UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

LUCAS FERNANDO COLLOCA

Desenvolvimento de software para análise da dinâmica longitudinal de aeronaves

SÃO CARLOS
2019

LUCAS FERNANDO COLLOCA

Desenvolvimento de software para análise da dinâmica longitudinal de aeronaves

Monografia apresentada ao Curso de Engenharia Aeronáutica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Aeronáutico.

Orientador: Prof. Dr. Jorge Henrique Bidinotto

VERSÃO CORRIGIDA

São Carlos
2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

C714d

Colloca, Lucas Fernando
Desenvolvimento de software para análise da dinâmica longitudinal de aeronaves / Lucas Fernando Colloca; orientador Jorge Henrique Bidinotto; coorientador Ricardo Afonso Angélico. São Carlos, 2019.

Monografia (Graduação em Engenharia Aeronáutica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. Python. 2. Longitudinal. 3. Matricial. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Candidato: Lucas Fernando Colloca
Título do TCC: *Desenvolvimento de software para análise da dinâmica longitudinal de aeronaves*
Data de defesa: 05/11/2019

Comissão Julgadora	**Resultado**
Professor Doutor Ricardo Afonso Angélico	Aprovado
Instituição: EESC - SAA	
Professor Titular Glauco Augusto de Paula Caurin	Aprovado
Instituição: EESC - SAA	

Presidente da Banca: Professor Doutor Ricardo Afonso Angélico

(assinatura)

RESUMO

COLLOCA, L. F. Desenvolvimento de software para análise da dinâmica longitudinal de aeronaves. 2019. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

Esse trabalho visa implementar leis de controle do movimento longitudinal de aeronaves, utilizando a linguagem de programação *Python* versão 3.7.2. O projeto consiste de um modelo matemático, no espaço de estados, que descreve as leis de controle em forma matricial, cujas equações estão relacionadas por meio de relações diferenciais de primeira ordem. Por fim, constrói-se um código computacional para implementar as equações que descrevem o movimento longitudinal de aeronaves.

Palavras-chave: 1. Python. 2. Longitudinal. 3. Matricial.

Glossário

Arquivo .exe: um arquivo executável usado em sistemas operacionais para abrir programas de software.

Curto período: oscilação amortecida em torno do eixo lateral y.

Fugóide: movimento oscilatório do C.G. da aeronave em torno de uma linha de referência.

numpy: biblioteca Python para operações matemáticas.

Pitch: rotação em torno do eixo lateral y.

Prompt de Comando: um interpretador de linha de comando.

Python: linguagem de programação.

Roll: rotação em torno do eixo longitudinal x.

Script: um conjunto de instruções para que uma função seja executada.

tkinter: biblioteca Python para construção de interfaces gráficas do usuário.

Yaw: rotação em torno do eixo normal z.

Símbolos

A matriz de estado

B matriz de *input*

C matriz constante

D matriz constante

g aceleração da gravidade local

h altitude da aeronave

i número imaginário

I matriz identidade

I_y momento de inércia no eixo lateral y

m massa da aeronave

q velocidade angular

t instante de tempo

$u(t)$ matriz de entrada

u_k ganho da entrada degrau

U_e velocidade linear no eixo longitudinal x

V matriz de autovetores

V_0 velocidade inicial da aeronave

W_e velocidade linear no eixo normal z

$x(t)$ matriz de estados

$y(t)$ matriz de saída

Δu variação de velocidade linear U_e

Δw variação de velocidade linear W_e

ζ taxa de amortecimento

η efeito do profundor

θ ângulo de Pitch

θ_e ângulo Pitch de equilíbrio

λ autovalor

Λ matriz de autovalores

v autovetor

τ efeito da propulsão

ω frequência natural não amortecida

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

O emprego de métodos computacionais vem se mostrando crescente na indústria, sendo que a tendência é que se perpetue, levando em conta o ganho de eficiência no que se refere à otimização de tempo e minimização de erros. Nesse contexto, *Python* é uma linguagem de programação de alto nível e *open source*, cujo uso tem aumentado consideravelmente no âmbito da engenharia.

Dessa maneira, é desenvolvido um *software* em *Python* voltado para usuários que desejam fazer uma análise da dinâmica de movimento longitudinal de aeronaves. O *software* fornece, em gráficos, 5 resultados:

- Variação de velocidade longitudinal U_e;
- Variação de velocidade normal W_e;
- Velocidade angular q;
- Ângulo de pitch θ;
- Altitude h.

Além dos resultados acima, são dados de saída do *software*, tanto para os movimentos de curto período como o de fugóide, a frequência natural não amortecida ω e a taxa de amortecimento ζ.

2 Modelo Matemático de Movimento Longitudinal

O objetivo desta seção é trazer uma explicação das equações utilizadas para descrever o movimento longitudinal de aeronaves. Como referência, foram utilizados os métodos registrados na bibliografia [1].

Inicia-se com o detalhamento do sistema de referência utilizado.

2.1 Sistema de Referência

O sistema de referência é ilustrado na figura 1, sendo que a convenção de sinais segue o sentido dos eixos lineares (x, y e z) e de rotação ($Roll$, $Pitch$ e Yaw):

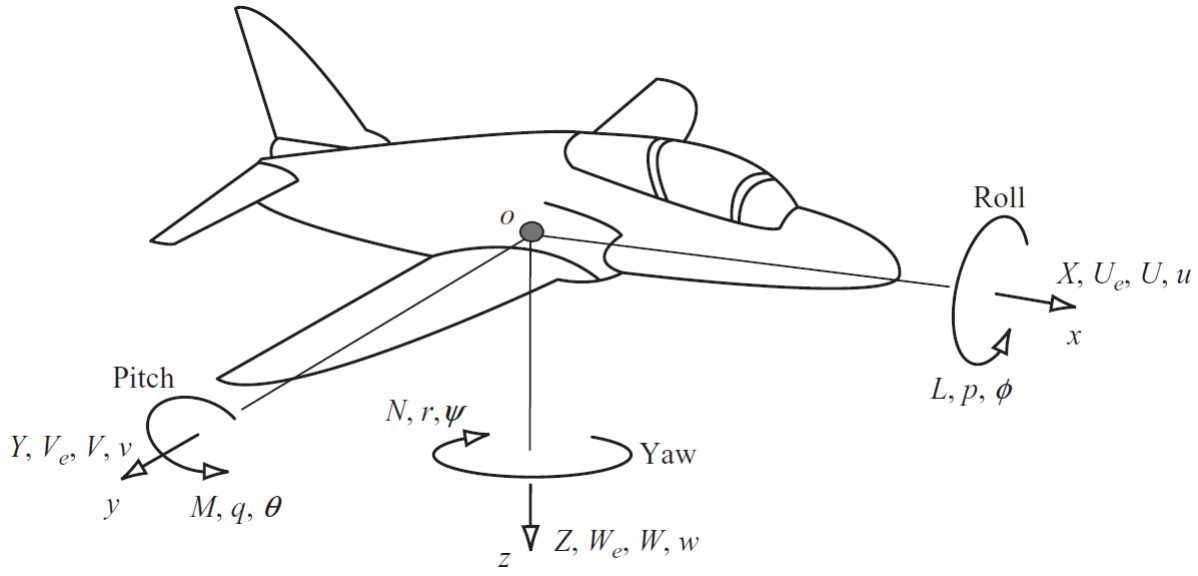

Figura 1: Sistema de Referência. Fonte: Bibliografia [1]

2.2 Equações

As equações em espaço de estados, que descrevem um sistema linear e dinâmico, são dadas abaixo:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1}$$

$$y(t) = Cx(t) + Du(t) \tag{2}$$

As matrizes $x(t)$, $u(t)$ e $y(t)$ são as matrizes de estados, de entrada e de saída, respectivamente.

Já as matrizes A, B, C e D são matrizes de constantes, características de cada aeronave. A seguir são detalhadas essas matrizes.

2.2.1 *Matriz de Estado A*

A matriz A possui dimensão 4x4 e é dada em termos das derivadas de estabilidade longitudinal, que são características de cada aeronave, na forma concisa:

$$A = \begin{bmatrix} x_u & x_w & x_q & x_\theta \\ z_u & z_w & z_q & z_\theta \\ m_u & m_w & m_q & m_\theta \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

A relação entre as derivadas longitudinais nas formas concisa e dimensional é mostrada na tabela 1.

Forma Concisa	Forma Dimensional	Forma Concisa	Forma Dimensional
x_u	$\frac{X_u}{m} + \frac{X_{\dot{w}} Z_u}{m(m - Z_{\dot{w}})}$	x_q	$\frac{X_q - mW_e}{m} + \frac{(Z_q + mU_e)X_{\dot{w}}}{m(m - Z_{\dot{w}})}$
z_u	$\frac{Z_u}{m - Z_{\dot{w}}}$	z_q	$\frac{Z_q + mU_e}{m - Z_{\dot{w}}}$
m_u	$\frac{M_u}{I_y} + \frac{Z_u M_{\dot{w}}}{I_y(m - Z_{\dot{w}})}$	m_q	$\frac{M_q}{I_y} + \frac{(Z_q + mU\,e)\dot{M}_{\dot{w}}}{I_y(m - Z_{\dot{w}})}$
x_w	$\frac{X_w}{m} + \frac{X_{\dot{w}} Z_w}{m(m - Z_{\dot{w}})}$	x_θ	$-g\cos(\theta_e) - \frac{X_{\dot{w}} g \sin(\theta_e)}{m - Z_{\dot{w}}}$
z_w	$\frac{Z_w}{m - Z_{\dot{w}}}$	z_θ	$\frac{-mg\sin(\theta_e)}{m - Z_{\dot{w}}}$
m_w	$\frac{M_w}{I_y} + \frac{Z_w M_{\dot{w}}}{I_y(m - Z_{\dot{w}})}$	m_θ	$\frac{-M_{\dot{w}} mg\sin(\theta_e)}{I_y(m - Z_{\dot{w}})}$

Tabela 1: Relação entre Derivadas Longitudinais - Matriz A. Fonte: Bibliografia [1]

Dessa posse da tabela 1, considerando o ângulo pitch θ_e de equilíbrio (figura 2) e pequenas perturbações γ_e, tem-se:

- a velocidade linear U_e, dada por $U_e = V_0\cos(\theta_e)$, onde V_0 é a velocidade da aeronave;

- A velocidade linear W_e, dada por $W_e = V_0\sin(\theta_e)$.

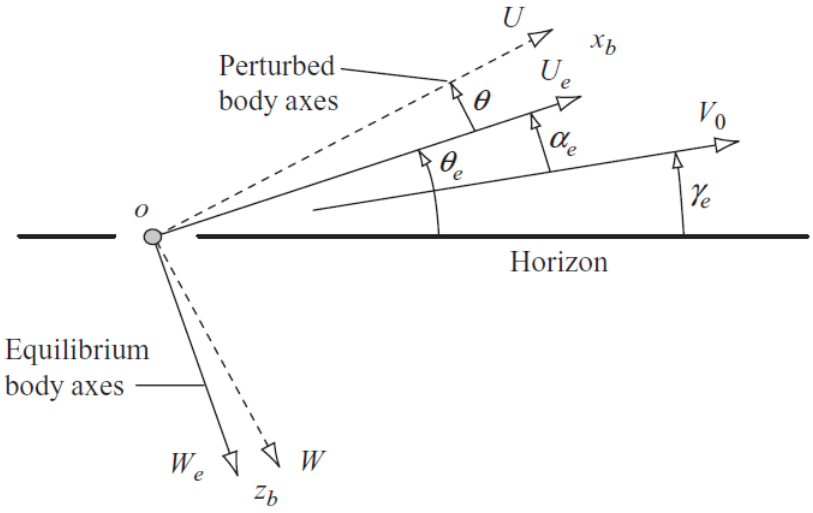

Figura 2: Ângulo Pitch θ_e de Equilíbrio. Fonte: Bibliografia [1]

As derivadas longitudinais na forma dimensional serão os *inputs* fornecidos pelo usuário, no *software*.

