Programação científica com python

Author: Arthur Chabole

Institute: UNESP - Equipe Zebra Aerodesign

Date: setembro. 18, 2021

Version: 0.1

Bio: arthur.chabole@unesp.br

"Nada é tão sério quanto parece quando você pensa sobre isso"

Daniel Kahneman

Conteúdo

Capítulo 1

Primeiros passos com PYTHON

Para iniciar seus primeiros passos com python é indicado o uso de algum ambiente de programação. O pacote *Anaconda* é muito útil, pois oferece as principais IDE's e nele vêm instaladas as principais bibliotecas de python como *numpy, matplotlib. scipy, pandas e seaborn*. Recomenda-se utilizar as IDE's mais preparadas para programação científica, que são o spyder e o pyLab, todavia, soluções online como *GoogleColab* são uma opção bastante viável.

- Tutorial pra instalação do anaconda: https://www.youtube.com/watch?v=$_eK0z5QbpKA$
- Drive com as bibliografias de python Link para o drive
- GitHub do Chabole https://github.com/Chabole/Python
- GitHub da Equipe Zebra Link para os arquivos
- GitHub dos notebooks de desempenho Link para os arquivos

1.1 Git e GitHub

Todos os códigos da equipe zebra são distribuidos pelo GitHub. GitHub é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. Vídeo direto ao ponto explicando o básico do GitHub desktop Link vídeo.

- Baixe o git em https://git-scm.com/
- Crie uma conta no Github em https://github.com
- Baixe e instale o Github desktop https://github.com
- Curso sobre Github assista a playlist do prof. Gustavo Guanabara Link do curso aqui

1.2 Configurando a biblioteca ZebraLib

A equipe zebra desenvolveu uma biblioteca para cálculos aeronáuticos. Ela será de grande ajuda para auxiliar no desenvolvimento de algoritmos para análise do aerodesign. Lista de passos para instala-la e o vídeo passo a passo para auxiliar Link do vídeo.

- GitHub da equipe zebra https://github.com/Zebra-Aerodesign/ZebraLib
- Clone o repositório do Github no seu computador em $C:/Users//..../Anaconda3$
- Vá em $C:/Users/arthu/Anaconda3/Lib/site-packages$
- Crie um arquivo no bloco de notas chamado $ZebraLib.pth$ e salve na pasta $site-packages$
- Dentro do arquivo coloque o local onde está a biblioteca exemplo: $C:/Users/arthu/Anaconda3/ZebraLib$

- Sucesso! Agora é só importar a biblioteca no seu código e se divertir ... :)

1.3 Onde encontrar ajuda

- **Curso básico de python**
  - Curso de algorimos e lógica de programação: Link playlist do mundo 0
  - Programação em python mundo 1: Link playlist do mundo 1
  - Programação em python mundo 2: Link playlist do mundo 2
  - Programação em python mundo 3: Link do mundo 3
  - Programação em python exercícios: Link dos exercícios
  - Conceitos de Prog. orientada a objeto (assistir aulas em vermelho): Link playlist POO
- **Curso avançado de python**
  - Jamal UFTPR - Cálculo numérico: Link playlist aqui
  - APMonitor - Computação para engenheiros: Link playlist aqui
  - APMonitor site - Tutoriais e exercícios: Link do site
- **Documentação das principais bibliotecas**
  - Aprender sobre *numpy*: documentação numpy
  - Aprender a plotar gráfico com *matplotlib*: documentação do matplotlib
  - Aprender computação ciêntifica com *scipy*: documentação do scipy

1.4 Roteiro recomendado de estudos para PYTHON

1. Estruturas de dados - GitHub Exemplos
   (a). Números e caracteres
      I. Operadores aritméticos - Aula
      II. tipos primitivos - Aula
      III. string - Aula
   (a). Sequências
      I. listas - Aula parte 1 e Aula parte 2
      II. dicionários - Aula

   Ao final você precisa conseguir resolver os desafios dos vídeos:
   - Desafio 1
   - Desafio 2
   - Desafio 3
2. Controle de fluxo
   (a). Básico
      I. Condicionais - Aula parte 1 e Aula parte 2
      II. For loop - Aula
      III. While loop - Aula parte 1 e Aula parte 2
   (b). Exceções - Aula
      I. Try - except
      II. Lidando com exceções

