec/r1;                             % unit vector of r1vec
433     r2unit  = r2vec/r2;                             % unit vector of r2vec
434     crsprod = cross(r1vec, r2vec, 2);               % cross product of r1vec ...
            and r2vec
435     mcrsprd = sqrt(crsprod*crsprod.');              % magnitude of that cross ...
            product
436     th1unit = cross(crsprod/mcrsprd, r1unit);       % unit vectors in the ...
            tangential-directions
437     th2unit = cross(crsprod/mcrsprd, r2unit);
438     % make 100.4% sure it's in (-1 ≤ x ≤ +1)
439     dth = acos( max(-1, min(1, (r1vec*r2vec.')/r1/r2)) ); % turn angle
440
441     % if the long way was selected, the turn-angle must be negative
442     % to take care of the direction of final velocity
443     longway = sign(tf); tf = abs(tf);
444     if (longway < 0), dth = dth-2*pi; end
445
446     % left-branch
447     leftbranch = sign(m); m = abs(m);
448
449     % define constants
450     c  = sqrt(r1^2 + r2^2 - 2*r1*r2*cos(dth));
451     s  = (r1 + r2 + c) / 2;
452     T  = sqrt(8*muC/s^3) * tf;
453     q  = sqrt(r1*r2)/s * cos(dth/2);
454
455     % general formulae for the initial values (Gooding)
456     % ---------------------------------------------------
457
458     % some initial values
459     T0  = LancasterBlanchard(0, q, m);
460     Td  = T0 - T;
461     phr = mod(2*atan2(1 - q^2, 2*q), 2*pi);
462
463     % initial output is pessimistic
464     V1 = NaN(1,3);     V2 = V1;     extremal_distances = [NaN, NaN];
465
466     % single-revolution case
467     if (m == 0)
468         x01 = T0*Td/4/T;
469         if (Td > 0)
470             x0 = x01;
471         else
472             x01 = Td/(4 - Td);
473             x02 = -sqrt( -Td/(T+T0/2) );
474             W   = x01 + 1.7*sqrt(2 - phr/pi);
475             if (W ≥ 0)
476                 x03 = x01;
477             else
478                 x03 = x01 + (-W).^(1/16).*(x02 - x01);
479             end
480             lambda = 1 + x03*(1 + x01)/2 - 0.03*x03^2*sqrt(1 + x01);
```

```
481             x0 = lambda*x03;
482         end
483
484         % this estimate might not give a solution
485         if (x0 < -1), exitflag = -1; return; end
486
487     % multi-revolution case
488     else
489
490         % determine minimum Tp(x)
491         xMpi = 4/(3*pi*(2*m + 1));
492         if (phr < pi)
493             xM0 = xMpi*(phr/pi)^(1/8);
494         elseif (phr > pi)
495             xM0 = xMpi*(2 - (2 - phr/pi)^(1/8));
496         % EMLMEX requires this one
497         else
498             xM0 = 0;
499         end
500
501         % use Halley's method
502         xM = xM0;   Tp = inf;   iterations = 0;
503         while abs(Tp) > tol
504             % iterations
505             iterations = iterations + 1;
506             % compute first three derivatives
507             [dummy, Tp, Tpp, Tppp] = LancasterBlanchard(xM, q, m);%#ok
508             % new value of xM
509             xMp = xM;
510             xM  = xM - 2*Tp.*Tpp ./ (2*Tpp.^2 - Tp.*Tppp);
511             % escape clause
512             if mod(iterations, 7), xM = (xMp+xM)/2; end
513             % the method might fail. Exit in that case
514             if (iterations > 25), exitflag = -2; return; end
515         end
516
517         % xM should be elliptic (-1 < x < 1)
518         % (this should be impossible to go wrong)
519         if (xM < -1) || (xM > 1), exitflag = -1; return; end
520
521         % corresponding time
522         TM = LancasterBlanchard(xM, q, m);
523
524         % T should lie above the minimum T
525         if (TM > T), exitflag = -1; return; end
526
527         % find two initial values for second solution (again with ...
                lambda-type patch)
528         % ...
                ------------------------------------------------------------------------
529
530         % some initial values
531         TmTM  = T - TM;     T0mTM = T0 - TM;
532         [dummy, Tp, Tpp] = LancasterBlanchard(xM, q, m);%#ok
533
534         % first estimate (only if m > 0)
```