2.2.2 *Matriz de input B*

A matriz *B* possui dimensão 4x2, originalmente:

$$B = \begin{bmatrix} x_\eta & x_\tau \\ z_\eta & z_\tau \\ m_\eta & m_\tau \\ 0 & 0 \end{bmatrix}$$

Nesse projeto, despreza-se os termos x_τ, z_τ e m_τ, relacionados aos efeitos de propulsão. Assim, a matriz *B* resulta em uma matriz 4x1, tal que:

$$B = \begin{bmatrix} x_\eta \\ z_\eta \\ m_\eta \\ 0 \end{bmatrix}$$

Analogamente à matriz *A*, a relação entre as derivadas longitudinais nas formas concisa e dimensional é mostrada na tabela 2.

Forma Concisa	Forma Dimensional
x_η	$\frac{X_\eta}{m} + \frac{X_{\dot{w}} Z_\eta}{m(m - Z_{\dot{w}})}$
z_η	$\frac{Z_\eta}{m - Z_{\dot{w}}}$
m_η	$\frac{M_\eta}{I_y} + \frac{M_{\dot{w}} Z_\eta}{I_y(m - Z_{\dot{w}})}$

Tabela 2: Relação entre Derivadas Longitudinais - Matriz B. Fonte: Bibliografia [1]

2.2.3 Matrizes C e D

Para o caso do presente trabalho, as matrizes C e D são dadas por:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

2.3 Autovalores e Autovetores

O cálculo dos autovalores e autovetores é necessário para a obtenção das características do movimento longitudinal de uma aeronave. Tais valores vêm da equação 3, característica da matriz de estado A:

$$\Delta(\lambda) = |\lambda I - A| \tag{3}$$

Onde λ são os autovalores e I é a matriz identidade.

2.3.1 Cálculo dos autovalores

Obtém-se os autovalores de A zerando-se a equação característica 3:

$$|\lambda I - A| = 0 \tag{4}$$

As soluções λ_i da equação 4 são os autovalores de A. Para o caso do projeto, como a matriz A possui dimensão 4x4, obtém-se 4 autovalores complexos conjugados 2 a 2:

$$\lambda_1 = a + bi \tag{5}$$
$$\lambda_2 = a - bi \tag{6}$$
$$\lambda_3 = c + di \tag{7}$$
$$\lambda_4 = c - di \tag{8}$$

Com o objetivo de efetuar as operações em forma matricial, cria-se a matriz de autovalores Λ de dimensão 4x4:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix}$$

2.3.2 Cálculo dos autovetores

Os autovetores v_i (v_1, v_2, v_3, v_4), de dimensão 4x1, são correspondentes de cada autovalor λ_i, satisfazendo a equação 9:

$$Av_i = \lambda_i v_i \tag{9}$$

Dessa forma, tem-se que:

$$[\lambda_i I - A]v_i = 0 \tag{10}$$

Os autovetores v_i são linearmente independentes.

A matriz de autovetores V, de dimensão 4x4, é formada pelas transpostas dos autovetores v_i:

$$V = \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ v_4^T \end{bmatrix}$$

Para o cálculo dos autovetores v_i, considerando as matriz Λ de autovalores e V de autovetores, tem-se a equação matricial 11, oriunda da equação 9:

$$AV = V\Lambda \tag{11}$$

O que resulta em uma relação de similaridade:

$$V^{-1}AV = \Lambda \tag{12}$$

Assim, as matrizes A e Λ são similares. Matrizes similares possuem a propriedade de seus autovalores serem iguais. Assim sendo, os autovalores de A são iguais aos de Λ.

Conforme a bibliografia [I], os autovetores v_i são determinados pela relação 13 abaixo, em função da matriz adjunta Adj:

$$v_i = Adj[\lambda_i I - A] \tag{13}$$

2.4 Frequência Natural Não Amortecida e Taxa de Amortecimento

O comportamento dinâmico do movimento longitudinal de uma aeronave pode ser caracterizado em termos de vários parâmetros, dentre eles:

- Frequência Natural Não Amortecida, calculada tanto para o movimento de curto período (ω_{sp}) como para o de fugóide (ω_{ph});

- Taxa de Amortecimento, calculada tanto para o movimento de curto período (ζ_{sp}) como para o de fugóide (ζ_{ph}).

Para o caso de um sistema dinâmico subamortecido, a taxa de amortecimento ζ satisfaz a condição $0 < \zeta < 1$. Nessa situação, os autovalores do sistema serão complexos conjugados 2 a 2, conforme a subseção 2.3.1.

Nas subseções 2.4.1 e 2.4.2 a seguir, faz-se o estudo do cálculo desses dois parâmetros, tanto para o movimento de curto período como para o de fugóide, respectivamente.

2.4.1 Movimento de Curto Período

Nesse caso, para um período mínimo, tem-se a máxima frequência. Portanto, a frequência natural não amortecida ω_{sp} é dada pelo máximo valor dentre os módulos das partes imaginárias dos autovalores:

$$\omega_{sp} = max(|Im(\lambda_1)|, |Im(\lambda_2)|, |Im(\lambda_3)|, |Im(\lambda_4)|) \tag{14}$$

Para os valores de ω_{sp} e λ_i que satisfazem a relação 14, a taxa de amortecimento ζ_{sp} é dada em função da parte real de λ_i e de ω_{sp}:

$$\zeta_{sp} = \frac{|Real(\lambda_i)|}{\omega_{sp}} \tag{15}$$

2.4.2 Movimento de Fugóide

Nesse caso, tem-se a mínima frequência. Portanto, a frequência natural não amortecida ω_{ph} é dada pelo mínimo valor dentre os módulos das partes imaginárias dos autovalores:

$$\omega_{ph} = min(|Im(\lambda_1)|, |Im(\lambda_2)|, |Im(\lambda_3)|, |Im(\lambda_4)|) \tag{16}$$

Para os valores de ω_{ph} e λ_i que satisfazem a relação 16, a taxa de amortecimento ζ_{ph} é dada em função da parte real de λ_i e de ω_{ph}:

$$\zeta_{ph} = \frac{|Real(\lambda_i)|}{\omega_{ph}} \tag{17}$$

São fornecidas aos usuários do *software* duas opções de entradas no profundor da aeronave:

- Entrada degrau de magnitude a: uma função que possui valor constante e igual a a para instantes de tempo $t \geq 0$ e valor nulo para $t < 0$;

- Entrada impulso unitário: uma função em forma de pulso retangular de área unitária. No limite, a base do retângulo é um intervalo de tempo infinitesimal, que tende a zero, e sua altura tende ao infinito.

Os métodos matemáticos utilizados para cada uma das entradas acima são detalhados nas subseções 2.5 e 2.6.

2.5 Entrada Degrau

Conforme a bibliografia [I], para uma entrada degrau de ganho constante (em graus), denotada por u_k, aplicada no instante $t_0 = 0$, a resposta da aeronave $y(t)$ a essa entrada é dada por:

$$y(t) = CVe^{\Lambda t}[V^{-1}x(0) + \Lambda^{-1}V^{-1}Bu_k] - [CA^{-1}B - D]u_k \tag{18}$$

A matriz exponencial $e^{\Lambda t}$ é formada pelos autovalores λ_i da matriz A:

$$e^{\Lambda t} = \begin{bmatrix} e^{\lambda_1 t} & 0 & 0 & 0 \\ 0 & e^{\lambda_2 t} & 0 & 0 \\ 0 & 0 & e^{\lambda_3 t} & 0 \\ 0 & 0 & 0 & e^{\lambda_4 t} \end{bmatrix}$$

Onde $e^{\lambda t} = e^{(a+bi)t} = e^{at}e^{(bi)t}$.

Para um número $e^{(bi)t}$, tem-se a relação:

$$e^{(bi)t} = cos(bt) + isin(bt)$$

E o número $e^{(a+bi)t}$ pode ser escrito como:

$$e^{(a+bi)t} = e^{at}[cos(bt) + isin(bt)] \tag{19}$$

Portanto, a matriz $e^{\Lambda t}$ será implementada em *Python* de acordo com os valores de autovalores λ_i a seguir:

$$e^{\lambda_1 t} = e^{at}[cos(bt) + isin(bt)] \tag{20}$$

$$e^{\lambda_2 t} = e^{at}[cos(bt) - isin(bt)] \tag{21}$$

$$e^{\lambda_3 t} = e^{ct}[cos(dt) + isin(dt)] \tag{22}$$

$$e^{\lambda_4 t} = e^{ct}[cos(dt) - isin(dt)] \tag{23}$$

Considerando condições iniciais nulas ($x(0) = 0$) e as já definidas matrizes C e D da seção 2.2.3, a resposta à entrada degrau resulta em:

$$y(t) = Ve^{\Lambda t}[\Lambda^{-1}V^{-1}Bu_k] - [A^{-1}B]u_k \tag{24}$$

A matriz de saída $y(t)$ é uma matriz de dimensão 4x1. Cada linha da mesma fornece 4 respostas, que ajudam a descrever o movimento longitudinal de uma aeronave:

$$y(t) = \begin{bmatrix} \Delta u(t) \\ \Delta w(t) \\ q(t) \\ \theta(t) \end{bmatrix}$$

Assim, por exemplo, caso deseja-se obter o ângulo de pitch θ em um determinado instante de tempo t, toma-se a 4ª linha da matriz $y(t)$.

2.6 Entrada Impulso Unitário

Conforme a bibliografia [I], a resposta longitudinal a uma entrada do tipo impulso unitário, $y(t)$, é dada pela expressão 25:

$$y(t) = CVe^{\Lambda t}V^{-1}[x(0) + B] \tag{25}$$

Considerando condições iniciais nulas ($x(0) = 0$) e a já definida matriz C da seção 2.2.3, a resposta à entrada impulso unitário resulta em:

$$y(t) = Ve^{\Lambda t}V^{-1}B \tag{26}$$

Analogamente ao caso da entrada degrau, a matriz $y(t)$ é uma matriz de dimensão 4x1 que fornece as 4 saídas:

$$y(t) = \begin{bmatrix} \Delta u(t) \\ \Delta w(t) \\ q(t) \\ \theta(t) \end{bmatrix}$$

2.7 Acréscimo da Resposta de Altitude

Da bibliografia [I], para pequenas perturbações, a variação de altitude $\dot{h}$ no tempo é dada pela expressão 27, sendo que foi considerado o sistema de referência da figura 1. Portanto, soma-se Δw, ao invés de subtrair:

$$\dot{h} = U_e\theta - W_e + \Delta w \tag{27}$$

Na condição de pequenas perturbações, pode-se considerar $U_e \approx V_0$ e $W_e \approx 0$. Portanto, a relação final de $\dot{h}$ pode ser aproximada para a expressão 28:

$$\dot{h} = V_0\theta + \Delta w \tag{28}$$

Para encontrar o valor da altitude $h(t)$ em um instante de tempo t, de posse de $\dot{h}(t)$, desenvolve-se a expressão da derivada da altitude h:

$$\dot{h} = \frac{dh}{dt} \tag{29}$$

$$dh = \dot{h}dt \tag{30}$$

Para um intervalo de tempo infinitesimal Δt, faz-se a aproximação $dt \approx \Delta t$, tal que $dh \approx \Delta h$:

$$\Delta t = t_{i+1} - t_i \tag{31}$$

$$\Delta h = h_{i+1} - h_i \tag{32}$$

Das relações 30, 31 e 32:

$$h_{i+1} - h_i = \dot{h}(t_{i+1} - t_i) \tag{33}$$

$$h_{i+1} = h_i + \dot{h}(t)(t_{i+1} - t_i) \tag{34}$$

Assim, de 34 e 28, em um instante $t = t_{i+1}$ a altitude da aeronave $h(t) = h_{i+1}$ é dada por:

$$h_{i+1} = h_i + [V_0\theta(t_{i+1}) + \Delta w(t_{i+1})](t_{i+1} - t_i) \tag{35}$$

3 Código em *Python*

O procedimento seguido pelo código consiste de duas partes:

- *Script Controlador.py*: interface gráfica do usuário, onde os dados de entrada são selecionados;

- *Script Calculos.py*: implementação do modelo matemático construído na seção 2, a partir dos *inputs* fornecidos na primeira parte. As cinco variáveis (Δu, Δw, q, θ e h) e os parâmetros ω e ζ serão os *outputs* do *software*. Para maximizar a velocidade de processamento dos cálculos, utiliza-se a biblioteca *numpy*.