   Ao final você precisa conseguir resolver os desafios dos vídeos:

- Desafio 1
- Desafio 2
- Desafio 3

3. Funções e modularização
   (a). Padrão
      I. Funções - Aula parte 1 e Aula parte 2
         A. Argumentos fixos
         B. Argumentos padrão e opcionais
         C. *args e *kwargs
         D. return - com mais valores
      II. Funções built-in
         A. Anonymous - lambda
         B. Map
         C. filter

   Ao final você precisa conseguir resolver os desafios dos vídeos:
   - Desafio 1
   - Desafio 2
   - Desafio 3
   - Desafio 4

   (b). Módulos - Aula
      I. importar módulos
         A. numpy, matplotlib e pandas - GitHub Exemplos
         B. seaborn - GitHub Exemplos
         C. sympy - GitHub Exemplos
      II. criar módulos
4. Programação orientada a objeto - Aula
   (a). Classes e objetos - GitHub Exemplos
   (b). Métodos e atributos
   (c). Polimorfismos e herança
   (d). Decorators

1.4.1 Cronograma recomendado

1. Semana 1 até (26/01/22) - Bloco 1
2. Semana 2 até (02/02/22) - Bloco 2
3. Semana 3 até (09/02/22) - Bloco 3

1.5 O Zen of Python de Peters

O Zen do **Python** é uma coleção de 19 princípios orientadores. Visando construir bons hábitos e costumes dentro da programação (PEP). Seguir esses principios é fundamental para se resolver problemas complicados ou extensos, como é o caso da Equipe Zebra.

- Bonito é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Linear é melhor do que aninhado.
- Esparso é melhor que denso.
- Legibilidade conta.
- Casos especiais não são especiais o bastante para quebrar as regras.
- Ainda que praticidade vença a pureza.
- Erros nunca devem passar silenciosamente.
- A menos que sejam explicitamente silenciados.
- Diante da ambiguidade, recuse a tentação de adivinhar.
- Dever haver um e preferencialmente apenas um modo óbvio para fazer algo.
- Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
- Agora é melhor que nunca.
- Apesar de que nunca normalmente é melhor do que exatamente agora.
- Se a implementação é difícil de explicar, é uma má ideia.
- Se a implementação é fácil de explicar, pode ser uma boa ideia.
- Namespaces são uma grande ideia vamos ter mais dessas!

Capítulo 2

Funções

Introduction

- *Funções*
- *Variais locais e globais*
- *returns e argumentos*
- *funções compostas*

2.1 Funções simples

Na programação, funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação. Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

Toda função começa com a palavra especial def seguido pelo nome da função e entre parenteses os argumentos que ela recebe para fazer suas operações e ao final termina com ":"(fonte de erros comum). Dentro da função é programado a sequencia de instruções que deve ser realizado. Por fim, a palavra return é uma palavra especial e determina quais variáveis locais da função serão retornadas para o programa principal. É importante destacar que qualquer outra variável atribuída dentro da função que não seja retornada não será visível pelo programa principal.

Vejamos um exemplo abaixo de como funciona a sintaxe de um função em python.

```
def alongamento(b, S): #argumentos
    #Onde a função faz
    y = b**2/S
    return y  #o que ela retorna
```

alongamento é o nome da função que recebe dois argumentos b (envergadura) e S(área alar da asa). A função faz uma operação sobre essas duas variáveis e retorna do resultado da operação y. A seguir é mostrado como chamar a função no código principal.

```
#Usando a função
AR = alongamento(1.84, 0.896)

#Mostrando resultados
print(AR)
3.78
```

2.2 Funções compostas

De forma simples podemos ter uma função dentro do escopo de uma outra função caso seja necessário. Tal técnica é muito poderosa para solução de problemas complicados e facilita bastante a manutenção do código por terceiros. Vamos implementar um função que calcule a coeficiente de arrasto induzido C_{Di}, *(Miranda pag 89 equação 2.48)*.