```
535         if leftbranch > 0
536             x   = sqrt( TmTM / (Tpp/2 + TmTM/(1-xM)^2) );
537             W   = xM + x;
538             W   = 4*W/(4 + TmTM) + (1 - W)^2;
539             x0  = x*(1 - (1 + m + (dth - 1/2)) / ...
540                 (1 + 0.15*m)*x*(W/2 + 0.03*x*sqrt(W))) + xM;
541
542             % first estimate might not be able to yield possible solution
543             if (x0 > 1), exitflag = -1; return; end
544
545         % second estimate (only if m > 0)
546         else
547             if (Td > 0)
548                 x0 = xM - sqrt(TM/(Tpp/2 - TmTM*(Tpp/2/T0mTM - 1/xM^2)));
549             else
550                 x00 = Td / (4 - Td);
551                 W = x00 + 1.7*sqrt(2*(1 - phr));
552                 if (W >= 0)
553                     x03 = x00;
554                 else
555                     x03 = x00 - sqrt((-W)^(1/8))*(x00 + sqrt(-Td/(1.5*T0 - ...
                            Td)));
556                 end
557                 W      = 4/(4 - Td);
558                 lambda = (1 + (1 + m + 0.24*(dth - 1/2)) / ...
559                     (1 + 0.15*m)*x03*(W/2 - 0.03*x03*sqrt(W)));
560                 x0     = x03*lambda;
561             end
562
563             % estimate might not give solutions
564             if (x0 < -1), exitflag = -1; return; end
565
566         end
567     end
568
569     % find root of Lancaster & Blancard's function
570     % ------------------------------------------------
571
572     % (Halley's method)
573     x = x0; Tx = inf; iterations = 0;
574     while abs(Tx) > tol
575         % iterations
576         iterations = iterations + 1;
577         % compute function value, and first two derivatives
578         [Tx, Tp, Tpp] = LancasterBlanchard(x, q, m);
579         % find the root of the *difference* between the
580         % function value [T_x] and the required time [T]
581         Tx = Tx - T;
582         % new value of x
583         xp = x;
584         x  = x - 2*Tx*Tp ./ (2*Tp^2 - Tx*Tpp);
585         % escape clause
586         if mod(iterations, 7), x = (xp+x)/2; end
587         % Halley's method might fail
588         if iterations > 25, exitflag = -2; return; end
589     end
```

```
590
591     % calculate terminal velocities
592     % ------------------------------
593
594     % constants required for this calculation
595     gamma = sqrt(muC*s/2);
596     if (c == 0)
597         sigma = 1;
598         rho   = 0;
599         z     = abs(x);
600     else
601         sigma = 2*sqrt(r1*r2/(c^2)) * sin(dth/2);
602         rho   = (r1 - r2)/c;
603         z     = sqrt(1 + q^2*(x^2 - 1));
604     end
605
606     % radial component
607     Vr1    = +gamma*((q*z - x) - rho*(q*z + x)) / r1;
608     Vr1vec = Vr1*r1unit;
609     Vr2    = -gamma*((q*z - x) + rho*(q*z + x)) / r2;
610     Vr2vec = Vr2*r2unit;
611
612     % tangential component
613     Vtan1    = sigma * gamma * (z + q*x) / r1;
614     Vtan1vec = Vtan1 * th1unit;
615     Vtan2    = sigma * gamma * (z + q*x) / r2;
616     Vtan2vec = Vtan2 * th2unit;
617
618     % Cartesian velocity
619     V1 = Vtan1vec + Vr1vec;
620     V2 = Vtan2vec + Vr2vec;
621
622     % exitflag
623     exitflag = 1; % (success)
624
625     % also determine minimum/maximum distance
626     a = s/2/(1 - x^2); % semi-major axis
627     extremal_distances = minmax_distances(r1vec, r1, r1vec, r2, dth, a, V1, ...
            V2, m, muC);
628
629 end
630
631 % Lancaster & Blanchard's function, and three derivatives thereof
632 function [T, Tp, Tpp, Tppp] = LancasterBlanchard(x, q, m)
633
634     % protection against idiotic input
635     if (x < -1) % impossible; negative eccentricity
636         x = abs(x) - 2;
637     elseif (x == -1) % impossible; offset x slightly
638         x = x + eps;
639     end
640
641     % compute parameter E
642     E  = x*x - 1;
643
644     % T(x), T'(x), T''(x)
```