3.1 Interface Gráfica do Usuário

A interface gráfica foi programada em *Controlador.py* (anexo), usando a biblioteca *tkinter*. O resultado é mostrado na figura 3.

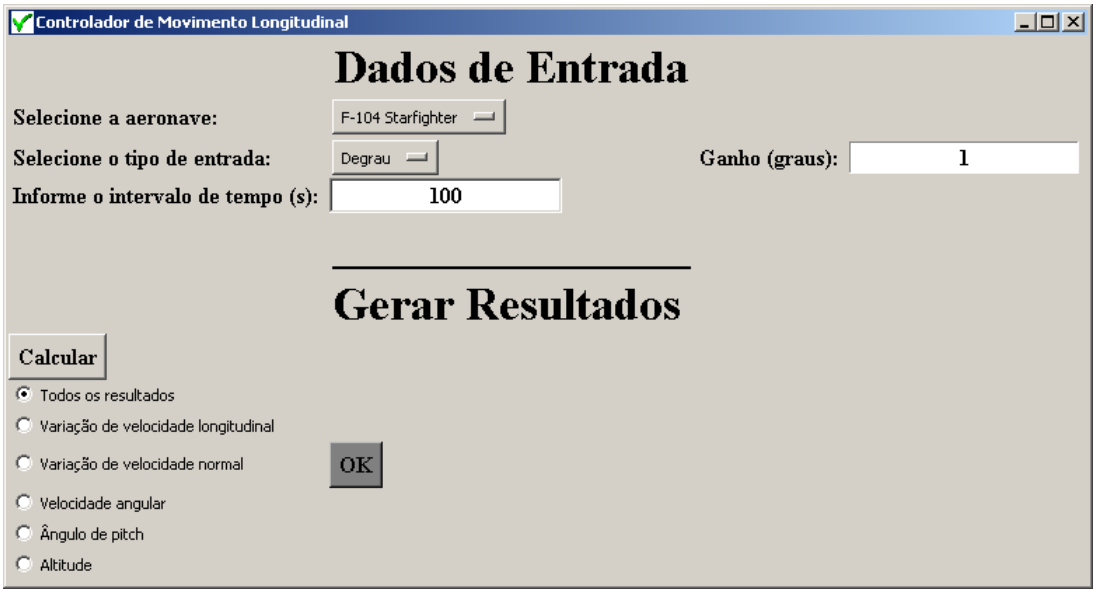

Figura 3: Interface Gráfica do Usuário. Fonte: Autor

Ao usuário, é permitido selecionar:

- a aeronave a ser analisada;

- o tipo de entrada (no caso da entrada degrau é permitido entrar com um ganho diferente de 1^o);

- o intervalo de tempo de análise (eixo das abscissas dos gráficos), em segundos;

- a opção de mostrar todos as respostas em uma única janela, ou gerar as respostas em janelas separadas.

3.1.1 Seleção da Aeronave

Para esse projeto foram colhidos os dados de 2 aeronaves, a saber:

- *F-104 Starfighter*: seus dados aerodinâmicos de estabilidade e controle (figura 5) foram retirados da bibliografia [1];

- *B747-100*: seus dados aerodinâmicos de estabilidade e controle (figura 6) foram retirados da bibliografia [2].

Os arquivos de texto estão salvos na pasta *Aeronaves*, conforme a figura 4. Esse endereço foi programado na segunda parte do código.

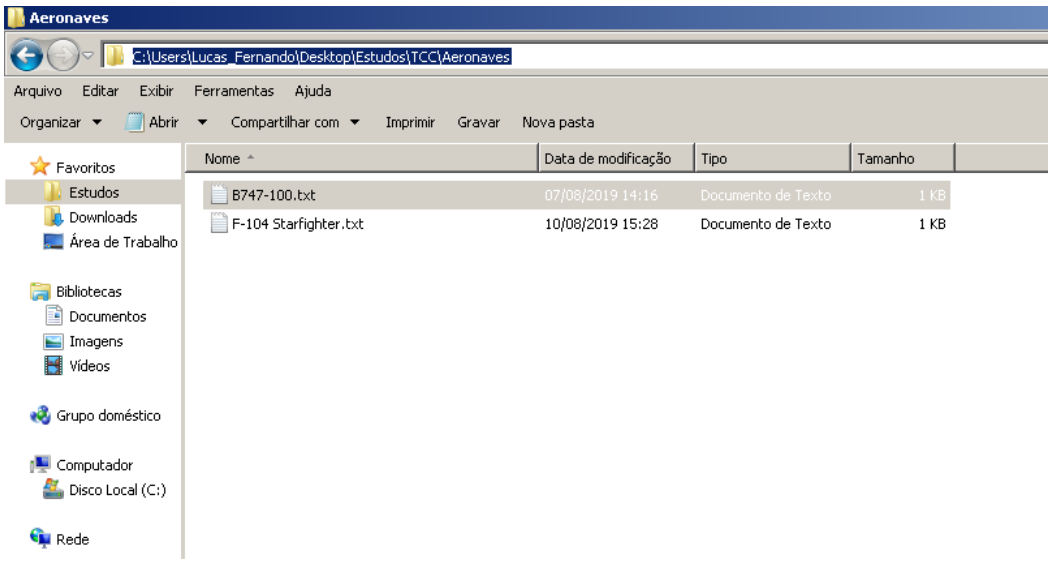

Figura 4: Local de Armazenamento dos Arquivos de Texto. Fonte: Autor

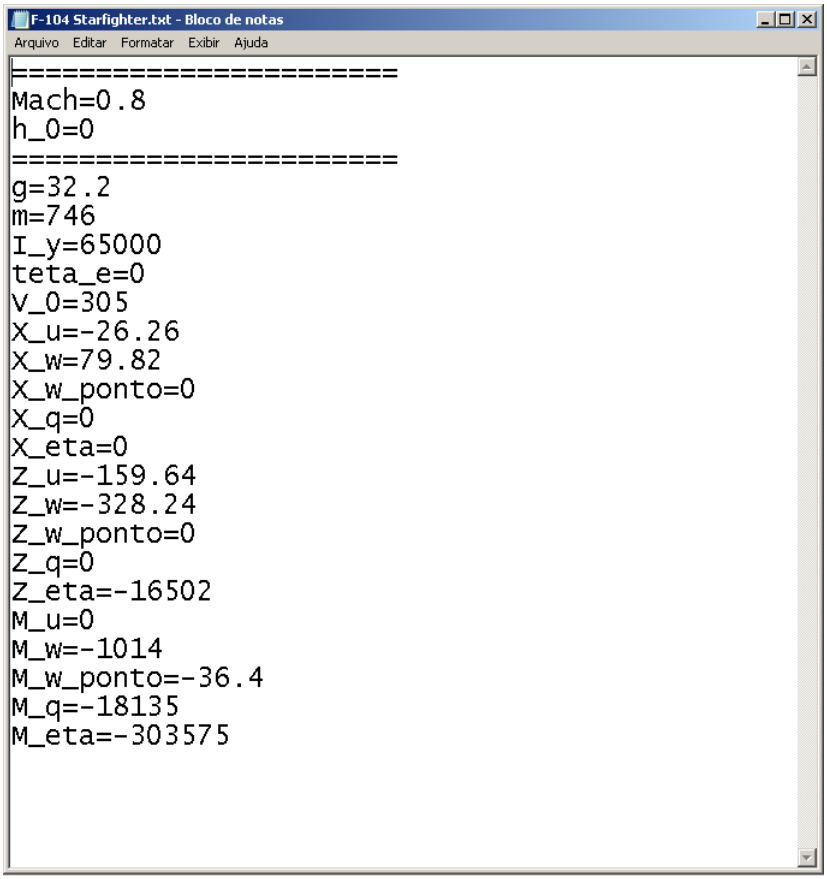

Figura 5: Arquivo de Texto com os Dados da Aeronave *F-104 Starfighter* (unidades no Sistema Inglês). Fonte: Autor

Nesse caso, para a condição de número de Mach = 0.8 e altitude h = 0, tem-se, em unidades no **Sistema Inglês**:

- as derivadas de estabilidade na **forma dimensional** (subseções 2.2.1 e 2.2.2, para as componentes Δu, Δw, q e η do profundor), por serem as fornecidas nas bibliografias [I] e [II];
- os demais *inputs* (g, m, I_y, θ_e e V_0).

A forma de preenchimento dos arquivos de texto deve seguir exatamente o modelo da figura 5, para que a segunda parte do código consiga interpretar esses dados.

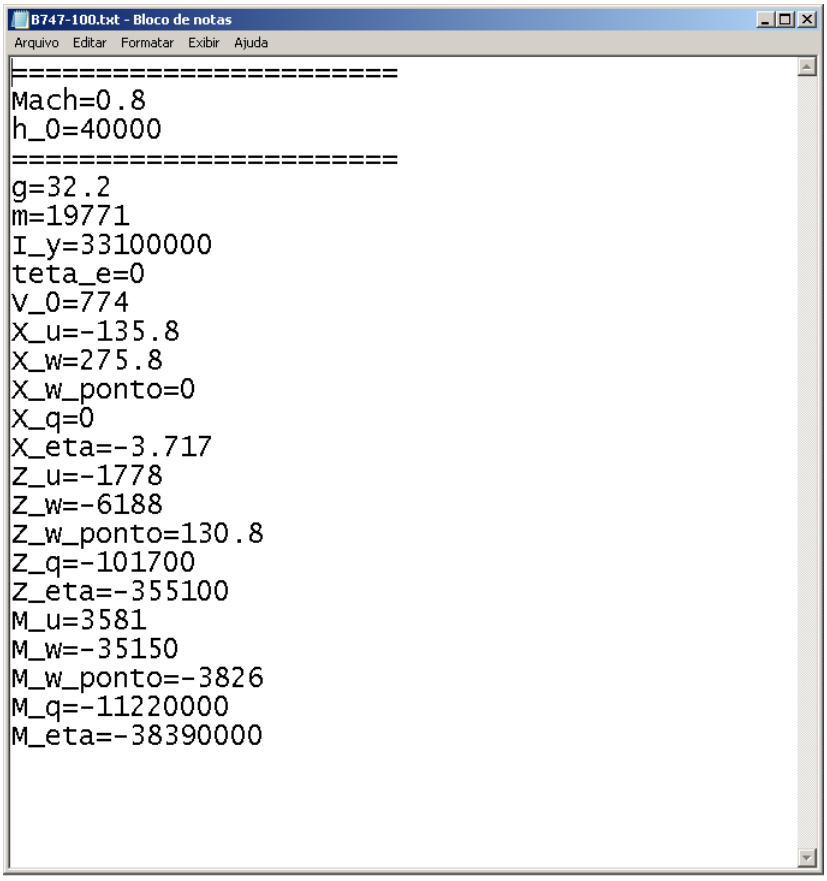

Figura 6: Arquivo de Texto com os Dados da Aeronave *B747-100* (unidades no Sistema Inglês). Fonte: Autor

A figura 7 mostra as aeronaves disponíveis no *software*.