$$C_{Di} = \frac{C_L^2}{\pi.e.AR} \tag{2.1}$$

Percebemos como já possuímos a uma função que calcula o alongamento (AR). Logo, para implementar uma solução eficiente precisamos desenvolver uma função *C_Di* que chame a função alongamento. Segue abaixo a aplicação.

```
def alongamento(b, S): #argumentos
    #Onde a função faz
    y = b**2/S
    return y  #o que ela retorna

def C_Di(CL, b, S):
    e = 0.8 #simplificação
    y = CL**2 / np.pi*e*alongamento(b, S)
    return y
```

É interessante destacar, a linha 6, que como a função alongamento esta dentro da função *C_Di* a função *C_Di* precisa dos seus argumentos *CL* mais os argumentos de alongamento *(b, S)*. Usando a função *C_Di.*

```
def alongamento(b, S): #argumentos
    #Onde a função faz
    y = b**2/S
    return y  #o que ela retorna

def C_Di(CL, b, S):
    e = 0.8 #simplificação
    y = CL**2 / np.pi*e*alongamento(b, S)
    return y

#Usando a função
Cdi = C_Di(1.2, 1.84, 0.896)
print(Cdi)
0.1516

#Erro da variável local
print(e)
NameError: name 'e' is not defined
```

Inicialmente, pode parecer que programar utilizando funções causa mais trabalho, contudo durante a utilização do programa as vantagens começam a aparecer. Por exemplo, agora é possível encontrar qualquer valor de arrasto induzido em qualquer asa somente utilizando uma linha de código.

A linha 20 mostra o erro causado por chamar uma variável local (atribuída dentro de uma função) pelo programa principal, qualquer variável não retorna não poderá ser acessada pela programa principal. Para resolver esse problema e mais outros foi desenvolvida a programação orientada a objeto.

Capítulo 3

Classes e objetos

Introduction

- ❏ *Classe e objetos*
- ❏ *Atributos e métodos*
- ❏ *Atributos da classe e do objeto*
- ❏ *Construtores*

Como foi visto anteriormente podemos fazer inúmeras funções para resolver um determinado problema, mas somos engenheiros queremos deixar as coisas mais simples. E se fosse possível criar uma estrutura que consiga guarda algumas informações do avião, por exemplo *(S e b)*, e substituir automaticamente os valor próprios do avião e deixar que o usuário insira somente os dados necessários. Sabemos que usuários são burros e vão quebrar o programa de algum modo.

Essa estrutura existe e é chamada de classe. vejamos como programar sua primeira classe.

```
class avião:
    def __init__(self, S, b):
```

Toda classe é iniciada com a palavra especial class seguido pelo nome da classe. Em seguida, na linha 2 a "função"*__init__* é obrigatória e é chamada toda vez que um objeto é criado. Todos as "variáveis"que se desejam que o computador guarde na memória do objeto precisa esta escrita como *self.variável*. Vamos fazer isso na classe.

```
class avião:
    def __init__(self, S, b):
        self.S = S
        self.b = b
```

O que fizemos agora é atribuir o valor de *S* para a variável *self.S* fazemos isso pois como já discutido anteriormente toda variável fora da função não é enxergada pelo programa principal.

Quando escrevemos *self.S* estamos falando pro python identificar a variável como especial e reservar um endereço de memória próprio pra ela assim ela poderá ser usada em qualquer lugar da classe avião. Esse truque parece trivial, mas abre um leque de possibilidades pra solução de problemas complexos e se tornou a base de todos os sistemas de informática rodando no mundo hoje.

Definition 3.1 (nomeclaturas)

Até aqui alguns nomes foram usados incorretamente. Vamos definir algumas nomeclaturas fundamentais sobre programação orientada a objeto. Toda "variável"dentro da classe é chamada de ***atributo*** *e toda "função"dentro da classe é chamada de* ***método****. Todo método criado dentro da classe precisa começa com self pois sempre o primeiro parâmetro de qualquer método precisa ser o objeto que se deseja modificar (isso é feito internamente pelo computador). Criar um novo objeto a partir de uma classe é chamado de* ***instância*** *da classe.*

♣

Agora vamos implementar as duas funções desenvolvidas acima e vermos como a programação orientada a objeto facilita a nossa vida. Começaremos implementando a função alongamento.

```
class avião:
    def __init__(self, S, b):
        self.S = S
        self.b = b

    def alongamento(self):
        y = self.b**2 / self.S
        return y
```

É notável a semelhança entre a função alongamento e o método alongamento. Basicamente mostrando que se você consegui programar uma função programar uma classe é quase que um passo natural.