```
645     if x == 1 % exactly parabolic; solutions known exactly
646         % T(x)
647         T = 4/3*(1-q^3);
648         % T'(x)
649         Tp = 4/5*(q^5 - 1);
650         % T''(x)
651         Tpp = Tp + 120/70*(1 - q^7);
652         % T'''(x)
653         Tppp = 3*(Tpp - Tp) + 2400/1080*(q^9 - 1);
654
655     elseif abs(x-1) < 1e-2 % near-parabolic; compute with series
656         % evaluate sigma
657         [sig1, dsigdx1, d2sigdx21, d3sigdx31] = sigmax(-E);
658         [sig2, dsigdx2, d2sigdx22, d3sigdx32] = sigmax(-E*q*q);
659         % T(x)
660         T = sig1 - q^3*sig2;
661         % T'(x)
662         Tp = 2*x*(q^5*dsigdx2 - dsigdx1);
663         % T''(x)
664         Tpp = Tp/x + 4*x^2*(d2sigdx21 - q^7*d2sigdx22);
665         % T'''(x)
666         Tppp = 3*(Tpp-Tp/x)/x + 8*x*x*(q^9*d3sigdx32 - d3sigdx31);
667
668     else % all other cases
669         % compute all substitution functions
670         y  = sqrt(abs(E));
671         z  = sqrt(1 + q^2*E);
672         f  = y*(z - q*x);
673         g  = x*z - q*E;
674
675         % BUGFIX: (Simon Tardivel) this line is incorrect for E==0 and f+g==0
676         % d  = (E < 0)*(atan2(f, g) + pi*m) + (E > 0)*log( max(0, f + g) );
677         % it should be written out like so:
678         if (E<0)
679             d = atan2(f, g) + pi*m;
680         elseif (E==0)
681             d = 0;
682         else
683             d = log(max(0, f+g));
684         end
685
686         % T(x)
687         T = 2*(x - q*z - d/y)/E;
688         %  T'(x)
689         Tp = (4 - 4*q^3*x/z - 3*x*T)/E;
690         % T''(x)
691         Tpp = (-4*q^3/z * (1 - q^2*x^2/z^2) - 3*T - 3*x*Tp)/E;
692         % T'''(x)
693         Tppp = (4*q^3/z^2*((1 - q^2*x^2/z^2) + 2*q^2*x/z^2*(z - x)) - 8*Tp ...
694             - 7*x*Tpp)/E;
695
696     end
697 end
698
699 % series approximation to T(x) and its derivatives
700 % (used for near-parabolic cases)
```

```
700 function [sig, dsigdx, d2sigdx2, d3sigdx3] = sigmax(y)
701
702     % preload the factors [an]
703     % (25 factors is more than enough for 16-digit accuracy)
704     persistent an;
705     if isempty(an)
706         an = [
707             4.000000000000000e-001;     2.142857142857143e-001;     ...
                    4.629629629629630e-002
708             6.628787878787879e-003;     7.211538461538461e-004;     ...
                    6.365740740740740e-005
709             4.741479925303455e-006;     3.059406328320802e-007;     ...
                    1.742836409255060e-008
710             8.892477331109578e-010;     4.110111531986532e-011;     ...
                    1.736709384841458e-012
711             6.759767240041426e-014;     2.439123386614026e-015;     ...
                    8.203411614538007e-017
712             2.583771576869575e-018;     7.652331327976716e-020;     ...
                    2.138860629743989e-021
713             5.659959451165552e-023;     1.422104833817366e-024;     ...
                    3.401398483272306e-026
714             7.762544304774155e-028;     1.693916882090479e-029;     ...
                    3.541295006766860e-031
715             7.105336187804402e-033];
716     end
717
718     % powers of y
719     powers = y.^(1:25);
720
721     % sigma itself
722     sig = 4/3 + powers*an;
723
724     % dsigma / dx (derivative)
725     dsigdx = ( (1:25).*[1, powers(1:24)] ) * an;
726
727     % d2sigma / dx2 (second derivative)
728     d2sigdx2 = ( (1:25).*(0:24).*[1/y, 1, powers(1:23)] ) * an;
729
730     % d3sigma / dx3 (third derivative)
731     d3sigdx3 = ( (1:25).*(0:24).*(-1:23).*[1/y/y, 1/y, 1, powers(1:22)] ) * an;
732
733 end
734
735
736 % ------------------------------------------------------------------------
737 % Helper functions
738 % ------------------------------------------------------------------------
739
740 % compute minimum and maximum distances to the central body
741 function extremal_distances = minmax_distances(r1vec, r1, ...
742                                                r2vec, r2, ...
743                                                dth, ...
744                                                a, ...
745                                                V1, V2, ...
746                                                m, ...
747                                                muC)
```

