UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

CURSO DE ENGENHARIA MECÂNICA

Rafael Santana de Paula

DESENVOLVIMENTO DE PROGRAMA EM PYTHON PARA OTIMIZAÇÃO DE PROJETO ESTRUTURAL DE ASA DE AERONAVE RÁDIO CONTROLADA

Ilha Solteira, SP
2024

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

CURSO DE ENGENHARIA MECÂNICA

Rafael Santana de Paula

DESENVOLVIMENTO DE PROGRAMA EM PYTHON PARA OTIMIZAÇÃO DE PROJETO ESTRUTURAL DE ASA DE AERONAVE RÁDIO CONTROLADA

Trabalho de Conclusão de Curso apresentado a Faculdade de Engenharia, da Universidade Estadual Paulista - Unesp, Campus de Ilha Solteira como parte das exigências para a obtenção do título de Bacharel em Engenharia Mecânica.
Orientador: Prof. Dr. Miguel Ângelo Menezes

Ilha Solteira, SP
2024

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

P324d

Paula, Rafael Santana de.
Desenvolvimento de programa em Python para otimização de projeto estrutural de asa de aeronave rádio controlada / Rafael Santana de Paula. -- Ilha Solteira: [s.n.], 2024
73 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Mecânica) - Universidade Estadual Paulista (UNESP), Faculdade de Engenharia, Ilha Solteira, 2024

Orientador: Miguel Ângelo Menezes

Inclui bibliografia

1. Longarinas. 2. Estruturas. 3. Python. 4. Otimização. 5. Aerodesign.

Elaborado por Raiane da Silva Santos - CRB: 8/9999

ATA DE DEFESA DO TRABALHO DE GRADUAÇÂO

Aos 13 dias do mês de dezembro do ano de dois mil e vinte e quatro, as 18h30min, por videoconferência, no Departamento de Engenharia Mecânica, do Campus da UNESP, da Faculdade de Ilha Solteira, o discente **Rafael Santana de Paula**, matriculado sob o número 161051529, tendo como banca examinadora, o orientador Prof. Ph.D. Miguel Angelo Menezes, o Engenheiro Mecânico M.Sc. Vinicius de Araújo Salmazo (Doutorando DEM) e o Prof. M.Sc. Ruhan Carlos Ponce de Oliveira (AKAER) apresentou o Trabalho de Graduação intitulado: "*Desenvolvimento de Programa em Python para Otimização de Projeto Estrutural de Asa de Aeronave Rádio Controlada*", obtendo a nota final (9,7) Nove e sete.

Por ser verdade, os membros da banca examinadora e o discente assinam em seguida.

Prof. Ph.D. Miguel Angelo Menezes (Orientador)

Eng. Mec. M.Sc. Vinicius de Araújo Salmazo

Prof. M. Sc. Ruhan Carlos Ponce de Oliveira (AKAER)

Rafael Santana de Paula (Discente)

Digitalizado com CamScanner

AGRADECIMENTOS

Primeiramente, agradeço a Deus por toda a saúde e por prover todas as ferramentas e oportunidades necessárias para eu chegar nesse momento.

Agradeço a minha esposa Yasmin, por todo o amor, carinho e cuidado que teve comigo desde o início dessa trajetória. Por sempre ser minha fiel companheira para dividir as alegrias e as lutas, e por compartilhar nossas vidas.

Agradeço a minha mãe Neuracy, sogra Iracema e sogro José Carlos que sempre me incentivaram e forneceram todo o suporte necessário até aqui. Que foram pilares fundamentais nesse caminho.

Agradeço a minha avós Emília e Silvia por todas as orações, todo amor e carinho que tiveram por mim.

Agradeço aos meus irmãos Gabriel e Rodrigo por todo o carinho, amor e respeito que tem por mim.

Agradeço ao meu orientador Miguel Ângelo Menezes e a Equipe Zebra, por todo o conhecimento transmitido, todas as experiencias e desafios vividos e por toda a confiança depositada em mim.

Agradeço aos meus Amigos, especialmente aos mais próximos: Gustavo, Guilherme (Acerola), Samuel, Rafael (Jamanta), Leonardo, Leopoldo, Matheus e Pedro.

RESUMO

Este trabalho apresenta o desenvolvimento de um programa em *Python* para otimização estrutural preliminar de asas de aeronaves rádio controladas, com foco em aplicações de aerodesign. O programa automatiza etapas fundamentais do projeto estrutural, como o cálculo de cargas aerodinâmicas, esforços internos e tensões, além de permitir a otimização geométrica da estrutura da asa. Para isso, foram aplicados conceitos de aerodinâmica, métodos numéricos e critérios de resistência estrutural, utilizando a madeira balsa como material principal. A metodologia incluiu a implementação de classes e funções em *Python*, com uso de bibliotecas como *Pandas*, *Matplotlib* e *Scipy* para manipulação de dados e visualizações gráficas. Os resultados obtidos permitiram não apenas a visualização detalhada das cargas e esforços internos, como também a determinação de uma configuração otimizada da asa. A melhor relação entre resistência e massa foi alcançada com duas longarinas de 1/8 de polegada, posicionadas a 23,5% e 40% da corda na raiz da asa, resultando em uma massa total de 102,5 g. Essa configuração representa uma redução de até 59% em relação a uma estrutura com mesma posição das longarinas, mas espessuras de 1/4, e uma economia de 5% em massa comparada a um indivíduo de mesma resistência e espessuras de parede, mas com posicionamentos diferentes. Esses resultados destacam o potencial do programa em gerar estruturas leves e eficientes, fundamentais para competições de aerodesign.

PALAVRAS CHAVE: Longarinas; Estruturas; *Python*; Otimização; Aerodesign.

ABSTRACT

This work presents the development of a *Python* program for the preliminary structural optimization of radio-controlled aircraft wings, focusing on aerodesign applications. The program automates fundamental steps of the structural design process, such as the calculation of aerodynamic loads, internal forces, and stresses, while enabling geometric optimization of the wing structure. To achieve this, concepts of aerodynamics, numerical methods, and structural resistance criteria were applied, using balsa wood as the primary material. The methodology involved the implementation of classes and functions in *Python*, utilizing libraries such as *Pandas*, *Matplotlib*, and Scipy for data manipulation and graphical visualizations. The results obtained allowed for a detailed analysis of loads and internal forces and the identification of an optimized wing configuration. The best balance between strength and weight was achieved with two spars of 1/8 inch thickness, positioned at 23.5% and 40% of the chord at the wing root, resulting in a total mass of 102.5 g. This configuration represents a reduction of up to 59% compared to a structure with the same spar positions but with 1/4 inch thicknesses, and a 5% weight saving compared to an individual with the same strength and wall thicknesses but different spar positions. These results highlight the program's potential to generate lightweight and efficient structures, essential for aerodesign competitions.

KEYWORDS: Spars; Structures; *Python*; Optimization; Aerodesign.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A equipe Zebra Aerodesign, vinculada ao projeto de extensão da Faculdade de Engenharia de Ilha Solteira – Universidade Estadual Paulista "Júlio de Mesquita Filho", desenvolve aeronaves rádio controladas para participar competição latino-americana anual SAE Aerodesign. Nesta competição, destacam-se as aeronaves que carregam a maior quantidade de carga possível, enquanto minimizam o peso estrutural. Nesse contexto, o dimensionamento estrutural da asa tem um papel crítico na redução de peso, visto que esse é o maior componente da aeronave.

O projeto estrutural da asa é complexo. Ele inicia-se com a determinação de cargas e da geometria da superfície aerodinâmica da asa. A partir dessas informações, a área de estruturas deve definir as configurações da geometria interna que suportem os esforços internos atuantes e possuam uma massa reduzida. Entretanto, o prazo reduzido para a execução desses projetos frequentemente limita a possibilidade de realizar análises detalhadas da estrutura e explorar diferentes alternativas de configuração.

Nesse cenário, ferramentas computacionais que automatizem etapas do projeto preliminar tornam-se indispensáveis. Com elas é possível reduzir custos e implementar métodos de otimização estrutural. Assim, torna-se possível adotar uma configuração preliminar da geometria da estrutura baseando-se nas informações fornecidas pelo programa.

O objetivo desse trabalho é desenvolver um programa em *Python* que automatize as etapas do projeto estrutural preliminar da asa da aeronave e otimize a geometria da estrutura, buscando a melhor relação entre resistência e massa. Para isso, foram determinadas hipóteses para a simplificação do modelo em análise. Assim, a partir dos dados de *input* e das equações disponíveis na literatura, foi possível desenvolver o programa e alcançar os objetivos determinados.

A implementação dessa ferramenta busca contribuir significativamente para o projeto da equipe Zebra Aerodesign, permitindo não apenas a definição de estruturas otimizadas, como também a realização de análises adicionais, como validações experimentais ou simulações numéricas mais detalhadas. Dessa forma, o trabalho oferece uma solução prática e flexível, alinhada às necessidades da competição e aos desafios enfrentados por equipes de aerodesign.

2 OBJETIVOS

O objetivo principal desse trabalho é desenvolver um programa em *Python* que execute as principais etapas do projeto estrutural preliminar, permitindo a otimização geométrica de uma asa de aerodesign. Para isso, o programa calcula cargas aerodinâmicas, esforços internos e tensões, além de explorar configurações geométricas que minimizem a massa e maximizem a resistência estrutural da asa. A ferramenta busca atender aos critérios de segurança e desempenho exigidos em competições de aerodesign, fornecendo análises rápidas e precisas para apoiar a equipe Zebra Aerodesign no desenvolvimento do projeto.

3 REVISÃO BIBLIOGRÁFICA

O desenvolvimento deste trabalho envolve a utilização da linguagem de programação *Python* e da aplicação de conceitos de cálculo numérico e estrutural, bem como a observância de normas aeronáuticas, entre outros tópicos relevantes. Dessa forma, é fundamental explorar esses temas para garantir uma compreensão mais aprofundada e embasar adequadamente o desenvolvimento do trabalho.

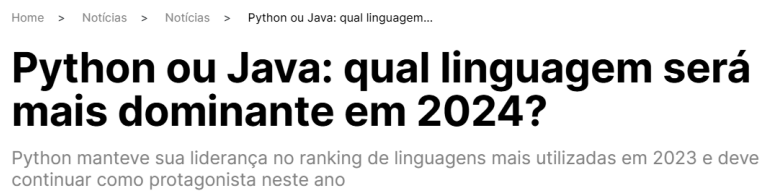
3.1 LINGUAGEM DE PROGRAMAÇÃO *PYTHON*

A linguagem *Python* é atualmente uma das mais utilizadas para análise de dados e modelagem de problemas em engenharia [1]. Seus principais destaques são:

- Curva de aprendizado rápida: A sintaxe da linguagem é muito mais simples se comparada a outras linguagens de programação. Ela evita uma quantidade excessiva de símbolos e estruturas complexas. Além disso, *Python* possibilita a execução de códigos linha a linha o que facilita o aprendizado, promove feedback imediato facilitando a experimentação e a aprendizagem iterativa.
- Extensa comunidade e material na web: A linguagem é uma das mais populares atualmente [1]. Isso possibilita um amplo suporte, grande disponibilidade de documentação e tutoriais e acima de tudo um grande compartilhamento de conhecimento.
- Ampla disponibilidade de bibliotecas e pacotes: Isso possibilita a implementação rápida e reduz trabalho repetitivo no desenvolvimento de rotinas. Ademais, as bibliotecas são uma coletânea de códigos testados e validados. O que reduz a possibilidade de *bugs*. É fundamental ainda ressaltar que as bibliotecas podem conferir maior desempenho e eficiência na execução dos códigos. Além disso, algumas bibliotecas mais populares, como *Matplotlib* e *Pandas* possuem amplo suporte da comunidade.

Para uma melhor compreensão do desenvolvimento do programa, é fundamental revisar as estruturas condicionais e laços de repetição, as classes e objetos e as bibliotecas *Pandas* e *Matplotlib*.

Figura 1 - Relevância de *Python* e Java em matéria do IT Fórum



Fonte: [1]

3.1.1 Estruturas condicionais e laços de repetição

As estruturas condicionais e de repetição são dois dos principais recursos presentes nas linguagens de programação, pois permitem que várias sequências lógicas sejam criadas nos códigos. Para esse trabalho, será abordada a estrutura condicional *if*, e o laço de repetição *for*.

A estrutura condicional *if* permite a tomada de decisão mediante a uma condição. Se a condição for verdadeira, um bloco de código é executado. Se não, é executado outro bloco. Na Figura 2, é possível verificar um exemplo de aplicação.

Figura 2 - Exemplo de aplicação da estrutura condicional *if*

```
numero = int(input("Digite um número: "))

if numero > 0:
    print("O número é positivo.")
elif numero < 0:
    print("O número é negativo.")
else:
    print("O número é zero.")
```

```
Digite um número:  2
O número é positivo.
```

Fonte: Próprio autor.

O laço de repetição *for* permite iterar sobre uma sequência e executar um bloco repetidas vezes. Um exemplo dessa estrutura é mostrado na Figura 3.

Figura 3 - Exemplo de aplicação da estrutura condicional *for*

```
for i in range(1, 11):
    print(f"Número: {i}")
```

```
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Número: 10
```

Fonte: Próprio autor

3.1.2 Classes e funções

Em *Python*, uma classe é um dos principais conceitos da programação orientada a objetos e serve como um modelo para criar objetos. Ela define um conjunto de atributos (dados) e métodos (funções) que descrevem o comportamento e o estado que esses objetos terão.

Quando uma classe é criada, estamos definindo um tipo de dado que combina dados e operações. Essa definição pode ser vista como uma "fábrica" de objetos que compartilham uma

mesma estrutura e comportamento. Cada objeto criado a partir de uma classe é chamado de instância e representa uma ocorrência específica com valores próprios para os atributos definidos pela classe.

Na Figura 4, por exemplo, é definida uma classe em *Python* denominada "*Circulo"*. Nela, o raio é dado como parâmetro e os atributos "*area"* e *"comprimento"* são calculados.

Figura 4 - Exemplo de classe em *Python*

```
class Circulo:
    def __init__(self, raio):
        self.raio = raio
        self.area = math.pi * (raio ** 2)  # Calcula a área
        self.comprimento = 2 * math.pi * raio  # Calcula o comprimento
```

Fonte: Próprio autor.

Ao se definir a classe, é possível gerar diversos objetos do tipo *"circulo"*. Cada um terá seus métodos e atributos de acordo com o parâmetro "*raio"* informado. Esse é um exemplo simples da aplicação de uma classe. Entretanto, é possível utilizar esse recurso casos mais complexos, em que mais parâmetros serão inseridos e mais métodos e atributos calculados.

As funções em *Python* podem ser definidas dentro ou fora das classes. Esse recurso é muito útil, pois permite a execução de um bloco de código repetidas vezes durante o programa, sem a necessidade de reescrevê-lo. Apenas com o fornecimento dos parâmetros é possível executar a função no código e obter seus *outputs*.

3.1.3 *Pandas*

A biblioteca *Pandas* é uma das ferramentas mais amplamente utilizadas para análise de dados em *Python* e é um componente fundamental do ecossistema de ciência de dados. *Pandas* foi projetada para fornecer estruturas de dados rápidas, flexíveis e expressivas que facilitam a manipulação e a análise de dados em *Python*.

Em *Pandas*, temos duas estruturas de dados principais: *DataFrames* e Series. Os *DataFrames* organizam-se como tabelas de bancos de dados, tendo várias linhas e colunas. Para cada coluna, existe um cabeçalho e um conjunto de dados (números, *strings*, datas, etc.). Cada linha possui um índice numérico, mas também podem receber rótulos personalizados. Essa estrutura é ideal para armazenar e manipular dados tabulares de forma organizada. As Series se diferem dos *DataFrames* apenas pelo fato de que são estruturas de dados unidimensionais, possuindo apenas uma coluna.

Dentre os principais recursos disponíveis em *Pandas*, destaca-se o suporte nativo para diversos formatos de dados, como CSV, Excel, JSON e HDF5. Isso possibilita a integração com várias fontes de dados bem como a exportação de resultados para outras extensões.

Pandas também possui um extenso portifólio de métodos para manipulação e tratamento de dados [2]. O principal objeto dessa biblioteca é o *DataFrame*, que se trata de uma estrutura de dados tabular organizado em linhas e colunas. Na biblioteca, é possível remover duplicatas, agrupar, remover linhas com valores ausentes, ordenar, inserir valores dentre outras ações, por exemplo. Alguns métodos utilizados na elaboração desse trabalho podem ser conferidos na Tabela 1.

Tabela 1 - Principais métodos da biblioteca *Pandas*

Método	**Descrição**
pd.read_excel()	Lê o conjunto de dados de uma planilha de Excel
pd.DataFrame()	Define um *DataFrame* a partir de um conjunto de dados
df["a"].sort_values	Organiza a coluna "a" do *DataFrame df* em ordem crescente
df["a"].max()	Encontra o valor máximo da coluna "a" do *DataFrame df*
df["a"].min()	Encontra o valor mínimo da coluna "a" do *DataFrame df*
df.iloc[i]	Retorna os dados da linha "i" do *DataFrame df*
df["a"]	Retorna os dados da coluna "a" do *DataFrame df*

Fonte: Adaptado [2]

Em resumo, a biblioteca *Pandas* é uma ferramenta poderosa para manipulação e análise de dados, especialmente quando utilizada em conjunto com outras bibliotecas disponíveis em *Python*. Sua integração com *Matplotlib* permite a visualização de dados de maneira prática e intuitiva, enquanto a compatibilidade com *Numpy* e *Scipy* potencializa as operações numéricas e computacionais. Dessa forma, *Pandas* se destaca como um recurso essencial para análise de dados, facilitando desde a manipulação até a visualização e processamento de informações complexas. A documentação completa a respeito da biblioteca pode ser acessada a partir de seu site na web [2].

3.1.4 *Matplotlib*

Matplotlib é a principal biblioteca utilizada para criação de visualizações de dados em *Python*, oferecendo uma vasta gama de recursos e personalizações para gráficos científicos. Na Tabela 2 são organizados alguns métodos disponíveis em seu portifólio que foram utilizados nesse trabalho.

Tabela 2 – Métodos de *Matplotlib*

Método	**Descrição**
plt.plot()	Plota um gráfico de linhas
plt.contourf ()	Plota gráfico de cores
plt.contour()	Plota gráfico de contornos

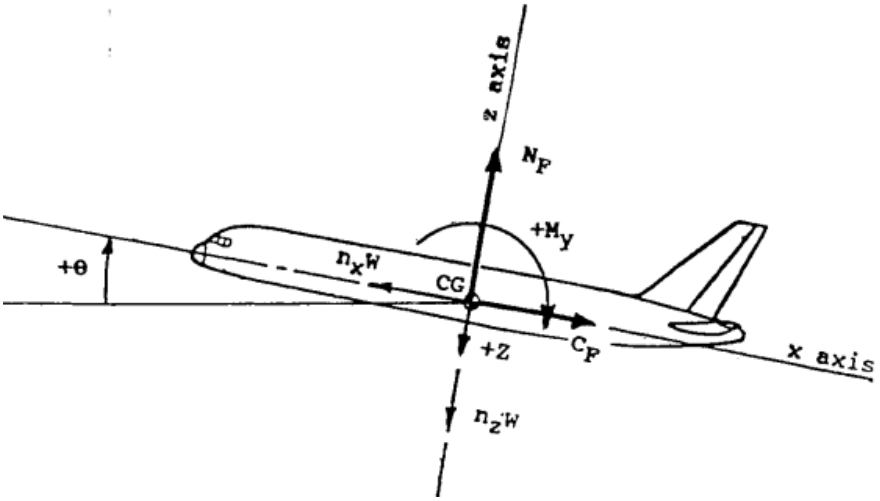
Fonte: Adaptado [3]

Para cada um desses métodos, é necessário informar seus devidos parâmetros para que seja possível gerar uma visualização dos dados. A documentação completa a respeito da biblioteca pode ser acessada a partir de seu site na web [3].

3.2 AERODINÂMICA DA ASA

O projeto aerodinâmico da asa é uma das primeiras etapas do desenvolvimento de uma aeronave. Sua finalidade consiste em dimensionar uma superfície que confira força de sustentação suficiente para que a aeronave "vença" seu peso e realize o voo. Além disso, é importante que essa superfície gere uma baixa força de arrasto, que é uma força de reação do ar, contrária ao movimento do veículo [4]. Assim, a aeronave necessitará de menor força de tração para conferir velocidade suficiente para o voo.

Figura 5- Diagrama de corpo livre em uma aeronave



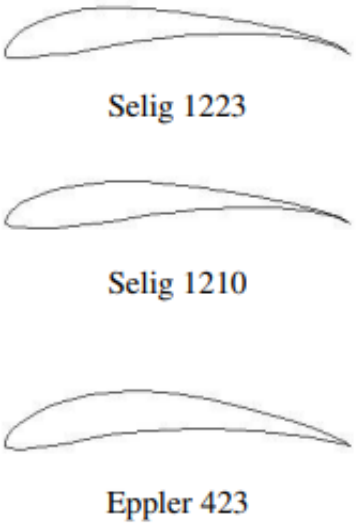
Fonte: [5]

Para esse trabalho, é fundamental determinar algumas propriedades aerodinâmicas básicas:

- Perfil aerodinâmico: Silhueta da secção transversal da asa. Sob um escoamento incidente nesse formato, é criado um gradiente de pressão e consequentemente a sustentação, o arrasto e o momento;
- Corda (c): É a distância entre o bordo de ataque e o de fuga do perfil aerodinâmico, que é medida em linha reta;
- Área Alar (S): É a área total da superfície da asa;
- Envergadura (b): É a distância entre as pontas da asa de uma aeronave;

- Ângulo de ataque (α): É o ângulo formado entre a corda da asa e a direção do vento relativo que incide sobre a asa. Um aumento de ângulo de ataque geralmente aumenta, até certo ponto, a sustentação;
- Coeficiente de Sustentação (Cl): É uma propriedade adimensional que descreve a eficiência de uma asa em gerar sustentação. É dependente das variáveis: ângulo de ataque, forma da asa, dentre outros fatores;
- Coeficiente de Arrasto (*Cd*): É um valor adimensional que descreve a resistência ao movimento da asa através do ar. Quanto menor o *Cd*, mais aerodinâmica é a asa;
- Coeficiente de Momento (*Cm*): É um valor adimensional que descreve o momento em torno do centro de gravidade da aeronave. Indica a tendência da aeronave em mudar sua atitude ou ângulo de voo. Esse momento surge em uma superfície aerodinâmica, pois a depender do ângulo de ataque, a distribuição de pressão ao longo da corda pode não ser homogênea e uniforme. Ao haver esses desequilíbrios, surge o momento aerodinâmico.

Figura 6 - Perfis aerodinâmicos



Fonte: [4]

Esse conjunto de características é fundamental para a definição da Força de sustentação *L*, da Força de arrasto *D* e do Momento *M*, dados pelas Equações 1, 2 e 3, respectivamente.

$$L = \frac{1}{2} \cdot \rho \cdot v^2 \cdot S \cdot C_l \tag{1}$$

$$D = \frac{1}{2} \cdot \rho \cdot v^2 \cdot S \cdot C_d \tag{2}$$

$$M = \frac{1}{2} \cdot \rho \cdot v^2 \cdot S^2 \cdot C_m \tag{3}$$

Sendo ρ a densidade do ar, v a velocidade relativa do ar em relação a aeronave, S a área alar da asa e C_l, C_d e C_m, os coeficientes de sustentação, arrasto e momento, respectivamente. Essas Equações podem ser reescritas para descreverem a distribuição dos carregamentos pela envergadura, como é mostrado nas Equações 4, 5 e 6, respectivamente.

$$l = \frac{1}{2} \cdot \rho \cdot v^2 \cdot c \cdot c_l \tag{4}$$

$$d = \frac{1}{2} \cdot \rho \cdot v^2 \cdot c \cdot c_d \tag{5}$$

$$m_c = \frac{1}{2} \cdot \rho \cdot v^2 \cdot c^2 \cdot c_{m\,c/4} \tag{6}$$

Sendo c_l, c_d e c_m os coeficientes C_l, C_d e C_m distribuídos pelo comprimento da envergadura. O cálculo desses coeficientes é feito computacionalmente a partir do software *XFRL5* pela área de aerodinâmica. Os valores dos coeficientes são dependentes de outras propriedades aerodinâmicas, tais como Número de Reynolds e Ângulo de Ataque. Para o cálculo estrutural, é fundamental determinar as condições de voo em que esses esforços apresentam o máximo módulo de intensidade.