Vamos com entendendo cada linha do método. A linha 6 mostra a assinatura do método. A única coisa de diferente em relação a função desenvolvida anteriormente é que não temos os argumentos (b e S) onde aconteceram com eles ? Eles foram passados automaticamente pela classe para o método pois guardamos eles como a palavra reservada *self.variável*. Por isso dentro do método nos referimos as variáveis S e b como atributos *self.S* e *self.b*. Assim, só precisamos instanciar o objeto e chamar o método. Exemplo abaixo.

```
#Instanciando o objeto
objeto_avião = avião(0.896, 1.84)
#chamando o método alongamento
AR = objeto_avião.alongamento()
print(AR)
3.78
```

Agora vamos implementar a função *C_Di* como método da classe avião. Faremos isso deixando como argumento somente a variável *CL* e deixando *Se b* a cargo do objeto passar para o método (utilizando *self*).

```
class avião:
    def __init__(self, S, b):
        self.S = S
        self.b = b

    def alongamento(self):
        y = self.b**2 / self.S
        return y

    def C_Di(self, CL):
        e = 0.8 #simplificação
        y = CL**2 / np.pi*e*self.alongamento()
        return y
```

Exemplo de utilização da classe com os dois métodos implementados.

```
#Instanciando o objeto
objeto_avião = avião(0.896, 1.84)

#Chamando o método alongamento
AR = objeto_avião.alongamento()

#Mostrando o valor
print(AR)
3.78

#Chamando o método C_Di
Cdi = objeto_avião.C_Di(1.2)

#Mostrando o valor
print(Cdi)
0.1516
```

Capítulo Exercício

1. Determine o AR, alongamento da asa, com programação orientada a objeto.

$$AR = \frac{b^2}{S} \tag{3.1}$$

Dados: $b = 2.48m$ *e* $S = 0.90m^2$

Algoritmo para programação

- Crie uma classe 'avião'.
- Crie o construtor da classe que receba 2 parametros (S, b).
- AR é um atributo da classe e deve ser calculado no construtor.

Resp: $AR = 6.83$

```
import numpy as np
import matplotlib.pyplot as plt

#Classe
class avião:
    def __init__(self, S, b): #Construtor da classe
        self.S = S
                        #Atributos
        self.AR =
```

2. Construa um método pra calcular a velocidade de estol utilizando a classe avião.

$$V_{estol} = \sqrt{\frac{2.W}{\rho.S.C_{L_{max}}}} \tag{3.2}$$

Dados: $\rho = 1.225kg/m^3, b = 2.48m, S = 0.90m^2, C_{Lmax} = 1,65, W = 150N$

Algoritmo para programação

- Crie uma classe 'avião'.
- Crie o construtor da classe que receba 3 parametros $(S, b, C_{L_{max}})$.
- AR é um atributo da classe e deve ser calculado no construtor.
- Implementar um método pra calcular a V_{stol} tendo como argumento do método $We\rho$

Resp: $V_{estol} = 12.84m/s$

```
#Classe
class avião:
    def __init__(self, S, b, CLmax): #Construtor da classe
        self.S = S
                        #Atributos da classe
        self.AR =

    #Método da classe abaixo

    #Método velocidade de estol
    def v_Stol(self, W, rho): # Argumentos do método (W e rho)
         pass
```

3. Implemente 3 novos métodos dentro da classe avião. Um para calcular a força de sustentação L, outro o arrasto D e para calcular o efeito solo ϕ.