```
748
749     % default - minimum/maximum of r1,r2
750     minimum_distance = min(r1,r2);
751     maximum_distance = max(r1,r2);
752
753     % was the longway used or not?
754     longway = abs(dth) > pi;
755
756     % eccentricity vector (use triple product identity)
757     evec = ((V1*V1.')*r1vec - (V1*r1vec.')*V1)/muC - r1vec/r1;
758
759     % eccentricity
760     e = sqrt(evec*evec.');
761     % apses
762     pericenter = a*(1-e);
763     apocenter  = inf;                    % parabolic/hyperbolic case
764     if (e < 1), apocenter = a*(1+e); end % elliptic case
765
766     % since we have the eccentricity vector, we know exactly where the
767     % pericenter lies. Use this fact, and the given value of [dth], to
768     % cross-check if the trajectory goes past it
769     if (m > 0) % obvious case (always elliptical and both apses are traversed)
770         minimum_distance = pericenter;
771         maximum_distance = apocenter;
772     else % less obvious case
773         % compute theta1&2 ( use (AxB)-(CxD) = [U+FFFD][U+FFFD]- [U+FFFD][U+FFFD])
774         pm1 = sign( r1*r1*(evec*V1.') - (r1vec*evec.')*(r1vec*V1.') );
775         pm2 = sign( r2*r2*(evec*V2.') - (r2vec*evec.')*(r2vec*V2.') );
776         % make 100.4% sure it's in (-1 ≤ theta12 ≤ +1)
777         theta1 = pm1*acos( max(-1, min(1, (r1vec/r1)*(evec/e).')) );
778         theta2 = pm2*acos( max(-1, min(1, (r2vec/r2)*(evec/e).')) );
779         % points 1&2 are on opposite sides of the symmetry axis -- minimum
780         % and maximum distance depends both on the value of [dth], and both
781         % [theta1] and [theta2]
782         if (theta1*theta2 < 0)
783             % if |th1| + |th2| = turnangle, we know that the pericenter was
784             % passed
785             if abs(abs(theta1) + abs(theta2) - dth) < 5*eps(dth)
786                 minimum_distance = pericenter;
787             % this condition can only be false for elliptic cases, and
788             % when it is indeed false, we know that the orbit passed
789             % apocenter
790             else
791                 maximum_distance = apocenter;
792             end
793         % points 1&2 are on the same side of the symmetry axis. Only if the
794         % long-way was used are the min. and max. distances different from
795         % the min. and max. values of the radii (namely, equal to the apses)
796         elseif longway
797             minimum_distance = pericenter;
798             if (e < 1), maximum_distance = apocenter; end
799         end
800     end
801
802     % output argument
803     extremal_distances = [minimum_distance, maximum_distance];
```

```
804
805 end
```

Modelagem das patched conics a partir do problema de Lambert