3.3 DIAGRAMA V-N

O fator de carga em uma aeronave pode ser calculado a partir da Equação 7 [5].

$$n = \frac{L}{W} \tag{7}$$

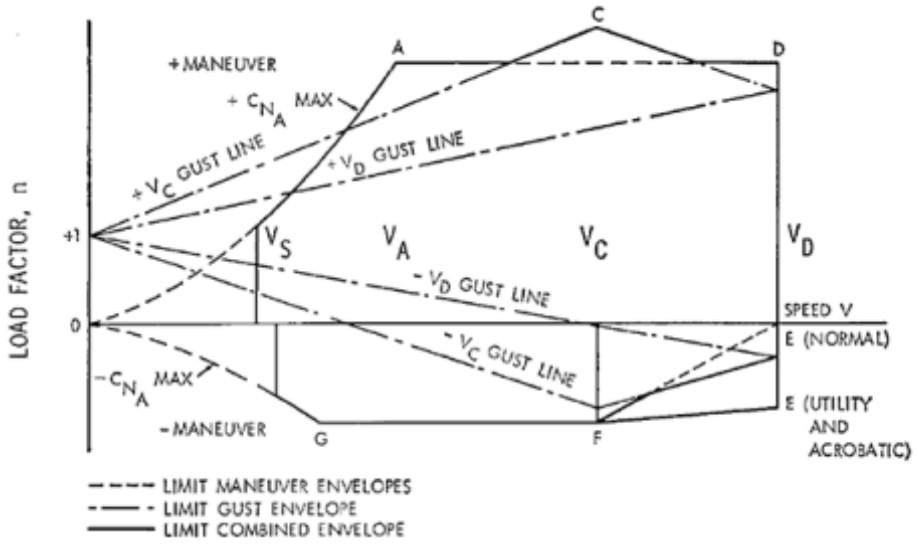
Sendo L a sustentação e W o peso da aeronave, ambos em Newtons. Conforme descrito pela norma da FAR Part 25.337 [6], dimensiona-se sua estrutura a partir de $n_{máx}$ calculado pela Equação 8, dada por:

$$n_{máx} = 2{,}1 + \frac{24.000}{(W + 10.000)} \tag{8}$$

As condições de operação e as dimensões de um aerodesign são bastante distintas de uma aeronave convencional, geralmente não é realizada a aplicação integral das normas para o desenvolvimento desse projeto. Portanto, cada equipe participante da competição adota seus critérios de projeto. Logo, é comum adotar-se valores menores de n para aeronaves dessa classe.

Ao se determinar o fator de carga máximo de operação da aeronave, é possível relacioná-lo a velocidade da aeronave por meio do diagrama V-n e o de rajada, como é mostrado na Figura 7.

Figura 7 - Diagrama V-n



Fonte: [5]

Nota-se pela Figura 7 que há alguns pontos de referência que devem ser calculados para a determinação do Diagrama. No segmento AD, temos o valor de $n_{máx}$ definido para o projeto. O segmento GF é dado pelo fator de carga máximo negativo. Pela norma da FAR 25.337 [6] é possível calcula-lo a partir da Equação 9, porém o valor obtido não deve ser menor que 1 [7].

$$n_{máx\,neg} = 0{,}4 \cdot n_{máx} \tag{9}$$

Na velocidade v_s tem-se uma condição de estol, quando a sustentação é igual ao peso do avião. Logo, temos que n = 1. Desse modo, relacionando as Equações 1 e 7 em uma condição de $C_{l\,máx}$, calculamos a velocidade de estol v_s a partir da Equação 10 [7].

$$v_s = \sqrt{\frac{2 \cdot W}{\rho \cdot S \cdot C_{l\,máx}}} \tag{10}$$

A velocidade de manobra v_a é calculada a partir do ponto em que $n = n_{máx}$, para uma condição de $C_{l\,máx}$. Portanto, analogamente a Equação 8, calculamos v_a a partir da Equação 11 [7].

$$v_a = \sqrt{\frac{2 \cdot n_{máx} \cdot W}{\rho \cdot S \cdot C_{l\,máx}}} \tag{11}$$

A norma 25.335 [5] estabelece que a velocidade de cruzeiro da aeronave não deve exceder 90% da velocidade de máximo alcance em voo nivelado, como é dado pela Equação 12 [7].

$$v_c = 0{,}9 \cdot v_{máx} \tag{12}$$

A velocidade $v_{máx}$ é fornecida pela área de Desempenho e é relacionada a máxima tração fornecida a aeronave em situação de voo nivelado.

A velocidade máxima de mergulho a não ser ultrapassada v_d é dada pela Equação 13 [7].

$$v_d = 1{,}11 \cdot v_{máx} \tag{13}$$

O valor da velocidade de estol negativa também pode ser obtido pela Equação 10. Para isso, *Cl* máximo negativo pode ser aproximado pela relação dada pela Equação 14 [7].

$$C_{l\,máx\,neg} = 0{,}6 \cdot C_{l\,máx\,neg} \tag{14}$$

O Diagrama de rajada tem por finalidade, descrever a influência das rajadas de vento no fator de carga máximo tanto para a velocidade de cruzeiro quanto para a velocidade de mergulho. Também leva em consideração as condições de voo com o “bico para baixo" e para cima. O cálculo de *n* devido as rajadas deve ser feito por meio da Equação 15 [7].

$$n = 1 \pm \frac{\frac{1}{2} \cdot v \cdot \alpha \cdot K_g \cdot U \cdot S}{W} \tag{15}$$

Sendo v a velocidade relativa da aeronave, α a variação do coeficiente de sustentação com o ângulo de ataque, *S* a área da asa, *W* o peso da aeronave, K_g é dado pela Equação 16 [7].

$$K_g = \frac{0{,}88 \cdot \mu_g}{5{,}3} \tag{16}$$

E μ_g sendo dado pela Equação 17.

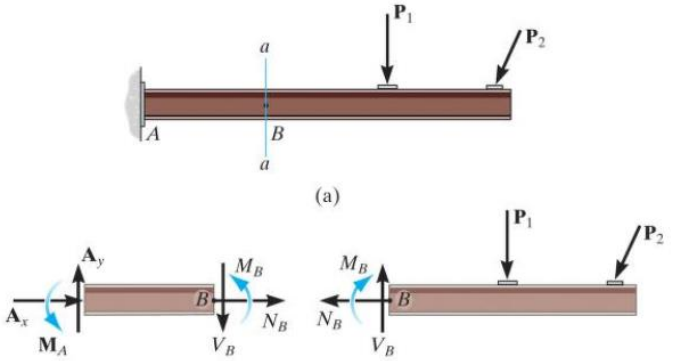
$$\mu_g = \frac{2 \cdot (\frac{M}{S})}{\rho \cdot \bar{c} \cdot \alpha} \tag{17}$$

Sendo M a massa e $\bar{c}$ a corda média da asa. *U* será a velocidade da rajada de vento. A norma FAR 25.345 define que para a velocidade v_C seu valor seja 15,24 m/s e para v_d seja 7,62 m/s. Entretanto, para um projeto em escalas reduzidas, como é o Aerodesign, pode-se assumir a metade desses valores [4].

3.4 ESFORÇOS INTERNOS

Os esforços internos são forças e momentos que surgem dentro dos elementos estruturais (como vigas, colunas e treliças) em resposta a cargas externas aplicadas. Esses esforços garantem o equilíbrio e a resistência da estrutura, distribuindo as cargas de forma a manter sua integridade. Suas intensidades podem ser determinadas a partir do método das seções [8]. Nesse método, a estrutura é seccionada transversalmente e para cada secção é feita uma análise de equilíbrio estático como é mostrado na Figura 8.

Figura 8 - Ilustração do método das secções



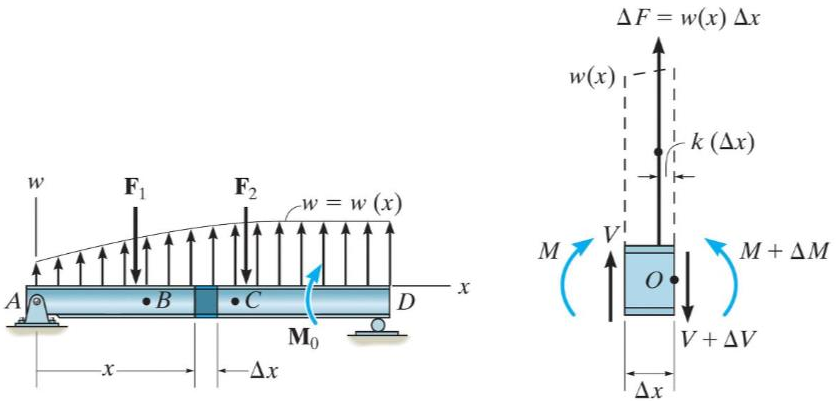
Fonte: [3]

Dado um sistema estático sob atuação de cargas externas e reações de apoio, a Força cortante *V* e o Momento fletor *M* podem ser calculados a partir das equações de equilíbrio estático dadas pelas Equações 18 e 19, respectivamente.

$$V + \sum_i F_i = 0 \tag{18}$$

$$M + \sum_i M_i = 0 \tag{19}$$

Figura 9 - Carga distribuída sobre viga



Fonte: [3]

Além disso, para sistemas submetidos a carregamentos distribuídos como o mostrado na Figura 9, é possível determinar relações que simplifiquem o cálculo do momento fletor M e da força cortante V [8]. Ao aplicar as equações de equilíbrio em um sistema de carga distribuída, é possível notar a relação mostrada na Equação 20.

$$\frac{dV}{dx} = w\,(x)$$

$$\begin{matrix}\text{Inclinação do} \\ \text{Diagrama de} \\ \text{esforço cortante}\end{matrix} \quad = \quad \begin{matrix}\text{Intensidade da carga} \\ \text{distribuída}\end{matrix} \tag{20}$$

Se a Equação 20 for reescrita na forma $dV = w(x)dx$ e for integrada entre dois pontos a e b da estrutura teremos V, como mostrado na Equação 21.

$$V = \int_a^b w\,(x)\,dx \tag{21}$$

Hibbeler [3] também destaca a relação dada pela Equação 22.

$$\frac{dM}{dx} = V\,(x) \tag{22}$$

$$\begin{matrix}\text{Inclinação do} \\ \text{Diagrama de} \\ \text{Momento Fletor}\end{matrix} \quad = \quad \begin{matrix}\text{Intensidade da Força} \\ \text{Cortante}\end{matrix} \tag{23}$$

Analogamente a Equação 21, por integração temos o momento fletor M dado pela Equação 24.

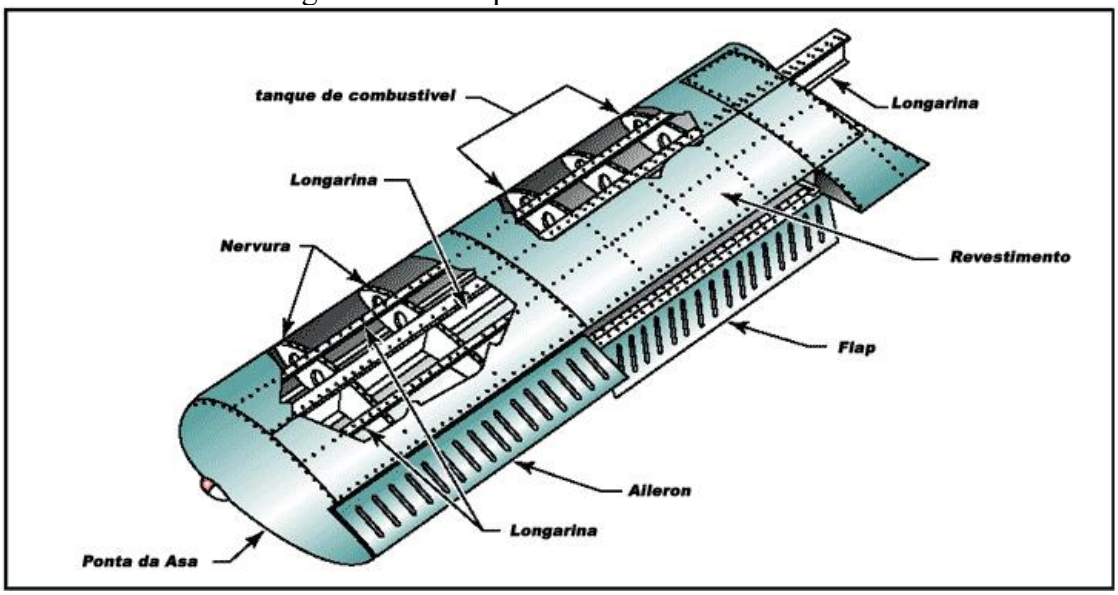
$$M = \int_a^b V\,(x)\,dx \tag{24}$$

A determinação dos esforços internos é de suma importância para o cálculo estrutural pois as tensões atuantes estão diretamente relacionadas com suas intensidades, como será abordado na próxima secção. Logo, em um projeto estrutural é fundamental estimar as cargas de forma criteriosa e de acordo com as normas disponíveis para que os dimensionamentos sejam executados de forma eficiente.

3.5 COMPONENTES E CÁLCULO DE TENSÕES EM ASAS DE AERONAVES

Os principais elementos que compõem a asa de uma aeronave são ilustrados na Figura 10.

Figura 10 – Componentes estruturais da asa.



Fonte: [9]

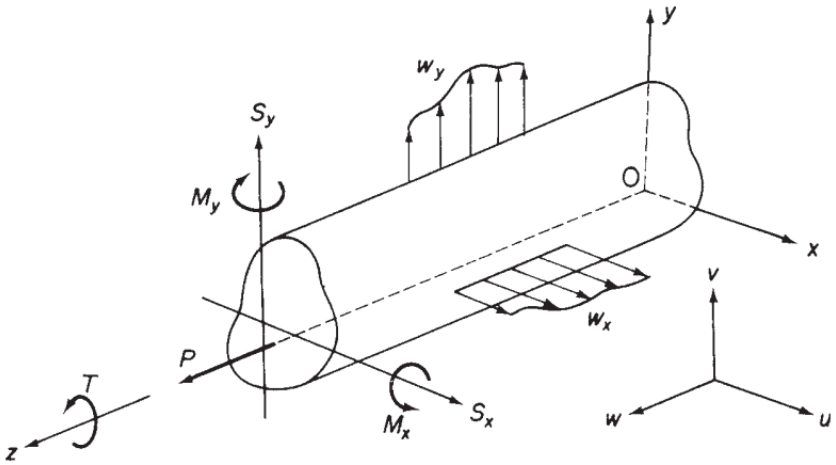
Sendo os principais:

- **Longarinas:** São vigas longitudinais que se estendem de uma extremidade da asa à outra. Normalmente, uma asa possui uma ou mais longarinas, e elas são projetadas para suportar tanto a carga de flexão quanto a carga de torção que ocorrem durante o voo. A longarina é essencial para manter a integridade estrutural da asa e garantir a distribuição das forças ao longo da estrutura;
- **Revestimento:** É a camada externa que cobre a estrutura interna da asa. Além de criar uma superfície aerodinâmica suave para minimizar a resistência ao ar, o revestimento protege os componentes internos da asa e também contribui para a resistência estrutural da asa. Combinado com as longarinas, podem gerar as "caixas de torção" e incrementar significantemente a rigidez a torção da asa.
- **Nervuras:** As nervuras são estruturas transversais que mantem a forma aerodinâmica ao revestimento. Elas são distribuídas perpendicularmente ao longo da envergadura e distribuem as cargas aerodinâmicas do revestimento para as longarinas, contribuindo para a rigidez e a resistência da asa. Geralmente, seu formato segue a silhueta do perfil utilizado no projeto da asa.

Para projetos de aerodesign, são realizadas algumas simplificações de projeto como por exemplo o tanque de combustível não estar alocado na asa, pode ou não haver flaps e o revestimento é parcialmente estrutural.

Dado que as aeronaves são projetadas para possuir o menor peso possível, essas estruturas geralmente apresentam paredes finas. Para esse trabalho abordaremos o cálculo de tensões normais e cisalhantes de torção [10].

Figura 11 - Sistema estático submetido a carregamentos



Fonte: [10]

Considerando um sistema submetido a flexão como mostrado na Figura 11, podemos calcular a tensão σ devido a flexão em qualquer ponto dessa secção a partir da Equação 25 [10].

$$\sigma_z = \frac{M_x(I_{yy}y - I_{xy}x)}{I_{xx}I_{yy} - I_{xy}^2} + \frac{M_y(I_{xx}x - I_{xy}y)}{I_{xx}I_{yy} - I_{xy}^2} \tag{25}$$

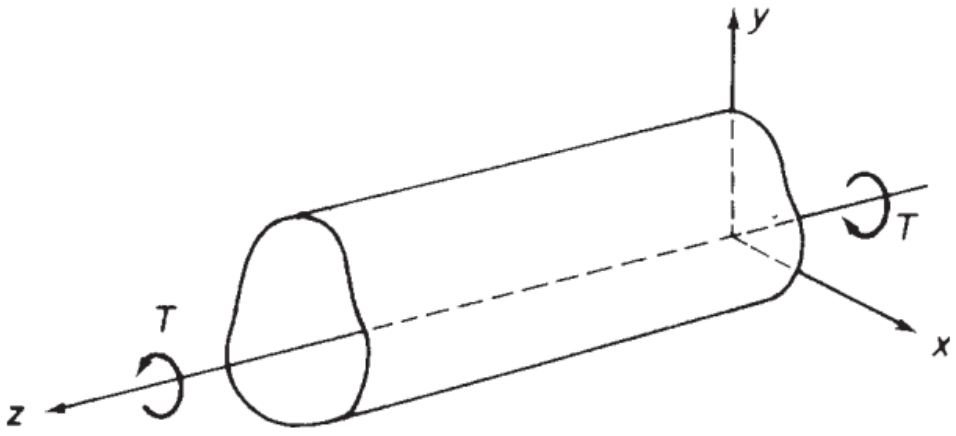
Onde I_{xx}, I_{yy}, I_{xy} são os momentos de inércia de área da secção transversal em análise, M_x e M_y são os momentos fletores atuantes na respectiva secção e, finalmente, x e y as distâncias do ponto em análise em relação ao centroide.

Considerando uma estrutura de secção transversal fechada e paredes finas, como mostrado na Figura 12, é possível calcular o cisalhamento dado pela Equação 26 [10].

$$q = \frac{T}{2A} \tag{26}$$

Sendo q o fluxo de cisalhamento, T o momento devido a torção e A é a área da secção transversal. Ao obter o fluxo de cisalhamento da secção, é possível calcular a tensão de cisalhamento simplesmente dividindo-se o valor obtido pela espessura da parede da estrutura em questão.

Figura 12 - Sistema estático submetido a torção



Fonte: [10]

A partir das equações apresentadas, nota-se que o cálculo de tensões é dependente dos momentos de inércia, centroide e área da secção transversal. Portanto, faz-se necessário abordar as equações que definem essas propriedades geométricas, visto que serão fundamentais para a compreensão e andamento do desenvolvimento do trabalho.

3.6 CENTRÓIDE

O centróide de uma secção transversal é o ponto médio de toda a área dessa figura. Se há várias Figuras de área A_i em uma secção transversal em posições x_i e y_i, para se calcular o centroide (x_c, y_c) utiliza-se as Equações 27 e 28.

$$x_c = \frac{\sum_{i=1}^{n} A_i \cdot x_i}{\sum_{i=1}^{n} A_i} \tag{27}$$

$$y_c = \frac{\sum_{i=1}^{n} A_i \cdot y_i}{\sum_{i=1}^{n} A_i} \tag{28}$$

O cálculo do centróide é de suma importância para o cálculo de tensões pois ele definirá o ponto que será referência para o cálculo de tensões nos demais pontos da secção transversal. A linha neutra é definida pelo conjunto de centróides de todas as secções transversais.

3.7 MOMENTO DE INÉRCIA

O momento de inércia de área pode ser definido como uma medida de distribuição de área em torno de um eixo. Quanto maior for seu valor, maior será a resistência dessa secção a flexão. O cálculo do momento de inércia é dado pela Equação 29 [8].

$$I = \int_A y^2 \, dA \tag{29}$$

Sendo y a distância entre o ponto e o centroide do elemento diferencial e dA o diferencial de área. Para secções transversais com diversas figuras geométricas distintas, é possível calcular o momento de inércia a partir da Equação 30 [8].

$$I = \sum_i A_i \cdot y_i^2 \tag{30}$$

Sendo A_i a área de cada uma das figuras e y_i a distância entre o centroide de cada figura e o centroide da secção resultante transversal. Essa Equação pode ser utilizada para realizar o cálculo do momento de inércia de secções transversais complexas, por meio de aproximações numéricas. Para secções transversais retangulares, resolução da integral da Equação 29 nos fornece a Equação 31 [8].

$$I = \frac{b \cdot h^3}{12} \tag{31}$$

Para secções transversais complexas compostas de várias formas geométricas *i* simples, utiliza-se o Teorema dos Eixos paralelos para o cálculo do momento de inércia resultante [8]. Para isso, é utilizada a Equação 32 [8].

$$I_c = \sum_{i=1}^{n} (I_i + A_i \cdot d_i^2) \tag{32}$$

Sendo I_c o momento de inércia resultante, I_i o momento de inércia, A_i a área da figura *i* e d_i a distância do centroide da figura até o centroide da secção transversal. É importante ressaltar que o momento de inércia é calculado em função da distância da geometria em relação a um eixo de referência. Portanto, para uma geometria em um sistema bidimensional *xy*, é possível calcular o momento de inércia em relação aos seus dois eixos, sendo em relação a *x* denotado por I_{xx} e *y* por I_{yy}.

3.8 MADEIRA BALSA: PROPRIEDADES MECÂNICAS E CRITÉRIO DE FALHA DE TSAI-HILL

O principal material utilizado na fabricação de asas de aeromodelos e aerodesigns é a madeira balsa. Esse material se destaca por ser extremamente leve e possuir boa resistência mecânica [11]. Além disso, apresenta ótimas propriedades de aderência com resinas epóxi, colas ciano acrilato e revestimentos poliméricos como o monokote. Além disso, é um material composto de fibras e uma matriz natural que as une e gera uma estrutura resistente e leve. Devido a essa estrutura, suas propriedades mecânicas dependem de suas direções ortogonais, podendo ser tratado como um material ortotrópico. As propriedades mecânicas da madeira balsa, estão organizadas na Tabela 3.

Tabela 3 – Propriedades mecânicas da madeira balsa

Propriedade	**Valor**
XT - Resistência a tração paralelo a fibra	10,12 MPa
YT - Resistência a tração transversal a fibra	0,82 MPa
XC - Resistência a compressão paralelo a fibra	8,05 MPa
YC - Resistência a compressão transversal a fibra	0,707 MPa
S_{xy} - Resistência a cisalhamento paralelo a fibra	1,35 MPa
S_{yz} - Resistencia a cisalhamento transversal a fibra	1,35 MPa
σ_{balsa} – *Densidade*	96 kg/m³

Fonte: Adaptado de [10]

Devido as características mecânicas e físicas da madeira balsa, ela é classificada como um material compósito natural. Para a análise de falha desse tipo de material, é possível utilizar a equação generalizada do critério de falha Tsai-Hill. Esse modelo é uma extensão do critério de Von Mises adaptado para materiais compósitos anisotrópicos [13]. Nele, são consideradas as tensões principais atuantes e suas interações. O critério leva em consideração ainda a natureza direcional dos compósitos. A expressão que define esse modelo é dada pela Equação 33.

$$\left(\frac{\sigma_1}{X_T}\right)^2 - \left(\frac{\sigma_1\sigma_2}{{X_T}^2}\right) + \left(\frac{\sigma_2}{Y_T}\right)^2 + \left(\frac{\sigma_3}{Z_T}\right)^2 + \left(\frac{\tau_{12}}{S_{12}^2}\right)^2 \leq 1 \tag{33}$$

Sendo:

- σ_1 a tensão normal na direção *x*;
- σ_2 a tensão normal na direção *y*;
- σ_3 a tensão normal na direção *z*;
- τ_{12} a tensão de cisalhamento no plano *xz*;

O critério de falha de Tsai-Hill é amplamente utilizado devido sua simplicidade computacional boa aproximação para prever falhas em compósitos laminados sob carregamentos complexos.