$$L = \frac{1}{2}.\rho.v^2.S.C_L \tag{3.3}$$

$$D = \frac{1}{2}.\rho.v^2.S.(C_{D0} + \phi.KC_L^2) \tag{3.4}$$

$$\phi = \frac{(16h/b)^2}{1 + (16h/b)^2} \tag{3.5}$$

Calcule o efeito solo ϕ, força de sustentação L e força de arrasto D. Para velocidade de $15m/s$, $C_L = 1.5$ e $\rho = 1.225kg/m^3$.
Dados: $b = 2.48, S = 0.90m^2, C_{Lmax} = 1.65, C_{D0} = 0.022, h = 0.35m$
Algoritmo para programação

- Adicione 2 novos atributos no construtor C_{D0} e h.
- Já será calcula pra vc a constante K no construtor.
- Implemente: efeito_Solo(self) um método que calcula (ϕ) e busca a altura h como atributo do avião (self.h).
- Implemente: sustentação(self, V, CL, rho) um método que calcula (L) com 3 argumentos de entrada. Velocidade, Cl qualquer, densidade do ar.
- Implemente: arrasto(self, V, CL, rho) um método que calcula (D) com 3 argumentos de entrada. Velocidade, Cl qualquer, densidade do ar.

Resp: $D = 17.22N, L = 186.05, \phi = 0.843$

```
#Classe
class avião:
    def __init__(self, S, b, CLmax, C_D0, h): #Construtor da classe
        self.S = S
                      #Atributos da classe

        self.AR = pass
        self.K = 1/(np.pi*0.75*self.AR)

    #Método da classe
    def v_Stol(self, W, rho): # Argumentos do método (W e rho)
        pass

    def efeito_Solo(self):  # Sem argumentos
        pass

    def sustentação(self, V, CL, rho):
        pass

    def arrasto(self, V, CL, rho):
        pass
```

4. *(Exemplo 4.9 - Pag 214. Miranda)* Determine o C_L ideal para se garantir uma decolagem com o menor

comprimento de pista possível, conhecido como (C_{LLO}).

$$C_{LLO} = \frac{\pi.e_0.AR.\mu}{2\phi} cruz \tag{3.6}$$

Dados: $b = 2.48m$, $S = 0.90m^2$, e_0 = 0.717, $C_{Lmax} = 1.65$, $\mu = 0,030$, $h = 0.35m$ e $C_D = 0.022$
Resp: $C_{LLO} = 0.276$

```
#Classe
class avião:
    def __init__(self, S, b, CLmax, C_D0, h): #Construtor da classe
        self.S = S
                        #Atributos da classe
        self.AR =
        self.K = 1/(3.14*0.75*self.AR)

    #Método da classe
    def v_Stol(self, W, rho): # Argumentos do método (W e rho)
        pass

    def efeito_Solo(self):
        pass

    def sustentação(self, V, CL, rho):
        pass

    def arrasto(self, V, CL, rho):
        pass

    def CLL0_Ideal(self, mi):
        phi = self.efeito_Solo()
        pass
```

5. *(Exemplo 4.10 - Pag 217. Miranda)* Determine o comprimento de pista S_{lo} necessário para a decolagem com um peso total de $150N$ em uma pista localizada ao nível do mar ρ = 1,225kg/m^3.

$$S_{lo} = \frac{1.44W^2}{gS\rho C_{Lmax}[T - [D + \mu(W - L)]]_{0.7V_{lo}}} \tag{3.7}$$

Dados: $b = 2.48, S = 0.90m^2, C_{Lmax} = 1.65, C_{D0} = 0.12, h = 0.35m, \mu = 0.03$
Algoritmo para programação

- Implemente: distancia_decolagem(self, W, rho) um método que calcula a distância de decolagem (Equação acima).