```
1  %function value = funcao_custo(x)
2  %Funcao para calculo do custo total de transferencia interplanetaria
3  %   Considera-se 4 pontos principais, com 3 orbita coladas, portanto
4  %   Uso: x = [ theta_a, theta_b, theta_c, theta_d, t_ab, t_bc, t_cd ]
5  %   Ex.: x = [ pi/2 -pi/2 pi/2 -pi/2 15 270 15 ];
6
7  base = [1 0 0];              % Vetor de referencia da base canonica de R3
8  M = @(a)[ cos(a)  sin(a)  0;
9            -sin(a) cos(a)  0;
10           0       0       1
11         ];                   % Matriz de rotacao
12 dV = zeros(4,1);                  % Custo total
13
14 % Carrega dados de referencia
15 dados;
16
17 % Em relacao a Terra
18 r_a_terra = r_ta * base*M(x(1));  % Definida em projeto
19 r_b_terra = r_tb * base*M(x(2));  % SOI da Terra
20
21 % Em relacao a Marte
22 r_c_marte = r_mc * base*M(x(3));  % SOI de Marte
23 r_d_marte = r_md * base*M(x(4));  % Definida em projeto
24
25 % Posicoes dos planetas
26 r_terra_sol = r_st * [0 -1 0];
27 r_marte_sol = r_sm * [0 1 0];
28
29 % Tempos de transferencia
30 t_ab = x(5);
31 t_bc = x(6);
32 t_cd = x(7);
33
34 % Transferencia Terra - SOI(Terra)
35 % Em torno da Terra
36 r1 = r_a_terra;
37 r2 = r_b_terra;
38 t_voo = t_ab;
39 GM = mi_terra;
40 [V1, V2, extremal_distances, exitflag] = lambert(r1, r2, t_voo, 0, GM);
41 v_a_1 = (mi_terra/norm(r1))^(0.5)*[-sin(x(1)) cos(x(1)) 0];
42 v_a_2 = V1;
43 v_b_1 = V2 + (mi_sol/norm(r_terra_sol))^(0.5)*[1 0 0];  % V em rel ao Sol
44 if (norm(v_a_2 - v_a_1) <= norm(-v_a_2 - v_a_1))
45     dV(1) = norm(v_a_2 - v_a_1);
46 else
47     dV(1) = norm(-v_a_2 - v_a_1);
48 end
```

```
49
50 % Transferencia SOI(Terra)-SOI(Marte)
51 % Em torno do Sol
52 r_b_sol = r_b_terra + r_terra_sol;
53 r_c_sol = r_c_marte + r_marte_sol;
54
55 r1 = r_b_sol;
56 r2 = r_c_sol;
57 t_voo = t_bc;
58 GM = mi_sol;
59 [V1, V2, extremal_distances, exitflag] = lambert(r1, r2, t_voo, 0, GM);
60 v_b_2 = V1;
61 v_c_1 = V2 - (mi_sol/norm(r_marte_sol))^(0.5)*[-1 0 0]; % V em rel a Marte
62 ΔV(2) = norm(v_b_2 - v_b_1);
63
64 % Transferencia SOI(Marte)-Marte
65 % Em torno de Marte
66 r1 = r_c_marte;
67 r2 = r_d_marte;
68 t_voo = t_cd;
69 GM = mi_marte;
70 [V1, V2, extremal_distances, exitflag] = lambert(r1, r2, t_voo, 0, GM);
71 v_c_2 = V1;
72 v_d_1 = V2;
73 ΔV(3) = norm(v_c_2 - v_c_1);
74
75 % Orbita estacionamento Marte
76 v_d_2 = (mi_marte/norm(r2))^(0.5)*[-sin(x(4)) cos(x(4)) 0];
77 if (norm(v_d_2 - v_d_1) ≤ norm(-v_d_2 - v_d_1))
78     ΔV(4) = norm(v_d_2 - v_d_1);
79 else
80     ΔV(4) = norm(-v_d_2 - v_d_1);
81 end
82
83 ΔV;
84 value = sum(ΔV);
85
86 if (isnan(value))
87     value = Inf;
88 end
```

Algoritmo de otimização

```
1 %function pso()
2 %   pso busca o minimo
3
4 best_global = zeros(1,7);
5 iteration = 0;
6 max_iteration = 100;
7
8 % Hyperparams struct
9 hyperparams.num_particles = 10^2;
10 hyperparams.lb = [0 0 0 0 0 0 0];
```