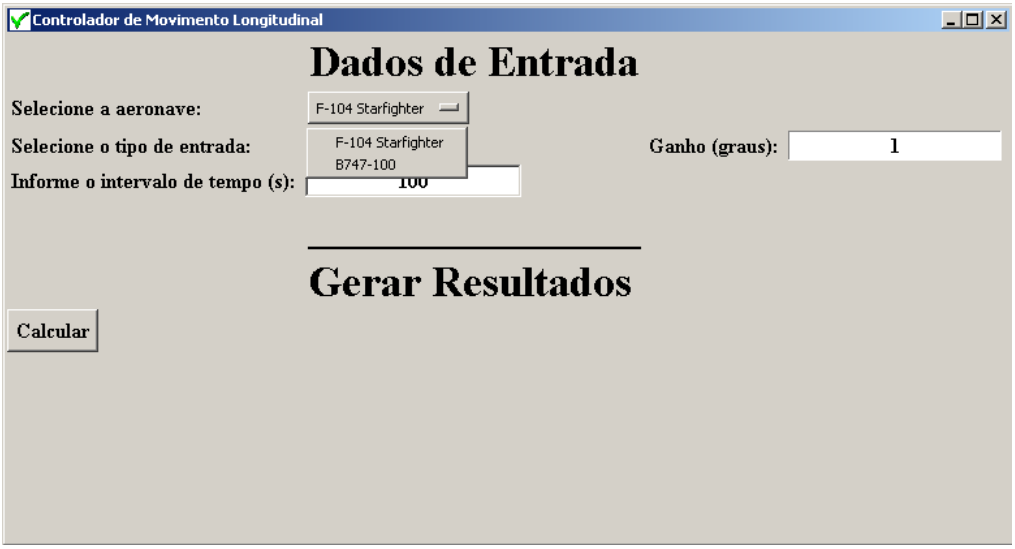

Figura 7: Aeronaves Disponíveis no *Software*. Fonte: Autor

3.1.2 *Seleção do Tipo de Entrada*

O usuário consegue optar por um dos dois tipos de entrada, *Degrau* ou *Impulso Unitário* (figura 8). Conforme a escolha, os métodos descritos nas seções 2.5 e 2.6, respectivamente, serão executados no *script* anexo *Calculos.py*.

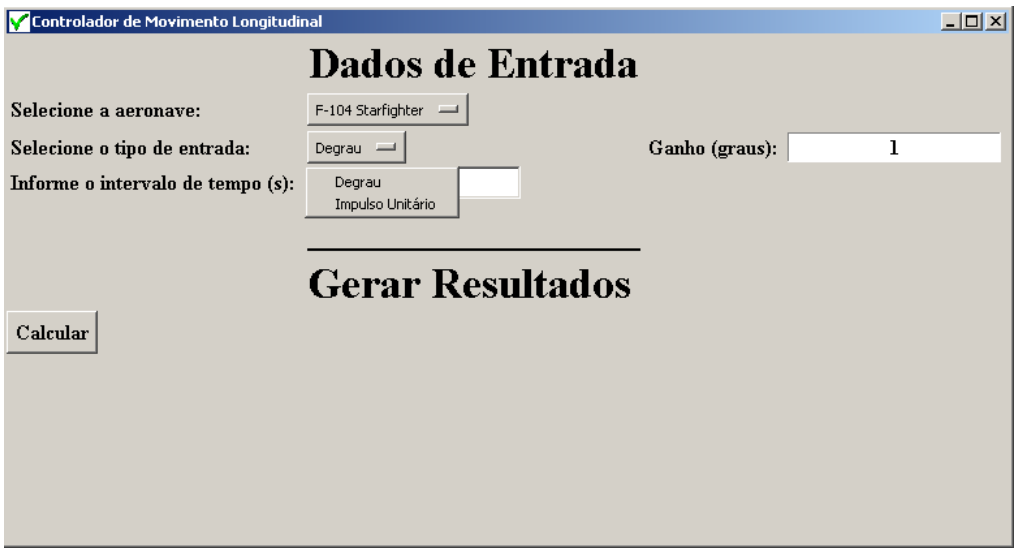

Figura 8: Entradas *Degrau* e *Impulso Unitário*. Fonte: Autor

3.1.3 *Intervalo de Tempo de Análise*

O usuário pode escolher o intervalo de tempo em que a análise é feita. Para o valor padrão de 100 s, o eixo das abscissas mostra esse intervalo, conforme as figuras 9 e 10.

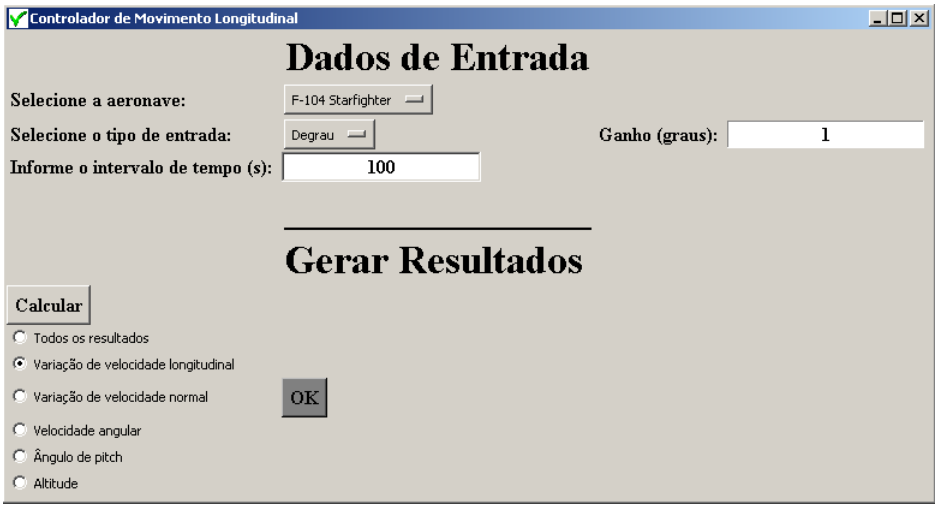

Figura 9: Intervalo de Tempo Padrão de 100 s. Fonte: Autor

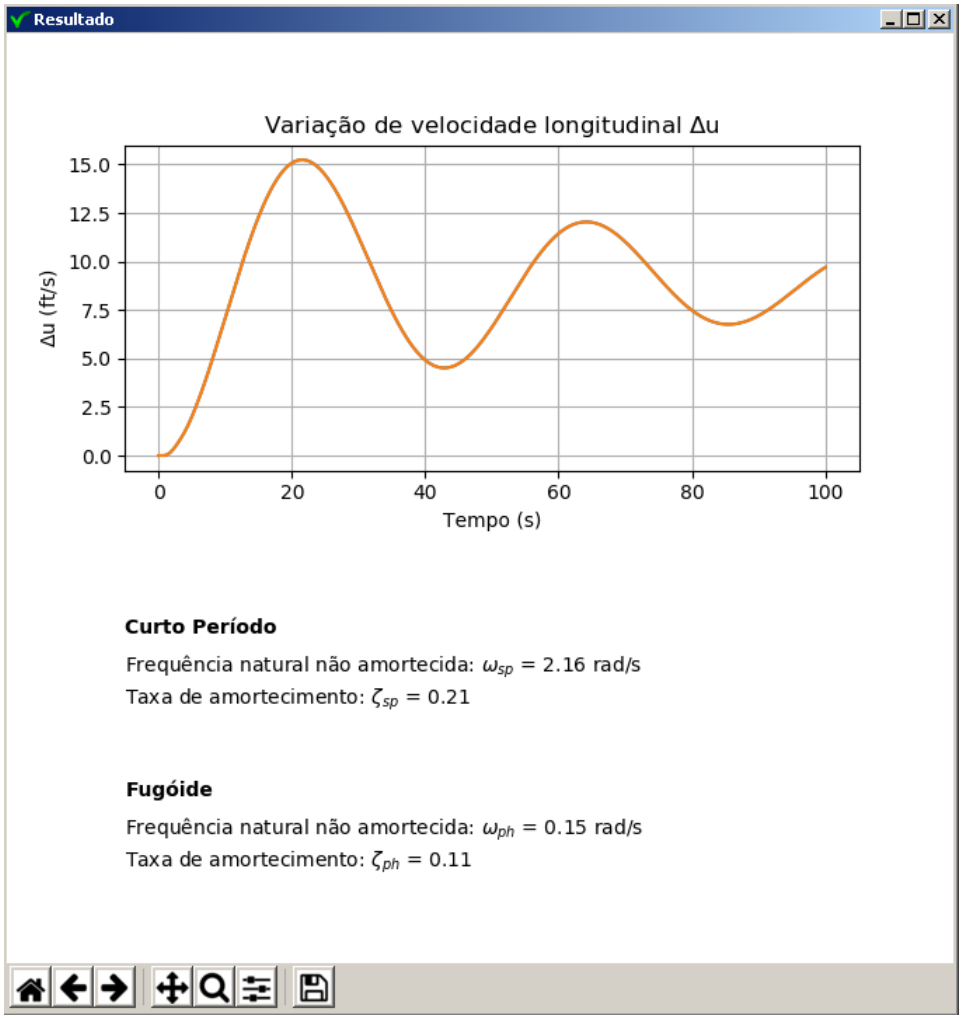

Figura 10: Eixo das Abscissas de 100 s. Fonte: Autor

3.1.4 Opções para os Resultados

Após clicar no botão *Calcular* (figura 8), ao usuário é permitido visualizar, além dos parâmetros ω e ζ, todos os resultados (Δu, Δw, q, θ e h) em uma mesma janela (figura 11), ou mostrá-los separadamente.

Caso o usuário escolher a opção *Variação de velocidade longitudinal* da figura 9, por exemplo, o resultado segue conforme a figura 10.

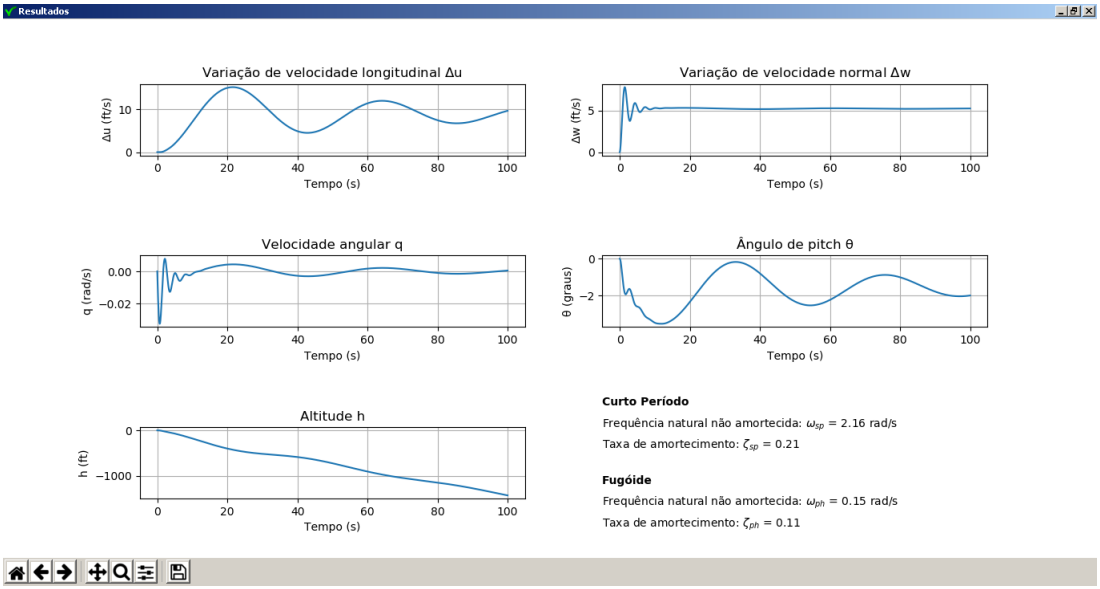

Figura 11: Todos os Resultados em uma Mesma Janela. Fonte: Autor

3.2 Compilando o Código

Há duas maneiras de executar o código implementado:
- executando o *script Controlador.py* diretamente no *software Python*;
- compilando-o para executá-lo em um arquivo *.exe*.