Para projetos estruturais que envolvem fatores de segurança FS, os termos X_T, Y_T, Z_T e S_{12} são substituídos pelas tensões admissíveis de projeto. O cálculo é feito como o exemplo da Equação 34.

$$X_{adm\,T} = \frac{X_T}{FS} \tag{34}$$

Para projetos aeronáuticos, por exemplo é comum utilizar-se de FS = 1,5 de acordo com as normas da FAR Part 23 [5].

3.9 MÉTODOS NUMÉRICOS

Os métodos numéricos são técnicas matemáticas utilizadas para encontrar soluções aproximadas para problemas matemáticos que não podem ser resolvidos de forma exata através de métodos analíticos. Esses métodos são especialmente úteis para resolver equações diferenciais, integrais, sistemas de equações lineares e não lineares, e para realizar interpolação e otimização. Para o desenvolvimento desse trabalho, é importante revisar o cálculo de comprimento de uma curva e da área sob uma curva visto que serão tratados dados discretizados.

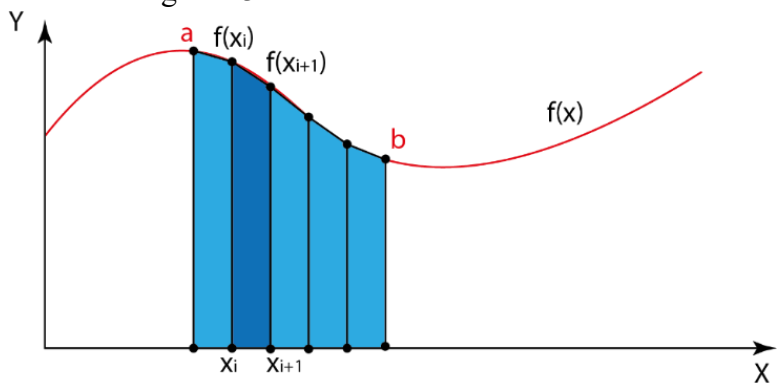
Seja uma curva *f(x)* dada entre *a* e *b*, a área sob essa curva é dada a partir da Equação 35.

$$\int_a^b f(x)dx \tag{35}$$

Para a resolução numérica desse problema, é possível utilizar o método da soma de trapézios. Essa integral pode ser resolvida de forma numérica caso de termos um operador *dx* não infinitesimal como mostrado na Figura 10. A partir da soma da área de trapézios entre a e b, obtêm-se a área aproximada sob a curva dada pela Equação 36.

$$A = \sum_{i=1}^{n-1} \frac{(f(x_{i+1}) + f(x_i))}{2} \cdot (x_{i+1} - x_i) \tag{36}$$

Figura 13 - Cálculo numérico em curvas



Fonte: Próprio autor

Seguindo o mesmo raciocínio, é possível calcular o comprimento de uma curva somando o comprimento dos segmentos entre x_i e x_{i+1}. O comprimento pode ser calculado a partir da Equação 37 [13].

$$l = \sum_{i=0}^{n} \sqrt{(f(x_{i+1}) - f(x_i))^2 + (x_{i+1} - x_i)^2} \tag{37}$$

Essa equação pode ser reescrita como uma integral, na forma mostrada pela Equação 38 [12].

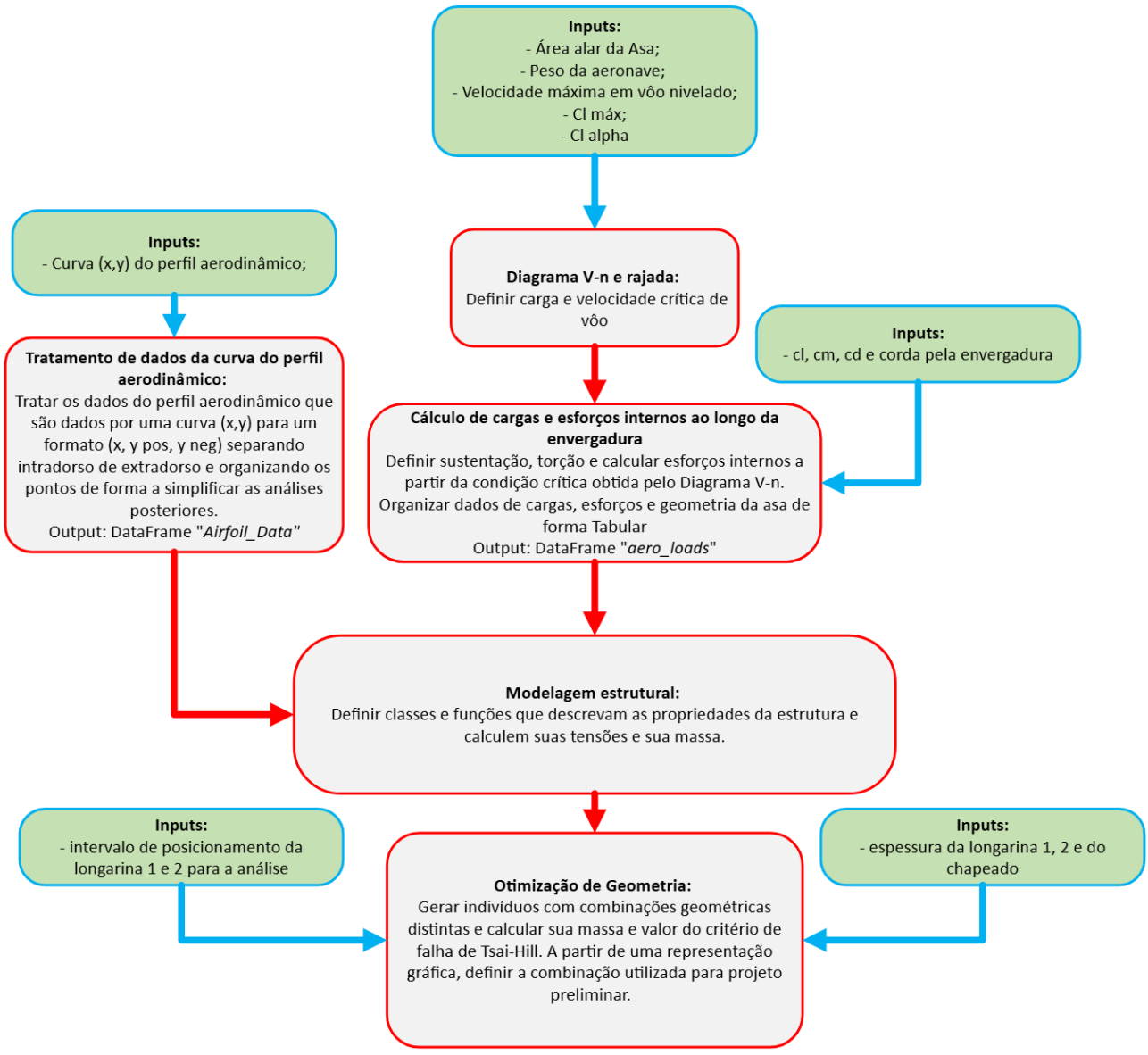
$$\int_a^b \sqrt{1 + [f'(x)]^2}\, dx \tag{38}$$

Sendo $f'(x)$ a primeira derivada da curva $f(x)$.

4 METODOLOGIA

O desenvolvimento do programa foi realizado de acordo com o fluxograma apresentado na Figura 14. Sua estrutura foi idealizada para que cada bloco de código execute uma etapa do projeto estrutural. Desse modo, é possível identificar a condição crítica de voo, calcular a distribuição de cargas e esforços internos na asa e por último, obter a otimização geométrica da estrutura. Cada bloco do programa foi desenvolvido utilizando-se de conceitos de programação na linguagem *Python* junto aos conceitos teóricos para cálculo estrutural aeronáutico.

Figura 14 - Fluxograma de desenvolvimento do programa



Fonte: Próprio autor

Para a implementação do código na linguagem *Python*, foram utilizadas as seguintes ferramentas da linguagem:

- Biblioteca *Pandas*: Utilizada para a importação, armazenamento e manipulação de dados por meio das estruturas de *DataFrames* e de seus métodos nativos;
- Classes: Utilizadas para criação de objetos em que seus atributos e métodos foram utilizados repetidas vezes ao longo do código;
- Funções: Utilizadas para execução de partes do código repetidas vezes;
- Laços de repetição e estruturas condicionais: Utilizadas para a execução de varreduras em intervalos de dados e tomadas de decisões de acordo com as condições impostas;
- Biblioteca Scipy: Utilizada para interpolação de dados e aplicação de métodos numéricos;
- Biblioteca *Matplotlib*: Utilizada para a construção de gráficos e diagramas que representam as estruturas de dados criadas a partir do programa.

Essas ferramentas foram fundamentais para integrar os conceitos teóricos abordados na Secção 3 com os principais objetivos desse trabalho. Ao longo dessa secção, serão apresentados os métodos e hipóteses adotados para a execução desse trabalho.

4.1 IMPLEMENTAÇÃO DO DIAGRAMA V-N

Para a confecção do diagrama V-n, utilizou-se os inputs organizados na Tabela 4.

Tabela 4 - Variáveis para definição do Diagrama V-n

Propriedade	Valor
$n_{máx}$	2,2
$v_{máx}$	17 m/s
W	98,1 N
S	1,24 m²
$\bar{c}$	0,53 m
a	5 rad^{-1}
ρ	1,22 kg/m³
$C_{l\,máx}$	1,47
U_{Vc}	7,62 m/s
U_{Vd}	3,81 m/s

Fonte: Próprio autor

Apesar da Equação 8 definir o $n_{máx}$ da aeronave em função de seu peso, para um projeto aerodesign a norma não é integralmente adotada dadas suas dimensões reduzidas e ao fato de a aeronave não transportar carga viva. Além disso, o projeto aerodesign não sofre alterações

significativas de escala e de condições de operação. Desse modo, foi adotado o fator de carga de 2,2 baseando-se no histórico da equipe.

No programa, foi desenvolvida uma classe denominada *V-n_Diagram*, como é mostrada no trecho de código da Figura 15. Nela, foram implementados os principais atributos e métodos necessários para a plotagem do Diagrama V-n de acordo com as normas da FAR Part 25 [6] e parâmetros de projeto (APÊNDICE A).

Figura 15 - Classe *Vn_Diagram*

```
class Vn_diagram:

    def rajada(self, U_de, pos_neg):

            mi_g = 2*(self.weight/(9.8 * self.wing_area))/(self.rho *self.med_chord * self.cl_alpha )
            K_g = (0.88 * mi_g)/(5.3 + mi_g)

            n_raj = 1 + pos_neg * (0.5 * self.rho * self.v_range * self.cl_alpha * K_g * U_de * self.wing_area)/(self.weight)

            return(n_raj)

    def __init__(self,n_max,wing_area,v_max,Cl_max,weight,rho,cl_alpha,med_chord,rajada_vc,rajada_vd):

        #Definindo os atributos da classe
        self.n_max = n_max
        self.v_max = v_max
        self.wing_area = wing_area
        self.v_max = v_max
        self.Cl_max = Cl_max
        self.weight = weight
        self.rho = rho
        self.n_max_neg =  1 #0.4 * self.n_max #Procurar referencia
        self.Cl_max_neg = 0.6 * self.Cl_max  # Deefinido por Iscold
        self.cl_alpha = cl_alpha
        self.med_chord = med_chord
        self.rajada_vc = rajada_vc
        self.rajada_vd = rajada_vd
        #-----------------------------------------------------------------------------------------------------
        #fator de carga positivo
        #Velocidade de estol:
        self.v_estol = np.sqrt(np.divide(2 * self.weight, self.rho * self.wing_area * self.Cl_max))
```

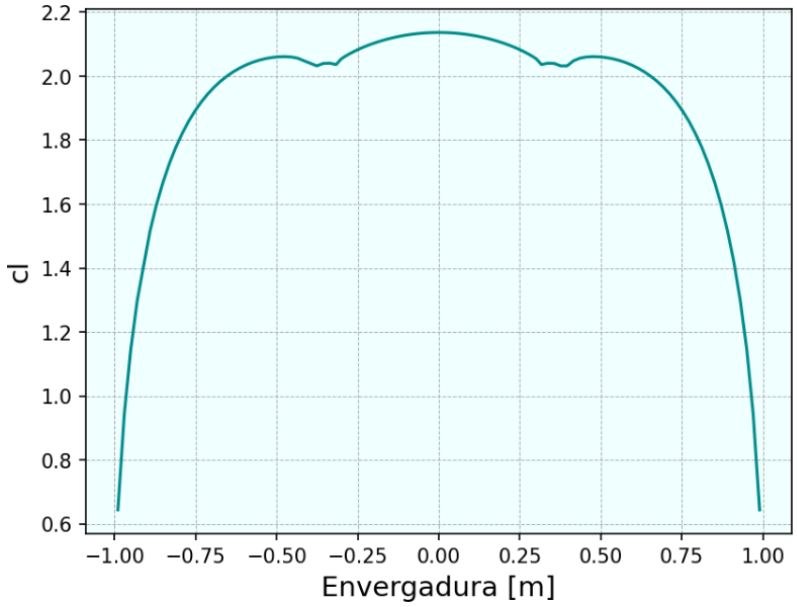
Fonte: Próprio autor

O código foi desenvolvido para calcular os envelopes de voo positivos e negativos tanto para o diagrama V-n quanto para o diagrama de rajada. Os dados obtidos foram organizados em *DataFrames* e plotados em *Matplotlib* por meio da função *Plot_Vn*. Dessa forma, é possível visualizar os dois diagramas e a intersecção entre eles. Com o valor máximo de fator de carga e velocidade obtidos, é possível dar início aos cálculos de cargas distribuídas na asa.

4.2 IMPLEMENTAÇÃO DO CÁLCULO DE CARGAS E ESFORÇOS INTERNOS

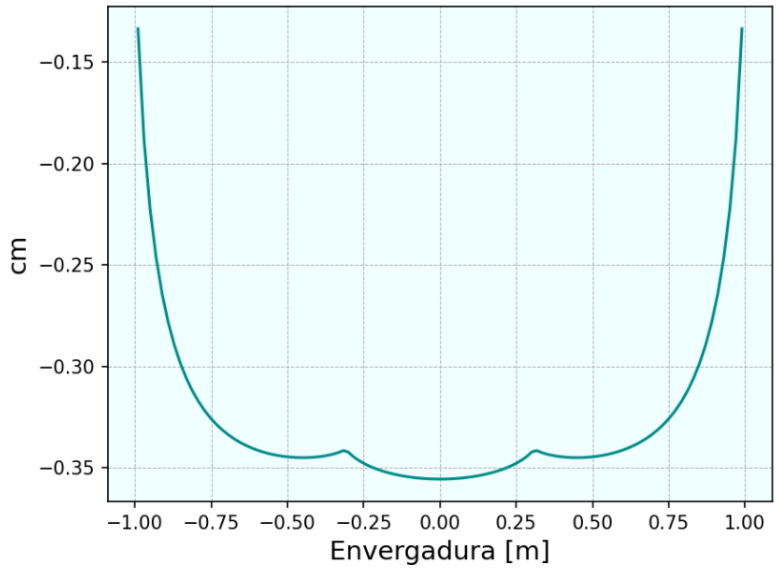
Além das informações obtidas a partir do diagrama V-n (fator de carga e velocidade críticos), é necessário importar as informações dos coeficientes aerodinâmicos *cl* e *cm* e da corda da asa *c* em função da envergadura *y* para ser realizado o cálculo das cargas na asa. Esses dados são fornecidos pela área da aerodinâmica em uma estrutura de dados tabular em arquivo *.xlsx*. Os dados importados são representados pelos gráficos de *cl x envergadura, cm x envergadura* e *c x envergadura* dados pela Figura 16, Figura 17 e Figura 18, respectivamente.

Figura 16 - cl x envergadura



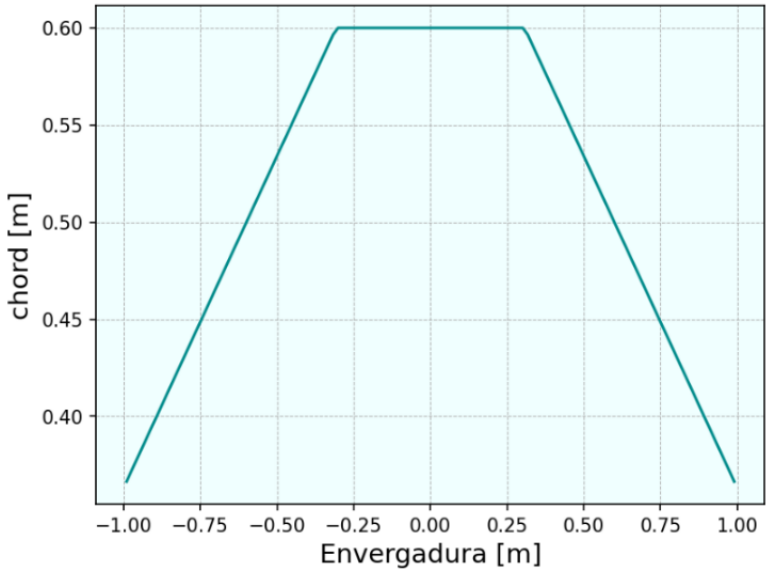
Fonte: Próprio autor

Figura 17 - cm x envergadura



Fonte: Próprio autor

Figura 18 - c x envergadura

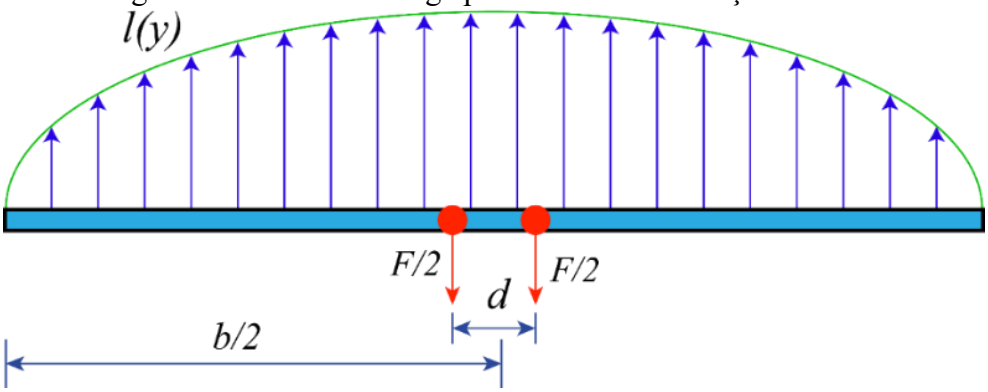


Fonte: Próprio autor

As informações foram importadas para o programa através do método *pd.read_excel()*. Assim, foi possível criar um *DataFrame* com essas informações e posteriormente realizar sua plotagem com os recursos de *Matplotlib*. A partir dos coeficientes aerodinâmicos e da distribuição de cordas na envergadura mostrados nas Figura 16, Figura 17 e Figura 18, foi possível calcular as cargas de sustentação e torção através da implementação das Equações 4 e 6 no código, respectivamente. A partir do cálculo dos esforços internos, foi obtida a secção transversal crítica de projeto.

Com os carregamentos externos obtidos, considerou-se o modelo apresentado pela Figura 19 para o cálculo dos esforços internos de força cortante, momento fletor e momento de torção. Nele, considera-se uma viga com carga distribuída e fixada na fuselagem por meio de dois pontos. Para isso, foram aplicadas as Equações 21, 24.

Figura 19 - Modelo de viga para cálculo de esforços internos



Fonte: Próprio autor

Dessa forma, obteve-se a força cortante *V(y)* por meio da Equação 39.

$$V(y) = \begin{cases} \int_{y}^{-b/2} l(y)dy \,, se\ -\frac{b}{2} < y \leq -\frac{d}{2} \\ \int_{y}^{-b/2} l(y)dy - \frac{F}{2}, se\ -\frac{d}{2} < y < \frac{d}{2} \\ \int_{y}^{-b/2} l(y)dy - F \,, se\ \frac{d}{2} \leq y < \frac{b}{2} \end{cases} \quad (39)$$

Sendo F a força de reação total nos dois apoios da asa, *d* a distância entre os apoios e *l(y)* a força de sustentação distribuída ao longo da asa. Visto que a torção aerodinâmica da asa dada pela Equação 6, também é distribuída por sua envergadura, é possível calcular o momento interno de torção de forma análoga a Equação 39, mas para isso considerando o momento total das reações de apoio. O momento fletor *M(y)* foi obtido a partir da integração de *V(y)* como foi mostrado pela Equação 24.

Para a implementação do cálculo de esforços internos no programa, foi utilizada uma estrutura de repetição *for* e condicionais *if* como é mostrado no trecho de código da Figura 20. Dessa forma, foi possível realizar uma varredura por todos os pontos da envergadura e dependendo de sua posição, calcular os esforços internos como mostrado pela Equação 39.

Figura 20 - Implementação do método das secções

```
for Y in b:

    if Y < (-d/2) :

        V += [ lift.integral(b.min(),Y) ]
        T += [ torsion.integral(b.min(),Y) ]

    elif Y > (-d/2) and Y < (d/2) :

        V += [  lift.integral(b.min(),Y) - F/2]
        T += [  torsion.integral(b.min(),Y) - T_reac/2]

    else:

        V += [  lift.integral(b.min(),Y) - F]
        T += [  torsion.integral(b.min(),Y) - T_reac]

cort = InterpolatedUnivariateSpline(b, V, k=1)

for Y in b:

    M += [cort.integral(b.min(),Y) ]

```

Fonte: Próprio autor

Como pode ser observado a partir do fluxograma mostrado na Figura 14, os esforços internos e as informações de geometria da asa são fundamentais para os cálculos estruturais que serão realizados na sequência. Desse modo, no desenvolvimento do programa optou-se por gerar um *DataFrame* denominado "*aero_loads"* que contêm todas as informações em função da envergadura da aeronave.

4.3 TRATAMENTO DE DADOS DO PERFIL AERODINÂMICO

Ao longo do desenvolvimento matemático e das hipóteses que serão assumidas nesse trabalho, a geometria das secções transversais da asa, dependem diretamente da curva do perfil aerodinâmico. O formato desse perfil pode ser obtido como um conjunto de pontos (x, y) em formato *.txt*, por meio da plataforma *airfoil tools* [13]. Contudo, a forma como os dados são organizados e a quantidade de pontos disponíveis compromete o desenvolvimento e precisão dos cálculos numéricos que foram implementados no programa. Para isso, as informações obtidas foram tratadas nas seguintes etapas:

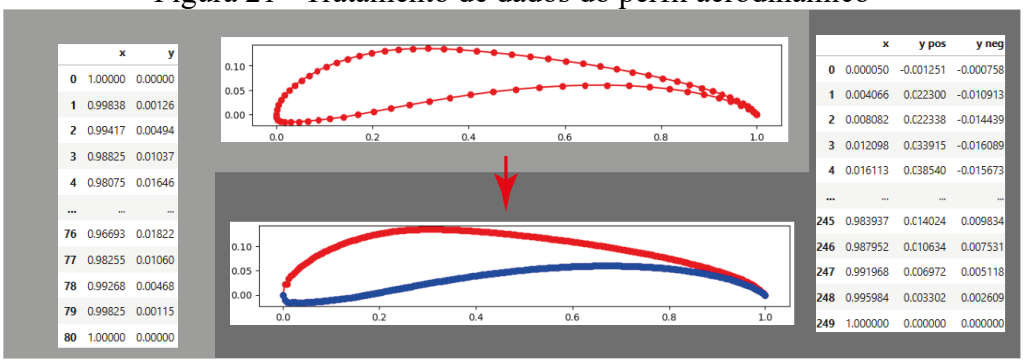
1. Separação da curva do perfil em extradorso e intradorso: Os pontos da curva importada foram divididos em extradorso e intradorso. Isso possibilita tratar as duas curvas como funções e consequentemente facilita a implementação de métodos numéricos;
2. Interpolação e organização dos pontos: A curva importada possui poucos pontos e eles não estão organizados de forma que o intervalo no eixo x entre eles seja equidistante. Para isso foi utilizado o método *interp1d* da biblioteca *scipy* para criar uma curva interpolada no mesmo intervalo de x, mas organizada e com mais pontos.