```
hyperparams.ub = [2*pi 2*pi 2*pi 2*pi 30 365 30];
hyperparams.w = 0.9;
hyperparams.phip = 0.6;
hyperparams.phig = 0.8;

%function initialize_particles(num_particles, lb, ub):
for i = 1:hyperparams.num_particles
    particles(i) = Particle;
    % random_uniform here operates on arrays
    particles(i).x = random_uniform(hyperparams.lb, hyperparams.ub);
    Δ = hyperparams.ub - hyperparams.lb;
    particles(i).v = random_uniform(-Δ, Δ);
    particles(i).best = zeros(1,length(particles(i).x));
end

while ¬check_stopping_condition(iteration, max_iteration)
    iteration = iteration + 1;
    %[particles, best_iteration] = update_particles(particles,...
    %    best_global, hyperparams);

    %function update_particles(particles, best_global, hyperparams):
    w = hyperparams.w;
    phip = hyperparams.phip;
    phig = hyperparams.phig;
    best_iteration = zeros(1,7);

    for i = 1:length(particles)
        rp = random_uniform(0.0, 1.0);
        rg = random_uniform(0.0, 1.0);
        particles(i).v = w * particles(i).v + phip * rp * ...
            (particles(i).best - particles(i).x) + phig * rg * (best_global ...
            - particles(i).x);
        particles(i).x = particles(i).x + particles(i).v;
        particles(i).x = min(max(particles(i).x, hyperparams.lb), ...
            hyperparams.ub);
        if custo(particles(i).x) < custo(particles(i).best)
            particles(i).best = particles(i).x;
            if custo(particles(i).x) < custo(best_iteration)
                best_iteration = particles(i).x;
            end
        end
    end

    if custo(best_iteration) < custo(best_global)
        best_global = best_iteration;
    end
end

best_global
custo(best_global)

% Create a table with the data and variable names
if exist('T','var') == 1
    T = [T; table(custo(best_global), best_global(1), best_global(2), ...
        best_global(3), best_global(4), best_global(5), best_global(6), ...
        best_global(7), hyperparams.num_particles, max_iteration, ...
```

```
            'VariableNames', { 'Custo_total', 'theta_a', 'theta_b', 'theta_c', ...
            'theta_d', 't_ab', 't_bc', 't_cd', 'Num_particles', ...
            'Max_iterations'} )];
elseif exist('resultados.txt', 'file') == 2
    T = readtable('resultados.txt');
else
    T = table(custo(best_global), best_global(1), best_global(2), ...
        best_global(3), best_global(4), best_global(5), best_global(6), ...
        best_global(7), hyperparams.num_particles, max_iteration, ...
        'VariableNames', { 'Custo_total', 'theta_a', 'theta_b', 'theta_c', ...
        'theta_d', 't_ab', 't_bc', 't_cd', 'Num_particles', ...
        'Max_iterations'} );
end
% Write data to text file
writetable(T, 'resultados.txt')
```

Obtenção dos parâmetros orbitais a partir de $\vec{r}$ e $\vec{v}$

```

function conica = orbita_from_rv(r, v, mi)

energia = norm(v)^2/2 - mi/norm(r);
h = cross(r,v);
theta = angular_coord(r);

a = -mi/(2*energia);
p = norm(h)^2/mi;
e = (1-p/a)^(0.5);
phi = theta - acos((1-a*(1-e^2)/norm(r))/e);

conica = @(nu) p./(1-e*cos(nu-phi));
er = @(nu) [cos(nu) sin(nu) 0];

%while (phi < 0)
%    phi = phi + pi;
%end

if norm(conica(theta)*er(theta)-r)/norm(r) > 10^-2
    phi = phi+pi;
    conica = @(nu) p./(1-e*cos(nu-phi));
end

phi = mod(phi, 2*pi);

end
```

Geração de material gráfico

```
% Vetor de estado que caracteriza nossa transferencia de interesse
```

```
x = [ 2.6147    6.2832    3.0095    0.8353 3.09 261.96 2.28];

funcao_custo;

UA = 1.496e8;