Para compilar o código no sistema operacional *Windows 7*, procede-se abrindo o *prompt* de comando, conforme ilustrado na figura 12.

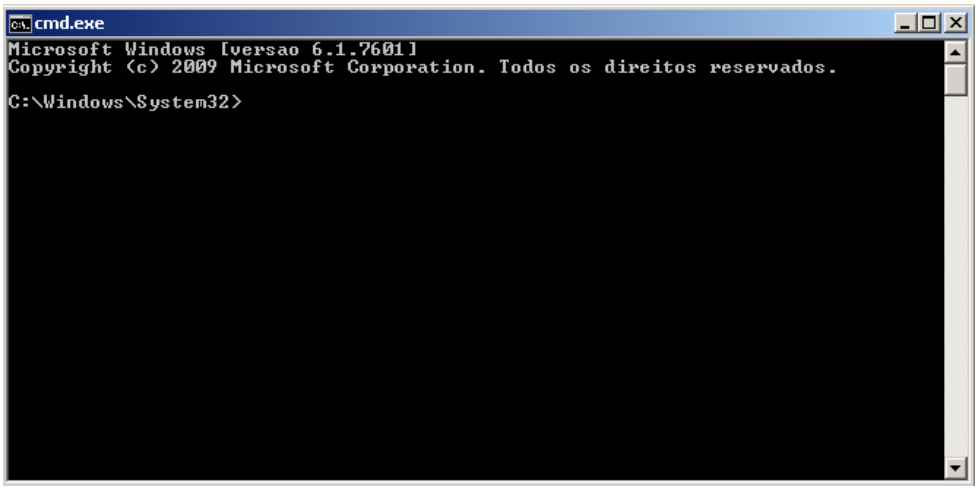

Figura 12: *Prompt* de Comando. Fonte: Autor

Em seguida, é preciso entrar com o comando *cd* e o caminho correspondente, conforme a figura 13:

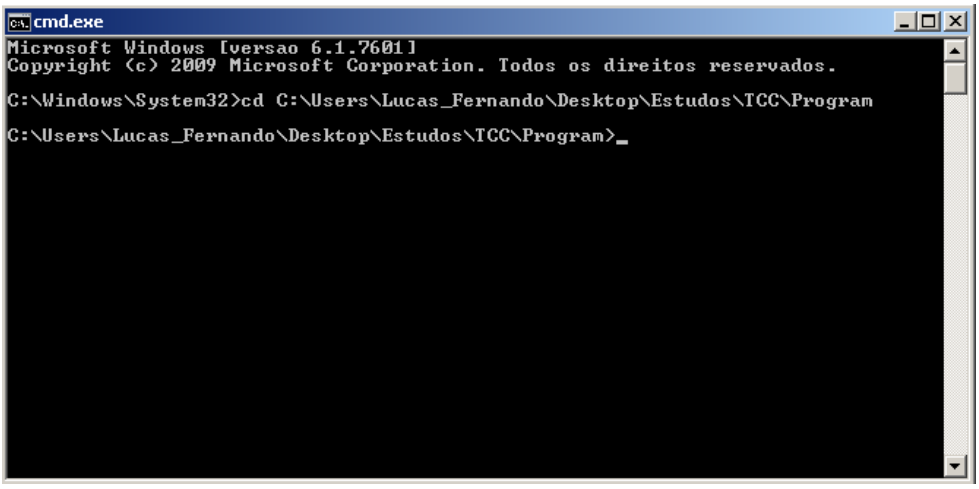

Figura 13: Comando *cd*. Fonte: Autor

No caso da figura 13 tem-se o caminho da pasta *Program*, mas o mesmo pode ser adaptado para o caso desejado.

Após o comando *cd*, digita-se o comando *pyinstaller --onefile Controlador.py*, onde o termo *Controlador.py* é o nome do *script*.

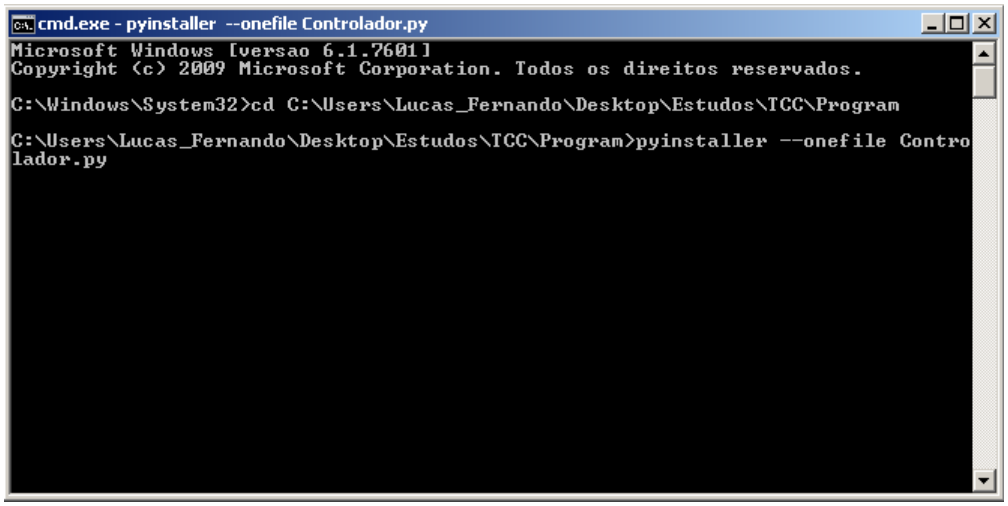

Figura 14: Comando *pyinstaller --onefile Controlador.py*. Fonte: Autor

Aguardar a compilação do código até que a mesma finalize, conforme a figura 15.

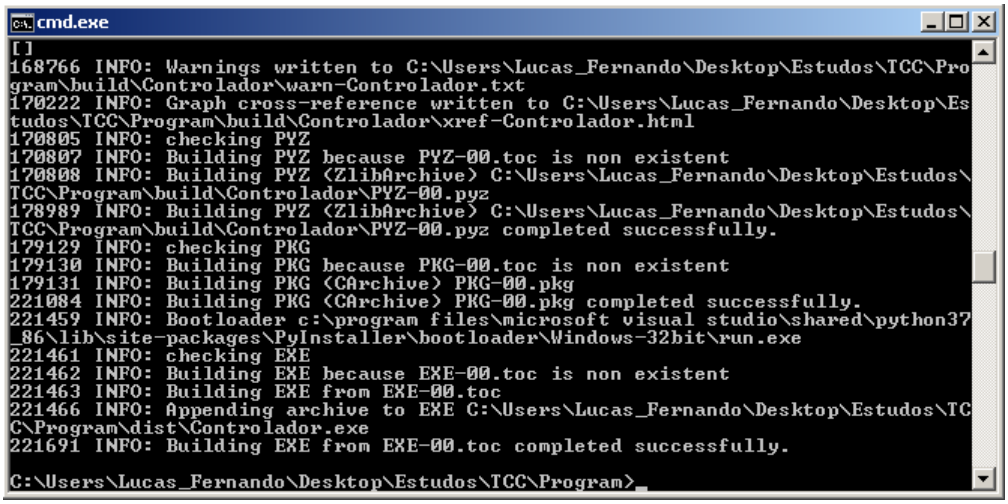

Figura 15: Término da Compilação. Fonte: Autor

O arquivo *.exe* estará na pasta *dist*, conforme as figuras 16 e 17.

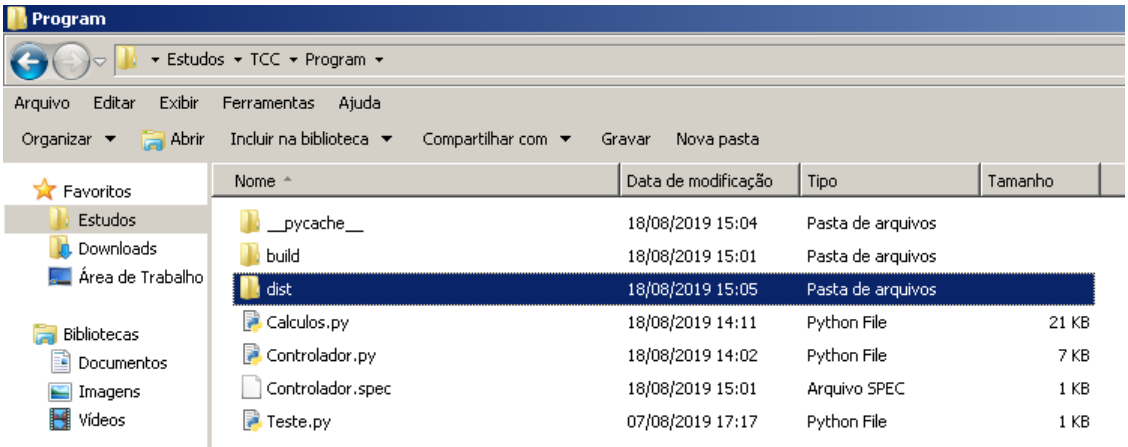

Figura 16: Pasta *dist*. Fonte: Autor

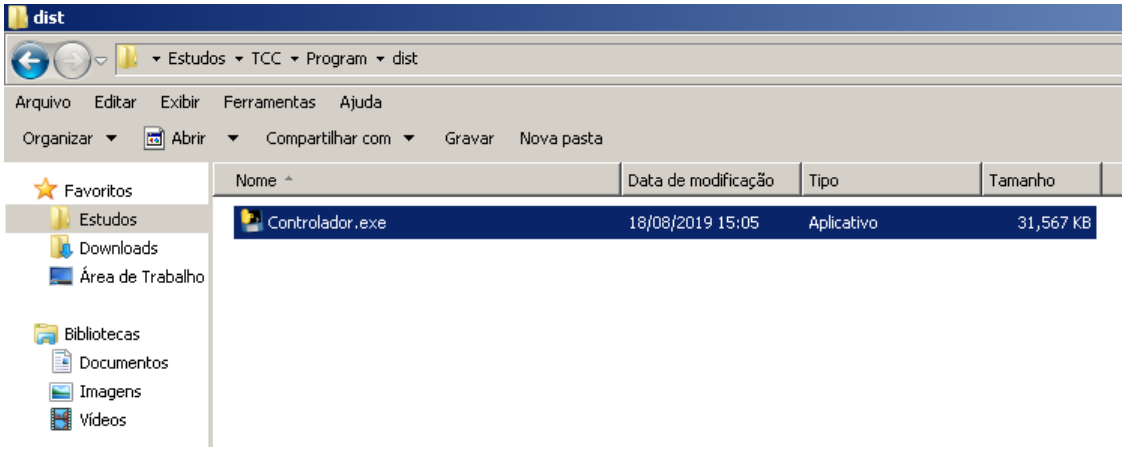

Figura 17: Arquivo *.exe*. Fonte: Autor

4 Validação

Para validar os resultados do *software*, considera-se a análise do ângulo de Pitch, tanto para o *software* (figura 18) como para a bibliografia [I] (figura 19):

