UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ESTRUTURAL

E CONSTRUÇÃO CIVIL

DAVI CARNEIRO PERES

DESENVOLVIMENTO DE UM PROGRAMA PARA O DIMENSIONAMENTO DE BLOCOS DE COROAMENTO RÍGIDOS SOB MÚLTIPLAS COMBINAÇÕES DE ESFORÇOS

FORTALEZA

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P51d Peres, Davi Carneiro.
Desenvolvimento de um programa para o dimensionamento de blocos de coroamento rígidos sob múltiplas combinações de esforços / Davi Carneiro Peres. – 2026.
121 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Civil, Fortaleza, 2026.
Orientação: Prof. Dr. Augusto Teixeira de Albuquerque.

1. Blocos de coroamento. 2. Programa de dimensionamento. 3. Modelo de Biela e Tirante. 4. Python. 5. Dimensionamento Estrutural. I. Título.

CDD 620

DAVI CARNEIRO PERES

DESENVOLVIMENTO DE UM PROGRAMA PARA O DIMENSIONAMENTO DE BLOCOS DE COROAMENTO RÍGIDOS SOB MÚLTIPLAS COMBINAÇÕES DE ESFORÇOS

Monografia apresentada ao Curso de Bacharelado em Engenharia Civil da Universidade Federal do Ceará, como requisito para obtenção do grau de Bacharel em Engenharia Civil.

Orientador: Prof. Dr. Augusto Teixeira de Albuquerque.

FORTALEZA

2026

DAVI CARNEIRO PERES

DESENVOLVIMENTO DE UM PROGRAMA PARA O DIMENSIONAMENTO DE BLOCOS DE COROAMENTO RÍGIDOS SOB MÚLTIPLAS COMBINAÇÕES DE ESFORÇOS

Monografia apresentada ao Curso de Bacharelado em Engenharia Civil da Universidade Federal do Ceará, como requisito para obtenção do grau de Bacharel em Engenharia Civil, apreciada pela Banca Examinadora composta pelos seguintes membros:

Aprovada em _06_/_01_/__2026__.

BANCA EXAMINADORA

_____

Prof. Dr. Augusto Teixeira de Albuquerque (Orientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Joaquim Eduardo Mota (Examinador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dra. Marcela Moreira da Rocha Almeida (Examinador)

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

RESUMO

O dimensionamento de blocos de coroamento sobre estacas é uma etapa fundamental na Engenharia Civil, cuja complexidade aumenta com a verticalização das edificações e a magnitude das cargas aplicadas. O processo de cálculo atual apresenta uma dicotomia: softwares comerciais de alto custo e planilhas eletrônicas acessíveis, mas que exigem a entrada manual de dados e são suscetíveis a erros, especialmente na análise de múltiplas combinações de carregamento. Observou-se, portanto, uma lacuna na disponibilidade de ferramentas automatizadas e acessíveis que otimizassem este fluxo de trabalho. Diante disso, este trabalho objetivou o desenvolvimento de um programa em linguagem Python, com interface gráfica intuitiva, para o dimensionamento automatizado de blocos de coroamento rígidos. A ferramenta foi desenvolvida para ser capaz de processar as combinações de esforços de força normal e momentos fletores, identificando a situação crítica para o dimensionamento. Com base nos modelos de cálculo, como o Modelo de Bielas e Tirantes, o programa possibilitou determinar a altura mínima do bloco e as armaduras necessárias. A proposta preencheu a lacuna existente, oferecendo uma solução eficiente, confiável e gratuita que contribuiu para a democratização do acesso a ferramentas de cálculo, o aumento da produtividade e a redução de erros no dimensionamento, servindo também como um recurso operacional e educacional valioso.

Palavras-chave: Blocos de Coroamento Rígidos, Dimensionamento Estrutural, Automação, Python, Modelo de Biela e Tirante.

ABSTRACT

The structural design of pile caps is a fundamental stage in Civil Engineering, the complexity of which increases with the verticalization of buildings and the magnitude of applied loads. The current calculation process presents a dichotomy: high-cost commercial software and accessible spreadsheets, which require manual data entry and are susceptible to errors, especially in the analysis of multiple load combinations. Therefore, a gap was observed in the availability of automated, accessible, and transparent tools to optimize this workflow. Given this, this work aimed to develop a program in Python, with an intuitive graphical interface, for the automated design of rigid pile caps. The tool was developed to be capable of processing combinations of normal force and bending moments, identifying the critical situation for the design. Based on calculation models such as the Strut-and-Tie Model, the program enabled the determination of the minimum height of the cap and the necessary reinforcement. The proposal filled the existing gap, offering an efficient, reliable, and free solution that contributed to the democratization of access to calculation tools, increased productivity, and reduced design errors, serving also as a valuable operational and educational resource.

Keywords: Rigid Pile Caps, Structural Design, Automation, Python, Strut-and-Tie Model.

AGRADECIMENTOS

Primeiramente, agradeço a Deus, por ter me concedido força e resiliência para que meus objetivos fossem alcançados, iluminando meu caminho durante todos os anos de estudo e permitindo que eu chegasse a este momento de conclusão.

Aos meus pais, Dioges e Ednizia, pilares da minha vida. Agradeço imensamente pelo apoio incondicional, pelos sacrifícios silenciosos e por todo o incentivo que me deram. Vocês são a base de quem sou e de tudo que conquistei até aqui.

Ao meu orientador, Professor Augusto Albuquerque, pela confiança depositada em mim e no meu trabalho e por suas correções.

À Professora Verônica, pelos ensinamentos fundamentais na disciplina de Projeto de Graduação, que contribuíram significativamente para o aprimoramento da minha escrita acadêmica e estruturação deste trabalho.

Aos demais professores do curso de Engenharia Civil da UFC, que compartilharam seu conhecimento e exigiram o melhor de mim, permitindo-me construir uma base técnica sólida para o exercício da profissão.

Aos meus colegas de turma e amigos, pela parceria nos momentos de estudo, pelo companheirismo diante dos desafios e por tornarem essa jornada de descobertas e aprendizado mais leve e memorável.

LISTA DE SIMBOLOS

α: Ângulo de inclinação da biela;

F_{ck}: Resistência característica do concreto;

F_{cd}: Resistência de cálculo do concreto;

F_{yd}: Resistência de cálculo do aço;

γ_c: Coeficiente de minoração da resistência do concreto;

γ_f: Coeficiente de ponderação das cargas;

N_k: Esforço vertical característico;

N_d: Esforço vertical de cálculo;

M_x: Momento em torno do eixo y;

M_y: Momento em torno do eixo x;

l: Distância entre eixos das estacas;

h: Altura do bloco;

d: Altura útil do bloco;

A_b: Largura no eixo x do bloco;

B_b: Largura no eixo y do bloco;

a_p: Largura no eixo x do pilar;

b_p: Largura no eixo y do pilar;

$a_{p,eqv}$: Largura equivalente do pilar;

$\emptyset_{est}$: Diâmetro da estaca;

K_r: Coeficiente de Rüsch;

$\sigma_{cd,lim}$: Tensão de compressão limite da biela;

A_p: Área do pilar;

A_{est}:Área da estaca;

$A_{b,pil}$: Área da biela no contato com o pilar;

$A_{b,est}$: Área da biela no contato com a estaca;

$\sigma_{cd,b,est}$: Tensão de compressão da biela no contato com a estaca;

$\sigma_{cd,b,pil}$: Tensão de compressão da biela no contato com o pilar;

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O advento do concreto armado em meados do século XIX, combinado à invenção do elevador moderno, catalisou uma revolução na engenharia civil, quebrando a barreira da altura e possibilitando o crescimento vertical das edificações. Este processo de verticalização, que hoje se estende globalmente para além das grandes metrópoles (GARREFA; GUERRA, 2011), impôs novos e complexos desafios à engenharia de estruturas.

No cenário brasileiro, essa evolução construtiva é notável ao longo do último século. Um marco desta progressão é o contraste entre o primeiro arranha-céu da América Latina, o edifício: “A Noite”, inaugurado no Rio de Janeiro em 1929 com cerca de 102 metros de altura, e o contemporâneo “One Tower” em Balneário Camboriú, cujo arranha-céu atinge 290 metros. Com o surgimento desses edifícios cada vez mais altos, o elevado número de pavimentos reflete diretamente na ordem de grandeza dos esforços transmitidos às fundações. Consequentemente, as soluções de fundações superficiais tornaram-se insuficientes para muitos projetos, exigindo uma análise cada vez mais cautelosa dos elementos de fundação (AGUIAR, 2024). Essa necessidade consolidou o uso de fundações profundas, como estacas e tubulões, capazes de transferir as cargas para camadas de solo mais resistentes.

Nesse contexto, os blocos de coroamento se apresentam como elementos de transição indispensáveis, projetados para receber os esforços concentrados dos pilares e distribuí-los de forma segura e equilibrada para o conjunto de estacas. No entanto, seu dimensionamento envolve verificações que requerem a consideração de múltiplas combinações de ações, conforme estabelecido pela ABNT NBR 6122:2022. A necessidade de avaliar diferentes casos de carregamentos, aliada a repetitividade dos cálculos, torna esse procedimento suscetível a erros quando realizado manualmente, além de demandar um tempo significativo do projetista.

Com o avanço tecnológico, os softwares comerciais se consolidaram e passaram a desempenhar papéis centrais no projeto estrutural. Contudo, tais ferramentas, por serem plataformas robustas e em constante atualização para atender às revisões normativas, o elevado custo de licenciamento representa uma barreira para estudantes e determinados profissionais. Como alternativa, surge a utilização de planilhas eletrônicas que embora acessíveis, exigem processos manuais e repetitivos e apresentam menos robustez frente à verificação de múltiplas combinações.

Diante da lacuna existente entre ferramentas de alto custo e planilhas com processos manuais onerosos, o presente trabalho se justifica pela necessidade de desenvolver uma solução que democratize o acesso a ferramentas deste tipo, superando essa barreira de custo dos

softwares já existentes. Simultaneamente, busca-se mitigar a suscetibilidade a erros inerentes ao uso de planilhas, otimizando o fluxo de trabalho através da automatização do dimensionamento a partir da leitura dos esforços da planta de cargas dos pilares.

A ABNT NBR 6118:2023 define os blocos de coroamento como as estruturas responsáveis por transmitir os esforços da superestrutura para as estacas ou tubulões. Para os blocos de comportamento rígido, alvos deste trabalho, um dos métodos que representa seu dimensionamento é o Modelo de Bielas e Tirantes, que depende da formação de escoras de compressão internas, exigindo uma altura de bloco suficiente para garantir uma inclinação adequada na transmissão das cargas conforme destaca Fusco (1994).

Portanto, o objetivo geral deste trabalho é o desenvolvimento de um programa, com interface gráfica intuitiva, para o dimensionamento automatizado de blocos de coroamento rígidos. A ferramenta deverá permitir ao usuário a entrada de parâmetros como os esforços atuantes no pilar, a geometria do arranjo de estacas e a resistência do concreto. Com base nesses dados, o programa realizará o cálculo das cargas em cada estaca, a verificação da altura mínima do bloco e o dimensionamento das armaduras necessárias, traduzindo as recomendações das bibliografias em um processo automático e eficiente.

2 PROBLEMA DE PESQUISA E QUESTÃO MOTIVADORA

2.1 Problema de pesquisa.

Apesar da consolidação teórica dos modelos de dimensionamento para blocos de coroamento rígidos, a prática de projeto enfrenta gargalos operacionais para projetos muito grandes. Ademais, observa-se uma dicotomia no mercado: de um lado, softwares robustos, porém onerosos; do outro, métodos manuais ou planilhas eletrônicas que, embora acessíveis, introduzem um risco de falha humana no processamento de múltiplas combinações de esforços. O problema de pesquisa centra-se, assim, na carência de uma ferramenta computacional que integre a leitura de combinações de esforços, a flexibilidade geométrica e a transparência metodológica.

2.2 Questões motivadoras

a) Interface Gráfica:

Quais recursos de interface e funcionalidades de sistema são essenciais para simplificar a modelagem geométrica do bloco e garantir a praticidade operacional do programa no cotidiano de projeto?

b) Impacto na Eficiência:

De que forma a automação da leitura de esforços pode otimizar a identificação dos esforços críticos para o dimensionamento em blocos de pilares submetidos a múltiplas combinações?

c) Desafios da Implementação:

Quais são os principais desafios técnicos e conceituais ao traduzir as prescrições dos métodos, particularmente o Modelo de Bielas e Tirantes e o Método do CEB-70, em um algoritmo computacional robusto e preciso, especialmente considerando diferentes arranjos de estacas e carregamentos?

d) Confiabilidade:

Os resultados da implementação computacional dos roteiros de cálculo demonstram plena convergência com os resultados obtidos através de soluções analíticas?

3 OBJETIVOS

3.1 Objetivo geral

Desenvolver um programa, capaz de dimensionar múltiplas variações de blocos de coroamento rígidos, a partir da importação de uma planilha de esforços solicitantes

3.2 Objetivos específicos

a) Interface Gráfica:

Desenvolver uma interface gráfica que permita a modelagem completa do bloco, integrando recursos de entrada de dados e um sistema de arquivos para o armazenamento e recuperação de estudos anteriores.

b) Leitura de Esforços dos Pilares:

Desenvolver um módulo para a importação e interpretação de uma planilha contendo os esforços dos pilares para suas múltiplas combinações de esforços.

c) Método de Cálculo:

Implementar computacionalmente os algoritmos de dimensionamento fundamentados no Modelo de Bielas e Tirantes e no Método da Flexão, abrangendo o cálculo das reações nas estacas, a verificação dos critérios de rigidez e a determinação das áreas de aço necessárias.

d) Validação:

Validar as funcionalidades e os resultados gerados pelo programa através da comparação dos resultados obtidos de forma analítica.

4 DELINEAMENTO E LIMITAÇÕES DO TRABALHO

O programa desenvolvido neste trabalho foca em resolver um problema prático específico de forma acessível e transparente. Como tal, é fundamental delimitar o escopo da ferramenta. As principais limitações são:

a) Foco Exclusivo em Blocos Rígidos:

Toda a metodologia de cálculo implementada (Modelo de Bielas e Tirantes e Método do CEB-70) assume um comportamento de corpo rígido. O dimensionamento de blocos flexíveis (que se comportam como lajes e exigem uma análise de flexão mais complexa) não é contemplado.

b) Verificação Geotécnica (Carga Admissível):

A ferramenta calcula a máxima reação de compressão ($P_{i,máx}$) nas estacas a partir das combinações fornecidas. No entanto, o programa não realiza a verificação geotécnica. É responsabilidade do engenheiro projetista comparar estas reações com a capacidade de carga admissível (P_{adm}) da estaca, fornecida pelo projetista de fundações.

c) Premissa de Simetria (Método de Alonso):

A validade do método de cálculo das reações nas estacas (superposição de efeitos) pressupõe que o arranjo do estaqueamento seja simétrico, fazendo com que os eixos principais de inércia do conjunto coincidam com os eixos x e y de aplicação dos momentos. Ao utilizar a função de "Entrada Customizada" de coordenadas, o usuário é responsável por garantir que esta premissa seja atendida.

d) Escopo dos Pilares:

O programa foi desenvolvido para o dimensionamento de blocos sob a ação de um único pilar central. Casos de pilares múltiplos sobre um mesmo bloco ou pilares com seções transversais muito alongadas ou complexas os quais alteram a distribuição das tensões não são cobertos por este estudo.

e) Esforços Horizontais (Fx e Fy):

O dimensionamento considera apenas os esforços de Força Normal (Fz) e Momentos Fletores (Mx e My). Forças horizontais na base do pilar (Fx e Fy), que podem induzir esforços adicionais no bloco e nas estacas, não são considerados.

f) Verificação ao Cisalhamento no Método da Flexão:

No que tange ao Método CEB-FIP, a ferramenta automatiza o cálculo dos momentos fletores e das respectivas armaduras de flexão A_{sx} e A_{sy}. Verificações adicionais como a verificação da força cortante, não integram a rotina atual, devendo ser verificadas conforme a necessidade do projeto.

g) Dimensionamento vs. Detalhamento:

A ferramenta é focada no dimensionamento (fornecendo os valores das áreas de aço necessárias para as armaduras principais e complementares) e não no detalhamento. O programa não gera desenhos, não calcula comprimentos de ancoragem ou o espaçamento final das barras, etapas que permanecem a cargo do projetista.

4.1 Fluxo de desenvolvimento do projeto

Para o desenvolvimento deste trabalho, será seguido o fluxograma apresentado a seguir na Figura 1:

Figura 1 – Fluxograma de etapas do trabalho.

Fonte: Autoria Própria.

O projeto se inicia pela etapa de **Fundamentação Teórica**, dedicada ao levantamento bibliográfico e normativo necessário para embasar os critérios de engenharia adotados. Na sequência, procede-se ao **Desenvolvimento do Programa**, fase que compreende a construção da ferramenta computacional em si e na tradução dos roteiros de cálculo para o código. Posteriormente, realiza-se a **Verificação dos Resultados**, etapa destinada a validar a conformidade do programa. O trabalho encerra-se com as **Considerações Finais**, onde são sintetizadas as conclusões obtidas e as contribuições do projeto.

5 FUNDAMENTAÇÃO TEÓRICA

5.1 Blocos rígidos

A NBR 6118:2023 classifica os blocos de coroamento em duas categorias:

> **Bloco Rígido**
> *O comportamento estrutural se caracteriza por:*
>
> > *a) trabalho à flexão nas duas direções, usualmente simulado por bielas e tirantes, mas com trações essencialmente concentradas nas linhas sobre as estacas (reticulado definido pelo eixo das estacas, com faixas de largura igual a 1,2 vez seu diâmetro);*
> >
> > *b) forças transmitidas do pilar para as estacas essencialmente por bielas de compressão, de forma e dimensões complexas;*
> >
> > *c) trabalho ao cisalhamento também em duas direções, não apresentando ruínas por tração diagonal, e sim por compressão das bielas, analogamente às sapatas.*
>
> **Bloco flexível**
> *Para esse tipo de bloco, deve ser realizada uma análise mais completa, desde a distribuição dos esforços nas estacas, dos tirantes de tração, do cisalhamento, até a necessidade da verificação da punção.*

A norma destaca também, que os modelos de cálculo a serem adotados devem ser consistentes com o comportamento estrutural do elemento. Desta maneira, considerando que o foco deste trabalho são os blocos rígidos, diante de suas características específicas, um dos modelos de cálculo mais adequados para o dimensionamento é o Modelo de Bielas e Tirantes que segundo Bastos (2023) é o modelo mais utilizado no Brasil, por se tratar um modelo de treliças intuitivo e possuir amplo suporte experimental.

Contudo, é importante ressaltar que a aplicabilidade e a representatividade deste modelo podem encontrar limitações em configurações geométricas mais complexas. Sakai (2010) ao revisar diversas bibliografias sobre o tema, incluindo os trabalhos de Blévot e Frémy (1967), além de outros como Moraes (1976), Alonso (1989) e Munhoz (2004), observa que a maioria dos autores descreve o comportamento das bielas predominantemente para blocos de até seis estacas. O autor aponta que estudos experimentais ou numéricos para blocos com arranjos mais extensos são consideravelmente menos comuns na literatura.

Essa perspectiva é corroborada por Souza e Delalibera (2022), que em seu estudo sobre o dimensionamento de um bloco de 12 estacas, destacam a existência de dúvidas quanto à real configuração das bielas em blocos sobre múltiplas estacas.

Nesse sentido, Ramos (2007) argumenta que, em virtude dessa incerteza quanto à formação de um sistema de bielas bem definido, o modelo convencional de bielas e tirantes

muitas vezes não é o mais empregado para blocos com grande número de estacas. Como alternativa para esses casos, o autor sugere a utilização do Método do CEB-FIP (1970).

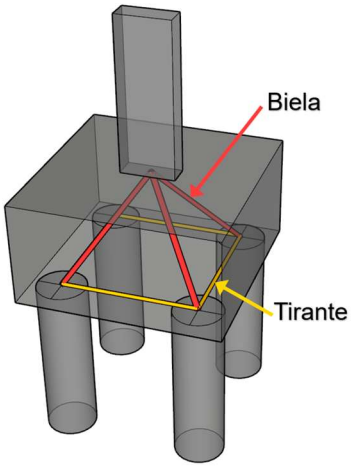
5.2 Bielas e tirantes

A origem do Modelo de Bielas e Tirantes remonta aos estudos de Ritter e Mörsch, no meio do século XIX (MÖRSCH, 1948), os quais observaram que o padrão de fissuração em vigas podia ser comparado ao comportamento de uma treliça. Nesse modelo idealizado, o banzo inferior é representado pelas armaduras tracionadas, enquanto as bielas inclinadas de concreto comprimido assumem o papel das diagonais da treliça.

Com base nesse conceito inicial, Blévot e Frémy (1967) deram continuidade ao desenvolvimento teórico ao aplicar o modelo de treliça ao comportamento de blocos de coroamento sobre estacas. Seus estudos representaram um marco importante na compreensão da formação das bielas de compressão nesses elementos. Por meio de ensaios experimentais realizados em modelos reduzidos, os autores propuseram um modelo racional, pautado no equilíbrio estático dos esforços internos. Durante os experimentos, foram avaliadas diferentes configurações geométricas de blocos de coroamento, o que permitiu o desenvolvimento de equações específicas para o dimensionamento das bielas e dos tirantes, possibilitando o cálculo direto das forças atuantes e da armadura necessária.

Fundamentalmente, o modelo assume que o equilíbrio do sistema depende exclusivamente da geometria do bloco e da direção das forças aplicadas, desconsiderando os efeitos de flexão. Conforme explicam Souza e Bittencourt (2006), no Modelo de Bielas e Tirantes são idealizadas bielas inclinadas de concreto, que formam o caminho direto da carga desde a base do pilar até o topo das estacas. Em contrapartida, as tensões de tração que surgem para equilibrar o sistema são representadas pelos tirantes, que são materializados pela armadura de aço principal posicionada entre as estacas. A Figura 2, representa o posicionamento das bielas e dos tirantes no bloco.

Figura 2 - Representação do modelo de bielas e tirantes.



Fonte: Autoria própria.

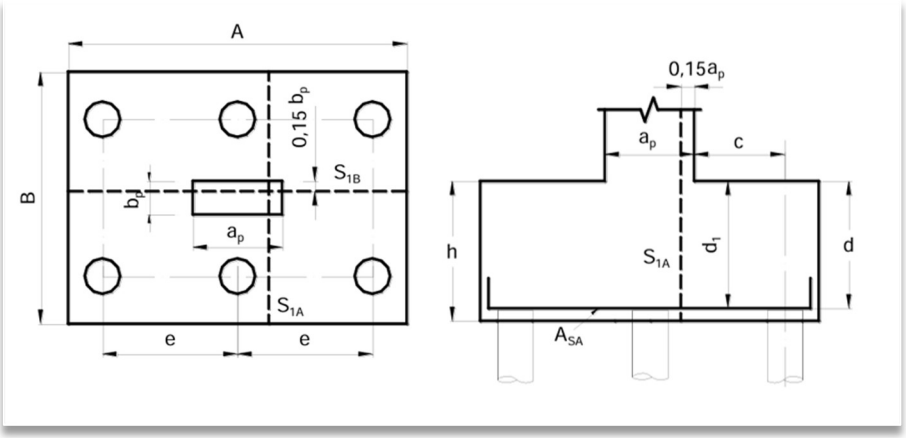
5.3 CEB-FIP (1970)

Conforme discutido anteriormente, para blocos com um número elevado de estacas, o Modelo de Bielas e Tirantes torna-se complexo e sua idealização, incerta. Como alternativa para esses casos, a literatura técnica, como aponta Ramos (2007), recomenda o método baseado na Teoria da Flexão, proposto no Boletim 73 do CEB-70.

Esta metodologia propõe que o dimensionamento da armadura principal seja realizado fundamentalmente para resistir aos esforços de flexão, tratando o bloco de forma análoga a uma laje espessa ou viga rígida.

O passo fundamental deste método, após o cálculo das reações nas estacas, é a determinação dos momentos fletores de cálculo (M_d). Para isso, o método define seções críticas (S), posicionadas no interior do pilar distanciadas ($0{,}15l$) da face do pilar, onde (l) é a dimensão do pilar na direção analisada. A Figura 3 ilustra o posicionamento dessas seções.

Figura 3 - Representação do Método do CEB-70.

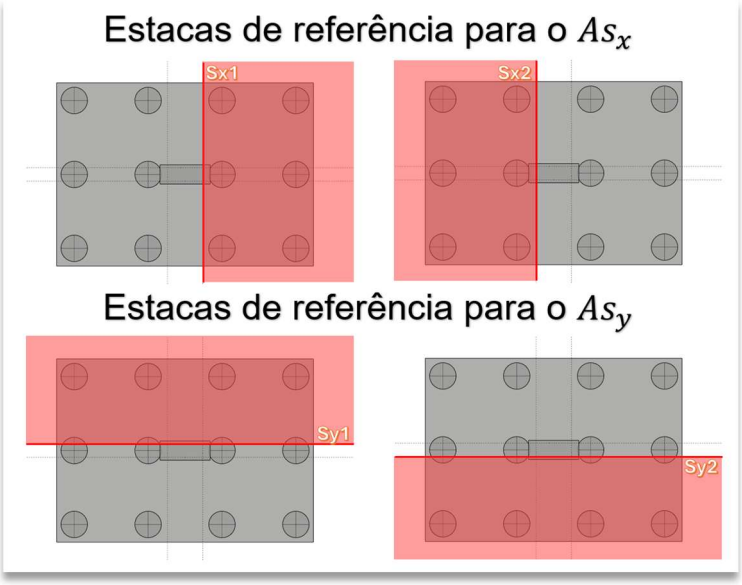


Fonte: Bastos (2023).

O momento fletor em cada seção é calculado somando os momentos gerados pelas reações de todas as estacas localizadas além daquela seção, ou seja, na região entre a seção de referência e a borda externa do bloco.

Para um dimensionamento completo, devem ser analisadas quatro seções, uma em cada sentido do pilar (duas seções para o eixo X e duas para o eixo Y) a fim de determinar a maior armadura naquela direção, conforme demonstra a Figura 4.

Figura 4 - Representação das seções nas duas direções.



Fonte: Autoria própria.

Matematicamente, os momentos fletores de cálculo nas direções X e Y são obtidos pelas somatórias:

$$M_x = \sum_i R_i \, |x_i - (0{,}5 - 0{,}15) l_x| \tag{1}$$

$$M_y = \sum_i R_i \, |y_i - (0{,}5 - 0{,}15) l_y| \tag{2}$$

Onde:

$R_i - Reação\ da\ estaca\ i;$

$(x_i, y_i) - Coordenadas\ do\ eixo\ da\ estaca\ i;$

$(l_x, l_y) - Dimensões\ do\ pilar\ nas\ direções\ x\ e\ y;$

Uma vez determinados os momentos fletores máximos, a área de aço necessária (A_S) é calculada perpendicularmente à seção de referência e governada pelo maior momento

encontrado naquela direção. O dimensionamento utiliza a hipótese de flexão simplificada, na qual o braço de alavanca interno (z) é estimado como ($0{,}85 \cdot d$). Essa simplificação é amplamente aceita para o dimensionamento de blocos de coroamento, pois considera a elevada rigidez do elemento, dispensando o refinamento do cálculo da posição exata da linha neutra no elemento. A equação resultante é:

$$As_x = \frac{M_{d,x}}{0{,}85\ d\ f_{yd}} \tag{3}$$

$$As_y = \frac{M_{d,y}}{0{,}85\ d\ f_{yd}} \tag{4}$$

$As - Armaduras\ do\ bloco\ nas\ direções\ x\ e\ y;$

$M_d - Momento\ de\ cálculo\ nas\ direções\ x\ e\ y;$

$d - Altura\ útil\ do\ bloco;$

$f_{yd} - Resistência\ de\ cálculo\ do\ aço;$

Esta abordagem, baseada no método das seções, simplifica significativamente o dimensionamento de arranjos complexos com muitas estacas, oferecendo um roteiro seguro para a determinação da armadura principal.

5.3.1 Verificação ao cisalhamento

Além do dimensionamento à flexão, este método preconiza a realização de verificações de segurança contra a ruptura por cisalhamento. Recomenda-se a análise de dois mecanismos principais de falha: o cisalhamento composto, que envolve a força cortante gerada por um conjunto de estacas além de uma seção estabelecida, e a verificação da força cortante localizada nas estacas de canto.

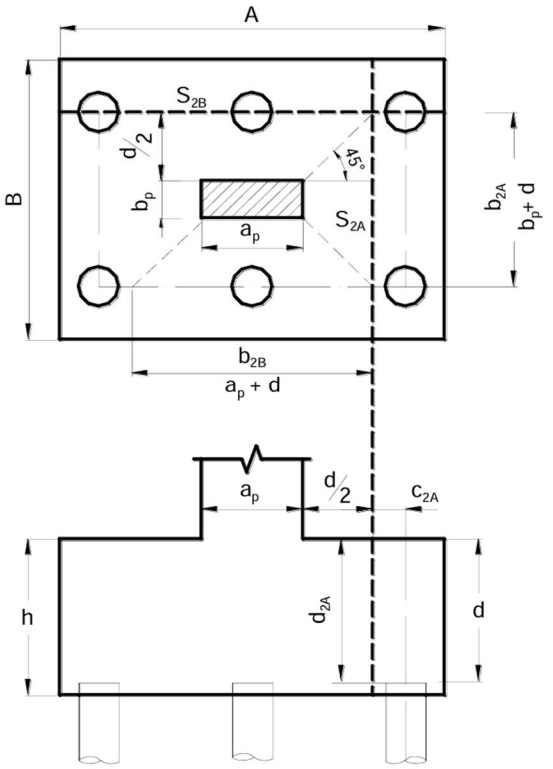
5.3.2 Verificação ao cisalhamento composto

Esta verificação analisa o bloco como uma viga larga. Para isso, define-se uma seção de referência S_2, crítica para o esforço cortante.

1. Posição da Seção S_2:

   A seção deve ser traçada a uma distância $d/2$ da face do pilar, perpendicularmente à direção analisada. Caso o eixo de alguma estaca se encontre entre a face do pilar e a seção crítica S_2, esta seção deve ser reposicionada para a face do pilar. A Figura 5, demonstra o posicionamento da seção no bloco.

Figura 5 - Representação da seção S_2 no método do CEB70



Fonte: Bastos (2023).