Algoritmo de para o método

- Considere tração disponível $T = 25N$
- Encontre a velocidade de decolagem $V_{lo} = 1.2V_{estol}$
- Calcule o C_{LLO} para substituir em L e D.
- Calcule L e D na velocidade igual a $0.7V_{lo}$.
- Por fim, calcule S_{lo}

Resp: $S_{lo} = 65.85m$

```
#Classe
class avião:
    def __init__(self, S, b, CLmax, C_D0, h): #Construtor da classe
        self.S = S
                        #Atributos da classe

        self.AR =
        self.K = 1/(3.14*0.75*self.AR)

    #Método da classe
    def v_Stol(self, W, rho): # Argumentos do método (W e rho)
        pass

    def efeito_Solo(self):
        pass

    def sustentação(self, V, CL, rho):
        pass

    def arrasto(self, V, CL, rho):
        pass

    def CLLO_Ideal(self, mi):
        phi = self.efeito_Solo()
        pass

    def distancia_decolagem(self, W, rho, mi):
        #Tração constante
        T = 33.207

        #Calculando velocidade de decolagem
        vlo = self.v_Stol(W, rho)*1.2

        #Calculando a força de sustentação
        L = self.sustentação(0.7*vlo, CLLO, rho)

        #Calcule a distância percorrida
        slo =
        return slo
```

3.1 Solução dos problemas

```
#Classe
class avião:
    def __init__(self, S, b, CLmax, C_D0, h): #Construtor da classe
        self.S = S
        self.b = b          #Atributos da classe
        self.CLmax = CLmax
        self. C_D0 = C_D0
        self.h = h

        self.AR = (self.b**2)/self.S
        self.K = 1/(3.14*0.75*self.AR)

    #Método da classe
    def v_Stol(self, W, rho): # Argumentos do método (W e rho)
        V_stol = (2*W/(rho*self.S*self.CLmax))**0.5
        return V_stol

    def efeito_Solo(self):
        zw = 16 * self.h /self.b
        phi = (zw ** 2) / (1 + zw ** 2)
        return phi

    def sustentação(self, V, CL, rho):
        L = 0.5 * (V ** 2) * rho * self.S * CL
        return L

    def arrasto(self, V, CL, rho):
        D = (0.5* (V ** 2)* rho* self.S* (self.C_D0 + self.efeito_Solo() * self.K * CL ** 2))
        return D

    def CLL0_Ideal(self, mi):
        cll0 = np.pi*0.717*self.AR*mi/(2*self.efeito_Solo())
        return cll0

    def distancia_decolagem(self, W, rho, mi):
        T = 33.207

        vlo = self.v_Stol(W, rho)*1.2
        CLLO = self.CLL0_Ideal(mi)

        L = self.sustentação(0.7*vlo, CLLO, rho)
        D = self.arrasto(0.7*vlo, CLLO, rho)

        slo = (1.44*(W**2))/(9.81*self.S*rho*self.CLmax*(T - (D + mi*(W - L))))
        return slo
```

Capítulo 4

ZebraLib

Introduction

- *Instância de uma classe (objeto)*
- *Modificando atributos do construtor*
- *Métodos e atributos da classe Airplane*
- *Métodos especiais da classe Airplane*

4.1 Instanciando e interagindo com o objeto

Instanciando o objeto.

```
#importando bibliotecas
import ZebraLib as zb

#Instanciando o objeto 'padrão'
avião_1 = zb.Airplane()
```

Instanciando o objeto e modificando os atributos do objeto. Por exemplo, vamos alterar o nome, S e Af.

```
#importando bibliotecas
import ZebraLib as zb

#Instanciando o objeto e modificando alguns atributos.
avião_2 = zb.Airplane(name='Avião modificado', S=0.950, Af=0.550)
```

Interagindo com os objetos

```
#Vendo os atributos dos objetos
>>> Avião_1.S
>>> 0.846
>>> Avião_1.AR
>>> 4.10
>>> Avião_2.S
>>> 0.950
>>> Avião_2.AR
>>> 3.64
```