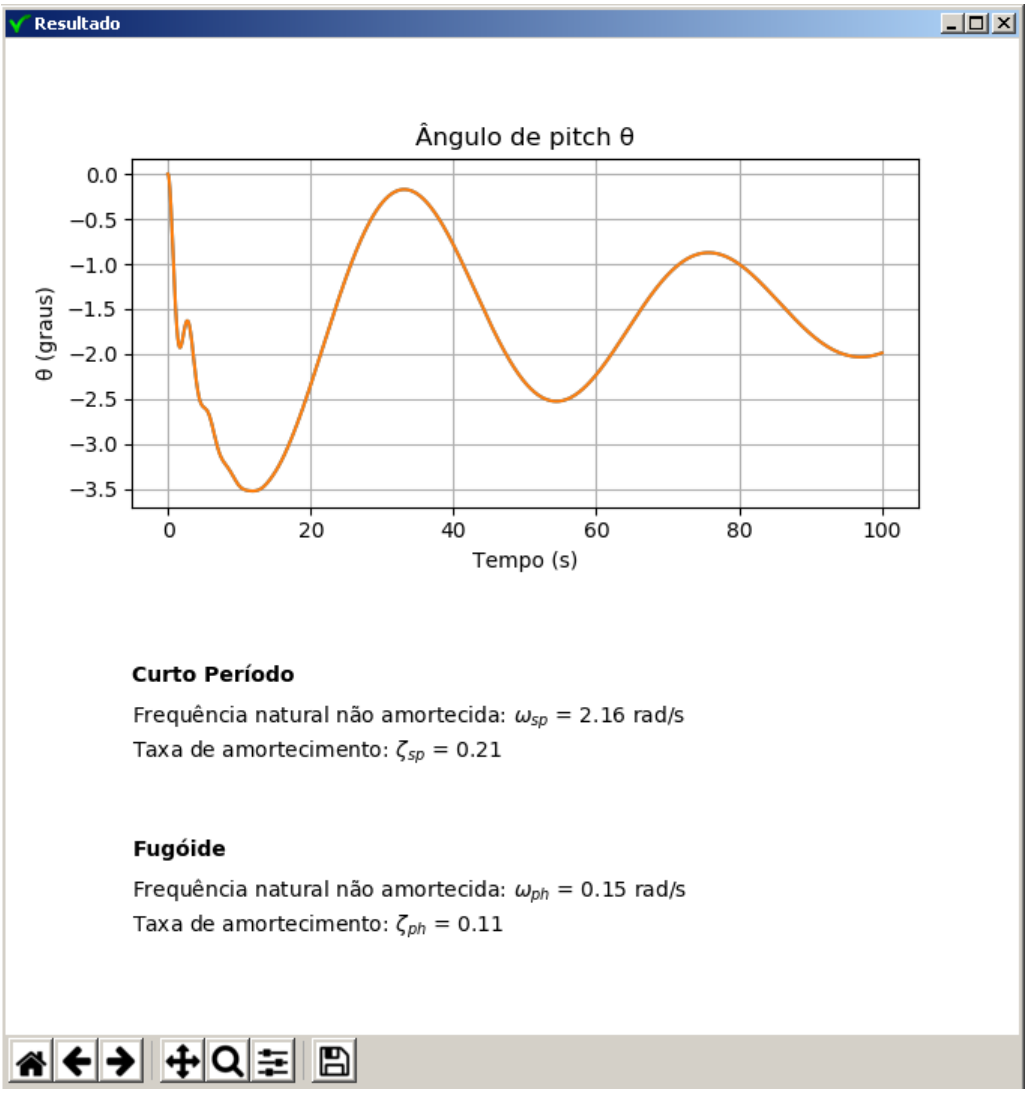

Figura 18: Resultado do *software*. Fonte: Autor

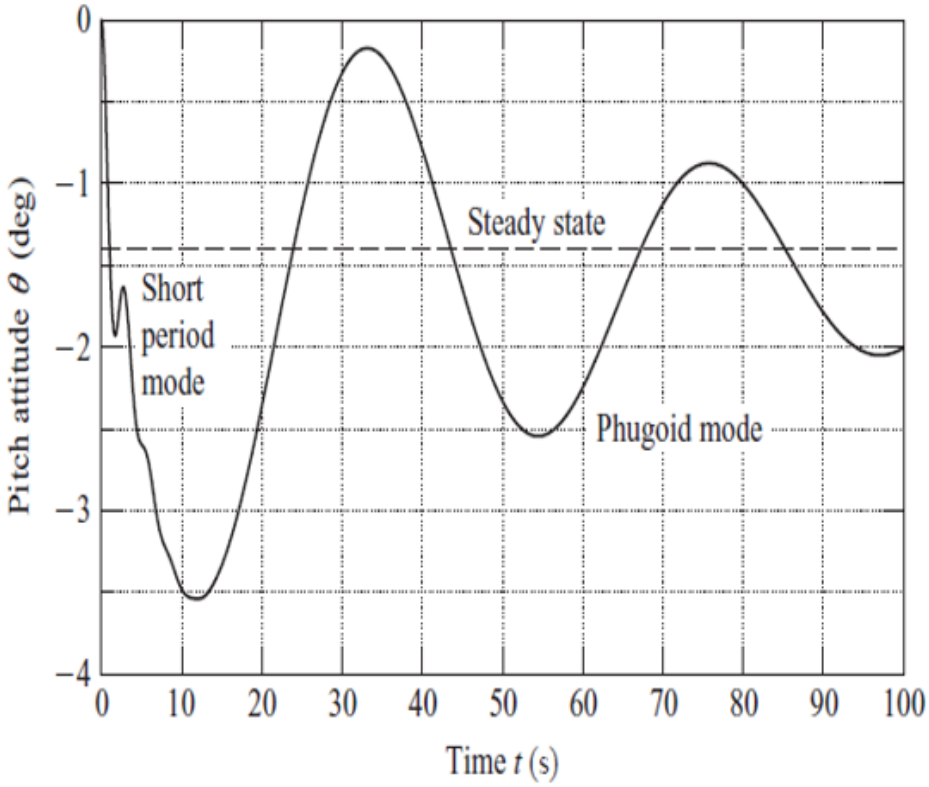

Figura 19: Resultado da bibliografia [I]. Fonte: Bibliografia [I]

Conclui-se que os resultados de ângulos de Pitch coincidem, além da frequência natural não amortecida e da taxa de amortecimento, conforme a bibliografia [I], que fornece:

$$\omega_{sp}(rad/s) = 2.1644 \tag{36}$$

$$\zeta_{sp} = 0.206 \tag{37}$$

$$\omega_{ph}(rad/s) = 0.1474 \tag{38}$$

$$\zeta_{ph} = 0.1126 \tag{39}$$

5 Conclusões

O uso da linguagem de programação *Python* mostra-se eficiente para efetuar operações matemáticas, visto que com o advento da biblioteca *numpy* a velocidade de processamento de cálculos aumentou significativamente. Além disso, os resultados fornecidos pelo *software* são condizentes com os demonstrados na bibliografia [I].

Assim sendo, tem-se uma linguagem gratuita e de código aberto que pode ser uma alternativa às ferramentas já existentes.

6 Anexos

As duas partes do código (*Controlador.py* e *Calculos.py*) estão registradas nas subseções 6.1 e 6.2, respectivamente.

6.1 *Controlador.py*

```
## Biblioteca "tkinter" - desenvolve interfaces grficas do usurio
from tkinter import *
import tkinter as tk
import tkinter.ttk



## Vetor com as aeronaves disponveis
options_airplane = ["F-104 Starfighter","B747-100"]



## Vetor com as entradas disponveis
options_input = ["Degrau","Impulso Unitrio"]



## Caractersticas da janela
root = Tk()
root.title("Controlador de Movimento Longitudinal") #Ttulo
root.geometry("770x390")                 #Dimenses

#cone da janela - arquivo "Logo.gif"
icon = PhotoImage(file="C:/Users/Lucas_Fernando/Desktop/Estudos/"+
"TCC/Logo.gif")
root.tk.call('wm', 'iconphoto', root._w, icon)



## Formataes de texto utilizadas
fonte_titulo = ("Times New Roman","25","bold")
fontePadrao = ("Times New Roman", "12","bold")



## Ttulo de entrada
# Opo  "anchor="w"" fixa a posio do objeto a esquerda
```

```
titulo = Label(root, font=fonte_titulo, text="Dados de Entrada",
anchor="w")

# Opo  "sticky=W" alinha as caixas de texto  esquerda
#Comandos "row" e "column" definem, respectivamente, quais as
#linhas e colunas de determinado objeto
titulo.grid(row=0, column=1, sticky=W)
root.grid_columnconfigure(0, minsize=200)



## Selecionar aeronave
selec_aeronave = Label(root, font=fontePadrao,
text="Selecione a aeronave: ", anchor="w")
selec_aeronave.grid(row=2, column=0, sticky=W)

#Criar varivel para usar os valores do vetor "options_airplane"
variable_airplane = StringVar(root)

#Valor padro - primeiro elemento do vetor "options_airplane"
variable_airplane.set(options_airplane[0])

#Criar lista dropdown com as aeronaves disponveis
aeronave = OptionMenu(root, variable_airplane, *options_airplane)
aeronave.grid(row=2, column=1, sticky=W)



## Selecionar o tipo de entrada
selec_input = Label(root, font=fontePadrao,
text="Selecione o tipo de entrada: ", anchor="w")
selec_input.grid(row=3, column=0, sticky=W)

#Criar varivel para usar os valores do vetor "options_input"
variable_input = StringVar(root)

#Valor padro - primeiro elemento do vetor "options_input"
variable_input.set(options_input[0])

# Funo  para mostrar/ocultar opo do ganho de entrada degrau,
#caso essa opo for a selecionada
def opcao_degrau(choice):
   if choice == "Degrau":
      valor_degrau.grid()
      texto_degrau.grid()
   else:
      valor_degrau.grid_remove()
      texto_degrau.grid_remove()
```

```
#Criar lista dropdown com as entradas disponveis
input_1 = OptionMenu(root, variable_input, *options_input,
command=opcao_degrau)
input_1.grid(row=3, column=1, sticky=W)

#Criar varivel de mostrar/ocultar texto de ganho da entrada Degrau
mostrar_ocultar_texto_degrau = StringVar(root)
mostrar_ocultar_texto_degrau.set("Ganho (graus): ")


#Criar varivel de mostrar/ocultar caixa de input de ganho da
#entrada Degrau
mostrar_ocultar_valor_degrau = StringVar(root)

#Valor padro = 1
mostrar_ocultar_valor_degrau.set("1")

texto_degrau = Label(root, font=fontePadrao,
textvariable = mostrar_ocultar_texto_degrau)
texto_degrau.grid(row=3, column=2)

#Criar caixa de texto para valor de ganho da entrada Degrau
valor_degrau = Entry(root, font=fontePadrao, justify="center",
textvariable = mostrar_ocultar_valor_degrau)
valor_degrau.grid(row=3, column=3, sticky=W)




## Intervalo de tempo analisado
informar_tempo = Label(root, font=fontePadrao,
text="Informe o intervalo de tempo (s): ", anchor="w")
informar_tempo.grid(row=4, column=0)

#Criar varivel para configurar o intervalo de tempo
intervalo_tempo = StringVar(root)

#Valor padro = 100
intervalo_tempo.set("100")

#Criar caixa de texto para valor de intervalo de tempo
valor_tempo = Entry(root, font=fontePadrao, justify="center",
bd=2, textvariable = intervalo_tempo)
valor_tempo.grid(row=4, column=1, sticky=W)




## Interface grfica
titulo_traco = Label(root, font=fonte_titulo,
text="_________________", anchor="w")
```

```
titulo_traco.grid(row=9, column=1, sticky=W)

#Ttulo de Sada
titulo = Label(root, font=fonte_titulo, text="Gerar Resultados",
anchor="w")
titulo.grid(row=10, column=1, sticky=W)

#Criar varivel de sada
option_selected = StringVar()

#Valor padro = 1
option_selected.set("1")

##  Funo  "gerar_grafico()" - aciona o script "Calculos.py"
def gerar_grafico():
   import Calculos

   # Executar funo "gerar_resultados" do script "Calculos.py"
   Calculos.gerar_resultados(variable_airplane.get(),
   variable_input.get(),mostrar_ocultar_valor_degrau.get(),
   intervalo_tempo.get(),option_selected.get())



##  Funo  "gerar_opcoes_resultados()"
def gerar_opcoes_resultados():

   #Cada valor atribudo  sua respectiva opo ser usado no
   #script "Calculos.py"
   #Atribuir valor = 1  opo
   result_all = Radiobutton(root, text="Todos os resultados",
   value="1", var=option_selected)
   result_all.grid(row=13, column=0, sticky=W)

   #Atribuir valor = 2  opo
   result_u = Radiobutton(root,
   text="Variao de velocidade longitudinal", value="2",
   var=option_selected)
   result_u.grid(row=14, column=0, sticky=W)

   #Atribuir valor = 3  opo
   result_w = Radiobutton(root,
   text="Variao de velocidade normal", value="3",
   var=option_selected)
   result_w.grid(row=15, column=0, sticky=W)

   #Atribuir valor = 4  opo
   result_q = Radiobutton(root, text="Velocidade angular",
   value="4", var=option_selected)
   result_q.grid(row=16, column=0, sticky=W)
```

```
    #Atribuir valor = 5  opo
    result_teta = Radiobutton(root, text="ngulo de pitch",
    value="5", var=option_selected)
    result_teta.grid(row=17, column=0, sticky=W)

    #Atribuir valor = 6  opo
    result_h = Radiobutton(root, text="Altitude", value="6",
    var=option_selected)
    result_h.grid(row=18, column=0, sticky=W)

    #Criar boto "OK" - aciona a funo "gerar_grafico", com o
    #valor da opo  j  escolhida pelo usurio
    button = Button(root, font=fontePadrao,text="OK",
    command=gerar_grafico, background = "gray")
    button.grid(row=15, column=1, sticky=W)



#Criar boto "Calcular" - aciona a funo "gerar_opcoes_resultados"
button = Button(root, font=fontePadrao,text="Calcular",
command=gerar_opcoes_resultados)
button.grid(row=12, column=0, sticky=W)


root.mainloop() #Encerrar loop de interface grfica
```