2. Cálculo da Força Cortante Solicitante (V_d):

A força cortante de cálculo na seção S_2 é obtida pelo somatório das reações de todas as estacas posicionadas além desta seção.

$$V_d = \sum_i R_{i,d} \tag{5}$$

3. Cálculo da Força Cortante Resistente $V_{d,lim}$:

A resistência do elemento é fornecida pela seção de concreto definida por ($b_{2A}\ x\ d_{2A}$) na qual d_{2A} é a altura útil do bloco naquela seção e b_{2A} é a largura obtida através da seguinte equação:

$$b_{2A} = b_p + d_{2A} \tag{6}$$

Para o caso em que o bloco possua uma altura útil constante, temos que ($d = d_1 = d_2$)

Com isso, a força cortante limite é dada por:

$$V_{d,lím}(kN) = \frac{0{,}25}{\gamma_c}\left(1 - \frac{c_{2A}}{5d}\right) b_{2A} d_{2A} \sqrt{f_{ck}\left(\frac{kN}{cm^2}\right)} \tag{7}$$

$c_{2A} - Distância\ da\ seção\ S_{2A}\ até\ o\ eixo\ da\ estaca.$

Com: $d_{2A} \leq 1{,}5\ c_{2A}$

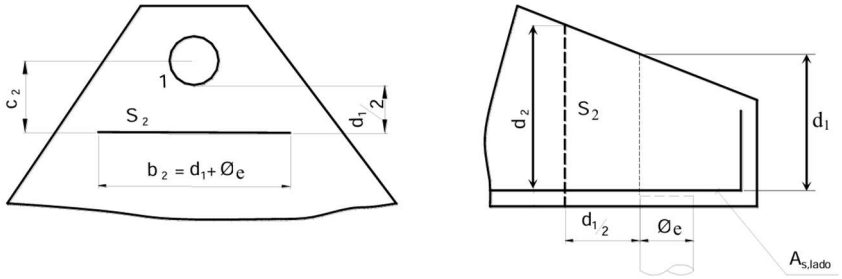
Para garantir a segurança, deve-se verificar que ($V_{sd} \leq V_{rd}$). Caso esta condição não seja atendida, faz-se necessário aumentar a altura do bloco ou a resistência do concreto.

5.3.3 Verificação ao cisalhamento na estaca de ponta

Para a verificação da estaca de ponta, temos que a seção de referência S_2 deve ser posicionada a uma distância de $d_1/2$ da face da estaca, perpendicular a direção que liga a estaca ao eixo do bloco, na qual d_1 é a altura útil do bloco na face da estaca. Neste caso, a seção para verificação é definida por ($b_2\ x\ d_2$) onde d_2 é a altura útil na seção S_2 como demonstra a Figura 6.

$$b_2 = d_1 + \emptyset_{est} \tag{8}$$

Figura 6 - Seção S para a estaca da ponta.



Fonte: Bastos (2023).

Da mesma maneira, que a verificação do cisalhamento composto, a verificação da estaca da ponta, consiste em garantir que a reação de cálculo da estaca (R_i) seja menor que a força cortante resistente ($R_{d,lím}$) naquela seção. A condição de segurança é dada por:

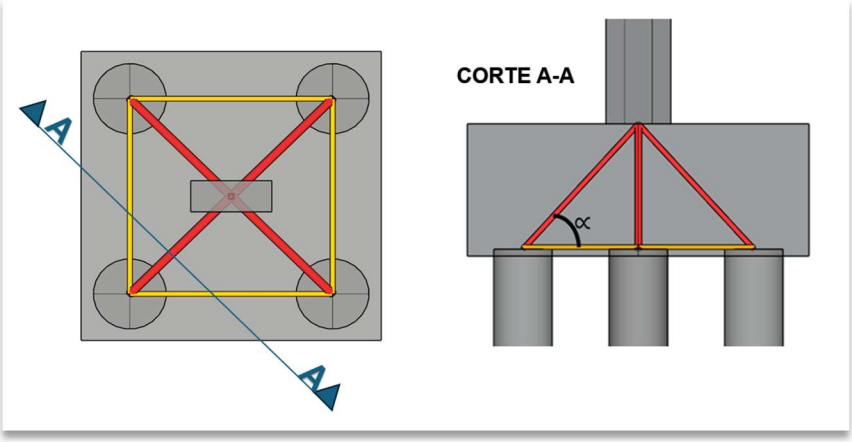
$$R_{d,lím} = \frac{0{,}12}{\gamma_c} b_2 d_2 \sqrt{f_{ck}} \tag{9}$$

Com: $R_i \leq R_{d,lím}$

5.4 Inclinações das bielas

Garantir que o bloco apresente um comportamento rígido é essencial, pois isso assegura a formação de um sistema de escoras internas bem definido, capaz de transferir as cargas do pilar para as estacas de forma eficiente. Um parâmetro crucial para essa verificação é a inclinação (α) das bielas de compressão em relação ao plano horizontal como demonstra a Figura 7.

Figura 7 - Representação da inclinação da biela.

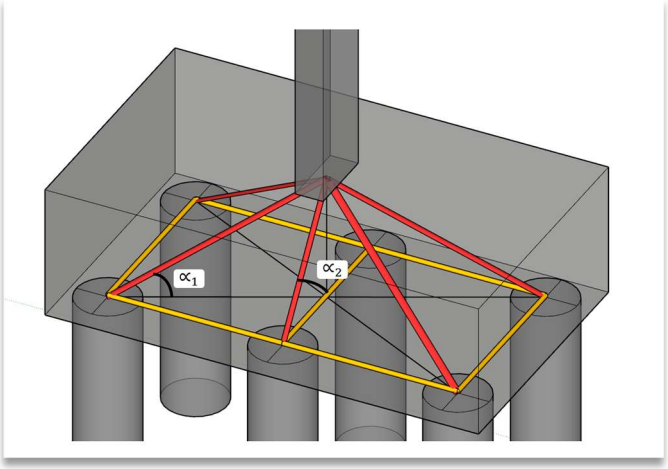


Fonte: Autoria própria.

Diversos autores estabelecem recomendações para esta inclinação. O método simplificado de Blévot e Frémy (1967), por exemplo, considera um intervalo válido para α entre 40° e 55°, sendo preferíveis valores próximos a 45° para garantir um comportamento mais seguro e eficiente.

É importante notar que, em um mesmo bloco, as bielas correspondentes a estacas em diferentes posições poderão ter inclinações distintas como demonstra a Figura 8. Fusco (1994) afirma que, por razões de segurança, a inclinação da biela correspondente à estaca mais afastada não deve ser inferior a 33,7°. De forma similar, Barros, Giongo e Oliveira (2014) sugerem que, para blocos com estacas mais afastadas, a biela mais crítica deve ter no mínimo uma inclinação de 40°.

Figura 8 - Bielas com inclinações diferentes.



Fonte: Autoria própria.

A fim de também garantir a rigidez, é comum encontrar na literatura técnica brasileira e em versões anteriores da NBR 6118 a utilização de critérios geométricos simplificados, como a relação abaixo:

$$h \geq \frac{Ab - Ap}{3} \tag{10}$$

$h - Altura\ do\ bloco;$

$A_b - Lado\ do\ bloco;$

$A_p - Lado\ do\ pilar;$

Embora práticas, essas equações podem conduzir a interpretações equivocadas sobre a rigidez real do elemento. Normas internacionais, como o ACI 318 (2014), enfatizam que a garantia do comportamento de corpo rígido não deve se basear exclusivamente nessas proporções, estabelecendo a inclinação das bielas de compressão como o critério determinante. Assim, a rigidez é assegurada quando o ângulo formado entre a biela e a horizontal respeita os limites que validam o modelo de treliça.

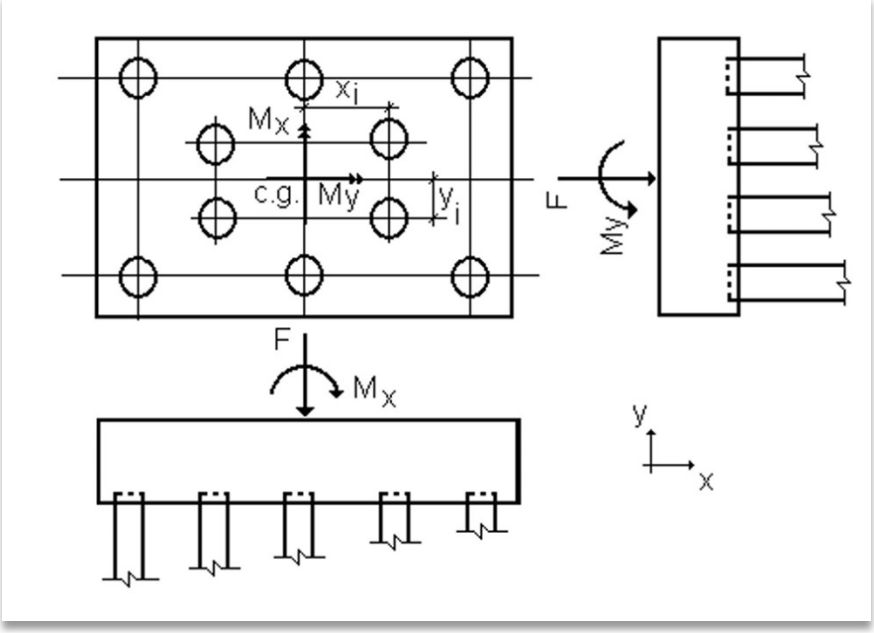
Resumindo, para arranjos que se enquadram claramente no Modelo de Bielas e Tirantes, a literatura aponta que o critério ideal é buscar uma inclinação mínima de 45°, conforme sugerido por Blévot e Frémy, para garantir um comportamento mais seguro. Já para blocos com múltiplas estacas nos quais o modelo de bielas é incerto, a verificação de rigidez ainda é necessária, sendo uma prática conservadora adotar um ângulo mínimo de 40° para a biela mais desfavorável, alinhando-se às recomendações de autores como Barros, Giongo e Oliveira (2014).

5.5 Determinação das cargas nas estacas.

A verificação das reações nas estacas é uma etapa crucial para o dimensionamento de blocos de coroamento. Essa etapa tem como objetivo determinar a reação individual de cada estaca do sistema, assegurando que a capacidade de carga destas não seja superada. A correta determinação destas reações é fundamental para prevenir recalques e potenciais danos estruturais.

Moraes (1976) observa uma importante correlação entre a rigidez do bloco e a distribuição das cargas entre as estacas. Segundo o autor, em blocos no limite mínimo de rigidez, a carga tende a se concentrar na estaca mais próxima ao pilar. Á medida que a altura do bloco cresce, conferindo-lhe maior rigidez, a distribuição dos esforços se aproxima do método da superposição de esforços o qual é descrito por ALONSO (1989). Este método, baseia-se na combinação da carga vertical com os momentos fletores atuantes conforme apresentado na Figura 9 a seguir.

Figura 9 - Representação do método de superposição dos esforços.



Fonte: Alonso (1989)

Nessas condições, para um pilar que transmite uma força normal N e momentos (M_x, M_y) em relação aos eixos principais (x, y) passando pelo centro geométrico do estaqueamento), a reação vertical N_i em uma estaca genérica "i", localizada nas coordenadas (x_i, y_i) em relação a esse centro, é dada pela seguinte expressão:

$$N_i = \frac{N}{n_{est}} + M_x \frac{x_i}{\sum {x_i}^2} - M_y \frac{y_i}{\sum {y_i}^2} \quad (11)$$

$N_i - Reação\ na\ estaca\ i;$

$N - Carga\ vertical\ atuante;$

$n_{est} - Número\ de\ estacas;$

$M_x - Momento\ em\ torno\ do\ eixo\ y;$

$M_y - Momento\ em\ torno\ do\ eixo\ x;$

$x_i - Absissa\ da\ posição\ da\ estaca;$

$y_i - Ordenada\ da\ posição\ da\ estaca;$

Para a validade da formulação de Alonso, assume-se que:

- As estacas sejam verticais;
- As estacas sejam idênticas (mesmo tipo, seção e comportamento);
- O bloco seja suficientemente rígido;
- Os eixos x e y considerados coincidam com os eixos principais de inércia do conjunto de estacas;

5.6 Combinações de cargas

Para realizar o dimensionamento dos blocos de coroamento e a verificação das tensões nas estacas, é fundamental determinar os esforços na situação mais desfavorável. Conforme esclarece Campos (2015), as cargas raramente atuam de forma isolada, sendo fundamental a análise de suas combinações para se obter os esforços críticos de projeto.

De acordo com a ABNT NBR 6120, essas combinações no Estado Limite Último são constituídas, essencialmente, por dois tipos de ações:

- Ações Permanentes: Atuam de forma contínua durante toda a vida útil da edificação (ex: peso próprio da estrutura, paredes, revestimentos).
- Ações Variáveis: Possuem intensidade que varia significativamente ao longo do tempo (ex: sobrecargas de uso de pessoas e veículos, ações do vento).

A correta combinação dessas ações, seguindo os preceitos normativos da ABNT NBR 8681, gera os esforços finais que devem ser utilizados para o dimensionamento seguro dos blocos de coroamento.

Na prática de projetos de edifícios, os relatórios de softwares de análise estrutural, como o TQS, já consolidam os resultados em um conjunto de casos críticos.

- Combinação Base (Gravitacional): Cenário que contempla a totalidade das cargas permanentes e acidentais, desconsiderando a atuação dos esforços de vento.
- Envoltórias de Esforços Máximos: Conjunto de casos que já apresentam os valores extremos finais para cada esforço isolado ($Fz_{máx}, Mx_{máx}, My_{máx}$). Estes valores são resultantes de processamentos que já consideram a atuação simultânea de diversas ações, incluindo possíveis efeitos de segunda ordem, imperfeições geométricas globais ou empuxos de solo.
- Casos de Vento Isolados: Esforços horizontais e momentos fletores gerados exclusivamente pela atuação da carga de vento nas direções principais ortogonais da edificação (Vento 0º, Vento 90º, Vento 180º, Vento 270º).

A devida interpretação desses dados é fundamental para a segurança do dimensionamento dos blocos de coroamento. Enquanto as envoltórias de máximos representam combinações finais características de carregamento, os casos de vento são frequentemente apresentados como esforços incrementais. Ou seja, exige-se que os esforços provenientes das ações do vento sejam adicionados aos esforços da Combinação Base. Esse procedimento gera novos cenários (ex: Base + Vento 0º, Base + Vento 90º, etc)

6 DIMENSIONAMENTO

6.1 Armaduras principais e secundárias

Para o Modelo de Bielas e tirantes, o detalhamento do bloco de coroamento é governado por um conjunto de armaduras que são classificadas como principais e complementares. As armaduras principais, as quais materializam os tirantes do modelo, são responsáveis por resistir aos esforços primários de tração, sendo calculadas diretamente a partir da geometria do modelo e da carga atuante. Já as armaduras complementares, são essenciais para controlar tensões secundárias a fim de assegurar a integridade e correto funcionamento do modelo. Estas incluem:

- Armadura de Malha (Superior e Inferior): Destina-se ao controle da fissuração superficial por efeitos de retração e temperatura e auxilia na distribuição de tensões localizadas, garantindo a coesão do bloco.

- Armadura de Suspensão: Esta armadura tem a função de evitar o surgimento de fissuras nas faces entre estacas ao suspender parte da força de tração para o topo do bloco. A NBR 6118 (22.7.4.1.3) afirma que "*Se for prevista armadura de distribuição para mais de 25 % dos esforços totais ou se o espaçamento entre estacas for maior que três vezes a altura do bloco, deve ser prevista armadura de suspensão para a parcela de carga a ser equilibrada.*"

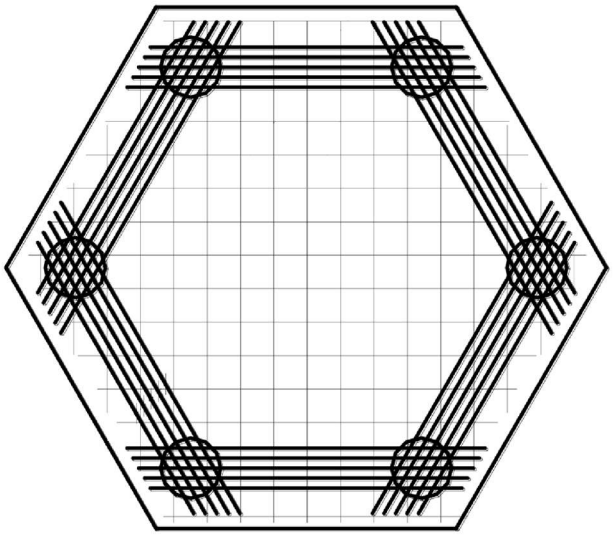
- Armadura de Costela: É a armadura horizontal disposta nas faces laterais do bloco. No Modelo de Bielas e Tirantes, sua função primordial é controlar o fendilhamento do concreto e confinar as bielas de compressão, evitando a ruptura do elemento por tensões de tração transversais.

No que tange à disposição da armadura principal, várias práticas foram empregadas ao longo do tempo como por exemplo, a distribuição de todo o aço em uma única malha inferior. Contudo, com as últimas revisões da ABNT NBR 6118 e a inclusão do (item 22.7.4.1.1), houve um esclarecimento com relação a este ponto. A norma passou a exigir que a armadura de tração seja concentrada em faixas sobre os eixos das estacas, especificando que ao menos 85% das barras de aço devem estar contidas nessas faixas.

Com isso, a disposição que atende a esta prescrição normativa, e que já era uma das mais consagrada na prática, é a que posiciona as barras de aço paralelamente aos lados do bloco, ligando os eixos das estacas. Esta abordagem, ilustrada na Figura 10, concentra o aço

exatamente onde o Modelo de Bielas e Tirantes prevê os maiores esforços de tração, resultando em um controle de fissuração mais eficiente em comparação com arranjos mais dispersos.

Figura 10 - Disposição das armaduras paralela aos lados.



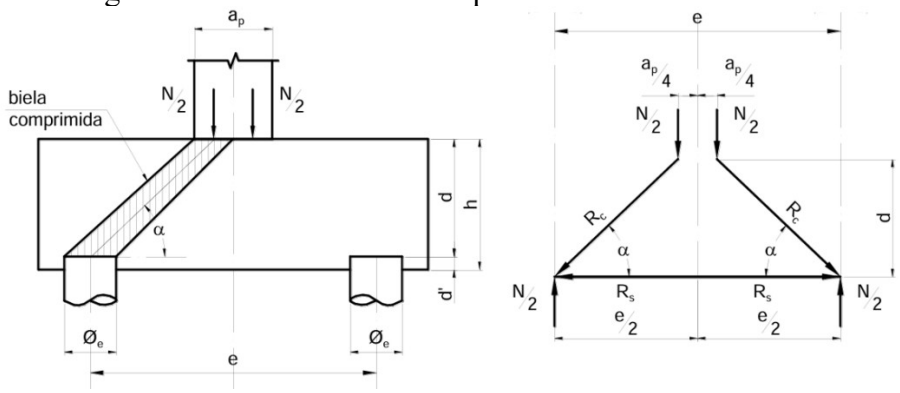
Fonte: Adaptado de Bastos (2023).

Nas seções a seguir, são apresentadas as formulações analíticas para o dimensionamento de blocos com arranjos de 2 a 6 estacas. As equações descritas para a determinação das armaduras e verificação das tensões nas bielas fundamentam-se no modelo clássico proposto por Blévot e Frémy (1967). Ressalta-se, contudo, que a abordagem adotada incorpora refinamentos e critérios mais recentes, sendo aqui reproduzidas conforme a sistematização apresentada por Bastos (2023), correlacionando a geometria do elemento com o equilíbrio estático das forças internas.

6.2 Blocos de 2 estacas

A configuração geométrica e o sistema de forças considerado para o bloco de duas estacas podem ser observados na Figura 11.

Figura 11 - Modelo de bielas para o bloco de 2 estacas.



Fonte: Adaptado de Bastos (2023).

Passo A) Determinação da inclinação das bielas e dos esforços internos.

Uma vez estabelecido o Modelo de Bielas e Tirantes, a geometria do sistema definida pela altura "d" e pela distância "l" estabelece, por semelhança de triângulos, uma proporção direta entre a reação da estaca e a força de tração no tirante.

$$\frac{\frac{N_d}{2}}{R_{sd}} = \frac{d}{\frac{l}{2} - \frac{a_{p,eqv}}{4}} \tag{12}$$

A partir desta relação, é possível determinar a inclinação da biela (α) que define a geometria do triângulo de forças, utilizando a seguinte relação trigonométrica:

$$\mathrm{tg}(\alpha) = \frac{\frac{N_d}{2}}{R_{sd}} \tag{13}$$

Substituindo:

$$\mathrm{tg}(\alpha) = \frac{d}{\frac{l}{2} - \frac{a_{p,eqv}}{4}} \tag{14}$$

$$\alpha = \mathrm{arctg}\left(\frac{d}{\frac{l}{2} - \frac{a_{p,eqv}}{4}}\right) \tag{15}$$

Após determinar o valor da inclinação da biela, é possível retornar ao sistema de equilíbrio para calcular as cargas na biela comprimida R_{cd} e no tirante tracionado R_{sd}.

$$R_{sd} = \frac{N_d}{2\ tg(\alpha)} \tag{16}$$

$$R_{cd} = \frac{N_d}{2\ sen(\alpha)} \tag{17}$$

Passo B) Verificação das tensões na biela.

Em seguida, é fundamental verificar as tensões de compressão nos nós do modelo de bielas, sendo os pontos críticos o nó na base do pilar e o nó no topo da estaca. Desta forma, as tensões atuantes calculadas devem ser inferiores às tensões limites resistentes de cálculo estabelecidas por Blévot para evitar o esmagamento da biela.

$$\sigma_{cd,b,pil} \leq \sigma_{cd,b,lim} \tag{18}$$

$$\sigma_{cd,b,est} \leq \sigma_{cd,b,lim} \tag{19}$$

Para determinar as tensões nestas duas regiões, primeiramente é necessário determinar a área de ação destas tensões, para isso, utiliza-se as seguintes equações.

$$A_{b,pil} = \frac{A_p}{2}\ sen(\alpha) \tag{20}$$

$$A_{b,est} = A_e\ sen(\alpha) \tag{21}$$

$$\sigma_{cd,b,pil} = \frac{N_d}{A_p\ sen^2(\alpha)} \tag{22}$$

$$\sigma_{cd,b,est} = \frac{N_d}{2\ A_e\ sen^2(\alpha)} \tag{23}$$

Já para a tensão limite, temos que esta é definida pela seguinte equação:

$$\sigma_{cd,b,lim} = 1{,}40\ K_r\ f_{cd} \tag{24}$$

Com: $(0{,}90 \leq K_r \leq 0{,}95)$

$K_r - Coeficiente\ de\ perda\ de\ resistência\ do\ concreto\ (Efeito\ Rüsch)$

Passo C) Determinação das armaduras.

i. Armadura Principal

A armadura principal é determinada dividindo a reação encontrada no tirante R_{sd} pela resistência de cálculo do aço f_{yd}.

$$A_s = 1.15 \cdot \frac{R_{sd}}{f_{yd}} \tag{25}$$

ii. Armadura Superior.

Neste caso, a armadura superior A_{sup} é geralmente, determinada como 20% da armadura principal A_s.

$$A_{sup} = 0{,}20 \,.\, A_s \tag{26}$$

iii. Estribos Verticais.

$$A_{s,estribos} = 0{,}075 \; B \; (cm^2/m) \tag{27}$$

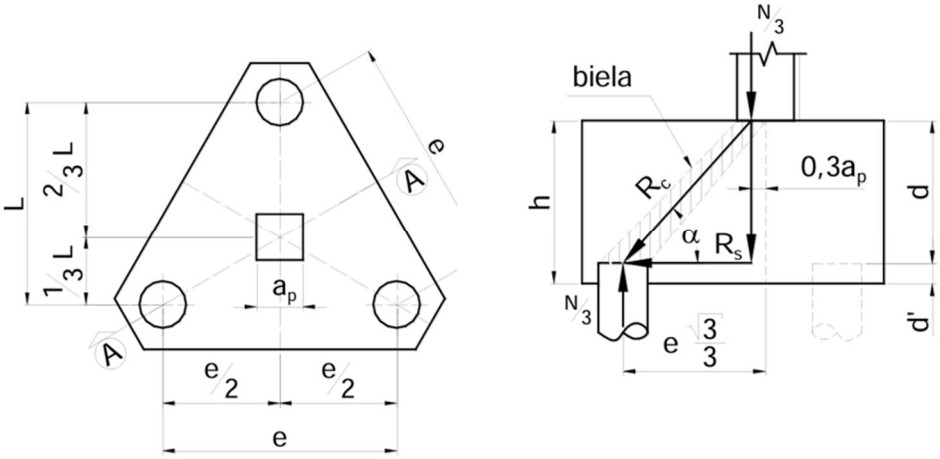
iv. Armadura de Costela.

$$A_{s,costela} = 0{,}1 \; \% \; B \,.\, h \; (cm^2) \tag{28}$$

6.3 Blocos de 3 estacas

A configuração espacial das bielas de compressão para o bloco de três estacas pode ser observada na Figura 12 a seguir.

Figura 12 – Modelo de bielas para o bloco de 3 estacas.



Fonte: Adaptado de Bastos (2023).

O dimensionamento de blocos com três ou mais estacas mantém a lógica fundamental do Modelo de Bielas e Tirantes, porém, apresenta particularidades na geometria da treliça e na distribuição dos esforços, devido à natureza tridimensional da estrutura.

Passo A) Determinação da inclinação das bielas e dos esforços internos.

$$\frac{\frac{N_d}{3}}{R_{sd}} = \frac{d}{\frac{l\sqrt{3}}{3} - 0{,}3\ a_{p,eqv}} \tag{29}$$

$$\mathrm{tg}(\alpha) = \frac{\frac{N_d}{3}}{R_{sd}} \tag{30}$$

Substituindo:

$$\mathrm{tg}(\alpha) = \frac{d}{\frac{l\sqrt{3}}{3} - 0{,}3\ a_{p,eqv}} \tag{31}$$

$$\alpha = \mathrm{arctg}\left(\frac{d}{\frac{l\sqrt{3}}{3} - 0{,}3\ a_{p,eqv}}\right) \tag{32}$$

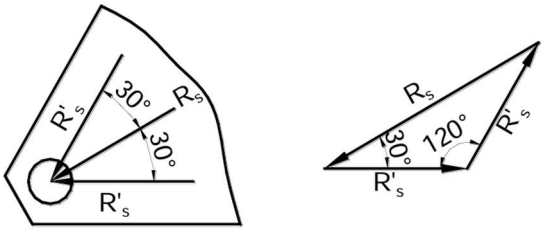
Em seguida, encontramos a carga na biela comprimida R_{cd} e no tirante tracionado R_{sd}

$$R_{sd} = \frac{N_d}{3\ tg(\alpha)} \tag{33}$$

$$R_{cd} = \frac{N_d}{3\ sen(\alpha)} \tag{34}$$

No caso de blocos com distribuição das estacas em plano, surge a necessidade de decompor novamente a tensão encontrada de tração na direção formada ligando o eixo entre as estacas. A Figura 13, demonstra a decomposição destas forças.

Figura 13 – Decomposição da força Rs em Rs'.



Fonte: Bastos (2023).

$$R'_{sd} = Rs\frac{\sqrt{3}}{3} \tag{35}$$

Passo B) Verificação das tensões na biela.

$$A_{b,pil} = \frac{A_p}{3}\ sen(\alpha) \tag{36}$$

$$A_{b,est} = A_e\ sen(\alpha) \tag{37}$$

$$\sigma_{cd,b,pil} = \frac{N_d}{A_p\ sen^2(\alpha)} \tag{38}$$

$$\sigma_{cd,b,est} = \frac{N_d}{3\ A_e\ sen^2(\alpha)} \tag{39}$$

$$\sigma_{cd,b,lim} = 1{,}75\ K_r\ f_{cd}\ (0{,}90 \leq K_r \leq 0{,}95) \tag{40}$$

Passo C) Determinação das armaduras.

i. Armadura Principal.

$$A_s = \frac{R'_{sd}}{f_{yd}} \tag{41}$$

ii. Armadura de Malha Inferior e Superior.

$$A_{s,malha} = 0{,}20\ A_s \tag{42}$$

iii. Armadura de Suspensão.

$$A_{s,susp,tot} = \frac{N_d}{4{,}5\ f_{yd}} \tag{43}$$

$$A_{s,susp/face} = \frac{A_{s,susp,tot}}{3} \tag{44}$$

iv. Armadura de Costela.

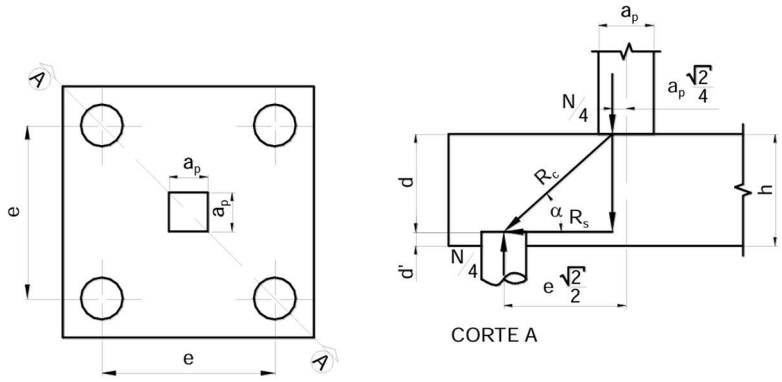
$$A_{s,costela} = \frac{1}{8}\ .\ A_{s,tot} \tag{45}$$

$$A_{s,tot} = A_s\ .\ n_{est} \tag{46}$$

6.4 Blocos de 4 estacas

O esquema estático considerado para o dimensionamento do bloco quadrado sobre quatro estacas é apresentado na Figura 14.

Figura 14 – Modelo de bielas para o bloco de 4 estacas.



Fonte: Adaptado de Bastos (2023).