% Orbita AB: Fuga da Terra
[x,y] = definir_orbita_cartesiana(r_a_terra, v_a_2, mi_terra);
figure;
legenda = string();
legenda(end+1) = plotar([0,0], "Terra");
legenda(end+1) = plotar(r_a_terra/R_t, "A");
legenda(end+1) = plotar(r_b_terra/R_t, "B");
legenda(end+1) = plotar([x' y']/R_t, "Trajetoria", '-');
grid on;
axis equal
legend(legenda(2:end));
xlabel('x [raios terrestres]');
ylabel('y [raios terrestres]');

% Orbita BC: Transferencia entre esferas de influencia
[x,y] = definir_orbita_cartesiana(r_b_sol, v_b_2, mi_sol);
figure;
legenda = string();
legenda(end+1) = plotar([0,0], "Sol");
legenda(end+1) = plotar(r_terra_sol/UA, "Terra");
legenda(end+1) = plotar(r_b_sol/UA, "B");
legenda(end+1) = plotar(r_marte_sol/UA, "Marte");
legenda(end+1) = plotar(r_c_sol/UA, "C");
legenda(end+1) = plotar([x' y']/UA, "Trajetoria", '-');
grid on;
axis equal
legend(legenda(2:end));
xlabel('x [UA]');
ylabel('y [UA]');

% Orbita CD: Captura gravitacional
[x,y] = definir_orbita_cartesiana(r_c_marte, v_c_2, mi_marte);
figure;
legenda = string();
legenda(end+1) = plotar([0,0], "Marte");
legenda(end+1) = plotar(r_c_marte/R_m, "C");
%legenda(end+1) = plotar(r_d_marte/R_m, "D");
legenda(end+1) = plotar([r_d_marte(1), -r_d_marte(2)]/R_m, "D");
legenda(end+1) = plotar([x' y']/R_m, "Trajetoria", '-');
grid on;
axis equal
legend(legenda(2:end));
xlabel('x [raios marcianos]');
ylabel('y [raios marcianos]');


function [x,y] = definir_orbita_cartesiana(r_,v_,mi_)

r = orbita_from_rv(r_, v_, mi_);
theta = pi/180:pi/180:2*pi;
```

```
58  raio = r(theta);
59  x = raio.*cos(theta);
60  y = raio.*sin(theta);
61
62  end
```

Referências

[1] SPHERE of influence (astrodynamics). In: WIKIPÉDIA: the free encyclopedia. Wikimedia, 2019. Disponível *aqui*. Acesso em: 16 jun. 2019.

[2] OLDENHUIS, R. *Robust solver for Lambert's orbital-boundary value problem*. In: MathWorks: Accelerating the pace of engineering and science. The MathWorks, Inc., 2019. Disponível *aqui*. Acesso em: 11 jun. 2019.

[3] CASTRO, F.M.M.; SILVA, F.C; SILVA, M.C *Transferˆencia Terra-Marte utilizando a metodologia de patched-conics*. Trabalho da disciplina MVO-41 - Mecânica Orbital. Departamento de Engenharia Aeronáutica e Aeroespacial. Instituto Tecnológico de Aeronáutica, 2019.

[4] SHI, Y.; EBERHART, R.C. *A modified particle swarm optimizer*. Proceedings of IEEE International Conference on Evolutionary Computation. pp. 69–73. 1998.

[5] PARTICLE swarm optimization. In: Wikipedia, the free encyclopedia. Wikimedia, 2019. Disponível *aqui*. Acesso em: 20 jun. 2019.

[6] DELTA-V budget. In: WIKIPÉDIA: the free encyclopedia. Wikimedia, 2019. Disponível *aqui*. Acesso em: 20 jun. 2019.

[7] MÁXIMO, M. *Métodos de Otimização Baseados em População*. Apresentação de slides da disciplina de Inteligência Artificial para Robótica Móvel. Departamento de Engenharia da Computação. Instituto Tecnológico de Aeronáutica, São José dos Campos-SP, 2019.

[8] PRUSSING, J.; E. CONWAY, B. A. *Orbital Mechanics*. 2nd Edition, pág. 68-71. Oxford University Press, 2012.

[9] CURTIS, H. D. *Orbital Mechanics: For Engineering Students*. Elsevier Butterworth-Heinemann. Oxford, 2005.