6.2 *Calculos.py*

```
import os                    #Usado para concatenar
#diretrios
from collections import defaultdict #Ler arquivo de texto
import numpy as np           # Funes  matemticas em geral,
#como operaes bsicas com matrizes
from numpy import linalg as LA #Obteno de autovalores e
#autovetores de uma matriz
import math                  #Pacote de operaes
#matemticas. No caso, o uso de nmeros complexos
import matplotlib.pyplot as plt #Pacote para plotagem de
#grficos
from matplotlib import pyplot as PLT #Permite plotar vrios
#grficos em uma mesma janela



##  Funo  que executa o modelo matemtico de movimento longitudinal
def
   gerar_resultados(name_file,tipo_entrada,valor_entrada,intervalo_tempo,
resultado_escolhido):

   #Converter o valor de intervalo de tempo de tipo texto
   # ("string") para um tipo numrico (nmero real "float")
   intervalo_tempo_final = float(intervalo_tempo)



   #Concatenar o nome do arquivo de texto (oriundo de
   #"Controlador.py") com a expresso ".txt"
   name_file_final = name_file + ".txt"

   #Ler dados numricos da aeronave escolhida, a partir dos dados
   #do arquivo de texto
   path_mine = os.path.join(r"C:/Users/Lucas_Fernando/Desktop/"+
   "Estudos/TCC/Aeronaves/", name_file_final)
   file_mine = open(path_mine,"r")
   data_mine = defaultdict(str)
   for line in file_mine:
      final_data = line.strip().split("=")
      data_mine[final_data[0].strip()] = final_data[1].strip()

   g = float(data_mine['g'])
   m = float(data_mine['m'])
   I_y = float(data_mine['I_y'])
   teta_0 = float(data_mine['teta_e'])
   h_0 = float(data_mine['h_0'])
```

```
u_0 = float(data_mine['V_0'])
X_u = float(data_mine['X_u'])
X_w = float(data_mine['X_w'])
X_w_ponto = float(data_mine['X_w_ponto'])
X_q = float(data_mine['X_q'])
X_eta = float(data_mine['X_eta'])
Z_u = float(data_mine['Z_u'])
Z_w = float(data_mine['Z_w'])
Z_w_ponto = float(data_mine['Z_w_ponto'])
Z_q = float(data_mine['Z_q'])
Z_eta = float(data_mine['Z_eta'])
M_u = float(data_mine['M_u'])
M_w = float(data_mine['M_w'])
M_w_ponto = float(data_mine['M_w_ponto'])
M_q = float(data_mine['M_q'])
M_eta = float(data_mine['M_eta'])

U_e = u_0*math.cos(teta_0) #velocidade linear no eixo
#longitudinal x
W_e = u_0*math.sin(teta_0) #velocidade linear no eixo
#normal z



#Matriz (4x4) A (em termos das derivadas na forma dimensional)
A = np.matrix([[(X_u/m)+(X_w_ponto*Z_u/(m*(m-Z_w_ponto))),
(X_w/m)+(X_w_ponto*Z_w/(m*(m-Z_w_ponto))),
((X_q-(m*W_e))/m)+((Z_q+(m*U_e))*X_w_ponto/(m*(m-Z_w_ponto))),
(-g*math.cos(teta_0))-
(X_w_ponto*g*(math.sin(teta_0))/(m-Z_w_ponto))],
[Z_u/(m-Z_w_ponto),Z_w/(m-Z_w_ponto),(Z_q+(m*U_e))/(m-Z_w_ponto),
(-m*g*math.sin(teta_0))/(m-Z_w_ponto)],
[(1/I_y)*(M_u+((M_w_ponto*Z_u)/(m-Z_w_ponto))),
(1/I_y)*(M_w+((M_w_ponto*Z_w)/(m-Z_w_ponto))),
(1/I_y)*(M_q+(M_w_ponto*(Z_q+(m*U_e))/(m-Z_w_ponto))),
(-M_w_ponto*m*g*math.sin(teta_0))/(I_y*(m-Z_w_ponto))],
[0,0,1,0]])

#Autovalores e autovetores de A, respectivamente
autovalores, autovetores = LA.eig(A)

#Obter elementos da matriz de autovalores
#Curto Perodo
lambda_s = autovalores[0]
lambda_s_conj = autovalores[1]

#Fugide
lambda_p = autovalores[2]
lambda_p_conj = autovalores[3]
```

```
#Inversa da matriz dos autovetores
autovetores_inv = np.linalg.inv(autovetores)

#Inversa da matriz A
A_inv = np.linalg.inv(A)



## Curto Perodo ##
#Frequncia natural no amortecida (rad/s)
ws_sp = max(abs(lambda_s.imag),abs(lambda_p.imag))

if ws_sp == abs(lambda_s.imag):
   #Taxa de amortecimento
   ratio_sp = abs(lambda_s.real)/ws_sp

if ws_sp == abs(lambda_p.imag):
   #Taxa de amortecimento
   ratio_sp = abs(lambda_p.real)/ws_sp



#Fugide
#Frequncia natural no amortecida (rad/s)
ws_ph = min(abs(lambda_s.imag),abs(lambda_p.imag))

if ws_ph == abs(lambda_s.imag):
   #Taxa de amortecimento
   ratio_ph = abs(lambda_s.real)/ws_ph

if ws_ph == abs(lambda_p.imag):
   #Taxa de amortecimento
   ratio_ph = abs(lambda_p.real)/ws_ph



#Matriz (4x1) B (em termos das derivadas na forma dimensional)
B = np.matrix([[(X_eta/m)+(X_w_ponto*Z_eta/(m*(m-Z_w_ponto)))],
[Z_eta/(m-Z_w_ponto)],
[(M_eta/I_y)+(M_w_ponto*Z_eta/(I_y*(m-Z_w_ponto)))],
[0]])
```

```
#Matriz (4x4) identidade I
I = np.matrix([[1,0,0,0],
[0,1,0,0],
[0,0,1,0],
[0,0,0,1]])



#Construir matriz (4x4) de autovalores
autovalores_44 = np.matrix([[lambda_s,0,0,0],
[0,lambda_s_conj,0,0],
[0,0,lambda_p,0],
[0,0,0,lambda_p_conj]])



#Inverter matriz de autovalores
autovalores_44_inv = np.linalg.inv(autovalores_44)



## Operaes finais ##
#Mult1
a = np.matmul(autovalores_44_inv,autovetores_inv)
mult1 = np.matmul(a,B)

#Mult2
mult2 = np.matmul(A_inv,B)



## Definir as matrizes de resultados ##
resultados_u = []            #Matriz de resultados u
resultados_w = []            #Matriz de resultados w
resultados_q = []            #Matriz de resultados q
resultados_teta = []         #Matriz de resultados
resultados_h = []            #Matriz de resultados h



#Intervalo de tempo discretizado em n pontos
n=5000
tt = np.linspace(0,intervalo_tempo_final, n)

#Valor de cada intervalo de tempo
delta_t = (intervalo_tempo_final-0)/(n-1)
```

```
#Criar objeto para criao de grficos
fig = PLT.figure()



## Construir um loop que calcula as variveis de interesse em
#intervalos discretizados de tempo
for t in tt:
   #Matriz e^(lambda*t)
   e_lambda_t = np.matrix([[math.exp((lambda_s.real)*t)*
   (math.cos((lambda_s.imag)*t)+
   (math.sin((lambda_s.imag)*t))*1j),0,0,0],
   [0,math.exp((lambda_s_conj.real)*t)*
   (math.cos((lambda_s_conj.imag)*t)+
   (math.sin((lambda_s_conj.imag)*t))*1j),0,0],
   [0,0,math.exp((lambda_p.real)*t)*
   (math.cos((lambda_p.imag)*t)+
   (math.sin((lambda_p.imag)*t))*1j),0],
   [0,0,0,math.exp((lambda_p_conj.real)*t)*
   (math.cos((lambda_p_conj.imag)*t)+
   (math.sin((lambda_p_conj.imag)*t))*1j)]])

   #Ve_lambda
   Ve_lambda_t = np.matmul(autovetores,e_lambda_t)

   #Para a entrada degrau
   Ve_lambda_tmult1 = np.matmul(Ve_lambda_t,mult1)

   #Para a entrada impulso unitrio
   mult3 = np.matmul(Ve_lambda_t,autovetores_inv)



   ## Se a entrada for a Degrau ##
   if tipo_entrada == "Degrau":

      #Converter o valor da entrada degrau de tipo "string"
      #para um tipo numrico "float"
      valor_degrau_final = float(valor_entrada)

      #Vetor de resultados
      y_t = valor_degrau_final*np.subtract(Ve_lambda_tmult1,mult2)

      # Variao  de velocidade longitudinal (u) - primeira
      #linha de y_t
      #Para esse clculo, o ngulo de step  convertido de
      #graus para radianos
      resultados_u.append((math.pi/180)*y_t[0].real)
```

```
    # Variao  de velocidade normal (w) - segunda linha de
    #y_t
    #Para esse clculo, o ngulo de step  convertido de
    #graus para radianos
    resultados_w.append(-(math.pi/180)*y_t[1].real)

    #Velocidade angular (q) - terceira linha de y_t
    #Para esse clculo, o ngulo de step  convertido de
    #graus para radianos
    resultados_q.append((math.pi/180)*y_t[2].real)

    #ngulo de pitch () - em graus - quarta linha de y_t
    resultados_teta.append(y_t[3].real)

    #Altitude (h)
    #Derivada da altitude
    h_ponto=u_0*(math.pi/180)*y_t[3].real +
    (math.pi/180)*y_t[1].real

    #Altitude final
    if len(resultados_h) == 0:
       resultados_h.append(h_0)
    else:
       resultados_h.append(resultados_h[-1] +
       (h_ponto*delta_t))



## Se a entrada for a Impulso Unitrio ##
if tipo_entrada == "Impulso Unitrio":

    #Vetor de resultados
    y_t = np.matmul(mult3,B)

    # Variao  de velocidade longitudinal (u) -
    #primeira linha de y_t
    resultados_u.append(y_t[0].real)

    # Variao  de velocidade normal (w) - segunda linha de
    #y_t
    resultados_w.append(-y_t[1].real)

    #Velocidade angular (q) - terceira linha de y_t
    resultados_q.append(y_t[2].real)

    #ngulo de pitch () - quarta linha de y_t
    resultados_teta.append(y_t[3].real)

    #Altitude (h)#
```

```
        #Derivada da altitude
        h_ponto=u_0*y_t[3].real + y_t[1].real

        #Altitude final
        if len(resultados_h) == 0:
            resultados_h.append(h_0)
        else:
            resultados_h.append(resultados_h[-1] +
            (h_ponto*delta_t))



## Exibio grfica ##
#Exibir todos os resultados
#Valor "1" da funo "gerar_opcoes_resultados()" do script
#"Controlador.py"
if resultado_escolhido == "1":

    #Grfico Velocidade Longitudinal u
    resultado_final_u = np.array(resultados_u).ravel() #Ajustar
    #a matriz para um vetor
    ax1 = fig.add_subplot(5,2,1)
    ax1.plot(tt,resultado_final_u)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Variao de velocidade longitudinal $\Delta$u')
        #Ttulo do grfico
    plt.xlabel('Tempo (s)')                 #Eixo x
    plt.ylabel('$\Delta$u (ft/s)')                  #Eixo y
    man = plt.get_current_fig_manager()
    man.canvas.set_window_title("Resultados") #Ttulo da janela
    #dos grficos
    man.window.wm_iconbitmap("C:/Users/Lucas_Fernando/Desktop/"+