Passo A) Determinação da inclinação das bielas e dos esforços internos.

$$\frac{\frac{N_d}{4}}{R_{sd}} = \frac{d}{\frac{l\sqrt{2}}{2} - \frac{a_{p,eqv}\sqrt{2}}{4}} \tag{47}$$

$$\text{tg}(\alpha) = \frac{\frac{N_d}{4}}{R_{sd}} \tag{48}$$

Desta forma:

$$\text{tg}(\alpha) = \frac{d}{\frac{l\sqrt{2}}{2} - \frac{a_{p,eqv}\sqrt{2}}{4}} \tag{49}$$

$$\alpha = \text{arctg}\left(\frac{d}{\frac{l\sqrt{2}}{2} - \frac{a_{p,eqv}\sqrt{2}}{4}}\right) \tag{50}$$

$$R_{sd} = \frac{N_d}{4\ tg(\alpha)} \tag{51}$$

$$R_{cd} = \frac{N_d}{4\ sen(\alpha)} \tag{52}$$

$$R'_{sd} = R_{sd}\frac{\sqrt{2}}{2} \tag{53}$$

Passo B) Verificação das tensões na biela.

$$A_{b,pil} = \frac{A_p}{4}\ sen(\alpha) \tag{54}$$

$$A_{b,est} = A_e\ sen(\alpha) \tag{55}$$

$$\sigma_{cd,b,pil} = \frac{N_d}{A_p\ sen^2(\alpha)} \tag{56}$$

$$\sigma_{cd,b,est} = \frac{N_d}{4\ A_e\ sen^2(\alpha)} \tag{57}$$

$$\sigma_{cd,b,lim} = 2{,}10\ K_r\ f_{cd}\ (0{,}90 \leq K_r \leq 0{,}95) \tag{58}$$

$$\sigma_{cd,b,pil} \leq \sigma_{cd,b,lim} \tag{59}$$

$$\sigma_{cd,b,est} \leq \sigma_{cd,b,lim} \tag{60}$$

Passo C) Determinação das armaduras.

i. Armadura Principal.

$$A_s = \frac{R'_{sd}}{f_{yd}} \tag{61}$$

ii. Armadura de Malha Inferior e Superior.

$$A_{s,malha} = 0{,}25\ A_s \tag{62}$$

iii. Armadura de Suspensão.

$$A_{s,susp,tot} = \frac{N_d}{6\ n_{est}\ f_{yd}}; \tag{63}$$

$$A_{s,susp/face} = \frac{A_{s,susp,tot}}{4} \tag{64}$$

iv. Armadura de Costela.

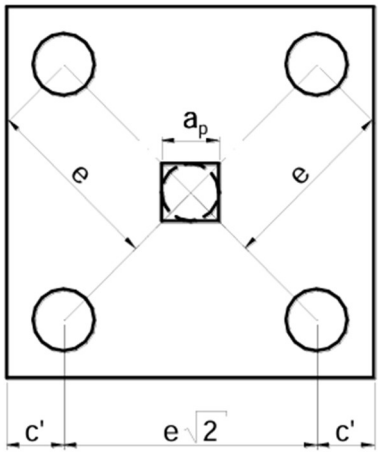
$$A_{s,costela} = \frac{1}{8}\ .\ A_{s,tot}\ ;\ A_{s,tot} = A_s\ .\ n_{est} \tag{65}$$

6.5 Blocos de 5 estacas

6.5.1 Caso com uma estaca central

A Figura 15 apresenta a geometria para o bloco de cinco estacas em configuração quadrada com estaca central

Figura 15 - Representação do bloco de 5 estacas com uma estaca central.



Fonte: Bastos (2023).

Passo A) Determinação da inclinação das bielas e dos esforços internos.

$$\mathrm{tg}(\alpha) = \frac{\frac{N_d}{5}}{R_{sd}} \tag{66}$$

Neste caso, temos que o triangulo formado pelas forças possui as mesmas inclinações que o bloco de 4 estacas;

$$\frac{\frac{N_d}{5}}{R_{sd}} = \frac{d}{\frac{l\sqrt{2}}{2} - \frac{a_{p,eqv}\sqrt{2}}{4}} \tag{67}$$

Desta forma:

$$\mathrm{tg}(\alpha) = \frac{d}{\frac{l\sqrt{2}}{2} - \frac{a_{p,eqv}\sqrt{2}}{4}} \tag{68}$$

$$\alpha = \mathrm{arctg}\left(\frac{d}{\frac{l\sqrt{2}}{2} - \frac{a_{p,eqv}\sqrt{2}}{4}}\right) \tag{69}$$

$$R_{sd} = \frac{N_d}{5\ tg(\alpha)} \tag{70}$$

$$R_{cd} = \frac{N_d}{5\ sen(\alpha)} \tag{71}$$

Passo B) Verificação das tensões nas bielas.

$$A_{b,pil} = \frac{A_p}{5}\ sen(\alpha) \tag{72}$$

$$A_{b,est} = A_e\ sen(\alpha) \tag{73}$$

$$\sigma_{cd,b,pil} = \frac{N_d}{A_p\ sen^2(\alpha)} \tag{74}$$

$$\sigma_{cd,b,est} = \frac{N_d}{5\ A_e\ sen^2(\alpha)} \tag{75}$$

$$\sigma_{cd,b,lim,pil} = 2{,}60\ K_r\ f_{cd} \tag{76}$$

$$\sigma_{cd,b,lim,est} = 2{,}10\ K_r\ f_{cd}\ ;\ (0{,}90 \leq K_r \leq 0{,}95) \tag{77}$$

$$\sigma_{cd,b,pil} \leq \sigma_{cd,b,lim,pil} \tag{78}$$

$$\sigma_{cd,b,est} \leq \sigma_{cd,b,lim,est} \tag{79}$$

Passo C) Determinação das armaduras.

i. Armadura Principal.

$$A_s = \frac{R'_{sd}}{f_{yd}} \tag{80}$$

ii. Armadura em Malha Inferior e Superior.

$$A_{s,malha} = 0{,}25\ A_s \tag{81}$$

iii. Armadura de Suspensão.

$$A_{s,susp,tot} = \frac{N_d}{7{,}5\ f_{yd}} \tag{82}$$

$$A_{s,susp/face} = \frac{A_{s,susp,tot}}{4} \tag{83}$$

iv. Armadura de Costela.

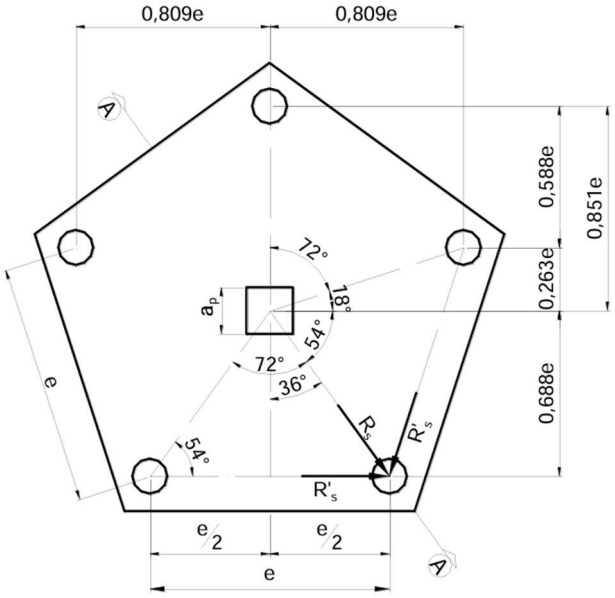
$$A_{s,costela} = \frac{1}{8}\ .A_{s,tot} \tag{84}$$

$$A_{s,tot} = A_s\ .n_{est} \tag{85}$$

6.5.2 Caso do bloco em forma pentagonal

A Figura 16 apresenta a geometria para o bloco de cinco estacas em configuração pentagonal.

Figura 16 - Representação do bloco de 5 estacas em formato pentagonal.



Fonte: Bastos (2023).

Passo A) Determinação da inclinação das bielas e dos esforços internos.

$$\text{tg}(\alpha) = \frac{\frac{N_d}{5}}{R_{sd}} \tag{86}$$

$$\frac{\frac{N_d}{5}}{R_{sd}} = \frac{d}{0{,}85\ l - 0{,}25\ a_{p,eqv}} \tag{87}$$

Desta forma;

$$\text{tg}(\alpha) = \frac{d}{0{,}85\ l - 0{,}25\ a_{p,eqv}} \tag{88}$$

$$\alpha = \text{arctg}\left(\frac{d}{0{,}85\ l - 0{,}25\ a_{p,eqv}}\right) \tag{89}$$

$$R_{sd} = \frac{N_d}{5\ tg(\alpha)} \tag{90}$$

$$R_{cd} = \frac{N_d}{5\ sen(\alpha)} \tag{91}$$

$$R'_{sd} = \frac{Rs}{2\cos(54°)} \tag{92}$$

Passo B) Verificação das tensões na biela.

Neste caso, quando a inclinação se encontra dentro dos limites estabelecidos ($45° \leq \alpha \leq 55°$) a verificação nos nós da biela não é necessária.

Passo C) Determinação das armaduras.

i. Armadura Principal.

$$A_s = \frac{R'_{sd}}{f_{yd}} \tag{93}$$

ii. Armadura em Malha Inferior e Superior.

$$A_{s,malha} = 0{,}25\ A_s \tag{94}$$

iii. Armadura de Suspensão.

$$A_{s,susp,tot} = \frac{N_d}{7{,}5\ f_{yd}} \tag{95}$$

$$A_{s,susp/face} = \frac{A_{s,susp,tot}}{5} \tag{96}$$

iv. Armadura de Costela.

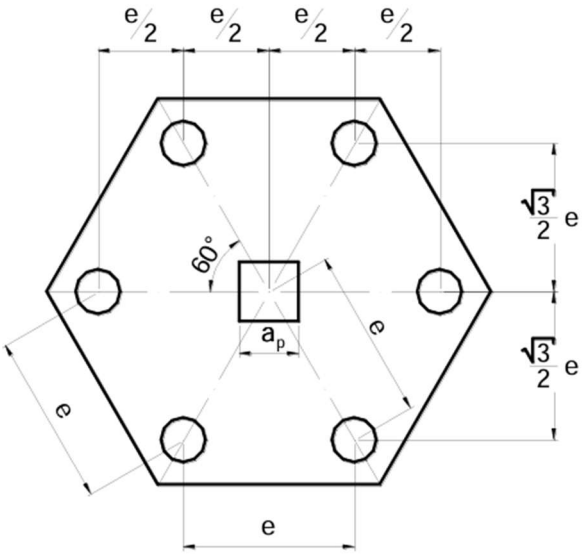
$$A_{s,costela} = \frac{1}{8}\ .A_{s,tot} \tag{97}$$

$$A_{s,tot} = A_s\ .n_{est} \tag{98}$$

6.6 Bloco de 6 estacas

A Figura 17 apresenta a geometria para o bloco de seis estacas em configuração hexagonal.

Figura 17 - Representação do bloco de 6 estacas.



Fonte: Bastos (2023).

Passo A) Determinação da inclinação das bielas e dos esforços internos.

$$\mathrm{tg}(\alpha) = \frac{\frac{N_d}{6}}{R_{sd}} \tag{99}$$

$$\frac{\frac{N_d}{6}}{R_{sd}} = \frac{d}{l - \frac{a_{p,eqv}}{4}} \tag{100}$$

Desta forma:

$$\mathrm{tg}(\alpha) = \frac{d}{l - \frac{a_{p,eqv}}{4}} \tag{101}$$

$$\alpha = \mathrm{arctg}\left(\frac{d}{l - \frac{a_{p,eqv}}{4}}\right) \tag{102}$$

$$R_{sd} = \frac{N_d}{6\ tg(\alpha)} \tag{103}$$

$$R_{cd} = \frac{N_d}{6\ sen(\alpha)} \tag{104}$$

$$R'_{sd} = R_s \tag{105}$$

Passo B) Verificação das tensões na biela.

Não é necessária quando $(45º \leq \alpha \leq 55º)$

Passo C) Determinação das armaduras.

i. Armadura Principal.

$$A_s = \frac{R'_{sd}}{f_{yd}} \tag{106}$$

ii. Armadura em Malha Inferior e Superior.

$$A_{s,malha,inf} = 0{,}25\ A_s \tag{107}$$

$$A_{s,malha,sup} = 0{,}2\ A_s \tag{108}$$

iii. Armadura de Suspensão.

$$A_{s,susp,tot} = \frac{N_d}{7{,}5\ f_{yd}} \tag{109}$$

$$A_{s,susp/face} = \frac{A_{s,susp,tot}}{5} \tag{110}$$

iv. Armadura de Costela.

$$A_{s,costela} = \frac{1}{8} \cdot A_{s,tot} \tag{111}$$

$$A_{s,tot} = A_s \cdot n_{est} \tag{112}$$

6.7 Particularidades dos métodos

Embora os modelos de cálculo adotados neste trabalho, o Modelo de Bielas e Tirantes e o Método da Flexão (CEB-FIP 1970), sejam consagrados pela prática e pelas normas técnicas, sua aplicação direta pressupõe certas idealizações geométricas. Uma das particularidades mais relevantes diz respeito à influência da geometria da seção transversal do pilar na distribuição real dos esforços, especialmente no caso de pilares alongados (pilares-parede).

O Modelo de Bielas e Tirantes clássico, conforme proposto por Blévot e Frémy (1967), foi desenvolvido experimentalmente com base em pilares de seção quadrada ou levemente retangular. Para a aplicação das equações de verificação das tensões nas zonas nodais, o método simplifica a geometria real do pilar, convertendo-o em um pilar quadrado equivalente de lado $a_{p,eqv} = \sqrt{A_p \cdot B_p}$.

No entanto, para pilares significativamente alongados, essa homogeneização da área de contato pode distorcer o comportamento físico das bielas. Em um pilar muito retangular, as bielas podem se formar a partir das extremidades do pilar ou distribuir-se de maneira não uniforme, invalidando a hipótese de um fluxo de carga centralizado e simétrico que o "pilar quadrado equivalente" sugere.

De forma análoga, o cálculo das reações nas estacas pelo Método da Superposição de Efeitos (Alonso, 1983) baseia-se na hipótese de rigidez absoluta do bloco. Essa premissa implica que o bloco rotaciona como um corpo rígido indeformável sobre as estacas.

Quando o pilar é excessivamente alongado em relação à geometria do bloco, a flexibilidade do próprio bloco na direção do menor lado do pilar pode se tornar relevante. A deformação por flexão do bloco pode fazer com que as estacas mais próximas ao pilar recebam uma parcela de carga maior do que a prevista pela distribuição linear plana do método de corpo rígido. Portanto, para geometrias de pilares que se aproximam de paredes estruturais, a hipótese de distribuição linear das reações pode não representar com precisão os picos de esforço nas estacas, sugerindo a necessidade de análises mais refinadas, como análises realizadas por métodos de elementos finitos, que fogem ao escopo das formulações implementadas neste trabalho.

Além disso, no dimensionamento de blocos com múltiplas estacas pelo Método da Flexão, o detalhamento das armaduras apresenta particularidades que exigem uma análise

crítica por parte do engenheiro, uma vez que a aplicação direta de critérios normativos pode levar a resultados incompatíveis com a natureza robusta destes elementos de fundação.

Para as armaduras complementares, especialmente a armadura de pele, observa-se uma lacuna na literatura quanto a critérios específicos para blocos de grandes dimensões. A ABNT NBR 6118 prescreve uma área mínima de pele equivalente a 0,10% da área de concreto da seção das vigas. No entanto, como os blocos de coroamento para múltiplas estacas possuem larguras significativas, a aplicação estrita dessa regra resulta em taxas de armadura excessivamente elevadas e muitas vezes impraticáveis construtivamente. Diante disso, muitos projetistas optam por não seguir a prescrição de vigas, adotando valores práticos ou construtivos que visam apenas o controle de fissurações como por exemplo ($\phi 10 c 20$).

Analogamente, a armadura de malha superior, essencial para o controle de fissuração por retração por temperatura, não é usualmente dimensionada por esforços de cálculo, mas sim definida por critérios de armadura mínima ou padrões de projeto.

Quanto a armadura principal, um aspecto divergente entre as metodologias reside na distribuição das barras. Enquanto o Modelo de Bielas e Tirantes e a recomendação da ABNT NBR 6118 preconizam a concentração de pelo menos 85% da armadura em faixas sobre os eixos das estacas afim de acompanhar o fluxo de tensões das bielas, o Método da Flexão parte de premissas distintas.

Ao idealizar o bloco como uma laje espessa ou viga rígida, o Método do CEB-70 assume que os momentos fletores atuam em toda a largura colaborante. Sob essa ótica, e considerando a complexidade de concentrar grandes quantidades de aço em faixas estreitas em blocos com muitas estacas, é prática corrente no uso deste método a distribuição uniforme da armadura principal ao longo de toda a largura útil do bloco. Embora essa abordagem difira da recomendação de concentração de barras da norma brasileira para modelos de bielas, ela é aceita no contexto do dimensionamento à flexão de blocos de grandes dimensões, facilitando a execução e garantindo a resistência ao momento fletor da seção.

7 METODOLOGIA

7.1 Descrição do método de desenvolvimento

A metodologia adotada neste trabalho consistiu na tradução dos conceitos teóricos e normativos, discutidos na Fundamentação Teórica e na seção de Dimensionamento, em uma ferramenta computacional funcional. O desenvolvimento do programa foi realizado utilizando a linguagem de programação Python selecionada por sua facilidade de escrita e ampla disponibilidade de bibliotecas voltadas para leitura de dados e interfaces gráficas.

Para a construção da interface gráfica, utilizou-se a biblioteca padrão “Tkinter”, complementada pela biblioteca “CustomTkinter” (SCHIMANSKY, 2023), que ofereceu componentes visuais modernos, garantindo uma experiência intuitiva. A importação de dados dos pilares foi implementada através da biblioteca Pandas (MCKINNEY, 2010), que permitiu que o arquivo de Excel fosse transformado em uma variável interpretável pelo código.

7.2 Descrição do método de validação

Para assegurar a confiabilidade técnica e a precisão dos algoritmos implementados, estabeleceu-se uma metodologia de validação baseada na análise comparativa. O procedimento consistiu em submeter o programa a cenários de teste e confrontar seus resultados com valores obtidos através de cálculos analíticos manuais, realizados conforme as prescrições da revisão bibliográfica.

A estratégia de validação foi estruturada em três casos de estudo distintos, selecionados para testar as principais funcionalidades e rotinas de decisão do programa:

a) Validação do Modelo de Bielas e Tirantes:

Foi modelado um bloco de geometria simples (3 estacas) para verificar a precisão das equações, o cálculo da força nos tirantes e a verificação das tensões de compressão nas bielas. O objetivo era confirmar a correta tradução das equações de Blévot e Frémy.

b) Validação da Rotina de Comparação de Esforços:

Foi simulado um cenário com dois pilares com esforços distintos para um bloco de 5 estacas. O objetivo era testar a capacidade do programa em processar simultaneamente diversos pilares e identificar automaticamente os esforços mais críticos para o dimensionamento.

c) Validação do Método da Flexão e Combinações de Carga:

Foi analisado um bloco de maior porte (6 estacas) tratado pelo Método CEB-FIP. Este caso visou a validação de duas frentes: a aplicação correta do método para o cálculo dos momentos fletores e a robustez do sistema na varredura de múltiplas combinações de carregamento incluindo os casos de vento, garantindo que o programa identificasse corretamente as armaduras nas duas direções.

Os critérios de aceitação para a validação basearam-se na convergência numérica entre os resultados do programa e os cálculos de referência, admitindo-se apenas desvios desprezíveis decorrentes de arredondamentos computacionais.

7.3 Premissas

Para viabilizar a automatização do dimensionamento, o funcionamento do programa fundamentou-se em um conjunto de regras decisórias baseadas na literatura técnica. Estas premissas orientaram tanto a seleção automática dos algoritmos de cálculo quanto os critérios de validação da geometria.

7.3.1 Critério para seleção do método de dimensionamento

Estabeleceu-se, como primeira etapa do processamento, a definição do método de cálculo a ser aplicado. O algoritmo foi programado para analisar a geometria do arranjo de estacas fornecido pelo usuário e adotar os seguintes critérios de seleção:

- **Modelo de Bielas e Tirantes:** Aplicado para arranjos padronizados de 2 a 6 estacas que configuram polígonos regulares (inscritos em uma circunferência). Para estas geometrias (B02 a B06), o programa utilizou as equações analíticas específicas de cada modelo para o cálculo das armaduras e a verificação das tensões nas bielas.
- **Método da Flexão (CEB-FIP 1970):** Adotado para todas as demais configurações, abrangendo blocos com mais de 6 estacas ou arranjos com posicionamento manual (coordenadas customizadas). Nestes casos, o programa aplicou a Teoria da Flexão baseada nas recomendações do CEB-FIP (1970).

7.3.2 Critério de ângulo mínimo das bielas

Visando assegurar a condição de rigidez do bloco, premissa fundamental para a validade dos modelos implementados, estabeleceu-se como critério de verificação a análise da inclinação da biela de compressão mais desfavorável (a mais abatida).

O algoritmo foi programado para calcular a inclinação (α) com base na disposição do arranjo de estacas e na altura útil definida pelo usuário, comparando o valor obtido com os limites inferiores recomendados pela literatura técnica:

- **Para o Modelo de Bielas e Tirantes:** Adotou-se $\alpha_{mín} = 45°$, conforme as recomendações de Blévot e Frémy (1967).
- **Para o Método de Flexão (CEB-FIP,1970):** Adotou-se $\alpha_{mín} = 40°$, seguindo as recomendações de Barros, Giongo e Oliveira (2014)

7.3.3 Critério de cálculo das reações nas estacas

Independentemente do método de dimensionamento a ser aplicado (Bielas ou Flexão), a determinação das solicitações individuais em cada estaca constitui a etapa preliminar de cálculo. O programa automatiza esse processo adotando o Método da Superposição de Efeitos, conforme a formulação clássica de Alonso (1983).

Para cada combinação de carregamento importada, o algoritmo processa a equação da flexão composta oblíqua, utilizando as coordenadas geométricas (x_i, y_i) do arranjo definido. Essa abordagem fundamenta-se na hipótese de comportamento de corpo rígido do bloco, o que permite calcular a reação vertical (N_i) em cada elemento e identificar automaticamente a estaca mais solicitada para as verificações de segurança.

7.4 Modulação do programa

Visando otimizar o desenvolvimento do código e sistematizar o programa, sua arquitetura foi estruturada de forma modular. Essa abordagem permitiu segmentar as complexas etapas de cálculo e lógica em unidades funcionais independentes, porém integradas, que atuam de maneira complementar. O sistema foi concebido em torno de três módulos fundamentais: o Módulo de Importação de Pilares, responsável pela importação e processamento dos esforços dos pilares presentes na planilha base; o Módulo de Modelagem, focado na definição dos parâmetros do bloco; e o Módulo de Dimensionamento e Verificações, que executa o processamento estrutural. A descrição e as atribuições específicas de cada componente são detalhadas a seguir.

7.5 Módulo de importação de pilares

O fluxo operacional do programa inicia-se com o módulo de importação, desenvolvido para processar uma planilha eletrônica em Excel. Este componente atua como a interface de entrada dos esforços solicitantes característicos, permitindo que o usuário importe, a planta de cargas proveniente de softwares de análise estrutural ou de memórias de cálculo externas.

A estrutura de dados da planilha foi concebida para contemplar todas as combinações de carregamento fundamentais descritas na Seção 5.6, abrangendo a Combinação Base, as Envoltórias de Máximos ($CasoFz_{máx}$; $CasoMx_{máx}$; $CasoMy_{máx}$) e os Casos de Vento isolados. Para assegurar a integridade dos dados e a correta interpretação pelo algoritmo, o programa opera mediante um arquivo modelo "pilares.xlsx" apresentado na Figura 18. O uso deste gabarito orienta a inserção das informações pelo usuário, garantindo a compatibilidade de leitura e mitigando erros de formatação que poderiam comprometer o processamento.

Figura 18 – Apresentação da planilha modelo "pilares.xlsx".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	PILARES	Todas as permanentes e acidentais dos pavimentos			Fz MAX (ELU)			Mx MAX (ELU)			My MAX (ELU)			Vento (0)°			Vento (90)°			Vento (180)°			Vento (270)°		
2		Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)	Fz (tf)	Mx (tf.m)	My (tf.m)
3	P1	440.2	-7.6	4.9	445.6	-7.3	11.3	401.04	-8.76	10.17	401.04	-6.57	13.56	4.2	-8.1	0.9	-4.2	8.1	-0.9	6.1	0	4.4	-6.1	0	-4.4
4	P2	481.4	15.9	0.1	488.2	16.3	7	439.38	19.56	6.3	439.38	14.67	8.4	-4.1	-7.9	-0.8	4.1	7.9	0.8	6	0.2	3.8	-6	-0.2	-3.8
5	P3	267.8	-8.4	-1.1	271.9	-8.1	-2.1	244.71	-9.72	-1.89	244.71	-7.29	-2.52	1.6	-7.5	0.3	-1.6	7.5	-0.3	-7.4	-0.5	1.5	7.4	0.5	-1.5
6	P4	706.9	-11.6	0.3	707.4	-16.6	0.6	636.66	-19.92	0.54	636.66	-14.94	0.72	2	-7.9	0.5	-2	7.9	-0.5	0.7	-0.3	2.5	-0.7	0.3	-2.5
7	P5	493.4	-0.1	0.2	526.7	0.2	-4.2	474.03	0.24	-3.78	474.03	0.18	-5.04	-1.7	-1.5	1.1	1.7	1.5	-1.1	-19.3	-0.1	8.8	19.3	0.1	-8.8
8	P6	1155.3	-0.5	11.8	1159.6	-0.5	27.3	1043.64	-0.6	24.57	1043.64	-0.45	32.76	1.2	-1.7	3.3	-1.2	1.7	-3.3	9	0	25.9	-9	0	-25.9
9	P7	796.8	-6.2	3.9	811.7	-6.9	19.9	730.53	-8.28	17.91	730.53	-6.21	23.88	-1.5	-1.2	1.6	1.5	1.2	-1.6	9.8	0	12.9	-9.8	0	-12.9
10	P8	437	0.6	1.6	453	0.9	-1.2	407.7	1.08	-1.08	407.7	0.81	-1.44	1.2	-1.5	0.5	-1.2	1.5	-0.5	-12.3	-0.1	9.9	12.3	0.1	-9.9
11	P9	943.1	2.8	13.6	944.2	1.9	14.5	849.78	2.28	13.05	849.78	1.71	17.4	2.9	-1.5	1.4	-2.9	1.5	-1.4	2.5	-0.1	27.2	-2.5	0.1	-27.2
12	P10	673.8	4.4	-1.7	682.3	4.5	7.5	614.07	5.4	6.75	614.07	4.05	9	2	-1.1	0.7	-2	1.1	-0.7	9.7	0	12.8	-9.7	0	-12.8
13	P11	319	-0.6	0.7	339.9	-0.3	-0.6	305.91	-0.36	-0.54	305.91	-0.27	-0.72	-0.3	-1.5	0	0.3	1.5	0	-11.5	-0.1	9.6	11.5	0.1	-9.6
14	P12	710.2	-0.4	11.9	710.6	-0.4	27.7	639.54	-0.48	24.93	639.54	-0.36	33.24	0.1	-1.5	0.1	-0.1	1.5	-0.1	2.1	-0.1	26.2	-2.1	0.1	-26.2
15	P13	535.6	-0.2	-11.7	540.9	-0.2	-4.5	486.81	-0.24	-4.05	486.81	-0.18	-5.4	0	-1.1	0.1	0	1.1	-0.1	9.3	0	11.9	-9.3	0	-11.9
16	P14	450	-1.4	-0.2	457.6	-1.2	-1.6	411.84	-1.44	-1.44	411.84	-1.08	-1.92	-1.1	-1.5	-0.4	1.1	1.5	0.4	-13.1	-0.1	9.6	13.1	0.1	-9.6
17	P15	923.1	-3.1	10.1	924	-2.1	10.8	831.6	-2.52	9.72	831.6	-1.89	12.96	-2.7	-1.5	-1.2	2.7	1.5	1.2	2.7	-0.1	26.3	-2.7	0.1	-26.3
18	P16	691	-4.2	-6.4	696.9	-4.3	0	627.21	-5.16	0	627.21	-3.87	0	-2.1	-1.1	-0.6	2.1	1.1	0.6	9.8	0	12.4	-9.8	0	-12.4
19																									

Fonte: Autoria Própria.

Posteriormente no programa, este módulo se integra à interface de modelagem, adicionando a funcionalidade de seleção de pilares na qual o usuário pode escolher entre o dimensionamento do bloco para um pilar ou selecionar um grupo de pilares. Nesta segunda modalidade, o módulo executa uma análise comparativa automática na qual o programa processa e compara os esforços de todos os pilares do grupo selecionado, identificando e adotando a condição mais desfavorável do conjunto para o dimensionamento do bloco, assegurando que o elemento atenda ao caso mais solicitante.

7.6 Módulo de modelagem geométrica

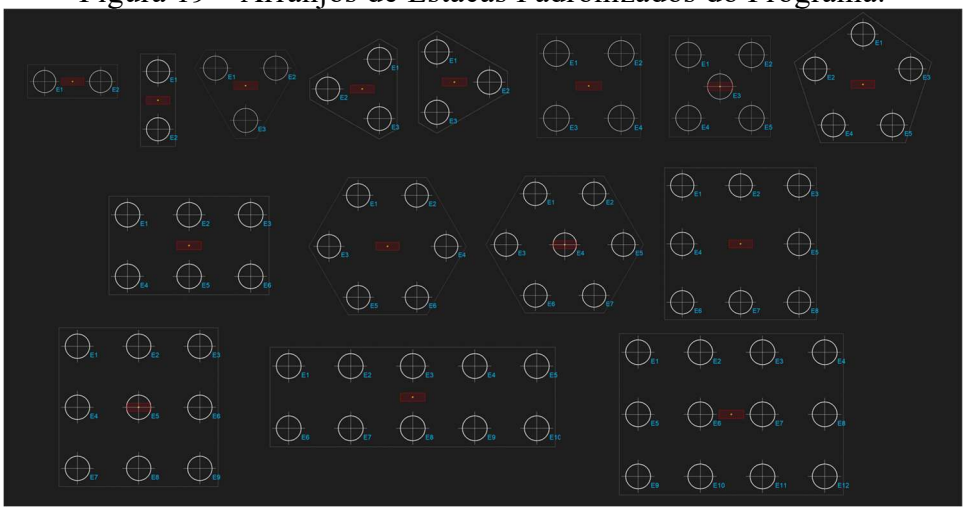
Na sequência do fluxo de trabalho, o programa apresenta o Módulo de Modelagem Geométrica o qual é composto por uma interface gráfica com a funcionalidade de centralizar a definição de todas as características físicas do sistema fundação-pilar e mostrar o elemento com suas dimensões.