Desse modo, considerando um intervalo de $0 \leq x_{airfoil} \leq 1$, foi possível realizar a transformação da estrutura de dados assim como é dado na Equação 41.

$$(x_{airfoil}, z_{airfoil}) \rightarrow (x_{airfoil}, z_{airfoil\ intra}(x_{airfoil}), z_{airfoil\ extra}(x_{airfoil})) \tag{40}$$

A Figura 21 ilustra a transformação e tratamento de dados realizada no código (APÊNDICE C). A partir dela, será possível a implementação de diversos métodos numéricos que serão demonstrados nas próximas secções.

Figura 21 - Tratamento de dados do perfil aerodinâmico



Fonte: Próprio autor

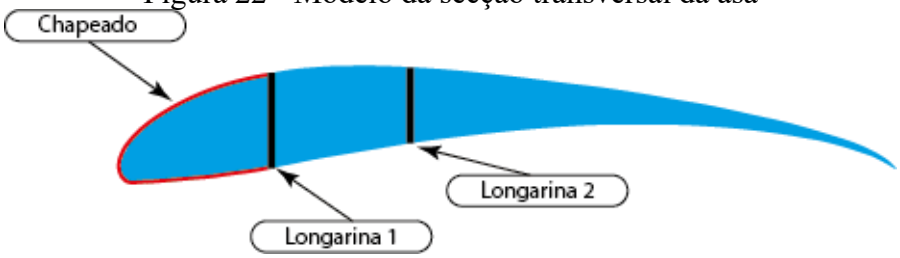
4.4 MODELAGEM ESTRUTURAL DA ASA

Para a definição do modelo estrutural da asa, foram assumidas as seguintes hipóteses:

- Duas longarinas na secção transversal;
- O posicionamento das longarinas é dado em percentual da corda na raiz da asa;
- Superfície aerodinâmica da asa pré-estabelecida (envergadura, perfil e cordas definidos);
- Altura de cada longarina igual a espessura da nervura na respectiva posição;
- O efeito das Cargas de arrasto é considerado desprezível se comparadas a sustentação e torção, devido a menor intensidade e maior rigidez na direção de atuação das forças;
- Desconsiderado o efeito das nervuras na rigidez estrutural;
- Desconsiderados os efeitos de fadiga visto que o aerodesign realiza poucos voos;
- Uma viga caixão definida entre a primeira longarina e o chapeado para resistir a torção;
- Cargas aplicadas na linha média do quarto de corda da secção transversal;
- As longarinas e o chapeado são considerados contínuos. Não possuem furos, encaixes para nervuras ou descontinuidades;

Na Figura 22, é possível conferir esquematicamente o modelo obtido a partir das hipóteses.

Figura 22 - Modelo da secção transversal da asa

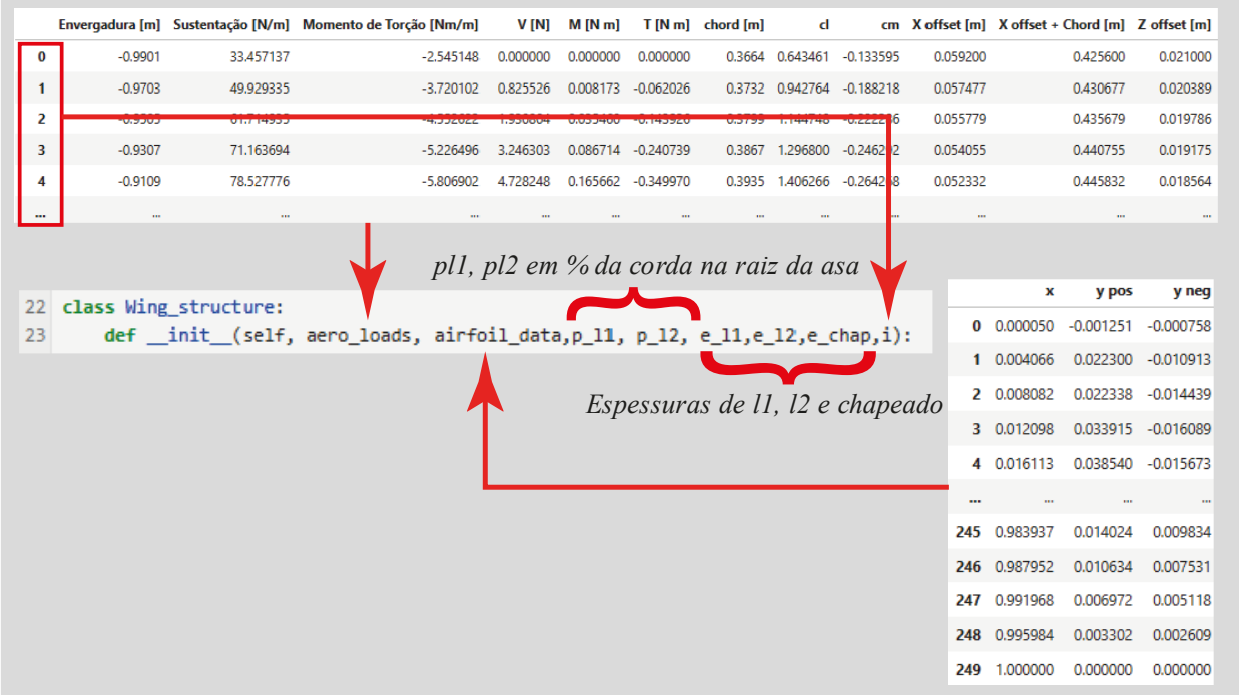


Fonte: Próprio autor

As hipóteses adotadas resultaram em um modelo simplificado do problema. Embora essas simplificações reduzam a precisão dos cálculos analíticos, as informações derivadas das análises deste modelo serão essenciais para estabelecer uma estrutura preliminar da asa, servindo como base para etapas mais detalhadas de projeto.

Dado que grande parte das análises estruturais dependem da geometria da secção transversal, foi implementada uma classe denominada "*Wing*_structure" para representá-la. Sua estrutura pode ser observada na Figura 23.

Figura 23 - Estrutura da classe *"Wing_structure"*



	Envergadura [m]	Sustentação [N/m]	Momento de Torção [Nm/m]	V [N]	M [N m]	T [N m]	chord [m]	cl	cm	X offset [m]	X offset + Chord [m]	Z offset [m]
0	-0.9901	33.457137	-2.545148	0.000000	0.000000	0.000000	0.3664	0.643461	-0.133595	0.059200	0.425600	0.021000
1	-0.9703	49.929335	-3.720102	0.825526	0.008173	-0.062026	0.3732	0.942764	-0.188218	0.057477	0.430677	0.020389
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.055779	0.435679	0.019786
3	-0.9307	71.163694	-5.226496	3.246303	0.086714	-0.240739	0.3867	1.296800	-0.246202	0.054055	0.440755	0.019175
4	-0.9109	78.527776	-5.806902	4.728248	0.165662	-0.349970	0.3935	1.406266	-0.264238	0.052332	0.445832	0.018564
...	...	...	...	...	...	...	...	...	...	...	...	...

	x	y pos	y neg
0	0.000050	-0.001251	-0.000758
1	0.004066	0.022300	-0.010913
2	0.008082	0.022338	-0.014439
3	0.012098	0.033915	-0.016089
4	0.016113	0.038540	-0.015673
...	...	...	...
245	0.983937	0.014024	0.009834
246	0.987952	0.010634	0.007531
247	0.991968	0.006972	0.005118
248	0.995984	0.003302	0.002609
249	1.000000	0.000000	0.000000

Fonte: Próprio autor

Sendo:

- "*aero_loads*": *DataFrame* contendo as informações de geometria da asa e esforços internos em função da envergadura;
- "*airfoil_data*": *DataFrame* contendo os pontos da curva do perfil aerodinâmico;
- P_l1 e pl2: Posicionamento da longarina 1 e 2, respectivamente;
- e_l1, e_l2 e e_chap: Espessuras das longarinas 1, 2 e chapeado, respectivamente;
- i: índice da linha do *DataFrames* "*aero_loads*". Define a posição na envergadura da secção transversal em análise.

Nessa classe, foram implementados vários atributos e métodos para calcular propriedades como centroide e as tensões, por exemplo (ANEXO D). Ao longo dessa secção, será demonstrado o desenvolvimento de algumas dessas equações e suas implementações no programa. Para uma melhor compreensão, os símbolos utilizados foram organizados na Tabela 5.

Tabela 5 - Variáveis empregadas no desenvolvimento matemático

Variável	Propriedade
x	Eixo x – Eixo horizontal da secção transversal
y	Eixo y – Direção da envergadura
z	Eixo z – Eixo vertical da secção transversal
c	Corda
L	Sustentação – distribuído pela envergadura
M	Momento Fletor
T	Momento de Torção
V	Força Cortante
I_{xx}	Momento de Inércia em x
I_{zz}	Momento de Inércia em z
$z_{airfoil\_intra}$	Coordenadas z do extradorso do perfil aerodinâmico
$z_{airfoil\_extra}$	Coordenadas z do intradorso do perfil aerodinâmico
$x_{airfoil}$	Coordenadas x do perfil aerodinâmico
x_{l1}	Posicionamento da longarina 1 no eixo x
x_{l2}	Posicionamento da longarina 2 no eixo x
e_{l1}	Espessura da longarina 1
e_{l2}	Espessura da longarina 2
e_{chap}	Espessura do chapeado
p_{L1}	% de x_{l1} em relação a corda na raiz da asa
p_{L2}	% de x_{l2} em relação a corda na raiz da asa
A	Área interna da viga caixão
l	Comprimento do chapeado
x_{offset}	Offset da asa no eixo x
y_{offset}	Offset da asa no eixo y

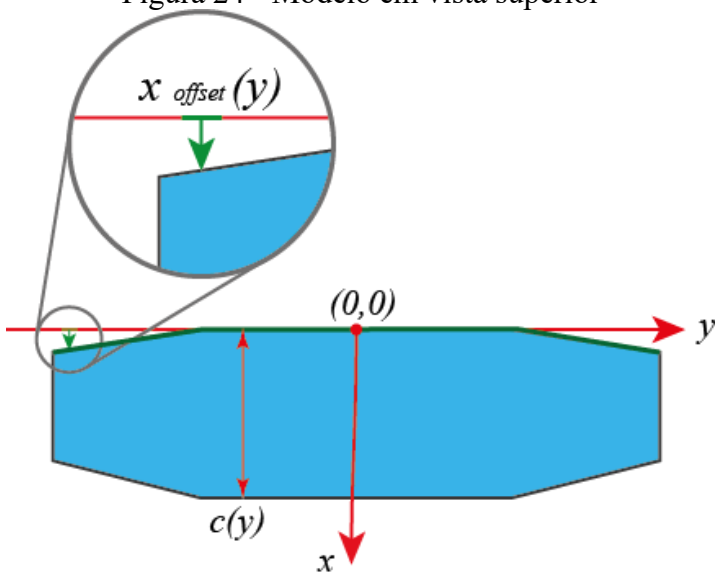
Fonte: Próprio autor

É possível notar que as variáveis obtidas por meio do *DataFrame* "*aero_loads"* são funções da posição na envergadura. Assim, tem-se

$$\begin{aligned} &\text{Geometria:} \quad c \rightarrow c(y);\ x_{offset} \rightarrow x_{offset}(y);\ z_{offset} \rightarrow z_{offset}(y) \\ &\text{Cargas:} \quad M \rightarrow M(y); V \rightarrow V(y);\ T \rightarrow T(y) \end{aligned} \tag{41}$$

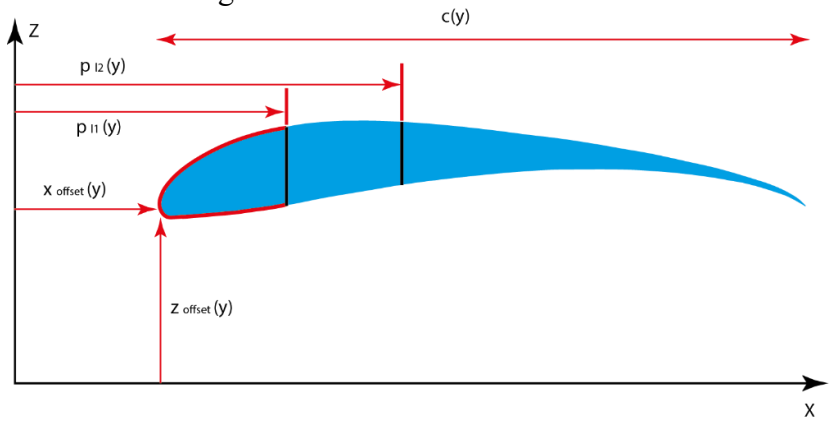
Na Figura 24 e Figura 25 é possível verificar as ilustrações que esquematizam o modelo adotado em relação aos eixos x, y e z, em vista superior e lateral, respectivamente.

Figura 24 - Modelo em vista superior



Fonte: Próprio autor

Figura 25 - Modelo em vista lateral



Fonte: Próprio autor

Por meio da Figura 25, é possível definir as equações que descrevem a geometria da superfície da asa. Para $0 \leq x_{airfoil} \leq 1$ tem-se:

$$x = x_{airfoil} * c(y) + x_{offset}(y) \quad (42)$$

$$z_{secção\ extra}(x_{airfoil}, y) = z_{airfoil\ extra}(x_{airfoil}) * c(y) + z_{offset}(y) \quad (43)$$

$$z_{secção\ intra}(x_{airfoil}, y) = z_{airfoil\ intra}(x_{airfoil}) * c(y) + z_{offset}(y) \quad (44)$$

O posicionamento das longarinas na classe "*Wing_structure*" é dado em percentual da corda na raiz da asa. Dessa forma, para obter-se o posicionamento no eixo x para *l1* e *l2*, obtêm-se:

$$x_{l1} = \frac{p_{l1} \cdot c(0)}{100} \quad (45)$$

$$x_{l2} = \frac{p_{l2} \cdot c(0)}{100} \quad (46)$$

Aplicando o valor de x_{l1} obtido na Equação 45 em x na Equação 42, é possível obter o valor $x_{\,airfoil\ l1}$ para o cálculo da coordenada z dos pontos superior e inferior da longarina *1* por meio das Equações 43 e 44, respectivamente. Repetindo esse método com a Equação 46, também foi possível calcular os pontos para a longarina *2*. Dessa forma, é possível descrever a altura da longarina *1* e *2* da seguinte forma:

$$h_{\ l1}\left(x_{airfoil\ l1}, y\right) = \left(z_{\,airfoil\ extra}\left(x_{airfoil\ l1}, y\right) - z_{\,airfoil\ intra}\left(x_{airfoil\ l1}\right)\right) \cdot c(y) \quad (47)$$

$$h_{\ l2}\left(x_{airfoil\ l2}, y\right) = \left(z_{\,airfoil\ extra}\left(x_{airfoil\ l2}, y\right) - z_{\,airfoil\ intra}\left(x_{airfoil\ l2}\right)\right) \cdot c(y) \quad (48)$$

De acordo com o modelo determinado, o chapeado constitui uma viga caixão com a longarina 1 e seu formato segue a silhueta do perfil aerodinâmico. Logo, as Equações 49 e 50 definem:

$$z_{\ chap\ extra}\left(x_{airfoil}, y\right) = z_{secção\ extra}\left(x_{airfoil}, y\right), para\ x \leq x_{airfoil\ l1} \quad (49)$$

$$z_{\ chap\ intra}\left(x_{airfoil}, y\right) = z_{secção\ intra}\left(x_{airfoil}, y\right), para\ x \leq x_{airfoil\ l1} \quad (50)$$

Na Figura 26, é possível verificar a implementação das Equações de 42 a 50 no código desenvolvido.

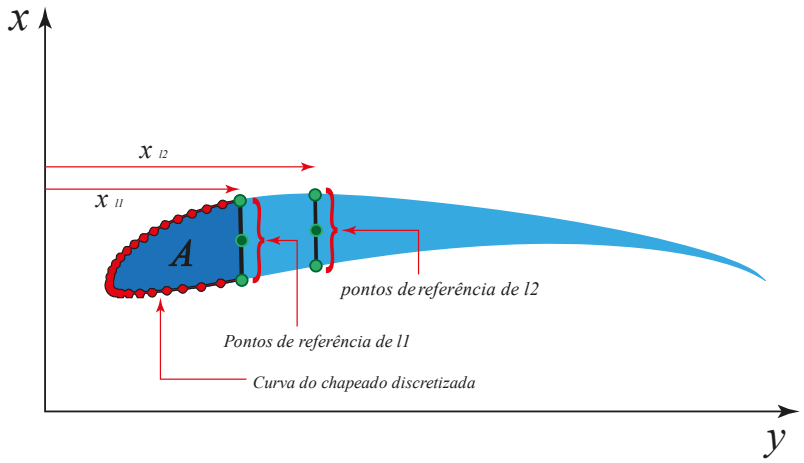
Figura 26 - Implementação das equações de geometria no programa

```
# Cálcular pontos da Longarina 1 e Longarina 2.
self.x_l1 = (self.p_l1/100) * self.chord_max
self.x_l2 = (self.p_l2/100) * self.chord_max

#secção transversal (calculando as curvas dos perfis nas secções transversais)
self.cross_section = self.airfoil_data * self.chord
self.cross_section["x"] =  self.cross_section["x"] + self.x_offset
self.cross_section["y pos"] =  self.cross_section["y pos"] + self.z_offset
self.cross_section["y neg"] =  self.cross_section["y neg"] + self.z_offset


#Cálcular pontos l1 e l2.
self.coord_l1 = self.cross_section[self.cross_section["x"] <= self.x_l1].iloc[-1]
self.coord_l2 = self.cross_section[self.cross_section["x"] <= self.x_l2].iloc[-1]

#Altura de l1 e de l2:
self.h_l1 = self.coord_l1["y pos"] - self.coord_l1["y neg"]
self.h_l2 = self.coord_l2["y pos"] - self.coord_l2["y neg"]

#Curva chapeado:
self.chapeado_shape = self.cross_section[self.cross_section["x"] <= self.x_l1]
```

Fonte: Próprio autor

As Equações de 41 a 53 determinam os pontos e curvas de referência para descrevermos a geometria do modelo adotado. Através delas será possível determinar a geometria de qualquer secção transversal da asa ao longo da envergadura. Na Figura 27 é representado o modelo de secção transversal discretizado. Foi possível defini-lo de acordo com as informações obtidas a partir dos *DataFrames "aero_loads"* e "*airfoil_data"* e com as Equações 42 a 50.

Figura 27 – Modelo discretizado



Fonte: Próprio autor

Por meio do modelo discretizado, foram implementados diversos métodos numéricos para realizar cálculos de propriedades geométricas e estruturais da asa (APÊNDICE D). Esses métodos possibilitaram o cálculo do centroide, momento de inércia, distribuição de tensões e massa, por exemplo.

O centroide e os momentos de inércia da secção transversal do modelo são os resultantes entre o conjunto das duas longarinas com o chapeado. Para as longarinas 1 e 2, seus centroides foram determinados como seu ponto médio e seus momentos de inércia foram calculados por meio da Equação 31. Para o chapeado, foi necessário implementar um cálculo numérico auxiliar nos cálculos. Para isso, foi desenvolvida a função "*area_pos"* mostrada na Figura 28.

Figura 28 - Função "*area_pos*"

```
e_chap = 3/32
def area_pos(x,y,e):  #vetor x, vetor y e espessura.

        xmed = (x[1::]+x[0:-1])/2
        ymed = (y[1::]+y[0:-1])/2
        A = (np.sqrt((x[1::]-x[0:-1])**2+(y[1::]-y[0:-1])**2))*e
        vector = np.array([xmed,ymed,A])
        return vector
```

Fonte: Próprio autor

Essa função importa a curva do chapeado em suas coordenadas (x, z) e a considera com uma espessura *e*. A partir disso, calcula as coordenadas do centroide e a área de cada segmento da curva e

retorna esses valores. Como é possível observar nas Equações 27, 28 e 30, esse cálculo é essencial na obtenção do centroide e momentos de inercia do chapeado.

O cálculo da tensão normal de flexão foi feito a partir da Equação 25. Considerando que temos apenas momento fletor atuante em *x*, simplificou-se para a Equação 50:

$$\sigma_y = \frac{M_x(I_{zz}z - I_{xz}x)}{I_{xx}I_{zz} - I_{xz}^2} \tag{50}$$

Sendo que as distâncias x e z tem como origem o centroide da secção transversal da asa, anteriormente calculado. No programa, σ_y foi calculado para 3 pontos equidistantes em cada longarina. O fluxo de cisalhamento devido a torção foi calculado por meio da Equação 26. Para isso, a área interna da viga caixão, mostrada na Figura 27, foi calculada a partir da implementação numérica da Equação 37. Foi calculada a tensão de cisalhamento em l1 e no chapeado dividindo – se o fluxo de cisalhamento pela respectiva espessura de parede. As tensões σ_{l1}, σ_{l2} e $t_{torção\ xz}$ foram calculados como atributos da classe *Wing_structure* assim como são mostrados na Figura 29.

Figura 29 - Cálculo de tensões no programa

```
#Sigma  l1 e l2:
self.sigma_l1 = (-self.M * (self.Izz * (self.zz_l1 - self.z_centroide)
                          - self.Ixz * (self.xx_l1 - self.x_centroide))/(self.Ixx * self.Izz - self.Ixz**2)) * 1e-6
self.sigma_l2 = (-self.M * (self.Izz * (self.zz_l2 - self.z_centroide)
                          - self.Ixz * (self.xx_l2 - self.x_centroide))/(self.Ixx * self.Izz - self.Ixz**2)) * 1e-6

# Cálculo das tensões devido a torção da asa:
self.q_torsion = self.T / ( 2 * self.chapeado_area)
self.tau_torsion_l1 = (self.q_torsion / self.e_l1) / 1e6
self.tau_torsion_chap = (self.q_torsion / self.e_chap) / 1e6
```

Fonte: Próprio autor

Na análise estrutural da asa, calcular sua massa é fundamental. Para isso, a função *Wing_mass* (Figura 30) foi desenvolvida. Nesta função, são fornecidos os parâmetros *pl1*, *pl2*, as espessuras das longarinas e do chapeado, além da densidade da madeira balsa especificada na Tabela 2. Ela itera ao longo da envergadura e calcula, por meio dos atributos da classe *Wing_structure,* as alturas das longarinas e comprimento do chapeado para cada secção transversal iterada. A partir dos pontos obtidos, é possível realizar uma integração numérica ao longo da envergadura e obter a área de cada uma das superfícies da estrutura. Ao se multiplicar os valores obtidos pelas respectivas espessuras e densidade do material, obtém-se a massa estrutural.

Figura 30 - Função para cálculo de massa da asa

```
def Wing_mass(pl1,pl2,rho,vertices,e_l1,e_l2,e_chap):
    e_l1 = e_l1 * 0.0254
    e_l2 = e_l2 * 0.0254
    e_chap = e_chap * 0.0254
    #extrair dados:
    comp_chapeado = []
    b = []
    h_l1 =[]
    h_l2 = []
    #massa l1 e l2:
    for id in vertices:
        data = Wing_structure(aero_loads, airfoil_data,pl1,pl2, e_l1,e_l2,e_chap,id)

        comp_chapeado.append(data.chapeado_length)
        b.append(data.b)
        h_l1.append(data.h_l1)
        h_l2.append(data.h_l2)

    b = np.array(b); h_l1 = np.array(h_l1)
    h_l2 = np.array(h_l2); comp_chapeado = np.array(comp_chapeado)

    massa_l1 = np.trapz(h_l1,b) * e_l1 * rho
    massa_l2 =  np.trapz(h_l2,b)* e_l2 * rho
    massa_chap =  np.trapz(comp_chapeado,b)* e_chap * rho

    massa_total = massa_l1 + massa_l2 + massa_chap
    return(massa_total)
```

Fonte: Próprio autor

A Figura 24 ilustra também o parâmetro "*vertices"*, que é uma lista contendo os índices *i* em *aero_loads* onde estão localizados os vértices da asa trapezoidal. Isso permite que a função itere sobre posições específicas de referência na longarina em *Wing_structure*, em vez de processar todas as 200 linhas de *aero_loads*. Essa abordagem otimiza o código, reduzindo o tempo de execução.