Detalhamentos de todos os argumentos do construtor e seus valores default da classe Airplane.

```
#Importando a biblioteca
import ZebraLib as zb

#Observando os argumentos do construtor da classe
Avião = zb.Airplane(
        name="Zb default",

        #AERODINÂMICA
        b=1.86,                         #Envergadura[m]
        S=0.843,                        #Área da asa [m^2]
        Af=0.601,                       #Afilamento
        CLn=1.080,                      # Coeficiente de sustentação com ângulo de incidencia
        CLmax=1.5193,                   # Coeficiente de sustentação máximo da aeronave, Cl de estol
        c=0.463,                        # *Corda média aerodinamica [m]
        ao=5,                           # *Ângulo de incidência da asa
        a_max=13,                       # *Ângulo mximo da asa (ângulo de estol)

        #ESTABILIDADE E CONTROLE
        St=0.2,                         # *Área da do profundor [m^2]
        CLt=1.2,                        # *CL do profundor
        zt=0,                           # *Altura do profundor (tail)
        xt=1.2,                         # *Distância entre o centro aerodinamico do profundor pro CG

        #DESEMPENHO
        Tc=(-0.001, -0.225, 35.225), # Coeficientes da curva de tração disponível dinâmica
        alt_Default=1212,               # Altitude-densidade que esta o avião por padrão
        rho_ISA=1.14,                   # Densidade do ar da international standard atmosphere
        mi=0.038,                       # Coeficiente de atrito com o solo
        Swet=10,  # ~C_D0=0.08          # Área molhada da aeronave

        #ESTRUTURAS
        Nmax=2.1,                       # Fator de carga estrutural máximo
        z=0.22,                         # Altura do avião
        mv=2.8,                         # Massa vazia [kg]
        Load=5.5,                       # Carga paga [kg]
        I=1,                            # *Momento de inercia do avião
        xw=0,                           # *Distância do centro aerodinamico em relação ao CG
        xm=0.25,                        # *Distância da bequilha em relação ao CG (centro gravidade)
        xn=0.15,                        # *Distância do trem de pouso em realçao ao CG
    )
```

Existem 3 maneiras de instanciar a classe Airplane. O primeiro modo e mais fácil pode-se instanciar o avião modificando os atributos default, contudo quando se deseja modificar vários parâmetros é um tanto quanto trabalhoso digitar todos os parâmetros por isso podemos instânciar a classe usando um arquivo excel do histórico da equipe ou um arquivo do seu computador. No segundo modo, podemos instanciar a classe utilizando um avião do histórico da equipe (todo final de ano os dados do ano são gravados no arquivo excel). Pelo terceiro modo, podemos instanciar o avião usando um próprio arquivo excel modificado ,dessa forma, podemos modificar os dados do avião pelo excel e a classe se encarregará de importar de criar uma instancia da classe. A seguir segue os exemplos:

```
import ZebraLib as zb

#0 modo - Instanciando normalmente s/ modificar
avião = zb.Airplane()

#1 modo - Instanciando normalmente modificando
avião = zb.Airplane(b=2, S=1.95)

#2 modo - Instanciar pelo histórico da equipe
avião = zb.Airplane(file='2021')

#3 modo - Instanciar por um arquivo excel
avião = zb.Airplane(file='C:/zebra/arquivos/avião_modificado.xlsx')
```

É importante ressaltar que o arquivo excel precisa seguir o padrão do arquivo 'Avioes_historicos' dentro do repositório zebralib, onde as colunas são os anos e as linhas são os parâmetros do avião.