Um dos pontos centrais deste módulo é a definição do arranjo das estacas, para o qual o programa oferece duas modalidades de entrada distintas, visando equilibrar agilidade e flexibilidade:

a) Arranjos Padrões:

O usuário tem acesso a uma biblioteca de configurações pré-definidas que abrangem as geometrias mais usuais na prática de engenharia, contemplando blocos de 2 a 12 estacas conforme ilustra a Figura 19.

Figura 19 – Arranjos de Estacas Padronizados do Programa.



Fonte: Autoria própria.

b) Entrada Customizada por Coordenadas:

Para atender a situações de projeto com geometrias atípicas não cobertas pelos padrões, foi implementada a funcionalidade de inserção manual da posição das estacas na qual, o usuário define as coordenadas (x, y) individuais de cada estaca. Neste caso, para qualquer geometria inserida via coordenadas, o programa adota automaticamente o Método do CEB-70.

Nota Importante: Ao optar pela entrada customizada, cabe ao usuário assegurar que o arranjo proposto respeite os eixos principais de inércia dos momentos aplicados.

Além da configuração das estacas, o módulo requer a entrada de um conjunto de parâmetros essenciais para o processamento:

1. Resistência característica à compressão do concreto (fck).
2. Espaçamento entre eixos (l).
3. Diâmetro das estacas (ϕ_{est})
4. Altura útil do bloco (d)
5. Dimensões da seção transversal do pilar (a_p, b_p)

Além disso, o programa solicita a distância da face da estaca à face externa do bloco. Embora este valor não intervenha diretamente no cálculo das armaduras principais ou secundárias, ele é fundamental para a definição das dimensões finais em planta do elemento, permitindo que a geometria completa seja salva e recuperada para futuras etapas de detalhamento.

7.7 Módulo de dimensionamento e verificações

Constituindo o núcleo de processamento do programa, este módulo é responsável por processar os parâmetros de entrada e executar os algoritmos de cálculo. O fluxo de operações segue uma rotina linear automatizada:

a) Processamento das Reações:

O algoritmo inicia calculando a reação vertical individual em cada estaca para todas as combinações de carregamento importadas, identificando a situação crítica de projeto conforme os métodos definidos nas premissas.

b) Direcionamento do Método:

Com base na geometria do arranjo de estacas, o programa direciona automaticamente o fluxo de cálculo para a rotina específica adequada: o Modelo de Bielas e Tirantes ou o Método da Flexão.

c) Execução do Dimensionamento:

Dentro da rotina selecionada, o programa realiza simultaneamente:

i. As verificações necessárias como a inclinação das bielas e suas tensões.
ii. O cálculo da armadura principal e das armaduras complementares (malha inferior, malha superior, armadura de pele e armadura de suspensão).

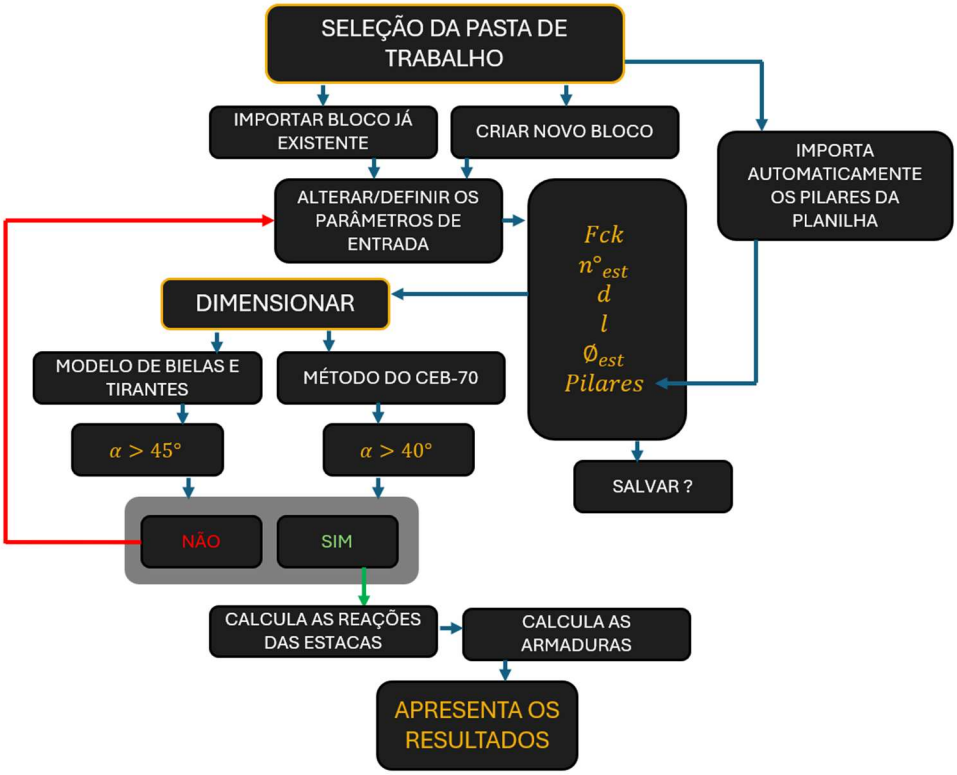
d) Síntese de Resultados:

Por fim, o módulo compila os dados processados e gera a interface de saída, exibindo as reações máximas das estacas para conferência, o resultado das armaduras calculadas e o valor da tensão na biela em comparação com a tensão limite.

8 APRESENTAÇÃO DO PROGRAMA DESENVOLVIDO

Este capítulo detalha a interface gráfica desenvolvida e o fluxo operacional do programa, evidenciando as funcionalidades implementadas para o dimensionamento automatizado dos blocos de coroamento. Para ilustrar a lógica de interação entre o usuário e o sistema, a Figura 20, apresenta o fluxograma completo do processo, mapeando as etapas desde a inicialização do aplicativo até a exibição final dos resultados.

Figura 20 - Mapa de etapas do aplicativo.



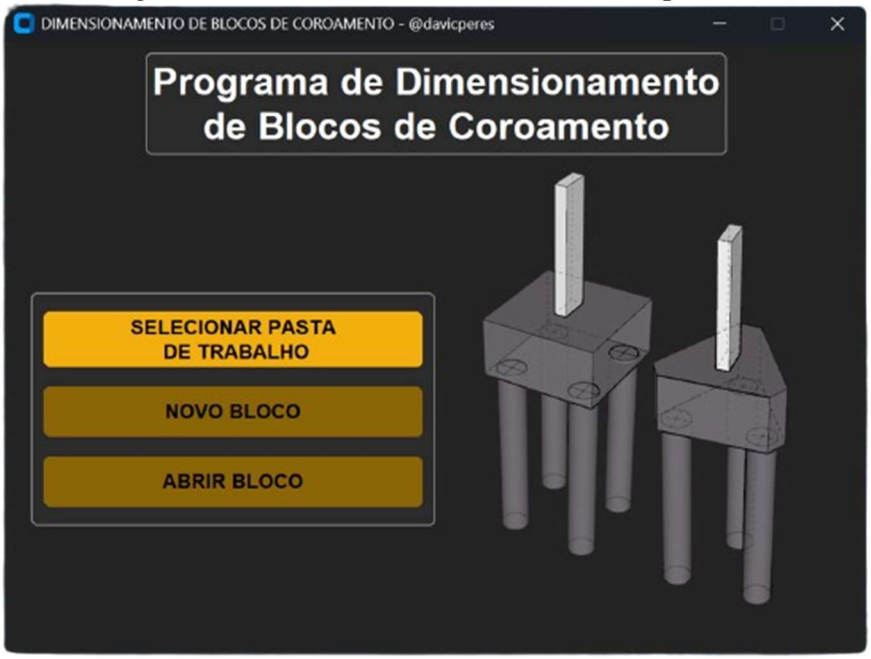
Fonte: Autoria Própria.

8.1 Tela inicial e seleção da pasta de trabalho

A interface do programa foi projetada com foco na simplicidade e intuitividade, adotando uma abordagem moderna e minimalista para facilitar o fluxo de trabalho do engenheiro. O programa é estruturado em duas janelas principais: a "Tela Inicial" para gerenciamento de projetos e a "Janela de Modelagem" para realização do dimensionamento.

Ao iniciar o programa, o usuário é apresentado à tela inicial. Esta interface possui três botões principais, dos quais, dois deles ("NOVO BLOCO" e "ABRIR BLOCO") permanecem desabilitados por padrão, até que uma pasta de trabalho seja definida conforme ilustra a Figura 21.

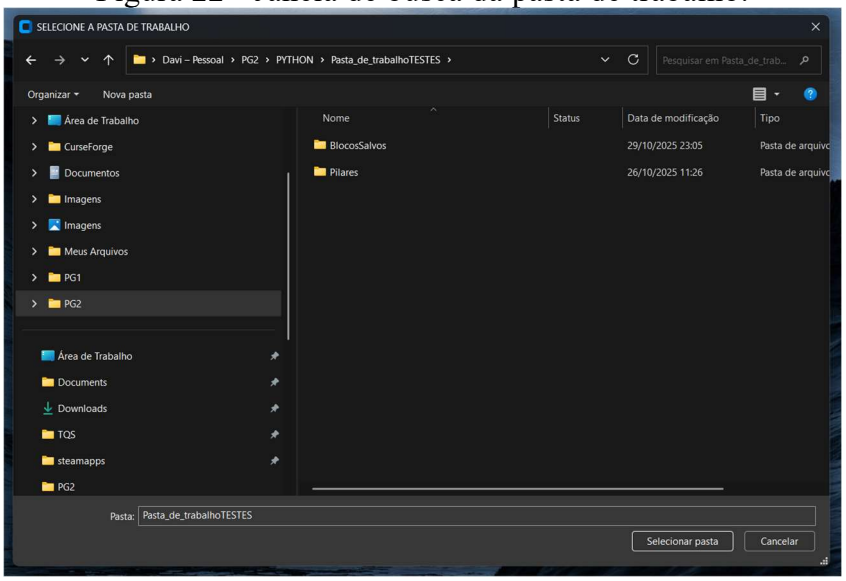
Figura 21 - Janela inicial com botões bloqueados.



Fonte: Autoria Própria.

A primeira ação obrigatória é a definição da pasta de trabalho onde os arquivos serão gerenciados. Ao clicar no botão "SELECIONAR PASTA DE TRABALHO", uma janela do explorador de arquivos é aberta, como demonstra a Figura 22, permitindo ao usuário indicar o caminho da pasta. É nesta pasta que o programa buscará a planilha “pilares.xlsx” e onde serão salvos os arquivos de projetos gerados.

Figura 22 - Janela de busca da pasta de trabalho.



Fonte: Autoria Própria.

Após a validação da pasta escolhida, os botões "NOVO BLOCO" e "ABRIR BLOCO" são habilitados conforme é apresentado na Figura 23.

Figura 23 - Janela inicial após a pasta de trabalho ser selecionada.

Fonte: Autoria Própria.

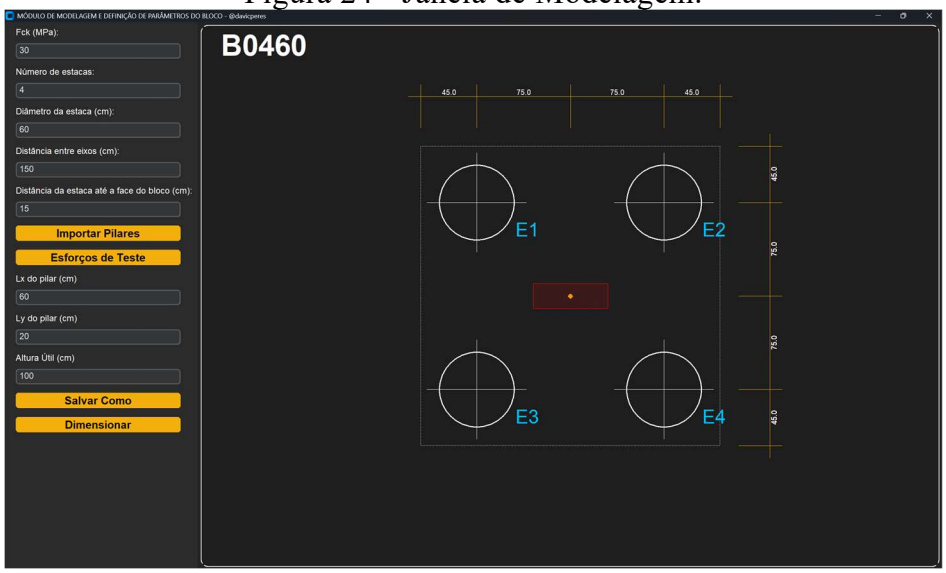
O botão "NOVO BLOCO" inicia um projeto do zero, enquanto o botão "ABRIR BLOCO" permite carregar um projeto existente com a extensão “.pkl” previamente salvo pelo usuário. Ambas as opções conduzem o usuário para a "Janela de Modelagem".

8.2 Janela de modelagem e definição de parâmetros

A "Janela de Modelagem" é o ambiente principal do programa. Sua interface, exibida na Figura 24, é dividida em duas áreas distintas: uma barra lateral à esquerda destinada à entrada de parâmetros e à direita, uma área que fornece a visualização em tempo real a geometria do bloco de coroamento e o arranjo de estacas cotado.

No painel lateral, o usuário define a resistência do concreto à compressão "fck" e os parâmetros geométricos como o diâmetro das estacas "$\emptyset_{est}$", a distância entre eixos "l", a distância da face da estaca à face do bloco "f", as dimensões do pilar "L_x" e "L_y" e por último, a altura útil "d".

Figura 24 - Janela de Modelagem.



Fonte: Autoria Própria.

O parâmetro que define a geometria do elemento é o número de estacas. O campo de entrada correspondente oferece flexibilidade, permitindo duas formas de inserção: o usuário pode optar pelos arranjos padronizados, inserindo valores de 2 a 12, ou, caso necessário, utilizar a entrada customizada para geometrias não previstas. Nesta segunda modalidade, deve-se inserir as coordenadas (x,y) de cada estaca, separadas por ponto e vírgula (ex: x1,y1; x2,y2; x3,y3...).

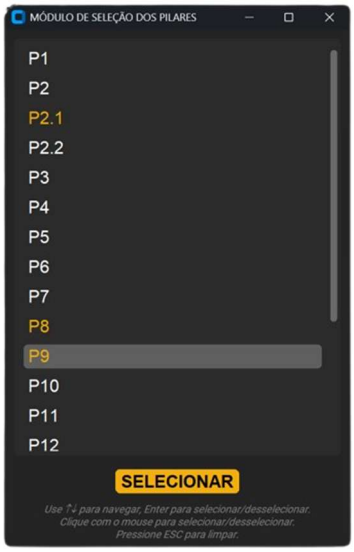
8.3 Importação de esforços e seleção de pilares

Definida a geometria, a etapa seguinte consiste na atribuição dos esforços solicitantes. O programa oferece duas modalidades para esta entrada de dados:

a) Importação via Planilha:

Através do botão "Importar Pilares", o programa acessa a planilha “pilares.xlsx” presente na pasta de trabalho e exibe uma lista de seleção, conforme detalhado na Figura 25. O usuário pode selecionar um único pilar ou um grupo de pilares simultaneamente. No caso de seleção múltipla, o algoritmo avaliará automaticamente a situação mais crítica entre todos os elementos selecionados para o dimensionamento.

Figura 25 - Janela de Seleção dos Pilares.



Fonte: Autoria Própria.

b) Entrada Manual:

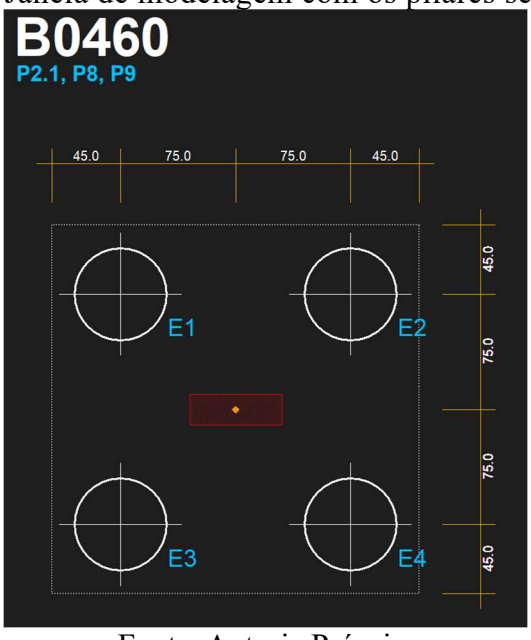
Para verificações rápidas ou casos de esforços não tabelados, o botão "Fz;Mx;My" abre uma janela simplificada, mostrada na Figura 26, para a inserção direta dos valores de Força Normal F_z e Momentos Fletores M_x e M_y, permitindo uma análise independente da planilha.

Figura 26 - Janela de Esforços de Teste

Fonte: Autoria Própria.

Após a confirmação, os identificadores dos pilares selecionados e o título gerado para o bloco (ex: B0460) são exibidos no topo da área gráfica, como pode ser observado na Figura 27, facilitando o controle do projeto pelo usuário. A seleção pode ser alterada a qualquer momento retornando-se ao menu de importação.

Figura 27 - Janela de modelagem com os pilares selecionados.



Fonte: Autoria Própria.

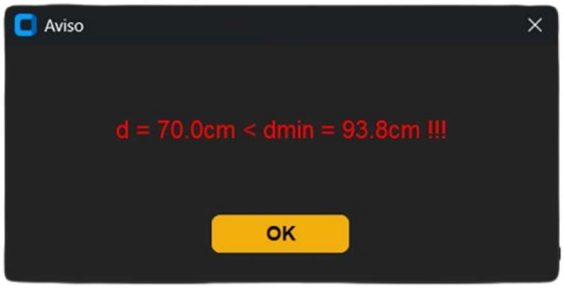
8.4 Dimensionamento e análise de resultados

Com todos os parâmetros de entrada definidos, o usuário dispõe das opções de "Salvar" o projeto “B0460.pkl” para edições futuras ou proceder com o cálculo através do botão "Dimensionar".

Ao acionar o comando de dimensionamento, o programa executa preliminarmente a verificação da condição de rigidez, conforme os critérios estabelecidos na metodologia.

Caso a altura útil informada resulte em uma inclinação das bielas inferior ao limite, o processamento é interrompido e um alerta é exibido, conforme exemplifica a Figura 28, sugerindo ao usuário a altura mínima necessária.

Figura 28 - Pop-up de aviso indicando "d<dmín".



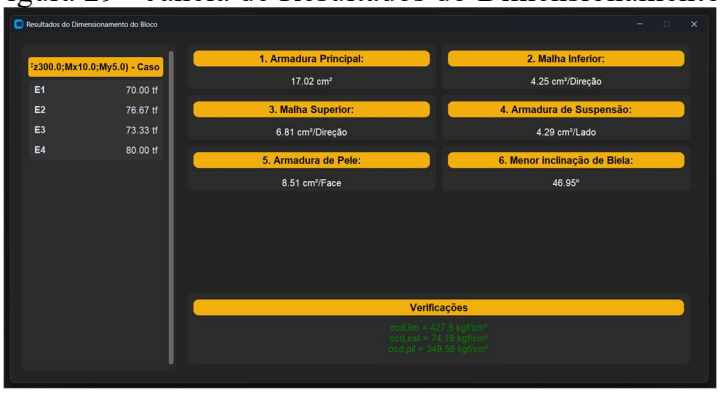
Fonte: Autoria Própria.

Satisfeita a condição de rigidez, o programa executa o roteiro de cálculo (Bielas e Tirantes ou Método da Flexão, a depender do número de estacas) e apresenta automaticamente a "Janela de Resultados", representada na Figura 29. Esta interface sintetiza as informações finais em dois painéis:

Reações nas Estacas: À esquerda, uma lista rolável exibe as reações das estacas para cada estaca em valores característicos, referente a combinação de carregamento mais desfavorável.

Detalhamento das Armaduras: À direita, são apresentadas as áreas de aço calculadas para as armaduras principais, malhas superior e inferior, e armaduras de suspensão e de pele além da inclinação da biela mais crítica e das tensões da biela comprimida em comparação com a tensão limite.

Figura 29 - Janela de Resultados do Dimensionamento.



Fonte: Autoria Própria.

8.5 Código do programa

Visando garantir a clareza lógica do programa e facilitar futuras atualizações ou expansões, a implementação computacional foi desenvolvida sob o paradigma da orientação a objetos. O código-fonte foi segmentado em seis arquivos distintos, cada um responsável por tarefas específicas dentro do fluxo de execução do programa.

A seguir, descrevem-se as atribuições de cada arquivo de código:

a) mainwindow.py (Gerenciamento de Sessão):

Presente no "ANEXO A", este arquivo contém a classe "MainWindow", responsável pela inicialização da aplicação. Ele gerencia a tela de boas-vindas, controla a criação de novos projetos ou a abertura de projeto existentes e executa funções, como a definição do diretório onde os arquivos de projeto e planilhas serão manipulados.

b) main.py (Interface Principal de Modelagem):

Apresentado no "ANEXO B", este é o módulo central da interface gráfica, onde reside a classe "AppModelagem". Ele atua como o organizador da experiência do usuário, integrando os campos de entrada de dados da barra lateral, o ambiente de visualização gráfica do bloco e os botões de comando.

c) blocos.py (Objeto e Geometria):

Este código, presente no "ANEXO C", define a classe "BLOCO", que serve como a estrutura de dados principal do sistema. Este módulo é responsável por receber e armazenar todos os parâmetros de entrada (geométricos e físicos), processar a lógica de geração de coordenadas das estacas (arranjos padrão ou customizados) e conter os métodos de desenho que geram a representação visual do bloco e suas estacas na tela.

d) modulodedimensionamento.py (Motor de Cálculo):

Apresentado no "ANEXO D", este algoritmo concentra toda a inteligência de engenharia do programa. Ele implementa a classe base e as classes derivadas para cada tipologia de bloco (ex: B03, B04, B06). É aqui que residem os algoritmos do Modelo de Bielas e Tirantes e do Método da Flexão (CEB-FIP), bem como as rotinas de verificação e de cálculo das reações nas estacas.

e) moduloselecaopilares.py (Integração de Dados):

Presente no “ANEXO E”, este módulo é responsável pela comunicação com a planilha de esforços. Ele implementa a classe “SelecaoPilares”, que utiliza a biblioteca “Pandas” para ler, interpretar e estruturar os dados da planilha “pilares.xlsx”. Além disso, gerencia a interface de seleção múltipla, permitindo ao usuário filtrar e escolher quais pilares serão dimensionados.

f) helpfunctions.py (Biblioteca Auxiliar):

Disponível no “ANEXO F”, este código reúne funções utilitárias de uso global, como algoritmos de ordenação alfanumérica de textos (ex: ordenar nomes de pilares como P1, P2, P10) e cálculos geométricos auxiliares. Sua função é reduzir a repetição de código e fornecer ferramentas comuns aos demais arquivos de código.

9 VALIDAÇÃO E ANÁLISE DOS RESULTADOS

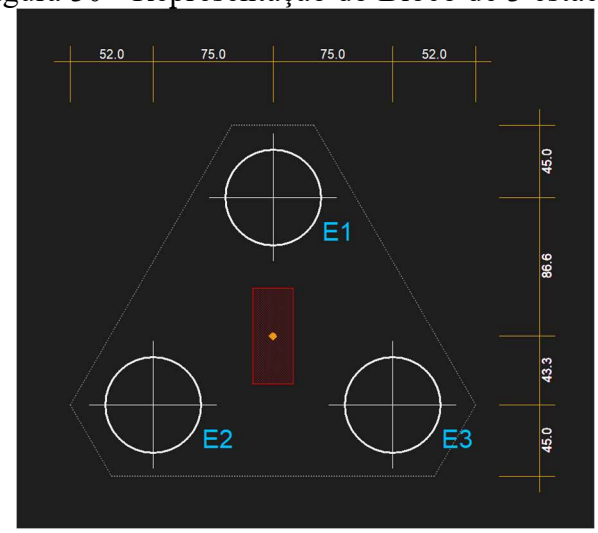
Com a finalidade de atestar a confiabilidade e a precisão da ferramenta computacional desenvolvida, este capítulo apresenta os procedimentos de validação realizados. O método consiste na aplicação do programa para o dimensionamento de três casos distintos, abrangendo diferentes geometrias e métodos de cálculo.

Para cada caso, os resultados obtidos pelo programa como as reações nas estacas e as áreas de armadura foram confrontados com resultados calculados de forma analítica, seguindo rigorosamente as prescrições teóricas detalhadas na Revisão Bibliográfica.

9.1 Validação do programa para o modelo de bielas

Este primeiro exemplo tem como objetivo validar a implementação do **Modelo de Bielas e Tirantes**. Para tal, foi selecionado como objeto de estudo um bloco de coroamento sobre três estacas, cuja configuração estrutural encontra-se esquematizada na Figura 30.

Figura 30 - Representação do Bloco de 3 estacas.



Fonte: Autoria Própria.

Dados do Problema:

Diâmetro da estaca: $\emptyset_{est} = 60\ cm$

Distância entre eixos: $l = 150\ cm$

Seção do Pilar P3: 25 x 60 cm

Materiais: Concreto C30 ($fck = 30MPa$), Aço CA-50, $\gamma_f = 1.4$, $f_{yd} = 1.15$

Esforços Solicitantes: $F_z = 120{,}0\ tf; M_x = 5{,}0\ tf.m; M_y = 2{,}0\ tf.m$

Passo 1 – Definição da Altura Útil.

Inicialmente, determina-se a altura útil mínima $d_{mín}$ necessária para garantir o comportamento rígido do elemento. Adotando o critério de inclinação mínima da biela de $\alpha = 45°$, e considerando a geometria de um triângulo equilátero, temos:

$$d_{mín} = \frac{l\sqrt{3}}{3} - 0{,}3\ a_{p,eqv}$$

$$a_{p,eqv} = \sqrt{0{,}25 \cdot 0{,}4} = 0{,}387\ m$$

$$d_{mín} = \frac{1{,}5 \cdot \sqrt{3}}{3} - 0{,}3 \cdot 0{,}387 = 0{,}750\ m$$

Adota-se $d = 80\ cm$.

$$\alpha = \text{arctg}\left(\frac{d}{\frac{l\sqrt{3}}{3} - 0{,}3\ a_{p,eqv}}\right) = 46{,}85°$$

Passo 2 – Determinação das Reações nas Estacas.

As reações verticais em cada estaca foram calculadas utilizando o Método da Superposição, considerando a carga axial e os momentos fletores. As coordenadas das estacas em relação ao centro geométrico do bloco são:

$$E_1\colon x_1 = 0{,}000\ m; y_1 = 0{,}866\ m$$
$$E_2\colon x_2 = -0{,}750\ m; y_2 = -0{,}433\ m$$
$$E_3\colon x_3 = 0{,}750\ m; y_3 = -0{,}433\ m$$

Temos que:

$$N_i = \frac{N}{n_{est}} \pm M_x\ \frac{x_i}{\sum {x_i}^2} \pm M_y\ \frac{y_i}{\sum {y_i}^2}$$

$$\sum {x_i}^2 = 1{,}125 \text{ e } \sum {y_i}^2 \approx 1{,}125$$

Com isso, obtém-se as reações em cada estaca:

$$N_1 = \frac{120{,}0\ tf}{3} + 5{,}0\ tf.m \cdot \frac{0{,}000\ m}{1{,}125\ m^2} - 2{,}0\ tf.m \cdot \frac{0{,}866\ m}{1{,}125\ m^2} = 38{,}46\ tf$$

$$N_2 = \frac{120{,}0\ tf}{3} + 5{,}0\ tf.m \cdot \frac{(-0{,}750\ m)}{1{,}125\ m^2} - 2{,}0\ tf.m \cdot \frac{(-0{,}433\ m)}{1{,}125\ m^2} = 37{,}44\ tf$$

$$N_3 = \frac{120{,}0\ tf}{3} + 5{,}0\ tf.m \cdot \frac{0{,}750\ m}{1{,}125\ m^2} - 2{,}0\ tf.m \cdot \frac{(-0{,}433\ m)}{1{,}125\ m^2} = 44{,}10\ tf$$

Observa-se que a estaca mais solicitada é a E3, com $N_3 = 44{,}10\ tf$. Para o dimensionamento, define-se a força de cálculo total majorada N_d considerando que as três estacas estivessem submetidas a essa carga crítica:

$$N_d = 3 \cdot 44{,}10 \cdot 1{,}4 = 185{,}22\ tf$$

Passo 3 – Verificação das Tensões nas Bielas.

A segurança contra o esmagamento do concreto nas bielas comprimidas é verificada comparando-se as tensões atuantes na base do pilar e no topo da estaca com a tensão limite de cálculo $\sigma_{cd,b,lim}$. Adotando o coeficiente de Rüsch ($k_r = 0{,}90$) de forma a obter um valor mais a favor da segurança.

$$\sigma_{cd,b,lim} = 1{,}75 \cdot\ K_r \cdot f_{cd}$$

$$\sigma_{cd,b,lim} = 1{,}75 \cdot 0{,}90 \cdot \frac{30\ MPa}{1{,}4} = 33{,}75\ MPa = 337{,}50\ kgf/cm^2$$

As tensões atuantes nas zonas nodais junto ao pilar $\sigma_{cd,b,pil}$ e junto à estaca $\sigma_{cd,b,est}$ são calculadas por:

$$\sigma_{cd,b,pil} = \frac{N_d}{A_p\ sen^2(\alpha)}$$

$$\sigma_{cd,b,pil} = \frac{185{,}22\ tf}{(0{,}25\ m \cdot 0{,}60\ m) \cdot sen^2(46{,}85°)} = 2319{,}89\frac{tf}{m^2}$$

$$\sigma_{cd,b,pil} = 231{,}99\ kg/cm^2$$

$$\sigma_{cd,b,est} = \frac{N_d}{3\ A_e\ sen^2(\alpha)}$$

$$\sigma_{cd,b,est} = \frac{185{,}22\ tf}{3 \cdot \left(\pi\ \cdot \frac{0{,}6^2\ m^2}{4}\right) \cdot sen^2(46{,}85°)} = 410{,}25\frac{tf}{m^2}$$

$$\sigma_{cd,b,est} = 41{,}02\ kgf/cm^2$$

Como ambas as tensões atuantes são menores que $\sigma_{cd,b,lim}$, o bloco atende ao critério de segurança da biela.

Passo 4 – Dimensionamento das Armaduras.