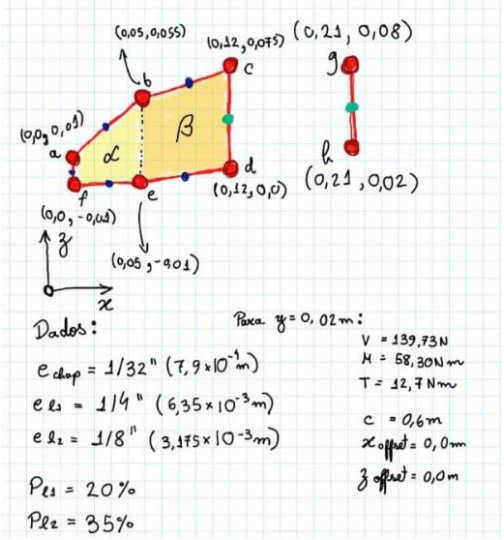
Tabela 6 - Parâmetros adotados para validação da rotina

Propriedade	**Valor**
p l1	20%
p l2	35%
e l1	1/4"
e l2	1/8"
e chap	1/32"

Fonte: Próprio autor

Para a validação do código, as equações implementadas foram calculadas analiticamente e os resultados obtidos foram comparados com os do programa. Para isso, foi selecionado uma secção transversal com as propriedades mostradas na Tabela 6. Para o cálculo analítico manual, foi utilizado o modelo ilustrado na Figura 31. A maior diferença percentual encontrada entre os cálculos foi de 2,52%. Logo, é possível afirmar que a implementação das equações utilizadas nesse trabalho foi feita de forma adequada no programa.

Figura 31 - Modelo para validação analítica dos cálculos do programa



Fonte: Próprio autor

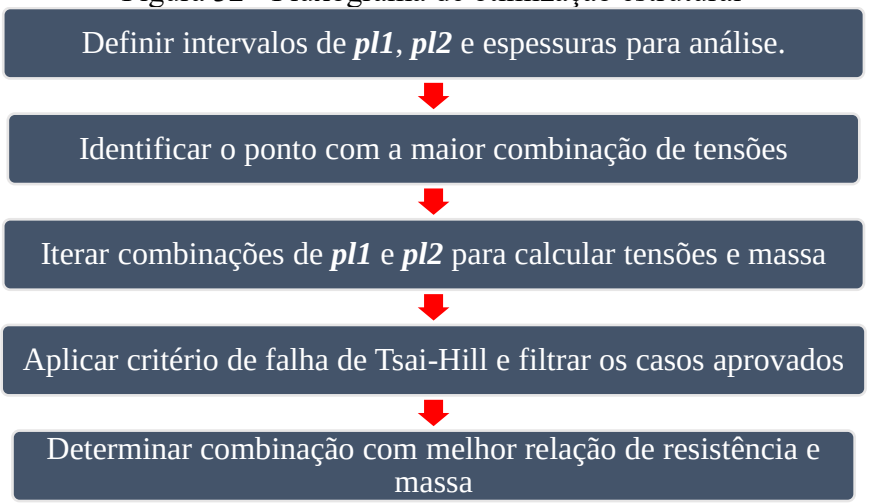
4.5 OTIMIZAÇÃO ESTRUTURAL

Com base na Figura 30, observa-se que tanto a classe *Wing_structure* (cálculo das tensões), quanto a função *Wing_mass* (cálculo da massa da asa), dependem dos parâmetros *pl1*, *pl2* e das espessuras das longarinas e do chapeado. Dessa forma, é possível concluir que tanto a resistência estrutural quanto a massa podem ser descritas como funções no formato da Equação 51.

$$f(p_{l1}, p_{l2}) \tag{51}$$

Sendo as espessuras, constantes da função. Para se obter uma função de massa $m(p_{l1}, p_{l2})$ e outra referente a resistência estrutural $r(p_{l1}, p_{l2})$ foi necessário seguir as etapas do fluxograma apresentado na Figura 32.

Figura 32 - Fluxograma de otimização estrutural



Fonte: Próprio autor

Conforme indicado no fluxograma apresentado na Figura 32, foi elaborado um conjunto de análises variando os parâmetros geométricos da estrutura. A Tabela 7 apresenta os intervalos contínuos definidos para *pl1* e *pl2*, bem como as opções de valores discretos selecionados para as espessuras das longarinas e do chapeado. Esses valores foram escolhidos com o objetivo de explorar uma grande possibilidade combinações estruturais.

Tabela 7 - Parâmetros geométricos para otimização

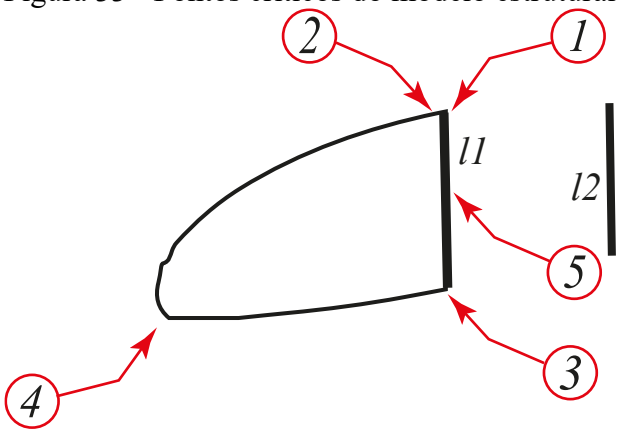
Parâmetro	**Valor**
p_{l1} [%]	14 a 28
p_{l2} [%]	32 a 45
e_{l1} [pol]	1/4 ou 1/8
e_{l2} [pol]	1/4 ou 1/8
e_{chap} [pol]	1/32

Fonte: Próprio autor

As opções de espessura de *l1* e *l2* foram determinadas de acordo com as opções disponíveis no mercado que são mais utilizadas para a confecção das longarinas. A espessura do chapeado foi fixada em 1/32 devido a facilidade construtiva de curvar as chapas. Os intervalos de *pl1* e *pl2* foram determinados de acordo com limitações geométricas e construtivas para o modelo adotado. No programa, o intervalo de *pl1* foi dividido em 29 pontos e *pl2* em 27 pontos. Desse modo, foram testados 783 indivíduos para cada combinação de espessuras adotada. No geral, 3132 indivíduos foram testados.

A segunda etapa consiste em avaliar o ponto com a combinação de tensões atuantes mais crítica e avaliar sua resistência de acordo com a Equação 33 do critério de falha de Tsai-Hill. Na Figura 33 temos alguns possíveis pontos críticos da estrutura.

Figura 33 - Pontos críticos do modelo estrutural



Fonte: Próprio autor.

Cada ponto foi avaliado de acordo com suas tensões atuantes.

1. Ponto de máxima compressão por σ_y. Nesse ponto também temos a atuação de τ_{xz} devido a torção da asa, contudo seu módulo será menor que no chapeado pois a espessura de *l1* é maior que no chapeado;
2. Esse ponto localiza-se no chapeado, muito próximo a *l1*. Nele, a tensão de compressão σ_y será muito próxima do obtido no ponto 1. Contudo, a tensão de cisalhamento será significativamente maior devido a menor espessura da parede do chapeado em relação a longarina;
3. Ponto de máxima tração por σ_y. Nesse ponto também temos a atuação de τ_{xz} devido a torção da asa, contudo seu módulo será menor que no chapeado pois a espessura de *l1* é maior que do chapeado;
4. Esse ponto localiza-se no chapeado, muito próximo a *l1*. Nele, a tensão de tração σ_y será muito próxima do obtido no ponto 3. Contudo, a tensão de cisalhamento será significativamente maior devido a menor espessura da parede do chapeado em relação a longarina;
5. Possui cisalhamento atuando devido a torção e a flexão. Contudo, nesse ponto o valor das tensões será menor se comparado aos outros pontos, devido a espessura das paredes.

A análise identificou os pontos 2 e 4 como os de maiores combinações de tensões. Para eles, foi implementada a função *beams_variation* (APÊNDICE E), que itera sobre as combinações dos intervalos *pl1* e *pl2*, gerando todas as possibilidades e calculando as tensões correspondentes, como é mostrado pela fração do código na Figura 34. Os resultados são armazenados no *DataFrame beams_data*.

Figura 34 - Trecho do Código da Função *beams_variation*

```
for pl1 in p_l1_range:
    for pl2 in p_l2_range:

        data = Wing_structure(aero_loads, airfoil_data,pl1,pl2, e_l1_thickness,e_l2_thickness,e_chap,61)
        sigma_max_l1.append(data.sigma_l1.max())
        sigma_max_l2.append(data.sigma_l2.max())
        sigma_min_l1.append(data.sigma_l1.min())
        sigma_min_l2.append(data.sigma_l2.min())
        tau_max_l1.append(data.tau_torsion_l1)
        tau_max_chap.append(data.tau_torsion_chap)
        pl1_dot.append(pl1)
        pl2_dot.append(pl2)
        Izz.append(data.Izz)
        Ixx.append(data.Ixx)
        massa.append(Wing_mass(pl1,pl2,rho_balsa,vertices,e_l1_thickness, e_l2_thickness,e_chap))


beams_data = pd.DataFrame({ "posição l1 [%]": pl1_dot,
                            "posição l2 [%]": pl2_dot,
                            "sigma max l1 [MPa]" : sigma_max_l1,
                            "sigma max l2 [MPa]" : sigma_max_l2 ,
                            "sigma min l1 [MPa]" : sigma_min_l1,
                            "sigma min l2 [MPa]" : sigma_min_l2 ,
                            "tau max torção l1 [MPa]" : tau_max_l1,
                            "tau max torção chap [MPa]" : tau_max_chap,
                            "massa [kg]" : massa })
return(beams_data)
```

Fonte: Próprio autor.

A função de resistência $r(p_{l1}, p_{l2})$ foi determinada a partir dos valores obtidos pela aplicação do critério de falha de Tsai-Hill (Figura 35). Para a armazenagem dos valores calculados, foi criada uma coluna em *beams_data* com o valor do critério de falha obtido para cada linha.

Figura 35 - Função Tsai-Hill

```
def Tsai_Hill(sigma_1 , sigma_2,tau, sigma_adm_1, sigma_adm_2, tau_adm):
    Tsai_Hill = ( sigma_1 / sigma_adm_1 )**2 + ( sigma_2 / sigma_adm_2 )**2
    - ((sigma_1 * sigma_2)/sigma_adm_1) + np.sqrt(( tau / tau_adm)**2)
    return(Tsai_Hill)

nalise_Otimização_Longarina = dados
Analise_Otimização_Longarina["Tsai Hill tração L1"] = Tsai_Hill(dados["sigma max l1 [MPa]"],
                                                               0,dados["tau max torção chap [MPa]"],
                                                               sigma_tracao_adm , 1 , tau_adm_max )
Analise_Otimização_Longarina["Tsai Hill compressão L1"] = Tsai_Hill(dados["sigma min l1 [MPa]"],
                                                                   0,dados["tau max torção chap [MPa]"],
                                                                   sigma_compressao_adm , 1 , tau_adm_max )
```

Fonte: Próprio autor.

Nessa função, a Equação 33 foi simplificada para a Equação 51, de acordo com as tensões atuantes no problema e as tensões admissíveis de projeto.

$$\left(\frac{\sigma_1}{X_{adm\,T}}\right)^2 - \left(\frac{\sigma_1\sigma_2}{X_{adm\,T}{}^2}\right) + \left(\frac{\tau_{12}}{S^2_{adm\,12}}\right)^2 \leq 1 \tag{51}$$

Obtidos os dados do *DataFrame beams_data*, notou-se que em todas as linhas o ponto 2 apresentou-se como mais crítico que o 4 em todos os indivíduos. A partir dessa confirmação, foram determinadas as colunas de *beams_data* que representam as funções $m(p_{l1}, p_{l2})$ e $r(p_{l1}, p_{l2})$. Desse modo, foram plotados quatro gráficos distintos, um para cada combinação de espessuras de longarina disponível. O eixo *y* representou o intervalo de *pl1* e o eixo *x*, o intervalo de *pl2*. Neste espaço, foram traçadas as curvas de nível de resistência estrutural ($r(p_{l1}, p_{l2})$) e o mapa de cores da massa ($m(p_{l1}, p_{l2})$). Isso possibilitou avaliar as correlações entre os parâmetros e identificar as combinações mais relevantes para o projeto. Por meio da análise, foi possível determinar quais parâmetros geométricos devem ser adotados para a obtenção do indivíduo com a melhor relação de massa e resistência.

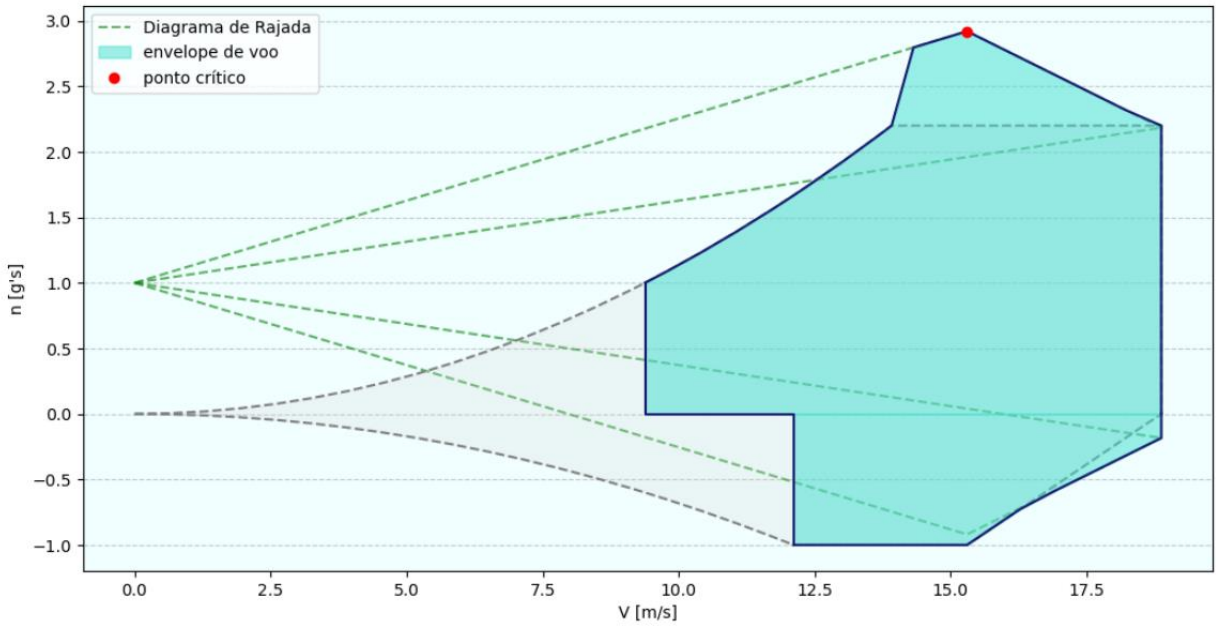
É fundamental reforçar que o modelo desenvolvido neste trabalho foi projetado para ser utilizado em fases preliminares do projeto. Ele serve como uma ferramenta para auxiliar na tomada de decisões durante as etapas iniciais de desenvolvimento, onde a otimização estrutural da asa é realizada de forma ágil e eficiente. Embora o modelo simplifique certos aspectos do comportamento estrutural e ignore algumas variáveis mais complexas, como efeitos de nervuras e concentradores de

tensões, ele proporciona uma base para identificar configurações geométricas e estruturais promissoras. Assim, o programa permite uma rápida avaliação de alternativas e fornece dados essenciais que podem ser refinados em etapas posteriores do projeto, quando recursos adicionais, como ensaios físicos ou simulações avançadas, estiverem disponíveis.

5 RESULTADOS E DISCUSSÃO

Por meio da classe *Vn_diagram* (ANEXO A) desenvolvida e das informações dadas na Tabela 4. É possível plotar o gráfico do Diagrama V-n do projeto como mostrado na Figura 36.

Figura 36 - Diagrama V-n desenvolvido

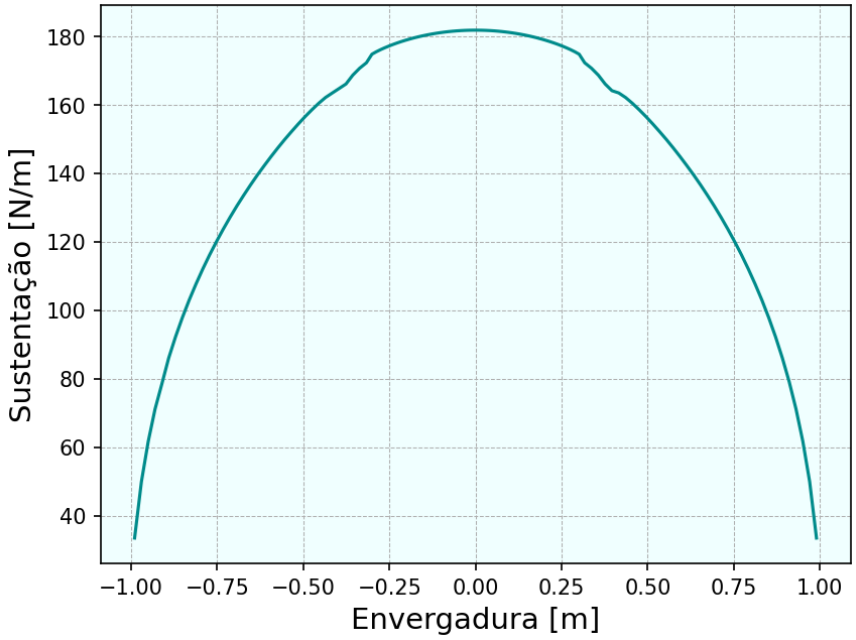


Fonte: Próprio autor

Nota-se a partir do Diagrama, que a condição crítica ocorre na velocidade v_c de 15,24 m/s e fator de carga 2,92. Dado que o peso de projeto da aeronave é de 98,0 N, calcula-se por meio da Equação 7 a força de sustentação máxima que será de 286,16 N. É possível observar pelo Diagrama V-n que as rajadas tiveram grande influência na determinação do fator de carga máximo, elevando-o de 2,2 para 2,92, um aumento de 32%. Isso pode se dever ao fato de que nesse projeto, a área da asa *S* é relativamente grande para o peso *W* da aeronave. Para essa faixa de peso, as aeronaves de outros anos apresentavam uma corda na raiz da asa de 450mm. Nesse projeto, o valor é de 600mm. Quando isso ocorre, os valores de *n* obtidos na Equação 15 tendem a ser maiores, causando maior influência dos efeitos de rajada de vento nas cargas críticas da estrutura.

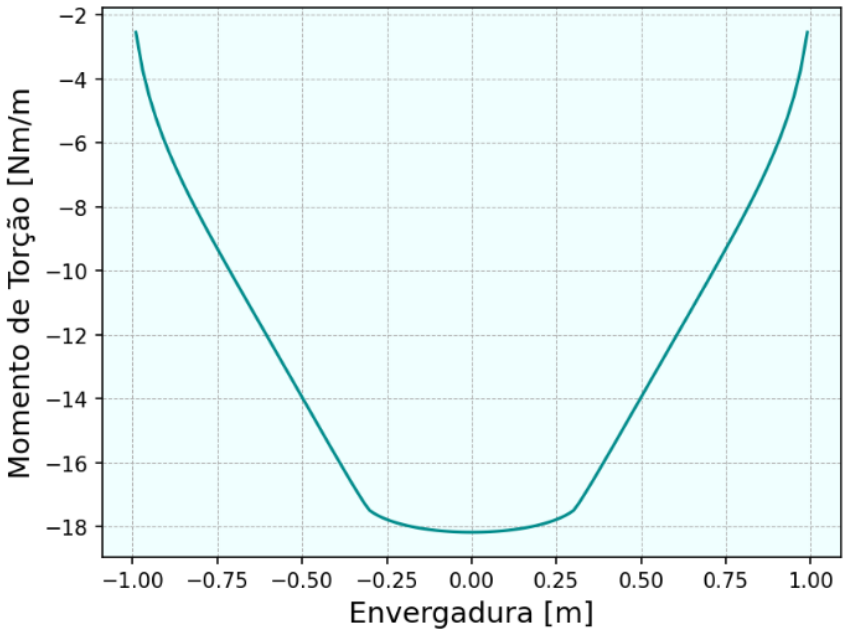
A partir da condição crítica obtida pelo Diagrama V-n e os dados fornecidos pela área de Aerodinâmica ilustrados pela Figura 16, Figura 17 e Figura 18 obteve-se a distribuição de sustentação e de torção ao longo da asa, dados pela Figura 37 e Figura 38, respectivamente.

Figura 37 - Força de sustentação x envergadura



Fonte: Próprio autor

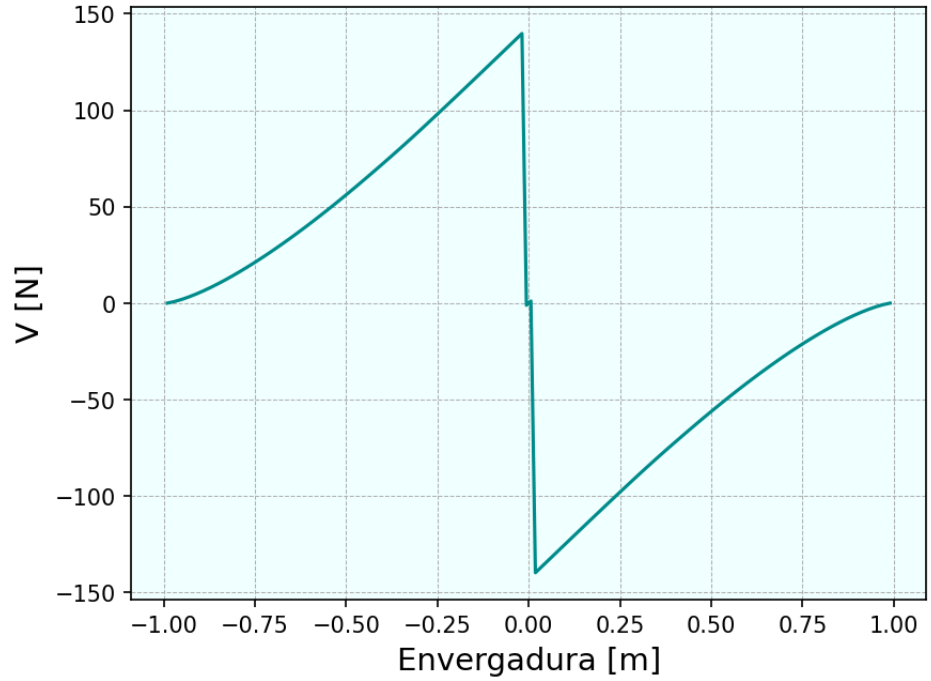
Figura 38 - Momento de torção x envergadura



Fonte: Próprio autor

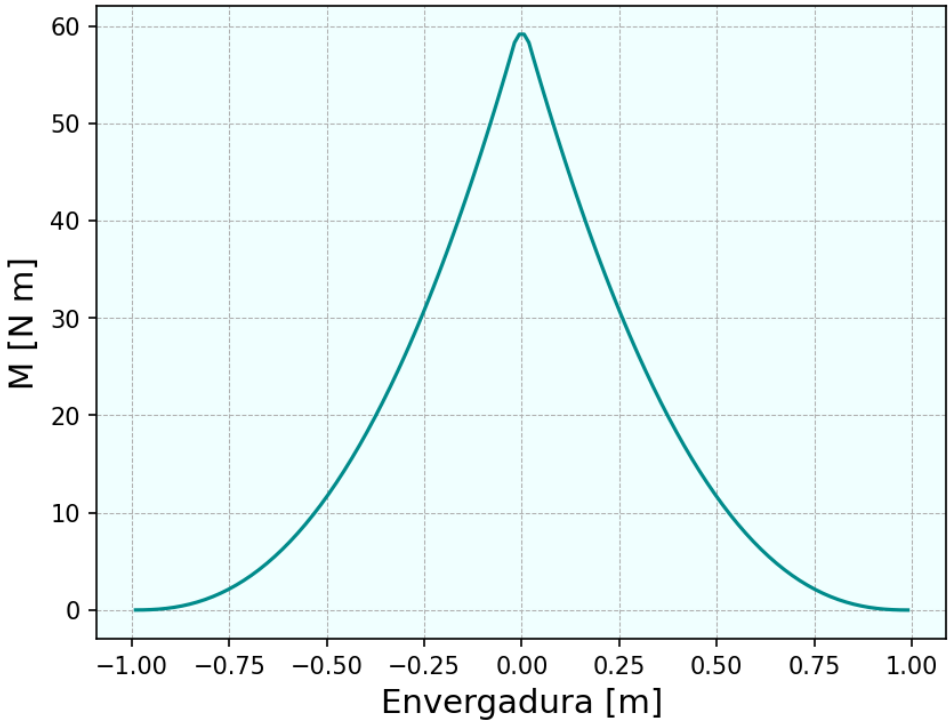
A partir dos carregamentos obtidos, foi realizado o cálculo dos esforços internos. Os diagramas de V, M e T distribuídos pela envergadura como mostrado pela Figura 39, Figura 40 e Figura 41, respectivamente.