4.2 Atributos da classe Airplane

Os atributos, para o caso específico da classe avião, são as propriedades principais do avião. Atributos podem ser acessados utilizando a sintaxe *objeto.Atributo* exemplo: *avião.Af*.

```
#Atributos introduzidos
self.b        # Envergadura [m]
self.S        # Área da asa [m^2]
self.Af       # Afilamento
self.CLn      # Coeficiente de sustentação com ângulo de incidencia ou natural
self.CLmax    # Coeficiente de sustentação máximo da aeronave, Cl de estol
self.Swet     # Área molhada do avião [m^2]
self.c        # Corda média [m]
self.ao       # Ângulo de incidência da asa
self.a_max    # Ângulo maximo da asa (ângulo de estol)
self.Nmax     # Fator de carga estrutural máximo
self.mv       # Massa vazia da aeroave [Kg]
self.Nmax     # Fator de carga máximo estrutural do Diagrama Vn != turn_Nmax
self.z        # Altura da asa em relação ao chão [m]
self.xm       # Distância da bequilha em relação ao CG (centro gravidade)
self.xn       # Distância do trem de pouso em realçao ao CG
self.xw       # Distância do centro aerodinamico em relação ao CG
self.I        # Momento de inercia do avião

#Atributos calculados internamente
self.AR           # Alongamento ou aspect ratio (AR) na asa retangular
self.e_0          # Fator eficiênte de Oswald
self.K            # cte de proporcionalidade
self.M            # Massa total da aeronave [kg]
self.W            # Peso total do avião     [N]
self.rho_Default  # Cálculo densidade do ar padrão normalmente para SJC
self.C_D0         # Coef de arrasto parasita (coef zero-lift)
self.C_D          # C_D total para o Clmáx --> Cd = C_D0 + K.CLmax^2
self.WS           # Relação crítica W/S
self.TW           # Relação crítica T/W
self.Eficmax      # Eficiência aeronâmica maxíma (L/D)máx
self.Clmax_Alc    # Calcula o CL de máximo alcance
self.Clmax_Aut    # Calcula o CL de máxima autonomia
self.CLLO         # CL que garante menor comprimento de pista durante a decolagem.
self.Vstall       # Calcula veloc de estol para (rho de SJC), output (Vestol)
self.Vlo          # Velocidade de decolagem lift-off 20% > Vstall para o rho de SJC
self.Vl           # Velocidade de pouso landing 30% > Vstall para o rho de SJC
self.Kt           # Constante proporcionalidade Kt do profundor
```

4.3 Utilizando alguns métodos da classe

Distância de decolagem

```
#Criando os objetos
avião_1 = zb.Airplane()
avião_2 = zb.Airplane(S=0.7, b=1.5)

#Aplicando os métodos
avião_1.takeOff_Distance_EDO()
>>> 26.05
avião_2.takeOff_Distance_EDO()
>>> 36.00
```

Método para ver informações gerais

```
#Criando os objetos
avião_1 = zb.Airplane()

#Aplicando o método
avião_1.airplane_Info()

#Resposta do método
                                Zb default
Alt-dens [m]                             0
Densidade [Kg/m^3]                   1.225
Informações de carga              ---------
Carga_paga                             5.5
Carga_vazia                            2.8
Carga_total                            8.4
Peso_total                          82.404
Informações aerodinâmicas         ---------
S                                      0.843
b                                       1.86
Swet                                      10
C_D0                                0.115658
CLn                                     1.08
CLmax                                 1.5193
c                                       0.463
AR                                  4.103915
Af                                      0.601
CLmax_Alc                           1.090895
CLmax_Aut                           1.889486
Eficmax                             4.716024
N_max                                    2.1
Informações de velocidades        ---------
V_estol                              10.249096
V_decolagem                         12.298915
V_aproximação                       13.323824
VstruturalMax                       14.852351
Vcruzeiro                           13.367116
VmaxAlcance                         12.095276
VmaxAutonomia                        9.190422
Informações de Decolagem/subida   ---------
```

```
Distancia_decolagem(c/efeito_solo)  24.805651
Distancia_decolagem(s/efeito_solo)  26.051005
CL_distancia_min_decolagem           0.225076
Razão_max_subida                     2.726462
Angulo_razao_max_subida             10.956573
Raio_subida                         74.526602
Angulo_max_subida                    12.46573
Velocidade_angulo_max_subida        14.061112
CL_Ang_max_subida                    1.090895
Informações de aproximação/pouso    ---------
Distancia_pouso(c/efeito_solo)      87.408404
Distancia_pouso(s/efeito_solo)      81.013459
Angulo_maxAlcance                   11.971845
Razao_maxAlcance                       2.4815
Angulo_min_planeio                  11.971845
Velocidade_Angulo_min_planeio       14.796916
Raio_descida                        80.999494
Flare_altura                         1.761767
Flare_distancia                     16.801807
```

Tempo de missão (ainda é um beta)

```
#Criando os objetos
avião_1 = zb.Airplane()

#Aplicando o método
avião_1.mission_Time()

#Resposta do método
                    Zb default
Alt-dens                  0.00
Tempo de decolagem        4.05
Tempo de subida          15.72
Tempo de cruzeiro        51.02
Tempo de descida         12.09
TEMPO TOTAL              82.88
```