    "Estudos/TCC/logo.ico") #cone da janela

    #Dados de amortecimento
    #Curto Perodo
    plt.annotate('Curto Perodo', weight = "bold",
    xy=(1.2, -3.5), xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    " $_{sp}$ = " + "%.2f" % ws_sp + " rad/s",
    xy=(1.2, -3.8), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{sp}$ = " +
    "%.2f" % ratio_sp,
    xy=(1.2, -4.1), xycoords='axes fraction')

    #Fugide
    plt.annotate('Fugide', weight = "bold", xy=(1.2, -4.6),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    " $_{ph}$ = " + "%.2f" % ws_ph + " rad/s",
```

```
    xy=(1.2, -4.9), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{ph}$ = " +
    "%.2f" % ratio_ph,
    xy=(1.2, -5.2), xycoords='axes fraction')


    #Grfico Velocidade Normal w
    resultado_final_w = np.array(resultados_w).ravel()
    ax2 = fig.add_subplot(5,2,2)
    ax2.plot(tt,resultado_final_w)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Variao de velocidade normal $\Delta$w')
    plt.xlabel('Tempo (s)')
    plt.ylabel('$\Delta$w (ft/s)')

    #Grfico Velocidade Angular q
    resultado_final_q = np.array(resultados_q).ravel()
    ax3 = fig.add_subplot(5,2,5)
    ax3.plot(tt,resultado_final_q)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Velocidade angular q')
    plt.xlabel('Tempo (s)')
    plt.ylabel('q (rad/s)')

    #Grfico de Pitch
    resultado_final_teta = np.array(resultados_teta).ravel()
    ax4 = fig.add_subplot(5,2,6)
    ax4.plot(tt,resultado_final_teta)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('ngulo de pitch ')
    plt.xlabel('Tempo (s)')
    plt.ylabel(' (graus)')

    #Grfico de Altitude h
    resultado_final_h = np.array(resultados_h).ravel()
    ax5 = fig.add_subplot(5,2,9)
    ax5.plot(tt,resultado_final_h)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Altitude h')
    plt.xlabel('Tempo (s)')
    plt.ylabel('h (ft)')

    PLT.show()  #Comando para mostrar os grficos


# Variao  de velocidade longitudinal (u)
#Valor "2" da funo "gerar_opcoes_resultados()" do script
#"Controlador.py"
if resultado_escolhido == "2":
    resultado_final_u = np.array(resultados_u).ravel()
    ax = fig.add_subplot(2,1,1)
```

```
    ax.plot(tt,resultado_final_u)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Variao de velocidade longitudinal $\Delta$u')
    plt.xlabel('Tempo (s)')
    plt.ylabel('$\Delta$u (ft/s)')
    plt.plot(tt,resultado_final_u)
    man = plt.get_current_fig_manager()
    man.canvas.set_window_title("Resultado")
    man.window.wm_iconbitmap("C:/Users/Lucas_Fernando/"+
    "Desktop/Estudos/TCC/logo.ico")

    #Dados de amortecimento
    #Curto Perodo
    plt.annotate('Curto Perodo', weight = "bold", xy=(0, -0.5),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{sp}$ = " + "%.2f" % ws_sp + " rad/s",
    xy=(0, -0.62), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{sp}$ = " +
    "%.2f" % ratio_sp,
    xy=(0, -0.72), xycoords='axes fraction')

    #Fugide
    plt.annotate('Fugide', weight = "bold", xy=(0, -1.0),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{ph}$ = " + "%.2f" % ws_ph + " rad/s",
    xy=(0, -1.12), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{ph}$ = " +
    "%.2f" % ratio_ph,
    xy=(0, -1.22), xycoords='axes fraction')

    plt.show()


# Variao  de velocidade normal (w)
#Valor "3" da funo "gerar_opcoes_resultados()" do script
#"Controlador.py"
if resultado_escolhido == "3":
    resultado_final_w = np.array(resultados_w).ravel()
    ax = fig.add_subplot(2,1,1)
    ax.plot(tt,resultado_final_w)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Variao de velocidade normal $\Delta$w')
    plt.xlabel('Tempo (s)')
    plt.ylabel('$\Delta$w (ft/s)')
    plt.plot(tt,resultado_final_w)
    man = plt.get_current_fig_manager()
    man.canvas.set_window_title("Resultado")
    man.window.wm_iconbitmap("C:/Users/Lucas_Fernando/"+
    "Desktop/Estudos/TCC/logo.ico")
```

```
    #Dados de amortecimento
    #Curto Perodo
    plt.annotate('Curto Perodo', weight = "bold", xy=(0, -0.5),
       xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    " $_{sp}$ = " + "%.2f" % ws_sp + " rad/s",
    xy=(0, -0.62), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{sp}$ = " +
    "%.2f" % ratio_sp,
    xy=(0, -0.72), xycoords='axes fraction')

    #Fugide
    plt.annotate('Fugide', weight = "bold", xy=(0, -1.0),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    " $_{ph}$ = " + "%.2f" % ws_ph + " rad/s",
    xy=(0, -1.12), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{ph}$ = " +
    "%.2f" % ratio_ph,
    xy=(0, -1.22), xycoords='axes fraction')

    plt.show()


# Variao  de velocidade angular (q)
#Valor "4" da funo "gerar_opcoes_resultados()" do script
#"Controlador.py"
if resultado_escolhido == "4":
    resultado_final_q = np.array(resultados_q).ravel()
    ax = fig.add_subplot(2,1,1)
    ax.plot(tt,resultado_final_q)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Velocidade angular q')
    plt.xlabel('Tempo (s)')
    plt.ylabel('q (rad/s)')
    plt.plot(tt,resultado_final_q)
    man = plt.get_current_fig_manager()
    man.canvas.set_window_title("Resultado")
    man.window.wm_iconbitmap("C:/Users/Lucas_Fernando/"+
    "Desktop/Estudos/TCC/logo.ico")

    #Dados de amortecimento
    #Curto Perodo
    plt.annotate('Curto Perodo', weight = "bold", xy=(0, -0.5),
       xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    " $_{sp}$ = " + "%.2f" % ws_sp + " rad/s",
    xy=(0, -0.62), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{sp}$ = " +
    "%.2f" % ratio_sp,
```

```
    xy=(0, -0.72), xycoords='axes fraction')

    #Fugide
    plt.annotate('Fugide', weight = "bold", xy=(0, -1.0),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{ph}$ = " + "%.2f" % ws_ph + " rad/s",
    xy=(0, -1.12), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{ph}$ = " +
    "%.2f" % ratio_ph,
    xy=(0, -1.22), xycoords='axes fraction')

    plt.show()


# Variao  de velocidade angular ()
#Valor "5" da funo "gerar_opcoes_resultados()" do script
#"Controlador.py"
if resultado_escolhido == "5":
    resultado_final_teta = np.array(resultados_teta).ravel()
    ax = fig.add_subplot(2,1,1)
    ax.plot(tt,resultado_final_teta)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('ngulo de pitch ')
    plt.xlabel('Tempo (s)')
    plt.ylabel(' (graus)')
    plt.plot(tt,resultado_final_teta)
    man = plt.get_current_fig_manager()
    man.canvas.set_window_title("Resultado")
    man.window.wm_iconbitmap("C:/Users/Lucas_Fernando/"+
    "Desktop/Estudos/TCC/logo.ico")

    #Dados de amortecimento
    #Curto Perodo
    plt.annotate('Curto Perodo', weight = "bold", xy=(0, -0.5),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{sp}$ = " + "%.2f" % ws_sp + " rad/s",
    xy=(0, -0.62), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{sp}$ = " +
    "%.2f" % ratio_sp,
    xy=(0, -0.72), xycoords='axes fraction')

    #Fugide
    plt.annotate('Fugide', weight = "bold", xy=(0, -1.0),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{ph}$ = " + "%.2f" % ws_ph + " rad/s",
    xy=(0, -1.12), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{ph}$ = " +
    "%.2f" % ratio_ph,
```

```
    xy=(0, -1.22), xycoords='axes fraction')

    plt.show()


#Altitude (h)
#Valor "6" da funo "gerar_opcoes_resultados()" do script
#"Controlador.py"
if resultado_escolhido == "6":
    resultado_final_h = np.array(resultados_h).ravel()
    ax = fig.add_subplot(2,1,1)
    ax.plot(tt,resultado_final_h)
    plt.grid(b=True, which='major', linestyle='-')
    plt.title('Altitude h')
    plt.xlabel('Tempo (s)')
    plt.ylabel('h (ft)')
    plt.plot(tt,resultado_final_h)
    man = plt.get_current_fig_manager()
    man.canvas.set_window_title("Resultado")
    man.window.wm_iconbitmap("C:/Users/Lucas_Fernando/"+
    "Desktop/Estudos/TCC/logo.ico")

    #Dados de amortecimento
    #Curto Perodo
    plt.annotate('Curto Perodo', weight = "bold", xy=(0, -0.5),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{sp}$ = " + "%.2f" % ws_sp + " rad/s",
    xy=(0, -0.62), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{sp}$ = " +
    "%.2f" % ratio_sp,
    xy=(0, -0.72), xycoords='axes fraction')

    #Fugide
    plt.annotate('Fugide', weight = "bold", xy=(0, -1.0),
    xycoords='axes fraction')
    plt.annotate("Frequncia natural no amortecida: " +
    "$_{ph}$ = " + "%.2f" % ws_ph + " rad/s",
    xy=(0, -1.12), xycoords='axes fraction')
    plt.annotate("Taxa de amortecimento: " + "$_{ph}$ = " +
    "%.2f" % ratio_ph,
    xy=(0, -1.22), xycoords='axes fraction')

    plt.show()
```

7 Bibliografia

[I] COOK, Michael V. Flight Dynamics Principles: A linear systems approach to aircraft stability and control. 2ed. Oxford: Elsevier. 2007.

[II] ETKIN, Bernard; REID, Lloyd. Dynamics of Flight: Stability and Control. 3ed. Nova Iorque: John Wiley e Sons. 1996.

[III] Site da comunidade online de programação: https://stackoverflow.com/