Figura 39 - Força cortante V x envergadura



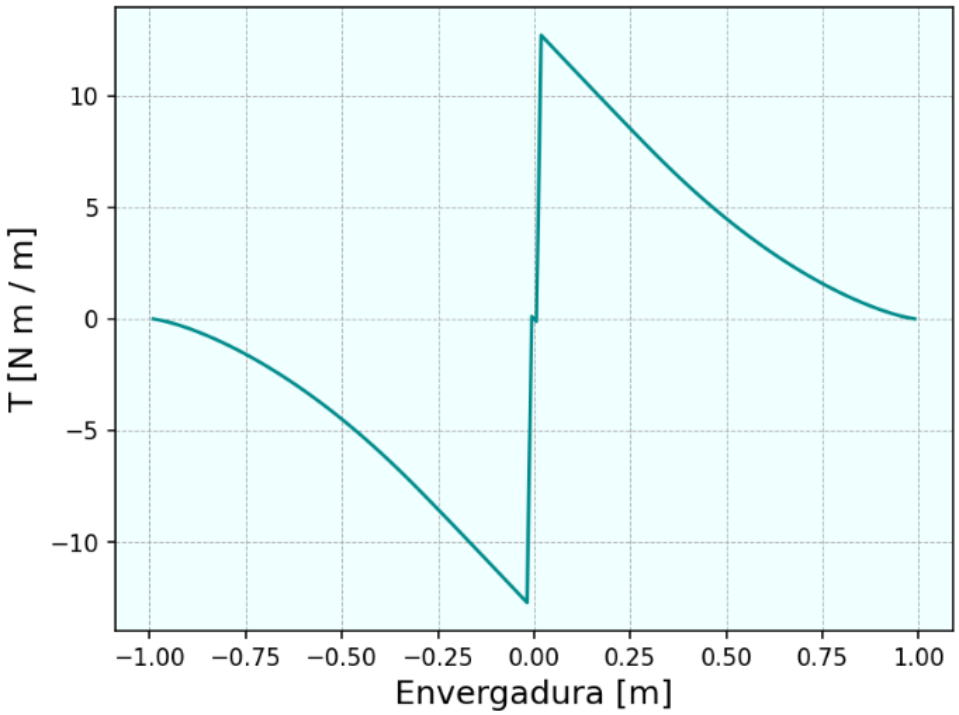
Fonte: Próprio autor

Figura 40 - Momento fletor M x envergadura



Fonte: Próprio autor

Figura 41 - Momento de torção x envergadura



Fonte: Próprio autor

A partir dos gráficos obtidos, nota-se que os maiores esforços internos se localizam próximos à raiz da asa. Isso é esperado pois nessa região encontram-se as estruturas de fixação da asa a fuselagem. Além disso, as curvas obtidas pelos diagramas são semelhantes as obtidas na literatura [2], mudando em intensidade. Desse modo, é possível concluir que a implementação dos cálculos de esforços internos no programa foi bem sucedida.

Referente ao ponto crítico, apesar do maior momento fletor M ser identificado no ponto 0,00 m da envergadura, nesse ponto a força cortante V e o momento de torção T são nulos. Para o dimensionamento estrutural, o ponto com a maior combinação dos três esforços ocorre nos pontos y = 0,018m e y = -0,018m, sendo o módulo dos esforços dados na Tabela 8.

Tabela 8 - Esforços internos máximos

Esforço	**Valor**
M	58,30 N m
V	139,73 N
T	12,7 N m
i	58

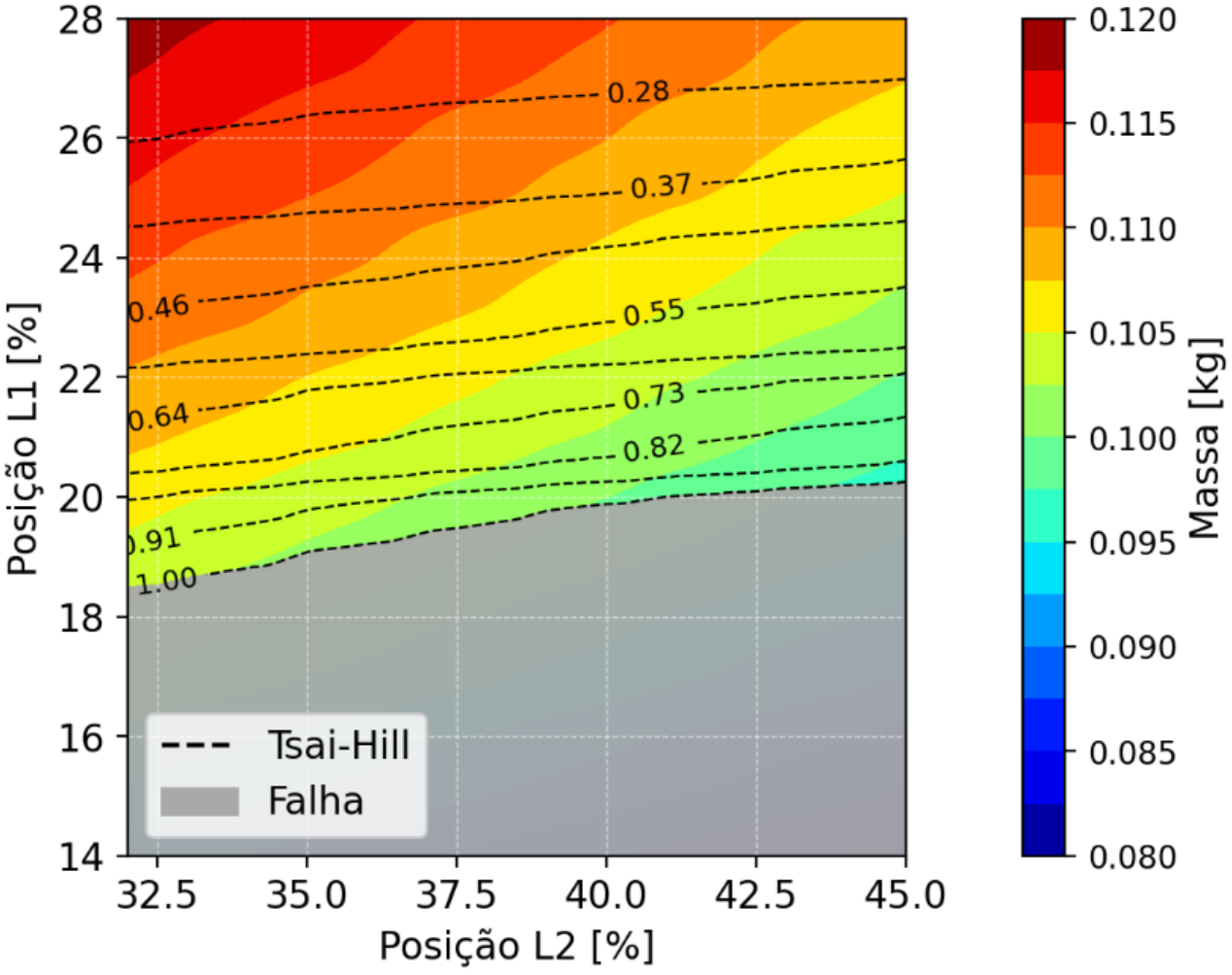
Fonte: Próprio autor

Determinar os esforços estruturais críticos é fundamental para o projeto estrutural na asa. A partir dessa condição, é possível estimar as tensões máximas atuantes na estrutura e dimensionar os componentes estruturais.

Na Figura 42, temos a configuração da primeira e segunda longarinas com 1/8 de polegada. É possível notar que grande parte dos indivíduos falham ao ser aplicado o critério de falha de Tsai-Hill. Para essa configuração, de 783 indivíduos testados, 469 apresentaram resistência suficiente para suportar as cargas atuantes. Ou seja, 40,10% falharam.

Essa configuração de espessuras é a que apresenta a menor massa possível dentro das possibilidades, variando de 80 g até 120 g. A partir do gráfico, também é possível concluir que assumindo a primeira longarina em 23.5% da corda da raiz e a segunda a 40% da corda é possível obter uma longarina com 102,50 g e um fator de Tsai-Hill de 0,55. Nota-se que é possível obter uma configuração mais leve, contudo pelo gráfico é possível concluir que a redução de segurança será muito maior que o ganho em redução de massa. Além disso, poderia ser adotada uma configuração mais leve e de mesma resistência se *pl2* = 45%. Entretanto, quanto mais na extremidade for adotada a longarina 2, mais limitações construtivas podem surgir devido a outros componentes estruturais que são embarcados na asa, como os servos dos ailerons.

Figura 42 - Resistencia mecânica x massa - l1: 1/8 pol; l2: 1/8 pol



Fonte: Próprio autor

As informações do indivíduo escolhido foram organizadas na Tabela 9.

Tabela 9 – Informações do indivíduo otimizado

Parâmetro	**Valor**
p_{l1} [%]	23,5
p_{l2} [%]	40,0
e_{l1} [pol]	1/8
e_{l2} [pol]	1/8
e_{chap} [pol]	1/32
$\tau_{chap\ xz}$ $[MPa]$	0,95
$\sigma_{chap\ comp}$ $[MPa]$	3,82
$\sigma_{chap\ traç}$ $[MPa]$	3,06
Massa [g]	102,5
$Tsai - Hill_{ponto\ 2}$ *[adim.]*	0,55

Fonte: Próprio autor

Pela Figura 42, a combinação de pl1 = 22% e pl2 = 32%, apresenta uma massa de aproximadamente 107,5 g e o mesmo fator de falha de Tsai-Hill que o indivíduo escolhido. Teríamos assim um aumento de 4,87% na massa sem nenhum ganho de resistência em relação ao indivíduo escolhido anteriormente. Se fosse adotada uma configuração de duas longarinas de ¼ pol (como era feito nos projetos até 2022), mas com o mesmo posicionamento das longarinas dado pela Tabela 9, teríamos uma estrutura de aproximadamente 164 g e um fator de falha de Tsai-Hill de aproximadamente 0,34, como é mostrado na Figura 42, nota-se um aumento de 59% na massa do indivíduo e com ganho de resistência desnecessário. Nesse caso, a estrutura da asa seria menos competitiva.

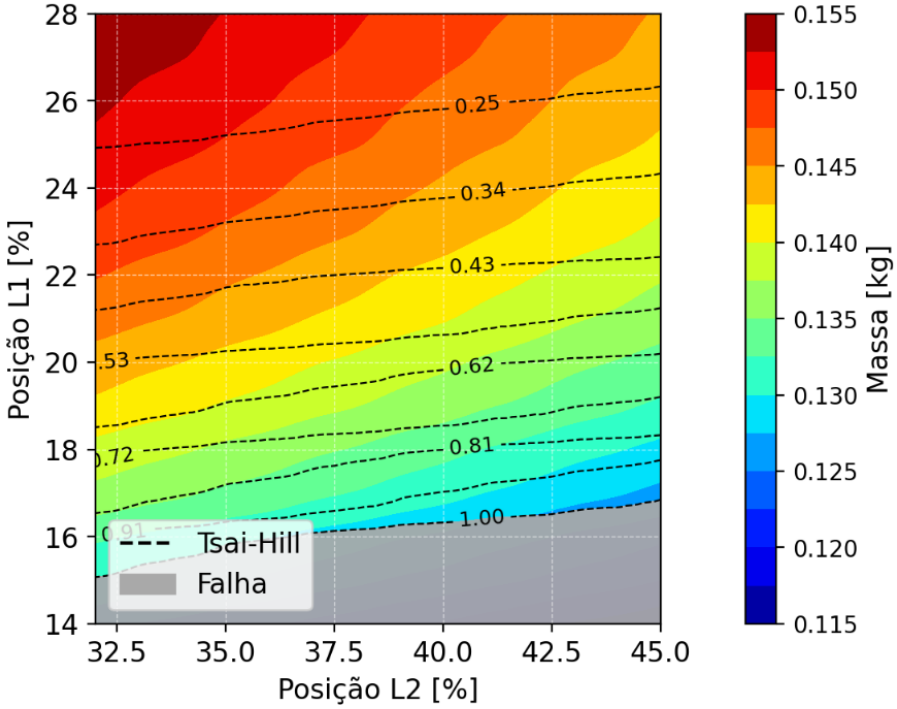
É também possível notar pelo gráfico, que o aumento de resistência tem mais influência pela alteração do posicionamento da longarina *1* que a longarina *2*. Isso se deve ao fato de que estamos aumentando a área interna da viga caixão, conferindo maior rigidez a torção para a estrutura. Entretanto, isso também pode indicar uma necessidade de melhoria do modelo adotado. Nele, o efeito da longarina *2* na resistência a torção é considerado desprezível.

Portanto, era esperado que a alteração da posição de l2 tivesse menor influência na resistência estrutural. Isso pode ser analisado implementando-se métodos mais avançados de cálculo da torção estrutural, abordados por Megson [10]. Visto que o programa desenvolvido representa a secção transversal de forma detalhada por meio da classe *Wing_structure,* é possível incrementar o modelo a partir de novos métodos e atributos da classe.

Nas Figuras 43, 44 e 45 estão organizados os gráficos obtidos para as demais combinações espessura disponíveis. Para esse projeto, não foi necessário analisar outro gráfico além do dado pela Figura 42. A partir dele, foi possível adotar a configuração dada pela Tabela 9. Um indivíduo leve e

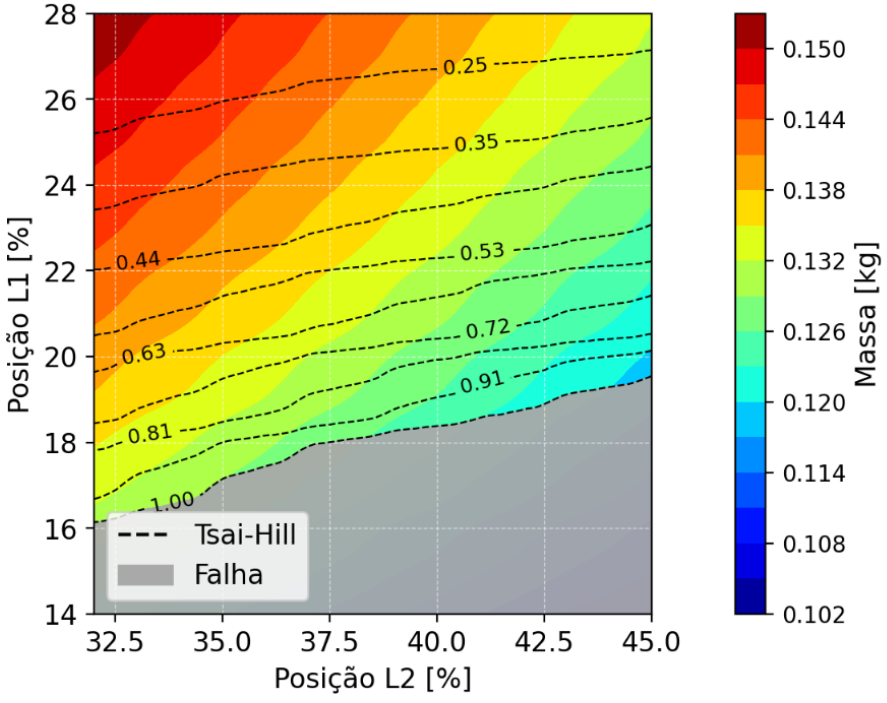
com elevada resistência estrutural. Entretanto, se tivéssemos um projeto com cargas mais intensas e geometria de asa diferente, por exemplo, é possível que a configuração com as longarinas mais finas não fornecesse indivíduos com segurança suficiente para o projeto.

Figura 43 - Resistencia mecânica x massa - l1: 1/4 pol; l2: 1/8 pol



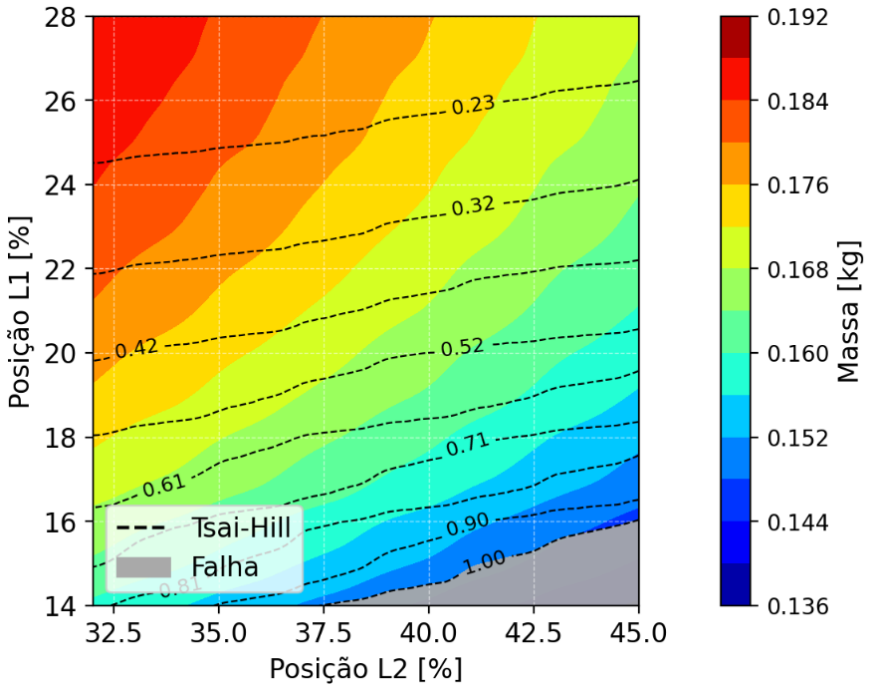
Fonte: Próprio autor

Figura 44 - Resistencia mecânica x massa - l1: 1/8 pol; l2: 1/4 pol



Fonte: Próprio autor

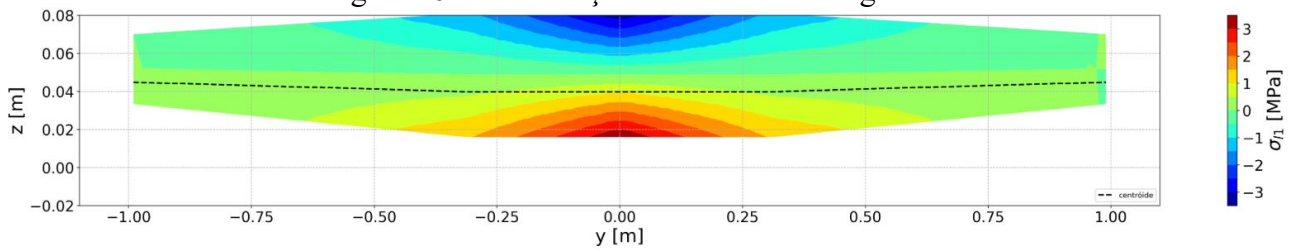
Figura 45 - Resistencia mecânica x massa - l1: 1/4 pol; l2: 1/4 pol



Fonte: Próprio autor

Além da otimização estrutural, é importante ressaltar que a rotina desenvolvida permite realizar análises adicionais para a estrutura. Na Figura 46, por exemplo é mostrada a distribuição de σ ao longo longarina 2. A partir dessa figura, poderiam ser feitas análises estruturais adicionais, como por exemplo verificar regiões da estrutura que necessitam de reforço com fibra de carbono.

Figura 46 - Distribuição de tensões na longarina 2



Fonte: Próprio autor

A partir dos resultados obtidos, é possível dizer que o código desenvolvido fornece informações fundamentais para a etapa de projeto preliminar da estrutura. Os dados obtidos pelo programa mostraram-se precisos se comparados com resultados analíticos obtidos para o modelo. Além disso, o programa é versátil para atualizações, implementações de novos métodos e adoção de modelos mais avançados para suas análises. Ademais, o programa foi desenvolvido para um *input* de estruturas de dados padronizadas como a curva de perfil aerodinâmico e os dados de aerodinâmica, isso permite que mesmo com alterações significativas no projeto, o programa funcione corretamente.

É importante ressaltar ainda, que sua estrutura vertical permite abordar todas as etapas do projeto fornecendo a equipe informações fundamentais como a distribuição dos esforços internos e as tensões máximas, por exemplo. Os resultados obtidos pelo programa podem ser comparados com ensaios estruturais e com análises numéricas de elementos finitos. Isso permitirá uma abordagem mais aprofundada no projeto estrutural da asa e beneficiará o desempenho da equipe na competição.

6 CONCLUSÃO

O programa desenvolvido em *Python* cumpriu os objetivos estabelecidos, oferecendo uma ferramenta eficiente para o projeto preliminar e otimização da estrutura da asa em um aerodesign. Ao integrar conceitos teóricos e técnicas numéricas utilizando-se *Python*, foi possível automatizar cálculos como distribuição de tensões, massa estrutural e resistência, promovendo análises mais ágeis. O programa elimina ajustes manuais, reduzindo erros e otimizando o tempo de execução. Ademais, sua estrutura permite explorar combinações geométricas que equilibram resistência e leveza.

Foi demonstrado, também, que o programa permite a implementação de outras análises e de incrementos ao modelo devido sua estrutura modular. O código possibilita a importação de diferentes perfis aerodinâmicos e geometrias de asa. Além disso, a classe *Wing_structure* foi programada para que seus atributos sejam validos mesmo com a alteração de seus parâmetros. Além disso, é possível a implementação de outros métodos e atributos a essa classe. Isso confere uma expressiva versatilidade a rotina, podendo ser utilizada em inúmeros projetos aerodesign. A otimização implementada resultou em uma considerável redução de massa estrutural. Além disso, foi possível compreender melhor as correlações entre os parâmetros geométricos e a resistência estrutural da asa.

6.1 SUJESTÕES PARA TRABALHOS FUTUROS

A partir do trabalho desenvolvido, sugere-se que sejam desenvolvidos os seguintes trabalhos:

- Revisar o desenvolvimento do diagrama V-n sobre a ótica de outras normas aeronáuticas, como a JAR-VLA por exemplo.
- Implementar a influência da rigidez das nervuras na estrutura e analisar os efeitos na rigidez estrutural da asa;
- Implementar o cálculo do centro de cisalhamento na asa para que os efeitos da torção na asa sejam estudados de forma mais abrangente;
- Realizar análise com ensaios estruturais ou modelos numéricos para a validação do modelo.

REFERÊNCIAS

[1] REDAÇÃO. *Python* ou Java: qual linguagem será mais dominante em 2024? *IT Forum*, 24 jan. 2024. Disponível em: https://itforum.com.br/noticias/*Python*-ou-java-qual-linguagem/. Acesso em: 23 nov. 2024.

[2] *PANDAS*. Disponível em: https://pandas.pydata.org/. Acesso em: 11 nov. 2024.

[3] *MATPLOTLIB*. Disponível em: https://matplotlib.org/. Acesso em: 11 nov. 2024.

[4] RODRIGUES, Luiz Eduardo Miranda José, 1973 - Fundamentos da Engenharia Aeronáutica com Aplicações ao Projeto SAE-AeroDesign: Volume Único / Luiz Eduardo Miranda José Rodrigues – Salto/SP: www.engbrasil.eng.br,

[5] LOMAX, T. L. Structural Loads Analysis for Commercial Transport Aircraft: Theory and Practice. Reston, VA: AIAA Education Series, 1996.

[6] U.S. GOVERNMENT. Federal Acquisition Regulation Part 25. [S.l.], 1965

[7] ISCOLD, P. H., Introdução as Cargas nas Aeronaves, CEA, UFMG

[8] HIBBELER, R. C. Resistência dos Materiais. 10ª. ed. São Paulo: Pearson, 2016

[9] RS BALS**.** Asas. Disponível em: http://rsbals.weebly.com/asas.html. Acesso em: 9 nov. 2024.

[10] MEGSON, T. Aircraft Structures for Engineering Students. 6th. ed. Oxford: Butterworth-Heinemann, 2016.

[11] ABDALSLAM, Suof Omran. *Impact damage analysis of balsawood sandwich composite materials.*2021. 83 f. (Doctorate in Mechanical Engineering) – Wayne State University.