Finalmente, calculam-se as áreas de aço necessárias.

$$A_s = \frac{R'_{sd}}{f_{yd}};\ R'_{sd} = Rs\frac{\sqrt{3}}{3};\ R_{sd} = \frac{N_d}{3\ tg(\alpha)}$$

$$R_{sd} = \frac{185{,}22\ tf}{3 \cdot tg(46{,}85°)} = 57{,}88\ tf$$

$$R'_{sd} = 57{,}88\ tf \cdot \frac{\sqrt{3}}{3} = 33{,}42\ tf$$

$$A_s = \frac{33{,}42\ tf}{\frac{5\ tf/cm^2}{1{,}15}} = 7{,}69\ cm^2$$

A armadura de malha inferior e superior são dadas por 20% da armadura principal.

$$A_{s,malha} = 0{,}2 \cdot 7{,}69 = 1{,}54\ cm^2$$

A armadura de costela é dada por:

$$A_{s,costela} = \frac{1}{8}\ A_{s,tot}\ ; A_{s,tot} = A_s \cdot n_{est}$$

$$A_{s,costela} = \frac{1}{8}\ (7{,}69 \cdot 3) = 2{,}88\ cm^2/face$$

Por fim, a armadura mínima de suspensão é dada por:

$$A_{s,susp,tot} = \frac{N_d}{4{,}5\ f_{yd}};\ A_{s,susp/face} = \frac{A_{s,susp,tot}}{3}$$

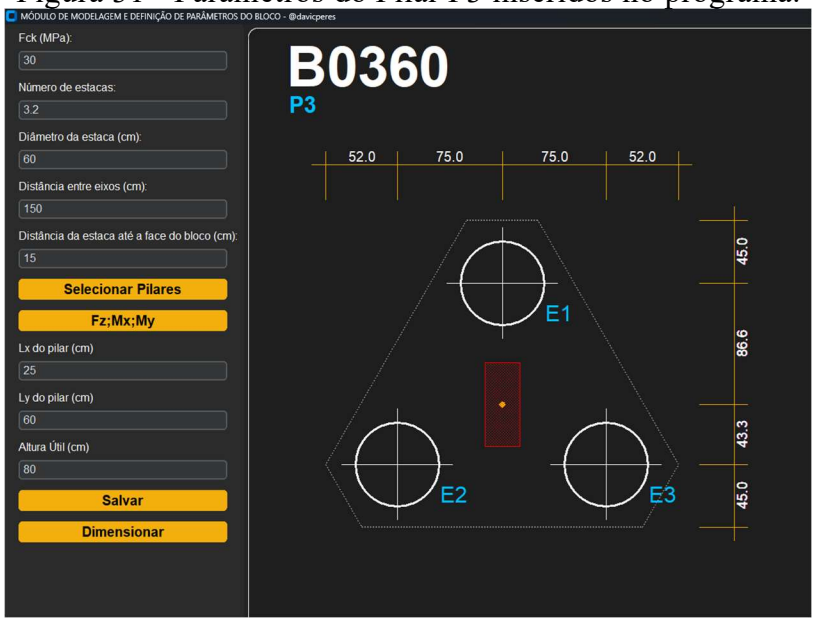
$$A_{s,susp,tot} = \frac{185{,}22\ tf}{4{,}5 \cdot \frac{5\ tf/cm^2}{1{,}15}} = 9{,}47\ cm^2$$

$$A_{s,susp/face} = \frac{9{,}47\ cm^2}{3} = 3{,}16\ cm^2/face$$

Passo 5 – Comparação com os Resultados do Programa.

A Figura 31, demonstra os dados do bloco de coroamento inseridos no programa.

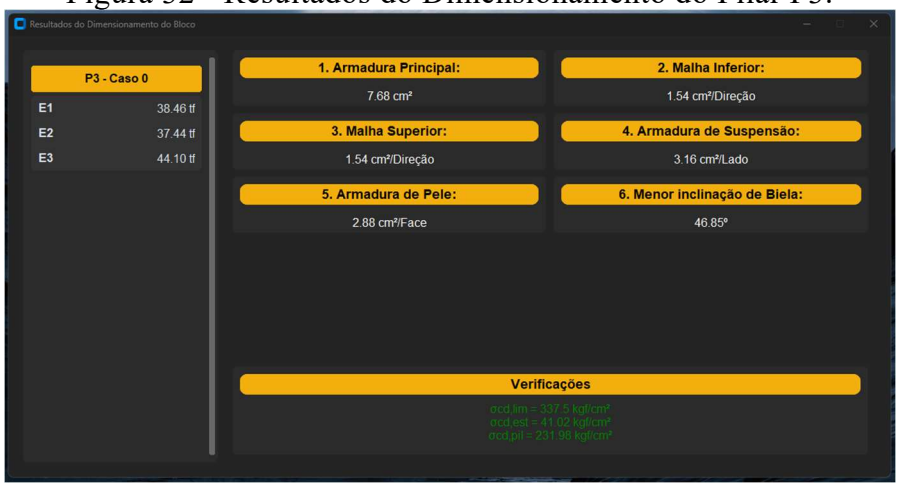
Figura 31 - Parâmetros do Pilar P3 inseridos no programa.



Fonte: Autoria Própria.

A Figura 32, apresenta os resultados gerados pelo programa.

Figura 32 - Resultados do Dimensionamento do Pilar P3.



Fonte: Autoria Própria.

Com base nos resultados gerados pelo programa, apresentados na Figura 32, elaborou-se a Tabela 1. Este quadro comparativo permite confrontar os valores obtidos computacionalmente com aqueles calculados pelo método analítico, evidenciando a precisão da ferramenta.

Tabela 1 - Comparação de resultados do bloco do Pilar P3

Parâmetro de Análise	Cálculo Analítico		Resultado Software		Diferença (%)
Reação nas Estacas					
E1	38,46	tf	38,46	tf	0,0%
E2	37,44	tf	37,44	tf	0,0%
E3	44,10	tf	44,10	tf	0,0%
Tensões					
σcd,lim	337,50	kgf/cm²	337,50	kgf/cm²	0,0%
σcd,b,est	41,02	kgf/cm²	41,02	kgf/cm²	0,0%
σcd,b,pil	231,99	kgf/cm²	231,98	kgf/cm²	0,0%
Armaduras					
As,principal	7,69	cm²	7,68	cm²	-0,1%
As,malha,inf	1,54	cm²/dir	1,54	cm²/dir	0,0%
As,malha,sup	1,54	cm²/dir	1,54	cm²/dir	0,0%
As,susp	3,16	cm²/lado	3,16	cm²/lado	0,0%
As,pele	2,88	cm²/face	2,88	cm²/face	0,0%

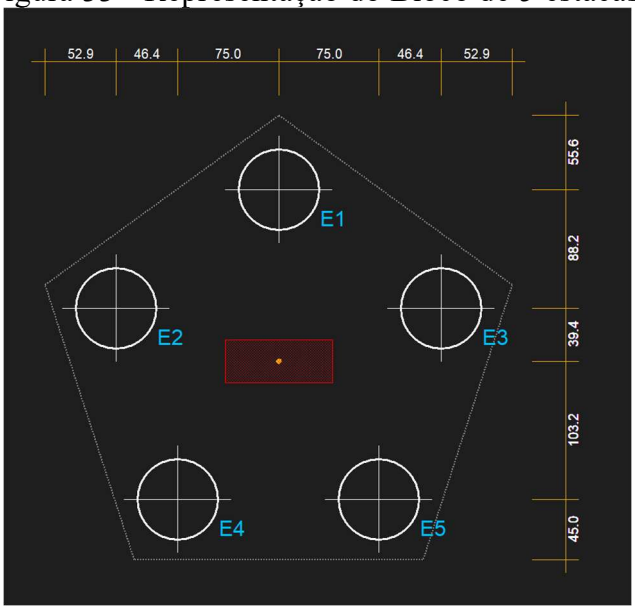
Fonte: Autoria Própria.

Conforme observado na Tabela 1, os resultados obtidos através programa apresentaram plena convergência com os valores calculados analiticamente, validando a precisão do algoritmo implementado para o Modelo de Bielas e Tirantes.

9.2 Validação da comparação de esforços

O segundo caso de estudo visa demonstrar e validar a funcionalidade de **Análise Comparativa de Pilares**, verificando a capacidade do algoritmo em processar múltiplos elementos simultaneamente para identificar a situação crítica de projeto. Para a validação desta rotina, selecionou-se um bloco pentagonal sobre cinco estacas, cuja geometria e arranjo estão ilustrados na Figura 33.

Figura 33 - Representação do Bloco de 5 estacas.



Fonte: Autoria Própria.

Dados do Problema:

Diâmetro da estaca: $\emptyset_{est} = 60\ cm$

Distância entre eixos: $l = 150\ cm$

Seção do Pilar P5 e P6 : 80 x 32 cm

Materiais: Concreto C30 ($fck = 30MPa$), Aço CA-50, $\gamma_f = 1.4$, $f_{yd} = 1.15$

Esforços Solicitantes:

$$Pilar\ P5\text{: } F_z = 188{,}0\ tf; M_x = 15{,}0\ tf.m; M_y = -10{,}0\ tf.m$$

$$Pilar\ P6\text{: } F_z = 200{,}0\ tf; M_x = 0{,}0\ tf.m; M_y = 0{,}0\ tf.m$$

Passo 1 – Definição da Altura Útil.

Para determinar a altura útil mínima neste caso, temos que:

$$d_{mín} = 0{,}85\ l - 0{,}25\ a_{p,eqv}$$

$$a_{p,eqv} = \sqrt{0{,}80 \cdot 0{,}32} = 0{,}506\ m$$

$$d_{mín} = 0{,}85 \cdot 1{,}5\ m - 0{,}25 \cdot 0{,}506\ m = 1{,}149\ m$$

Adota-se, $d = 120\ cm$.

Neste caso, o valor de α é dado por:

$$\alpha = \text{arctg}\left(\frac{d}{0{,}85\ l - 0{,}25\ a_{p,eqv}}\right)$$

$$\alpha = \text{arctg}\left(\frac{1{,}20\ m}{0{,}85 \cdot 1{,}50\ m - 0{,}25 \cdot 0{,}506\ m}\right) = 46{,}26°$$

Passo 2 – Determinação das Reações nas Estacas.

As coordenadas das estacas em relação ao centro geométrico do bloco são:

$$E_1\colon x_1 = 0{,}000\ m; y_1 = 1{,}276\ m$$
$$E_2\colon x_2 = -1{,}214\ m; y_2 = 0{,}394\ m$$
$$E_3\colon x_3 = 1{,}214\ m; y_3 = 0{,}394\ m$$
$$E_4\colon x_4 = -0{,}750\ m; y_4 = -1{,}032\ m$$
$$E_5\colon x_5 = 0{,}750\ m; y_5 = -1{,}032\ m$$
$$\sum {x_i}^2 = 4{,}070 \text{ e } \sum {y_i}^2 = 4{,}070$$

Para o pilar P5 temos:

$$N_1 = \frac{188{,}0\ tf}{5} + 15{,}0\ tf.m \cdot \frac{0{,}00\ m}{4{,}070\ m^2} - (-10{,}0\ tf.m) \cdot \frac{1{,}276\ m}{4{,}070\ m^2} = 40{,}73\ tf$$

$$N_2 = \frac{188{,}0\ tf}{5} + 15{,}0\ tf.m \cdot \frac{(-1{,}214\ m)}{4{,}070\ m^2} - (-10{,}0\ tf.m) \cdot \frac{0{,}394\ m}{4{,}070\ m^2} = 34{,}09\ tf$$

$$N_3 = \frac{188{,}0\ tf}{5} + 15{,}0\ tf.m \cdot \frac{1{,}214\ m}{4{,}070\ m^2} - (-10{,}0\ tf.m) \cdot \frac{0{,}394\ m}{4{,}070\ m^2} = 43{,}04\ tf$$

$$N_4 = \frac{188{,}0\ tf}{5} + 15{,}0\ tf.m \cdot \frac{(-0{,}750\ m)}{4{,}070\ m^2} - (-10{,}0\ tf.m) \cdot \frac{(-1{,}032\ m)}{4{,}070\ m^2} = 32{,}30\ tf$$

$$N_5 = \frac{188{,}0\ tf}{5} + 15{,}0\ tf.m \cdot \frac{0{,}750\ m}{4{,}070\ m^2} - (-10{,}0\ tf.m) \cdot \frac{(-1{,}032\ m)}{4{,}070\ m^2} = 37{,}82\ tf$$

Para o pilar P6, como os momentos são nulos, as reações em cada estaca são idênticas:

$$N_{1a5} = \frac{200{,}0\ tf}{5} = 40{,}0\ tf$$

A partir do cálculo das reações, constata-se que, embora o pilar P6 apresente uma carga vertical superior ao pilar P5, a influência dos momentos fletores atuantes neste último resulta em uma solicitação individual mais elevada na estaca crítica. Portanto, caso se opte pela padronização do detalhamento do bloco para ambos os pilares, o dimensionamento deve ser governado pela situação mais desfavorável, adotando-se as solicitações provenientes do pilar P5.

Portanto:

$$N_d = 5 \cdot 43{,}04 \cdot 1{,}4 = 301{,}28\ tf$$

Passo 3 – Dimensionamento das Armaduras.

Neste bloco de 5 estacas, a verificação da biela não se faz necessária, uma vez que a inclinação α se encontra dentro dos limites preestabelecidos ($45° \leq \alpha \leq 55°$). Portanto, parte-se diretamente para a determinação das armaduras.

A armadura principal é dada por:

$$A_s = \frac{R'_{sd}}{f_{yd}};\ R_{sd} = \frac{N_d}{5\ tg(\alpha)};\ R'_{sd} = \frac{Rs}{2\cos(54°)}$$

$$R_{sd} = \frac{301{,}28\ tf}{5\ tg(46{,}26°)} = 57{,}66\ tf$$

$$R'_{sd} = \frac{57{,}66\ tf}{2\cos(54°)} = 49{,}05\ tf$$

$$A_s = \frac{49{,}05\ tf}{\frac{5\ tf/cm^2}{1{,}15}} = 11{,}28\ cm^2$$

As armaduras de malha inferior e superior são dadas por 25% da armadura principal:

$$A_{s,malha,inf} = 0{,}25 \cdot 11{,}28 = 2{,}82\ cm^2$$

A armadura de costela é dada por:

$$A_{s,costela} = \frac{1}{8}\ .A_{s,tot}\ ;A_{s,tot} = A_s\ .n_{est}$$

$$A_{s,costela} = \frac{1}{8}(11{,}28\ cm^2 \cdot 5) = 7{,}05\ cm^2$$

Por fim, a armadura de suspensão é dada por:

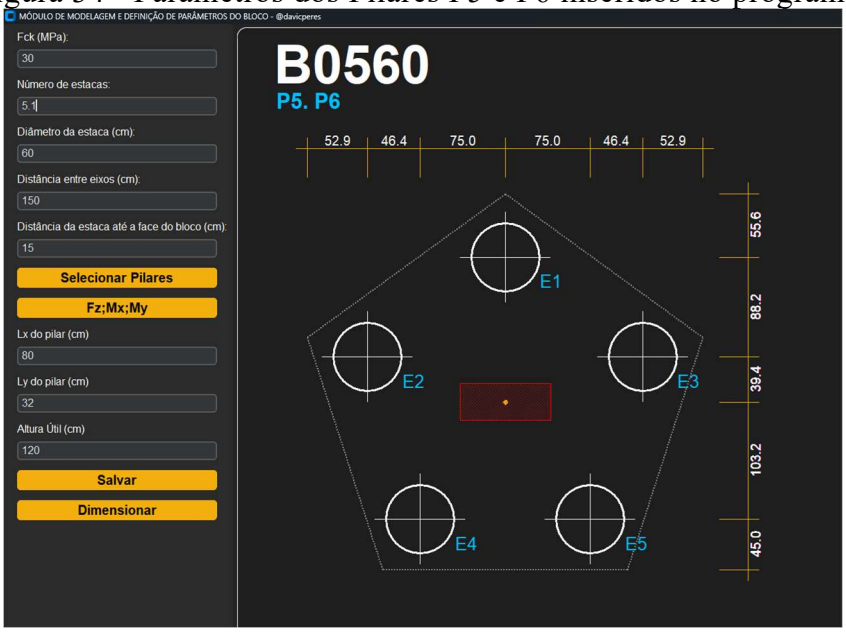
$$A_{s,susp,tot} = \frac{N_d}{7{,}5\ f_{yd}};\ A_{s,susp/face} = \frac{A_{s,susp,tot}}{5}$$

$$A_{s,susp,tot} = \frac{301{,}28\ tf}{7{,}5 \cdot \frac{5\ tf/cm^2}{1{,}15}} = 9{,}24\ cm^2$$

Passo 4 – Comparação com os Resultados do Programa.

A Figura 34, demonstra os dados do bloco de coroamento inseridos no programa.

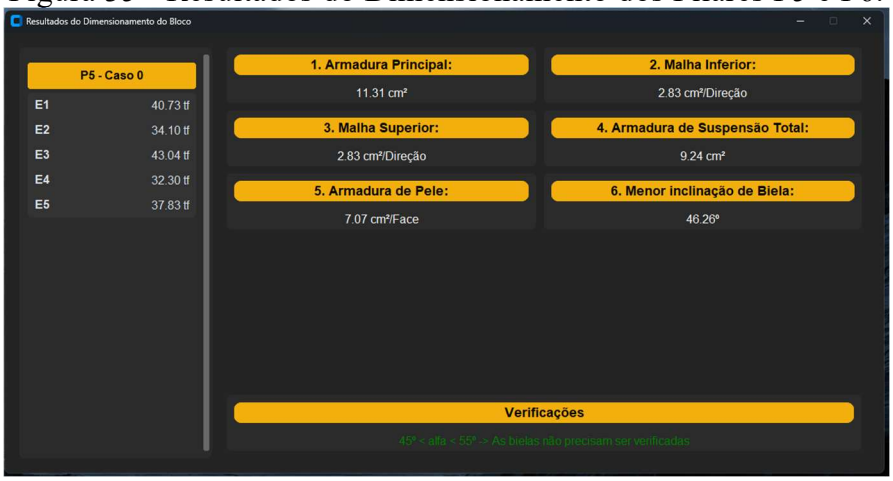
Figura 34 - Parâmetros dos Pilares P5 e P6 inseridos no programa.



Fonte: Autoria própria.

A Figura 35, apresenta os resultados do dimensionamento.

Figura 35 - Resultados do Dimensionamento dos Pilares P5 e P6.



Fonte: Autoria própria.

Com base nos resultados gerados pelo programa, elaborou-se a Tabela 2.

Tabela 2 - Comparação de resultados do bloco dos Pilares P5 e P6.

Parâmetro de Análise	Cálculo Analítico	Resultado Software	Diferença (%)
Reação nas Estacas			
E1	40,73 tf	40,73 tf	0,0%
E2	34,09 tf	34,10 tf	0,0%
E3	43,04 tf	43,04 tf	0,0%
E3	32,30 tf	32,30 tf	0,0%
E3	37,82 tf	37,83 tf	0,0%
Armaduras			
As,principal	11,28 cm²	11,31 cm²	0,3%
As,malha,inf	2,82 cm²/dir	2,83 cm²/dir	0,4%
As,malha,sup	2,82 cm²/dir	2,83 cm²/dir	0,4%
As,susp,total	9,24 cm²	9,24 cm²	0,0%
As,pele	7,05 cm²/face	7,07 cm²/face	0,3%

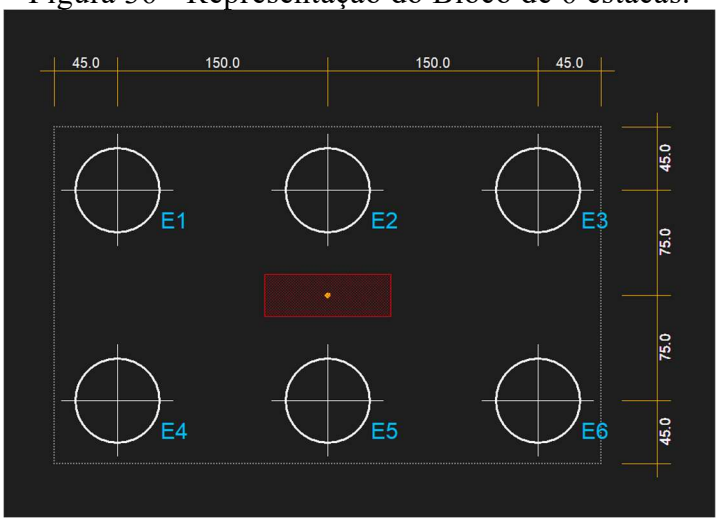
Fonte: Autoria própria.

Conforme evidenciado nos resultados, o programa foi capaz de distinguir corretamente qual pilar gerava a solicitação mais desfavorável para o bloco. A consistência dos valores de armadura obtidos válida a lógica de seleção automática da situação crítica, confirmando que a ferramenta garante a segurança do dimensionamento ao considerar o pior cenário entre os pilares selecionados.

9.3 Validação do programa para o método do CEB70

Este terceiro exemplo tem como objetivo validar a aplicação do **Método da Flexão (CEB-FIP 1970)** e o **processamento de múltiplas combinações de carregamento**. Este caso foi selecionado para demonstrar que, em blocos maiores, o programa é capaz de identificar que as armaduras em direções distintas (A_{sx} e A_{sy}) podem ser governadas por diferentes combinações. O caso de estudo selecionado consiste em um bloco sobre seis estacas, cuja configuração geométrica é apresentada na Figura 36.

Figura 36 - Representação do Bloco de 6 estacas.



Fonte: Autoria Própria.

Dados do Problema:

Diâmetro da estaca: $\emptyset_{est} = 60\ cm$

Distância entre eixos: $l = 150\ cm$

Seção do Pilar P10: 90 x 30 cm

Materiais: Concreto C30 ($fck = 30MPa$), Aço CA-50, $\gamma_f = 1.4$, $f_{yd} = 1.15$

Esforços Solicitantes: Para validar a robustez do algoritmo na leitura das combinações, foram considerados os seguintes esforços de entrada apresentados na Tabela 3, contemplando todas as cargas permanentes e acidentais presentes no Caso 0, o caso $Fz_{máx}$ que representa o caso com a maior carga vertical encontrada para aquele pilar e as ações geradas apenas pelo vento nas quatro direções principais.

Tabela 3 - Esforços do Pilar P10.

	Caso 0	Fz MAX (ELU)	Vento 1	Vento 2	Vento 3	Vento 4
Fz (tf)	168.0	177.0	-0.7	0.7	1.6	-1.6
Mx (tf.m)	-0.7	-2.2	-0.4	0.3	0.0	0.0
My (tf.m)	-11.5	5.5	1.0	-1.0	8.7	-8.7

Fonte: Autoria Própria.

Passo 1 – Definição da Altura Útil.

A definição da altura útil "d" partiu da verificação da condição de rigidez. Para o método da flexão, adota-se a recomendação de que a inclinação da biela da estaca mais afastada não seja inferior a $\alpha_{mín} = 40°$.

Calculando a distância do centro do pilar até a estaca mais afastada temos:

$$dist = \sqrt{(1{,}50\ m)^2 + (0{,}75\ m)^2} = 1{,}677\ m$$

A altura mínima é dada por:

$$d = dist \cdot tg(\alpha_{mín})$$

$$d = 1{,}677\ m \cdot tg(40°) = 1{,}41\ m$$

Com base nesse critério, adotou-se uma altura útil de $d = 150\ cm$.

Passo 2 – Determinação das Reações nas Estacas.

O dimensionamento requer a identificação das reações críticas em cada estaca. O programa processa automaticamente a superposição dos esforços, somando os casos de vento à combinação base (gravitacional). Foram geradas as seguintes combinações de cálculo:

1. Caso 0 (apenas gravitacional)
2. Caso de $Fz_{máx}$
3. Caso 0 + Vento 1
4. Caso 0 + Vento 2
5. Caso 0 + Vento 3
6. Caso 0 + Vento 4

A Tabela 4 apresenta os esforços já combinados para todos os casos apresentados.

Tabela 4 - Esforços combinados do Pilar P10.

	Caso 0	Fz MAX (ELU)	Caso0 + Vento 1	Caso0 + Vento 2	Caso0 + Vento 3	Caso0 + Vento 4
Fz (tf)	168.0	177.0	167.3	168.7	169.6	166.4
Mx (tf.m)	-0.7	-2.2	-1.1	-0.4	-0.7	-0.7
My (tf.m)	-11.5	5.5	-10.5	-12.5	-2.8	-20.2

Fonte: Autoria Própria.

A partir destes cenários, é calculado as reações verticais em cada estaca conforme apresenta a Tabela 5.

Tabela 5 - Reações nas estacas do bloco do Pilar P10.

REAÇÃO (tf)	Caso 0	Fz MAX (ELU)	Caso0 + Vento 1	Caso0 + Vento 2	Caso0 + Vento 3	Caso0 + Vento 4
RE1	30.7	28.6	30.4	31.0	29.0	32.3
RE2	30.6	28.3	30.2	30.9	28.9	32.2
RE3	30.4	27.9	30.0	30.8	28.8	32.1
RE4	25.6	31.1	25.7	25.4	27.8	23.4
RE5	25.4	30.7	25.6	25.3	27.6	23.2
RE6	25.3	30.4	25.4	25.3	27.5	23.1
Total	168.0	177.0	167.3	168.7	169.6	166.4

Fonte: Autoria Própria.

Passo 3 – Dimensionamento das Armaduras.

No Método do CEB-70, a armadura é dimensionada para resistir aos momentos fletores máximos atuantes nas seções de referência distanciadas $0{,}15\ a_p$ da face do pilar onde a_p é o lado do pilar naquela direção.

Para cada combinação de carga, calcularam-se os momentos em quatro seções críticas:

$$M_{x,dir}\ e\ M_{x,esq}\ para\ a\ armadura\ A_{s,x}$$

$$M_{y,sup}\ e\ M_{y,inf}\ para\ a\ armadura\ A_{s,y}$$

Os sufixos utilizados na notação dos momentos ($M_{dir}, M_{esq}, M_{sup}, M_{inf}$) indicam a posição geométrica da seção de referência em relação ao centro do pilar. Assim, os índices "*dir*" e "*esq*" referem-se às seções verticais localizadas, respectivamente, à direita e à esquerda do

pilar, enquanto "*sup*" e "*inf*" designam as seções horizontais situadas acima (superior) e abaixo (inferior) do pilar.

O momento em cada seção é obtido pelo somatório do produto das reações das estacas pelos seus respectivos braços de alavanca em relação à seção de referência. Estes momentos estão apresentados na Tabela 6.

Tabela 6 - Momentos nas seções do bloco do Pilar P10.

M (tf.m)	Caso 0	Fz MAX (ELU)	Caso0 + Vento 1	Caso0 + Vento 2	Caso0 + Vento 3	Caso0 + Vento 4
Mx (dir)	66.1	69.0	65.7	66.5	66.7	65.5
Mx (esq)	66.6	70.8	66.5	66.8	67.3	66.0
My (sup)	59.1	54.7	58.5	59.8	55.9	62.4
My (inf)	49.2	59.4	49.4	49.0	53.5	45.0

Fonte: Autoria Própria.

Com os momentos fletores determinantes identificados, as áreas de aço foram calculadas pela equação de flexão simples:

$$A_s = \frac{M_{d,máx}}{0{,}85 \cdot d \cdot f_{yd}}$$

A Tabela 7, apresenta as armaduras calculadas no eixo x e y para todos as combinações.

Tabela 7 - Armaduras para o Bloco do Pilar P10.

As (cm²)	Caso 0	Fz MAX (ELU)	Caso0 + Vento 1	Caso0 + Vento 2	Caso0 + Vento 3	Caso0 + Vento 4
Asx (dir)	16.69	17.44	16.58	16.79	16.85	16.53
Asx (esq)	16.83	17.88	16.79	16.87	16.99	16.67
Asy (sup)	14.93	13.82	14.77	15.10	14.12	15.75
Asy (inf)	12.43	15.01	12.49	12.38	13.51	11.36

Fonte: Autoria Própria.

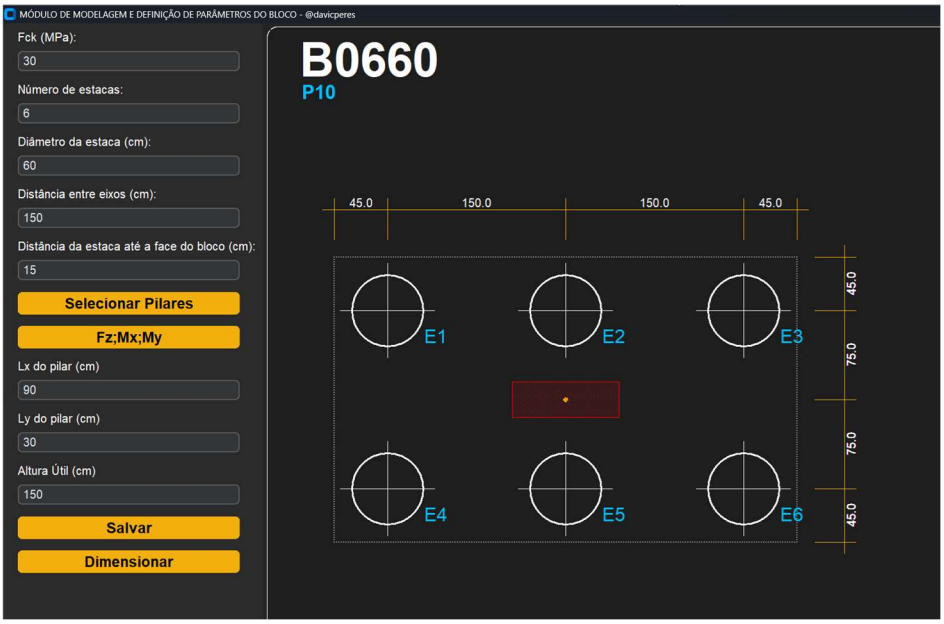
A análise revelou que o dimensionamento não foi governado por um único caso de carga, a armadura $A_{s,x}$ foi definida peça combinação $F_{z,máx}$ resultando em 17,88 cm^2.

A armadura $A_{s,y}$, por sua vez, foi governada pela combinação $Base + Vento\ 4$, resultando em 15,75 cm^2

Passo 4 - Comparação com os Resultados do Programa.

A Figura 37, representa os dados do bloco inseridos na janela de modelagem do programa.

Figura 37 - Parâmetros do Pilar P10 inseridos no programa.



Fonte: Autoria Própria.

A Figura 38, apresenta os resultados do dimensionamento.

Figura 38 - Resultados do Dimensionamento do Pilar P10.

Fonte: Autoria Própria.

Observa-se que o programa obteve os mesmos resultados que os cálculos analíticos, o que valida o módulo de flexão. Além disso, o fato de a ferramenta ter identificado corretamente que as armaduras em X e Y são regidas por combinações distintas demonstra a importância

prática de automatizar essa varredura de cargas, um processo que seria exaustivo e propenso a falhas se realizado manualmente.

10 CONCLUSÃO

No presente trabalho, propôs-se o desenvolvimento de uma ferramenta computacional para o dimensionamento automatizado de blocos de coroamento rígidos, abrangendo várias geometrias de bloco. Para a fundamentação dos algoritmos de cálculo, realizou-se uma revisão bibliográfica aprofundada, consolidando critérios de projeto alinhados à ABNT NBR 6118 e à literatura técnica consagrada. A metodologia adotada integrou o Modelo de Bielas e Tirantes, para blocos de até seis estacas, e o Método da Flexão (baseado no CEB-FIP, 1970), para arranjos mais complexos, garantindo a verificação da segurança através de critérios de inclinação das bielas e tensões de compressão nas bielas.

Diferentemente de soluções baseadas exclusivamente em planilhas, a implementação foi realizada na linguagem de programação Python, estruturada em módulos interconectados. A interface gráfica desenvolvida demonstrou-se eficaz em tornar a manipulação de parâmetros uma tarefa intuitiva e segura, respondendo à necessidade de praticidade no ambiente de projeto. Um dos principais diferenciais da ferramenta, o módulo de importação de pilares, atingiu o objetivo de otimizar o fluxo de trabalho: ao automatizar a leitura e o processamento de múltiplas combinações de esforços diretamente da planilha padrão, o programa não apenas reduziu o tempo de dimensionamento, mas mitigou significativamente a probabilidade de erros inerentes ao processo manual.

Não obstante a eficácia dos módulos implementados, reconhece-se que a ferramenta apresenta oportunidades de aprimoramento. Destaca-se, principalmente, a ausência de um módulo de detalhamento gráfico “cad”, funcionalidade que auxiliaria visualmente o usuário na interpretação do posicionamento das armaduras. Ademais, reitera-se que o programa opera dentro das fronteiras estabelecidas na seção de Delineamento e Limitações, restringindo-se às verificações de compressão das bielas e das inclinações, sem contemplar, nesta versão, análises complementares como o cisalhamento no Método da Flexão ou o dimensionamento de blocos com pilares associados ou de geometria complexa.

Ressalta-se, contudo, que tais delimitações de escopo não comprometem a utilidade da ferramenta para os fins propostos. A confiabilidade dos algoritmos de cálculo foi assegurada na etapa de validação, onde os resultados computacionais apresentaram plena convergência com os métodos analíticos de referência. Conclui-se, portanto, que o trabalho atingiu seu objetivo geral com êxito, entregando à comunidade técnica uma ferramenta intermediária que concilia a automação e a segurança dos softwares comerciais com a acessibilidade e a transparência fundamentais para o ambiente acadêmico e profissional.

Por fim, visando a continuidade da pesquisa e o aprimoramento da ferramenta computacional desenvolvida, sugerem-se as seguintes linhas de investigação e desenvolvimento:

1. Expansão do Detalhamento e Verificações: Implementar funcionalidades para a geração automática do detalhamento gráfico das armaduras (desenhos de execução e tabelas de ferro) para todas as geometrias de blocos.
2. Blocos com Pilares Associados: Desenvolver um módulo ou um software específico dedicado ao dimensionamento de blocos de coroamento submetidos à ação de múltiplos pilares ou pilares-parede, ampliando o escopo da ferramenta para situações de carregamento e geometria mais complexas.
3. Módulo de Quantitativo: Desenvolver um módulo para que o usuário do programa possa extrair quantitativos de aço, concreto e forma com base nos blocos gerados na pasta de trabalho.
4. Memória de Cálculo: Desenvolver um módulo para extração de um memorial de cálculo.

REFERÊNCIAS

AGUIAR, M. F. P. et al. **Estudo da eficiência de fundações com solução em radier estaqueado para prédios altos**. [S. l.: s. n.]. Disponível em: https://marcosporto.eng.br/wp-content/uploads/2024/09/Estudo-da-Eficiencia-de-Fundacoes-com-Solucao-em-Radier-1.pdf. Acesso em: 2 out. 2025.

ALONSO, U. R. **Exercícios de Fundações**. São Paulo: Edgard Blucher, 2000.

ALONSO, U. R. **Dimensionamento de fundações profundas.** São Paulo: Edgard Blucher, 1989.

AMERICAN CONCRETE INSTITUTE. ACI 318-14: **building code requirements for structural concrete**. Farmington Hills: ACI, 2014.

ARAÚJO, J. M. de. Curso de Concreto Armado. 4. ed. Rio Grande: Dunas, 2014.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 6118: **projeto de estruturas de concreto**. Rio de Janeiro: ABNT, 2023.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 6120: **ações para o cálculo de estruturas de edificações**. Rio de Janeiro: ABNT, 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 6122: **projeto e execução de fundações**. Rio de Janeiro: ABNT, 2019.

BLÉVOT, J.; FRÉMY, R. **Blocos sobre estacas – Parte A.** Tradução para o português 1967. Revista Ibracon de Estruturas e Materiais, São Paulo, n. 45, p. 15–24, 2021.

BLÉVOT, J.; FRÉMY, R. Semelles sur piex. **Analles d'Institut Techique du Bâtiment et des Travaux Publics**, Paris, v. 20, n. 230, p. 223-295, fev. 1967.

BARROS, R.; GIONGO, J. S.; OLIVEIRA, D. S. de. **Blocos de concreto armado sobre seis estacas:** simulação numérica e dimensionamento pelo método de bielas e tirantes. **Revista IBRACON de Estruturas e Materiais**, São Paulo, v. 7, n. 1, p. 1-23, fev. 2014. DOI: https://doi.org/10.1590/s1983-41952014000100002. Disponível em: Riem - vol 7 - nº 1 - artigo 1 - portugues.indd. Acesso em: 10 out. 2025.

BASTOS, P. S. S. **Blocos de Fundação.** Bauru: UNESP, 2023. (Apostila do curso de Estruturas de concreto III da Faculdade de Engenharia da Universidade Estadual Paulista)
1MORAES, M. C. **Estruturas de fundações.** Editora McGraw-HILL do Brasil, 1976.

CAMPO, J.C. de. **Elementos de Fundações em Concreto.** São Paulo: Oficina de Textos, 2015.

FUSCO, P. B. **Técnicas de armar as estruturas de concreto**. São Paulo: Editora Pini, 1994.
GARREFA, Fernando. GUERRA, Maria Eliza Alves. **Adoção de Parâmetros para a Verticalização em Araxá- MG**. Disponível em:

https://seer.ufu.br/index.php/Observatorium/article/download/45067/24020. Acesso em 02 de outubro de 2025.

MCKINNEY, W. Data structures for statistical computing in Python. *In*: PROCEEDINGS OF THE 9TH PYTHON IN SCIENCE CONFERENCE, 2010, Austin. **Proceedings [...]**. Austin: SciPy, 2010. p. 56-61.

MORAES, M. C. Estruturas de fundações. 3. ed. São Paulo: McGraw-HILL do Brasil, 1976.

MÖRSCH, E. **Teoría y práctica del hormigón armado**. v. 2. Barcelona: Gustavo Gili, 1948.

MUNHOS, F. S. **Análise do comportamento de blocos de concreto armado sobre estacas submetidos à ação de força centrada.** Dissertação (Mestrado em Engenharia Civil) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2004.

PYTHON SOFTWARE FOUNDATION. **Python Language Reference**: version 3.12. [S. l.], 2024. Disponível em: https://www.python.org/. Acesso em: 10 out. 2025.

RAMOS, F. A. C. **Análise numérica de blocos sobre dez estacas: cálculo das reações de apoio.** 2007. 156 p. Dissertação (Mestrado em Engenharia de estruturas) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

SAKAI, E. **Análise de blocos de concreto armado sobre estacas**. 2010. 107 f. Dissertação (Mestrado em Geotecnia e Construção Civil) – Universidade Federal de Goiás, Goiânia, 2010.

SCHIMANSKY, T. **CustomTkinter**: a modern and customizable python UI-library based on Tkinter. Versão 5.2.0. [S. l.], 2023. Disponível em: https://github.com/TomSchimansky/CustomTkinter. Acesso em: 10 out. 2025.

SOUZA, R.A; BITTENCOURT, T. N. **Non-Linear Analysis of Four-Pile Caps.** Revista IBRACON de Estruturas, v. 02, p310-319, 2006.

SOUZA, D. B. de; DELALIBERA, R. G. Análise numérica de blocos sobre doze estacas: estudo de caso. **Revista Principia**, *[S. l.]*, v. 61, n. 4, p. 890–914, 2024. DOI: 10.18265/1517-0306a2022id7248. Disponível em: https://periodicos.ifpb.edu.br/index.php/principia/article/view/7248. Acesso em: 25 out. 2025.

APÊNDICE

ANEXO A – Código mainwindow.py

```
from helpfunctions import *
from tkinter.filedialog import askdirectory,askopenfilename
from main import *
from moduloselecaopilares import Pilar
import pandas as pd

class MainWindow(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.screen_height = int(self.winfo_screenheight()/2.2)
        self.screen_width = int(self.winfo_screenwidth()/2.5)
        self.geometry(f"{self.screen_width}x{self.screen_height}+0+0")
        self.title("DIMENSIONAMENTO DE BLOCOS DE COROAMENTO - @davicperes")
        self.resizable(width=False, height=False)
        self.font_label = ("Arial", 16)
        self.font_entry = ("Arial", 16)
        self.font_button = ("Arial", 16, "bold")
        self.pastadetrabalho = None
        self.buttons_width = 300
        self.buttons_height = 40
        self.buttons_fgcolor = "#f2af0d"
        self.buttons_textcolor = "black"
        self.buttons_hovercolor = "#1ca188"
        self.buttons_disablecolor = "#8c6507"
        self.buttons_disablehovercolor = "#8c6507"
        self.draw()

    def GerarPastaPilares(self):
        diretorio = os.path.join(self.pastadetrabalho, "pilares.xlsx")

        try:
            planilha_dados = pd.read_excel(diretorio)
            planilha_dados = planilha_dados.fillna(0)
            PILARES = []
            for i in range(1, len(planilha_dados)):
                linhaplanilha = list(planilha_dados.iloc[i, :])
                PILARES.append(Pilar(linhaplanilha))
        except Exception as e:
            print(f"Erro ao ler pasta de pilares': {e}")
            PILARES = []

        pastapilares = os.path.join(self.pastadetrabalho, "Pilares")
        if os.path.exists(pastapilares):
            # Limpa a pasta removendo todos os arquivos e subpastas
            pass
        else:
            os.makedirs(pastapilares)
            print(f"Pasta '{pastapilares}' foi criada.")

        for pilar in PILARES:
            arquivo = os.path.join(pastapilares, pilar.name)
            with open(arquivo, "wb") as arquivo:
                pickle.dump(pilar, arquivo)

    def selectpath(self):
        self.pastadetrabalho = askdirectory(title="SELECIONE A PASTA DE TRABALHO")
        if self.pastadetrabalho:
            self.openprojectbutton.configure(fg_color=self.buttons_fgcolor,
hover_color=self.buttons_hovercolor)
            self.newprojectbutton.configure(fg_color=self.buttons_fgcolor,
hover_color=self.buttons_hovercolor)
            self.GerarPastaPilares()
```

```
    def newproject(self):
        if self.pastadetrabalho != None:
            self.withdraw()
            AppModelagem(pastadetrabalho=self.pastadetrabalho)

    def openproject(self):
        if self.pastadetrabalho != None:
            caminho = askopenfilename(title="SELECIONE O ARQUIVO .pkç DO BLOCO")
            if caminho:
                self.withdraw()
                AppModelagem(pastadetrabalho=self.pastadetrabalho,
caminhoblocoinicial=caminho, )

    def draw(self):
        img2 = MyImage(self, caminhoimagem="images/b3.png", scale=0.7)
        img2.place(relx=0.84,rely=0.64,anchor="center")

        img3 = MyImage(self, caminhoimagem="images/b4.png", scale=0.6)
        img3.place(relx=0.64,rely=0.56,anchor="center")

        self.frame =
ctk.CTkFrame(self,bg_color="#242424",border_color="gray",border_width=1)
        self.frame.grid_rowconfigure(0, weight=1)
        self.frame.grid_rowconfigure(1, weight=1)
        self.frame.grid_rowconfigure(2, weight=1)
        self.frame.grid_columnconfigure(0, weight=0)

        self.selectworkpath = ctk.CTkButton(master=self.frame,text="SELECIONAR
PASTA\n DE TRABALHO",
                                            width=self.buttons_width,
                                            height=self.buttons_height,
                                            command=self.selectpath,
                                            font=self.font_button,
                                            fg_color=self.buttons_fgcolor,
                                            text_color=self.buttons_textcolor,
                                            hover_color=self.buttons_hovercolor,)
        self.newprojectbutton = ctk.CTkButton(master=self.frame,
                                            text="NOVO BLOCO",
                                            width=self.buttons_width,
                                            height=self.buttons_height,
                                            command=self.newproject,
                                            font=self.font_button,
                                            fg_color=self.buttons_disablecolor,
                                            text_color=self.buttons_textcolor,

hover_color=self.buttons_disablehovercolor, )
        self.openprojectbutton = ctk.CTkButton(master=self.frame,
                                            text="ABRIR BLOCO",
                                            width=self.buttons_width,
                                            height=self.buttons_height,
                                            command=self.openproject,
                                            font=self.font_button,
                                            fg_color=self.buttons_disablecolor,
                                            text_color=self.buttons_textcolor,

hover_color=self.buttons_disablehovercolor, )

        frametitulo = ctk.CTkFrame(self,border_color="gray",border_width=1)
        titulo = ctk.CTkLabel(frametitulo, text="Programa de Dimensionamento\nde
Blocos de Coroamento", text_color="white",
                      font=("Arial", 30, "bold"))
        titulo.pack(padx=5,pady=5)
        frametitulo.place(relx=0.5, rely=0.1, anchor="center")
        self.frame.place(relx=0.03, rely=0.6, anchor="w")

        self.selectworkpath.grid(column=0,row=0,pady=(15,15),padx=10)
        self.newprojectbutton.grid(column=0,pady=(0,15))
```

```
        self.openprojectbutton.grid(column=0,row=2, pady=(0,15))

if __name__ == "__main__":
    mainwindow = MainWindow()
    mainwindow.mainloop()
```

ANEXO B – Código main.py

```
import tkinter
from tkinter.filedialog import askopenfilename, asksaveasfilename
import os
import customtkinter as ctk
import pickle
import tkinter as tk
from helpfunctions import *
from moduloselecaopilares import *
from blocos import *
from modulodedimensionamento import *
from typing import Union, Callable, List, Any

class MyButton(ctk.CTkButton):
    def __init__(self,master,**kwargs):
        kwargs.setdefault("text_color", "black")
        kwargs.setdefault("fg_color", "#f2af0d")
        kwargs.setdefault("width", 300)
        kwargs.setdefault("font", ("Arial", 20,"bold"))
        kwargs.setdefault("hover_color", "#1ca188")
        super().__init__(master,**kwargs)

    def pack_default(self, **kwargs):
        kwargs.setdefault("padx", (20,10))
        kwargs.setdefault("pady", (10,5))
        super().pack(**kwargs)

class MyLabel(ctk.CTkLabel):
    def __init__(self,master,**kwargs):
        kwargs.setdefault("text_color", "white")
        kwargs.setdefault("font", ("Arial", 16))
        super().__init__(master, **kwargs)

    def pack_default(self, **kwargs):
        kwargs.setdefault("padx", (20, 10))
        kwargs.setdefault("pady", (5, 0))
        super().pack(**kwargs)

class MyEntry(ctk.CTkEntry):
    def __init__(self,master,**kwargs):
        kwargs.setdefault("width", 300)
        kwargs.setdefault("font", ("Arial",16))
        super().__init__(master, **kwargs)

    def pack_default(self, **kwargs):
        kwargs.setdefault("padx", (20, 10))
        kwargs.setdefault("pady", (5, 5))
        super().pack(**kwargs)

# CLASSE PRINCIPAL DA INTERFACE GRÁFICA (GUI)
# ================================================================================#
class AppModelagem(ctk.CTkToplevel):
    def __init__(self, pastadetrabalho = None,caminhoblocoinicial = None):
        super().__init__()
        # --- Configurações da Janela Principal ---
        self.title("MÓDULO DE MODELAGEM E DEFINIÇÃO DE PARÂMETROS DO BLOCO -
@davicperes")
        self.screen_height = self.winfo_screenheight()
        self.screen_width = self.winfo_screenwidth()
        # self.geometry(f"{self.screen_width}x{self.screen_height}+0+0")
        self.state('zoomed')  # Abre maximizado
        self.canvas_scale = 2.0

        self.pastadetrabalho = pastadetrabalho

        self.cor_canvas = "#1f1e1f"
```

```
        self.lift()
        self.focus_set()
        #self.grab_set()

        # --- Lógica de fechamento ---
        self.protocol("WM_DELETE_WINDOW", self.on_closing)

        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(0, weight=1)

        sidebar_width = 350
        self.sidebar = ctk.CTkFrame(self, width=sidebar_width)
        self.sidebar.grid(row=0, column=0, sticky="ns")

        self.canvas_container = ctk.CTkFrame(self, corner_radius=15,
fg_color="#1f1e1f", border_width=1, border_color="white")
        self.canvas_container.grid(row=0, column=1, padx=5, pady=5, sticky="nsew")

        # Configure o grid do container para o canvas expandir dentro dele
        self.canvas_container.grid_rowconfigure(0, weight=1)
        self.canvas_container.grid_columnconfigure(0, weight=1)

        self.canvas = tk.Canvas(self.canvas_container, bg=self.cor_canvas,
highlightthickness=0)
        self.canvas.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)

        self.canvas.bind("<Configure>", self.update_canvas)

        self.bind("<MouseWheel>", self.zoom_scroll)  # Windows e Mac
        # self.bind("<Control-Up>", self.zoom_in)
        # self.bind("<Control-Down>", self.zoom_out)

        self.fck_var = ctk.StringVar(value=30)
        self.estaqueamento_var = ctk.StringVar(value="4")
        self.diametro_estacas_var = ctk.StringVar(value=60)
        self.dist_estacas_var = ctk.StringVar(value=150)
        self.angulo_var = ctk.StringVar(value=0)
        self.excentricidade_var = ctk.StringVar(value="0,0")
        self.distF_var = ctk.StringVar(value=15)
        self.pilar_lx_var = ctk.StringVar(value=60)
        self.pilar_ly_var = ctk.StringVar(value=20)
        self.alturautilbloco_var = ctk.StringVar(value=100)

        self.pilaresselecionados = []
        self.nomepilares = ""

        if caminhoblocoinicial != None:
            self.abrir_bloco_salvo(caminhoblocoinicial)

        self.draw()
        self.update()

    def abrir_bloco_salvo(self,caminhobloco):
        with open(f"{caminhobloco}", "rb") as arquivo:
            self.bloco_atual = pickle.load(arquivo)

        self.fck_var.set(self.bloco_atual.fck)
        self.estaqueamento_var.set(self.bloco_atual.estacas_input)
        self.diametro_estacas_var.set(self.bloco_atual.diametro_estacas)
        self.dist_estacas_var.set(self.bloco_atual.dist_estacas)
        self.angulo_var.set(self.bloco_atual.ang_rotacao)
        self.excentricidade_var.set(",".join([str(i) for i in
self.bloco_atual.excentricidade]))
        self.distF_var.set(self.bloco_atual.distF)
        self.pilar_lx_var.set(self.bloco_atual.pilar_lx)
        self.pilar_ly_var.set(self.bloco_atual.pilar_ly)
        self.pilaresselecionados = self.bloco_atual.listapilares
        self.alturautilbloco_var.set(self.bloco_atual.alturautilbloco)
```

```
    def draw(self):
        # --- DEFINIÇÃO DE ESTILOS E LISTAS ---

        # Definições para os pares de Label e Entry
        objects_efinitions = [
            ("entry", "Fck (MPa):", self.fck_var, "entry_fck"),
            ("listspinbox", "Fck (MPa):"),
            ("entry", "Número de estacas:", self.estaqueamento_var,
"entry_qtd_estacas"),
            ("entry", "Diâmetro da estaca (cm):", self.diametro_estacas_var,
"entry_diametro_estacas"),
            ("entry", "Distância entre eixos (cm):", self.dist_estacas_var,
"entry_dist_estacas"),
            #("entry", "Ângulo de rotação (°):", self.angulo_var, "entry_angulo"),
            #("entry", "Excentricidade (x,y) (cm) Máx=30cm:",
self.excentricidade_var, "entry_excentricidade"),
            ("entry", "Distância da estaca até a face do bloco (cm):",
self.distF_var, "entry_distF"),
            ("button", "Selecionar Pilares", self.selecionarpilares),
            ("button", "Fz;Mx;My", self.CriarPilar),
            ("entry", "Lx do pilar (cm)", self.pilar_lx_var, "entry_Apilar"),
            ("entry", "Ly do pilar (cm)", self.pilar_ly_var, "entry_Bpilar"),
            ("entry", "Altura Útil (cm)", self.alturautilbloco_var,
"entry_alturautilbloco"),
            ("button", "Salvar", self.salvar_bloco),
            ("button", "Dimensionar", self.dimensionar),
        ]

        for obj in objects_efinitions:
            if obj[0] == "entry":
                label_text = obj[1]
                text_var = obj[2]
                attr_name = obj[3]
                label = MyLabel(self.sidebar, text=label_text)
                label.pack_default(anchor="w")
                entry = MyEntry(self.sidebar, textvariable=text_var)
                setattr(self, attr_name, entry)
                entry.bind("<KeyRelease>", self.update_canvas)
                entry.pack_default(anchor="w")

            elif obj[0] == "button":
                text = obj[1]
                command = obj[2]
                button = MyButton(self.sidebar, text=text, command=command)
                button.pack_default(anchor="w")

    def on_closing(self):
        # Esta função é chamada quando o usuário clica no 'X'
        try:
            self.master.deiconify()  # Reexibe a janela principal
        except:
            pass

        self.destroy()  # Destrói a janela de modelagem

    def dimensionar(self):
        if self.bloco_atual.estaqueamento_custom == False and
(int(self.bloco_atual.n_estacas) in [2,3,4,5] or (self.bloco_atual.n_estacas > 6
and self.bloco_atual.n_estacas < 7)) :
            BielasTirantes(self,self.bloco_atual)
        else:
            CEBFIP(self,self.bloco_atual)

    def salvar_bloco(self):
        # 1. Garante que o self.bloco_atual esteja com os dados mais recentes da
tela
        self.update_canvas()
```

```
        if not hasattr(self.bloco_atual, 'n_estacas') or self.bloco_atual.n_estacas
== 0:
            PopupAviso(self, text=f"Não é possível salvar um bloco sem estacas !",
color="red")
            return

        # 2. Define o diretório e o cria se ele não existir
        if self.pastadetrabalho is not None:
            diretorio = os.path.join(self.pastadetrabalho, "BlocosSalvos")
            if not os.path.exists(diretorio):
                os.makedirs(diretorio)
        else:
            diretorio = ""

        # 3. Gera o nome de arquivo sugerido (ex: B0390.pkl)
        # :02d formata o número para ter sempre dois dígitos (ex: 3 -> 03)
        nome_sugerido = f"{self.bloco_atual.titulobloco}.pkl"

        # 4. Abre o diálogo "Salvar Como..."
        caminho_final = asksaveasfilename(
            initialdir=diretorio,
            title="Salvar Bloco Como...",
            initialfile=nome_sugerido,
            filetypes=(("Arquivos de Bloco Pickle", "*.pkl"), ("Todos os arquivos",
"*.*")),
            defaultextension=".pkl"
        )

        # 5. Verifica se o usuário selecionou um caminho ou cancelou
        if not caminho_final:
            PopupAviso(self, text="Salvamento cancelado pelo usuário.")
            return

        # 6. Salva o objeto do bloco no caminho escolhido pelo usuário

        self.pilaresselecionados = [pilar for pilar in self.pilaresselecionados if
pilar.name != 'Pilar de Teste']
        try:
            with open(caminho_final, "wb") as arquivo:
                pickle.dump(self.bloco_atual, arquivo)

            nome_final_arquivo = os.path.basename(caminho_final)
            PopupAviso(self, text=f"Bloco salvo com sucesso como:
{nome_final_arquivo}",color="green")

        except Exception as e:
            PopupAviso(self, text=f"Erro ao salvar o bloco: {e}", color="red")

    def zoom_in(self, event=None):
        """Aumenta o fator de escala (Zoom In), com limite máximo de 4.0."""
        incremento = 0.1
        nova_escala = self.canvas_scale + incremento

        # Garante que a escala não passe de 4.0
        self.canvas_scale = min(4.0, nova_escala)
        self.update_canvas()  # Redesenha o canvas com a nova escala

    def zoom_out(self, event=None):
        """Reduz o fator de escala (Zoom Out), com limite mínimo de 1.0."""
        incremento = 0.1
        nova_escala = self.canvas_scale - incremento

        # Garante que a escala não seja menor que 1.0
        self.canvas_scale = max(1.0, nova_escala)
        self.update_canvas()  # Redesenha o canvas com a nova escala

    def zoom_scroll(self,event=None):
```

```
        if event.delta > 0:
            self.zoom_in()
        else:
            self.zoom_out()

    def CriarPilar(self):
        self.pilaresselecionados = []
        dialogo = CriacaoPilar(master=self, callback=self.getpilaresselecionados)
        self.wait_window(dialogo)
        print(self.pilaresselecionados)

    def selecionarpilares(self):
        self.pilaresselecionados = [pilar for pilar in self.pilaresselecionados if
pilar.name != 'Pilar de Teste']
        dialogo = SelecaoPilares(pastadetrabalho =
self.pastadetrabalho,callback=self.getpilaresselecionados,jaselecionados=self.pilar
esselecionados)
        self.wait_window(dialogo)

    def getpilaresselecionados(self,pilares):
        self.pilaresselecionados = pilares
        self.update_canvas()

    def update_canvas(self, event=None):
        """Função central que lê os dados dos Entries e redesenha o canvas."""
        try:
            diametro = float(self.diametro_estacas_var.get())
        except ValueError:
            diametro = 60
        try:
            distancia = float(self.dist_estacas_var.get())
        except ValueError:
            distancia = 150

        try:
            angulo = float(self.angulo_var.get())
        except ValueError:
            angulo = 0.0

        try:
            excentricidade = [int(x) for x in
self.excentricidade_var.get().split(",")]
            if len(excentricidade) < 2:
                excentricidade.append(0)

            if excentricidade[0] > 30:
                excentricidade[0] = 30
            elif excentricidade[0] < -30:
                excentricidade[0] = -30

            if excentricidade[1] > 30:
                excentricidade[1] = 30
            elif excentricidade[1] < -30:
                excentricidade[1] = -30

        except ValueError:
            excentricidade = [0,0]


        try:
            fck = (int(self.fck_var.get()))
        except ValueError:
            fck = 30

        try:
            pilar_lx = int(self.pilar_lx_var.get())
        except ValueError:
            pilar_lx = 60
```

```
        try:
            pilar_ly = int(self.pilar_ly_var.get())
        except ValueError:
            pilar_ly = 25

        try:
            alturautilbloco = float(self.alturautilbloco_var.get())
        except ValueError:
            alturautilbloco = 100

        try:
            distF = int(self.distF_var.get())
        except ValueError:
            distF = 15

        # --- Cria o objeto BLOCO passando a string bruta ---
        # A lógica de decisão foi movida para dentro da classe Bloco

        self.bloco_atual = BLOCO(
            estacas_input=self.estaqueamento_var.get(),  # Passa o texto
diretamente
            dist_estacas=distancia,
            ang_rotacao=angulo,
            diametro_estaca=diametro,
            excentricidade = excentricidade,
            distF = distF,
            listapilares=self.pilaresselecionados,
            fck = fck,
            pilar_lx = pilar_lx,
            pilar_ly = pilar_ly,
            alturautilbloco = alturautilbloco
        )

        canvas_width = self.canvas.winfo_width()
        canvas_height = self.canvas.winfo_height()
        cx = canvas_width / 2
        cy = canvas_height / 2

        self.bloco_atual.draw(self.canvas, cx, cy, escala=self.canvas_scale)

if __name__ == "__main__":
    app = AppModelagem()
    app.mainloop()
```