[12] CHAPRA, Steven C.; CANALE, Raymond P. Numerical Methods for Engineers. 7. ed. New York: McGraw-Hill, 2015.

[13] TSAI, S. W.; PATRICK, R. M.; SHI, H. K. Composite Materials: Engineering and Science. 2. ed. Lancaster: Technomic Publishing, 2002.

[14] SELIG 1223. Disponível em: http://airfoiltools.com/airfoil/details?airfoil=s1223-il. Acesso em: 9 nov. 2024

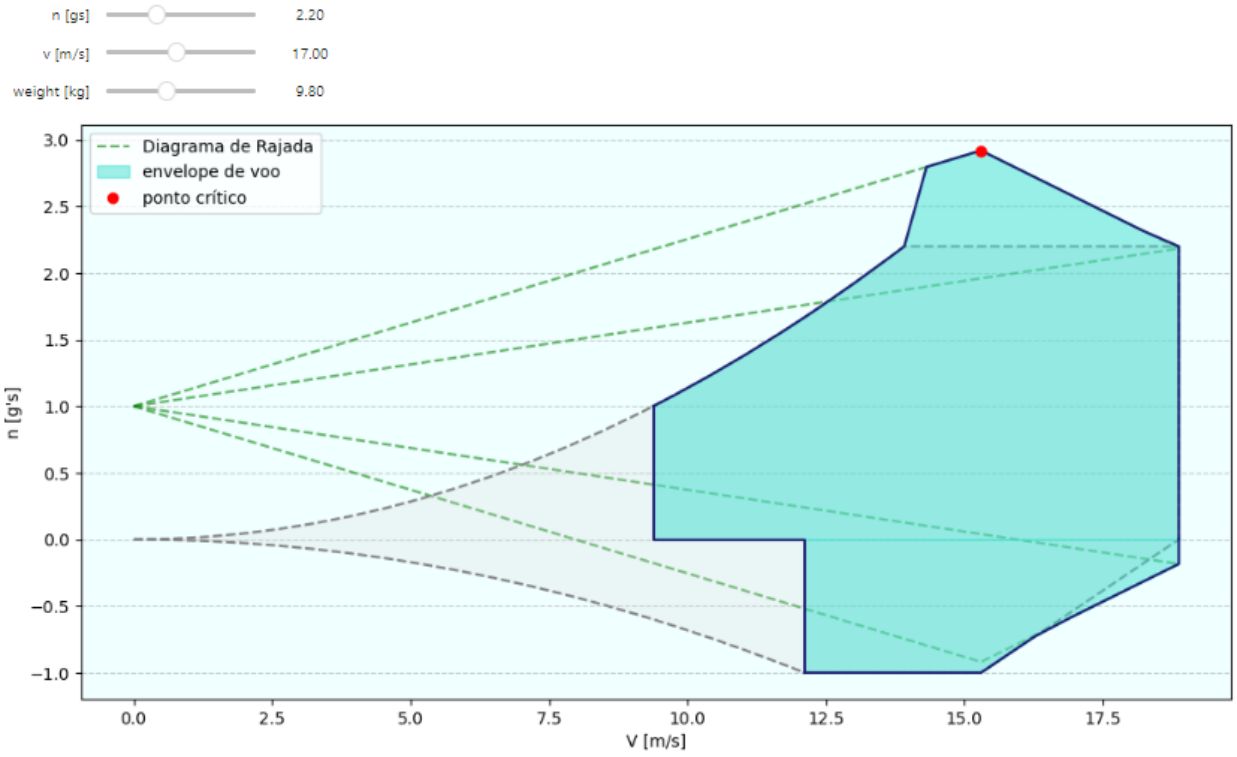
APÊNDICE A – CÓDIGO DO DIAGRAMA V-N

```
# IMPORTAÇÃO DE BIBLIOTECAS E PACOTES

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


from scipy.interpolate import InterpolatedUnivariateSpline

from scipy.interpolate import interp1d #iremos criar uma curva interpolada para o perfil

import ipywidgets as wd
from IPython.display import display

#-----------------------------------------------------------------------------------------------------------------------------------------------------------
# DIAGRAMA VN

#Variáveis Diagrama Vn:
aircraft_weight = 10.00 * 9.8    # [N] Peso da aeronave
wing_area = 1.24 # [m^2] #Área alar da Asa
rho =   1.222         # [kg/m^3] velocidade máxima em vôo nivelado
v_max = 19 #[m/s] Velocidade de máximo alcance en voo nivelado
n_max  = 2.5 # Fator de carga máximo

Cl_max = 1.47                 #[adim.] Coeficiente de sustentação máximo
Cd_max = 0.19                 #[adim.] Coeficiente de arrasto máximo

Cn_max = np.sqrt(Cl_max**2+Cd_max**2) # Coeficiente resultante máximo

cl_alpha = 5 #Variação da sustentação

med_chord = 0.530875 # Corda média

rajada_vc = 15.24/2 #Maior  #Velocidade de rajada em V_c
rajada_vd = 7.62/2 #Menor   #Velocidade de rajada em V_d

#Propriedades Madeira Balsa
rho_balsa = 200 #kg/m3


def n_calc(rho, S, Cl_max, v, weight):
    n = (0.5 * rho * S * Cl_max * v**2)/(weight)
    return(n)

#Definição de classe para o plote do Diagrama V-n:
class Vn_diagram:

    def rajada(self, U_de, pos_neg):

            mi_g = 2*(self.weight/(9.8 * self.wing_area))/(self.rho *self.med_chord * self.cl_alpha )
            K_g = (0.88 * mi_g)/(5.3 + mi_g)

            n_raj = 1 + pos_neg * (0.5 * self.rho * self.v_range * self.cl_alpha * K_g * U_de * self.wing_area)/(self.weight)

            return(n_raj)

    def __init__(self,n_max,wing_area,v_max,Cl_max,weight,rho,cl_alpha,med_chord,rajada_vc,rajada_vd):

        #Definindo os atributos da classe
        self.n_max = n_max
        self.v_max = v_max
        self.wing_area = wing_area
        self.v_max = v_max
        self.Cl_max = Cl_max
        self.weight = weight
        self.rho = rho
        self.n_max_neg =  1 #0.4 * self.n_max #Procurar referencia
        self.Cl_max_neg = 0.6 * self.Cl_max  # Deefinido por Iscold
        self.cl_alpha = cl_alpha
        self.med_chord = med_chord
        self.rajada_vc = rajada_vc
        self.rajada_vd = rajada_vd
        #-----------------------------------------------------------------------------------------------------
        #fator de carga positivo
        #Velocidade de estol:
        self.v_estol = np.sqrt(np.divide(2 * self.weight, self.rho * self.wing_area * self.Cl_max))

        #Velocidade de manobra:
        self.v_manobra = np.sqrt(np.divide(2 * self.weight * self.n_max, self.rho * self.wing_area * self.Cl_max))

        #Velocidade de cruzeiro:
        self.v_c = 0.9* self.v_max

        #Velocidade de mergulho:
        self.v_d = 1.11 *  self.v_max
```

```
        #Velocidade de mergulho:
        self.v_d = 1.11 *  self.v_max

        self.v_manobra_neg = np.sqrt(np.divide(2 * self.weight * (self.n_max_neg), self.rho * self.wing_area * self.Cl_max_neg))
        self.v_estol_neg = np.sqrt(np.divide(2 * self.weight , self.rho * self.wing_area * self.Cl_max_neg))


        self.main_speeds = pd.DataFrame({"$V_{estol}$":self.v_estol,
                                         "$V_{manobra}$":self.v_manobra,
                                         "$V_{c}$":self.v_c,
                                         "$V_{d}$":self.v_d,
                                         "$V_{manneg}$":self.v_manobra_neg,
                                         "$V_{estol neg}$":self.v_estol_neg}, index = ["Velocidade [m/s]:"])


        self.v_range = np.linspace(0,self.v_d,30)[:-1]

        self.v_range = np.sort(np.append(self.v_range, np.array(self.main_speeds.iloc[0].tolist())))

        self.N_Vn_pos = np.array([self.n_max if i > self. v_manobra else n_calc(self.rho, self.wing_area, self.Cl_max, i, self.weight) for i in self.v_range])


        self.N_Vn_neg = np.array([-self.n_max_neg if i > self. v_manobra_neg else -n_calc(self.rho,
                                                                                          self.wing_area,
                                                                                          self.Cl_max_neg,
                                                                                          i, self.weight) for i in self.v_range])

        self.N_Vn_neg[self.v_range >= self.v_c] = - self.n_max_neg +((self.v_range[self.v_range>=self.v_c]- self.v_c)/(self.v_range[-1] - self.v_c))*self.n_max_neg

        self.v_range = np.append(self.v_range,self.v_range[-1])
        self.N_Vn_pos = np.append(self.N_Vn_pos,0)
        self.N_Vn_neg = np.append(self.N_Vn_neg,0)

        # diagrama de rajada:

    #Definida a função de rajada, definir agora as 4 retas.

        self.rajada1_pos = self.rajada(self.rajada_vc,1)
        self.rajada1_neg = self.rajada(self.rajada_vc,-1)

        self.rajada2_pos = self.rajada(self.rajada_vd,1)
        self.rajada2_neg = self.rajada(self.rajada_vd,-1)
```

```


     # Definindo o Diagrama de rajada após a velocidade Vc
        self.rajada1_pos[self.v_range >= self.v_c] =  (self.rajada1_pos[self.v_range == self.v_c]
                                                     - ((self.rajada1_pos[self.v_range == self.v_c]
                                                      - self.rajada2_pos.max())/(self.v_d -self.v_c))*(self.v_range[self.v_range >= self.v_c]-self.v_c))
        self.rajada1_neg[self.v_range >= self.v_c] =  (self.rajada1_neg[self.v_range == self.v_c]
                                                     + ((self.rajada1_neg[self.v_range == self.v_c]
                                                         -self.rajada2_neg.min())/(self.v_d -self.v_c))*(-self.v_range[self.v_range >= self.v_c]+self.v_c))

    # Organizando os dataframes para o plot final do V-n:
        #Etapas definidas para manipulação de dados e organização do plot do Diagrama.
        self.Vn_data_pos = pd.DataFrame({"v":self.v_range,"n_vn":self.N_Vn_pos,"n_raj":self.rajada1_pos})
        self.Vn_data_neg = pd.DataFrame({"v":self.v_range,"n_vn":self.N_Vn_neg,"n_raj":self.rajada1_neg})


        self.Vn_data_pos = self.Vn_data_pos[self.Vn_data_pos["v"]>=self.v_estol]
        self.Vn_data_neg = self.Vn_data_neg[self.Vn_data_neg["v"]>=self.v_estol_neg]

        self.Vn_data_pos["n_intersec"] = self.Vn_data_pos.apply(lambda row:max(row['n_vn'], row['n_raj'])if row['v'] > self.v_manobra else row['n_vn'], axis=1)
        self.Vn_data_pos.at[self.Vn_data_pos.index[-1], 'n_intersec'] = 0
        self.Vn_data_neg["n_intersec"] = self.Vn_data_neg.apply(lambda row:min(row['n_vn'], row['n_raj'])if row['v'] > self.v_manobra_neg else row['n_vn'], axis=1)
        self.Vn_data_neg.at[self.Vn_data_neg.index[-1], 'n_intersec'] = 0


        self.x_vn_first_points = [self.Vn_data_pos["v"].iloc[0],self.Vn_data_pos["v"].iloc[0],self.Vn_data_neg["v"].iloc[0],self.Vn_data_neg["v"].iloc[0]]
        self.y_vn_first_points = [self.Vn_data_pos["n_intersec"].iloc[0],0,0,self.Vn_data_neg["n_intersec"].iloc[0]]


    def Plot_Vn(self):

        # Definir configurações do gráfico:
        fig = plt.figure(dpi = 100)
        ax = fig.add_axes([0,0,1.5,1])
        ax.grid(ls='--', c = 'grey',axis = 'y',alpha = 0.4)
        ax.set_facecolor('azure')
        ax.set_xlabel('V [m/s]')
        ax.set_ylabel("n [g's]")

        x = self.v_range
        y = self.N_Vn_pos
        y_neg = self.N_Vn_neg

        ax.plot(self.v_range,self.rajada1_pos,"--", c = "green",alpha = 0.6, label = "Diagrama de Rajada")
        ax.plot(self.v_range,self.rajada1_neg,"--", c = "green",alpha = 0.6)
        ax.plot(self.v_range,self.rajada2_pos,"--", c = "green",alpha = 0.6)
        ax.plot(self.v_range,self.rajada2_neg,"--", c = "green",alpha = 0.6)
```

```
        ax.fill_between(x,y,y_neg, color = 'lightgrey',alpha = 0.2,zorder = 2)
        ax.plot(x,y,"grey",ls = "--")
        ax.plot(x,y_neg,"grey", ls = "--")

        ax.fill_between(self.Vn_data_pos["v"],self.Vn_data_pos["n_intersec"], color = 'turquoise', label= 'envelope de voo',alpha = 0.5,zorder = 2)
        ax.fill_between(self.Vn_data_neg["v"],self.Vn_data_neg["n_intersec"], color = 'turquoise' ,alpha = 0.5,zorder = 2)

        ax.plot(self.Vn_data_pos["v"],self.Vn_data_pos["n_intersec"],c = "midnightblue")
        ax.plot(self.Vn_data_neg["v"],self.Vn_data_neg["n_intersec"], c = "midnightblue")

        ax.plot(self.x_vn_first_points,self.y_vn_first_points, c = "midnightblue")

        self.ponto_critico =  self.Vn_data_pos[ self.Vn_data_pos["n_intersec"]==self.Vn_data_pos["n_intersec"].max()]

        ax.plot(self.ponto_critico["v"],self.ponto_critico["n_intersec"],"ro",label = "ponto crítico")
        ax.legend()

def Vn_plot(n,v, aircraft_weight):
    Vn = Vn_diagram(n,wing_area,v,Cl_max,aircraft_weight * 10 ,rho,cl_alpha,med_chord,rajada_vc,rajada_vd)
    Vn.Plot_Vn()

aircraft_weight = wd.FloatSlider(value = 9, min = 7, max = 14,step = 0.025, description = 'weight [kg]')
n = wd.FloatSlider(value = 2.3, min = 1.8, max = 3,step = 0.05, description = 'n [gs]')
v = wd.FloatSlider(value = 18, min = 10, max = 25,step = 0.25, description = 'v [m/s]')# porcentagem da corda - longarina principal

wd.interact(Vn_plot,n = n, v = v, aircraft_weight = aircraft_weight)
```



APÊNDICE B – CÓDIGO DO CÁLCULO DE ESFORÇOS INTERNOS

```
wing_data = pd.read_excel("C:\\TCC\\wing_data2.xlsx",sheet_name="Wing")
wing_data
b = wing_data["y-span"]
chord = wing_data["Chord"]
Cl = wing_data["Cl"]
Cm = wing_data["Cm (chord/4)"]
v = 15.24
L = (1/2)*Cl*rho*v**2*chord

M_c4 = (1/2)*Cm*rho*v**2*chord**2
```

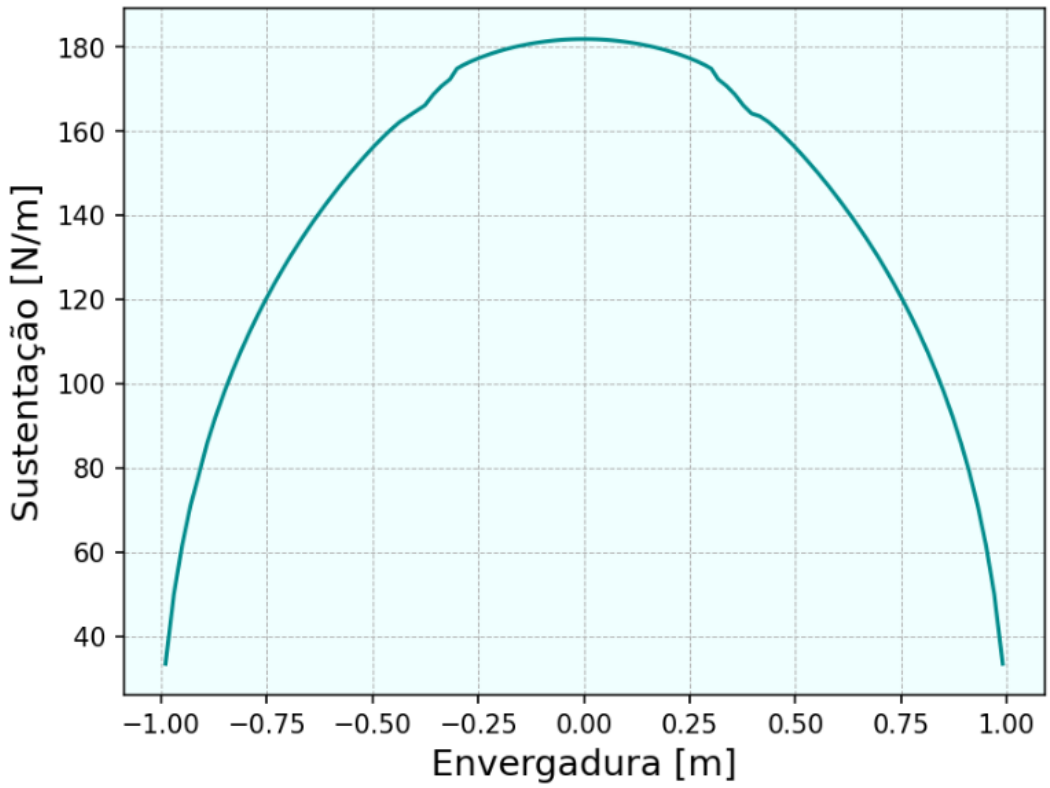
```
# CALCULO DE ESFORÇOS INTERNOS
d = 0.02 # [m]
V = []; M = []; T = []

lift = InterpolatedUnivariateSpline(b, L, k=1)

torsion = InterpolatedUnivariateSpline(b, M_c4, k=1)

F = lift.integral(b.min(),b.max()) #Cálculo da sustentação total da aeronave. Utilizado para calcular a carga total nos apoios
T_reac = torsion.integral(b.min(),b.max())

for Y in b:

    if Y < (-d/2) :

        V += [ lift.integral(b.min(),Y) ]
        T += [ torsion.integral(b.min(),Y) ]

    elif Y > (-d/2) and Y < (d/2) :

        V += [  lift.integral(b.min(),Y) - F/2]
        T += [  torsion.integral(b.min(),Y) - T_reac/2]

    else:

        V += [  lift.integral(b.min(),Y) - F]
        T += [  torsion.integral(b.min(),Y) - T_reac]

cort = InterpolatedUnivariateSpline(b, V, k=1)

for Y in b:

    M += [cort.integral(b.min(),Y) ]

```

```
#Dataframe que irá armazenar as principais infos da Asa:
aero_loads = pd.DataFrame({
                           'Envergadura [m]': b,
                           'Sustentação [N/m]': L,
                           'Momento de Torção [Nm/m': M_c4,
                           'V [N]': V,
                           'M [N m]': M,
                           'T [N m / m]': T,
                           'chord [m]': chord,
                           'cl' : Cl,
                           'cm' : Cm

})

#cálculo posição bordo de ataque e de fuga:

Offset_wingtip = 0.0592 #[m]  Inserir offset da ponta da asa - eixo x
Y_offset_wingtip = 0.021 #[m]  Inserir offset da ponta da asa - eixo z

# Para calcular a posição do bordo de ataque da asa, foi feita uma correlação com a distribuição de cordas ao longo da envergadura.
front_edge_position = (aero_loads["chord [m]"].max()-aero_loads["chord [m]"])/(aero_loads["chord [m]"].max())
front_edge_position = (front_edge_position/front_edge_position.max())*Offset_wingtip


trailing_edge_position = front_edge_position + aero_loads["chord [m]"]

#-----------------------------------------------------------------------------------------------------------------------
aero_loads["X offset [m]"] = front_edge_position
aero_loads["X offset + Chord [m]"] = trailing_edge_position
aero_loads["Z offset [m]"] = (front_edge_position/front_edge_position.max())*Y_offset_wingtip  #REVISAR
aero_loads
```

```python
def plot_per_wingspan(column):
    fig, ax = plt.subplots(dpi=150)
    ax.grid(ls='--', lw=0.5)
    ax.plot(aero_loads['Envergadura [m]'], aero_loads[column],"darkcyan")
    ax.set_facecolor('azure')
    ax.set_ylabel(column, fontsize=14)  # Using the column name directly
    ax.set_xlabel('Envergadura [m]', fontsize=14)  # Assuming 'Envergadura [m]' is the X-axis

# Assuming aero_loads and Y are defined somewhere in your code
# Y should be the DataFrame containing the columns to be plotted

column_options = list(aero_loads.columns[1:])  # Excluding the first column ('Envergadura [m]')

# Creating an interactive widget with RadioButtons
wd.interact(plot_per_wingspan, column= wd.widgets.RadioButtons(options=column_options, description='Eixo y:'))
```

Eixo y:

- (•) Sustentação [N/m]
- ( ) Momento de Torção [Nm/m
- ( ) V [N]
- ( ) M [N m]
- ( ) T [N m / m]
- ( ) chord [m]
- ( ) cl
- ( ) cm
- ( ) X offset [m]
- ( ) X offset + Chord [m]
- ( ) Z offset [m]

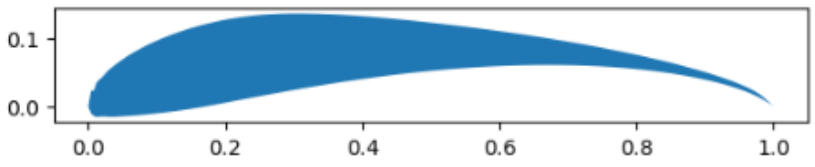


APÊNDICE C – TRATAMENTO DE DADOS DO PERFIL AERODINÂMICO

```
#Importação Perfil:
airfoil_dots = pd.read_excel("C:\\TCC\\wing_data.xlsx",sheet_name="Selig 1223")

#-------------------------------------------------------------------------------------

#encontrar indice no dataframe onde ocorre a passagem do extradorso para o intradorso:
indice_proximo_zero = (airfoil_dots['x'] - 0).abs().idxmin()

# dividir a curva entre extradorso e intradorso:
extradorso = airfoil_dots.iloc[:indice_proximo_zero + 2]
extradorso = extradorso.sort_values(by="x", ascending = True)
intradorso = airfoil_dots.iloc[indice_proximo_zero :]


# Criação de curvas interpoladas para o intradorso e extradorso
X_airfoil_interpol = np.linspace(0.00005,1,250) #(primeiro parâmetro é próximo de zero pois ocorre erro no método)

Y_airfoil_extra_interpol = interp1d(extradorso["x"], extradorso["y"], kind='cubic')
Y_airfoil_intra_interpol = interp1d(intradorso["x"], intradorso["y"], kind='cubic')

Y_airfoil_extra = (Y_airfoil_extra_interpol(X_airfoil_interpol))
Y_airfoil_intra = (Y_airfoil_intra_interpol(X_airfoil_interpol))


#-------------------------------------------------------------------------------------

airfoil_data = pd.DataFrame({"x": X_airfoil_interpol,"y pos":Y_airfoil_extra,"y neg":Y_airfoil_intra})

plt.fill_between(airfoil_data["x"],airfoil_data["y pos"],airfoil_data["y neg"])
plt.axis("scaled")
airfoil data
```

	x	y pos	y neg
0	0.000050	-0.001251	-0.000758
1	0.004066	0.022300	-0.010913
2	0.008082	0.022338	-0.014439
3	0.012098	0.033915	-0.016089
4	0.016113	0.038540	-0.015673
...	...	...	...
245	0.983937	0.014024	0.009834
246	0.987952	0.010634	0.007531
247	0.991968	0.006972	0.005118
248	0.995984	0.003302	0.002609
249	1.000000	0.000000	0.000000

250 rows × 3 columns



APÊNDICE D – CÓDIGO DA CLASSE WING_STRUCTURE

```python
# Essa classe vai calcular as propriedades em uma secção transversal da asa

e_chap = 1/32
def area_pos(x,y,e):  #vetor x, vetor y e espessura.

        xmed = (x[1::]+x[0:-1])/2
        ymed = (y[1::]+y[0:-1])/2
        A = (np.sqrt((x[1::]-x[0:-1])**2+(y[1::]-y[0:-1])**2))*e
        vector = np.array([xmed,ymed,A])
        return vector

# Essa é a classe mais importante desse programa. O objetivo dela é calcular todas as propriedades da Asa em uma secção transversal.
# Seus inputs serão:
# - As cargas do DataFrame aeroloads
# - A curva do perfil aerodinâmico airfoil_data (no formato x, y_pos, y_neg)
# - Posição da longarina 1 [% da corda na raíz da asa]
# - Posição da longarina 2 [% da corda na raíz da asa]
# - Espessuras das longarinas e do chapeado
# - i é o contador da secção transversal. Equivale ao indice das linhas em aero_loads.

class Wing_structure:
    def __init__(self, aero_loads, airfoil_data,p_l1, p_l2, e_l1,e_l2,e_chap,i):

        # "i" é o nosso contador. Ele será usado para fazer a varredura na asa.
        self.i = i

        self.aero_loads = aero_loads
        self.airfoil_data = airfoil_data

        #Dados de cargas e geometria:
        self.b = self.aero_loads["Envergadura [m]"].iloc[i]
        self.V = self.aero_loads["V [N]"].iloc[i]
        self.M = self.aero_loads["M [N m]"].iloc[i]
        self.T = self.aero_loads["T [N m]"].iloc[i]
        self.chord = self.aero_loads["chord [m]"].iloc[i]
        self.chord_max = self.aero_loads["chord [m]"].max()
        self.x_offset = self.aero_loads["X offset [m]"].iloc[i]
        self.z_offset = self.aero_loads["Z offset [m]"].iloc[i]

        #Dados da longarina:

        # Posição em porcentagem da corda máxima;
        self.p_l1 = p_l1  #[%]
        self.p_l2 =  p_l2 #[%]

        self.inch_2_mm = 0.0254

        # inserir em metros;
        self.e_l1 = e_l1 * self.inch_2_mm
        self.e_l2 = e_l2 * self.inch_2_mm
        self.e_chap = e_chap * self.inch_2_mm

        # Cálcular pontos da longarina 1 e longarina 2.
        self.x_l1 = (self.p_l1/100) * self.chord_max
        self.x_l2 = (self.p_l2/100) * self.chord_max

        #secção transversal (calculando as curvas dos perfis nas secções transversais)
        self.cross_section = self.airfoil_data * self.chord
        self.cross_section["x"] =  self.cross_section["x"] + self.x_offset
        self.cross_section["y pos"] =  self.cross_section["y pos"] + self.z_offset
        self.cross_section["y neg"] =  self.cross_section["y neg"] + self.z_offset


        #Cálcular pontos l1 e l2.
        self.coord_l1 = self.cross_section[self.cross_section["x"] <= self.x_l1].iloc[-1]
        self.coord_l2 = self.cross_section[self.cross_section["x"] <= self.x_l2].iloc[-1]

        #Altura de l1 e de l2:
        self.h_l1 = self.coord_l1["y pos"] - self.coord_l1["y neg"]
        self.h_l2 = self.coord_l2["y pos"] - self.coord_l2["y neg"]

        #Curva chapeado:
        self.chapeado_shape = self.cross_section[self.cross_section["x"] <= self.x_l1]

        #Pontos nas longarinas:
        self.zz_l1 = np.linspace(self.coord_l1["y neg"],self.coord_l1["y pos"],3)
        self.zz_l2 = np.linspace(self.coord_l2["y neg"],self.coord_l2["y pos"],3)

        self.xx_l1 = self.coord_l1["x"] * np.ones(len(self.zz_l1))
        self.xx_l2 = self.coord_l2["x"] * np.ones(len(self.zz_l2))

        self.yy = self.b * np.ones(len(self.zz_l2))
```

```
        #Área de l1 e l2:

        self.A_l1 = self.e_l1 * self.h_l1
        self.A_l2 = self.e_l2 * self.h_l2

        # Centro de l1 e l2:

        self.centro_l1 = [self.coord_l1["x"], (self.coord_l1["y pos"] + self.coord_l1["y neg"])/2]
        self.centro_l2 = [self.coord_l2["x"], (self.coord_l2["y pos"] + self.coord_l2["y neg"])/2]

        #Curva chapeado:
        self.chapeado_shape = self.cross_section[self.cross_section["x"] <= self.x_l1]

        # Cálculo da área sob o chapeado:
        self.chapeado_area = np.trapz(self.chapeado_shape['y pos'] - self.chapeado_shape['y neg'], x=self.chapeado_shape['x'])

        # Cálculo do comprimento da curva:

        self.chapeado_length = 0
        for i in range(1, len(self.chapeado_shape)):
            dx = self.chapeado_shape['x'].iloc[i] - self.chapeado_shape['x'].iloc[i - 1]
            dy_positivo = self.chapeado_shape['y pos'].iloc[i] - self.chapeado_shape['y pos'].iloc[i - 1]
            dy_negativo = self.chapeado_shape['y neg'].iloc[i] - self.chapeado_shape['y neg'].iloc[i - 1]
            segmento_positivo = np.sqrt(dx**2 + dy_positivo**2)
            segmento_negativo = np.sqrt(dx**2 + dy_negativo**2)
            self.chapeado_length += (segmento_positivo + segmento_negativo)

        # Calculando o centroide

        x = self.chapeado_shape["x"]
        z_pos = self.chapeado_shape["y pos"]
        z_neg = self.chapeado_shape["y neg"]


        self.Area_Chap_sup = area_pos(np.array(x.tolist()),np.array(z_pos.tolist()),self.e_chap)
        self.Area_Chap_inf = area_pos(np.array(x.tolist()),np.array(z_neg.tolist()),self.e_chap)

        #vector = np.array([xmed,ymed,A])
        self.Area_chapeado = (sum(self.Area_Chap_sup[2]) + sum(self.Area_Chap_inf[2]))
        self.x_centroide_chapeado = (sum(self.Area_Chap_sup[2]*self.Area_Chap_sup[0])+ sum(self.Area_Chap_inf[2]*self.Area_Chap_inf[0]))/ self.Area_chapeado
        self.z_centroide_chapeado =(sum(self.Area_Chap_sup[2]*self.Area_Chap_sup[1])+ sum(self.Area_Chap_inf[2]*self.Area_Chap_inf[1]))/ self.Area_chapeado
```

```
        #Momento Inércia Chapeado
        self.Ixx_chap = sum(self.Area_Chap_sup[2]*(self.x_centroide_chapeado-self.Area_Chap_sup[0])**2) + sum(self.Area_Chap_inf[2]*(self.x_centroide_chapeado-self.Area_Chap_inf[0])**2)
        self.Izz_chap = sum(self.Area_Chap_sup[2]*(self.z_centroide_chapeado-self.Area_Chap_sup[1])**2) + sum(self.Area_Chap_inf[2]*(self.z_centroide_chapeado-self.Area_Chap_inf[1])**2)

        #Cálculo do Centróide da Secção Transversal:

        self.x_centroide = (self.A_l1 * self.centro_l1[0] + self.A_l2 * self.centro_l2[0] + self.x_centroide_chapeado * self.Area_chapeado)/(self.A_l1 + self.A_l2 + self.Area_chapeado)
        self.z_centroide = (self.A_l1 * self.centro_l1 [1] + self.A_l2 * self.centro_l2[1] + self.z_centroide_chapeado * self.Area_chapeado)/(self.A_l1 + self.A_l2 + self.Area_chapeado)

        #CÁLCULO DE TENSÕES

        self.Ixx_l1 = (self.h_l1 ** 3 * self.e_l1) / 12
        self.Ixx_l2 = (self.h_l2 ** 3 * self.e_l2) / 12

        self.Izz_l1 = (self.h_l1 * self.e_l1 ** 3) / 12
        self.Izz_l2 = (self.h_l1 * self.e_l1 ** 3) / 12

        #Cálculo do Momento de Inércia total da secção transversal:

        self.Ixx = ((self.Ixx_l1 + self.A_l1 * (self.centro_l1[1] - self.z_centroide) ** 2) + (self.Ixx_l2 + self.A_l2 * (self.centro_l2[1] - self.z_centroide ) ** 2)
                    + (self.Ixx_chap + self.Area_chapeado * (self.z_centroide_chapeado - self.z_centroide) **2))
        self.Izz =  ((self.Izz_l1 + self.A_l1 * (self.centro_l1[0] - self.x_centroide) ** 2) + (self.Ixx_l2 + self.A_l2 * (self.centro_l2[0] - self.x_centroide ) ** 2)
                     + (self.Izz_chap + self.Area_chapeado * (self.x_centroide_chapeado - self.x_centroide) **2))
        self.Ixz = ((self.A_l1 * (self.centro_l1[1] - self.z_centroide) * (self.centro_l1[0] - self.x_centroide)
                     + self.A_l2 * (self.centro_l2[1] - self.z_centroide) * (self.centro_l2[0] - self.x_centroide)
                     + self.Area_chapeado * (self.z_centroide_chapeado - self.z_centroide) * (self.x_centroide_chapeado - self.x_centroide) ) )
        # Cálculo das tensões devido a flexão da Asa:

        #Sigma  l1 e l2:
        self.sigma_l1 = (-self.M * (self.Izz * (self.zz_l1 - self.z_centroide)- self.Ixz * (self.xx_l1 - self.x_centroide))/(self.Ixx * self.Izz - self.Ixz**2)) * 1e-6
        self.sigma_l2 = (-self.M * (self.Izz * (self.zz_l2 - self.z_centroide)- self.Ixz * (self.xx_l2 - self.x_centroide))/(self.Ixx * self.Izz - self.Ixz**2)) * 1e-6

        self.sigma_chap_intra = (-self.M * (self.Izz * (self.chapeado_shape["y neg"] - self.z_centroide) - self.Ixz * (self.chapeado_shape["x"] - self.x_centroide))/(self.Ixx * self.Izz - self.Ixz**2)) * 1e-6



        # Cálculo das tensões devido a torção da asa:
        self.q_torsion = self.T / ( 2 * self.chapeado_area)
        self.tau_torsion_l1 = (self.q_torsion / self.e_l1) / 1e6
        self.tau_torsion_chap = (self.q_torsion / self.e_chap) / 1e6
```

```
        #Tau
        self.tau_l1 = (self.V *  ((self.zz_l1 + self.coord_l1["y pos"])/2 - self.z_centroide) * (self.coord_l1["y pos"] - self.zz_l1) * (self.e_l1 + self.e_l2))/(self.Ixx * self.e_l1 * 1e6)
        self.tau_l2 = (self.V *  ((self.zz_l2 + self.coord_l2["y pos"])/2 - self.z_centroide) * (self.coord_l2["y pos"] - self.zz_l2) * (self.e_l1 + self.e_l2))/(self.Ixx * self.e_l2 * 1e6)


def plot_cross_section(p11, p12, e_l1, e_l2,i):

    data = Wing_structure(aero_loads, airfoil_data,p11, p12, e_l1,e_l2,e_chap,i)
    fig = plt.figure(dpi=150)
    ax = fig.add_subplot(1, 1, 1, aspect='equal')
    ax.grid(ls='--', lw=0.5)
    ax.set_facecolor('azure')
    ax.set_ylabel("Z [m]", fontsize=14)  # Using the column name directly
    ax.set_xlabel('X [m]', fontsize=14)  # Assuming 'Envergadura [m]' is the X-axis
    ax.plot([0,0,0.7],[0.15,0,0], color = "lightgrey", ls = "--")

    #Plote Chapeado:
    ax.plot(data.chapeado_shape["x"], data.chapeado_shape["y pos"], color = "darkblue",linewidth = 1 , )
    ax.plot(data.chapeado_shape["x"], data.chapeado_shape["y neg"], color = "darkblue",linewidth = 1 , )
    ax.fill_between(data.chapeado_shape["x"], data.chapeado_shape["y pos"],data.chapeado_shape["y neg"], color = "dodgerblue")

    #
    ax.fill_between(data.cross_section["x"],data.cross_section["y pos"],data.cross_section["y neg"],alpha = 0.5, color = "deepskyblue")

    ax.plot(data.xx_l1, data.zz_l1, color = "darkblue", linewidth = 10*e_l1, label = "$l_{1}$")
    ax.plot(data.xx_l2, data.zz_l2, color = "seagreen", linewidth = 10*e_l2, label = "$l_{2}$")
```

```
#  ax.plot(data.x_centroide_chapeado,data.z_centroide_chapeado,"o")

    ax.plot(data.x_centroide,data.z_centroide,"o",color = "red", label = "$centroide$")

    ax.legend()

#-------------------------------------------------------------------------------
#Definindo dados das longarinas:

p_l1 = wd.FloatSlider(value = 24, min = 14, max = 28,step = 0.50, description = 'L1 [% corda]:') # porcentagem da corda - longarina principal
p_l2 = wd.FloatSlider(value = 40, min = 32, max = 45,step = 0.50, description = 'L2 [% corda]') # porcentagem da corda - longarina secundária

e_l1 = main_beam_thickness = wd.RadioButtons(value = 1/4,options = [1/8,1/4], description = 'thickness L1 [pol]') #Como mostrar o 1/8 e o 1/4 como frações
e_l2 = main_beam_thickness = wd.RadioButtons(value = 1/8,options = [1/8,1/4], description = 'thickness L2 [pol]') #Como mostrar o 1/8 e o 1/4 como frações
#e_chap = 3/32 * inch_2_mm
#-------------------------------------------------------------------------------

i = wd.IntSlider(value = int(len(aero_loads["Envergadura [m]"])/2), min = 0, max = len(aero_loads["Envergadura [m]"])-1,step = 1, description = 'i') # porcentagem da corda - longarina secund



#calculando a posição das longarinas na asa:

l1_position = aero_loads["chord [m]"].max() * (p_l1.value/100) * np.ones(len(aero_loads["chord [m]"]))
l2_position = aero_loads["chord [m]"].max() * (p_l2.value/100) * np.ones(len(aero_loads["chord [m]"]))



#-------------------------------------------------------------------------------------------------------------------------------------------------------
def plot_beams(l1,l2,enverg,front_offset, back_offset):


    l1_pos = aero_loads["chord [m]"].max() * (11/100)
    l2_pos = aero_loads["chord [m]"].max() * (12/100)


    fig1 = plt.figure(dpi = 100)
    ax = fig1.add_axes([0,0,1.5,1])
    ax.grid(ls='--', c = 'grey',axis = 'y',alpha = 0.4)
    ax.set_facecolor('azure')
    ax.set_xlabel('x [m]')
    ax.set_ylabel("y [m]")

    # Plotar os dados nos eixos
    #ax.plot(enverg,front_offset,"blue")
    #ax.plot(enverg,back_offset,"blue")
    ax.fill_between(enverg,front_offset,back_offset,color = "turquoise", alpha = 0.8)


    ax.plot(enverg,l1_pos * np.ones(len(aero_loads['Envergadura [m]'])),"midnightblue")
    ax.plot(enverg,l2_pos * np.ones(len(aero_loads['Envergadura [m]'])),"midnightblue")
    print()

    plt.axis("scaled")
#-------------------------------------------------------------------------------------------------------------------------------------------------------
#wd.interact(plot_beams, l1 = p_l1, l2 = p_l2, enverg = wd.fixed(aero_loads['Envergadura [m]']), front_offset = wd.fixed(front_edge_position), back_offset = wd.fixed(trailing_edge_position))
#display(e_l1,e_l2)
```

```
wd.interact(plot_cross_section, pl1 = p_l1, pl2 = p_l2,e_l1 = e_l1, e_l2 = e_l2, i = i)
```

L1 [% corda]: 24.00

L2 [% corda] 40.00

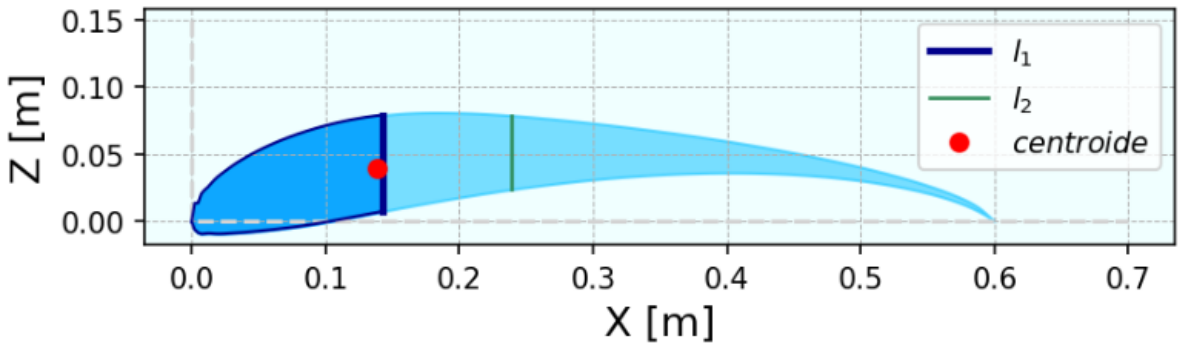
thickness L1 [pol]

- 0.125
- (•) 0.25

thickness L2 [pol]

- (•) 0.125
- 0.25

i 60




```
<function __main__.plot_cross_section(pl1, pl2, e_l1, e_l2, i)>
```

APÊNDICE E – CÓDIGO DAS FUNÇÕES BEAMS_VARIATION E TSAI-HILL

```
p_l1_range = np.arange(p_l1.min , p_l1.max + p_l1.step,p_l1.step)
p_l2_range = np.arange(p_l2.min , p_l2.max + p_l2.step,p_l2.step)

# Variação da posição das longarinas:

e_l1_thickness = 1/4
e_l2_thickness = 1/4
e_chap = 1/32

def beams_variation(p_l1_range, p_l2_range,e_l1_thickness,e_l2_thickness):
    pl1_dot = []
    pl2_dot = []

    sigma_max_l1 = []
    sigma_min_l1 = []

    sigma_max_l2 = []
    sigma_min_l2 = []

    tau_max = []
    Izz = []
    Ixx = []
    massa = []
    tau_max_l1 = []
    tau_max_chap = []

    for pl1 in p_l1_range:
        for pl2 in p_l2_range:

            data = Wing_structure(aero_loads, airfoil_data,pl1,pl2, e_l1_thickness,e_l2_thickness,e_chap,61)
            sigma_max_l1.append(data.sigma_l1.max())
            sigma_max_l2.append(data.sigma_l2.max())
            sigma_min_l1.append(data.sigma_l1.min())
            sigma_min_l2.append(data.sigma_l2.min())
            tau_max_l1.append(data.tau_torsion_l1)
            tau_max_chap.append(data.tau_torsion_chap)
            pl1_dot.append(pl1)
            pl2_dot.append(pl2)
            Izz.append(data.Izz)
            Ixx.append(data.Ixx)
            massa.append(Wing_mass(pl1,pl2,rho_balsa,vertices,e_l1_thickness, e_l2_thickness,e_chap))

    beams_data = pd.DataFrame({ "posição l1 [%]": pl1_dot,
                                "posição l2 [%]": pl2_dot,
                                "sigma max l1 [MPa]" : sigma_max_l1,
                                "sigma max l2 [MPa]" : sigma_max_l2 ,
                                "sigma min l1 [MPa]" : sigma_min_l1,
                                "sigma min l2 [MPa]" : sigma_min_l2 ,
                                "tau max torção l1 [MPa]" : tau_max_l1,
                                "tau max torção chap [MPa]" : tau_max_chap,
                                "massa [kg]" : massa })
    return(beams_data)

dados = beams_variation(p_l1_range, p_l2_range, e_l1_thickness,e_l2_thickness)
dados
```

	posição l1 [%]	posição l2 [%]	sigma max l1 [MPa]	sigma max l2 [MPa]	sigma min l1 [MPa]	sigma min l2 [MPa]	tau max torção l1 [MPa]	tau max torção chap [MPa]	massa [kg]
0	14.0	32.0	5.448005	5.244644	-5.821453	-5.333513	0.234693	1.877540	0.125686
1	14.0	32.5	5.466897	5.237238	-5.851696	-5.314473	0.234693	1.877540	0.125290
2	14.0	33.0	5.505019	5.220113	-5.912323	-5.274836	0.234693	1.877540	0.124897
3	14.0	33.5	5.524229	5.210382	-5.942697	-5.254195	0.234693	1.877540	0.124586
4	14.0	34.0	5.543527	5.199861	-5.973101	-5.232974	0.234693	1.877540	0.124272
...	...	...	...	...	...	...	...	...	...
778	28.0	43.0	1.877131	1.704356	-2.269134	-1.366598	0.097045	0.776359	0.146247
779	28.0	43.5	1.875414	1.695421	-2.274576	-1.346950	0.097045	0.776359	0.145921
780	28.0	44.0	1.873681	1.686379	-2.279920	-1.327289	0.097045	0.776359	0.145597
781	28.0	44.5	1.871933	1.677237	-2.285165	-1.307617	0.097045	0.776359	0.145165
782	28.0	45.0	1.868395	1.658691	-2.295357	-1.268267	0.097045	0.776359	0.144746

```python
#Dados da madeira Balsa:

sigma_tracao_balsa_max = 10.12
sigma_compressao_balsa_max = 8.05
tau_max_balsa =  1.35 # Resistencia a tração na direção perpendicular as fibras da madeira

#Fator de segurança:
# Segundo as normas da FAR ....
FS = 1.5

#Tensões admissíveis:
sigma_tracao_adm = sigma_tracao_balsa_max / FS
sigma_compressao_adm = sigma_compressao_balsa_max / FS
tau_adm_max = tau_max_balsa / FS


# Função Tsai Hill:

def Tsai_Hill(sigma_1 , sigma_2,tau, sigma_adm_1, sigma_adm_2, tau_adm):
    Tsai_Hill = ( sigma_1 / sigma_adm_1 )**2 + ( sigma_2 / sigma_adm_2 )**2
    - ((sigma_1 * sigma_2)/sigma_adm_1) + np.sqrt(( tau / tau_adm)**2)
    return(Tsai_Hill)

Analise_Otimização_Longarina = dados
Analise_Otimização_Longarina["Tsai Hill tração L1"] = Tsai_Hill(dados["sigma max l1 [MPa]"],
                                                               0,dados["tau max torção chap [MPa]"],
                                                               sigma_tracao_adm , 1 , tau_adm_max )
Analise_Otimização_Longarina["Tsai Hill compressão L1"] = Tsai_Hill(dados["sigma min l1 [MPa]"],
                                                                   0,dados["tau max torção chap [MPa]"],
                                                                   sigma_compressao_adm , 1 , tau_adm_max )


```

APÊNDICE F - CÓDIGO DO PLOTE DO GRÁFICO DE OTIMIZAÇÃO

```python
import matplotlib.patches as mpatches
from matplotlib.lines import Line2D
# Arrays de dados
x = np.array(compression_data["posição 12 [%]"])
y = np.array(compression_data["posição 11 [%]"])
z = np.array(compression_data["massa [kg]"])
z2 = np.array(compression_data["Tsai Hill compressão L1"])

# Criar uma grade para os dados
xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)

# Interpolação para os dados de z e z2
zi = griddata((x, y), z, (xi[None, :], yi[:, None]), method='cubic')
zii = griddata((x, y), z2, (xi[None, :], yi[:, None]), method='cubic')


# Configurar o gráfico conforme o código inicial
wing_colormap, wcmap = plt.subplots(dpi=200)
wcmap.grid(ls='--', lw=0.5, color="white", alpha=0.6)
wcmap.set_facecolor('azure')
wcmap.set_ylabel('Posição L1 [%]', fontsize=12)
wcmap.set_xlabel('Posição L2 [%]', fontsize=12)
wcmap.tick_params(axis="both", which='major', labelsize=12)

cmap_massa = wcmap.contourf(xi, yi, zi, 15, cmap="jet")

barra_de_cores = plt.colorbar(cmap_massa)
barra_de_cores.set_label('Massa [kg]', fontsize=12)

levels_resistencia = np.linspace(np.nanmin(zii), 1, 10)
curvanivel_resistencia = wcmap.contour(xi, yi, zii, levels=levels_resistencia, colors="black", linestyles="--", linewidths=0.8)
curvanivel_falha = wcmap.contourf(xi, yi, zii, levels=[1, np.nanmax(zii)], colors='darkgrey', alpha=0.95)

plt.clabel(curvanivel_resistencia, inline=True, fontsize=9, fmt=" %.2f ")


# Criar uma legenda para "Falha" com a cor cinza claro
falha_patch = mpatches.Patch(color='darkgrey', label='Falha')
falha_patch2 = Line2D([0], [0], color='black', linestyle='--', label='Tsai-Hill')
```