ANEXO C – Código blocos.py

```
import math
from helpfunctions import *

class BLOCO:
    def __init__(self, estacas_input, dist_estacas=225, ang_rotacao=0,
diametro_estaca=90, excentricidade = [0,0], distF = 15, listapilares = [], fck=30,
pilar_lx = 50, pilar_ly = 50, alturautilbloco = 100 ):
        self.listapilares = listapilares
        self.nomepilares = [pilar.name for pilar in self.listapilares]

        self.dist_estacas = dist_estacas
        self.ang_rotacao = ang_rotacao
        self.ang_rad = math.radians(self.ang_rotacao)
        self.diametro_estacas = diametro_estaca
        self.excentricidade = excentricidade
        self.distF = distF
        self.estacas_input = estacas_input
        self.fck = fck
        self.pilar_lx = pilar_lx
        self.pilar_ly = pilar_ly
        self.alturautilbloco = alturautilbloco

        self.cor_estaca = "white"
        self.custom_coords = self.parse_coords_string(self.estacas_input)
        self.gerar_estaqueamento()

    def gerar_estaqueamento(self):
        if self.custom_coords:
            coords_base = self.custom_coords
            self.n_estacas = len(coords_base)
            self.estaqueamento_custom = True
        else:
            self.estaqueamento_custom = False
            try:
                self.n_estacas = float(self.estacas_input)

                coords_base = self.definir_estacas_padrao()
            except (ValueError, TypeError):
                self.n_estacas = 0
                coords_base = []

        self.titulobloco =
f"B{str(int(self.n_estacas)).zfill(2)}{int(self.diametro_estacas)}"

        self.estacas_coords = []
        for x, y in coords_base:
            x_rot = x * math.cos(self.ang_rad) - y * math.sin(self.ang_rad)
            y_rot = x * math.sin(self.ang_rad) + y * math.cos(self.ang_rad)
            self.estacas_coords.append([x_rot, y_rot])

    def parse_coords_string(self, coord_str):
        """Tenta converter uma string como 'x1,y1; x2,y2' em uma lista de
coordenadas."""
        try:
            coords_list = []
            pairs = str(coord_str).split(';')
            if len(pairs) == 1 and ',' not in pairs[0]: return None
            for pair in pairs:
                if ',' in pair:
                    x_str, y_str = pair.split(',')
                    coords_list.append([float(x_str), float(y_str)])
            return coords_list if coords_list else None
        except (ValueError, IndexError):
            return None

    def definir_estacas_padrao(self):
```

```
        """Calcula as posições (x, y) das estacas com base no número e
geometria."""
        n = self.n_estacas
        coords_base = []

        if n == 1:
            coords_base = [[0, 0]]

        elif int(n) == 2:
            if n == 2:
                x = self.dist_estacas / 2
                coords_base = [[-x, 0], [x, 0]]
            else:
                x = self.dist_estacas / 2
                coords_base = [[0, -x], [0, x]]


        elif int(n) == 3:
            caso = 10*(n-int(n))

            r = self.dist_estacas / (3 ** 0.5)
            for i in range(3):
                ang_i = (math.pi / 2) + (i * 2 * math.pi / 3) + caso*math.pi/2
                coords_base.append([r * math.cos(ang_i), r * math.sin(ang_i)])

        elif n == 4:
            d = self.dist_estacas / 2
            coords_base = [[-d, -d], [d, -d], [-d, d], [d, d]]

        elif int(n) == 5:
            if n == 5:
                d = self.dist_estacas / 2
                coords_base = [[0, 0], [-d, -d], [d, -d], [-d, d], [d, d]]

            else:
                caso = 10*(n-int(n))
                r = (self.dist_estacas/2) / (math.sin(36/180*math.pi))
                for i in range(5):
                    ang_i = 0 + i * math.pi*0.4 - 18/180*math.pi * caso
                    coords_base.append([r * math.cos(ang_i), r * math.sin(ang_i)])

        elif int(n) == 6:
            if n == 6:
                coords_base = self.gerar_grid_estacas(colunas=3, linhas=2)

            else:
                caso = 10*(n-int(n))
                r = (self.dist_estacas/2) / (math.sin(30/180*math.pi))
                for i in range(6):
                    ang_i = 0 + i * math.pi/3 + 15/180*math.pi * (caso-1)
                    coords_base.append([r * math.cos(ang_i), r * math.sin(ang_i)])

        elif n == 7:
            coords_base = [[0, 0]]
            r = self.dist_estacas
            for i in range(6):
                ang_i = (i * math.pi / 3)
                coords_base.append([r * math.cos(ang_i), r * math.sin(ang_i)])

        elif n == 8:
            coords_base = self.gerar_grid_estacas(colunas=3, linhas=3)
            coords_base.remove([0, 0])

        elif n == 9:
            coords_base = self.gerar_grid_estacas(colunas=3, linhas=3)

        elif n == 10:
            coords_base = self.gerar_grid_estacas(colunas=5, linhas=2)
```

```
        elif n == 12:
            coords_base = self.gerar_grid_estacas(colunas=4, linhas=3)

        tolerance = 2
        coords_base = sorted(coords_base,
                             key=lambda coord: (round(coord[1], tolerance),
round(coord[0], tolerance)))

        return coords_base

    def gerar_grid_estacas(self, colunas, linhas):
        """Função auxiliar para criar arranjos retangulares centrados na origem."""
        coords = []
        offset_x = -self.dist_estacas * (colunas - 1) / 2
        offset_y = -self.dist_estacas * (linhas - 1) / 2
        for i in range(linhas):
            y = offset_y + i * self.dist_estacas
            for j in range(colunas):
                x = offset_x + j * self.dist_estacas
                coords.append([x, y])
        return coords

    def desenhar_contorno_bloco(self, canvas, cx, cy, escala, distF=15.0):

        """
        Calcula e desenha o retângulo de contorno do bloco de coroamento.
        """
        # Se não houver estacas, não faz nada.
        if not self.estacas_coords:
            self.p1lim = (0,0)
            self.p2lim = (0,0)

        # Inicializa os limites com valores extremos
        min_x, max_x = float('inf'), float('-inf')
        min_y, max_y = float('inf'), float('-inf')

        raio = self.diametro_estacas / 2

        # Encontra a "caixa" que envolve todas as estacas
        for estaca_x, estaca_y in self.estacas_coords:
            min_x = min(min_x, estaca_x - raio)
            max_x = max(max_x, estaca_x + raio)
            min_y = min(min_y, estaca_y - raio)
            max_y = max(max_y, estaca_y + raio)

        # Calcula as coordenadas do retângulo do bloco no canvas,
        # incluindo a borda e aplicando a escala.
        p1_x = cx + (min_x - distF) * escala
        p1_y = cy + (min_y - distF) * escala
        p2_x = cx + (max_x + distF) * escala
        p2_y = cy + (max_y + distF) * escala

        self.p1lim = (p1_x, p1_y)
        self.p2lim = (p2_x, p2_y)

        self.bloco_lx = (p2_x - p1_x)/escala
        self.bloco_ly = (p2_y - p1_y)/escala

        if int(self.n_estacas)==3 and self.estaqueamento_custom == False:
            l = self.dist_estacas
            r = self.diametro_estacas / 2
            R = l / 3 ** 0.5
            f = self.distF

            teta = math.atan((l / 3 + (r + f - R) / 3 ** 0.5) / (R + r + f))
            j = (R + r + f) / math.cos(teta)
```

```
            alfas = [
                teta,
                teta + (math.pi * 2 / 3 - 2 * teta),
                teta + (math.pi * 2 / 3 - 2 * teta) + 2 * teta,
                teta + (math.pi * 2 / 3 - 2 * teta) + 2 * teta + (math.pi * 2 / 3 -
2 * teta),
                teta + (math.pi * 2 / 3 - 2 * teta) + 2 * teta + (math.pi * 2 / 3 -
2 * teta) + 2 * teta,
                teta + (math.pi * 2 / 3 - 2 * teta) + 2 * teta + (math.pi * 2 / 3 -
2 * teta) + 2 * teta + (math.pi * 2 / 3 - 2 * teta),
                     ]

            pontos = []
            rotacao = (self.n_estacas - 3)*10*math.pi/2

            for alfa in alfas:
                x = cx + j * math.sin(alfa-rotacao) * escala
                y = cy + j * math.cos(alfa-rotacao) * escala
                pontos.append([x, y])

            p1_x = min([p[0] for p in pontos])
            p1_y = min([p[1] for p in pontos])
            p2_x = max([p[0] for p in pontos])
            p2_y = max([p[1] for p in pontos])

            self.p1lim = (p1_x, p1_y)
            self.p2lim = (p2_x, p2_y)

            for i,ponto in enumerate(pontos):
                xa,ya = pontos[i-1]
                x,y = ponto
                canvas.create_line(xa, ya, x, y, fill="#888888", width=2, dash=(5,
3))

        elif int(self.n_estacas) == 5 and self.n_estacas > int(self.n_estacas) and
self.estaqueamento_custom == False:
            l = self.dist_estacas
            r = self.diametro_estacas / 2
            f = self.distF
            R = (l/2)/math.sin(36*math.pi/180) + (r + f)/math.cos(36*math.pi/180)

            nrot = (self.n_estacas - int(self.n_estacas))*10-1
            ang = -math.pi/2 - nrot*(18/180*math.pi)
            pontos = []
            for i in range(5):
                x = cx + R * math.cos(ang) * escala
                y = cy + R * math.sin(ang) * escala
                pontos.append([x, y])
                ang += 72*math.pi/180

            p1_x = min([p[0] for p in pontos])
            p1_y = min([p[1] for p in pontos])
            p2_x = max([p[0] for p in pontos])
            p2_y = max([p[1] for p in pontos])

            self.p1lim = (p1_x, p1_y)
            self.p2lim = (p2_x, p2_y)

            for i,ponto in enumerate(pontos):
                xa,ya = pontos[i-1]
                x,y = ponto
                canvas.create_line(xa, ya, x, y, fill="#888888", width=2, dash=(5,
3))
```

```
        elif (int(self.n_estacas) == 6 and self.n_estacas > int(self.n_estacas)) or
self.n_estacas == 7 and self.estaqueamento_custom == False:

            l = self.dist_estacas
            r = self.diametro_estacas / 2
            f = self.distF
            R = (l / 2) / math.sin(30 * math.pi / 180) + (r + f) / math.cos(30 *
math.pi / 180)

            if self.n_estacas == 7:
                nrot = 0
            else:
                nrot = (self.n_estacas - int(self.n_estacas)) * 10 - 1

            ang = nrot * (15 / 180 * math.pi)
            pontos = []
            for i in range(6):
                x = cx + R * math.cos(ang) * escala
                y = cy + R * math.sin(ang) * escala
                pontos.append([x, y])
                ang += 60 * math.pi / 180

            p1_x = min([p[0] for p in pontos])
            p1_y = min([p[1] for p in pontos])
            p2_x = max([p[0] for p in pontos])
            p2_y = max([p[1] for p in pontos])

            self.p1lim = (p1_x, p1_y)
            self.p2lim = (p2_x, p2_y)

            for i, ponto in enumerate(pontos):
                xa, ya = pontos[i - 1]
                x, y = ponto
                canvas.create_line(xa, ya, x, y, fill="#888888", width=2, dash=(5,
3))

        else:
            # Desenha o retângulo do bloco
            canvas.create_rectangle(
                p1_x, p1_y, p2_x, p2_y,
                outline="#888888",  # Cor do contorno
                width=2,
                dash=(5, 3)  # Linha tracejada para o contorno
            )

    def update_titulo_bloco(self,titulo):
        self.titulo_bloco = titulo

    def draw(self, canvas, cx, cy, escala=1.0):
        """Desenha as estacas no canvas na posição correta."""
        canvas.delete("all")
        cg_x = cx + self.excentricidade[0] * escala
        cg_y = cy + self.excentricidade[1] * escala

        canvas.create_oval(cx - 1, cy - 1, cx + 1, cy + 1, fill="gray",
outline="gray", width=2*escala)
        canvas.create_oval(cg_x - 1, cg_y - 1, cg_x + 1, cg_y + 1, fill="#f2af0d",
outline="#f2af0d", width=3*escala)


        self.desenhar_contorno_bloco(canvas, cx, cy, escala, distF = self.distF)

        # Desenha o título do bloco
        canvas.create_text(50, 50, text=self.titulobloco,
                           font=("Arial", 60, "bold"), fill="white",
anchor="w")

        canvas.create_text(54, 105, text=",
```

```
".join(organizar_titulos(self.nomepilares)), font=("Arial", 25,"bold"),
                                fill="deep sky blue", anchor="w")

        # Desenha a cotagem
        positions = [(cx+dx*escala,cy+dy*escala) for (dx,dy) in
self.estacas_coords]
        positions.append(self.p1lim)
        positions.append(self.p2lim)
        positions.append((cg_x,cg_y))
        positions.append((cx,cy))

        for i, (x, y) in enumerate(positions):
            canvas.create_line((x, self.p1lim[1] - 15 * escala), (x, self.p1lim[1]
- 50 * escala), fill="#f2af0d")
            canvas.create_line((self.p2lim[0] + 15*escala, y), (self.p2lim[0] +
50*escala, y), fill="#f2af0d")

        listaX = sorted([pos[0] for pos in positions])
        listaY = sorted([pos[1] for pos in positions])

        for i, x in enumerate(listaX):
            if i > 0:
                xa,xb = listaX[i-1], listaX[i]
                xcentro = media(xa,xb)
                dist = round((xb - xa) / escala, 1)
                if dist != 0:
                    canvas.create_text(xcentro, self.p1lim[1]-40*escala,
text=str(dist), font=("Arial", 15),anchor="s",fill="white")

            for i, y in enumerate(listaY):
                if i > 0:
                    ya, yb = listaY[i - 1], listaY[i]
                    ycentro = media(ya, yb)
                    dist = round((yb - ya) / escala, 1)
                    if dist != 0:
                        canvas.create_text(self.p2lim[0]+40*escala,ycentro,
text=str(dist), font=("Arial", 15),
                                           anchor="n", fill="white",angle=90)

        canvas.create_line((self.p1lim[0]-10*escala, self.p1lim[1] - 40*escala),
(self.p2lim[0] + 10 * escala, self.p1lim[1]-40*escala), fill="#f2af0d")
        canvas.create_line((self.p2lim[0] + 40 * escala, self.p1lim[1] -
10*escala), (self.p2lim[0] + 40 * escala, self.p2lim[1] + 10*escala),
fill="#f2af0d")

        # Desenha as estacas
        for i, (dx, dy) in enumerate(self.estacas_coords):
            raio = (self.diametro_estacas / 2) * escala
            x = cx + (dx * escala)
            y = cy + (dy * escala)

            canvas.create_oval(x - raio, y - raio, x + raio, y + raio,
outline=self.cor_estaca, width=3)

            e = 10 * escala
            canvas.create_line(x - raio - e, y, x + raio + e, y,
fill=self.cor_estaca, width=1)
            canvas.create_line(x, y - raio - e, x, y + raio + e,
fill=self.cor_estaca, width=1)

            # Desenha o texto da estaca
            font_size = max(8, int(15 * escala))  # Garante um tamanho mínimo para
a fonte

            canvas.create_text(x + raio + 1 * escala, y + raio + 1 * escala,
text=f"E{i + 1}", fill="deep sky blue",
                               font=("Arial", int(12*escala)), anchor="sw")
```

```
 # Desenha o pilar
        canvas.create_rectangle(cg_x - self.pilar_lx * escala / 2, cg_y -
self.pilar_ly * escala / 2,
                                cg_x + self.pilar_lx * escala / 2, cg_y +
self.pilar_ly * escala / 2, outline="red", fill="red",stipple = "gray12")
```

ANEXO D – Código modulodedimensionamento.py

```
import math
from helpfunctions import *

def verificarestaqueamento(Fz,Mx,My,estaqueamento):
    estaqueamento = [(pos[0]/100, pos[1]/100) for pos in estaqueamento]

    n_estacas = len(estaqueamento)
    Sx2 = sum([pos[0] ** 2 for pos in estaqueamento])
    Sy2 = sum([pos[1] ** 2 for pos in estaqueamento])

    reacoes = []
    for estaca in estaqueamento:
        xi = estaca[0]
        yi = estaca[1]

        if Sx2 == 0:
            Mx=0
            Sx2 = 1
        if Sy2 == 0:
            My=0
            Sy2 = 1

        R = Fz/n_estacas + Mx * xi/Sx2 + My * yi/Sy2
        reacoes.append(R)

    return reacoes

class BielasTirantes:
    def __init__(self, mainapp, classebloco):
        self.bloco = classebloco
        self.mainapp = mainapp
        self.pegar_parametros()
        self.gerar_combinacoes()

        if not len(self.pilares) > 0:
            PopupAviso(self.mainapp, text="Nenhum pilar foi
selecionado !",color="red")
        elif self.d < self.dmin_blocorigido():
            PopupAviso(self.mainapp, text=f"d = {self.d}cm < dmin =
{self.dmin_blocorigido()}cm !!!", color="red")
        elif self.alfa*180/math.pi > 55:
            alfa = round(self.alfa*180/math.pi,1)
            PopupAviso(self.mainapp, text=f"alfa = {alfa}° > 55° !!!", color="red")
        else:
            self.dimensionar()

    def pegar_parametros(self):
        self.pilares = self.bloco.listapilares
        self.fck = self.bloco.fck  # Mpa
        self.fcd = self.fck / 140  # tf/cm2
        self.fyd = 500 / 1.15 / 100  # tf/cm2
        self.diametro_estaca = self.bloco.diametro_estacas
        self.estaqueameto = self.bloco.estacas_coords
        self.area_pilar = self.bloco.pilar_lx * self.bloco.pilar_ly
        self.area_estaca = math.pi * self.diametro_estaca ** 2 / 4
        self.dist_estacas = self.bloco.dist_estacas
        self.d = self.bloco.alturautilbloco
        self.bloco_lx = self.bloco.bloco_lx
        self.bloco_ly = self.bloco.bloco_ly
        self.leqv_pilar = self.area_pilar ** 0.5

        if self.estaqueameto[0][1] == self.estaqueameto[1][1]:
            self.Lpilar = self.bloco.pilar_lx
            self.Bbloco = self.bloco_ly

        else:
```

```
            # Bloco Vertical
            self.Lpilar = self.bloco.pilar_ly
            self.Bbloco = self.bloco_lx

        self.alfa = self.calc_alfa(self.d)

    def gerar_combinacoes(self):
        self.combinacoes = []
        for pilar in self.pilares:
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.caso0[0], "MxTotal": pilar.caso0[1],
                                     "MyTotal": pilar.caso0[2], "NomeCombinacao":
"Caso 0"})
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.fzmax[0], "MxTotal": pilar.fzmax[1],
                                     "MyTotal": pilar.fzmax[2], "NomeCombinacao":
"Fz Max"})
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.mxmax[0], "MxTotal": pilar.mxmax[1],
                                     "MyTotal": pilar.mxmax[2], "NomeCombinacao":
"My Max"})
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.mymax[0], "MxTotal": pilar.mymax[1],
                                     "MyTotal": pilar.mymax[2], "NomeCombinacao":
"My Max"})

            for i, combinacao in enumerate(pilar.casos_adicionais):
                FzTotal = combinacao[0] + pilar.caso0[0]
                MxTotal = combinacao[1] + pilar.caso0[1]
                MyTotal = combinacao[2] + pilar.caso0[2]

                self.combinacoes.append(
                    {"TituloPilar": pilar.name, "FzTotal": FzTotal, "MxTotal":
MxTotal, "MyTotal": MyTotal,
                     "NomeCombinacao": f"Vento {i + 1}"})

    def verificar_estaqueamento(self):
        resultados = []
        nomes_combinacao = []
        for i, combinacao in enumerate(self.combinacoes):
            Fz = combinacao["FzTotal"]
            Mx = combinacao["MxTotal"]
            My = combinacao["MyTotal"]
            TituloPilar = combinacao["TituloPilar"]
            resultados.append(verificarestaqueamento(Fz, Mx, My,
self.estaqueameto))

            nomes_combinacao.append(f"{TituloPilar} -
{combinacao["NomeCombinacao"]}")

        pior_carga = max(resultados, key=lambda x: max(x))
        indice_pior_combincao = resultados.index(pior_carga)
        pior_combinacao = nomes_combinacao[indice_pior_combincao]

        return pior_carga, pior_combinacao

    def dmin_blocorigido(self):
        if int(self.bloco.n_estacas) == 2:
            if self.estaqueameto[0][1] == self.estaqueameto[1][1]:
                # Bloco Horizontal
                Lpilar = self.bloco.pilar_lx
            else:
                # Bloco Vertical
                Lpilar = self.bloco.pilar_ly

            self.dmin = (self.dist_estacas/2 - Lpilar/4)
            return self.dmin
```

```
        elif int(self.bloco.n_estacas) == 3:
            self.dmin = round(self.dist_estacas / 3 ** 0.5 - 0.3 * self.leqv_pilar,
1)
            return self.dmin

        elif int(self.bloco.n_estacas) == 4:
            self.dmin = round(self.dist_estacas / 2 ** 0.5 - self.leqv_pilar / (2 *
2 ** 0.5), 1)
            return self.dmin

        elif self.bloco.n_estacas == 5:
            self.dmin = round(2 ** 0.5 / 2 * (self.dist_estacas - self.leqv_pilar /
2), 1)
            return self.dmin

        elif int(self.bloco.n_estacas) == 5:
            self.dmin = round(0.85 * self.dist_estacas - 0.25 * self.leqv_pilar, 1)
            return self.dmin

    def calc_alfa(self,d):
        if int(self.bloco.n_estacas) == 2:
            alfa = math.atan(d / (self.dist_estacas / 2 - self.Lpilar / 4))
        elif int(self.bloco.n_estacas) == 3:
            alfa = math.atan(d / (self.dist_estacas / 3 ** 0.5 - 0.3 *
self.leqv_pilar))
        elif int(self.bloco.n_estacas) == 4:
            alfa = math.atan(d / (self.dist_estacas / 2 ** 0.5 - self.leqv_pilar /
(2 * 2 ** 0.5)))
        elif self.bloco.n_estacas == 5:
            alfa = math.atan(d / (2 ** 0.5 / 2 * (self.dist_estacas -
self.leqv_pilar / 2)))
        elif int(self.bloco.n_estacas) == 5:
            alfa = math.atan(d / (0.85 * self.dist_estacas - 0.25 *
self.leqv_pilar))

        return alfa

    def dimensionar(self):
        self.Nk = int(self.bloco.n_estacas) *
max(self.verificar_estaqueamento()[0])
        self.Nd = self.Nk * 1.4

        d = self.d
        alfa = self.alfa
        Nd = self.Nd
        AreaP = self.area_pilar
        AreaE = self.area_estaca
        l_pilareqv = self.leqv_pilar  # Lado do pilar equivalente
        l = self.dist_estacas
        fcd = self.fcd
        fyd = self.fyd
        l_pilar = self.Lpilar
        B_bloco = self.Bbloco

        if int(self.bloco.n_estacas) == 2:
            Kr = 0.95
            Rs = Nd * (2 * l - l_pilar) / (8 * d)
            As = 1.15 * Rs / fyd
            AsSup = 0.2 * As
            AsPele = 0.075 * B_bloco
            AsEstribo = AsPele  # cm2/m
            AsCostela = AsPele * (d + 5) / 100  # cm2/face
            self.sgmcd_pil = round(Nd / (AreaP * math.sin(alfa) ** 2) * 1000, 2)  #
kgf/cm2
            self.sgmcd_est = round(Nd / (2 * AreaE * math.sin(alfa) ** 2) * 1000,
2)  # kgf/cm2
            self.sgmcd_lim = round(1.4 * Kr * fcd * 1000, 2)  # kgf/cm2
            self.As = round(As, 2)
```

```
            self.AsSup = round(AsSup, 2)
            self.AsPele = AsPele
            self.AsEstribo = round(AsEstribo, 2)
            self.AsCostela = round(AsCostela, 2)

            self.textos = {"1. Armadura Principal:": f"{round(self.As, 2)} cm²",
                           "2. Armadura Superior:": f"{round(self.AsSup, 2)} cm²",
                           "3. Estribos:": f"{round(self.AsEstribo, 2)} cm²/m",
                           "4. Costela:": f"{round(self.AsCostela, 2)} cm²/face",
                           "5. Menor inclinação de Biela:": f"{round(self.alfa *
180 / math.pi, 2)}°"}

            self.textoverificacao = f"""σcd,lim = {self.sgmcd_lim} kgf/cm²\nσcd,est
= {self.sgmcd_est} kgf/cm²\nσcd,pil = {self.sgmcd_pil} kgf/cm²
                                       """

            if self.sgmcd_lim >= self.sgmcd_pil and self.sgmcd_lim >=
self.sgmcd_est:
                self.cor_textoverificacao = "green"
            else:
                self.cor_textoverificacao = "red"

        elif int(self.bloco.n_estacas) == 3:
            Kr = 0.9
            self.sgmcd_pil = round(Nd / (AreaP * math.sin(alfa) ** 2) * 1000, 2)  #
kgf/cm2
            self.sgmcd_est = round(Nd / (3 * AreaE * math.sin(alfa) ** 2) * 1000,
2)  # kgf/cm2
            self.sgmcd_lim = round(1.75 * Kr * fcd * 1000, 2)  # kgf/cm2
            self.As = Nd * 3 ** 0.5 / (27 * d * fyd) * (l * 3 ** 0.5 - 0.9 *
l_pilareqv)
            self.AsMalhaInf = 0.2 * self.As
            self.AsSuspFace = Nd / (3 * 4.5 * fyd)
            self.AsMalhaSup = 0.2 * self.As
            self.AsPele = 1 / 8 * 3 * self.As
            self.textos = {"1. Armadura Principal:": f"{round(self.As, 2)} cm²",
                           "2. Malha Inferior:": f"{round(self.AsMalhaInf, 2)}
cm²/Direção",
                           "3. Malha Superior:": f"{round(self.AsMalhaSup, 2)}
cm²/Direção",
                           "4. Armadura de Suspensão:": f"{round(self.AsSuspFace,
2)} cm²/Lado",
                           "5. Armadura de Pele:": f"{round(self.AsPele, 2)}
cm²/Face",
                           "6. Menor inclinação de Biela:": f"{round(self.alfa *
180 / math.pi, 2)}°"}

            self.textoverificacao = f"""σcd,lim = {self.sgmcd_lim} kgf/cm²\nσcd,est
= {self.sgmcd_est} kgf/cm²\nσcd,pil = {self.sgmcd_pil} kgf/cm²
                                       """

            if self.sgmcd_lim >= self.sgmcd_pil and self.sgmcd_lim >=
self.sgmcd_est:
                self.cor_textoverificacao = "green"
            else:
                self.cor_textoverificacao = "red"

        elif int(self.bloco.n_estacas) == 4:
            Kr = 0.95
            self.sgmcd_pil = round(Nd / (AreaP * math.sin(alfa) ** 2) * 1000, 2)  #
kgf/cm2
            self.sgmcd_est = round(Nd / (4 * AreaE * math.sin(alfa) ** 2) * 1000,
2)  # kgf/cm2
            self.sgmcd_lim = round(2.1 * Kr * fcd * 1000, 2)  # kgf/cm2
            self.As = Nd / (16 * d * fyd) * (2 * l - l_pilareqv)
            self.AsMalhaInf = 0.25 * self.As
            self.AsSuspFace = Nd / (4 * 6 * fyd)
            self.AsMalhaSup = 0.2 * 4 * self.As / 2
```

```
            self.AsPele = 1 / 8 * 4 * self.As
            self.textos = {"1. Armadura Principal:": f"{round(self.As, 2)} cm²",
                           "2. Malha Inferior:": f"{round(self.AsMalhaInf, 2)}
cm²/Direção",
                           "3. Malha Superior:": f"{round(self.AsMalhaSup, 2)}
cm²/Direção",
                           "4. Armadura de Suspensão:": f"{round(self.AsSuspFace,
2)} cm²/Lado",
                           "5. Armadura de Pele:": f"{round(self.AsPele, 2)}
cm²/Face",
                           "6. Menor inclinação de Biela:": f"{round(self.alfa *
180 / math.pi, 2)}°"}

            self.textoverificacao = f"""σcd,lim = {self.sgmcd_lim} kgf/cm²\nσcd,est
= {self.sgmcd_est} kgf/cm²\nσcd,pil = {self.sgmcd_pil} kgf/cm²
                                        """

            if self.sgmcd_lim >= self.sgmcd_pil and self.sgmcd_lim >=
self.sgmcd_est:
                self.cor_textoverificacao = "green"
            else:
                self.cor_textoverificacao = "red"

        elif self.bloco.n_estacas == 5:
            Kr = 0.9
            self.sgmcd_pil = round(Nd / (AreaP * math.sin(alfa) ** 2) * 1000, 2)  #
kgf/cm2
            self.sgmcd_est = round(Nd / (5 * AreaE * math.sin(alfa) ** 2) * 1000,
2)  # kgf/cm2
            self.sgmcd_lim = round(2.6 * Kr * fcd * 1000, 2)  # kgf/cm2
            # Neste caso existe um sgmcd_lim_est e um sgmcd_lim_pil com 2.1 e 2.6
respectivamente
            self.As = (4 / 5) * Nd / (16 * d * fyd) * (2 * l - l_pilareqv)
            self.AsMalhaInf = 0.25 * self.As
            self.AsSuspFace = Nd / (4 * 6 * fyd)
            self.AsMalhaSup = 0.25 * self.As
            self.AsPele = 1 / 8 * 4 * self.As
            self.textos = {"1. Armadura Principal:": f"{round(self.As, 2)} cm²",
                           "2. Malha Inferior:": f"{round(self.AsMalhaInf, 2)}
cm²/Direção",
                           "3. Malha Superior:": f"{round(self.AsMalhaSup, 2)}
cm²/Direção",
                           "4. Armadura de Suspensão:": f"{round(self.AsSuspFace,
2)} cm²/Lado",
                           "5. Armadura de Pele:": f"{round(self.AsPele, 2)}
cm²/Face",
                           "6. Menor inclinação de Biela:": f"{round(self.alfa *
180 / math.pi, 2)}°"}
            self.textoverificacao = f"""σcd,lim = {self.sgmcd_lim} kgf/cm²\nσcd,est
= {self.sgmcd_est} kgf/cm²\nσcd,pil = {self.sgmcd_pil} kgf/cm²
                                        """
            if self.sgmcd_lim >= self.sgmcd_pil and self.sgmcd_lim >=
self.sgmcd_est:
                self.cor_textoverificacao = "green"
            else:
                self.cor_textoverificacao = "red"

        elif int(self.bloco.n_estacas) == 5:
            Kr = 0.9
            self.sgmcd_pil = "SE alfa entre 45 e 55° as bielas não precisam ser
verificadas"
            self.sgmcd_est = ""
            self.sgmcd_lim = ""

            self.As = 0.725 * Nd / (5 * d * fyd) * (l - l_pilareqv / 3.4)
            self.AsMalhaInf = 0.25 * self.As
            self.AsSuspFace = Nd / (7.5 * fyd)
            self.AsMalhaSup = 0.25 * self.As
```

```
            self.AsPele = 1 / 8 * (5 * self.As)

            self.textos = {"1. Armadura Principal:": f"{round(self.As, 2)} cm²",
                           "2. Malha Inferior:": f"{round(self.AsMalhaInf, 2)}
cm²/Direção",
                           "3. Malha Superior:": f"{round(self.AsMalhaSup, 2)}
cm²/Direção",
                           "4. Armadura de Suspensão Total:":
f"{round(self.AsSuspFace, 2)} cm²",
                           "5. Armadura de Pele:": f"{round(self.AsPele, 2)}
cm²/Face",
                           "6. Menor inclinação de Biela:": f"{round(self.alfa *
180 / math.pi, 2)}°"}

            if alfa * 180 / math.pi < 55 and alfa * 180 / math.pi >= 45:
                self.textoverificacao = f"45° < alfa < 55° -> As bielas não
precisam ser verificadas"
                self.cor_textoverificacao = "green"
            elif alfa * 180 / math.pi >= 55:
                self.textoverificacao = f"alfa > 55° -> Inclinação não recomendada"
                self.cor_textoverificacao = "red"

        print("=" * 40)
        pprint([
            f"Altura Útil = {d} cm",
            f"Angulo = {alfa * 180 / math.pi} graus",
            f"Kr = {Kr}",
            f"Nd = {Nd} tf",
            f"Area do Pilar = {AreaP} cm2",
            f"Aread da Estaca = {AreaE} cm2",
            f"Lado Eqv = {l_pilareqv} cm",
            f"Dist Estacas = {l} cn",
            f"Fcd = {fcd} tf/cm2",
            f"Fyd = {fyd} tf/cm2",
        ])
        print("=" * 40)

        APPRESULTADOS = AppResultados(self)
        reacoes_pior_combinacao, titulo_pior_combinacao =
self.verificar_estaqueamento()
        APPRESULTADOS.definir_lista_reacoes([reacoes_pior_combinacao],
[titulo_pior_combinacao])
        APPRESULTADOS.definir_texto_verificao(self.textoverificacao,
self.cor_textoverificacao)
        APPRESULTADOS.criar_widgets_relatorio(self.textos)
        APPRESULTADOS.mainloop()

class CEBFIP:
    def __init__(self,mainapp, classebloco):
        self.bloco = classebloco
        self.d = self.bloco.alturautilbloco
        self.pilares = classebloco.listapilares
        self.estaqueameto = classebloco.estacas_coords
        self.mainapp = mainapp

        self.menor_alfa_rad = self.calc_alfa()[0]
        self.menor_alfa_graus = round(self.menor_alfa_rad*180/math.pi,2)
        self.d_min = round(self.calc_alfa()[2],1)
        self.maior_alfa_rad = self.calc_alfa()[1]
        self.maior_alfa_graus = round(self.maior_alfa_rad * 180 / math.pi, 2)
        self.pilar_lx = classebloco.pilar_lx
        self.pilar_ly = classebloco.pilar_ly

        if not len(self.pilares) > 0:
            PopupAviso(self.mainapp, text="Nenhum pilar foi selecionado !",
color="red")

        elif self.menor_alfa_graus < 40:
```

```
            #PopupAviso(self.mainapp, text=f"Menor alfa = {self.menor_alfa_graus}°
< 40°", color="red")
            PopupAviso(self.mainapp, text=f"alfa < 40°, dmín = {self.d_min}",
color="red")

        else:
            self.dimensionar()

    def gerar_combinacoes(self):
        self.combinacoes = []
        for pilar in self.pilares:
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.caso0[0], "MxTotal": pilar.caso0[1],
                                     "MyTotal": pilar.caso0[2], "NomeCombinacao":
"Caso 0"})
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.fzmax[0], "MxTotal": pilar.fzmax[1],
                                     "MyTotal": pilar.fzmax[2], "NomeCombinacao":
"Fz Max"})
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.mxmax[0], "MxTotal": pilar.mxmax[1],
                                     "MyTotal": pilar.mxmax[2], "NomeCombinacao":
"My Max"})
            self.combinacoes.append({"TituloPilar": pilar.name, "FzTotal":
pilar.mymax[0], "MxTotal": pilar.mymax[1],
                                     "MyTotal": pilar.mymax[2], "NomeCombinacao":
"My Max"})

            for i, combinacao in enumerate(pilar.casos_adicionais):
                FzTotal = combinacao[0] + pilar.caso0[0]
                MxTotal = combinacao[1] + pilar.caso0[1]
                MyTotal = combinacao[2] + pilar.caso0[2]

                self.combinacoes.append(
                    {"TituloPilar": pilar.name, "FzTotal": FzTotal, "MxTotal":
MxTotal, "MyTotal": MyTotal,
                     "NomeCombinacao": f"Vento {i + 1}"})

    def gerar_cargas_estacas(self):
        resultados = []
        nomes_combinacao = []
        for i, combinacao in enumerate(self.combinacoes):
            Fz = combinacao["FzTotal"]
            Mx = combinacao["MxTotal"]
            My = combinacao["MyTotal"]
            TituloPilar = combinacao["TituloPilar"]
            resultados.append(verificarestaqueamento(Fz, Mx, My,
self.estaqueameto))
            nomes_combinacao.append(f"{TituloPilar} -
{combinacao["NomeCombinacao"]}")

        return resultados,nomes_combinacao

    def calc_alfa(self):
        menoralfa = math.pi / 2
        maioralfa = 0.0
        d_min = 0

        # Garante que a altura útil 'd' exista e seja positiva
        if not hasattr(self, 'd') or self.d <= 0:
            print("Aviso: 'self.d' (altura útil) não definido ou é zero. Retornando
0, 0.")
            return 0.0, 0.0

        # Garante que a lista de estacas exista
        if not hasattr(self, 'estaqueameto') or not self.estaqueameto:
            print("Aviso: 'self.estaqueameto' está vazio. Retornando 0, 0.")
            return 0.0, 0.0
```

```
        # 2. Itera por todas as estacas
        for estaca_x, estaca_y in self.estaqueameto:
            distpilarestaca = (estaca_x ** 2 + estaca_y ** 2) ** 0.5

            # 3. Lógica de cálculo do ângulo (alfa)
            if distpilarestaca == 0:
                alfa=None
            else:
                # Caso padrão: calcula o ângulo com base na distância
                alfa = math.atan(self.d / distpilarestaca)

            # 4. Atualiza o maior e o menor ângulo encontrado
            if alfa is not None:
                if alfa > maioralfa:
                    maioralfa = alfa
                if alfa < menoralfa:
                    menoralfa = alfa
                    d_min = distpilarestaca * math.tan(40*math.pi/180)

        # 5. CORREÇÃO: Retorna os valores *após* o loop ter terminado
        return menoralfa, maioralfa, d_min

    def dimensionar(self):
        self.gerar_combinacoes()
        self.cargas_estacas, self.nomes_combinacoes = self.gerar_cargas_estacas()

        self.estacas = []
        for i, (x, y) in enumerate(self.estaqueameto):
            # Cria uma lista de cargas para esta estaca, uma para cada combinação
            cargas_da_estaca = [grupo_cargas[i] for grupo_cargas in
self.cargas_estacas]
            self.estacas.append({"x": x, "y": y, "cargas": cargas_da_estaca})

        # Variáveis para armazenar os maiores valores encontrados
        self.max_Asx = 0
        self.max_Asy = 0
        self.combinacao_Asx = "Nenhuma"
        self.combinacao_Asy = "Nenhuma"

        # Constantes
        fyd = 500 / (100 * 1.15)  # tf/cm2
        d = self.bloco.alturautilbloco / 100  # Converte altura útil para metros
        braco_alavanca_z = 0.85 * d  # Simplificação para z = 0.85d

        if d == 0:
            print("Erro: Altura útil (d) não pode ser zero.")
            return

        # Itera por cada combinação de carga
        for i, nome_combinacao in enumerate(self.nomes_combinacoes):

            # Inicializa os momentos para esta combinação específica
            Mesq, Mdir, Mcim, Mbaix = 0, 0, 0, 0

            # Loop único para calcular todos os momentos da combinação
            for estaca in self.estacas:
                # CORREÇÃO 1: Usa a carga da combinação atual 'i'
                carga_estaca = estaca["cargas"][i]
                x, y = estaca["x"], estaca["y"]

                # Calcula momentos em X (para armadura Asx)
                if x < 0:
                    Mesq += carga_estaca * (-x - self.pilar_lx *0.35) / 100
                elif x > 0:
                    Mdir += carga_estaca * (x - self.pilar_lx *0.35) / 100

                # Calcula momentos em Y (para armadura Asy)
```

```
                if y < 0:
                    Mcim += carga_estaca * (-y - self.pilar_ly *0.35) / 100
                elif y > 0:
                    Mbaix += carga_estaca * (y - self.pilar_ly *0.35) / 100

            # Calcula a armadura necessária para ESTA combinação
            Asx_comb = max(Mesq, Mdir)  * 1.4/ (fyd * braco_alavanca_z)

            # CORREÇÃO 2: Asy deve usar Mcim e Mbaix
            Asy_comb = max(Mcim, Mbaix)  * 1.4/ (fyd * braco_alavanca_z)

            # Atualiza os valores máximos se os valores desta combinação forem
maiores
            if Asx_comb > self.max_Asx:
                self.max_Asx = Asx_comb
                self.combinacao_Asx = nome_combinacao

            if Asy_comb > self.max_Asy:
                self.max_Asy = Asy_comb
                self.combinacao_Asy = nome_combinacao

        # Salva os resultados finais no objeto
        self.Asx_final = self.max_Asx
        self.Asy_final = self.max_Asy

        # Imprime o resultado final, que era o seu objetivo
        print("--- Dimensionamento Concluído ---")
        print(f"Maior Asx: {self.Asx_final:.2f} cm² (Combinação:
{self.combinacao_Asx})")
        print(f"Maior Asy: {self.Asy_final:.2f} cm² (Combinação:
{self.combinacao_Asy})")

        self.textos = {"Maior Asx:":f"{self.combinacao_Asx} - {round(self.max_Asx,
2)} cm2",
                       "Maior Asy:":f"{self.combinacao_Asy} - {round(self.max_Asy,
2)} cm2"}

        if self.combinacao_Asx == self.combinacao_Asy:
            self.piores_cargas_estacas =
[self.cargas_estacas[self.nomes_combinacoes.index(self.combinacao_Asx)]]
            self.pior_combinacao = [self.combinacao_Asx]
        else:
            self.piores_cargas_estacas =
[self.cargas_estacas[self.nomes_combinacoes.index(self.combinacao_Asx)],

self.cargas_estacas[self.nomes_combinacoes.index(self.combinacao_Asy)]]
            self.pior_combinacao = [self.combinacao_Asx, self.combinacao_Asy]

        APPRESULTADOS = AppResultados(self)
        APPRESULTADOS.criar_widgets_relatorio(self.textos)
        APPRESULTADOS.definir_lista_reacoes(self.piores_cargas_estacas,
self.pior_combinacao)
        #APPRESULTADOS.definir_texto_verificao(self.textoverificacao,
self.cor_textoverificacao)
        #APPRESULTADOS.criar_widgets_relatorio(self.textos)
        APPRESULTADOS.mainloop()

class FrameRelatorio(ctk.CTkFrame):
    def __init__(self,master,titulo, texto=None, texto_color = "white",texto_font =
("Arial",16), **kwargs):
        super().__init__(master,**kwargs)
        self.titulo = titulo
        self.texto = texto
        self.texto_color = texto_color
        self.texto_font = texto_font

        self.fg_color = "#f2af0d" #3B3B3B
        self.titulo_color = "black"
```

```
        self.label_width = 300
        self.label_titulo = ctk.CTkLabel(self, text=self.titulo, font=("Arial",
18,"bold"),fg_color=self.fg_color, text_color=self.titulo_color,corner_radius=10,
width=self.label_width)
        self.label_texto = ctk.CTkLabel(self, text=self.texto,
font=self.texto_font, text_color=self.texto_color)

        self.padx = (0,10)
        self.pady = (0, 10)

    def grid_default(self,**kwargs):
        if self.texto is None:
            return

        kwargs.setdefault("pady", self.pady)
        kwargs.setdefault("padx", self.padx)
        kwargs.setdefault("sticky", "we")
        super().grid(**kwargs)

        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(0, weight=0)
        self.grid_rowconfigure(1, weight=0)

        self.label_titulo.grid(row=0,column=0,padx=10, pady=(10, 10), sticky="we")
        self.label_texto.grid(row=1,column=0, pady=(0,0))

class AppResultados(ctk.CTk):
    def __init__(self, dimensionamento):
        super().__init__()

        self.title("Resultados do Dimensionamento do Bloco")
        self.geometry("1200x600")
        self.resizable(width=False, height=False)

        self.lift()
        self.focus_set()
        # self.grab_set()

        # Configura o grid principal da janela para ter duas colunas que se
expandem
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=5)
        self.grid_rowconfigure(0, weight=1)

        self.frame_relatorio = ctk.CTkFrame(self, fg_color="transparent")
        self.frame_relatorio.grid(row=0, column=1, padx=(0, 20), pady=20,
sticky="nsew")
        self.frame_relatorio.grid_rowconfigure(0, weight=0)
        self.frame_relatorio.grid_rowconfigure(1, weight=0)
        self.frame_relatorio.grid_rowconfigure(2, weight=0)
        self.frame_relatorio.grid_rowconfigure(3, weight=0)
        self.frame_relatorio.grid_rowconfigure(4, weight=1)
        self.frame_relatorio.grid_columnconfigure(0, weight=1)
        self.frame_relatorio.grid_columnconfigure(1, weight=1)

        self.dimensionamento = dimensionamento

    def definir_lista_reacoes(self, listas_de_cargas, listas_de_titulos):
        # Armazena as listas de cargas e títulos que serão exibidas.
        # listas_de_cargas pode ser [[...]] ou [[...], [...]]
        self.listas_de_cargas_estacas = listas_de_cargas
        # listas_de_titulos pode ser ["Comb A"] ou ["Comb A", "Comb B"]
        self.listas_de_titulos_combinacoes = listas_de_titulos

        # Cria o frame principal para a seção de reações
        self.frame_reacoes = ctk.CTkFrame(self, fg_color="transparent")
        self.frame_reacoes.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
```

```
        # Chama a função que vai desenhar os widgets
        self.criar_widgets_reacoes()

    def definir_texto_verificao(self,texto, texto_color="white"):
        self.label_verificao = FrameRelatorio(self.frame_relatorio,
titulo="Verificações", texto=texto,
                                              texto_color=texto_color)
        self.label_verificao.grid_default(row=4, column=0,
                                          columnspan=2, sticky="sew")

    def criar_widgets_reacoes(self):
        """
        Cria a seção da lista rolável, agora com capacidade para
        exibir múltiplas combinações de carga.
        """
        # Configura o grid do frame da esquerda
        self.frame_reacoes.grid_rowconfigure(0, weight=1)  # Permite que a lista
rolável expanda
        self.frame_reacoes.grid_columnconfigure(0, weight=1)

        # 1. Cria a lista rolável principal
        # Note que o label foi removido daqui, pois teremos labels dinâmicos dentro
dela.
        lista_rolavel = ctk.CTkScrollableFrame(self.frame_reacoes)
        lista_rolavel.grid(row=0, column=0, sticky="nsew")

        # 2. Itera sobre cada combinação que foi passada
        #    zip() junta o título "Combinação Asx" com a lista de cargas [20,
30, ...]
        #    e depois o título "Combinação Asy" com a lista [15, 35, ...]
        for titulo_comb, cargas_da_combinacao in
zip(self.listas_de_titulos_combinacoes, self.listas_de_cargas_estacas):

            # 3. Cria um Label de Título para esta combinação
            label_titulo = ctk.CTkLabel(lista_rolavel,
                                        text=f"{titulo_comb}",
                                        font=("Arial", 16, "bold"),
                                        fg_color="#f2af0d",
                                        text_color="black",
                                        corner_radius=5)
            # 'ipady' dá um preenchimento interno vertical para o label ficar mais
bonito
            label_titulo.pack(fill="x", padx=5, pady=(15, 5), ipady=4)

            # 4. Cria um frame para agrupar as reações desta combinação
            frame_cargas = ctk.CTkFrame(lista_rolavel, fg_color="#333333")
            frame_cargas.pack(fill="x", padx=5, pady=(0, 10))

            # 5. Itera sobre as cargas (E1, E2, ...) desta combinação específica
            for i, reacao in enumerate(cargas_da_combinacao):
                estaca_nome = f"E{i + 1}"

                item_frame = ctk.CTkFrame(frame_cargas, fg_color="transparent")
                item_frame.pack(fill="x", padx=10, pady=3)
                item_frame.grid_columnconfigure(1, weight=1)

                label_estaca = ctk.CTkLabel(item_frame, text=estaca_nome,
font=("Arial", 16, "bold"))
                label_estaca.grid(row=0, column=0, sticky="w")

                label_reacao = ctk.CTkLabel(item_frame, text=f"{reacao:.2f} tf",
font=("Arial", 16))
                label_reacao.grid(row=0, column=1, sticky="e", padx=(10, 0))
```

```
 def criar_widgets_relatorio(self,textos):
        titulos = list(textos.keys())
        textos = list(textos.values())

        x = -1
        y = 0
        for i,titulo in enumerate(titulos):
            x += 1
            if x == 2:
                y += 1
                x = 0

            texto = textos[i]
            FrameRelatorio(self.frame_relatorio,
titulo=titulo,texto=texto).grid_default(row=y, column=x)
```

ANEXO E – Código moduloselecaopilares.py

```
import os
import tkinter.filedialog

import customtkinter as ctk
import pickle

from ezdxf.render.mesh import normalize_mesh

from helpfunctions import *

class Pilar:
    def __init__(self, linhaplanilha):
        self.linhaplanilha = linhaplanilha

        self.name = linhaplanilha[0]
        self.caso0 = linhaplanilha[1:4]
        self.fzmax = linhaplanilha[4:7]
        self.mxmax = linhaplanilha[7:10]
        self.mymax = linhaplanilha[10:13]
        self.casos_adicionais = []

        for i in range(15,len(linhaplanilha),3):
            self.casos_adicionais.append([linhaplanilha[i-2],linhaplanilha[i-
1],linhaplanilha[i]])

        print(f"""
        Pilar: {self.name}
        Caso0: {self.caso0}
        FzMaz: {self.fzmax}
        Mmax: {self.mxmax}
        Mymax: {self.mymax}
        Casos adicionais: {self.casos_adicionais}
        """)

class SelecaoPilares(ctk.CTkToplevel):
    def __init__(self,pastadetrabalho=None, callback=None, jaselecionados=None):
        super().__init__()

        if jaselecionados is None:
            jaselecionados = []

        self.callback = callback

        self.pastadetrabalho = pastadetrabalho
        if self.pastadetrabalho is None:
            self.pastadetrabalho = tkinter.filedialog.askdirectory()

        self.pastapilares = os.path.join(self.pastadetrabalho, "Pilares")

        self.PILARES =[]

        nomepilares = [nomepilar for nomepilar in os.listdir(self.pastapilares)]
        nomepilares = organizar_titulos(nomepilares)

        for nomepilar in nomepilares:
            camihnopilar = os.path.join(self.pastapilares, nomepilar)
            PILAR = pickle.load(open(camihnopilar, "rb"))
            self.PILARES.append(PILAR)

        self.title("MÓDULO DE SELEÇÃO DOS PILARES")
        self.geometry("400x600")
```

```
        # Cria uma cópia da lista para evitar modificar a original diretamente
        self.selected_objetos = list(jaselecionados)
        self.highlighted_index = 0
        self.labels = []

        # Cria um conjunto com os nomes dos pilares já selecionados para uma busca
rápida
        nomes_ja_selecionados = {pilar.name for pilar in self.selected_objetos}
        # Define o status inicial de cada pilar (selecionado ou não)
        self.selection_status = {obj.name: (obj.name in nomes_ja_selecionados) for
obj in self.PILARES}
        # --- FIM DA FUNCIONALIDADE ---

        self.setup_ui()
        self.bind_events()

        self.lift()
        self.focus_set()
        self.grab_set()

        self.update_highlight()

    def setup_ui(self):
        self.label_frame = ctk.CTkScrollableFrame(self)
        self.label_frame.pack(pady=10, padx=10, fill="both", expand=True)

        for i, obj in enumerate(self.PILARES):
            label = ctk.CTkLabel(master=self.label_frame, text=obj.name,
font=("Arial",20), anchor="w", fg_color="transparent",
                                 corner_radius=5, justify="center")
            label.pack(pady=3, padx=5, fill="x")
            self.labels.append(label)
            label.bind("<Button-1>", lambda event, idx=i: self.on_click(idx))

        select_button = ctk.CTkButton(
            master=self,
            text="SELECIONAR",
            command=self.on_select_button_click,
            font = ("Arial",20,"bold"),
            fg_color = "#f2af0d",
            text_color = "black",
        )
        select_button.pack(pady=5, padx=10)

        instructions_label = ctk.CTkLabel(
            master=self,
            text="Use ↑↓ para navegar, Enter para selecionar/desselecionar.\nClique
com o mouse para selecionar/desselecionar.\nPressione ESC para limpar.",
            font=ctk.CTkFont(size=12, slant="italic"),
            text_color="gray"
        )
        instructions_label.pack(pady=5, padx=10)

    def bind_events(self):
        self.bind("<Up>", self.navigate_up)
        self.bind("<Down>", self.navigate_down)
        self.bind("<Return>", self.toggle_selection)
        self.bind("<Escape>", self.deselect_all)
        self.bind("<Shift-Return>", self.on_select_button_click)

    def on_click(self, index):
        self.highlighted_index = index
        self.toggle_selection()
```

```
    def update_highlight(self):
        for i, label in enumerate(self.labels):
            obj_name = self.PILARES[i].name
            bg_color = "#5f5f5f" if i == self.highlighted_index else "transparent"
            text_color = "#f2af0d" if self.selection_status[obj_name] else "white"
            label.configure(fg_color=bg_color, text_color=text_color)



    def navigate_up(self, event=None):
        if self.highlighted_index > 0:
            self.highlighted_index -= 1
            self.update_highlight()

    def navigate_down(self, event=None):
        if self.highlighted_index < len(self.labels) - 1:
            self.highlighted_index += 1
            self.update_highlight()

    def toggle_selection(self, event=None):
        if self.labels:
            current_obj = self.PILARES[self.highlighted_index]
            self.selection_status[current_obj.name] = not
self.selection_status[current_obj.name]
            self.update_selection_list(current_obj)
            self.update_highlight()



    def update_selection_list(self, obj):
        is_selected = self.selection_status[obj.name]
        is_in_list = any(p.name == obj.name for p in self.selected_objetos)

        if is_selected and not is_in_list:
            self.selected_objetos.append(obj)
        elif not is_selected and is_in_list:
            # Remove o objeto correspondente pelo nome
            self.selected_objetos = [p for p in self.selected_objetos if p.name !=
obj.name]



    def deselect_all(self, event=None):
        """Deseleciona todos os pilares marcados."""
        for name in self.selection_status:
            self.selection_status[name] = False
        self.selected_objetos.clear()
        self.update_highlight()
        print("Toda a seleção foi limpa.")

    def on_select_button_click(self,event=None):
        self.callback(self.selected_objetos)
        self.destroy()



class CriacaoPilar(ctk.CTkToplevel):
    def __init__(self, master, callback):
        """
        Janela modal para inserção manual de esforços (Fz, Mx, My)
        para um pilar de teste.
        """
        super().__init__(master)

        self.callback = callback
        self.title("Esforços de Teste")
        self.geometry("300x300")
        self.resizable(False, False)
```

```
        # Variáveis para armazenar os valores dos Entries
        self.fz_var = ctk.StringVar(value="100")  # Valor padrão Fz=100tf
        self.mx_var = ctk.StringVar(value="0")
        self.my_var = ctk.StringVar(value="0")



        # Título
        ctk.CTkLabel(self, text="Insira os esforços", font=("Arial", 16,
"bold")).pack(pady=(20, 10))



         # Frame para os campos de entrada
        frame_entradas = ctk.CTkFrame(self, fg_color="transparent")
        frame_entradas.pack(pady=10, padx=20, fill="x")
        frame_entradas.grid_columnconfigure(1, weight=1)



        # Campo Fz
        ctk.CTkLabel(frame_entradas, text="Carga Vertical (Fz) (tf):").grid(row=0,
column=0, padx=(0, 10),

       pady=5,

sticky="w")
        self.entry_fz = ctk.CTkEntry(frame_entradas, textvariable=self.fz_var)
        self.entry_fz.grid(row=0, column=1, pady=5, sticky="ew")

        # Campo Mx
        ctk.CTkLabel(frame_entradas, text="Momento em X (Mx) (tf.m):").grid(row=1,
column=0, padx=(0, 10), pady=5,

sticky="w")
        self.entry_mx = ctk.CTkEntry(frame_entradas, textvariable=self.mx_var)
        self.entry_mx.grid(row=1, column=1, pady=5, sticky="ew")

        # Campo My
        ctk.CTkLabel(frame_entradas, text="Momento em Y (My) (tf.m):").grid(row=2,
column=0, padx=(0, 10), pady=5,

sticky="w")
        self.entry_my = ctk.CTkEntry(frame_entradas, textvariable=self.my_var)
        self.entry_my.grid(row=2, column=1, pady=5, sticky="ew")

        # Label de erro (inicialmente vazia)
        self.label_erro = ctk.CTkLabel(self, text="", text_color="red")
        self.label_erro.pack(pady=5, padx=20)

        # Botão Confirmar
        confirm_button = ctk.CTkButton(self, text="Usar Esforços",
command=self.confirmar, fg_color="#f2af0d",
                                       text_color="black", font=("Arial", 16,
"bold"))

        confirm_button.pack(pady=10, padx=20, fill="x")

        # Configurações da janela modal
        self.transient(master)  # Mantém a janela na frente da principal
        self.grab_set()  # Impede interação com a janela principal
        self.focus_set()  # Foca automaticamente nesta janela
        self.entry_fz.focus()  # Foca no primeiro campo de entrada

        # Bind 'Enter' para confirmar
        self.bind("<Return>", self.confirmar)
```

```
        def confirmar(self, event=None):
         """
         Coleta os valores, chama o callback e fecha a janela.
         """


         try:
             # Tenta converter os valores para float
             fz = float(self.fz_var.get())
             mx = float(self.mx_var.get())
             my = float(self.my_var.get())

             # --- Simulação de um Objeto Pilar ---
             # Criamos um objeto simples que "imita" a estrutura do seu objeto
'Pilar'
             # para que o módulo de dimensionamento possa usá-lo da mesma forma.
             class PilarTeste:
                 def __init__(self, fz, mx, my):
                     self.name = f"(Fz{fz};Mx{mx};My{my})"
                     self.caso0 = [fz, mx, my]
                     self.vento0 = [0, 0, 0]
                     self.vento90 = [0, 0, 0]
                     self.vento180 = [0, 0, 0]
                     self.vento270 = [0, 0, 0]
                     self.fzmax = [fz, mx, my]
                     self.mxmax = [fz, mx, my]
                     self.mymax = [fz, mx, my]
                     self.casos_adicionais = []

             pilar_teste = PilarTeste(fz, mx, my)

             # Chama o callback da AppModelagem, passando o pilar de teste
             # dentro de uma lista (para ser igual ao retorno da SelecaoPilares)
             self.callback([pilar_teste])

             # Fecha a janela de diálogo
             self.destroy()

         except ValueError:
             # Caso o usuário digite algo que não seja um número
             self.label_erro.configure(text="Erro: Insira apenas números.")
         except Exception as e:
             print(f"Erro inesperado: {e}")
             self.destroy()
```

ANEXO F – Código helpfunctions.py

```
import math

import customtkinter as ctk
from PIL import Image  # Importa a classe Image da biblioteca Pillow
import re

def distp1p2(p1,p2):
    dx = p1[0]-p2[0]
    dy = p1[1]-p2[1]
    return math.sqrt(dx**2 + dy**2)

def media(x1,x2):
    return(x1+x2)/2

def pprint(objeto,type="LIST"):
    if type.upper()=="DICT":
        for i in objeto:
            print(i, [n for n in objeto[i]])

    elif type.upper()=="LIST":
        for i in objeto:
            print(i)

class MyImage(ctk.CTkLabel):
    def __init__(self, master, caminhoimagem, scale = 1):
        ctk.CTkLabel.__init__(self, master=master, text="")
        self.master = master
        self.scale = scale
        self.caminhoimagem = caminhoimagem

        self.imagem_pil = Image.open(self.caminhoimagem)
        self.largura, self.altura = self.imagem_pil.size[0] * self.scale,
self.imagem_pil.size[1] * self.scale

        self.minha_imagem = ctk.CTkImage(light_image=self.imagem_pil,
                                         dark_image=self.imagem_pil,
                                         size=(self.largura, self.altura))

        self.configure(image=self.minha_imagem)

def natural_sort_key(s):
    """
    Função auxiliar que gera uma chave para ordenação natural (alfanumérica).
    Ela divide a string em partes de texto e partes numéricas, permitindo
    que a ordenação trate números corretamente (ex: 'P2' vem antes de 'P10').
    """
    # Divide a string em uma lista de partes de texto e partes numéricas. Ex:
'P101A' -> ['P', '101', 'A']
    parts = re.split(r'(\d+)', s)
    # Converte as partes numéricas para inteiros para que a comparação seja
numérica.
    # As partes de texto são convertidas para minúsculas para a ordenação não
diferenciar maiúsculas/minúsculas.
    return [int(part) if part.isdigit() else part.lower() for part in parts]
```

```
def organizar_titulos(lista_de_strings):
    """
    Recebe uma lista de strings contendo letras e números e a retorna
    organizada em ordem alfanumérica natural.

    Args:
        lista_de_strings (list): Uma lista de strings para ser ordenada.

    Returns:
        list: A lista de strings organizada.
    """
    lista_organizada = sorted(lista_de_strings, key=natural_sort_key)
    return lista_organizada

class PopupAviso(ctk.CTkToplevel):
    def __init__(self, master, title="Aviso", text="Mensagem padrão.",
color="white"):
        """
        Cria uma janela de popup modal (Toplevel) para exibir uma mensagem.
        A janela espera ser fechada antes de devolver o controle.
        """
        super().__init__(master)

        self.title(title)
        self.geometry("400x170")  # Tamanho fixo para o popup
        self.resizable(False, False)

        # --- Configuração Modal ---
        self.transient(master)  # Mantém o popup na frente da janela 'master'
        self.grab_set()  # Bloqueia interações com a janela 'master'
        self.focus_set()  # Foca nesta janela

        # --- Widgets ---

        # Frame principal para centralizar o conteúdo
        main_frame = ctk.CTkFrame(self, fg_color="transparent")
        main_frame.pack(pady=20, padx=20, fill="both", expand=True)
        main_frame.grid_rowconfigure(0, weight=1)
        main_frame.grid_columnconfigure(0, weight=1)

        # Label da Mensagem
        label_mensagem = ctk.CTkLabel(main_frame,
                                      text=text,
                                      text_color=color,
                                      font=("Arial", 18),
                                      wraplength=360)  # Quebra de linha automática
        label_mensagem.grid(row=0, column=0, sticky="nsew")

        # Botão OK
        botao_ok = ctk.CTkButton(main_frame,
                                 text="OK",
                                 font=("Arial", 14, "bold"),
                                 width=100,
                                 fg_color="#f2af0d",
                                 text_color="black",
                                 command=self.destroy)  # Fecha o popup ao ser
clicado
        botao_ok.grid(row=1, column=0, pady=(20, 0))
```

```
        # Bind "Enter" para fechar a janela
        self.bind("<Return>", lambda event: self.destroy())
        self.bind("<Escape>", lambda event: self.destroy())

        # --- Espera ---
        # Pausa a execução do código que a chamou até que esta janela (self) seja
destruída.
        self.wait_window(self)
```